CAPÍTULO 4: PROGRAMACIÓN PERSISTENTE

4.2 FUNCIONES (ESCALAR, TABULAR, OTRAS)

4.2.1 Fundamentos Teóricos

Definición

Una Función es un objeto de base de datos que encapsula lógica reutilizable y retorna un valor. Las funciones aceptan parámetros de entrada, realizan operaciones o cálculos, y devuelven un resultado que puede ser un valor escalar (simple) o una tabla completa.

A diferencia de los procedimientos almacenados, las funciones:

- **SIEMPRE retornan un valor**
- Pueden ser utilizadas dentro de consultas SELECT
- No pueden modificar el estado de la base de datos (en funciones puras)
- Son determinísticas o no determinísticas

Características Principales

1. **Reutilización:** Lógica de negocio centralizada
2. **Modularidad:** Código organizado y mantenible
3. **Encapsulación:** Oculta la complejidad de cálculos
4. **Rendimiento:** Reduce duplicación de código
5. **Integración:** Se usan en SELECT, WHERE, JOIN, etc.

4.2.2 Tipos de Funciones

1. Funciones Escalares (Scalar Functions)

Retornan un **único valor** de un tipo de dato específico (NUMBER, VARCHAR2, DATE, etc.)

Características:

- Retornan UN solo valor
- Pueden usarse en expresiones SQL
- Aceptan cero o más parámetros
- Útiles para cálculos, conversiones, validaciones

Ejemplo de uso:

```
SELECT nombre, CALCULAR_EDAD(fecha_nacimiento) AS edad
FROM empleados;
```

2. Funciones Tabulares (Table-Valued Functions)

Retornan una **tabla completa** (conjunto de filas y columnas)

Características:

- Retornan un conjunto de resultados
- Se usan en la cláusula FROM como si fueran tablas
- Pueden reemplazar vistas complejas
- Aceptan parámetros para filtrado dinámico

Tipos en SQL Server:

- **Inline Table-Valued Functions (ITVF):** Una sola sentencia SELECT
- **Multi-Statement Table-Valued Functions (MSTVF):** Múltiples sentencias

Ejemplo de uso:

```
SELECT * FROM OBTENER_ESTUDIANTES_POR_CARRERA('Sistemas')
WHERE promedio > 80;
```

3. Funciones Pipelined (Oracle)

Funciones especiales que retornan datos de forma **incremental**, fila por fila, sin esperar a completar todo el procesamiento.

Características:

- Optimizan memoria y rendimiento
- Procesan grandes volúmenes de datos
- Permiten paralelización
- Usan PIPE ROW para retornar filas

4. Funciones Determinísticas vs No Determinísticas

Determinísticas:

- Siempre retornan el mismo resultado con los mismos parámetros
- Ejemplo: CALCULAR_IVA(100) siempre retorna 12
- Oracle: Declaradas con DETERMINISTIC

No Determinísticas:

- Pueden retornar diferentes valores con los mismos parámetros

- Ejemplo: SYSDATE, DBMS_RANDOM.VALUE
- Dependen de factores externos

4.2.3 Sintaxis y Creación

FUNCIONES ESCALARES

Oracle

```
CREATE [OR REPLACE] FUNCTION nombre_funcion
(
    parametro1 tipo_dato,
    parametro2 tipo_dato DEFAULT valor_default,
    ...
)
RETURN tipo_dato_retorno
[DETERMINISTIC]
IS
    -- Declaración de variables locales
    variable_local tipo_dato;
BEGIN
    -- Lógica de la función
    variable_local := expresion;

    RETURN variable_local;

EXCEPTION
    WHEN exception_name THEN
        RETURN valor_por_defecto;
END nombre_funcion;
/
```

Elementos clave:

- **`OR REPLACE`:** Reemplaza si existe
- **`DETERMINISTIC`:** Indica que es determinística
- **`IS / AS`:** Inicia el bloque de declaración
- **`BEGIN...END`:** Cuerpo de la función
- **`RETURN`:** Obligatorio, retorna el valor
- **`EXCEPTION`:** Manejo de errores (opcional)

SQL Server

```
CREATE OR ALTER FUNCTION nombre_funcion
(
    @parametro1 tipo_dato,
    @parametro2 tipo_dato = valor_default,
    ...
)
RETURNS tipo_dato_retorno
[WITH SCHEMABINDING]
AS
BEGIN
    -- Declaración de variables locales
    DECLARE @variable_local tipo_dato;

    -- Lógica de la función
    SET @variable_local = expresion;

    RETURN @variable_local;
END;
```

Elementos clave:

- **`OR ALTER`:** Modifica si existe (SQL Server 2016+)
- **`@`:** Prefijo obligatorio para parámetros y variables
- **`RETURNS`:** Define tipo de retorno
- **`SCHEMABINDING`:** Liga a objetos referenciados
- **`SET / SELECT`:** Para asignar valores
- **`RETURN`:** Obligatorio

IR A ANEXO1. DATOS DE PRUEBAS

IR A ANEXO2. PARA MODIFICAR TABLA ESTUDIANTE (ANADIR CAMPO est_fecha_nacimiento E INSERTAR DATOS IR A ANEXO)

FUNCIONES TABULARES

Oracle - Función Pipelined

Qué es una función pipelined?

