

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

A tarefa consiste en crear e probar un disparador na base de datos *praticas1* 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.empregado
(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:

## Tarefa 2. Consultar información sobre os disparadores creados

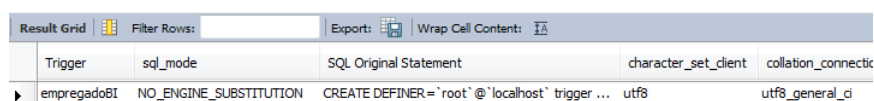
A tarefa consiste en consultar os disparadores asociados á base de datos *praticas1*, e mostrar información do disparador *praticas1.empregadoBI* creado na *tarefa1*.

### 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 praticas1
select * from information_schema.triggers
  where trigger_schema = tendabd;
-- consulta dos triggers de praticas1
show triggers from praticas1;
-- consulta da información do trigger praticas1.empregadoBI
show create trigger praticas1.empregadoBI;
```

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_connection
empregadoBI	NO_ENGINE_SUBSTITUTION	CREATE DEFINER='root'@'localhost' trigger ...	utf8	utf8_general_ci

## 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 dunha 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;

```

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_employado, 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

Columnas actualizadas

```

-- 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

## 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;
```

Result Grid		Filter Rows:	Export:	Wrap Cell Contents:			
Trigger	Event	Table	Statement	Timing	Created	sql_mode	
empregadoBI	INSERT	empregado	begin if (select count(*) from departamento wh...	BEFORE	NULL	NO_ENGINE_SUBSTITU	

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 *registroOperacions*.

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

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

## 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.registroOperacio...	AFTER	NO_ENGINE_SU	NO_ENGINE_SU
departamentoAU	UPDATE	departamento	begin insert into traballadores.registroOperacio...	AFTER	NO_ENGINE_SU	NO_ENGINE_SU
departamentoAD	DELETE	departamento	begin insert into traballadores.registroOperacio...	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:

- Registrar na táboa *registroOperacions* 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

```

## 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;
```

## 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 *ventas* 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 ;
```