UNIDAD DE TRABAJO 3: PROGRAMACIÓN DE BASES DE DATOS

TEMA 8: INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN

8.1. INTRODUCCIÓN AL SQL PROCEDIMENTAL

Casi todos los grandes Sistemas Gestores de Datos incorporan utilidades que permiten ampliar el lenguaje **SQL** para producir pequeñas utilidades que añaden al SQL mejoras de la programación estructurada (bucles, condiciones, funciones,….). La razón es que hay diversas acciones en la base de datos para las que SQL no es suficiente.
Por ello todas las bases de datos incorporan algún lenguaje de tipo procedimental (de tercera generación) que permite manipular de forma más avanzada los datos de la base de datos.
PL/SQL es el lenguaje procedimental que es implementado por el precompilador de **Oracle**. Es una extensión procedimental del lenguaje SQL; es decir, se trata de un lenguaje creado para dar a SQL nuevas posibilidades. Esas posibilidades permiten utilizar condiciones y bucles al estilo de los lenguajes de tercera generación (como **Basic**, **Cobol**, **C++, Java**, etc.).
En otros sistemas gestores de bases de datos existen otros lenguajes procedimentales: **SQL Server** utiliza **Transact SQL**, **Informix** usa **Informix 4GL**,…
Lo interesante del lenguaje PL/SQL es que integra SQL por lo que gran parte de su sintaxis procede de dicho lenguaje.
PL/SQL es un lenguaje pensado para la gestión de datos. La creación de aplicaciones sobre la base de datos se realiza con otras herramientas (**Oracle Developer**) o lenguajes externos como **Visual Basic** o **Java**. El código PL/SQL puede almacenarse:

- En la propia base de datos
- En archivos externos

Para hacer un programa PL SQL:

- **Desde un bloque anónimo**: nos metemos en SQL PLUS: lo escribimos y para ejecutarlo: / (barra del 7)
- **También podemos editar un .sql**: así quedará almacenado. Al final del código, debemos poner la barra del 7. Después llamarlo desde SQLPLUS con la @ delante.
- **Guardarlos dentro del esquema del usuario:** si tiene privilegios (create procedure o create function) podrá crearlos (para ver los privilegios: select * from user_privileges)

- Cuando creamos un procedimiento desde un .sql y lo ejecutamos con la @ desde SQLPLUS, lo que estamos haciendo es **crear** ese procedimiento (bajo el esquema del usuario que lo ejecuta). Comprueba privilegios sintaxis. Para ejecutarlo: exec nombre_procedimiento;

8.2. FUNCIONES QUE PUEDEN REALIZAR LOS PROGRAMAS PL/SQL

Las más destacadas son:

- Facilitar la realización de tares administrativas sobre la base de datos (copia de valores antiguos, auditorías, control de usuarios,…)
- Validación y verificación avanzada de usuarios
- Consultas muy avanzadas
- Tareas imposibles de realizar con SQL

8.3. CONCEPTOS BÁSICOS

bloque PL/SQL
Se trata de un trozo de código que puede ser interpretado por Oracle. Se encuentra inmerso dentro de las palabras BEGIN y END.

programa PL/SQL
Conjunto de bloques que realizan una determinada labor.

procedimiento
Programa PL/SQL almacenado en la base de datos y que puede ser ejecutado si se desea con solo saber su nombre (y teniendo permiso para su acceso).

función
Programa PL/SQL que a partir de unos datos de entrada obtiene un resultado (datos de salida). Una función puede ser utilizada desde cualquier otro programa PL/SQL e incluso desde una instrucción SQL.

trigger (disparador)
Programa PL/SQL que se ejecuta automáticamente cuando ocurre un determinado suceso a un objeto de la base de datos.

paquete
Colección de procedimientos y funciones agrupados dentro de la misma estructura. Similar a las bibliotecas y librerías de los lenguajes convencionales.