En Oracle, una función pipelined es una función que devuelve un conjunto de resultados de manera pipelined o en "tubería". Esto significa que los resultados de la función pueden ser devueltos gradualmente a medida que se procesan, en lugar de tener que esperar hasta que todo el conjunto de datos haya sido procesado y devuelto de una vez.

```
-- 1. Crear tipo de objeto (registro)
CREATE OR REPLACE TYPE tipo_estudiante AS OBJECT (
    est_id NUMBER,
    nombre_completo VARCHAR2(100),
    promedio NUMBER
);
/
```

```
-- 2. Crear tipo tabla
CREATE OR REPLACE TYPE tabla_estudiantes AS TABLE OF tipo_estudiante;
/
```

```
-- 3. Crear función pipelined
CREATE OR REPLACE FUNCTION obtener_estudiantes_top(p_limite NUMBER)
RETURN tabla_estudiantes PIPELINED
IS
BEGIN
    FOR rec IN (
        SELECT est_id,
               est_nombre || ' ' || est_apellido AS nombre_completo,
               est_promedio
        FROM ESTUDIANTES
        ORDER BY est_promedio DESC
        FETCH FIRST p_limite ROWS ONLY
    )
    LOOP
        PIPE ROW(tipo_estudiante(rec.est_id, rec.nombre_completo,
rec.est_promedio));
    END LOOP;

    RETURN;
END;
/
```

Uso:

```
-- Top 10 TABLA ESTUDIANTES
SELECT * FROM TABLE(obtener_estudiantes_top (10));
```

SQL Server - Inline Table-Valued Function

```
CREATE OR ALTER FUNCTION obtener_estudiantes_top
(
    @limite INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT TOP (@limite)
        est_id,
        est_nombre + ' ' + est_apellido AS nombre_completo,
        est_promedio
    FROM ESTUDIANTES
    ORDER BY est_promedio DESC
);
```

SQL Server - Multi-Statement Table-Valued Function

```
CREATE OR ALTER FUNCTION obtener_estadisticas_carrera
(
    @carrera VARCHAR(30)
)
RETURNS @resultado TABLE
(
    carrera VARCHAR(30),
    total_estudiantes INT,
    promedio_general DECIMAL(4,2),
    mejor_estudiante VARCHAR(100)
)
AS
BEGIN
    INSERT INTO @resultado
    SELECT
        @carrera,
        COUNT(*),
        AVG(est_promedio),
        (SELECT TOP 1 est_nombre + ' ' + est_apellido
         FROM ESTUDIANTES
         WHERE est_carrera = @carrera
         ORDER BY est_promedio DESC)
    FROM ESTUDIANTES
    WHERE est_carrera = @carrera;
```

```
    RETURN;
END;
```

CÓMO EJECUTAR FUNCIONES EN SQL*PLUS

*SQL*Plus es el cliente de línea de comandos de Oracle. A continuación se explica cómo crear y ejecutar funciones.*

1. Conectarse a SQL*Plus

Abrir terminal o símbolo del sistema y ejecutar:

```
sqlplus usuario/contraseña@servidor


-- Ejemplo:
sqlplus bchancusig/bchancusig@localhost:1521/orcl
```

2. Configurar el entorno

Antes de crear funciones, configurar SQL*Plus:

```
SET SERVEROUTPUT ON      -- Habilita salida de DBMS_OUTPUT
SET LINESIZE 200          -- Ancho de línea
SET PAGESIZE 100          -- Líneas por página
SET TIMING ON             -- Muestra tiempo de ejecución
```

3. Crear una función

Escribir el código de la función y finalizar con / en una línea separada:

```
CREATE OR REPLACE FUNCTION FN_FORMATEAR_NOMBRE(
    p_nombre VARCHAR2,
    p_apellido VARCHAR2
) RETURN VARCHAR2
IS
BEGIN
    RETURN UPPER(p_apellido) || ', ' || INITCAP(p_nombre);
END;
/
```

IMPORTANTE:

- El carácter / DEBE estar en una línea separada
- Es necesario para que SQL*Plus compile el código PL/SQL

4. Verificar errores de compilación

Si hay errores, verlos con:

```
SHOW ERRORS


-- O más específicamente:
SHOW ERRORS FUNCTION FN_FORMATEAR_NOMBRE
```

5. Ejecutar la función

Método 1: En una consulta SELECT

```
SELECT FN_FORMATEAR_NOMBRE('Juan', 'Pérez') AS nombre_formateado FROM DUAL;
```

Método 2: Con variables en bloque anónimo

```
DECLARE
    v_nombre_completo VARCHAR2(100);
BEGIN
    v_nombre_completo := FN_FORMATEAR_NOMBRE('Juan', 'Pérez');
    DBMS_OUTPUT.PUT_LINE('Nombre: ' || v_nombre_completo);
END;
/
```

6. Listar funciones existentes

```
SELECT object_name, status FROM USER_OBJECTS
WHERE object_type = 'FUNCTION'
ORDER BY object_name;
```

7. Ver código fuente de una función

```
SELECT text FROM USER_SOURCE
WHERE name = 'FN_FORMATEAR_NOMBRE'
ORDER BY line;
```

8. Eliminar una función

```
DROP FUNCTION FN_FORMATEAR_NOMBRE;
```

4.2.4 Ejemplos Prácticos

EJEMPLO 1: Función Escalar - Calcular Edad

Oracle:

```
CREATE OR REPLACE FUNCTION FN_CALCULAR_EDAD(
    p_fecha_nacimiento DATE    -- Parámetro de entrada: fecha de nacimiento
) RETURN NUMBER                 -- Retorna un número entero (edad en años)
IS
    v_edad NUMBER;              -- Variable local para almacenar la edad
BEGIN
    -- Validación: Si la fecha es NULL, retornar NULL
    IF p_fecha_nacimiento IS NULL THEN
        RETURN NULL;
    END IF;

    -- MONTHS_BETWEEN calcula meses entre dos fechas
```

```
    -- Dividimos entre 12 para convertir meses a años
    -- TRUNC elimina decimales para obtener años completos
    v_edad := TRUNC(MONTHS_BETWEEN(SYSDATE, p_fecha_nacimiento) / 12);

    -- Retornar el valor calculado
    RETURN v_edad;
END FN_CALCULAR_EDAD;
/
```

