

Administración de Bases de Datos

Operación y Mantenimiento

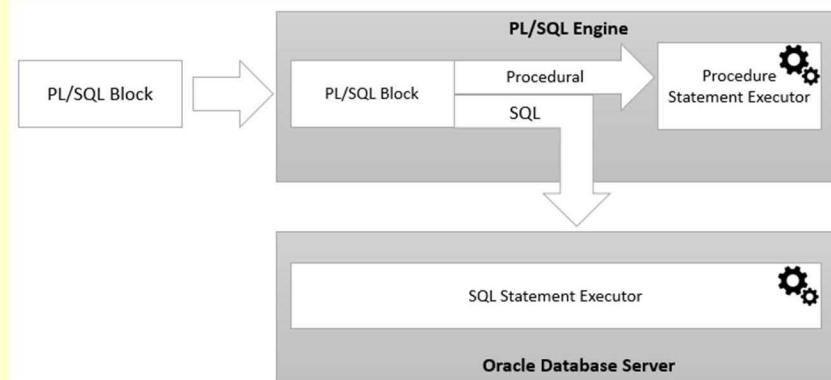
Introducción al PL/SQL

Se llama PL/SQL al lenguaje procedural que complementa y agrega diversos constructos procedurales al SQL, para superar las limitaciones de éste, proporcionando soluciones a la construcción de aplicaciones críticas en las bases de datos.

PL/SQL es un lenguaje estructurado muy comprensible, sus constructos buscan la claridad y es bastante fácil de aprender. Es un lenguaje necesario para desarrollar bases de datos en Oracle y su estándar permite la portabilidad a otras bases de datos compatibles con Oracle. Pero no está diseñado para ser ejecutado fuera de Oracle, pero si permite la inclusión de código en otros lenguajes de programación.

Arquitectura de PL/SQL

El motor de PL/SQL compila el código de los programas a código binario que puede ser ejecutado por la base de datos Oracle.



El servidor de la base de datos recibe el bloque PL/SQL y ejecuta el código mediante el motor de PL/SQL, que ejecuta los elementos procedurales y transmite las sentencias SQL al motor de SQL.

Bloques de PL/SQL

PL/SQL es un lenguaje estructurado en bloques, cada bloque consiste de tres secciones:

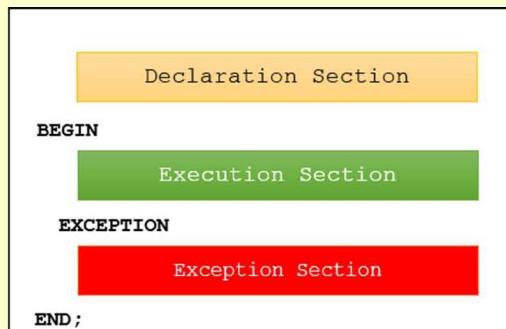
Declaraciones
Código Ejecutable
Manejo de Excepciones

En un bloque, la sección de código ejecutable es la prioritaria, mientras que las otras dos secciones son opcionales.

Los bloques de código PL/SQL tienen un nombre, como es el caso de las funciones y procedimientos. Los bloques con nombre pueden ser almacenados como parte de la base de datos y utilizados posteriormente.

Un bloque anónimo es aquel que no tiene nombre y, por ello, no puede ser guardado en el servidor de la base de datos, solo se crea para un solo uso y suelen ser útiles para propósitos de prueba.

La siguiente figura muestra la estructura de un bloque PL/SQL:



Sección de Declaraciones

En esta sección se declaran variables, se definen tipos de datos y se asigna memoria para los cursos.

Sección de código ejecutable

Esta sección siempre inicia con la palabra BEGIN y termina con la palabra END. Debe contener al menos una instrucción, incluso si se trata de la sentencia NULL que no hace nada.

Sección de manejo de Excepciones

Esta tercera sección inicia con la palabra EXCEPTION y es la sección donde se manejan todas las excepciones que se producen en la ejecución del código.

Considere que un bloque es en sí una sentencia ejecutable, por lo que resulta lógico poder anidar bloques dentro de otros bloques.

