INTRODUCCIÓN A PL/SQL

¿Qué es PL/SQL?

PL/SQL provee una manera muy cómoda de relacionar los conceptos de bases de datos y manejarlos mediante ciertas estructuras de control, dentro del contexto de una herramienta netamente de programación.
Su utilización es dentro del administrador de bases de datos "Oracle" y sus principales características son la posibilidad que brinda de utilizar sentencias SQL para manipular datos y sentencias de control de flujo para organizar esta manipulación de datos.
Dentro del lenguaje, es posible declarar constantes y variables, definir procedimientos y funciones y atrapar errores en tiempo de ejecución. Así visto, PL/SQL combina la el poder de la manipulación de datos, con SQL, y las facilidades del procesamiento de los mismos, tal como en los más modernos lenguajes de programación.

Ventajas en la utilización de PL/SQL

PL/SQL es un lenguaje de procesamiento de transacciones completamente portable y con un alto rendimiento, que proporciona las siguientes ventajas al ser utilizado:

- Soporte para SQL
- Soporte para la programación orientada a objetos
- Mejor rendimiento
- Completa portabilidad
- Integración con Oracle garantizada
- Seguridad

Soporte para SQL

SQL se ha convertido en el lenguaje estándar de bases de datos por su flexibilidad de uso y facilidad de aprenderlo. Unos pocos comandos permiten la fácil manipulación de prácticamente toda la información almacenada en una base de datos.

SQL es no-procedural, lo cual significa que el motor de base de datos es quien se preocupará de cómo ejecutar de la mejor manera un requerimiento señalado en una sentencia SQL no es necesaria la conexión entre varias sentencias porque los motores las ejecuta de a una a la vez.
PL/SQL permite una completa manipulación de los datos almacenados en una base de datos, proporciona comandos de control de transacciones y permite utilizar las funciones de SQL, operadores y consultas. Así, se puede manipular los datos en la base de datos de una manera flexible y segura. Además, PL/SQL soporta tipos de datos de SQL, lo que reduce la necesidad de convertir los datos al pasar de una a otra aplicación.
PL/SQL también soporta SQL dinámico, una avanzada técnica de programación que convierte las aplicaciones en más flexibles y versátiles.

Soporte para Programación Orientada a Objetos

Los objetos se han convertido en una herramienta ideal para modelar situaciones de la vida real. Con su utilización es posible reducir el costo y tiempo de construcción de aplicaciones complejas. Otra ventaja es que utilizando una metodología de este tipo es posible mantener diferentes equipos de programadores construyendo aplicaciones basadas en el mismo grupo de objetos.
Permitir el encapsulamiento del código en bloques es el primer paso para la implementación de métodos asociados a diferentes tipos de objetos construidos también con PL/SQL.

Mejor rendimiento

Sin PL/SQL, las bases de datos tendrían que procesar las instrucciones una a una. Cada llamada produciría un exceso de utilización de memoria considerable, sobre todo si consideramos que estas consultas viajan a través de la red.
Por el contrario, con PL/SQL, un bloque completo de sentencias puede ser enviado cada vez a Oracle, lo que reduce drásticamente la intensidad de comunicación con la base de datos. Los procedimientos almacenados escritos con PL/SQL son compilados una vez y almacenados en formato ejecutable, lo que produce que las llamadas sean más rápidas y eficientes. Además, ya que los procedimientos

almacenados se ejecutan en el propio servidor, el tráfico por la red se reduce a la simple llamada y el envío de los parámetros necesarios para su ejecución.
El código ejecutable se almacena en caché y se comparte a todos los usuarios, redundando en mínimos requerimientos de memoria.

Completa portabilidad

Las aplicaciones escritas con PL/SQL son portables a cualquier sistema operativo y plataforma en la cual se encuentre corriendo la base de datos. Esto significa que se pueden codificar librerías que podrán ser reutilizadas en otros ambientes.

Integración con Oracle

PL/SQL y los lenguajes SQL en general se encuentran perfectamente integrados. PL/SQL soporta todos los tipos de datos de SQL. Los atributos %TYPE y %ROWTYPE integran PL/SQL con SQL, permitiendo la declaración de variables basado en tipos de columnas de tablas de la base de datos. Lo anterior provee independencia de los datos, reduce costos de mantención y permite a los programas adaptarse a los cambios en la base de datos para cumplir con las nuevas necesidades del negocio.

