# Eventos en MySQL. Triggers After Update.

## Introducción a los disparadores de MySQL AFTER UPDATE

Los disparadores de MySQL AFTER UPDATE se invocan automáticamente después de que se produce un evento de actualización en la tabla asociada con los disparadores.
A continuación se muestra la sintaxis de la creación de un trigger de MySQL AFTER UPDATE:

```
CREATE TRIGGER trigger_name
AFTER UPDATE
ON table_name FOR EACH ROW
trigger_body
```

En esta sintaxis vemos:
Primero, el nombre del disparador que desea crear en la cláusula CREATE TRIGGER.
En segundo lugar, use la cláusula AFTER UPDATE para especificar el momento en que invocará el disparador.
En tercer lugar, el nombre de la tabla a la que pertenece el trigger después de la palabra clave ON.
Finalmente, el cuerpo del disparador que consiste en una o más declaraciones.
Si el cuerpo del disparador tiene más de una instrucción, debe usar el bloque BEGIN END. También necesita cambiar el delimitador predeterminado como se muestra en el siguiente código:

```
DELIMITER $$

CREATE TRIGGER trigger_name
    AFTER UPDATE
    ON table_name FOR EACH ROW
BEGIN
    -- statements
END$$
```

En un disparador AFTER UPDATE, se puede acceder a filas ANTIGUAS (OLD) y NUEVAS (NEW), pero no puede actualizarlas.

### Ejemplo de disparador AFTER UPDATE

Veamos un ejemplo de creación de un trigger AFTER UPDATE.

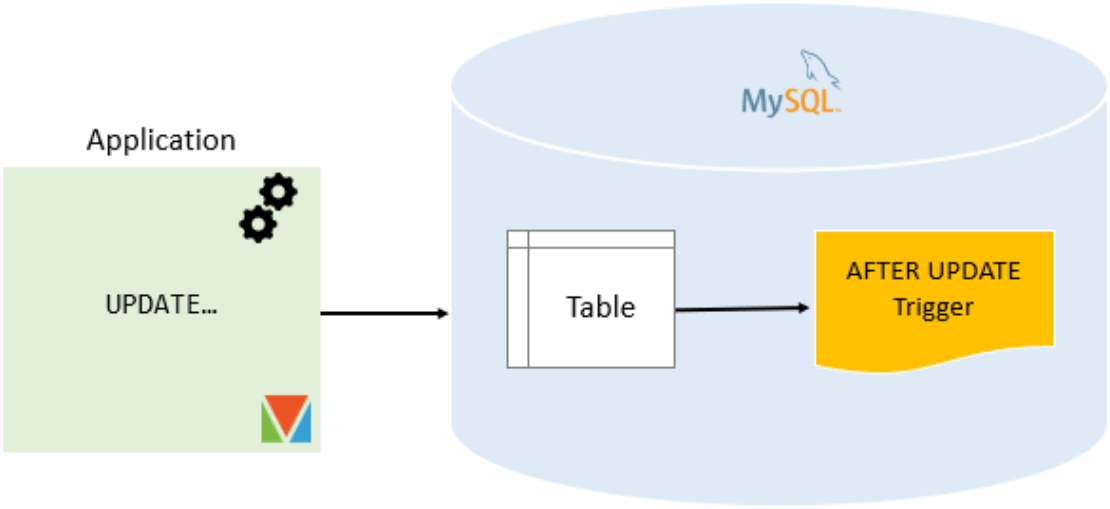


Figura 1: MySQL AFTER UPDATE Trigger

**Configurar una tabla de muestra**

Primero, cree una tabla llamada Sales(Ventas):

```
DROP TABLE IF EXISTS Sales;

CREATE TABLE Sales (
    id INT AUTO_INCREMENT,
    product VARCHAR(100) NOT NULL,
    quantity INT NOT NULL DEFAULT 0,
    fiscalYear SMALLINT NOT NULL,
    fiscalMonth TINYINT NOT NULL,
    CHECK(fiscalMonth >= 1 AND fiscalMonth <= 12),
    CHECK(fiscalYear BETWEEN 2000 and 2050),
    CHECK (quantity >=0),
    UNIQUE(product, fiscalYear, fiscalMonth),
    PRIMARY KEY(id)
);
```

En segundo lugar, inserte datos de muestra en la tabla Ventas:

```
INSERT INTO Sales(product, quantity, fiscalYear, fiscalMonth)
VALUES
    ('2001 Ferrari Enzo',140, 2021,1),
    ('1998 Chrysler Plymouth Prowler', 110,2021,1),
    ('1913 Ford Model T Speedster', 120,2021,1);
```

Tercero, consulte los datos de la tabla Ventas para mostrar su contenido:

```
SELECT * FROM Sales;
```

Finalmente, cree una tabla que almacene los cambios en la columna de cantidad (quantity) de la tabla de ventas(sales):

```
DROP TABLE IF EXISTS SalesChanges;

CREATE TABLE SalesChanges (
    id INT AUTO_INCREMENT PRIMARY KEY,
    salesId INT,
    beforeQuantity INT,
    afterQuantity INT,
    changedAt TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

**Ejemplo de creación de un disparador AFTER UPDATE**

La siguiente instrucción crea un trigger AFTER UPDATE en la tabla de ventas:

```
DELIMITER $$

CREATE TRIGGER after_sales_update
AFTER UPDATE
ON sales FOR EACH ROW
BEGIN
    IF OLD.quantity <> new.quantity THEN
        INSERT INTO SalesChanges(salesId,beforeQuantity, afterQuantity)
        VALUES(old.id, old.quantity, new.quantity);
    END IF;
END$$

DELIMITER ;
```

Este disparador after_sales_update se activa automáticamente después de que ocurra un evento de actualización para cada fila en la tabla de **sales**.
Si actualiza el valor en la columna **quantity** a un nuevo valor, el disparador inserta una nueva fila para registrar los cambios en la tabla SalesChanges.
Examinemos el disparador en detalle:
Primero, el nombre del desencadenante es after_sales_update especificado en la cláusula CREATE TRIGGER:

```
CREATE TRIGGER after_sales_update
```

En segundo lugar, el evento desencadenante es:

```
AFTER UPDATE
```

En tercer lugar, la tabla con la que está asociado el desencadenante es **sales**:

```
ON Sales FOR EACH ROW
```

Finalmente, usamos la instrucción IF-THEN dentro del cuerpo del disparador para verificar si el nuevo valor (new) no es el mismo que el anterior (old), luego insertamos los cambios en la tabla SalesChanges:

```
IF OLD.quantity <> new.quantity THEN
    INSERT INTO SalesChanges(salesId,beforeQuantity, afterQuantity)
    VALUES(old.id, old.quantity, new.quantity);
END IF;
```

**Probando el trigger MySQL AFTER UPDATE**

Primero, actualice la cantidad de la fila con id 1 a 350:

```
UPDATE Sales
SET quantity = 350
WHERE id = 1;
```

El trigger after_sales_update se invocó automáticamente.
En segundo lugar, consulta los datos de la tabla SalesChanges:

```
SELECT * FROM SalesChanges;
```

Tercero, aumente la cantidad de ventas de todas las filas en un 10 %:

```
UPDATE Sales
SET quantity = CAST(quantity * 1.1 AS UNSIGNED);
```

Cuarto, consultar datos de la tabla SalesChanges:

```
SELECT * FROM SalesChanges;
```

El disparador se activó tres veces debido a las actualización de las tres filas.