Ejemplo simple

El siguiente ejemplo muestra un bloque anónimo formado únicamente por la sección de código ejecutable, con la que se envía el mensaje "Hola Mundo." A la pantalla.

```

BEGIN
    DBMS_OUTPUT.put_line ('Hola Mundo.');
END;
/
  
```

Para poder ver la respuesta en pantalla, previamente se deberá ejecutar la

sentencia:

```
SET SERVEROUTPUT ON;
```

Una vez escrito este código en la consola de SQL*Plus, se teclea una diagonal (/) para ejecutarlo como cualquier comando:

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
 2   DBMS_OUTPUT.put_line ('Hola Mundo.');
 3 END;
 4 /
Hola Mundo.
```

Procedimiento PL/SQL terminado correctamente.

Se puede volver a ejecutar el bloque anónimo tecleando la diagonal nuevamente.

Si tiene algo que modificar, puede editar el buffer (afiedt.buf) en el bloc de notas, mediante el comando EDIT, guardar los cambios y luego ejecutarlo con la diagonal, como se muestra:

```
SQL> edit
Escrito file afiedt.buf

 1 BEGIN
 2   DBMS_OUTPUT.put_line ('Hola Todos');
 3* END;
SQL> /
Hola Todos
```

Procedimiento PL/SQL terminado correctamente.

Si prefiere utilizar Oracle SQL Developer, puede crear un bloque anónimo estableciendo una conexión y escribiendo el bloque anónimo en la hoja de trabajo iniciando con la sentencia SET SERVEROUTPUT ON; para luego dar clic en Ejecutar.

Ejemplo completo

En este ejemplo, se modifica el bloque anónimo anterior para incluir las secciones de declaraciones y de manejo de excepciones. Aquí se define una variable numérica para guardar el resultado de una división, pero se provoca un error al dividir entre cero y entonces se controla la excepción generando un mensaje indicando el error ocurrido:

```
SQL> DECLARE
 2   resultado NUMBER;
 3 BEGIN
 4   resultado := 5/0;
 5 EXCEPTION
 6   WHEN ZERO_DIVIDE THEN
 7     DBMS_OUTPUT.PUT_LINE(SQLERRM);
 8 END;
 9 /
```

ORA-01476: el divisor es igual a cero

Procedimiento PL/SQL terminado correctamente.

Tipos de Datos

PL/SQL maneja dos distintos tipos de datos: los escalares y los compuestos.

Los tipos escalares se agrupan en cuatro familias, que a su vez tienen algunos subtipos, además de que reconoce diversos sinónimos usados en otros gestores de bases de datos con fines de compatibilidad:

- Number, que representa números reales, enteros y de punto flotante
- Boolean, que poseen solamente tres valores: TRUE, FALSE y NULL.
- Character, principalmente: CHAR(n), VARCHAR2(n) y LONG.
- Datetime, los principales son: DATE y TIMESTAMP.

Mientras que los tipos compuestos almacenan múltiples valores, tal es el caso de los registros y las colecciones.

VARIABLES

En PL/SQL, una variable es una localidad de memoria definida para almacenar un particular tipo de datos mediante un nombre. El valor de la variable cambia durante la ejecución del programa. Antes de usar una variable, esta debe ser declarada en la sección de declaraciones de un bloque.

La sintaxis para declarar una variable es:

```
nombre_variable tipo [NOT NULL] [:= valor_inicial];
```

Ejemplo

Se puede emplear la palabra DEFAULT para asignar el valor inicial en vez de :=.

```
SQL> DECLARE
 2      ventas_totales NUMBER(15,2);
 3      limite_credito NUMBER(10,0);
 4      nombre_contacto VARCHAR2(255);
 5      estado VARCHAR2(20) NOT NULL DEFAULT 'Activo';
 6  BEGIN
 7      NULL;
 8  END;
 9 /
```

```
Procedimiento PL/SQL terminado correctamente.
```

Si se declara la condición NOT NULL, significa que no permite valores nulos y deberá ser inicializada.

Declaraciones ancladas

Es frecuente que se declare una variable para guardar el valor de una columna de una tabla, por lo tanto, se puede escribir el programa para que asigne el tipo de dato correspondiente mediante el sufijo %TYPE.

Este tipo de declaraciones también son válidas para asignar a unas variables los tipos de variables previamente definidas.

Ejemplo

