UNIDAD12. Introducción al lenguaje PL/SQL

1. INTRODUCCIÓN
2. ESTRUCTURA DE UN PROGRAMA EN PL/SQL
3. VARIABLES Y DEFINICIÓN DE DATOS
4. OPERADORES
5. FUNCIONES
6. SOPORTE PARA CONSULTA
7. INTERACCION CON EL USUARIO
8. ESTRUCTURAS DE CONTROL
9. SUBPROGRAMAS: PROCEDIMEINTOS Y FUNCIONES

1. INTRODUCCIÓN

Hasta ahora hemos utilizado SQL en modo interactivo o elaborando pequeños scripts, pero sin control de la secuencia de instrucciones, módulos, iteraciones, etc., . Para este propósito Oracle incorpora PL/SQL.

PL/SQL es un lenguaje procedimental diseñado por ORACLE para trabajar con la base de datos. Esta incluido en el servidor y en algunas herramientas del cliente.

CARACTERÍSTICAS:

- Soporta todos los comandos de consulta y manipulación de datos, aportando a SQL las estructuras de control (bucles, bifurcaciones, etc.,)
- Su unidad de trabajo es el bloque.
- Puede utilizar variables dentro de consultas.
- Es portable entre distintas versiones de ORACLE o entre distintas máquinas y Sistemas Operativos.
- Mejora el rendimiento de Oracle enviando bloques de instrucciones al núcleo de la base de datos para ser procesadas, en lugar de enviar instrucción a instrucción, lo que evita muchas operaciones de entrada/salida.

2. ESTRUCTURA DE UN PROGRAMA EN PL/SQL

BLOQUES es la unidad básica de todos los programas PL/SQL. Su formato incluye tres secciones:

- Zona declaraciones: donde se declaran objetos locales (variables, constantes, etc.,)

  Suele ir precedida de la cl. DECLARE (IS/AS)

  Es opcional.

- Zona instrucciones: comienza por cláusula BEGIN
- Zona tratamiento de excepciones: comienza con EXCEPTION. Nos permite gestionar errores y excepciones. Es opcional.

Formato:

```
[DECLARE
        <declaraciones>]
BEGIN
        <ordenes>
[EXCEPTION
        <gestión de excepciones>]
END;
```

Cada orden debe acabar con ;

Se pueden distinguir tres tipos de programas:

- BLOQUE ANÓNIMO: sin nombre, se ejecutan en el servidor pero no se guardan.
- SUBPROGRAMAS: procedimientos y funciones. Se guardan y se ejecutan en el servidor.
- DISPARADORES DE LA B.D.(TRIGGERS).

COMENTARIOS:

1- De varias líneas /* <comentario>/*

Ejemplo:

/* Este programa de prueba ha sido realizado por C.S./*

2- Al final de una línea de programa utilizando "--"

Ejemplo:

```
Total := total+dinero ;     -- Total invertido en la cuenta
```

3. VARIABLES Y DEFINICIÓN DE DATOS

Las variables en PL/SQL se deben definir y declarar en la sección DECLARE. Para ello se utilizará el siguiente formato:

```
Nombre_variable tipo_variable;
```

Tipos de datos:

PL/SQL dispone de tipos de datos compatibles con los que utilizan las columnas de SQL y de otros propios.

- Igual que SQL: NUMBER, CHAR, VARCHAR2, LONG, RAW, LONG RAW y DATE.
- Propios:

  BINARY_INTEGER: Enteros con signo. Usa números en formato binario lo que evita la conversión como ocurriría con NUMBER. Se utiliza para definir índices en ARRAYS.

  BOOLEAN: puede almacenar tres valores TRUE, FALSE o NULL.

Además PL/SQL maneja otros tipos más complejos:

- Tipos compuestos:registros, tablas y arrays
- Referencias – parecidas a punteros en C
- LOB (objetos de gran tamaño)

Asimismo permite que el programador defina sus propios subtipos.

Identificadores

Se utilizan para nombrar objetos que intervienen en un programa como variables, constantes, cursores, procedimientos, excepciones,etc.,

Reglas para nombrarlos:

. Hasta 30 caracteres, empezando por letra que puede ir seguida de letra, nº, $,# y _.

. No distingue mayúsculas de minúsculas.

Declaración e inicialización de variables

En la sección DECLARE.

