ORACLE PL/SQL

MÓDULO 05

ESTRUCTURAS DE DATOS

GUSTAVO CORONEL
desarrollasoftware.com

ORACLE PL/SQL

Contenido

INTRODUCCIÓN

Oracle provee dos Tipos de Datos Compuestos (Composite Data Types), estos son: Los Registros y Las Colecciones.

Un registro está compuesto por una serie de datos de diferentes, pero relacionados entre sí, muy semejante a una fila de una tabla.

Las Colecciones de PL/SQL son como los arreglos en otros Lenguajes de Programación como: C, C++ y Java. Por lo cual las podemos definir como un conjunto de datos homogéneos almacenados en forma consecutiva en memoria.

Al decir que son un conjunto de datos homogéneos se entiende que todos los valores deben ser del mismo tipo de dato, pero es prudente tener presente que dicho tipo de dato puede ser un tipo de dato compuesto, o sea, una Colección de otra Colección o de un Registro.

En síntesis, usamos las Colecciones de PL/SQL cuando queremos almacenar una lista de valores del mismo tipo de dato.

Por ejemplo, una lista con los nombres (VARCHAR2) de los empleados de la tabla EMPLOYEES. De igual manera, podrías tener una lista con todos los campos de la tabla COUNTRIES, en este caso definirías un Tipo Record equivalente a COUNTRIES%ROWTYPE y posteriormente una Colección de dicho tipo.

Al igual que los Tipos de Datos Escalares (VARCHAR2, NUMBER, etc), las colecciones pueden ser usadas como parámetros de entrada y/o salida de Procedimientos y Funciones, así como también pueden ser el valor de retorno de una función.

REGISTROS

Definición

Sintaxis:

```
type tipo_registro is record (
   campo1 tipo1 [not null] [:= valor1],
   campo2 tipo2 [not null] [:= valor2],
   --------------
   --------------
);
```

Declaración de variable

Sintaxis:

```
nombre_variable tipo_registro;
```

Acceso a los campos

Sintaxis:

```
nombre_variable.nombre_campo
```

Script 1

Consultar el nombre y salario de un empleado.

```
create or replace procedure pr105( cod emp.empno%type )
is
   type reg is record (
      nombre emp.ename%type,
      salario emp.sal%type
   );
   r reg;
begin
   select ename, sal into r
      from emp where empno = cod;
   dbms_output.put_line( 'Nombre: ' || r.nombre );
   dbms_output.put_line( 'Salario: ' || r.salario );
end;
```

Ejecución:

```
SQL> execute pr105( 7698 );
Nombre: BLAKE
Salario: 2850

PL/SQL procedure successfully completed.
```

%RowType

Se utiliza para declarar registros con la misma estructura de una tabla.

Sintaxis:

```
   NombreVariable NombreTable%RowType;
```

Script 2

Consultar los datos de un departamento.

```
create or replace procedure pr106( cod dept.deptno%type )
is
   r dept%rowtype;
begin
   select * into r
      from dept where deptno = cod;
   dbms_output.put_line('Codigo: ' || r.deptno);
   dbms_output.put_line('Nombre: ' || r.dname);
   dbms_output.put_line('Localización: ' || r.loc);
end;
```

Ejecución:

```
SQL> execute pr106(10);
Codigo: 10
Nombre: ACCOUNTING
Localización: NEW YORK

PL/SQL procedure successfully completed.
```

