Table of Contents

[CHAPTER 8: ADVANCED COMPOSITE DATA TYPES 2](#_Toc416701832)

[Theory 2](#_Toc416701833)

[AIM 4](#_Toc416701834)

[Lab Exercise 8: ADVANCED COMPOSITE DATA TYPES 5](#_Toc416701835)

[1. Loop Through Collection 6](#_Toc416701836)

[2. Using Set Operations 20](#_Toc416701837)

[3. Record of Record 28](#_Toc416701838)

[4. Record of Array 32](#_Toc416701839)

[5. Multi-Dimensions Collection 34](#_Toc416701840)

[SUMMARY 40](#_Toc416701841)

[REFERENCES 41](#_Toc416701842)

[INDEX 42](#_Toc416701843)

# CHAPTER 8: ADVANCED COMPOSITE DATA TYPES

## Theory

The most important note in the previous chapter is to recognize when to use each kind of collection or when to use a record data type. In this chapter, you may extend you understanding of VARRAY internal features in comparison to Nested Table. You will also explore the advanced use of composite data types such as multi-dimension collection and a combination of records and collections.

Nested Table is somehow similar to VARRAY; the only difference you can see so far that a Nested table has an unlimited number of elements while the size of VARRAY must be specified in a declaration section. However, there are other important differences that you must bear in your mind: (1) Set operations are only applicable for Nested Table variable and (2) VARRAY is always dense while Nested Table may become spare. Please see the following figure:

Nested Table

(6)

(5)

(4)

(2)

(1)



VARRAY

(6)

(5)

(4)

(3)

(2)

(1)

(1)

(2)

(3)

(4)

(5)

MAX



(3)

MAX

(1)

(2)

(4)

(5)

(6)

After deleting the element (3), VARRAY rearranges the index to keep the array dense while Nested Table keep the index as it was. For this reason, you may use an integer increased by 1 to refer to the next item in VARRAY but you can't do that in Nested Table. Instead, you must use NEXT or PRIOR collection attributes to the next or previous element.

Now you may wonder if there is any other collection attributes that would help managing collection. The answer is YES. Please see the following figure to display some of these attributes:

VARRAY

**LIMIT**

**LAST**

**EXISTS(6) = FALSE**

**PRIOR(4)**

**NEXT(4)**

**FIRST**



Collection and record can be combined to form more complex data type. For example, you may create VARRAY of VARRAY to emulate multi-dimension array or you may create a Nested Table of Record to emulate a complete table (ROWS and COLUMNS). It is important to distinguish between how to access a record (using "." notation) and how to access a collection (using "(INDEX)" syntax). Thus, it is difference on how access a Record of Collection than to access a Collection of Record.

**Draw.DOB(4)**

Record of VARRAYs

Draw



Cap





Prof

**Elm(2)(4)**

Multi-Dimension VARRAY



(3)

(1)



(2)

(4)

(3)

(2)

(1)

(5)

## AIM

The AIM of the following exercise is to demonstrate the advanced using of composite data types (Record and Collection).

The steps involved will include:

* Loop Through Collection
* Using Set Operations
* Record of Record
* Record of Array
* Multi-Dimensions Collection

In general, lab exercises are done in sequential order. Thus, it is assumed that you successfully completed the previous labs. However, not all previous labs are required. Please be sure to run the following lab before proceeding:

* Installing Oracle Database 12c.
* Composite Data Types

Estimated Completion Time:

30 minutes

# Lab Exercise 8: ADVANCED COMPOSITE DATA TYPES

|  |
| --- |
|  |

## Loop Through Collection

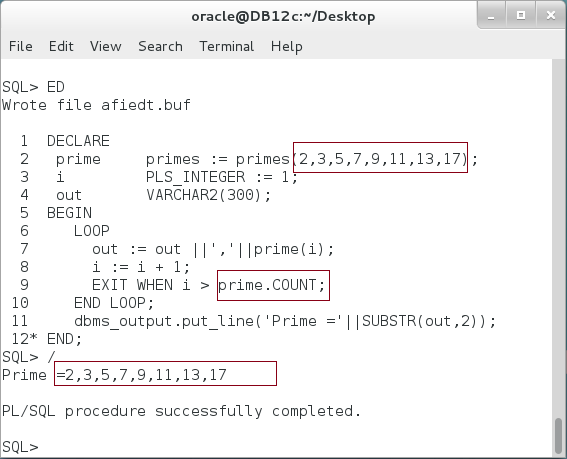
**Step 1:** Open the Terminal, open SQL\*Plus console and connect to hr schema.

|  |  |
| --- | --- |
| Command | Description |
| sqlplus | Open SQL\*Plus console. |
| hr/oracle | connect to **hr** schema. |

****

**Step 2:** Execute the following block:

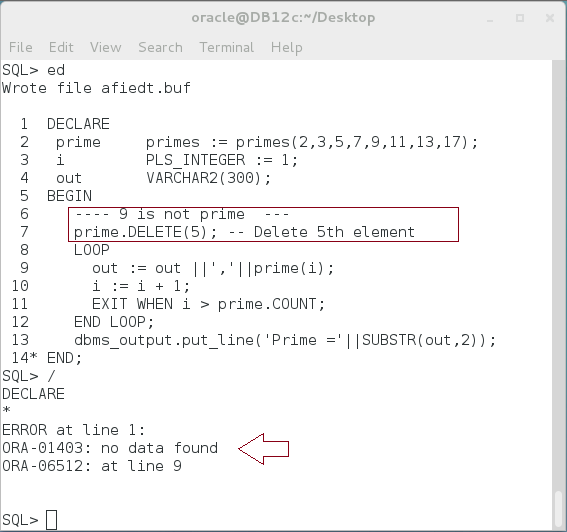
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); | primes is a Nested Table defined in the previous chapter. |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| LOOP | **Normal LOOP** with i+1 syntax. The loop stop when i is greater than **COUNT attribute**: the number of elements in the nested table. |
| out := out ||','||prime(i); |
| i := i + 1; |
| EXIT WHEN i > prime.**COUNT;** |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
|  |
| END; |  |
| / |  |

