Sesión	***29***
Tema	Implementación de procedimientos almacenados.
Propósito	Fortalecer las capacidades de los participantes respecto de la implementación de procedimientos almacenados para bases de datos de acuerdo a requerimientos pre establecidos, en una sesión práctica.
Fecha	C.18.12.2025
Hora	18:30

Implementación de Procedimientos Almacenados.

Parte I

1. CONCEPTOS BÁSICOS

¿Qué es un procedimiento almacenado?

Es un programa almacenado en el servidor de base de datos que contiene:

- Lógica SQL
- Estructuras de control (IF, CASE, WHILE, etc.)
- Capacidad de recibir parámetros
- Posibilidad de retornar valores o conjuntos de resultados

Ventajas principales:

-- Ejemplo conceptual

1. RENDIMIENTO: Precompilado y almacenado en caché
2. SEGURIDAD: Control granular de permisos (EXECUTE)
3. REDUCCIÓN DE TRÁFICO: Una llamada vs múltiples queries
4. MANTENIBILIDAD: Lógica centralizada
5. TRANSACCIONALIDAD: Operaciones ACID encapsuladas

2. SINTAXIS POR MOTOR DE BASE DE DATOS

MySQL / MariaDB:

```
DELIMITER $$

CREATE PROCEDURE sp_clientes_activos()
BEGIN
    SELECT * FROM clientes
    WHERE estado = 'activo'
    ORDER BY nombre;
END$$

DELIMITER ;

-- Ejecutar
CALL sp_clientes_activos();
```

SQL Server:

```
CREATE PROCEDURE usp_ObtenerPedidos
    @FechaInicio DATE,
    @FechaFin DATE
AS
BEGIN
    SELECT * FROM pedidos
    WHERE fecha BETWEEN @FechaInicio AND @FechaFin;
END;

-- Ejecutar
EXEC usp_ObtenerPedidos '2024-01-01', '2024-12-31';
```

PostgreSQL:

```
CREATE OR REPLACE PROCEDURE registrar_pago(
    p_cliente_id INT,
    p_monto DECIMAL(10,2)
)
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO pagos(cliente_id, monto, fecha)
    VALUES (p_cliente_id, p_monto, CURRENT_DATE);

    UPDATE clientes
    SET saldo = saldo - p_monto
    WHERE id = p_cliente_id;

    COMMIT;
END $$;

-- Ejecutar
CALL registrar_pago(123, 500.00);
```

---

3. CASOS DE USO PRÁCTICOS

Caso 1: Validación compleja + transacción

```
-- MySQL Ejemplo: Procesar orden con validación de inventario
DELIMITER $$

CREATE PROCEDURE procesar_orden(
    IN p_producto_id INT,
    IN p_cantidad INT,
    IN p_cliente_id INT,
    OUT p_mensaje VARCHAR(100)
)
```

```
BEGIN
	DECLARE stock_actual INT;
	DECLARE EXIT HANDLER FOR SQLEXCEPTION
	BEGIN
		ROLLBACK;
		SET p_mensaje = 'Error en el proceso';
	END;

	START TRANSACTION;

	-- Verificar stock
	SELECT cantidad INTO stock_actual
	FROM inventario
	WHERE producto_id = p_producto_id FOR UPDATE;

	IF stock_actual >= p_cantidad THEN
		-- Actualizar inventario
		UPDATE inventario
		SET cantidad = cantidad - p_cantidad
		WHERE producto_id = p_producto_id;

		-- Registrar orden
		INSERT INTO ordenes(producto_id, cliente_id, cantidad, fecha)
		VALUES (p_producto_id, p_cliente_id, p_cantidad, NOW());

		SET p_mensaje = 'Orden procesada exitosamente';
		COMMIT;
	ELSE
		SET p_mensaje = CONCAT('Stock insuficiente. Disponible: ',
stock_actual);
		ROLLBACK;
	END IF;
END$$
DELIMITER ;
-- Uso
CALL procesar_orden(101, 5, 500, @mensaje);
SELECT @mensaje;
```

