

# **PRODUCTO 5.**

## **PROCEDIMIENTOS ALMACENADOS, FUNCIONES UDF Y DISPARADORES**

### **COMPONENTES DEL GRUPO C2047 LAC**

M<sup>a</sup>Celia Garcia Molina

Liliana Díaz Ibáñez

Agnès Garcia Mateo

### **CONSULTORA**

Rita de la Torre Chirivella

<b>Descripción</b>	<b>3</b>
<b>Objetivos</b>	<b>3</b>
<b>Realizar los siguientes procedimientos almacenados y Funciones y anotar la sentencia SQL y su salida en un documento:</b>	<b>3</b>
Crear un procedimiento almacenado que calcule y actualice el precio de compra de cada producto basado en el último precio de compra.	3
Crear un procedimiento almacenado que genere automáticamente las donaciones del día. La política de la Rosticería es donar a comedores sociales aquellos productos con fecha de caducidad igual a la de la fecha en curso.	4
Crear una función que calcule el stock actual de un producto, basado en las comandas y donaciones.	5
Inventar una función que brinde algún tipo de información sobre las ventas del día.	5
Inventar un procedimiento almacenado que tenga un parámetro de entrada.	5
Realizar los siguientes disparadores y anotar la sentencia SQL y su salida en un documento:	6
Crear dos disparadores que, al realizar el registro en comanda de un producto o plato, verifique que la cantidad solicitada sea mayor a cero.	6
Crear un disparador que, al realizar la compra de un producto, copie su código, tipo (ingrediente o elaborado), cantidad y unidad en una nueva tabla (esta tabla deberá ser creada por vosotros).	6
Inventar y justificar un disparador.	6

## Descripción

En este producto continuaremos trabajando con nuestra base de datos pero procederemos a aprender a trabajar con los procedimientos almacenados y las funciones, así como a implementar los disparadores. Son funciones que nos pueden ayudar a tener más agilidad o a tener mayor control de los cambios en nuestra base de datos.

## Objetivos

- Crear procedimientos o funciones en una base de datos e implementar los disparadores.

# 1. Realizar los siguientes procedimientos almacenados y Funciones y anotar la sentencia SQL y su salida en un documento:

## A. Crear un procedimiento almacenado que calcule y actualice el precio de compra de cada producto basado en el último precio de compra.

Un **procedimiento almacenado** es un conjunto de instrucciones SQL Server compila, se encuentran almacenados en la base de datos, los cuales pueden ser ejecutados en cualquier momento.

Código:

```
delimiter //
CREATE PROCEDURE actualizar_precios()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE i VARCHAR(5);
    DECLARE p DECIMAL(12,2);
    DECLARE com_producto CURSOR FOR Select idProdComProd, preComProd FROM
    COMPRAS_PRODUCTO inner join PRODUCTO on PRODUCTO.idProd = idProdComProd;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN com_producto;

    read_loop: LOOP
        FETCH com_producto INTO i, p;

        IF done THEN
            LEAVE read_loop;
        END IF;

        update PRODUCTO set Precio_Compra = p where idProd = i/* idProdComProd*/;

    END LOOP;

    CLOSE com_producto;
```

```
END  
//
```