Formato:

<nombre_variable> <tipo> [NOT NULL] [{:=|DEFAULT} <valor>]

Ejemplo:

```
DECLARE
   Importe      Number(8,2);
   Contador     Number(2)   DEFAULT 0;
   Nombre       Char(20)      NOT NULL: =’MIGUEL’;
   Nuevo        Varchar2(15);
BEGIN.....
```

- Como podemos observar la asignación se realiza con los símbolos “:=” o la palabra DEFAULT.
- Podemos asignar a una variable la opción NOT NULL y por tanto ya en la definición debemos asignar un valor.
- no se pueden poner una lista de variables para un tipo en una sola línea.
- Si no definimos valor por defecto, no debemos hacer referencia a la variable antes de que se le asigne un valor.

Ámbito y viabilidad de las variables

El ámbito de una variable se ciñe al bloque en el que se declara y sus bloques hijos.

La var. será local para el bloque padre y global para los bloques hijos.

Las var. definidas en los bloques hijos no son accesibles desde el bloque padre.

Ejemplo:

```
DECLARE  ---------------------------------- bloque padre
        V1 char;
BEGIN
        .................
        DECLARE ---------------------------- bloque hijo
                V2 CHAR;
        BEGIN
                ..............................
                V2 := 2;
                V1 :=v2;
                ..............................
        END;
        V2 := v1;   -- --------------------Error :v2 es desconocida en este bloque
END;
```

ATRIBUTOS DE VARIABLES

Además de poder asignar a una variable un determinado tipo, tenemos la posibilidad de asignarle uno de estos atributos: %TYPE y %ROWTYPE.

- *%TYPE*

  Se puede declarar una variable del mismo tipo que otra o que una columna de una tabla

  Ejemplo:

```
Cumple  DATE;
Cumple1 cumple%TYPE;
/* asignarle el mismo tipo que una columna de una tabla*/
nom  alumnos.nombre%TYPE  –tabla.columna%TYPE
```

- %ROWTYPE

Con él podemos declarar una variable para guardar una fila completa con %ROWTYPE.

MIFILA EMPLEADOS %ROWTYPE;

Podríamos hacer SELECT * INTO mifila FROM EMPLE;

(*) Para hacer una referencia a un elemento de MIFILA diremos MIFILA.<nombre_columna>

Al declarar una var. de esta forma esta heredará tipo y longitud pero no atributos NOT NULL o valor por defecto, definidos en la columna de la tabla.

Constantes

Se declaran mediante el siguiente formato:

<nombre_constante> CONSTANT <tipo> := <valor>

Siempre hay que asignarles un valor.

Ejemplo:

Tipo_IVA CONSTANT number := 16;

4. OPERADORES

OPERADORES LÓGICOS
AND OR Y NOT
OPERADOR DE CONCATENACIÓN
||
OPERADORES DE COMPARACIÓN
< <= > >= != IS NULL BETWEEN IN LIKE
OPERADORES ARITMÉTICOS
+ - * / ** Con fechas se puede: f1-f2 f+n f-n

5 .FUNCIONES

Se utilizan todas las funciones de SQL,

Tener en cuenta:

1.- Algunas funciones como las de agrupamiento sólo se pueden utilizar dentro de la cláusula SELECT.

2.- Si a una función se le pasa el valor nulo normalmente devolverá un valor nulo.

6. SOPORTE PARA CONSULTA

FORMATO:

SELECT <columna/s> INTO <variable/s> FROM tabla WHERE.....

Ejemplo:

SQL>SELECT COUNT(*) FROM EMPLE; --ERROR dentro de un bloque PL.

SQL>SELECT COUNT(*) INTO V_TOTAL_EMPLE FROM EMPLE;

7. INTERACCION CON EL USUARIO

PL/SQL no dispone de órdenes para introducir datos por teclado ni para visualizarlos por pantalla. Oracle incorpora la cláusula DBMS_OUTPUT.

FORMATO:

DBMS_OUTPUT.PUT_LINE(<expresiones>);

Para que funcione la cláusula SERVEROUTPUT debe estar en ON.

SQL>SET SERVEROUTPUT ON;

Para poder ver los posibles errores utilizaremos:

SQL>SHOW ERRORS;

Para ejecutar un bloque anónimo cargado en memoria usaremos / o RUN.

8. ESTRUCTURAS DE CONTROL

Como cualquier otro lenguaje procedimental PL/SQL dispone de ordenes para establecer controles y condiciones a nuestros programas.

La mayoría de estas estructuras requieren evaluar una condición, que en PL/SQL pueda dar 3 resultados TRUE, FALSE o NULL. A estos efectos NULL=FALSE, la condición se cumplirá si es TRUE.

