

## ¿Qué es un Trigger (disparador)?

Un trigger es un conjunto de instrucciones SQL que se ejecutan automáticamente en respuesta a ciertos eventos en una tabla. Los eventos pueden ser:

- **INSERT:** Cuando se inserta un nuevo registro.
- **UPDATE:** Cuando se actualiza un registro existente.
- **DELETE:** Cuando se elimina un registro.

### Tipos de Triggers

Los triggers pueden ejecutarse **antes** o **después** del evento que los activa:

- **BEFORE:** El trigger se ejecuta **antes** de que el evento ocurra.
- **AFTER:** El trigger se ejecuta **después** de que el evento ocurra.

### Sintaxis Básica

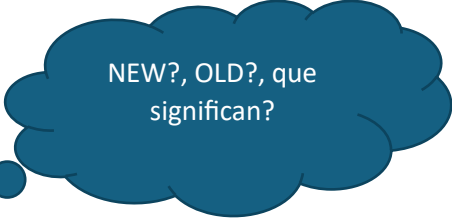
La sintaxis para crear un trigger en MySQL es la siguiente:

```
CREATE TRIGGER nombre_del_trigger
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON nombre_de_la_tabla
FOR EACH ROW //se ejecuta una vez p/c/fila afectada
BEGIN
    -- Instrucciones SQL
END;
```

### Ejemplo

Supongamos que tenemos una tabla llamada productos y queremos crear un **trigger** que actualice automáticamente el precio de los productos cada vez que se actualice su costo. El trigger se llamará *actualizarPrecioProducto* y se ejecutará **antes** de la actualización (**BEFORE UPDATE**).

```
DELIMITER $$
CREATE TRIGGER actualizarPrecioProducto
BEFORE UPDATE ON productos
FOR EACH ROW
BEGIN
    IF NEW.costo <> OLD.costo THEN
        SET NEW.precio = NEW.costo * 2;
    END IF;
END$$
DELIMITER ;
```



NEW?, OLD?, que significan?

### En este ejemplo:

- **NEW** se refiere a los valores nuevos que se están insertando o actualizando.
- **OLD** se refiere a los valores antiguos que están siendo reemplazados.

### ¿Qué significa **NEW**?

En el contexto de un **trigger**, **NEW** se utiliza para referirse a los nuevos valores que están siendo introducidos en una tabla como parte de la operación (en este caso, una actualización).

Pensemos en **NEW** como una representación de cómo quedará la fila después de que se complete la operación.

Por ejemplo, si estamos actualizando el campo **COSTO** de un producto en la tabla **PRODUCTOS**, el valor asociado a **NEW.COSTO** sería el nuevo valor que el usuario está ingresando. Es como el *futuro* de esa fila.

### ¿Qué significa **OLD**?

Por otro lado, **OLD** hace referencia a los valores que ya existen en la tabla antes de que se ejecute la operación. Es decir, representa el estado actual (o anterior) de los datos en la fila que está siendo modificada.

En el ejemplo de actualizar el campo **COSTO**, el valor de **OLD.COSTO** sería el valor antiguo de ese campo, antes de que ocurriera la actualización. Es como el *pasado* de esa fila.

### Usos Comunes de los Triggers

- **Auditoría:** Registrar cambios en los datos para mantener un historial de modificaciones.
- **Validación:** Asegurar que los datos cumplen con ciertas reglas antes de ser insertados o actualizados.
- **Automatización:** Realizar cálculos o actualizaciones automáticas basadas en cambios en los datos.

### Consideraciones

- Los triggers pueden afectar el rendimiento de la base de datos si no se usan con cuidado, ya que se ejecutan automáticamente y pueden añadir **carga adicional**.
- Es importante tener permisos adecuados para crear y gestionar triggers en la base de datos.

### Utilidades...

#### Ver Todos los Triggers en la Base de Datos Actual

Para listar todos los triggers en la base de datos actual, simplemente ejecutamos:

- SHOW TRIGGERS;

#### Ver Triggers en una Base de Datos Específica

Si queremos ver los triggers de una base de datos específica, usamos la siguiente sintaxis:

- SHOW TRIGGERS FROM nombre\_de\_la\_base\_de\_datos;

### Ver Triggers Asociados a una Tabla Específica

Para listar los triggers asociados a una tabla específica dentro de una base de datos, podemos usar una cláusula WHERE:

- `SHOW TRIGGERS FROM nombre_de_la_base_de_datos WHERE `Table` = 'nombre_de_la_tabla';`

