MySQL - Ejercicios sobre procedimientos almacenados

MySQL - Ejercicios sobre procedimientos almacenados
Ejercicio 1: Crear un Procedimiento *Simple*
Enunciado:
Crea un procedimiento almacenado llamado hola_mundo que, al ejecutarse, seleccione un
mensaje que diga "*Hola mundo*".

```
delimiter $
create procedure hola_mundo()
BEGIN
    select 'hola mundo';
end$

DELIMITER ;
call hola_mundo;
```

Ejercicio 2: Procedimiento con Parámetros de Entrada
Enunciado:
Desarrolla un procedimiento almacenado saludar_usuario que acepte un nombre de usuario
como parámetro y devuelva un saludo personalizado.

```
drop PROCEDURE saludar_usuariodelimiter $
create procedure saludar_usuario(USER varchar(100))
begin
    select concat ('hola ', USER);
end$
delimiter ;

call saludar_usuario('Antonio');
```

Ejercicio 3: Procedimiento con Parámetros de Entrada y Salida
Enunciado:
Escribe un procedimiento almacenado llamado calcular_iva que tome como entrada el precio
de un producto y un porcentaje de IVA, y que devuelva el precio final incluyendo el IVA.
Tabla y datos:

```
CREATE TABLE productos (
producto_id INT AUTO_INCREMENT PRIMARY KEY,
precio DECIMAL(10, 2)
);
INSERT INTO productos (precio) VALUES (100.00);
INSERT INTO productos (precio) VALUES (150.00);
```

```
delimiter $
create procedure calcular_iva (precio decimal (10.2), tasa_iva decimal
(2.2), out precio_final decimal (10,2))
begin
    set precio_final = precio + (precio * tasa_iva / 100);
end $

delimiter ;

call calcular_iva('100.00','21');
```

Ejercicio 4: Procedimiento con Bucle y Condicional
Enunciado:
Crea un procedimiento almacenado listar_precios que reciba un rango de precios mínimo y
máximo y que devuelva una lista de productos cuyos precios estén dentro de ese rango.
Tabla y datos:

```
CREATE TABLE productos (
producto_id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100),
precio DECIMAL(10, 2),
stock INT
);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto A',
120.00, 10);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto B',
80.00, 3);
```

```
delimiter $
create procedure lista_precios (MAX€ decimal (10.2), min€ decimal(10.2))
begin
    select * from productos where precio<=MAX€ and min€<=precio;
end $
delimiter ;


call lista_precios ('120.00','80.00')
```

Ejercicio 5: Procedimiento con *Cursor* y Manejo de Excepciones
Enunciado:
Implementa un procedimiento almacenado actualizar_stock que actualice el stock de los
productos. Si el producto tiene un stock menor a 5, debe incrementarse en 10 unidades.

```
CREATE TABLE productos (
producto_id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100),
precio DECIMAL(10, 2),
stock INT
);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto A',
120.00, 10);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto B',
80.00, 3);

delimiter $
create procedure actualizar_stock()
begin
    declare v_finalizado int default 0;
    declare v_producto_id int;
    declare v_stock_actual int;

    -- declarar el cursor
    declare producto_cursor cursor for
        select producto_id, stock from productos;

    --Declarar el handler de finalización
    declare CONTINUE handler for not FOUND set v_finalizado = 1;
```

```
    Open producto_cursor;

    get_stock: LOOP
    fetch producto_cursor into v_producto_id, v_stock_actual;
    if v_finalizado = 1 then
        LEAVE get_stock
        end if;

    --Condición para incrementar el stock
    if v_stock_actual < 5 then
    update productos set stock = stock + 10 where producto_id =
v_producto_id;
    end if,
    end loop get_stock;

    close producto_cursor;
    end $
    delimiter ;
```

Ejercicio 6: Procedimiento con Parámetros de Entrada y de Salida Múltiples

Enunciado:

Desarrolla un procedimiento almacenado llamado obtener_estadisticas_producto que reciba el ID de un producto y devuelva el precio más alto, el precio más bajo y el precio promedio de ese producto según las ventas históricas.

