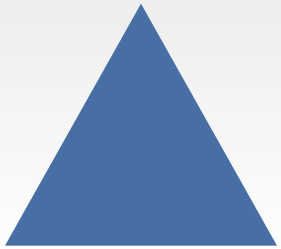
IES Valle del Jerte
PLASENCIA
Bilingual Section

# TEMA 9: PROGRAMACIÓN EN BASES DE DATOS

## FUNCIONES ALMACENADAS



UNION EUROPEA
Fondo Social Europeo
"Una manera de hacer Europa"

"**Una manera de hacer Europa**". Cofinanciación a cargo del Programa Operativo del FSE 2014-2020 para Extremadura gastos de Ciclos Formativos de Grados Medio y Superior.

# Índice

# INTRODUCCIÓN

- Las funciones son el segundo tipo de rutinas que vamos a utilizar en nuestras bases de datos.
- Pueden ser utilizadas para un gran número de tareas y permiten mejorar la integridad de datos y la seguridad de los mismos.
- Al igual que los procedimientos almacenados, sirven para mejorar el rendimiento y la legibilidad del código.

# CARACTERÍSTICAS DE LAS FUNCIONES ALMACENADAS

- Una función está formada por un bloque de instrucciones que se ejecutan y que 'devuelven' un valor o un dato que es recogido por la sentencia que realiza la llamada a la función. (sentencia select generalmente)
- Podríamos ver a las funciones almacenadas como un procedimiento con un parámetro de salida (OUT).
- Es aconsejable el uso de funciones almacenadas frente al uso de procedimientos con parámetros de salida.

# DIFERENCIAS ENTRE PROCEDIMIENTOS ALMACENADOS Y FUNCIONES ALMACENADAS

- La función se diferencia del procedimiento almacenado en:
  - Que sólo admiten parámetros de entrada.
  - Que retornan un valor.
  - Que se pueden usar dentro de instrucciones SQL.
  - Que no se llaman explícitamente con una instrucción, sino que se invocan dentro de una sentencia SELECT, como por ejemplo: SELECT funcion(),col1 FROM Tabla.

# SINTAXIS DE CREACIÓN DE FUNCIONES ALMACENADAS

- La sintaxis general para la creación de una función es:

```
CREATE FUNCTION nombre_funcion ([parámetro[,…]])
 RETURNS TIPO_DATOS;
 [características]
 begin
      instrucciones de la función
 return variable o valor;
 end;
```

# SINTAXIS DE CREACIÓN DE FUNCIONES ALMACENADAS

- Los parámetros siempre son de entrada (IN), por lo que no es necesario indicarlo.
- Los valores posibles a la opcion CARACERISTICAS son los mismos que en los procedimientos.
- Todas las funciones deben ser DETERMINISTAS.
- Para que las funciones compilen correctamente es posible que sea necesario ejecutar la sentencia:

  SET GLOBAL log_bin_trust_function_creators = 1;

  para evitar el error de compilación:

  Error Code: 1418. This function has none of DETERMINISTIC, NO SQL, or READS SQL DATA

# BORRAR UNA FUNCIÓN ALMACENADA

Como el resto de objetos de la BD, las funciones se borran con el comando DROP.

**DROP FUNCTION Nombre_funcion;**

# EJEMPLOS DE FUNCIONES

# Ejemplo 1.
# FUNCIÓN ALMACENADA SIN PARÁMETROS

```
USE BDTEMA9;
/* cambiar la variable para que las funciones sean deterministas*/
SET GLOBAL log_bin_trust_function_creators = 1;
DELIMITER //
DROP FUNCTION IF EXISTS EJEMPLOFUNCION//
CREATE FUNCTION EJEMPLOFUNCION() RETURNS VARCHAR(50)
BEGIN
    RETURN 'Esto es un ejemplo de función almacenada';
END
//
```

```
18 •  select EJEMPLOFUNCION() AS "RESULTADO_FUNCION";
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| RESULTADO_FUNCION |
| --- |
| Esto es un ejemplo de función almacenada |

# EJEMPLO 1

En el ejemplo anterior:

- Creamos una función que no acepta ningún parámetro y que devolverá una cadena de caracteres cuyo valor asignamos en el cuerpo de la función.
- Utilizamos el cambio de delimitadores igual que en los procedimientos almacenados.
- La instrucción RETURNS seguida de un tipo de datos determina la salida de la función.
- La declaración de variables en una función es similar a la utilizada en procedimientos almacenados.
- La sentencia final RETURN devuelve el valor calculado en la función, en este caso una cadena de caracteres.
- La función se llama dentro de una sentencia select que muestra el retorno de la función.

# Ejemplo 2
# FUNCIÓN ALMACENADA CON PARÁMETROS

```
-- Función que reciba dos números enteros y devuelve su suma
DELIMITER //
DROP FUNCTION IF EXISTS SUMA//
CREATE FUNCTION SUMA(PAR1 INT, PAR2 INT) RETURNS INT
BEGIN
    RETURN PAR1+PAR2;
END
//
```

```
28 •  SELECT SUMA(5,12) AS RESULTADO;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| | RESULTADO |
|---|---|
| ▶ | 17 |

# Ejemplo 3
# FUNCIÓN ALMACENADA CON PARÁMETROS Y DECLARACIÓN DE VARIABLES

CREAMOS LA TABLA PRODUCTOS.

```
CREATE TABLE productos (
    id INT NOT NULL AUTO_INCREMENT,
    nombre VARCHAR(20) NOT NULL,
    coste FLOAT NOT NULL DEFAULT 0.0,
    precio FLOAT NOT NULL DEFAULT 0.0,
    PRIMARY KEY(id) );
```

INSERTAMOS VARIOS REGISTROS:

```
INSERT INTO productos (nombre, coste, precio)
VALUES ('Producto A', 4, 8), ('Producto B', 1, 1.5),('Producto C', 50, 80);
```

# Ejemplo 3
# FUNCIÓN ALMACENADA CON PARÁMETROS Y DECLARACIÓN DE VARIABLES

```
DELIMITER //
DROP FUNCTION IF EXISTS calcularBeneficio//
CREATE FUNCTION calcularBeneficio(coste FLOAT, precio FLOAT) RETURNS DECIMAL(9,2)
BEGIN
    DECLARE beneficio DECIMAL(9,2);
    SET beneficio = precio - coste;
    RETURN beneficio;
END//
```

```
50 • SELECT *, calcularBeneficio(coste, precio) AS beneficio FROM productos;
51
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| id | nombre | coste | precio | beneficio |
|---|---|---|---|---|
| 7 | Producto A | 4 | 8 | 4.00 |
| 8 | Producto B | 1 | 1.5 | 0.50 |
| 9 | Producto C | 50 | 80 | 30.00 |

# Ejemplo 4
# FUNCIÓN QUE CUENTA EL NÚMERO DE ACTORES DE LA TABLA ACTOR (BD VIDEOTECA)

```
USE videoteca;
DELIMITER //
CREATE FUNCTION fa_actores_cantidad()
RETURNS INT
BEGIN
DECLARE actores INT;
SELECT   COUNT(*) INTO  actores
FROM   actor;
RETURN actores;
END
//
DELIMITER ;
```

Llamada a la función almacenada:

```
USE videoteca;

SELECT fa_actores_cantidad();
```