Índice

1. REGISTROS (RECORD)

La sintaxis general para definir e inicializar un registro es:

```
TYPE   tipo_registro    IS  RECORD (
                Campo1   Tipo1   [NOT   NULL]  [:= expr1],
                Campo2   Tipo2   [NOT   NULL]  [:= expr2],
                Campo3   Tipo3   [NOT   NULL]  [:= expr3],
                  ……        …         .. ……          ……
                Campo(n)   Tipo(n)   [NOT   NULL]  [:= expr(n)]  );
```

Es típico en el trabajo con bases de datos el declarar registros con el mismo formato que las filas de las tablas. Para ello o bien sabes el formato de las tablas y creas un registro con ese formato, o bien utilizas el operador **%rowtype**, similar a **%type.**

Emp.**ename%TYPE** /* equivale a una columna */

Emp%ROWTYPE /* define un registro o elemento como una fila de la tabla empleados. */

Ejemplo 1.-

```
DECLARE
TYPE  StockItem  IS RECORD (
        Item_no       INTEGER(3),
        Description   VARCHAR2(50),
        Quantity      INTEGER,
        Price         REAL(7,2));

item_info StockItem;  -- declaramos la variable item_info del tipo StockItem

BEGIN
                ….
END;
```

Ejemplo 2.-

```
DECLARE
TYPE  EmpRec   IS RECORD (
        Emp_id        emp.empno%TYPE,      --  del mismo tipo que ese campo
        Last_name     VARCHAR2(10),
        Job_title     VARCHAR2(9),
        SALARY        NUMBER(7,2));
…..
PROCEDURE raise_salary (emp_info EmpRec); -- usamos como argumento la variable
emp_info del tipo EmpRec

BEGIN
                ….
END;
```

Ejemplo 3.-

```
DECLARE
TYPE  TimeRec   IS RECORD (
        Secs          SMALLINT     :=0,
        Mins          SMALLINT     :=0,
        Hrs           SMALLINT     :=0);
…..
```

Como referenciar un registro:

Nombre_registro.nombre_campo
emp_info.Emp_id

Cuando se llame a una función que devuelva un registro, hay que referenciar al campo dentro del registro.

Nombre_funcion(lista_parametros)**.nombre_campo**

```
DECLARE
TYPE  EmpREc   IS RECORD (
        Emp_id        NUMBER(4),
        Job_title     VARCHAR2(9),
        Salary        number(7,2));
        Middle_sal    number(7,2);
FUNCTION  nth_highest_sal (n INTEGER) RETURN  EmpRec IS
        Emp_info   EmpREc;
BEGIN
…..
RETURN Emp_info;      --devuelve un registro
```

Ejemplo. Declarar un registro que contiene los campos nombre y cliente. Donde en nombre se va a guardar el nombre de un cliente que se introduce por teclado y cliente va a ser el contenido de todas las columnas de la tabla customer de dicho cliente. Visualizar el identificativo del cliente introducido.

```
declare
   type tipo_registro is record
     (nombre varchar2(20),
      cliente customer%rowtype);
    registro tipo_registro;
begin
    registro.nombre:='&nombre';
    select * into registro.cliente
    from customer
    where name=registro.nombre;
    dbms_output.put_line(registro.cliente.customer_id);
end;
/
```

2. COLECCIÓN (TABLE/VARRAY)

Se define colección a un conjunto de elementos ordenados y PL/SQL define 3 tipos:

2.1. **Arrays asociativos,** son tablas exclusivas de PL/SQL, 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. Aunque el rango real permitido de índice va desde -2**31 ... 2**31.
Es opcional fijar el límite.
Están compuestas de dos columnas a las cuales no es posible asignarles nombres:
- La primera columna es el índice (index). X(1) X(2) X(3) …
- La segunda columna contiene el dato almacenado (value).
- El índice (index) 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 secuenciar 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.

