



Introducción al lenguaje PL/SQL

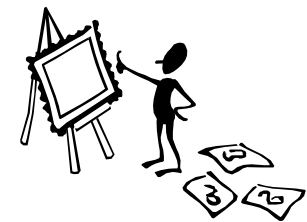
**Diseño de Bases de Datos y Seguridad de
la Información**





Índice

1. Introducción al PL/SQL
2. Conceptos Básicos de PL/SQL
 - Estructura de Bloques
 - Reglas y Convenciones del Lenguaje
 - Entrada y Salida de Datos
3. Variables de PL/SQL
4. Estructuras de Control
5. Cursores
6. Procedimientos y Funciones Almacenados
7. Paquetes
8. Tratamiento de los Errores y Excepciones
9. Disparadores
 - Bases de Datos Activas
 - Disparadores en ORACLE





Introducción

ORACLE PL/SQL

- **Procedural Language/Structured Query Language**
- Apareció por primera vez en ORACLE versión 6 (1988).
- Es un entorno de programación que reside **directamente** en la **BD**.
- Lenguaje de programación **sencillo** similar a C y ADA.
- Lenguaje **procedimental** que amplía la funcionalidad de SQL añadiendo estructuras habituales en otros lenguajes: variables y tipos, estructuras de control, procedimientos y funciones, ...



ORACLE PL/SQL

Creación de Programas

- Se pueden crear con cualquier editor y ejecutarlos desde el *prompt* del SQL*PLUS con **START** o **@**.
- Los ficheros creados serán de texto y tendrán la extensión **.sql**.
- Para que un fichero se ejecute correctamente debe tener en su **última línea** el símbolo **/**.



Conceptos Básicos de PL/SQL

Estructura de Bloques

- La unidad básica en PL/SQL es el **bloque**.
- Todos los programas PL/SQL están **compuestos por bloques**, que pueden definirse de forma secuencial o estar anidados.
- Normalmente cada bloque realiza una *unidad lógica* de *trabajo* en el programa, separando así unas tareas de otras.



Estructura de Bloques

Hay diferentes tipos de bloques:

- **Anónimos (*Anonymous blocks*)**.- se construyen de forma dinámica y se ejecutan una sola vez. Su código no se almacena en la BD.
- **Con nombre (*Named blocks*)**.- son bloques con nombre (incluyen una cabecera) que se compilan y almacenan en la base de datos para su posterior ejecución.
 - **Subprogramas**.- procedimientos, funciones o paquetes almacenados en la BD. No suelen cambiar después de su construcción y se ejecutan múltiples veces mediante una llamada al mismo.
 - **Disparadores (*Triggers*)**.- son bloques con nombre que también se almacenan en la BD. Tampoco suelen cambiar después de su construcción y se ejecutan varias veces. Se ejecutan ante algún suceso de disparo, que será una orden del lenguaje de manipulación de datos (INSERT, UPDATE o DELETE) que se ejecuta sobre una tabla de la BD.



Estructura de Bloques

Todos los bloques tienen tres secciones diferenciadas:

- **Sección Declarativa:**
 - Donde se localizan todas las **variables**, **cursores** y **tipos** usados por el bloque. También se pueden declarar en esta sección las **funciones** y **procedimientos locales**. Estos subprogramas estarán disponibles solo para ese bloque.
- **Sección Ejecutable:**
 - Donde se lleva a cabo el trabajo del bloque. En esta sección pueden aparecer tanto **órdenes SQL** (LDD y LMD) como **órdenes procedimentales**.
- **Sección Errores:**
 - El código de esta sección no se ejecutará a menos que ocurra un **error**.



Estructura de Bloques

DECLARE

Declaración de variables y subprogramas

Esta sección es opcional

BEGIN → requerido

Sección ejecutable, se incluyen las sentencias de SQL y llamadas a procedimientos

Es la sección principal del bloque y es obligatoria (*a/ menos debe haber una orden ejecutable*)

EXCEPTION

Zona de excepción, se pueden colocar aquí las sentencias en caso de error

Esta sección es opcional

END; → requerido

/ → requerido



Conceptos Básicos de PL/SQL

Estructura de Bloques

Ejemplo:

```
DECLARE

V_Num NUMBER; --primero las variables y luego procedimientos y funciones locales
PROCEDURE Ejemplo IS
    BEGIN
        DBMS_OUTPUT.PUT_LINE('Llamada al procedimiento local.');
```

END Ejemplo;

```

BEGIN
    DBMS_OUTPUT.PUT_LINE('Mi primer ejemplo.');
```

V_Num:= 7;

```

    DBMS_OUTPUT.PUT_LINE(V_Num);
    Ejemplo();

END;
/
```



Reglas y Convenciones del Lenguaje

Unidades Léxicas:

- Secuencia de caracteres admitida
- Conjunto de caracteres permitido en PL/SQL :
 - Letras mayúsculas y minúsculas: A – Z y a – z
 - Dígitos: 0 – 9
 - Espacios en blanco: tabuladores, caracteres de espaciado y retorno de carro
 - Símbolos matemáticos: + - * / < > =
 - Símbolos de puntuación: () { } [] ¿ ¡ ; : . ' " @ # % ~ & _



Reglas y Convenciones del Lenguaje

Identificadores:

- Se emplean para dar nombre a los objetos PL/SQL, tales como variables, cursores, tipos y subprogramas.
- Los identificadores constan de una **letra**, seguida por una secuencia opcional de caracteres, que pueden incluir **letras**, **números**, signos de dólar (\$), caracteres de subrayado (_) y símbolos de almohadilla (#). Los demás caracteres no pueden emplearse.
- La longitud **máxima** de un identificador es de **30 caracteres** y todos los caracteres son significativos.

– Ejemplos válidos

- X, V_ENAME, CodEmp, V1, V2_, ES_UNA_VARIABLE_#, V_\$_Cod

– Variables MAL declaradas

- X+Y, _ENAME, Cod Emp, 1V, ESTA ES VARIABLE (+ de 30 caracteres)



Reglas y Convenciones del Lenguaje

Mayúsculas y minúsculas:

Hay que tener en cuenta que PL/SQL **no** diferencia entre mayúsculas y minúsculas, así todos los siguientes identificadores serían equivalentes desde el punto de vista de PL/SQL:

- NUM_EMP
- Num_emP
- num_emp
- nUM_Emp

Palabras reservadas:

- *A la hora de declarar identificadores hemos de tener en cuenta las **palabras reservadas**, que son palabras que tienen un significado especial para PL/SQL y que no podemos utilizar como variables porque tendríamos problemas de compilación, estas palabras son tales como **DECLARE, BEGIN, END...***



Conceptos Básicos de PL/SQL

Reglas y Convenciones del Lenguaje

Delimitadores:

Son símbolos que tienen un significado especial para PL/SQL y que se utilizan para separar unos identificadores de otros:

SIMBOLO	DESCRIPCIÓN	SIMBOLO	DESCRIPCIÓN
+	Operador de suma	-	Operador de resta
*	Operador de multiplicación	/	Operador de división
=	Operador de igualdad	<	Operador “menor que”
>	Operador “mayor que”	(Delimitador inicial de expresión
)	Delimitador final de expresión	;	Terminador de orden
%	Indicador de atributo	,	Separador de elementos
.	Selector de componente	@	Delimitador de enlace a BD.
‘	Delimitador cadena caracteres	“	Delimitador cadena entrecomillada
:	Indicador variable asignación	**	Operador de exponenciación
<>	Operador “distinto de”	!=	Operador “distinto de”
<=	Operador “menor o igual que”	>=	Operador “mayor o igual que”
:=	Operador de asignación	=>	Operador de asociación
	Operador de concatenación	--	Comentario, una sola línea
<<	Comienzo de etiqueta	>>	Fin de etiqueta
*/	Cierre de comentario multilínea	<space>	Espacio
<tab>	Carácter de tabulación	<cr>	Retorno de carro



Reglas y Convenciones del Lenguaje

Literales (*Valores*):

- **Carácter**

Constan de uno o más caracteres delimitados por comillas simples. Se pueden asignar a variables de tipo CHAR o VARCHAR2, sin tener que hacer ningún tipo de conversión:

`'12345'`

`'100%'`

- **Numérico**

Representa un valor entero o real, puede asignarse a una variable de tipo NUMBER sin tener que efectuar conversión alguna. Los literales **enteros** consisten de una serie de dígitos, precedidos opcionalmente por un signo (+ o -). No se permite utilizar un punto decimal en un literal entero. `123` `+7` `-9`

Un literal **real** consta de signo, opcional, y una serie de dígitos que contiene punto decimal. También pueden escribirse utilizando notación científica.

