



1. Introducció a la programació SQL submergit

NF1. Introducció a SQL submergit

UF3 - Llenguatges SQL: DCL i extensió procedimental

Desenvolupament d'Aplicacions Multiplataforma

M02 – Bases de dades. Versió 1.0

© M^a Carmen Brito Ruiz



1.1. Introducció.

1.2. Bloque PL/SQL

1.2.1. Ejecución de un programa PL/SQL

1.2.2. Tipos de bloques

1.3. Identificadores, Variables y Constantes

1.3.1. Identificadores

1.3.2. Variables

1.3.2.1. Tipos de datos

1.3.2.2. Variables no PL/SQL o variables de HOST o variables de sustitución

1.3.3. Constantes

1.3.4. Los atributos %TYPE y %ROWTYPE

1.3.5. Ejemplos de variables

1.3.6. Variables de sistema

1.4. Tipos de datos compuestos

Desenvolupament d'Aplicacions Multiplataforma – M02 Bases de dades

UF3: Llenguatges SQL: DCL i extensió procedimental - NF1: Introducció a SQL submergit -

EA 3.1.1 Introducció a la programació SQL submergit

Versió 1.0 - © M^a Carmen Brito

1.1. Introducció

El **lenguaje PL/SQL**, incorpora todas las características propias de los lenguajes de tercera generación, es decir, manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles, etc.), control de excepciones, etc.

El lenguaje PL/SQL es un lenguaje procedimental diseñado por Oracle para trabajar con bases de datos. Este lenguaje está incluido en el servidor y en algunas herramientas que se encuentran en el cliente.

Algunas ventajas de PL/SQL son:

- Mejor rendimiento.
- Desarrollo de programas modulares.
- Es portable: es un lenguaje nativo de Oracle Server y por tanto los programas se pueden ejecutar en cualquier sistema operativo y/o plataforma.
- Permite declarar variables.
- Se puede programar con estructuras de control de lenguaje procedural.
- Se puede manejar errores.
- Fácil mantenimiento.
- Mayor integridad y seguridad de los datos.
- Mayor claridad del código.

Instrucciones y sintaxis de los bloques PL/SQL:

- Las sentencias pueden ocupar varias líneas.
- Tenemos las siguientes unidades léxicas:
 - delimitadores:
 - identificadores (se estudiara en el apartado 5.3. de este tema)
 - literales
 - comentarios
- Se pueden usar funciones de SQL en PL/SQL y los operadores de SQL en PL/SQL (se estudiaran en este tema en el apartado 5.5.).

El código ha de estar sangrado.

```
VARIABLE v_salario NUMBER
BEGIN
    SELECT salary
    INTO :v_salario
    FROM employees
    WHERE employee_id= 100;
END;
/
```

Delimitadores: son símbolos simples o compuestos.

Símbolos simples	
Símbolo	Descripción
+	Operador suma
-	Operador resta
*	Operador multiplicación
/	Operador división
=	Operador relacional
@	Indicador acceso remoto
;	Terminador de sentencias

Símbolos compuestos	
Símbolo	Descripción
<>	Operador relacional
!=	Operador relacional
	Operador concatenación
--	Indicador de comentarios de una línea
/*	Delimitador del principio del comentario
*/	Delimitador del final del comentario
:=	Operador de asignación

Literales: es un valor numérico, de carácter, de cadena o booleano. Y se ha de tener en cuenta que:

- Los literales de fecha y carácter han de ir entre comillas simples.
- Los números pueden ser valores simples o notaciones complejas.

Por ejemplo:

```
v_nombre := 'Maria';
```


Comentarios: para incluir comentarios en el código se usa dos guiones (--) para comentar una línea, o bien, /* y */ para comentar más de una línea.

```
VARIABLE v_salario NUMBER    -- variable no PL/SQL
BEGIN
  SELECT salary
  INTO :v_salario /* hacer referencia a una variable no PL/SQL */
  FROM employees
  WHERE employee_id= 100; /* imprimir una variable no PL/SQL */
END;
/
```

Funciones SQL en PL/SQL

Las funciones que estudiamos en SQL, se usan también en PL/SQL, como son:

- Número de una fila
- Carácter de una fila
- Conversión de tipos de dato: TO_CHAR, TO_DATE y TO_NUMBER.
- Fecha
- Registro de hora
- etc

Las que no se pueden usar en PL/SQL, pues no son sentencias procedurales son el DECODE y las funciones de grupo.

