# CAPÍTULO 4: PROGRAMACIÓN PERSISTENTE

## 4.3 PROCEDIMIENTOS ALMACENADOS

*(Tablas Temporales, Ciclos de Repetición, Cursores)*

### 4.3.1 Fundamentos Teóricos

#### Definición

Un Procedimiento Almacenado (Stored Procedure) es un conjunto de instrucciones SQL y de control de flujo que se almacenan en la base de datos como un objeto ejecutable. A diferencia de las funciones, los procedimientos almacenados:

- **NO necesariamente retornan un valor**
- Pueden modificar datos (INSERT, UPDATE, DELETE)
- Pueden ejecutar lógica de negocio compleja
- Aceptan parámetros de entrada, salida y entrada/salida
- Son compilados y optimizados por el DBMS

#### Lenguajes Procedurales por Gestor de Base de Datos

Cada sistema de gestión de bases de datos (DBMS) implementa su propio lenguaje procedural para extender las capacidades de SQL estándar. Estos lenguajes permiten crear procedimientos almacenados, funciones, triggers y otros objetos programables con estructuras de control de flujo, manejo de excepciones y lógica compleja.

| Gestor de BD | Lenguaje | Características |
|---|---|---|
| **Oracle** | PL/SQL (Procedural Language/SQL) | Lenguaje maduro y robusto. Soporta packages, tipos definidos por usuario, cursores avanzados. |
| **SQL Server** | T-SQL (Transact-SQL) | Extensión de Microsoft. Incluye TRY-CATCH, variables de tabla, CTEs avanzados. |
| **PostgreSQL** | PL/pgSQL | Similar a PL/SQL. También soporta PL/Python, PL/Perl, PL/Java. |
| **MySQL** | SQL/PSM (Persistent Stored Modules) | Sintaxis más simple. Funcionalidad básica de |

| Gestor de BD | Lenguaje | Características |
|---|---|---|
| | | procedimientos y funciones. |
| **IBM DB2** | SQL PL | Basado en estándar SQL/PSM. Compatible con PL/SQL de Oracle. |
| **MariaDB** | SQL/PSM | Compatible con MySQL. Incluye Oracle PL/SQL compatibility mode. |
| **SQLite** | N/A (No soporta) | SQLite no soporta procedimientos almacenados nativamente. |

**Características Comunes:**

- **Variables y tipos de datos:** Todos los lenguajes soportan declaración de variables locales con tipos de datos específicos
- **Control de flujo:** IF-THEN-ELSE, CASE, WHILE, FOR, LOOP según el gestor
- **Manejo de excepciones:** EXCEPTION (Oracle), TRY-CATCH (SQL Server), DECLARE...HANDLER (MySQL)
- **Cursores:** Todos soportan cursores para procesamiento fila por fila
- **Transacciones:** COMMIT, ROLLBACK, SAVEPOINT en todos los gestores

**Diferencias Principales:**

- **Sintaxis:** PL/SQL usa := para asignación, T-SQL usa SET o SELECT, PostgreSQL acepta ambos
- **Estructuras avanzadas:** Oracle ofrece PACKAGES para agrupar procedimientos, SQL Server tiene módulos CLR
- **Optimización:** Cada gestor tiene sus propias técnicas de optimización y compilación
- **Portabilidad:** El código no es directamente portable entre gestores sin adaptaciones

**Nota importante:** En este curso nos enfocaremos principalmente en **PL/SQL (Oracle)** y **T-SQL (SQL Server)**, ya que son los más utilizados en el entorno empresarial ecuatoriano y en instituciones públicas.

## Características Principales

1. **Encapsulación:** Agrupa lógica de negocio compleja
2. **Seguridad:** Control de acceso granular a datos
3. **Rendimiento:** Pre-compilados y en caché
4. **Reducción de tráfico:** Ejecutan múltiples operaciones en el servidor
5. **Mantenibilidad:** Cambios centralizados sin modificar aplicaciones
6. **Transaccionalidad:** Soporte completo de ACID

## Diferencias: Procedimientos vs Funciones

| Aspecto | Procedimientos | Funciones |
|---|---|---|
| **Retorno** | Opcional (OUT params) | Obligatorio (RETURN) |
| **Uso en SELECT** | NO | SÍ |
| **Modificación datos** | SÍ | Limitado |
| **Transacciones** | SÍ (COMMIT/ROLLBACK) | NO |
| **Parámetros OUT** | SÍ | NO (solo RETURN) |
| **Llamada** | EXECUTE/CALL | Dentro de SQL |

### 4.3.2 Sintaxis y Creación

#### Oracle

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
(
    p_param1 IN tipo_dato,
    p_param2 OUT tipo_dato,
    p_param3 IN OUT tipo_dato,
    p_param4 IN tipo_dato DEFAULT valor
)
IS
    -- Declaración de variables locales
    v_variable tipo_dato;
    v_cursor SYS_REFCURSOR;
BEGIN
    -- Cuerpo del procedimiento
    -- Lógica de negocio

    COMMIT;  -- Opcional

EXCEPTION
    WHEN exception_name THEN
        ROLLBACK;
        RAISE;
END nombre_procedimiento;
/
```

**Elementos clave:**

- **`IN`**: Parámetro de entrada (por defecto)
- **`OUT`**: Parámetro de salida
- **`IN OUT`**: Parámetro de entrada y salida
- **`IS/AS`**: Inicia declaraciones
- **`COMMIT/ROLLBACK`**: Control de transacciones
- **`EXCEPTION`**: Manejo de errores

## SQL Server

```
CREATE OR ALTER PROCEDURE nombre_procedimiento
(
    @param1 tipo_dato,              -- IN por defecto
    @param2 tipo_dato OUTPUT,       -- OUT
    @param3 tipo_dato = valor_default
)
AS
BEGIN
    -- Configuración inicial
    SET NOCOUNT ON;  -- Evita mensajes de filas afectadas

    -- Declaración de variables
    DECLARE @variable tipo_dato;

    BEGIN TRY
        -- Lógica de negocio
        BEGIN TRANSACTION;

        -- Operaciones

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        THROW;  -- Re-lanza el error
    END CATCH
END;
```

**Elementos clave:**

- `@`: Prefijo para parámetros y variables
- `OUTPUT`: Marca parámetro de salida
- `SET NOCOUNT ON`: Mejora rendimiento
- `BEGIN TRANSACTION/COMMIT`: Control transaccional
- `TRY...CATCH`: Manejo estructurado de errores

### 4.3.3 Tablas Temporales

Las tablas temporales permiten almacenar datos intermedios durante la ejecución de un procedimiento.

#### Oracle - Tablas Temporales Globales (GTT)

```
-- Crear tabla temporal global
CREATE GLOBAL TEMPORARY TABLE temp_estudiantes
(
    est_id NUMBER,
    nombre VARCHAR2(100),
    promedio NUMBER
)
ON COMMIT DELETE ROWS;  -- Borra datos al hacer COMMIT
-- ON COMMIT PRESERVE ROWS;  -- Mantiene datos después de COMMIT
```

**Características Oracle GTT:**

- Definición permanente, datos temporales
- Visibles solo en la sesión que los creó
- **`ON COMMIT DELETE ROWS`**: Borra al COMMIT (por defecto)
- **`ON COMMIT PRESERVE ROWS`**: Mantiene hasta fin de sesión

## SQL Server - Tablas Temporales

```
CREATE OR ALTER PROCEDURE procesar_estudiantes
AS
BEGIN
    -- Tabla temporal local (# prefijo)
    CREATE TABLE #temp_estudiantes
    (
        est_id INT,
        nombre VARCHAR(100),
        promedio DECIMAL(4,2)
    );

    -- Insertar datos
    INSERT INTO #temp_estudiantes
    SELECT est_id, est_nombre + ' ' + est_apellido, est_promedio
    FROM ESTUDIANTES
    WHERE est_promedio >= 80;

    -- Procesar
    SELECT * FROM #temp_estudiantes;

    -- La tabla se elimina automáticamente al terminar
END;
```

**Tipos de tablas temporales en SQL Server:**

7. **Tablas Temporales Locales (#):** Solo visible en la sesión actual
8. **Tablas Temporales Globales (##):** Visible en todas las sesiones
9. **Variables de Tabla (@):** Almacenadas en memoria, ámbito limitado

## 4.3.4 Ciclos de Repetición

Los ciclos permiten ejecutar bloques de código repetidamente.

### Oracle - Tipos de Ciclos

#### 1. LOOP Básico

```
CREATE OR REPLACE PROCEDURE ejemplo_loop
IS
    v_contador NUMBER := 1;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('Iteración: ' || v_contador);
        v_contador := v_contador + 1;

        EXIT WHEN v_contador > 10;  -- Condición de salida
    END LOOP;
END;
/
```

#### 2. WHILE LOOP

```
CREATE OR REPLACE PROCEDURE ejemplo_while
IS
    v_contador NUMBER := 1;
BEGIN
    WHILE v_contador <= 10 LOOP
        DBMS_OUTPUT.PUT_LINE('Iteración: ' || v_contador);
        v_contador := v_contador + 1;
    END LOOP;
END;
/
```

#### 3. FOR LOOP (Numérico)

```
CREATE OR REPLACE PROCEDURE ejemplo_for
IS
BEGIN
    FOR i IN 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE('Iteración: ' || i);
    END LOOP;