COLECCIONES

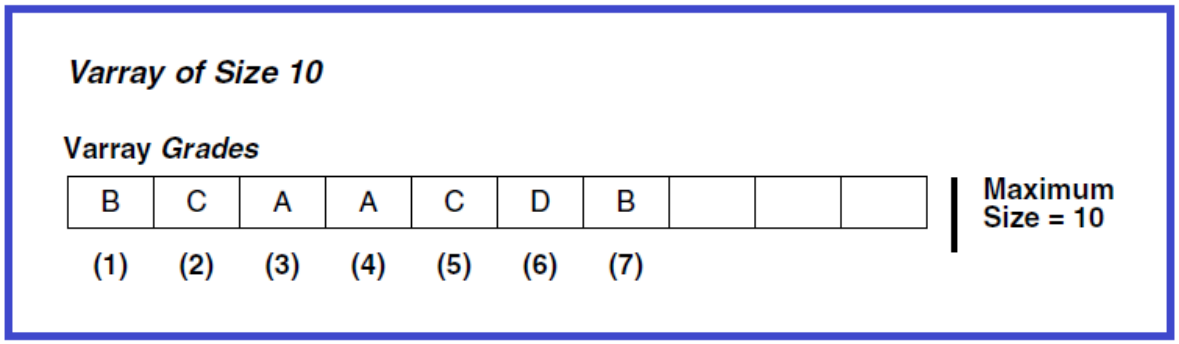
Métodos para Colecciones

METODO	DESCRIPCIÓN.
COUNT	Devuelve el número de elementos que contiene la Colección. En el caso de los VARRAYs, el valor de COUNT siempre es igual a LAST. Para las Tablas Anidadas y las Matrices Asociativas, COUNT es normalmente igual a LAST. A menos que se eliminen elementos del centro, si esto ocurre, COUNT sería menor que LAST.
DELETE	Solo puede ser usado con Tablas Anidadas y Matrices Asociativas. Tiene Tres variantes: DELETE Elimina todos los elementos de una Colección. DELETE(n) Elimina el elemento **n**. Si **n** es nulo, no hace nada. DELETE(m, n) Elimina todos los elementos del rango **m..n**. Si **m** es mayor que **n**, o si **m** o **n** es nulo no hace nada.
EXISTS	EXISTS(n) devuelve TRUE si existe el elemento n de una Colección, de lo contrario, devuelve FALSE. Puede utilizar EXISTS para evitar una excepción al hacer referencia a un elemento inexistente. Cuando se le pasa un subíndice fuera de rango, EXISTS devuelve FALSE en lugar de lanzar la excepción SUBSCRIPT_OUTSIDE_LIMIT.
EXTEND	Es un método exclusivo de las Tablas Anidadas y Los VARRAYs. Este procedimiento tiene tres formas: **EXTEND** Agrega un elemento nulo. **EXTEND(n)** Agrega **n** elementos nulos. **EXTEND(n, i)** Agrega **n** copias del elemento **i**. EXTEND opera sobre el tamaño interno de una Colección. Si EXTEND encuentra elementos eliminados, los incluye en su recuento.
FIRST	FIRST retorna el primer valor (más pequeño) de subíndice en una colección. Los valores de subíndice suelen ser enteros, pero en el caso de las Matrices Asociativas también pueden ser cadenas. Si la Colección está vacía, FIRST retorna NULL. Para Los VARRAYs, FIRST siempre retorna 1.
LAST	LAST retorna el último valor (más grande) de subíndice en una Colección. Al igual que FIRST, si la Colección está vacía, LAST retorna NULL. Para Los VARRAYs, LAST siempre es igual a COUNT. Para las Tablas Anidadas y las Matrices Asociativas, LAST es normalmente igual a COUNT. A menos que se eliminen elementos del centro, si esto ocurre, LAST sería mayor que COUNT.
PRIOR(n)	Retorna el número de índice que antecede al índice **n**.
NEXT(n)	Retorna el número de índice que sigue (sucede) al índice **n**.

LIMIT	Para las Tablas Anidadas y las Matrices Asociativas (No tienen un tamaño máximo), LIMIT retorna NULL. Para los VARRAYs, LIMIT retorna el número máximo de elementos que un VARRAY puede contener (Especificado en la definición de tipo).
TRIM	Es un método exclusivo de las Tablas Anidadas y los VARRAYs. Este procedimiento tiene dos formas: TRIM Elimina un elemento del final de una Colección. TRIM(n) Elimina **n** elementos del final de una Colección. Si **n** es mayor que COUNT, TRIM(n) lanza la excepción SUBSCRIPT_BEYOND_COUNT. TRIM opera en el tamaño interno de una Colección. Si TRIM encuentra elementos eliminados, los incluye en su recuento.

