

*For Better, Faster, Smarter, Oracle Solutions*

**Oracle PLSQL Tutorial (Part 11)**

In this part of our Oracle PLSQL tutorial we continue our introduction to the use of collections in Oracle PL/SQL by looking at varrays (varying arrays).

**Oracle PLSQL Collection Types**

We've already seen that there are [3 different types of PL/SQL collections](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-9.htm) :-

* nested tables - covered in [part 10](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-10.htm) of this Oracle PLSQL tutorial
* varrays - covered in this tutorial
* PL/SQL associative arrays - covered in [part 12](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-12.htm) of this PL/SQL tutorial .

**PLSQL Varrays**

Varrays are one-dimensional, variable length arrays, with the maximum length defined in the declaration. These are used when it is known in advance that you will only need a few of the items to be stored and you know in advance the maximum number that you will need to hold.

A good example of the use of these would be for storing telephone numbers or addresses for say employees - each person is unlikely to have more than half a dozen telephone numbers or addresses. We can create a table for storing these as follows:-

TYPE phone\_no\_tab\_typ IS VARRAY(6) OF VARCHAR2(20;

phone\_nos phone\_no\_tab\_typ;

The preceding 2 statements have created an un-initialised array that is available within our PL/SQL session. Inserting and retrieving data is done in the same way as with nested tables. First of all we have to initialise the array, then we can insert data into and retrieve data from the varray.

As with nested tables, if you attempt to read from or write to an un-initialised varray in PLSQL, Oracle will generate an error which will cause an exception to be raised, as in the following examples:

SQL> declare  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20) ;  
3 phone\_nos phone\_no\_tab;  
4 begin  
5 dbms\_output.put\_line('phone\_no(1) is '||phone\_nos(1));  
6 end;  
7 /  
declare  
\*  
ERROR at line 1:  
ORA-06531: Reference to uninitialized collection  
ORA-06512: at line 5

SQL> declare  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20) ;  
3 phone\_nos phone\_no\_tab;  
4 begin  
5 phone\_nos(1) := 1;  
6 end;  
7 /  
declare  
\*  
ERROR at line 1:  
ORA-06531: Reference to uninitialized collection  
ORA-06512: at line 5

We can initialise our PLSQL varrays either in the declaration or separately. If the varray is only partially initialised we have to use the EXTEND method to define extra cells. Let's see how to do this in the next example in our PLSQL tutorial.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab := phone\_no\_tab('+44 (0)117 942 2508'); -- only 1st cell initailised  
4 BEGIN  
5 phone\_nos (1) := '0117 942 2508';  
6 DBMS\_OUTPUT.PUT\_LINE('phone\_no(1) is '||phone\_nos(1));  
7 END;  
8 /  
phone\_no(1) is 21

This example partially initialised the table when it was declared . Notice the use of the constructor for our collection on line 3 of our anonymous PLSQL block. The use of this enabled us to create one cell in the array. We we could have used the constructor to define as many elements as we liked, to do this we would just have to separate each element by a comma. For example to define two elements, the syntax would be like this:

phone\_nos phone\_no\_tab := phone\_no\_tab('+44 (0)117 942 2508','0117 942 2508');

This next example in our PLSQL tutorial will demonstrate the use of the EXTEND method.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab ;  
4 BEGIN  
5 phone\_nos := phone\_no\_tab(); --create empty varray  
6 phone\_nos.EXTEND(2); -- create first 2 cells  
7 phone\_nos(1) := '0117 942 2508';  
8 DBMS\_OUTPUT.PUT\_LINE('phone\_no(1) is '||phone\_nos(1));  
9 END;  
10 /  
phone\_no(1) is 0117 942 2508

Note the use of the constructor again on line 5, without this we would get the same error as before (ORA-06531 Reference to uninitialized collection). This is because we created a PL/SQL varray with no elements, so we have to use the EXTEND method to create one or more NULL elements ( in this example we created two) .

