Eventos en MySQL. Triggers.

Triggers (disparador) en MySQL

En MySQL, un "disparador.[es] un programa almacenado invocado automáticamente en respuesta a un evento, como insertar, actualizar o eliminar que ocurre en la tabla asociada. Por ejemplo, puede definir un "disparador"que se invoca automáticamente antes de insertar una nueva fila en una tabla.
MySQL admite "disparadores"que se invocan en respuesta al evento INSERT, UPDATE o DELETE. El estándar SQL define dos tipos de "disparadores": disparadores de fila y disparadores de instrucción.

- Un disparador de fila se activa para cada fila que se inserta, actualiza o elimina. Por ejemplo, si una tabla tiene 100 filas insertadas, actualizadas o eliminadas, el disparador se invoca automáticamente 100 veces para las 100 filas afectadas.
- Un disparador "nivel de instrucción"se ejecuta una vez para cada transacción, independientemente de cuántas filas se inserten, actualicen o eliminen.

MySQL solo admite disparadores de nivel de fila. No admite disparadores de "nivel de sentencia(instrucción)".

Ventajas de los disparadores

- Los disparadores proporcionan otra forma de verificar la integridad de los datos.
- Los disparadores manejan los errores desde la capa de base de datos.
- Los disparadores brindan una forma alternativa de ejecutar tareas programadas. Al usar desencadenantes, no tiene que esperar a que se ejecuten los eventos programados porque los desencadenantes se invocan automáticamente antes o después de que se realice un cambio en los datos de una tabla.
- Los disparadores pueden ser útiles para auditar los cambios de datos en las tablas.

Desventajas de los disparadores

- Los disparadores sólo pueden proporcionar validaciones extendidas, no todas las validaciones. Para validaciones simples, puede usar las restricciones NOT NULL, UNIQUE, CHECK y FOREIGN KEY.

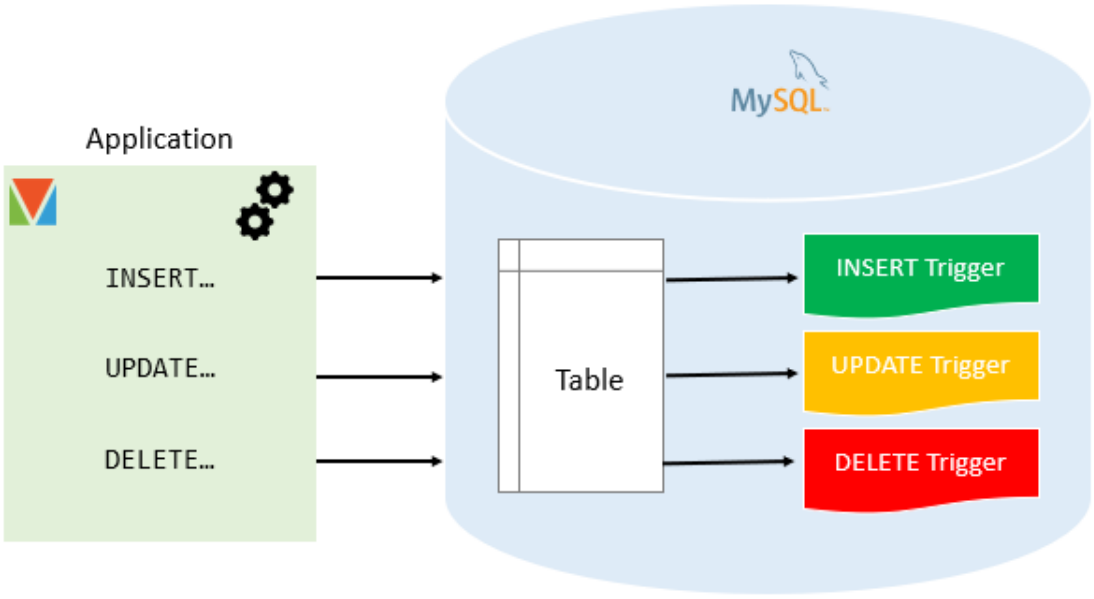

Figura 1: MySQL Triggers

- Los disparadores pueden ser difíciles de arreglar porque se ejecutan automáticamente en la base de datos, lo que puede no ser invisible para las aplicaciones cliente.
- Los disparadores pueden aumentar la sobrecarga del servidor MySQL.

Lo que vamos a ver

- Crear disparadores: los pasos para crear un disparador en MySQL.
- Desencadenar disparadores: cómo lanzar un disparador.
- Crear un trigger BEFORE INSERT: le muestra cómo crear un trigger BEFORE INSERT para mantener una tabla de resumen de otra tabla.
- Crear un disparador AFTER INSERT: describa cómo crear un disparador AFTER INSERT para insertar datos en una tabla después de insertar datos en otra tabla.
- Crear un trigger BEFORE UPDATE: aprenda a crear un trigger BEFORE UPDATE que valide los datos antes de actualizarlos a la tabla.
- Crear un disparador AFTER UPDATE: le muestra cómo crear un disparador AFTER UPDATE para registrar los cambios de datos en una tabla.
- Crear un disparador BEFORE DELETE: muestra cómo crear un disparador BEFORE DELETE.
- Crear un disparador AFTER DELETE: describe cómo crear un disparador AFTER DELETE.
- Cree múltiples disparadores para una tabla que tenga el mismo evento y tiempo de activación - MySQL 8.0 le permite definir múltiples disparadores para una tabla que tenga el mismo evento y tiempo de activación.

- Mostrar disparadores: enumera los disparadores en una base de datos, tabla por patrones específicos.