    -- FOR LOOP inverso
    FOR i IN REVERSE 1..10 LOOP
        DBMS_OUTPUT.PUT_LINE('Cuenta regresiva: ' || i);
    END LOOP;
END;
/
```

### 4. FOR LOOP (Cursor)

```
CREATE OR REPLACE PROCEDURE ejemplo_for_cursor
IS
BEGIN
    FOR rec IN (SELECT * FROM ESTUDIANTES WHERE est_promedio >= 80)
    LOOP
        DBMS_OUTPUT.PUT_LINE(rec.est_nombre || ': ' || rec.est_promedio);
    END LOOP;
END;
/
```

## SQL Server - Tipos de Ciclos

### 1. WHILE

```
CREATE OR ALTER PROCEDURE ejemplo_while
AS
BEGIN
    DECLARE @contador INT = 1;

    WHILE @contador <= 10
    BEGIN
        PRINT 'Iteración: ' + CAST(@contador AS VARCHAR);
        SET @contador = @contador + 1;

        -- BREAK: Sale del ciclo
        -- CONTINUE: Salta a la siguiente iteración
    END
END;
```

**Nota:** SQL Server no tiene FOR LOOP nativo. Se usa WHILE o cursores.

### 4.3.5 Cursores

Los cursores permiten procesar conjuntos de resultados fila por fila.

#### Oracle - Cursores

**1. Cursor Implícito (FOR LOOP)**

```
CREATE OR REPLACE PROCEDURE cursor_implicito
IS
BEGIN
    FOR rec IN (SELECT * FROM ESTUDIANTES WHERE est_promedio >= 80)
    LOOP
        -- Oracle maneja automáticamente OPEN, FETCH, CLOSE
        DBMS_OUTPUT.PUT_LINE(rec.est_nombre);
    END LOOP;
END;
/
```

**2. Cursor Explícito**

```
CREATE OR REPLACE PROCEDURE cursor_explicito
IS
    -- Declarar cursor
    CURSOR c_estudiantes IS
        SELECT est_id, est_nombre, est_promedio
        FROM ESTUDIANTES
        WHERE est_promedio >= 80
        ORDER BY est_promedio DESC;

    -- Variables para almacenar datos del cursor
    v_id ESTUDIANTES.est_id%TYPE;
    v_nombre ESTUDIANTES.est_nombre%TYPE;
    v_promedio ESTUDIANTES.est_promedio%TYPE;
BEGIN
    -- Abrir cursor
    OPEN c_estudiantes;

    LOOP
        -- Obtener siguiente fila
        FETCH c_estudiantes INTO v_id, v_nombre, v_promedio;

        -- Salir si no hay más filas
        EXIT WHEN c_estudiantes%NOTFOUND;

        -- Procesar fila
        DBMS_OUTPUT.PUT_LINE(v_nombre || ': ' || v_promedio);
```

```
    END LOOP;

    -- Cerrar cursor
    CLOSE c_estudiantes;
END;
/
```

**Atributos de Cursores Oracle:**

- `%FOUND`: TRUE si la última operación encontró filas
- `%NOTFOUND`: TRUE si no encontró filas
- `%ISOPEN`: TRUE si el cursor está abierto
- `%ROWCOUNT`: Número de filas procesadas

**GESTION DE CURSORES CON LAZO FOR**

```
FOR record_name IN cursor_name LOOP
   statement1;
   statement2;
   . . .
END LOOP;
```

```
SET SERVEROUTPUT ON
DECLARE
  CURSOR emp_cursor IS
   SELECT employee_id, last_name FROM
employees
   WHERE department_id =30;
BEGIN
   FOR emp_record IN emp_cursor LOOP
     DBMS_OUTPUT.PUT_LINE(
emp_record.employee_id
     ||' ' ||emp_record.last_name);
    END LOOP;
END;
/
```

## SQL Server - Cursores

### Cursor Básico

```
CREATE OR ALTER PROCEDURE cursor_basico
AS
BEGIN
    -- Declarar variables
    DECLARE @id INT;
    DECLARE @nombre VARCHAR(100);
    DECLARE @promedio DECIMAL(4,2);

    -- Declarar cursor
    DECLARE c_estudiantes CURSOR FOR
        SELECT est_id, est_nombre, est_promedio
        FROM ESTUDIANTES
        WHERE est_promedio >= 80
        ORDER BY est_promedio DESC;

    -- Abrir cursor
    OPEN c_estudiantes;

    -- Primera lectura
    FETCH NEXT FROM c_estudiantes INTO @id, @nombre, @promedio;

    -- Iterar mientras haya datos
    WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT @nombre + ': ' + CAST(@promedio AS VARCHAR);

        -- Siguiente fila
        FETCH NEXT FROM c_estudiantes INTO @id, @nombre, @promedio;
    END

    -- Cerrar y liberar cursor
    CLOSE c_estudiantes;
    DEALLOCATE c_estudiantes;
END;
```

### Tipos de Cursores SQL Server:

- **FAST_FORWARD:** Solo lectura hacia adelante (más rápido)
- **SCROLL:** Permite movimiento en cualquier dirección
- **STATIC:** Copia temporal de los datos
- **DYNAMIC:** Refleja cambios en tiempo real

### 4.3.8 Ventajas y Desventajas

#### Ventajas

- **Rendimiento:** Pre-compilados y optimizados
- **Seguridad:** Control de acceso granular
- **Reducción de tráfico:** Menos datos entre cliente y servidor
- **Reutilización:** Lógica centralizada
- **Mantenibilidad:** Cambios sin modificar aplicaciones
- **Transaccionalidad:** Control completo de transacciones
- **Modularidad:** Código organizado

#### Desventajas

- **Debugging:** Más difícil de depurar
- **Portabilidad:** Sintaxis específica del DBMS
- **Versionamiento:** Difícil integrar con control de versiones
- **Testing:** Requiere acceso a base de datos
- **Carga en servidor:** Procesamiento en el DBMS
- **Curva de aprendizaje:** Requiere conocer PL/SQL o T-SQL

### 4.3.9 Buenas Prácticas

#### 1. Nomenclatura

Usar prefijos consistentes:

- `SP_nombre_procedimiento:` Procedimiento general
- `SP_INS_nombre:` Insert
- `SP_UPD_nombre:` Update
- `SP_DEL_nombre:` Delete
- `SP_GET_nombre:` Select/Retrieve
- `SP_PROC_nombre:` Procesamiento

#### 2. Manejo de Transacciones

Siempre incluir manejo apropiado de transacciones:

- Iniciar transacción explícitamente
- Hacer COMMIT al completar exitosamente
- Hacer ROLLBACK en caso de error
- Usar bloques TRY-CATCH (SQL Server) o EXCEPTION (Oracle)

#### 3. Uso de Cursores

- Evitar cuando sea posible (usar operaciones basadas en conjuntos)
- Si son necesarios, usar cursores implícitos (más rápidos)
- Siempre cerrar cursores explícitamente
- Usar FAST_FORWARD en SQL Server para mejor rendimiento

#### 4. Tablas Temporales

- Usar para datos intermedios complejos
- Preferir variables de tabla para conjuntos pequeños (SQL Server)
- Limpiar explícitamente si es necesario
- Considerar índices en tablas temporales grandes

#### 5. Parámetros y Validación

- Validar todos los parámetros de entrada
- Usar nombres descriptivos con prefijos (p_ para parámetros, v_ para variables)
- Documentar tipo y propósito de cada parámetro
- Establecer valores por defecto cuando sea apropiado

## EJEMPLOS

## PROCEDIMIENTO 1: SP_ACTUALIZAR_PROMEDIO_ESTUDIANTE

**Descripción:**

Recalcula automáticamente el promedio ponderado de un estudiante basándose en sus matrículas y actualiza el campo est_promedio.

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_ACTUALIZAR_PROMEDIO_ESTUDIANTE(
```

```
    -- Parámetro de entrada: ID del estudiante
    p_est_id IN NUMBER
)
IS
    -- Variables locales
    v_promedio_nuevo NUMBER(4,2);    -- Promedio calculado
    v_total_creditos NUMBER;          -- Total de créditos cursados
    v_nombre_estudiante VARCHAR2(100);-- Nombre para mensajes
BEGIN
    -- PASO 1: Verificar que el estudiante existe
    BEGIN
        SELECT est_nombre || ' ' || est_apellido
        INTO v_nombre_estudiante
        FROM ESTUDIANTES
        WHERE est_id = p_est_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('ERROR: Estudiante con ID ' || p_est_id ||
' no existe');
            RETURN;
    END;

    -- PASO 2: Calcular promedio ponderado
    -- Fórmula: SUM(nota * créditos) / SUM(créditos)
    SELECT
        ROUND(SUM(m.mat_nota * c.curso_creditos) /
              NULLIF(SUM(c.curso_creditos), 0), 2),
        SUM(c.curso_creditos)
    INTO v_promedio_nuevo, v_total_creditos
    FROM MATRICULAS m
    INNER JOIN CURSOS c ON m.curso_id = c.curso_id
    WHERE m.est_id = p_est_id
      -- Solo considerar cursos con nota (cursados completamente)
      AND m.mat_nota IS NOT NULL;

    -- PASO 3: Si no tiene cursos con nota, establecer promedio en 0
    IF v_promedio_nuevo IS NULL THEN
        v_promedio_nuevo := 0;
        DBMS_OUTPUT.PUT_LINE('Estudiante ' || v_nombre_estudiante ||
                             ' no tiene cursos completados.');
    END IF;

    -- PASO 4: Actualizar el promedio en la tabla ESTUDIANTES
```

```
    UPDATE ESTUDIANTES
    SET est_promedio = v_promedio_nuevo
    WHERE est_id = p_est_id;

    -- PASO 5: Confirmar cambios
    COMMIT;

    -- PASO 6: Mostrar resultado
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('Promedio actualizado exitosamente');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Nuevo promedio: ' || v_promedio_nuevo);
    DBMS_OUTPUT.PUT_LINE('Créditos cursados: ' || NVL(v_total_creditos,
0));
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    -- Manejo de errores generales
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLERRM);
END SP_ACTUALIZAR_PROMEDIO_ESTUDIANTE;
/
```

