

MP0484.
Bases de datos
UF6. Programación básica de acceso a datos

6.1. Elementos del lenguaje PL/SQL

Índice

☰	Objetivos	3
☰	Ventajas de PL/SQL	4
☰	Bloques Anónimos	6
☰	Conceptos del Lenguaje	9
☰	Sentencias SQL permitidas	15
☰	Declaración de variables: Tipos de datos	19
☰	Alternativas	25
☰	Repetitivas	29
☰	Trabajar con varios bloques	33
☰	Un ejemplo resuelto	39

Objetivos

Incrustado dentro del motor de Oracle, nos permite acceder a las bases de datos a través de un lenguaje de programación estructurado y potente. Solo necesitas tener conocimientos de manipulación de datos (DML), y tener conocimientos de programación.

Objetivos:

- Conocer cómo se estructura un programa PL/SQL
- Definir las variables de un programa
- Saber que tipo de sentencias SQL se puede ejecutar en PL
- Ver las estructuras de control de PL: alternativas y repetitivas
- Cómo se trabaja con varios bloques PL.
- Estudiar el ámbito de las variables en PL.

Ventajas de PL/SQL

Con PL / SQL se pueden utilizar sentencias SQL para manipular datos y estructuras de control para poder interactuar con ellos. Se pueden declarar variables y constantes, definir procedimientos y funciones. De esta manera se puede aunar todo el poder de SQL y la capacidad procedural de un lenguaje.

PL / SQL es un lenguaje estructurado en bloques. Cada bloque puede contener a su vez otros sub-bloques y así sucesivamente. Cada parte de un bloque o sub-bloque resuelve, normalmente, un problema o un sub-problema.

Un bloque o sub-bloque relaciona lógicamente declaraciones y comandos. Las declaraciones son propias de ese bloque y dejan de existir cuando el bloque finaliza.

Existen dos tipos de Bloques: Bloque anónimo y Bloque Almacenado.

Bloque anónimo

Un bloque que no se guarda en la base de datos y se analizará cada vez que sea invocado.

El bloque queda en el Buffer de memoria de la aplicación en donde se esté ejecutando el programa, y por tanto se pierde al cerrar la sesión de trabajo.

Si queremos recuperarlo, lo almacenamos en un fichero de tipo texto, en nuestro sistema operativo. Pero no es un Componente Oracle, y por tanto no sabe que existe.

Cada vez que queremos ejecutarlo lo haremos de dos formas:

- Copiamos el texto del bloque y lo pegamos en la consola, para ejecutarlo directamente.
- Lo ejecutamos como un script, dando el nombre y la dirección donde se encuentra el fichero que almacena el código.

Por ejemplo en SQL*PLUS el comando es:

```
start "direccion\nombrecompleto"
-- o también
@ "direccion\nombrecompleto"
```

Se construyen bloques anónimos para realizar scripts de visualización de datos, para procesar actividades que se van a ejecutar una sola vez, para probar bloques almacenados.

Bloques Almacenados

Un código PL / SQL se dice que está almacenado, porque al compilarle, el motor de Oracle lo guarda como un componente más de la Base de Datos, y por tanto se puede solicitar su ejecución desde el otro PL / SQL, o desde otros entornos, siempre y cuando se haya establecido la conexión a nuestra base de datos, tenga los permisos oportunos y este compilado correctamente.

Por tanto, como componente, Oracle, el mandato, DDL, empleado para su cualificación, es CREAR.

Los bloques almacenados que se pueden crear son:

- Procedimientos(PROCEDURE)
- Funciones(FUNCTION)
- Paquetes(PAKAGE)
- Disparadores(TRIGGER)

Su estudio lo veremos en la siguiente Unidad Formativa.

Bloques Anónimos

De momento trabajaremos con programas de un solo bloque. Al final de este capítulo veremos como estructurar un programa en varios bloques.

La estructura de un bloque es la siguiente:

```
DECLARE
```

```
BEGIN
```

```
EXCEPTION
```

```
END;
```

```
/
```

DECLARE

Esta sección se usa para declarar cada una de las variables que sean referenciadas en la sección de instrucciones(BEGIN).

Si no hay variables, su codificación no es obligatoria

BEGIN

Es la sección de instrucciones de control, sentencias de asignación y operación, y sentencias SQL/DML, embebidas en el programa. Una sentencia DML, se considera una instrucción en secuencia dentro del bloque.

Su codificación es obligatoria.

EXCEPTION

Las excepciones en PL/SQL, se levantan, y se capturan en esta sección. Hay dos tipos de excepciones las que genera Oracle al ejecutar determinadas sentencias, y las que levanta el programador, buscando un fin alternativo al bloque.

Esta sección no es obligatorio codificarla, pero es muy recomendable su uso.

END;

Marca el final de un bloque, su codificación es obligatoria, y no hay que olvidarse del punto y coma.

Muy recomendable teclear la barra "/", debajo de esta sección, obligatorio si el programa se ejecuta con SQL*PLUS

Los bloques anónimos son interpretados por el motor de la base de datos, de forma semejante a un script de sentencias SQL.

El bloque más pequeño que se puede ejecutar sin errores, y por tanto con resultado satisfactorio (eso sí no genera nada), en PL/SQL, es:

```
BEGIN
    NULL;
END;
/
```

La sentencia NULL no realiza ninguna acción y pasa el control a la siguiente sentencia. Se utiliza en zonas en donde es necesario escribir código, pero no se quiere realizar ninguna acción.

