PL/SQL

Antonio Espín Herranz

Fundamentos

- Introducción
- Tipos de datos
- Operadores
- Estructuras de Control
- Bloques

Fundamentos de PL/SQL

- PL/SQL es el lenguaje procedimental de Oracle. Al igual que otros gestores de BD tienen su lenguaje (SQL Server → Transact SQL), Oracle dispone de PL/SQL.

- PL/SQL no es CASE-SENSITIVE, es decir, no diferencia mayúsculas de minúsculas como otros lenguajes de programación como C o Java. **Sin embargo debemos recordar que ORACLE es CASE-SENSITIVE en la búsquedas de texto.**

Fundamentos de PL/SQL

- Dentro de este lenguaje se pueden mezclar la instrucciones de SQL con el lenguaje procedimental, pudiendo utilizar sentencias de control, cursores, excepciones, transacciones, etc.
- Nos permite escribir bloques de código y grabarlos en ficheros (Scripts).
- Las **sentencias** de **PL/SQL terminan** en **;** a no ser que sean una instrucción de bloque.

Comentarios

- Dentro de los scripts podemos utilizar comentarios de dos tipos:
  - De línea: **-- Línea comentada**
  - De Bloque: **/* esto es un bloque**

    comentado ***/**

- Utilidad, para aclarar y hacer anotaciones dentro del código.

Declaración de Variables

- Se pueden definir en mayúsculas y minúsculas. No se tiene en cuenta.
- Hay que declararlas antes de utilizarlas y es obligatorio indicar el tipo.
- A las variables se les puede asignar el resultado de un SQL, u otra variable o un valor constante.
- El nombre: 30 char. Máximo, letras, números y los chars: $ _ #.

Declaración de Variables

- La declaración de variables puede ir dentro de un bloque de código: clausula **Declare**.
  - provincia number:=32;
  - Se asigna un valor con **:=**
- En **SQL-Developer** se utilizan así:

```
set serveroutput on;
declare
  numero number;
Begin
   numero:=0;
  select count(*) into numero from tbproductos;
  dbms_output.put_line('Número de registros : ' || numero);
  end;
/
```

Declaración de Variables

- La sintaxis general para declarar una variable (*también se pueden definir contantes*):
- nombre_variable [**CONSTANT**] <*tipo_dato*> [**NOT NULL**][:=valor_inicial]

- /* Se declara la variable de tipo VARCHAR2(15) identificada por v_location y se le asigna el valor "Granada"*/
  v_location **VARCHAR2**(15) := 'Granada';

- /*Se declara la constante de tipo NUMBER identificada por PI y se le asigna el valor 3.1416*/
  PI **CONSTANT NUMBER** := 3.1416;

- /*Se declara la variable del mismo tipo que tenga el campo nombre de la tabla tabla_empleados identificada por v_nombre y no se le asigna ningún valor */
  v_nombre tabla_empleados.nombre**%TYPE**;

- /*Se declara la variable del tipo registro correspondiente a un supuesto cursor, llamado micursor, identificada por reg_datos*/
  reg_datos micursor**%ROWTYPE**;

Declaración de Variables

- **tipo_dato:** es el tipo de dato que va a poder almacenar la variable, este puede ser cualquiera de los tipos soportandos por ORACLE, es decir **NUMBER** , **DATE** , **CHAR** , **VARCHAR, VARCHAR2, BOOLEAN ...** Además para algunos tipos de datos (NUMBER y VARCHAR) podemos especificar la longitud.

- **La cláusula CONSTANT** indica la definición de una constante cuyo valor no puede ser modificado. Se debe incluir la inicialización de la constante en su declaración.

- **La cláusula NOT NULL** impide que a una variable se le asigne el valor nulo, y por tanto debe inicializarse a un valor diferente de NULL.

Declaración de Variables

- Las variables que no son inicializadas toman el **valor inicial es NULL**.

- **La inicialización puede incluir cualquier expresión legal de PL/SQL**, que lógicamente debe corresponder con el tipo del identificador definido.