Caso 2: Reporte analítico con múltiples consultas

```sql
-- SQL Server: Reporte de ventas mensual con resúmenes
CREATE PROCEDURE generar_reporte_ventas
    @año INT,
    @mes INT
AS
BEGIN
    -- 1. Detalle de ventas
    SELECT
        v.fecha,
        c.nombre AS cliente,
        p.nombre AS producto,
        v.cantidad,
        v.total
    FROM ventas v
    JOIN clientes c ON v.cliente_id = c.id
    JOIN productos p ON v.producto_id = p.id
    WHERE YEAR(v.fecha) = @año
        AND MONTH(v.fecha) = @mes
    ORDER BY v.fecha;

    -- 2. Total general
    SELECT
        SUM(total) AS total_mes,
        COUNT(*) AS transacciones,
        AVG(total) AS promedio_venta
    FROM ventas
    WHERE YEAR(fecha) = @año
        AND MONTH(fecha) = @mes;

    -- 3. Top 5 productos
    SELECT TOP 5
        p.nombre,
        SUM(v.cantidad) AS unidades_vendidas,
        SUM(v.total) AS ingresos_generados
    FROM ventas v
```

```
    JOIN productos p ON v.producto_id = p.id
    WHERE YEAR(v.fecha) = @año
        AND MONTH(v.fecha) = @mes
    GROUP BY p.nombre
    ORDER BY ingresos_generados DESC;
END;
```

---

4. BUENAS PRÁCTICAS

Convenciones de nomenclatura:

```
-- Prefijos según tipo
```
- sp_ -- *Procedimiento almacenado (SQL Server)*
- usp_ -- *User stored procedure*
- pr_ -- *Procedimiento*
- pkg_ -- *Paquete (Oracle)*

```
-- Ejemplos claros
sp_Clientes_Insertar
usp_GenerarReporteVentas
pr_calcular_impuesto
```

Seguridad:

```
-- 1. Usar parámetros con tipo específico
CREATE PROCEDURE sp_login
    @usuario VARCHAR(50),  -- TIPO EXPLÍCITO
    @contrasena_hash VARBINARY(64)
AS ...

-- 2. Minimizar permisos
GRANT EXECUTE ON sp_procesar_pago TO rol_cajero;
-- NO dar acceso directo a tablas
```

```
-- 3. Validar inputs
IF @monto <= 0
        RAISERROR('Monto inválido', 16, 1);
```

Optimización:

```
-- 1. Usar SET NOCOUNT ON (SQL Server)
CREATE PROCEDURE sp_ejemplo
AS
BEGIN
        SET NOCOUNT ON; -- Reduce tráfico de red
        -- ... código ...
END

-- 2. Evitar SELECT * (especificar columnas)
SELECT id, nombre, email FROM clientes;

-- 3. Usar transacciones apropiadamente
BEGIN TRY
        BEGIN TRANSACTION;
        -- Operaciones críticas
        COMMIT TRANSACTION;
END TRY
BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;
END CATCH
```

---

5. COMPARATIVA: PROCEDIMIENTOS vs FUNCIONES vs TRIGGERS

Característica	Procedimiento	Función	Trigger
Retorno	Cero/múltiples valores o result sets	Un solo valor	Ninguno
Uso en SELECT	No	Sí	No
Transacciones	Sí (explícitas)	No (implícitas)	Depende del motor
Ejecución	EXEC/CALL	En expresiones	Automática por evento
Parámetros	Input/Output	Solo Input	Ninguno (contexto especial)

Ejemplo comparativo:

```
-- PROCEDIMIENTO: Lógica compleja con transacción
CREATE PROCEDURE sp_transferencia(
    @cuenta_origen INT,
    @cuenta_destino INT,
    @monto DECIMAL(10,2)
)
AS
BEGIN
    BEGIN TRANSACTION;
    UPDATE cuentas SET saldo = saldo - @monto WHERE id =
@cuenta_origen;
    UPDATE cuentas SET saldo = saldo + @monto WHERE id =
@cuenta_destino;
    COMMIT;
END;
```

```
-- FUNCIÓN: Cálculo reutilizable
CREATE FUNCTION calcular_iva(@monto DECIMAL(10,2))
RETURNS DECIMAL(10,2)
AS
BEGIN
    RETURN @monto * 0.16;
END;

-- TRIGGER: Validación automática
CREATE TRIGGER tr_auditar_cambio
ON empleados
AFTER UPDATE
AS
BEGIN
    INSERT INTO auditoria(tabla, accion, fecha)
    VALUES ('empleados', 'UPDATE', GETDATE());
END;
```

---

6. PATRONES AVANZADOS

Patrón: Cursor para procesamiento fila por fila

```
-- SQL Server: Procesamiento batch con cursor
CREATE PROCEDURE sp_actualizar_precios_categoria
    @categoria_id INT,
    @incremento DECIMAL(5,2)
AS
BEGIN
    DECLARE @producto_id INT;
    DECLARE @precio_actual DECIMAL(10,2);

    DECLARE cur_productos CURSOR FOR
    SELECT id, precio FROM productos
    WHERE categoria_id = @categoria_id;
```

```
    OPEN cur_productos;

    FETCH NEXT FROM cur_productos INTO @producto_id, @precio_actual;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        UPDATE productos
        SET precio = @precio_actual * (1 + @incremento/100)
        WHERE id = @producto_id;

        FETCH NEXT FROM cur_productos INTO @producto_id,
@precio_actual;
    END;

    CLOSE cur_productos;
    DEALLOCATE cur_productos;
END;
```