```
SQL> DECLARE
  2      sucursal_cliente "Cliente"."Sucursal"%TYPE;
  3      nombre_cliente "Cliente". "Nombre"%TYPE;
  4      empresa_cliente "Cliente"."Empresa"%TYPE;
  5  BEGIN
  6      SELECT "Sucursal", "Nombre", "Empresa" INTO sucursal_cliente,
  7      nombre_cliente, empresa_cliente
  8      FROM "Cliente" WHERE "Clave" = 35;
  9      DBMS_OUTPUT.PUT_LINE('Sucursal: ' || sucursal_cliente);
 10     DBMS_OUTPUT.PUT_LINE('Nombre Cliente: ' || nombre_cliente);
 11     DBMS_OUTPUT.PUT_LINE('Empresa: ' || empresa_cliente);
 12  END;
 13 /
Sucursal: 281
Nombre Cliente: Blanca Nieves
Empresa: Textiles Finos

Procedimiento PL/SQL terminado correctamente.
```

Constantes

Una constante conserva un valor que no puede cambiar durante la ejecución de un programa. El uso de constantes tiene una finalidad práctica y de legibilidad de un programa.

Para declarar una constante, se usa la palabra CONSTANT después del nombre de la constante, se asigna un tipo y luego su valor, su sintaxis es:

`nombre_constante CONSTANT tipo [NOT NULL] := expresión`

Ejemplo

```
SQL> DECLARE
  2      iva CONSTANT NUMBER := 0.16;
  3      precio_lista NUMBER(10,2) := 100;
  4      precio_venta NUMBER(10,2) := precio_lista + precio_lista * iva;
  5  BEGIN
  6      NULL;
  7  END;
  8 /

Procedimiento PL/SQL terminado correctamente.
```

Cualquier intento de modificar el valor de una constante provocará un error.

Comentarios

Los comentarios permiten documentar el propósito de una línea o de un bloque de código. Es importante usar comentarios para hacer más comprensible el código.

Hay dos formas de incluir comentarios:

Comentarios de una sola línea

Los comentarios de una sola línea se anteceden de dos guiones, puede ser al inicio de una línea o después del código, y se extienden hasta el final de la línea.

A veces se colocan los dos guiones para deshabilitar una línea de código.

Ejemplo

```
SQL> -- el I.V.A. aplicable es del 16%
SQL>
```

Comentarios multilínea

Un comentario multilínea inicia con diagonal-asterisco y termina con asterisco-diagonal, y puede abarcar una o varias líneas de texto. A veces se utilizan para deshabilitar un bloque de varias líneas de código de un programa.

Ejemplo

```
SQL> /* El propósito de este programa consiste en
SQL>     Bla, bla, bla.
SQL> */
SQL>
```

No se permiten comentarios anidados dentro de otros comentarios.

Sentencias IF

Una sentencia IF permite ejecutar o saltar a una secuencia de instrucciones dependiendo del cumplimiento de una condición. Y se usa de tres formas:

Caso IF THEN

La estructura de la sentencia IF THEN es:

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

La condición es una expresión Booleana que puede ser TRUE, FALSE o NULL. Si su valor es TRUE, entonces se ejecuta el bloque de instrucciones a partir de la palabra THEN, en caso contrario, se salta hasta el END IF.

Ejemplo

```
SQL> DECLARE
 2      ventas NUMBER := 200000;
 3  BEGIN
 4      IF ventas > 100000 THEN
 5          DBMS_OUTPUT.PUT_LINE('Ingreso por ventas mayor a $100,000');
 6      END IF;
 7  END;
 8 /
Ingreso por ventas mayor a $100,000

Procedimiento PL/SQL terminado correctamente.
```

Caso IF THEN ELSE

La estructura de la sentencia IF THEN ELSE es:

```
IF condición THEN
```

```

    instrucciones;
ELSE
    otras_instrucciones;
END IF;
```

Si la condición es TRUE, entonces se ejecuta el bloque de instrucciones entre la palabra THEN y la palabra ELSE, en caso contrario, se ejecutan las instrucciones desde la palabra ELSE hasta el END IF.

Ejemplo

```

SQL> DECLARE
 2      ventas NUMBER := 200000;
 3      comision NUMBER;
 4  BEGIN
 5      IF ventas > 200000 THEN
 6          comision := ventas * 0.1;
 7      ELSE
 8          comision := ventas * 0.05;
 9      END IF;
10      DBMS_OUTPUT.PUT_LINE('Comisión por ventas: ' || comision);
11  END;
12 /
```

Comisión por ventas: 10000

Procedimiento PL/SQL terminado correctamente.

Caso IF THEN ELSIF

La estructura de la sentencia IF THEN ELSIF es:

