▼1.Triggers▼

Los triggers son objetos que puedes crear en tu base de datos. Estos permiten **desencadenar *un evento* de forma automática** en la tabla a la cúal están asociados. *En Java, por ejemplo, un evento es la acción que ocurre después de pulsar un botón.*

En nuestro caso, los triggers se pueden activar cuando:

- Realizamos un `INSERT` en una tabla.
- Realizamos un `DELETE` en una tabla.
- Realizamos un `UPDATE` en una tabla.

1.2 Casos de uso

Su principal uso es la realización de tareas de mantenimiento y administración de las BDD.

- **Copia de seguridad automática:** Puedes crear un trigger que se active después de cada inserción, actualización o eliminación de registros y realice una copia de la tabla en otro archivo o tabla.

- **Notificaciones:** Puedes crear un trigger que capture el evento de inserción en la tabla de empleados (por ejemplo) y envíe un correo electrónico a 'X'.
- **Control de stock:** Puedes crear un trigger que se active cada vez que se actualiza la cantidad de un producto y verifique si ha alcanzado 'X' límite. Cuando se alcance ese límite, el trigger puede desencadenar cualquier evento, como solicitar más unidades del producto.

1.3 Sintaxis

```
    DELIMITER $$
    CREATE TRIGGER trigger_1
    AFTER UPDATE  --> Cuando se ejecuta
    ON nombre_tabla FOR EACH ROW
    BEGIN
        *lo que hace trigger;*
    END$$

[BEFORE OR AFTER] --> Se desencadena antes o después de un INSERT, DELETE o
UPDATE.
[INSERT, UPDATE, DELETE] --> Con que
[FOR EACH ROW]: Indica que el trigger se repite por cada registro.
[FOR EACH STATEMENT]: Indica que el trigger se repite por cada sentencia SQL.
```

1. En MySQL es imprescindible usar `DELIMITER`.
2. Se declara el nombre del trigger.
3. Se establece cuando se va activar.
4. Se define la tabla en la que se va a ejecutar y las columnas afectadas.
5. Se describe la acción del trigger que siempre comenzará con `BEGIN` y acabará con `END`.

*La palabra reservada `DELIMITER` señala el principio y el final de **un bloque de instrucciones** puede ser cualquier signo que quieras establecer como: %%, ; $$...*

1.4 SENTENCIAS `OLD` && `NEW`

Permiten diferenciar y hacer referencia a los datos antiguos (OLD) y a los nuevos (NEW) dentro del trigger.

Ejemplo de trigger con `INSERT`

Ejemplo de `TRIGGER` para recopilar información de cada `nuevo fabricante` que se vaya a insertar en la tabla `fabricante`. Se guardará también quien lo hizo en la columna de `autor` y cuando lo hizo en la columna `fecha`.

Primero se crea la tabla auxiliar donde se van a volcar/guardar los datos

```
CREATE TABLE info_fabricante (
    nombre_f varchar (50),
    autor varchar(50),
    fecha date
);
```

Después se crea el trigger

```
DELIMITER $$
CREATE TRIGGER info_new_fabricantes_AI
AFTER INSERT
ON fabricante FOR EACH ROW
BEGIN
    INSERT INTO info_fabricante (nombre_f,autor,fecha)
    VALUES (NEW.nombre,CURRENT_USER(),NOW());
END$$
```

```
INSERT INTO `fabricante`(`nombre`) VALUES ('Apple');
```

nombre_f	autor	fecha
Apple	root@%	2023-06-12

Ejemplo de trigger con `UPDATE`

En este ejemplo se creará una tabla secundaria para almacenar los posibles cambios sobre el `nombre` y el `precio` de la tabla `producto`. Cada vez que alguien realice una sentencia `UPDATE` sobre la tabla productos se guardará el nombre y precio antiguos del producto, el nombre y precio nuevos, quién hizo los cambios y cuando.

