

DEFINICIÓN DE DESENCADENADORES, DISPARADORES O TRIGGERS.

El lenguaje SQL3 estandariza los disparadores (*triggers*), que ya incluyen prácticamente todos los productos relacionales, lo que permite mejorar considerablemente la integridad semántica de la base de datos. Esto es lo que hace que se llamen Bases de datos activas.

Podemos definir **SGBD activo** como aquel que, cuando se producen ciertas condiciones, ejecuta de forma automática, sin la intervención del usuario, las acciones especificadas de antemano en la fase de definición de la base de datos.

Las reglas de un SGBD activo se denominan ECA porque constan de Evento, Condición y Acción: cuando ocurre el **evento** se evalúa la **condición** y si ésta se satisface se ejecuta la **acción**. Por ejemplo, ante la subida de un valor bursátil (evento), si este incremento es superior a un veinte por ciento (condición), emitir órdenes de venta de los títulos (acción).

Las principales ventajas de los SGBD activos son:

- . Mayor productividad: se simplifican los programas al descargarlos de características que corresponden a la semántica de los datos.
- . Mejor mantenimiento: las reglas se almacenan una sola vez en el diccionario y, por tanto, en caso de precisarse una modificación sólo hay que cambiarlas en un sitio.
- . Reutilización de código: al tener parte de los procesos implementadas como reglas facilita su reutilización.
- . Reducción del tráfico de mensajes en las redes de ordenadores: al tener más procedimientos almacenados en los "servidores de datos", se optimiza la comunicación con los clientes y/u otros servidores.
- . Mayor seguridad: los usuarios podrán acceder a través de lenguajes de cuarta generación o de otro tipo sin atentar contra la integridad de la base de datos, puesto que esta integridad se encuentra preservada por el propio SGBD.

CREACIÓN DE TRIGGERS EN MySQL

Una de las principales características que se incorporaron a partir de la versión 5.0 de MySQL fueron los "Triggers". Los triggers son objetos relacionados a tablas que son ejecutados cuando sucede algún evento que afecte a sus tablas asociadas. Estos eventos son aquellas sentencias (INSERT, DELETE, UPDATE) que modifican los datos dentro de la tabla a la que está asociado el trigger y pueden ser disparados, ejecutados, antes (BEFORE) y/o después (AFTER) de que la fila sea modificada.

Los triggers son muy parecidos a los procedimientos almacenados, de tal forma que si deseamos ejecutar múltiples acciones cuando un trigger es disparado, podemos encapsular estas acciones dentro de una construcción BEGIN, END. Los triggers tienen un par de palabras claves extras - OLD y NEW - las cuales se refieren respectivamente a los valores de las columnas antes y después de que la sentencia fue procesada. Las sentencias INSERT únicamente permiten NEW, las sentencias UPDATE permiten ambos, NEW y OLD, y las sentencias DELETE permiten sólo OLD. La razón para esto debe ser obvia.

Sintaxis:

```
CREATE TRIGGER <nombre> [BEFORE|AFTER] [INSERT|UPDATE|DELETE]
ON <tabla> FOR EACH ROW
BEGIN
... <sentencias a ejecutar>
END
```

Para mostrar los triggers de una base de datos desde una sentencia SQL, se puede ejecutar alguno de los siguientes comandos:

- `SELECT * FROM INFORMATION_SCHEMA.TRIGGERS`
- `SHOW TRIGGERS LIKE '%';`

PRÁCTICA 1 TRIGGERS MySQL

Construye el siguiente esquema:

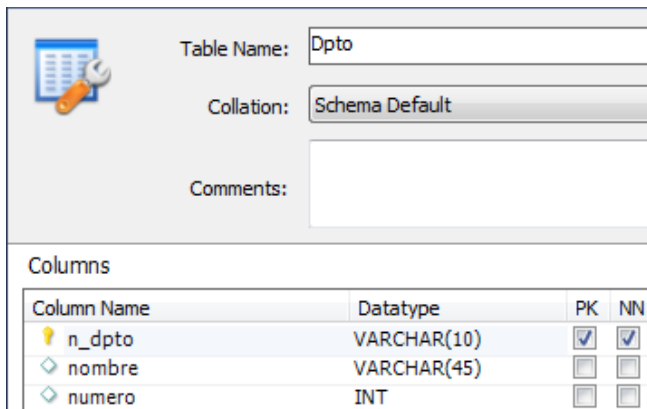


Table Name: Dpto

Collation: Schema Default

Comments:

Column Name	Datatype	PK	NN
n_dpto	VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nombre	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>
numero	INT	<input type="checkbox"/>	<input type="checkbox"/>

Se introducirán valores en esta tabla teniendo en cuenta que el campo número se deberá poner a 0. Este campo se refiere al número de empleados del departamento y se le irá sumando o restando 1, mediante un trigger, según se inserte o borre un empleado, también se modificará cuando se cambie a un empleado de departamento.