IMPORTANTE: No olvides el / al final en SQL*Plus

PROBAR LA FUNCIÓN

Prueba 1: Calcular edad de una fecha específica

```
SELECT FN_CALCULAR_EDAD(DATE '2000-05-15') AS edad FROM DUAL;
```

Resultado esperado (en 2025):

```
EDAD
----
  24
```

Prueba 2: Calcular edad de todos los estudiantes

```
SELECT est_id,
       est_nombre || ' ' || est_apellido AS estudiante,
       est_fecha_nacimiento,
       FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       est_carrera
FROM ESTUDIANTES
ORDER BY edad DESC;
```

Prueba 3: Estadísticas por edad

```
SELECT FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY FN_CALCULAR_EDAD(est_fecha_nacimiento)
ORDER BY edad;
```

Prueba 4: Estudiantes mayores de edad específica

```
SELECT est_nombre || ' ' || est_apellido AS estudiante,
       FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       est_promedio
FROM ESTUDIANTES
WHERE FN_CALCULAR_EDAD(est_fecha_nacimiento) >= 24
ORDER BY edad DESC;

-- mayores s 18
SELECT est_nombre,
       FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad
```

```
FROM ESTUDIANTES
WHERE FN_CALCULAR_EDAD(est_fecha_nacimiento) >= 18;
```

SQL Server:

```
CREATE OR ALTER FUNCTION FN_CALCULAR_EDAD
(
    @fecha_nacimiento DATE
)
RETURNS INT
AS
BEGIN
    DECLARE @edad INT;

    SET @edad = DATEDIFF(YEAR, @fecha_nacimiento, GETDATE()) -
                CASE
                    WHEN MONTH(@fecha_nacimiento) > MONTH(GETDATE()) OR
                         (MONTH(@fecha_nacimiento) = MONTH(GETDATE()) AND
                          DAY(@fecha_nacimiento) > DAY(GETDATE()))
                    THEN 1
                    ELSE 0
                END;

    RETURN @edad;
END;
```

```
-- Uso
SELECT est_nombre,
       dbo.FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad
FROM ESTUDIANTES
WHERE dbo.FN_CALCULAR_EDAD(est_fecha_nacimiento) >= 18;
```

EJEMPLO 2: Función Escalar - Estado Académico

Esta función clasifica a los estudiantes según su promedio y créditos:

- EXCELENCIA: promedio >= 90 y créditos >= 120
- HONOR: promedio >= 80 y créditos >= 100
- REGULAR: promedio >= 70 y créditos >= 80
- CONDICIONAL: promedio >= 60
- CRÍTICO: promedio < 60

Oracle:

```
CREATE OR REPLACE FUNCTION FN_ESTADO_ACADEMICO
(
    p_promedio NUMBER,
    p_creditos NUMBER
)
RETURN VARCHAR2
DETERMINISTIC
IS
    v_estado VARCHAR2(50);
BEGIN
    IF p_promedio >= 90 AND p_creditos >= 120 THEN
        v_estado := 'EXCELENCIA';
    ELSIF p_promedio >= 80 AND p_creditos >= 100 THEN
        v_estado := 'HONOR';
    ELSIF p_promedio >= 70 AND p_creditos >= 80 THEN
        v_estado := 'REGULAR';
    ELSIF p_promedio >= 60 THEN
        v_estado := 'CONDICIONAL';
    ELSE
        v_estado := 'CRÍTICO';
    END IF;

    RETURN v_estado;
END FN_ESTADO_ACADEMICO;
/
```

```
-- Uso
SELECT est_nombre,
       est_promedio,
       est_creditos,
       FN_ESTADO_ACADEMICO(est_promedio, est_creditos) AS estado
FROM ESTUDIANTES
ORDER BY est_promedio DESC;
```

Ejemplo 3: Función de Formateo de Cadenas

Formatear nombre completo en formato 'APELLIDO, Nombre'

```
CREATE OR REPLACE FUNCTION FN_FORMATEAR_NOMBRE(
    p_nombre VARCHAR2,      -- Nombre del estudiante
    p_apellido VARCHAR2     -- Apellido del estudiante
) RETURN VARCHAR2            -- Retorna nombre formateado
IS
BEGIN
    -- UPPER convierte apellido a mayúsculas
    -- INITCAP capitaliza primera letra de cada palabra
    -- || es el operador de concatenación en Oracle
    RETURN UPPER(p_apellido) || ', ' || INITCAP(p_nombre);
END FN_FORMATEAR_NOMBRE;
/
```

Pruebas de ejecución:

```
-- Caso 1: Formatear un nombre específico
SELECT FN_FORMATEAR_NOMBRE('Juan', 'Pérez') AS nombre_formateado FROM DUAL;
-- Resultado: PÉREZ, Juan

-- Caso 2: Formatear todos los estudiantes
SELECT est_id,
       FN_FORMATEAR_NOMBRE(est_nombre, est_apellido) AS nombre_completo,
       est_carrera,
       est_promedio
FROM ESTUDIANTES
ORDER BY est_apellido;
```