### Ejemplo

Tenemos una base de datos llamada tienda y queremos ver los triggers asociados a la tabla productos. El comando sería:

- `SHOW TRIGGERS FROM tienda WHERE `Table` = 'productos';`

### Información Devuelta

El comando **SHOW TRIGGERS** devuelve una serie de columnas con información sobre cada trigger, incluyendo:

- **Trigger:** El nombre del trigger.
- **Event:** El evento que activa el trigger (INSERT, UPDATE, DELETE).
- **Table:** La tabla a la que pertenece el trigger.
- **Statement:** El cuerpo del trigger.
- **Timing:** El momento en que se activa el trigger (BEFORE o AFTER).
- **Created:** La fecha y hora de creación del trigger.
- **SQL Mode:** El modo SQL en el momento de la creación del trigger.
- **Definer:** El usuario que creó el trigger.

### ¿Recorrido de Registros?

El trigger no recorre todos los registros de la tabla. En su lugar, **se activa por cada fila que se ve afectada por el evento** (en este caso, una actualización). Esto es lo que significa la cláusula **FOR EACH ROW**. Cada vez que se actualiza una fila en la tabla productos, el trigger se ejecuta para esa fila específica.

### Importancia de FOR EACH ROW

La cláusula FOR EACH ROW es esencial en los triggers de MySQL porque indica que el trigger debe ejecutarse una vez por cada fila afectada por el evento. Sin esta cláusula, **el trigger no sabría cómo manejar los cambios a nivel de fila**.

### Ejecución del Trigger

En este ejemplo:

```
CREATE TRIGGER actualizarPrecioProducto
BEFORE UPDATE ON productos
FOR EACH ROW
BEGIN
    IF NEW.costo <> OLD.costo THEN
        SET NEW.precio = NEW.costo * 2;
    END IF;
END
```

el trigger se ejecuta **antes** de que se actualice una fila en la tabla productos (BEFORE UPDATE). Para cada fila que se actualiza:

1. Compara el valor nuevo (NEW.costo) con el valor antiguo (OLD.costo).
2. Si los valores son diferentes, actualiza el campo precio con el nuevo valor calculado (NEW.coste \* 2).

### ¿Cómo borrar o actualizar un trigger en MySQL?

- DROP TRIGGER IF EXISTS nombre\_del\_trigger;

### ¿Cómo actualizo un trigger?

MySQL, no se puede actualizar un trigger directamente. Si necesitamos modificar un trigger, debemos eliminarlo primero y luego crear uno nuevo con las modificaciones necesarias. Por ejemplo:

#### 1. Eliminar el trigger existente

```
DROP TRIGGER IF EXISTS nombre_del_trigger;
```

## 2. Crear un nuevo trigger con las modificaciones

```
CREATE TRIGGER nombre_del_trigger  
BEFORE/AFTER (INSERT/UPDATE/DELETE)  
ON nombre_de_la_tabla  
FOR EACH ROW  
BEGIN  
    -- código del trigger  
END;
```

### ¿Cuántas veces se ejecuta un trigger al actualizar un registro?

Si tenemos un trigger configurado para ejecutarse al actualizar un registro en una tabla (por ejemplo, un trigger AFTER UPDATE o BEFORE UPDATE), ese trigger se ejecutará **una vez por cada registro afectado** por la operación de UPDATE.

Esto significa que si la operación de UPDATE afecta a múltiples registros (por ejemplo, si usamos una cláusula WHERE que selecciona varios registros), el trigger se ejecutará una vez para cada registro que se actualice. Por otro lado, si solo se actualiza un registro, el trigger se ejecutará una sola vez.

En resumen, el trigger se ejecuta tantas veces como registros se actualicen en la operación UPDATE.

### Uso de NEW y OLD en un trigger de MySQL

Como ya comentamos, en MySQL, el uso de los alias NEW y OLD en un trigger depende del tipo de operación (INSERT, UPDATE, DELETE) para la cual el trigger está diseñado.

#### 1. Trigger INSERT:

- NEW: Está disponible y representa los valores nuevos que se están insertando en la tabla.
- OLD: No está disponible ya que no hay valores "viejos" en una operación de inserción.

#### 2. Trigger UPDATE:

- NEW: Está disponible y representa los valores nuevos que se están estableciendo en los registros actualizados.
- OLD: Está disponible y representa los valores anteriores, es decir, los valores antes de la actualización.