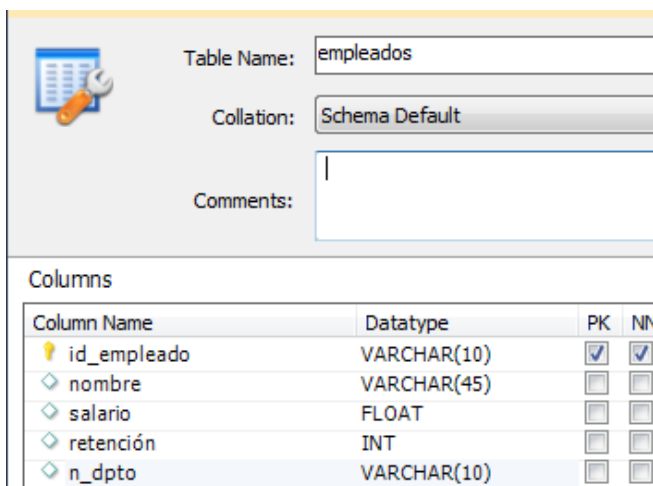


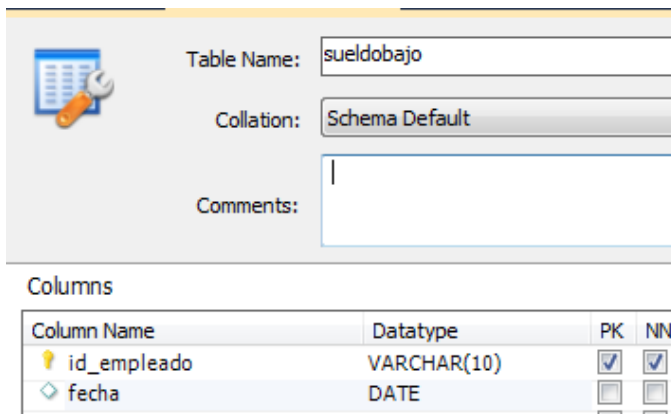
Table Name: empleados

Collation: Schema Default

Comments:

Column Name	Datatype	PK	NN
id_empleado	VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
nombre	VARCHAR(45)	<input type="checkbox"/>	<input type="checkbox"/>
salario	FLOAT	<input type="checkbox"/>	<input type="checkbox"/>
retención	INT	<input type="checkbox"/>	<input type="checkbox"/>
n_dpto	VARCHAR(10)	<input type="checkbox"/>	<input type="checkbox"/>

Antes de introducir valores en esta tabla se deberán crear los triggers que se describen más adelante.



The screenshot shows the MySQL Workbench interface for creating or editing a table named 'sueldobajo'. The 'Table Name' field is set to 'sueldobajo', the 'Collation' is 'Schema Default', and the 'Comments' field is empty. Below this, the 'Columns' section displays a table with the following structure:

Column Name	Datatype	PK	NN
id_empleado	VARCHAR(10)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
fecha	DATE	<input type="checkbox"/>	<input type="checkbox"/>