Seguridad

Los procedimientos almacenados construidos con PL/SQL habilitan la división de la lógica del cliente con la del servidor. De esta manera, se previene que se efectúe manipulación de los datos desde el cliente. Además, se puede restringir el acceso a los datos de la base de datos, permitiendo a los usuarios la ejecución de los procedimientos almacenados para los cuales tengan privilegios solamente.

ESTRUCTURAS DE BLOQUE

PL/SQL es un lenguaje *estructurado en bloques*, lo que quiere decir que la unidad básica de codificación son bloques lógicos, los que a su vez pueden contener otros sub-bloques dentro de ellos, con las mismas características.

Un bloque (o sub-bloque) permite agrupar en forma lógica un grupo de sentencias. De esta manera se pueden efectuar declaraciones de variables que sólo tendrán validez en los bloques donde éstas se definan.

Un bloque PL/SQL tiene tres partes:

1. Una sección de *declaración*
2. Una sección de *ejecución*
3. Una sección de manejo de *excepciones*.

Sólo el bloque de ejecución es obligatorio en un programa PL/SQL.

Es posible anidar sub-bloques en la sección ejecutable y de excepciones, pero no en la sección de declaraciones.

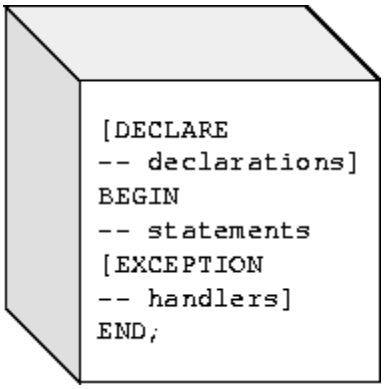

Figura Estructura de bloques de un programa PL/SQL

FUNDAMENTOS DEL LENGUAJE

Variables y Constantes

PL/SQL permite declarar constantes y variables para ser utilizadas en cualquier expresión dentro de un programa. La única condición exigida por PL/SQL es que cada variable (o constante) debe estar declarada antes de ser utilizada en una expresión.

Las variables pueden corresponder a cualquier tipo de dato de SQL, tal como *char*, *date* o *number*, o algún tipo de PL/SQL, como *boolean* o *binary_integer*.

Por ejemplo, si desea declarar una variable llamada "profesor_no" que almacene cuatro dígitos numéricos y otra variable "total_stock" de tipo booleano, es decir, que almacene solamente los valores True o False, la declaración se vería como sigue:

```
profesor_num        number(4) ;
total_stock         boolean ;
```

Cómo asignar valores a variables

Es posible asignar valores a las variables de dos formas. La primera utiliza el operador "**:=**". La variable se ubica al lado izquierdo y la expresión al lado derecho del símbolo.

Por ejemplo:

```
total := precio * unidad ;
salario := bono * 0.10 ;
```

Declaración de Constantes

En la declaración de una constante (muy similar a la de una variable), se debe incorporar la palabra reservada "constant" e inmediatamente asignar el valor deseado. En adelante, no se permitirán reasignaciones de valores para aquella constante que ya ha sido definida.

Ejemplo:

```
limite_credito CONSTANT real := 5000.00 ;
```

Conjunto de Caracteres y Unidades Léxicas

Las instrucciones del lenguaje deben ser escritas utilizando un grupo de caracteres válidos. PL/SQL no es sensible a mayúsculas o minúsculas. El grupo de caracteres incluye los siguientes:

- Letras mayúsculas y minúsculas de la A a la Z
- Números del 0 al 9
- Los símbolos ( ) + - * / < > = ! ~ ^ ; . ' @ % , " # $ & _ | { } ? [ ]
- Tabuladores, espacios y saltos de carro

Una línea de texto en un programa contiene lo que se conoce como unidades léxicas, los que se clasifican como sigue:

- Delimitadores (símbolos simples y compuestos)
- Identificadores (incluye palabras reservadas)
- Literales
- Comentarios

Por ejemplo en la instrucción:

```
Salario_mensual := sueldo * 0.10;   -- cálculo del sueldo mensual
```

se observan las siguientes unidades léxicas:

- Los identificadores Salario_mensual y sueldo
- El símbolo compuesto **:=**
- Los símbolos simples ***** y **;**
- El literal numérico *0.10*
- El comentario "*cálculo del sueldo mensual*"

Para asegurar la fácil comprensión del código se pueden añadir espacios entre identificadores o símbolos. También es una buena práctica utilizar saltos de línea e indentaciones para permitir una mejor legibilidad.