1.2. Bloque PL/SQL

[DECLARE

<declaraciones de variables, cursores,
excepciones definidas por el usuario>]

} Opcional

BEGIN

<cuerpo: sentencias SQL y PL/SQL>

} Obligatorio

[EXCEPTION

<gestión de las excepciones (control de
errores)>]

} Opcional

END;

} Obligatorio


1.2.1. Ejecución de un programa PL/SQL

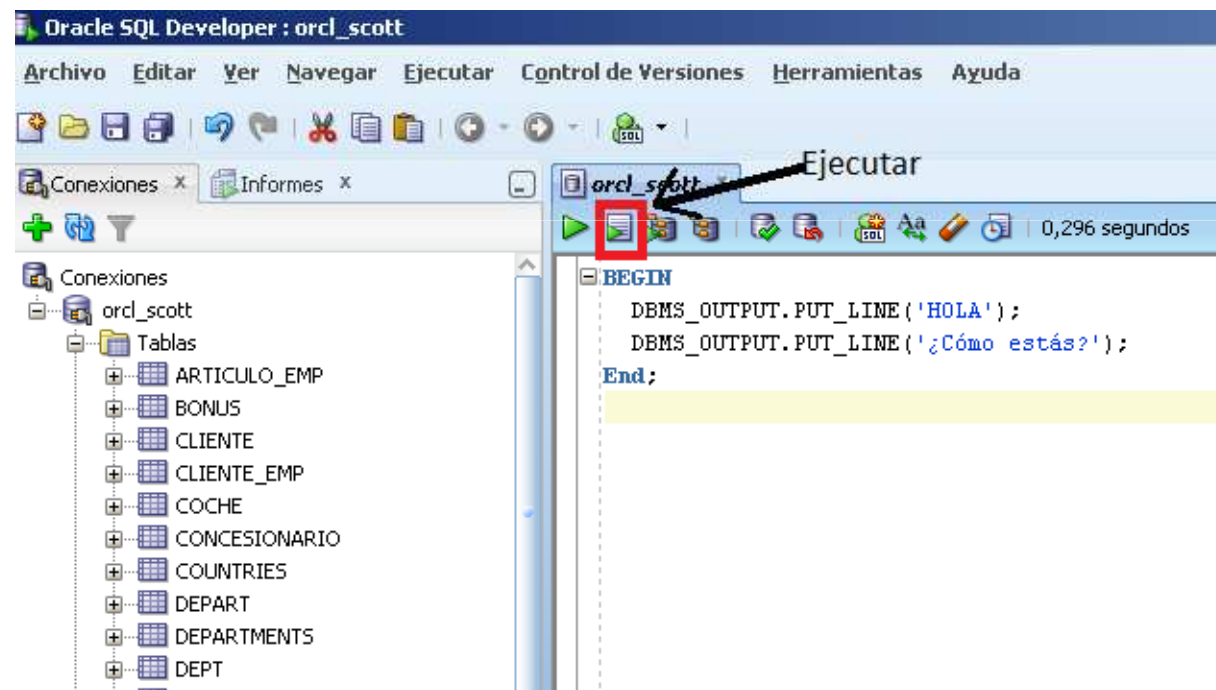
Si tenemos el siguiente código:

```
BEGIN  
    DBMS_OUTPUT.PUT_LINE( 'HOLA' );  
    DBMS_OUTPUT.PUT_LINE( '¿Cómo estás?' );  
END;
```

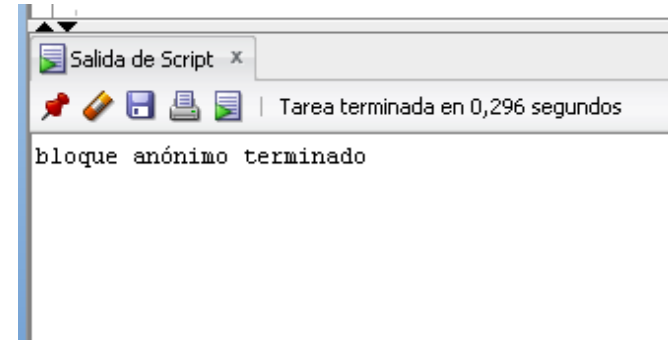
```
HOLA  
¿Cómo estás?
```

Ejecución de un programa PL/SQL con Sqldeveloper

Se escribe edita el programa correspondiente en la parte derecha superior, o bien, se carga el script si ya existe con la opción Archivo / Abrir. Después se da al botón  o F5 para ejecutar el script.



En caso de que no aparezca ningún resultado y no se imprima ningún mensaje.

