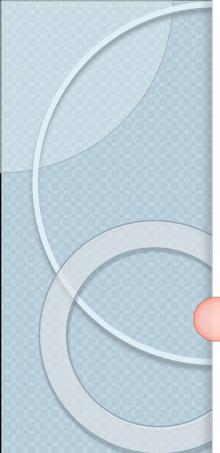


# **Subprogramas: PROCEDIMIENTOS Y FUNCIONES**



# Subprogramas

Son bloques PL/SQL identificados por un nombre. Eventualmente, estos bloques pueden ser invocados con parámetros.

# **Los Subprogramas también cuentan con:**

- **Una sección Declarativa**, con la declaración de tipos, cursores, variables, etc. Estos ítems son locales y visibles solo dentro del subprograma
- **Una sección Ejecutable**, con sentencias que asignan valores, control de ejecución y manipulación de datos
- **Una sección de Excepciones**, que es opcional, para el manejo de errores en tiempo de ejecución

# ***Componentes del Subprograma***

## **CABECERA – obligatorio**

- Nombre del subprograma, tipo y argumentos

## **AREA DECLARATIVA – opcional**

- Identificadores locales

## **AREA EJECUTABLE – obligatorio**

- Sentencias SQL
- Sentencias de control PL/SQL

## **MANEJO DE EXCEPCIONES – opcional**

- Acciones que se ejecutan cuando hay error

## **END; – obligatorio**

# PROCEDIMIENTOS

- Un procedimiento es un subprograma que realiza una acción específica. Esta unidad de programa es compilada en el esquema de la base de datos.
- Pueden aceptar argumentos
- Una vez compilado, el identificador del procedimiento (creado con CREATE PROCEDURE) se convierte en el nombre de un objeto de la base de datos que puede ser ejecutado desde cualquier interfaz que pueda ejecutar sentencias SQL

# Declaración de Procedimientos: Sintaxis

```
PROCEDURE nombre  
  [ (parámetro, . . .) ]  
IS  
Bloque pl/sql;
```

Donde la sintaxis del parámetro es :

