Dirección Xeral de Educación, Formación Profesional e Innovación Educativa

Material para

a formación profesional inicial

A03. Disparadores e eventos

|  |  |  |
| --- | --- | --- |
| Familia profesional | IFC | Informática e comunicacións |
| Ciclo formativo | CSIFC03  CSIFC02 | Desenvolvemento de aplicacións web  Desenvolvemento de aplicacións multiplataforma |
| Grao |  | Superior |
| Módulo profesional | MP0484 | Bases de datos |
| Unidade didáctica | UD07 | Programación de bases de datos |
| **Actividade** | **A03** | **Disparadores e eventos** |
| Autores |  | Marta Fernández García  María del Carmen Fernández Lameiro  Miguel Fraga Vila  María Carmen Pato González  Andrés del Río Rodríguez |
| Nome do arquivo |  | CSIFC02\_ MP0484\_V000703\_UD07\_A03\_DisparadorEvento.docx |
| © 2015 Xunta de Galicia.  Consellería de Cultura, Educación e Ordenación Universitaria.  Este traballo foi realizado durante unha licenza de formación retribuída pola Consellería de Cultura, Educación e Ordenación Universitaria e ten licenza CreativeCommons BY-NC-SA (recoñecemento - non comercial - compartir igual). Para ver unha copia desta licenza, visitar a ligazón http://creativecommons.org/licenses/by-nc-sa/3.0/es/. | | |

Índice

[1. Ficha técnica 4](#_Toc439856421)

[Contexto da actividade 4](#_Toc439856422)

[Título da actividade 5](#_Toc439856423)

[Resultados de aprendizaxe do currículo 5](#_Toc439856424)

[Obxectivos didácticos e título e descrición da actividade 5](#_Toc439856425)

[Criterios de avaliación 5](#_Toc439856426)

[Contidos 5](#_Toc439856427)

[Actividades de ensino e aprendizaxe e de avaliación, métodos, recursos e instrumentos de avaliación 6](#_Toc439856428)

[2. A03. Disparadores e eventos 7](#_Toc439856429)

[2.1 Introdución 7](#_Toc439856430)

[2.1.1 Obxectivos 7](#_Toc439856431)

[2.1.2 Software 7](#_Toc439856432)

[2.1.3 Bases de datos de traballo 8](#_Toc439856433)

[2.2 Actividade 8](#_Toc439856434)

[2.2.1 Disparadores (*triggers*) 8](#_Toc439856435)

[2.2.1.1 Sentenza CREATE TRIGGER 8](#_Toc439856436)

[2.2.1.2 Sentenzas SHOW TRIGGERS e SHOW CREATE TRIGGER 9](#_Toc439856437)

[2.2.1.3 Identificadores NEW e OLD en disparadores 10](#_Toc439856438)

[2.2.1.4 Utilización de disparadores 10](#_Toc439856439)

[2.2.1.5 Sentenza DROP TRIGGER 14](#_Toc439856440)

[2.2.2 Planificador de eventos 14](#_Toc439856441)

[2.2.2.1 Sentenza CREATE EVENT 15](#_Toc439856442)

[2.2.2.2 Exemplos de creación de eventos 16](#_Toc439856443)

[2.2.2.3 Sentenzas SHOW EVENTS e SHOW CREATE EVENT 17](#_Toc439856444)

[2.2.2.4 Sentenza ALTER EVENT 17](#_Toc439856445)

[2.2.2.5 Sentenza DROP EVENT 18](#_Toc439856446)

[2.3 Tarefas 18](#_Toc439856447)

[2.3.1 Tarefa 1. Crear disparadores para validar a entrada de datos 18](#_Toc439856448)

[Solución 19](#_Toc439856449)

[2.3.2 Tarefa 2. Consultar información sobre os disparadores creados 19](#_Toc439856450)

[Solución 20](#_Toc439856451)

[2.3.3 Tarefa 3. Crear disparadores para actualizar atributos derivados 20](#_Toc439856452)

[Solución 20](#_Toc439856453)

[2.3.4 Tarefa 4. Crear disparadores para levar rexistros de operacións 23](#_Toc439856454)