Se crea la tabla auxiliar

```
CREATE TABLE info_producto (
    nombre_OLD varchar (50),
    precio_OLD double,
    nombre_NEW varchar (50),
    precio_NEW double,
    autor varchar(50),
    fecha date
);
```

Se crea el trigger

```
DELIMITER #
CREATE TRIGGER info_producto_BU
BEFORE UPDATE
ON producto FOR EACH ROW
BEGIN
    INSERT INTO info_producto
```

```
(nombre_OLD,precio_OLD,nombre_NEW,precio_NEW,autor,fecha)
    VALUES
(OLD.nombre,OLD.precio,NEW.nombre,NEW.precio,CURRENT_USER(),CURRENT_DATE());
END#
```

Se hace un update para comprobar que funciona

```
UPDATE producto SET precio=500 WHERE id_producto = 9;
```

nombre_OLD	precio_OLD	nombre_NEW	precio_NEW	autor	fecha
Portátil Ideapd 320	444	Portátil Ideapd 320	500	root@%	2023-06-12

1.5 Instrucciones adicionales

- `SHOW TRIGGERS`: muestra los triggers de la BDD.
- `DROP TRIGGER`nombre_trigger: elimina el trigger. *La instrucción `ALTER TRIGGER` no existe, si se quiere modificar un trigger se debe borrar y crear de nuevo.*

2. Estructuras de Control

En MySQL, existen varias estructuras de control que permiten realizar operaciones condicionales y repetitivas tanto en los triggers como en los procedimientos almacenados.

2.1 Sintaxis

{`Condicional IF`} :: Permite ejecutar una acción o un bloque si se cumple una condición. Por ejemplo:

```
IF condición THEN
    -- acciones a ejecutar si la condición es verdadera
ELSE
    -- acciones a ejecutar si la condición es falsa
END IF;
```

{`Bucle WHILE`} :: Ejecuta una acción repetidamente mientras se cumple una condición específica. Por ejemplo:

```
WHILE condición DO
    -- acciones a ejecutar mientras se cumpla la condición
END WHILE;
```

{`Bucle REPEAT`} :: Es el do while de Java. El bloque de código se ejecuta al menos una vez. Por ejemplo:

```
REPEAT
    -- acciones a ejecutar
UNTIL condición;
```

{`Bucle FOR`} :: Permite recorrer un rango de valores y ejecutar un bloque de código para cada valor. Por ejemplo:

```
FOR variable IN rango DO
    -- acciones a ejecutar para cada valor
END FOR;
```

3. ⚜ Procedimientos ⚜

Un procedimiento consiste en crear un objeto que pueda almacenar instrucciones `SQL`repetitivas. *En cristiano, un método de Java para consultas SQL.*

Permite crear una "consulta genérica" para las diferentes aplicaciones que usen la BDD ganando en eficiencia y seguridad. También puede servir para crear `TRIGGERS` más completos y complejos.

Un procedimiento se crea con la sentencia `CREATE PROCEDURE` y se invoca con `CALL`. Además, al igual que los métodos de Java puede tener 0, 1 o varios parámetros.

3.1 Parámetros de entrada, salida y entrada/salida

En los procedimientos podemos tener tres tipos de palabras clave que definimos antes del párametro:

{`IN`} :: Parámetros que no cambian su valor. Se considera paso por valor. {`OUT`} :: Estos parámetros pueden cambiar su valor dentro del procedimiento. En programación sería equivalente al paso por referencia. {`INOUT`} :: Combinación de los tipos IN y OUT que nos permite modificar la variable a nuestro antojo.

3.2 Sintaxis

```
DELIMITER #
CREATE PROCEDURE nombre_metodo(párametro?)
BEGIN
    *lo que hace el metodo;*
END#
```

- Para asignar una nueva variable usaremos: `DECLARE nombre_var`.
- Para asignar un valor a una variable usaremos podemos usar `SET nombre_var` o `TIPO_DATO DEFAULT valor`.

3.3 Ejemplo de creación de una variable con `DEFAULT`

```
CREATE PROCEDURE ejemplo_txt()
    BEGIN
        DECLARE var INT DEFAULT 12345;
    END
```

3.4 Ejemplo de procedimiento

En el siguiente ejemplo se crea un procedimiento que permite actualizar el precio de un producto determinado. Recibe por `parametro` el `precio nuevo` y la `ID` del producto que se quiere modificar.

