UT6: PL/SQL

Contenido

Introducción

Tenemos una sintaxis propia en Oracle, y tiene cosas comunes con otros entornos
Es un lenguaje procedimental (le dices los pasos que tienes que seguir) que soporta todos los comandos y tipos de datos y operadores de SQL.
Desde PL/SQL se pueden usar ordenes de DML (Delete/Update/Insert/Select) pero no se puede usar ordenes DDL ni DCL

Bloques PL/SQL

Formato Genérico:

```
[DECLARE
    declaraciones]              --> Declaración de Objetos
BEGIN
    <ordenes>                   --> Conjunto de Instrucciones
EXCEPTION
    <gestión de excepciones>]   --> Tratamiento de Excepciones
END;
```

Ejemplo de uso:

```
DECLARE
    anios NUMBER(2);
    f_alta DATE;
    incremento NUMBER(4,3);
BEGIN
    SELECT fecha_alta INTO f_alta
        FROM USUARIOS
        WHERE num_socio = 122;
    anos:= TRUNC(MONTHS_BETWEEN(SYSDATE, f_alta) / 12);

    IF anos > 6 THEN
        incremento := 1,02;
    ELSIF anos > 4 THEN
        incremento := 1.06;
    ELSE
        incremento := 1,04;
    ENDIF;
END;
```

Declaración de variables en DECLARE

Sigue la estructura <nombre> <tipo> [NOT NULL] [(:= | DEFAULT) VALOR]

Ejemplos:

```
DECLARE
    iva NUMBER(4,2); --No inicia el valor
    aprobado BOOLEAN DEFAULT FALSE; --Lo inicia en falso
    pi CONSTANT NUMBER(7,6) := 3.1416;  --Lo inicia en 3.1416 y hace
que no sea editable
BEGIN
END;
```

Hay que tener en cuenta que se pueden anidar begin dentro de otros, metiéndolos entre ellos

Estructuras repetitivas de PL/SQL:

Se parecen en estructura al Pseudocódigo inglés (por eso es importante saber de todo un poco en cuanto programación se refiere)

Bucle While:

Estructura:

```
WHILE CONDICION LOOP
     [instrucciones]
END LOOP;
```

Funciona igual que en Java: Mientras se cumpla la condición de arriba, seguirá en el bucle

Ejemplo:

```
WHILE CONTADOR<10 LOOP
    CONTADOR:=CONTADOR+1;
END LOOP;
```

Bucle Do-While:

Estructura:

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

La condición aparece en el EXIT WHEN. La diferencia se encuentra en que, por ejemplo, en que si tenemos un contador que llega hasta 10 en el WHEN y estamos imprimiendo todos los numeros, no nos imprimirá el número 10. El DO_WHILE en las mismas condiciones escribirá el 10.

Ejemplo:

```
LOOP
    CONTADOR:=CONTADOR+1;
EXIT WHEN CONTADOR=10;
END LOOP;
```

Bucle For:

Estructura:

```
FOR variable IN [REVERSE] valor1..valor2 LOOP
    [instrucciones]
END LOOP;
```

Su lectura es literalmente: “Para una variable que va desde valor1 hasta valor2 hacer instrucciones”. Es un for que siempre va en paso 1, decidiendo si es positivo o negativo con REVERSE.

Ejemplo de uso:

```
FOR CONTADOR IN REVERSE 20..10 LOOP
    DBMS_OUTPUT.PUT_LINE(CONTADOR) --Imprime el valor
END LOOP;
```

Estructuras de control alternativas en PL/SQL:

Hay 3 formas principales, siendo equivalente a las estructuras if, if-else-if y switch

Estructura If:

Estructura:

```
IF (condición) THEN
[Instrucciones]
END IF;
```

Estructura If-Else-If

Estructura:

```
IF condicion2 THEN
    [instrucciones];
ELSIF condicion2 THEN
    [instrucciones]
ELSE
    [instrucciones]
END IF;
```

Estructura switch (CASE)

Este tiene 2 formas de hacerse, una que hace que escribas una condición completa (CASE 1), y otra que hace que tengas que escribir la variable al principio y los diferentes valores que puede tomar (CASE 2)

