COLECCIONES

COLECCIONES

- ***Oracle*** provee dos ***Tipos de Datos Compuestos***
  - ***Records***
  - ***Las Colecciones*** (Ambos ***de PL/SQL***).
- En Oracle una "colección" PL/SQL es un tipo de datos compuesto consistente en una matriz de una sola dimensión que está compuesta por uno o más elementos accesibles a través de un índice.

COLECCIONES

- Una colección es un grupo ordenado de elementos, todos del mismo tipo
- Funcionan como arrays
- Pueden almacenar instancias de un tipo objeto y ser atributos de un tipo objeto.
- Las tablas indexadas de PL/SQL son también colecciones

COLECCIONES

- Se tienen los siguientes tipos de colecciones
  - TABLAS INDEXADAS (Index-by Tables)
  - TABLAS ANIDADAS (Nested Tables)
  - TABLAS DE LONGITUD VARIABLE (Variable-size arrays or VARRAYS)

I - TABLAS INDEXADAS (Index-by Tables)

- Llamadas también 'tablas asociativas' porque cada valor de la tabla está asociado a una clave única utilizada para la localización de dicho valor.
  - La clave puede ser un entero del tipo binary_integer
  - Un string

1- TABLAS INDEXADAS

- 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 objetos o 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 (*index*) es usado para localizar el dato almacenado en la segunda columna.

1- TABLAS INDEXADAS

- ✓ Los valores del índice pueden ser números, positivos o negativos y de tipo caracteres y no necesariamente tienen que ser insertados de forma consecutiva, se 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.

Nota: Los valores de índice para tablas asociativas deben ser únicos. Si se asigna un nuevo valor(columna value) a un índice previamente usado, el valor anterior es reemplazado. Esto es:

- *v_assoc_array(1) = 'Hola';*
- *v_assoc_array(1) = 'PL/SQL';*

Métodos de Colecciones

- Los siguientes métodos hacen a las tablas mas fáciles de usar

  -EXISTS

  -DELETE

  -FIRST AND LAST

  -PRIOR

  -NEXT

  -COUNT

  -EXTEND

TABLAS INDEXADAS

Ejemplo de tabla anidada usando un binary_integer como clave:

```
SET SERVEROUTPUT ON
DECLARE
--Definición del Tipo Tabla:
TYPE typ_assoc_array IS TABLE OF   VARCHAR2(25) INDEX BY BINARY_INTEGER
 --Declaramos una Variable del Tipo Tabla anterior:
    v_assoc_array   typ_assoc_array;
BEGIN
            v_assoc_array(1) :=  'Valor para el Indice 1';
           v_assoc_array(2)  :=  'Valor para el Indice 2';
           v_assoc_array(3)  :=  'Valor para el Indice 3';
        DBMS_OUTPUT.PUT_LINE(v_assoc_array(1)||', '||v_assoc_array(2)||',
'||v_assoc_array(3));
END;
```

TABLAS INDEXADAS

Ejemplo de tabla anidada usando un string como clave:

```
DECLARE
 TYPE t_poblacion IS TABLE OF NUMBER INDEX BY
 VARCHAR2(64);
 poblacion_pais t_poblacion;
 cantidad NUMBER;
BEGIN
 poblacion_pais('China')  := 1321000000;
 poblacion_pais('EEUU')   := 302750000;
 poblacion_pais('Brasil') := 186990000;
 DBMS_OUTPUT.PUT_LINE('El primer índice es ' ||
poblacion_pais.FIRST);
 DBMS_OUTPUT.PUT_LINE('El primer valor  es ' ||
poblacion_pais(poblacion_pais.FIRST));
END;
/
```

```
declare
  type t_vector is table of number(3) index by
varchar2(10);
  v_vector t_vector;
  subind varchar2(10);
begin
  v_vector('Cuarenta') := 40;
  v_vector('cuarenta') := -40;
  v_vector('Cinco') := 5;
  v_vector('Quince') := 15;
  dbms_output.put_line('count: '||
v_vector.count);

  subind := v_vector.first;
  while subind is not NULL loop
   dbms_output.put_line(subind |':'||
      v_vector(subind));
    subind := v_vector.next(subind);
  end loop;
end;/
```

2 -NESTED TABLES (Tablas Anidadas)

- La funcionalidad de una tabla anidada es la misma que de una tabla indexada, pero pueden almacenarse en la base de datos
- Dos características importantes:
  - Como las tablas indexadas, no requieren la definición de un tamaño y pueden crecer dinámicamente.
  - El posicionamiento inicia de 1 y debe asignarse secuencialmente. Pero puede borrarse un elemento y su posición puede quedar vacía.

