

PARTE 1

Tarea Funciones de Usuario

Tarea: Funciones de Usuario en Bases de Datos

Objetivo:

El objetivo de esta tarea es que los estudiantes aprendan a crear funciones de usuario en bases de datos

Escenario:

Vas a crear una base de datos para una tienda en línea que maneja clientes, productos, pedidos y detalles de los pedidos.

Pasos a Seguir:

1. Crear la Base de Datos y Tablas:

- Crea una base de datos llamada `tienda_online`.

```
create database tienda_online;  
use tienda_online;
```

- Dentro de la base de datos, crea las siguientes tablas:
 - Clientes: Contendrá información básica sobre los clientes (id, nombre, apellido, email, teléfono, fecha de registro).

```
CREATE TABLE Clientes (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    apellido VARCHAR(100) NOT NULL,  
    email VARCHAR(100) NOT NULL UNIQUE,  
    telefono VARCHAR(15),  
    fecha_registro DATETIME DEFAULT CURRENT_TIMESTAMP,  
    CONSTRAINT chk_email CHECK (email <> '')  
);
```

- Productos: Contendrá información sobre los productos (id, nombre, precio, stock, descripción).

```
CREATE TABLE Productos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL UNIQUE,
  precio DECIMAL(10, 2) NOT NULL CHECK (precio > 0),
  stock INT NOT NULL CHECK (stock >= 0),
  descripcion TEXT
);
```

- Pedidos: Registra los pedidos realizados por los clientes (id, cliente_id, fecha del pedido, total).

```
CREATE TABLE Pedidos (
  id INT AUTO_INCREMENT PRIMARY KEY,
  cliente_id INT NOT NULL,
  fecha_pedido DATETIME DEFAULT CURRENT_TIMESTAMP,
  total DECIMAL(10, 2) NOT NULL CHECK (total >= 0),
  FOREIGN KEY (cliente_id) REFERENCES Clientes(id)
);
```

- Detalles_Pedido: Registra los detalles de cada pedido (id, pedido_id, producto_id, cantidad, precio unitario).

```
CREATE TABLE Detalles_Pedido (
  id INT AUTO_INCREMENT PRIMARY KEY,
  pedido_id INT NOT NULL,
  producto_id INT NOT NULL,
  cantidad INT NOT NULL CHECK (cantidad > 0),
  precio_unitario DECIMAL(10, 2) NOT NULL CHECK (precio_unitario > 0),
  FOREIGN KEY (pedido_id) REFERENCES Pedidos(id),
  FOREIGN KEY (producto_id) REFERENCES Productos(id)
);
```

2. Restricciones:

- No se permiten valores nulos en campos como nombre, apellido, email, precio, y cantidad.
- Los precios deben ser positivos.
- El stock de los productos no puede ser negativo.
- Los nombres de los productos no deben repetirse.

- El email de los clientes debe ser único.
- 3. Crear Funciones de Usuario
- 4. Función para obtener el nombre completo de un cliente:
 - Esta función debe aceptar un `cliente_id` como parámetro y devolver el nombre completo (nombre + apellido) del cliente.

```
/* Función para obtener el nombre completo de un cliente:*/  
DELIMITER //  
CREATE FUNCTION obtener_nombre_completo(cliente_id INT)  
RETURNS VARCHAR(255)  
DETERMINISTIC  
BEGIN  
    DECLARE nombre_completo VARCHAR(255);  
    SELECT CONCAT(nombre, ' ', apellido) INTO nombre_completo  
    FROM Clientes  
    WHERE id = cliente_id;  
    RETURN nombre_completo;  
END //  
DELIMITER ;
```

Result Grid		Filter Rows
	NombreCompleto	
▶	Juan Pérez	

- Función para calcular el descuento de un producto:
 - Esta función debe aceptar el `precio` y el `descuento` como parámetros y devolver el precio con descuento.

```
/*Función para calcular el descuento de un producto:*/  
DELIMITER //  
CREATE FUNCTION calcular_precio_con_descuento(precio DECIMAL(10, 2), descuento DECIMAL(5, 2))  
RETURNS DECIMAL(10, 2)  
DETERMINISTIC  
BEGIN  
    DECLARE precio_final DECIMAL(10, 2);  
    SET precio_final = precio - (precio * (descuento / 100));  
    RETURN precio_final;  
END //  
DELIMITER ;
```

Result Grid			
Filter Rows:			
Export:			
	PrecioOriginal	Descuento	PrecioConDescuento
▶	800.00	80.0000	720.0000

