

# Bases de datos

Capítulo 6

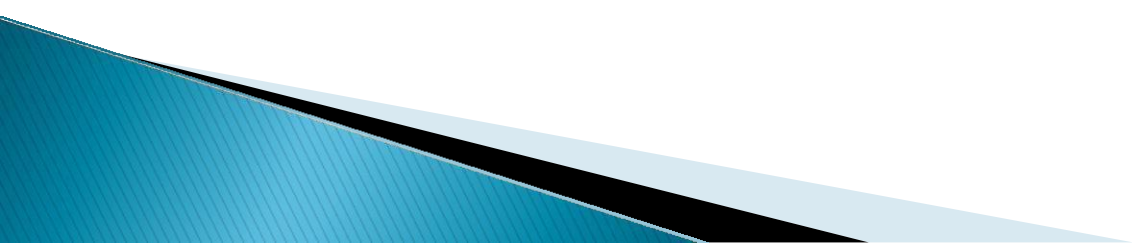
Disparadores y Eventos

# Disparadores (triggers)

Un disparador es un procedimiento almacenado que **se ejecuta automáticamente** cuando sobre una tabla se realiza alguna operación que implique modificar sus datos (DELETE, INSERT, UPDATE).

Por lo tanto, está asociado a una tabla y a un **evento** sobre la tabla.

Antes o después de que se ejecute ese evento, **por cada fila modificada** de la tabla, se ejecutará el conjunto de sentencias del disparador.



## Crear un disparador

La orden sql utilizada es CREATE TRIGGER

```
CREATE TRIGGER nombre_disp  
momento_disp evento_disp  
ON nombre_tabla  
FOR EACH ROW  
sentencias_disp
```

***nombre\_tabla***: no puede ser una tabla  
TEMPORARY ni una vista.

***momento\_disp*** : momento en que el disparador  
entra en acción.

**BEFORE** (antes de la sentencia que lo activa)

**AFTER** (despues de la sentencia que lo activa)

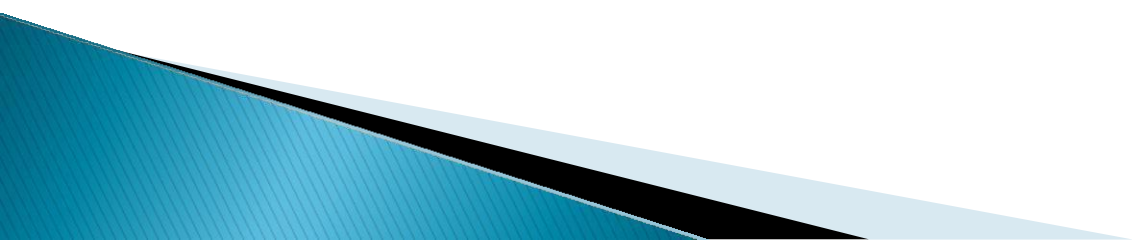
***evento\_disp*** : sentencia que activa al  
disparador ( **INSERT**, **UPDATE**, **DELETE**)

***sentencias\_disp*** : sentencias que se ejecutan  
cuando se activa el disparador.

BEGIN ... END si múltiples sentencias



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 después de ser actualizada.
- 

# Eliminar un disparador

## DROP TRIGGER

[nombre\_esquema.]nombre\_disp

Elimina un disparador. El nombre de esquema es opcional. Si el esquema se omite, el disparador se elimina en el esquema.

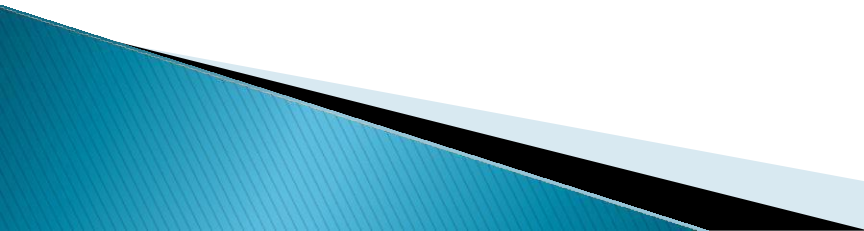
Ejemplo :

```
DROP TRIGGER mascota.dp_insetar;
```

# Base de datos de ejemplo

```
USE veterinario;
```

```
CREATE TABLE mascotas_ext (  
    nomb  VARCHAR(10),  
    carpeta VARCHAR(50) NOT NULL,  
    PRIMARY KEY(nomb)  
    FOREIGN KEY(nomb) REFERENCES mascotas(nombre)  
)  
ENGINE = InnoDB;
```



