**1. Declaración de una variable**

**variable\_name data\_type [NOT NULL] [DEFAULT valor\_inicial];**

* **variable\_name**: Nombre de la variable (debe cumplir las reglas de nombres en Oracle, como empezar con una letra).
* **data\_type**: Tipo de dato de la variable (puede ser un tipo básico como NUMBER, VARCHAR2, o derivado de una tabla usando %TYPE).
* **NOT NULL** *(opcional)*: Indica que la variable no puede contener valores nulos.
* **DEFAULT** *(opcional)*: Permite asignar un valor inicial a la variable.
* **:=**: Alternativa a DEFAULT para asignar valores iniciales.

**Ejemplos:**

v\_salario NUMBER; -- Variable de tipo número sin valor inicial

v\_nombre VARCHAR2(50) := 'Juan'; -- Variable de tipo texto con valor inicial

v\_fecha DATE DEFAULT SYSDATE; -- Variable de tipo fecha con valor inicial usando DEFAULT

**2. Declaración de una variable basada en una columna de una tabla**

variable\_name table\_name.column\_name%TYPE;

* **variable\_name**: Nombre de la variable.
* **table\_name**: Nombre de la tabla de la base de datos.
* **column\_name**: Nombre de la columna en la tabla.

**Ejemplo:**

sql

CopiarEditar

v\_salario EMPLEADOS.SALARIO%TYPE; -- Variable del mismo tipo que la columna "SALARIO" de la tabla "EMPLEADOS"

Esto asegura que si el tipo de la columna cambia en la base de datos, la variable sigue siendo compatible automáticamente.

**3. Asignación de valores a variables**

En PL/SQL puedes asignar valores de dos formas principales:

**a) Asignación directa**

variable\_name := valor;

v\_nombre := 'Carlos';

v\_salario := 5000;

v\_fecha := SYSDATE;

**b) Asignación mediante una consulta SELECT ... INTO**

SELECT columna

INTO variable\_name

FROM tabla

WHERE condición;

**Ejemplo:**

DECLARE

v\_salario EMPLEADOS.SALARIO%TYPE;

BEGIN

SELECT SALARIO

INTO v\_salario

FROM EMPLEADOS

WHERE NOMBRE = 'KING';

END;

* **columna**: La columna de la que deseas obtener el valor.
* **variable\_name**: La variable que recibirá el valor.
* **tabla**: La tabla de la base de datos.
* **condición**: La condición para filtrar los datos.

**4. Sintaxis de un bloque PL/SQL completo**

DECLARE

variable\_name data\_type [:= valor\_inicial];

BEGIN

-- Asignación de valores

variable\_name := nuevo\_valor;

-- Operaciones

SELECT columna

INTO variable\_name

FROM tabla

WHERE condición;

-- Mostrar resultados

DBMS\_OUTPUT.PUT\_LINE('Mensaje: ' || variable\_name);

END;

/

**Ejemplo completo:**

DECLARE

v\_nombre EMPLEADOS.NOMBRE%TYPE := 'KING'; -- Variable con valor inicial

v\_salario EMPLEADOS.SALARIO%TYPE; -- Variable basada en la columna "SALARIO"

BEGIN

-- Consultar el salario del empleado "KING"

SELECT SALARIO

INTO v\_salario

FROM EMPLEADOS

WHERE NOMBRE = v\_nombre;

-- Mostrar el salario

DBMS\_OUTPUT.PUT\_LINE('El salario de ' || v\_nombre || ' es: ' || v\_salario);

END;

/

**Elementos básicos**

| **Elemento** | **Significado** | **Ejemplo** |
| --- | --- | --- |
| variable\_name | Nombre de la variable | v\_nombre, v\_salario |
| data\_type | Tipo de dato de la variable | NUMBER, VARCHAR2, DATE |
| tabla | Nombre de la tabla de la base de datos | EMPLEADOS |
| columna | Nombre de la columna dentro de la tabla | SALARIO, NOMBRE |

