**Guía Paso a Paso: Creación de una Función PL/SQL**

Vamos a crear una función que calcule un bono basado en el salario de un empleado (usando la tabla empleados que creamos antes).

**Paso 1: Definición y Creación**

Usamos la sentencia CREATE OR REPLACE FUNCTION. Debes definir el tipo de datos de los parámetros de entrada y el tipo de datos que devuelve la función.

```
CREATE OR REPLACE FUNCTION calcular_bono (
  p_salario IN empleados.salario%TYPE -- Parámetro de entrada
)
RETURN NUMBER -- Tipo de dato que devuelve la función
IS
  v_bono NUMBER; -- Variable local para el cálculo
BEGIN
  -- Lógica de negocio:
  IF p_salario >= 8000 THEN
    v_bono := p_salario * 0.15; -- 15% de bono
  ELSIF p_salario >= 4000 THEN
    v_bono := p_salario * 0.10; -- 10% de bono
  ELSE
    v_bono := p_salario * 0.05; -- 5% de bono
  END IF;

  RETURN v_bono;

EXCEPTION
  -- Manejo básico de errores (devuelve 0 si hay un problema)
  WHEN OTHERS THEN
    RETURN 0;
END;
/
```

**Nota:** El CREATE OR REPLACE permite modificar la función sin tener que borrarla primero.

**Paso 2: Uso en una Sentencia SQL (SELECT)**

Una vez creada, puedes usar la función como si fuera una función nativa de Oracle.

```
SELECT
  nombre,
  salario,
  calcular_bono(salario) AS bono_calculado, -- Aquí se invoca
  salario + calcular_bono(salario) AS salario_total
FROM
  empleados
ORDER BY
  salario DESC;
```

**Ventaja Clave:** La lógica de negocio (IF/THEN/ELSE) está encapsulada en la base de datos. Si la regla del bono cambia (ej. de 15% a 20%), solo modificas la función, y todos los reportes que la usan se actualizan automáticamente.

**Paso 3: Uso en Cláusulas (WHERE)**

También puedes usar funciones para filtrar datos.

**Escenario:** Mostrar solo a los empleados cuyo bono calculado es mayor a 1000.

```
SELECT
  nombre,
  salario,
  calcular_bono(salario) AS bono
FROM
  empleados
WHERE
  calcular_bono(salario) > 1000; -- Invocada en el filtro
```

**Rendimiento: El "Context Switching"**

Las funciones son poderosas, pero tienen una desventaja de rendimiento que debes conocer: el **Cambio de Contexto**.

1. Cuando ejecutas una consulta puramente SQL, Oracle utiliza su **Motor SQL** altamente optimizado.
2. Cuando el Motor SQL encuentra una llamada a una función PL/SQL (como calcular_bono(salario)), tiene que pausar el procesamiento SQL y cambiar el control al **Motor PL/SQL** para ejecutar la lógica IF/THEN/ELSE.
3. Una vez que la función devuelve el resultado (RETURN v_bono), el control vuelve al Motor SQL.

Si llamas a la función para 10,000 filas, habrá 10,000 costosos cambios de contexto.

**La Regla del Rendimiento:**

- **Evitar:** Usar funciones PL/SQL en el WHERE o SELECT de consultas que leen millones de filas.
- **Preferir:** Si la lógica puede resolverse con **Funciones de Ventana** (CASE WHEN + OVER()) o **CTEs** (WITH), hazlo ahí, ya que todo queda dentro del eficiente Motor SQL.
- **Aceptar:** Usarlas cuando la lógica es demasiado compleja para SQL (ej. cálculos iterativos, manejo de excepciones) y el impacto del rendimiento es aceptable (ej. en reportes que leen pocas filas).

**Funciones que Devuelven Tablas (Table Functions)**

Como complemento, existe un tipo avanzado de función llamada **Table Function**. Esta no devuelve un solo valor, sino una **colección de filas** (una tabla virtual) que puede ser tratada como una tabla normal en la cláusula FROM usando el operador **TABLE()**.

- **Uso:** Procesamiento de datos complejo donde la salida son múltiples filas.

**Ejemplo de Uso:**

```
SELECT t.columna_1, t.columna_2
FROM TABLE(mi_funcion_table(parametro_entrada)) t;
```

**Creación de un Procedimiento Almacenado**

Un procedimiento es un bloque de código PL/SQL que se nombra, se almacena en la base de datos y se ejecuta bajo demanda. No están obligados a devolver un valor, pero pueden devolver múltiples datos mediante parámetros de salida.