# Inserciones

```
INSERT INTO mascotas  
VALUES('Luna', '2019-12-08', 'Ana Vals');
```

```
INSERT INTO mascotas  
VALUES('Tintin', '2017-10-25', 'Pedro Lopez');
```

```
INSERT INTO mascotas_ext  
VALUES('Luna', 'ana vals_luna');
```

```
INSERT INTO mascotas_ext  
VALUES('Tintin', 'pedro lopez_tintin');
```





# Disparador de inserción

USE veterinario;

DELIMITER //

CREATE TRIGGER dp\_mascota\_insertar AFTER INSERT ON mascotas  
FOR EACH ROW

BEGIN

DECLARE nueva\_carpeta VARCHAR(50);

SET nueva\_carpeta = CONCAT( LOWER(NEW.propietario), '\_',  
LOWER(NEW.nombre) );

INSERT INTO mascotas\_ext

VALUES(NEW.nombre, nueva\_carpeta);

END //

DELIMITER ;

Inserción de una nueva mascota:

INSERT INTO mascotas

VALUES('Robin', '2020-05-16', 'Pepa Sanz', );

# Disparador de actualización

```
USE veterinario;
```

```
DELIMITER //
```

```
CREATE TRIGGER dp_mascota_actual AFTER UPDATE ON mascota  
FOR EACH ROW
```

```
BEGIN
```

```
    DECLARE nueva_carpeta VARCHAR(50);
```

```
    SET nueva_carpeta = CONCAT( LOWER(NEW.propietario),  
    '_', LOWER(NEW.nombre) );
```

```
    UPDATE mascota_ext SET carpeta = nueva_carpeta
```

```
    WHERE nomb = NEW.nombre;
```

```
END //
```

```
DELIMITER ;
```

Cambiar el nombre del propietario:

```
UPDATE mascotas SET propietario = 'Pepi Sanz'
```

```
WHERE nombre = 'Robin';
```

# Disparador de borrado

```
USE veterinario;
```

```
DELIMITER //
```

```
CREATE TRIGGER dp_mascota_eliminar BEFORE DELETE ON mascotas  
FOR EACH ROW
```

```
BEGIN
```

```
    DELETE FROM mascota_ext
```

```
    WHERE nomb= OLD.nombre;
```

```
END //
```

```
DELIMITER ;
```

Borrar una mascota:

```
DELETE FROM mascotas WHERE nombre = 'Robin';
```



# Eventos

En MySQL los eventos son tareas que se ejecutan de acuerdo a un horario ( eventos **programados.**)

También se conocen como ***triggers* temporales** ya que son similares, diferenciándose en que :

- el *trigger* se activa por un evento sobre la base de datos
- el evento se activa según una marca de tiempo.

# Planificador/scheduler

Antes de crear eventos hay que comprobar si el planificador de eventos (**scheduler**) está arrancado

La variable global **event\_scheduler** determina si el planificador de eventos está habilitado y en ejecución en el servidor. Puede tomar los valores:

- **ON** planificador activado
- **OFF** planificador parado
- **DISABLED** si queremos imposibilitar la activación en tiempo de ejecución.

Comando para comprobar si el programador de eventos está activo

***SHOW PROCESSLIST*** \G

Para activar el planificador hay que modificar el fichero de my.cnf de MySQL y reiniciar o bien modificar el valor de la variable con el siguiente comando:

***SET GLOBAL*** *event\_scheduler* = ON;



# Creación eventos

La orden sql utilizada es CREATE EVENT

**CREATE**

[DEFINER = { *user* | CURRENT\_USER } )

**EVENT**

[IF NOT EXISTS]

event name

**ON SCHEDULE** *schedule*

[ON COMPLETION [NOT] PRESERVE]

[ENABLE | DISABLE | DISABLE ONSLAVE]

[COMMENT '*comment*']

**DO**

event\_body;

## **schedule:**

*AT timestamp* [+ INTERVAL *interval* ] ...

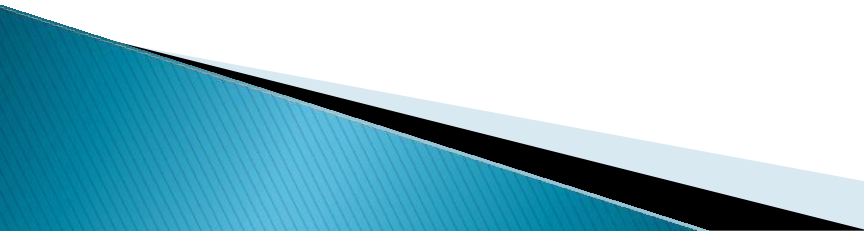
I EVERY interval

[STARTS *timestamp* [+ INTERVAL interval ] ...]


[ENDS *timestamp* [+ INTERVAL interval ] ...]

