Lab 8. Collections

Databases II

A collection is a group of elements of the same datatype. Each element is identified by a unique subscript that represents its position in the collection.

8.1.PL/SQL Tables

A PL/SQL table is similar to a one-column database table. The rows of a PL/SQL table are not stored in any predefined order, yet when they are retrieved in a variable, each row is assigned a consecutive subscript starting at 1.

There are two types of PL/SQL tables: associative tables (formerly called index-by tables) and nested tables. They have the same structure, and their rows are accessed in the same way using subscript notation. The main difference between these two types is that nested tables can be stored in a database column, and associative arrays cannot.

ASSOCIATIVE ARRAYS

The general syntax for creating an associative array is as follows :

```
TYPE type_name IS TABLE OF element_type [NOT NULL]
   INDEX BY element_type;
table_name TYPE_NAME;
```

Declaring an associative array has two steps. First, a table structure is defined using the TYPE statement, where `type_name` is the name of the type that is used in the second step to declare an actual table. An `element_type` is any PL/SQL datatype, such as NUMBER, VARCHAR2, or DATE, with some restrictions.

EXAMPLE

```
DECLARE
   TYPE last_name_type IS TABLE OF student.last_name%TYPE
      INDEX BY BINARY_INTEGER;
   last_name_tab last_name_type;
```

The individual elements of a PL/SQL table are referenced using subscript notation as follows:

```
table_name(subscript)
```

EXAMPLE

```
DECLARE
   CURSOR name_cur IS
      SELECT last_name
        FROM student
       WHERE rownum <= 10;
   TYPE last_name_type IS TABLE OF student.last_name%TYPE
      INDEX BY BINARY_INTEGER;
   last_name_tab last_name_type;
   v_counter INTEGER := 0;
```

```
BEGIN
   FOR name_rec IN name_cur LOOP
      v_counter := v_counter + 1;
      last_name_tab(v_counter) := name_rec.last_name;
      DBMS_OUTPUT.PUT_LINE ('last_name('||v_counter||'): '||
         last_name_tab(v_counter));
   END LOOP;

END;
```

NESTED TABLES

The general syntax for creating a nested table is:

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

The declaration is very similar to the one of an associative array, except that it has no INDEX BY BINARY_INTEGER clause. As in the case of an associative array, restrictions apply to an `element_type` of a nested table. A nested table must be initialized before its individual elements can be referenced.

```
DECLARE
   CURSOR name_cur IS
      SELECT last_name
        FROM student
      WHERE rownum <= 10;
   TYPE last_name_type IS TABLE OF student.last_name%TYPE;
   last_name_tab last_name_type := last_name_type();
   v_counter INTEGER := 0;
BEGIN

FOR name_rec IN name_cur LOOP
v_counter := v_counter + 1; last_name_tab.EXTEND;
last_name_tab(v_counter) := name_rec.last_name;

      DBMS_OUTPUT.PUT_LINE ('last_name('||v_counter||'): '||
         last_name_tab(v_counter));
   END LOOP;
END;
```

The nested table is initialized at the time of declaration. This means that it is empty, but non-null. In the cursor loop is a statement with one of the collection methods, EXTEND. This method allows you to increase the size of the collection. Note that the EXTEND method cannot be used with associative arrays.

COLLECTION METHODS

A collection method is a built-in function that is called using dot notation as follows:

```
collection_name.method_name
```

The following list explains collection methods that allow you to manipulate or gain information about a particular collection:

- EXISTS returns TRUE if a specified element exists in a collection. This method can be used to avoid SUBSCRIPT_OUTSIDE_LIMIT exceptions.
- COUNT returns the total number of elements in a collection.

- EXTEND increases the size of a collection.
- DELETE deletes either all elements, elements in the specified range, or a particular element from a collection.
- FIRST and LAST return subscripts of the first and last elements of a collection. Note that if the first elements of a nested table are deleted, the FIRST method returns a value greater than 1. If elements have been deleted from the middle of a nested table, the LAST method returns a value greater than the COUNT method.
- PRIOR and NEXT return subscripts that precede and succeed a specified collection subscript.
- TRIM removes either one or a specified number of elements from the end of a collection.

EXAMPLE

```
DECLARE
   TYPE index_by_type IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
   index_by_table index_by_type;
   TYPE nested_type IS TABLE OF NUMBER;
   nested_table nested_type := nested_type(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
BEGIN
   -- Populate index by table
   FOR i IN 1..10 LOOP
      index_by_table(i) := i;
   END LOOP;
   IF index_by_table.EXISTS(3) THEN
      DBMS_OUTPUT.PUT_LINE ('index_by_table(3) ='||index_by_table(3));

   END IF;

   -- delete 10th element from a collection
   nested_table.DELETE(10);
   -- delete elements 1 through 3 from a collection
   nested_table.DELETE(1,3);
   index_by_table.DELETE(10);
   DBMS_OUTPUT.PUT_LINE ('nested_table.COUNT ='||nested_table.COUNT);
   DBMS_OUTPUT.PUT_LINE ('index_by_table.COUNT = '||index_by_table.COUNT);
   DBMS_OUTPUT.PUT_LINE ('nested_table.FIRST ='||nested_table.FIRST);
   DBMS_OUTPUT.PUT_LINE ('nested_table.LAST = '||nested_table.LAST);
   DBMS_OUTPUT.PUT_LINE ('index_by_table.FIRST = '||index_by_table.FIRST);
   DBMS_OUTPUT.PUT_LINE ('index_by_table.LAST ='||index_by_table.LAST);
   DBMS_OUTPUT.PUT_LINE ('nested_table.PRIOR(2) = '||nested_table. PRIOR(2));
   DBMS_OUTPUT.PUT_LINE ('nested_table.NEXT(2) = '||nested_table.NEXT(2));
   DBMS_OUTPUT.PUT_LINE ('index_by_table.PRIOR(2) = '||index_by_table.PRIOR(2));
   DBMS_OUTPUT.PUT_LINE ('index_by_table.NEXT(2) = '||index_by_table.NEXT(2));
   -- Trim last two elements
   nested_table.TRIM(2);
   -- Trim last element
   nested_table.TRIM;
   DBMS_OUTPUT.PUT_LINE('nested_table.LAST = '||nested_table.LAST);
END;
```