Además, se estableció una condición para que, en caso de que se introduzca un precio negativo, se haga una consulta interna y se re-asigne la variable al precio original del producto.

```
DELIMITER $$
CREATE PROCEDURE actualizarPrecio (INOUT precioPrd double, id int)
BEGIN
    IF precioPrd < 0 THEN
            SET precioPrd = (SELECT precio FROM producto WHERE id_producto = id);
    END IF;

    UPDATE producto SET precio = precioPrd WHERE id_producto = id; --el return de
SQL
END$$
```

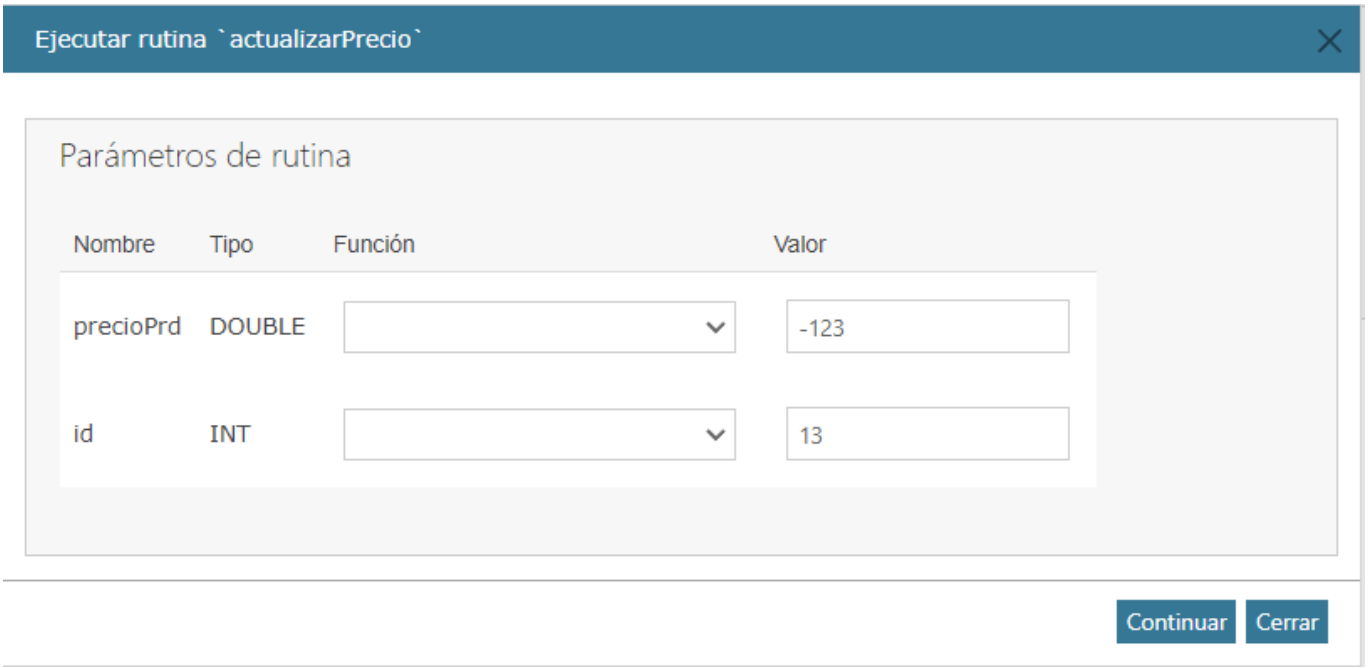
Llamamos al procedimiento...

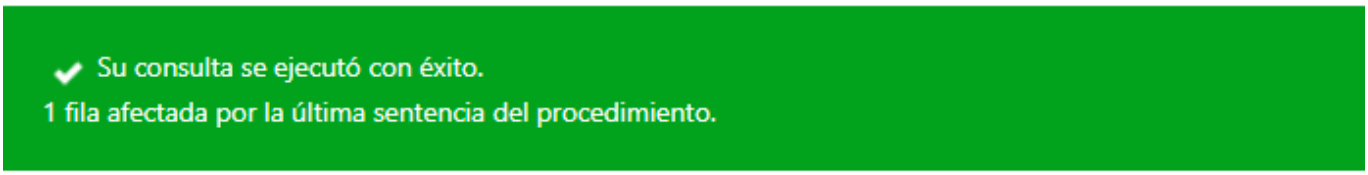
```
CALL actualizarPrecio(nuevoPrecio,id);
```