**Forma de Uso:**

```
-- Habilitar salida de mensajes
SET SERVEROUTPUT ON

-- Ejemplo 1: Actualizar promedio de Juan Pérez (ID 1001)
BEGIN
    SP_ACTUALIZAR_PROMEDIO_ESTUDIANTE(1001);
END;
/

-- Ejemplo 2: Actualizar promedios de todos los estudiantes
BEGIN
    FOR est IN (SELECT est_id FROM ESTUDIANTES) LOOP
        SP_ACTUALIZAR_PROMEDIO_ESTUDIANTE(est.est_id);
    END LOOP;
END;
/
```

## PROCEDIMIENTO 2: SP_REGISTRAR_MATRICULA

**Descripción:**

Registra una nueva matrícula con validaciones completas: verifica que el estudiante existe, que el curso existe, que no esté matriculado previamente, y genera automáticamente el ID de matrícula.

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_REGISTRAR_MATRICULA(
    -- Parámetros de entrada
    p_est_id IN NUMBER,          -- ID del estudiante
    p_curso_id IN NUMBER,        -- ID del curso
    p_periodo IN VARCHAR2,       -- Periodo académico (ej: '2024-1')
    -- Parámetro de salida
    p_mat_id OUT NUMBER          -- ID de matrícula generado
)
IS
    -- Variables locales
    v_estudiante_existe NUMBER;
    v_curso_existe NUMBER;
    v_matricula_existe NUMBER;
    v_nombre_estudiante VARCHAR2(100);
    v_nombre_curso VARCHAR2(60);
BEGIN
    -- VALIDACIÓN 1: Verificar que el estudiante existe
    SELECT COUNT(*), MAX(est_nombre || ' ' || est_apellido)
    INTO v_estudiante_existe, v_nombre_estudiante
    FROM ESTUDIANTES
    WHERE est_id = p_est_id;

    IF v_estudiante_existe = 0 THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: El estudiante con ID ' || p_est_id ||
' no existe');
        RETURN;
    END IF;

    -- VALIDACIÓN 2: Verificar que el curso existe
    SELECT COUNT(*), MAX(curso_nombre)
    INTO v_curso_existe, v_nombre_curso
    FROM CURSOS
    WHERE curso_id = p_curso_id;

    IF v_curso_existe = 0 THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: El curso con ID ' || p_curso_id || '
no existe');
```

```
        RETURN;
    END IF;

    -- VALIDACIÓN 3: Verificar que no esté matriculado en el mismo curso y
periodo
    SELECT COUNT(*)
    INTO v_matricula_existe
    FROM MATRICULAS
    WHERE est_id = p_est_id
      AND curso_id = p_curso_id
      AND mat_periodo = p_periodo;

    IF v_matricula_existe > 0 THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: El estudiante ya está matriculado en
este curso');
        RETURN;
    END IF;

    -- PASO 4: Generar nuevo ID de matrícula
    -- Obtener el máximo ID y sumar 1
    SELECT NVL(MAX(mat_id), 3000) + 1
    INTO p_mat_id
    FROM MATRICULAS;

    -- PASO 5: Insertar la matrícula
    INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota,
mat_periodo)
    VALUES (p_mat_id, p_est_id, p_curso_id, NULL, p_periodo);

    -- PASO 6: Confirmar cambios
    COMMIT;

    -- PASO 7: Mostrar confirmación
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('MATRÍCULA REGISTRADA EXITOSAMENTE');
    DBMS_OUTPUT.PUT_LINE('ID Matrícula: ' || p_mat_id);
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Periodo: ' || p_periodo);
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN OTHERS THEN
```

```
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR al registrar matrícula: ' || SQLERRM);
END SP_REGISTRAR_MATRICULA;
/
```

### Forma de Uso:

```
SET SERVEROUTPUT ON

-- Ejemplo 1: Matricular a Juan Pérez en Base de Datos
DECLARE
    v_mat_id NUMBER;
BEGIN
    SP_REGISTRAR_MATRICULA(
        p_est_id => 1001,          -- Juan Pérez
        p_curso_id => 2003,        -- Base de Datos
        p_periodo => '2024-2',     -- Periodo actual
        p_mat_id => v_mat_id       -- Variable de salida
    );
    DBMS_OUTPUT.PUT_LINE('ID generado: ' || v_mat_id);
END;
/
```

## PROCEDIMIENTO 3: SP_LISTAR_ESTUDIANTES_CARRERA

**Descripción:**

Lista todos los estudiantes de una carrera específica usando un cursor explícito. Muestra información detallada y estadísticas generales.

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_LISTAR_ESTUDIANTES_CARRERA(
    -- Parámetro de entrada: nombre de la carrera
    p_carrera IN VARCHAR2
)
IS
    -- Declaración del cursor
    CURSOR cur_estudiantes IS
        SELECT est_id,
               est_nombre || ' ' || est_apellido AS nombre_completo,
               est_creditos,
               est_promedio
        FROM ESTUDIANTES
        WHERE UPPER(est_carrera) = UPPER(p_carrera)
        ORDER BY est_promedio DESC;

    -- Variables para el cursor
    v_est_id ESTUDIANTES.est_id%TYPE;
    v_nombre VARCHAR2(100);
    v_creditos ESTUDIANTES.est_creditos%TYPE;
    v_promedio ESTUDIANTES.est_promedio%TYPE;

    -- Variables de estadísticas
    v_contador NUMBER := 0;
    v_suma_promedios NUMBER := 0;
    v_suma_creditos NUMBER := 0;
BEGIN
    -- Encabezado del reporte
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('ESTUDIANTES DE ' || UPPER(p_carrera));
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('');

    -- Abrir cursor y procesar estudiantes
    OPEN cur_estudiantes;

    LOOP
        -- Obtener siguiente registro
```

```
        FETCH cur_estudiantes INTO v_est_id, v_nombre, v_creditos,
v_promedio;

        -- Salir cuando no haya más registros
        EXIT WHEN cur_estudiantes%NOTFOUND;

        -- Incrementar contador
        v_contador := v_contador + 1;

        -- Acumular estadísticas
        v_suma_promedios := v_suma_promedios + NVL(v_promedio, 0);
        v_suma_creditos := v_suma_creditos + NVL(v_creditos, 0);

        -- Mostrar información del estudiante
        DBMS_OUTPUT.PUT_LINE(v_contador || '. ' || v_nombre);
        DBMS_OUTPUT.PUT_LINE('   ID: ' || v_est_id ||
                             ' | Créditos: ' || v_creditos ||
                             ' | Promedio: ' || v_promedio);
        DBMS_OUTPUT.PUT_LINE('');
    END LOOP;

    -- Cerrar cursor
    CLOSE cur_estudiantes;

    -- Mostrar estadísticas
    IF v_contador = 0 THEN
        DBMS_OUTPUT.PUT_LINE('No hay estudiantes en la carrera ' ||
p_carrera);
    ELSE
        DBMS_OUTPUT.PUT_LINE('========================================');
        DBMS_OUTPUT.PUT_LINE('ESTADÍSTICAS GENERALES');
        DBMS_OUTPUT.PUT_LINE('----------------------------------------');
        DBMS_OUTPUT.PUT_LINE('Total estudiantes: ' || v_contador);
        DBMS_OUTPUT.PUT_LINE('Promedio general: ' ||
                             ROUND(v_suma_promedios / v_contador, 2));
        DBMS_OUTPUT.PUT_LINE('Créditos promedio: ' ||
                             ROUND(v_suma_creditos / v_contador, 0));
        DBMS_OUTPUT.PUT_LINE('========================================');
    END IF;

EXCEPTION
    WHEN OTHERS THEN
        -- Asegurar que el cursor se cierre en caso de error
```

```
        IF cur_estudiantes%ISOPEN THEN
            CLOSE cur_estudiantes;
        END IF;
        DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLERRM);
END SP_LISTAR_ESTUDIANTES_CARRERA;
/
```

### Forma de Uso:

```
SET SERVEROUTPUT ON

-- Ejemplo 1: Listar estudiantes de Sistemas
BEGIN
    SP_LISTAR_ESTUDIANTES_CARRERA('Sistemas');
END;
/

-- Ejemplo 2: Listar estudiantes de Electrónica
EXEC SP_LISTAR_ESTUDIANTES_CARRERA('Electrónica');
```

## PROCEDIMIENTO 4: SP_GENERAR_REPORTE_CURSO

**Descripción:**

Genera un reporte completo de estadísticas de un curso específico: total de matrículas, promedio de notas, tasa de aprobación, mejor y peor calificación.