CASE 1: Estructura:

```
CASE
     WHEN condicion1 THEN instrucciones;
      WHEN condicion2 THEN instrucciones;
      WHEN condicion3 THEN instrucciones;
    [...]
      ELSE instrucciones;
END CASE;
```

CASE 2: Estructura:

```
CASE VARIABLE
    WHEN valor1 THEN
        instrucciones;
    WHEN valor2 THEN
        instrucciones;
    WHEN valor3 THEN
        instrucciones;
    ELSE
        instrucciones;
END CASE;
```

Tipos de Bloques en PL/SQL:

Bloques Anónimos:

Son bloques sin nombres, son los que empiezan con DECLARE.

```
[DECLARE
    declaraciones]              --> Declaración de Objetos
BEGIN
    <ordenes>                   --> Conjunto de Instrucciones
END;
```

Subprogramas:

Tienen nombre y se usan por llamadas, empiezan todos por IS o AS. Hay varios tipos:

- **Procedimientos:** Se almacenan en la base de datos y no se devuelven los valores

-BLOQUE ANONIMO: No tienen nombre, empieza con DECLARE

-SUBPROGRAMAS: Tienen nombre y empiezan por la palabra IS o AS

-Procedimientos: Se almacena en la base de datos

-Funciones: Su formato es similar a los procedimintos pero devuelven valor

-TRIGGERS: Se ejecutan automáticamente cuando ocurre algún evento

Subprogramas

Son bloques de PLSQL que tienen nombre, reciben parámetros y pueden devolver valores.

Permiten invocarlos desde otros subprogramas o herramientas

Para ver todos los subprogramas:

```
SELECT * FROM USER_OBJECTS;
```

Procedimientos

FORMATO:

```
CREATE [OR REPLACE] PROCEDURE <nombre procedimiento> [lista de
parametros]
    IS
        <Declaraciones>
    BEGIN
        <Instrucciones>;
    EXCEPTION
        <excepciones>;
    END [<nombre de procedimiento>];
```

INVOCACION:

```
    EXECUTE <nombre> [parametros]
```

Funciones

FORMATO:

```
CREATE OR REPLACE FUNCTION nombreFuncion (parametros)
RETURN tipoDeParametro
IS
[Declaración de variables]
BEGIN
    [Instrucciones SQL]
    RETURN valor; -->Tiene que ser del mismo tipo que arriba
END num_empleados;
```

Solo tiene una auténtica utilidad cuando se la llama desde otra función o un procedimiento.

Cursores

Son elementos que sacan la información de tablas de la base de datos.

- Cursores Implícitos (SELECT .. INTO ..): Solo pueden almacenar un valor, si la consulta da m s de un resultado o no da ninguno salta una excepci n.
- Cursores Explícitos: Los crea el programador y los utiliza por medio de sentencias

Cursores explicitos:

Declaración:

```
CURSOR nombre IS SELECT (sentencia);
```

Apertura:

```
OPEN nombre;
```

Búsqueda de información:

```
FETCH nombre INTO {valor|lista de valores};
```

Cierre de cursor:

```
CLOSE nombre;
```

Hay varias variables que se pueden usar:

- %FOUND: Devuelve true si ha encontrado un valor
- %NOTFOUND: Devuelve true si no encuentra valores
- %ROUCOUNT: Devuelve el n mero de línea que corresponde a la búsqueda actual
- %ISOPEN: Devuelve true si actualmente está el cursor abierto, pero Oracle siempre lo cierra después de cada orden SQL, así que siempre da falso

Además, con la declaración de **una variable %ROWTYPE** del cursor, se puede recorrer un bucle for similar al forEach de Java

```
DECLARE
    CURSOR CUR IS [SENTENCIA SELECT];
    v_reg CUR%ROWTYPE; --Creación del elemento del cursor
BEGIN
    FOR v_reg IN CUR LOOP
    [instrucciones];
    END LOOP;
END;
```

Cursores con parámetros

DECLARACIÓN:

```
DECLARE
CURSOR nombre (parametro1, parametro2...) IS SELECT (sentencia
select donde se usan los parámetros)
BEGIN
[instrucciones]
END;
```

Los parámetros después del cursor tienen la siguiente forma:

```
 nombrevariable [IN] tipoDato [{:= | DEFAULT} valor]
```

Si se abren o se utilizan en bucles for, se tiene que declarar los valores al principio

Uso de cursores para actualizar filas:

Hasta ahora los cursores eran para seleccionar datos, pero también se puede usar el nombre de un cursor que apunte a una fila para realizar una actualización.