8.4. ESCRITURA DE PL/SQL

8.4.1 Estructura de un bloque PL/SQL

Ya se ha comentado antes que los programas PL/SQL se agrupan en estructuras llamadas **bloques**. Cuando un bloque no tiene nombre, se le llama **bloque anónimo**. Un bloque consta de las siguientes secciones:

Declaraciones. Define e inicializa las variables, constantes, excepciones de usuario y cursores utilizados en el bloque. Va precedida de la palabra **DECLARE**

Comandos ejecutables. Sentencias para manipular la base de datos y los datos del programa. Todas estas sentencias van precedidas por la palabra **BEGIN.**

Tratamiento de excepciones. Para indicar las acciones a realizar en caso de error. Van precedidas por la palabra **EXCEPTION**

Final del bloque. La palabra **END** da fin al bloque.

La estructura en sí es:

[**DECLARE**
declaraciones **]**
BEGIN
instrucciones ejecutables
[**EXCEPTION**
instrucciones de manejo de errores **]**
END;

A los bloques se les puede poner nombre usando (así se declara un procedimiento):

PROCEDURE nombre **IS**
bloque

Para una función se hace:

FUNCTION nombre
RETURN tipoDedatos **IS**
bloque

Cuando un bloque no se declara como procedimiento o función, se trata de un bloque anónimo.

8.4.2. Escritura de instrucciones PL/SQL

Normas básicas

La mayor parte de las normas de escritura en PL/SQL proceden de SQL, por ejemplo:

- Las palabras clave, nombres de tabla y columna, funciones,… no distinguen entre mayúsculas y minúsculas
- Todas las instrucciones finalizan con el signo del punto y coma (**;**), excepto las encabezan un bloque
- Los bloques comienzan con la palabra BEGIN y terminan con END
- Las instrucciones pueden ocupar varias líneas

Comentarios

Pueden ser de dos tipos:

Comentarios de varias líneas. Comienzan con **/*** y terminan con ***/**

Comentarios de línea simple. Son los que utilizan los signos **--** (doble guión). El texto a la derecha de los guiones se considera comentario (el de la izquierda no)

Ejemplo:

```
DECLARE
        v NUMBER := 17;
BEGIN
        /* Este es un comentario que
        ocupa varias líneas */
        v:=v*2; -- este sólo ocupa esta línea
        DBMS_OUTPUT.PUT_LINE(v) -- escribe 34
END;
```

8.5. VARIABLES

8.5.1. Uso de variables

Declarar variables

Las variables se declaran en el apartado **DECLARE** del bloque. PL/SQL no acepta entrada ni salida de datos por sí mismo (para conseguirlo se necesita software auxiliar). La sintaxis de la declaración de variables es:

DECLARE
identificador **[CONSTANT]** tipoDeDatos **[**:= valorInicial**];**
[siguienteVariable…**]**

Ejemplos:
DECLARE
pi **CONSTANT NUMBER**(9,7):=3.1415927;
radio **NUMBER**(5);
area **NUMBER**(14,2) := 23.12;

El operador := sirve para asignar valores a una variable. Este operador permite inicializar la variable con un valor determinado. La palabra **CONSTANT** indica que la variable no puede ser modificada (es una constante). Si no se inicia la variable, ésta contendrá el valor NULL.
Los identificadores de Oracle deben de tener 30 caracteres, empezar por letra y continuar con letras, números o guiones bajos (**_**) (también vale el signo de dólar (**$**) y la almohadilla (**#**). No debería coincidir con nombres de columnas de las tablas ni con palabras reservadas (como SELECT).
En PL/SQL sólo se puede declarar una variable por línea.