`-17.7`

`23.0`

`1.345E7` `-7.12e+12`



Conceptos Básicos de PL/SQL

Reglas y Convenciones del Lenguaje

Literales (*Valores*):

- **BOOLEAN**

Los literales booleanos representan la verdad o falsedad de una condición y se utilizan en las órdenes IF o LOOP, sólo existen tres posibles literales booleanos (*sin comillas*):

TRUE

Verdadero

FALSE

Falso

NULL

Nulo

- **DATE**

Los literales de tipo DATE almacenan información del año, mes y día (hora, minutos y segundos en el caso de TIMESTAMP)

'05-JUN-07'

'23-01-2007'



Reglas y Convenciones del Lenguaje

Comentarios:

- **Monolínea.-** Comienzan con dos guiones y continua hasta el final de la línea.

-- Esto es un comentario

- **Multilínea.-** Comienzan con el delimitador `/*` y terminan con el delimitador `*/`.

/* Esto es otro comentario, que puede abarcar
varias líneas */



Entrada y Salida de Datos

- **Para mostrar un valor por pantalla:**

`DBMS_OUTPUT.PUT_LINE(cadena);`

En caso de que el valor no sea una cadena, se puede usar la función *TO_CHAR* para transformarlo.

Es necesario activar la opción *SERVEROUTPUT* mediante la siguiente instrucción en el SQL*Plus:

`SET SERVEROUTPUT ON;`

`SET SERVEROUTPUT ON SIZE tamaño del buffer;`

/ Se puede incluir en el fichero glogin.sql*/*



Entrada y Salida de Datos

- Para **leer valores por pantalla** se puede usar el comando ACCEPT y las variables de sustitución:
 - **Variable de sustitución:** pueden aparecer directamente en la sentencia SELECT sin necesidad de definirla, anteponiéndola el símbolo **&** y SQL nos preguntará el valor que queremos asignarle.
 - **ACCEPT** permite declarar una variable de entorno y leer su valor poniendo el mensaje deseado en el Prompt.

ACCEPT variable [NUMBER|CHAR|DATE] [FORMAT]
[PROMPT text] [HIDE]

Para utilizar la variable accedemos a ella anteponiéndole el símbolo &.

PERO: No podemos utilizar ACCEPT para leer variables **dentro de un bloque PL/SQL**, si queremos utilizarlo debemos hacerlo fuera del mismo.



Conceptos Básicos de PL/SQL

Entrada y Salida de Datos

Ejemplo (usando variables de sustitución):

```
SELECT numEmp, nombreEmp, salario, numDept  
FROM Empleados  
WHERE numEmp = &Num_Emp;
```

En el *prompt* aparecerá el siguiente mensaje:

Enter value for Num_Emp:



Conceptos Básicos de PL/SQL

Entrada y Salida de Datos

Ejemplo (usando variables de sustitución):

```
SET VERIFY OFF -- para no mostrar por pantalla el valor anterior de la variable
SET SERVEROUTPUT ON
DECLARE
    v_fecha DATE:='&fecha';
BEGIN
    DBMS_OUTPUT.PUT_LINE('La fecha introducida es: '||TO_CHAR(v_fecha));
END;
/
```

>Introduzca un valor para **fecha**: 11/12/2007

>La fecha introducida es: 11/12/07

>Procedimiento PL/SQL terminado correctamente.
SQL>



Conceptos Básicos de PL/SQL

Entrada y Salida de Datos

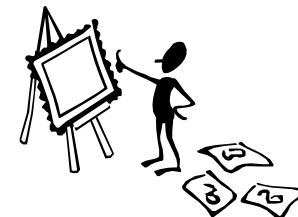
Ejemplo (usando ACCEPT):

```
SET VERIFY OFF -- para no mostrar por pantalla el valor anterior de la  
variable  
SET SERVEROUTPUT ON  
ACCEPT producto NUMBER FORMAT 9999 PROMPT 'Introduce el precio: '  
ACCEPT iva NUMBER FORMAT 99.99 PROMPT 'Introduce el IVA: '  
DECLARE  
    v_producto NUMBER:= &producto;  
    v_iva NUMBER:= &iva;  
  
...  
BEGIN  
  
...  
END;  
/
```



Índice

1. *Introducción al PL/SQL*
2. *Conceptos Básicos de PL/SQL*
 - *Estructura de Bloques*
 - *Reglas y Convenciones del Lenguaje*
 - *Entrada y Salida de Datos*
- ⇒ **3. Variables de PL/SQL**
4. Estructuras de Control
5. Cursores
6. Procedimientos y Funciones Almacenados
7. Paquetes
8. Tratamiento de los Errores y Excepciones
9. Disparadores
 - Bases de Datos Activas
 - Disparadores en ORACLE





Variables de PL/SQL

- Se definen en la sección DECLARE.
- Las variables deben tener un tipo asociado.
- Su formato es el siguiente:

Nombre_variable tipo_variable [:= valor_inicial];

DECLARE

V_Fecha_Ingreso DATE := SYSDATE;



Fecha actual
del sistema

Es preferible inicializar una variable siempre que su valor se pueda determinar, no obstante, PL/SQL define el contenido de las variables no inicializadas, asignándolas el valor NULL (valor desconocido o no definido).



Variables de PL/SQL

NUMÉRICO: Contienen un valor numérico entero o de punto flotante

- Number**: numérico
- Dec, decimal**: numérico decimal
- Double precision**: doble precisión
- Integer, int**: enteros
- Real**: reales
- Smallint**: entero corto
- Binary_integer**: enteros (se usa para almacenar valores que sólo se van a usar en cálculos y no se van a almacenar en la BD).
- Natural**: números naturales
- Positive**: números positivos

CARÁCTER: contienen un conjunto de caracteres

- Varchar**
- Varchar2**
- Char**
- Character**
- Long**

ROWID: Se puede almacenar un identificador de columna que es una clave que identifica unívocamente a cada fila de la base de datos

- Rowid**

RAW (Tipo Binario); Se emplean para almacenar datos binarios

- Raw**
- Long Raw**

BOOLEAN: Sólo pueden contener los valores TRUE, FALSE o NULL

TIPOS COMPUESTOS: Consta de una serie de componentes (TYPE ... IS RECORD| IS TABLE OF ...)

- Record**
- Table**



Variables de PL/SQL

UTILIZACIÓN DE %TYPE

- Hay ocasiones que las variables que usamos en PL/SQL se emplean para manipular datos almacenados en una tabla de la BD. En estos casos tendrá que ser la variable del mismo tipo que las columnas de las tablas.

DECLARE

v_salarios Empleado.salario%TYPE;



REGISTROS

La sintaxis general para definir un tipo de registro es:

```
TYPE tipo_registro IS RECORD (  
    Campo1 Tipo1 [NOT NULL] [:= expr1],  
    Campo2 Tipo2 [NOT NULL] [:= expr2],  
    Campo3 Tipo3 [NOT NULL] [:= expr3],  
    .....  
    Campo(n) Tipo(n) [NOT NULL] [:= expr(n)]);
```

Nombre_registro tipo_registro;

Para hacer referencia a los campos de un registro se utiliza la notación punto (.).

Nombre_registro.Campo1

Para poder asignar un registro a otro, ambos deben ser del mismo tipo.



Variables de PL/SQL

REGISTROS

Ejemplo:

DECLARE

TYPE tipoDireccion **IS RECORD** (
 tipoVia VARCHAR2(10) NOT NULL := 'CALLE',
 nombreVia VARCHAR2(50),
 ciudad VARCHAR2(25),
 cp VARCHAR2(5));

V_Direccion tipoDireccion;

BEGIN

-- Inicialización de campos de la variable

V_Direccion.nombreVia := 'ALCALA';

V_Direccion.ciudad := 'MADRID';

V_Direccion.cp := '28901';

DBMS_OUTPUT.PUT_LINE(V_Direccion.tipoVia || ' ' || V_Direccion.nombreVia);

...

END;



REGISTROS

Es típico en el trabajo con BD el declarar registros con el mismo formato que las filas de las tablas.

Si se conoce la estructura de las tablas se crea un registro con ese formato, si no se utiliza el operador **%ROWTYPE**, similar a %TYPE. El registro tendrá los mismos campos y del mismo tipo que la tupla de la tabla correspondiente de la BD.



Variables de PL/SQL

REGISTROS

Ejemplo:

DECLARE

--

V_Empleados Empleado%ROWTYPE;

BEGIN

-- Inicialización de campos de la variable

V_Empleados.codigo:='Emp01';

V_Empleados.nombre:='Pablo Martínez';

V_Empleados.ciudad:='MADRID';

...

