**📌 Explicación de TABLE en PL/SQL**

El tipo **TABLE** en PL/SQL es una colección que funciona como un **array dinámico**. Esto significa que: ✅ Puede crecer según sea necesario.  
✅ Se accede a sus elementos usando un **índice numérico**.  
✅ Permite almacenar **datos del mismo tipo**, que pueden ser valores simples o incluso **registros completos**.  
✅ Se usa frecuentemente para **almacenar los resultados de consultas SQL** usando BULK COLLECT.

**🔹 Sintaxis básica de TABLE**

TYPE nombre\_tipo\_tabla IS TABLE OF tipo\_elemento INDEX BY {BINARY\_INTEGER | PLS\_INTEGER};

* nombre\_tipo\_tabla: Es el nombre del tipo que estamos definiendo.
* tipo\_elemento: Es el tipo de datos que almacenará la tabla (por ejemplo, NUMBER, VARCHAR2, o %ROWTYPE).
* INDEX BY BINARY\_INTEGER: Define que el índice de la tabla será un número entero, similar a un array.

**🔹 Ejemplo básico con números**

DECLARE

-- Definimos un tipo de tabla que almacenará números

TYPE tablaNumeros IS TABLE OF NUMBER INDEX BY BINARY\_INTEGER;

-- Creamos una variable de tipo tablaNumeros

datos tablaNumeros;

BEGIN

-- Insertamos valores en la tabla usando índices

datos(1) := 10;

datos(2) := 20;

datos(5) := 50; -- No es necesario que los índices sean consecutivos

-- Recorremos los valores con un bucle

FOR i IN datos.FIRST..datos.LAST LOOP

IF datos.EXISTS(i) THEN -- Verificamos si existe el índice

dbms\_output.put\_line('Valor en índice ' || i || ': ' || datos(i));

END IF;

END LOOP;

END;

/

**🔥 ¿Qué hace este código?**

1. Define una **tabla de números** llamada datos.
2. Asigna valores en los índices **1, 2 y 5**.
3. Usa un **bucle** para recorrer los valores entre datos.FIRST y datos.LAST, validando que existen con EXISTS().
4. Muestra los valores en la salida con dbms\_output.put\_line.

**🔹 Ejemplo usando registros**

Podemos almacenar registros completos en una tabla. Es decir, cada posición del array contendrá una fila completa de datos.

DECLARE

-- Definimos un tipo de registro

TYPE emp\_registro IS RECORD (

emp\_no NUMBER,

emp\_name VARCHAR2(70),

salary NUMBER

);

-- Definimos un tipo de tabla que almacenará registros del tipo emp\_registro

TYPE tablaEmpleados IS TABLE OF emp\_registro INDEX BY BINARY\_INTEGER;

-- Creamos una variable de tipo tablaEmpleados

empleados tablaEmpleados;

BEGIN

-- Insertamos registros en la tabla

empleados(1).emp\_no := 1001;

empleados(1).emp\_name := 'John';

empleados(1).salary := 3000;

empleados(2).emp\_no := 1002;

empleados(2).emp\_name := 'Alice';

empleados(2).salary := 3500;

-- Recorremos la colección

FOR i IN empleados.FIRST..empleados.LAST LOOP

IF empleados.EXISTS(i) THEN

dbms\_output.put\_line('Empleado: ' || empleados(i).emp\_name || ', Salario: ' || empleados(i).salary);

END IF;

END LOOP;

END;

/

**🔥 ¿Qué hace este código?**

1. Define un **tipo de registro** emp\_registro con 3 campos (emp\_no, emp\_name, salary).
2. Crea un **tipo de tabla** tablaEmpleados que almacena registros completos.
3. Inserta datos en los índices **1 y 2**.
4. Recorre la colección y muestra los datos de los empleados.

**🔹 Funciones y atributos útiles en TABLE**

| **Función** | **Descripción** |
| --- | --- |
| **datos.FIRST** | Devuelve el menor índice en la tabla. |
| **datos.LAST** | Devuelve el mayor índice en la tabla. |
| **datos.EXISTS(i)** | Devuelve TRUE si el índice i existe en la tabla. |
| **datos.COUNT** | Devuelve el número total de elementos en la tabla. |
| **datos.DELETE** | Borra todos los elementos de la tabla. |
| **datos.DELETE(i)** | Borra solo el elemento con índice i. |
| **datos.DELETE(i, j)** | Borra los elementos en el rango de índices i a j. |
| **datos.NEXT(i)** | Devuelve el siguiente índice después de i. |
| **datos.PRIOR(i)** | Devuelve el índice anterior a i. |

**🔹 Ejemplo de DELETE, COUNT, EXISTS**

DECLARE

TYPE tablaNumeros IS TABLE OF NUMBER INDEX BY BINARY\_INTEGER;

datos tablaNumeros;

BEGIN

datos(1) := 10;

datos(2) := 20;

datos(3) := 30;

dbms\_output.put\_line('Total elementos: ' || datos.COUNT); -- 3

datos.DELETE(2); -- Borrar el índice 2

IF NOT datos.EXISTS(2) THEN

dbms\_output.put\_line('Índice 2 eliminado');

END IF;

dbms\_output.put\_line('Primer índice: ' || datos.FIRST); -- 1

dbms\_output.put\_line('Último índice: ' || datos.LAST); -- 3

datos.DELETE; -- Borra todos los datos

dbms\_output.put\_line('Total después de DELETE: ' || datos.COUNT); -- 0

END;

/

**🔹 Usando BULK COLLECT con TABLE**

Si queremos almacenar **varias filas de una consulta** en una tabla, podemos usar BULK COLLECT.

DECLARE

-- Definir tabla para almacenar múltiples registros

TYPE tablaEmpleados IS TABLE OF emp%ROWTYPE INDEX BY BINARY\_INTEGER;

-- Variable para almacenar los empleados

empleados tablaEmpleados;

BEGIN

-- Almacenar varias filas en la colección

SELECT \* BULK COLLECT INTO empleados FROM emp WHERE salary > 3000;

-- Recorrer la colección

FOR i IN empleados.FIRST..empleados.LAST LOOP

dbms\_output.put\_line('Empleado: ' || empleados(i).emp\_name || ', Salario: ' || empleados(i).salary);

END LOOP;

END;

/

**🔥 ¿Qué hace este código?**

1. Define un **tipo de tabla** basado en la estructura de emp%ROWTYPE.
2. Usa BULK COLLECT INTO para cargar todos los empleados con salario mayor a 3000.
3. Recorre la colección y muestra los resultados.

**🔥 Resumen Final**

✔ TABLE es una **colección** que crece dinámicamente y se accede con un índice.  
✔ Puede almacenar **datos simples** (NUMBER, VARCHAR2, etc.) o **registros completos** (%ROWTYPE).  
✔ **No es necesario que los índices sean consecutivos**.  
✔ Se usa mucho para **manejar múltiples registros en memoria** antes de insertarlos o procesarlos.  
✔ Con BULK COLLECT, podemos almacenar **varios registros de una consulta** en una colección y procesarlos más rápido.

¿Te queda más claro? 😃