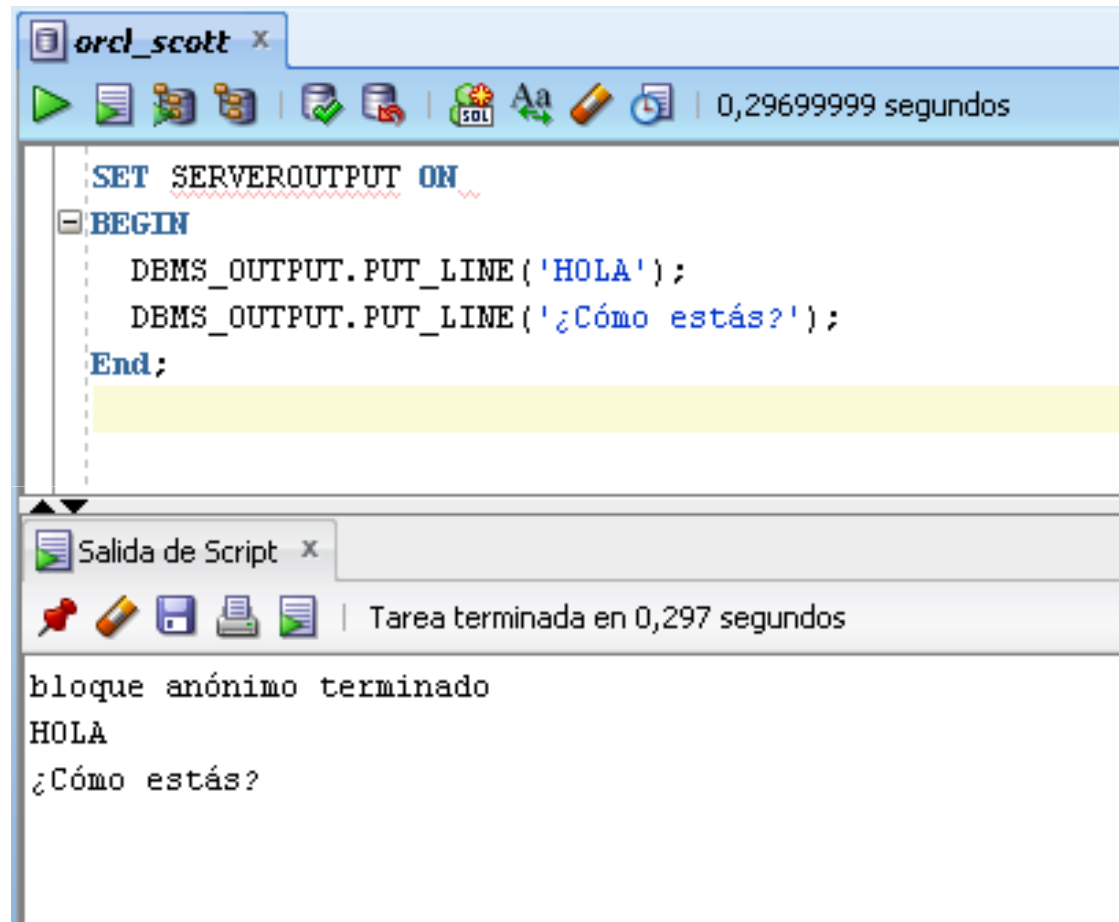


Se ha de activar la librería `DBMS_OUTPUT.PUT_LINE`, es decir, escribir desde el prompt SQL y ejecutar: `SET SERVEROUTPUT ON`

O bien, al principio del script:

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE( 'HOLA' );
    DBMS_OUTPUT.PUT_LINE( '¿Cómo estás?' );
END ;
/
```

Introducció a la programació SQL submergit



The screenshot shows the Oracle SQL Developer interface. The main window, titled 'orcl_scott x', contains a PL/SQL script with the following code:

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.PUT_LINE('HOLA');
  DBMS_OUTPUT.PUT_LINE('¿Cómo estás?');
End;
```

Below the script editor is a 'Salida de Script' (Script Output) window. It displays the execution results:

```
bloque anónimo terminado
HOLA
¿Cómo estás?
```

The status bar of the output window indicates 'Tarea terminada en 0,297 segundos'.

Ejecución de un programa PL/SQL desde SQL *Plus:

- a) **RUN** <nombre_bloque> \Rightarrow ejecuta el bloque, imprimiendo por pantalla el resultado de las instrucciones que contiene dicho bloque y el proceso que efectúa el bloque (no carga ninguna información en el buffer).
- b) **START** <nombre_bloque> \Rightarrow carga y ejecuta el bloque, no imprime el proceso que efectúa el bloque. En caso de que Oracle se quede esperando algo, poner la barra (/).

Ejecución de un programa PL/SQL con iSqlPlus:

- 1) Se examina para buscar el fichero .sql que contiene el script.
- 2) Se carga el archivo de comandos a ejecutar.
- 3) Se ejecuta.



Pantalla de Trabajo

Archivo o URL:

Introducir Sentencias:

```
BEGIN
  DBMS_OUTPUT.PUT_LINE('HOLA');
  DBMS_OUTPUT.PUT_LINE('¿Cómo estás?');
END;
```

HOLA
¿Cómo estás?

1.2.2. Tipos de bloques

Anónimo

Son sentencias SQL. Se ejecutan una sola vez, es decir, es semejante a un script.

La sintaxis es la siguiente:

```
[ DECLARE ]  
  
BEGIN  
  
...  
  
[ EXCEPTION ]  
  
END ;
```

Procedimiento

En este bloque se almacena en el servidor y se ejecuta cada vez que se invoque. La sintaxis es la siguiente:

```
PROCEDURE nombre_procedimiento  
  
IS/AS  
  
    Variable  
  
BEGIN  
  
    ...  
  
    [EXCEPTION]  
  
END nombre_procedimiento;
```

Función

Es semejante a un procedimiento, con la diferencia de que se le pasa un parámetro (como mínimo) y devuelve un valor. La sintaxis es la siguiente:

```
FUNCTION nombre_funcion  
RETURN tipo_dato_a_devolver  
IS/AS  
BEGIN  
...  
RETURN valor_a_devolver;  
[EXCEPTION]  
END nombre_funcion;
```

1.3. Identificadores, Variables y Constantes

1.3.1. Identificadores

Los **identificadores** se usan para nombrar los objetos que intervienen en un programa y son: variables, constantes, cursores, excepciones, procedimientos, funciones, etiquetas, etc.

