Instituto Tecnológico y de Estudios Superiores de Monterrey

Construcción de software y toma de decisiones

(Gpo 404)

Tarea 5.

SQL Avanzado

Alumno(s):

Luis Eduardo Diaz Valle - A00835871

Profesor:

Benjamín Gutiérrez Padilla

Fecha de Entrega:

2 de Abril del 2025

Realiza los siguientes pasos, para cada uno de ellos debes de subir el script que usaste, mas lo que indique cada paso.

Debes crear un sistema de pedidos para una tienda en línea. Los usuarios podrán hacer pedidos de productos, y se llevará un control de stock, registro de órdenes, auditoría de cambios, y reporte de ventas.

Paso 1: Crear las tablas necesarias (Solo script)

- Productos: ID, nombre, precio, stock.
- Pedidos: ID, fecha, total, cliente.
- DetallePedidos: ID, ID del pedido, ID del producto, cantidad, precio.
- Auditoría: ID, fecha, acción, tabla afectada, registro afectado.

Unset

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

CREATE TABLE Pedidos (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    fecha DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
    total DECIMAL(10, 2) NOT NULL,
    cliente VARCHAR(100) NOT NULL
);

CREATE TABLE DetallePedidos (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    ID_pedido INT NOT NULL,
    ID_producto INT NOT NULL,
    cantidad INT NOT NULL,
    precio DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (ID_pedido) REFERENCES Pedidos(ID),
    FOREIGN KEY (ID_producto) REFERENCES Productos(ID)
);

CREATE TABLE Auditoria (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    fecha DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
```

```
    accion VARCHAR(50) NOT NULL,
    tabla_afectada VARCHAR(50) NOT NULL,
    registro_afectado INT NOT NULL,
    usuario VARCHAR(50) NOT NULL DEFAULT USER()
);
```

Paso 2: Crear índices (Solo script)

• Crear un índice en la tabla Productos sobre el campo nombre para optimizar las búsquedas por nombre de producto.
• Crear un índice compuesto en DetallePedidos sobre ID del pedido y ID del producto para acelerar las consultas que busquen detalles de pedidos.

```
Unset
CREATE INDEX idx_productos_nombre ON Productos(nombre);

CREATE INDEX idx_detallepedidos_pedido_producto ON DetallePedidos(ID_pedido,
ID_producto);
```

Paso 3: Crear un trigger para auditoría (Script + Captura de consulta)

• Crear un trigger que registre cualquier actualización o eliminación de productos o pedidos en la tabla Auditoría (registro de cambios, quién realizó la acción, etc.).
• Ejecuta el trigger y muestra en una consulta la tabla auditoria.

```
Unset
DELIMITER //
CREATE TRIGGER trg_productos_update AFTER UPDATE ON Productos
FOR EACH ROW
BEGIN
    INSERT INTO Auditoria (accion, tabla_afectada, registro_afectado)
    VALUES ('UPDATE', 'Productos', OLD.ID);
END//
```

```
DELIMITER ;

DELIMITER //
CREATE TRIGGER trg_productos_delete AFTER DELETE ON Productos
FOR EACH ROW
BEGIN
    INSERT INTO Auditoria (accion, tabla_afectada, registro_afectado)
    VALUES ('DELETE', 'Productos', OLD.ID);
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER trg_pedidos_update AFTER UPDATE ON Pedidos
FOR EACH ROW
BEGIN
    INSERT INTO Auditoria (accion, tabla_afectada, registro_afectado)
    VALUES ('UPDATE', 'Pedidos', OLD.ID);
END//
DELIMITER ;

DELIMITER //
CREATE TRIGGER trg_pedidos_delete AFTER DELETE ON Pedidos
FOR EACH ROW
BEGIN
    INSERT INTO Auditoria (accion, tabla_afectada, registro_afectado)
    VALUES ('DELETE', 'Pedidos', OLD.ID);
END//
DELIMITER ;
```

Unset

```
-- Consulta para verificar la tabla de auditoría y que los triggers funcionan
correctamente.
INSERT INTO Productos (nombre, precio, stock) VALUES ('Producto Test', 100.00,
10);
UPDATE Productos SET precio = 120.00 WHERE ID = 1;
DELETE FROM Productos WHERE ID = 1;
SELECT * FROM Auditoria;
```

←T→	ID	fecha	accion	tabla_afectada	registro_afectado	usuario
Edit Copy Delete	1	2025-04-02 18:19:39	UPDATE	Productos	1	root@localhost
Edit Copy Delete	2	2025-04-02 18:19:39	DELETE	Productos	1	root@localhost

Paso 4: Crear funciones personalizadas (Script + captura de consulta)

• Crear una función que calcule el total de un pedido, aplicando un descuento del 10% si el total es superior a $1000.