## **interval:**

*quantity* {YEAR I QUARTER I MONTH I DAY I HOUR  
I MINUTE I WEEK I SECOND I YEAR\_MONTH I  
DAY\_HOUR I DAY\_MINUTE I DAY SECOND I  
HOUR\_MINUTE I HOUR\_SECOND I  
MINUTE\_SECOND}





- **ON SCHEDULE:** permite establecer cómo y cuando se ejecutará el evento. Una sola vez, durante un intervalo, cada cierto tiempo o en una fecha hora de inicio y fin determinadas.
  - **DEFINER:** especifica el usuario cuyos permisos se tendrán en cuenta en la ejecución del evento.
  - **event\_body:** es el contenido o código del evento que se va a ejecutar. Consiste en una o varias instrucciones SQL dentro de un bloque *BEGIN/END*.
  - **COMPLETION:** permiten mantener el evento aunque haya expirado mientras que *DISABLE* permite crear el evento en estado inactivo.
  - **DISABLE ON SLAVE:** sirve para indicar que el evento se creó en el *master* de una replicación y que, por tanto, no se ejecutará en el esclavo.
- 

# Consulta y eliminación de eventos

**SHOW EVENTS** [{FROM | IN} *nombre\_bd*] [LIKE '*patrón*' | WHERE *expr*]

- Muestra información de eventos asociados a una base de datos filtrándolo con un patrón o cláusula *WHERE*.

**SHOW CREATE EVENT** *nombre\_evento*\G

- Muestra información sobre cómo se ha creado el evento

Cuando creamos *eventos* se crea un nuevo registro en la tabla *INFORMATION\_SCHEMA.EVENTS*.

Por eso, también podemos consultar los eventos almacenados con la siguiente sentencia:

```
SELECT *FROM information_schema.EVENTS\G;
```

Para **eliminar** un **evento** de utiliza el comando

```
DROP EVENT nombre_evento
```

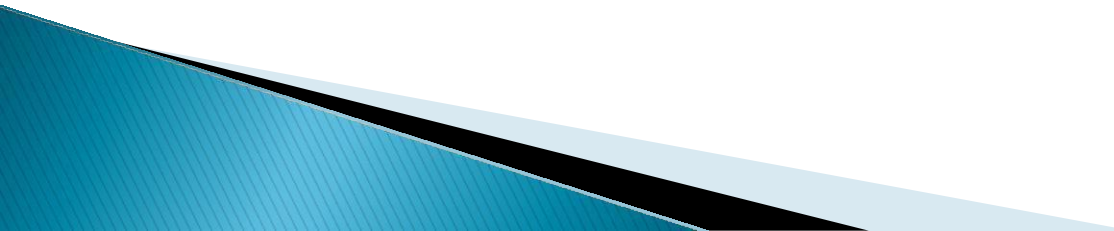


## Creación de la base de datos de prueba:

```
CREATE DATABASE bd_evento;  
USE bd_evento;  
CREATE TABLE prueba(  
    fecha_hora datetime  
);
```

## Creación del evento:

```
CREATE EVENT evento_prueba  
ON schedule every 1 MINUTE  
DO  
    INSERT INTO prueba VALUES (now());
```



Comprobación:

```
SELECT * FROM prueba;
```

Para ver si el evento está activo:

- SHOW EVENTS \G;
- SHOW CREATE EVENT evento\_prueba;
- SELECT \*  
FROM information\_schema.EVENTS;

# Modificación de eventos

## ALTER

[DEFINER = { user | CURRENT\_USER }]

**EVENT** *event\_name*

[ON SCHEDULE schedule]

[ON COMPLETION [NOT] PRESERVE]

[RENAME TO new\_event\_name]

[ENABLE | DISABLE | DISABLE ON SLAVE]

[COMMENT 'comment']

[DO event\_body]



# Ejemplos

- **Desactivar** un evento:

```
ALTER event evento_prueba disable;
```

Comprobación:

```
SHOW events\G;
```

- **Activar** el evento:

```
ALTER event evento_prueba enable;
```



- **Modificar la programación**

```
ALTER EVENT evento_prueba  
ON schedule every 30 second;
```

Comprobamos:

```
SELECT * FROM prueba;
```

- **Modificar el cuerpo del evento**

- ✓ Desactivamos el evento

```
ALTER EVENT evento_prueba disable;
```

- ✓ Creamos un procedimiento almacenado al que llamaremos desde el evento:



delimiter //

```
CREATE PROCEDURE inserta()  
BEGIN  
    INSERT INTO prueba VALUES(now());  
END //  
delimiter ;
```

✓ Modificamos el evento:

```
ALTER EVENT evento_prueba  
ON schedule every 1minute  
do  
    call inserta();
```