Ejemplo 4: Función de Clasificación

Clasificar estudiantes según créditos aprobados

```
CREATE OR REPLACE FUNCTION FN_CLASIFICAR_CREDITOS(
    p_creditos NUMBER       -- Créditos aprobados
) RETURN VARCHAR2            -- Retorna nivel del estudiante
IS
BEGIN
    -- Clasificar según créditos acumulados
    -- Típicamente una carrera tiene 180-200 créditos totales
    RETURN CASE
        -- Validación: si es NULL retornar SIN DATOS
        WHEN p_creditos IS NULL THEN 'SIN DATOS'
        -- Más de 150 créditos: último año
        WHEN p_creditos >= 150 THEN 'ÚLTIMO AÑO'
```

```
        -- 100-149 créditos: penúltimo año
        WHEN p_creditos >= 100 THEN 'PENÚLTIMO AÑO'
        -- 50-99 créditos: mitad de carrera
        WHEN p_creditos >= 50 THEN 'MITAD DE CARRERA'
        -- Menos de 50: inicio de carrera
        ELSE 'INICIO DE CARRERA'
    END;
END FN_CLASIFICAR_CREDITOS;
/
```

Uso de la función:

```
-- Ver clasificación de todos los estudiantes
SELECT est_nombre || ' ' || est_apellido AS estudiante,
       est_creditos,
       FN_CLASIFICAR_CREDITOS(est_creditos) AS nivel
FROM ESTUDIANTES
ORDER BY est_creditos DESC;

-- Contar estudiantes por nivel
SELECT FN_CLASIFICAR_CREDITOS(est_creditos) AS nivel,
       COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY FN_CLASIFICAR_CREDITOS(est_creditos);

-- Con ordenamiento
SELECT FN_CLASIFICAR_CREDITOS(est_creditos) AS nivel,
       COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY FN_CLASIFICAR_CREDITOS(est_creditos)
ORDER BY cantidad DESC;
```

Ejemplo 5: Función con Consulta SQL

Calcular promedio ponderado de un estudiante

```
CREATE OR REPLACE FUNCTION FN_CALCULAR_PROMEDIO_PONDERADO(
    p_est_id NUMBER         -- ID del estudiante
) RETURN NUMBER              -- Retorna el promedio ponderado
IS
    v_promedio NUMBER;      -- Variable para almacenar el resultado
BEGIN
    -- Calcular promedio ponderado:
    -- Suma de (nota * créditos) / Suma de créditos
    SELECT
        ROUND(                              -- Redondear a 2 decimales
            SUM(m.mat_nota * c.curso_creditos) /  -- Suma ponderada
            NULLIF(SUM(c.curso_creditos), 0),     -- Total créditos (evita
/0)
            2
        )
    INTO v_promedio                         -- Guardar en variable
    FROM MATRICULAS m
    INNER JOIN CURSOS c ON m.curso_id = c.curso_id
    WHERE m.est_id = p_est_id               -- Filtrar por estudiante
      AND m.mat_nota IS NOT NULL;           -- Solo cursos con nota

    -- NVL convierte NULL a 0 si el estudiante no tiene cursos
    RETURN NVL(v_promedio, 0);

EXCEPTION
    -- Si no se encuentran datos (NO_DATA_FOUND), retornar 0
    WHEN NO_DATA_FOUND THEN
        RETURN 0;
END FN_CALCULAR_PROMEDIO_PONDERADO;
/
```

Uso de la función:

```
-- Comparar promedio simple vs ponderado
SELECT
    est_id,
    est_nombre || ' ' || est_apellido AS estudiante,
    est_promedio AS promedio_simple,
    FN_CALCULAR_PROMEDIO_PONDERADO(est_id) AS promedio_ponderado,
    -- Diferencia entre ambos promedios
    ROUND(est_promedio - FN_CALCULAR_PROMEDIO_PONDERADO(est_id), 2) AS
diferencia
```

```
FROM ESTUDIANTES
ORDER BY promedio_ponderado DESC;
```

Ejemplo 6: Función con CASE

Determinar estado académico según promedio

```
CREATE OR REPLACE FUNCTION FN_ESTADO_ACADEMICO(
    p_promedio NUMBER       -- Promedio del estudiante
) RETURN VARCHAR2            -- Retorna el estado académico
IS
BEGIN
    -- Usar CASE para clasificar según rangos de promedio
    RETURN CASE
        -- Si el promedio es NULL, no hay datos
        WHEN p_promedio IS NULL THEN 'SIN DATOS'
        -- Clasificación por rangos
        WHEN p_promedio >= 85 THEN 'EXCELENTE'  -- 85-100
        WHEN p_promedio >= 70 THEN 'BUENO'      -- 70-84
        WHEN p_promedio >= 60 THEN 'REGULAR'    -- 60-69
        ELSE 'RIESGO'                           -- <60
    END;
END FN_ESTADO_ACADEMICO;
/
```

Ejemplo de uso con agrupación:

```
-- Contar estudiantes por estado académico
SELECT
    FN_ESTADO_ACADEMICO(est_promedio) AS estado,
    COUNT(*) AS cantidad,
    -- Calcular porcentaje
    ROUND(COUNT(*) * 100.0 / (SELECT COUNT(*) FROM ESTUDIANTES), 2) AS
porcentaje
FROM ESTUDIANTES
GROUP BY FN_ESTADO_ACADEMICO(est_promedio)
-- Ordenar por prioridad (Excelente primero, Riesgo al final)
ORDER BY
    CASE FN_ESTADO_ACADEMICO(est_promedio)
        WHEN 'EXCELENTE' THEN 1
        WHEN 'BUENO' THEN 2
        WHEN 'REGULAR' THEN 3
        WHEN 'RIESGO' THEN 4
        ELSE 5
    END;
```

