Exercises for Chapter 15: Collections

The Labs below provide you with exercises and suggested answers with discussion related to how those answers resulted. The most important thing to realize is whether your answer works. You should figure out the implications of the answers here and what the effects are from any different answers you may come up with.

Lab 15.1 PL/SQL Tables

Answer the following questions:

Associative Arrays

In this exercise, you will modify a script that populates an associative array with course descriptions. Create the following PL/SQL script:

For Example *ch15_5a.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE
      INDEX BY PLS_INTEGER;
   course_tab course_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab(v_counter) := course_rec.description;
   END LOOP;
END;
```

Answer the following questions:

a) Explain the script ch15_5a.sql.

Answer: The declaration section of the script contains definition of the associative array type, `course_type`. This type is based on the column `DESCRIPTION` of the table `COURSE`. Next, the actual associative array is declared as `course_tab`.

The executable section of the script populates associative array `course_tab` in the cursor `FOR` loop. Each element of the associative array is referenced by its subscript, `v_counter`. For each iteration of the loop, the value of `v_counter` is incremented by 1 so that each new description value is stored in the new row of the associative array.

b) Modify the script so that rows of the associative array are displayed on the screen.

Answer: The script should look similar to the following script. Newly added statements are shown in bold.

For Example *ch15_5b.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE
      INDEX BY PLS_INTEGER;
   course_tab course_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab(v_counter):= course_rec.description;
      DBMS_OUTPUT.PUT_LINE('course('||v_counter||'): '||course_tab(v_counter));
   END LOOP;
END;
```

Consider another version of the same script.

For Example *ch15_5c.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE
      INDEX BY PLS_INTEGER;
   course_tab course_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab(v_counter):= course_rec.description;
   END LOOP;

   FOR i IN 1..v_counter
   LOOP
      DBMS_OUTPUT.PUT_LINE('course('||i||'): '||course_tab(i));
   END LOOP;
END;
```

When run, both versions produce the same output:

```
course(1): Technology Concepts
course(2): Intro to Information Systems
course(3): Intro to Programming
course(4): Programming Techniques
course(5): Hands-On Windows
course(6): Intro to Java Programming
course(7): Intermediate Java Programming
course(8): Advanced Java Programming
course(9): Java Developer I
course(10): Intro to Unix
course(11): Basics of Unix Admin
course(12): Advanced Unix Admin
course(13): Unix Tips and Techniques
course(14): Systems Analysis
course(15): Project Management
course(16): Database Design
course(17): Internet Protocols
course(18): Java for C/C++ Programmers
course(19): GUI Design Lab
course(20): Intro to SQL
course(21): Oracle Tools
course(22): PL/SQL Programming
course(23): Intro to the Internet
course(24): Intro to the BASIC Language
course(25): Operating Systems
course(26): Network Administration
course(27): Java Developer II
course(28): Database System Principles
course(29): Java Developer III
course(30): DB Programming with Java
```

c) Modify the script so that only first and last rows of the associative array are displayed on the screen.

   Answer: The script should look similar to the following script. Changes are shown in bold.

   For Example *ch15_5d.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE
```

```
      INDEX BY PLS_INTEGER;
   course_tab course_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab(v_counter) := course_rec.description;
   END LOOP;
   DBMS_OUTPUT.PUT_LINE('course('||course_tab.FIRST||'): '||
      course_tab(course_tab.FIRST));
   DBMS_OUTPUT.PUT_LINE('course('||course_tab.LAST||'): '||
      course_tab(course_tab.LAST));
END;
```

Consider the statements

```
course_tab(course_tab.FIRST)
```

and

```
course_tab(course_tab.LAST)
```

used in this example. While these statements look somewhat different from the statements that you have seen so far, they produce the same effect as

```
course_tab(1)
```

and

```
course_tab(30)
```

statements because, as mentioned in Chapter 15, the `FIRST` and `LAST` methods return the subscripts of the first and last elements of a collection, respectively. In this example, the associative array contains 30 elements, where the first element has subscript of 1, and the last element has subscript of 30.

This version of the script produces the following output:

```
course(1): Technology Concepts
course(30): DB Programming in Java
```

d) Modify the script by adding the following statements and explain the output produced:

   i) Display the total number of elements in the associative array after it has been populated on the screen.

   ii) Delete the last element, and display the total number of elements of the associative array again.

   iii) Delete the fifth element, and display the total number of elements and the subscript of the last element of the associative array again.