[Solución 23](#_Toc439856455)

[2.3.5 Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais 25](#_Toc439856456)

[Solución 25](#_Toc439856457)

[2.3.6 Tarefa 6. Borrar disparadores 26](#_Toc439856458)

[Solución 26](#_Toc439856459)

[2.3.7 Tarefa 7. Planificar eventos 26](#_Toc439856460)

[Solución 26](#_Toc439856461)

[3. Materiais 28](#_Toc439856462)

[3.1 Documentos de apoio ou referencia 28](#_Toc439856463)

[3.2 Recursos didácticos 28](#_Toc439856464)

[3.3 Material auxiliar 28](#_Toc439856465)

[4. Avaliación 29](#_Toc439856466)

[Criterios de avaliación 29](#_Toc439856467)

[Modelo de proba para TO.7 29](#_Toc439856468)

[Proba para CA7.10 30](#_Toc439856469)

[Exemplo de solución para entregar en papel 31](#_Toc439856470)

[Exemplo de lista de valoración para TO.7 32](#_Toc439856471)

1. Ficha técnica

Contexto da actividade

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Módulo | Duración  horas | Unidade didáctica. | Sesións 50´ | Actividades | Sesións 50´ |
| MP0484. Bases de datos | 187 | UD01. Bases de datos e sistemas de almacenamento da información | 12 | A01. Bases de datos e sistemas de almacenamento da información | 7 |
| A02. Introdución aos sistemas xestores de bases de datos | 5 |
| UD02. Deseño conceptual de bases de datos | 38 | A01. Fases do deseño de bases de datos | 3 |
| A02.Descrición e representación gráfica dos elementos do Modelo Entidade-Interrelación (MER) | 10 |
| A03. Descrición e representación gráfica dos elementos do Modelo Entidade-Interrelación Estendido(MERE) | 9 |
| A04. Construción e validación do modelo de datos | 11 |
| A05. Notación Martin e ferramentas CASE | 5 |
| UD03. Deseño lóxico de base de datos | 32 | A01. O deseño lóxico empregando o modelo relacional | 15 |
| A02. Normalización de relacións | 10 |
| A03. Operacións do MR: álxebra e cálculo | 7 |
| UD04. Deseño físico de bases de datos | 28 | A01. Creación da estrutura de bases de datos relacionais | 13 |
| A02. Modificación da estrutura de bases de datos relacionais | 9 |
| A03. Verificación e probas | 3 |
| A04. Utilización de ferramentas gráficas para a creación da estrutura de bases de datos relacionais | 3 |
| UD05. Consultas de datos | 39 | A01. Consultas simples cunha táboa. | 12 |
| A02. Consultas con datos de máis dunha táboa | 10 |
| A03. Consultas resumo con agrupamentos | 7 |
| A04. Consultas con subconsultas | 7 |
| A05. Optimización de consultas | 3 |
| UD06. Tratamento de datos | 24 | A01. Manipulación de datos | 12 |
| A02. Vistas | 5 |
| A03. Transaccións e bloqueos | 7 |
| UD07. Programación de bases de datos | **22** | A01. Introdución á programación con SQL | 4 |
| A02. Rutinas almacenadas | 9 |
| **A03. Disparadores e eventos** | **5** |
| A04. Cursores | 4 |
| UD08. Administración de bases de datos | 19 | A01. Copias de seguridade e intercambio de datos entre SXBDs | 10 |
| A02. Índices e xestión de usuarios | 9 |
| UD09. Bases de datos obxecto-relacionais | 10 | A01. Uso de bases de datos obxecto-relacionais | 10 |

NOTA: Esta actividade está vinculada á programación recollida no arquivo CSIFC02\_MP0487\_V000700\_UD07\_ProgramacionBD.pdf

Título da actividade

|  |  |  |  |
| --- | --- | --- | --- |
| Nº | Título | Descrición | Duración |
| A03 | Disparadores e eventos. | Crearanse disparadores e planificaranse eventos comprobando o seu funcionamento | 5 |