Aspectos a tener en cuenta con los identificadores:

- Pueden contener hasta 30 caracteres y siempre ha de comenzar por un carácter. Pueden contener números, signos de dólar, subrayados y almohadillas. No pueden contener caracteres como guiones, barras inclinadas y espacios.
- No deberían tener el mismo nombre que una columna de la tabla de base de datos o tabla.
- No debería ser palabras reservadas.

1.3.2. Variables

En PL/SQL se pueden declarar **variables** y se usan entre otras cosas para almacenar temporalmente un dato y manipular valores previamente almacenados.

La sintaxis para declarar una variable, es:

```
identificador [CONSTANT] tipo_dato [NOT NULL]  
[ {:=|DEFAULT} valor];
```

donde:

`identificador` ⇒ es el nombre de la variable.

`CONSTANT` ⇒ restringe la variable para que no se pueda modificar el valor. Las constantes se han de inicializar.

`tipo_dato` ⇒ tipo de dato de la variable (escalar, compuesto, LOB, de referencia).

`NOT NULL` ⇒ restringe la variable para que obligatoriamente tenga un valor. Las variables `NOT NULL` se han de inicializar.

`valor` ⇒ es cualquier expresión PL/SQL que pueda ser una expresión literal, otra variable o una expresión que implique el uso de operadores y funciones.

Ejemplos variables

```
DECLARE

    var1      NUMBER(5);

    radio     NUMBER(3) := 2;

    fecha     DATE;

    cont      NUMBER(2) NOT NULL := 10;      ...

    v_comm    CONSTANT NUMBER := 200;

    pobla     VARCHAR2(15) := 'Cornella';
```

Reglas de nomenclatura

- Dos variables pueden tener el mismo nombres si están en diferentes bloques.
- El nombre o identificador de las variables no deben coincidir con el nombre de las columnas de la tabla o el de las tablas.

Por ejemplo:

```
DECLARE /* Ejemplo ERRONEO, se usa
como identificador de una variable el
mismo identificador que un campo de una
tabla */
```

```
    employee_id NUMBER(2);
BEGIN
    SELECT employee_id
    INTO employee_id
    FROM employees
    WHERE UPPER(last_name) = 'KING';
END;
/
```

```
DECLARE /* Ejemplo CORRECTO */
```

```
    v_employee_id NUMBER(2);
BEGIN
    SELECT employee_id
    INTO v_employee_id
    FROM employees
    WHERE UPPER(last_name) = 'KING';
END;
/
```

Inicializar variables

Para inicializar variables se usa la asignación `:=`. La sintaxis es:

```
identificador := expr;
```

Ejemplo:

```
v_nombre := 'Manel';
```

```
v_fecha := '10-ENE-2010';
```

```
v_sal := 1000;
```

Palabras reservadas

- la palabra reservada DEFAULT
- la restricción NOT NULL

1.3.2.1. Tipos de datos (para declarar variables):

ESCALAR ⇒ Contienen un valor único. Son los tipos más comunes que podemos encontrar en PL/SQL, como son VARCHAR2, NUMBER, DATE, CHAR, BOOLEAN, etc.

COMPUESTA ⇒ Se usan para definir y manipular grupos de campos en bloques PL/SQL. Son las tablas PL/SQL y registros PL/SQL. Este tipo de variables la estudiaremos brevemente.

REFERENCIADA ⇒ Son las variables llamadas también punteros y se usan para designar otros artículos de programa. Este curso no vamos a estudiar dicha variable.

LOB ⇒ Contienen los valores llamados *localizadores*, que especifican la ubicación de imágenes gráficas. Este curso no vamos a estudiar dicha variable.

Tipos de datos escalares

Tipo de dato	Descripción
CHAR [(long_max)]	Carácter de longitud fija hasta 32.767 bytes y no se especifica la longitud, por defecto es 1.
VARCHAR2 [(long_max)]	Carácter de longitud variable hasta 32.767 bytes y no existe un tamaño por defecto para este tipo de dato y tampoco para las constantes.
LONG	Caracteres de variable de longitud variable hasta 32.760 bytes.
LONG RAW	Datos binarios y cadenas de bytes de hasta 32.760 bytes. PL/SQL no interpreta los datos LONG RAW.
NUMBER [(precisión, escala)]	Número con una precisión (que va entre 1 y 38) y una escala que va entre -84 y 127.
BINARY_INTEGER	Enteros.
BOOLEAN	Tipo de dato lógico que almacena uno de los tres valores siguientes: TRUE, FALSE o NULL:
DATE	Fecha y hora.