**B. Crear un procedimiento almacenado que genere automáticamente las donaciones del día. La política de la Rosticería es donar a comedores sociales aquellos productos con fecha de caducidad igual a la de la fecha en curso.**

```
delimiter //  
CREATE PROCEDURE donaciones_dia()  
BEGIN  
    DECLARE done INT DEFAULT FALSE;  
    DECLARE ENCCREADO INT DEFAULT FALSE;  
    DECLARE i VARCHAR(5);  
    DECLARE c INT;  
    DECLARE cur_productos CURSOR FOR SELECT idProd, Stock FROM PRODUCTO where  
    Fecha_Caducidad >= CURDATE();  
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;  
  
    OPEN cur_productos;  
  
    select 1 into @linea;  
  
    read_loop: LOOP  
        FETCH cur_productos INTO i, c;  
  
        IF done THEN  
            LEAVE read_loop;  
        END IF;  
  
        IF (NOT ENCCREADO) THEN  
            INSERT INTO DONACIONES VALUES (CURDATE());  
            SELECT TRUE INTO ENCCREADO;  
        END IF;  
  
        IF (c IS NULL) THEN  
            select 1 into c;  
        END IF;
```

```

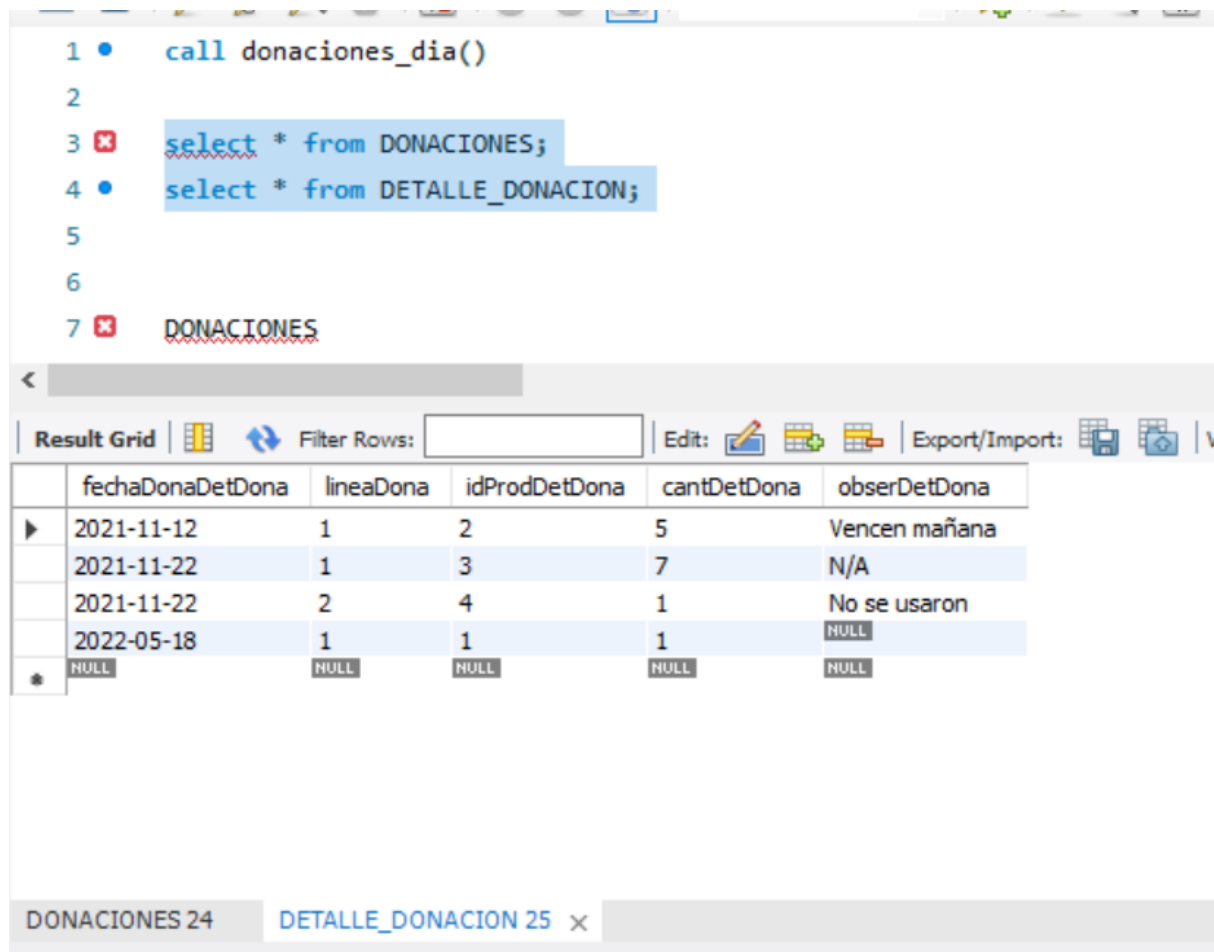
INSERT INTO DETALLE_DONACION (fechaDonaDetDona, lineaDona, idProdDetDona,
cantDetDona)
VALUES (CURDATE(), @linea, i, c);

        select @linea + 1 into @linea;
END LOOP;

CLOSE cur_productos;

END
//