END;



REGISTROS

- También se pueden asignar valores de un registro completo mediante una SELECT que extraería los datos de la BD y los almacena en el registro.

```
SELECT nombre, apellido, carrera  
INTO V_Registro  
FROM Estudiantes  
WHERE ID_Estudiante=1000;
```

V_Registro es una variable de tipo T_Registro:

```
TYPE T_Registro IS RECORD(  
  Nombres Estudiantes.nombre%TYPE,  
  Apellidos Estudiantes.apellido%TYPE,  
  Carrera Estudiantes.carrera%TYPE);
```



Variables de PL/SQL

TABLAS

Para poder declarar una tabla es necesario primero definir su tipo y luego una variable de dicho tipo.

La sintaxis general para definir un tipo de tabla es:

TYPE *tipotabla* **IS TABLE OF** *tipo* [NOT NULL];

Donde:

tipotabla.- es el nombre del nuevo tipo que está siendo definido

tipo.- es un tipo predefinido o una referencia a un tipo mediante
%TYPE

Una vez declarados el tipo y la variable, podemos hacer referencia a un elemento determinado de la tabla PL/SQL mediante la sintaxis:

nombretabla (índice)

Donde nombretabla es el nombre de una tabla, e índice es la posición del elemento dentro de la tabla.



Variables de PL/SQL

TABLAS

Ejemplo:

```
DECLARE
```

```
TYPE tipo_tabla IS TABLE OF INTEGER;
```

```
V_tabla tipo_tabla := tipo_tabla (0,1,2,3,4,5,6,7,8,9);
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE(V_tabla(1));
```

```
    DBMS_OUTPUT.PUT_LINE(V_tabla(5));
```

```
    DBMS_OUTPUT.PUT_LINE(V_tabla(10));
```

```
END;
```

```
/
```

```
0
```

```
4
```

```
9
```




ATRIBUTOS DE LA TABLA

COUNT. Devuelve el número de elementos de la tabla

V_Table.Count ⇒ 10

DELETE(I). Borra fila de la tabla **V_Table.Delete(2)** ⇒ borra elemento de la posición 2 de la tabla V_Table (*no se redistribuyen*)

DELETE(I,S). Borra filas de la tabla **V_Table.Delete(2,4)** ⇒ borra de la posición 2 a la 4 de la tabla V_Table

EXISTS BOOLEAN. Devuelve TRUE si existe en la tabla el elemento especificado. **V_Table.Exists(1)** ⇒ TRUE

FIRST BINARY_INTEGER. Devuelve el índice de la primera fila de la tabla. **V_Table.First** ⇒ 1

LAST BINARY_INTEGER. Devuelve el índice de la última fila de la tabla. **V_Table.Last** ⇒ 10

NEXT BINARY_INTEGER. Devuelve el índice de la fila de la tabla que sigue a la fila especificada. **V_Table.Next** ⇒ índice siguiente

PRIOR BINARY_INTEGER. Devuelve el índice de la fila de la tabla que antecede a la fila especificada, **V_Table.Prior** ⇒ índice anterior



Variables de PL/SQL

Consideraciones a tener en cuenta:

- DELETE constituye una orden completa por sí mismo; no se lo utiliza como parte de una expresión como sucede con los otros atributos.
- EXISTS devolverá TRUE si existe el elemento buscado en caso contrario devolverá FALSE. Este atributo es útil para evitar el error ORA-1403 que se produce cuando el elemento no existe.
- Tanto FIRST como LAST devolverán el índice, no el valor contenido en dichas filas.
- Excepto usando el atributo DELETE no hay manera de borrar todas las filas de una tabla



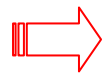
Índice

1. *Introducción al PL/SQL*

2. *Conceptos Básicos de PL/SQL*

- *Estructura de Bloques*
- *Reglas y Convenciones del Lenguaje*
- *Entrada y Salida de Datos*

3. *Variables de PL/SQL*



4. Estructuras de Control

5. **Cursores**

6. **Procedimientos y Funciones Almacenados**

7. **Paquetes**

8. **Tratamiento de los Errores y Excepciones**

9. **Disparadores**

- **Bases de Datos Activas**
- **Disparadores en ORACLE**





Estructuras de Control

1) Estructuras lógicas: **IF – THEN – ELSE**

- A) IF-THEN
- B) IF-THEN-ELSE
- C) IF-THEN-ELSIF

2) Expresiones **CASE**

3) Estructuras de **BUCLE**

- A) Bucles simples
- B) Bucles WHILE
- C) Bucles FOR

4) **GOTO y ETIQUETAS**

- A) Restricciones de GOTO
- B) Etiquetado



A) Estructuras lógicas: IF – THEN – ELSE

Su sintaxis es:

```
IF <expresión_booleana> THEN
    Secuencia_de_órdenes;
[ELSIF <expresión_booleana> THEN
    Secuencia_de_órdenes;]
...
[ELSE
    Secuencia_de_órdenes;]
END IF;
```

Donde *<expresiones booleanas>* es cualquier expresión que de como resultado un valor booleano. Las cláusulas ELSIF y ELSE son opcionales y puede haber tantas cláusulas ELSIF como se quiera.



Estructuras de Control

1) Estructuras lógicas: IF – THEN – ELSE

A) IF – THEN

Si se evalúa la condición y resulta **verdadera**, se ejecutan uno o más líneas de código de programa.

En el caso de que la condición resulte ser **falsa**, **NO se realiza NINGUNA acción**.

```
IF fecha_nac<'1-01-1970' THEN -- No termina con un ;  
    salario:= salario *1.15; -- aumento de salario en un 15%  
END IF;
```

Se pueden anidar varias instrucciones:

```
IF fecha_nac<'1-01-1970' THEN  
    IF apellido ='Martínez' THEN -- IF ANIDADO  
        salario:= salario *1.15; -- aumento de salario en un 15%  
    END IF; -- END IF OBLIGATORIO  
END IF;
```



Estructuras de Control

1) Estructuras lógicas: IF – THEN – ELSE

B) IF – THEN – ELSE

Si se evalúa la condición y resulta **verdadera**, se ejecutan uno o más líneas de código de programa.

En el caso de que la condición resulte ser **falsa**, se ejecutan las instrucciones que siguen a la instrucción ELSE.

```
IF fecha_nac<'1-01-1970' THEN -- No termina con un ;  
    salario:= salario *1.15; -- aumento de salario en un 15%  
ELSE -- No termina con un ;  
    salario:= salario* 1.05; -- aumento de salario en un 5%  
END IF;
```

Se pueden anidar varias instrucciones IF-THEN-ELSE.

Sólo se permite una instrucción ELSE en cada instrucción IF .



Estructuras de Control

1) Estructuras lógicas: IF – THEN – ELSE

C) IF – THEN – ELSIF

Si se evalúa la condición y resulta **verdadera**, se ejecutan uno o más líneas de código de programa.

En el caso de que la condición resulte ser **falsa**, se evalúa la condición especificada en el ELSIF.

```
IF apellido ='Pérez' THEN
    salario:= salario *1.10; -- aumento de salario en un 10%
ELSIF apellido ='Martínez' THEN
    salario:= salario *1.15; -- aumento de salario en un 15%
ELSIF apellido='Alvarez' THEN
    salario:= salario *1.20; -- aumento de salario en un 20%
ELSE
    salario:= salario* 1.05; -- aumento de salario en un 5%
END IF; -- Sólo se necesita un único END IF
```




Estructuras de Control

Condiciones Nulas

Procedimiento que nos dice si el número 1 (Num1) es mayor que el número 2 (Num2) :

```
DECLARE
    Num1  NUMBER :=3;
    Num2  NUMBER; -- Como no inicializamos la variable, su valor es NULL
    EsMayor  VARCHAR2(15);
BEGIN
    ...
    IF Num1 < Num2 THEN
        EsMayor := 'Yes';
    ELSE
        EsMayor := 'No';
    END IF;
END;
```

**¡¡Es muy importante
inicializar siempre las
Variables!!**

¿¿¿Solución???



Estructuras de Control

2) Expresiones CASE

La instrucción CASE es una evolución en el control lógico.

Se diferencia de las estructuras IF-THEN-ELSE en que se puede utilizar una estructura simple para realizar selecciones lógicas en una lista de valores.

Puede utilizarse también para establecer el valor de una variable.

Su sintaxis es:

```
CASE [variable]
  WHEN expresión1 THEN valor1;
  WHEN expresión2 THEN valor2;
  WHEN expresión3 THEN valor3;
  WHEN expresión4 THEN valor4;
  ELSE valor5;
END CASE;
```

No existe límite para el número de expresiones que se pueden definir en una expresión CASE.