```

IF condición1 THEN
    instrucciones1;
ELSIF condición2 THEN
    instrucciones2;
...
[ELSE
    otras_instrucciones;]
END IF;
```

Si se cumple la primera condición, se ejecuta el primer bloque de instrucciones. En caso contrario, se evalúa cada una de las condiciones ELSIF hasta que alguna sea verdadera, y se ejecuta el bloque de instrucciones correspondientes.

En caso de no cumplirse ninguna de las condiciones se alcanza la cláusula ELSE y se ejecutan las instrucciones correspondientes.

Ejemplo

```
SQL> DECLARE
  2      ventas NUMBER := 40000;
  3      comision NUMBER;
  4  BEGIN
  5      IF ventas > 200000 THEN
  6          comision := ventas * 0.1;
  7      ELSIF ventas > 100000 THEN
  8          comision := ventas * 0.08;
  9      ELSIF ventas > 50000 THEN
 10          comision := ventas * 0.05;
 11      ELSE
 12          comision := ventas * 0.03;
 13      END IF;
 14      DBMS_OUTPUT.PUT_LINE('Comision por ventas:' || comision);
 15  END;
 16 /
Comision por ventas:1200

Procedimiento PL/SQL terminado correctamente.
```

Condiciones IF anidadas

Se pueden anidar sentencias IF según se requieran:

```
IF condición1 THEN
    IF condición2 THEN
        sentencias;
    END IF;
ELSE
    otras_sentencias;
END IF;
```

Manejar demasiados niveles de anidamiento convierten al código en algo difícil de comprender, por lo que es aconsejable evitarlo en la medida de lo posible.

Sentencias CASE

Una sentencia CASE sirve para elegir una secuencia de instrucciones de entre varias otras posibles secuencias. Existen dos tipos de sentencias CASE:

Tipo CASE simple

El tipo simple de sentencias CASE evalúa una simple expresión y compara su resultado con diversos valores. Su estructura es como sigue:

```
CASE selector
    WHEN valor1 THEN
        sentencias1
    WHEN valor2 THEN
        sentencias2
    ...
    [ELSE
```

```

        otras_sentencias]
END CASE;

```

Cuando el valor del selector coincide con alguno de los valores en WHEN, se ejecutan las sentencias asociadas a ese valor y ya no se requieren evaluar los demás valores y la ejecución se pasa hasta el END CASE.

Cuando el selector no coincide con ningún valor, se ejecutan las sentencias en la cláusula ELSE. Aunque usar ELSE es opcional, PL/SQL lanzará una excepción CASE_NOT_FOUND cuando una expresión CASE no coincide con ninguna cláusula WHEN y no exista un ELSE.

Ejemplo

```

SQL> DECLARE
 2      grado CHARACTER := 'B';
 3      nota STRING(15);
 4  BEGIN
 5      CASE grado
 6          WHEN 'A' THEN
 7              nota := 'Excelente' ;
 8          WHEN 'B' THEN
 9              nota := 'Muy Bueno' ;
10          WHEN 'C' THEN
11              nota := 'Bueno' ;
12          ELSE
13              nota := 'Insuficiente' ;
14      END CASE;
15      DBMS_OUTPUT.PUT_LINE('Nota obtenida:' || nota);
16  END;
17 /
Nota obtenida:Muy Bueno

Procedimiento PL/SQL terminado correctamente.

```

Tipo CASE calculado

Una sentencia CASE tipo calculado evalúa múltiples expresiones condicionales y ejecuta las sentencias asociadas con la primera condición que sea TRUE. Su estructura es:

```

CASE
    WHEN condición1 THEN sentencias1
    WHEN condición2 THEN sentencias2
    ...
    [ELSE
        otras_sentencias]
END CASE;

```

Ejemplo

```

SQL> DECLARE
 2      ventas NUMBER := 40000;
 3      comision NUMBER;
 4  BEGIN
 5      CASE
 6          WHEN ventas > 200000 THEN
 7              comision := ventas * 0.1;
 8          WHEN ventas > 100000 THEN
 9              comision := ventas * 0.08;
10          WHEN ventas > 50000 THEN
11              comision := ventas * 0.05;
12      ELSE
13          comision := ventas * 0.03;
14      END CASE;
15      DBMS_OUTPUT.PUT_LINE('Comision por ventas:' || comision);
16  END;
17 /
Comision por ventas:1200