Consulta adicional:

```
-- Ver estudiantes por estado con detalles
SELECT est_nombre || ' ' || est_apellido AS estudiante,
       est_carrera,
```

```
       est_promedio,
       FN_ESTADO_ACADEMICO(est_promedio) AS estado
FROM ESTUDIANTES
ORDER BY est_promedio DESC;
```

Ejemplo 7: Función con Validaciones

Validar formato de email

```
CREATE OR REPLACE FUNCTION FN_VALIDAR_EMAIL(
    p_email VARCHAR2        -- Email a validar
) RETURN VARCHAR2            -- Retorna 'VÁLIDO' o 'INVÁLIDO'
IS
    v_pos_arroba NUMBER;    -- Posición del carácter @
    v_pos_punto NUMBER;     -- Posición del punto después del @
BEGIN
    -- VALIDACIÓN 1: Verificar que no sea NULL o vacío
    IF p_email IS NULL OR LENGTH(TRIM(p_email)) = 0 THEN
        RETURN 'INVÁLIDO';
    END IF;

    -- VALIDACIÓN 2: Buscar posición de @ usando INSTR
    -- INSTR retorna 0 si no encuentra el carácter
    v_pos_arroba := INSTR(p_email, '@');

    -- El @ debe existir y no estar al inicio
    IF v_pos_arroba <= 1 THEN
        RETURN 'INVÁLIDO';
    END IF;

    -- VALIDACIÓN 3: Buscar punto DESPUÉS del @
    -- Empezamos a buscar desde la posición del @
    v_pos_punto := INSTR(p_email, '.', v_pos_arroba);

    -- El punto debe existir y no ser el último carácter
    IF v_pos_punto = 0 OR v_pos_punto = LENGTH(p_email) THEN
        RETURN 'INVÁLIDO';
    END IF;

    -- Si pasó todas las validaciones, es válido
    RETURN 'VÁLIDO';
END FN_VALIDAR_EMAIL;
/
```

Pruebas de validación:

```
SELECT
    -- Caso válido: tiene @ y punto después
    FN_VALIDAR_EMAIL('[email protected]') AS prueba1,

    -- Inválido: tiene @ pero no tiene punto
    FN_VALIDAR_EMAIL('correo@dominio') AS prueba2,

    -- Inválido: no tiene @
    FN_VALIDAR_EMAIL('correo.dominio.com') AS prueba3,

    -- Inválido: @ está al inicio
    FN_VALIDAR_EMAIL('@dominio.com') AS prueba4
FROM DUAL;
```

4.2.8 Diferencias Oracle vs SQL Server

Aspecto	Oracle	SQL Server
Variables	Sin prefijo	Prefijo @ obligatorio
Asignación	:= o SELECT INTO	SET o SELECT
Bloques	BEGIN...END;	BEGIN...END
Funciones tabulares	Requiere tipos + PIPELINED	RETURNS TABLE directa
Concatenación	||	+
Determinística	Cláusula DETERMINISTIC	No disponible
Manejo errores	EXCEPTION block	TRY...CATCH

4.2.6 Ventajas y Desventajas

Ventajas

- **Reutilización:** Código centralizado usado en múltiples lugares
- **Mantenibilidad:** Cambios en un solo lugar
- **Modularidad:** Lógica organizada y estructurada
- **Seguridad:** Encapsula lógica de negocio sensible
- **Rendimiento:** Compiladas y optimizadas por el DBMS
- **Legibilidad:** SQL más limpio y comprensible
- **Validación:** Centraliza reglas de negocio

Desventajas

- **Overhead:** Llamadas a funciones tienen costo computacional
- **Debugging:** Más difícil de depurar que SQL inline
- **Portabilidad:** Sintaxis diferente entre DBMS
- **Restricciones:** Funciones escalares no pueden modificar datos
- **Performance:** Uso excesivo puede impactar rendimiento
- **Complejidad:** Funciones complejas son difíciles de mantener

4.2.7 Buenas Prácticas

1. Nomenclatura

Usar prefijos consistentes:

- `FN_nombre_funcion:` Función escalar
- `FN_TVF_nombre_funcion:` Función tabular
- `FN_GET_nombre:` Funciones de obtención
- `FN_CALC_nombre:` Funciones de cálculo
- `FN_VAL_nombre:` Funciones de validación

2. Documentación

Incluir comentarios descriptivos en cada función:

- Propósito de la función
- Descripción de parámetros
- Tipo de dato retornado
- Autor y fecha de creación
- Ejemplos de uso

3. Performance

- Evitar funciones escalares en WHERE sobre grandes tablas
- Preferir funciones tabulares inline sobre multi-statement
- Marcar funciones como DETERMINISTIC cuando sea posible
- No usar funciones para operaciones simples que SQL hace nativamente

4. Manejo de Errores

- Siempre incluir manejo de excepciones
- Retornar valores apropiados en caso de error (NULL, -1, etc.)
- Documentar comportamiento ante errores

4.2.9 Ejercicios Propuestos

EJERCICIO 1: Funciones Escalares Básicas

a) Crear función que calcule el descuento según créditos matriculados: 0-50 (0%), 51-100 (5%), 101-150 (10%), más de 150 (15%)
b) Crear función que determine si un estudiante puede graduarse (requiere: promedio >= 70 y créditos >= 160)
c) Crear función que calcule el costo total de un curso considerando créditos y precio por crédito

EJERCICIO 2: Funciones Escalares con Consultas

d) Crear función que retorne el número de cursos aprobados por un estudiante
e) Crear función que calcule el porcentaje de asistencia de un estudiante en un periodo
f) Crear función que retorne el ranking de un estudiante dentro de su carrera