Resultados de aprendizaxe do currículo

|  |  |
| --- | --- |
| Resultados de aprendizaxe do currículo | Completo |
| * RA7 - Desenvolve procedementos almacenados e guións de sentenzas, para o que utiliza e avalía as sentenzas da linguaxe incorporada no sistema xestor de bases de datos. | Si |

Obxectivos didácticos e título e descrición da actividade

| Obxectivos específicos | | Actividade | | Descrición básica | Duración |
| --- | --- | --- | --- | --- | --- |
| O3.1 | Crear, eliminar e comprobar o funcionamento dos disparadores. | A03 | Disparadores e eventos. | Crearanse disparadores e planificaranse eventos comprobando o seu funcionamento. | 5 |
| O3.2 | Crear, eliminar, modificar e comprobar o funcionamento dos eventos. |

Criterios de avaliación

|  |
| --- |
| Criterios de avaliación |
| * CA7.10 Definíronse disparadores.   CA 7.10.1 Creáronse e elimináronse disparadores. Creáronse, modificáronse e elimináronse eventos. Comprobáronse os resultados de disparadores e eventos. |

Contidos

| Contidos |
| --- |
| * Eventos e disparadores. |

Actividades de ensino e aprendizaxe e de avaliación, métodos, recursos e instrumentos de avaliación

| Que e para que | Como | | | Con que | Como e con que se valora | Duración (sesións) |
| --- | --- | --- | --- | --- | --- | --- |
| Actividade (título e descrición) | Profesorado (en termos de tarefas) | Alumnado (tarefas) | Resultados  ou produtos | Recursos | Instrumentos e procedementos de avaliación |
| A03. Disparadores e eventos.   * Crearanse disparadores e planificaranse eventos comprobando o seu funcionamento. | * **Tp3.1** Explicación sobre sintaxe de sentenzas SQL relacionadas coa definición e eliminación de disparadores, definición, eliminación e modificación de eventos, e comprobación da execución. | **Ta3.1** Creación, mantemento, documentación e execución ou utilización de rutinas almacenadas (procedementos almacenados e funcións de usuario) utilizando sentenzas SQL e ferramentas gráficas. | * Guións de sentenzas SQL coa definición de disparadores e eventos. * Resultado da execución dos guións. | * Material didáctico subministrado polo profesorado en papel e/ou formato dixital. * Ordenadores con conexión a Internet, que terán instalado MySQL e MySQL Workbench. * Proxector. * Manual de referencia de MySQL |  | 4.5 |
|  | * **Ta3.2** Tarefa de avaliación utilizando o instrumento TO.7. | * Exame en papel e en formato dixital. | * Máquina virtual para exame que terá instalado MySQL e MySQL Workbench. * Manual de referencia de MySQL. | * **TO.7** Documento de rexistro de creación de disparadores, planificación de eventos e comprobación da execución de disparadores e eventos. Escala de valores (observación indirecta). | 0.5 |

1. A03. Disparadores e eventos
   1. Introdución
      1. Obxectivos

Os obxectivos desta actividade son:

* Crear disparadores (*triggers*).
* Planificar eventos.
  + 1. 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 utilizalo 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, mostraranse 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. Bases de datos de traballo

As bases de datos *practicas1*, *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 scritps atópanse no cartafol anexo a esta actividade descrito no apartado '3.3 Material auxiliar'.

* 1. Actividade
     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. 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 borralo. 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. 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:

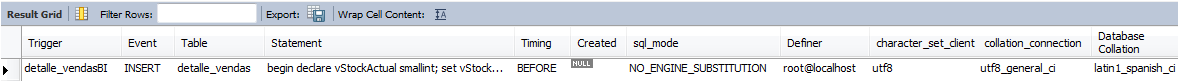
SHOW TRIGGERS [FROM nome\_bd]

[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 sentenza anterior móstrase na zona *Result Grid* de Workbench.



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.

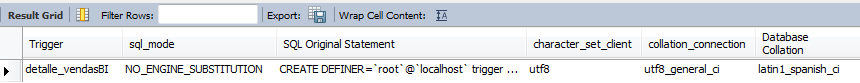
* A sentenza SHOW CREATE TRIGGER, permite ver información de como foi creado o disparador, incluído o código SQL utilizado para a creación. Sintaxe:

SHOW CREATE TRIGGER nome\_disparador

Exemplo:

**show** **create** **trigger** tendabd**.**detalle\_vendasBI**;**

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



* + - 1. 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.
  + - 1. 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.