Nuestro primer programa, emulando a otros lenguajes de programación, escrito en SQL*PLUS, que es el programa interactivo de consola de Oracle:

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
 2 DBMS_OUTPUT.PUT_LINE('Hola soy tu profe de PL/SQL');
 3 END;
 4 /
Hola soy tu profe de PL/SQL
```

```
PL/SQL procedure successfully completed.
```

```
SQL>
```

Como en otros lenguajes tenemos que explicar estas dos sentencias.

- DBMS_OUTPUT.PUT_LINE: es la instrucción(procedimiento PUT_LINE, dentro del paquete DBMS_OUTPUT) para mostrar literales y/o variables en la consola del ordenador. Los literales alfanuméricos se encierran entre comillas sencillas.
- En Oracle toda la funcionalidad está almacenada en paquetes: DBMS_OUTPUT, DBMS_FILE, STANDARD..... Y en cada paquete existen procedimientos y funciones para tareas específicas.
- SET SERVEROUTPUT ON: es una directiva para que lo que se manda a la consola a través del procedimiento PUT_LINE aparezca.

Esta es la salida, cuando en la sesión no se ha especificado la directiva set serveroutput on. Fíjate que no aparece el literal codificado en el procedimiento PUT_LINE('Hola....');

```
BEGIN
 2 DBMS_OUTPUT.PUT_LINE('Hola soy tu profe PL/SQL');
 3 END;
 4 /
```

PL/SQL procedure successfully completed.

SQL>

Conceptos del Lenguaje

Como en cualquier lenguaje de programación PL/SQL tiene también reglas de sintaxis, palabras reservadas, tipos de datos propios, puntuación, etc.

1

Conjuntos de caracteres.

Utilizar:

- Letras del alfabeto inglés.
- Números.
- Los símbolos ()+-*;/<>=!~^;%,"#\$&_{}?[], espacio, tabulación y retorno de carro.

PL/SQL no distingue entre mayúsculas y minúsculas, excepto cadenas y caracteres literales (texto entre comillas).

2

Delimitadores.

Los delimitadores son símbolos simples o compuestos que tienen un significado para PL/SQL. Por ejemplo se utilizan delimitadores para expresar una operación aritmética o lógica. El espacio se utiliza para dar una mayor legibilidad al programa PL/SQL.

Una lista de los delimitadores más comunes:

+	Operador de suma.
/	Operador de división.
*	Operador de multiplicación.
-	Operador de resta / negación.
=	Operador de relación (Igualdad).
<	Operador de relación (Menor que).
>	Operador de relación (Mayor que).
;	Delimitador de final de sentencia.
'liter al'	Identificador de literal.
:	Identificador de Host Variable.
:=	Operador de asignación.
	Operador de concatenación.
**	Operador de potencia. (X elevado a Y).
/*	Delimitador de inicio de comentario de varias líneas.
*/	Delimitador de fin de comentario de varias líneas.
--	Delimitador de comentario de una sola línea.
..	Operador de rango.

3

Identificadores.

Los identificadores dan nombre a las variables, constantes, cursores, variables de cursor, subprogramas, paquetes y excepciones.

Los identificadores deben comenzar por una letra y puede ser seguida por números, letras, signos dólar (\$), guiones bajos (_) y el signo numérico (#). Cualquier otro signo provocará un error.

Se pueden utilizar indistintamente las mayúsculas y las minúsculas. Los signos dólar, guión bajo y signo numérico cuentan a la hora de diferenciar un identificador.

La longitud máxima de un identificador es de 30 caracteres. Es aconsejable que los identificadores tengan significado para una mayor legibilidad en los programas.

X	Correcto.
Total\$	Correcto.
Dep_Trabajo	Correcto.
1980Total	Incorrecto por empezar con un número.
Nombre-Jefe	Incorrecto por llevar un guión.
Si/No	Incorrecto por la barra.
IdPedido	Correcto.
idpedido	Correcto pero es igual que IdPedido.
IDPEDIDO	Correcto pero es igual a los 2 anteriores.
ID_PEDIDO	Correcto y diferente a los 3 anteriores.

Existe también un conjunto de palabras llamadas palabras reservadas que no pueden ser utilizadas, como las palabras BEGIN y END, que forman parte del bloque o subprograma y son reservadas. Muchas palabras en PL/SQL, propias del lenguaje, no se consideran reservadas, pero se desaconseja su uso para evitar errores de compilación.

4

Literales.

Un literal es un número, una cadena, un carácter o un valor Booleano que no está representado por un identificador. Es un valor constante.

Literales numéricos

Literales numéricos: Representan números y pueden ser de dos tipos:

- Enteros. Pueden tener signo y son representados sin punto decimal.
- Reales. Pueden tener signo y se representan con un punto decimal.

El único carácter que se puede utilizar en los literales es la E, la cual significa “potencia de”.

El rango de valores de los números es de 1E-130 hasta 10E125.

12	Entero
-34	Entero
67.7	Real
-927.	Real
1e28	Real
1893e130	Incorrecto por salir del rango admitido

Literales de carácter y cadena de caracteres.

Son conjuntos de 1 o más caracteres que están entre comillas simples (' '). Pueden contener todos los caracteres de PL/SQL y sí que son sensibles a las mayúsculas y minúsculas.

'S'	
'El pedido ha sido procesado'	
'Introduzca S/N'	
'Entrada'	
'ENTRADA'	--Este literal es diferente al superior

Literales Booleanos

Existen tres valores, no cadenas, para los literales booleanos.

TRUE
FALSE
NULL

Literales de tipo fecha

Los literales de tipo fecha dependen de la base de datos, pero siempre van encerrados entre comillas simples.