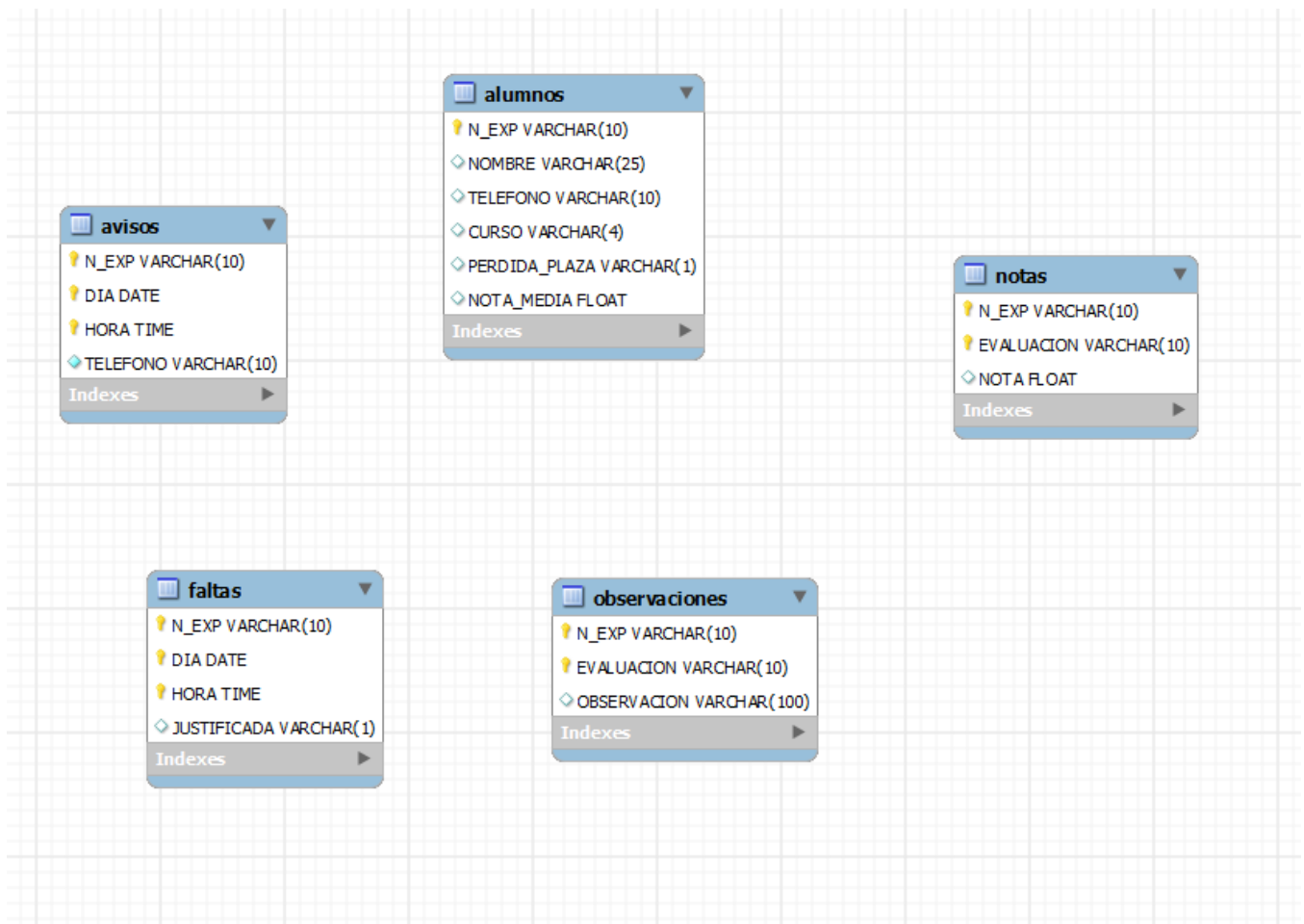
En esta tabla no se debe introducir ningún valor puesto que se mantendrá mediante triggers.

1. Controlar que cuando se produce una inserción sobre la tabla empleados, se actualice de forma conveniente la tabla Dpto, incrementándose el número de empleados.
2. Realizar una regla para controlar el borrado.
3. Realiza una regla para controlar cuando a un empleado se le cambie de departamento.
4. Regla en la cual si se inserta un trabajador en la tabla **EMPLEADOS**, con un sueldo mayor de 3000 euros y una retención menor a 20, se incremente la retención en 5.
5. Regla en la cual si al actualizar un sueldo en la tabla **EMPLEADOS**, si ese sueldo era menor a 3000 euros y después de la actualización es mayor a 3000 y tenía una retención menor a 20, se debe incrementar la retención en 5.
6. Antes de hacer este ejercicio debes borrar el trigger I_EMPLEADO del ejercicio 1 y a lo que ya realizaba en dicho ejercicio añadirle lo siguiente: Regla en la cual si al insertar un trabajador en la tabla **EMPLEADOS**, tiene un sueldo menor de 1000 euros, se debe insertar en la tabla **SueldoBajo** su id y la fecha actual.