Tipos de datos para las variables

Las variables PL/SQL pueden pertenecer a uno de los siguientes datos (sólo se listan los tipos básicos, los llamados **escalares**), la mayoría son los mismos del SQL de Oracle.

tipo de datos	descripción
CHAR(n)	Texto de anchura fija
VARCHAR2(n)	Texto de anchura variable
NUMBER[(p[,s])]	Número. Opcionalmente puede indicar el tamaño del número (*p*) y el número de decimales (s)
DATE	Almacena fechas
TIMESTAMP	Almacena fecha y hora
INTERVAL YEAR TO MONTH	Almacena intervalos de años y meses
INTERVAL DAY TO SECOND	Almacena intervalos de días, horas, minutos y segundos
LONG	Para textos de más de 32767 caracteres
LONG RAW	Para datos binarios. PL/SQL no puede mostrar estos datos directamente
INTEGER	Enteros de -32768 a 32767
BINARY_INTEGER	Enteros largos (de -2.147.483.647 a -2.147.483.648)
PLS_INTEGER	Igual que el anterior pero ocupa menos espacio
BOOLEAN	Permite almacenar los valores TRUE (verdadero) y FALSE (falso)
BINARY_DOUBLE	Disponible desde la versión 10g, formato equivalente al double del lenguaje C. Representa números decimales en coma flotante.
BINARY_FLOAT	Otro tipo añadido en la versión 10g, equivalente al float del lenguaje C.

expresión %TYPE

Se utiliza para dar a una variable el mismo tipo de otra variable o el tipo de una columna de una tabla de la base de datos. La sintaxis es:

identificador variable | tabla.columna**%TYPE**;

Ejemplo:

nom **personas.nombre%TYPE**;
precio **NUMBER**(9,2);
precio_iva precio**%TYPE**;

La variable ***precio_iva*** tomará el tipo de la variable precio (es decir ***NUMBER(9,2)***) la variable ***nom*** tomará el tipo de datos asignado a la columna ***nombre*** de la tabla ***personas***.

8.5.2 DBMS_OUTPUT.PUT_LINE

Para poder mostrar datos (fechas, textos y números), Oracle proporciona una función llamada **put_line** en el paquete **dbms_output**. Ejemplo:

```
DECLARE
        a NUMBER := 17;
BEGIN
        DBMS_OUTPUT.PUT_LINE(a);
END;
```

Eso escribiría el número 17 en la pantalla. Pero para ello se debe habilitar primero el paquete en el entorno de trabajo que utilicemos. En el caso de iSQL*Plus hay que colocar la orden interna (no lleva punto y coma):

```
SET SERVEROUTPUT ON
```

hay que escribirla antes de empezar a utilizar la función.

8.5.3 Alcance de las variables

Ya se ha comentado que en PL/SQL puede haber un bloque dentro de otro bloque. Un bloque puede anidarse dentro de:

Un apartado **BEGIN**

Un apartado **EXCEPTION**

Hay que tener en cuenta que las variables declaradas en un bloque concreto, son eliminadas cuando éste acaba (con su END correspondiente).

Ejemplo:

```
DECLARE
        v NUMBER := 2;
BEGIN
        v:=v*2;
        DECLARE
                z NUMBER := 3;
        BEGIN
                z:=v*3;
                DBMS_OUTPUT.PUT_LINE(z); --escribe 12
                DBMS_OUTPUT.PUT_LINE(v); --escribe 4
        END;
        DBMS_OUTPUT.PUT_LINE(v*2); --escribe 8
        DBMS_OUTPUT.PUT_LINE(z); --error
END;
```

En este ejemplo se produce un error porque ***z*** no es accesible desde ese punto, el bloque interior ya ha finalizado. Sin embargo desde el bloque interior sí se puede acceder a ***v***

8.5.4. Operadores y funciones

operadores

En PL/SQL se permiten utilizar todos los operadores de SQL: los operadores aritméticos (**+-*/**), condicionales (**> < != <> >= <= OR AND NOT**) y de cadena (**||**).
A estos operadores, PL/SQL añade el operador de potencia ******. Por ejemplo ***4**3*** es 4_3.

