Apuntes PL-SQL

Triggers II

Tipos de datos compuestos

Registros PL/SQL:

- Son similares a un tipo struct en C o a un tipo Registro en otros 3GL.
- Se componen de otros datos más simples.
- Son adecuados para recuperar una fila de datos de una tabla, pero pueden usarse para crear arrays de estructuras.
- A cada campo se le puede asignar un valor inicial y pueden definirse como NOT NULL.
- Ya hemos usado registros cuando hemos declarado variables usando %ROWTYPE.

Definición de Registros en PL/SQL

- La sintaxis para crear un tipo registro es la siguiente:

```
TYPE NombreTipoDatos IS RECORD
(
    NombreCampo TipoCampo [NOT NULL] [:= ValorInicial],
    NombreCampo TipoCampo [NOT NULL] [:= ValorInicial],
    …
);
```

- En TipoCampo se puede usar %TYPE.

- Una vez creado este tipo de datos, ya pueden declararse variables de este tipo (igual que con las struct de C).

Definición de Registros en PL/SQL

- La definición de tipos de datos compuestos y la declaración de variables del nuevo tipo se realizan en la zona de declaraciones del bloque PL/SQL, así:

```
DECLARE

    TYPE TipoRegEmpleado IS RECORD
    (
            nombre     emp.ename%TYPE,
            oficio                  emp.job%TYPE,
            salario                 emp.sal%TYPE
    );

    registroempleado TipoRegEmpleado;
…
```

Para referirnos a cada parte de una variable tipo registro usaremos la sintáxis:

nombrevariable.nombrecampo

Arrays o tablas PL

- No tienen nada que ver con las tablas de la base de datos, son tablas que se guardan en memoria.
- Es similar al concepto de array en C.
- Tienen dos partes: el índice o clave y los datos, que pueden ser de tipo simple o de tipo registro.
- El índice no tiene porque ser secuencial y los elementos no se guardan contiguos en memoria (se parece más a una lista que a un array). Además, no tienen una dimensión predeterminada, pueden crecer en tiempo de ejecución.

Declaración de tablas PL

- Se hace en dos pasos: Primero se crea un tipo de datos Tabla y luego se declaran variables de ese tipo, así:

```
--Declaro el tipo
TYPE TipoTablaNombresEmpleados IS TABLE OF emp.ename%TYPE
INDEX BY BINARY_INTEGER;

-- Declaro la variable
MiTabla TipoTablaNombresEmpleados;
```

Operaciones con tablas PL

- Un elemento se crea cuando se le asigna un valor, así:

  MiTabla(8):='SMITH';

  crea el elemento de índice 8 de la tabla. Los índices pueden ser negativos o positivos.

- Para referirnos a un elemento, ponemos el índice entre paréntesis tras el nombre de la tabla.

- Si intento leer un elemento de la tabla que no existe, se levanta una excepción NO_DATA_FOUND

- Para trabajar con tablas, existen una serie de métodos que nos permiten realizar todas las operaciones necesarias, se llaman con la sintaxis habitual:

  NombreTabla.NombreMétodo(parámetros)

Métodos de las tablas PL

- Son los siguientes:
  - EXISTS(n): Devuelve TRUE si existe el elemento n.
  - COUNT: Devuelve el número de elementos de la tabla.
  - FIRST: Devuelve el índice del primer elemento.
  - LAST: Devuelve el índice del último elemento.
  - PRIOR: Devuelve el índice del elemento anterior. Si ya está en el primero devuelve NULL
  - NEXT: Devuelve el índice del elemento siguiente. Si ya está en el último devuelve NULL
  - DELETE: Borra todos los elementos.
  - DELETE(n): Borra el elemento de índice n.