A continuación te presento una tabla con la sintaxis, el uso y las reglas de los procedimientos y funciones en PL/SQL que mencionaste:

| **Concepto** | **Sintaxis** | **Uso** | **Reglas** |
| --- | --- | --- | --- |
| **Procedimientos y Funciones** | CREATE PROCEDURE <nombre\_proc> [parametros] IS ... BEGIN ... END; | Definición de un procedimiento almacenado. | Se pueden declarar con parámetros IN, OUT, o IN OUT. No devuelven valores en los procedimientos. |
| **Procedimiento Almacenado** | CREATE OR REPLACE PROCEDURE <nombre\_proc> IS ... BEGIN ... END; | Crear un procedimiento que se almacena en la base de datos. | Puede ser reemplazado si ya existe. |
| **Función Almacenada** | CREATE OR REPLACE FUNCTION <nombre\_funcion>(parametros) RETURN <tipo> IS ... BEGIN ... RETURN <expresión>; END; | Definición de una función almacenada. Devuelve un valor. | Debe incluir RETURN en su declaración para indicar el tipo de valor devuelto. |
| **Llamada a Procedimiento** | EXECUTE <nombre\_proc>(param1, param2, ...); | Invocar procedimientos almacenados desde SQL\*PLUS o PL/SQL. | El número y tipo de parámetros debe coincidir con los definidos en la declaración. |
| **Llamada a Función** | <variable> := <nombre\_funcion>(param1, param2, ...); | Invocar funciones almacenadas desde un bloque de código o SQL\*PLUS. | El valor devuelto se asigna a una variable. |
| **Parámetros de Procedimiento** | `param [IN | OUT | IN OUT] tipo\_dato [:= |
| **Notación de Parámetros** | **Posicional:** param1, param2**Nominal:** param1 => valor1, param2 => valor2**Mixta:** Combinación de las dos anteriores | Métodos para pasar parámetros a un subprograma. | La notación posicional sigue el orden definido. La nominal permite especificar el valor por nombre. La mixta combina ambas. |
| **Funciones Predefinidas - Caracteres** | LENGTH(<cadena>), INSTR(<cadena>, <subcadena>, <inicio>, <ocurrencia>), REPLACE(<expresión>, <busqueda>, <reemplazo>), SUBSTR(<expresión>, <inicio>, <longitud>) | Funciones para manipular cadenas de texto. | Algunas funciones como REPLACE modifican el texto, otras como LENGTH devuelven un valor. |
| **Funciones Predefinidas - Numéricas** | MOD(<dividendo>, <divisor>), TRUNC(<número>), ROUND(<número>) | Funciones para realizar operaciones matemáticas. | Devuelven el resto de la división, truncan decimales o redondean números, respectivamente. |
| **Funciones Predefinidas - Fechas** | SYSDATE, TRUNC(<fecha>) | Funciones para obtener y manipular fechas. | SYSDATE devuelve la fecha actual, mientras que TRUNC elimina horas, minutos y segundos de una fecha. |
| **Funciones Predefinidas - Conversión** | TO\_DATE(<expresión>, <formato>), TO\_CHAR(<expresión>, <formato>), TO\_NUMBER(<expresión>, <formato>) | Funciones para convertir entre tipos de datos. | TO\_DATE convierte texto a fecha, TO\_CHAR convierte cualquier tipo a cadena, y TO\_NUMBER convierte a número. |
| **Funciones Predefinidas - Nulos** | NVL(<expresión>, <valor>) | Manejo de valores NULL. Devuelve un valor si la expresión es NULL. | Útil para reemplazar NULL por un valor predeterminado. |
| **Funciones Predefinidas - Misceláneas** | DECODE(<expresión>, <cond1>, <valor1>, ..., <condN>, <valorN>, <default>) | Implementa una estructura condicional tipo IF para seleccionar valores según condiciones. | El uso de DECODE es similar a una cadena de sentencias IF. |
| **Sobrecarga de Subprogramas** | CREATE OR REPLACE PROCEDURE <nombre\_subprograma>(<parametros>) | Se puede definir más de un procedimiento o función con el mismo nombre, pero con diferentes parámetros. | Se distingue por el número o tipo de parámetros. |
| **Recursividad** | FUNCTION factorial(n INTEGER) RETURN INTEGER IS ... BEGIN ... RETURN n \* factorial(n-1); END; | Funciones o procedimientos que se llaman a sí mismos, como en el cálculo del factorial. | Es necesario tener una condición base para evitar una llamada infinita. |
| **Eliminar Procedimientos o Funciones** | DROP PROCEDURE <nombre\_proc>;DROP FUNCTION <nombre\_funcion>; | Eliminar procedimientos o funciones almacenadas en la base de datos. | Se debe tener permisos adecuados para eliminar los subprogramas. |