- Función para calcular el total de un pedido:
 - Esta función debe aceptar un `pedido_id` y calcular el total del pedido sumando los precios de los productos multiplicados por sus respectivas cantidades.

```

/*Función para calcular el total de un pedido:*/
DELIMITER //
CREATE FUNCTION calcular_total_pedido(pedido_id INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
  DECLARE total DECIMAL(10, 2);
  SELECT SUM(dp.cantidad * dp.precio_unitario) INTO total
  FROM Detalles_Pedido dp
  WHERE dp.pedido_id = pedido_id;
  IF total IS NULL THEN
    SET total = 0;
  END IF;
  RETURN total;
END //
DELIMITER ;

```

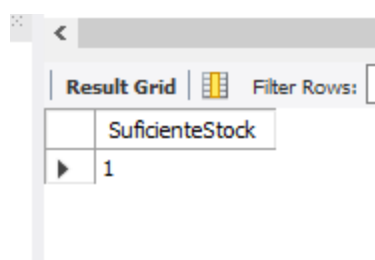
Result Grid	
TotalPedido	
▶	800.00

- Función para verificar la disponibilidad de stock de un producto:
 - Esta función debe aceptar un `producto_id` y una `cantidad` como parámetros y devolver `TRUE` si el stock disponible es suficiente, de lo contrario, debe devolver `FALSE`.

```

/*Función para verificar la disponibilidad de stock de un producto:*/
DELIMITER //
CREATE FUNCTION verificar_disponibilidad_stock(producto_id INT, cantidad INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE stock_disponible INT;
    SELECT stock INTO stock_disponible
    FROM Productos
    WHERE id = producto_id;
    IF stock_disponible IS NULL THEN
        RETURN FALSE;
    ELSEIF stock_disponible >= cantidad THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END //
DELIMITER ;

```



The screenshot shows a database client interface. At the top, there is a text area containing the SQL code for the function. Below the text area, there is a 'Result Grid' tab. The 'Result Grid' shows a single row with the column name 'SuficienteStock' and the value '1'. To the right of the 'Result Grid' is a 'Filter Rows' input field.

SuficienteStock
1

- Función para calcular la antigüedad de un cliente:
 - Esta función debe aceptar un `cliente_id` y calcular la antigüedad del cliente en años a partir de la fecha de registro.

```

/* Función para calcular la antigüedad de un cliente*/
DELIMITER //
CREATE FUNCTION calcular_antigüedad_cliente(cliente_id INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE fecha_registro DATETIME;
    DECLARE antigüedad INT;
    SELECT fecha_registro INTO fecha_registro
    FROM Clientes
    WHERE id = cliente_id;
    IF fecha_registro IS NULL THEN
        RETURN NULL;
    ELSE
        SET antigüedad = TIMESTAMPDIFF(YEAR, fecha_registro, CURDATE());
        RETURN antigüedad;
    END IF;
END //

DELIMITER ;

```

5. Consultas de Uso de Funciones:

- Consulta para obtener el nombre completo de un cliente dado su `cliente_id`.

```

SELECT CONCAT(nombre, ' ', apellido) AS NombreCompleto
FROM Clientes
WHERE id = 1;

```

- Consulta para calcular el descuento de un producto dado su `precio` y un `descuento` del 10%.

```


SELECT
    precio AS PrecioOriginal,
    (precio * 0.10) AS Descuento,
    (precio - (precio * 0.10)) AS PrecioConDescuento
FROM
    Productos
WHERE
    id = 1;

```

- Consulta para calcular el total de un pedido dado su `pedido_id`.

```
SELECT
    SUM(dp.cantidad * dp.precio_unitario) AS TotalPedido
FROM
    Detalles_Pedido dp
WHERE
    dp.pedido_id = 1;
```

- Consulta para verificar si un producto tiene suficiente stock para una cantidad solicitada.



```
SELECT
    CASE
        WHEN stock >= 5 THEN TRUE
        ELSE FALSE
    END AS SuficienteStock
FROM
    Productos
WHERE
    id = 1;
```

PARTE 2

Aprendizaje de Funciones SQL: Creación, Análisis y Ejecución

Objetivo:

El objetivo de esta actividad es aprender a crear y utilizar funciones definidas por el usuario en SQL, analizar su estructura y lógica, y practicar la creación de tablas y consultas con funciones personalizadas. También se incluirán ejemplos prácticos para mostrar cómo utilizar estas funciones en un contexto real.

Instrucciones:

1. Transcripción y análisis del código SQL.
2. Creación de las tablas necesarias para almacenar los datos.
3. Ejecución de las funciones SQL creadas y captura de los resultados.
4. Explicación detallada de cada línea del código.