- **Las variables** se puede definir de **tipos escalares o tipos mas complejos** (registros, etc.)

Declaración de Variables

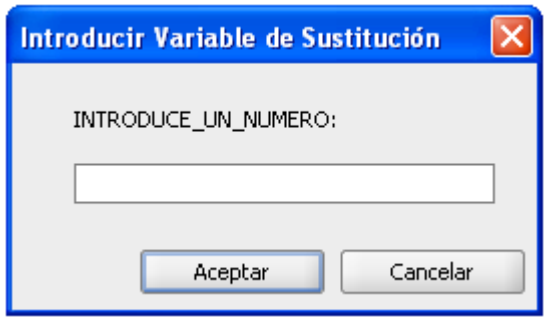
- Podemos definir el tipo de la variable en función del tipo de otra variable o de una columna de una tabla → **%TYPE**.

- También se puede definir una variable en función del tipo de una tabla, de tal forma que la variable contiene todos los campos de la tabla y almacena el valor de una fila → **%ROWTYPE**.
  - Esta se suele utilizar para definir cursores.

Lectura de Variables

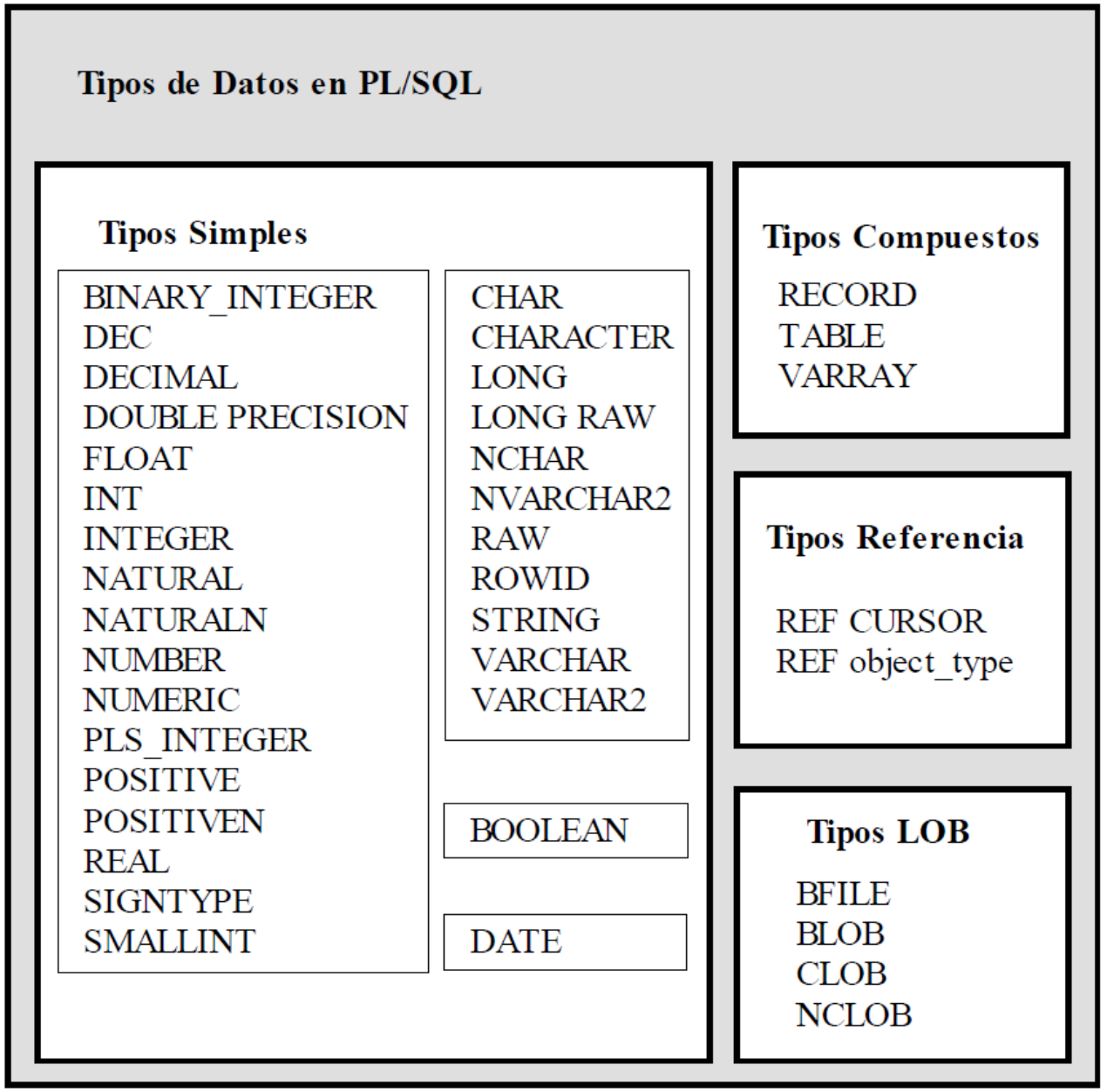
- Se pueden solicitar variables a partir del teclado utilizando un **parámetro**.

```
Declare
      Num number;
Begin
      Num:= &introduce_un_numero;
      -- Trabajar con el número …
End;
/
```