Esta tabla resume los puntos clave que has proporcionado sobre los procedimientos y funciones en PL/SQL.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Concepto | Definición | Sintaxis |
|  |  |  |
| Declarar variable | Crear una variable para almacenar datos temporales en un bloque PL/SQL. | variable\_name data\_type [:= valor\_inicial]; |
| Variable con %TYPE | Crear una variable basada en el tipo de una columna de una tabla. | variable\_name table\_name.column\_name%TYPE; |
| Asignar valor | Asignar un valor a una variable en el bloque PL/SQL. | variable\_name := valor; |
| Consulta con INTO | Recuperar datos de una tabla y almacenarlos en una variable. | SELECT columna INTO variable\_name FROM tabla WHERE condición; |
| Operador de asignación | Operador utilizado para asignar valores a variables. | variable\_name := valor; |
| Declarar bloque PL/SQL | Estructura general de un bloque PL/SQL. | DECLARE ... BEGIN ... END; |
| Mostrar resultados | Imprimir mensajes o valores en la consola usando DBMS\_OUTPUT. | `DBMS\_OUTPUT.PUT\_LINE('Texto: ' |
| Inicio del programa | Sección ejecutable de un bloque PL/SQL donde se realizan operaciones. | BEGIN ... END; |
| Ciclo WHILE | Ejecuta un conjunto de instrucciones mientras se cumpla una condición. | WHILE condición LOOP ... END LOOP; |
| Longitud de cadena | Función que obtiene el número de caracteres de una cadena. | LENGTH(cadena); |
| Extraer subcadena | Función que extrae una porción de texto de una cadena. | SUBSTR(cadena, posición\_inicial, longitud); |
| Fecha actual | Obtiene la fecha y hora actuales del sistema. | SYSDATE |
| Formato de fecha | Convierte una fecha en texto o extrae partes específicas de una fecha. | TO\_CHAR(fecha, 'formato'); |
| Conversión a número | Convierte texto en un valor numérico. | TO\_NUMBER(texto); |

Aquí tienes una tabla que resume las definiciones y sintaxis básicas de los elementos en PL/SQL:

¡Aquí tienes la tabla actualizada, incluyendo la estructura de %ROWTYPE! 👇

**Diferencia entre SQL%ROWCOUNT y %ROWTYPE**

| **Concepto** | **SQL%CONTADORDEFILAS** | **%TIPO DE FILA** |
| --- | --- | --- |
| **Tipo** | Atributo implícito en PL/SQL | Tipos de datos en PL/SQL |
| **Uso** | Devuelve el número de filas afectadas por un DML | Almacena una fila completa de una tabla o resultado de una consulta |
| **Alcance** | Operaciones DML ( INSERT, UPDATE, DELETE) | Consultas que devuelven una sola fila completa |
| **Resultado** | Valor numérico | Estructura tipo registro (fila completa, con columnas y valores) |
| **Estructura** | No tiene estructura; solo devuelve un número | Se basa en la estructura de la tabla o consulta que representa. Por ejemplo:  - Si la tabla empleadostiene columnas: id, nombre, sueldo, la variable declarada con %ROWTYPEtendrá los mismos campos: v\_empleado.id, v\_empleado.nombre, v\_empleado.sueldo. |
| **Ejemplo típico** | Saber cuantas filas se actualizaron en una tabla | Guarde los datos de un empleado completo con todas sus columnas |

