PL/SQL Tables - Introduction

Creating a PL/SQL table will allow the Developer to store rows of data in the process without the need to return to the Database, in addition, rows can be compared with each other as well as their data manipulated before updating the underlying tables.

A PL/SQL table can consist of one simple datatype or be defined as a type of record and is sometimes referred to as an Index by table.

Rows in a PL/SQL table work like an array in the sense that a particular row can be referenced by its index …

<plsql_table>(10) := '<value>';

PL/SQL Tables - Introduction

Unlike an array however the table is one dimensional and stores in rows rather than cells ..

Rows in a PL/SQL table do not have to be contiguous, empty rows do not cause storage problems so therefore the Developer can use keys based on the data rather than sequences.

PL/SQL Tables - Declaring

The declaration is as follows ….

type <pl_table_name> is table of <datatype>
index by binary_integer;

Datatype can be a simple datatype, %rowtype, %type or a record.

This type can then be assigned to a local variable ..

<variable> <pl_table_name>;

In 9i the indexing has been extended to include

index by varchar2(<length>);

PL/SQL Tables - Populating

Tables with simple datatypes can be populated thus ..

<variable>(<integer>) := <value>;

Tables with complex datatypes will need the columns populated individually ..

<variable>(<integer>).<column_name> := <value>;

Or from a cursor ..

fetch <cursor_name> into <variable>(<integer>);

PL/SQL Tables - Populating

Although we use an integer to signify the position of the record within the table, it is just as valid to place a record in a negative value row ..

l_table(-12445) := 'Mr Jones';

PL/SQL Tables - Using

If a row is specified which doesn’t have a value the exception No_Data_Found will be raised, therefore the Developer should cater for this event within the coding.

It is recommended that records are inserted into the table in sequence, however, it is possible to store records in whatever order and position is required, this will necessitate the use of the above exception.

PL/SQL Tables - Using Count

There are several functions available for the Developer to use when using PL/SQL tables …

Count - Counts the number of rows populated within the table

```
open get_stock;
fetch get_stock into l_tab(1);
close get_stock;

l_count := l_tab.count;

dbms_output.put_line('There are '||to_char(l_count)||' records');
```

PL/SQL Tables - Using Delete

Delete - deletes all records in a table, a range of records, or a specified record

l_tab.delete;
l_tab.delete(1,10);
l_tab.delete(12);

PL/SQL Tables - Using Exists

Exists - Boolean function which returns true if a row is populated, this can be very useful if avoiding using No_Data_Found

```
if l_tab.exists(1) then

    dbms_output.put_line('There is a record in position 1');

else

    dbms_output.put_line('There is no record in position 1');

end if;
```

PL/SQL Tables - Using First/Last

First - Gives the position of the first record in the table .. That is, the one with the lowest index value, the opposite to this is Last and between them will give the range of records but not necessarily a contiguous range

l_first := l_tab.first;

PL/SQL Tables - Using Next/Prior

Next - Gives the position of the first record in the table following the value specified in the function, Prior gives the position of the first record in the table before the value specified.

l_next := l_tab.next(245);

PL/SQL Tables - Passing as Parameters

Entire tables can be passed between processes, in the following example a Package has been created with a PL/SQL table defined, this is then accessed by two processes, the first passing the entire contents of its table to the second ..

```
create or replace package plsql_tabs is

    type pl_table is table of stock_information%rowtype
                        index by binary_integer;

end plsql_tabs;
```

PL/SQL Tables - Passing as Parameters

```
create or replace procedure process_one is

   cursor get_stock is
          select * from stock_information;

   l_tab plsql_tabs.pl_table;
   l_count number := 0;

begin

   for stock_rec in get_stock loop
      l_count := l_count + 1;
      l_tab(l_count) := stock_rec;
   end loop;
   process_two(l_tab);

end;
```

PL/SQL Tables - Passing as Parameters

```
create or replace procedure process_two(p_tab in plsql_tabs.pl_table) is

l_count number;

begin

   l_count := p_tab.count;

   for stock_rec in 1..l_count loop

      dbms_output.put_line(p_tab(stock_rec).si_stock_description);

   end loop;

end;
```