Agenda

- Introduction
- PL/SQL Collections
- Associative Arrays, Nested Tables, VARRAY
- Collections Similarities and Differences
- Defining, Declaring and Initializing Collections
- Multidimensional Collections
- Collection Methods
Introduction

• Variable of scalar data type can hold only one value.
  • Numeric, Character, Boolean, Date, etc.

• Variable of composite data type can hold multiple values.
  • PL/SQL Records: The internal components can be of different data types and are called fields. Record can hold a table row or some columns from a table row.
    
    1) `<variable_name> <table_name / cursor_name> %ROWTYPE;
    2) TYPE `<record_name> IS RECORD (field1 datatype1, field2 datatype2, field3 datatype3, ......);

• PL/SQL Collections: The internal components are always of the same data type and are called elements. Array is an example of collection.
PL/SQL Collections

- An ordered group of elements
- All of the same type
- Each element can be accessed by its unique subscript
- Are used to treat data as a single unit like an array or list
- Data is easier to manage, relate and transport

Types of PL/SQL Collections

- Associative Arrays or PL/SQL Table or index-by tables
- Nested Tables
- Varrays
Associative Arrays

- Also known as PL/SQL Table or index-by tables
- Can be used only in PL/SQL blocks
- Are sets of key-value pairs
- Subscripts can be either strings or integers
- Subscripts are stored in sort order, not creation order
- Cannot specify an initial value when you declare an associative array variable

<table>
<thead>
<tr>
<th>Index Key (Integer)</th>
<th>Element Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100001</td>
</tr>
<tr>
<td>2</td>
<td>100002</td>
</tr>
<tr>
<td>3</td>
<td>100003</td>
</tr>
<tr>
<td>4</td>
<td>100004</td>
</tr>
<tr>
<td>5</td>
<td>100005</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index Key (String)</th>
<th>Element Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scott</td>
<td>2000</td>
</tr>
<tr>
<td>Smith</td>
<td>3500</td>
</tr>
<tr>
<td>Jones</td>
<td>5200</td>
</tr>
<tr>
<td>James</td>
<td>4500</td>
</tr>
<tr>
<td>Allen</td>
<td>3500</td>
</tr>
</tbody>
</table>
Nested Tables

• Can be used in SQL and PL/SQL Blocks both

• Column type that stores an unbounded set of rows in no particular order

• Sequential numbers as subscripts

• Uninitialized nested table variable is a null collection

• Must initialize it either by making it empty or by assigning a non-NULL

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>700</td>
<td>800</td>
<td>900</td>
<td>1000</td>
</tr>
<tr>
<td>x(1)</td>
<td>x(2)</td>
<td>x(3)</td>
<td>x(4)</td>
<td>x(5)</td>
<td>x(6)</td>
<td>x(7)</td>
<td>x(8)</td>
<td>x(9)</td>
<td>x(10)</td>
</tr>
</tbody>
</table>
VARRAY

- Can be used in SQL and PL/SQL Blocks both
- Equivalent of array types in other languages
- Sequential numbers as subscripts
- Uninitialized VARRAY variable is a null collection
- Must initialize it either by making it empty or by assigning a non-NULL

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
<th>600</th>
<th>700</th>
<th>800</th>
<th>900</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>x(1)</td>
<td>x(2)</td>
<td>x(3)</td>
<td>x(4)</td>
<td>x(5)</td>
<td>x(6)</td>
<td>x(7)</td>
<td>x(8)</td>
<td>x(9)</td>
<td>x(10)</td>
<td></td>
</tr>
</tbody>
</table>
# Collections similarities and differences

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Elements Limit</th>
<th>Subscript Type</th>
<th>Uninitialized Status</th>
<th>Always Dense</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associative array</td>
<td>Unlimited</td>
<td>String or Integer</td>
<td>Empty</td>
<td>NO</td>
</tr>
<tr>
<td>Nested table</td>
<td>Unlimited</td>
<td>Integer</td>
<td>NULL</td>
<td>NO</td>
</tr>
<tr>
<td>VARRAY</td>
<td>Fixed</td>
<td>Integer</td>
<td>NULL</td>
<td>YES</td>
</tr>
</tbody>
</table>
# Defining and Declaring Collections

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Defining Collection</th>
<th>Declaring Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associative Array (index by integer)</td>
<td>TYPE sno_at IS TABLE OF NUMBER INDEX BY PLS_INTEGER;</td>
<td>v_sno sno_at;</td>
</tr>
<tr>
<td>Associative Array (index by string)</td>
<td>TYPE marks_at IS TABLE OF NUMBER INDEX BY VARCHAR2(100);</td>
<td>v_marks marks_at;</td>
</tr>
<tr>
<td>Nested Table</td>
<td>TYPE fruits_nt IS TABLE OF VARCHAR2;</td>
<td>v_fruits fruits_nt;</td>
</tr>
<tr>
<td>VARRAY</td>
<td>TYPE fruits_vt IS VARRAY(10) OF NUMBER;</td>
<td>v_fruits fruits_vt;</td>
</tr>
</tbody>
</table>

sno_at, marks_at, fruits_nt and fruits_vt are type names  
v_sno, v_marks and v_fruits are collection variables for the respective collection types
### Initializing Collections

