Working with Collections in Oracle PL/SQL

Need for Collection Types & Their Characteristics

Pankaj Jain

@twit_pankajj

Course Overview

FORALL

Nested Table Comparisons & Operators

Varrays

Multilevel Collections & Conversions

Bulk Collect

Introduction

Associative Arrays

Collection Methods

Nested Tables

Pre-requisites

Basic Oracle Programming Knowledge

Recommendations

- Oracle PL/SQL Fundamentals - Part 1
- Oracle PL/SQL Fundamentals - Part 2
- Oracle PL/SQL: Transactions, Dynamic SQL & Debugging

Audience

Oracle Database Programmers

Web Developers

Other Programmers

Tools

Oracle Database

SQL Developer

SQLPLUS

Toad

SQL Navigator

Module Overview

Overview & Need

Structure & Notation

Characteristics

Collections | Composite Datatypes

Associative Arrays	Varrays	Nested Tables

Need

Interplay with other languages

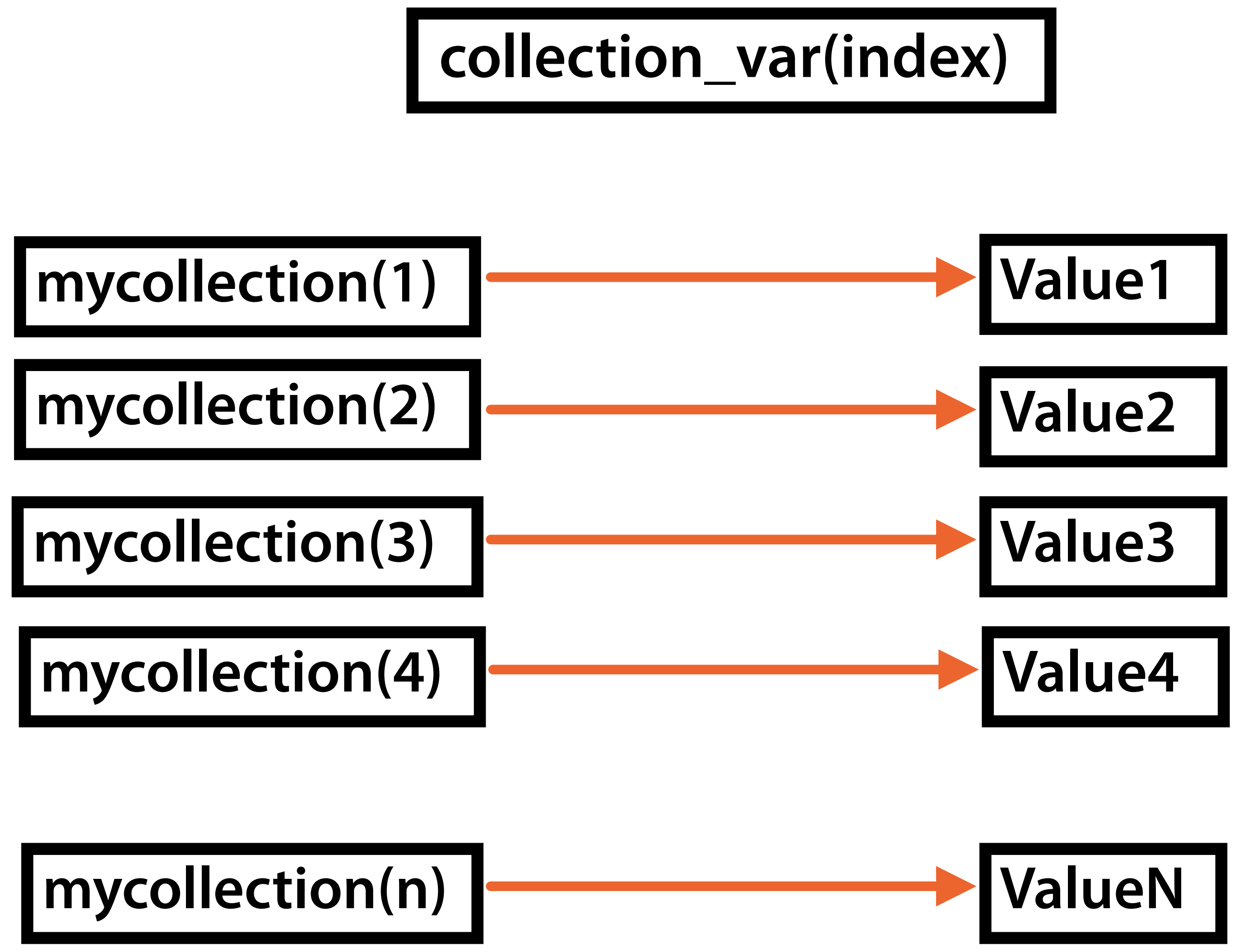
Compact code

Data grouping and manipulation

Unknown number of variables

Performance

Notating Collections



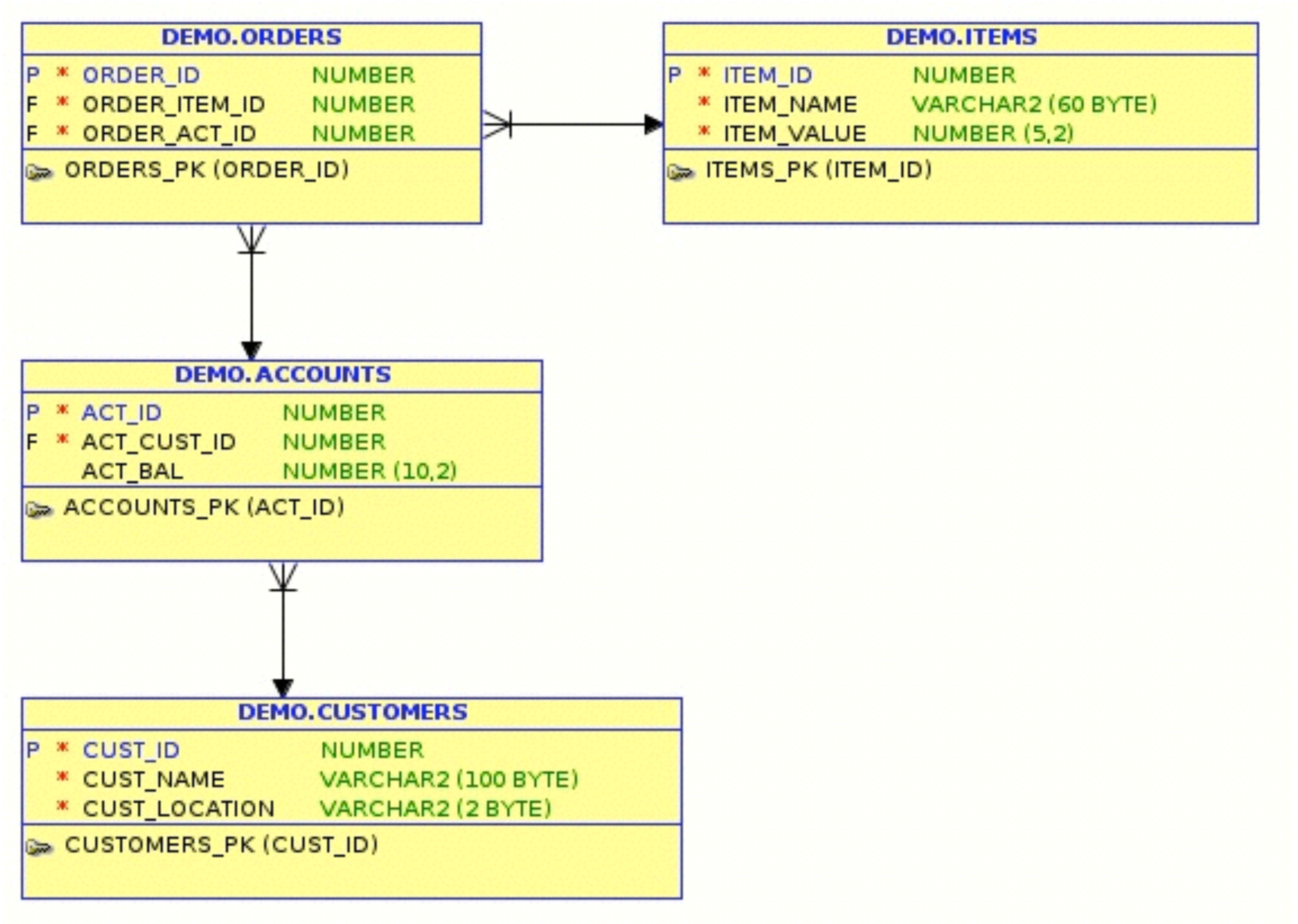
Characteristics

- Density
- Number of elements
- Index datatype
- Where defined
- Collection methods