Procedimiento PL/SQL terminado correctamente.

```

Sentencia NULL

La sentencia NULL es un comando que no hace nada. Sirve como complemento de una sentencia que no se desea que haga nada, pero permite dar legibilidad al código o para apartar un sitio donde colocar un bloque posteriormente.

Ejemplo

```

SQL> DECLARE
 2      estado STRING(10):= 'Normal';
 3  BEGIN
 4      CASE estado
 5          WHEN 'Activo'
 6              THEN DBMS_OUTPUT.PUT_LINE(' Enviar correo ');
 7          WHEN 'Suspendido'
 8              THEN DBMS_OUTPUT.PUT_LINE(' Bloquear pedidos ');
 9      ELSE
10          NULL;
11      END CASE;
12  END;
13 /

Procedimiento PL/SQL terminado correctamente.

```

Bucles LOOP

Una sentencia LOOP es una estructura de control que ejecuta un bloque de código hasta que se cumpla una determinada condición. La sintaxis básica es:

```

LOOP
    sentencias;
END LOOP;

```

Para poder salir de un bloque LOOP se debe encontrar una sentencia EXIT dentro del bloque de instrucciones, para evitar que se convierta en un ciclo infinito.

Generalmente se usa LOOP cuando se desea que al menos se ejecute una vez el bloque de instrucciones o cuando no se conoce el número de veces que se va a ejecutar.

Sentencia EXIT

La sentencia EXIT permite salir del bloque LOOP. Usualmente se coloca dentro de una sentencia condicional IF.

Ejemplo

```
SQL> DECLARE
  2      Contador NUMBER := 0;
  3  BEGIN
  4      LOOP
  5          contador := contador + 1 ;
  6          IF contador > 10 THEN
  7              EXIT;
  8          END IF;
  9          dbms_output.put_line( 'Contador: ' || contador ) ;
10      END LOOP;
11  END;
12 /
```

Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
Contador: 6
Contador: 7
Contador: 8
Contador: 9
Contador: 10

Procedimiento PL/SQL terminado correctamente.

Sentencia EXIT WHEN

Otra forma de abandonar un bucle LOOP es mediante la sentencia EXIT WHEN, cuya sintaxis es:

EXIT WHEN condición;

Esencialmente la sentencia EXIT WHEN es equivalente a la sentencia EXIT dentro de una sentencia IF THEN, cada vez que se encuentre con la sentencia EXIT WHEN se evalúa la condición y si su valor es TRUE, entonces el LOOP termina.

Ejemplo

```

SQL> DECLARE
  2      Contador NUMBER := 0;
  3  BEGIN
  4      LOOP
  5          contador := contador + 1 ;
  6          EXIT WHEN contador > 10;
  7          dbms_output.put_line( 'Contador: ' || contador )
  8      END LOOP;
  9  END;
10 /
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
Contador: 6
Contador: 7
Contador: 8
Contador: 9
Contador: 10

```

Procedimiento PL/SQL terminado correctamente.

Es posible anidar una sentencia LOOP dentro de otra sentencia LOOP, y en este caso se aconseja usar etiquetas para su identificación:

```

SQL> DECLARE
  2      i NUMBER := 0;
  3      j NUMBER;
  4  BEGIN
  5      <<loop_externo>>
  6      LOOP
  7          i := i + 1;
  8          EXIT loop_externo WHEN i > 4;
  9          dbms_output.put_line('Valor de i: ' || i);
10          j := 0;
11          <<loop_interno>> LOOP
12              j := j + 1;
13              EXIT loop_interno WHEN j > 3;
14              dbms_output.put_line('Valor de j: ' || j);
15          END LOOP loop_interno;
16      END LOOP loop_externo;
17  END;
18 /
Valor de i: 1
Valor de j: 1
Valor de j: 2
Valor de j: 3
Valor de i: 2
Valor de j: 1
Valor de j: 2
Valor de j: 3
Valor de i: 3
Valor de j: 1
Valor de j: 2
Valor de j: 3
Valor de i: 4
Valor de j: 1
Valor de j: 2
Valor de j: 3

```

Procedimiento PL/SQL terminado correctamente.

Bucles FOR LOOP

El bucle FOR LOOP ejecuta una secuencia de instrucciones un determinado número de veces. Tiene la siguiente estructura:

```
FOR índice IN límite_inferior .. límite_superior
LOOP
    sentencias;
END LOOP;
```

La variable *índice* es implícita y es una variable interna del bucle FOR LOOP, por lo que no se puede referenciar fuera de éste. Y dentro del bucle se puede referenciar, pero no se le puede cambiar el valor actual.

Los límites inferior y superior son números o expresiones numéricas que se evalúan al iniciar la sentencia FOR LOOP, en caso necesario, se redondean al entero más cercano y no se pueden modificar.

El bucle inicia con el índice con el valor del límite inferior y se incrementa de uno en uno, mientras el valor del índice no supere al límite superior.

Ejemplo simple

```
SQL> BEGIN
  2  FOR contador IN 1..5
  3  LOOP
  4      DBMS_OUTPUT.PUT_LINE(contador);
  5  END LOOP;
  6 END;
  7 /
1
2
3
4
5
```

Procedimiento PL/SQL terminado correctamente.

Ejemplo con incremento de 2.

Equivale a un contador de 2 al 10 con incrementos de 2.

```
SQL> BEGIN
  2  FOR contador IN 1..5
  3  LOOP
  4      DBMS_OUTPUT.PUT_LINE(contador*2);
  5  END LOOP;
  6 END;
  7 /
2
4
6
8
10
```

Procedimiento PL/SQL terminado correctamente.

FOR LOOP con la cláusula REVERSE

Cuando se emplea la cláusula REVERSE en la estructura FOR LOOP, el límite superior se decrementa hasta alcanzar al límite inferior, su sintaxis es:

```
FOR índice IN REVERSE límite_inferior .. límite_superior
LOOP
    sentencias;
END LOOP;
```

Ejemplo

```
SQL> BEGIN
  2  FOR contador IN REVERSE 1..5
  3  LOOP
  4      DBMS_OUTPUT.PUT_LINE(contador);
  5  END LOOP;
  6  END;
  7 /
5
4
3
2
1

Procedimiento PL/SQL terminado correctamente.
```

Bucles WHILE LOOP

Un bucle WHILE LOOP es una estructura de control que repite un bloque de código mientras se cumpla una determinada condición. Su sintaxis es:

```
WHILE condición
LOOP
    sentencias;
END LOOP;
```

Donde condición es una expresión booleana y el bloque dentro del bucle se ejecuta mientras su valor sea TRUE. La condición se evalúa antes de cada iteración. Puede suceder que la condición sea inicialmente FALSE o NULL, y el bucle WHILE LOOP no ejecutaría nada. Es posible que se finalice la ejecución del bucle dentro del bloque de instrucciones mediante el empleo de EXIT o EXIT WHEN.

Ejemplo simple

```
SQL> DECLARE
  2      contador NUMBER := 0;
  3  BEGIN
  4  WHILE contador <= 5
  5  LOOP
  6      DBMS_OUTPUT.PUT_LINE(contador);
  7      Contador := contador + 1;
  8  END LOOP;
  9  END;
10 /
0
1
2
3
4
5

Procedimiento PL/SQL terminado correctamente.
```

Ejemplo con EXIT WHEN

```
SQL> DECLARE
  2      contador NUMBER := 0;
  3  BEGIN
  4  WHILE contador <= 5
  5  LOOP
  6      DBMS_OUTPUT.PUT_LINE(contador);
  7      Contador := contador + 1;
  8      EXIT WHEN contador = 3;
  9  END LOOP;
10 END;
11 /

0
1
2

Procedimiento PL/SQL terminado correctamente.
```

Cláusula CONTINUE

La cláusula CONTINUE permite finalizar una iteración de un bucle LOOP y continuar con la siguiente iteración. Su sintaxis es:

CONTINUE;

Generalmente se usa dentro de una condición IF THEN para salir de la iteración cuando se alcance determinada condición, como se muestra:

```
IF condición THEN
    CONTINUE;
END IF;
```

Ejemplo

```
SQL> BEGIN
 2 FOR indice IN 1 .. 10
 3 LOOP
 4     -- se salta en caso de ser impar
 5     IF MOD(indice, 2) = 1 THEN
 6         CONTINUE;
 7     END IF;
 8     DBMS_OUTPUT.PUT_LINE( indice );
 9 END LOOP;
10 END;
11 /
2
4
6
8
10
```

Procedimiento PL/SQL terminado correctamente.

Cláusula CONTINUE WHEN

La cláusula CONTINUE WHEN se usa para salir de la iteración de un bucle LOOP cuando se alcance determinada condición, y continuar con la siguiente iteración. Su sintaxis es:

CONTINUE WHEN condición;

Ejemplo

