GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: BCA - IOP

Course Code: BCAS3031

Course Name: PL/SQL & Cursors and Triggers

Dr. T. Poongodi

Associate Professor

PL/SQL - Collections (Associative array, VARRAY and Nested Tables)

A composite data type stores values that have internal components and internal components can be either scalar or composite.

Internal components can be of same data type and different data type.

PL/SQL allows us to define two kinds of composite data types:

1. Collection - The internal components must have the same data type and access each element of a collection variable by its unique index, **Syntax: variable_name(index).**

2. Record - The internal components can have different data types and access each field of a record variable by its name, **Syntax: variable_name.field_name**.

Collections in PL/SQL

Oracle provides three types of collections.

1. Index-by Table(associative array),
2. Nested Tables, and
3. VARRAY.

All these collections are like a single dimension array. Syntax of collection declaration:-

```
TYPE  type  IS  -- type is collection variable name, a valid identifier
{ assoc_array_type_def
| varray_type_def
| nested_table_type_def
} ;
```

Collection can be created in following ways:

1. Defines a collection type and then declare a variable of that type.
2. Use %TYPE to declare a collection variable of the same type as a previously declared collection variable.

Associative array(Indexed Tables):-

- Associative array is a set of **key-value pairs** and each key should be unique index.
- The data type of index can be either a string type or PLS_INTEGER.
- Indexes are stored in sort order, not creation order.
- Syntax of associative array type creation :

```
TYPE type IS {
--assoc_array_type_def
TABLE OF datatype [ NOT NULL ]
INDEX BY { PLS_INTEGER | BINARY_INTEGER | VARCHAR2 ( v_size ) |
data_type } };
```

- On combining both collection declaration and associate table type declaration, create associative array and store key value pairs and perform various operation on it.
- Create TYPE of associative array named as address and then create a variable employee_address of TYPE address.

```
DECLARE
--Associative array type indexed by BINARY_NUMBER
TYPE address IS TABLE OF VARCHAR2(200) INDEX BY BINARY_INTEGER
--Associative array variable of type address
employees_address address;
BEGIN
employees_address('01') := 'Hyderabad, INDIA';
employees_address('02') := 'Banglore, INDIA’;
employees_address('03') := 'NY, USA’;

--Collection operations
-- FIRST and NEXT gives first and next element of collection
```

```
DBMS_OUTPUT.PUT_LINE('FIRST and LAST ELEMENT key of collection are '
|| employees_address.FIRST || ' and ' || employees_address.LAST);
--COUNT()method gives total no of elements in collection
DBMS_OUTPUT.PUT_LINE('Total no of elements in collection '
|| employees_address.COUNT);
--EXISTS check for existence of key
IF employees_address.EXISTS(02) THEN
      employees_address.DELETE(02);
END IF;
DBMS_OUTPUT.PUT_LINE('Total no of elements in collection after delete '
|| employees_address.COUNT);
END;
```

```
=========Sample output==========
FIRST and LAST ELEMENT key of collection are
1 and 3
Total no of elements in collection 3
Total no of elements in collection after delete 2
```

- **VARRAY**:- It is variable-size array and element counts in it can vary from 0 to declared maximum size.

Characteristics of VARRAY:-

- Elements of VARRAY can be accessed by variable_name(index).VARRY index starts from 1 (lowest_index = 1) and it can go up to maximum size of VARRAY.
- As contrast to associative array, **it can be persisted in database table** and order of elements (indexes and element order) remain stable.

- VARRAY has constructor support as contrast to Associative array that does not support collection constructor.
- VARRAY is stored as a single object in a column in database table.(if size of object is more than 4KB then it is stored separately but in same namespace).

VARRAY creation and its initialization:-

Syntax:

varray_type_def with collection

```
-- size_limit: upper limit of VARRAY(maximum that many elements can be stored)
TYPE type IS { VARRAY | [ VARYING ] ARRAY } ( size_limit )
OF datatype [ NOT NULL ]
```

Consider following sample program which creates a VARRY to store address information of employees and initialize it with constructor.