2 -NESTED TABLES (Tablas Anidadas)

- Pueden ser declaradas en bloques de ***PL/SQL*** así como también en la Base de Datos.
- 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 **Tablas *Asociativas***, las ***Tablas Anidadas*** no pueden contener índices negativos. Otro dato importante es que aunque se hace referencia a la primera columna como 'Indice', las ***Tablas Anidadas*** no tienen índices, mas bien es una columna con números.

Ejemplo BLOQUE PL/SQL

```
DECLARE    -- --Definición del Tipo Tabla:
   TYPE  typ_nest_tab IS TABLE OF  VARCHAR2(25);
      --Declaramos una Variable del Tipo Tabla: typ_nest_tab
     v_nest_tab     typ_nest_tab;
BEGIN
      --A continuación Inicializamos la variable tipo tabla
     v_nest_tab  :=  typ_nest_tab();

     --EXTEND: Inserta un Registro Nulo a la Tabla.
    v_nest_tab.EXTEND;
   v_nest_tab(1)  :=  'Valor para el Indice 1';
   v_nest_tab.EXTEND;
   v_nest_tab(2)  :=  'Valor para el Indice 2';
   v_nest_tab.EXTEND;
   v_nest_tab(3)  :=  'Valor para el Indice 3';
    -- despliega los valores en pantalla
   DBMS_OUTPUT.PUT_LINE(v_nest_tab(1)||', '||v_nest_tab(2)||',
'||v_nest_tab(3));
END;
```

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

Una tabla anidada es un conjunto de elementos del mismo tipo sin ningún orden predefinido.

Estas tablas solamente pueden tener una columna que puede ser de un tipo de datos básico de Oracle, o de un tipo de objetos definido por el usuario.
En este último caso, la tabla anidada también puede ser considerada como una tabla con tantas columnas como atributos tenga el tipo de objetos.

En el siguiente ejemplo, se declara una tabla que después es anidada en el tipo ordenes_t.
Los pasos de todo el diseño son los siguientes.

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

Los pasos de todo el diseño son los siguientes.

1- Se define el tipo de objetos linea_t para las filas de la tabla anidada

```
CREATE TYPE linea_t  AS OBJECT
( linum NUMBER, item VARCHAR2(30),
 cantidad NUMBER, descuento NUMBER(6,2) ) ;
```

2- Se define el tipo de colección tabla lineas_pedido_t para después anidarla.

```
CREATE TYPE lineas_pedido_t  AS TABLE OF linea_t ;
```

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

Esta definición permite utilizar el tipo colección lineas_pedido_t para definir:

- El tipo de datos de una columna de una tabla relacional.
- El tipo de datos de un atributo de un tipo de objetos.
- Una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

3- Se define el tipo de objetos ordenes_t que en el atributo pedido almacena una tabla anidada del tipo lineas_pedido_t.

```
CREATE TYPE ordenes_t  AS OBJECT
 ( ordnum NUMBER,
  fechpedido DATE, fechentrega DATE,
  pedido lineas_pedido_t
  ) ;
```

4- Se define la tabla de objetos ordenes_tab y se especifica la tabla anidada del tipo lineas_pedido_t.

```
CREATE TABLE ordenes_tab OF ordenes_t
NESTED TABLE pedido STORE AS pedidos_tab
;
```

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

Este último paso es necesario hacerlo porque la declaración de una tabla anidada no reserva ningún espacio para su almacenamiento.

Lo que se hace es indicar en qué tabla (pedidos_tab) se deben almacenar todas las lineas de pedido que se representen en el atributo pedido de cualquier objeto de la tabla ordenes_tab.

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

Al definir una tabla tipada que contiene un atributo que es de tipo tabla, es necesario darle un nombre de almacenamiento a la NESTED TABLE mediante la cláusula STORE AS que es un nombre interno y no se puede utilizar para acceder directamente a la tabla anidada.

La tabla anidada es un tipo colección y funciona como tal. Por tanto, el acceso a sus miembros (bien para inserción, borrado o consulta)
es necesario hacerlo a través de la tabla principal.

Creación de una tabla anidada en PL/SQL

```
DECLARE
    TYPE t_personas IS TABLE OF tip_cli;
     v_personas t_personas := t_personas();
```

La tabla debe ser inicializada como los objetos.

Si trata de asignar a un elemento de una tabla que no ha sido inicializada, dará un error por ser aún nula.

TABLAS ANIDADAS (NESTED TABLES) EN LA BD

```
CREATE TYPE tipo_producto AS OBJECT
     (id_producto NUMBER(8),
     precio NUMBER(9));
CREATE TYPE tabla_producto AS TABLE OF
tipo_producto;


CREATE TABLE ventas
(No_venta NUMBER(8),
fecha  DATE,
producto tabla_producto)
 NESTED TABLE producto STORE AS
      producto_vendidos;
```

Incorporando elementos en la tabla ventas:

Note que la columna “producto” es del tipo tabla_producto, el cual a su vez es una colección de objetos tipo_producto. Por lo tanto, para asignar a la columna producto, debe emplear el constructor, e instanciar cada elemento:

```
INSERT INTO VENTAS
  (no_venta, fecha, producto)
   VALUES  (1, SYSDATE,
tabla_producto(tipo_producto(1,30000),
                  tipo_producto(2,2500),
                  tipo_producto(3,10000));
```

Es decir, en la columna producto, asignamos 3 elementos de tipo_producto.