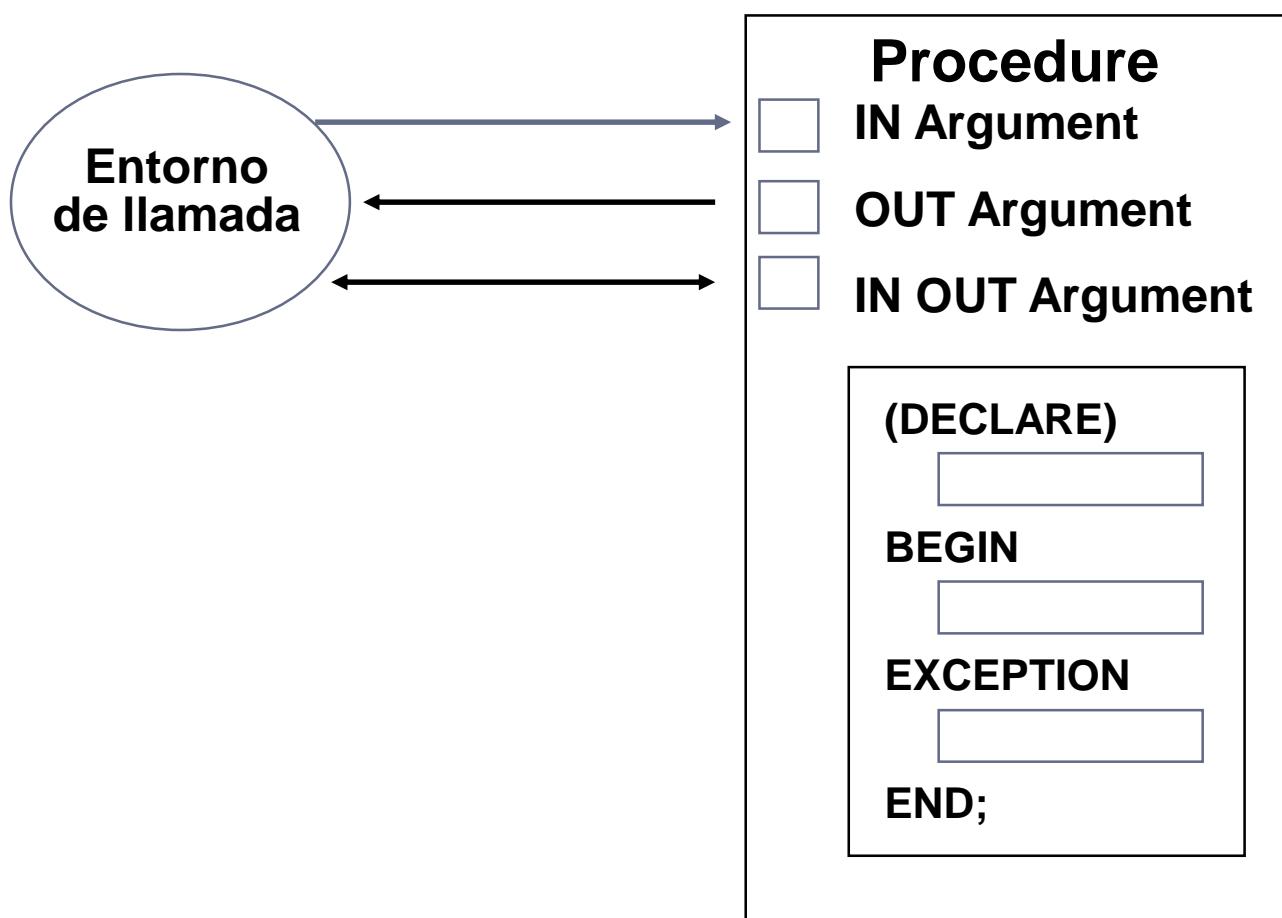
```
Nombre_parámetro [IN | OUT | IN OUT] tipo_dato  
  [{ := | DEFAULT} expresión]
```

No debe especificar la longitud del tipo de dato, y  
tampoco puede incluir constraints.

# Creando un Procedimiento: Pautas

- Use la cláusula CREATE O REPLACE al construir su procedimiento en SQL\*Plus.
- Indique los parámetros y su tipo
- Empiece los bloques PL/SQL con IS.
- Ingrese declaración de variables o la sentencia BEGIN (no se usa DECLARE)

# Modos de utilización de parámetros



# Modos para los Parámetros Formales

IN (Por default)	OUT (debe especificarse)	IN OUT (debe especificarse)
El valor es pasado al subprograma	El valor es retorna al ambiente que llama	Se pasa un valor al subprograma y éste retorna a su vez un valor al ambiente que llama
El parámetro puede ser un literal, una expresión, <b>una constante</b> o variable inicializada	Debe definirse una variable en el programa 'llamador', pero dicha variable no requiere ser inicializada	Debe definirse una variable en el programa 'llamador' y dicha variable debe estar inicializada

# ***Creación de Procedimientos: Ejemplo***

```
CREATE OR REPLACE PROCEDURE p_actualizar_stock
    (p_id_art    IN NUMBER,
     p_cantidad  IN NUMBER)
IS
BEGIN
    UPDATE b_articulos
    SET stock_actual = stock_actual + p_cantidad
    WHERE id = p_id_art;
    COMMIT;
END p_actualizar_stock;
```

# Declarando Variables y Constantes: Sintaxis

```
identificador [CONSTANT] tipodato [NOT NULL]  
[ := | DEFAULT expr] ;
```