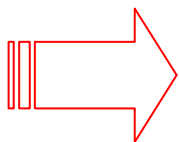


Estructuras de Control

2) Expresiones CASE

EJEMPLO:

```
DECLARE
  equipo varchar(100);
  ciudad varchar(50):= 'MADRID';
BEGIN
  CASE ciudad
    WHEN 'MADRID' THEN equipo:='RealMadrid';
    WHEN 'BARCELONA' THEN equipo:='FCBarcelona';
    WHEN 'LACORUÑA' THEN equipo:= 'Deportivo de La Coruña';
    ELSE equipo:='SIN EQUIPO';
  END CASE;
  DBMS_OUTPUT.PUT_LINE(equipo);
END;
/
```



Expresión a evaluar se pasa al principio de la sentencia CASE:
ciudad= 'VALOR'



Estructuras de Control

2) Expresiones CASE

Cada cláusula WHEN puede tener su propia expresión a evaluar. En este caso, después del CASE no aparece ninguna expresión.

EJEMPLO:

CASE

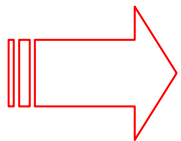
WHEN precio<11 THEN descuento:=2;

WHEN precio>10 and precio<25 THEN descuento:=5;

WHEN precio>24 THEN descuento:=10;

ELSE descuento:=15;

END CASE;



Expresión a evaluar se pasa en cada cláusula WHEN



3) Estructuras de BUCLE

A) Bucles simples

Su sintaxis es:

```
LOOP  
    <Secuencia_de_órdenes>;  
END LOOP;
```

⇒ Este bucle sería *infinito*, no tiene condición de parada.

Para salir de un bucle le pondremos la **orden EXIT**, que su sentencia es:

```
EXIT [WHEN <Condición>;
```

Esta orden sería equivalente a:

```
IF <Condición> THEN  
    EXIT;  
END IF;
```



Estructuras de Control

3) Estructuras de BUCLE

A) Bucles simples

EJEMPLO:

```
DECLARE
    V_Contador BINARY_INTEGER:=1;
BEGIN
    LOOP
        INSERT INTO Tabla (Valor)
        VALUES (V_Contador);
        V_Contador:=V_Contador +1;
        EXIT WHEN V_Contador =10;
    END LOOP;
END;
```



Estructuras de Control

3) Estructuras de BUCLE

B) Bucles WHILE

Su sintaxis es:

```
WHILE <Condición> LOOP  
    <Secuencia_de_órdenes>;  
END LOOP;
```

Antes de entrar en el bucle evalúa la condición, si es verdadera, entrará.
Si la condición es falsa o nula el bucle se termina.

Hay que tener en cuenta que si la condición del bucle no toma el valor TRUE la primera vez que se le comprueba el bucle, no llegará nunca a ejecutarse.

Pueden usarse las órdenes **EXIT** o **EXIT WHEN** dentro de un bucle WHILE para salir del bucle, sin llegar a terminar la condición.



Estructuras de Control

3) Estructuras de BUCLE

B) Bucles WHILE

EJEMPLO:

```
DECLARE
    V_Contador BINARY_INTEGER:=1;
BEGIN
    WHILE Contador <11 LOOP
        INSERT INTO Tabla (Valor)
        VALUES (V_Contador);
        V_Contador:=V_Contador +1;
    END LOOP;
END;
```




Estructuras de Control

3) Estructuras de BUCLE

C) Bucles FOR

En el caso en que sepamos el número de iteraciones en que se ejecutarán los bucles simples y WHILE utilizaremos los bucles FOR.

Su sintaxis es:

```
FOR <contador_bucle> IN [ REVERSE ] menor . . mayor LOOP  
    Secuencia_de_órdenes  
END LOOP;
```

donde <contador_bucle> es una variable que no hace falta que se declare ya que lo hace de forma implícita como BINARY_INTEGER.

Los valores *menor...mayor* muestra el rango en que se ejecutará el bucle.



3) Estructuras de BUCLE

C) Bucles FOR

EJEMPLO:

```
BEGIN
  FOR V_Contador IN 1..10 LOOP
    INSERT INTO Tabla (Valor)
      VALUES (V_Contador);
  END LOOP;
END;
```



4) GOTO y etiquetas

Su sintaxis es:

GOTO *<Etiqueta>;*

donde *<Etiqueta>* es una etiqueta definida en el bloque PL/SQL.

Al evaluar una orden GOTO el control pasa inmediatamente a la orden identificada por la etiqueta, por ejemplo:

BEGIN

DBMS_OUTPUT.PUT_LINE('Esto es un ejemplo.');

GOTO Etiqueta_1;

DBMS_OUTPUT.PUT_LINE('No hace el GOTO.');

<<Etiqueta_1>>

DBMS_OUTPUT.PUT_LINE('Entra en el GOTO.');

END;



Estructuras de Control

4) GOTO y etiquetas

USO:

P.e.: Para hacer más legible el bloque de ejecución con manejadores de excepciones complejos en bloques anidados.

Restricciones de GOTO

- No se puede saltar al interior de un bloque anidado
- No se puede saltar al interior de un bucle
- No se puede saltar al interior de una orden IF

Etiquetado

A los bucles pueden ponérseles etiquetas de forma que las usemos en la sentencia EXIT. En el caso de que se le añada una etiqueta a un bucle habrá que ponerla también al final del bucle.



Índice

1. *Introducción al PL/SQL*
2. *Conceptos Básicos de PL/SQL*
 - *Estructura de Bloques*
 - *Reglas y Convenciones del Lenguaje*
 - *Entrada y Salida de Datos*
3. *Variables de PL/SQL*
4. *Estructuras de Control*
- ⇒ **5. Cursores**
6. Procedimientos y Funciones Almacenados
7. Paquetes
8. Tratamiento de los Errores y Excepciones
9. Disparadores
 - Bases de Datos Activas
 - Disparadores en ORACLE





Procesamiento de un CURSOR explícito

Un CURSOR es un puntero al ***área de contexto***, que es un área de memoria que contiene información sobre el procesamiento, el número de filas procesadas y el conjunto de filas resultado de la consulta.

Pasos necesarios para procesar un cursor:

- Declaración de un CURSOR (*DECLARE*).
- Apertura del CURSOR para una consulta.
- Recuperar los resultados de las variables PL/SQL.
- Cierre del CURSOR.



Declaración de un CURSOR

Sintaxis:

CURSOR nombre_cursor IS sentencia_SELECT;

Nota:

El nombre del cursor es un identificador PL/SQL, y ha de ser declarado en la sección declarativa del bloque antes de poder hacer referencia a él.



Apertura de un CURSOR

La sintaxis para abrir un cursor es:

OPEN *nombre_cursor*;

Al abrir un cursor suceden tres cosas:

- 1.- Se examinan los valores de las variables aceptadas.
- 2.- Se determina el conjunto activo, basándose en los valores de dichas variables
- 3.- Se hace apuntar el puntero del conjunto activo a la primera fila.

- **El conjunto activo:** es el conjunto de filas que satisfacen los criterios de la consulta se determina en el momento de abrir un cursor.
- La cláusula **WHERE se evalúa** para la tabla o tablas referenciadas en la cláusula FROM de la consulta, y todas las filas para las cuales se evalúe la condición como TRUE son añadidas al conjunto activo.
- También se establece un **puntero** al conjunto activo en el momento de abrir el cursor. Este puntero indica cuál es la siguiente fila que el cursor extraerá.
- Es legal abrir un cursor que ya esté abierto. PL/SQL ejecutará implícitamente una orden CLOSE antes de reabrir el cursor con la segunda orden OPEN.
- También puede haber más de un cursor abierto al mismo tiempo (*OPEN_CURSORS*).



Cursores

Ejemplo:

DECLARE

CURSOR C_Libros IS SELECT Titulo, Fecha FROM T_LIBRO WHERE Autor ='DATE';

BEGIN

OPEN C_Libros;

...

T_LIBRO

Titulo	Fecha	Autor
Introducción a las BD	1990	DATE
SGBD	2002	DE MIGUEL
Oracle 10g	2007	URMAN
BD Relacionales	1995	DATE

OPEN
CURSOR

CURSOR
SELECT
C_Libros

Conjunto Activo

Titulo	Fecha
Introducción a las BD	1990
BD Relacionales	1995

Puntero a
la primera fila



Extracción de los datos de un CURSOR

La cláusula INTO de la consulta es parte de la orden FETCH. Dicha orden tiene dos formas posibles:

```
FETCH nombre_cursor INTO lista_variables;  
FETCH nombre_cursor INTO registro_PL/SQL;
```

Nombre_cursor: Identifica a un cursor abierto y ya declarado.

Lista_variables: Es una lista de variables PL/SQL previamente declaradas y separadas por comas.

Registro_PL/SQL: Es un registro PL/SQL previamente declarado.

En ambos casos, la variable o variables de la cláusula INTO deben ser compatibles en cuanto a tipo con la lista de selección de la consulta.

Después de cada FETCH, se incrementa el puntero activo, para que apunte a la siguiente fila. De esta forma, cada FETCH devolverá filas sucesivas del conjunto activo, hasta que se devuelve el conjunto completo.