Introducción a la sentencia MySQL CREATE TRIGGER

La instrucción CREATE TRIGGER crea un nuevo disparador. Aquí está la sintaxis básica de la instrucción CREATE TRIGGER:

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE }
ON table_name FOR EACH ROW
trigger_body;
```

En esta sintaxis:

- Primero, especifique el nombre del desencadenante que desea crear, después de las palabras clave CREATE TRIGGER. Tenga en cuenta que el nombre del disparador debe ser **único** dentro de una base de datos.
- A continuación, especifique el momento de activación del trigger, que puede ser BEFORE o AFTER, lo que indica que el disparador se invoca antes o después de que se modifique cada fila.
- Luego, especifique la operación que activa el disparador, que puede ser INSERT, UPDATE o DELETE.
- Después de eso, especifique el nombre de la tabla a la que pertenece el desencadenador después de la palabra clave ON.
- Finalmente, especifique la instrucción que se ejecutará cuando se active el disparador. Si desea ejecutar varias declaraciones, use la declaración compuesta BEGIN END.

El cuerpo del disparador puede acceder a los valores de la columna afectada por la instrucción DML.
Para distinguir entre el valor de las columnas BEFORE y AFTER de que se haya activado el DML, utilice los modificadores NUEVO y ANTERIOR.
Por ejemplo, si actualiza la descripción de la columna, en el cuerpo del disparador, puede acceder al valor de la descripción antes de la actualización OLD.description y el nuevo valor NEW.description.
La siguiente tabla ilustra la disponibilidad de los modificadores VIEJO y NUEVO:

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Ejemplo de creación de un disparador

Primero, cree una nueva tabla llamada employee_audit para mantener los cambios en la tabla de empleados:

```
CREATE TABLE employees_audit (
    id INT AUTO_INCREMENT PRIMARY KEY,
    employeeNumber INT NOT NULL,
    lastname VARCHAR(50) NOT NULL,
    changedat DATETIME DEFAULT NULL,
    action VARCHAR(50) DEFAULT NULL
);
```

A continuación, cree un trigger BEFORE UPDATE que se invoque antes de realizar un cambio en la tabla de empleados.

```
CREATE TRIGGER before_employee_update
    BEFORE UPDATE ON employees
    FOR EACH ROW
 INSERT INTO employees_audit
 SET action = 'update',
     employeeNumber = OLD.employeeNumber,
     lastname = OLD.lastname,
     changedat = NOW();
```

Dentro del cuerpo del disparador, utilizamos la palabra clave OLD para acceder a los valores de las columnas employeeNumber y lastname de la fila afectada por el disparador.
Luego, mostramos todos los triggers en la base de datos actual utilizando la instrucción SHOW TRIGGERS:

```
SHOW TRIGGERS;
```

Además, si observas el esquema utilizando MySQL Workbench, bajo empleados>triggers, verás el disparador before_employee_update.
Después de eso, actualiza una fila en la tabla de empleados:

```
UPDATE employees
SET
    lastName = 'Phan'
WHERE
    employeeNumber = 1056;
```

Finalmente, consulta la tabla employee_audit para verificar si el disparador ha sido activado por la instrucción UPDATE:

```
SELECT * FROM employees_audit;
```

Como puede ver claramente en la salida, el desencadenador se invocó automáticamente e insertó una nueva fila en la tabla employee_audit.

Introducción a la sentencia MySQL DROP TRIGGER

La sentencia DROP TRIGGER borra un disparador de la base de datos. En su forma básica su sintaxis es la siguiente:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name;
```

En la sintaxis vemos:

- Primero, se especifica el nombre del disparador que desea eliminar, con las palabras clave DROP TRIGGER.
- Segundo, se especifica el nombre del esquema al que pertenece el disparador. Si omite el nombre del esquema, la instrucción eliminará el disparador en la base de datos actual.
- Tercero, se use la opción IF EXISTS para descartar el disparador existe. La cláusula IF EXISTS es opcional.

Si se intenta eliminar un disparador que no existe sin usar la cláusula IF EXISTS, MySQL emite un error. Sin embargo, si usa la cláusula IF EXISTS, MySQL emite una WARNING en su lugar.
DROP TRIGGER requiere el privilegio TRIGGER sobre la tabla asociada con el disparador.
Tenga en cuenta que si elimina una tabla, MySQL eliminará automáticamente todos los triggers asociados con la tabla.

Ejemplo de MySQL DROP TRIGGER

Primero, cree una tabla llamada facturación para demostración:

```
CREATE TABLE billings (
    billingNo INT AUTO_INCREMENT,
    customerNo INT,
    billingDate DATE,
    amount DEC(10 , 2 ),
    PRIMARY KEY (billingNo)
);
```

En segundo lugar, cree un nuevo trigger que se disparará BEFORE UPDATE, asociado a la tabla de facturación creada:

```
DELIMITER $$
CREATE TRIGGER before_billing_update
    BEFORE UPDATE
    ON billings FOR EACH ROW
BEGIN
    IF new.amount > old.amount * 10 THEN
        SIGNAL SQLSTATE '45000'
            SET MESSAGE_TEXT = 'New amount cannot be 10 times greater than the
                                current amount.';
    END IF;
END$$
DELIMITER ;
```

Si no estás familiarizado con la declaración DELIMITER, mira en los procedimientos almacenados. El disparador se activa antes de cualquier actualización. Si la nueva cantidad es 10 veces mayor que la cantidad actual, el disparador genera un error.
Tercero, mostramos los disparadores:

```
SHOW TRIGGERS;
```

Cuarto, eliminamos el trigger y mostramos los que quedan:

```
DROP TRIGGER before_billing_update;
SHOW TRIGGERS;
```