PRECIO DEL `IPHONE X` ANTES DE LA MODIFICACIÓN:

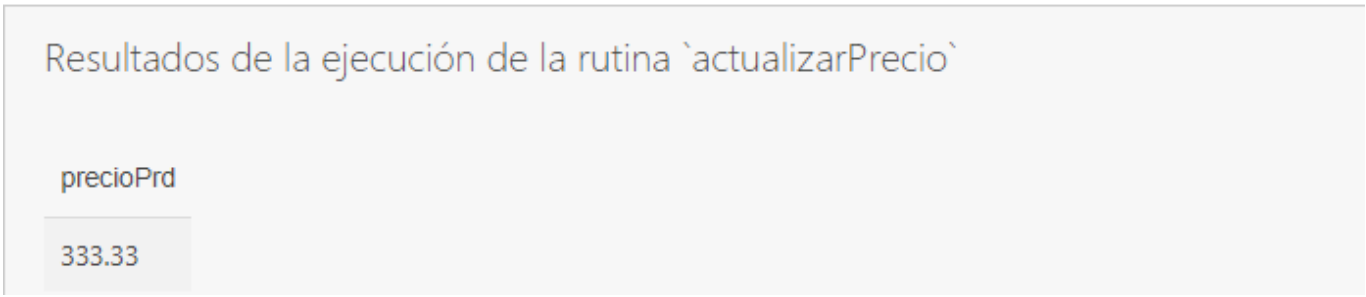
☐	Editar	Copiar	Borrar	12	Iphone XX	999.99	17
☐	Editar	Copiar	Borrar	13	Iphone X	333.33	17
☐	Editar	Copiar	Borrar	14	Iphone 13	544.44	17

PRECIO DESPUÉS DE LA MODIFICACIÓN:






```
SET @p0='-123'; SET @p1='13'; CALL `actualizarPrecio`(@p0, @p1); SELECT @p0 AS `precioPrd`;
```



4. Índices

Un índice es una estructura de base de datos que permite realizar búsquedas o consultas con mayor rapidez. Existen diferentes tipos de índices y diferentes formas de implementarlos en nuestra BDD.

4.1 Tipos de índices

- `Índices de clave primaria.` Identifican de forma única una fila dentro de una tabla y no admiten valores nulos. Hacen uso de la palabra clave: `PRIMARY KEY`
- `Índices únicos.` Garantiza que los valores de una columna son únicos. Son similares a los índices de clave primaria, pero permiten valores nulos. Hacen uso de la palabra clave: `UNIQUE`
- `Índices compuestos.` Se forman con varias columnas y permiten valores nulos.

4.2 Creación de índices al crear la tabla

Se pueden definir los índices en el momento de la creación de la tabla:

```
    CREATE TABLE PRODUCTOS (
        id_producto INTEGER AUTO_INCREMENT,
        nombre VARCHAR(50),
        precio DOUBLE NOT NULL,
        PRIMARY KEY (id_producto),
        UNIQUE (precio),
        INDEX idx_nombre (nombre)
    )
```

4.3 Creación de índices después de crear la tabla

Es posible crear diferentes tipos de índices con la sentencia `ALTER TABLE`:

```
ALTER TABLE cliente ADD INDEX idx_nombre (nombre);
```

También se puede omitir el nombre:

```
ALTER TABLE empleado ADD UNIQUE INDEX (email);
```

También es posible crear un `ÍNDEX` con su propio comando:

```
CREATE UNIQUE INDEX precio_cli ON cliente (precio);
```

Por último, podemos crear un `índex compuesto` de varias columnas para evitar nombres duplicados. La clave evitará tener nombres | apellidos iguales y también permitirá dejar los campos vacios.

```
CREATE UNIQUE INDEX myIndex ON empleado (nombre,apellido1,apellido2);
```

4.4 Instrucciones adicionales

- `SHOW INDEX`: muestra los índices de la BDD.
- `DESC tabla`: muestra los indices de una tabla en particular.