CIERRE de un CURSOR

Cuando se ha terminado de extraer el conjunto activo, debe cerrarse el cursor.

Esta acción informa a PL/SQL de que el programa **ha terminado** de utilizar el cursor, y de que se pueden **liberar los recursos** con él asociados. Estos recursos incluyen las áreas de almacenamiento empleadas para contener el conjunto activo, así como cualquier espacio temporal utilizado en la determinación de dicho conjunto.

La sintaxis para el cierre del cursor es:

`CLOSE nombre_cursor;`

Si intentamos cerrar un cursor que ya está cerrado nos daría un **error ORA-1001**.



Excepciones `NO_DATA_FOUND` y `%NOTFOUND`

La excepción `NO_DATA_FOUND` se produce sólo en las órdenes `SELECT.. INTO..`, cuando la cláusula `WHERE` de la consulta no se corresponde con **ninguna fila**.

En el caso de los cursores explícitos, por el contrario, cuando la cláusula `WHERE` no se corresponde con ninguna fila, lo que sucede es que el atributo `%NOTFOUND` toma el valor `TRUE`.

Si la cláusula `WHERE` de una orden `UPDATE` o `DELETE` no se corresponde con ninguna fila, el atributo SQL `%NOTFOUND` toma el valor `TRUE`, en lugar de producirse la excepción `NO_DATA_FOUND`.

⇒ Debido a esta circunstancia todos los **bucles de extracción** usan `%NOTFOUND` o `%FOUND` para determinar la condición de salida del bucle, en lugar de la excepción `NO_DATA_FOUND`.



NO_DATA_FOUND y %NOTFOUND

Ejemplo:

BEGIN

UPDATE Rooms

SET number_seats=100

WHERE room_id=10101;

IF **SQL%NOTFOUND** THEN

INSERT INTO Rooms (room_id, number_seats)
VALUES (10101,100);

END IF;

END;

/

Si la anterior orden UPDATE no se aplica a **ninguna fila**, inserta una nueva fila en la tabla

Alternativa: IF **SQL%ROWCOUNT=0** THEN



Ejemplo

```
CREATE TABLE Autor
(Nombre_A VARCHAR(50) PRIMARY KEY,
Fecha_Nac DATE);

INSERT INTO AUTOR VALUES ('Mario Piattini', '01/01/1960');
INSERT INTO AUTOR VALUES ('Adoración de Miguel', '01/01/1940');
```

```
SET SERVEROUTPUT ON;
DECLARE
```

```
V_Autor AUTOR.Nombre_A%TYPE;
CURSOR C_Autores IS SELECT Nombre_A FROM AUTOR;
```

```
BEGIN
```

```
OPEN C_Autores;
DBMS_OUTPUT.PUT_LINE('Lista de autores');
```

```
LOOP
```

```
FETCH C_Autores INTO V_Autor;
```

```
EXIT WHEN C_Autores%NOTFOUND;
```

```
DBMS_OUTPUT.PUT_LINE('El nombre del autor es: '||V_Autor);
```

```
END LOOP;
```

```
CLOSE C_Autores;
```

```
END;
```

```
/
```

V_Autor y Nombre_A
tienen que tener el
mismo tipo



Ejemplo

Resultado de la Ejecución:

```
/
Lista de autores
El nombre del autor es: Mario Piattini
El nombre del autor es: Adoración de Miguel

Procedimiento PL/SQL terminado correctamente.

SQL>
```



Bucles FOR para CURSORES

Hay una manera de manejar cursores sin tener que realizar los pasos de: apertura, mover y cierre de un cursor. Esto se consigue mediante la **utilización de bucles FOR**.

Ejemplo:

DECLARE

--Declaramos el cursor

CURSOR C_Estudiantes_Inf IS

SELECT Num_Mat, Nombre, Apellidos

FROM Estudiantes WHERE Titulacion = 'Informatica';

BEGIN

--Ahora es cuando empieza el bucle, se realiza **OPEN implícito** de C_Estudiantes_Inf

FOR V_Estudiantes IN C_Estudiantes_Inf LOOP

--El FETCH lo ejecuta de forma implícita

INSERT INTO **Estudiantes_Inf** (Matricula, NombreEstudiantes)

VALUES (V_Estudiantes.Num_Mat, V_Estudiantes.Nombre || ' ' || V_Estudiantes.Apellidos);

--Ahora, antes de que el bucle continúe, implícitamente se chequea que haya datos en el

--cursor C_Estudiantes_Inf;

END LOOP;

--Ahora se cierra el cursor

END;

No se ha declarado la variable **V_Estudiantes**, pero el propio compilador la declara al analizar el bucle FOR. Es una forma más sencilla y clara para el uso de los cursores.



SELECT FOR UPDATE (CURSORES)

Para el caso de que en un bucle se deseen **modificar las filas extraídas por el cursor**, PL/SQL proporciona una sintaxis específica.

Consta de dos partes:

- La cláusula **FOR UPDATE** en la declaración del CURSOR
- La cláusula **WHERE CURRENT OF** en la orden UPDATE o DELETE.

La cláusula FOR UPDATE es parte de una orden SELECT. Es la última cláusula de la orden, después de la cláusula ORDER BY (si es que existe).

Su sintaxis es:

CURSOR ... IS

SELECT... FROM...

FOR UPDATE [OF *referencia_columna*]

Donde *referencia_columna* es una columna o lista de columnas de la tabla sobre la que se realiza la consulta (la/s que se va/n a modificar).



SELECT FOR UPDATE (CURSORES)

Si el cursor se creó con la cláusula FOR UPDATE, en las sentencias UPDATE y DELETE se puede incluir la cláusula:

UPDATE *tabla*

...

WHERE **CURRENT OF** cursor;

DELETE FROM *tabla*

WHERE **CURRENT OF** cursor;



SELECT FOR UPDATE (CURSORES)

Ejemplo:

```
DECLARE
```

```
-- Variable para añadir estos créditos al total de cada estudiante  
V_Creditos Clases.num_creditos%TYPE;
```

```
-- Cursor que selecciona los estudiantes matriculados en 3º de Informática
```

```
CURSOR c_EstudiantesRegistrados IS
```

```
SELECT *
```

```
FROM Estudiantes
```

```
WHERE Id_Estudiante IN (SELECT Id_Estudiante  
                        FROM Estudiante_Registrado  
                        WHERE Departamento='INF'  
                        AND Curso='3')
```

```
FOR UPDATE OF Creditos_Actuales;
```

**Atributo que se
va a modificar**



SELECT FOR UPDATE (CURSORES)

Ejemplo (continuación):

BEGIN

FOR *V_Estudiantes* IN **C_EstudiantesRegistrados** LOOP

-- Selecciona los créditos de 3º de Informática

SELECT num_creditos

INTO V_Creditos

FROM Clases

WHERE Departamento='INF'

AND Curso='3';

-- Actualiza la fila que acaba de recuperar con el cursor

UPDATE Estudiantes

**Atributo que
se modifica**

Set Creditos_Actuales=Creditos_Actuales+V_Creditos

WHERE CURRENT OF C_EstudiantesRegistrados;

END LOOP;

COMMIT;

END;



CURSORES PARAMETRIZADOS

Permiten utilizar la orden OPEN para pasarle al cursor el valor de uno o varios parámetros:

```
DECLARE
```

```
    CURSOR C_Estudiantes (V_Titulacion Estudiante.titulacion%TYPE) IS  
        SELECT iden, nombre, apellidos  
        FROM Estudiantes  
        WHERE titulación=V_Titulacion;
```

```
BEGIN
```

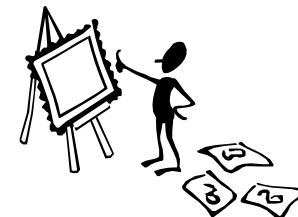
```
    OPEN C_Estudiantes ('Ing. Informática');
```

```
...
```



Índice

1. *Introducción al PL/SQL*
2. *Conceptos Básicos de PL/SQL*
 - *Estructura de Bloques*
 - *Reglas y Convenciones del Lenguaje*
 - *Entrada y Salida de Datos*
3. *Variables de PL/SQL*
4. *Estructuras de Control*
5. *Cursores*
- ⇒ 6. **Procedimientos y Funciones Almacenados**
7. Paquetes
8. Tratamiento de los Errores y Excepciones
9. Disparadores
 - Bases de Datos Activas
 - Disparadores en ORACLE





Procedimientos y Funciones Almacenados

CREACIÓN DE UN PROCEDIMIENTO

Un procedimiento PL/SQL es similar a los procedimientos de otros lenguajes de programación. Es un bloque con nombre que tiene la misma estructura que los bloques anónimos.