Answer: The script should look similar to the following script. All changes are shown in bold.

For Example *ch15_5e.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
      FROM course;
   TYPE course_type IS TABLE OF course.description%TYPE
      INDEX BY PLS_INTEGER;
   course_tab course_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab(v_counter) := course_rec.description;
   END LOOP;

   -- Display the total number of elements in the associative array
   DBMS_OUTPUT.PUT_LINE ('1. Total number of elements: '||course_tab.COUNT);

   -- Delete the last element of the associative array
   -- Display the total number of elements in the associative array
   course_tab.DELETE(course_tab.LAST);
   DBMS_OUTPUT.PUT_LINE ('2. Total number of elements: '||course_tab.COUNT);

   -- Delete the fifth element of the associative array
   -- Display the total number of elements in the associative array
   -- Display the subscript of the last element of the associative array
   course_tab.DELETE(5);
   DBMS_OUTPUT.PUT_LINE ('3. Total number of elements: '||course_tab.COUNT);
   DBMS_OUTPUT.PUT_LINE ('3. The subscript of the last element: '||course_tab.LAST);
END;
```

When run, this version of the script produces the following output:

```
1. Total number of elements: 30
2. Total number of elements: 29
3. Total number of elements: 28
3. The subscript of the last element: 29
```

First, the total number of the elements in the associative array is calculated via the COUNT method and displayed on the screen.

Second, the last element is deleted via the DELETE and LAST methods, and the total number of the elements in the associative array is displayed on the screen again.

Third, the fifth element is deleted, and the total number of the elements in the associative array and the subscript of the last element are displayed on the screen.

Consider the last two lines on the output. After the fifth element of the associative array is deleted, the COUNT method returns value 28, and the LAST method returns the value 29. Usually, the values returned by the COUNT and LAST methods are equal. However, when an element is deleted from the middle of the associative array, the value returned by the LAST method is greater than the value returned by the COUNT method because the LAST method ignores deleted elements.

Nested Tables

In this exercise, you will modify script created in the previous section of this lab. Instead of using associative arrays you will be asked to use nested tables.

Answer the following questions:

a) Modify the script 15_5a.sql. Instead of using an associative array, use a nested table.

Answer: The new script should look similar to the following script. Changes are highlighted in bold.

For Example *ch15_6a.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE;
   course_tab course_type := course_type();

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab.EXTEND;
      course_tab(v_counter) := course_rec.description;
   END LOOP;
END;
```

b) Modify the script by adding the following statements and explain the output produced:

   i) Delete the last element of the nested table, and then reassign a new value to it. Execute the script.

   ii) Trim the last element of the nested table, and then reassign a new value to it. Execute the script.

Answer:

i) This version of the script should look similar to the following script. Newly added statements are shown in bold.

For Example *ch15_6b.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE;
   course_tab course_type := course_type();

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab.EXTEND;
      course_tab(v_counter) := course_rec.description;
   END LOOP;
   course_tab.DELETE(30);
   course_tab(30) := 'New Course';
END;
```

ii) This version of the script should look similar to the following script. Newly added statements are shown in bold.

For Example *ch15_6c.sql*

```
DECLARE
   CURSOR course_cur IS
      SELECT description
      FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE;
   course_tab course_type := course_type();

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab.EXTEND;
      course_tab(v_counter) := course_rec.description;
   END LOOP;

   course_tab.TRIM;
   course_tab(30) := 'New Course';
END;
```

When run, this version of the script produces the following error:

```
ORA-06533: Subscript beyond count
ORA-06512: at line 19
```

In the previous version of the script, the last element of the nested table is removed via the `DELETE` method. As mentioned in Chapter 15, when the `DELETE` method is used, the PL/SQL keeps a placeholder of the deleted element. Therefore, the statement

```
course_tab(30) := 'New Course';
```

does not cause any errors.

In the current version of the script, the last element of the nested table is removed via the `TRIM` method. In this case, the PL/SQL does not keep placeholder of the trimmed element because the `TRIM` method manipulates the internal size of a collection. As a result, the reference to the trimmed elements causes ‘Subscript beyond count’ error.

c) How would you modify the script created, so that there is no error generated when a new value is assigned to the trimmed element?

Answer: The script should be modified as follows. Changes are shown in bold.

For Example *ch15_6d.sql*