- Si la variable es de tipo texto, el valor de sustitución debe ir entre comillas.
- Nombre:='&introduzca_nombre';

Tipos


Tipos de Datos en PL/SQL
Tipos Simples
BINARY_INTEGER
DEC
DECIMAL
DOUBLE PRECISION
FLOAT
INT
INTEGER
NATURAL
NATURALN
NUMBER
NUMERIC
PLS_INTEGER
POSITIVE
POSITIVEN
REAL
SIGNTYPE
SMALLINT
CHAR
CHARACTER
LONG
LONG RAW
NCHAR
NVARCHAR2
RAW
ROWID
STRING
VARCHAR
VARCHAR2
BOOLEAN
DATE
Tipos Compuestos
RECORD
TABLE
VARRAY
Tipos Referencia
REF CURSOR
REF object_type
Tipos LOB
BFILE
BLOB
CLOB
NCLOB

Tipos predefinidos

- **Tipos de caracteres:**
  - VARCHAR2 [(n)]. Para textos de longitud variable de hasta 4000 caracteres
  - CHAR [(n)]. Para textos de longitud fija de hasta 2000 caracteres.
  - NCHAR [(n)]. Para el almacenamiento de caracteres nacionales de texto fijo
  - NVARCHAR2[(n)]. Para el almacenamiento de caracteres nacionales de longitud variable.
  - LONG: Cadena de longitud variable. Máximo hasta 32.760 bytes.
  - RAW [(n)]. Cadena de caracteres o datos binarios entre 1 y 32767.

- Si no se indica el tamaño se supone 1.
- **Normalmente se suele utilizar VARCHAR2, ya que si el espacio indicado no se ocupa, Oracle lo aprovecha.**

Tipos predefinidos

- **Tipos numéricos:**
  - **NUMBER**[(p,s)]:
    - Almacena números enteros o de punto flotante, virtualmente de cualquier longitud, aunque puede ser especificada la precisión (Número de digitos) y la escala que es la que determina el número de decimales.
    - -- Ejemplo: saldo **NUMBER**(16,2);
      /* Indica que puede almacenar un valor numérico de 16 posiciones, 2 de ellas decimales. Es decir, 14 enteros y dos decimales */
  - INTEGER, FLOAT:
    - Enteros y reales.
  - BINARY_INTEGER, PLS_INTEGER:
    - Números comprendidos entre -2.147 M a 2.147 M.
    - Necesitan menos espacio que el number y realizan cálculos más rápidos.

Tipos predefinidos

- **Tipos para objetos de gran tamaño:**
  - **BFILE**:
    - Permite almacenar la referencia a un archivo del sistema operativo que contiene datos físicos.
  - **BLOB**:
    - Almacenar un objeto en binario.