Ejemplo de creación y recorrido de una tabla PL

```
 TYPE tEmpleados IS RECORD
 (
  NUMEMP  EMP.EMPNO%TYPE,
  NUMDEPT                EMP.DEPTNO%TYPE,
  PUESTO                 EMP.JOB%TYPE,
  JEFE                   EMP.MGR%TYPE
 );

TYPE tTablaEmpleados IS TABLE OF tEmpleados
INDEX BY BINARY_INTEGER;

 empleados tTablaEmpleados;

i NUMBER;

BEGIN
      rellenar_tabla(empleados);

      FOR i IN empleados.FIRST..empleados.LAST LOOP
            DBMS_OUTPUT.PUT_LINE(empleados(i).numemp||empleados(i).numdept);
      END LOOP;
END;
```

Ejemplo de uso de una tabla PL

- Veamos como sería el procedimiento rellenartabla, que rellena una tabla PL a partir de una tabla de la base de datos:

```
DECLARE
    CURSOR c_empleados IS
        SELECT EMPNO, DEPTNO, JOB, MGR FROM EMP;
    I NUMBER:=0;
    v_emp empleados%ROWTYPE;
BEGIN
    OPEN empleados;
    FETCH c_empleados INTO v_emp;
    WHILE  c_empleados%FOUND LOOP
        empleados(I).NUMEMP := v_emp.EMPNO;
        empleados(I).NUMDEPT := v_emp.DEPTNO;
        empleados(I).PUESTO := v_emp.JOB;
        empleados(I).JEFE := v_emp.MGR;
        I := I + 1;
        FETCH c_empleados INTO v_emp;
    END LOOP;
    CLOSE c_empleados;
END;
```

Paquetes PL/SQL

- Parten de un concepto similar al de las librerías de objetos o de funciones que hay en C++ y C respectivamente.
- Son una forma de agrupar procedimientos, funciones, tipos de datos, cursores, variables, etc… que están relacionados entre si.
- Al igual que pasa con las librerías en C, existen algunos paquetes que ya han sido programados por ORACLE y cuyas funciones y procedimientos podemos usar, por ejemplo, el paquete DBMS_OUTPUT.
- Para referirnos a un objeto de un paquete desde fuera del mismo, usamos la notación NombrePaquete.NombreObjeto, por ejemplo, cuando ponemos DBMS_OUTPUT.PUT_LINE nos referimos al procedimiento PUT_LINE del paquete DBMS_OUTPUT.

Creación de Paquetes PL/SQL

- Los paquetes PL/SQL tienen dos partes: cabecera o especificación del paquete y cuerpo del paquete.
- En la cabecera se declaran las partes públicas del paquete (es como un .h en C), pudiendo ser: prototipos de procedimientos y funciones, cursores, tipos de datos, variables, excepciones, etc... Todo lo que se declara aquí es accesible desde fuera del paquete.
- En el cuerpo del paquete se define el código de los procedimientos y funciones declarados en la cabecera y se incluyen declaraciones de objetos privados que no son accesibles desde fuera del paquete.

Sintaxis de la creación de paquetes

- Para crear la cabecera:

```
CREATE OR REPLACE PACKAGE NombrePaquete AS
    Declaraciones de tipos y variables públicas
    Declaraciones de cursores y excepciones públicas
    Prototipos de procedimientos y funciones públicas
END;
```

Sintaxis de la creación de paquetes

- Para crear el cuerpo del paquete:

```
CREATE OR REPLACE PACKAGE BODY NombrePaquete AS

    Declaraciones de tipos y variables privados
    Declaraciones de cursores y excepciones privados
    Definición del código de los procedimientos y funciones públicas
[BEGIN
    Instrucciones de inicialización ]

END;
```

Aspectos importantes de los paquetes

- Las variables que se declaran en la cabecera de un paquete mantienen su valor durante toda la sesión, se llaman variables persistentes.
- Los objetos del paquete se cargan en memoria cuando se referencia alguno de ellos por primera vez y se descargan al cerrar la sesión.
- Cuando se crean los objetos valen NULL, a no ser que se añadan instrucciones de inicialización en el cuerpo del paquete.
- Para referirme a un objeto del paquete desde un procedimiento del paquete no tengo que poner más que su nombre, pero para hacerlo desde fuera debo anteponer el nombre del paquete al del objeto.
- Se permite el uso de procedimientos sobrecargados.