funciones

Se pueden utilizar las funciones de Oracle procedentes de SQL (**TO_CHAR, SYSDATE, NVL, SUBSTR, SIN,** etc., etc.) excepto la función **DECODE** y las funciones de grupo (**SUM, MAX, MIN, COUNT**,...), salvo en las instrucciones SQL permitidas.
A estas funciones se añaden diversas procedentes de paquetes de Oracle o creados por los programadores y las funciones **GREATEST** y **LEAST**

8.5.5 Paquetes estándar

Oracle incorpora una serie de paquetes para ser utilizados dentro del código PL/SQL. Es el caso del paquete **DBMS_OUTPUT** que sirve para utilizar funciones y procedimientos de escritura como **PUT_LINE**.

Por ejemplo **DBMS_OUTPUT.NEW_LINE()** sirve para escribir una línea en blanco en el buffer de datos.

Números aleatorios

El paquete **DBMS_RANDOM** contiene diversas funciones para utilizar número aleatorios. Quizá la más útil es la función **DBMS_RANDOM.RANDOM** que devuelve un número entero (positivo o negativo) aleatorio (y muy grande).

Por ello si deseáramos un número aleatorio entre 1 y 10 se haría con la expresión:

MOD(ABS(DBMS_RANDOM.RANDOM),10)+1

Entre 20 y 50 sería:

MOD(ABS(DBMS_RANDOM.RANDOM),31)+20

8.6. INSTRUCCIONES SQL PERMITIDAS

8.6.1 Instrucciones SELECT en PL/SQL

PL/SQL admite el uso de un SELECT que permite almacenar valores en variables.

Es el llamado **SELECT INTO**.

Su sintaxis es:

SELECT listaDeCampos
INTO listaDeVariables
FROM tabla
[**JOIN** …]
[**WHERE** condición]

La cláusula INTO es obligatoria en PL/SQL y además la expresión SELECT **sólo puede devolver una única fila;** de otro modo, ocurre un error.

La cláusula INTO es obligatoria en PL/SQL y además la expresión SELECT **sólo puede devolver una única fila;** de otro modo, ocurre un error.

Ejemplo:

```
DECLARE
        v_salario NUMBER(9,2);
        v_nombre VARCHAR2(50);
BEGIN
        SELECT salario,nombre INTO v_salario, v_nombre
        FROM empleados WHERE id_empleado=12344;
        SYSTEM_OUTPUT.PUT_LINE('El nuevo salario será de ' ||
        salario*1.2 || 'euros');
END;
```

8.6.2 Instrucciones DML y de transacción

Se pueden utilizar instrucciones DML dentro del código ejecutable. Se permiten las instrucciones INSERT, UPDATE, DELETE y MERGE (hacer en una sola sentencia lo que haríamos en dos: comprobar que un registro existe y, o bien, cambiar un valor (update) o bien insertar valor (insert); con la ventaja de que en PL/SQL pueden utilizar variables.
Las instrucciones de transacción ROLLBACK y COMMIT también están permitidas para anular o confirmar instrucciones.

Ejemplo MERGE:

```
SQL> SELECT * FROM dept;

   DEPTNO DNAME        LOC
--------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       60 HELP DESK      PITTSBURGH
       40 OPERATIONS     BOSTON

SQL> SELECT * FROM dept_online;

   DEPTNO DNAME        LOC
--------- -------------- -------------
       40 OPERATIONS     BOSTON
       20 RESEARCH DEV   DALLAS
       50 ENGINEERING    WEXFORD


SQL> MERGE INTO dept d //en esta insertamos o actualizamos
     USING (SELECT deptno, dname, loc
            FROM dept_online) o
     ON (d.deptno = o.deptno)
     WHEN MATCHED THEN
        UPDATE SET d.dname = o.dname, d.loc = o.loc
     WHEN NOT MATCHED THEN
        INSERT (d.deptno, d.dname, d.loc)
        VALUES (o.deptno, o.dname, o.loc);
```

8.7. INSTRUCCIONES DE CONTROL DE FLUJO

Son las instrucciones que permiten ejecutar un bloque de instrucciones u otro dependiendo de una condición. También permiten repetir un bloque de instrucciones hasta cumplirse la condición (es lo que se conoce como bucles).
La mayoría de estructuras de control PL/SQL son las mismas que las de los lenguajes tradicionales como C o Pascal. En concreto PL/SQL se basa en el lenguaje Ada.