| **Criterio** | **Un soloBEGIN** | **Dos bloques BEGINanidados** |
| --- | --- | --- |
| **Sencillez del código** | Ideal para código simple y directo. | Útil para código más complejo con múltiples operaciones. |
| **Cantidad de operaciones** | Para 1 o pocas operaciones dependientes entre sí. | Para varias operaciones que pueden fallar independientemente. |
| **Manejo de excepciones** | Las excepciones se manejan de forma general para todo el bloque. | Permite manejar errores específicos de cada operación. |
| **independencia de las operaciones** | Todas las operaciones comparten el mismo manejo de errores. | Cada bloque puede manejar sus errores sin afectar al resto. |
| **Flujo del programa** | Si ocurre una excepción, el bloque completo puede terminar. |  |

**Diferencias Clave**

| **Característica** | **Constante** | **Variable No Constante** |
| --- | --- | --- |
| **Valor inicial** | Se debe asignar un valor inicial al declarar. | Se debe asignar un valor inicial, pero puede cambiar. |
| **Cambio de valor** | No puede cambiar después de ser asignado. | Puede cambiar durante la ejecución. |
| **Declaración** | CONSTANTes la palabra clave. | No se usa CONSTANT, solo se declara el tipo. |
| **Uso típico** | Usada cuando el valor debe ser fijo (ej. tasas, umbrales). | Usada cuando el valor cambia en el tiempo (ej. contadores, acumuladores). |

La diferencia entre usar o no usar las palabras clave CREATE, OR REPLACE y simplemente declarar una función con FUNCTION depende del contexto y del propósito. Aquí te explico en detalle cómo funcionan y las diferencias:

**3. Usar solo FUNCTION dentro de un bloque anónimo**

* Si utilizas FUNCTION sin CREATE o CREATE OR REPLACE, estás declarando una **función local** dentro de un bloque PL/SQL anónimo o en el cuerpo de un procedimiento/función ya existente.
* Las funciones locales solo están disponibles **dentro del bloque** donde se definen y no se almacenan permanentemente en la base de datos.

**Ejemplo de una función local en un bloque anónimo:**

DECLARE

-- Declaración de una función dentro de un bloque anónimo

FUNCTION MiFuncion(fecha IN DATE) RETURN NUMBER IS

anio NUMBER;

BEGIN

anio := TO\_NUMBER(TO\_CHAR(fecha, 'YYYY'));

RETURN anio;

END;

BEGIN

-- Llamar a la función local

DBMS\_OUTPUT.PUT\_LINE('El año es: ' || MiFuncion(SYSDATE));

END;

/

* **Resultado:**
  + La función MiFuncion se define y se usa **solo dentro del bloque**.
  + No está disponible para ser utilizada fuera del bloque.

**Diferencias clave**

| **Palabra clave** | **Propósito** | **Permanencia** | **Disponibilidad** |
| --- | --- | --- | --- |
| CREATE FUNCTION | Crea una función por primera vez. Si la función ya existe, genera un error. | Permanente (en la base de datos) | Disponible globalmente |
| CREATE OR REPLACE FUNCTION | Crea o reemplaza una función existente. Si no existe, la crea; si ya existe, la sobrescribe. | Permanente (en la base de datos) | Disponible globalmente |
| FUNCTION (sin CREATE) | Declara una función **local** dentro de un bloque anónimo o un procedimiento. | Temporal (dura mientras el bloque se ejecuta) | Disponible solo en el bloque local |