EJERCICIO 3: Funciones Tabulares

g) Crear función tabular que liste todos los cursos disponibles para matricular (sin prerrequisitos pendientes) para un estudiante específico
h) Crear función tabular que retorne el top N de estudiantes por promedio en una carrera específica
i) Crear función tabular que genere un reporte de cursos con su tasa de aprobación por periodo

EJERCICIO 4: Funciones Complejas

j) Crear función que simule el cálculo del promedio final si el estudiante obtiene cierta nota en un curso específico
k) Crear función que recomiende cursos a matricular basándose en prerrequisitos, carga académica y promedio
l) Crear función que valide si un estudiante puede matricularse considerando prerrequisitos, cupos y no estar matriculado

EJERCICIO 5: Optimización y Testing

m) Comparar el rendimiento de una función escalar vs. una vista para el mismo cálculo
n) Crear una suite de pruebas para validar todas las funciones creadas con diferentes casos de prueba
o) Documentar todas las funciones creadas con comentarios y casos de uso

--- ANEXOS—

ANEXO1.

DATOS DE PRUEBA – INSERTS

Scripts de Inserción para las Tablas Base

Scripts corregidos según la estructura real de las tablas ESTUDIANTES, CURSOS y MATRICULAS.

ESTRUCTURA DE LAS TABLAS

Tabla ESTUDIANTES

```
CREATE TABLE ESTUDIANTES (
    est_id NUMBER(6) PRIMARY KEY,
    est_nombre VARCHAR2(50),
    est_apellido VARCHAR2(50),
    est_carrera VARCHAR2(30),
    est_creditos NUMBER(3),
    est_promedio NUMBER(4,2)
);
```

Tabla CURSOS

```
CREATE TABLE CURSOS (
    curso_id NUMBER(4) PRIMARY KEY,
    curso_nombre VARCHAR2(60),
    curso_creditos NUMBER(2),
    curso_departamento VARCHAR2(30)
);
```

Tabla MATRICULAS

```
CREATE TABLE MATRICULAS (
    mat_id NUMBER(8) PRIMARY KEY,
    est_id NUMBER(6),
    curso_id NUMBER(4),
    mat_nota NUMBER(4,2),
    mat_periodo VARCHAR2(10),
    FOREIGN KEY (est_id) REFERENCES ESTUDIANTES(est_id),
    FOREIGN KEY (curso_id) REFERENCES CURSOS(curso_id)
);
```

INSERTS – TABLA ESTUDIANTES

12 estudiantes de diferentes carreras

```
-- ========================================
-- TABLA ESTUDIANTES
-- ========================================

-- Estudiantes de Sistemas
INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1001, 'Juan', 'Pérez', 'Sistemas', 120, 85.50);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1002, 'María', 'González', 'Sistemas', 150, 92.30);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1003, 'Miguel', 'Torres', 'Sistemas', 85, 68.30);

-- Estudiantes de Electrónica
INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1004, 'Carlos', 'Rodríguez', 'Electrónica', 110, 78.90);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1005, 'Ana', 'Martínez', 'Electrónica', 140, 88.75);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1006, 'Roberto', 'Vargas', 'Electrónica', 95, 58.20);

-- Estudiantes de Civil
INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1007, 'Luis', 'Fernández', 'Civil', 90, 82.40);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1008, 'Andrea', 'López', 'Civil', 130, 91.20);

INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1009, 'Valentina', 'Morales', 'Civil', 155, 87.90);
```

```
-- Estudiantes de Mecánica
INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1010, 'Diego', 'Sánchez', 'Mecánica', 145, 76.50);


INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1011, 'Sofía', 'Ramírez', 'Mecánica', 135, 95.60);


INSERT INTO ESTUDIANTES (est_id, est_nombre, est_apellido, est_carrera,
est_creditos, est_promedio)
VALUES (1012, 'Daniela', 'Castro', 'Mecánica', 80, 73.80);


COMMIT;
```

INSERTS – TABLA CURSOS

12 cursos de diferentes departamentos

```
-- ========================================
-- TABLA CURSOS
-- ========================================

-- Cursos de Sistemas
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2001, 'Fundamentos de Programación', 5, 'Sistemas');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2002, 'Estructura de Datos', 5, 'Sistemas');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2003, 'Base de Datos', 5, 'Sistemas');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2004, 'Ingeniería de Software', 4, 'Sistemas');

-- Cursos de Electrónica
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2005, 'Circuitos Eléctricos I', 5, 'Electrónica');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2006, 'Electrónica Digital', 5, 'Electrónica');

-- Cursos de Civil
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2007, 'Resistencia de Materiales', 5, 'Civil');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2008, 'Estructuras de Hormigón', 5, 'Civil');

-- Cursos de Mecánica
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2009, 'Termodinámica', 4, 'Mecánica');
```

```
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2010, 'Mecánica de Fluidos', 5, 'Mecánica');

-- Cursos de Matemáticas (compartidos por todas las carreras)
INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2011, 'Cálculo Diferencial', 5, 'Matemáticas');

INSERT INTO CURSOS (curso_id, curso_nombre, curso_creditos,
curso_departamento)
VALUES (2012, 'Cálculo Integral', 5, 'Matemáticas');

COMMIT;
```