[SUBIR A GIT HUB EL SCRIPT Y EL PDF](#)

EJERCICIO 1

```
CREATE FUNCTION CalcularTotalOrden(id_orden INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10, 2);
    DECLARE iva DECIMAL(10, 2);

    SET iva = 0.15;

    SELECT SUM(P.precio * O.cantidad) INTO total
    FROM Ordenes O
    JOIN Productos P ON O.producto_id = P.ProductoID
    WHERE O.OrdenID = id_orden;

    SET total = total + (total * iva);

    RETURN total;
END $$

DELIMITER ;
```



```
CREATE DATABASE tiendaaaa;  
USE tiendaaaa;
```

```
CREATE TABLE Productos (  
    ProductoID INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    precio DECIMAL(10,2) NOT NULL  
);
```

```
CREATE TABLE Ordenes (  
    OrdenID INT AUTO_INCREMENT PRIMARY KEY,  
    producto_id INT NOT NULL,  
    cantidad INT NOT NULL,  
    FOREIGN KEY (producto_id) REFERENCES Productos(ProductoID)  
);
```

```
INSERT INTO Productos (nombre, precio) VALUES  
( 'Producto A', 10.00),  
( 'Producto B', 20.00),  
( 'Producto C', 15.50);
```

```
INSERT INTO Ordenes (producto_id, cantidad) VALUES  
(1, 2),  
(2, 1),  
(3, 3);
```

```
DELIMITER $$ -- Cambia el delimitador a $$ para la función
```

```
CREATE FUNCTION CalcularTotalOrden(id_orden INT) -- Define la función con un parámetro id_orden  
RETURNS DECIMAL(10,2) -- Devuelve un valor decimal (10 dígitos, 2 decimales)  
DETERMINISTIC -- Asegura resultados consistentes para los mismos parámetros  
BEGIN -- Inicio del bloque de código
```

```
    DECLARE total DECIMAL(10,2); -- Variable para almacenar el total  
    DECLARE iva DECIMAL(10,2); -- Variable para almacenar el IVA
```

```
    SET iva = 0.15; -- Asigna el valor del IVA (15%)
```

```

-- Calcula el total de la orden
SELECT SUM(P.precio * O.cantidad) INTO total
FROM Ordenes O
JOIN Productos P ON O.producto_id = P.ProductoID
WHERE O.OrdenID = id_orden; -- Filtra por el ID de la orden

-- Establece total a 0 si no hay productos
> IF total IS NULL THEN
    SET total = 0;
- END IF;

SET total = total + (total * iva); -- Suma el IVA al total

```

```

RETURN total; -- Devuelve el total final
END $$ -- Fin del bloque de código

```

```




DELIMITER ; -- Restablece el delimitador a ;

```

```

SELECT CalcularTotalOrden(1) AS TotalOrden;

```

Result Grid   	
	TotalOrden
▶	23.00

EJERCICIO 2

```
DELIMITER $$

CREATE FUNCTION CalcularEdad(fecha_nacimiento DATE)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE edad INT;
    SET edad = TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE());
    RETURN edad;
END $$

DELIMITER ;
```

```
CREATE DATABASE IF NOT EXISTS personas;
USE personas;
```



```
CREATE TABLE IF NOT EXISTS Personas (
    PersonaID INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(100) NOT NULL,
    fecha_nacimiento DATE NOT NULL
);
```

```
INSERT INTO Personas (nombre, fecha_nacimiento) VALUES
('Juan Pérez', '1990-05-15'),
('María López', '1985-08-20'),
('Carlos García', '2000-12-30');
```

```
DELIMITER $$ -- Cambia el delimitador a $$
CREATE FUNCTION CalcularEdad(fecha_nacimiento DATE) -- Define la función con un parámetro de fecha
RETURNS INT -- Devuelve un entero (edad)
DETERMINISTIC -- Resultados consistentes para los mismos parámetros
BEGIN -- Inicio del bloque de la función

    DECLARE edad INT; -- Declara la variable edad
    SET edad = TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE()); -- Calcula la edad en años
    RETURN edad; -- Devuelve la edad calculada
END $$ -- Fin del bloque de la función
DELIMITER ; -- Restablece el delimitador a ;
```

```
SELECT nombre, CalcularEdad(fecha_nacimiento) AS Edad FROM Personas;
```

Result Grid   Filter Rows		
	nombre	Edad
▶	Juan Pérez	34
	María López	39
	Carlos García	23

EJERCICIO 3

DELIMITER \$\$

```
CREATE FUNCTION VerificarStock(producto_id INT)
RETURNS BOOLEAN
DETERMINISTIC
```

BEGIN

```
    DECLARE stock INT;
    SELECT Existencia INTO stock
    FROM Productos
    WHERE ProductoID = producto_id;
```

```
    IF stock > 0 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
```

END IF;