```
SQL> BEGIN
 2 FOR indice IN 1 .. 10
 3 LOOP
 4     -- se salta en caso de ser par
 5     CONTINUE WHEN MOD(indice, 2) = 0;
 6     DBMS_OUTPUT.PUT_LINE( indice );
 7 END LOOP;
 8 END;
 9 /
1
3
5
7
9
```

Procedimiento PL/SQL terminado correctamente.

Registros de PL/SQL

Un registro PL/SQL es una estructura de datos compuesta, formada por múltiples campos, cada uno conteniendo un valor propio.

Los registros ayudan a simplificar el código, pasando de operaciones de campos a operaciones de registros.

Existen tres tipos de registros, en todos los casos, primero se deben declarar antes de utilizarlos.

Registros basados en tablas

Un registro basado en tablas se declara usando el atributo %ROWTYPE con el nombre de la tabla correspondiente. Este tipo de registros contiene cada uno de los campos definidos en la estructura de la tabla.

```
DECLARE
    nombre_registro nombre_tabla%ROWTYPE;
```

Comando SELECT INTO

El comando SELECT INTO es la forma más simple para pasar un registro de una tabla a un conjunto de variables. Su sintaxis es:

```
SELECT lista_columnas INTO lista_variables FROM tabla
WHERE condición;
SELECT * INTO nombre_registro FROM tabla
WHERE condición;
```

Se debe cuidar que la cantidad de columnas de la lista coincida con la cantidad de variables, además de que los tipos correspondientes deben ser compatibles. Se pueden emplear sentencias SELECT más complejas, siempre y cuando se tenga la seguridad que se devuelve exactamente un registro.

Ejemplo

```
SQL> DECLARE
  2      reg carrera%ROWTYPE;
  3      nom alumnos.nombre%TYPE;
  4      ape alumnos.apellido%TYPE;
  5  BEGIN
  6      SELECT * INTO reg from carrera WHERE clave = (SELECT carrera
  7      FROM alumnos WHERE clave='23120346');
  8      SELECT nombre, apellido INTO nom, ape FROM alumnos WHERE
  9      clave='23120346';
 10      DBMS_OUTPUT.PUT_LINE('Alumno:' || nom || ' ' || ape);
 11      DBMS_OUTPUT.PUT_LINE('Carrera:' || reg.clave || ' ' || reg.descripcion);
 12  END;
 13 /
Alumno:Marcela Lara
Carrera:LIAR Arquitecto

Procedimiento PL/SQL terminado correctamente.
```

Mensajes de error más frecuentes

Los errores más frecuentes que se presentan en este tipo de sentencias son:

- PL/SQL: ORA-00947: no hay suficientes valores.
- PL/SQL: ORA-00913: demasiados valores.
- ORA-01722: número no válido.
- ORA-01422: la recuperación exacta devuelve un número mayor de filas que el solicitado.
- ORA-01403: No se ha encontrado ningún dato.

Registro basado en Cursos

Un registro basado en cursos tiene cada columna correspondiente a la columna o al alias de una sentencia SELECT asociada al cursor.

Para declarar un registro basado en cursos, primero se declara el cursor como una sentencia SELECT y luego se utiliza el atributo %ROWTYPE con el cursor explícito:

```
DECLARE
    CURSOR nombre_cursor IS sentencia_Select
        nombre_registro nombre_cursor%ROWTYPE;
```

Con la sentencia SELECT INTO se puede asignar los valores de la consulta a un registro completo:

Con el comando Fetch se puede extraer de un cursor un registro completo o campos individuales:

Ejemplo

```
SQL> DECLARE
  2      CURSOR cur_alumnos IS SELECT nombre, apellido, carrera FROM
  3          alumnos;
  4      reg_alumnos cur_alumnos%ROWTYPE;
  5  BEGIN
  6      OPEN cur_alumnos;
  7      FETCH cur_alumnos INTO reg_alumnos;
  8      DBMS_OUTPUT.PUT_LINE('Alumno:' || reg_alumnos.nombre || ' ' || reg_alumnos.apellido);
  9      DBMS_OUTPUT.PUT_LINE('Carrera:' || reg_alumnos.carrera);
 10 END;
 11 /
Alumno:Kevin Torres
Carrera:LIDE