Tabla y datos:

```
CREATE TABLE ventas (
venta_id INT AUTO_INCREMENT PRIMARY KEY,
producto_id INT,
precio DECIMAL(10, 2)
);
INSERT INTO ventas (producto_id, precio) VALUES (1, 99.99);
INSERT INTO ventas (producto_id, precio) VALUES (1, 199.99);

delimiter $
DELIMITER $
CREATE PROCEDURE obtener_estadisticas_producto (IN producto_id INT, OUT
precio_maximo DECIMAL(10,2), OUT precio_medio DECIMAL(10,2), OUT
precio_minimo DECIMAL(10,2))
BEGIN
    SELECT MAX(precio) INTO precio_maximo FROM ventas;
```

```
    SELECT AVG(precio) INTO precio_medio FROM ventas;
    SELECT MIN(precio) INTO precio_minimo FROM ventas;
END$

DELIMITER ;

call obtener_estadisticas_producto (1, @precio_maximo, @precio_medio,
@precio_minimo);
SELECT @precio_maximo, @precio_medio, @precio_minimo;
```

Ejercicio 7: Procedimiento con Bucle que Modifica Datos en Múltiples Tablas

Enunciado:

Crea un procedimiento almacenado reajustar_precios que aumente el precio de todos los
productos en un porcentaje dado y actualice la fecha de última modificación del precio en otra
tabla de auditoría.

Tabla y datos:

```
CREATE TABLE ventas (
venta_id INT AUTO_INCREMENT PRIMARY KEY,
producto_id INT,
precio DECIMAL(10, 2)
);
CREATE TABLE auditoria_precios (
auditoria_id INT AUTO_INCREMENT PRIMARY KEY,
producto_id INT,
nuevo_precio DECIMAL(10, 2),
fecha_actualizacion DATE
);
INSERT INTO ventas (producto_id, precio) VALUES (1, 99.99);
INSERT INTO ventas (producto_id, precio) VALUES (1, 199.99);

delimiter $
create procedure reajustar_precios (porcentaje decimal (3,2), fecha date)
begin
create loop: reajuste
    UPDATE auditoria_precios set nuevo_precio=(select precio from
ventas)*(@porcentaje);
    UPDATE auditoria_precios set fecha_actualizacion where
fecha_actualizacion=@fecha;
end loop
```

```
-------------------------------------------------------------------------
-----------------------------
Ejercicio 8: Procedimiento con Transacciones y Manejo de Excepciones
Enunciado:
Escribe un procedimiento almacenado procesar_pago que intente procesar un
pago para una
orden de compra. Debe usar transacciones para asegurar que si cualquier
parte del proceso
falla, no se apliquen cambios a la base de datos.
Tablas y datos:
CREATE TABLE ordenes (
id INT AUTO_INCREMENT PRIMARY KEY,
estado VARCHAR(20),
monto_pagado DECIMAL(10, 2)
);
CREATE TABLE transacciones (
transaccion_id INT AUTO_INCREMENT PRIMARY KEY,
orden_id INT,
monto DECIMAL(10, 2),
fecha DATETIME
);
INSERT INTO ordenes (estado, monto_pagado) VALUES ('Pendiente', 0.00);
INSERT INTO ordenes (estado, monto_pagado) VALUES ('Pendiente', 0.00);

DELIMITER $
CREATE PROCEDURE procesar_pago(IN orden_id INT, IN monto_pagado DECIMAL
(10,2))
BEGIN
    DECLARE exit handler FOR sqlexception
    -- Una excepcion es un error, hay de distintos niveles, entre ellos,
los        fatales
    -- Las excepciones: Cuando ocurre un error, en vez de hacer que el
error        se propague, intenta minimizarlo.
    BEGIN
    --Manejo de errores
        ROLLBACK;
    END;

    -- auto commit, cada vez que el sistema encuentra un punto y
coma,      ejecuta.
    -- Para eso, sirve el  Start Transaction, el auto commit ya no
funciona         como antes, si hay dos select, o se ejecutan las dos o
no se ejecutan,  por el contrario, no see ejecuta ninguno de los dos.

    START TRANSACTION;
```

```
    UPDATE ordenes SET estado = 'Pagado', monto_pagado = monto_pagado
WHERE         id = orden_id;
    INSERT INTO transacciones(orden_id, monto, fecha) VALUES
(orden_id,         monto_pagado, NOW());
    -- El comando NOW() pone la fecha y hora dependiendo del uso horario
de         cada país.

    COMMIT;
END$

DELIMITER ;

CALL procesar_pago(123, 299.99);

--------------------------------------------------------------------------
-----------------------------
```

