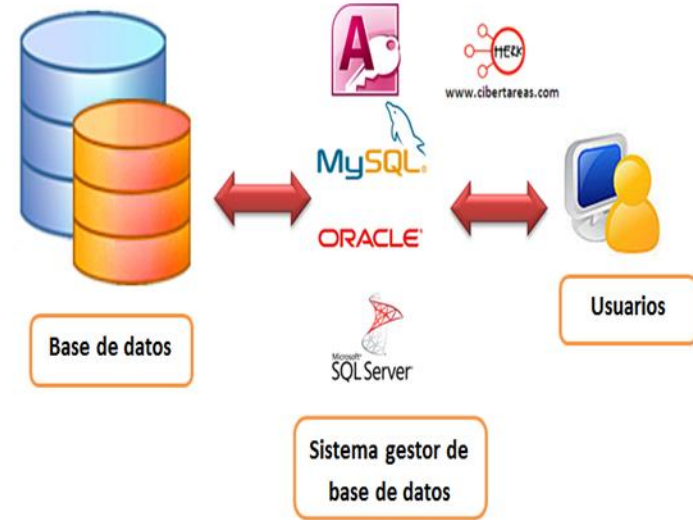


---

# Sesion 10

# Triggers



---

Maite Chirivella

2º Asix

---

# Indice

---

1.- Introducción

2.- Definición

3.- Ejercicios

# 1.- Introducción

---

- Un **trigger SQL** es un conjunto de sentencias SQL almacenadas en el catálogo de la base de datos .
  - Un **trigger SQL** se ejecuta cuando se produce un evento asociado a una tabla, por ejemplo, insertar, actualizar o borrar.
  - Un **trigger SQL** es un tipo especial de procedimiento almacenado.
  - La principal diferencia entre un trigger y un procedimiento es que un **trigger se llama automáticamente** cuando se realiza un evento de modificación de datos en una tabla, mientras que un **procedimiento se debe llamar de forma explícita (CALL)**.
-

# 1.- Introducción

---

- Un **trigger** es un conjunto de sentencias SQL que se invoca automáticamente cuando se realiza un cambio en los datos de la tabla asociada.
  - Un **trigger** se puede invocar ya sea **antes o después** de los datos se cambien por una operación **de Inserción actualización o borrado**.
  - Se pueden definir como máximo 6 triggers por tabla
    1. **BEFORE INSERT** - activado antes de introducir datos en la tabla.
    2. **AFTER INSERT** - activado después de que los datos se inserten.
    3. **BEFORE UPDATE** - activado antes de que se actualicen los datos.
    4. **AFTER UPDATE** - activado después de que los datos se actualicen.
    5. **BEFORE DELETE** - activado antes de borrar datos de la tabla.
    6. **AFTER DELETE** - activado después de eliminar los datos de la tabla.
-

# 1.- Introducción

---

- Se debe utilizar un nombre único para cada trigger asociado con una tabla.
- Se puede tener el mismo nombre de trigger definido para diferentes tablas a pesar de que no es una buena práctica.

1. Se puede utilizar esta nomenclatura:

**( BEFORE | AFTER )\_tableName\_( INSERT | UPDATE | DELETE )**

2. O esta otra nomenclatura.

**tablename\_( BEFORE | AFTER )\_( INSERT | UPDATE | DELETE )**

Por ejemplo, **pedidos\_before\_update** es un disparador invocado antes de actualizar un pedido. **Nosotros utilizaremos esta nomenclatura.**

---

# 1.- Introducción

---

MySQL almacena los triggers en un directorio de datos, por ejemplo, /data con los archivos llamados **tablename.TRG** y **triggername.TRN**

1. El archivo **tablename.TRG** asigna el trigger a la tabla .
2. El archivo **triggername.TRN** contiene la definición trigger.

Para hacer **copias de seguridad**:

1. Copiando los archivos de activación a la carpeta de copia de seguridad.
  2. Utilizando la herramienta **mysqldump**.
-

## 2.- Definición

---

- Para crear un trigger se utiliza la sentencia **Create Trigger**, que tiene la siguiente sintaxis:

```
1 CREATE TRIGGER trigger_name trigger_time trigger_event
2   ON table_name
3   FOR EACH ROW
4   BEGIN
5   ...
6   END ;
```

- El **nombre del disparador** debe seguir la convención de nombres
  - El tiempo de activación puede ser **BEFORE** (el trigger se ejecuta antes del evento) o **AFTER** (el trigger se ejecuta después del evento)
  - El evento puede ser **INSERT, UPDATE o DELETE**
-