LOS VARRAYS

Entendiendo los ARRAYs:



Sintaxis:

```
TYPE NuevoTipoDeDato IS VARRAY(Tamaño) OF TipoDeDato;
```

Script 3

Ejemplo ilustrativo de cómo usar VARRAYs.

```
DECLARE
   -- Definimos los tipos de datos
   TYPE AlumnosArray IS VARRAY(5) OF VARCHAR2(100);
   TYPE NotasArray IS VARRAY(5) OF NUMBER(4);
   -- Definiendo las variables
   alumnos AlumnosArray;
   notas   NotasArray;
BEGIN
   -- Creando los arreglos
   alumnos := AlumnosArray('Gustavo','Lucero','Ricardo','Andrea','Laura');
   notas := NotasArray(20,18,16,10,15);
   -- Mostrando los arreglos
   FOR i IN 1 .. alumnos.count LOOP
      dbms_output.PUT_LINE( alumnos(i) || ' - ' || notas(i) );
   END LOOP;
END;
```

Ejecución:

```
Gustavo - 20
Lucero - 18
Ricardo - 16
Andrea - 10
Laura – 15
```

Script 4

Segundo ejemplo de cómo usar VARRAYs:

```
DECLARE
   -- Definimos los tipos de datos
   TYPE VARRAY_EMPLEADOS IS VARRAY(5000) OF HR.EMPLOYEES%ROWTYPE;
   -- Definiendo las variables
   V_EMPLEADOS VARRAY_EMPLEADOS;
   V_CONT NUMBER(8);
BEGIN
   V_EMPLEADOS := VARRAY_EMPLEADOS();
   DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
   FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
      V_EMPLEADOS.EXTEND;
      V_CONT := V_EMPLEADOS.COUNT;
      V_EMPLEADOS(V_CONT) := REC;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
   FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
      DBMS_OUTPUT.PUT_LINE( I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
   END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```

Matrices Asociativas (Associative Array)

Características:

- Son tablas exclusivas de PL/SQL, esto significa que pueden existir en estructuras de memoria de PL/SQL (Paquetes, Funciones, Procedimientos etc.) pero no pueden ser creadas como objectos / columnas de Base de Datos.
- Están compuestas de dos columnas a las cuales no es posible asignarles nombres:
  - ✓ La primera columna es el índice (index).
  - ✓ La segunda columna contiene el dato almacenado (value).
  - ✓ El índice es usado para localizar el dato almacenado en la segunda columna.
  - ✓ Los valores del índice pueden ser tanto negativos como positivos y no necesariamente tienen que ser insertados de forma secuencial o consecutiva, esto es, puede agregar el índice 4 antes que el 3.
- No son inicializadas al momento de su declaración.
- Tienen un tamaño dinámico, por lo cual pueden crecer tanto como sea necesario.

Sintaxis:

```
TYPE nuevo_tipo_dato IS TABLE OF
{
      column_type
      | variable%TYPE
      | table.column%TYPE} [NOT NULL]
      | table%ROWTYPE
}
INDEX BY {PLS_INTEGER|BINARY_INTEGER|VARCHAR2(<tamaño>);
```

Script 5

En el presente ejemplo se ilustra de una manera sencilla el uso de matrices asociativas:

```
DECLARE
   TYPE ARRAY_NOTAS IS TABLE OF NUMBER
   INDEX BY BINARY_INTEGER;
   NOTAS ARRAY_NOTAS;
BEGIN
   -- CARGAR NOTAS
   NOTAS(1) := 20;
   NOTAS(2) := 18;
   NOTAS(3) := 15;
   NOTAS(4) := 17;
   -- MOSTRAR NOTAS
   FOR I IN 1..NOTAS.COUNT LOOP
      DBMS_OUTPUT.PUT_LINE('NOTA ' || I || ': ' || NOTAS(I));
   END LOOP;
END;
```

Ejecución:

```
NOTA 1: 20
NOTA 2: 18
NOTA 3: 15
NOTA 4: 17
```

Script 6

Segundo ejemplo de arreglo asociativo.