- **Otros tipos de datos:**
  - **BOOLEAN**:
    - Valores TRUE, FALSE o NULL.
  - **DATE**:
    - Almacena fechas del 1 de enero de 4712 a.c. al 31/12/9999 d.c.
  - **TIMESTAMP:**
    - Almacena valores de día, mes y año, junto con hora, minuto y segundos (incluso con decimales). Con lo que representa un instante concreto en el tiempo.

Tipos definidos por el usuario

- PL/SQL permite definir nuevos tipo.
- La función es clarificar el programa.
- Se apoya en los tipos ya conocidos.
- Dentro de estos, se pueden definir tipos que agrupen varias variables mediante el mismo nombre (record).
- **Sintaxis**: (*se declaran en la sección declare*).
  - **SUBTYPE nombre_subtipo IS Tipo**;

Ejemplo

```
Declare
    -- Declarar un tipo basado en el tipo Date:
        Subtype fechaNacimiento is date;
    -- Declarar una variable de ese tipo.
        Nacido_el fechaNacimiento;
Begin
    --Poner la fecha del sistema:
        Nacido_el := sysdate;
End;
/
```

Tipos derivados

- **%TYPE**
  - Hace referencia a una columna de una tabla o una variable que fue declarada con anterioridad.
    - Tendrá el mismo tipo.
  - Sintaxis:

    -- Referenciando a una columna de una tabla:

    Nombre_variable nombre_tabla.nombre_col%TYPE;

    -- Referenciando a otra variable previamente declarada:

    Nombre_variable2 nombre_variable1%TYPE;

    EJEMPLO:

```
set serveroutput on;
declare
  x number;
  y x%TYPE;
begin
  y:=100;
  dbms_output.put_line('y=' || y);
end;
/
```

Tipos derivados

- **%ROWTYPE**
  - Hace referencia a una estructura completa de tabla o cursor. La variable tiene la misma estructura que la tabla o cursor.
  - Sintaxis:
    - Nombre-de-variable {nombre_tabla | nombre_cursor} %ROWTYPE;
  - Ejemplo:

```
set serveroutput on;
declare
 fila personas%rowtype;

Begin -- Ojo, si devuelve mas de una fila dará una excepción.
 select * into fila from personas where nombre='Jose';
 dbms_output.put_line(fila.nombre || ' ' || fila.edad);
end;
/
```

Operadores

- Tabla de operadores de PL/SQL:

Tipo de operador	Operadores
Operador de asignación	:= (dos puntos + igual)
Operadores aritméticos	+ (suma) - (resta) * (multiplicación) / (división) ** (exponente)
Operadores relacionales o de comparación	= (igual a) <> (distinto de) < (menor que) > (mayor que) >= (mayor o igual a) <= (menor o igual a)
Operadores lógicos	AND (y lógico) NOT (negacion) OR (o lógico)
Operador de concatenación	||

Expresiones

- Podemos utilizar variables, valores constantes y el conjunto de operadores.
- A la hora de escribir expresiones podemos utilizar paréntesis, para establecer prioridades.
- Ejemplos:

```
x : = 0;
Vnombre:= ‘Señor’ || Vnombre;
y := (x + 5) * y;
```

Valores constantes

- **Cadenas y caracteres**:
  - Entre comillas simples: ‘esto es una cadena’.
- **Números enteros**:
  - Positivos y negativos: -3, +6, 4.
- **Número reales**:
  - El punto separador decimal: 5.7, -8.8, 25e-04
- **Fechas**:
  - Entre comillas: ’31/05/1970’, ’30-06-1952’, ‘09072010’
- **Boolean**:
  - True, False.
- **NULL**:
  - No tiene valor.

Sentencias de control

- Condicionales:
  - **If**: Permite anidamientos, else, etc.
  - **Case**: Evaluación múltiple. Mas cómodo de leer y mejora el rendimiento.

- Bucles:
  - **Loop**
  - **For**
  - **While**

Sentencia IF

If cond then procesamiento1;

[Elsif cond then procesamiento2;]

Else

Procesamiento3;

End if;

- Permite **elsif** anidados y prescindir de la parte Else.
- Los operadores para montar las condiciones son los mismos que en SQL:

  =, >, <, !, >=, <=, is null, is not null, between, like, and, or.