```
DECLARE
   CURSOR course_cur IS
```

```
      SELECT description
        FROM course;

   TYPE course_type IS TABLE OF course.description%TYPE;
   course_tab course_type := course_type();

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR course_rec IN course_cur
   LOOP
      v_counter := v_counter + 1;
      course_tab.EXTEND;
      course_tab(v_counter) := course_rec.description;
   END LOOP;

   course_tab.TRIM;
   course_tab.EXTEND;
   course_tab(30) := 'New Course';
END;
```

In order to reference the trimmed element, the `EXTEND` method is use to increase the size on the collection. As a result, the assignment statement

```
course_tab(30) := 'New Course';
```

does not cause any errors.

Lab 15.2 Varrays

In this exercise, you will need to debug the following script, which populates `city_varray` with 10 cities selected from the `ZIPCODE` table and displays its individual elements on the screen. Create the following PL/SQL script:

For Example *ch15_7a.sql*

```
DECLARE
   CURSOR city_cur IS
      SELECT city
        FROM zipcode
       WHERE rownum <= 10;

   TYPE city_type IS VARRAY(10) OF zipcode.city%TYPE;
   city_varray city_type;

   v_counter PLS_INTEGER := 0;
BEGIN
   FOR city_rec IN city_cur
   LOOP
      v_counter := v_counter + 1;
      city_varray(v_counter) := city_rec.city;
      DBMS_OUTPUT.PUT_LINE('city_varray('||v_counter||'): '||city_varray(v_counter));
   END LOOP;
END;
```

Execute the script, and then answer the following questions:

a) What output was produced by the script? Explain it.

   Answer: The output should look similar to the following:

```
ORA-06531: Reference to uninitialized collection
ORA-06512: at line 15
```

   Recall that when a varray is declared, it is automatically `NULL`. In other words, the collection itself is `NULL`, not its individual elements. Therefore, before it can be used, it must be initialized via the constructor function with the same name as the varray type. Furthermore, once the collection is initialized, the `EXTEND` method must be used before its individual elements can be referenced in the script.

b) Modify the script so that no errors are returned at the runtime.

   Answer: The script should look similar to the following script. Changes are highlighted in bold.

For Example *ch15_7b.sql*

```
DECLARE
   CURSOR city_cur IS
      SELECT city
        FROM zipcode
       WHERE rownum <= 10;

   TYPE city_type IS VARRAY(10) OF zipcode.city%TYPE;
   city_varray city_type := city_type();

   v_counter INTEGER := 0;
BEGIN
   FOR city_rec IN city_cur
   LOOP
      v_counter := v_counter + 1;
      city_varray.EXTEND;
      city_varray(v_counter) := city_rec.city;
      DBMS_OUTPUT.PUT_LINE('city_varray('||v_counter||'): '||
         city_varray(v_counter));
   END LOOP;
END;
```

When run, this version of the script produces the following output:

```
city_varray(1): New York
city_varray(2): Santurce
city_varray(3): North Adams
city_varray(4): Dorchester
city_varray(5): Tufts Univ. Bedford
city_varray(6): Weymouth
city_varray(7): Sandwich
city_varray(8): Ansonia
city_varray(9): Middlefield
city_varray(10): Oxford
```

c) Modify the script as follows: Double the size of the varray and populate the last ten elements with the first ten elements. In other words, the value of the eleventh element should be equal to the value of the first element; the value of the twelfth element should be equal to the value of the second element; and so forth.

   Answer: The script should look similar to the following script. Newly added statements are shown in bold.

   For Example *ch15_7c.sql*

```
DECLARE
   CURSOR city_cur IS
      SELECT city
        FROM zipcode
       WHERE rownum <= 10;

   TYPE city_type IS VARRAY(20) OF zipcode.city%TYPE;
   city_varray city_type := city_type();

   v_counter INTEGER := 0;
BEGIN
   FOR city_rec IN city_cur
   LOOP
      v_counter := v_counter + 1;
      city_varray.EXTEND;
      city_varray(v_counter) := city_rec.city;
   END LOOP;

   FOR i IN 1..v_counter
   LOOP
      -- extend the size of varray by 1 and copy the current element
      -- to the last element
      city_varray.EXTEND(1, i);
   END LOOP;