The EXTEND method works the same way as it does with PLSQL nested tables - if we don't specify the number of elements to create, then just one will be created. We can also optionally specify that each new element should be set to the same value as an existing element as in the following example in our PLSQL tutorial.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab ;  
4 BEGIN  
5 phone\_nos := phone\_no\_tab();  
6 phone\_nos.EXTEND(2);  
7 phone\_nos(1) := '0117 942 2508';  
8 phone\_nos.EXTEND(2,1);  
9 DBMS\_OUTPUT.PUT\_LINE('phone\_no(4) is '||phone\_nos(4));  
10 END;  
11 /  
phone\_no(4) is 0117 942 2508

On line 8, we specified that we two new elements should be created and that each new element should have the same value as the first element (the 2nd parameter specifies the cell which should be copied).

It is also worth noting that as with nested tables we have no control over where these elements are created, Oracle automatically appends them to the existing table so that we always have a dense structure unless we specifically delete elements. However unlike nested tables, varrays have an upper limit, so if we attempt to add more rows than the upper limit this causes the error ORA-06532: Subscript outside of limit.

So far in this PLSQL tutorial we've learnt how to initialise the table and add new elements to it. The next example in our PLSQL tutorial shows what happens when we reference an element that doesn't exist after we've initialised the table:

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab() ;  
4 BEGIN  
5 phone\_nos.EXTEND(2);  
6 DBMS\_OUTPUT.PUT\_LINE(  
7 'phone\_no(4) is '||phone\_nos(4));  
8 END;  
9 /  
declare  
\*  
ERROR at line 1:  
ORA-06533: Subscript beyond count  
ORA-06512: at line 6

As you can see Oracle raises an exception and we get the Oracle error ORA-06533.

Let's finish off this Oracle plsql tutorial by having a look at the other methods we can use on varrays in PL/SQL.

COUNT: This function returns the number of elements (cells) in the collection

DELETE: This deletes all the elements in the collection as varrays cannot be sparse. After deleting the whole collection any subsequent attempt to read or write to an element in the collection gives the following error: "ORA-06533: Subscript beyond count".

EXISTS(n) : used to determine if the specified element has been created and not deleted, returns TRUE if the element exists, FALSE if not

TRIM(n) : deletes the last n elements in the PLSQL collection, n defaults to 1 . An Oracle error (ORA-06533: Subscript beyond count) is generated if you attempt to TRIM elements that have already been TRIMmed or if the collection has been DELETEd.

The next PLSQL tutorial looks at PL/SQL associative arrays.

**Take advantage of one of our classroom-based [Oracle training](http://www.smart-soft.co.uk/Oracle/oracle-9i-training.htm) courses to extend, enhance and advance your knowledge of PL/SQL.** See the [Oracle training](http://www.smart-soft.co.uk/training/indexOracle.htm) page for details.

[Go to PLSQL tutorial Part 10 to learn about nested tables](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-11.htm)

[Go to PLSQL tutorial Part 12 to learn about associative arrays](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-12.htm)

[Go to PLSQL overview](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part1.htm)

---------------------------------------

**Looking for more Oracle tips and tricks?** To expand your knowledge [subscribe](http://www.asktheoracle.net/oracle-tips-signup.html) to our ezine.

*Smartsoft Computing Ltd  
Bristol, England  
Tel: 0845 0031320  
  
[Contact Us](http://www.smart-soft.co.uk/ContactUs.html)*

[Click here to view our privacy policy](http://www.smart-soft.co.uk/privacy_policy.htm) .

This site uses woopra.com to gather statistical information about our visitors. [View woopra privacy policy](http://www.woopra.com/privacy/) .

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright Smartsoft Computing Ltd 2001-2009. All rights reserved.

Top of Form

Search for: 

Bottom of Form



*For Better, Faster, Smarter, Oracle Solutions*

**Oracle PLSQL Tutorial (Part 11)**

In this part of our Oracle PLSQL tutorial we continue our introduction to the use of collections in Oracle PL/SQL by looking at varrays (varying arrays).