Ejemplo

```
If varnumcli = 10 then
    Update clientes set nombre='Juan' where numcli = varnumcli;
    Commit;
Else
    Rollback;
End if;
```

Sentencia Case

- Nos permite la evaluación múltiple.
- Se puede realizar la evaluación por valor o por condición.
- Nos permite añadir una clausula else por si no se cumple ninguno de los casos.
- Sintaxis: **Evaluando por valor.**
  - [<<etiqueta>>]
  - **Case** elemento
    - When valor1 then instrucciones1;
    - When valor2 then instrucciones2;
    - [Else instrucciones;]
  - **End Case** [etiqueta];

Sentencia Case

- Sintaxis **evaluando por condición**:
  - [<<Etiqueta>>]
  - **Case**
    - When condición1 then instrucciones1;
    - When condición2 then instrucciones2;
    - [Else instrucciones];
  - **End case** [etiqueta];

Ejemplos

- **Evaluación por valor:**

```
declare
 provincia number:=32;
 nombre VARCHAR2(40);

 begin
  case provincia
   when 32 then nombre:='Orense';
   when 36 then nombre:='Pontevedra';
   else nombre:= 'fuera de la comunidad';
  end case;

 end;
 /
```

- **Evaluación por condición:**

```
declare
  provincia number:=32;
  comunidad VARCHAR2(40);
 begin
  case
   when provincia in (02,13,16,19,45) THEN
    comunidad:='Castilla la mancha';
   when provincia in (04,11,14,18,21,23,29,41) then
    comunidad:='Andalucía';
   else
    comunidad:='Otra comunidad';
   end case;
 end;
 /
```

Sentencia Case

- En caso de que no se añada la sentencia else, Oracle lo añade de forma implícita de tal forma que si la evaluación no se ajusta a ningún caso se producirá una excepción.
  - **ELSE Raise Case_Not_Found**

- **La sentencia case se puede utilizar dentro del SQL.**

Bucle Loop

- Se trata de la variedad más simple de la sentencia LOOP, y se corresponde con el bucle básico (o infinito), el cual incluye una secuencia de sentencias entre las palabras claves LOOP y END LOOP.
- Ejemplo:
  - *LOOP*
    - *secuencia_de_sentencias;*
  - *END LOOP;*

Bucle Loop

- Tenemos dos formas de salir del bucle:
  - EXIT y EXIT WHEN.

- La sentencia **EXIT** provoca la **salida** de un bucle de forma **incondicional**. Cuando se encuentra un EXIT, el bucle acaba inmediatamente y el control pasa a la siguiente instrucción que esté fuera del bucle.

- Ejemplo:
  - LOOP
    - …
    - IF limite_credito < 3 THEN
    - …
    - EXIT; -- Fuerza la salida inmediata…
    - END IF;
  - END LOOP;

Bucle Loop

- La sentencia **EXIT-WHEN**, nos va a permitir salir de un bucle de forma condicional. Cuando PL/SQL encuentra una sentencia de este tipo, la condición del WHEN será evaluada… en caso de devolver TRUE, se provocará la salida del bucle… en caso contrario, se continuará la iteración.

- Ejemplo:

```
LOOP
   …
      EXIT WHEN contador>100;
END LOOP
```

Bucles con etiquetas

- Al igual que los bloques de PL/SQL, los bucles pueden ser etiquetados. La etiqueta es un identificador no declarado que se escribe entre los símbolos << y >>; deben aparecer al principio de las sentencias LOOP.

```
<<mi_loop>>
LOOP
        secuencia_de_sentencias;
END LOOP mi_loop;
```

- Con las etiquetas podemos forzar la salida de bucles internos, salir hasta el bucle mas externo.