Algúns 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.

/\*

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\_venda

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\_cantidade **>** 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:



* 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\_venda

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\_cantidade

**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\_cantidade* 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 *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.

/\*

u703exemplo\_triggerAU.sql

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

NOME DISPARADOR: clientes\_AU

DATA CREACIÓN: 16/11/2015

AUTOR: Grupo licenza 2015

TAREFA A AUTOMATIZAR: - Rexistra na táboa rexistro\_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

rexistro\_cambios\_desconto

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

**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 UDPDATE. 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** rexistro\_cambios\_desconto**;**



* 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:



* Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais.
  + - 1. Sentenza 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. 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'**;**



* + - 1. Sentenza 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. 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 *practicas1* 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 practicas1

**use** practicas1**;**

-- 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 practicas1

**use** practicas1**;**

-- 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. Sentenzas SHOW EVENTS e SHOW CREATE EVENT

A información dos eventos creados gárdase no dicionario 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.

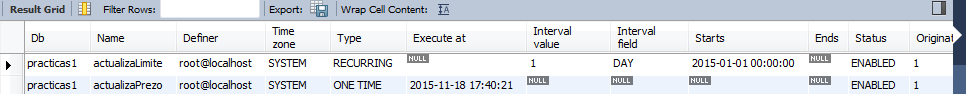
SHOW EVENTS permite ver os eventos creados. A sintaxe é:

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**;**

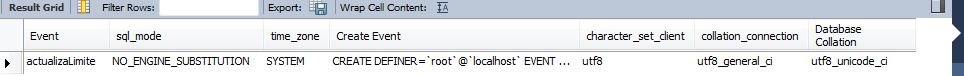


* A sentenza SHOW CREATE EVENT, permite ver información de como foi creado o evento, incluído o código SQL utilizado para a creación. Sintaxe:

SHOW CREATE EVENT nome\_evento

Exemplo:

**SHOW** **CREATE** **EVENT** practicas1**.**actualizaLimite**;**



* + - 1. Sentenza 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. Sentenza 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. 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. Tarefa 1. Crear disparadores para validar a entrada de datos

A tarefa consiste en crear e probar un disparador na base de datos *practicas1* 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:



* + 1. 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.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 practicas1

**select** **\*** **from** information\_schema**.triggers**

**where** trigger\_schema **=** tendabd**;**

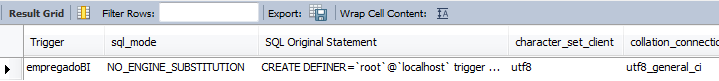
-- consulta dos triggers de practicas1

**show** **triggers** **from** practicas1**;**

-- consulta da información do trigger practicas1.empregadoBI

**show** **create** **trigger** practicas1**.**empregadoBI**;**

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



* + 1. 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\_venda* na táboa *clientes*. A columna *clt\_vendas* garda información do número de ventas que se lle fixeron ao cliente, e a columna *clt\_ultima\_venda* 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\_venda, 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\_venda actualizadas

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*/

-- disparador que actualiza as columnas clt\_vendas e clt\_ultima\_venda 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\_venda **=** **date(new.**ven\_data**)**

**where** clt\_id **=** **new.**ven\_cliente**;**

**end**

**//**

**delimiter;**

-- disparador que actualiza as columnas clt\_vendas e clt\_ultima\_venda 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\_venda

**if** **date(new.**ven\_data**)** **>** **(select** clt\_ultima\_venda

**from** clientes

**where** clt\_id **=** **new.**ven\_cliente**)**

**then**

**update** clientes

**set** clt\_ultima\_venda **=** **date(new.**ven\_data**)**

**where** clt\_id **=** **new.**ven\_cliente**;**

**end** **if;**

**end**

**//**

**delimiter;**

-- disparador que actualiza as columnas clt\_vendas e clt\_ultima\_venda 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\_venda