**Oracle PLSQL Collection Types**

We've already seen that there are [3 different types of PL/SQL collections](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-9.htm) :-

* nested tables - covered in [part 10](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-10.htm) of this Oracle PLSQL tutorial
* varrays - covered in this tutorial
* PL/SQL associative arrays - covered in [part 12](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-12.htm) of this PL/SQL tutorial .

**PLSQL Varrays**

Varrays are one-dimensional, variable length arrays, with the maximum length defined in the declaration. These are used when it is known in advance that you will only need a few of the items to be stored and you know in advance the maximum number that you will need to hold.

A good example of the use of these would be for storing telephone numbers or addresses for say employees - each person is unlikely to have more than half a dozen telephone numbers or addresses. We can create a table for storing these as follows:-

TYPE phone\_no\_tab\_typ IS VARRAY(6) OF VARCHAR2(20;

phone\_nos phone\_no\_tab\_typ;

The preceding 2 statements have created an un-initialised array that is available within our PL/SQL session. Inserting and retrieving data is done in the same way as with nested tables. First of all we have to initialise the array, then we can insert data into and retrieve data from the varray.

As with nested tables, if you attempt to read from or write to an un-initialised varray in PLSQL, Oracle will generate an error which will cause an exception to be raised, as in the following examples:

SQL> declare  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20) ;  
3 phone\_nos phone\_no\_tab;  
4 begin  
5 dbms\_output.put\_line('phone\_no(1) is '||phone\_nos(1));  
6 end;  
7 /  
declare  
\*  
ERROR at line 1:  
ORA-06531: Reference to uninitialized collection  
ORA-06512: at line 5

SQL> declare  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20) ;  
3 phone\_nos phone\_no\_tab;  
4 begin  
5 phone\_nos(1) := 1;  
6 end;  
7 /  
declare  
\*  
ERROR at line 1:  
ORA-06531: Reference to uninitialized collection  
ORA-06512: at line 5

We can initialise our PLSQL varrays either in the declaration or separately. If the varray is only partially initialised we have to use the EXTEND method to define extra cells. Let's see how to do this in the next example in our PLSQL tutorial.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab := phone\_no\_tab('+44 (0)117 942 2508'); -- only 1st cell initailised  
4 BEGIN  
5 phone\_nos (1) := '0117 942 2508';  
6 DBMS\_OUTPUT.PUT\_LINE('phone\_no(1) is '||phone\_nos(1));  
7 END;  
8 /  
phone\_no(1) is 21

This example partially initialised the table when it was declared . Notice the use of the constructor for our collection on line 3 of our anonymous PLSQL block. The use of this enabled us to create one cell in the array. We we could have used the constructor to define as many elements as we liked, to do this we would just have to separate each element by a comma. For example to define two elements, the syntax would be like this:

phone\_nos phone\_no\_tab := phone\_no\_tab('+44 (0)117 942 2508','0117 942 2508');

This next example in our PLSQL tutorial will demonstrate the use of the EXTEND method.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab ;  
4 BEGIN  
5 phone\_nos := phone\_no\_tab(); --create empty varray  
6 phone\_nos.EXTEND(2); -- create first 2 cells  
7 phone\_nos(1) := '0117 942 2508';  
8 DBMS\_OUTPUT.PUT\_LINE('phone\_no(1) is '||phone\_nos(1));  
9 END;  
10 /  
phone\_no(1) is 0117 942 2508

Note the use of the constructor again on line 5, without this we would get the same error as before (ORA-06531 Reference to uninitialized collection). This is because we created a PL/SQL varray with no elements, so we have to use the EXTEND method to create one or more NULL elements ( in this example we created two) .