   FOR i IN 1..20
   LOOP
      DBMS_OUTPUT.PUT_LINE('city_varray('||i||'): '||
         city_varray(i));
   END LOOP;
END;
```

   In this version of the script, the maximum size of the varray has been increased to 20. Next, the first 10 elements of the varray `city_varray` are populated via cursor `FOR LOOP` just like in the previous versions of the script. After the first 10 elements of the varray are populated, the last ten elements are populated via numeric `FOR LOOP` and the `EXTEND` method as follows:

```
FOR i IN 1..v_counter
LOOP
   -- extend the size of varray by 1 and copy the current element
   -- to the last element
    city_varray.EXTEND(1, i);
END LOOP;
```

In this loop, the loop counter is implicitly incremented by one. So for the first iteration of the loop, the size of the varray is increased by one and the first element of the varray is copied to the eleventh element. In the same manner, the second element of the varray is copied to the twelfth element, and so forth.

Finally, in order to display all elements of the varray, the `DBMS_OUTPUT.PUT_LINE` statement has been moved to its own numeric `FOR` loop that iterates 20 times.

When run, this version of the script produces the following output:

```
city_varray(1): New York
city_varray(2): Santurce
city_varray(3): North Adams
city_varray(4): Dorchester
city_varray(5): Tufts Univ. Bedford
city_varray(6): Weymouth
city_varray(7): Sandwich
city_varray(8): Ansonia
city_varray(9): Middlefield
city_varray(10): Oxford
city_varray(11): New York
city_varray(12): Santurce
city_varray(13): North Adams
city_varray(14): Dorchester
city_varray(15): Tufts Univ. Bedford
city_varray(16): Weymouth
city_varray(17): Sandwich
city_varray(18): Ansonia
city_varray(19): Middlefield
city_varray(20): Oxford
```

Lab 15.3 Multilevel Collections

In this exercise, you will experiment with multilevel collections. Create the following PL/SQL script:

For Example *ch15_8a.sql*

```
DECLARE
    TYPE table_type1 IS TABLE OF INTEGER     INDEX BY PLS_INTEGER;
    TYPE table_type2 IS TABLE OF TABLE_TYPE1 INDEX BY PLS_INTEGER;

   table_tab1 table_type1;
   table_tab2 table_type2;
BEGIN
   FOR i IN 1..2
   LOOP
      FOR j IN 1..3
      LOOP
         IF i = 1
         THEN
            table_tab1(j) := j;
         ELSE
            table_tab1(j) := 4 - j;
         END IF;
         table_tab2(i)(j) := table_tab1(j);
         DBMS_OUTPUT.PUT_LINE ('table_tab2('||i||')('||j||'): '||table_tab2(i)(j));
      END LOOP;
   END LOOP;
END;
```

Execute the script, and then answer the following questions:

a) Execute the script above and explain the output produced.

Answer: The output should look similar to the following:

```
table_tab2(1)(1): 1
table_tab2(1)(2): 2
table_tab2(1)(3): 3
table_tab2(2)(1): 3
table_tab2(2)(2): 2
table_tab2(2)(3): 1
```

This script uses multilevel associative arrays or an associative array of associative arrays. The declaration portion of the script defines a multilevel associative array `table_tab2`. Each row of this table is an associative array consisting of multiple rows.

The executable portion of the script populates the multilevel table via nested numeric `FOR LOOP`. In the first iteration of the outer loop, the inner loop populates the associative array `table_tab1` with values 1, 2, 3, and the first row of the multilevel table `table_tab2`. In the second iteration of the outer loop, the inner loop populates the associative array `table_tab1` with values 3, 2, 1, and the second row of the multilevel table `table_tab2`.

b) Modify the script so that instead of using multilevel associative arrays it uses a nested table of associative arrays.

Answer: The new version of the script should look similar to the following. Affected statements are highlighted in bold.

For Example *ch15_8b.sql*