Ejemplo de `DESCRIBE` de la tabla Jardinería

```
DESCRIBE cliente;

+---------------------------+---------------+------+-----+---------+-------+
| Field                     | Type          | Null | Key | Default | Extra |
+---------------------------+---------------+------+-----+---------+-------+
```

```
| codigo_cliente             | int(11)       | NO   | PRI | NULL    |       |
| nombre_cliente             | varchar(50)   | NO   |     | NULL    |       |
| nombre_contacto            | varchar(30)   | YES  |     | NULL    |       |
| apellido_contacto          | varchar(30)   | YES  |     | NULL    |       |
| telefono                   | varchar(15)   | NO   |     | NULL    |       |
| fax                        | varchar(15)   | NO   |     | NULL    |       |
| linea_direccion1           | varchar(50)   | NO   |     | NULL    |       |
| linea_direccion2           | varchar(50)   | YES  |     | NULL    |       |
| ciudad                     | varchar(50)   | NO   |     | NULL    |       |
| region                     | varchar(50)   | YES  |     | NULL    |       |
| pais                       | varchar(50)   | YES  | MUL | NULL    |       |
| codigo_postal              | varchar(10)   | YES  |     | NULL    |       |
| codigo_empleado_rep_ventas | int(11)       | YES  | MUL | NULL    |       |
| limite_credito             | decimal(15,2) | YES  |     | NULL    |       |
+----------------------------+---------------+------+-----+---------+-------+
```

5. ⚜ Vistas ⚜

Una vista es una **tabla virtual** generada a partir de la ejecución de varias consultas sobre una o más tablas. Una vista tiene la misma estructura de filas y columnas que cualquier otra tabla MySQL y se almacenan del mismo modo.

5.1 Sintaxis

```
CREATE VIEW nombre_vista
[LISTA DE COLUMNAS]
AS consulta
```

- nombre_vista: Representa el nombre de la tabla virtual.
- lista_columnas: Es el listado de columnas que creará y contendrá la vista.
- consulta: Se trata de una consulta `SELECT` que nos devuelve los datos que forman de la vista.

5.2 ¿Cual es la ventaja de usar vistas?

La mayor ventaja de utilizar vistas se obtiene en forma de **rendimiento**, ya que no estaremos generando constantemente una vista si ésta ya existe (al contrario de las tablas). Cuanto más complejas sean las consultas que se deben ejecutar para obtener la vista, mayor será la ganancia de rendimiento.

Por lo tanto, una vista en términos generales, puede ayudarte a construir una interfaz simplificada y abstracta a una base de datos compleja.

5.3 Ejemplos

Crea una vista para que liste el nombre de los productos con un precio inferior a 100 euros

```
CREATE VIEW productos_baratos
AS SELECT precio
```

```
FROM producto
WHERE precio < 100;
```

Crea una vista que muestre para cada uno de los pedidos, el código del pedido, la fecha, el nombre del cliente que realizó el pedido y el importe total del pedido.

```
CREATE VIEW resumen_pedidos AS
SELECT
    pedido.codigo_pedido,
    pedido.fecha_pedido,
    cliente.nombre_cliente,
    SUM(detalle_pedido.cantidad * detalle_pedido.precio_unidad) AS total

FROM cliente INNER JOIN pedido
    ON cliente.codigo_cliente = pedido.codigo_cliente
INNER JOIN detalle_pedido
    ON pedido.codigo_pedido = detalle_pedido.codigo_pedido
GROUP BY pedido.codigo_pedido
```

Cuando crees una vista, podrás ejecutar consultas sobre ella como si de una tabla más se tratase. Por ejemplo:

```
SELECT * FROM vistaProductosBarato;
```

La consulta anterior devolverá los siguientes resultados:

```
+------------+
| nombre     |
+------------+
| Producto A |
| Producto B |
+------------+
```

Por otro lado, la consulta de Jardinería devolverá los siguientes resultados:

```
SELECT * FROM resumen_pedido;
```

```
+---------------+------------+----------------+----------+
| codigo_pedido |fecha_pedido| nombre_cliente |  total   |
+---------------+------------+----------------+----------+
| 123           | 2023-12-20 | Alpaca         | 4534.56  |
```

```
| 124           | 2020-02-15 | Almerimar      | 67876.32 |
+---------------+------------+----------------+----------+
```

6. Ejemplo completo de TRIGGER

Vamos a crear un TRIGGER que sea capaz de registrar los ascensos/descensos de los empleados.

Para el ejemplo usaremos la base de datos de JARDINERIA y modificaremos la tabla empleado para que tenga 2 columnas adicionales: comentarios y fecha de modificación.