## Pautas

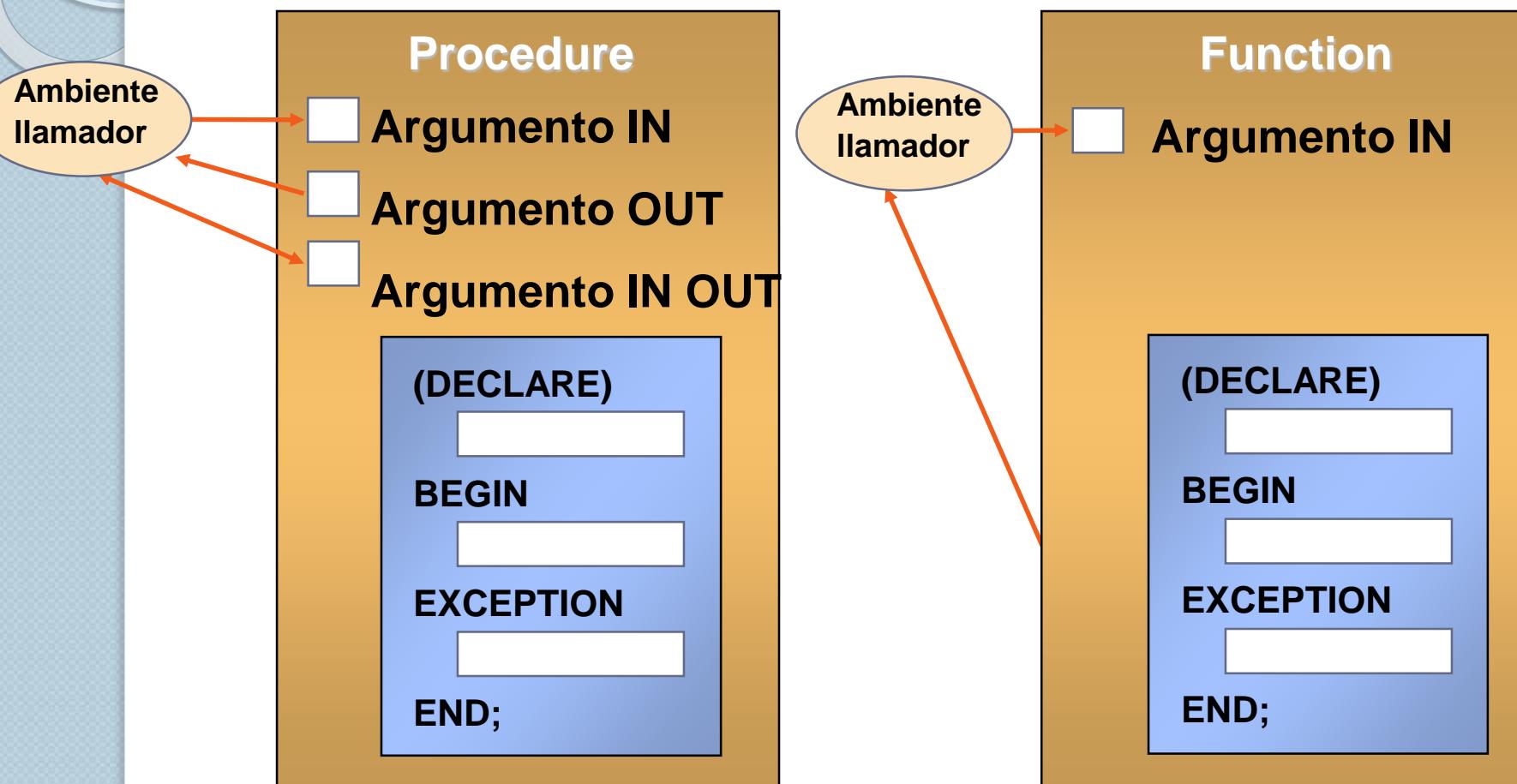
- Nombre de acuerdo a convenciones.
- Inicialice las constantes y variables no designadas como NOT NULL.
- Inicialice identificadores usando al operador de la asignación (:=) o por la palabra reservada DEFAULT.
- Declare un identificador a lo sumo por línea.