**if** **date(old.**ven\_data**)** **=** **(select** clt\_ultima\_venda

**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\_venda **=** **(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** **;**

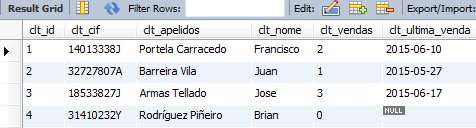
* Proba de funcionamento

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\_apelidos**,** clt\_nome**,** clt\_vendas**,** clt\_ultima\_venda

**from** tendabd**.**clientes**;**



**insert** **into** vendas **(**ven\_tenda**,** ven\_empregado**,** ven\_cliente**,** ven\_data**)**

**values** **(**1**,**1**,**1**,**now**());**

**select** clt\_id**,** clt\_cif**,** clt\_apelidos**,** clt\_nome**,** clt\_vendas**,** clt\_ultima\_venda

**from** tendabd**.**clientes**;**

u7a3_tareafa03_probas2.emf

-- 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\_apelidos**,** clt\_nome**,** clt\_vendas**,** clt\_ultima\_venda

**from** tendabd**.**clientes**;**

u7a3_tareafa03_probas3.emf

-- 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\_apelidos**,** clt\_nome**,** clt\_vendas**,** clt\_ultima\_venda

**from** tendabd**.**clientes**;**

u7a3_tareafa03_probas4.emf

* + 1. 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**)**

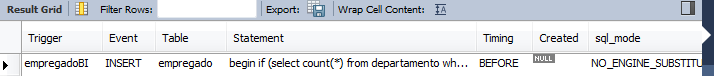
**)engine** **=** myisam**;**

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**;**



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**;**



* + 1. Tarefa 5. Crear disparadores para controlar as restricións referenciais en táboas non transacionais

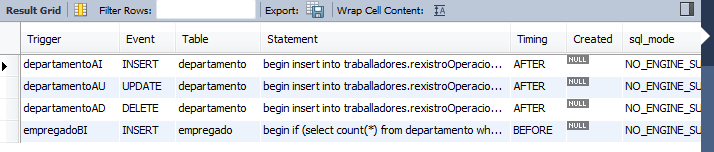
A tarefa consiste en simular o comportamento de borrado en cascada asociado á restrición de integridade referencial. Para iso, créase e próbase un disparador na base de datos *traballadores*, que faga que cada vez que se borre unha fila na táboa *departamento,* se borren as filas da táboa *empregado* 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 mesta 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**;**



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. Tarefa 6. Borrar disparadores

A tarefa consiste en borrar o disparador *practicas1.empregadoBI* creado na tarefa 1.

Solución

* Código de creación

**drop** **trigger** **if** **exists** practicas1**.**empregadoBI**;**

* + 1. 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 ventas 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 *hvendas* 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** hvendas **(**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\_venda**,** hdet\_numero**,** hdet\_artigo**,** hdet\_cantidade**,** hdet\_importe**)**

**select** dev\_venda**,** dev\_numero**,** dev\_artigo**,** dev\_cantidade**,**

**(**dev\_prezo\_unitario**\***dev\_cantidade**)\***dev\_desconto**/**10 # Cálculo de hdet\_importe

**from** detalle\_vendas

**where** dev\_venda **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\_venda **in** **(select** ven\_id **from** vendas **where** ven\_factura **is** **not** **null);**

**delete** **from** vendas

**where** ven\_factura **is** **not** **null;**

**end**

**//**

**delimiter** **;**

1. Materiais
   1. Documentos de apoio ou referencia

* HUESO IBAÑEZ, Luis. *Bases de datos*. Ciclos Formativos Ra-Ma, 2012.
* SANCHEZ, Jorge. *Manual de Gestión e bases de datos*.

<http://www.jorgesanchez.net/bd/sgbd.html>

* *Bases de datos. Máster oficial de Software libre*. UOC.