```
Unset
DELIMITER //
CREATE FUNCTION CalcularTotalPedido(pedido_id INT) RETURNS DECIMAL(10, 2)
BEGIN
    DECLARE total_pedido DECIMAL(10, 2);

    SELECT SUM(cantidad * precio) INTO total_pedido
    FROM DetallePedidos
    WHERE ID_pedido = pedido_id;

    IF total_pedido > 1000 THEN
        SET total_pedido = total_pedido * 0.9;
    END IF;

    RETURN total_pedido;
END//
DELIMITER ;
```

```
Unset
-- Consulta para probar la función CalcularTotalPedido.
INSERT INTO Productos (ID, nombre, precio, stock) VALUES (1, 'Producto A',
800.00, 10);
INSERT INTO Pedidos (ID, fecha, total, cliente) VALUES (1, NOW(), 0, 'Cliente
Test');
INSERT INTO DetallePedidos (ID_pedido, ID_producto, cantidad, precio) VALUES
(1, 1, 2, 800.00);
SELECT CalcularTotalPedido(1) AS total_con_descuento;
```

total_con_descuento
1440.00

Paso 5: Crear stored procedure con transaction que permita realizar un pedido (Script + capturas de pruebas de caso COMMIT y caso ROLLBACK)

• Insertar el pedido en la tabla Pedidos.
• Insertar los detalles del pedido en DetallePedidos.
• Actualizar el stock de los productos (restar la cantidad de cada producto).
• Validar si el total del pedido requiere un descuento usando la función CalcularTotalPedido.
• Si todo es correcto, realizar un COMMIT, si no, hacer un ROLLBACK.

Unset

```
DELIMITER //
CREATE PROCEDURE RealizarPedido(
    IN p_cliente VARCHAR(100),
    IN p_productos TEXT -- Formato: "id_producto:cantidad", simulando un
diccionario en python"
)
BEGIN
    DECLARE v_error VARCHAR(255) DEFAULT '';
    DECLARE v_total DECIMAL(10, 2) DEFAULT 0;
    DECLARE v_id_pedido INT;
    DECLARE v_producto_id INT;
    DECLARE v_cantidad INT;
    DECLARE v_precio DECIMAL(10, 2);
    DECLARE v_stock_actual INT;
    DECLARE v_producto_existe INT;
    DECLARE v_done INT DEFAULT FALSE;
    DECLARE v_idx INT DEFAULT 1;
    DECLARE v_comma_pos INT;
    DECLARE v_colon_pos INT;
    DECLARE v_part VARCHAR(100);
    DECLARE v_exit_with_error BOOLEAN DEFAULT FALSE;

    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
        SET v_error = 'Error SQL al procesar el pedido';
        SET v_exit_with_error = TRUE;
        ROLLBACK;
    END;

    START TRANSACTION;

    INSERT INTO Pedidos (fecha, total, cliente) VALUES (NOW(), 0, p_cliente);
```

```
    SET v_id_pedido = LAST_INSERT_ID();

    SET v_idx = 1;
    SET v_part = p_productos;

    procesar_productos: WHILE v_part != '' AND NOT v_exit_with_error DO
        SET v_comma_pos = LOCATE(',', v_part);
        IF v_comma_pos > 0 THEN
            SET v_producto_id = CAST(SUBSTRING_INDEX(SUBSTRING(v_part, 1,
v_comma_pos - 1), ':', 1) AS UNSIGNED);
            SET v_cantidad = CAST(SUBSTRING_INDEX(SUBSTRING(v_part, 1,
v_comma_pos - 1), ':', -1) AS UNSIGNED);
            SET v_part = SUBSTRING(v_part, v_comma_pos + 1);
        ELSE
            SET v_producto_id = CAST(SUBSTRING_INDEX(v_part, ':', 1) AS
UNSIGNED);
            SET v_cantidad = CAST(SUBSTRING_INDEX(v_part, ':', -1) AS
UNSIGNED);
            SET v_part = '';
        END IF;