## 3.- Ejemplos

---

- Crea una tabla alumne\_Auditoria

```
CREATE TABLE alumne_Auditoria(  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  NumExp VARCHAR(4) NOT NULL,  
  nom VARCHAR(10) NOT NULL ,  
  cognom1 VARCHAR(30) NOT NULL ,  
  cognom2 VARCHAR(30) NOT NULL ,  
  edat INT(10),  
  telefon VARCHAR(9),  
  pes FLOAT,  
  dataAct DATETIME DEFAULT NULL ,  
  accio SET ('i','u','d')  
);
```

---




## 3.- Ejemplos

---

1. Crea un trigger **AFTER INSERT** que crea un registro en la tabla de auditoria\_alumne cada vez que se **INSERTA en la tabla alumne**.

```
CREATE TRIGGER alumne_after_insert AFTER INSERT ON alumne
BEGIN
INSERT    INTO alumne_Auditoria (NumExp ,nom ,cognom1,cognom2,
                                edat ,telefon ,pes ,dataAct,accio )
VALUES (NEW.NumExp, NEW.nom, NEW.cognom1, NEW.cognom2, NEW.edat,
        NEW.telefon, NEW.pes, NOW(), 'i' ) ;
END
```



- Para comprobar que se ha ejecutado has de **insertar un alumno**

```
INSERT    INTO alumne VALUES ( '0010', 'Juan', 'Moscardo', 'Sanchez',
                                19, '666554221', 70 ) ;
SELECT * FROM alumne
SELECT * FROM alumne_Auditoria;
```

### 3.- Ejemplos

---

Las columnas de la tabla asociada con el disparador pueden referenciarse empleando los alias OLD y NEW.

**OLD.nombre\_col** hace referencia a una columna de una fila existente, antes de ser **actualizada o borrada**.

**NEW.nombre\_col** hace referencia a una columna en una nueva fila a punto de **ser insertada, o en una fila existente luego de que fue actualizada**.

En **insert** solo se puede usar **NEW** para indicar la nueva fila insertada.

En **delete** solo se puede usar **OLD** para indicar la fila borrada.

En **update** se puede usar **NEW** para indicar los nuevos campos actualizados **OLD** los campos con el valor antes de actualizar.


---

## 3.- Ejemplos

---

2. Crea un trigger **BEFORE UPDATE** que crea un registro en la tabla de auditoria\_alumne cada vez que se **ACTUALIZA la tabla alumne**.

```
CREATE TRIGGER alumne_before_update BEFORE UPDATE ON alumne
BEGIN
INSERT INTO alumne_Auditoria (NumExp ,nom ,cognom1,cognom2,
edat ,telefon ,pes ,dataAct ,accio )
VALUES (OLD.NumExp, OLD.nom, OLD.cognom1,OLD.cognom2, OLD.edat,
OLD.telefon, OLD.pes, NOW(), 'u' ) ;
END
```



- Para comprobar que se ha ejecutado has de **actualizar un alumno**


```
UPDATE alumne SET nom='Pepe' WHERE NumExp='0010';
SELECT * FROM alumne;
SELECT * FROM alumne_Auditoria;
```

## 3.- Ejemplos

---

3. Crea un trigger **AFTER DELETE** que crea un registro en la tabla de auditoria\_alumne cada vez que se **BORRA en la tabla alumne**.

```
CREATE TRIGGER alumne_after_delete AFTER DELETE ON alumne
BEGIN
INSERT    INTO alumne_Auditoria (NumExp ,nom ,cognom1,cognom2,
                                edat ,telefon ,pes ,dataAct, accio )
VALUES (OLD.NumExp, OLD.nom, OLD.cognom1, OLD.cognom2, OLD.edat,
        OLD.telefon, OLD.pes, NOW(), 'd') ;
END
```



- Para comprobar que se ha ejecutado has de **insertar un alumno**

```
DELETE FROM alumne WHERE NumExp='0010' ;
SELECT * FROM alumne;
SELECT * FROM alumne_Auditoria;
```

### 3.- Ejemplos

---

4. ¿Puedes insertar en tabla `alumne_auditoria`, antes de insertar en la tabla `alumne`?
  5. ¿Cuántos registros aparecen en la tabla `alumne_auditoria`?
  6. ¿Qué opción piensas que es la más adecuada?
  7. ¿Puedes insertar en tabla `alumne_auditoria`, después de borrar de la tabla `alumne`?
  8. ¿Por qué?
-

### 3.- Ejemplos

---

4. Crea un trigger **BEFORE INSERT** inserte los datos en la tabla en mayúsculas, (usando la función UPPER), y guardamos NULL en vez de la edad si la edad es menor o igual que 0.

```
CREATE TRIGGER alumne_before_insert BEFORE INSERT ON
    alumne
BEGIN
    BEGIN
        SET NEW.numExp= UPPER (NEW.numExp) ;
        SET NEW.nom=UPPER (NEW.nom) ;
        SET NEW.cognom1=UPPER (NEW.cognom1) ;
        SET NEW.cognom2=UPPER (NEW.cognom2) ;
        SET NEW.edat = IF (NEW.edat = 0, NULL, NEW.edat) ;
    END
END
```