Creación de cursores en paquetes

- El uso de cursores dentro del paquete es un poco especial, veamos como se hace:

- En la cabecera se pone el nombre del cursor y el tipo de datos que devuelve, así:

  CURSOR c_emp RETURN emp%ROWTYPE;

- En el cuerpo, se completa la declaración:

  CURSOR c_emp RETURN emp%ROWTYPE
  SELECT * FROM emp;

Algunos paquetes de ORACLE

- ORACLE incorpora una serie de paquetes que facilitan al programador su tarea (al igual que pasa en C con stdio, string, conio, stdlib, etc…)

- Algunos de ellos son:
  - DBMS_STANDARD
  - DBMS_OUTPUT
  - DBMS_MAIL
  - DBMS_JOB
  - DBMS_SQL
  - DBMS_SESSION
  - DBMS_HTML

- Vamos a verlos con un poco más de detalle.

Algunos paquetes de ORACLE

DBMS_STANDARD

- Este paquete se carga por defecto y contiene, por ejemplo, el procedimiento RAISE_APPLICATION_ERROR o las funciones SQL que hemos estudiado (ABS, TO_CHAR, etc...)

- Es el único paquete cuyos objetos se pueden usar desde cualquier parte sin anteponer el nombre del paquete.

DBMS_OUTPUT

- Tiene procedimientos para leer y escribir el buffer de pantalla (por ejemplo PUT_LINE). Estas funciones solo se suelen usar para mostrar mensajes de depuración.

Más paquetes de ORACLE

- **DBMS_MAIL**
  Permite la preparación y el envío de mensajes de correo electrónico.

- **DBMS_JOB**
  Permite lanzar trabajos batch aprovechando las horas en las que no hay gente trabajando contra la base de datos.

- **DBMS_SQL**
  Permite el uso de SQL dinámico, es decir, permite la ejecución de instrucciones DDL desde un procedimiento PL/SQL.

- **DBMS_SESSION**
  Permite ejecutar desde un procedimiento instrucciones para cambiar las variables de entorno y de sesión.

- **DBMS_HTML**
  Permite generar código HTML de forma dinámica. Para funcionar necesita que esté instalado Oracle 9i Application Server.

El problema de las tablas mutantes

- ¿En qué consiste?

  Si recordamos las restricciones que tenían los triggers:

  - Por sentencia: No pueden hacer referencia a los valores :old y :new
  - Por fila: No pueden hacer una SELECT sobre la tabla que los disparó.
    Si se intenta hacer, ORACLE nos devolverá un error en el que nos dice que no se puede consultar una tabla que está mutando.

  Veremos que hay muchos problemas que aún no podemos resolver, como por ejemplo los ejercicios 3 y 5 de la práctica.

  En general, esto nos va a pasar con todos aquellos problemas que exijan controlar si se cumple alguna condición entre los datos que estamos insertando y los que ya se encuentran en la tabla.

Un ejemplo de problema de tablas mutantes…

- Para hacernos una idea de cuando se presenta este problema, vamos a ver un ejemplo de una situación que no podíamos resolver hasta ahora, como hacer un trigger para:

  - Impedir que algún empleado gane más que el presidente o que el manager de su departamento.

    Si hago el trigger por sentencia, no podré acceder al valor del sueldo del nuevo empleado.

    Si hago el trigger por fila, no podré consultar en la tabla emp cual es el sueldo del presidente o el del manager de su departamento.

Resolución de problemas de tablas mutantes

La solución a este problema es sencilla (si alguien te la cuenta…), no vamos a hacer un único trigger, sino dos:

- Un trigger por sentencia: Este trigger buscará la información necesaria en la tabla que dispara el trigger y la guardará en una o varias variables persistentes, esto es, variables declaradas dentro de un paquete.

- Un trigger por fila: Este trigger por fila comparará el valor del nuevo registro con el almacenado en las variables persistentes y verá si la operación se puede realizar o no, levantando en su caso el correspondiente error de aplicación.