Translation

Other Language Composite Types	Eqivalent PL/SQL Composite Type
Hash table	Associative array
Unordered table	Associative array
Set	Nested table
Bag	Nested table
Array	VARRAY

Tables



DEMO.ORDERS

P	*	ORDER_ID	NUMBER
F	*	ORDER_ITEM_ID	NUMBER
F	*	ORDER_ACT_ID	NUMBER

ORDERS_PK (ORDER_ID)

DEMO.ITEMS

P	*	ITEM_ID	NUMBER
	*	ITEM_NAME	VARCHAR2 (60 BYTE)
	*	ITEM_VALUE	NUMBER (5,2)

ITEMS_PK (ITEM_ID)

DEMO.ACCOUNTS

P	*	ACT_ID	NUMBER
F	*	ACT_CUST_ID	NUMBER
		ACT_BAL	NUMBER (10,2)

ACCOUNTS_PK (ACT_ID)

DEMO.CUSTOMERS

P	*	CUST_ID	NUMBER
	*	CUST_NAME	VARCHAR2 (100 BYTE)
	*	CUST_LOCATION	VARCHAR2 (2 BYTE)

CUSTOMERS_PK (CUST_ID)

Summary

Need

Collection Structure

Characteristics

Next up.. Associative Arrays

Associative Arrays

Pankaj Jain

@twit_pankajj

Module Overview

Usage guidelines

Where can they be declared?

What is an associative array?

Definition and use

Associative array index

What is an Associative Array?

PL/SQL table

Index-by table

Key-Value pair

String or PLS_INTEGER key type

PL/SQL Only datatype

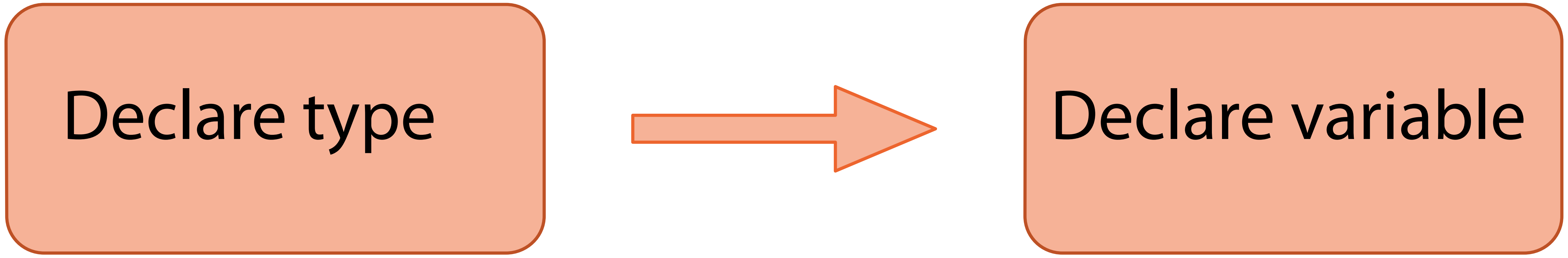
In-Memory table

Defining Associative Array

```
TYPE <type_name> IS TABLE OF <element_type> [NOT NULL]

INDEX BY [BINARY_INTEGER | PLS_INTEGER |VARCHAR2(size_limit)]
```

```
TYPE mytype_aa IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;

TYPE mytype_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;

TYPE mytype_aa IS TABLE OF emp%ROWTYPE INDEX BY BINARY_INTEGER;
```

Declaring Associative Array



```
DECLARE
      TYPE items_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;

..
       l_items_aa  items_aa;
```

Initialized
as an empty but not
null array

Assigning Value to an Associative Array

```
DECLARE
  TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;

  l_items_aa      items_aa;

BEGIN

  l_Items_aa(1) := 'Treadmill';

  l_Items_aa(2) := 'Bike';

  l_items_aa(3) := 'Elliptical';

 dbms_output.put_line(l_items_aa(2));

END;
```