Patrón: Tabla temporal para procesamiento intermedio

```
-- MySQL: Reporte complejo con tabla temporal
CREATE PROCEDURE generar_reporte_comision()
BEGIN
    -- Crear tabla temporal
    CREATE TEMPORARY TABLE temp_comisiones (
        vendedor_id INT,
        ventas_totales DECIMAL(10,2),
        comision DECIMAL(10,2)
    );

    -- Calcular ventas
    INSERT INTO temp_comisiones (vendedor_id, ventas_totales)
    SELECT vendedor_id, SUM(total)
    FROM ventas
    WHERE fecha >= DATE_SUB(NOW(), INTERVAL 30 DAY)
    GROUP BY vendedor_id;
```

```
    -- Calcular comisión (escalonada)
    UPDATE temp_comisiones
    SET comision = CASE
        WHEN ventas_totales > 10000 THEN ventas_totales * 0.10
        WHEN ventas_totales > 5000 THEN ventas_totales * 0.07
        ELSE ventas_totales * 0.05
    END;

    -- Resultado final
    SELECT
        v.nombre,
        t.ventas_totales,
        t.comision
    FROM temp_comisiones t
    JOIN vendedores v ON t.vendedor_id = v.id;

    -- Limpiar temporal
    DROP TEMPORARY TABLE temp_comisiones;
END;
```

EJERCICIO PRÁCTICO INTEGRADOR

```
-- Sistema de biblioteca: Préstamo con validaciones
CREATE PROCEDURE prestar_libro(
    IN p_libro_id INT,
    IN p_usuario_id INT,
    OUT p_codigo_error INT,
    OUT p_mensaje VARCHAR(200)
)
BEGIN
    DECLARE v_disponibles INT;
    DECLARE v_prestamos_activos INT;
    DECLARE v_fecha_devolucion DATE;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
```

```
    BEGIN
        SET p_codigo_error = 99;
        SET p_mensaje = 'Error interno del sistema';
        ROLLBACK;
    END;

    START TRANSACTION;

    -- 1. Verificar disponibilidad (con bloqueo)
    SELECT ejemplares_disponibles INTO v_disponibles
    FROM libros WHERE id = p_libro_id FOR UPDATE;

    IF v_disponibles <= 0 THEN
        SET p_codigo_error = 1;
        SET p_mensaje = 'Libro no disponible';
        ROLLBACK;
        RETURN;
    END IF;

    -- 2. Verificar préstamos activos del usuario
    SELECT COUNT(*) INTO v_prestamos_activos
    FROM prestamos
    WHERE usuario_id = p_usuario_id
        AND fecha_devolucion IS NULL;

    IF v_prestamos_activos >= 3 THEN
        SET p_codigo_error = 2;
        SET p_mensaje = 'Límite de préstamos alcanzado';
        ROLLBACK;
        RETURN;
    END IF;

    -- 3. Registrar préstamo
    SET v_fecha_devolucion = DATE_ADD(CURDATE(), INTERVAL 15 DAY);

    INSERT INTO prestamos(libro_id, usuario_id, fecha_prestamo,
fecha_devolucion_estimada)
    VALUES (p_libro_id, p_usuario_id, CURDATE(), v_fecha_devolucion);
```

```
    -- 4. Actualizar inventario
    UPDATE libros
    SET ejemplares_disponibles = ejemplares_disponibles - 1
    WHERE id = p_libro_id;

    -- 5. Confirmar
    SET p_codigo_error = 0;
    SET p_mensaje = CONCAT('Préstamo exitoso. Devolver antes de: ',
v_fecha_devolucion);

    COMMIT;
END;
```

RECOMENDACIONES FINALES

- **Versiona tus procedimientos** - Usa scripts de migración
- **Documenta** - Comentarios sobre propósito, parámetros y autor
- **Testea** - Casos de éxito y error
- **Monitoriza** - Performance con herramientas del motor
- **Evita lógica de negocio compleja** - Mantén equilibrio con capa de aplicación

BIBLIOGRAFÍA

- "Fundamentos de Bases de Datos" de Abraham Silberschatz, Henry F. Korth y S. Sudarshan
- "Sistemas de Bases de Datos: un enfoque práctico" de Thomas M. Connolly y Carolyn Begg
- "Desarrollo de Bases de Datos: casos prácticos desde el análisis a la implementación" de Dolores Cuadra, Elena Castro, Ana M. Iglesias.
- "Tecnología y Diseño de Bases de Datos" de Marcos, C. Calero y B. Vela
- https://docs.oracle.com/en/database/