```



	fechaDonaDetDona	lineaDona	idProdDetDona	cantDetDona	obserDetDona
▶	2021-11-12	1	2	5	Vencen mañana
	2021-11-22	1	3	7	N/A
	2021-11-22	2	4	1	No se usaron
	2022-05-18	1	1	1	NULL
*	NULL	NULL	NULL	NULL	NULL

DONACIONES 24    DETALLE\_DONACION 25 x

**c. Crear una función que calcule el stock actual de un producto, basado en las comandas y donaciones.**

```
drop function CalcularStock;
```

```

delimiter //
CREATE FUNCTION CalcularStock (idProducto INT) RETURNS
INT
BEGIN
    DECLARE res INT;

    SELECT Stock INTO res from PRODUCTO where idProd = idProducto;

    SELECT res + cantDetDona INTO res from DETALLE_DONACION where idProdDetDona
= idProducto;

    RETURN res;
END
//

```

The screenshot shows a database management tool interface. The top toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. The main query editor displays the following SQL code:

```

1
2 • SELECT CalcularStock(1)
3
4
5

```

Below the editor, the 'Result Grid' tab is active, showing a single row of results:

CalcularStock(1)
2

On the right side, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' pane shows the execution log:

#	Time	Action	Message
1	00:19:45	SELECT CalcularStock(1) LIMIT 0, 1000	1 row(s) returned

## D. Inventar una función que brinde algún tipo de información sobre las ventas del día.

Creemos una función que nos calcule el beneficio del día.

```
CREATE FUNCTION CalcularBeneficioDia (PVP float, coste float) RETURNS  
FLOAT (5,2)  
BEGIN  
    DECLARE beneficio FLOAT(5,2);  
    SET beneficio = PVP - coste;  
    RETURN beneficio  
END //  
DELIMITER ;
```

## E. Inventar un procedimiento almacenado que tenga un parámetro de entrada.

```
DELIMITER $$  
    CREATE PROCEDURE  
        `ver_categoria_proveedor`(IN codemp char(12))  
        SELECT nifProv FROM PROVEEDOR WHERE nif = codemp$$  
DELIMITER ;
```

```
CALL ver_categoria_proveedor('A28w647451');
```



## 2. Realizar los siguientes disparadores y anotar la sentencia SQL y su salida en un documento:

**A. Crear dos disparadores que, al realizar el registro en comanda de un producto o plato, verifique que la cantidad solicitada sea mayor a cero.**

```
delimiter $$
create trigger `validar comanda`
before insert on COMANDA
for each row
begin
    if length(new.idCo) < 0
    then signal sqlstate '02000' set message_text = 'Error';
    end if;
end $$
delimiter ;
```

**B. Crear un disparador que, al realizar la compra de un producto, copie su código, tipo (ingrediente o elaborado), cantidad y unidad en una nueva tabla (esta tabla deberá ser creada por vosotros).**

```
CREATE TABLE OtraTabla
(
    idProd VARCHAR(5),
    Cantidad INT,
    Tipo enum('INGREDIENTE','ELABORADO'),
    Unidad enum('INGREDIENTE','ELABORADO')
);

delimiter //
create trigger `copiarProducto`
before insert on COMPRAS_PRODUCTO
for each row
```

```

begin
    select Tipo, uniProd into @t, @u from PRODUCTO where idProd = new.idProdComProd;

    INSERT INTO OtraTabla (idProd, Tipo, Cantidad, Unidad)
    VALUE (new.idProdComProd, @t, new.cantComProd, @u);

end
//

```

### **C. Inventar y justificar un disparador.**

Supongamos que queremos que el total de la compra se actualice automaticamente al insertar

```

DELIMITER $$
CREATE TRIGGER `actualizar_total` BEFORE INSERT ON
`COMPRAS_PRODUCTO`
FOR EACH ROW BEGIN

    SET NEW.total = NEW.cantComProd * NEW.preComProd;
END$$
DELIMITER ;

```