La sintaxis es:

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
    [ (argumento [ { IN | OUT | IN OUT } ] tipo,
      ...
      argumento [ { IN | OUT | IN OUT } ] tipo) ] { IS | AS }
[SECCIÓN DE DECLARACIÓN DE VARIABLES SIN palabra clave DECLARE]
BEGIN
    cuerpo_procedimiento
[EXCEPTION]
END [nombre_procedimiento];
```

Tras crear el **procedimiento**, éste se compila y luego se almacena en la BD de forma compilada. Este procedimiento luego puede ser invocado desde cualquier bloque PL/SQL.

Para borrar un procedimiento:

```
DROP PROCEDURE nombre_procedimiento;
```



Procedimientos y Funciones Almacenados

CREACIÓN DE UN PROCEDIMIENTO

nombre_procedimiento .- Nombre del procedimiento que se quiere crear

argumento .- Nombre de un parámetro del procedimiento

tipo .- Tipo del parámetro asociado

cuerpo_procedimiento .- Bloque PL/SQL que contiene el código del procedimiento

OR REPLACE.- Es para poder cambiar el código de un procedimiento que ya existía con anterioridad. Ya que esto es bastante común mientras se está desarrollando un procedimiento, para evitar eliminarlo y crearlo de nuevo, se utilizan estas palabras. Si está creado se elimina sin generar ningún mensaje de aviso, si no está creado se crea.

En el caso de que exista pero no se han incluido las palabras clave OR REPLACE, la orden CREATE devolverá el error ORACLE: *ORA – 00955 : name is already used by an existing object*



Procedimientos y Funciones Almacenados

CREACIÓN DE UN PROCEDIMIENTO

Los parámetros formales: IN, OUT o IN OUT.

Por defecto, **el modo IN**.

Las diferencias entre los tres modos son:

- **IN** .- El valor del parámetro se pasa al procedimiento cuando éste es invocado. Dentro del procedimiento el parámetro formal se considera como de solo lectura, y **no puede ser cambiado**. Cuando termina el procedimiento, y se devuelve el control al entorno que realizó la invocación, el parámetro no sufre cambios.
- **OUT** .- Se ignora cualquier valor que tenga el parámetro cuando se llama al procedimiento. Dentro del procedimiento, el parámetro formal se considera como de solo escritura, **no puede ser leído**, sino que tan solo pueden asignársele valores. Cuando termina el procedimiento y se devuelve el control al entorno que realizó la llamada, los contenidos del procedimiento formal se asignan al parámetro OUT.
- **IN OUT** .- Este modo es una combinación de IN y OUT. El valor del parámetro se pasa al procedimiento cuando éste es invocado: Dentro del procedimiento, el parámetro puede ser tanto leído como escrito. Cuando termina el procedimiento y se devuelve el control al entorno que realizó la llamada, los contenidos del parámetro se asignan al parámetro IN OUT.

En el caso de que no haya parámetros no se pondrán los paréntesis.



Tipos de los parámetros formales: IN, OUT o IN OUT.

En una declaración de procedimiento, no está permitido restringir un parámetro CHAR o VARCHAR2 con una determinada longitud, o un parámetro NUMBER con un valor de precisión y/o escala.

Como conclusión:

A la hora de definir los parámetros no se deben indicar los tamaños, ya que heredan las restricciones de las variables que se les asocian en la llamada al procedimiento.

Por lo que si una variable es de tipo VARCHAR2(3) y al parámetro se le asigna el siguiente valor "abcdefg" lo que se obtiene es un error (ORA – 6502). El único modo de que los parámetros adquieran restricciones completas es por medio de %TYPE por defecto.



VALORES POR DEFECTO (PARÁMETROS)

De igual manera que con las variables, los parámetros de un procedimiento o función, pueden asignárseles valores por defecto.

Su sintaxis es:

`Nombre_del_parámetro [modo] tipo_de_parámetro [DEFAULT valor_defecto]`

Cuando se llame al procedimiento en el caso de que no se especifique el valor para el parámetro, éste cogerá el valor por defecto.

Por comodidad es **recomendable** colocar los parámetros con valores por defecto al final de los argumentos para que a la hora de declarar la llamada del procedimiento y no se pasen valores no existan errores o confusiones.



Procedimientos y Funciones Almacenados

Ejemplo:

-- Creación de un procedimiento almacenado

```
CREATE OR REPLACE PROCEDURE Ejemplo (  
    Parametro1      NUMBER DEFAULT 10,  
    Parametro2      VARCHAR2 DEFAULT 'ABCD',  
    Parametro3      DATE DEFAULT SYSDATE) AS
```

```
BEGIN
```

```
    DBMS_OUTPUT.PUT_LINE(Parametro1||Parametro2||Parametro3);
```

```
END Ejemplo;
```

```
/
```

--Llamadas:

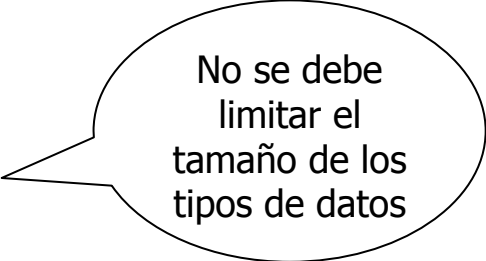
Si llamásemos así al procedimiento, el Parametro3 toma el valor por defecto.

```
Ejemplo ( 7, 'Cadena');
```

Si hubiésemos llamado así:

```
Ejemplo('Cadena','30/12/2005');
```

Al ejecutarse mostraría un error (**ORA-06502: PL/SQL**), ya que colocaría el primer valor en el primer parámetro, cuando este es de tipo NUMBER y no VARCHAR2.



No se debe
limitar el
tamaño de los
tipos de datos



CREACIÓN DE FUNCIONES

- Una función es bastante similar a un procedimiento:
 - Ambos aceptan argumentos, y estos pueden ser de cualquiera de los modos reseñados.
 - Ambos son formas diferentes de bloque PL/SQL, con sus secciones declarativa, ejecutable y de excepciones.
 - Ambos pueden ser almacenados en la BD o ser declarados dentro de un bloque.
- Sin embargo, una llamada a un procedimiento es una orden PL/SQL en sí misma, mientras que una **llamada a función** se realiza como **parte de su expresión**.



Procedimientos y Funciones Almacenados

CREACIÓN DE FUNCIONES

La sintaxis es:

```
CREATE [ OR REPLACE ] FUNCTION nombre_función
    [ ( argumento [ {IN | OUT | INOUT} ] tipo,
      ...
      argumento [ {IN | OUT | INOUT } ] tipo ) ]
RETURN return_tipo { IS | AS }
[SECCIÓN DE DECLARACIÓN DE VARIABLES]
BEGIN
    cuerpo_funcion
[EXCEPTION]
END [nombre_función];
```

Dentro del cuerpo de la función existirá la sentencia **RETURN** que devolverá el valor de la variable deseada, ésta será del mismo tipo que el asignado en la cláusula : **RETURN tipo IS ...** . Su sintaxis es: **RETURN** *variable*;

Puede haber varias instrucciones RETURN, pero sólo se ejecutará la primera que se encuentre dentro de la lógica del programa.

Para borrar una función:

```
DROP FUNCTION nombre_función;
```



Procedimientos y Funciones Almacenados

CREACIÓN DE PROCEDIMIENTOS Y FUNCIONES LOCALES

DECLARE

```
PROCEDURE EjemploP (  
    Parametro1  NUMBER DEFAULT 10,  
    Parametro2  VARCHAR2 DEFAULT 'ABCD',  
    Parametro3  DATE DEFAULT SYSDATE) AS  
BEGIN  
    DBMS_OUTPUT.PUT_LINE(Parametro1 || ' ' || Parametro2 || ' ' || Parametro3);  
END EjemploP;
```

```
FUNCTION EjemploF (Nombre VARCHAR2, Apellidos VARCHAR2)  
RETURN VARCHAR2 IS  
BEGIN  
    RETURN Nombre || ' ' || Apellidos;  
END EjemploF;
```

BEGIN

```
EjemploP();  
DBMS_OUTPUT.PUT_LINE('Mi nombre es: ' || EjemploF('Belén','Vela Sánchez'));  
END;  
/
```



Procedimientos y Funciones Almacenados

CREACIÓN DE PROCEDIMIENTOS Y FUNCIONES LOCALES