8.7.1 Instrucción IF

Se trata de una sentencia tomada de los lenguajes estructurados. Desde esta sentencia se consigue que ciertas instrucciones se ejecuten o no dependiendo de una condición

sentencia IF simple

Sintaxis:

```
IF condicion THEN
        instrucciones
END IF;
```

Las instrucciones se ejecutan en el caso de que la condición sea verdadera. La condición es cualquier expresión que devuelva verdadero o falso. Ejemplo:

```
IF departamento=134 THEN
        salario := salario * 13;
        departamento := 123;
END IF;
```

sentencia IF-THEN-ELSE

Sintaxis:

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

En este caso las instrucciones bajo el ELSE se ejecutan si la condición es falsa.

sentencia IF-THEN-ELSIF

Cuando se utilizan sentencias de control es común desear anidar un IF dentro de otro IF.

Ejemplo:

```
IF saldo>90 THEN
        DBMS_OUTPUT.PUT_LINE('Saldo mayor que el esperado');
ELSE
        IF saldo>0 THEN
                DBMS_OUTPUT.PUT_LINE('Saldo menor que el esperado');
        ELSE
                DBMS_OUTPUT.PUT_LINE('Saldo NEGATIVO');
        END IF;
END IF;
```

Otra solución es utilizar esta estructura:

```
IF condición1 THEN
        instrucciones1
ELSIF condición2 THEN
        instrucciones3
[ELSIF.... ]
[ELSE
        instruccionesElse ]
END IF;
```

En este IF (que es el más completo) es evalúa la primera condición; si es verdadera se ejecutan las primeras instrucciones y se abandona el IF; si no es así se mira la siguiente condición y si es verdadera se ejecutan las siguientes instrucciones, si es falsa se va al siguiente ELSIF a evaluar la siguiente condición, y así sucesivamente. La cláusula ELSE se ejecuta sólo si no se cumple ninguna de las anteriores condiciones.

Ejemplo (equivalente al anterior):

```
IF saldo>90 THEN
        DBMS_OUTPUT.PUT_LINE('Saldo mayor que el esperado');
ELSIF saldo>0 THEN
        DBMS_OUTPUT.PUT_LINE('Saldo menor que el esperado');
ELSE
        DBMS_OUTPUT.PUT_LINE('Saldo NEGATIVO');
END IF;
```

8.7.2 sentencia CASE

La sentencia CASE devuelve un resultado tras evaluar una expresión. Sintaxis:

```
CASE selector
        WHEN expresion1 THEN resultado1
        WHEN expresion2 THEN resultado2
        ...
        [ELSE resultadoElse]
END;
```

Ejemplo:

```
texto:= CASE actitud
        WHEN 'A' THEN 'Muy buena'
        WHEN 'B' THEN 'Buena'
        WHEN 'C' THEN 'Normal'
        WHEN 'D' THEN 'Mala'
        ELSE 'Desconocida'
END;
```

Hay que tener en cuenta que la sentencia CASE sirve para devolver un valor y no t€ para ejecutar una instrucción.
También se pueden escribir sentencias CASE más complicadas. Por ejemplo:

```
aprobado:= CASE
        WHEN actitud='A' AND nota>=4 THEN TRUE
        WHEN nota>=5 AND (actitud='B' OR actitud='C') THEN TRUE
        WHEN nota>=7 THEN TRUE
        ELSE FALSE
END;
```