Bucles con etiquetas

```
<<exterior>>
LOOP
  …
  LOOP
        …
        EXIT exterior WHEN …
        -- Sale de los dos bucles LOOP…
  END LOOP;
  …
END LOOP exterior;
```

Sentencia Continue

- Esta sentencia permite interrumpir la iteración en curso y pasar directamente a la siguiente.
- Sintaxis:

  Continue [label] [when condición]
- Ejemplo: insertar los clientes impares:

```
x:=0
<<incremento>>
loop
    x:=x+1
    exit incremento when x > 10
    continue incremento when mod(x, 2) =0;
    -- Cuando se cumple la condición esta línea no se ejecuta:
    insert into clientes (numerocliente)  values (x);
end loop incremento;
commit;
```

Bucle For

- Permite ejecutar las instrucciones dentro del bucle haciendo variar un índice.
- Las instrucciones se ejecutan tantas veces como cambia el índice de valor.
- El índice se puede utilizar dentro de las instrucciones pero siempre en modo lectura.

Bucle For

- Sintaxis:

  <<etiqueta>>

  For indice in [reverse] exp1…exp2 loop

    Instrucciones

    …

  End loop [etiqueta]

  - El índice se declara de forma implícita.
  - Exp1, exp2 son constantes, expresiones o variables.
  - Si no se utiliza reverse el índice avanza de exp1 a exp2 de uno en uno.
  - Si se utiliza reverse, el índice varía de exp2 a exp1 con incrementos de -1.

Bucle For

- Ejemplo:

```
For n in 100..110 loop
    Insert into clientes (numcli) values (n);
End loop;
Commit;
```

- El bucle se ejecuta 10 veces, y se ejecuta el sql.

Bucle While

- La evaluación del bucle se realiza al inicio.
- Si la condición es true el bucle se ejecuta.
- Sintaxis:
  - <<etiqueta>>
  - While condición loop
    - Instrucciones;
    - …
  - End loop [etiqueta];
- En la condición se pueden utilizar todos los operadores: <,>,=,<>, AND, OR, LIKE, …

Ejemplo

```
x:= 1;
while x <= 10 loop
    insert into clientes (numcli) values (x);
    x := x + 1;
end loop;
commit;
```

- Crea clientes del 1 al 10, ambos incluidos.

Bloque anónimos de PL/SQL

- Los bloques de PL/SQL representan un conjunto de instrucciones que se ejecutan de una vez.
- El bloque se compila y se ejecuta.
- En bloque tendremos declaraciones de variables, instrucciones PL/SQL, sentencias SQL y podremos tener tratamiento de excepciones.

Estructura del bloque

- DECLARE
  - (Declaraciones de variables, constantes, excepciones y cursores).
    - Si no necesitamos variables no es necesario.
  - BEGIN
    - Instrucciones SQL, PL/SQL, estructuras de control.
  - EXCEPTION
    - (tratamiento de errores).
  - END;

Estructura del bloque

- Las etiquetas (label), con el formato <<nombreetiqueta>>, permiten marcar ciertas partes del código.
- Ejemplo:

```
x:=0
<<incremento>>
Loop
        x:=x+1
        Exit incremento when x > 10;
        Insert into …
End Loop incremento;
Commit;
```

Sentencias SQL dentro de PL/SQL

```
DECLARE
        codigo EMPLEADOS.CODIGO EMPLEADO%TYPE;
        nombre EMPLEADOS.NOMBRE%TYPE;
        primerApellido EMPLEADOS.APELLIDO1%TYPE;
        segundoApellido EMPLEADOS.APELLIDO2%TYPE;
        maximo EMPLEADOS.CODIGOEMPLEADO%TYPE;
BEGIN
        codigo:=1;
        SELECT Max(CodigoEmpleado) INTO maximo FROM EMPLEADOS;
        FOR indice IN 1..maximo LOOP
                SELECT codigoEmpleado, nombre, apellido1, apellido2
                INTO codigo, nombre, primerApellido, segundoApellido
                FROM EMPLEADOS WHERE codigoEmpleado=indice;
                INSERT INTO LISTADO
                VALUES (codigo, nombre, primerApellido, segundoApellido);
                DBMS OUTPUT.PUT LINE('La tupla se ha insertado');
        END LOOP;
END;
```