****

**Please note**: The output order matches the initial order.

**Step 3:** Modify the previous block as the following:

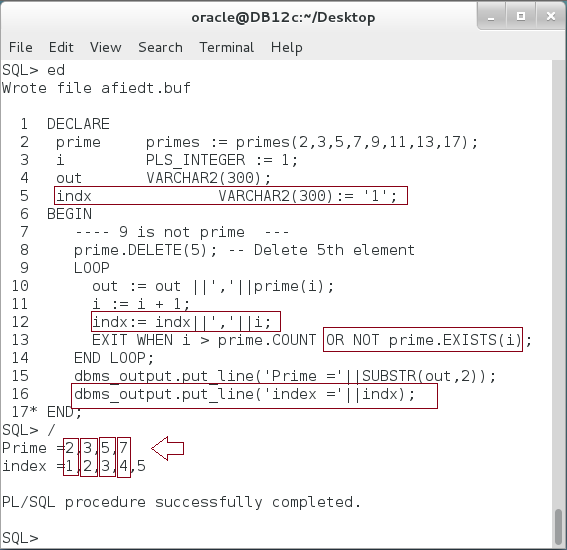
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| ---- 9 is not prime --- | **Delete** un related element: 9. |
| **prime.DELETE(5);** -- Delete 5th element |
| LOOP | **Try to loop through a nested table.** |
| out := out ||','||prime(i); |
| i := i + 1; |
| EXIT WHEN i > prime.COUNT; |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |



**Please note:** The previous block raised an exception of "no data found". **Can you explain the result?**

**Step 4:** Change the previous block again as shown below:

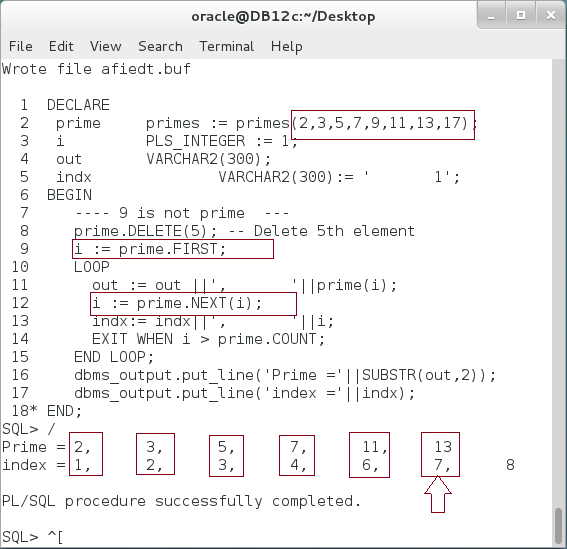
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |
| out VARCHAR2(300); |  |
| indx VARCHAR2(300):= '1'; |  |
| BEGIN |
| ---- 9 is not prime --- |  |
| prime.DELETE(5); -- Delete 5th element |
| LOOP |  |
| out := out ||','||prime(i); |  |
| i := i + 1; | **Stop the loop when the element is not existing.** |
| indx:= indx||','||i; |
| EXIT WHEN i > prime.COUNT **OR NOT prime.EXISTS(i);** |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| dbms\_output.put\_line('index ='||indx); |  |
| END; |  |
| / |  |

****

**Please note:** Nested Table is sparse collection. You should not use "i+1" syntax to loop through its elements. Instead, you must use "NEXT" or "PRIOR" attributes.

**Step 5:** Modify a previous block as the following:

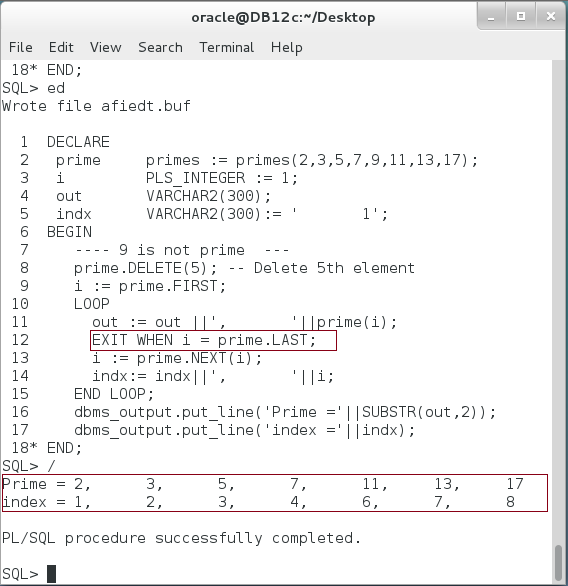
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| indx VARCHAR2(300):= ' 1'; |  |
| BEGIN |  |
| ---- 9 is not prime --- |  |
| prime.DELETE(5); -- Delete 5th element |  |
| **i := prime.FIRST;** | Use **FIRST** and **NEXT** attributes to navigate through collection. |
| LOOP |
| out := out ||', '||prime(i); |
| **i := prime.NEXT(i);** |
| indx:= indx||', '||i; |
| EXIT WHEN i > prime.COUNT; |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| dbms\_output.put\_line('index ='||indx); |  |
| END; |  |
| / |  |



**Ops**: The loop didn't raise un exception but it didn't finish yet. The last element (17) didn't show in the result. **Can you explain what happened?**   
COUNT attribute returns 7: (Total number elements 8 - Deleted elements 1 = 7). **How can we fix this?**