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_GENERAR_REPORTE_CURSO(
    -- Parámetro de entrada: ID del curso
    p_curso_id IN NUMBER
)
IS
    -- Variables para información del curso
    v_nombre_curso VARCHAR2(60);
    v_creditos NUMBER;
    v_departamento VARCHAR2(30);

    -- Variables para estadísticas
    v_total_matriculas NUMBER;
    v_total_con_nota NUMBER;
    v_promedio_notas NUMBER;
    v_nota_maxima NUMBER;
    v_nota_minima NUMBER;
    v_aprobados NUMBER;
    v_reprobados NUMBER;
    v_tasa_aprobacion NUMBER;
BEGIN
    -- PASO 1: Obtener información del curso
    BEGIN
        SELECT curso_nombre, curso_creditos, curso_departamento
        INTO v_nombre_curso, v_creditos, v_departamento
        FROM CURSOS
        WHERE curso_id = p_curso_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('ERROR: El curso con ID ' || p_curso_id ||
' no existe');
            RETURN;
    END;

    -- PASO 2: Calcular estadísticas
    SELECT
        COUNT(*) AS total,
        COUNT(mat_nota) AS con_nota,
```

```
        ROUND(AVG(mat_nota), 2) AS promedio,
        MAX(mat_nota) AS maxima,
        MIN(mat_nota) AS minima,
        COUNT(CASE WHEN mat_nota >= 70 THEN 1 END) AS aprobados,
        COUNT(CASE WHEN mat_nota < 70 THEN 1 END) AS reprobados
    INTO v_total_matriculas, v_total_con_nota, v_promedio_notas,
         v_nota_maxima, v_nota_minima, v_aprobados, v_reprobados
    FROM MATRICULAS
    WHERE curso_id = p_curso_id;

    -- PASO 3: Calcular tasa de aprobación
    IF v_total_con_nota > 0 THEN
        v_tasa_aprobacion := ROUND((v_aprobados / v_total_con_nota) * 100,
2);
    ELSE
        v_tasa_aprobacion := 0;
    END IF;

    -- PASO 4: Generar reporte
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('REPORTE DEL CURSO');
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Departamento: ' || v_departamento);
    DBMS_OUTPUT.PUT_LINE('Créditos: ' || v_creditos);
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('ESTADÍSTICAS DE MATRÍCULAS');
    DBMS_OUTPUT.PUT_LINE('----------------------------------------');
    DBMS_OUTPUT.PUT_LINE('Total matrículas: ' || v_total_matriculas);
    DBMS_OUTPUT.PUT_LINE('Con nota: ' || v_total_con_nota);
    DBMS_OUTPUT.PUT_LINE('En curso: ' || (v_total_matriculas -
v_total_con_nota));
    DBMS_OUTPUT.PUT_LINE('');

    IF v_total_con_nota > 0 THEN
        DBMS_OUTPUT.PUT_LINE('ESTADÍSTICAS DE NOTAS');
        DBMS_OUTPUT.PUT_LINE('----------------------------------------');
        DBMS_OUTPUT.PUT_LINE('Promedio: ' || v_promedio_notas);
        DBMS_OUTPUT.PUT_LINE('Nota máxima: ' || v_nota_maxima);
        DBMS_OUTPUT.PUT_LINE('Nota mínima: ' || v_nota_minima);
        DBMS_OUTPUT.PUT_LINE('Aprobados: ' || v_aprobados);
        DBMS_OUTPUT.PUT_LINE('Reprobados: ' || v_reprobados);
```

```
        DBMS_OUTPUT.PUT_LINE('Tasa aprobación: ' || v_tasa_aprobacion ||
'%');
    ELSE
        DBMS_OUTPUT.PUT_LINE('No hay calificaciones registradas aún.');
    END IF;

    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLERRM);
END SP_GENERAR_REPORTE_CURSO;
/
```

### Forma de Uso:

```
SET SERVEROUTPUT ON

-- Ejemplo: Reporte del curso 'Fundamentos de Programación'
EXEC SP_GENERAR_REPORTE_CURSO(2001);
```

## PROCEDIMIENTO 5: SP_CALCULAR_CREDITOS_APROBADOS

**Descripción:**

Calcula y actualiza los créditos aprobados de un estudiante (cursos con nota >= 70) y los almacena en el campo est_creditos.

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_CALCULAR_CREDITOS_APROBADOS(
    -- Parámetro de entrada: ID del estudiante
    p_est_id IN NUMBER
)
IS
    -- Variables locales
    v_creditos_aprobados NUMBER := 0;
    v_nombre_estudiante VARCHAR2(100);
    v_creditos_antiguos NUMBER;
BEGIN
    -- PASO 1: Verificar que el estudiante existe y obtener datos
    BEGIN
        SELECT est_nombre || ' ' || est_apellido,
               est_creditos
        INTO v_nombre_estudiante, v_creditos_antiguos
        FROM ESTUDIANTES
        WHERE est_id = p_est_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.PUT_LINE('ERROR: Estudiante no encontrado');
            RETURN;
    END;

    -- PASO 2: Calcular créditos aprobados
    -- Solo cursos con nota >= 70 (aprobados)
    SELECT NVL(SUM(c.curso_creditos), 0)
    INTO v_creditos_aprobados
    FROM MATRICULAS m
    INNER JOIN CURSOS c ON m.curso_id = c.curso_id
    WHERE m.est_id = p_est_id
      -- Condiciones: nota registrada Y nota aprobatoria
      AND m.mat_nota IS NOT NULL
      AND m.mat_nota >= 70;

    -- PASO 3: Actualizar el campo est_creditos
    UPDATE ESTUDIANTES
    SET est_creditos = v_creditos_aprobados
```

```
    WHERE est_id = p_est_id;

    -- PASO 4: Confirmar cambios
    COMMIT;

    -- PASO 5: Mostrar resultado
    DBMS_OUTPUT.PUT_LINE('=====================================');
    DBMS_OUTPUT.PUT_LINE('CRÉDITOS ACTUALIZADOS');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Créditos anteriores: ' || v_creditos_antiguos);
    DBMS_OUTPUT.PUT_LINE('Créditos aprobados: ' || v_creditos_aprobados);

    IF v_creditos_aprobados - v_creditos_antiguos >= 0 THEN
        DBMS_OUTPUT.PUT_LINE('Diferencia: +' ||
                             (v_creditos_aprobados - v_creditos_antiguos));
    ELSE
        DBMS_OUTPUT.PUT_LINE('Diferencia: ' ||
                             (v_creditos_aprobados - v_creditos_antiguos));
    END IF;
    DBMS_OUTPUT.PUT_LINE('=====================================');

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: ' || SQLERRM);
END SP_CALCULAR_CREDITOS_APROBADOS;
/
```

### Forma de Uso:

```
SET SERVEROUTPUT ON

-- Ejemplo 1: Actualizar créditos de un estudiante específico
EXEC SP_CALCULAR_CREDITOS_APROBADOS(1001);

-- Ejemplo 2: Actualizar créditos de todos los estudiantes
BEGIN
    FOR est IN (SELECT est_id FROM ESTUDIANTES) LOOP
        SP_CALCULAR_CREDITOS_APROBADOS(est.est_id);
    END LOOP;
END;
/
```

# EJEMPLOS MANEJO DE EXCEPCIONES EN PROCEDIMIENTOS

## PROCEDIMIENTO 6: SP_REGISTRAR_NOTA_VALIDADA

**Descripción:**

Registra la nota de un estudiante con validaciones exhaustivas. Demuestra excepciones predefinidas (NO_DATA_FOUND, TOO_MANY_ROWS) y excepciones personalizadas con RAISE_APPLICATION_ERROR.

**Excepciones demostradas:**

- NO_DATA_FOUND: Cuando no existe la matrícula
- TOO_MANY_ROWS: Cuando hay registros duplicados
- RAISE_APPLICATION_ERROR: Validación de rango de nota
- Excepciones personalizadas: nota_invalida, ya_calificado

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_REGISTRAR_NOTA_VALIDADA(
    p_mat_id IN NUMBER,         -- ID de la matrícula
    p_nota IN NUMBER            -- Nota a registrar (0-100)
)
IS
    -- DECLARACIÓN DE EXCEPCIONES PERSONALIZADAS
    -- Se declaran con la palabra clave EXCEPTION
    nota_invalida EXCEPTION;         -- Nota fuera de rango
    ya_calificado EXCEPTION;         -- Matrícula ya tiene nota
    matricula_no_existe EXCEPTION;   -- Matrícula no encontrada

    -- Variables locales
    v_nota_actual NUMBER;
    v_est_id NUMBER;
    v_curso_id NUMBER;
    v_nombre_estudiante VARCHAR2(100);
    v_nombre_curso VARCHAR2(60);
BEGIN
    -- VALIDACIÓN 1: Nota debe estar entre 0 y 100
    IF p_nota < 0 OR p_nota > 100 THEN
        -- RAISE lanza la excepción personalizada
        RAISE nota_invalida;
    END IF;

    -- VALIDACIÓN 2: Verificar que la matrícula existe
    -- Esta consulta puede lanzar NO_DATA_FOUND o TOO_MANY_ROWS
    BEGIN
        SELECT mat_nota, est_id, curso_id
        INTO v_nota_actual, v_est_id, v_curso_id
        FROM MATRICULAS
```