Ejemplo de variables de datos escalares

```
DECLARE  
  
  v_oficio          VARCHAR2(9);  
  
  v_contador        BINARY_INTEGER := 0;  
  
  v_total           NUMBER(9,2) := 0;  
  
  v_fecha           DATE := SYSDATE + 9;  
  
  v_valido           BOOLEAN NOT NULL := TRUE;  
  
  v_comision         CONSTANT NUMBER(3) := 100;
```

Resumen de tipos de datos

Tipo	Descripción
CHAR(n)	Almacena cadenas de caracteres de longitud fija. Se puede dar opcionalmente la longitud máxima que ha de tener la cadena. Este parámetro se ha de dar entre paréntesis (n).
VARCHAR2(n)	Almacena cadenas de caracteres de longitud variable, cuyo límite se debe especificar en número de bytes (n).
LONG(n)	Almacena cadenas de caracteres de longitud variable. Es parecido al VARCHAR2.
NUMBER(n,m)	Almacena datos numéricos donde <i>n</i> es el número total de dígitos y <i>m</i> es el número de decimales. Tanto <i>n</i> y <i>m</i> son opcionales. PL/SQL dispone de subtipos de NUMBER, que son: DECIMAL, NUMERIC, INTEGER, REAL, SMALLINT, etc.
BINARY_INTEGER	Es un tipo numérico entero que se almacena en memoria en formato binario, con el fin de facilitar los cálculos. Se usa para contadores, índices, etc.
PLS_INTEGER	Es un tipo nuevo y es similar a BINARY_INTEGER; aunque tiene dos ventajas: es más rápido y en caso de desbordamiento en el cálculo, se produce un error y se da la excepción correspondiente.
BOOLEAN	Almacena los valores Verdadero, Falso y Nulo. Las bases de datos no soporta este tipo.
DATE	Almacena fechas y el formato estándar es 'dd-mmm-aaaa'. También almacena la hora.
RAW(n)	Almacena datos binarios en longitud fija y se usa para almacenar cadenas de caracteres evitando las conversiones entre conjuntos de caracteres que realiza Oracle.
LONG RAW	Almacena datos binarios en longitud fija evitando conversiones entre conjuntos de caracteres.
ROWID	Almacena identificadores de fila.

Desenvolupament d'Aplicacions Multiplataforma– M02 Bases de dades

UF3: Llenguatges SQL: DCL i extensió procedimental - NF1: Introducció a SQL submergit -

EA 3.1.1 Introducció a la programació SQL submergit

Versió 1.0 - © M^a Carmen Brito

1.3.2.2. Variables no PL/SQL o variables de HOST o variables de substitució

Cabe destacar que las variables que se definen dentro de un bloque PL/SQL en la zona DECLARE, sea el tipo de bloque que sea, reciben el nombre de **Variables PL/SQL**.

Pero también, podemos tener variables que no se definen dentro de un bloque PL/SQL y reciben el nombre de **Variables no PL/SQL** o **variables de Host** o **variables de substitució**.

Hay dos tipos de definir una variable no PL/SQL:

- a) Definirla mientras se le pregunta al usuario un valor determinado a introducir:

```
ACCEPT nom_variable PROMPT 'literal'
```

- b) Definirla para poder operar con ella (asignarle un valor)

```
VARIABLE nom_variable tipo_dato
```


➤ Referencia a variables no PL/SQL

Ahora nos podemos preguntar, **¿cómo referencia a variables No PL/SQL?**

Para hacer referencia a las variables que no son de tipo PL/SQL (variables HOST o variables de sustitución):

- a) Si la variables se ha definido con la sintaxis (para preguntar al usuario un valor)

```
ACCEPT nom_variable PROMPT 'literal'
```

Se ha de coger el valor que contiene la variable \Rightarrow se ha de preceder del signo & a la variable.

- b) Si la variable se ha definido con la sintaxis

```
VARIABLE <nom_variable> <tipo_variable>
```

Se le quiera asignar un valor \Rightarrow se ha de preceder de dos puntos (:) a la variable.

➤ Impresión de variables no PL/SQL

Para imprimir variables que no son de tipo PL/SQL (variables HOST o variables de sustitución):

- a) Si la variables se ha definido con la sintaxis (para preguntar al usuario un valor)

```
ACCEPT nom_variable PROMPT 'literal'
```

Se puede imprimir (si es necesario) con `DBMS_OUTPUT.PUT_LINE` dentro del bloque PL/SQL, anteponiendo el ampersand (&) antes de la variable.

- b) Si la variable se ha definido con la sintaxis

```
VARIABLE <nom_variable> <tipo_variable>
```

No se puede imprimir dentro del bloque PL/SQL y tampoco se puede usar el `DBMS_OUTPUT.PUT_LINE`. Por tanto, se ha de imprimir fuera del programa y con el comando `PRINT`.

Ejemplo de variables no PL/SQL o variables ligadas

VARIABLE v_salario NUMBER*/* variable no PL/SQL, declarada fuera del programa PL/SQL*/*

BEGIN

SELECT salary

INTO :v_salario/* hacer referencia a una variable no PL/SQL, para escribir un valor en ella */

FROM employees

WHERE employee_id = 100;

END;

/

PRINT v_salario

/ imprimir el resultado de una variable no PL/SQL, no se puede imprimir con DBMS_OUTPUT.PUT_LINE dentro del programa PL/SQL */*

Desenvolupament d'Aplicacions Multiplataforma– M02 Bases de dades

UF3: Llenguatges SQL: DCL i extensió procedimental - NF1: Introducció a SQL submergit -

EA 3.1.1 Introducció a la programació SQL submergit

Versió 1.0 - © M^a Carmen Brito

Introducir Sentencias:

```
VARIABLE v_salario NUMBER
BEGIN
  SELECT sal
  INTO :v_salario
  FROM emp
  WHERE empno = 7839;
END;
/
PRINT v_salario
```

Ejecutar

Guardar Archivo de Comandos

Procedimiento PL/SQL terminado correctamente.