**Step 6:** Modify the previous block as shown below:

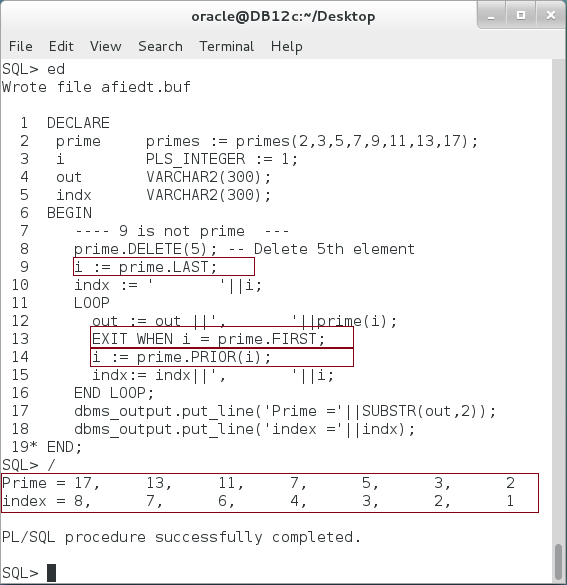
|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| indx VARCHAR2(300):= ' 1'; |  |
| BEGIN |  |
| ---- 9 is not prime --- |  |
| prime.DELETE(5); -- Delete 5th element |  |
| **i := prime.FIRST;** | Use **FIRST**, **NEXT** and **LAST** attributes to navigate through collection. |
| LOOP |
| out := out ||', '||prime(i); |
| **EXIT WHEN i = prime.LAST;** |
| **i := prime.NEXT(i);** |
| indx:= indx||', '||i; |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| dbms\_output.put\_line('index ='||indx); |  |
| END; |  |
| / |  |

****

**Please note:** You must place the condition before executing NEXT. If you try to execute NEXT outside the limit of the collection, a pre-define exception will be raised.

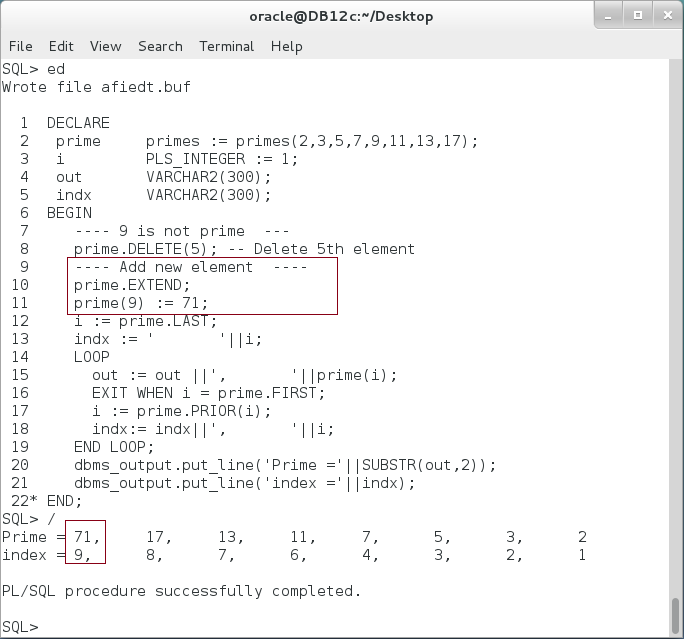
**Step 7:** You may want to navigate through a collection in reverse. In this case, use PRIOR instead of NEXT as shown below:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| indx VARCHAR2(300; |  |
| BEGIN |  |
| ---- 9 is not prime --- |  |
| prime.DELETE(5); -- Delete 5th element |  |
| **i := prime.LAST;** | Use **LAST**, **PRIOR** and **FIRST** attributes to navigate through collection in reverse. |
| **indx := ' '||i;** |
| LOOP |
| out := out ||', '||prime(i); |
| **EXIT WHEN i = prime.FIRST;** |
| **i := prime.PRIOR(i);** |
| indx:= indx||', '||i; |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| dbms\_output.put\_line('index ='||indx); |  |
| END; |  |
| / |  |



**Step 8:** Nested Table can be extended. The element, however, has to be initialized first. One option is use a constructor. Another option is use EXTEND function which extends the Nested Table with one additional element. Execute the following block:

|  |  |
| --- | --- |
| Command | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,9,11,13,17); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| indx VARCHAR2(300; |  |
| BEGIN |  |
| ---- 9 is not prime --- |  |
| prime.DELETE(5); -- Delete 5th element |  |
| ---- Add new element ---- | Add and initialize new element. |
| **prime.EXTEND;** |
| **prime(9) := 71;** |
| i := prime.LAST; |  |
| indx := ' '||i; |  |
| LOOP |  |
| out := out ||', '||prime(i); |  |
| EXIT WHEN i = prime.FIRST; |  |
| i := prime.PRIOR(i); |  |
| indx:= indx||', '||i; |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| dbms\_output.put\_line('index ='||indx); |  |
| END; |  |
| / |  |



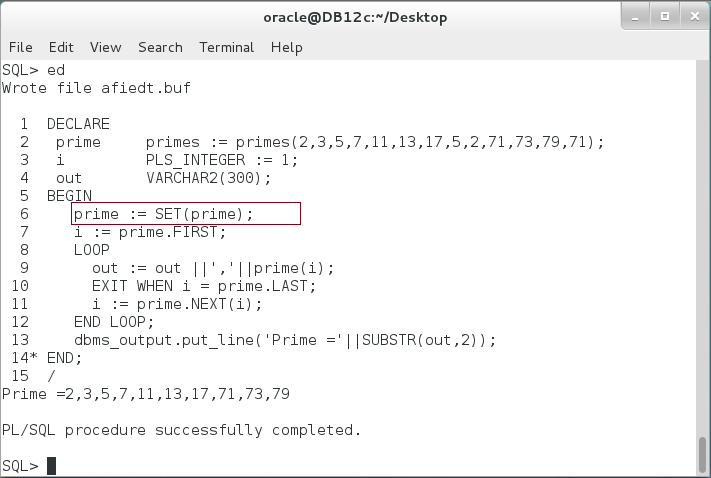
## Using Set Operations

**Step 1:** Set operations (SET, MULTISET) are exclusively applicable to Nested Tables. Before proceeding, please take a look on the table below:

|  |  |
| --- | --- |
| Operation | Description |
| **SET** | Converts a nested table into a set by eliminating duplicates. |
| **MULTISET EXCEPT** | Takes as arguments two nested tables and returns a nested table whose elements are in the first nested table but not in the second nested table. |
| **MULTISET EXCEPT DISTINCT** | Eliminate any element in nested\_table1 which is also in nested\_table2, regardless of the number of occurrences. |
| **MULTISET INTERSECT** | Takes as arguments two nested tables and returns a nested table whose values are common in the two input nested tables. |
| **MULTISET INTERSECT DISTINCT** | Eliminates duplicates from the returned nested table, including duplicates of NULL, if they exist |
| **MULTISET UNION** | Takes as arguments two nested tables and returns a nested table whose values are those of the two input nested tables. |
| **MULTISET UNION DISTINCT** | Instructs Oracle to eliminate duplicates from the returned nested table, including duplicates of NULL, if they exist. |