```
Dia      V_DIA := DATE '18-04-2000';
Fecha2  V_STAMP := TIMESTAMP '2002-02-20 15:01:01';
Fecha3  V_TIME_ZONE := TIMESTAMP '2002-01-31 19:26:56.66 +02:00';
Fecha4  V_INTERVAL_YM := INTERVAL '3-2' YEAR TO MONTH;
```

5

Comentarios.

Los comentarios son necesarios para poder documentar un programa PL/SQL. Permiten clarificar partes de código para una posible revisión posterior del código. PL/SQL ofrece 2 tipos de comentarios.

- Comentarios de línea. Para comentar una línea o parte de una línea se utiliza dos guiones (--).
- Comentarios multilínea o de varias líneas. Para iniciar el bloque de comentarios PL/SQL utiliza /*) y para acabar el bloque de líneas comentadas (*).

6

Normas de nombrado.

Las siguientes convenciones se aplican a todos los objetos PL/SQL. Es decir, a las variables, constantes, nombres de cursores, programas y subprogramas. Los nombres pueden ser de 4 tipos:

- Simples. Se invoca al objeto por su nombre.
- Cualificado. Se antepone el nombre, del paquete o estructura que contiene este objeto, separado por un punto([registro.dato](#)).

- Remoto. Se pospone la base de datos remote detrás del objeto nombrado. Se accede a la base de datos remota mediante un database link y es donde se encuentra el objeto nombrado(nombre@bbddremota).
- Cualificado y remota. Es la suma de las 2 anteriores. Se referencia el paquete, el procedimiento y, mediante el indicador de acceso remoto, la base de datos remota(registr.datos@bbddremota).

7

Expresiones y Comparaciones.

PL/SQL evalúa el resultado de una operación examinando la expresión y el contexto de ésta. En las expresiones u operaciones pueden intervenir variables, constantes, literales y funciones.

En la siguiente figura se muestra el orden de preferencia (de mayor a menor) de los diversos operadores que pueden aparecer en una expresión u operación.

Operador	Operación
**	Exponenciación (A elevado a B)
+ , -	Identidad, negación
* , /	multiplicación, división
+, -,	Suma, resta, concatenación
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparación
NOT	Negación Lógica
AND	Conjunción (A y B)
OR	Inclusión (A o B)

El orden de las operaciones se puede alterar mediante los paréntesis. Se recomienda el uso de los paréntesis; para que de una manera explícita, quede claro lo que se quiere.



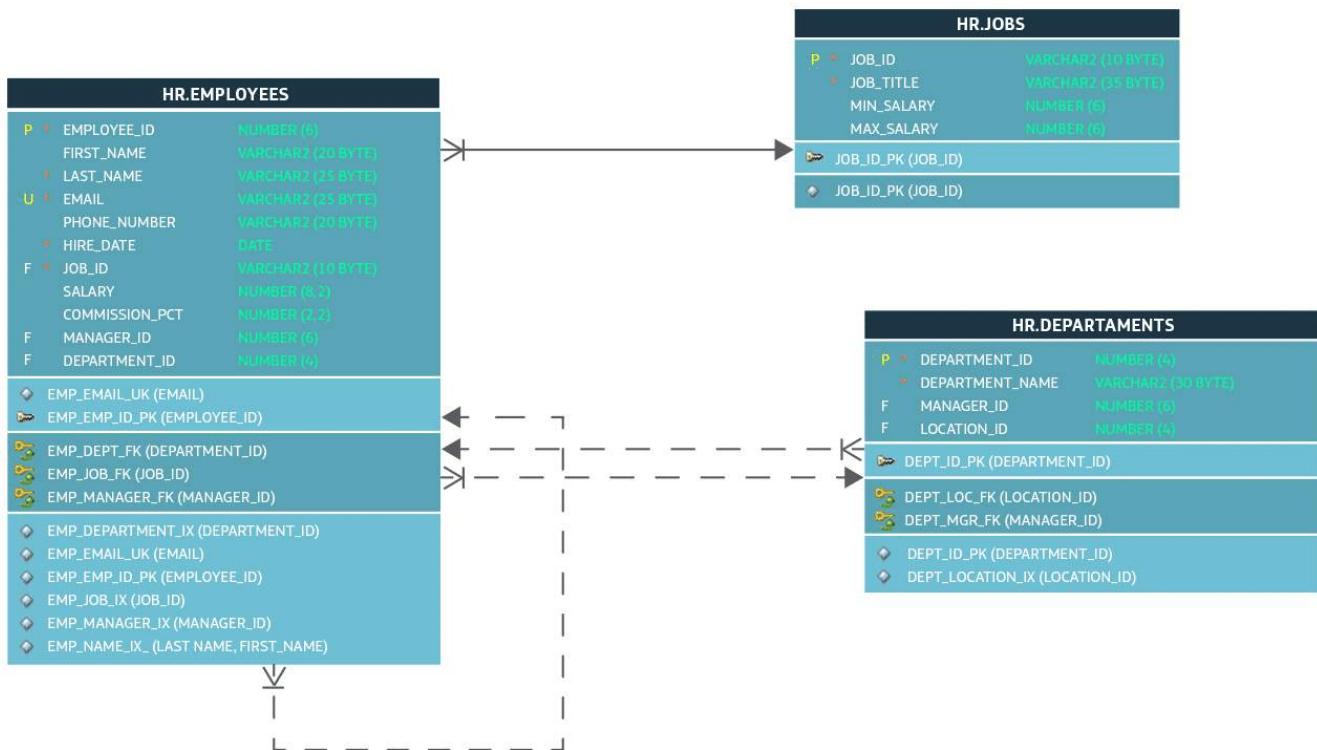
Como vemos en el cuadro anterior, a los operadores normales de comparación (=,<,<=,>,>=,<>), PL/SQL incorpora, al estar metido en el motor de Oracle, los operadores especiales IN, BETWEEN y LIKE(y sus correspondientes negaciones NOT IN, NOT BETWEEN y NOT LIKE), pero asociado a variables, y con el mismo tipo de funcionamiento que en SQL.

Además en PLSQL, puedo preguntar si una variable tiene activado el nulo : IS NULL, IS NOT NULL.

Sentencias SQL permitidas

PL/SQL no es un lenguaje para cálculos científicos, es un lenguaje para gestionar la Base de Datos y trabajar con las tablas de nuestros esquemas. Allá donde una sentencia SQL no genera lo requerido, PL es necesario.

Un bloque PL tanto anónimo como almacenado, las sentencias que puede manejar son las de DML: select, insert, update y delete. Las sentencias de tipo DDL, directamente no se pueden especificar en un bloque, necesitan funciones especiales aportadas por Oracle.



Tablas, columnas y sus tipos, PK, FK's y relaciones

Tablas de ejemplos

El esquema de tablas que vamos a seguir en los ejemplos de PL son las tablas correspondientes al usuario HR que viene incorporado en la versión 11gR2 del Oracle Express Edition.

Select

PL admite como instrucción cualquier select siempre y cuando la sentencia devuelva **una sola fila**.

La pregunta es, ¿qué select estoy absolutamente seguro que devuelve una sola fila?:

- La que accede a la columna (o columnas) que forma la PRIMARY KEY por igualdad.
- La que accede a la columna (o columnas) con la restricción UNIQUE por igualdad.
- La que tiene en la select funciones de agrupamiento, sin group by.