```
        WHERE mat_id = p_mat_id;
    EXCEPTION
        -- NO_DATA_FOUND: Oracle la lanza cuando SELECT INTO no encuentra
datos
        WHEN NO_DATA_FOUND THEN
            RAISE matricula_no_existe;
        -- TOO_MANY_ROWS: Oracle la lanza cuando SELECT INTO devuelve >1
fila
        WHEN TOO_MANY_ROWS THEN
            -- RAISE_APPLICATION_ERROR: genera un error personalizado
            -- Formato: RAISE_APPLICATION_ERROR(código_error, mensaje)
            -- Código debe estar entre -20000 y -20999
            RAISE_APPLICATION_ERROR(-20001,
                'ERROR: Existen múltiples matrículas con ID ' || p_mat_id
||
                '. Contacte al administrador.');
    END;

    -- VALIDACIÓN 3: Verificar que no tenga nota registrada
    IF v_nota_actual IS NOT NULL THEN
        RAISE ya_calificado;
    END IF;

    -- VALIDACIÓN 4: Obtener nombres para el mensaje
    SELECT e.est_nombre || ' ' || e.est_apellido, c.curso_nombre
    INTO v_nombre_estudiante, v_nombre_curso
    FROM ESTUDIANTES e, CURSOS c
    WHERE e.est_id = v_est_id
      AND c.curso_id = v_curso_id;

    -- PASO 5: Registrar la nota
    UPDATE MATRICULAS
    SET mat_nota = p_nota
    WHERE mat_id = p_mat_id;

    COMMIT;

    -- Mensaje de éxito
    DBMS_OUTPUT.PUT_LINE('=======================================');
    DBMS_OUTPUT.PUT_LINE('NOTA REGISTRADA EXITOSAMENTE');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Nota: ' || p_nota);
```

```
    IF p_nota >= 70 THEN
        DBMS_OUTPUT.PUT_LINE('Estado: APROBADO');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Estado: REPROBADO');
    END IF;
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    -- MANEJO DE EXCEPCIONES PERSONALIZADAS
    WHEN nota_invalida THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: La nota debe estar entre 0 y 100');
        DBMS_OUTPUT.PUT_LINE('Nota ingresada: ' || p_nota);

    WHEN ya_calificado THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: Esta matrícula ya tiene nota
registrada');
        DBMS_OUTPUT.PUT_LINE('Nota actual: ' || v_nota_actual);
        DBMS_OUTPUT.PUT_LINE('Use SP_MODIFICAR_NOTA para cambiar la
calificación');

    WHEN matricula_no_existe THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: No existe matrícula con ID ' ||
p_mat_id);

    -- OTHERS captura cualquier otra excepción no manejada
    -- SQLCODE: código numérico del error
    -- SQLERRM: mensaje descriptivo del error
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO:');
        DBMS_OUTPUT.PUT_LINE('Código: ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('Mensaje: ' || SQLERRM);
END SP_REGISTRAR_NOTA_VALIDADA;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Caso exitoso
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 85.5);
```

```
-- Probará excepción: nota_invalida
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 150);

-- Probará excepción: matricula_no_existe
EXEC SP_REGISTRAR_NOTA_VALIDADA(9999, 75);

-- Probará excepción: ya_calificado (ejecutar 2 veces)
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 80);
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 90);  -- Segunda vez = ERROR
```

## PROCEDIMIENTO 7: SP_TRANSFERIR_ESTUDIANTE

**Descripción:**

Transfiere un estudiante de una carrera a otra. Demuestra el uso de transacciones con SAVEPOINT, manejo de múltiples excepciones y validaciones de integridad referencial.

**Excepciones demostradas:**

• Uso de SAVEPOINT y ROLLBACK TO SAVEPOINT
• Excepciones personalizadas múltiples
• Validación de datos antes de actualizar
• Log de auditoría en caso de error

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_TRANSFERIR_ESTUDIANTE(
    p_est_id IN NUMBER,              -- ID del estudiante
    p_carrera_nueva IN VARCHAR2      -- Nueva carrera
)
IS
    -- Excepciones personalizadas
    estudiante_no_existe EXCEPTION;
    misma_carrera EXCEPTION;
    carrera_invalida EXCEPTION;
    tiene_cursos_en_progreso EXCEPTION;

    -- Variables
    v_carrera_actual VARCHAR2(30);
    v_nombre_estudiante VARCHAR2(100);
    v_cursos_pendientes NUMBER;
    v_carreras_validas NUMBER;
BEGIN
    -- SAVEPOINT: Punto de guardado dentro de la transacción
    -- Permite hacer ROLLBACK parcial si algo falla
    SAVEPOINT inicio_transferencia;

    -- VALIDACIÓN 1: Verificar que el estudiante existe
    BEGIN
        SELECT est_nombre || ' ' || est_apellido, est_carrera
        INTO v_nombre_estudiante, v_carrera_actual
        FROM ESTUDIANTES
        WHERE est_id = p_est_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE estudiante_no_existe;
    END;
```

```
-- VALIDACIÓN 2: Verificar que no sea la misma carrera
IF UPPER(v_carrera_actual) = UPPER(p_carrera_nueva) THEN
    RAISE misma_carrera;
END IF;

-- VALIDACIÓN 3: Verificar que la nueva carrera sea válida
-- En un sistema real, habría una tabla CARRERAS
SELECT COUNT(DISTINCT est_carrera)
INTO v_carreras_validas
FROM ESTUDIANTES
WHERE UPPER(est_carrera) = UPPER(p_carrera_nueva);

IF v_carreras_validas = 0 THEN
    RAISE carrera_invalida;
END IF;

-- VALIDACIÓN 4: Verificar que no tenga cursos en progreso
SELECT COUNT(*)
INTO v_cursos_pendientes
FROM MATRICULAS
WHERE est_id = p_est_id
  AND mat_nota IS NULL;  -- Sin nota = en progreso

IF v_cursos_pendientes > 0 THEN
    RAISE tiene_cursos_en_progreso;
END IF;

-- PASO 5: Realizar la transferencia
UPDATE ESTUDIANTES
SET est_carrera = INITCAP(p_carrera_nueva)
WHERE est_id = p_est_id;

-- Verificar que se actualizó
IF SQL%ROWCOUNT = 0 THEN
    -- SQL%ROWCOUNT: número de filas afectadas por el último DML
    RAISE_APPLICATION_ERROR(-20002,
        'No se pudo actualizar el registro');
END IF;

COMMIT;

-- Mensaje de éxito
```

```
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('TRANSFERENCIA EXITOSA');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Carrera anterior: ' || v_carrera_actual);
    DBMS_OUTPUT.PUT_LINE('Carrera nueva: ' || INITCAP(p_carrera_nueva));
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN estudiante_no_existe THEN
        -- ROLLBACK TO: revierte hasta el SAVEPOINT
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: Estudiante ID ' || p_est_id || ' no
existe');

    WHEN misma_carrera THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: El estudiante ya está en ' ||
v_carrera_actual);

    WHEN carrera_invalida THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: Carrera "' || p_carrera_nueva || '" no
es válida');
        DBMS_OUTPUT.PUT_LINE('Carreras disponibles: Sistemas, Electrónica,
Civil, Mecánica');

    WHEN tiene_cursos_en_progreso THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: No se puede transferir');
        DBMS_OUTPUT.PUT_LINE('El estudiante tiene ' || v_cursos_pendientes
||
                             ' curso(s) en progreso');
        DBMS_OUTPUT.PUT_LINE('Debe completar todos los cursos primero');

    WHEN OTHERS THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO: ' || SQLERRM);
END SP_TRANSFERIR_ESTUDIANTE;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Transferencia exitosa
EXEC SP_TRANSFERIR_ESTUDIANTE(1003, 'Electrónica');
```

```
-- Error: misma carrera
EXEC SP_TRANSFERIR_ESTUDIANTE(1001, 'Sistemas');

-- Error: carrera inválida
EXEC SP_TRANSFERIR_ESTUDIANTE(1001, 'Medicina');
```

## PROCEDIMIENTO 8: SP_ELIMINAR_CURSO_SEGURO

**Descripción:**

Elimina un curso verificando que no tenga dependencias (matrículas). Demuestra manejo de integridad referencial, excepciones de restricción de llave foránea y validaciones complejas.