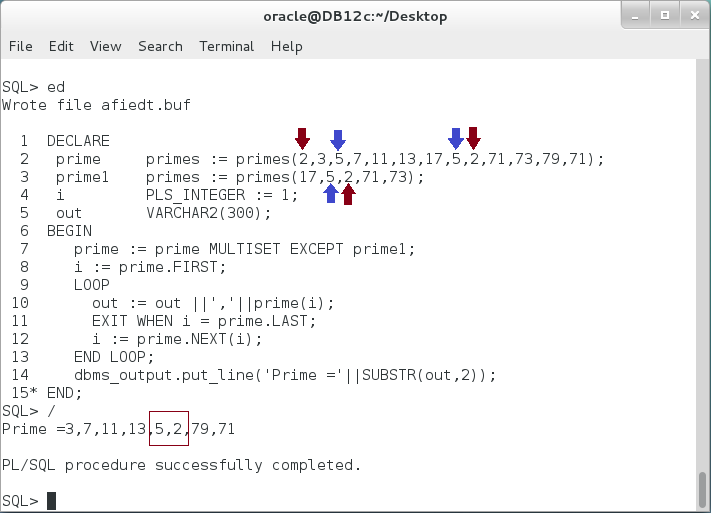
**Step 2:** Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); | There are prime numbers occurring multiple times. |
| i PLS\_INTEGER := 1; |
| out VARCHAR2(300); |
| BEGIN |
| **prime := SET(prime);** | Use SET operator to remove duplication. |
| i := prime.FIRST; |  |
| LOOP |  |
| out := out ||','||prime(i); | Print prime content. |
| EXIT WHEN i = prime.LAST; |
| i := prime.NEXT(i); |
| END LOOP; |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |



**Step 3:** Modify the previous block as show below:

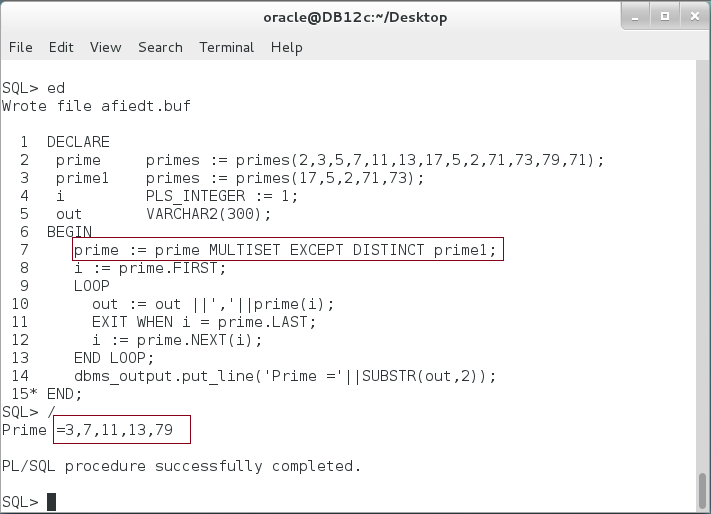
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); |  |
| **prime1 primes := primes(17,5,2,71,73);** |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| **prime := prime MULTISET EXCEPT prime1;** | Returns elements in "**prime**" but not in "**prime1**" |
| i := prime.FIRST; |
| LOOP |
| out := out ||','||prime(i); |  |
| EXIT WHEN i = prime.LAST; |  |
| i := prime.NEXT(i); |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |

****

**Please note:** Although MULTISET EXCEPT should return elements which in "prime" but not in "prime1", it returns two elements (2,5) which are existing in "prime1"! **WHY?**

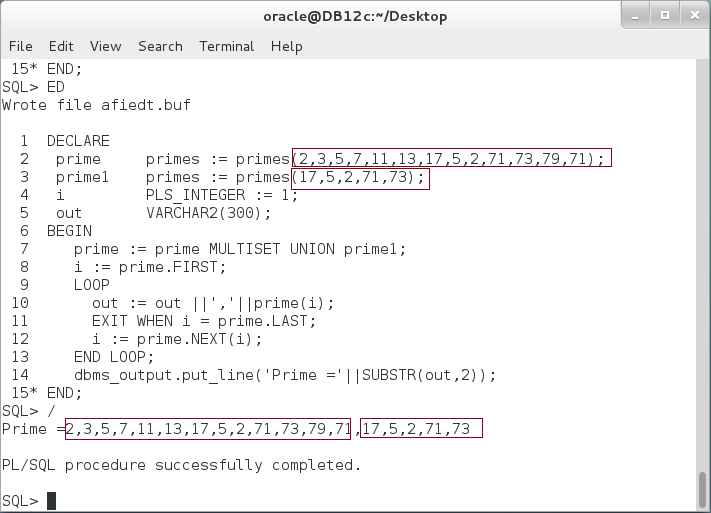
**Step 4:** Modify the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); |  |
| prime1 primes := primes(17,5,2,71,73); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| **prime := prime MULTISET EXCEPT DISTINCT prime1;** | Use **DISTINCT** keyword to enforce EXCEPT operation no matter how many the elements occur in "prime" table. |
| i := prime.FIRST; |
| LOOP |
| out := out ||','||prime(i); |
| EXIT WHEN i = prime.LAST; |
| i := prime.NEXT(i); |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |

****

**Step 5:** Modify the previous block as shown below:

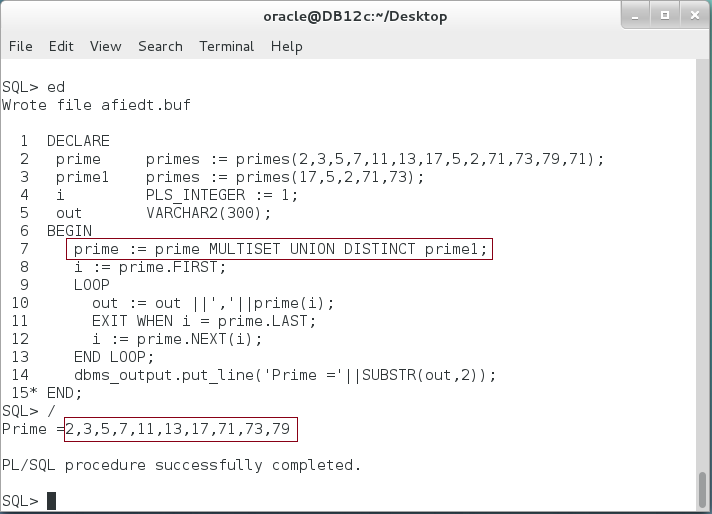
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); |  |
| prime1 primes := primes(17,5,2,71,73); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| **prime := prime MULTISET UNION prime1;** | Use **UNION** keyword to combine elements from two nested tables into one nested table. |
| i := prime.FIRST; |
| LOOP |
| out := out ||','||prime(i); |
| EXIT WHEN i = prime.LAST; |
| i := prime.NEXT(i); |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |

****

**Please note:** UNION keyword just append the second table to the first table even if the elements in the second table already existed in the first.

**Step 6:** Modify the previous block as shown below:

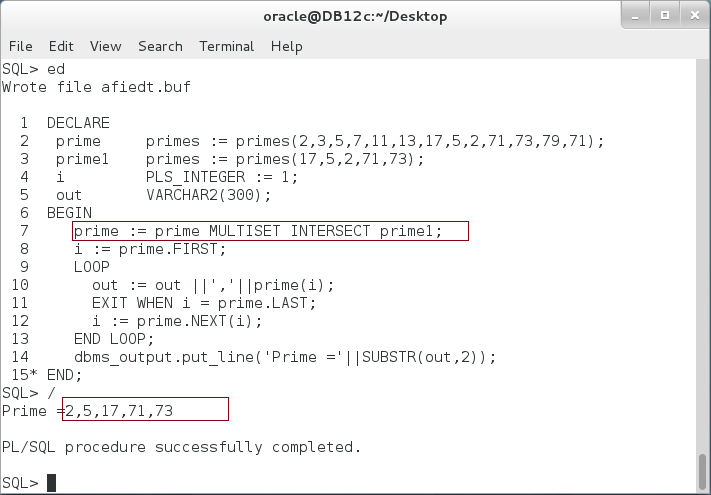
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); |  |
| prime1 primes := primes(17,5,2,71,73); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| **prime := prime MULTISET UNION DISTINCT prime1;** | Use **DISTINCT** keyword to enforce UNION operation by eliminating redundant elements. |
| i := prime.FIRST; |
| LOOP |
| out := out ||','||prime(i); |
| EXIT WHEN i = prime.LAST; |
| i := prime.NEXT(i); |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |

****

**Please note:** DISTINCT keyword removes not only the redundancy between the nested tables, but also within each nested tables. **Do you think it has something in common with SET function?**

**Step 7:** Modify the previous block as shown below:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| prime primes := primes(2,3,5,7,11,13,17,5,2,71,73,79,71); |  |
| prime1 primes := primes(17,5,2,71,73); |  |
| i PLS\_INTEGER := 1; |  |
| out VARCHAR2(300); |  |
| BEGIN |  |
| **prime := prime MULTISET INTERSECT prime1;** | Find the elements in common between two nested tables. |
| i := prime.FIRST; |
| LOOP |
| out := out ||','||prime(i); |
| EXIT WHEN i = prime.LAST; |
| i := prime.NEXT(i); |  |
| END LOOP; |  |
| dbms\_output.put\_line('Prime ='||SUBSTR(out,2)); |  |
| END; |  |
| / |  |

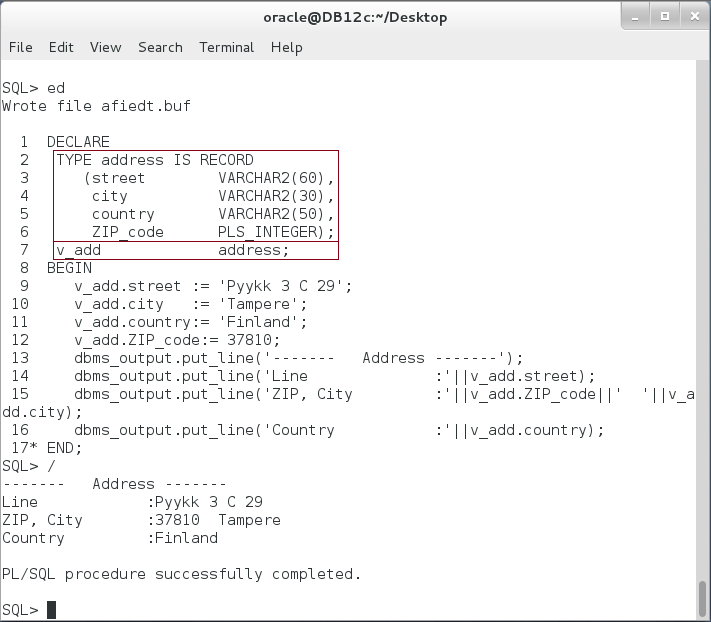
****

**Please note:** INTERSECT function returns the element in common between "prime" and "prime1" nested tables. **Do you think the result will change if you use INTERSECT DISTINCT? Explain?**