Here ADDRESS is VARRAY type with upper limit of container 3 and using constructor collection of type ADDRESS created is returned to emp_address.

```
DECLARE
-- VARRAY type declaration of type VARCHAR, upperlimit 3
TYPE ADDRESS IS VARRAY(3) OF VARCHAR2(45);
-- varray variable initialized with constructor of type ADDRESS
emp_address ADDRESS := ADDRESS('HYD,IND',  'NY,USA', 'BANG,IND’);
BEGIN
DBMS_OUTPUT.PUT_LINE('VARRAY elements count is ‘
        || emp_address.COUNT);
DBMS_OUTPUT.PUT_LINE('Address display - Iteration over VARRAY’);
        --emp_address.FIRST= 1 and emp_address.LAST = 3
```

```
FOR i IN emp_address.FIRST..emp_address.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(i || '. address is ' || emp_address(i));
END LOOP;
DBMS_OUTPUT.PUT_LINE('Modify emp_address VARRAY ‘);
emp_address(1) := 'Sydney, AUS’;
DBMS_OUTPUT.PUT_LINE('Accessing VARRAY based on index, modified
address is ‘      ||emp_address(1)); -- notice modified value here.
--emp_address.DELETE(2);--Delete operation on VARRAY is not allowed.
END;
```

```
======Sample output=======

VARRAY elements count is 3
Address display - Iteration over VARRAY
1. address is HYD,IND
2. address is NY,USA
3. address is BANG,IND
Modify emp_address VARRAY
Accessing VARRAY based on index, modified
address is Sydney, AUS
```

Where do we use VARRAY:-

If we have prior info of maximum number of elements and we want sequential access of collection.

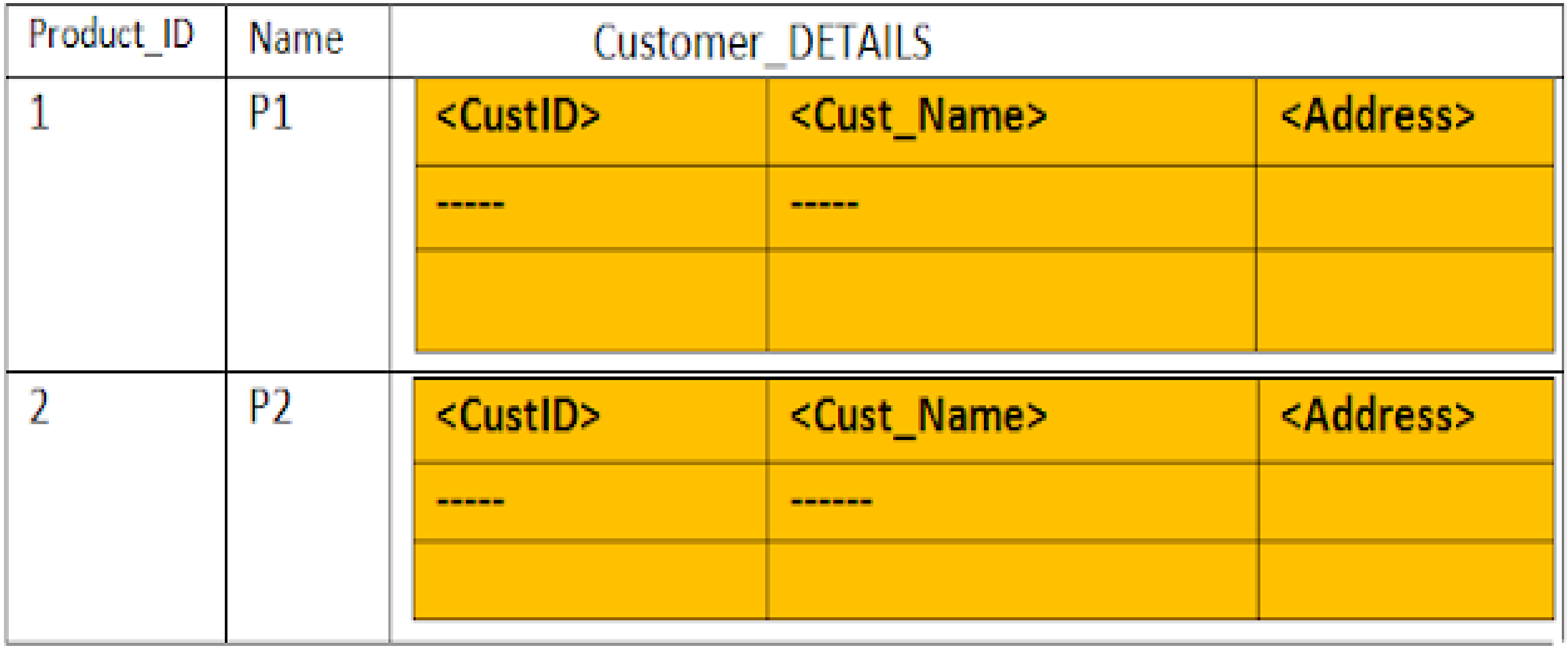
It is not good idea to use VARRAY when collection size is very large, because VARRAY is retrieved at once from database.