INSERTS – TABLA MATRICULAS

26 matrículas con notas variadas

```
-- ========================================
-- TABLA MATRICULAS
-- ========================================

-- Estudiante 1001 – Juan Pérez (Sistemas)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3001, 1001, 2001, 88.50, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3002, 1001, 2011, 82.00, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3003, 1001, 2002, 85.75, '2021-1');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3004, 1001, 2003, NULL, '2021-2');

-- Estudiante 1002 – María González (Sistemas)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3005, 1002, 2001, 95.00, '2019-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3006, 1002, 2002, 91.50, '2020-1');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3007, 1002, 2003, 90.00, '2020-2');

-- Estudiante 1003 – Miguel Torres (Sistemas – bajo rendimiento)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3008, 1003, 2001, 65.00, '2021-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3009, 1003, 2011, 70.50, '2021-2');

-- Estudiante 1004 – Carlos Rodríguez (Electrónica)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3010, 1004, 2005, 75.50, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3011, 1004, 2011, 80.00, '2020-2');
```

```
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3012, 1004, 2006, 78.25, '2021-1');

-- Estudiante 1005 – Ana Martínez (Electrónica)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3013, 1005, 2005, 92.00, '2019-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3014, 1005, 2006, 87.50, '2020-1');

-- Estudiante 1006 – Roberto Vargas (Electrónica – en riesgo)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3015, 1006, 2005, 55.00, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3016, 1006, 2011, 60.00, '2020-2');

-- Estudiante 1007 – Luis Fernández (Civil)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3017, 1007, 2011, 85.00, '2021-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3018, 1007, 2007, 79.50, '2021-2');

-- Estudiante 1008 – Andrea López (Civil)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3019, 1008, 2007, 93.00, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3020, 1008, 2008, 89.75, '2021-1');

-- Estudiante 1009 – Valentina Morales (Civil)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3021, 1009, 2007, 90.00, '2019-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3022, 1009, 2008, 85.75, '2020-1');

-- Estudiante 1010 – Diego Sánchez (Mecánica)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3023, 1010, 2009, 72.00, '2020-1');
```

```
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3024, 1010, 2010, 78.50, '2020-2');

-- Estudiante 1011 – Sofía Ramírez (Mecánica – excelente)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3025, 1011, 2009, 97.00, '2020-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3026, 1011, 2010, 94.25, '2021-1');

-- Estudiante 1012 – Daniela Castro (Mecánica)
INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3027, 1012, 2009, 70.00, '2021-2');

INSERT INTO MATRICULAS (mat_id, est_id, curso_id, mat_nota, mat_periodo)
VALUES (3028, 1012, 2011, 76.50, '2021-2');

COMMIT;
```

CONSULTAS DE VERIFICACIÓN

Scripts para verificar los datos insertados:

```
-- Contar registros por 34erio
SELECT 'ESTUDIANTES' AS 34erio, COUNT(*) AS total FROM ESTUDIANTES
UNION ALL
SELECT 'CURSOS', COUNT(*) FROM CURSOS
UNION ALL
SELECT 'MATRICULAS', COUNT(*) FROM MATRICULAS;

-- Estudiantes por 34eriod34
SELECT est_carrera, COUNT(*) AS total, ROUND(AVG(est_promedio), 2) AS
promedio
FROM ESTUDIANTES
GROUP BY est_carrera
ORDER BY promedio DESC;

-- Cursos por departamento
SELECT curso_departamento, COUNT(*) AS total_cursos,
       SUM(curso_creditos) AS total_creditos
FROM CURSOS
GROUP BY curso_departamento
ORDER BY total_cursos DESC;

-- Ver matrículas con detalles
SELECT e.est_nombre || ' ' || e.est_apellido AS estudiante,
       c.curso_nombre,
       m.mat_periodo,
       m.mat_nota
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
ORDER BY e.est_apellido, m.mat_periodo;

-- Estadísticas de notas
SELECT COUNT(*) AS total_matriculas,
       COUNT(mat_nota) AS con_nota,
       COUNT(*) - COUNT(mat_nota) AS sin_nota,
       ROUND(AVG(mat_nota), 2) AS promedio_general,
       COUNT(CASE WHEN mat_nota >= 70 THEN 1 END) AS aprobados,
       COUNT(CASE WHEN mat_nota < 70 THEN 1 END) AS reprobados
FROM MATRICULAS;
```

RESUMEN DE DATOS

ESTUDIANTES: 12 registros

• Sistemas: 3 estudiantes (1001, 1002, 1003)
• Electrónica: 3 estudiantes (1004, 1005, 1006)
• Civil: 3 estudiantes (1007, 1008, 1009)
• Mecánica: 3 estudiantes (1010, 1011, 1012)
• Promedios: desde 58.20 hasta 95.60
• Créditos: desde 80 hasta 155

CURSOS: 12 registros

• Sistemas: 4 cursos
• Electrónica: 2 cursos
• Civil: 2 cursos
• Mecánica: 2 cursos
• Matemáticas: 2 cursos (compartidos)
• Créditos: 4-5 por curso

MATRICULAS: 28 registros

• Periodos: 2019-2 hasta 2021-2
• Notas aprobadas (≥70): 23 registros
• Notas reprobadas (<70): 4 registros
• Sin nota (en curso): 1 registro
• Promedio general de notas: ~81.5

ANEXO2.