FOR UPDATE: Para que las filas seleccionadas se actualicen o borren.

```
CURSOR nombre IS SELECT ... FOR UPDATE
```

Con **CURRENT OF** en la cláusula where se actualizará o borrará la última fila del FETCH del cursor.

Para actualizar X columna de un cursor es:

```
 CURSOR nombre IS SELECT ... FOR UPDATE OF columna
```

Excepciones

Sirven para tratar errores en tiempo de ejecución y otras situaciones definidas por el usuario.

El bloque de excepción se crea entre begin y end, y en un mismo bloque se puede meter varias excepciones

Formato:

```
EXCEPTION
    WHEN nombreExcepcion THEN
        [...];
    WHEN nombreExcepcion2 THEN
        [...];
  WHEN OTHERS THEN
        [...];
END;
```

Hay dos tipos de excepciones:

- Las excepciones por defecto de ORACLE:
- Las excepciones definidas por el usuario:
    - Se declaran en DECLARE:

```
nombreExcepcion EXCEPTION;
```

    - Se disparan o levantan en la sección ejecutable del programa.

```
RAISE nombreExcepcion;
```

    - Se tratan en el bloque de excepción:

```
EXCEPTION
     WHEN nombreException THEN
        [...];
END;
```

Triggers o disparadores:

Son bloques PL/SQL que se ejecutan o disparan automáticamente cuando hay algún suceso como Insert, Update o Delete en tablas.

Cuando se dispara un trigger, forma parte de la operación de actualización que lo disparó.

Es decir, si el trigger falla, oracle dará como fallida la actualización y automáticamente se hará un rollback

FORMATO:

```
CREATE [OR REPLACE] TRIGGER nombre
{BEFORE|AFTER|INSTEAD OF} {insert, delete, update [OF COL] [OR
{insert...]
ON nombretablaovista
[REFERENCING OLD AS nombreold, NEW AS nombrenew]
[{FOR EACH ROW [WHEN (condicion)] | FOR EACH STATEMENT}]
[DECLARE]
BEGIN
[EXCEPTION]
END;
```

Valores new y old

Hacen referencia a los valores anterior y posterior de una actualización de una fila.

Sintaxis -> :old.(nombrecolumna)y :new.(nombrecolumna)

A new y old se le puede poner otros alias poniendo REFERENCING new AS _ , old AS _, aunque no es lo mejor que se recomienda.

Disparadores de sustitucion (instead of)

Se pueden crear triggers que se ejecuten en lugar de la orden de manipulación (insert, update o delete)

Solo se usan en triggers asociados a vistas y son siempre a nivel de fila.

No se pueden especificar restricciones de disparo con WHEN

Disparadores del sistema:

Se disparan cuando ocurre un evento del sistema (arranque o parada de base de datos, conexión o desconexión de un usuario...) o una instrucción del DDL

Hay que tener el privilegio de ADMINISTER DATABASE TRIGGER para editarlos

Sintaxis:

```
CREATE OR REPLACE TRIGGER nombre
{BEFORE|AFTER} {lista_eventos_definicion |
lista_eventos_sistema}
ON {DATABASE | CHEMA} [WHEN condicion]
    [BLOQUE PL/SQL]
```

La lista de eventos de definición son las instrucciones DDL

La lista de eventos de sistema son los eventos del sistema (arranque o parada de base de datos, etc)

ON DATABASE se dispara siempre que ocurre el evento en toda la base de datos

ON CHEMA se dispara solo si el evento se produce en el esquema del trigger