Nested Tables:-

- It is a table (with rows and columns) that is stored in database table as data of a column in no particular order.
- When that table is retrieved form database in PL/SQL context, PL/SQL indexes all rows starting from 1 and based on index we can access each row of nested table using method: nested_table_var(index).

- Following diagram shows how Nested tables is stored in database table.
- Highlighted inner table in CUSTOMER_DETAILS column refers to Nested table type and stored as part of column data.

Product table in database with a column of NESTED TABLE type (CUSTOMER DETAILS):



Product_ID	Name	Customer_DETAILS		
1	P1	<CustID>	<Cust_Name>	<Address>
		-----	-----	
2	P2	<CustID>	<Cust_Name>	<Address>
		-----	------	

Nested table creation and its initialization:-

Syntax:

(nested_table_type_def with collection):

```
TYPE type IS {TABLE OF datatype [ NOT NULL ] }
```

Where do we use Nested tables:-

Nested table finds it's usage when index values are not consecutive, maximum number of elements storage is not fixed.

Nested Tables

A nested table is like a one-dimensional array with an arbitrary number of elements. However, a nested table differs from an array in the following aspects –

- An array has a declared number of elements, but a nested table does not. The size of a nested table can increase dynamically.
- An array is always dense, i.e., it always has consecutive subscripts. A nested array is dense initially, but it can become sparse when elements are deleted from it.

Syntax:

```
TYPE type_name IS TABLE OF element_type [NOT NULL];

table_name type_name;
```

- This declaration is similar to the declaration of an **index-by** table, but there is no **INDEX BY** clause.
- A nested table can be stored in a database column. It can further be used for simplifying SQL operations where you join a single-column table with a larger table. An associative array cannot be stored in the database.

```
DECLARE
  TYPE names_table IS TABLE OF VARCHAR2(10);
  TYPE grades IS TABLE OF INTEGER;
  names names_table;
  marks grades;
  total integer;
BEGIN
  names := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
  marks:= grades(98, 97, 78, 87, 92);
  total := names.count;
  dbms_output.put_line('Total '|| total || ' Students');
  FOR i IN 1 .. total LOOP
    dbms_output.put_line('Student:'||names(i)||', Marks:' || marks(i));
  end loop;
END;
/
```

```
Total 5 Students
Student:Kavita, Marks:98
Student:Pritam, Marks:97
Student:Ayan, Marks:78
Student:Rishav, Marks:87
Student:Aziz, Marks:92


PL/SQL procedure successfully completed.
```

Collection Methods

PL/SQL provides the built-in collection methods that make collections easier to use. The following table lists the methods and their purpose

EXISTS(n)
Returns TRUE if the nth element in a collection exists; otherwise returns FALSE.

COUNT
Returns the number of elements that a collection currently contains.

LIMIT
Checks the maximum size of a collection.

FIRST
Returns the first (smallest) index numbers in a collection that uses the integer subscripts.

LAST
Returns the last (largest) index numbers in a collection that uses the integer subscripts.