**Paso 1: Definición y Parámetros**

Vamos a crear un procedimiento para **dar de alta un nuevo empleado** y **actualizar el número de empleados** de su jefe, todo en una sola transacción.

```
CREATE OR REPLACE PROCEDURE crear_empleado (
  p_nombre IN empleados.nombre%TYPE,   -- Parámetro de entrada (IN)
```

```
    p_salario IN empleados.salario%TYPE,
    p_jefe_id IN empleados.jefe_id%TYPE,
    p_nuevo_id OUT empleados.emp_id%TYPE -- Parámetro de salida (OUT)
)
IS
    -- Declaramos una variable para el ID, usando una secuencia
    v_next_id NUMBER;
BEGIN
    -- 1. Generar el nuevo ID (asumiendo que tienes una secuencia llamada EMP_SEQ)
    -- SELECT EMP_SEQ.NEXTVAL INTO v_next_id FROM dual;
    -- Para Live SQL, usaremos MAX+1 si no hay secuencia:
    SELECT NVL(MAX(emp_id), 0) + 1 INTO v_next_id FROM empleados;

    -- 2. Insertar el nuevo empleado
    INSERT INTO empleados (emp_id, nombre, salario, jefe_id)
    VALUES (v_next_id, p_nombre, p_salario, p_jefe_id);

    -- 3. (Opcional) Lógica de negocio adicional: Actualizar otra tabla (no se muestra aquí)

    -- 4. Devolver el nuevo ID al exterior
    p_nuevo_id := v_next_id;

    -- 5. Confirmar los cambios
    COMMIT;

EXCEPTION
    WHEN OTHERS THEN
        -- Si hay un error, deshacer la operación
        ROLLBACK;
        RAISE_APPLICATION_ERROR(-20001, 'Error al crear empleado: ' || SQLERRM);
END;
```

/

**Nota:** En un entorno real, usarías una secuencia (EMP_SEQ.NEXTVAL) en lugar de MAX+1.

**Ejecución de un Procedimiento**

Los procedimientos **no se pueden llamar directamente desde un SELECT** como las funciones. Debes ejecutarlos usando la sentencia EXECUTE (o EXEC en SQL*Plus/Live SQL) o un bloque BEGIN...END en PL/SQL.

**Paso 2: Uso del Procedimiento**

Para capturar el parámetro de salida (p_nuevo_id), necesitamos un bloque DECLARE/BEGIN/END.

S

```
DECLARE
  v_id_generado NUMBER; -- Variable para capturar el valor de salida
BEGIN
  -- Llamada al procedimiento:
  crear_empleado(
    p_nombre => 'Roberto Nuevo', -- Nombre
    p_salario => 3500,         -- Salario
    p_jefe_id => 3,            -- Jefe (Luis Gerente)
    p_nuevo_id => v_id_generado  -- Variable de salida
  );

  DBMS_OUTPUT.PUT_LINE('✔ Empleado creado exitosamente con ID: ' ||
v_id_generado);

  -- Verificar el nuevo empleado
  FOR r IN (SELECT nombre, jefe_id FROM empleados WHERE emp_id = v_id_generado)
LOOP
    DBMS_OUTPUT.PUT_LINE('   - Nombre: ' || r.nombre || ', Reporta a: ' || r.jefe_id);
  END LOOP;
END;
/
```

**Funciones vs. Procedimientos**

La elección entre una función y un procedimiento es una decisión de diseño fundamental, dictada por su propósito.

| Característica | Procedimiento Almacenado (Procedure) | Función Definida por el Usuario (Function) |
|---|---|---|
| **Propósito** | Realizar una **Acción** o Tarea (DML, I/O, Transacciones). | Realizar un **Cálculo** y devolver un valor. |
| **Valor Devuelto** | Cero o múltiples valores (vía parámetros OUT). | **Siempre un único valor** (RETURN). |
| **Manipulación de Datos (DML)** | **Permitida** (INSERT, UPDATE, DELETE). | **No permitida** si se usa en SELECT (debe ser pura). |
| **Ejecución** | Vía EXECUTE o bloque BEGIN…END. | Directamente en sentencias SQL (SELECT, WHERE, etc.). |
| **Commit/Rollback** | Puede contener explícitamente COMMIT y ROLLBACK. | No debe contener COMMIT o ROLLBACK. |

**La Regla Simple:**

- Si quieres **hacer algo** (ej. actualizar, guardar en un log, mandar un email) Usa un **Procedimiento**.
- Si quieres **obtener un valor** (ej. calcular un bono, transformar un dato) Usa una **Función**.