END \$\$

DELIMITER ;

```
CREATE DATABASE IF NOT EXISTS tienda_3;  
USE tienda_3;
```

```
CREATE TABLE IF NOT EXISTS Productos (  
    ProductoID INT AUTO_INCREMENT PRIMARY KEY,  
    nombre VARCHAR(100) NOT NULL,  
    Existencia INT NOT NULL  
);
```

```
INSERT IGNORE INTO Productos (nombre, Existencia) VALUES  
( 'Producto A', 10),  
( 'Producto B', 0),  
( 'Producto C', 5);
```

```
DELIMITER $$ -- Cambia el delimitador a $$
```

```
DROP FUNCTION IF EXISTS VerificarStock; -- Elimina la función si existe
```

```
CREATE FUNCTION VerificarStock(producto_id INT) -- Define la función con un parámetro  
RETURNS BOOLEAN -- Devuelve un valor booleano  
DETERMINISTIC -- Resultados consistentes
```

```
BEGIN -- Inicio del bloque  
  
    DECLARE stock INT; -- Declara la variable stock  
    SELECT Existencia INTO stock -- Selecciona existencia y almacena en stock  
    FROM Productos  
    WHERE ProductoID = producto_id; -- Filtra por ID del producto
```

```

IF stock > 0 THEN -- Verifica si hay stock
    RETURN TRUE; -- Devuelve TRUE si hay stock
ELSE
    RETURN FALSE; -- Devuelve FALSE si no hay stock
END IF; -- Fin de la condicional
END $$ -- Fin del bloque

DELIMITER ; -- Restablece el delimitador a ;

```

```

SELECT nombre, VerificarStock(ProductoID) AS StockDisponible FROM Productos;

```

Result Grid			Filter Rows:
	nombre	StockDisponible	
▶	Producto A	1	
	Producto B	0	
	Producto C	1	

EJERCICIO 4

```
CREATE FUNCTION CalcularSaldo(id_cuenta INT)
RETURNS DECIMAL(10, 2)
DETERMINISTIC
BEGIN
    DECLARE saldo DECIMAL(10, 2);

    SELECT SUM(CASE
        WHEN tipo_transaccion = 'deposito' THEN monto
        WHEN tipo_transaccion = 'retiro' THEN -monto
        ELSE 0
    END) INTO saldo
    FROM Transacciones
    WHERE cuenta_id = id_cuenta;

    RETURN saldo;
END $$

DELIMITER ;
```

```
CREATE DATABASE IF NOT EXISTS banco;
USE banco;
```

```
CREATE TABLE IF NOT EXISTS Transacciones (
    transaccion_id INT AUTO_INCREMENT PRIMARY KEY,
    cuenta_id INT NOT NULL,
    tipo_transaccion ENUM('deposito', 'retiro') NOT NULL,
    monto DECIMAL(10,2) NOT NULL,
    fecha TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
INSERT INTO Transacciones (cuenta_id, tipo_transaccion, monto) VALUES
(1, 'deposito', 1000.00),
(1, 'retiro', 200.00),
(1, 'deposito', 500.00),
(2, 'retiro', 300.00);
```

```

DELIMITER $$
-- Eliminar la función si ya existe (para evitar errores)
DROP FUNCTION IF EXISTS CalcularSaldo;
-- Crear la función CalcularSaldo
CREATE FUNCTION CalcularSaldo(id_cuenta INT)
RETURNS DECIMAL(10,2) -- Devuelve un valor decimal (el saldo)
DETERMINISTIC -- La función produce siempre el mismo resultado para la misma entrada
BEGIN
    DECLARE saldo DECIMAL(10,2); -- Declara una variable para almacenar el saldo
    -- Calcula el saldo sumando depósitos y restando retiros
    SELECT SUM(CASE
        WHEN tipo_transaccion = 'deposito' THEN monto -- Suma los depósitos
        WHEN tipo_transaccion = 'retiro' THEN -monto -- Resta los retiros
        ELSE 0 -- Devuelve 0 para cualquier otro caso (seguridad)
        ELSE 0 -- Devuelve 0 para cualquier otro caso (seguridad)
    END) INTO saldo
    FROM Transacciones
    WHERE cuenta_id = id_cuenta; -- Filtra las transacciones de la cuenta específica
    RETURN saldo; -- Devuelve el saldo calculado
END $$
-- Restablecer el delimitador a su valor original
DELIMITER ;

SELECT CalcularSaldo(1) AS SaldoCuenta1;
SELECT CalcularSaldo(2) AS SaldoCuenta2;

```

	SaldoCuenta1
▶	1300.00

	SaldoCuenta2
▶	-300.00

LINK DEL REPOSITORIO: https://github.com/Sebastian-Betancourt/Deber_BD_Tarea_Funciones_de_Usuario.git