# FUNCIÓN

Las funciones son bloques de PL/SQL construidos al igual que los procedimientos, con las siguientes diferencias:

- Se invoca a la función como parte de una expresión
- Una función **DEBE** retornar un valor

# PROCEDIMIENTO O FUNCIÓN?



# COMPARACIÓN DE FUNCIONES Y PROCEDIMIENTOS

<b>PROCEDURE</b>	<b>FUNCION</b>
Debe ejecutarse como una sentencia SQL	Se puede llamar como parte de una expresión
No contiene RETURN	Debe contener un RETURN
Puede retornar un valor (parámetros OUT)	Necesariamente DEBE retornar un valor

# SINTAXIS DE LA FUNCIÓN

```
FUNCTION nombre
    [ (parámetro, ... ) ]
    RETURN tipo_de_dato
IS | AS
Cuerpo del PL/SQL;
```

**DEBE** haber al menos un RETURN en una función. Puede haber más de uno.

# INVOCACIÓN DE FUNCIONES

- **PROPÓSITO**

- Permitir la ejecución de cálculos no predefinidos en SQL
- Incrementar la eficiencia de las consultas.

- **RECOMENDACIONES**

- Los parámetros deben ser IN
- Si se usarán en SQL, no utilizar sentencias DML dentro de la función
- Utilizar tipos de datos de ORACLE y no tipos de datos de PL/SQL
- Se requiere privilegios de EXECUTE

# **Ejemplos de llamado a funciones**

- Como parte de una sentencia SELECT
- Como condición de una cláusula WHERE o HAVING
- Como parte de cláusulas ORDER BY y GROUP BY
- Como parte de los valores incluidos en la cláusula VALUES de la sentencia INSERT
- Como parte de la cláusula SET de la sentencia UPDATE

# LLAMADA A FUNCIÓN: EJEMPLO

```
SELECT V.ID, V.NUMERO_FACTURA, V.MONTO_TOTAL,  
       DEDUCCION(V.ID) DEDUCCION  
FROM   B_VENTAS V  
WHERE  ID = 1;
```

“deducion” es una función definida así:

```
CREATE OR REPLACE FUNCTION DEDUCCION(PID_VENTA NUMBER) RETURN  
NUMBER  
IS  
    V_DEDUCCION NUMBER;  
BEGIN  
    SELECT SUM(ROUND(V.PRECIO * A.PORC_IVA))  
      INTO V_DEDUCCION  
     FROM B_DETALLE_VENTAS V JOIN B_ARTICULOS A  
       ON A.ID = V.ID_ARTICULO  
      WHERE V.ID_VENTA = PID_VENTA;  
    RETURN V_DEDUCCION;  
END;
```

## ***La instrucción RETURN***

- La instrucción RETURN devuelve el valor que genera, así como el control al ambiente llamador.
- Puede haber más de un RETURN en una función, pero sólo se ejecuta uno de ellos. Tome en cuenta que todas las instrucciones que siguen al RETURN ya no se ejecutan

# Procedimientos y Funciones locales

- Los procedimientos y funciones también pueden declararse localmente.
- En ese caso, no se usa la palabra clave CREATE, y toda la unidad PL/SQL va en la parte declarativa
- Las funciones y procedimientos locales sólo pueden invocarse dentro de la unidad de programa que les contiene.