321	17	407	83	622	105	19	67	278	Fija Límite superior
X(1)	X(2)	X(3)	X(4)	X(5)	X(6)	X(7)	X(8)	X(9)	

2.2. **Varrays,** VARRAY significa Matriz de Tamaño Variable(Variable-Size Array). Son matrices que pueden ser almacenadas y recuperadas a través de SQL.
El rango del índice va desde 1 hasta el *tamaño_límite* especificado en la declaración.
Los VARRAYs son colecciones que guardan más similitud con las Tablas Anidadas que con las Matrices Asociativas. Características:
- Al igual que las Tablas Anidadas pueden ser declaradas en bloques de PL/SQL así como también en la Base de Datos.
- A diferencia de las Matrices Asociativas y las Tablas Anidadas los VARRAYs no tienen un tamaño dinámico, a estos últimos hay que especificarles el límite máximo al momento de su declaración.
- Deben ser inicializadas antes de ser usadas (al igual que las Tablas Anidadas). Una variable Tipo VARRAY 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.
- Los índices siempre son secuenciales/consecutivos, esto significa que no es posible eliminar elementos del centro de un VARRAY, solo es posible eliminarlos del final de la Colección (con el método TRIM). Los índices no pueden ser negativos siendo 1 el límite inferior.

B	C	A	A	A	C	D		
(1)	(2)	(3)	(4)	(5)	(6)	(7)		

Tamaño Máximo = 9

2.3 **Tablas anidadas**, una Tabla Anidada (Nested Table) puede ser considerada como una tabla de una sola columna que puede ser alojada en memoria, aunque es prudente decir que también puede ser una columna en una tabla de Base de Datos.
Sin límites y huecos en los borrados. El rango de índice permitido va desde 1 ... 2**31
- Pueden ser declaradas en bloques de PL/SQL así como también en la Base de Datos.
- Al igual que las Matrices Asociativas (Associative Array), Las Tablas Anidadas tienen un tamaño dinámico y puede contener elementos vacíos, o sea, sus índices no tienen 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. Otro dato importante es que, aunque se hace referencia a la primera columna como 'índice', las Tablas Anidadas no tienen índices, más bien es una columna con números.

321		407	83		105	19		278
X(1)		X(3)	X(4)		X(6)	X(7)		X(9)

Sin limites →

2.1. Arrays asociativos

```
TYPE tipotabla IS TABLE OF tipoelemento INDEX BY BINARY_INTEGER
                                                PLS_INTEGER          ;
                                                VARCHAR2 [size_limit]

 Nombre_tabla tipotabla;
```

Donde:

tipotabla.- es el nombre del nuevo tipo que está siendo definido

tipoelemento.- es un tipo escalar predefinido o una referencia a un tipo escalar Mediante %TYPE

Nombre_tabla.- es el nombre que le damos realmente declarando la variable

Una vez declarados el tipo y la variable, podemos hacer referencia a un elemento determinado de la tabla PL/SQL mediante la sintaxis:

```
Nombre_tabla(índice)
```

donde Nombre_tabla es el nombre de la colección, e índice es una variable de tipo BINARY_INTEGER

Nombre_tabla*(índice).columna*

Ejemplo 1.-

```
DECLARE
TYPE associative_array_type  IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
V3     associative_array_type;
```

Ejemplo_2.-

```
DECLARE
 TYPE       EmpTabTyp   IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
 EMP_TAB  EmpTabTyp;
BEGIN
  SELECT * INTO  EMP_TAB(7369) FROM employee where employee_id = 7369;
END
```

ATRIBUTOS DE LA TABLA

Atributo	Tipo devuelto	Descripción
COUNT	NUMBER	Devuelve el número de filas de la tabla
DELETE	N/A	Borra filas de la tabla
EXISTS	BOOLEAN	Devuelve TRUE si existe en la tabla el elemento especificado
FIRST	BINARY_INTEGER	Devuelve el índice de la primera fila de la tabla
LAST	BINARY_INTEGER	Devuelve el índice de la primera fila de la tabla
NEXT	BINARY_INTEGER	Devuelve el índice de la fila de la tabla que sigue a la fila especificada
PRIOR	BINARY_INTEGER	Devuelve el índice de la fila de la tabla que antecede a la fila especificada

Hay que tener en cuenta lo siguiente:

- **DELETE** constituye una orden completa por sí mismo; no se lo utiliza como parte de una expresión como sucede con los otros atributos
- **EXISTS** devolverá TRUE si existe el elemento buscado en caso contrario devolverá FALSE. Este atributo es útil para evitar el error ORA-1403 que se produce cuando el elemento no existe
- Tanto FIRST como LAST devolverán el índice, **no el valor contenido en dichas filas**
- Excepto por el atributo **DELETE** no hay manera de borrar todas las filas de una tabla

Ejemplos

Ntabla.count devuelve el número de filas que contiene la tabla

Ntabla.delete borra todos los elementos de la tabla

Ntabla.delete(n) borra el elemento n de la tabla. Si este elemento es null no hará nada.