```
DECLARE
   -- Definimos el tipo de dato
   TYPE ARRAY_EMPLEADOS IS TABLE OF HR.EMPLOYEES%ROWTYPE
   INDEX BY BINARY_INTEGER;
   -- Definiendo las variables
   V_EMPLEADOS ARRAY_EMPLEADOS;
BEGIN
   DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
   FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
      V_EMPLEADOS(V_EMPLEADOS.COUNT + 1) := REC;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
   FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
      DBMS_OUTPUT.PUT_LINE( I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
   END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```

Tablas Anidadas (Nested Table)

Características:

- Pueden ser declaradas en bloques de PL/SQL, así como también en la Base de Datos.
- Al igual que las Matrices Asociativas, las Tablas Anidadas tienen un tamaño dinámico y puede contener elementos vacíos, o sea, sus Índices no tiene que ser consecutivos.
- Deben ser Inicializadas antes de ser usadas. Una variable de Tabla Anidada no inicializada es una colección nula.
- Para ser inicializadas es necesario hacer uso de su constructor. Dicho Constructor es una función con el mismo nombre que el **Tipo Colección**, el cual devuelve una colección de ese tipo.
- A diferencia de las Matrices Asociativas, las Tablas Anidadas no pueden contener índices negativos.
- Es importante que tengas en cuenta que, aunque se hace referencia a la primera columna como **INDICE**, las Tablas Anidadas no tienen índices, más bien es una columna con números.

Sintaxis:

```
TYPE nuevo_tipo_dato IS TABLE OF
   {
      column_type
      | variable%TYPE
      | table.column%TYPE} [NOT NULL]
      | table%ROWTYPE
   }
```

Script 7

Ejemplo simple de una tabla anidada de tipo VARCHAR2.

```
DECLARE
   TYPE tabla_varchar2 IS TABLE OF VARCHAR2(100);
   empleados   tabla_varchar2 := tabla_varchar2();
BEGIN
   -- Tamaño Inicial
   DBMS_OUTPUT.PUT_LINE('Tamaño Inicial: ' || empleados.COUNT);
   -- Se añaden 4 elementos
   empleados.EXTEND (4);
   empleados (1) := 'Pepe';
   empleados (2) := 'Elena';
   empleados (3) := 'Carmen';
   empleados (4) := 'Juan';
   -- Se añade un elemento mas
   empleados.EXTEND;
   empleados (empleados.LAST) := 'Gustavo';
   -- Tamaño Final
   DBMS_OUTPUT.PUT_LINE('Tamaño Final: ' || empleados.COUNT);
   -- Mostrar lista
   FOR I IN 1 .. empleados.COUNT
   LOOP
      DBMS_OUTPUT.put_line ( empleados(I) );
   END LOOP;
END;
```

Ejecución:

```
Tamaño Inicial: 0
Tamaño Final: 5
Pepe
Elena
Carmen
Juan
Gustavo
```

Script 8

Segundo ejemplo de tablas anidadas.

```
DECLARE
   -- Definimos los tipo de datos
   TYPE TABLA_EMPLEADOS IS TABLE OF HR.EMPLOYEES%ROWTYPE;
   -- Definiendo las variables
   V_EMPLEADOS TABLA_EMPLEADOS;
BEGIN
   V_EMPLEADOS := TABLA_EMPLEADOS();
   DBMS_OUTPUT.PUT_LINE('TAMAÑO INICIAL: ' || V_EMPLEADOS.COUNT);
   FOR REC IN (SELECT * FROM HR.EMPLOYEES) LOOP
      V_EMPLEADOS.EXTEND;
      V_EMPLEADOS(V_EMPLEADOS.LAST) := REC;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('TAMAÑO FINAL: ' || V_EMPLEADOS.COUNT);
   FOR I IN V_EMPLEADOS.FIRST..V_EMPLEADOS.LAST LOOP
      DBMS_OUTPUT.PUT_LINE( I || '.- ' || V_EMPLEADOS(I).FIRST_NAME);
   END LOOP;
END;
```

Ejecución:

```
TAMAÑO INICIAL: 0
TAMAÑO FINAL: 107
1.- Steven
2.- Neena
. . .
106.- Shelley
107.- William
```

ORACLE PL/SQL

CURSOS VIRTUALES

En estos enlaces se publican cupones de descuento:

- https://github.com/gcoronelc/UDEMY
- https://www.facebook.com/groups/bolsa.sistemas
- https://www.facebook.com/groups/universidadjava
- https://www.facebook.com/groups/desarrollasoftware
- https://chat.whatsapp.com/H6BBebNDZHEAqAk6gTU5ZS

JAVA ORIENTADO A OBJETOS

https://bit.ly/2B3ixUW

PROGRAMACIÓN DE BASE DE DATOS CON JAVA JDBC

https://bit.ly/31apy0O