```
set serveroutput on
declare

begin
    select * from jobs
    into fila_jobs
    from jobs
    where job_id = 'IT_PROG';

    select last_name, salary, hire_date
    into v_apel, v_salario, v_fecha
    from employees
    where employee_id = 107;

    select count(*), count (distinct(jobs_id)), count(department_id),
           max(salary), min(salary), avg(salary)
    into v_count, v_count_dist, v_count_dep,
         v_max_sal, v_min_sal, v_avg_sal
    from employees
    where department_id in (10,20,30,90);

end;
/
```

Ejemplos de sentencias select en PL

La única variación en la sentencia es la cláusula INTO, en donde se especifican las variables de mi programa, se recogen cada una de las expresiones de la cláusula select, y que deben estar declaradas en la sentencia DECLARE.

Cada sentencia SQL termina en punto y coma(;), y se comporta como una instrucción mas de PL, y por tanto, se codifican en el BEGIN.

En le siguiente apartado veremos cómo se declaran estas variables.

Insert, Update y Delete

Se codifican tal cual se conocen de SQL interactivo, la diferencia es que en la cláusula values de insert, set y where de update, y where de delete, puedo usar además de literales, variables de mi programa, previamente declaradas en la DECLARE.

```
set serveroutput on
declare

begin
    insert into departments
    (department_id, department_name)
    values(v_dep,v_dname);

    insert into jobs
    (job_id, job_title,min_salary,max_salary)
    values fila_jobs;

    update employees
        set commission_pct = v_comision
        where job_id = 'ST_PRU';

    delete employees
        where job_id = v_jobid;
end;
/
```

v_ y fila_ son variables de mi programa

Cada sentencia representa una instrucción y por tanto termina en punto y coma(;).



Cada sentencia SQL en PL, levantan excepciones. Las excepciones las trataremos en la siguiente lección, por tanto todos los ejemplos de este capítulo, están codificados de forma que se ejecuten de forma normal, sin levantar excepciones.

Declaración de variables: Tipos de datos

Todas las variables que me hacen falta para mi bloque anónimo, las tengo que poner en la sección DECLARE, separadas por ":";



PL usa para definir el tipo de dato de las variables y constantes, todos los tipos de datos de SQL. E incorpora dos fundamentalmente: PLS_INTEGER, para variables enteras y BOOLEAN, cuyos valores posibles son TRUE, FALSE Y NULL.

1

Variables simples.

La zona de declaraciones permite reservar espacio para las variables y las constantes que se utilizarán en el programa PL/SQL.

Una variable en PL se declara especificando el nombre y el tipo, y si es preciso el valor inicial.

Las variables no inicializadas toman el valor NULL.

Las variables pueden ser de tipo NOT NULL, esto significa que la variable tiene que contener un valor y por lo tanto es obligatoria en este caso inicializarla cuando se declara la misma. Las variables de tipo NOT NULL, está desaconsejado su uso.

La definición de constantes se realiza mediante el uso de la cláusula CONSTANT. La inicialización de una constante se debe realizar en la zona de declaraciones puesto que si no se producirá un error de compilación. Este valor no se podrá cambiar mientras exista, se evita así que en el bloque o subprograma pueda ser modificada por error en el código del programa.

La manera de asignar un valor a una variable es utilizar la notación de (:=) (Dos puntos y el signo igual).