<http://ocw.uoc.edu/informatica-tecnologia-y-multimedia/bases-de-datos/Course_listing>

* *Manual de referencia de MySQL 5.6*. MySQL. <http://dev.mysql.com/doc/>
* DUBOIS, Paul. *La biblia de MySQL*. Anaya Multimedia, 2009.
* GROFF, James.*SQL.* McGraw-Hill.
* *Crear triggers en MySQL.* Hermosa Programacion.

<http://www.hermosaprogramacion.com/2014/06/mysql-trigger/>

* 1. Recursos didácticos
* Ordenadores con conexión a Internet, que terán instalado MySQL e MySQL Workbench.
* Material didáctico subministrado polo profesorado en papel e/ou formato dixital.
* Máquina virtual para exame que terá instalado MySQL e MySQL Workbench.
* Manual de referencia de MySQL.
* Proxector.
  1. Material auxiliar

O material auxiliar anexo a esta actividade está almacenado na carpeta CSIFC02\_MP0484\_V000703\_UD07\_A03\_DisparadorEvento\_Anexo que contén:

* O arquivo GuiaWorkbench.docx que é unha guía básica de MySQL Workbench 6.3.
* O arquivo V000703\_scriptsSQL.zip comprimido que contén os scripts necesarios para crear as bases de datos empregadas nos exemplos, tarefas e proba de avaliación da actividade.

1. Avaliación

Criterios de avaliación

|  |  |  |  |
| --- | --- | --- | --- |
| Criterios de avaliación seleccionados  para esta actividade | Tipo de  evidencia de aprendizaxe | Instrumento de avaliación | Peso na cualificación  da UD |
| * CA7.10 Definíronse disparadores.   CA 7.10.1 Creáronse e elimináronse disparadores. Creáronse, modificáronse e elimináronse eventos. Comprobáronse os resultados de disparadores e eventos. | * Exame en papel e en formato dixital. | * **TO.7** Documento de rexistro de creación de disparadores, planificación de eventos e comprobación da execución de disparadores e eventos. Escala de valores (observación indirecta). | 15 |

Modelo de proba para TO.7

Proponse unha proba de dous apartados de tipo práctico. A solución deberá ser entregada polo alumnado nun documento escrito e nun arquivo dixital.

* O documento escrito debe conter:
* Código do disparador ou evento solicitado.
* Sentenza ou sentenzas que permitan utilizar ou executar o código anterior.
* Imaxe capturada co resultado da execución do código pedido dende o cliente Workbench.
* O arquivo dixital debe conter:
* Arquivo .sql cos códigos solicitados.

A proba realizarase na base de datos *liga\_acb*. Subminístrase o arquivo *ligaAcb.sql* co script que permite crear esa base de datos. A estrutura da base de datos é a seguinte:

* Táboa *equipo*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome columna | Tipo | Null | Clave | Observacións |
| codigo | char(4) |  |  | Código do equipo. Para enlazar con outras táboas |
| nome\_club | varchar(80) |  |  | Nome do club ao que pertence o equipo |
| pavillon | varchar(70) |  |  | Nome do pavillón no que xogan os partidos como local |
| capacidade\_pavillon | smallint unsigned |  |  | Número de espectadores que entran no seu pavillón |
| web | varchar(100) |  |  | Url da web |
| nome\_equipo | varchar(80) |  |  | Nome oficial do equipo na liga |

* Táboa *xogador*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome columna | Tipo | Null | Clave | Observacións |
| codigo\_equipo | char(4) |  |  | Código do equipo no que xoga o xogador |
| numero | tinyint unsigned |  |  | Número de dorsal do xogador |
| nome | char(30) |  |  | Nome do xogador |
| posicion | char(1) |  |  | Posición na que xoga: B = base, P = pivot, ...... |
| nacionalidade | char(3) |  |  | Nacionalidade do xogador |
| ficha | char(3) |  |  | Tipo de ficha: EXT=estranxeiro EUR=europeo JFL=nacional |
| estatura | Decimal(3,2) |  |  | Estatura do xogador en metros |
| idade | tinyint unsigned |  |  | Idade do xogador |
| temporadas | tinyint |  |  | Número de tempadas que leva no equipo |

