

# Trigger

Sitio:

[Agencia de Aprendizaje a lo largo de la Vida](#)

Curso:

Administración de Base de Datos 1° F

Libro:

Trigger

Imprimido por:

MARIO DAVID GONZALEZ BENITEZ

Día:

domingo, 20 de octubre de 2024, 22:45

# Tabla de contenidos

- 1. ¿Qué es un trigger?
- 2. Crear un Trigger
- 3. New y Old
- 4. Trigger: Ejemplo 1
- 5. Trigger: ejemplo 2

## Introducción



Un **disparador** o *Trigger* es un **procedimiento almacenado** invocado automáticamente por el sistema gestor como reacción a cambios específicos en la Base de Datos.

Su definición es responsabilidad del **DBA** ( profesional responsable de los aspectos técnicos, tecnológicos, científicos, y legales de las bases de datos).

Las bases de datos que tienen asociado un conjunto de disparadores se llaman **Bases de datos activas**.

Cuando trabajamos con *trigger* debemos tener en cuenta el **evento**, la **condición**, la **acción** y el **momento**.

- El **evento** es un cambio en la base de datos que “activa” al disparador.
- La **condición** es una *query* que se ejecuta al activarse el disparador.
- La **acción** es un procedimiento que se invoca cuando el disparador se “activa” y se cumple su “condición”
- El **momento** es cuando se ejecuta la acción, si es “antes” o “después” del evento.



## Crear un trigger



Veamos la siguiente imagen

```
create trigger << nombre momento evento >>
on << tabla >> for each row << cuerpo >>
```



¿Qué significa cada << >> de la creación?

**Nombre:** Identifica al disparador, no puede existir dos nombres iguales para el mismo esquema, es decir, para la misma base de datos.

**Momento:** Indica cuándo se activa, puede ser *"before"*, se produce antes de realizar la operación que activa el disparador o *after*. Se produce después de realizar la operación.

**Evento:** Indica ante qué operación se activa ( *Insert, Update, Delete.*)

**Tabla:** La tabla involucrada debe ser permanente, es decir, pertenecer a la [base de datos](#). No puede ser temporal, ni una vista.

**Cuerpo:** Es la acción a realizar cuando se activa. Son sentencias [SQL](#), como las de los *Stored Procedures*.

La particularidad es que se lo puede eliminar con un simple *"drop trigger nombre"* o eliminando la tabla asociada él.

Empleando el constructor *begin ... end*, se puede definir un disparador que ejecute sentencias múltiples.

Dentro del bloque *begin*, también pueden utilizarse otras sintaxis permitidas en rutinas almacenadas, tales como condicionales y bucles.



Como sucede con las [rutinas almacenadas](#), cuando se [crea un disparador que ejecuta sentencias múltiples](#), se hace necesario redefinir el delimitador de sentencias, de forma que se pueda utilizar el [carácter ";"](#) dentro de la definición del disparador.



### ¿Para qué sirven?

Las palabras clave *new* y *old* permiten acceder a los campos en las filas afectadas por la acción que desencadena el *trigger*. *Old* y *new* son extensiones de MySQL para los disparadores.

Vimos que los eventos que dan pie al disparador son tres, veamos como intervienen estas palabras claves en ellos.

#### *Insert trigger:*

- *NEW* refiere al valor que ingresa a la tabla.

#### *Update trigger:*

- *OLD* refiere al valor anterior a la acción.
- *NEW* refiere al valor posterior.

#### *Delete trigger:*

- No hay *NEW*, sólo existe el *OLD*.

Una columna precedida por *old* es de sólo lectura. Es posible hacer referencia a ella, pero no modificarla.

Una columna precedida por *new* puede ser referenciada si se tiene el privilegio *select* sobre ella.

En un disparador *before*, también es posible cambiar su valor con “*Set new.nombre\_col = valor*” si se tiene el privilegio de *update* sobre ella. Esto significa que un disparador puede usarse para modificar los valores antes que se inserten en un nuevo registro o se empleen para actualizar uno existente.

En un disparador *before* el valor de *new* para una columna *auto\_increment* es 0, no el número secuencial que se generará en forma automática cuando el registro sea realmente insertado.

## Trigger: Ejemplo 1



Suponemos la siguiente situación problemática usando la base de datos **“Taller”** de la semana 7 que se encuentra en la bibliografía de consulta de esa semana.

Cuando se agrega una nueva tupla a la tabla **“presurep”**, que contiene los repuestos destinados al presupuesto, se debe actualizar el stock de ese repuesto. Se debe crear un **store procedure** con parámetros de entrada para el **insert** y desencadenar un proceso para la actualización.



**Nota:** asumimos que siempre hay stock suficiente.

Así lo podemos ver en la siguiente imagen:

```
delimiter //
```

```
create procedure ingreso(in presupuesto int, in repuesto int, in cantidad int)
/* ===== datos a ingresar NO PUEDEN LLAMARSE IGUAL que el atributo === */
begin
    insert into presurep values(presupuesto, repuesto, cantidad);
    select "se realizo la actualizacion del repuesto" as mensaje;
end //
```

```
/* =====
    El trigger puede ejecutarse ANTES o DESPUES
===== */
```

```
create trigger Actualiza before insert on presurep
for each row
begin
    update repuesto set stock = stock - new.cant where codrep = new.codrep;
end//
```

Se ejecuta el **INSERT** una ALERTA indica que hay un **Trigger Asociado**

Es **BEFORE** → primero ejecuta el trigger, luego el **insert**

Ejecuta las líneas posteriores

Contenido nuevo de esas columnas → Uso del **NEW**

Trigger: ejemplo 2

La lógica del *trigger* puede ser tan compleja como lo requiera el razonamiento para resolver la consigna.

Este ejemplo es totalmente diferente al anterior y la finalidad es mostrar que son independientes de los conceptos teóricos.



La base de datos a usar está en el archivo [BasePreparatoria.sql](#) y la resolución en SQL en el archivo [Semana12-EjemploTrigger2.sql](#)

Accedé a los archivos haciendo clic [acá](#)



Consigna:

Los alumnos de los últimos años de las escuelas se deben anotar en cursos destinados a su ingreso al mundo universitario.

La inscripción está informatizada y se deberá tener en cuenta el momento de la inscripción dado que pueden quedar en un listado de espera para el próximo trimestre de cursada.

Todas las inscripciones quedan almacenadas (asistan o no al trimestre en cuestión), para asignarlos automáticamente al curso entrante.



Se pide :

Ingresar la inscripción que hace cada alumno. Este ingreso debe desencadenar la lógica que se detalla a continuación:

- La estructura de Inscripción tiene los siguientes datos:

IdInsc	codCurso	Aspirante	Fecha	Estado
--------	----------	-----------	-------	--------

**Estado** → es nulo en el momento del “Insert” y cambia cuando se comprueba el cupo. **Aspirante** → nombre alumno

- Comprobar que hay cupo para el aspirante sabiendo que la estructura curso tiene un campo “cupo” que muestra la cantidad de asientos disponibles y que la estructura inscripción cuenta con un campo “estado” que es *True* cuando la inscripción fue satisfactoria.
- Cambiar el “Estado” de Inscripción a “*False*” si NO hay cupo o a “*True*” si SI hay cupo.
- Registrar el pago de la matrícula siempre que la inscripción sea satisfactoria.

En este caso el *trigger* debe ser *before*.