Procedimiento PL/SQL terminado correctamente.
```

Registro definido por el Programador

Si se desea crear un registro con una estructura que no se base en las estructuras existentes, se emplea un registro definido por el programador, utilizando los siguientes pasos:

- Se define un tipo RECORD que contenga la estructura deseada de Registro, de manera similar a la definición de los registros de una tabla.
- Se declara un registro basado en ese tipo de registro.

Para referirse a un campo dentro de un registro se emplea la notación siguiente:

`nombre_registro.nombre_campo`

Ejemplo

```
SQL> DECLARE
  2      TYPE tipo_reg_alumno IS RECORD (
  3          nom alumnos.nombre%TYPE,
  4          ape alumnos.apellido%TYPE,
  5          car carrera.descripcion%TYPE );
  6  BEGIN
  7      NULL;
  8  END;
  9 /
```

Procedimiento PL/SQL terminado correctamente.

Referenciando un campo de un registro

Asignando registros

Se puede asignar un registro a otro registro del mismo tipo:

```
reg_contacto1 := reg_contacto2;
```

Pero no se pueden comparar directamente:

```
IF reg_contacto1 = reg_contacto2 THEN ...
```

Sino que se deben comparar individualmente los campos de cada registro:

```
IF reg_contacto1.nombre_empresa = reg_contacto2.nombre_empresa AND
    reg_contacto1.nombre_cliente = reg_contacto2.nombre_cliente AND
    reg_contacto1.apellido_cliente = reg_contacto2.apellido_cliente
THEN
```

Similarmente, se pueden asignar valores a los campos de un registro de forma individual, por ejemplo:

```
reg_contacto1.nombre_empresa := 'Bimbo';
reg_contacto1.nombre_cliente := 'Juan';
reg_contacto1.apellido_cliente := 'Pérez';
```

Sentencia INSERT con registros

Se puede insertar un nuevo registro a una tabla usando un registro %ROWTYPE sin necesidad de especificar cada campo.

Ejemplo

```
SQL> DECLARE
  2      reg_est estudiantes%ROWTYPE;
  3  BEGIN
  4      reg_est.ncontrol := '23120458';
  5      reg_est.nombre := 'Juan';
  6      reg_est.apellido := 'Pérez';
  7      reg_est.sexo := 'M';
  8      reg_est.carrera := 'ISC';
  9      INSERT INTO estudiantes VALUES reg_est;
 10  END;
 11 /
```

Procedimiento PL/SQL terminado correctamente.

Sentencia UPDATE con registros

Para actualizar un registro usando %ROWTYPE se utiliza la cláusula SET ROW.

Ejemplo

```
SQL> DECLARE
 2      reg_est estudiantes%ROWTYPE;
 3  BEGIN
 4      SELECT * INTO reg_est FROM estudiantes WHERE ncontrol = '23120458';
 5      reg_est.nombre := 'Luis';
 6      UPDATE estudiantes SET ROW = reg_est WHERE ncontrol = reg_est.ncontrol;
 7  END;
 8 /
```

Procedimiento PL/SQL terminado correctamente.

Registros anidados

Un registro puede contener un campo que a su vez es otro registro. Anidar registros es una forma poderosa de estructurar los datos y ocultar la complejidad dentro del código.

Ejemplo

```
SQL> DECLARE
 2      TYPE dirección IS RECORD (
 3          calle VARCHAR2(255),
 4          ciudad VARCHAR2(100),
 5          estado VARCHAR2(100),
 6          código_postal VARCHAR(10)
 7      );
 8      TYPE cliente IS RECORD(
 9          razón_social VARCHAR2(100),
10          dirección_envío dirección,
11          dirección_factura dirección
12      );
13      reg_cliente cliente;
14  BEGIN
15      reg_cliente.razón_social := 'Bimbo, S.A.';
16      reg_cliente.dirección_envío.calle := 'Madero 4000';
17      reg_cliente.dirección_envío.ciudad := 'Morelia';
18      reg_cliente.dirección_envío.estado := 'Mich.';
19      reg_cliente.dirección_envío.código_postal := '58134';
20      reg_cliente.dirección_factura := reg_cliente.dirección_envío;
21  END;
22 /
```

Procedimiento PL/SQL terminado correctamente.