V_SALARIO
5000

Ejemplo de variables no PL/SQL o variables ligadas

```
ACCEPT var_emp PROMPT 'Introduce el código del empleado:'  
  
VARIABLE v_salario NUMBER  
  
BEGIN  
  
  SELECT salary  
  INTO :v_salario /* escribe en una variable no PL/SQL */  
  
  FROM employees  
  
  WHERE employee_id= &var_emp; /* var_deptno es una variable de sustitución,  
                                que se declara en el ACCEPT cuando se le pregunta al usuario por el  
                                valor */  
  
END;  
  
/  
  
PRINT v_salario
```

```
bloque anónimo terminado  
V_SALARIO  
-----  
24000
```

1.3.3. Constantes

Las **constantes** se han de declarar mediante la siguiente sintaxis:

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

Por ejemplo:

```
DECLARE
```

```
    IVA CONSTANT    NUMBER := 16;
```

```
    pi CONSTANT     NUMBER(9,7) := 3.1415927;
```

```
    ...
```

1.3.4. Los atributos %TYPE y %ROWTYPE

La idea de usar estos atributos es declarar variables que sean del mismo tipo que otros objetos que ya están definidos. Es decir, hereda de la variable el tipo y la longitud del objeto, pero no los atributos NOT NULL, ni valores por defecto.

Estos atributos son el %TYPE y el %ROWTYPE, vamos a estudiar cada uno de ellos:

%TYPE:

Este atributo declara una variable del mismo tipo que otra, o que una columna de una tabla. Por tanto, declara una variable basada en otras variables previamente declaradas, o en la definición de una columna de la base de datos.

Por ejemplo, si queremos declarar una variable llamada **var_nombre** que sea del mismo tipo que el **nombre de un empleado** (`first_name`) que tenemos en nuestra base de datos, concretamente en la tabla **empleado** (`employees`).

Tenemos la siguiente sintaxis dentro de un bloque PL/SQL:

```
DECLARE

    var_nombre      employees.first_name%TYPE;

    ...
```

%ROWTYPE:

Este atributo crea una variable de registro cuyos campos se corresponden con las columnas de una tabla o vista de la base de datos.

En esta unidad didáctica estudiaremos con más detalle este atributo.

Por ejemplo, si declaras una variable que tenga la estructura de la tabla **empleado** (EMPLOYEES), seria:

```
DECLARE  
  
    var_empleado    employees%ROWTYPE;  
  
    ...
```


1.3.5. Ejemplos de variables

Recordemos que, para que el resultado de los programas en PL/SQL se puedan observar por pantalla, se ha de ejecutar primero la librería:

```
SET SERVEROUTPUT ON
```

Ejemplo1:

```
SET SERVEROUTPUT ON
DECLARE                                -- Bloque padre
    var1 NUMBER;
BEGIN
    var1 := 10;
    DBMS_OUTPUT.PUT_LINE('El valor de la variable1 es
' || var1);
    DECLARE                            -- Bloque hijo
        var2 NUMBER;
    BEGIN
        var2:=20;
        DBMS_OUTPUT.PUT_LINE('El valor de la variable2 es
' || var2);
        var2:=var1;
        DBMS_OUTPUT.PUT_LINE('El valor de la variable2 es
' || var2);
    END;                               -- Fin bloque hijo
END;                                   -- Fin bloque padre
/
```

El valor de la variable1 es 10
El valor de la variable2 es 20
El valor de la variable2 es 10

Ejemplo2:

```

SET SERVEROUTPUT ON
DECLARE                                -- Bloque padre
    var1 NUMBER;
BEGIN
    var1 := 10;
    DBMS_OUTPUT.PUT_LINE('El valor de la variable1 es
' || var1);
    DECLARE                            -- Bloque hijo
        var2 NUMBER;
    BEGIN
        var2:=var1;
        DBMS_OUTPUT.PUT_LINE('El valor de la variable2 es
' || var2);
    END;                               -- Fin bloque hijo
var2:=var1;                            -- ERROR: var2 no se conoce en este ámbito, puesto que
                                       -- es local en el bloque hijo y se ha finalizado
END;                                   -- Fin bloque padre
var2:=var1;
*
```

ERROR en línea 12:
ORA-06550: línea 12, columna 1:
PLS-00201: el identificador 'VAR2' se debe declarar
ORA-06550: línea 12, columna 1:
PL/SQL: Statement ignored

Ejemplo3:

```
SET SERVEROUTPUT ON
<<bloquepadre>>      -- Etiqueta que identifica el bloque padre
DECLARE               -- Bloque padre
    var1 NUMBER;
BEGIN
    var1 := 10;
    DBMS_OUTPUT.PUT_LINE('El valor de la variable1 es
' || var1);
    DECLARE           -- Bloque hijo
        var2 NUMBER:=20;
    BEGIN
        var2:=bloquepadre.var1; /* se asigna a una variable local (del
                                bloque hijo) una variable que se encuentra
                                declarada en el bloque padre */
        DBMS_OUTPUT.PUT_LINE('El valor de la variable2 es
' || var2);
    END;               -- Fin bloque hijo      El valor de la variable1 es 10
END;                  -- Fin bloque padre     El valor de la variable2 es 10
/
```