```
set serveroutput on

declare
    v_contador pls_integer := 0; -- variable inicializada
    v_nombre varchar2(30); -- variable sin inicializar
    v_empresa constant char(3) := 'TED'; -- constante
    v_apellido varchar2(30) not null := 'Sebastian'; -- variable not nu
    v_fecha date := '02-11-2017';
begin
    dbms_output.put_line('contador : ' || v_contador);
    dbms_output.put_line('nombre : ' || v_nombre);
    dbms_output.put_line('empresa : ' || v_empresa);
    dbms_output.put_line('apellido : ' || upper(v_apellido));
    dbms_output.put_line('fecha : ' || v_fecha);

end;
/
```

Definición de variables

```
contador : 0
nombre :
empresa : TED
apellido : SEBASTIAN
fecha : 02/11/17
```

```
PL/SQL procedure successfully completed.
```

Resultado de salida

Aclaraciones sobre el programa y su resultado.

- Cuando una variable no se inicializa, en la salida por consola no sale nada.
- Cuando una variable numérica no se inicializa, y se opera con ella, cualquier operador aritmético sobre ella, el resultado es NULL.
- Sobre las variables podemos aplicarles cualquier función escalar usada en SQL, para alterar el resultado (UPPER(v_apellido)).
- El formato de salida de una variable de tipo DATE, depende del formato de fecha y hora de la sesión de Oracle donde se ha ejecutado el bloque PL/SQL. Por eso, al igual que en SQL, se emplea con frecuencia la función escalar TO_CHAR.
- Se declara **una** variable en cada linea.

2

Variables sentencias DML.

Cuando en el programa aparecen sentencias DML, select, insert, update y delete, hemos visto que puedo referenciar variables en varias de sus cláusulas.

Normalmente de una columna de una tabla en concreto me se su nombre y como mucho el tipo de dato, lo que me es muy difícil es saber su precisión.

Por ejemplo, la columna LAST_NAME de la tabla Employees, se que es de tipo varachar2, pero lo que no se es si es de 25, 30, 100 caracteres.

Para la definición de las variables correspondientes a cada columna de una tabla referenciada en mandatos DML, PL nos proporciona la cláusula %TYPE.

v_apel employees.last_name%type.

La variable v_apel, es del mismo tipo que la columna last_name de la tabla employees.

Vemos como quedaría en el ejemplo :

```
set serveroutput on
declare
    v_apel employees.last_name%TYPE;
    v_salario employees.salary%type;
    v_fecha employees.hire_date%type;

begin
    select last_name, salary, hire_date
    into v_apel, v_salario, v_fecha
    from employees
    where employee_id = 107;

    dbms_output.put('apellido : ' || v_apel);
    dbms_output.put(' salario : ' || v_salario);
    dbms_output.put_line(' fecha ing : ' || to_char(v_fecha,'dd-mm-yy'));

end;
/
```

Las variables corresponden a las de la cláusula INTO de la select

```
apellido : Lorentz salario : 4200 fecha ing : 07-02-07
```

```
PL/SQL procedure successfully completed.
```

Resultado

3

Variables DML compuestas (Registros).

Fíjate en estas dos sentencias : select * e insert into jobs . Ambas hacen referencia a **todas las columnas**. Por tanto tendría que definir una a una las variables, como en el caso anterior.

Para estas situaciones PL nos proporciona %ROWTYPE: nombre_de_tabla%rowtype.

```
set serveroutput on
declare
    fila_jobs jobs%rowtype;

begin

    select *
        into fila_jobs
        from jobs
        where job_id = 'IT_PROG';
    dbms_output.put_line('datos del trabajo IT_PROG');
    dbms_output.put('trabajo : ' || fila_jobs.job_title);
    dbms_output.put_line('max salario : ' || fila_jobs.max_salary);

    fila_jobs.job_id := 'FOR_ORACLE';
    fila_jobs.job_title := 'Formador';
    fila_jobs.min_salary := 12000;
    fila_jobs.max_salary := 50000;

    insert into jobs
        values fila_jobs;
    dbms_output.put_line('fila insertada');
    commit;

end;
/

```

fila_jobs

datos del trabajo IT_PROG
trabajo : Programmer max salario : 10000
fila insertada

PL/SQL procedure successfully completed.

Resultado

- La variable **fila_jobs** es un registro o campo compuesto, formado por la concatenación de tantas variables como columnas tiene la tabla, con el mismo nombre de variable que de columna y con el tipo de dato (%type) correspondiente a la columna de la tabla.
- Como ves en el ejemplo anterior la forma de referenciar las variables es : **fila_jobs.job_title**, es decir **registro.variable**.
- En el caso del insert, primero alimento las variables del registro (una a una), y luego inserto la fila con el registro



!!! OJO!!! los values del insert van entre paréntesis, excepto cuando se especifica una variable de tipo registro: `insert into jobs values fila_jobs;`

Como ves se escribe sin paréntesis.

Alternativas

Permite tomar decisiones dependiendo de la información obtenida.

IF-THEN-ELSE

El comando IF-THEN-ELSE ejecutará una serie de comandos dependiendo de la información.

IF

evalúa la condición. Las condiciones **no** es necesario encerrarlas entre paréntesis

THEN

realiza la serie de comandos en caso que la condición sea cierta. Y es obligatorio codificarla.

ELSE

realiza la serie de comandos en caso que la condición ser falsa o nula.

ELSIF

permite evaluar las varias condiciones y en caso que se cumpla una ya no continuará evaluando el resto.

END IF;

Cierre de cada IF que codifiques en la sentencia, es obligatorio su uso.

Si en vez de usar ELSE, se emplea ELSIF, solo se codifica un END IF; al final del IF principal.

En el siguiente código se muestra las variantes que ofrece la sentencia IF:

```
IF condición THEN
    instrucciones;
[ELSE
    instrucciones;]
END IF;

-----
```

```
IF condición THEN
    instrucciones;

[ELSE
    IF condicion2 THEN
        instrucciones;
    [ELSE
        instrucciones;]
    END IF;
    instrucciones;
]

END IF;

-----
```

```
IF condición THEN
    instrucciones;
ELSIF condicion2 THEN
    instrucciones;
ELSIF condicion3 THEN
    instrucciones;
[ELSE
    instrucciones;]
END IF;
```

CASE

Para poder escoger entre varios valores y realizar diferentes acciones se puede utilizar la sentencia CASE.

La sentencia CASE permite crear bloques de sentencias como si se tratara de sentencias IF anidadas.

CASE selector

```
WHEN Expresión1 THEN secuencia_de_comandos1  
WHEN Expresión2 THEN secuencia_de_comandos2  
...  
WHEN ExpresiónN THEN secuencia_de_comandosN  
[ELSE secuencia_de_comandosN+1]  
END CASE;
```

CASE Nombrada

La cláusula ELSE realiza los comandos cuando ninguna de las cláusulas WHEN se ha cumplido. No es obligatoria pero si no se utiliza y no se ha cumplido ninguna cláusula WHEN se levantará la excepción CASE_NOT_FOUND.

A continuación se muestra un ejemplo de la sentencia **CASE nombrada**. Solicitamos a través de la variable de entorno **sysdate**, el día de la semana, y el algoritmo devuelve el literal correspondiente:

```
set serveroutput on  
declare  
    v_dia char(1);  
    literal varchar2(15);  
begin  
    v_dia:= to_char(sysdate,'d');  
  
    case v_dia  
        when '1' then literal:='lunes';  
        when '2' then literal:='martes';  
        when '3' then literal:='miercoles';  
        when '4' then literal:='jueves';  
        when '5' then literal:='viernes';  
        when '6' then literal:='sabado';  
        when '7' then literal:='domingo';  
    end case;  
  
    dbms_output.put_line('el dia es : ' || literal);  
end;  
/  
el dia es : martes
```

PL/SQL procedure successfully completed.

También está la posibilidad de utilizar los bloques CASE con condiciones, lo que se conoce como **CASE Buscadas**. En el caso de que se cumpla una condición el control pasará a la siguiente sentencia después del END CASE.

```
CASE
    WHEN Condición1 THEN
        secuencia_de_comandos1;
    WHEN Condición2 THEN
        secuencia_de_comandos2;
    ...
    WHEN CondiciónN THEN
        secuencia_de_comandosN;
    [ELSE secuencia_de_comandosN+1;]
END CASE;
```

CASE Buscada

El ejemplo siguiente es una CASE Buscada, almacenamos en la variable v_bono un valor en función del salario obtenido de un empleado.

```
DECLARE
    v_Salary          employees.salary%TYPE;
    v_Bono           PLS_INTEGER;
BEGIN
    SELECT salary
    INTO   v_Salary
    FROM   employees
    WHERE  employee_id=154;

    CASE
        WHEN v_Salary< 5000 THEN
            v_Bono:= 500;
        WHEN v_Salary> 12000 THEN
            v_Bono:= 100;
        ELSE
            v_Bono:= 200;
    END CASE;
    Dbms_output.put_line('Bono:'||v_bono);
END;
/
```

Si no se cumpliera una de las condiciones y no existiera la cláusula ELSE, provocaría un error. Por optimización y legibilidad, es recomendable implementar una sentencia CASE en vez de anidar sentencias IF.

Repetitivas

Bucles de 1 a n veces: LOOP

La instrucción LOOP permite ejecutar una serie de sentencias repetidamente. Se coloca la instrucción LOOP en la primera sentencia a repetir y la instrucción END LOOP; después de la última sentencia que forma parte del bucle.

La sentencia EXIT WHEN permite salir del bucle. La condición después del WHEN es evaluada y si se cumple (TRUE) el control pasa a la sentencia inmediatamente posterior a la sentencia END LOOP.

EXIT sin ningún parámetro saldrá inmediatamente del bucle.

Si dentro de la instrucción LOOP no existen las sentencias EXIT WHEN o EXIT, se creara un bucle sin fin.

Ejemplo:

Queremos sacar los diez primeros números naturales:

```
SET SERVEROUTPUT ON
DECLARE
    V_CONT PLS_INTEGER := 0;
BEGIN
    LOOP
        DBMS_OUTPUT.PUT_LINE('NUMERO : ' || V_CONT);

        V_CONT := V_CONT + 1;
        EXIT WHEN V_CONT >= 10;
    END LOOP;
END;
/
```

Bucles de 0 a n veces: WHILE

La sentencia WHILE-LOOP asocia una condición a la repetición de las sentencias del bucle. La condición es evaluada antes de iniciar el bucle. Si se cumple la condición las sentencias son ejecutadas y la sentencia END LOOP devolverá el control al WHILE. Si no se cumple la condición el control del programa pasará a la línea siguiente al END LOOP.

La sentencia EXIT WHEN permite salir del bucle. La condición después del WHEN es evaluada y si se cumple (TRUE) el control pasa a la sentencia inmediatamente posterior a la sentencia END LOOP.

EXIT sin ningún parámetro saldrá inmediatamente del bucle.

Ejemplo:

Queremos sacar los diez primeros números naturales:

```
SET SERVEROUTPUT ON
DECLARE
  V_CONT PLS_INTEGER := 0;
BEGIN
  WHILE V_CONT < 10 LOOP
    DBMS_OUTPUT.PUT_LINE('NUMERO : ' || V_CONT);
    V_CONT := V_CONT + 1;

  END LOOP;
END;
/
```

Bucles de n veces: FOR

La sentencia FOR-LOOP especifica un rango de número enteros, siempre correlativos de uno en uno. Se ejecutarán tantas veces las sentencias del bucle como números enteros se especifiquen en el rango. Se puede referenciar la variable que definimos en el bucle pero no se puede asignar valores a esta variable.

La sentencia EXIT WHEN permite salir del bucle. La condición después del WHEN es evaluada y si se cumple (TRUE) el control pasa a la sentencia inmediatamente posterior a la sentencia END LOOP.

EXIT sin ningún parámetro saldrá inmediatamente del bucle.

Ejemplo:

Queremos sacar los diez primeros números naturales:

```
BEGIN
  FOR I IN 0..9 LOOP
    DBMS_OUTPUT.PUT_LINE('NUMERO : ' || I);
  END LOOP;
END;
/
```

A la vista del ejemplo tenemos que hacer las siguientes consideraciones:

- La variable "I", es local al FOR, y dentro del bucle es de referencia, es decir no se puede alterar su valor.
- En el rango "0..9", se pueden poner literales enteros y/o variables, pero siempre el valor o variable a la izquierda del rango debe ser menor que el de la derecha. Si ambos son iguales, el bucle solo se ejecuta una vez, y si es mayor el de la izquierda el bucle no se ejecuta, y el flujo de control pasa a la siguiente instrucción al END LOOP.

Si se desea que la secuencia de valores sea de mayor a menor se utilizará la cláusula REVERSE.

```
BEGIN
    FOR numero IN REVERSE 1..5 LOOP
        Dbms_output.put_line(numero);
    END LOOP;
END;
/
5
4
3
2
1
```

PL/SQL procedure successfully completed.

En Oracle 11g se introdujo la sentencia CONTINUE; marca el fin inmediato de una iteración de un bucle, y salta a la primera sentencia del bucle. El siguiente código ilustra su funcionamiento, el bucle se salta los números pares (la función escalar MOD(I,2) devuelve el resto de una división entera):

```
BEGIN
    FOR i IN 1..6 LOOP

        IF MOD(i,2) = 0 THEN
            CONTINUE;
        END IF;
        DBMS_OUTPUT.PUT_LINE ('i vale: ' || i);

    END LOOP;
END;
/
i vale: 1
i vale: 3
i vale: 5
```

PL/SQL procedure successfully completed.

Trabajar con varios bloques

Se pueden construir, para realizar un programa cuantos sub-bloques nos sean necesarios, tanto secuenciales como anidados.

```
DECLARE
...
BEGIN
...
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
...
BEGIN
...
BEGIN
...
END;
...
EXCEPTION
...
END;
EXCEPTION
WHEN ... THEN
...
DECLARE
...
BEGIN
...
EXCEPTION
...
END;
...
END;
/

```

Programa PL con varios sub-bloques

Muchos programas PL se construyen en un solo bloque.

Pero en algunas ocasiones nos es interesante encerrar parte del código en un sub-bloque, que depende del bloque principal. Un caso típico es para tratar excepciones asociadas a mandatos SQL/DML, que veremos en el capítulo siguiente.

Visibilidad y ámbito de variables

Todas las variables declaradas en el bloque principal, son **globales**, es decir se pueden referenciar en cualquier parte del programa.

Todas las variables definidas en un sub-bloque, son **locales** al sub-bloque, y no se pueden referenciar en el bloque principal.

1

Variables globales y locales con distinto nombre.

Veamos un ejemplo:

```
set serveroutput on
declare
    v_contador pls_integer := 20;

begin
    dbms_output.put_line('antes subbloque :' || v_contador);
    declare
        v_numero pls_integer := 55;
    begin
        v_contador := v_numero;
        dbms_output.put_line('alterada en subbloque :' || v_conta
    end;

end;
/
antes subbloque : 20
alterada en subbloque : 55
```

PL/SQL procedure successfully completed.

v_contador: variable global

¿Qué pasa si referencio la variable v_numero al final del bloque principal?

```
set serveroutput on
declare
    v_contador pls_integer := 20;

begin
    dbms_output.put_line('antes subbloque : ' || v_contador);
    declare
        v_numero pls_integer := 55;
    begin
        v_contador := v_numero;
        dbms_output.put_line('alterada en subbloque : ' || v_contador);
    end;

    -- referencia a variable local del subbloque anterior
    dbms_output.put_line('contenido v_numero : ' || v_numero);
end;
/
dbms_output.put_line('contenido v_numero : ' || v_numero);
*
```

ERROR at line 12:
ORA-06550: line 12, column 49:
PLS-00201: identifier 'V_NUMERO' must be declared
ORA-06550: line 12, column 1:
PL/SQL: Statement ignored

Que se produce un error. la variable v_numero no la encuentra el bloque principal, te dice que no está declarada.

Si al final del bloque principal hago referencia a la variable global v_contador, como ha sido modificada por el sub-bloque, obtendremos ese valor (55).