PRIOR(n)
Returns the index number that precedes index n in a collection.

NEXT(n)
Returns the index number that succeeds index n.

EXTEND
Appends one null element to a collection.

EXTEND(n)
Appends n null elements to a collection.

EXTEND(n,i)
Appends n copies of the ith element to a collection.

TRIM
Removes one element from the end of a collection.

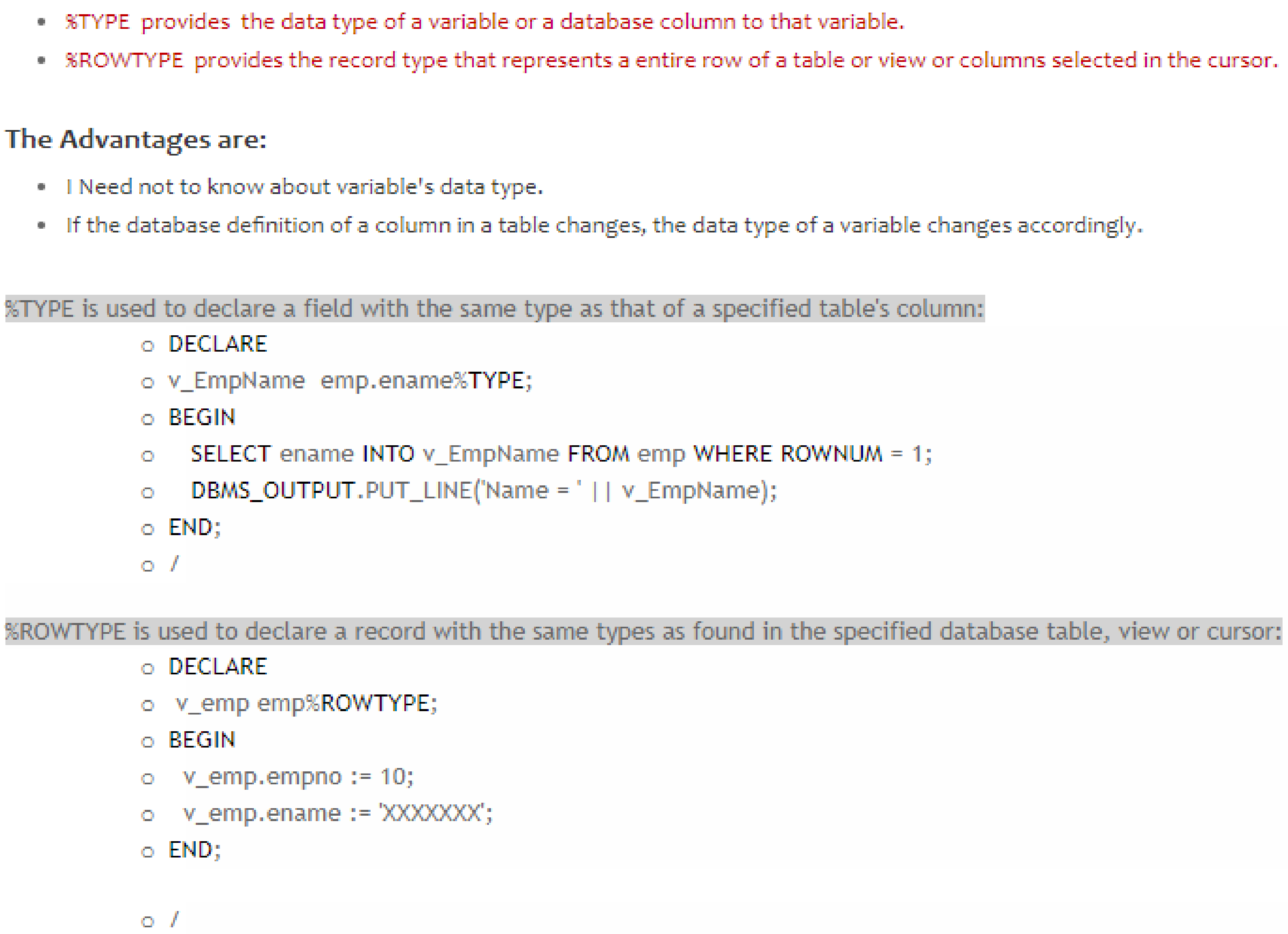
TRIM(n)
Removes n elements from the end of a collection.