7. Antes de hacer este ejercicio debes borrar el trigger **B_EMPLEADO** del ejercicio 2 y a lo que ya realizaba en dicho ejercicio añadirle lo siguiente: Regla en la cual si se borra un trabajador en la tabla **EMPLEADOS** que tenía un sueldo menor de 1000 euros, y se encontraba en la tabla **SueldoBajo** se debe borrar de dicha tabla.
8. Antes de hacer este ejercicio debes borrar el trigger **C_EMPLEADO** del ejercicio 3 y a lo que ya realizaba en dicho ejercicio añadirle lo siguiente: Regla que borre al trabajador de la tabla **SueldoBajo**, si se actualiza su sueldo con una cantidad superior a 1000 euros en la tabla **EMPLEADOS**.

PRÁCTICA 2 TRIGGERS MySQL

Disponemos de la siguiente base de datos:



Hacer un procedimiento almacenado para crear la base de datos y tablas.

Cuando se inserta un alumno, el campo *PERDIDA_PLAZA* tiene la letra 'N'.

1. Crea un disparador que controle que si se va a insertar una falta en la que el campo *JUSTIFICADA* es distinto de 'S' o 'N', se le ponga 'N' al contenido de dicho campo.

2. Crea un disparador que actúe después de realizar la inserción de un registro en la tabla *FALTAS*, haciendo:

- Insertar un registro en la tabla *AVISOS* si la falta es no justificada.

- Si el alumno tiene 50 o más faltas se actualice el campo *PERDIDA_PLAZA* de la tabla *ALUMNOS* poniendo la letra 'S'.

3. Crea un disparador que actúe después de realizar la actualización del campo *JUSTIFICADA* en un registro en la tabla *FALTAS*, poniéndole 'S' si estaba con el valor 'N', nos haga:

- Borrar el registro correspondiente en la tabla *AVISOS*.

- Se comprueba si el alumno tiene o supera las 50 faltas, y si no es así, se actualiza de nuevo el campo *PERDIDA_PLAZA* de la tabla *ALUMNOS*.

4. Realiza un trigger que actúe cuando se está insertando un registro en la tabla *NOTAS*, de tal manera que si la nota que se intenta insertar es menor que 0, registre un 0 y si es mayor que 10, registre un 1.

5. Una cuestión importante que hay que mencionar es que dentro de los triggers se pueden llamar procedimientos (*procedures*), otra característica que ya tiene MySQL desde la versión 5.0.10 en adelante.

(a) Crea un procedimiento que calcule la nota media de un alumno cuyo número de expediente entra de parámetro de entrada en el procedimiento.

(b) Crea un trigger que después de insertar una nota en la tabla *notas*, actualice la nota media de éste en la tabla *alumnos*. Deberá llamar al procedimiento creado en el punto anterior.

PRÁCTICA 3 TRIGGERS MySQL (BASE DE DATOS A_RURAL)

CONSTRUCCIÓN DE UN SISTEMA DE AUDITORÍA CON TRIGGERS

La auditoria de sistemas es un punto importante a considerar en sistemas cliente/servidor en los cuales existe una multitud de usuarios trabajando sobre una misma fuente de datos. Cuando dichos sistemas son pequeños y poseen pocos usuarios el problema podría parecer trivial, pero a medida que la envergadura del sistema crece y (sobre todo) la cantidad y variedad de usuarios es mayor, la auditoria se convierte en un punto difícil de manejar.

Con auditoria nos referimos al seguimiento de las operaciones realizadas por cada usuario, básicamente llevar un control de "qué se hace", "quién lo hace (el Usuario)", "dónde se hace", y "cuándo se hace".