**Excepciones demostradas:**

• Validación de integridad referencial
• PRAGMA EXCEPTION_INIT (asociar excepciones Oracle a nombres)
• Análisis detallado de dependencias
• Múltiples niveles de validación

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_ELIMINAR_CURSO_SEGURO(
    p_curso_id IN NUMBER
)
IS
    -- Declaración de excepción con PRAGMA EXCEPTION_INIT
    -- Asocia un nombre a un código de error de Oracle
    -- -2292 es el código para violación de llave foránea
    foreign_key_violation EXCEPTION;
    PRAGMA EXCEPTION_INIT(foreign_key_violation, -2292);

    curso_no_existe EXCEPTION;
    tiene_matriculas EXCEPTION;

    -- Variables
    v_nombre_curso VARCHAR2(60);
    v_departamento VARCHAR2(30);
    v_total_matriculas NUMBER;
    v_matriculas_activas NUMBER;
    v_matriculas_historicas NUMBER;
BEGIN
    -- VALIDACIÓN 1: Verificar que el curso existe
    BEGIN
        SELECT curso_nombre, curso_departamento
        INTO v_nombre_curso, v_departamento
        FROM CURSOS
        WHERE curso_id = p_curso_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE curso_no_existe;
    END;
```

```
    -- VALIDACIÓN 2: Análisis detallado de matrículas
    SELECT
        COUNT(*) AS total,
        COUNT(CASE WHEN mat_nota IS NULL THEN 1 END) AS activas,
        COUNT(CASE WHEN mat_nota IS NOT NULL THEN 1 END) AS historicas
    INTO v_total_matriculas, v_matriculas_activas, v_matriculas_historicas
    FROM MATRICULAS
    WHERE curso_id = p_curso_id;

    -- Si tiene matrículas, no se puede eliminar
    IF v_total_matriculas > 0 THEN
        RAISE tiene_matriculas;
    END IF;

    -- PASO 3: Confirmar con el usuario (simulado)
    DBMS_OUTPUT.PUT_LINE('Preparando para eliminar:');
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Departamento: ' || v_departamento);

    -- PASO 4: Eliminar el curso
    DELETE FROM CURSOS
    WHERE curso_id = p_curso_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('CURSO ELIMINADO EXITOSAMENTE');
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN curso_no_existe THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: El curso con ID ' || p_curso_id || '
no existe');

    WHEN tiene_matriculas THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('========================================');
        DBMS_OUTPUT.PUT_LINE('ERROR: NO SE PUEDE ELIMINAR EL CURSO');
        DBMS_OUTPUT.PUT_LINE('========================================');
        DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
        DBMS_OUTPUT.PUT_LINE('Total matrículas: ' || v_total_matriculas);
```

```
        DBMS_OUTPUT.PUT_LINE('  - Activas (sin nota): ' ||
v_matriculas_activas);
        DBMS_OUTPUT.PUT_LINE('  - Históricas (con nota): ' ||
v_matriculas_historicas);
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('SOLUCIÓN:');
        DBMS_OUTPUT.PUT_LINE('Primero debe eliminar todas las matrículas de
este curso');
        DBMS_OUTPUT.PUT_LINE('========================================');

    -- Esta excepción se captura si se intenta DELETE y hay FK
    WHEN foreign_key_violation THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: Violación de integridad referencial');
        DBMS_OUTPUT.PUT_LINE('El curso tiene registros dependientes en
MATRICULAS');

    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Código: ' || SQLCODE);
END SP_ELIMINAR_CURSO_SEGURO;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Error: curso con matrículas
EXEC SP_ELIMINAR_CURSO_SEGURO(2001);

-- Error: curso no existe
EXEC SP_ELIMINAR_CURSO_SEGURO(9999);
```

## PROCEDIMIENTO 9: SP_PROCESAR_LOTE_NOTAS

**Descripción:**

Procesa múltiples notas en un solo procedimiento. Demuestra manejo de excepciones en loops, contador de errores, log de operaciones y procesamiento por lotes con rollback parcial.

**Excepciones demostradas:**

- Manejo de excepciones dentro de loops
- Continuar procesamiento después de errores
- Log de errores y éxitos
- Reporte final de resultados

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_PROCESAR_LOTE_NOTAS(
    p_periodo IN VARCHAR2    -- Periodo a procesar (ej: '2024-1')
)
IS
    -- Cursor para obtener matrículas sin nota
    CURSOR cur_matriculas IS
        SELECT m.mat_id,
               e.est_nombre || ' ' || e.est_apellido AS estudiante,
               c.curso_nombre
        FROM MATRICULAS m
        INNER JOIN ESTUDIANTES e ON m.est_id = e.est_id
        INNER JOIN CURSOS c ON m.curso_id = c.curso_id
        WHERE m.mat_periodo = p_periodo
          AND m.mat_nota IS NULL;

    -- Contadores
    v_total_procesados NUMBER := 0;
    v_exitosos NUMBER := 0;
    v_errores NUMBER := 0;

    -- Variables
    v_nota_aleatoria NUMBER;
    v_mensaje_error VARCHAR2(500);
BEGIN
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('PROCESAMIENTO DE LOTE DE NOTAS');
    DBMS_OUTPUT.PUT_LINE('Periodo: ' || p_periodo);
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('');

    -- Procesar cada matrícula
```

```
    FOR rec IN cur_matriculas LOOP
        v_total_procesados := v_total_procesados + 1;

        -- Bloque BEGIN-EXCEPTION dentro del loop
        -- Permite capturar errores SIN detener el loop
        BEGIN
            -- Generar nota aleatoria entre 60 y 100
            -- DBMS_RANDOM.VALUE(min, max) genera número aleatorio
            v_nota_aleatoria := ROUND(DBMS_RANDOM.VALUE(60, 100), 2);

            -- Registrar nota
            UPDATE MATRICULAS
            SET mat_nota = v_nota_aleatoria
            WHERE mat_id = rec.mat_id;

            v_exitosos := v_exitosos + 1;

            DBMS_OUTPUT.PUT_LINE('✓ ' || rec.estudiante || ' - ' ||
                                 rec.curso_nombre || ' = ' ||
v_nota_aleatoria);

        EXCEPTION
            -- Si hay error en UNA matrícula, se registra y continúa
            WHEN OTHERS THEN
                v_errores := v_errores + 1;
                v_mensaje_error := SQLERRM;

                DBMS_OUTPUT.PUT_LINE('✗ ERROR: ' || rec.estudiante ||
                                     ' - ' || v_mensaje_error);
                -- CONTINUE hace que el loop siga con el siguiente registro
                CONTINUE;
        END;
    END LOOP;

    -- Si hubo al menos una operación exitosa, hacer COMMIT
    IF v_exitosos > 0 THEN
        COMMIT;
    END IF;

    -- Reporte final
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('RESUMEN DEL PROCESAMIENTO');
```

```
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('Total procesados: ' || v_total_procesados);
    DBMS_OUTPUT.PUT_LINE('Exitosos: ' || v_exitosos);
    DBMS_OUTPUT.PUT_LINE('Errores: ' || v_errores);

    IF v_total_procesados = 0 THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('No hay matrículas pendientes para el periodo
' || p_periodo);
    ELSIF v_errores = 0 THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('✓ Procesamiento completado sin errores');
    ELSE
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('⚠ Procesamiento completado con errores');
    END IF;
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR CRÍTICO EN EL PROCESAMIENTO:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END SP_PROCESAR_LOTE_NOTAS;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Procesar todas las matrículas sin nota del periodo 2024-1
EXEC SP_PROCESAR_LOTE_NOTAS('2024-1');

-- Si no hay matrículas pendientes
EXEC SP_PROCESAR_LOTE_NOTAS('2024-2');
```

## MANEJO DE EXCEPCIONES EN PROCEDIMIENTOS.

## PROCEDIMIENTO 1: SP_REGISTRAR_NOTA_VALIDADA

**Descripción:**

Registra la nota de un estudiante con validaciones exhaustivas. Demuestra excepciones predefinidas (NO_DATA_FOUND, TOO_MANY_ROWS) y excepciones personalizadas con RAISE_APPLICATION_ERROR.

**Excepciones demostradas:**

• NO_DATA_FOUND: Cuando no existe la matrícula
• TOO_MANY_ROWS: Cuando hay registros duplicados
• RAISE_APPLICATION_ERROR: Validación de rango de nota
• Excepciones personalizadas: nota_invalida, ya_calificado

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_REGISTRAR_NOTA_VALIDADA(
    p_mat_id IN NUMBER,         -- ID de la matrícula
    p_nota IN NUMBER            -- Nota a registrar (0-100)
)
IS
    -- DECLARACIÓN DE EXCEPCIONES PERSONALIZADAS
    -- Se declaran con la palabra clave EXCEPTION
    nota_invalida EXCEPTION;         -- Nota fuera de rango
    ya_calificado EXCEPTION;         -- Matrícula ya tiene nota
    matricula_no_existe EXCEPTION;   -- Matrícula no encontrada

    -- Variables locales
    v_nota_actual NUMBER;
    v_est_id NUMBER;
    v_curso_id NUMBER;
    v_nombre_estudiante VARCHAR2(100);
    v_nombre_curso VARCHAR2(60);
BEGIN
    -- VALIDACIÓN 1: Nota debe estar entre 0 y 100
    IF p_nota < 0 OR p_nota > 100 THEN
        -- RAISE lanza la excepción personalizada
        RAISE nota_invalida;
    END IF;

    -- VALIDACIÓN 2: Verificar que la matrícula existe
    -- Esta consulta puede lanzar NO_DATA_FOUND o TOO_MANY_ROWS
    BEGIN
        SELECT mat_nota, est_id, curso_id
        INTO v_nota_actual, v_est_id, v_curso_id
        FROM MATRICULAS
        WHERE mat_id = p_mat_id;
```

```
    EXCEPTION
        -- NO_DATA_FOUND: Oracle la lanza cuando SELECT INTO no encuentra
datos
        WHEN NO_DATA_FOUND THEN
            RAISE matricula_no_existe;
        -- TOO_MANY_ROWS: Oracle la lanza cuando SELECT INTO devuelve >1
fila
        WHEN TOO_MANY_ROWS THEN
            -- RAISE_APPLICATION_ERROR: genera un error personalizado
            -- Formato: RAISE_APPLICATION_ERROR(código_error, mensaje)
            -- Código debe estar entre -20000 y -20999
            RAISE_APPLICATION_ERROR(-20001,
                'ERROR: Existen múltiples matrículas con ID ' || p_mat_id
||
                '. Contacte al administrador.');
    END;