MODIFICACIÓN DE TABLA ESTUDIANTES

Añadir columna est_fecha_nacimiento

Guía paso a paso para modificar la tabla ESTUDIANTES y actualizar los datos

PASO 1: VERIFICAR ESTRUCTURA ACTUAL

Antes de modificar, verificar la estructura actual de la tabla:

```
DESC ESTUDIANTES;
```

Resultado esperado:

```
Name           Null?    Type
-------------- -------- ------------
EST_ID         NOT NULL NUMBER(6)
EST_NOMBRE              VARCHAR2(50)
EST_APELLIDO            VARCHAR2(50)
EST_CARRERA             VARCHAR2(30)
EST_CREDITOS            NUMBER(3)
EST_PROMEDIO            NUMBER(4,2)
```

PASO 2: AÑADIR LA NUEVA COLUMNA

Comando ALTER TABLE para añadir la columna:

```
ALTER TABLE ESTUDIANTES
ADD est_fecha_nacimiento DATE;
```

Explicación:

• ALTER TABLE: Modifica la estructura de una tabla existente
• ADD: Añade una nueva columna
• est_fecha_nacimiento: Nombre de la nueva columna
• DATE: Tipo de dato para almacenar fechas

NOTA: La columna se crea con NULL en todos los registros existentes

Verificar que se añadió correctamente:

```
DESC ESTUDIANTES;
```

Ahora debe aparecer:

```
EST_FECHA_NACIMIENTO    DATE
```

PASO 3: ACTUALIZAR LOS DATOS

Actualizar la fecha de nacimiento de cada estudiante:

```
-- ========================================
-- ACTUALIZAR FECHAS DE NACIMIENTO
-- Estudiantes universitarios: 18-24 años
-- Fechas entre 2000-2006
-- ========================================

-- Estudiante 1001 - Juan Pérez (24 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2000-05-15'
WHERE est_id = 1001;

-- Estudiante 1002 - María González (25 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '1999-08-22'
WHERE est_id = 1002;

-- Estudiante 1003 - Miguel Torres (22 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2002-09-20'
WHERE est_id = 1003;

-- Estudiante 1004 - Carlos Rodríguez (23 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2001-11-10'
WHERE est_id = 1004;

-- Estudiante 1005 - Ana Martínez (25 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '1999-03-18'
WHERE est_id = 1005;

-- Estudiante 1006 - Roberto Vargas (23 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2001-02-28'
WHERE est_id = 1006;

-- Estudiante 1007 - Luis Fernández (21 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2003-01-25'
WHERE est_id = 1007;
```

```sql
-- Estudiante 1008 - Andrea López (24 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2000-07-30'
WHERE est_id = 1008;

-- Estudiante 1009 - Valentina Morales (25 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '1999-06-08'
WHERE est_id = 1009;

-- Estudiante 1010 - Diego Sánchez (24 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2000-12-05'
WHERE est_id = 1010;

-- Estudiante 1011 - Sofía Ramírez (23 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2001-04-14'
WHERE est_id = 1011;

-- Estudiante 1012 - Daniela Castro (21 años)
UPDATE ESTUDIANTES
SET est_fecha_nacimiento = DATE '2003-10-12'
WHERE est_id = 1012;

-- Confirmar cambios
COMMIT;
```

PASO 4: VERIFICAR LAS ACTUALIZACIONES

Consulta para verificar que se actualizaron todos los registros:

```
SELECT est_id,
       est_nombre || ' ' || est_apellido AS estudiante,
       est_fecha_nacimiento,
       -- Calcular edad manualmente para verificar
       TRUNC(MONTHS_BETWEEN(SYSDATE, est_fecha_nacimiento) / 12) AS edad
FROM ESTUDIANTES
ORDER BY est_id;
```

Verificar que NO haya NULLs:

```
SELECT COUNT(*) AS total,
       COUNT(est_fecha_nacimiento) AS con_fecha,
       COUNT(*) - COUNT(est_fecha_nacimiento) AS sin_fecha
FROM ESTUDIANTES;
```

Resultado esperado:

```
TOTAL    CON_FECHA    SIN_FECHA
-----    ---------    ---------
  12         12            0
```

PASO 5: CREAR LA FUNCIÓN FN_CALCULAR_EDAD

Ahora podemos crear la función para calcular edad:

```
CREATE OR REPLACE FUNCTION FN_CALCULAR_EDAD(
    p_fecha_nacimiento DATE    -- Parámetro de entrada: fecha de nacimiento
) RETURN NUMBER                 -- Retorna un número entero (edad en años)
IS
    v_edad NUMBER;              -- Variable local para almacenar la edad
BEGIN
    -- Validación: Si la fecha es NULL, retornar NULL
    IF p_fecha_nacimiento IS NULL THEN
        RETURN NULL;
    END IF;

    -- MONTHS_BETWEEN calcula meses entre dos fechas
    -- Dividimos entre 12 para convertir meses a años
    -- TRUNC elimina decimales para obtener años completos
    v_edad := TRUNC(MONTHS_BETWEEN(SYSDATE, p_fecha_nacimiento) / 12);

    -- Retornar el valor calculado
    RETURN v_edad;
END FN_CALCULAR_EDAD;
/
```

IMPORTANTE: No olvides el / al final en SQL*Plus

PASO 6: PROBAR LA FUNCIÓN

Prueba 1: Calcular edad de una fecha específica

```
SELECT FN_CALCULAR_EDAD(DATE '2000-05-15') AS edad FROM DUAL;
```

Resultado esperado (en 2024):

```
EDAD
----
  24
```

Prueba 2: Calcular edad de todos los estudiantes

```
SELECT est_id,
       est_nombre || ' ' || est_apellido AS estudiante,
       est_fecha_nacimiento,
       FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       est_carrera
FROM ESTUDIANTES
ORDER BY edad DESC;
```

Prueba 3: Estadísticas por edad

```
SELECT FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY FN_CALCULAR_EDAD(est_fecha_nacimiento)
ORDER BY edad;
```

Prueba 4: Estudiantes mayores de edad específica

```
SELECT est_nombre || ' ' || est_apellido AS estudiante,
       FN_CALCULAR_EDAD(est_fecha_nacimiento) AS edad,
       est_promedio
FROM ESTUDIANTES
WHERE FN_CALCULAR_EDAD(est_fecha_nacimiento) >= 24
ORDER BY edad DESC;
```