Primero buscaremos donde almacenar la información, más allá de cómo hacerlo. El formato de la tabla puede variar dependiendo de las necesidades específicas del entorno, se propone la siguiente, a los efectos del desarrollo:

Tabla audita_maestro:

```
CREATE TABLE AUDITA_MAESTRO(  
ID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,  
TIPO_MODIFICACION VARCHAR( 1 ) ,  
NOMBRE_TABLA VARCHAR( 50 ) ,  
CODIGO_PER VARCHAR( 10 ) ,  
USUARIO VARCHAR( 50 ) ,  
FECHA DATE,  
TERMINAL VARCHAR( 50 )  
)
```

Tabla audita_detalle:

```
CREATE TABLE AUDITA_DETALLE(  
ID INTEGER NOT NULL PRIMARY KEY AUTO_INCREMENT,
```

```
NOMBRE_CAMPO VARCHAR( 50 ) ,
VALOR_NUEVO VARCHAR( 50 ) ,
VALOR_ANTERIOR VARCHAR( 50 )
)
```

Crear disparadores en cada tabla que se desee auditar. EN ESTE EJERCICIO HAY QUE HACERLO PARA LA TABLA PERSONAL DE LA BASE DE DATOS A_RURAL.

De este modo, cada tabla (que se desee auditar) tendrá un disparador (o trigger) que, cada vez que detecte una modificación en la tabla, ejecuta una serie de sentencias SQL previamente configuradas.

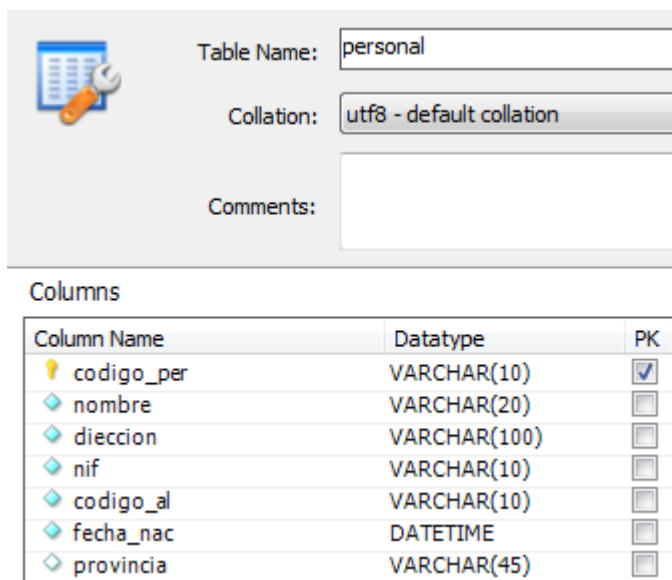
Para poder comprender correctamente las secciones siguientes, es necesario tener claros algunos conceptos de teoría de bases de datos y la sintaxis de algunas funciones que necesitamos utilizar:

Función USER(): devuelve el usuario.

Función CURDATE(): devuelve la fecha del sistema.

Función CONNECTION_ID(): Devuelve la identificación del sistema.

La tabla PERSONAL, presenta la siguiente estructura:



Column Name	Datatype	PK
codigo_per	VARCHAR(10)	<input checked="" type="checkbox"/>
nombre	VARCHAR(20)	<input type="checkbox"/>
direccion	VARCHAR(100)	<input type="checkbox"/>
nif	VARCHAR(10)	<input type="checkbox"/>
codigo_al	VARCHAR(10)	<input type="checkbox"/>
fecha_nac	DATETIME	<input type="checkbox"/>
provincia	VARCHAR(45)	<input type="checkbox"/>

Los disparadores deberán ser los siguientes:

1. Realiza un trigger que actúe antes de **insertar** un registro en la tabla personal, añadiendo los siguientes datos en la tabla AUDITA_MAESTRO (tipo_modificacion ('I'), nombre_tabla, CODIGO_PER, usuario(USER()), fecha (CURDATE()), terminal (CONNECTION_ID()))
2. Realiza un trigger que actúe antes de **actualizar** un registro en la tabla personal, añadiendo los siguientes datos en la tabla AUDITA_MAESTRO (tipo_modificacion ('U'), nombre_tabla, CODIGO_PER, usuario(USER()), fecha (CURDATE()), terminal (CONNECTION_ID())) y los siguientes en la tabla AUDITA_DETALLE (nombre_campo, valor_nuevo, valor_anterior). Tiene que servir para si se modifica el codigo_al, la dirección o la provincia.
3. Realiza un trigger que actúe antes de **borrar** un registro en la tabla personal, añadiendo los siguientes datos en la tabla AUDITA_MAESTRO (tipo_modificacion ('D'), nombre_tabla, CODIGO_PER, usuario(USER()), fecha (CURDATE()), terminal (CONNECTION_ID())).