```
DECLARE
   TYPE table_type1 IS TABLE OF INTEGER INDEX BY PLS_INTEGER;
   TYPE table_type2 IS TABLE OF TABLE_TYPE1;

   table_tab1 table_type1;
   table_tab2 table_type2 := table_type2();
BEGIN
   FOR i IN 1..2
   LOOP
      table_tab2.EXTEND;
      FOR j IN 1..3 LOOP
         IF i = 1
         THEN
            table_tab1(j) := j;
         ELSE
            table_tab1(j) := 4 - j;
         END IF;
         table_tab2(i)(j) := table_tab1(j);
         DBMS_OUTPUT.PUT_LINE ('table_tab2('||i||')('||j||'): '||table_tab2(i)(j));
      END LOOP;
   END LOOP;
```

```
END;
```

In this version of the script, the `table_type2` is declared as a nested table of associative arrays. Next, `table_tab2` is initialized prior to its use, and its size is extended before a new element is assigned a value.

c) Modify the script so that instead of using multilevel associative arrays it uses a nested table of varrays.

Answer: The script should look similar to the following script. Modifications are shown in bold.

For Example *ch15_8c.sql*

```
DECLARE
   TYPE table_type1 IS VARRAY(3) OF PLS_INTEGER;
   TYPE table_type2 IS TABLE     OF TABLE_TYPE1;

   table_tab1 table_type1 := table_type1();
   table_tab2 table_type2 := table_type2(table_tab1);
BEGIN
   FOR i IN 1..2
   LOOP
      table_tab2.EXTEND;
      table_tab2(i) := table_type1();
      FOR j IN 1..3
      LOOP
         IF i = 1
         THEN
            table_tab1.EXTEND;
            table_tab1(j) := j;
         ELSE
            table_tab1(j) := 4 - j;
         END IF;
         table_tab2(i).EXTEND;
         table_tab2(i)(j):= table_tab1(j);
         DBMS_OUTPUT.PUT_LINE ('table_tab2('||i||')('||j||'): '||table_tab2(i)(j));
      END LOOP;
   END LOOP;
END;
```

In the declaration portion of the script, the `table_type1` is defined as a varray with maximum of three integer elements, and the `table_type2` is declared as a nested table of varrays. Next, `table_tab1` and `table_tab2` are initialized prior to their uses.
In the executable portion of the script, the size of the `table_tab2` is incremented via the `EXTEND` method and its individual elements are initialized as follows:

```
table_tab2(i) := table_type1();
```

Notice that that each element is initialized via the constructor associated with the varray type `table_type1`. Furthermore, in order to populate a nested table, a new varray element must be added to the each nested table element as shown:

```
table_tab2(i).EXTEND;
```

Without this statement, the script causes the following error:

```
ORA-06533: Subscript beyond count
ORA-06512: at line 21
```

When run, this version of the script produces output identical to the original example:

```
table_tab2(1)(1): 1
table_tab2(1)(2): 2
table_tab2(1)(3): 3
table_tab2(2)(1): 3
table_tab2(2)(2): 2
table_tab2(2)(3): 1
```

Try It Yourself

The projects in this section are meant to have you use all of the skills that you have acquired throughout this chapter. Here are some exercises that will help you test the depth of your understanding.

1) Create the following script. Create an associative array and populate it with the instructor's full name. In other words, each row of the associative array should contain first and last names. Display this information on the screen.

   Answer: The script should look similar to the following:

   For Example *ch15_9a.sql*

```
DECLARE
   CURSOR name_cur IS
      SELECT first_name||' '||last_name name
        FROM instructor;

   TYPE name_type IS TABLE OF VARCHAR2(50) INDEX BY PLS_INTEGER;
   name_tab name_type;

   v_counter INTEGER := 0;
BEGIN
   FOR name_rec IN name_cur
   LOOP
      v_counter := v_counter + 1;
      name_tab(v_counter) := name_rec.name;

      DBMS_OUTPUT.PUT_LINE ('name('||v_counter||'): '||name_tab(v_counter));
   END LOOP;
END;
```

   In the preceding example, the associative array `name_tab` is populated with instructor full names. Notice that the variable `v_counter` is used as a subscript to reference individual array elements. This example produces the following output:

```
name(1): Fernand Hanks
name(2): Tom Wojick
name(3): Nina Schorin
name(4): Gary Pertez
name(5): Anita Morris
name(6): Todd Smythe
name(7): Marilyn Frantzen
name(8): Charles Lowry
name(9): Rick Chow
name(10): Irene Willig
```

2) Modify the script created in the previous exercise (step 1 above). Instead of using an associative array, use a varray.

   Answer: The script should look similar to the following. Affected statements are highlighted in bold.

   For Example *ch15_9b.sql*