## Record of Record

**Step 1:** The record is a composite data type which consists of one or more fields. The fields may consist of scalar data type as well as other composite data types. In this block, you will create a simple record type "address" which consists of scalar data types. Execute the following block:

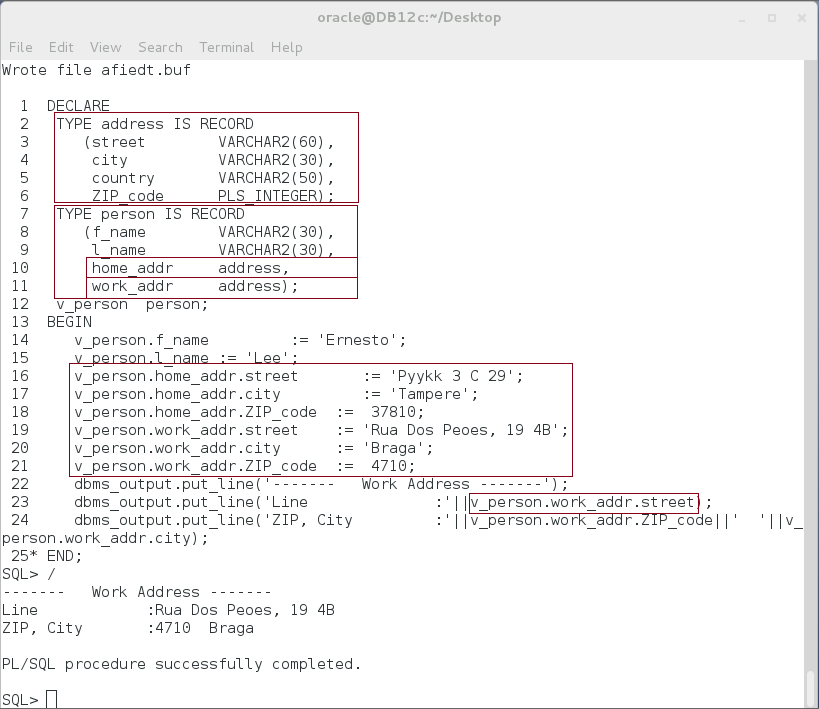
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **TYPE address IS RECORD** | Declare a User-Defined Record. |
| (street VARCHAR2(60), |
| city VARCHAR2(30), |
| country VARCHAR2(50), |
| ZIP\_code PLS\_INTEGER); |
| v\_add address; | Declare variable |
| BEGIN |  |
| v\_add.street := 'Pyykk 3 C 29'; |  |
| v\_add.city := 'Tampere'; |  |
| v\_add.country:= 'Finland'; |  |
| v\_add.ZIP\_code:= 37810; |  |
| dbms\_output.put\_line('------- Address -------'); |  |
| dbms\_output.put\_line('Line:'||v\_add.street); |  |
| dbms\_output.put\_line('ZIP, City :'||v\_add.ZIP\_code||' '||v\_add.city); |  |
| dbms\_output.put\_line('Country:'||v\_add.country); |  |
| END; |  |
| / |  |

****

**Please note:** This is a simple Record variable that contains scalar variables.

**Step 2:** Extend the previous block as the following:

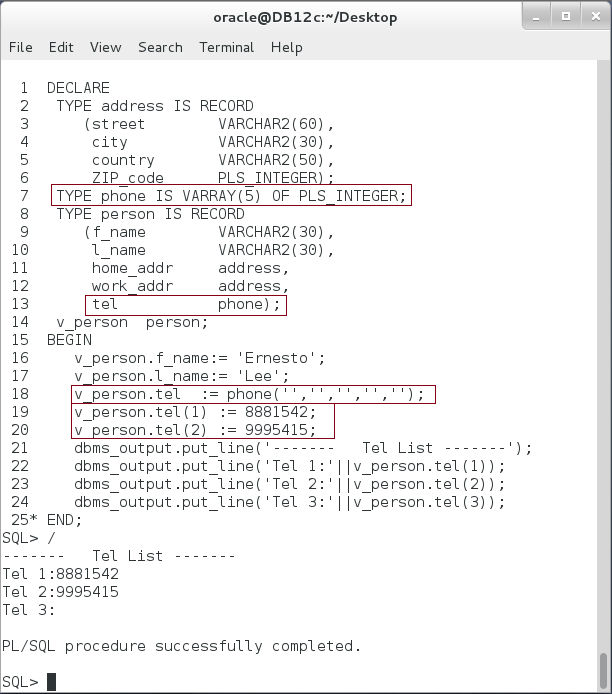
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **TYPE address IS RECORD** | Declare the inner Record "address". |
| (street VARCHAR2(60), |
| city VARCHAR2(30), |
| country VARCHAR2(50), |
| ZIP\_code PLS\_INTEGER); |
| **TYPE person IS RECORD** | Declare the outer Record "person". |
| (f\_name VARCHAR2(30), |
| l\_name VARCHAR2(30), |
| **home\_addr address**, |
| **work\_addr address**); |
| v\_person person; | Declare a variable |
| BEGIN |  |
| v\_person.f\_name := 'Ernesto'; | Note how you set the inner record fields. |
| v\_person.l\_name := 'Lee'; |
| **v\_person.home\_addr.street** := 'Pyykk 3 C 29'; |
| v\_person.home\_addr.city := 'Tampere'; |
| v\_person.home\_addr.ZIP\_code := 37810; |
| v\_person.work\_addr.street := 'Rua Dos Peoes, 19 4B'; |
| v\_person.work\_addr.city := 'Braga'; |
| v\_person.work\_addr.ZIP\_code := 4710; |
| dbms\_output.put\_line('------- Work Address -------'); |  |
| dbms\_output.put\_line('Line :'||v\_person.work\_addr.street); | Call the inner fields' variable. |
| dbms\_output.put\_line('ZIP, City :'||v\_person.work\_addr.ZIP\_code||' '||v\_person.work\_addr.city); |
| END; |  |
| / |  |



