A collection is an ordered group of elements having the same data type. Each element is identified by a unique subscript that represents its position in the collection.

PL/SQL provides three collection types:

- Index-by tables or Associative array
- Nested table
- Variable-size array or Varray

Oracle documentation provides the following characteristics for each type of collections:

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Number of Elements</th>
<th>Subscript Type</th>
<th>Dense or Sparse</th>
<th>Where Created</th>
<th>Can Be Object Type Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associative array (Index-by table)</td>
<td>Unbounded</td>
<td>String or integer</td>
<td>Either</td>
<td>Only in PL/SQL block</td>
<td>No</td>
</tr>
<tr>
<td>Nested table</td>
<td>Unbounded</td>
<td>Integer</td>
<td>Starts dense, can become sparse</td>
<td>Either in PL/SQL block or at schema level</td>
<td>Yes</td>
</tr>
<tr>
<td>Variable-size array (Varray)</td>
<td>Bounded</td>
<td>Integer</td>
<td>Always dense</td>
<td>Either in PL/SQL block or at schema level</td>
<td>Yes</td>
</tr>
</tbody>
</table>

We have already discussed varray in the chapter 'PL/SQL arrays'. In this chapter, we will discuss PL/SQL tables.

Both types of PL/SQL tables, i.e., index-by tables and nested tables have the same structure and their rows are accessed using the subscript notation. However, these two types of tables differ in one aspect; the nested tables can be stored in a database column and the index-by tables cannot.

**Index-By Table**

An index-by table also called an associative array is a set of key-value pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

An index-by table is created using the following syntax. Here, we are creating an index-by table named `table_name` whose keys will be of `subscription_type` and associated values will be of `element_type`

```sql
TYPE type_name IS TABLE OF element_type [NOT NULL] INDEX BY subscript_type;
```

### Example:

Following example shows how to create a table to store integer values along with names and later it prints the same list of names.

```sql
DECLARE
    TYPE salary IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
    salary_list salary;
    name VARCHAR2(20);
BEGIN
```

```sql
```
-- adding elements to the table
salary_list('Rajnish') := 62000;
salary_list('Minakshi') := 75000;
salary_list('Martin') := 100000;
salary_list('James') := 78000;

-- printing the table
name := salary_list.FIRST;
WHILE name IS NOT null LOOP
dbms_output.put_line
    ('Salary of ' || name || ' is ' || TO_CHAR(salary_list(name)));
    name := salary_list.NEXT(name);
END LOOP;
END;
/

When the above code is executed at SQL prompt, it produces the following result:

Salary of Rajnish is 62000
Salary of Minakshi is 75000
Salary of Martin is 100000
Salary of James is 78000

PL/SQL procedure successfully completed.

Example:

Elements of an index-by table could also be a %ROWTYPE of any database table or %TYPE of any database table field. The following example illustrates the concept. We will use the CUSTOMERS table stored in our database as:

Select * from customers;

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>AGE</th>
<th>ADDRESS</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ramesh</td>
<td>32</td>
<td>Ahmedabad</td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>Khilan</td>
<td>25</td>
<td>Delhi</td>
<td>1500.00</td>
</tr>
<tr>
<td>3</td>
<td>kaushik</td>
<td>23</td>
<td>Kota</td>
<td>2000.00</td>
</tr>
<tr>
<td>4</td>
<td>Chaitali</td>
<td>25</td>
<td>Mumbai</td>
<td>6500.00</td>
</tr>
<tr>
<td>5</td>
<td>Hardik</td>
<td>27</td>
<td>Bhopal</td>
<td>8500.00</td>
</tr>
<tr>
<td>6</td>
<td>Komal</td>
<td>22</td>
<td>MP</td>
<td>4500.00</td>
</tr>
</tbody>
</table>

DECLARE
    CURSOR c_customers is
    select name from customers;
    TYPE c_list IS TABLE of customers.name%type INDEX BY binary_integer;
    name_list c_list;
    counter integer :=0;
BEGIN
    FOR n IN c_customers LOOP
        counter := counter +1;
        name_list(counter) := n.name;
        dbms_output.put_line('Customer('||counter||'):'||name_list(counter));
    END LOOP;
END;
/

When the above code is executed at SQL prompt, it produces the following result:

Customer(1): Ramesh
Customer(2): Khilan
Customer(3): kaushik
Customer(4): Chaitali
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.