---

## 3.- Ejemplos

---

- Para comprobar que se ha ejecutado has de **insertar un alumno**

```
INSERT INTO alumne VALUES ('0011', 'Ana',  
'Sanjuan', 'Navarro', 0, '666333555', 50 );
```

```
SELECT * FROM alumne;
```

---

### 3.- Ejemplo

---

5. Crea una tabla **asignatura\_log** para registro de modificaciones.

```
CREATE TABLE asignatura_log (  
  
    id int (11) NOT NULL AUTO_INCREMENT PRIMARY KEY,  
    asigOld varchar (4) DEFAULT NULL ,  
    asigNew varchar (4) DEFAULT NULL ,  
    data_act timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP  
    ON UPDATE CURRENT_TIMESTAMP,  
    usuari varchar (30) NOT NULL  
  
) ENGINE=InnoDB;
```

---



### 3.- Ejemplo

---

5. Crea un trigger **AFTER DELETE** que crea un registro en la tabla de asignatura\_log, con usuario y fecha actual cada vez que se **ACTUALIZA la tabla asignatura.**

**BEGIN**

```
DECLARE v_usuario varchar(50);
```

```
SELECT USER() INTO v_usuario;
```

```
INSERT INTO asignatura_log(asigOld, asigNew ,usuario  
) VALUES (old.codAsig, new.codAsig, v_usuario);
```

**END**

Para comprobar que se ha ejecutado has de **actualizar una asignatura**

```
update asignatura set codAsig='AD1'
```

```
where codAsig='ADA';
```

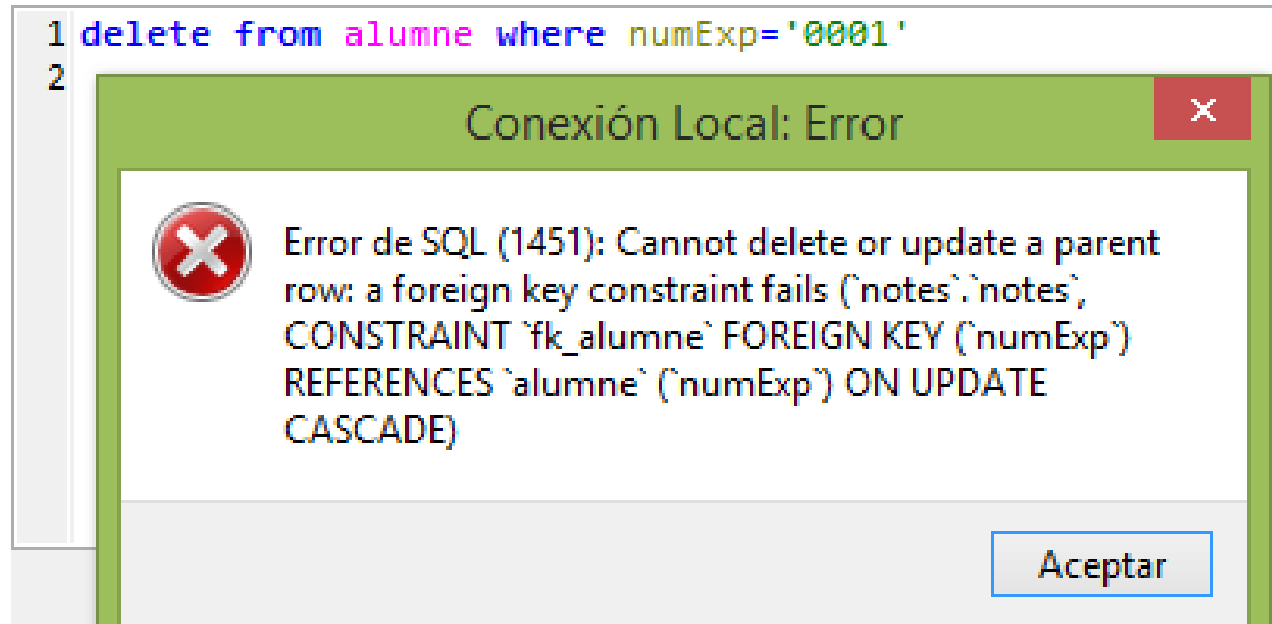
```
SELECT * FROM asignatura_log;
```

---

### 3.- Ejemplo

---

6. Comprueba que no puedes borrar un alumno si tiene notas asociadas.



### 3.- Ejemplo

---

6. Crea un trigger **BEFORE DELETE** que borra todas las notas de un alumno antes de borrar dicho alumno..

```
BEGIN
```

```
DELETE FROM notes WHERE numExp=OLD.numExp;
```

```
END
```

- Para comprobar que se ha ejecutado has de **borrar un alumno**

```
delete from alumne where numExp='0001';
```

```
select * from notes;
```

---