## Record of Array

**Step 1:** Extend the previous block to include an array field as the following:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE address IS RECORD |  |
| (street VARCHAR2(60), |  |
| city VARCHAR2(30), |  |
| country VARCHAR2(50), |  |
| ZIP\_code PLS\_INTEGER); |  |
| **TYPE phone IS VARRAY(5) OF PLS\_INTEGER;** | Define VARRAY phone |
| TYPE person IS RECORD |  |
| (f\_name VARCHAR2(30), |  |
| l\_name VARCHAR2(30), |  |
| home\_addr address, |  |
| work\_addr address, |  |
| **tel phone);** | **Use "phone" as field** |
| v\_person person; |  |
| BEGIN |  |
| v\_person.f\_name:= 'Ernesto'; |  |
| v\_person.l\_name:= 'Lee'; |  |
| **v\_person.tel := phone('','','','','');** | Initialize, set value, and call array value. |
| **v\_person.tel(1) := 8881542;** |
| **v\_person.tel(2) := 9995415;** |
| dbms\_output.put\_line('------- Tel List -------'); |  |
| dbms\_output.put\_line('Tel 1:'||v\_person.tel(1)); |  |
| dbms\_output.put\_line('Tel 2:'||v\_person.tel(2)); |  |
| dbms\_output.put\_line('Tel 3:'||v\_person.tel(3)); |  |
| END; |  |
| / |  |

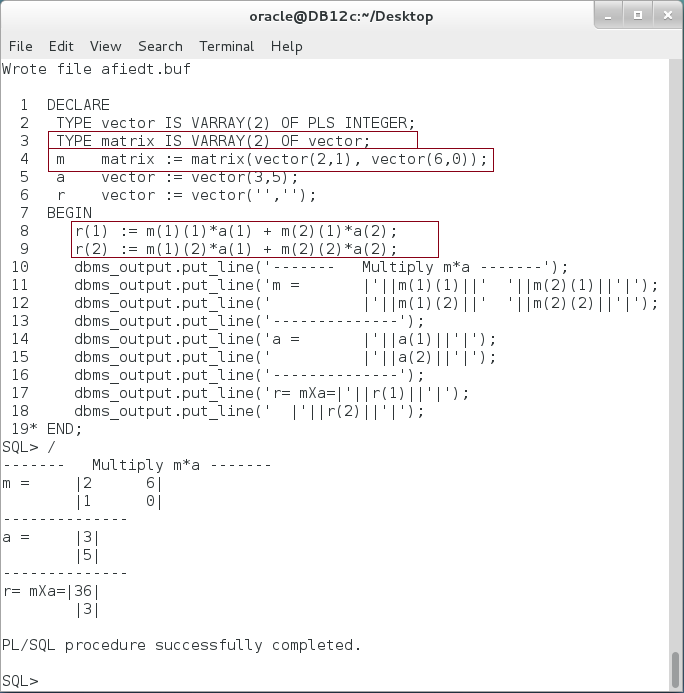


**Please note:** A Record is a flexible data type which may contain any scalar data type, other Record, VARRAY and Nested Tables. It is important to distinguish between how to access the inner record and how to access the inner collection.

## Multi-Dimensions Collection

**Step 1:** Start with VARRAY of VARRAY. Execute the following block:

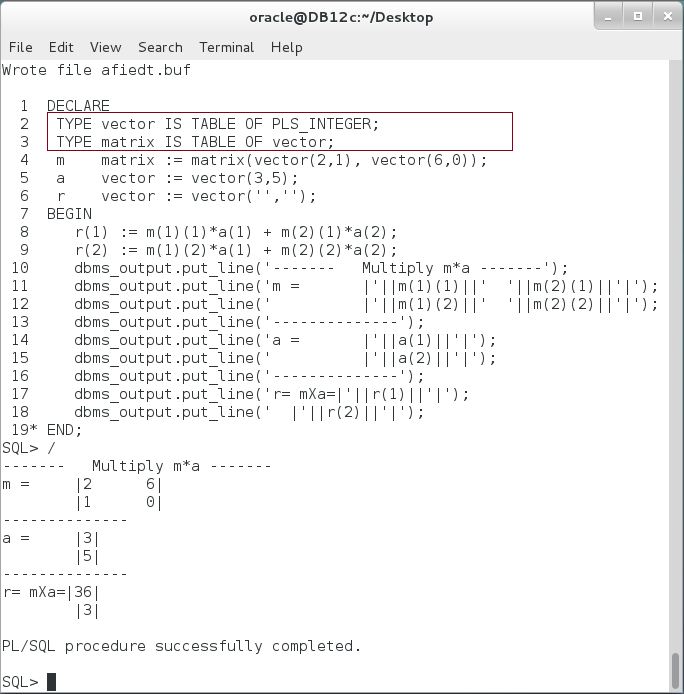
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| TYPE vector IS VARRAY(2) OF PLS\_INTEGER; |  |
| **TYPE matrix IS VARRAY(2) OF vector;** | VARRAY of VARRAY |
| **m matrix := matrix(vector(2,1), vector(6,0));** | Declare variables and initialize them. |
| a vector := vector(3,5); |
| r vector := vector('',''); |
| BEGIN |  |
| r(1) := m(1)(1)\*a(1) + m(2)(1)\*a(2); | Note how to access/set VARRAY-OF-VARRAY variable. |
| r(2) := m(1)(2)\*a(1) + m(2)(2)\*a(2); |
| dbms\_output.put\_line('------- Multiply m\*a -------'); |
| dbms\_output.put\_line('m = |'||m(1)(1)||' '||m(2)(1)||'|'); |
| dbms\_output.put\_line(' |'||m(1)(2)||' '||m(2)(2)||'|'); |
| dbms\_output.put\_line('--------------'); |  |
| dbms\_output.put\_line('a = |'||a(1)||'|'); |  |
| dbms\_output.put\_line(' |'||a(2)||'|'); |  |
| dbms\_output.put\_line('--------------'); |  |
| dbms\_output.put\_line('r= mXa=|'||r(1)||'|'); |  |
| dbms\_output.put\_line(' |'||r(2)||'|'); |  |
| END; |  |
| / |  |