Ejemplo:

```
IF x>y THEN max := x; ELSE max := y; END IF;
```

Puede reescribirse de la siguiente manera para mejorar el aspecto y legibilidad:

```
IF x > y THEN
        max := x;
ELSE
        max := y;
END IF;
```

Delimitadores e Identificadores

Un delimitador es un símbolo simple o compuesto que tiene un significado especial dentro de PL/SQL. Por ejemplo, es posible utilizar delimitadores para representar operaciones aritméticas, por ejemplo:

Símbolo	Significado
+	operador de suma
%	indicador de atributo
'	delimitador de caracteres
.	selector de componente
/	operador de división
(	expresión o delimitador de lista
)	expresión o delimitador de lista
:	indicador de variable host
,	separador de ítems
*****	operador de multiplicación
"	delimitador de un identificador entre comillas
=	operador relacional
<	operador relacional
>	operador relacional
@	indicador de acceso remoto
;	terminador de sentencias
-	negación u operador de substracción

Los delimitadores compuestos consisten de dos caracteres, como por ejemplo:

Símbolo	Significado
:=	operador de asignación
=>	operador de asociación
||	operador de concatenación
**	operador de exponenciación
<<	comienzo de un rótulo
>>	fin de un rótulo
/*	comienzo de un comentario de varias líneas
*/	fin de un comentario de varias líneas
..	operador de rango
<>	operador relacional
!=	operador relacional
^=	operador relacional
<=	operador relacional
>=	operador relacional
--	comentario en una línea

Los identificadores incluyen constantes, variables, excepciones, cursores, subprogramas y paquetes. Un identificador se forma de una letra, seguida opcionalmente de otras letras, números, signo de moneda y otros signos numéricos.

La longitud de un identificador no puede exceder los 30 caracteres. Se recomienda que los nombres de los identificadores utilizados sean descriptivos.

Algunos identificadores especiales, llamados *palabras reservadas*, tienen un especial significado sintáctico en PL/SQL y no pueden ser redefinidos. Son palabras reservadas, por ejemplo, BEGIN, END, ROLLBACK, etc.

Tipos de Datos

Cada constante y variable posee un tipo de dato el cual especifica su forma de almacenamiento, restricciones y rango de valores válidos. Con PL/SQL se proveen diferentes tipos de datos predefinidos. Un tipo *escalar* no tiene componentes internas; un tipo *compuesto* tiene otras componentes internas que pueden ser manipuladas individualmente. Un tipo de *referencia* almacena valores, llamados *punteros*, que designan a otros elementos de programa. Un tipo *lob* (large object) especifica la ubicación de un tipo especial de datos que se almacenan de manera diferente.

En la siguiente figura se muestran los diferentes tipos de datos predefinidos y disponibles para ser utilizados.