The EXTEND method works the same way as it does with PLSQL nested tables - if we don't specify the number of elements to create, then just one will be created. We can also optionally specify that each new element should be set to the same value as an existing element as in the following example in our PLSQL tutorial.

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab ;  
4 BEGIN  
5 phone\_nos := phone\_no\_tab();  
6 phone\_nos.EXTEND(2);  
7 phone\_nos(1) := '0117 942 2508';  
8 phone\_nos.EXTEND(2,1);  
9 DBMS\_OUTPUT.PUT\_LINE('phone\_no(4) is '||phone\_nos(4));  
10 END;  
11 /  
phone\_no(4) is 0117 942 2508

On line 8, we specified that we two new elements should be created and that each new element should have the same value as the first element (the 2nd parameter specifies the cell which should be copied).

It is also worth noting that as with nested tables we have no control over where these elements are created, Oracle automatically appends them to the existing table so that we always have a dense structure unless we specifically delete elements. However unlike nested tables, varrays have an upper limit, so if we attempt to add more rows than the upper limit this causes the error ORA-06532: Subscript outside of limit.

So far in this PLSQL tutorial we've learnt how to initialise the table and add new elements to it. The next example in our PLSQL tutorial shows what happens when we reference an element that doesn't exist after we've initialised the table:

SQL> DECLARE  
2 TYPE phone\_no\_tab IS VARRAY(6) OF VARCHAR2(20);  
3 phone\_nos phone\_no\_tab() ;  
4 BEGIN  
5 phone\_nos.EXTEND(2);  
6 DBMS\_OUTPUT.PUT\_LINE(  
7 'phone\_no(4) is '||phone\_nos(4));  
8 END;  
9 /  
declare  
\*  
ERROR at line 1:  
ORA-06533: Subscript beyond count  
ORA-06512: at line 6

As you can see Oracle raises an exception and we get the Oracle error ORA-06533.

Let's finish off this Oracle plsql tutorial by having a look at the other methods we can use on varrays in PL/SQL.

COUNT: This function returns the number of elements (cells) in the collection

DELETE: This deletes all the elements in the collection as varrays cannot be sparse. After deleting the whole collection any subsequent attempt to read or write to an element in the collection gives the following error: "ORA-06533: Subscript beyond count".

EXISTS(n) : used to determine if the specified element has been created and not deleted, returns TRUE if the element exists, FALSE if not

TRIM(n) : deletes the last n elements in the PLSQL collection, n defaults to 1 . An Oracle error (ORA-06533: Subscript beyond count) is generated if you attempt to TRIM elements that have already been TRIMmed or if the collection has been DELETEd.

The next PLSQL tutorial looks at PL/SQL associative arrays.

**Take advantage of one of our classroom-based [Oracle training](http://www.smart-soft.co.uk/Oracle/oracle-9i-training.htm) courses to extend, enhance and advance your knowledge of PL/SQL.** See the [Oracle training](http://www.smart-soft.co.uk/training/indexOracle.htm) page for details.

[Go to PLSQL tutorial Part 10 to learn about nested tables](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-11.htm)

[Go to PLSQL tutorial Part 12 to learn about associative arrays](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part-12.htm)

[Go to PLSQL overview](http://www.smart-soft.co.uk/Oracle/oracle-plsql-tutorial-part1.htm)

---------------------------------------

**Looking for more Oracle tips and tricks?** To expand your knowledge [subscribe](http://www.asktheoracle.net/oracle-tips-signup.html) to our ezine.

*Smartsoft Computing Ltd  
Bristol, England  
Tel: 0845 0031320  
  
[Contact Us](http://www.smart-soft.co.uk/ContactUs.html)*

[Click here to view our privacy policy](http://www.smart-soft.co.uk/privacy_policy.htm) .

This site uses woopra.com to gather statistical information about our visitors. [View woopra privacy policy](http://www.woopra.com/privacy/) .

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright Smartsoft Computing Ltd 2001-2009. All rights reserved.

Top of Form

Search for: 

Bottom of Form