ESTRUCTURAS DE CONTROL ALTERNATIVAS		
Alternativa Simple	**Alternativa Doble**	**Alternativa Múltiple**

IF <condición> THEN
Instrucciones;
END IF;
	IF <condición> THEN
	Instrucciones;
	ELSE
	Instrucciones
	END IF;
		IF <condición> THEN
		Instrucciones;
		ELSIF <condición 2> THEN
		Instrucciones;
		ELSIF <condición 3> THEN
		Instrucciones;
		.....
		[ELSE instruccionesotras;]
		END IF;

Ejemplo:

```
SELECT SUM(matricula) INTO renta FROM ALUMNOS WHERE codigo_curso=3;
IF renta>1000000 THEN
        INSERT INTO temp(COL3) values ('Rentable');
ELSE
        IF renta > 500000 THEN
                INSERT INTO TEMP (col3) VALUES ('No rentable')
        ELSE
                INSERT INTO TEMP (col3) VALUES ('Cerrarla')
        END IF;
END IF;
```

ESTRUCTURAS DE CONTROL REPETITIVAS	
Mientras	**Para**
WHILE <condición> LOOP Instrucciones; END LOOP; Se ejecuta mientras se cumple la cond.. Se evalúa la cond., y si se cumple se realizan las instrucciones.	FOR <variable_control> IN [REVERSE] <valor inicio>..<valor final> LOOP Instrucciones; END LOOP; Variablecontrol- es una var. De tipo BINARY__INTEGER que se declara de manera implícita como variable local del bucle. No se puede utilizar fuera de él. Dentro del bucle se puede usar en una expresión pero no se le pueden asignar valores. Cuando se especifica REVERSE va del v. Final al inicial.
Iterar	**Case**
LOOP [instrucciones;] {IF <condición> THEN EXIT; ENDIF; | EXIT WHEN <condición>} Instrucciones; END LOOP;	CASE expressión WHEN <valorcomprbac1> THEN instrucciones1; WHEN <valorcomprbac2> THEN instrucciones2; WHEN <valorcomprbac3> THEN instrucciones3; …….. [ELSE instruccionesotras;] END CASE;

Ejemplos:

```
LOOP
        C := c+1;
        IF c=100 THEN
                EXIT
        END IF;
END LOOP;
```

```
LOOP
        c:= c+1;
        EXIT WHEN c=100;
END LOOP;
```

Consideraciones para el manejo de bucles

Los bucles se pueden etiquetar para conseguir mayor legibilidad:

Ejemplo1:

```
<<mibucle>>
LOOP
    <secuencia de instrucciones>
END LOOP mibucle;
```

Ejemplo2:

```
<<mibucle>>
LOOP
.....
        LOOP
.       .....
        EXIT mibucle WHEN ..... –se sale de ambos
        END LOOP
END LOOP mibucle;
```

9. SUBPROGRAMAS: PROCEDIMIENTOS Y FUNCIONES

Los procedimientos y las funciones son bloques en PL que tienen un **nombre** pudiendo recibir y devolver valores. Se guardan en la base de datos y podemos ejecutarlos, invocándolos desde otros programas.

PROCEDIMIENTOS

FORMATO:

```
CREATE [OR REPLACE]  PROCEDURE <nbprocedimiento>
        [(<lista de parámetros>)]
IS
        [<declaraciones>]
BEGIN
        <instrucciones>;
[EXCEPTION
        <excepciones>;]
END [<nbprocedimiento>];
```

FUNCIONES

FORMATO:

```
CREATE [OR REPLACE]  FUNCTION <nbfuncion>
        [(<lista de parámetros>)]
RETURN <tipo de valor devuelto>
IS
        [<declaraciones>]
BEGIN
        <instrucciones>;
        RETURN <expresión>;

[EXCEPTION
        <excepciones>;]
END [<nbfuncion>];
```

RETURN de la cabecera especifica el tipo de valor que retorna la función.

RETURN en el cuerpo del programa devuelve el control al programa que llamo la función, asignando el valor de la expresión (que sigue al return) a la variable que figura en la llamada de la función.

Los procedimientos y las funciones serán almacenados en la b.d. donde quedarán disponibles para ser ejecutados mediante la cláusula EXECUTE.

SQL>**EXECUTE nbproc/fun (valor/es);**

Para que funcione la cláusula SERVEROUTPUT debe estar ON.

SQL>SET SERVEROUTPUT ON;

Para invocar a los procedimientos o las funciones desde otro bloque, procedimiento o función:

Nombredelprocedimientoalquellamamos (lista de parámetros)