A nested table is created using the following syntax:

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

table_name type_name;
```

This declaration is similar to declaration of an index-by table, but there is no INDEX BY clause.

A nested table can be stored in a database column and so it could 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.

Example:

The following examples illustrate the use of nested table:

```sql
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;
/```

When the above code is executed at SQL prompt, it produces the following result:

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

Example:

Elements of a nested table could also be a %ROWTYPE of any database table or %TYPE of any database table field. The following example illustrates the concept. We will use the CUSTOMERS table stored in our database as:
Select * from customers;

<table>
<thead>
<tr>
<th>ID</th>
<th>NAME</th>
<th>AGE</th>
<th>ADDRESS</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ramesh</td>
<td>32</td>
<td>Ahmedabad</td>
<td>2000.00</td>
</tr>
<tr>
<td>2</td>
<td>Khilan</td>
<td>25</td>
<td>Delhi</td>
<td>1500.00</td>
</tr>
<tr>
<td>3</td>
<td>kaushik</td>
<td>23</td>
<td>Kota</td>
<td>2000.00</td>
</tr>
<tr>
<td>4</td>
<td>Chaitali</td>
<td>25</td>
<td>Mumbai</td>
<td>6500.00</td>
</tr>
<tr>
<td>5</td>
<td>Hardik</td>
<td>27</td>
<td>Bhopal</td>
<td>8500.00</td>
</tr>
<tr>
<td>6</td>
<td>Komal</td>
<td>22</td>
<td>MP</td>
<td>4500.00</td>
</tr>
</tbody>
</table>

DECLARE
CURSOR c_customers is
SELECT  name FROM customers;

TYPE c_list IS TABLE of customers.name%type;
name_list c_list := c_list();
counter integer :=0;
BEGIN
FOR n IN c_customers LOOP
  counter := counter +1;
  name_list.extend;
  name_list(counter) := n.name;
  dbms_output.put_line('Customer('||counter||'):'||name_list(counter));
END LOOP;
END;
/

When the above code is executed at SQL prompt, it produces the following result:

Customer(1): Ramesh
Customer(2): Khilan
Customer(3): kaushik
Customer(4): Chaitali
Customer(5): Hardik
Customer(6): Komal

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:

<table>
<thead>
<tr>
<th>S.N.</th>
<th>Method Name &amp; Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EXISTS[n]</td>
</tr>
<tr>
<td></td>
<td>Returns TRUE if the nth element in a collection exists; otherwise returns FALSE.</td>
</tr>
<tr>
<td>2</td>
<td>COUNT</td>
</tr>
<tr>
<td></td>
<td>Returns the number of elements that a collection currently contains.</td>
</tr>
<tr>
<td>3</td>
<td>LIMIT</td>
</tr>
<tr>
<td></td>
<td>Checks the Maximum Size of a Collection.</td>
</tr>
<tr>
<td>4</td>
<td>FIRST</td>
</tr>
<tr>
<td></td>
<td>Returns the first smallest index numbers in a collection that uses integer subscripts.</td>
</tr>
<tr>
<td>5</td>
<td>LAST</td>
</tr>
<tr>
<td></td>
<td>Returns the last largest index numbers in a collection that uses integer subscripts.</td>
</tr>
<tr>
<td>6</td>
<td>PRIOR[n]</td>
</tr>
</tbody>
</table>
Returns the index number that precedes index n in a collection.

NEXT<sub>n</sub>
Returns the index number that succeeds index n.

EXTEND
Appends one null element to a collection.

EXTEND<sub>n</sub>
Appends n null elements to a collection.

EXTEND<sub>n, i</sub>
Appends copies of the ith element to a collection.

TRIM
Removes one element from the end of a collection.

TRIM<sub>n</sub>
Removes n elements from the end of a collection.

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

DELETE<sub>n</sub>
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<sub>n</sub> does nothing.

DELETE<sub>m, n</sub>
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<sub>m, n</sub> does nothing.

Collection Exceptions

The following table provides the collection exceptions and when they are raised:

<table>
<thead>
<tr>
<th>Collection Exception</th>
<th>Raised in Situations</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLLECTION_IS_NULL</td>
<td>You try to operate on an atomically null collection.</td>
</tr>
<tr>
<td>NO_DATA_FOUND</td>
<td>A subscript designates an element that was deleted, or a nonexistent element of an associative array.</td>
</tr>
<tr>
<td>SUBSCRIPT_BEYOND_COUNT</td>
<td>A subscript exceeds the number of elements in a collection.</td>
</tr>
<tr>
<td>SUBSCRIPT_OUTSIDE_LIMIT</td>
<td>A subscript is outside the allowed range.</td>
</tr>
<tr>
<td>VALUE_ERROR</td>
<td>A subscript is null or not convertible to the key type. This exception might occur if the key is defined as a PLS_INTEGER range, and the subscript is outside this range.</td>
</tr>
</tbody>
</table>