    -- VALIDACIÓN 3: Verificar que no tenga nota registrada
    IF v_nota_actual IS NOT NULL THEN
        RAISE ya_calificado;
    END IF;

    -- VALIDACIÓN 4: Obtener nombres para el mensaje
    SELECT e.est_nombre || ' ' || e.est_apellido, c.curso_nombre
    INTO v_nombre_estudiante, v_nombre_curso
    FROM ESTUDIANTES e, CURSOS c
    WHERE e.est_id = v_est_id
      AND c.curso_id = v_curso_id;

    -- PASO 5: Registrar la nota
    UPDATE MATRICULAS
    SET mat_nota = p_nota
    WHERE mat_id = p_mat_id;

    COMMIT;

    -- Mensaje de éxito
    DBMS_OUTPUT.PUT_LINE('=======================================');
    DBMS_OUTPUT.PUT_LINE('NOTA REGISTRADA EXITOSAMENTE');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Nota: ' || p_nota);
    IF p_nota >= 70 THEN
```

```
        DBMS_OUTPUT.PUT_LINE('Estado: APROBADO');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Estado: REPROBADO');
    END IF;
    DBMS_OUTPUT.PUT_LINE('=====================================');

EXCEPTION
    -- MANEJO DE EXCEPCIONES PERSONALIZADAS
    WHEN nota_invalida THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: La nota debe estar entre 0 y 100');
        DBMS_OUTPUT.PUT_LINE('Nota ingresada: ' || p_nota);

    WHEN ya_calificado THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: Esta matrícula ya tiene nota
registrada');
        DBMS_OUTPUT.PUT_LINE('Nota actual: ' || v_nota_actual);
        DBMS_OUTPUT.PUT_LINE('Use SP_MODIFICAR_NOTA para cambiar la
calificación');

    WHEN matricula_no_existe THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: No existe matrícula con ID ' ||
p_mat_id);

    -- OTHERS captura cualquier otra excepción no manejada
    -- SQLCODE: código numérico del error
    -- SQLERRM: mensaje descriptivo del error
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO:');
        DBMS_OUTPUT.PUT_LINE('Código: ' || SQLCODE);
        DBMS_OUTPUT.PUT_LINE('Mensaje: ' || SQLERRM);
END SP_REGISTRAR_NOTA_VALIDADA;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Caso exitoso
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 85.5);

-- Probará excepción: nota_invalida
```

```
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 150);

-- Probará excepción: matricula_no_existe
EXEC SP_REGISTRAR_NOTA_VALIDADA(9999, 75);

-- Probará excepción: ya_calificado (ejecutar 2 veces)
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 80);
EXEC SP_REGISTRAR_NOTA_VALIDADA(3001, 90);  -- Segunda vez = ERROR
```

## PROCEDIMIENTO 2: SP_TRANSFERIR_ESTUDIANTE

**Descripción:**

Transfiere un estudiante de una carrera a otra. Demuestra el uso de transacciones con SAVEPOINT, manejo de múltiples excepciones y validaciones de integridad referencial.

**Excepciones demostradas:**

• Uso de SAVEPOINT y ROLLBACK TO SAVEPOINT
• Excepciones personalizadas múltiples
• Validación de datos antes de actualizar
• Log de auditoría en caso de error

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_TRANSFERIR_ESTUDIANTE(
    p_est_id IN NUMBER,              -- ID del estudiante
    p_carrera_nueva IN VARCHAR2      -- Nueva carrera
)
IS
    -- Excepciones personalizadas
    estudiante_no_existe EXCEPTION;
    misma_carrera EXCEPTION;
    carrera_invalida EXCEPTION;
    tiene_cursos_en_progreso EXCEPTION;

    -- Variables
    v_carrera_actual VARCHAR2(30);
    v_nombre_estudiante VARCHAR2(100);
    v_cursos_pendientes NUMBER;
    v_carreras_validas NUMBER;
BEGIN
    -- SAVEPOINT: Punto de guardado dentro de la transacción
    -- Permite hacer ROLLBACK parcial si algo falla
    SAVEPOINT inicio_transferencia;

    -- VALIDACIÓN 1: Verificar que el estudiante existe
    BEGIN
        SELECT est_nombre || ' ' || est_apellido, est_carrera
        INTO v_nombre_estudiante, v_carrera_actual
        FROM ESTUDIANTES
        WHERE est_id = p_est_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE estudiante_no_existe;
    END;
```

```
-- VALIDACIÓN 2: Verificar que no sea la misma carrera
IF UPPER(v_carrera_actual) = UPPER(p_carrera_nueva) THEN
    RAISE misma_carrera;
END IF;

-- VALIDACIÓN 3: Verificar que la nueva carrera sea válida
-- En un sistema real, habría una tabla CARRERAS
SELECT COUNT(DISTINCT est_carrera)
INTO v_carreras_validas
FROM ESTUDIANTES
WHERE UPPER(est_carrera) = UPPER(p_carrera_nueva);

IF v_carreras_validas = 0 THEN
    RAISE carrera_invalida;
END IF;

-- VALIDACIÓN 4: Verificar que no tenga cursos en progreso
SELECT COUNT(*)
INTO v_cursos_pendientes
FROM MATRICULAS
WHERE est_id = p_est_id
  AND mat_nota IS NULL;  -- Sin nota = en progreso

IF v_cursos_pendientes > 0 THEN
    RAISE tiene_cursos_en_progreso;
END IF;

-- PASO 5: Realizar la transferencia
UPDATE ESTUDIANTES
SET est_carrera = INITCAP(p_carrera_nueva)
WHERE est_id = p_est_id;

-- Verificar que se actualizó
IF SQL%ROWCOUNT = 0 THEN
    -- SQL%ROWCOUNT: número de filas afectadas por el último DML
    RAISE_APPLICATION_ERROR(-20002,
        'No se pudo actualizar el registro');
END IF;

COMMIT;

-- Mensaje de éxito
```

```
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('TRANSFERENCIA EXITOSA');
    DBMS_OUTPUT.PUT_LINE('Estudiante: ' || v_nombre_estudiante);
    DBMS_OUTPUT.PUT_LINE('Carrera anterior: ' || v_carrera_actual);
    DBMS_OUTPUT.PUT_LINE('Carrera nueva: ' || INITCAP(p_carrera_nueva));
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN estudiante_no_existe THEN
        -- ROLLBACK TO: revierte hasta el SAVEPOINT
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: Estudiante ID ' || p_est_id || ' no
existe');

    WHEN misma_carrera THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: El estudiante ya está en ' ||
v_carrera_actual);

    WHEN carrera_invalida THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: Carrera "' || p_carrera_nueva || '" no
es válida');
        DBMS_OUTPUT.PUT_LINE('Carreras disponibles: Sistemas, Electrónica,
Civil, Mecánica');

    WHEN tiene_cursos_en_progreso THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR: No se puede transferir');
        DBMS_OUTPUT.PUT_LINE('El estudiante tiene ' || v_cursos_pendientes
||
                             ' curso(s) en progreso');
        DBMS_OUTPUT.PUT_LINE('Debe completar todos los cursos primero');

    WHEN OTHERS THEN
        ROLLBACK TO inicio_transferencia;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO: ' || SQLERRM);
END SP_TRANSFERIR_ESTUDIANTE;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Transferencia exitosa
EXEC SP_TRANSFERIR_ESTUDIANTE(1003, 'Electrónica');
```

```
-- Error: misma carrera
EXEC SP_TRANSFERIR_ESTUDIANTE(1001, 'Sistemas');

-- Error: carrera inválida
EXEC SP_TRANSFERIR_ESTUDIANTE(1001, 'Medicina');
```

## PROCEDIMIENTO 3: SP_ELIMINAR_CURSO_SEGURO

**Descripción:**

Elimina un curso verificando que no tenga dependencias (matrículas). Demuestra manejo de integridad referencial, excepciones de restricción de llave foránea y validaciones complejas.