#### 3. Trigger DELETE:

- NEW: No está disponible, ya que no hay valores "nuevos" en una operación de eliminación.
- OLD: Está disponible y representa los valores que se están eliminando de la tabla.

### Resumen hasta aquí:

- NEW se usa para acceder a los valores que se están insertando o actualizando. Representa los nuevos valores de los registros que se están insertando o actualizando en una operación INSERT o UPDATE. Estos valores **se pueden modificar** dentro del trigger.
- OLD se usa para acceder a los valores antes de una actualización o para los valores que se están eliminando. Representa los valores antiguos de los registros antes de la operación UPDATE o DELETE. Estos valores son de **solo lectura** y no se pueden modificar.

### **Disponibilidad de NEW y OLD según la operación:**

- INSERT: Solo NEW está disponible.
- UPDATE: Tanto NEW como OLD están disponibles.
- DELETE: Solo OLD está disponible.

**Finalmente**, podríamos decir que **NEW** y **OLD** son referencias temporales a los datos de una fila específica durante la ejecución del trigger, pero no son tablas ni CTEs.

Ambas son accesibles solo mientras el trigger está ejecutándose, y su ámbito es limitado únicamente al código del trigger. No existen físicamente como tablas ni ocupan espacio en la base de datos; son conceptos internos manejados por el motor de MySQL para darnos acceso a esos valores.

No son registros permanentes ni almacenados en tu base de datos. Tampoco son tablas temporales ni CTEs ya que no podemos consultar NEW y OLD fuera del contexto del trigger.

### **Casos donde no necesitamos usar NEW o OLD**

Existen casos en los que podemos escribir un trigger sin necesidad de utilizar los alias NEW y OLD. Esto ocurre cuando las acciones que deseamos realizar dentro del trigger no dependen de los valores específicos de los registros afectados, sino de la operación en sí.

### **Ejemplos:**

#### **Ejemplo 1: Trigger AFTER INSERT sin usar NEW**

Supongamos que tenemos una tabla de auditoría donde queremos registrar simplemente que se ha realizado una inserción en una tabla, sin importar los valores específicos insertados.

```
CREATE TRIGGER after_insert_log
AFTER INSERT ON tablita
FOR EACH ROW
BEGIN
  INSERT INTO log_auditoria (acción_ejecutada, nombre_tabla, fecha_hora)
  VALUES ('INSERT', 'tablita', NOW());
END;
```

En este ejemplo, no necesitamos acceder a NEW porque solo nos interesa registrar que se ha realizado una inserción.

### Ejemplo 2: Trigger BEFORE DELETE sin usar OLD

Imaginemos que deseamos realizar una “limpieza” automática en otra tabla relacionada cuando se elimina un registro en la tabla principal. Por ejemplo, eliminar registros en una tabla de detalles (detalle\_pedido) cuando se elimina un registro en una tabla de pedidos (pedidos).

```
CREATE TRIGGER before_delete_limpieza  
BEFORE DELETE ON pedidos  
FOR EACH ROW  
BEGIN  
    DELETE FROM detalle_pedido WHERE id_pedido = OLD.id_pedido;  
END;
```

En este caso, aunque se utiliza OLD para identificar los detalles relacionados, es posible que en otros casos no necesitemos OLD si, por ejemplo, la “limpieza” no depende del valor específico del registro eliminado.

### Ejemplo 3: Trigger AFTER DELETE que borra registros en otra tabla

Si estamos eliminando registros en una tabla de historial cuando se elimina un registro en la tabla principal:

```
CREATE TRIGGER after_delete_cleanup  
AFTER DELETE ON employees  
FOR EACH ROW  
BEGIN  
    DELETE FROM employee_history WHERE employee_id = OLD.employee_id;  
END;
```

En este ejemplo, aunque se utiliza OLD, podríamos tener un trigger que realice una limpieza en otra tabla sin depender de los valores específicos, por ejemplo, eliminando registros más antiguos que una cierta fecha.

### Para ir cerrando...

- Triggers de Auditoría: Podemos registrar operaciones (INSERT, UPDATE, DELETE) sin acceder a NEW o OLD.
- Limpieza o Mantenimiento de Tablas: Podemos borrar o actualizar registros en otras tablas sin necesariamente usar NEW u OLD.

En general, la necesidad de usar **NEW** y **OLD** depende de si el trigger **necesita trabajar con los valores específicos de los registros afectados**. Si la lógica del trigger se basa únicamente en la operación que se está realizando y no en los valores de los datos, es posible que no necesitemos utilizar NEW ni OLD.