DELETE
Removes all elements from a collection, setting COUNT to 0.

DELETE(n)
Removes the nth element from an associative array with a numeric key or a nested table. If the associative array has a string key, the element corresponding to the key value is deleted. If n is null, DELETE(n) does nothing.

DELETE(m,n)
Removes all elements in the range m..n from an associative array or nested table. If m is larger than n or if m or n is null, DELETE(m,n)does nothing.

%TYPE Vs. %RowType



- %TYPE provides the data type of a variable or a database column to that variable.
- %ROWTYPE provides the record type that represents a entire row of a table or view or columns selected in the cursor.

The Advantages are:

- I Need not to know about variable's data type.
- If the database definition of a column in a table changes, the data type of a variable changes accordingly.

%TYPE is used to declare a field with the same type as that of a specified table's column:

```
DECLARE
v_EmpName  emp.ename%TYPE;
BEGIN
   SELECT ename INTO v_EmpName FROM emp WHERE ROWNUM = 1;
   DBMS_OUTPUT.PUT_LINE('Name = ' || v_EmpName);
END;
/
```

%ROWTYPE is used to declare a record with the same types as found in the specified database table, view or cursor:

```
DECLARE
 v_emp emp%ROWTYPE;
BEGIN
  v_emp.empno := 10;
  v_emp.ename := 'XXXXXXX';
END;

/
```

- A Collection is an ordered group of elements of particular data types. It can be a collection of simple data type or complex data type (like user-defined or record types).
- In the collection, each element is identified by a term called **"subscript."** Each item in the collection is assigned with a unique subscript. The data in that collection can be manipulated or fetched by referring to that unique subscript.

- Collections are classified based on the structure, subscript, and storage as shown below.
    - Index-by-tables (Associative Array)
    - Nested tables
    - Varrays (Variable size Array)
- At any point, data in the collection can be referred by three terms Collection name, Subscript, Field/Column name as

"<collection_name>(<subscript>).<column_name>"

Varrays (Variable-size array)

- Varray is a collection method in which the size of the array is fixed. The array size cannot be exceeded than its fixed value. The subscript of the Varray is of a numeric value.

Following are the attributes of Varrays.

- Upper limit size is fixed
- Starting with the subscript '1'
- It is always dense, i.e. we cannot delete any array elements. Varray can be deleted as a whole, or it can be trimmed from the end.
- Since it always is dense in nature, it has very less flexibility.
- It is more appropriate to use when the array size is known and to perform similar activities on all the array elements.
- The subscript and sequence always remain stable, i.e. the subscript and count of the collection is always same.
- They need to be initialized before using them in programs.
- It can be created as a database object, which is visible throughout the database or inside the subprogram, which can be used only in that subprogram.

The below figure will explain the memory allocation of Varray (dense) diagrammatically.

Subscript	1	2	3	4	5	6	7
Value	Xyz	Dfv	Sde	Cxs	Vbc	Nhu	Qwe

Syntax for VARRAY:

In the syntax, type_name is declared as VARRAY of the type 'DATA_TYPE' for the given size limit. The data type can be either simple or complex type.

```
TYPE <type_name> IS VARRAY (<SIZE>) OF <DATA_TYPE>;
```

Nested Tables

- A Nested table is a collection in which the size of the array is not fixed. It has the numeric subscript type.
- The Nested table has no upper size limit.
- Since the upper size limit is not fixed, the collection, memory needs to be extended each time before we use it. We can extend the collection using 'EXTEND' keyword.
- Starting with the subscript '1'.
- It can be of both **dense and sparse**, i.e. we can create the collection as a dense, and we can also delete the individual array element randomly, which make it as sparse.
- It gives more flexibility regarding deleting the array element.
- It is stored in the system generated database table and can be used in the select query to fetch the values.
- The subscript and sequence are not stable, i.e. the subscript and the count of the array element can vary.
- They need to be initialized before using them in programs.
- It can be created as a database object, which is visible throughout the database or inside the subprogram, which can be used only in that subprogram.