Insertar registros en la tabla ventas con un LOOP

```
 DECLARE
  CURSOR C_VENTAS IS SELECT * FROM B_VENTAS;
  CURSOR C_ART(P_VENTA NUMBER) IS
          SELECT DISTINCT V.ID_ARTICULO, A.PRECIO
        FROM   B_DETALLE_VENTAS V JOIN B_ARTICULOS A
        ON     A.ID = V.ID_ARTICULO
        WHERE V.ID_VENTA = P_VENTA;
  V_PROD tabla_producto := tabla_producto(); -- CONSTRUCTOR
  ind number;
BEGIN
  FOR REG IN C_VENTAS LOOP
      ind := 1;
      V_PROD.DELETE; --LIMPIA LA TABLA (COLECCIÓN)
      FOR REG2 IN C_ART(REG.ID) LOOP
          v_PROD.EXTEND; -- INICIALIZA CADA ELEMENTO
          V_PROD(ind) := tipo_producto(REG2.ID_ARTICULO,
REG2.PRECIO);
          ind := ind + 1;
      END LOOP;
      INSERT INTO VENTAS VALUES (REG.ID, REG.FECHA, V_PROD);
   END LOOP;
END;
/
```

Leer elementos insertados en ventas:

```
SELECT v.no_venta, v.fecha, c.id_producto, c.precio
FROM ventas v, TABLE(v.producto) c;

 NO_VENTA FECHA    ID_PRODUCTO     PRECIO
---------- -------- ----------- ----------
         1 27/10/15           1      30000
         1 27/10/15           2       2500
         1 27/10/15           3      10000
```

Con la cláusula TABLE() se obtiene los elementos de la colección almacendada en la columna producto.

3- Varying Arrays -VARRAYS

- Son colecciones de elementos homogéneos
- Los varrays deben tener un límite y se almacenan en lugares de memoria consecutivos. El límite se especifica en la definición del tipo.
- Sus elementos se acceden con subíndice empezando de 1

varray

Un tipo VARRAY se puede utilizar para definir:

- El tipo de datos de una columna de una tabla de BD
- El tipo de datos de un atributo de un tipo de objetos.
- Una variable PL/SQL, un parámetro, o el tipo que devuelve una función.

Cuando se declara un tipo VARRAY no se produce ninguna reserva de espacio. Si el espacio que requiere lo permite, se almacena junto con el resto de columnas de su tabla, pero si es demasiado largo (más de 4000 bytes) se almacena aparte de la tabla como un BLOB.

VARRAYs

- Desde una sentencia SELECT no es posible acceder a los elementos individuales del VARRAY indexándolos;
- Para el tratamiento individual de cada elemento del VARRAY, es necesario hacerlo mediante PL/SQL, que proporciona cláusulas para posicionarse en los distintos elementos del array.
- Es posible definir un tipo de datos como una tabla, y utilizar dicho tipo como el tipo de datos de la columna de otra tabla.
- De este modo, la columna contendrá una colección de valores, objetos o referencias, que se almacenarán en formato de tabla.

VARRAYs

```
TYPE nombre_tipo IS { VARRAY |
   VARYING ARRAY } (tamaño_maximo)
   OF tipo_elemento [NOT NULL];
```

(*Los tipos no pueden ser BOOLEAN, REF CURSOR, TABLE u otro tipo VARRAY*)

Para alterar el tamaño máximo del varray:

```
ALTER TYPE nombre_tipo MODIFY LIMIT
   nuevo_tamaño
   [INVALIDATE|CASCADE];
```

Ejemplo de VARRAYs

```
CREATE TYPE var_tab AS VARRAY (10) OF
NUMBER;
```

- Utilización del tipo varray:

```
DECLARE
   a var_tab;
BEGIN
    a := var_tab( 1,2,3,5);
    DBMS_OUTPUT.PUT_LINE(a(1));
END;
```

Agregar elementos a un varray

```
DECLARE
TYPE tipo_var  IS VARRAY(5) OF
   VARCHAR2(10);
 v_var tipo_var := tipo_var('Uno',
   'Dos', 'Tres', 'Cuatro');
BEGIN
    v_var.EXTEND;
     v_var(5) := 'Cinco';
END;
```

varray

```
CREATE TYPE lista_tel_t AS VARRAY(10) OF VARCHAR2(20) ;

CREATE TYPE clientes_t AS OBJECT
  ( clinum NUMBER,
   clinomb VARCHAR2(200),
    lista_tel lista_tel_t );
```