Ntabla.delete(n1,n2) borra los elementos de n1 a n2 de una tabla siempre que n1 >n2 sino no hará nada.

Ntabla.exits(n) devuelve true si existe el elemento n. en caso contrario devuelve false.

Ntabla.first devuelve el índice del primer elemento de la tabla

Ntabla.last devuelve el índice del último elemento de la tabla

Ntabla.next(n) devuelve el índice del elemento posterior al elemento n

Ntabla.prior(n) devuelve el índice del elemento anterior al elemento n

2.2. Varrays

```
TYPE tipotabla IS | VARRAY         | (size_limit) OF tipo_elemento [not null];
                  | VARYING ARRAY |

Nombre_tabla tipotabla;
```

- **Son equivalentes a los arrays de una dimensión de los lenguajes de programación tradicionales.**
- **Tienen un índice secuencial que permite el acceso a sus elementos. A diferencia de otros lenguajes de programación el índice comienza en uno.**
- **Tienen una longitud fija determinada en el momento de su creación.**
- **Los elementos del array han de ser inicializados antes de ser referenciados. Evidentemente hemos de inicializar el array completo.**

Ejemplo 1.- definición

```
DECLARE
    TYPE     CalendarList  IS VARRAY(366) OF DATE;
   Calendar  CalendarList;
```

Ejemplo 2.-

```
DECLARE
        TYPE      varray_type   IS VARRAY(50) OF  INTEGER;
        V2        varray_type;
```

Ejemplo 3.-

```
DECLARE
        TYPE      DeptFile   IS VARRAY(20) OF  c1%ROWTYPE;   -- basado en cursor
        C1        DeptFile;
```

Ejemplo 4.- definición e inicialización

```
  DECLARE
        TYPE                        ProjectList  IS VARRAY(3) OF  VARCHAR2(16);
        Accounting_projects         ProjectList;
  BEGIN
        Accounting_projects :=
          ProjectList('Expense Report', 'Outsourcing', ' Auditing');
END;
```

Ejemplo 5. - cargar la tabla con valores de una select de tabla oracle

```
  DECLARE
        TYPE            ProjectList  IS VARRAY(50) OF  VARCHAR2(16);
        My_projects  ProjectList;
  BEGIN
        SELECT projects INTO  My_projects FROM department where dept_id =30;
END;
```

Ejemplo 6.- Utilización de Registro (RECORD) y VARRAY

```
DECLARE
TYPE   AnEntry  IS RECORD (
            Term   VARCHAR(20),
            Meaning  VARCHAR(200));

TYPE Glossary   IS VARRAY(250) OF  AnEntry;
```

Ejemplo. realizar un bloque pl/sql que permita introducir un número y calcule la tabla de multiplicar de dicho número del 1 al 9.

```
declare
   type ejem is varray(9) of number;
   tabla ejem;
   i integer;
   num integer;
begin
   tabla:=ejem(0,0,0,0,0,0,0,0,0);
   num:=&num;
   for i in 1..9 loop
       tabla(i):=num*i;
   end loop;
   for i in 1..9loop
      dbms_output.put_line(num||' x '||i||' = '||tabla(i));
   end loop;
end;
/
```

2.3. Tablas anidadas

TYPE *tipotabla* **IS TABLE OF** *tipoelemento* [not null] ;

Nombre_tabla *tipotabla*;

Estructuras similares a los VARRAYS.
Tienen en comun: practicamente todo. La gran diferencia es que **no tienen una longitud fija**

Los elementos no pueden ser BYNARY_INTEGER, PLS_INTEGER, BOOLEAN, LONG, CURSOR, STRING.

Ejemplo 1.- definición

```
DECLARE
 TYPE nested_type IS TABLE OF VARCHAR2(20);
 V1    nested_type;
```

Ejemplo 2.- definición e inicialización

```
DECLARE
  TYPE          CourseList IS TABLE OF VARCHAR2(16);
  My_courses    CourseList;
   BEGIN
  My_courses :=
              CourseList('Econ 2010', 'Acct 3401', 'Mgmt 3100');
  END;
```

Ejemplo 3.-

```
DECLARE
  TYPE          CourseList IS TABLE OF VARCHAR2(16);
  My_courses    CourseList;
   BEGIN
  My_courses :=
              CourseList('Econ 2010', 'Acct 3401', 'Mgmt 3100');
My_courses.extend;
My_courses(4):=' Econ 2011';
 END;
```