```
SQL> BEGIN  
  2 EjemploP();  
  3 end;  
  4 /
```

EjemploP();

ERROR en línea 2:

ORA-06550: línea 2, columna 1:

PLS-00201: el identificador 'EJEMPLOP' se debe declarar

ORA-06550: línea 2, columna 1:

PL/SQL: Statement ignored



Índice

1. *Introducción al PL/SQL*
2. *Conceptos Básicos de PL/SQL*
 - *Estructura de Bloques*
 - *Reglas y Convenciones del Lenguaje*
 - *Entrada y Salida de Datos*
3. *Variables de PL/SQL*
4. *Estructuras de Control*
5. *Cursores*
6. *Procedimientos y Funciones Almacenados*
- ⇒ **7. Paquetes**
8. *Tratamiento de los Errores y Excepciones*
9. *Disparadores*
 - *Bases de Datos Activas*
 - *Disparadores en ORACLE*





Paquetes

- Un **paquete** es un **conjunto de bloques de PL/SQL** que se encuentran guardados en un mismo objeto.
- Un paquete es, en esencia, una **sección declarativa con nombre**.
- Cualquier cosa que pueda incluirse en la sección declarativa de un bloque, puede incluirse también en un paquete. Esto abarca ***procedimientos, funciones, cursores, tipos y variables***.
- Una ventaja de incluir estos objetos en un paquete es la posibilidad de referenciarlos desde otros bloques PL/SQL, con lo que los paquetes permiten disponer de **variables globales** en PL/SQL.
- Cada paquete está formado por dos partes, la **especificación** y el **cuerpo**.



ESPECIFICACIÓN DE UN PAQUETE

También denominada **cabecera**, contiene información acerca del contenido del paquete. Sin embargo, **no contiene el código** de los procedimientos o funciones.

La sintaxis general para la creación de una cabecera de paquete es:

```
CREATE [OR REPLACE] PACKAGE nombre_paquete {IS | AS}  
    Especificación_procedimiento /  
    Especificación_función /  
    Declaración_variable /  
    Definición_tipo /  
    Declaración_excepción /  
    Declaración_cursor  
END [nombre_paquete];
```



ESPECIFICACIÓN DE UN PAQUETE

Las reglas sintácticas para una **cabecera de paquete** son las mismas que para una sección declarativa, excepto en lo que se refiere a la declaración de procedimientos y funciones.

- Estas reglas son:
 - Los elementos del paquete pueden aparecer en cualquier orden. Sin embargo y al igual que sucede en una sección declarativa, un objeto debe ser declarado antes de poderlo referenciar. Si un cursor contiene una variable como parte de la cláusula WHERE, por ejemplo, la variable debe ser declarada antes que el cursor.
 - No es necesario que estén presentes todos los tipos de elementos. Un paquete puede contener solo especificaciones de procedimientos y funciones, por ejemplo, sin declarar ninguna excepción de ningún tipo.
 - Las declaraciones de procedimientos y funciones deben ser declaraciones formales, a diferencia de la sección declarativa de un bloque, que puede contener tanto declaraciones formales como el código de los procedimientos y funciones. El código que implementa los procedimientos y funciones del paquete se encuentra en el cuerpo de este.



CUERPO DE UN PAQUETE

- Es un objeto del diccionario de datos distinto de la cabecera. El cuerpo no puede ser compilado a menos que se haya previamente compilado la cabecera correspondiente. El cuerpo contiene el código para las declaraciones formales de subprogramas incluidas en la cabecera.
- El cuerpo del paquete es opcional. Si la cabecera del paquete no contiene ningún procedimiento o función (solo declaraciones de variables, cursores, tipos, etc.), entonces no es necesario que el cuerpo esté presente. Esta técnica resulta útil para declarar **variables globales**, dado que todos los objetos de un paquete son visibles fuera de él.
- Cualquier declaración formal de la cabecera del paquete debe ser implementada en el cuerpo del paquete. La especificación del procedimiento o función debe ser la misma en ambos sitios, incluyendo el nombre del subprograma, los nombres de sus parámetros y los modos de éstos.



CUERPO DE UN PAQUETE

La sintaxis para la creación de un cuerpo de paquete es:

```
CREATE [OR REPLACE] PACKAGE BODY nombre_paquete  
{IS | AS}
```

```
    Implementación_procedimiento /  
    Implementación_función /
```

```
END [nombre_paquete];
```



ÁMBITO PARA LOS PAQUETES

- Cualquier objeto declarado en la cabecera de un paquete está dentro de ámbito y es visible fuera del paquete, sin más que cualificar el objeto con el nombre del paquete.
- La llamada al procedimiento es igual que si fuera un procedimiento independiente. La única diferencia es que hay que incluir como prefijo el nombre del paquete.
- Los procedimientos empaquetados pueden tener parámetros predeterminados, y pueden ser llamados utilizando notación posicional o nominal (indicando el valor para cada parámetro *parametro1=>Variable1*), al igual que los procedimientos almacenados independientes.
- Dentro del cuerpo del paquete se puede hacer referencia a los objetos definidos en la cabecera sin necesidad de utilizar el nombre del paquete.



SOBRECARGA DE LOS SUBPROGRAMAS DE UN PAQUETE

Dentro de un paquete pueden **sobrecargarse** los procedimientos y funciones, es decir, puede haber más de un procedimiento o función con el mismo nombre, pero con distintos parámetros.

Esta característica es muy útil, dado que permite aplicar la misma operación a objetos de tipos diferentes.



Paquetes

SOBRECARGA DE LOS SUBPROGRAMAS DE UN PAQUETE

Ejemplo:

Supongamos que queremos añadir un estudiante a una clase, bien especificando el **número de identificación** del estudiante, bien especificando el **nombre y apellidos** :

CREATE OR REPLACE PACKAGE PaqueteCurso AS

```
PROCEDURE EstudianteNuevo (P_ID_Estudiante IN Estudiantes.ID%TYPE,  
    P_Departamento IN CLASES.DEPARTAMENTO%TYPE,  
    P_Curso IN CLASES.Curso%TYPE);
```

```
PROCEDURE EstudianteNuevo (P_Nombre IN Estudiantes.Nombre%TYPE,  
    P_Apellidos IN Estudiantes.Apellidos%TYPE,  
    P_Departamento IN CLASES.DEPARTAMENTO%TYPE,  
    P_Curso IN CLASES.Curso%TYPE);
```

```
END PaqueteCurso;
```



Paquetes

CREATE OR REPLACE PACKAGE PaqueteCurso **BODY AS**

```
PROCEDURE EstudianteNuevo (P_ID_Estudiante IN Estudiantes.ID%TYPE,  
    P_Departamento IN Clases.Departamento%TYPE,  
    P_Curso IN Clases.Curso%TYPE) IS  
BEGIN  
    INSERT INTO Estudiantes_Matriculados (Id_Estudiante, Departamento, Curso)  
        VALUES ( P_ID_Estudiante, P_Departamento, P_Curso);  
    COMMIT;  
END;
```



Paquetes

```
PROCEDURE EstudianteNuevo (P_Nombre IN Estudiantes.Nombre% TYPE,  
    P_Apellidos IN Estudiantes.Apellidos%TYPE,  
    P_Departamento IN Clases.Departamento%TYPE,  
    P_Curso IN Clases.Curso%TYPE) IS  
DECLARE  
    V_ID_Estudiante Estudiantes.ID%TYPE;  
BEGIN  
    SELECT ID INTO V_ID_Estudiante  
    FROM Estudiantes  
    WHERE Nombre = P_Nombre  
    AND Apellidos = P_Apellidos;  
    INSERT INTO Estudiantes_Matriculados(Id_Estudiante, Departamento, Curso)  
    VALUES ( V_ID_Estudiante, P_Departamento, P_Curso);  
    COMMIT;  
END EstudianteNuevo;  
  
END PaqueteCurso;
```



Paquetes

Con esto podremos añadir un estudiante a la clase de dos formas:

Primera opción:

```
BEGIN
```

```
PaqueteCurso.EstudianteNuevo (10000, 'ORA',150);
```

```
END;
```

Segunda opción:

```
BEGIN
```

```
PaqueteCurso.EstudianteNuevo ('JAVIER', 'LOPEZ' , 'ORA' ,150);
```

```
END;
```

La sobrecarga puede ser muy útil cuando se pueda hacer la misma operación con argumentos de tipos diferentes. Sin embargo, la sobrecarga está sujeta a diversas restricciones.

- **No** se puede sobrecargar dos subprogramas si sus parámetros **sólo difieren en el nombre o en el modo**.
- **No** pueden sobrecargarse dos funciones basándose sólo en su **tipo de retorno**.
- Finalmente, los parámetros de las funciones sobrecargadas deben diferir también en cuanto a **familia de tipos**, no pudiendo realizarse sobrecargas dentro de la misma familia. Por ejemplo, CHAR y VARCHAR2, que pertenecen a la misma familia de tipos.



Índice

1. *Introducción al PL/SQL*
2. *Conceptos Básicos de PL/SQL*
 - *Estructura de Bloques*
 - *Reglas y Convenciones del Lenguaje*
 - *Entrada y Salida de Datos*
3. *Variables de PL/SQL*
4. *Estructuras de Control*
5. *Cursores*
6. *Procedimientos y Funciones Almacenados*
7. *Paquetes*
- ⇒ **8. Tratamiento de los Errores y Excepciones**
9. *Disparadores*
 - *Bases de Datos Activas*
 - *Disparadores en ORACLE*





Tratamiento de los Errores y Excepciones

PL/SQL implementa los mecanismos de tratamiento de errores mediante **excepciones**.

Objetivo: tratar los **errores** producidos en tiempo de **ejecución** y no en tiempo de compilación. Las excepciones y los gestores de excepciones son el método a través del cual el programa reacciona a los errores de ejecución y realiza su tratamiento.

Los errores que se producen en la fase de compilación son detectados por el motor PL/SQL y comunicados al usuario.

Cuando se produce un **error**, se genera **una excepción**. Cuando esto sucede, el control pasa al **gestor de excepciones**, que es una sección independiente del programa.

Esto permite **separar** la gestión de errores del resto del programa, lo que hace que sea más fácil entender la lógica de éste, y también asegura que todos los errores serán interceptados.



Tratamiento de los Errores y Excepciones

DECLARACIÓN DE EXCEPCIONES

Las excepciones:

- son declaradas en la sección de declaración,
- son lanzadas en la sección de ejecución,
- y resueltas en la sección de excepciones.

Existen dos tipos de excepciones: las definidas por los usuarios y las predefinidas.

Excepciones definidas por el usuario:

Es un error cuya definición se realiza en el programa. El error puede ser un **error Oracle** o, p.e., un **error** relativo a los **datos**.

Las excepciones definidas por el usuario se declaran en la sección declarativa de un bloque PL/SQL. Al igual que las variables, las excepciones tienen un tipo asociado (*EXCEPTION*) y un ámbito.

La sintaxis es:

Nombre_excepcion EXCEPTION;



Tratamiento de los Errores y Excepciones

MANEJO DE LOS ERRORES:

Una vez que se haya lanzado una excepción, el manejador de errores pasa a la sección EXCEPTION. Aquí se utiliza la siguiente estructura:

EXCEPTION

WHEN *nombre_excepción1* **THEN**

Conjunto de sentencias 1

WHEN *nombre_excepción2* **THEN**

Conjunto de sentencias 2

...

WHEN OTHERS THEN

Conjunto de sentencias N

END;

Cada excepción tendrá un **WHEN ... THEN** y un conjunto de sentencias que intentarán subsanar el error acaecido.



Tratamiento de los Errores y Excepciones

Generación de excepciones definidas por el usuario:

```
DECLARE
    A EXCEPTION; -- Declaración de la excepción A

BEGIN
    ...
    RAISE A;      --Generación de la excepción A
    ...          -- Este código no se ejecuta al generarse la excepción

EXCEPTION      -- El control pasa al gestor de excepciones;
    WHEN A THEN
        ...
    WHEN OTHERS THEN
        ...

END;
```



Tratamiento de los Errores y Excepciones

Además podemos añadir dos o más excepciones a un mismo conjunto de sentencias.

```
...  
WHEN EXCEPTION OR EXCEPTION THEN  
    Conjunto de sentencias  
  
...  
WHEN OTHERS THEN  
  
...
```

En la parte **WHEN OTHERS THEN** contiene las sentencias que se ejecutarán al ocurrir un error que no tiene excepción declarada.

Debe ser siempre el último gestor de un bloque.

Para saber el error que provocó la excepción dentro del gestor de excepciones OTHERS podemos usar las funciones **SQLCODE** y **SQLERRM** para obtener el código del error y el mensaje asociado.



Tratamiento de los Errores y Excepciones

Funciones SQLCODE Y SQLERRM.- En el manejador de las demás excepciones, a veces es útil conocer el error que lanzó la excepción. Para ello usamos **sqlcode** que devuelve el código de error que ha ocurrido, y usamos **sqlerrm** que muestra el mensaje de error.

...

WHEN OTHERS THEN

v_codigo_Error:=SQLCODE;

v_texto_Error:=SUBSTR(SQLERRM,1,200);

INSERT INTO Tabla_Log

VALUES (v_codigo_Error, v_texto_Error);

...

Sólo se asignan los 200 primeros caracteres a la variable previamente definida *v_texto_Error*

En el INSERT **no** podemos meter directamente las llamadas a funciones. Se usan variables (previamente declaradas).



Tratamiento de los Errores y Excepciones

Excepciones predefinidas de Oracle:

- **ORA – 0001 DUP_VAL_ON_INDEX** Violación de una restricción de unicidad
- **ORA – 0051 TIMEOUT_ON_RESOURCE** Se produjo un fin de intervalo mientras se esperaba un determinado recurso
- **ORA – 0061 TRANSACTION_BACKED_OUT** La transacción fue cancelada debido a un bloqueo
- **ORA – 1001 INVALID_CURSOR** Operación ilegal con un cursor
- **ORA – 1012 NOT_LOGGED_ON** No existe conexión con Oracle
- **ORA – 1017 LOGIN_DENIED** Nombre de usuario o contraseña inválidos
- **ORA – 1403 NO_DATA_FOUND** No se ha encontrado ningún dato
- **ORA – 1422 TOO_MANY_ROWS** Hay más de una línea que corresponde a una orden SELECT... INTO
- **ORA – 1476 ZERO_DIVIDE** División por cero
- **ORA – 1722 INVALID_NUMBER** Falló la conversión a un número
- **ORA – 6500 STORAGE_ERROR** Error interno PL/SQL, generado cuando PL/SQL se queda sin memoria



Tratamiento de los Errores y Excepciones

Excepciones predefinidas de Oracle:

- **ORA – 6501 PROGRAM_ERROR** Error interno PL/SQL
- **ORA – 6502 VALUE_ERROR** Error de truncamiento, aritmético o de conversión
- **ORA – 6504 ROWTYPE_MISMATCH** Una variable de cursor del HOST y una variable del cursor PL/SQL tienen tipo de fila incompatibles
- **ORA – 6511 CURSOR_ALREADY_OPEN** Se ha intentado abrir un cursor que ya estaba abierto
- **ORA – 6530 ACCESS_INTO_NULL** Se ha intentado asignar valores a los atributos de un objeto que tiene el valor NULL
- **ORA – 6531 COLLECTION_IS_NULL** Se ha intentado aplicar métodos de colección distintos de EXISTS a una tabla o array PL/SQL con valor NULL
- **ORA – 6532 SUBSCRIPT_OUTSIDE_LIMIT** Una referencia a una tabla anidada o índice de array se encuentra fuera del rango declarado
- **ORA – 6533 SUBSCRIPT_BEYOND_COUNT** Una referencia a una tabla animada o índice de array es mayor que el número de elementos de la colección.



Tratamiento de los Errores y Excepciones

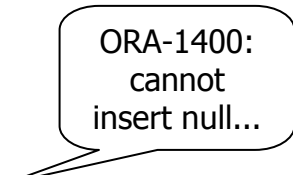
EXCEPTION_INIT.- Se puede asociar nombres de excepciones declaradas por el usuario con errores particulares de ORACLE, lo que permite parchear las excepciones y manejarlas por nuestra cuenta.

La sintaxis es:

PRAGMA EXCEPTION_INIT (*excepción, número de error*);

Ejemplo:

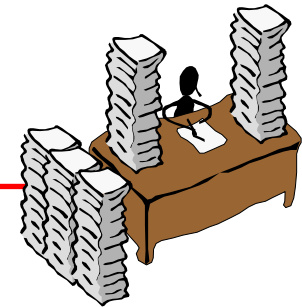
```
DECLARE
  E_valor_nulo EXCEPTION;
  PRAGMA EXCEPTION_INIT (E_valor_nulo, -1400);
BEGIN
  INSERT INTO Empleados (ID_Emp) VALUES (NULL);
EXCEPTION
  WHEN E_valor_nulo THEN
    INSERT INTO Tabla_Log (info) VALUES ('Error de ORACLE ORA-1400');
END;
```



ORA-1400:
cannot
insert null...



Bibliografía



Bibliografía Básica

- Abbey, Corey y Abramson. ***ORACLE8i: Guía de Aprendizaje (versiones 7.x, 8 y 8i)***. McGraw-Hill.
- Urman S. ***ORACLE 8: Programación PL/SQL***. McGraw-Hill.



- Urman S., Hardman, R., y McLaughlin, M. ***ORACLE DATABASE 10g. PL/SQL Programming***. Oracle Press, 2004.



Bibliografía Complementaria

- Elmasri, R. y Navathe, S. B. ***Fundamentos de Sistemas de Bases de Datos***. Tercera Edición. Addison-Wesley Iberoamericana, 2000.