6.1 Modificación de la tabla

Añadimos la tabla de comentarios

```
ALTER TABLE empleado ADD COLUMN comentarios TEXT NULL;
```

Añadimos la tabla de fecha_mod

```
ALTER TABLE empleado ADD COLUMN fecha_mod DATE NULL;
```

nombre	apellido1	comentarios	fecha_mod
Marcos	Magaña	*NULL*	*NULL*
Ruben	López	*NULL*	*NULL*
Alberto	Soria	*NULL*	*NULL*
Maria	Solís	*NULL*	*NULL*
Felipe	Rosas	*NULL*	*NULL*
Juan Carlos	Ortiz	*NULL*	*NULL*
Carlos	Soria	*NULL*	*NULL*
Mariano	López	*NULL*	*NULL*
Lucio	Campoamor	*NULL*	*NULL*
Hilario	Rodriguez	*NULL*	*NULL*

6.2 Creamos el trigger

```
DELIMITER $$
CREATE TRIGGER ver_ascensos
BEFORE UPDATE ON empleado FOR EACH ROW
BEGIN
    IF OLD.PUESTO != NEW.puesto THEN
        IF NEW.puesto = 'Representante Ventas' THEN
                SET NEW.comentarios = 'El empleado ha sido asignado a ser
Representante de ventas';
                SET NEW.fecha_mod = CURRENT_DATE();

        ELSEIF NEW.puesto = 'Secretaria' THEN
                SET NEW.comentarios = 'El empleado ha sido asignado a ser
Secretaria';
                SET NEW.fecha_mod = CURRENT_DATE();

        ELSEIF NEW.puesto = 'Director Oficina' THEN
                SET NEW.comentarios = 'El empleado ha sido asignado a ser Director
Oficina';
                SET NEW.fecha_mod = CURRENT_DATE();

        ELSEIF NEW.puesto = 'Subdirector Marketing' THEN
                SET NEW.comentarios = 'El empleado ha sido asignado a ser
```

```
Subdirector Marketing';
                SET NEW.fecha_mod = CURRENT_DATE();

        ELSEIF NEW.puesto = 'Subdirector Ventas' THEN
                SET NEW.comentarios = 'El empleado ha sido asignado a ser
Subdirector ventas';
                SET NEW.fecha_mod = CURRENT_DATE();

            ELSE
                SET NEW.puesto = OLD.puesto;
                SET NEW.comentarios = 'Sin cambios';
                SET NEW.fecha_mod = CURRENT_DATE();
        END IF;
    END IF;
END;$$
```

6.3 Consultas de prueba

Cargo del empleado 4 antes de la modificación

codigo_empleado	nombre	apellido1	apellido2	puesto
1	Marcos	Magaña	Perez	Director General
2	Ruben	López	Martinez	Subdirector Marketing
3	Alberto	Soria	Carrasco	Subdirector Ventas
4	Maria	Solís	Jerez	Secretaria

*Cargo del **empleado 4** después de la modificación*

```
UPDATE empleado SET puesto="Limpiadora" WHERE codigo_empleado = 4;
```

codigo_empleado	nombre	apellido1	apellido2	puesto	comentarios	fecha_mod
1	Marcos	Magaña	Perez	Director General	*NULL*	*NULL*
2	Ruben	López	Martinez	Subdirector Marketing	*NULL*	*NULL*
3	Alberto	Soria	Carrasco	Subdirector Ventas	*NULL*	*NULL*
4	Maria	Solís	Jerez	Secretaria	Sin cambios	2023-06-18

*Cargo del **empleado 3** después de la modificación*

```
UPDATE empleado SET puesto="Director Oficina" WHERE codigo_empleado = 3;
```

codigo_empleado	nombre	apellido1	apellido2	puesto	comentarios	fecha_mod
1	Marcos	Magaña	Perez	Director General	*NULL*	*NULL*
2	Ruben	López	Martinez	Subdirector Marketing	*NULL*	*NULL*
3	Alberto	Soria	Carrasco	Director Oficina	El empleado ha sido asignado a ser Director Oficin...	2023-06-18
4	Maria	Solís	Jerez	Secretaria	Sin cambios	2023-06-18