****

**Please note:** VARRAY OF VARRAY can efficiently be use to manipulate Matrices. And it still easy of use.

**Step 2:** Similarly, you may use Nested Table instead of VARRAY if you don't know the number of elements. Modify the previous example as shown below:

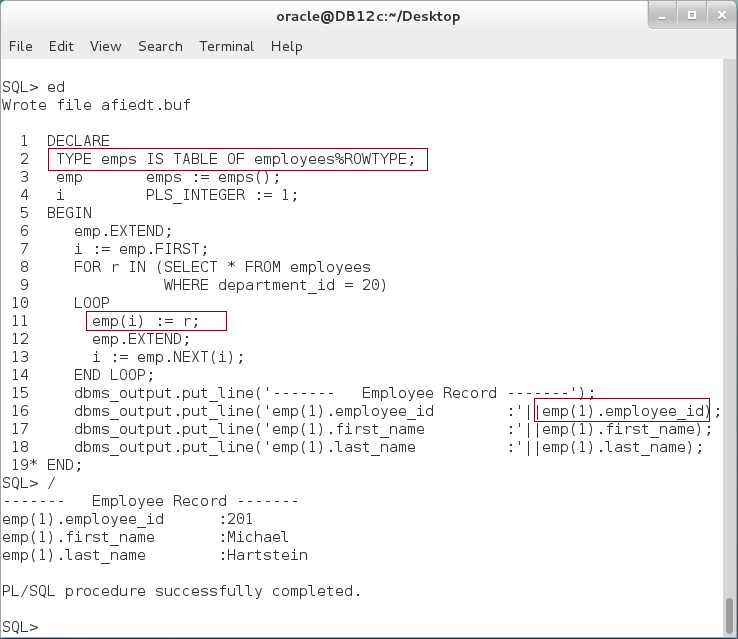
|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **TYPE vector IS TABLE OF PLS\_INTEGER;** | **NESTED TABLE OF NESTED TABLE.** |
| **TYPE matrix IS TABLE OF vector;** |
| m matrix := matrix(vector(2,1), vector(6,0)); |  |
| a vector := vector(3,5); |
| r vector := vector('',''); |
| BEGIN |  |
| r(1) := m(1)(1)\*a(1) + m(2)(1)\*a(2); |  |
| r(2) := m(1)(2)\*a(1) + m(2)(2)\*a(2); |
| dbms\_output.put\_line('------- Multiply m\*a -------'); |
| dbms\_output.put\_line('m = |'||m(1)(1)||' '||m(2)(1)||'|'); |
| dbms\_output.put\_line(' |'||m(1)(2)||' '||m(2)(2)||'|'); |
| dbms\_output.put\_line('--------------'); |  |
| dbms\_output.put\_line('a = |'||a(1)||'|'); |  |
| dbms\_output.put\_line(' |'||a(2)||'|'); |  |
| dbms\_output.put\_line('--------------'); |  |
| dbms\_output.put\_line('r= mXa=|'||r(1)||'|'); |  |
| dbms\_output.put\_line(' |'||r(2)||'|'); |  |
| END; |  |
| / |  |

****

**Please note:** You can mix between VARRAY, Nested Table, and Associative Array to create a multi-dimensional collection of your need.

**Step 3:** You can mix Collection with Record to create a TABLE like data type. Execute the following block:

|  |  |
| --- | --- |
| Line | Description |
| DECLARE |  |
| **TYPE emps IS TABLE OF employees%ROWTYPE;** | **Declare Nested Table of Records.** |
| emp emps := emps(); |
| i PLS\_INTEGER := 1; |
| BEGIN |  |
| emp.EXTEND; |  |
| i := emp.FIRST; |  |
| FOR r IN (SELECT \* FROM employees |  |
| WHERE department\_id = 20) |  |
| LOOP |  |
| **emp(i) := r;** | Note how you can assign a Record of Nested Table (emp) to an implicit variable Record (r) |
| emp.EXTEND; |
| i := emp.NEXT(i); |
| END LOOP; |
| dbms\_output.put\_line('------- Employee Record -------'); |
| dbms\_output.put\_line('emp(1).employee\_id :'||**emp(1).employee\_id**); | Note how you can set/get a Record field of a Nested Table. |
| dbms\_output.put\_line('emp(1).first\_name :'||**emp(1).first\_name**); |
| dbms\_output.put\_line('emp(1).last\_name :'||emp(1).last\_name); |
| END; |
| / |  |

****

# SUMMARY

Composite Data types are those types that contain sub-types. Sub-types can be scalar or composite data types. Thus, it may develop types that are more complex. The table below may help you see you progress through this chapter. You may fill in the blank field of your example:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Record** | **Nested Table** | **VARRAY** | **Associative Array** |
| **Record** | Address(s) in person Record | Phone(s) in a person Record |  |  |
| **Nested Table** | Emulate TABLE | Unbounded Martix |  |  |
| **VARRAY** |  |  | Bounded Martix |  |
| **Associative Array** |  |  |  |  |

It is important, however, to recognize the differences between collection types. VARRAY is a bounded dense collection while Nested Table is an unbounded sparse collection. Associative Array is mostly used as lookup table. It does the best when you need find element from a collection in a constant time; it does not depend on the number of elements in the collection.

After completing this lab exercise, you should be able to use a combination of Record and Collection data types.

# REFERENCES

* http://docs.oracle.com/database/121/LNPLS/composites.htm#LNPLS005

# INDEX

Collection 4, 34

multi-dimension collection 2

Nested Table 4, 29

Record 4, 16, 34