Initializing collections in executable part (BEGIN … END) of PL/SQL Block

<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Initializing Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associative Array (index by integer)</td>
<td>v_sno(-50) := 1;</td>
</tr>
<tr>
<td>Associative Array (index by integer)</td>
<td>v_sno(20) := 250;</td>
</tr>
<tr>
<td>Associative Array (index by string)</td>
<td>v_marks('Scott') := 90;</td>
</tr>
<tr>
<td>Nested Table</td>
<td>v_fruits := fruits_nt('APPLE','BANANA','GUAVA');</td>
</tr>
<tr>
<td>VARRAY</td>
<td>v_fruits := fruits_vt('APPLE','BANANA','GUAVA');</td>
</tr>
</tbody>
</table>
Comparing Collections

- You can check whether a collection is null.
- Comparisons such as greater than, less than, etc are not allowed.
- For nested tables, you can check whether two nested table of the same declared type are equal or not equal.

\[
\begin{array}{ccccc}
100 & 200 & 300 & 400 & 500 \\
\text{x(1)} & \text{x(2)} & \text{x(3)} & \text{x(4)} & \text{x(5)} \\
\end{array}
= 
\begin{array}{ccccc}
500 & 300 & 200 & 100 & 400 \\
\text{y(1)} & \text{y(2)} & \text{y(3)} & \text{y(4)} & \text{y(5)} \\
\end{array}
\]

\[
\begin{array}{ccccc}
100 & 200 & 300 & 400 & 500 \\
\text{x(1)} & \text{x(2)} & \text{x(3)} & \text{x(4)} & \text{x(5)} \\
\end{array}
\neq 
\begin{array}{cccc}
100 & 200 & 300 & 400 \\
\text{y(1)} & \text{y(2)} & \text{y(3)} & \text{y(4)} \\
\end{array}
\]
Multidimensional Collections

- You can create a nested table of varrays, a varray of varrays, a varray of nested tables, etc.
- Multidimensional collection elements are also collections.

| Associative Array (index by string) | TYPE sno_at IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
| TYPE sno_mat IS TABLE OF sno_at INDEX BY PLS_INTEGER; |
| Nested Table | TYPE fruits_nt IS TABLE OF NUMBER;
| TYPE fruits_mnt IS TABLE OF fruits_nt; |
| VARRAY | TYPE fruits_vt IS VARRAY(10) OF NUMBER;
| TYPE fruits_mvt IS VARRAY(10) OF fruits_vt; |
Collection Methods - Introduction

- A collection method is a built-in function or procedure that operates on collection.

- Collections methods make collections easier to use and make your applications easier to maintain.

- Collection methods cannot be called from SQL statements.

- The following is the basic syntax of collection method.

  `<collection_name>.method`
## Collection Methods

<table>
<thead>
<tr>
<th>Collection Method</th>
<th>Method Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>Procedure</td>
<td>Deletes elements from collection.</td>
</tr>
<tr>
<td>TRIM</td>
<td>Procedure</td>
<td>Deletes elements from end of varray or nested table.</td>
</tr>
<tr>
<td>EXTEND</td>
<td>Procedure</td>
<td>Adds elements to end of varray or nested table.</td>
</tr>
<tr>
<td>EXISTS</td>
<td>Function</td>
<td>Returns TRUE if and only if specified element of varray or nested table exists.</td>
</tr>
<tr>
<td>FIRST</td>
<td>Function</td>
<td>Returns first subscript in collection.</td>
</tr>
<tr>
<td>LAST</td>
<td>Function</td>
<td>Returns last subscript in collection.</td>
</tr>
<tr>
<td>COUNT</td>
<td>Function</td>
<td>Returns number of elements in collection.</td>
</tr>
<tr>
<td>LIMIT</td>
<td>Function</td>
<td>Returns maximum number of elements that collection can have.</td>
</tr>
<tr>
<td>PRIOR</td>
<td>Function</td>
<td>Returns subscript that precedes specified subscript.</td>
</tr>
<tr>
<td>NEXT</td>
<td>Function</td>
<td>Returns subscript that succeeds specified subscript.</td>
</tr>
</tbody>
</table>

**NOTE:** EXTEND and TRIM cannot be used with Associative Arrays.
DECLARE

    TYPE  names_at  IS TABLE OF NUMBER INDEX BY VARCHAR2(100);

    v_names   names_at;

BEGIN

    v_names('Scott') := 90;
    v_names('Smith') := 92;
    v_names('Jones') := 95;
    DBMS_OUTPUT.PUT_LINE('Scott marks ' || v_names('Scott'));

END;
/

ASSOCIATIVE ARRAY - Example
DECLARE

    TYPE  ascii_nt  IS TABLE OF VARCHAR2(20);
    v_ascii      ascii_nt := ascii_nt(97,98,99,100,101);

BEGIN

    FOR i IN v_ascii.FIRST..v_ascii.LAST LOOP
        DBMS_OUTPUT.PUT_LINE(CHR(v_ascii(i)));
    END LOOP;

END;
/
DECLARE

TYPE  ascii_vt  IS VARRAY(26) OF VARCHAR2(20);
  v_ascii      ascii_vt :=
      ascii_vt(65,66,67,68,69,70,71,72,73,74,75,76,77,78,79,80
          ,81,82,83,84,85,86,87,88,89,90);

BEGIN

  FOR i IN v_ascii.FIRST..v_ascii.LAST LOOP
    DBMS_OUTPUT.PUT_LINE(CHR(v_ascii(i)));
  END LOOP;

END;
/

VARRAY - Example
DEMO Scripts
Questions
Learn More From:  http://docs.oracle.com

Thanks For Coming !!

Sailaja Pasupuleti