8.2 Varray

A varray has a maximum size. A subscript of a varray has a fixed lower bound equal to 1, and an upper bound that is extensible if such a need arises. The general syntax for creating a varray is :

```
TYPE type_name IS {VARRAY | VARYING ARRAY} (size_limit) OF
  element_type [NOT NULL];
varray_name TYPE_NAME;
```

First, a varray structure is defined using the TYPE statement, where `type_name` is the name of the type that is used in the second step to declare an actual varray. Notice that there are two variations of the type, VARRAY and VARYING ARRAY. A `size_limit` is a positive integer literal that specifies the upper bound of a varray.

EXAMPLE

```
DECLARE
   TYPE last_name_type IS VARRAY(10) OF student.last_name%TYPE;
   last_name_varray last_name_type;
```

In this example, type `last_name_type` is declared as a varray of ten elements based on the column LAST_NAME of the STUDENT table. Next, the actual varray `last_name_varray` is declared based on the `last_name_type`.

Similar to nested tables, a varray is automatically NULL when it is declared and must be initialized before its individual elements can be referenced.

EXAMPLE

```
DECLARE
          CURSOR name_cur IS
             SELECT last_name
               FROM student
              WHERE rownum <= 10;
          TYPE last_name_type IS VARRAY(10) OF student.last_name%TYPE;
          last_name_varray last_name_type := last_name_type();
          v_counter INTEGER := 0;
       BEGIN
          FOR name_rec IN name_cur LOOP
             v_counter := v_counter + 1;
             last_name_varray.EXTEND;
             last_name_varray(v_counter) := name_rec.last_name;
             DBMS_OUTPUT.PUT_LINE ('last_name('||v_counter||'): '||
                last_name_varray(v_counter));
          END LOOP;

       END;
```

8.3 Multilevel Collections

A varray of varrays consists of three elements, where each element is a varray consisting of four integers. To reference an individual element of a varray of varrays, you use the following syntax:

```
varray_name(subscript of the outer varray)(subscript of the inner varray)
```

EXAMPLE

```
       DECLARE
          TYPE varray_type1 IS VARRAY(4) OF INTEGER;
          TYPE varray_type2 IS VARRAY(3) OF varray_type1;
          varray1 varray_type1 := varray_type1(2, 4, 6, 8);
          varray2 varray_type2 := varray_type2(varray1);
       BEGIN
          DBMS_OUTPUT.PUT_LINE ('Varray of integers');
          FOR i IN 1..4 LOOP
             DBMS_OUTPUT.PUT_LINE ('varray1('||i||'): '||varray1(i));
```

```
   END LOOP;
   varray2.EXTEND;
   varray2(2) := varray_type1(1, 3, 5, 7);
   DBMS_OUTPUT.PUT_LINE (chr(10)||'Varray of varrays of integers');
   FOR i IN 1..2 LOOP
      FOR j IN 1..4 LOOP
         DBMS_OUTPUT.PUT_LINE
            ('varray2('||i||')('||j||'): '||varray2(i)(j));
      END LOOP;
   END LOOP;
END;
```

The declaration portion of this example defines two varray types. The first, `varray_type1`, is based on the INTEGER datatype and can contain up to four elements. The second, `varray_type2`, is based on `varray_type1` and can contain up to three elements where each individual element may contain up to four elements. Next, you declare two varrays based on the types just described. The first varray, `varray1`, is declared as `varray_type1` and is initialized so that its four elements are populated with the first four even numbers. The second varray, `varray2`, is declared as `varray_type2` so that each element is a varray consisting of four integers and is initialized so that its first varray element is populated.

Exercices:

1. Create an associative array, and populate it with the instructor's full name. In other words, each row of the associative array should contain the first name, middle initial, and last name. Display this information on the screen.
2. Modify the script you just created. Instead of using an associative array, use a varray.
3. Modify the script you just created. Create an additional varray, and populate it with unique course numbers for the courses that each instructor teaches. Display the instructor's name and the list of courses he or she teaches.
4. Find and explain the errors in the following script:

```
DECLARE
   TYPE varray_type1 IS VARRAY(7) OF INTEGER;
   TYPE table_type2 IS TABLE OF varray_type1 INDEX BY
      BINARY_INTEGER;
   varray1 varray_type1 := varray_type1(1, 2, 3);
   table2 table_type2 := table_type2(varray1,
                                     varray_type1(8, 9, 0));
BEGIN
   DBMS_OUTPUT.PUT_LINE ('table2(1)(2): '||table2(1)(2));
   FOR i IN 1..10 LOOP
      varray1.EXTEND;
      varray1(i) := i;
      DBMS_OUTPUT.PUT_LINE ('varray1('||i||'): '||varray1(i));
   END LOOP;
END;
```