```
set serveroutput on
declare
    v_contador pls_integer := 20;

begin
    dbms_output.put_line('antes subbloque : ' || v_contador);
    declare
        v_numero pls_integer := 55;
    begin
        v_contador := v_numero;
        dbms_output.put_line('alterada en subbloque : ' || v_contador);
    end;
    -- referencia a variable global modificada por el subbloque anterior
    dbms_output.put_line('contenido v_contador : ' || v_contador);
end;
/

antes subbloque : 20
alterada en subbloque : 55
contenido v_contador : 55

PL/SQL procedure successfully completed.
```

2

Variables globales y locales con el mismo nombre.

Utilizando el ejemplo anterior, vamos a crear una segunda variable en el sub-bloque, que se llama igual que la del bloque principal: v_contador.

No da error. Pero PL construye en memoria dos variables v_contador, distintas, y cada una se usa en su ámbito.

Observa el resultado tras hacer PUT_LINE de v_contador dentro del sub-bloque y al final del bloque principal.

```
set serveroutput on
declare
    v_contador pls_integer := 20;

begin
    dbms_output.put_line('antes subbloque : ' || v_contador);
    declare
        v_numero pls_integer := 55;
        v_contador pls_integer := 10;
    begin
        v_contador := v_numero;
        dbms_output.put_line('alterada en subbloque : ' || v_contador);
    end;
    -- referencia a variable global modificada por el subbloque anterior
    dbms_output.put_line('contenido v_contador : ' || v_contador);
end;
/
antes subbloque : 20
alterada en subbloque : 55
contenido v_contador : 20

PL/SQL procedure successfully completed.
```

- v_contador en el bloque principal está inicializada a 20. El sub-bloque crea otra variable con el mismo nombre con valor inicial 10, y se cambia por 55. Pero al final del bloque principal imprimo por consola el valor de v_contador, y sigue a 20. Es decir PL está trabajando con dos copias de la variable.
- El sub-bloque crea una variable v_contador con valor inicial a 10, lo cambia por 55, y al imprimirla en el sub-bloque aparece 55. Pero esta variable es independiente de la del bloque principal

3

Referenciar variables con el mismo nombre en distintos bloques.

Para poder utilizar la variable global dentro del sub-bloque, tenemos que cualificarla en el sub-bloque . Para ello damos un nombre al bloque principal, encerrando el nombre entre "<<nombre>>". Veamos un ejemplo:

```
<<bloque>>
DECLARE
    x PLS_INTEGER:=5;
BEGIN

    DECLARE
        x PLS_INTEGER:=3;
        y PLS_INTEGER;
    BEGIN
        y:= x + bloque.x;
        dbms_output.put_line('El resultado es: ||y);
    END;
    dbms_output.put_line('El valor de la variable X es: ||x);
END;
/
El resultado es: 8
El valor de la variable X es: 5

PL/SQL procedure successfully completed.
```

El bloque principal le hemos denominado <<bloque>>. El sub-bloque suma su variable x con la variable x del bloque principal (y: = x + bloque.x;).

Por eso el resultado de la suma es 8.

Cuando en el bloque principal imprimimos la variable x, el resultado es 5 (su contenido inicial).

Un ejemplo resuelto

Queremos obtener el apellido, el salario y la fecha de ingreso (con formato dd-mm-yy), del empleado 107. Además queremos saber el numero de empleados que hay en el departamento al que pertenece y la media de los salarios de las personas de su departamentos. (esta sentencia en Oracle, solo con los datos de las tablas, sin crear vistas ni nada, es imposible de hacer).

Intenta hacer el ejercicio y luego consulta la siguiente 'Solución':

```
set serveroutput on
declare
    v_apel employees.last_name%TYPE;
    v_salario employees.salary%type;
    v_fecha employees.hire_date%type;
    v_dep employees.department_id%type;
    v_cuantos pls_integer :=0;
    v_media employees.salary%type;

begin
    select last_name, salary, hire_date, department_id
    into v_apel, v_salario, v_fecha, v_dep
    from employees
    where employee_id = 107;

    select count(*), avg(salary)
    into v_cuantos, v_media
    from employees
    where department_id = v_dep;
    dbms_output.put('apellido : ' || v_apel);
    dbms_output.put(' salario : ' || v_salario);
    dbms_output.put_line(' fecha ing : ' || to_char(v_fecha,'dd-mm-yy')
    -- sacamos una linea de guiones para separar
    dbms_output.put_line(rpad('-',60,'-'));

    dbms_output.put('departamento : ' || v_dep);
    dbms_output.put(' empleados : ' || v_cuantos);
    dbms_output.put_line(' media salario : ' || round(v_media,2));
```

```
dbms_output.put('apellido : ' || v_apel);
dbms_output.put(' salario : ' || v_salario);
dbms_output.put_line(' fecha ing : ' || to_char(v_fecha,'dd-mm-yy')
-- sacamos una linea de guiones para separar
dbms_output.put_line(rpad('-',60,'-'));

dbms_output.put('departamento : ' || v_dep);
dbms_output.put(' empleados : ' || v_cuantos);
dbms_output.put_line(' media salario : ' || round(v_media,2));

end;
/
apellido : Lorentz salario : 4200 fecha ing : 07-02-07
-----
departamento : 60 empleados : 5 media salario : 5760

PL/SQL procedure successfully completed.
```



PROEDUCA