8.7.3 Bucles

bucle LOOP

Se trata de una instrucción que contiene instruccies que se repiten indefinidamente (bucle infinito). Se inicia con la palabra **LOOP** y finaliza con la palabra **END LOOP** y dentro de esas palabras se colocan las instrucciones que se repetirán.
Lógicamente no tiene sentido utilizar un bucle infinito, por eso existe una instrucción llamada **EXIT** que permite abandonar el bucle. Cuando Oracle encuentra esa instrucción, el programa continúa desde la siguiente instrucción al END LOOP.
Lo normal es colocar EXIT dentro de una sentencia IF a fin de establecer una condición de salida del bucle. También se puede acompañar a la palabra EXIT de la palabra WHEN seguida de una condición. Si se condición es cierta, se abandona el bucle, sino continuamos dentro.

Sintaxis

```
LOOP
        instrucciones
        ...
        EXIT [WHEN condición]
END LOOP;
```

Ejemplo (bucle que escribe los números del 1 al 10):

```
DECLARE
        cont NUMBER :=1;
BEGIN
        LOOP
                DBMS_OUTPUT.PUT_LINE(cont);
                EXIT WHEN cont=10;
                cont:=cont+1;
        END LOOP;
END;
```

bucle WHILE

Genera un bucle cuyas instrucciones se repiten mientras la condición que sigue a la palabra WHILE sea verdadera. Sintaxis:

```
WHILE condición LOOP
        instrucciones
END LOOP;
```

En este bucle es posible utilizar (aunque no es muy habitual en este tipo de bucle) la instrucción **EXIT** o **EXIT WHEN**. La diferencia con el anterior es que este es más estructurado (más familiar para los programadores de lenguajes como Basic, Pascal, C, Java,...)
Ejemplo (escribir números del 1 al 10):

```
DECLARE
        cont NUMBER :=1;
BEGIN
        WHILE cont<=10 LOOP
                DBMS_OUTPUT.PUT_LINE(cont);
                cont:=cont+1;
        END LOOP;
END;
```

bucle FOR

Se utilizar para bucles con contador, bucles que se recorren un número concreto de veces. Para ello se utiliza una variable (contador) que no tiene que estar declarada en el **DECLARE**, esta variable es declarada automáticamente en el propio **FOR** y se elimina cuando éste finaliza.
Se indica el valor inicial de la variable y el valor final (el incremento irá de uno en uno). Si se utiliza la cláusula **REVERSE**, entonces el contador cuenta desde el valor alto al bajo restando 1.

Sintaxis:

```
FOR contador IN [REVERSE] valorBajo..valorAlto
        instrucciones
END LOOP;
```

bucles anidados

Se puede colocar un bucle dentro de otro sin ningún problema, puede haber un **WHILE** dentro de un **FOR**, un **LOOP** dentro de otro LOOP, etc.
Hay que tener en cuenta que en ese caso, la sentencia EXIT abandonaría el bucle en el que estamos:

```
FOR i IN 1..10 LOOP
        FOR j IN 1..30 LOOP
                EXIT WHEN j=5;
                ...
        END LOOP;
        ...
END LOOP;
```

El bucle más interior sólo cuenta hasta que j vale 5 ya que la instrucción EXIT abandona el bucle más interior cuando ***j*** llega a ese valor.
No obstante hay una variante de la instrucción EXIT que permite salir incluso del bucle más exterior. Eso se consigue poniendo una etiqueta a los bucles que se deseen. Una etiqueta es un identificador que se coloca dentro de los signos **<<** y **>>** delante del bucle. Eso permite poner nombre al bucle.

Por ejemplo:

```
<<buclei>>
FOR i IN 1..10 LOOP
        FOR j IN 1..30 LOOP
                EXIT buclei WHEN j=5;
        ...
        END LOOP;
        ...
END LOOP buclei;
```

En este caso cuando ***j*** vale 5 se abandonan ambos bucles. No es obligatorio poner la etiqueta en la instrucción END LOOP (en el ejemplo en la instrucción ***END LOOP buclei***), pero se suele hacer por dar mayor claridad al código.