**Excepciones demostradas:**

• Validación de integridad referencial
• PRAGMA EXCEPTION_INIT (asociar excepciones Oracle a nombres)
• Análisis detallado de dependencias
• Múltiples niveles de validación

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_ELIMINAR_CURSO_SEGURO(
    p_curso_id IN NUMBER
)
IS
    -- Declaración de excepción con PRAGMA EXCEPTION_INIT
    -- Asocia un nombre a un código de error de Oracle
    -- -2292 es el código para violación de llave foránea
    foreign_key_violation EXCEPTION;
    PRAGMA EXCEPTION_INIT(foreign_key_violation, -2292);

    curso_no_existe EXCEPTION;
    tiene_matriculas EXCEPTION;

    -- Variables
    v_nombre_curso VARCHAR2(60);
    v_departamento VARCHAR2(30);
    v_total_matriculas NUMBER;
    v_matriculas_activas NUMBER;
    v_matriculas_historicas NUMBER;
BEGIN
    -- VALIDACIÓN 1: Verificar que el curso existe
    BEGIN
        SELECT curso_nombre, curso_departamento
        INTO v_nombre_curso, v_departamento
        FROM CURSOS
        WHERE curso_id = p_curso_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN
            RAISE curso_no_existe;
    END;
```

```
    -- VALIDACIÓN 2: Análisis detallado de matrículas
    SELECT
        COUNT(*) AS total,
        COUNT(CASE WHEN mat_nota IS NULL THEN 1 END) AS activas,
        COUNT(CASE WHEN mat_nota IS NOT NULL THEN 1 END) AS historicas
    INTO v_total_matriculas, v_matriculas_activas, v_matriculas_historicas
    FROM MATRICULAS
    WHERE curso_id = p_curso_id;

    -- Si tiene matrículas, no se puede eliminar
    IF v_total_matriculas > 0 THEN
        RAISE tiene_matriculas;
    END IF;

    -- PASO 3: Confirmar con el usuario (simulado)
    DBMS_OUTPUT.PUT_LINE('Preparando para eliminar:');
    DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
    DBMS_OUTPUT.PUT_LINE('Departamento: ' || v_departamento);

    -- PASO 4: Eliminar el curso
    DELETE FROM CURSOS
    WHERE curso_id = p_curso_id;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('CURSO ELIMINADO EXITOSAMENTE');
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN curso_no_existe THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: El curso con ID ' || p_curso_id || '
no existe');

    WHEN tiene_matriculas THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('========================================');
        DBMS_OUTPUT.PUT_LINE('ERROR: NO SE PUEDE ELIMINAR EL CURSO');
        DBMS_OUTPUT.PUT_LINE('========================================');
        DBMS_OUTPUT.PUT_LINE('Curso: ' || v_nombre_curso);
        DBMS_OUTPUT.PUT_LINE('Total matrículas: ' || v_total_matriculas);
```

```
        DBMS_OUTPUT.PUT_LINE('  - Activas (sin nota): ' ||
v_matriculas_activas);
        DBMS_OUTPUT.PUT_LINE('  - Históricas (con nota): ' ||
v_matriculas_historicas);
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('SOLUCIÓN:');
        DBMS_OUTPUT.PUT_LINE('Primero debe eliminar todas las matrículas de
este curso');
        DBMS_OUTPUT.PUT_LINE('========================================');

    -- Esta excepción se captura si se intenta DELETE y hay FK
    WHEN foreign_key_violation THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR: Violación de integridad referencial');
        DBMS_OUTPUT.PUT_LINE('El curso tiene registros dependientes en
MATRICULAS');

    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR INESPERADO: ' || SQLERRM);
        DBMS_OUTPUT.PUT_LINE('Código: ' || SQLCODE);
END SP_ELIMINAR_CURSO_SEGURO;
/
```

### Ejemplos de Uso:

```
SET SERVEROUTPUT ON

-- Error: curso con matrículas
EXEC SP_ELIMINAR_CURSO_SEGURO(2001);

-- Error: curso no existe
EXEC SP_ELIMINAR_CURSO_SEGURO(9999);
```

## PROCEDIMIENTO 4: SP_PROCESAR_LOTE_NOTAS

**Descripción:**

Procesa múltiples notas en un solo procedimiento. Demuestra manejo de excepciones en loops, contador de errores, log de operaciones y procesamiento por lotes con rollback parcial.

**Excepciones demostradas:**

- Manejo de excepciones dentro de loops
- Continuar procesamiento después de errores
- Log de errores y éxitos
- Reporte final de resultados

**Código del Procedimiento:**

```
CREATE OR REPLACE PROCEDURE SP_PROCESAR_LOTE_NOTAS(
    p_periodo IN VARCHAR2    -- Periodo a procesar (ej: '2024-1')
)
IS
    -- Cursor para obtener matrículas sin nota
    CURSOR cur_matriculas IS
        SELECT m.mat_id,
               e.est_nombre || ' ' || e.est_apellido AS estudiante,
               c.curso_nombre
        FROM MATRICULAS m
        INNER JOIN ESTUDIANTES e ON m.est_id = e.est_id
        INNER JOIN CURSOS c ON m.curso_id = c.curso_id
        WHERE m.mat_periodo = p_periodo
          AND m.mat_nota IS NULL;

    -- Contadores
    v_total_procesados NUMBER := 0;
    v_exitosos NUMBER := 0;
    v_errores NUMBER := 0;

    -- Variables
    v_nota_aleatoria NUMBER;
    v_mensaje_error VARCHAR2(500);
BEGIN
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('PROCESAMIENTO DE LOTE DE NOTAS');
    DBMS_OUTPUT.PUT_LINE('Periodo: ' || p_periodo);
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('');

    -- Procesar cada matrícula
```

```
    FOR rec IN cur_matriculas LOOP
        v_total_procesados := v_total_procesados + 1;

        -- Bloque BEGIN-EXCEPTION dentro del loop
        -- Permite capturar errores SIN detener el loop
        BEGIN
            -- Generar nota aleatoria entre 60 y 100
            -- DBMS_RANDOM.VALUE(min, max) genera número aleatorio
            v_nota_aleatoria := ROUND(DBMS_RANDOM.VALUE(60, 100), 2);

            -- Registrar nota
            UPDATE MATRICULAS
            SET mat_nota = v_nota_aleatoria
            WHERE mat_id = rec.mat_id;

            v_exitosos := v_exitosos + 1;

            DBMS_OUTPUT.PUT_LINE('✓ ' || rec.estudiante || ' - ' ||
                                 rec.curso_nombre || ' = ' ||
v_nota_aleatoria);

        EXCEPTION
            -- Si hay error en UNA matrícula, se registra y continúa
            WHEN OTHERS THEN
                v_errores := v_errores + 1;
                v_mensaje_error := SQLERRM;

                DBMS_OUTPUT.PUT_LINE('✗ ERROR: ' || rec.estudiante ||
                                     ' - ' || v_mensaje_error);
                -- CONTINUE hace que el loop siga con el siguiente registro
                CONTINUE;
        END;
    END LOOP;

    -- Si hubo al menos una operación exitosa, hacer COMMIT
    IF v_exitosos > 0 THEN
        COMMIT;
    END IF;

    -- Reporte final
    DBMS_OUTPUT.PUT_LINE('');
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('RESUMEN DEL PROCESAMIENTO');
```

```
    DBMS_OUTPUT.PUT_LINE('========================================');
    DBMS_OUTPUT.PUT_LINE('Total procesados: ' || v_total_procesados);
    DBMS_OUTPUT.PUT_LINE('Exitosos: ' || v_exitosos);
    DBMS_OUTPUT.PUT_LINE('Errores: ' || v_errores);

    IF v_total_procesados = 0 THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('No hay matrículas pendientes para el periodo
' || p_periodo);
    ELSIF v_errores = 0 THEN
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('✓ Procesamiento completado sin errores');
    ELSE
        DBMS_OUTPUT.PUT_LINE('');
        DBMS_OUTPUT.PUT_LINE('⚠ Procesamiento completado con errores');
    END IF;
    DBMS_OUTPUT.PUT_LINE('========================================');

EXCEPTION
    WHEN OTHERS THEN
        ROLLBACK;
        DBMS_OUTPUT.PUT_LINE('ERROR CRÍTICO EN EL PROCESAMIENTO:');
        DBMS_OUTPUT.PUT_LINE(SQLERRM);
END SP_PROCESAR_LOTE_NOTAS;
/
```

**Ejemplos de Uso:**

```
SET SERVEROUTPUT ON

-- Procesar todas las matrículas sin nota del periodo 2024-1
EXEC SP_PROCESAR_LOTE_NOTAS('2024-1');

-- Si no hay matrículas pendientes
EXEC SP_PROCESAR_LOTE_NOTAS('2024-2');
```

## 4.3.10 Ejercicios Propuestos

**EJERCICIO 1: Procedimientos Básicos**

a) Crear procedimiento que registre una nueva matrícula validando prerrequisitos, cupos disponibles y que no esté ya matriculado
b) Crear procedimiento que calcule y actualice el promedio general de un estudiante
c) Crear procedimiento que registre asistencia de estudiantes a una clase

**EJERCICIO 2: Procedimientos con Cursores**

d) Crear procedimiento que genere un reporte de estudiantes en riesgo académico (promedio < 70) y envíe notificaciones
e) Crear procedimiento que actualice masivamente los estados académicos de todos los estudiantes usando cursor
f) Crear procedimiento que identifique y marque estudiantes elegibles para graduación

**EJERCICIO 3: Procedimientos con Tablas Temporales**

g) Crear procedimiento que genere un ranking temporal de estudiantes por carrera y periodo
h) Crear procedimiento que simule la asignación de horarios usando tabla temporal para validar conflictos
i) Crear procedimiento que genere estadísticas consolidadas por departamento usando tablas temporales

**EJERCICIO 4: Procedimientos con Ciclos**

j) Crear procedimiento que calcule el costo total de matrícula para cada estudiante iterando sobre sus cursos
k) Crear procedimiento que distribuya cupos disponibles entre estudiantes en lista de espera
l) Crear procedimiento que genere códigos únicos secuenciales para nuevos estudiantes

**EJERCICIO 5: Procedimientos Complejos**

m) Crear procedimiento que implemente un sistema de recomendación de cursos basado en historial académico, prerrequisitos y carga actual
n) Crear procedimiento que realice simulación de escenarios: ¿Qué pasaría si obtengo X nota en Y curso?
o) Crear procedimiento que automatice el cierre de periodo académico: calcular promedios finales, actualizar estados, generar reportes y preparar siguiente periodo