```
DECLARE
   CURSOR name_cur IS
      SELECT first_name||' '||last_name name
        FROM instructor;

   TYPE name_type IS VARRAY(15) OF VARCHAR2(50);
   name_varray name_type := name_type();

   v_counter INTEGER := 0;
BEGIN
   FOR name_rec IN name_cur
   LOOP
      v_counter := v_counter + 1;
      name_varray.EXTEND;
      name_varray(v_counter) := name_rec.name;

      DBMS_OUTPUT.PUT_LINE ('name('||v_counter||'): '||name_varray(v_counter));
   END LOOP;
END;
```

   In this version of the script, you define a varray of 15 elements. It is important to remember to initialize the array before referencing its individual elements. In addition, the array must be extended before new elements are added to it.

3) Modify the script created in the previous exercise (step 2 above). Create an additional varray and populate it with unique course numbers that each instructor teaches. Display instructor's name and the list of courses he or she teaches.

   Answer: The script should look similar to the following:

   For Example *ch15_10a.sql*

```
DECLARE
   CURSOR instructor_cur IS
      SELECT instructor_id, first_name||' '||last_name name
        FROM instructor;

   CURSOR course_cur (p_instructor_id NUMBER) IS
      SELECT unique course_no course
```

```
         FROM section
        WHERE instructor_id = p_instructor_id;

   TYPE name_type IS VARRAY(15) OF VARCHAR2(50);
   name_varray name_type := name_type();

   TYPE course_type IS VARRAY(10) OF NUMBER;
   course_varray course_type;

   v_counter1 INTEGER := 0;
   v_counter2 INTEGER;
BEGIN
   FOR instructor_rec IN instructor_cur
   LOOP
      v_counter1 := v_counter1 + 1;
      name_varray.EXTEND;
      name_varray(v_counter1) := instructor_rec.name;

      DBMS_OUTPUT.PUT_LINE ('name('||v_counter1||'): '||name_varray(v_counter1));

      -- Initialize and populate course_varray
      v_counter2 := 0;
      course_varray := course_type();
      FOR course_rec in course_cur (instructor_rec.instructor_id)
      LOOP
         v_counter2 := v_counter2 + 1;
         course_varray.EXTEND;
         course_varray(v_counter2) := course_rec.course;

         DBMS_OUTPUT.PUT_LINE ('course('||v_counter2||'): '||
            course_varray(v_counter2));
      END LOOP;
      DBMS_OUTPUT.PUT_LINE ('===========================');
   END LOOP;
END;
```

Consider the script just created. First, you declare two cursors, `instructor_cur` and `course_cur`. The `course_cur` accepts a parameter because it returns a list of courses taught by a particular instructor. Notice that the `SELECT` statement uses function `UNIQUE` to retrieve distinct course numbers. Second, you declare two varray types and variables, `name_varray` and `course_varray`. Notice that you do not initialize the second varray at the time of declaration. Next, you declare two counters and initialize the first counter only.

In the body of the block, you open `instructor_cur` and populate `name_varray` with its first element. Next, you initialize the second counter and `course_varray`. This step is necessary because you need to repopulate `course_varray` for the next instructor. Finally, you open `course_cur` to retrieve corresponding courses and display them on the screen.

When run, the script produces the following output:

```
name(1): Fernand Hanks
course(1): 25
course(2): 450
course(3): 134
course(4): 120
course(5): 240
course(6): 125
course(7): 140
course(8): 146
course(9): 122
===========================
name(2): Tom Wojick
course(1): 25
course(2): 100
course(3): 134
course(4): 120
course(5): 240
course(6): 125
course(7): 140
course(8): 146
course(9): 124
course(10): 10
===========================
name(3): Nina Schorin
course(1): 25
course(2): 310
course(3): 100
course(4): 147
course(5): 134
course(6): 120
course(7): 20
course(8): 130
course(9): 142
course(10): 124
===========================
name(4): Gary Pertez
course(1): 25
course(2): 100
course(3): 330
course(4): 120
course(5): 20
course(6): 135
course(7): 130
course(8): 142
course(9): 124
course(10): 204
===========================
name(5): Anita Morris
course(1): 25
course(2): 100
course(3): 20
course(4): 210
course(5): 350
course(6): 135
course(7): 130
course(8): 142
course(9): 124
course(10): 122
===========================
name(6): Todd Smythe
course(1): 25
course(2): 100
course(3): 144
course(4): 20
course(5): 220
course(6): 350
course(7): 135
course(8): 125
course(9): 130
course(10): 122
===========================
name(7): Marilyn Frantzen
course(1): 25
course(2): 120
course(3): 132
course(4): 230
course(5): 350
course(6): 135
course(7): 125
course(8): 145
course(9): 122
===========================
name(8): Charles Lowry
course(1): 25
course(2): 120
course(3): 132
course(4): 230
course(5): 125
course(6): 140
course(7): 420
course(8): 145
course(9): 122
===========================
name(9): Rick Chow
===========================
name(10): Irene Willig
===========================
```

As mentioned earlier, it is important to reinitialize the variable `v_counter2` that is used to reference individual elements of `course_varray`. When this step is omitted and the variable is initialized only once at the time declaration, the script generates the following runtime error:

```
ORA-06533: Subscript beyond count
ORA-06512: at line 33
```

4) Find and explain errors in the following script:

For Example *ch15_11a.sql*

```
DECLARE
   TYPE varray_type1 IS VARRAY(7) OF INTEGER;
   TYPE table_type2  IS TABLE OF varray_type1 INDEX BY PLS_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;
```

Answer: Consider the error generated by the preceding script:

```
ORA-06550: line 6, column 26:
PLS-00222: no function with name 'TABLE_TYPE2' exists in this scope
ORA-06550: line 6, column 11:
PL/SQL: Item ignored
ORA-06550: line 9, column 44:
PLS-00320: the declaration of the type of this expression is incomplete or
malformed
ORA-06550: line 9, column 4:
PL/SQL: Statement ignored
```

Notice that this error refers to the initialization of the collection variable `table2`, which has been declared as an associative array of varrays. You will recall that associative arrays are not initialized prior to their use. As a result, the declaration of `table2` must be modified. Furthermore, additional assignment statement must be added to the executable portion of the script as follows:

For Example *ch15_11b.sql*

```
DECLARE
   TYPE varray_type1 IS VARRAY(7) OF INTEGER;
   TYPE table_type2  IS TABLE OF varray_type1 INDEX BY PLS_INTEGER;

   varray1 varray_type1 := varray_type1(1, 2, 3);
   table2  table_type2;
BEGIN
   -- These statements populate associative array
   table2(1) := varray1;
   table2(2) := varray_type1(8, 9, 0);

   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;
```

When run, this version of the script produces a different error:

```
ORA-06532: Subscript outside of limit
ORA-06512: at line 16
```

Notice that this is a runtime error that refers to the collection variable `varray1`. This error occurs because varray is extended beyond its limit. `Varray1` can contain up to seven integers. After initialization, the varray contains three integers. As a result, it can be populated with no more than four additional integer numbers. When the fifth iteration of the loop tries to extend the varray to eighth element, it causes causes a subscript beyond count error.

It is important to note that there is no correlation between the loop counter and the `EXTEND` method. Every time the `EXTEND` method is called, it increases the size of the varray by one element. Since the varray has been initialized to three elements, the `EXTEND` method adds a fourth element to the array for the first iteration of the loop. At this same time, the first element of the varray is assigned a value of 1 via the loop counter. For the second iteration of the loop, the `EXTEND` method adds a fifth element to the varray while the second element is assigned a value of 2, and so forth.

Finally, consider the error-free version of the script and its output:

For Example *ch15_11c.sql*

```
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;
BEGIN
   -- These statements populate associative array
   table2(1) := varray1;
   table2(2) := varray_type1(8, 9, 0);

   DBMS_OUTPUT.PUT_LINE ('table2(1)(2): '||table2(1)(2));

   FOR i IN 4..7
   LOOP
      varray1.EXTEND;
      varray1(i) := i;
   END LOOP;

   -- Display elements of the varray
   FOR i IN 1..7
   LOOP
      DBMS_OUTPUT.PUT_LINE ('varray1('||i||'): '||varray1(i));
   END LOOP;
END;
```

The script output should look similar to the following:

```
table2(1)(2): 2
varray1(1): 1
varray1(2): 2
varray1(3): 3
varray1(4): 4
varray1(5): 5
```

```
varray1(6): 6
varray1(7): 7
```