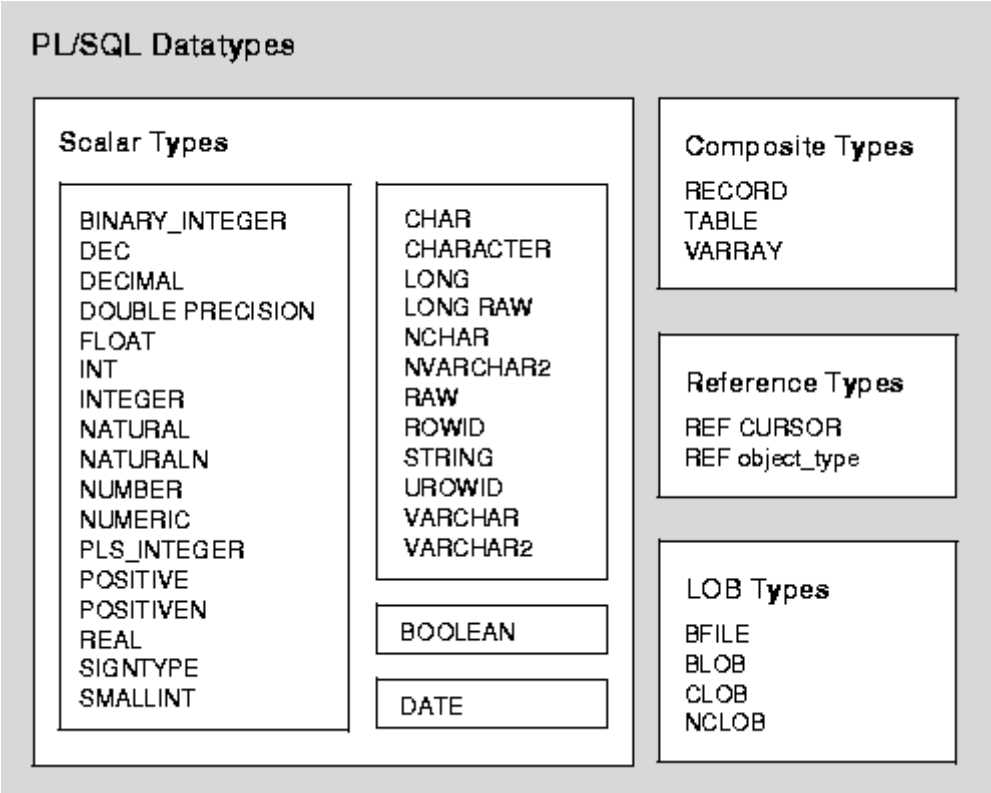

Figura Tipos de datos de PL/SQL

Uso de %TYPE

El atributo %TYPE define el tipo de una variable utilizando una definición previa de otra variable o columna de la base de datos.

Ejemplo:

```
DECLARE
credito        REAL(7,2);
debito         credito%TYPE;
…
```

También se podría declarar una variable siguiendo el tipo de un campo de alguna tabla, como por ejemplo en:

```
debito         cuenta.mora%TYPE;
```

La ventaja de esta última forma es que no es necesario conocer el tipo de dato del campo "mora" de la tabla "cuenta", manteniendo la independencia necesaria para proveer más flexibilidad y rapidez en la construcción de los programas.

ESTRUCTURAS DEL LENGUAJE

Control Condicional: Sentencia IF

A menudo es necesario tomar alternativas de acción dependiendo de las circunstancias. La sentencia IF permite ejecutar una secuencia de acciones condicionalmente. Esto es, si la secuencia es ejecutada o no depende del valor de la condición a evaluar. Existen tres modos para esta instrucción: IF – THEN, IF – THEN – ELSE y IF – THEN – ELSIF.

IF – THEN

Este es el modo más simple y consiste en asociar una condición con una secuencia de sentencias encerradas entre las palabras reservadas THEN y END IF (no ENDIF).

Ejemplo:

```
IF condición THEN
        secuencia_de_sentencias
END IF;
```

La secuencia de sentencias es ejecutada sólo si la condición es verdadera. Si la condición es falsa o nula no realiza nada. Un ejemplo real de su utilización es la siguiente:

```
IF condición THEN
        calcular_bonus (emp_id) --llamada a un procedimiento creado
        UPDATE sueldos SET pago = pago + bonus WHERE emp_no = emp_id;
END IF;
```

IF – THEN – ELSE

Esta segunda modalidad de la sentencia IF adiciona una nueva palabra clave: ELSE, seguida por una secuencia alternativa de acciones:

```
IF condición THEN
        secuencia_de_sentencias_1
ELSE
        secuencia_de_sentencias_2
END IF;
```

La secuencia de sentencias en la cláusula ELSE es ejecutada solamente si la condición es falsa o nula. Esto implica que la presencia de la cláusula ELSE asegura la ejecución de alguna de las dos secuencias de estamentos. En el ejemplo siguiente el primer UPDATE es ejecutado cuando la condición es verdadera, en el caso que sea falsa o nula se ejecutará el segundo UPDATE:

```
IF tipo_trans = 'CR' THEN
        UPDATE cuentas SET balance = balance + credito WHERE …
ELSE
        UPDATE cuentas SET balance = balance – debito WHERE …
END IF;
```

Las cláusulas THEN y ELSE pueden incluir estamentos IF, tal como lo indica el siguiente ejemplo:

```
IF tipo_trans = 'CR' THEN
        UPDATE cuentas SET balance = balance + credito WHERE …
ELSE
        IF  nuevo_balance >= minimo_balance THEN
                UPDATE cuentas SET balance = balance – debito WHERE …
        ELSE
                DBMS_OUTPUT.PUT_LINE('fondos_insuficientes';
        END IF;
END IF;
```