Ejemplo4:

```
-- Variables NO PL/SQL
VARIABLE nombre VARCHAR2(10)
VARIABLE edad NUMBER
SET SERVEROUTPUT ON
DECLARE
    var_nombre VARCHAR2(10):='JOSE LOPEZ';
    var_edad NUMBER:=20;
BEGIN
-- Asignación a variables No PL/SQL el valor de variables PL/SQL
    :nombre:=var_nombre;
    :edad:=var_edad;
END;
/
print nombre
print edad
/
```

NOMBRE

JOSE LOPEZ

EDAD
--
20

Los datos del empleado son:

CODIGO: 100

NOMBRE: Steven

SALARIO: 24000

Ejemplo5:

```
SET SERVEROUTPUT ON
ACCEPT codigo PROMPT 'Introduce el codigo del empleado a
consultar'
DECLARE
    var_empno employees.employee_id%TYPE;
    var_name   employees.first_name%TYPE;
    var_sal    employees.salary%TYPE;
BEGIN
    SELECT employee_id, first_name, salary
    INTO var_empno, var_name, var_sal
    FROM employees
    WHERE employee_id = &codigo;
    DBMS_OUTPUT.PUT_LINE('Los datos del empleado son: ');
    DBMS_OUTPUT.PUT_LINE('CODIGO: ' || var_empno);
    DBMS_OUTPUT.PUT_LINE('NOMBRE: ' || var_name);
    DBMS_OUTPUT.PUT_LINE('SALARIO: ' || var_sal);
END;
/
```

Desenvolupament d'Aplicacions Multiplataforma– M02 Bases de dades

UF3: Llenguatges SQL: DCL i extensió procedimental - NF1: Introducció a SQL submergit -

EA 3.1.1 Introducció a la programació SQL submergit

Versió 1.0 - © M^a Carmen Brito

1.3.6. Variables de sistema

`SET ECHO {ON | OFF}` \Rightarrow Por defecto el ECHO está activado y muestra las instrucciones SQL que se van ejecutando. Para desactivarlo tenemos la cláusula OFF.

`SET VERIFY {ON | OFF}` \Rightarrow Imprime los valores de las variables de sustitución e imprime dos líneas para cada variable. Por defecto está activada.

Y además, hemos de tener en cuenta

`SET SERVEROUTPUT ON`

para que se pueda visualizar los resultados mostrados con `DBMS_OUTPUT.PUT_LINE`. Por defecto esta variable está a OFF.

1.4. Tipos de datos compuestos

Hasta ahora hemos estudiado las variables de tipo escalar, y ahora vamos a estudiar brevemente las variables compuestas, que son los **registros PL/SQL** y las **tablas PL/SQL**.

Registros PL/SQL

Son un grupo de elementos de datos relacionados en campos, cada uno con su propio nombre y tipo de datos; es decir, tratan una colección de campos como una unidad lógica.

La sintaxis para crear un registro PL/SQL, es la siguiente:

```
TYPE nombre_registro IS RECORD  
    (declaración_campo [,declaración_campo])
```

Es importante destacar que los registros no son los mismo que las filas de una tabla de una base de datos.

Ejemplo de registros PL/SQL:

- Creación de un registro usando tipos de variables escalares:

```
...
TYPE tipo_registro_empleado IS RECORD
  (employee_id  NUMBER(4),
   first_name    VARCHAR2(10),
   job_id        VARCHAR2(9),
   salary        NUMBER(7,2));
registro_empleado  tipo_registro_empleado;
...
```
- Creación de un registro usando el atributo %TYPE:

```
...
TYPE tipo_registro_empleado IS RECORD
  (employee_id  employees.employee_id%TYPE,
   first_name    employees.first_name%TYPE,
   job_id        employees.job_id%TYPE,
   salary        employees.salary%TYPE);
registro_empleado  tipo_registro_empleado;
...
```

Estructura del registro de nuestro ejemplo será:

Campo1 (tipo de datos)	Campo2 (tipo de datos)	Campo3 (tipo de datos)	Campo4 (tipo de datos)
department_id NUMBER(4)	first_name VARCHAR2(10)	job_id VARCHAR2(9)	salary NUMBER(7,2)

Esta estructura es similar a trabajar con el atributo %ROWTYPE :

```
DECLARE
```

```
var_emp employees%ROWTYPE;
```



LOS DATOS DEL EMPLEADO QUE HA CONSULTADO SON:

CODIGO EMPLEADO: 100

NOMBRE EMPLEADO: Steven

SALARIO EMPLEADO: 24000

```
SET SERVEROUTPUT ON
SET VERIFY OFF
SET ECHO OFF
ACCEPT codigo PROMPT 'Introduce el código del empleado a consultar: ';
DECLARE
    variable employees%ROWTYPE;
BEGIN
    SELECT *
    INTO variable
    FROM employees
    WHERE employee_id = &codigo;
    DBMS_OUTPUT.PUT_LINE ('LOS DATOS DEL EMPLEADO QUE HA CONSULTADO
SON: ');
    DBMS_OUTPUT.PUT_LINE ('CODIGO EMPLEADO: ' || variable.employee_id);
    DBMS_OUTPUT.PUT_LINE ('NOMBRE EMPLEADO: ' || variable.first_name);
    DBMS_OUTPUT.PUT_LINE ('SALARIO EMPLEADO: ' || variable.salary);
END;
/
UNDEFINE codigo
SET VERIFY ON
SET ECHO ON
```

Tablas PL/SQL

Son tipos de datos compuestos y están estructurados como tablas de la base de datos, aunque no se ha de confundir con las propias tablas donde se almacenan información en la base de datos

Clave Principal

...
1
2
3
...

BINARY_INTEGER

Columna

...
Juan
María
José
...

Dato Escalar

La sintaxis para crear una tabla PL/SQL es la siguiente:

```
TYPE tipo_nombre_tabla IS TABLE OF  
    {tipo_campo | variable%TYPE |  
        nombretabla.campo%TYPE} [NOT NULL]  
    INDEX BY BINARY_INTEGER;  
  
identificador    tipo_nombre_tabla;
```

Ejemplo: en este ejemplo vamos a trabajar con tablas PL/SQL e indicaremos los tres pasos a efectuar:

1. Crear una tabla (física) donde se almacenará el nombre y el salario de los empleados. Asegurarse de que la tabla no existe y si existe, eliminarla.
2. Declarar dos tablas PL/SQL: `tabla_nombre` (para almacenar el nombre del empleado o empleados) y `tabla_salario` (para almacenar el salario).
3. Almacenar los datos en las tablas, para insertarlo posteriormente en la tabla nueva creada. Asegurarse de que la tabla está vacía.

1. Crear una tabla (física de la base de datos) donde se almacenará el nombre y el salario de los empleados. Asegurándose de que la tabla no existe y si existe, eliminarla.

```
-- ELIMINAR TABLA  
DROP TABLE empleado_salario;  
  
-- CREAR TABLA  
CREATE TABLE empleado_salario  
  (nombre VARCHAR(25),  
   salario NUMBER(8,2));
```

2. Declarar dos tablas PL/SQL: tabla_nombre (para almacenar el nombre del empleado o empleados) y tabla_salario (para almacenar el salario).

```
--BLOQUE PL/SQL
SET VERIFY OFF
SET ECHO OFF
ACCEPT var_codigo PROMPT 'Introduce el codigo del empleado: '

DECLARE                                /* Declaración de variables y tablas PL/SQL */
    TYPE tipo_tabla_nombre IS TABLE OF VARCHAR2(20)
        INDEX BY BINARY_INTEGER;

    TYPE tipo_tabla_salario IS TABLE OF NUMBER(8,2)
        INDEX BY BINARY_INTEGER;

    var_ename          employees.first_name%TYPE;
    var_sal             employees.salary%TYPE;
    tabla_nombre        tipo_tabla_nombre;
    tabla_salario        tipo_tabla_salario;
    i BINARY_INTEGER := 0; /*Posición de la tabla PL/SQL*/
```


3. Almacenar los datos en las tablas, para insertarlo posteriormente en la tabla nueva creada.
Asegurarse de que la tabla está vacía.

```
BEGIN    -- Inicio del programa PL/SQL
-- Eliminar contenido de la tabla empleado_salario
DELETE FROM empleado_salario;

-- Consultar los datos del empleado a consultar
SELECT first_name, salary
INTO var_ename, var_sal
FROM employees
WHERE employee_id = &var_codigo;

-- Llenar las tablas PL/SQL con los datos correspondientes
del empleado
tabla_nombre(i) := var_ename;
tabla_salario(i) := var_sal;
```

```
-- Insertar en la tabla física de la base de datos,  
-- los datos que contiene las tablas PL/SQL  
INSERT INTO empleado_salario (nombre, salario)  
VALUES (tabla_nombre(i), tabla_salario(i));  
  
-- Guardar los cambios en la base de datos  
COMMIT;  
  
-- Mostrar mensajes correspondientes al usuario  
DBMS_OUTPUT.PUT_LINE ('SE HA CREADO LA TABLA  
CORRECTAMENTE');  
DBMS_OUTPUT.PUT_LINE ('VAMOS A COMPROBARLO ....');  
END;  
/
```

```
-- Fuera del bloque PL/SQL:
-- Consulta para comprobar que se han insertado en la
  tabla física de la base de datos

SELECT *
FROM empleado_salario;

-- Se limpia la variable no PL/SQL var_codigo y se pone
  a ON el SET VERIFY y SET ECHO.
UNDEFINE var_codigo
SET VERIFY ON
SET ECHO ON
```

NOMBRE	SALARIO
-----	-----
Steven	24000

Preguntes!!!!



Desenvolupament d'Aplicacions Multiplataforma— M02 Bases de dades
UF3: Llenguatges SQL: DCL i extensió procedimental - NF1: Introducció a SQL submergit -
EA 3.1.1 Introducció a la programació SQL submergit
Versió 1.0 - © M^a Carmen Brito