La principal limitación que presenta los tipos VARRAY es que en las consultas es imposible poner condiciones sobre los elementos almacenados dentro. Desde una consulta SQL, los valores de un VARRAY solamente pueden ser accedidos y recuperados en un bloque. Es decir no se puede acceder a los elementos de un VARRAY individualmente. Sin embargo desde un programa PL/SQL si que es posible definir un bucle que itere sobre los elementos de un VARRAY

Diferencias entre VARRAY y NESTED TABLE:

- Los VARRAYS tienen un tamaño máximo fijo que se especifica en la definición del tipo, mientras que las NESTED TABLES son de tamaño variable (no se dimensionan).
- A la hora del almacenamiento en la base de datos, los VARRAYS se guardan en el mismo espacio que la tabla. Las NESTED TABLES se almacenan como otra tabla independiente asociada a la tabla sobre la que está definida, pero sobre la que solo se pueden realizar consultas a través de la tabla en la que está definida.
- Cuando se necesite emplear un tipo colección, a la hora de elegir entre una NESTED TABLE o un VARRAY hay varios criterios a seguir:
  - Si el orden en que se almacenan los elementos de la colección es relevante, se emplea un VARRAY, puesto que la NESTED TABLE no conservan el orden.

varray

- En caso de saber de antemano el número de elementos que tendrá la colección, se emplea un VARRAY porque permite limitar su tamaño en el momento de su definición.
- En general, si el tamaño y el orden no son especialmente relevantes: si se necesitan consultas sobre la colección, que nos permitan tratar los elementos de la colección por separado, se emplea una NESTED TABLE, mientras que si se desea recuperar la colección como un todo se empleará un VARRAY, aunque también se permitiría el acceso a los elementos individuales.

Recorrer colecciones PL/SQL

- Una operación que se debe realizar muy comúnmente con las colecciones PLSQL es tener que **recorrerse todos los elementos de la misma**.
- Las razones para tener que recorrerse totalmente los datos de una colección pueden se múltiples, desde mostrar la información que almacenan hasta buscar un elemento específico o ejecutar una sentencia DML (de manipulación de datos dentro de la base de datos Oracle) utilizando los datos almacenados en los diferentes elementos de la colección.

Recorrer colecciones PL/SQL

- El tipo de código PL/SQL que se debe utilizar para **operar y recorrer una colección** viene determinado por el tipo de colección con la que estemos trabajando y la forma en que dicha colección fue inicializada con datos. Lo normal es elegir entre utilizar un **bucle WHILE** o utilizar un **bucle numérico FOR**.

Excepción NO_DATA_FOUND en colecciones

Uno de los primeros conceptos básicos a tener en cuenta cuando se trabaja con colecciones es que La base de datos Oracle generará un error **NO_DATA_FOUND** cuando intentemos leer un elemento de una colección cuyo valor de índice no haya sido definido todavía.

Por ejemplo, el siguiente código PLSQL generará el error:

```
DECLARE  TYPE  coleccion_numeros  IS TABLE
OF NUMBER   INDEX BY PLS_INTEGER;
    v_col   coleccion_numeros;
    BEGIN DBMS_OUTPUT.PUT_LINE (v_col(5));
END;
```

Cuándo utilizar un bucle numérico FOR

Los bucles numéricos FOR se deben utilizar **cuando la colección ha sido poblada en su totalidad**, es decir, cuando a todos los elementos desde el índice inicial al final se les ha asignado de forma secuencial un valor. El bucle numérico FOR también está pensado para los casos en los que inexcusablemente debemos recorrernos todos los elementos de la colección.

Cuando se sabe que la colección está completamente rellena y que no debería generarse la excepción NO_DATA_FOUND por ningún motivo, **utilizar el bucle FOR** es la forma más sencilla de recorrerse dicha colección.

Cuándo utilizar un bucle WHILE

Los bucles WHILE son adecuados cuando los datos de la colección PLSQL son dispersos o cuando podríamos necesitar **parar** la ejecución del bucle antes de haber recorrido todos los elementos de la colección. Ej

```
DECLARE
    TYPE t_tab IS TABLE OF VARCHAR2(30)
    INDEX BY PLS_INTEGER;
    v_elementos t_tab;
    v_ind  PLS_INTEGER;
BEGIN
     v_elementos(1):= 'PRIMER';
     v_elementos(10):= 'DECIMO';
     v_ind := v_elementos.FIRST;
     WHILE v_ind <= v_elementos.LAST LOOP
        DBMS_OUTPUT.put_line (v_elementos(v_ind));
           v_ind:= v_elementos.NEXT (v_ind);
    END LOOP;
 END;
```

EJERCICIOS