IF – THEN – ELSIF

Algunas veces se requiere seleccionar una acción de una serie de alternativas mutuamente exclusivas. El tercer modo de la sentencia IF utiliza la clave ELSIF (no ELSEIF) para introducir condiciones adicionales, como se observa en el ejemplo siguiente:

```
IF condición_1 THEN
        secuencia_de_sentencias_1
ELSIF condición_2 THEN
        secuencia_de_sentencias_2
ELSE
        secuencia_de_sentencias_3
END IF;
```

Si la primera condición es falsa o nula, la cláusula ELSIF verifica una nueva condición. Cada sentencia IF puede poseer un número indeterminado de cláusulas ELSIF; la palabra clave ELSE que se encuentra al final es opcional.

Las condiciones son evaluadas una a una desde arriba hacia abajo. Si alguna es verdadera, la secuencia de sentencias que corresponda será ejecutada. Si cada una de las condiciones analizadas resultan ser falsas, la secuencia correspondiente al ELSE será ejecutada:

```
BEGIN
        …
        IF  sueldo > 50000 THEN
                bono : = 1500;
        ELSIF sueldo > 35000 THEN
                bono : = 500;
        ELSE
                bono : = 100;
        END IF;
        INSERT INTO sueldos VALUES (emp_id, bono, );
END;
```

Si el valor de sueldo es mayor que 50000, la primera y segunda condición son verdaderas, sin embargo a *bono* se le asigna 1500, ya que la segunda condición jamás es verificada. En este caso sólo se verifica la primera condición para luego pasar el control a la sentencia INSERT.

Control Condicional: Sentencia CASE

Formato:

```
CASE
  WHEN condición THEN
  WHEN condición THEN
  ELSE
END CASE;
```

Ejemplo:

```
declare
        x number;
        r varchar(100);
        begin
        x:=1;
       CASE
          WHEN x=1 THEN
         r:='aa';
          dbms_output.put_line (r);
          WHEN x=2 THEN
         r:='bb';
          dbms_output.put_line (r);
          ELSE r:='cc';

        END CASE;
end;
```

Sentencia SELECT en PL/SQL

Formato:

```
SELECT lista de clolumnas
INTO variable[, variable, variable...]
FROM tabla
WHERE condición;
```

- El SELECT almacenará los valores que obtenga en las variables indicadas tras INTO y en el mismo orden.
- Es obligatorio incluir la cláusula INTO.
- El SELECT debe prepararse para que sólo devuelva una fila.

Ejemplo:

```
DECLARE
v_num number(10);
v_anexo  departamento.anexo%type;
BEGIN
SELECT numero, anexo INTO v_num, v_anexo
FROM departamento
WHERE nombre='Informática';
------
```

SENTENCIA INSERT EN PL/SQL

Formato:

```
INSERT INTO <nombre-tabla> (<columna1>, <columna2>.....) VALUES (<valor1>, <valor2>....)
```

SENTENCIA UPDATE EN PLSQL

Formato:

```
UPDATE  <nombre-tabla>
        SET  <columna1> = valor1  [, <columna2> = valor2 ...]
[WHERE <condición>]
```

SENTENCIA DELETE EN PL/SQL

Formato:

```
DELETE FROM <nombre-tabla>
[WHERE <condición>]
```

Si no se pone condición de selección, borra todas las filas de la tabla.

Observación:

La sintaxis de INSERT, UPDATE Y DELETE es igual en SQL que en PL/SQL

SALIDA DE DATOS

Sintaxis

DBMS_OUTPUT.PUT_LINE(‘Mensaje’ || variable);

Ejemplo:

DBMS_OUTPUT.PUT_LINE(‘El nombre del usuario es : ‘|| nombre);

EJEMPLO

```
declare
v_num number(10);
v_anexo  departamento.anexo%type;
BEGIN
SELECT numero, anexo INTO v_num, v_anexo
FROM departamento
WHERE nombre='Informática';

DBMS_OUTPUT.PUT_LINE('El nombre del departamento es: ' ||v_anexo);
DBMS_OUTPUT.PUT_LINE('El numero del departamento es  : ' ||v_num);
END;
```