# Ejemplo: Unidad de Programa local

```
CREATE OR REPLACE PROCEDURE P_PROCESAR_VENTAS
IS
FUNCTION DEDUCCION(PID_VENTA NUMBER) RETURN NUMBER
IS
    V_DEDUCCION NUMBER;
BEGIN
    SELECT SUM(ROUND(V.PRECIO * A.PORC_IVA))
        INTO V_DEDUCCION
    FROM B_DETALLE_VENTAS V JOIN B_ARTICULOS A
    ON A.ID = V.ID_ARTICULO
    WHERE V.ID_VENTA = PID_VENTA;
    RETURN NVL(V_DEDUCCION,0);
END;
BEGIN
    FOR REG IN (SELECT * FROM B_VENTAS) LOOP
        DBMS_OUTPUT.PUT_LINE('Deducción de factura
'||TO_CHAR(REG.numero_factura)|| ' es: '
|| TO_CHAR(DEDUCCION(REG.id)));
    END LOOP;
END;
/
```

# **PROCEDIMIENTOS Y FUNCIONES**

## **Más sobre los parámetros:**

- De manera predeterminada, PL/SQL pasa los parámetros IN por referencia y los parámetros IN OUT y OUT por valor.
- Se puede agregar la indicación NOCOPY para indicar explícitamente que el parámetro será por referencia. Ej:

```
CREATE OR REPLACE PROCEDURE
Ver_uso_nocopy(p_parametro IN OUT nocopy
NUMBER)
IS . . . . .
```

# PROCEDIMIENTOS Y FUNCIONES

## Notación Posicional y notación nominal

- Los argumentos reales en general se encuentran asociados a los argumentos formales por su posición.
- Alternativamente se puede usar una notación nominal:

Si la declaración formal del procedimiento P1 fuera:

```
PROCEDURE P1 (p_par1 number, p_par2 number);
```

Se puede invocar al procedimiento así:

```
BEGIN
```

```
P_PROCEDIMIENTO(p_par1 => 10, p_par2 =>3);
```

```
END;
```

# **PROCEDIMIENTOS Y FUNCIONES**

## **Valores predeterminados (DEFAULT)**

- Los parámetros formales pueden tener valores predeterminados. Para ello se usa la expresión DEFAULT:

```
PROCEDURE P1 (p_par1 number DEFAULT 1);
```

```
PROCEDURE P1 (p_par1 number := 1);
```

- Puede de esta manera, invocarse al procedimiento sin el parámetro, el cual asume el valor predeterminado.
- Si existen varios parámetros, y se usa notación posicional, solo puede dejarse de utilizar los últimos en la lista.

# **PRIVILEGIOS REQUERIDOS**

- Los subprogramas requieren el privilegio EXECUTE .
- Aun cuando el usuario no tenga acceso a los objetos referenciados por el subprograma, indirectamente, los accede a través de dicho subprograma

# **SINTAXIS PARA ELIMINACIÓN DE FUNCIONES Y PROCEDIMIENTOS**

- **DROP PROCEDURE**  
nombre\_procedimiento;
  
- **DROP FUNCTION** nombre\_funcion;



# **DEPENDENCIA DE LOS SUBPROGRAMAS**

- Cuando se compila una función o un procedimiento almacenado, todos los objetos de Oracle a los que hace referencia se registran en el diccionario de datos.
- Si se alteran algunos de estos objetos (ej. Una tabla) a los que hace referencia un subprograma, éste puede quedar inválido.

# DEPENDENCIA DE LOS SUBPROGRAMAS

Ej:

- Se tiene un procedimiento sobre la tabla `b_detalle_ventas`
- Se altera la tabla `b_detalle_ventas` para agregar o modificar una columna
- La función que hace referencia al objeto queda inválido

# **PARA PODER VER LAS DEPENDENCIAS**

- **USER\_DEPENDENCIES**
- **ALL\_DEPENDENCIES**
- **DBA\_DEPENDENCIES**

## **VER LAS DEPENDENCIAS**

```
SELECT NAME, TYPE, REFERENCED_NAME,  
REFERENCED_TYPE FROM USER_DEPENDENCIES;
```

## **VER TODOS LOS OBJETOS QUE DEPENDEN DE LA TABLA 'B\_DETALLE\_VENTAS'**

```
SELECT NAME, TYPE FROM USER_DEPENDENCIES  
WHERE REFERENCED_NAME = 'B_DETALLE_VENTAS';
```