Pasos concretos a dar para resolver estos problemas…

1. Leer bien los requisitos del problema e identificar la información de la tabla que debo guardar en variables persistentes.

2. Crear el paquete declarando los tipos de datos y las variables necesarias para guardar dicha información.

3. Hacer un trigger before por sentencia que rellene dichas variables consultando la tabla mutante.

4. Hacer un trigger before por fila que compruebe si el registro que se está manejando cumple la condición especificada consultando las variables persistentes.

5. En muchas ocasiones, el trigger por fila también tendrá que ir actualizando la información que está en las variables.

Resolución del ejemplo anterior paso a paso

- Paso 1: Leer requisitos e identificar la información que voy a necesitar guardar.

  El problema pide que ningún empleado gane más que el presidente o que el jefe de su departamento.

  Parece lógico guardar en sendas variables el sueldo del presidente y el de cada uno de los jefes de departamento.

  Así, las variables que voy a necesitar son dos:

  sueldo_presi Tipo Number

  sueldos_jefes Tipo Tabla, en la que cada fila tendrá un número de departamento y el salario de su jefe, así:

DEPTNO	SAL_JEFE
10	1800
20	1300
30	1500

Resolución del ejemplo anterior paso a paso

- **Paso 2: Crear el paquete con tipos de datos y variables necesarias.**

```
CREATE OR REPLACE PACKAGE ControlSueldos
AS

SueldoPresi NUMBER;                                    -- aquí guardo el sueldo del presidente

 TYPE tRegistroTablaSueldos IS RECORD                  --defino el tipo de datos registro
 (
     DEPTNO          EMP.DEPTNO%TYPE,
     SAL_JEFE        EMP.SAL%TYPE
 );

TYPE tTablaSueldosJefes IS TABLE OF tRegistroTablaSueldos  -- defino el tipo de datos tabla
INDEX BY BINARY_INTEGER;

SueldosJefes tTablaSueldosJefes;                -- declaro una variable del tipo tabla antes creado

END ControlSueldos;
/
```

Nota: No necesito crear un cuerpo del paquete puesto que no he incluido procedimientos ni funciones.

Resolución del ejemplo anterior paso a paso

- **Paso 3: Hacer un trigger before sentencia que consulte la tabla y rellene las variables del paquete.**

```
CREATE OR REPLACE TRIGGER RELLENARVARIABLES
BEFORE INSERT OR UPDATE ON EMP
DECLARE
    CURSOR CUR_SUELDOS_JEFES IS        SELECT DEPTNO, SAL
                                       FROM EMP
                                       WHERE job='MANAGER'
                                       ORDER BY DEPTNO;
    INDICE NUMBER:=0;

BEGIN
    SELECT SAL INTO ControlSueldos.SueldoPresi -- busco el sueldo del presidente
    FROM EMP
    WHERE job='PRESIDENT';

    -- relleno la tabla de sueldos de los jefes de cada departamento
    FOR V_CUR IN CUR_SUELDOS_JEFES LOOP
        ControlSueldos.SueldosJefes(INDICE).DEPTNO := V_CUR.DEPTNO;
        ControlSueldos.SueldosJefes(INDICE).SAL_JEFE := V_CUR.SAL;
        INDICE := INDICE + 1;
    END LOOP;

END RELLENARVARIABLES;
/
```

Resolución del ejemplo anterior paso a paso

- Paso 4: Hacer un trigger before fila que compruebe si los registros introducidos cumplen las condiciones.

```
CREATE OR REPLACE TRIGGER ControlarSueldos
BEFORE INSERT OR UPDATE ON EMP
FOR EACH ROW
DECLARE
    SueldodelJefe NUMBER;
BEGIN
    SueldodelJefe := ControlSueldos.BuscarSueldoJefe (:new.deptno);
    IF :new.sal > SueldodelJefe THEN
        RAISE_APPLICATION_ERROR(-20001,'No puede ganar más que su jefe');
    END IF;
    IF :new.sal > ControlSueldos.SueldoPresi THEN
        RAISE_APPLICATION_ERROR(-20002,'No puede ganar más que el presidente');
    END IF;
END;
```

Nota: Hemos usado una función BuscarSueldoJefe que buscará el sueldo del jefe del departamento del empleado que estamos insertando o modificando.