Ejemplo 4.- como referenciar a un elemento

```
DECLARE
   TYPE              CourseList  IS TABLE  OF  VARCHAR2(16);
    My_courses       CourseList;
    BEGIN
      My_courses := CourseList('Econ 2010', 'Acct 3401', 'Mgmt 3100');
      My_courses.extend;
      My_courses(4):=' Econ 2011';
      For i in My_courses.first... My_courses.last loop
          If My_courses(i)= 'Econ 2010'
             Then
               My_courses(i):= null;
          End if;
       End loop;
  END;
```

Ejemplo 5. – Cargar la tabla anidada con los valores de una select de una tabla oracle

```
DECLARE
  TYPE              CourseList IS TABLE OF VARCHAR2(16);
  English_courses   CourseList;
   BEGIN
    SELECT courses INTO English_courses from department where name='English';
  END;
```

Se pueden crear objetos que contengan estas listas.

```
CREATEY TYPE Student IS OBJET (id_num INTEGER(4),
                               Name     VARCHAR2(25),
                               Address  VARCHAR2(35),
                               Status   CHAR(2),
                               courses  CourseList)
```

Ejemplo: Realizar un bloque pl/sql que permita introducir un número y calcule la tabla de multiplicar de dicho número del 1 al 9.

```
declare
    type tipotabla is table of number(4);
    tabla tipotabla;
    i integer;
    num integer;
begin
    tabla:=tipotabla(0,0,0,0,0,0,0,0,0);
    tabla.extend;
    tabla(9):=0;
    num:=&num;
    for i in 1..9 loop
        tabla(i):=num*i;
    end loop;
    for i in 1..9 loop
        dbms_output.put_line(num||' x '||i||' = '||tabla(i));
    end loop;
end;
/
```

3.- EJERCICIOS PROPUESTOS

1º. Realizar un bloque pl/sql que permita calcular la serie de Fibonacci. Esta serie se va obteniendo de la siguiente forma: introducido un número por teclado me ha de dar el valor 1 si introducimos el 1 el 2 si introducimos el 2 el valor 3 si introducimos el 3, el valor 5 si introducimos 4, el valor 8 si introducimos 5 y así sucesivamente.
(Explicación serie de Fibonacci: sucesión infinita de números naturales que comienza con los números 1 y 1, y a partir de ellos, cada término se obtiene sumando los dos anteriores: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1.597 …)

2º. realizar un bloque pl/sql que visualice cuales de los 50 primeros números naturales son primos y cuáles no. Ha de verse algo como lo siguiente:

El 1 es primo	El 2 es primo	El 3 es primo	El 4 no es primo	El 5 es primo	El 6 no es primo	El 7 es primo	…….	……	El 50 no es primo

3º. Realizar un bloque pl/sql que permita introducir una frase por teclado y que cada palabra se almacene en una celda de una tabla.

4º. Realizar un bloque pl/sql para declarar un registro que permita insertar un departamento en la tabla **DEPARTMENT** y posteriormente inserte los valores introducidos por teclado en el registro en dicha tabla.

5º. Realizar un bloque pl/sql que permita realizar lo siguiente: Dado el nombre de un cliente que llega hoy a la empresa, Guardar en la tabla **CUSTOMER** su nuevo código, su fecha de hoy y en el campo comments 'es un cliente llegado hoy' y la fecha. Además, asignarle el representante de ventas que más clientes tenga.

6º. Realizar un bloque pl/sql que permita realizar lo siguiente: Hay un cliente que realiza un pedido nuevo de un solo producto solicitar al usuario: nombre del cliente, nombre del producto y unidades compradas y reflejar todos estos datos en la base de datos grabando en **SALES_ORDER** e **ITEM**.

7º. Realizar un bloque pl/sql que permita realizar lo siguiente: Se introduce un cliente por teclado si se ha gastado más de 1000 euros en sus compras, ponerle como límite de crédito el 10% de su dinero gastado, en caso contrario el crédito limite que le concedemos es el del 15% de lo gastado.

8º. Realizar un bloque pl/sql que permita realizar lo siguiente: Dado el nombre de un empleado que escribe el usuario a través del teclado si tienen comisión ponerle como comisión el 20% de su salario y si no tiene comisión calculamos el 20% de las compras del total de pedidos de sus clientes.