- The below figure will explain the memory allocation of Nested Table (dense and sparse) diagrammatically. The space denotes the empty element in a collection i.e. sparse.

Subscript	1	2	3	4	5	6	7
Value (dense)	Xyz	Dfv	Sde	Cxs	Vbc	Nhu	Qwe
Value(sparse)	Qwe		Asd	Afg		Asd	Wer

Syntax for Nested Table:

TYPE <tvpe name> IS TABLE OF <DATA TYPE>;

In the above syntax, type_name is declared as Nested table collection of the type 'DATA_TYPE'. The data type can be either simple or complex type.

Index-by-table

- Index-by-table is a collection in which the array size is not fixed. The subscript can consist be defined by the user.
- The subscript can of integer or strings. At the time of creating the collection, the subscript type should be mentioned.
- Not stored sequentially.
- Always sparse in nature.
- Array size is not fixed.
- They cannot be stored in the database column. They shall be created and used in any program in that particular session.
- They give more flexibility in terms of maintaining subscript.
- Appropriate to use for relatively smaller collective values in which the collection can be initialized and used within the same subprograms.
- They need not be initialized before start using them.
- It cannot be created as a database object. It can only be created inside the subprogram, which can be used only in that subprogram.

- The below figure will explain the memory allocation of Nested Table (sparse) diagrammatically. The space denotes the empty element in a collection i.e. sparse.

Subscript (varchar)	FIRST	SECOND	THIRD	FOURTH	FIFTH	SIXTH	SEVENTH
Value(sparse)	Qwe		Asd	Afg		Asd	Wer

Syntax for Index-by-Table

TYPE <type_name> IS TABLE OF <DATA_TYPE> INDEX BY VARCHAR2 (10);

In the above syntax, type_name is declared as an index-by-table collection of the type 'DATA_TYPE'. The data type can be either simple or complex type. The subsciprt/index variable is given as VARCHAR2 type with maximum size as 10.

Use index by tables when:

- The collection can be made at runtime in the memory when the package/ procedure is initialized
- The data volume is unknown beforehand
- The subscript values are flexible (e.g. strings, negative numbers, non-sequential)
- Do not need to store the collection in the database

Use nested tables when:

- The data needs to be stored in the database
- The number of elements in the collection is not known in advance
- The elements of the collection may need to be retrieved out of sequence
- Updates and deletions affect only some elements, at arbitrary locations

Use varrays when:

- The data needs to be stored in the database
- The number of elements of the varray is known in advance
- The data from the varray is accessed in sequence
- Updates and deletions happen on the varray as a whole and not on arbitrarily located elements in the varray

```
DECLARE
  TYPE nested_type IS TABLE OF VARCHAR2(30);
  TYPE varray_type IS VARRAY(5) OF INTEGER;
  TYPE assoc_array_num_type IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  TYPE assoc_array_str_type IS TABLE OF VARCHAR2(32) INDEX BY PLS_INTEGER;
  TYPE assoc_array_str_type2 IS TABLE OF VARCHAR2(32) INDEX BY VARCHAR2(64);
  v1 nested_type;
  v2 varray_type;
  v3 assoc_array_num_type;
  v4 assoc_array_str_type;
  v5 assoc_array_str_type2;
BEGIN
-- an arbitrary number of strings can be inserted v1
  v1 := nested_type('Shipping','Sales','Finance','Payroll');
  v2 := varray_type(1, 2, 3, 4, 5); -- Up to 5 integers
  v3(99) := 10; -- Just start assigning to elements
  v3(7) := 100; -- Subscripts can be any integer values
  v4(42) := 'Smith'; -- Just start assigning to elements
  v4(54) := 'Jones'; -- Subscripts can be any integer values
  v5('Canada') := 'North America'; -- Just start assigning to elements
  v5('Greece') := 'Europe';       -- Subscripts can be string values
END;
/
```

Thank You