Ejercicio 9: Procedimiento Complejo con Múltiples Cursores y Lógica Condicional
Enunciado:
Implementa un procedimiento almacenado verificar_inventario que revise el stock de cada
producto y, si es necesario, realice pedidos automáticos para aquellos productos cuyo stock
sea inferior a un umbral específico.
Tabla y datos:

```
CREATE TABLE productos (
producto_id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100),
precio DECIMAL(10, 2),
stock INT
);
CREATE TABLE pedidos(
pedido_id INT AUTO_INCREMENT PRIMARY KEY,
producto_id INT,
cantidad INT,
fecha_pedido DATE
);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto A',
120.00, 10);
INSERT INTO productos (nombre, precio, stock) VALUES ('Producto B',
80.00, 3);

DELIMITER //
CREATE PROCEDURE verificar_inventario()
BEGIN
  DECLARE v_finished INTEGER DEFAULT 0;
  DECLARE v_producto_id INT;
  DECLARE v_stock_actual INT;
```

```
  DECLARE v_umbral INT DEFAULT 10;

  DECLARE producto_cursor CURSOR FOR
      SELECT producto_id, stock FROM productos;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

  OPEN producto_cursor;

  inventory_check: LOOP --Esta parte se encarga de comprobar los
productos.
    FETCH producto_cursor INTO v_producto_id, v_stock_actual;
    IF v_finished = 1 THEN
      LEAVE inventory_check;
    END IF;

    IF v_stock_actual < v_umbral THEN --esta parte se encarga de reponer
el número de articulos de un producto.
      INSERT INTO pedidos(producto_id, cantidad, fecha_pedido) VALUES
(v_producto_id, v_umbral - v_stock_actual, NOW());
    END IF;
  END LOOP inventory_check;

  CLOSE producto_cursor;
END//
DELIMITER ;
```

Para ejecutar el procedimiento:

```
CALL verificar_inventario();
```

Ejercicio 10: Procedimiento con Dinámica de Optimización y Análisis
Enunciado:
Desarrolla un procedimiento almacenado optimizar_categorías que analice las ventas por
categoría de producto y reasigne productos a nuevas categorías basándose en su desempeño
de ventas.
Tablas y datos:

```
CREATE TABLE categorias (
id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100)
);
CREATE TABLE productos (
producto_id INT AUTO_INCREMENT PRIMARY KEY,
nombre VARCHAR(100),
categoria_id INT,
```

```
FOREIGN KEY (categoria_id) REFERENCES categorias(id)
);
CREATE TABLE ventas (
venta_id INT AUTO_INCREMENT PRIMARY KEY,
producto_id INT,
monto DECIMAL(10, 2)
);
INSERT INTO categorias (nombre) VALUES ('Básica');
INSERT INTO categorias (nombre) VALUES ('Premium');
INSERT INTO productos (nombre, categoria_id) VALUES ('Producto X', 1);
INSERT INTO productos (nombre, categoria_id) VALUES ('Producto Y', 1);
INSERT INTO ventas (producto_id, monto) VALUES (1, 5000.00);
INSERT INTO ventas (producto_id, monto) VALUES (2, 12000.00);

DELIMITER //
CREATE PROCEDURE optimizar_categorías()
BEGIN
  DECLARE v_finished INTEGER DEFAULT 0;
  DECLARE v_producto_id INT;
  DECLARE v_categoria_id INT;
  DECLARE v_ventas_total DECIMAL(10,2);

  DECLARE ventas_cursor CURSOR FOR
      SELECT producto_id, SUM(monto) FROM ventas GROUP BY producto_id;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET v_finished = 1;

  OPEN ventas_cursor;

  WHILE v_finished = 0 DO
    FETCH ventas_cursor INTO v_producto_id, v_ventas_total;
    SELECT categoria_id INTO v_categoria_id FROM productos WHERE
producto_id = v_producto_id;

    IF v_ventas_total > 10000 THEN
      -- Reasignar a categoría premium
      UPDATE productos SET categoria_id = (SELECT id FROM categorias
WHERE nombre = 'Premium') WHERE producto_id = v_producto_id;
    ELSEIF v_ventas_total BETWEEN 5000 AND 10000 THEN
      -- Reasignar a categoría estándar
      UPDATE productos SET categoria_id = (SELECT id FROM categorias
WHERE nombre = 'Estándar') WHERE producto_id = v_producto_id;
    END IF;
  END WHILE;

  CLOSE ventas_cursor;
END//
DELIMITER ;
```

Para ejecutar el procedimiento:

```
CALL optimizar_categorías();
```