        SELECT COUNT(*) INTO v_producto_existe FROM Productos WHERE ID =
v_producto_id;
        IF v_producto_existe = 0 THEN
            SET v_error = CONCAT('Producto con ID ', v_producto_id, ' no
existe');
            SET v_exit_with_error = TRUE;
            ROLLBACK;
            LEAVE procesar_productos;
        END IF;

        SELECT precio, stock INTO v_precio, v_stock_actual FROM Productos WHERE
ID = v_producto_id;
        IF v_stock_actual < v_cantidad THEN
            SET v_error = CONCAT('Stock insuficiente para el producto ID ',
v_producto_id);
            SET v_exit_with_error = TRUE;
            ROLLBACK;
            LEAVE procesar_productos;
        END IF;

        INSERT INTO DetallePedidos (ID_pedido, ID_producto, cantidad, precio)
        VALUES (v_id_pedido, v_producto_id, v_cantidad, v_precio);
```

```
        UPDATE Productos SET stock = stock - v_cantidad WHERE ID =
v_producto_id;

        SET v_idx = v_idx + 1;
    END WHILE procesar_productos;

    IF NOT v_exit_with_error THEN
        SET v_total = CalcularTotalPedido(v_id_pedido);
        UPDATE Pedidos SET total = v_total WHERE ID = v_id_pedido;

        COMMIT;

        SELECT 'Pedido realizado correctamente' AS mensaje, v_id_pedido AS
id_pedido, v_total AS total;
    ELSE
        SELECT 'Error en el pedido' AS mensaje, v_error AS error_descripcion;
    END IF;
END//
DELIMITER ;
```

Unset

```
-- Ejemplo para probar el caso COMMIT.
CALL RealizarPedido('Luis Diaz', '1:3,2:2');
```

mensaje	id_pedido	total
Pedido realizado correctamente	12	3330.00

Unset

```
-- Ejemplo para probar el caso ROLLBACK.
CALL RealizarPedido('Luis Diaz', '1:300');
```

mensaje	error_descripcion
Error en el pedido	Stock insuficiente para el producto ID 1

Paso 6: Crear vistas (Script + Consulta a ambas vistas)

• Crear una vista que muestre todos los pedidos realizados, con el nombre del cliente, productos pedidos y el total del pedido
• Crear una vista que muestre el total de ventas por mes.

Unset

```
CREATE VIEW vista_pedidos_detallados AS
SELECT
    p.ID AS id_pedido,
    p.fecha,
    p.cliente,
    GROUP_CONCAT(pr.nombre, ' (', dp.cantidad, ')' SEPARATOR ', ') AS
productos,
    p.total
FROM
    Pedidos p
    JOIN DetallePedidos dp ON p.ID = dp.ID_pedido
    JOIN Productos pr ON dp.ID_producto = pr.ID
GROUP BY
    p.ID, p.fecha, p.cliente, p.total;

CREATE VIEW vista_ventas_mensuales AS
SELECT
    YEAR(fecha) AS año,
    MONTH(fecha) AS mes,
    SUM(total) AS total_ventas,
    COUNT(*) AS num_pedidos
FROM
    Pedidos
GROUP BY
    YEAR(fecha), MONTH(fecha)
ORDER BY
    año DESC, mes DESC;
```

Unset

```
SELECT * FROM vista_pedidos_detallados;
```

id_pedido	fecha	cliente	productos	total
8	2025-04-02 18:57:58	Juan Pérez	Laptop (3), Mouse (2)	3330.00
10	2025-04-02 19:12:41	Luis Diaz	Laptop (3), Mouse (2)	3330.00
12	2025-04-02 19:15:26	Luis Diaz	Mouse (2), Laptop (3)	3330.00

Unset

```
SELECT * FROM vista_ventas_mensuales;
```

año	mes	total_ventas	num_pedidos
2025	4	9990.00	3