Lo más lógico es incluirla en el paquete que hemos diseñado en el paso 2.

Resolución del ejemplo anterior paso a paso

- Para incluir la función BuscarSueldoJefe, habrá que modificar la cabecera del paquete incluyendo la línea siguiente:

```
FUNCTION BuscarSueldoJefe(NumDep EMP.DEPTNO%TYPE) RETURN NUMBER;
```

- Después incluimos la definición de la función en el cuerpo del paquete, así que habrá que crear un cuerpo de paquete de la siguiente forma:

```
CREATE OR REPLACE PACKAGE BODY ControlSueldos
AS
    FUNCTION BuscarSueldoJefe (NumDep EMP.DEPTNO%TYPE) RETURN NUMBER
    IS
        I NUMBER :=0;
    BEGIN
        FOR I IN ControlSueldos.SueldosJefes.FIRST.. ControlSueldos.SueldosJefes.LAST LOOP
            IF ControlSueldos.SueldosJefes (I).DEPTNO = NumDep THEN
                RETURN ControlSueldos.SueldosJefes(I).SAL_JEFE;
            END IF;
        END LOOP;
    END BuscarSueldoJefe;

END ControlSueldos;
```

Resolución del ejemplo anterior paso a paso

- Paso 5: El trigger por fila debe ir actualizando la información que está en las variables persistentes.

  Esto es, si en la operación que dispara los triggers se modifica el sueldo de un jefe o del presidente, se debe cambiar en las variables para que en las siguientes filas la comprobación sea correcta.

  Para ello, el trigger por fila deberá modificar la variable SueldoPresi si :new.job es President y la tabla SueldosJefes si :new.job es Manager.

  La variable SueldoPresi la modificaremos directamente y la tabla SueldosJefes por medio de un procedimiento que reciba el número de departamento y el nuevo sueldo del jefe.

Resolución del ejemplo anterior paso a paso

- Así, al código del trigger por fila habrá que añadirle al final las siguientes líneas:

```
IF :new.job='PRESIDENT' THEN
        ControlSueldos.SueldoPresi := :new.sal;
END IF;
IF :new.job = 'MANAGER' THEN
        ControlSueldos.ActualizarSueldoJefe(:new.deptno, :new.sal);
END IF;
```

- Y al paquete habrá que añadirle el procedimiento ActualizarSueldoJefe, poniendo el prototipo en la cabecera y la definición en el cuerpo.

Resolución del ejemplo anterior paso a paso

- Así, hay que incluir en la cabecera la línea:

```
PROCEDURE ActualizarSueldoJefe(NumDep EMP.DEPTNO%TYPE, NuevoSueldo EMP.SAL%TYPE)
```

- Y en el cuerpo del paquete:

```
PROCEDURE ActualizarSueldoJefe(NumDep EMP.DEPTNO%TYPE, NuevoSueldo EMP.SAL%TYPE)
IS
    I NUMBER :=0;
BEGIN
    FOR I IN ControlSueldos.SueldosJefes.FIRST.. ControlSueldos.SueldosJefes.LAST LOOP
            IF ControlSueldos.SueldosJefes (I).DEPTNO = NumDep THEN
                ControlSueldos.SueldosJefes(I).SAL_JEFE:=NuevoSueldo;
            END IF;
    END LOOP;
END ActualizarSueldoJefe;
```

Probando triggers de tablas mutantes

- Es muy importante probar bien este tipo de triggers, puesto que es fácil equivocarse en el diseño.
- Para probar el trigger en Insert, es imprescindible hacer una consulta de datos anexados que haga saltar la restricción en alguna fila.
- Para probarlo en Update también hay que probar modificando registros que hagan cambiar los valores guardados en la tabla.