* Táboa *partido*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome columna | Tipo | Null | Clave | Observacións |
| id | smallint unsigned | N | **P** | Un identificador para cada partido. AUTOINCREMENTAL |
| xornada | smallint unsigned |  |  | Número da xornada na que se xoga o partido |
| equipo\_local | char(4) |  |  | Código do equipo que xoga como local |
| equipo\_visitante | char(4) |  |  | Código do equipo que xoga como visitante |
| puntos\_local | tinyint unsigned |  |  | Puntos conseguidos polo equipo local no partido |
| puntos\_visitante | tinyint unsigned |  |  | Puntos conseguidos polo equipo visitante no partido |

* Táboa *estatistica*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome columna | Tipo | Null | Clave | Observacións |
| equipo | char(4) |  |  | Código do equipo |
| xogados\_local | tinyint unsigned |  |  | Número de partidos xogados como local |
| xogados\_visitante | tinyint unsigned |  |  | Número de partidos xogados como visitante |
| ganados\_local | tinyint unsigned |  |  | Número de partidos ganados como local |
| ganados\_visitante | tinyint unsigned |  |  | Número de partidos ganados como visitante |
| puntos\_favor\_local | smallint |  |  | Número de puntos conseguidos xogando como local |
| puntos\_favor\_visitante | smallint |  |  | Número de puntos conseguidos xogando como visitante |
| puntos\_contra\_local | smallint |  |  | Número de puntos recibidos xogando como local |
| puntos\_contra\_visitante | smallint |  |  | Número de puntos recibidos xogando como visitante |

* Táboa *clasificacion*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Nome columna | Tipo | Null | Clave | Observacións |
| posicion | tinyint unsigned |  |  | Nº de orde que ocupa o equipo na clasificación |
| equipo | char(4) |  |  | Código do equipo |
| xogados | tinyint unsigned |  |  | Número de partidos xogados |
| ganados | tinyint unsigned |  |  | Número de partidos ganados |
| puntos\_favor | smallint |  |  | Número de puntos conseguidos en total |
| puntos\_contra | smallint |  |  | Número de puntos recibidos en total |

Proba para CA7.10

* 1.- Crear un disparador que actualice a táboa *estatística* cada vez que se insira unha fila na táboa *partido*. Facer unha proba de funcionamento.
* Para o equipo que xoga como local, a actualización consistirá en sumarlle os valores que correspondan ás columnas: *jugados\_local*, *ganados\_local* (no caso que gane o equipo local)*,* *puntos\_favor\_local* e *puntos\_contra\_local.*
* Para o equipo que xoga como visitante, a actualización consistirá en sumarlle os valores que correspondan ás columnas: *jugados\_visitante, ganados\_visitante* (no caso que gane o visitante)*, puntos\_favor\_visitante* e *puntos\_contra\_visitante.*
* 2.- Planificar un evento que actualice a táboa *clasificacion* o día de remate da liga (7 de maio de 2016), utilizando a información da táboa *estatisticos*. A actualización empezará borrando as filas que teña a táboa *clasificacion*, e continuará inserindo nela información dos equipos ordenados polo número de partidos ganados (de maior a menor). No caso de empate, ordenarase pola diferenza entre puntos a favor e puntos en contra (de maior a menor). No caso de novo empate, non se seguirá ningún novo criterio para ordenar.

Exemplo de solución para entregar en papel

* Solución ao apartado 1
* Código do disparador

**drop** **trigger** **if** **exists** liga\_acb**.**partido\_insert**;**

delimiter **//**

**create** **trigger** liga\_acb**.**partido\_insert **after** **insert** **on** liga\_acb**.**partido

**for** **each** **row**

**begin**

**declare** vLocal**,** vVisitante **tinyint** **default** 0**;**

**if** **new.**puntos\_local **>** **new.**puntos\_visitante **then**

**set** vLocal **=** 1**;**

**else**

**set** vVisitante **=** 1**;**

**end** **if;**

**update** liga\_acb**.**estatistica

**set** xogados\_local **=** xogados\_local**+**1**,**

ganados\_local **=** ganados\_local**+** vLocal**,**

puntos\_favor\_local **=** puntos\_favor\_local**+new.**puntos\_local**,**

puntos\_contra\_local **=** puntos\_contra\_local**+new.**puntos\_visitante

**where** equipo **=** **new.**equipo\_local**;**

**update** liga\_acb**.**estatistica

**set** xogados\_visitante **=** xogados\_visitante**+**1**,**

ganados\_visitante **=** ganados\_visitante**+** vVisitante**,**

puntos\_favor\_visitante **=** puntos\_favor\_visitante**+new.**puntos\_visitante**,**

puntos\_contra\_visitante **=** puntos\_contra\_visitante**+new.**puntos\_local

**where** equipo **=** **new.**equipo\_visitante**;**

**end**

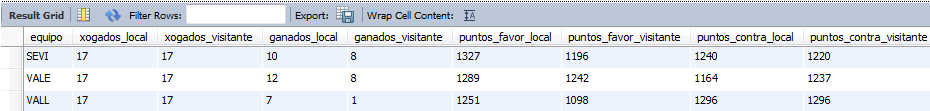
**//**

delimiter **;**

* Proba de funcionamento

Información da táboa *estatisticos* antes da inserción dun novo partido

**select** **\*** **from** estatistica**;**



Inserción do novo partido

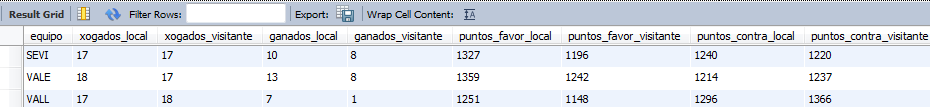
**insert** **into**

partido **(**xornada**,** equipo\_local**,** equipo\_visitante**,** puntos\_local**,** puntos\_visitante**)**

**values** **(**35**,**'VALE'**,** 'VALL'**,**70**,**50**);**

Información da táboa *estatisticos* despois da inserción do novo partido

**select** **\*** **from** estatistica**;**



* Solución ao apartado 2
* Código correspondente á planificación do evento

-- habilitar o planificador de eventos

**set** **global** event\_scheduler **=** **on;**

-- crear o evento

**drop** event **if** **exists** liga\_acb**.**obter\_clasificacion**;**

delimiter **//**

**create** event liga\_acb**.**obter\_clasificacion

**on** schedule **at** '2016-05-07 00:00:00'

**do**

**begin**

/\*Borrado de todas as filas da táboa clasificacion\*/

**delete** **from** liga\_acb**.**clasificacion**;**

/\*inserción de filas na táboa clasificacion \*/

**set** @posicion**=**0**;**

**insert** **into** liga\_acb**.**clasificacion **(**posicion**,** equipo**,** xogados**,** ganados**,** puntos\_favor**,** puntos\_contra**)**

**select**

@posicion**:=**@posicion**+**1**,**

equipo**,**

xogados\_local**+**xogados\_visitante**,**

ganados\_local**+**ganados\_visitante **as** ganados**,**

puntos\_favor\_local**+**puntos\_favor\_visitante **as** puntos\_favor**,**

puntos\_contra\_local**+**puntos\_contra\_visitante **as** puntos\_contra

**from** liga\_acb**.**estatistica

**order** **by** ganados **desc,** puntos\_favor **-** puntos\_contra **desc;**

**end**

**//**

delimiter **;**

Exemplo de lista de valoración para TO.7

Proponse a seguinte lista de valoración para o instrumento de avaliación TO.7 (EV) (observación indirecta: documento de rexistro de creación de disparadores, planificación de eventos e comprobación da execución de disparadores e eventos).

| Nome | Data | |
| --- | --- | --- |
| Indicadores de  CA 7.10.1 Creáronse e elimináronse disparadores. Creáronse, modificáronse e elimináronse eventos. Comprobáronse os resultados de disparadores e eventos. | Valoración máxima  15 | Cualificación |
| Escribiuse o código de creación do disparador proposto. | 7 |  |
| Escribíronse as sentenzas que farán que o disparador se execute e mostrouse o funcionamento do disparador mediante imaxes capturadas en Workbench. | 4 |  |
| Escribiuse o código de planificación do evento proposto. | 4 |  |