Assigning Value to an Associative Array

Assigning Another Collection

```
DECLARE
  TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa       items_aa;
  l_copy_aa        items_aa;
BEGIN
  l_Items_aa(1) := 'Treadmill';
  l_Items_aa(2) := 'Bike';
  l_items_aa(3) := 'Elliptical';

  l_copy_aa := l_items_aa;

 dbms_output.put_line(l_copy_aa(2));
END;
```

Assigning Value to an Associative Array

Same Type

```
DECLARE
 TYPE items_aa   IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 TYPE dup_aa     IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa      items_aa;
 l_dup_aa        dup_aa;
BEGIN
 l_Items_aa(1) := 'Treadmill';
 l_Items_aa(2) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 l_dup_aa := l_items_aa; ✗

END;
```

Initializing an Associative Array

Assigning Empty Array

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa        items_aa;
 l_empty_aa        items_aa;
BEGIN
 l_Items_aa(1) := 'Treadmill';
 l_Items_aa(2) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 l_items_aa := l_empty_aa;

END;
```

Bulk Collect Query

Associative Array Index Type

String

VARCHAR, VARCHAR2, String, Long

Numeric

PLS_INTEGER / BINARY_INTEGER

Associative Array Index

- Maximum size unspecified
  - BINARY_INTEGER   -2147483647..2147483647

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa          items_aa;

BEGIN







END;
```

Associative Array Index

Can Hold Negative Values

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa         items_aa;

BEGIN
 l_Items_aa(-10) := 'Treadmill';





END;
```

Associative Array Index

Can Be Sparse

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa         items_aa;

BEGIN
  l_Items_aa(-10) := 'Treadmill';
  l_Items_aa(20) := 'Bike';






END;
```

Associative Array Index

Index Values Need Not be Consecutive

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa        items_aa;

BEGIN
   l_Items_aa(-10) := 'Treadmill';

   l_Items_aa(20) := 'Bike';

   l_Items_aa(3) := 'Weights';



END;
```

Associative Array Index

Reassigning Overwrites Previous Value at that Index

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa        items_aa;

BEGIN
   l_Items_aa(-10) := 'Treadmill';
   l_Items_aa(20) := 'Bike';
   l_Items_aa(3) := 'Weights';
   l_Items_aa(20) := 'Elliptical';
  DBMS_OUTPUT.PUT_LINE(l_Items_aa(20)) ;
END;
```

Exceptions During Assignment

Not null constraint

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) NOT NULL INDEX BY BINARY_INTEGER;

 l_items_aa      items_aa;

BEGIN

 l_Items_aa(1) := NULL;

 l_Items_aa(2) := 'Bike';

 l_items_aa(3) := 'Elliptical';

 dbms_output.put_line(l_items_aa(2));

END;
```

PLS-00382: expression is of wrong type

Exceptions During Assignment

Value Error

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(4) INDEX BY BINARY_INTEGER;

 l_items_aa      items_aa;

BEGIN

 l_Items_aa(1) := 'Treadmill';

 l_Items_aa(2) := 'Bike';

EXCEPTION
 WHEN VALUE_ERROR THEN
   DBMS_OUTPUT.PUT_LINE(SQLCODE);
   RAISE;
END;
```

ORA-6502: PL/SQL numeric or value error

Exceptions During Assignment

No data found

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;

 l_items_aa      items_aa;

BEGIN

 l_Items_aa(1) := 'Treadmill';

 DBMS_OUTPUT.PUT_LINE(l_Items_aa(2));

EXCEPTION
 WHEN NO_DATA_FOUND  THEN
   DBMS_OUTPUT.PUT_LINE(SQLCODE);
   RAISE;
END;
```

ORA-01403: no data found

Exceptions During Assignment

Numeric overflow

BINARY_INTEGER Range

-2,147,483,648 through 2,147,483,647

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;

 l_items_aa      items_aa;

BEGIN

 l_Items_aa(2,147,483,648) := 'Treadmill';

END;
```

ORA-01426:
numeric overflow

First and Next

```
<collection_variable>.<collection_method>
```

FIRST

First index counter in collection

NEXT

Next index counter in collection

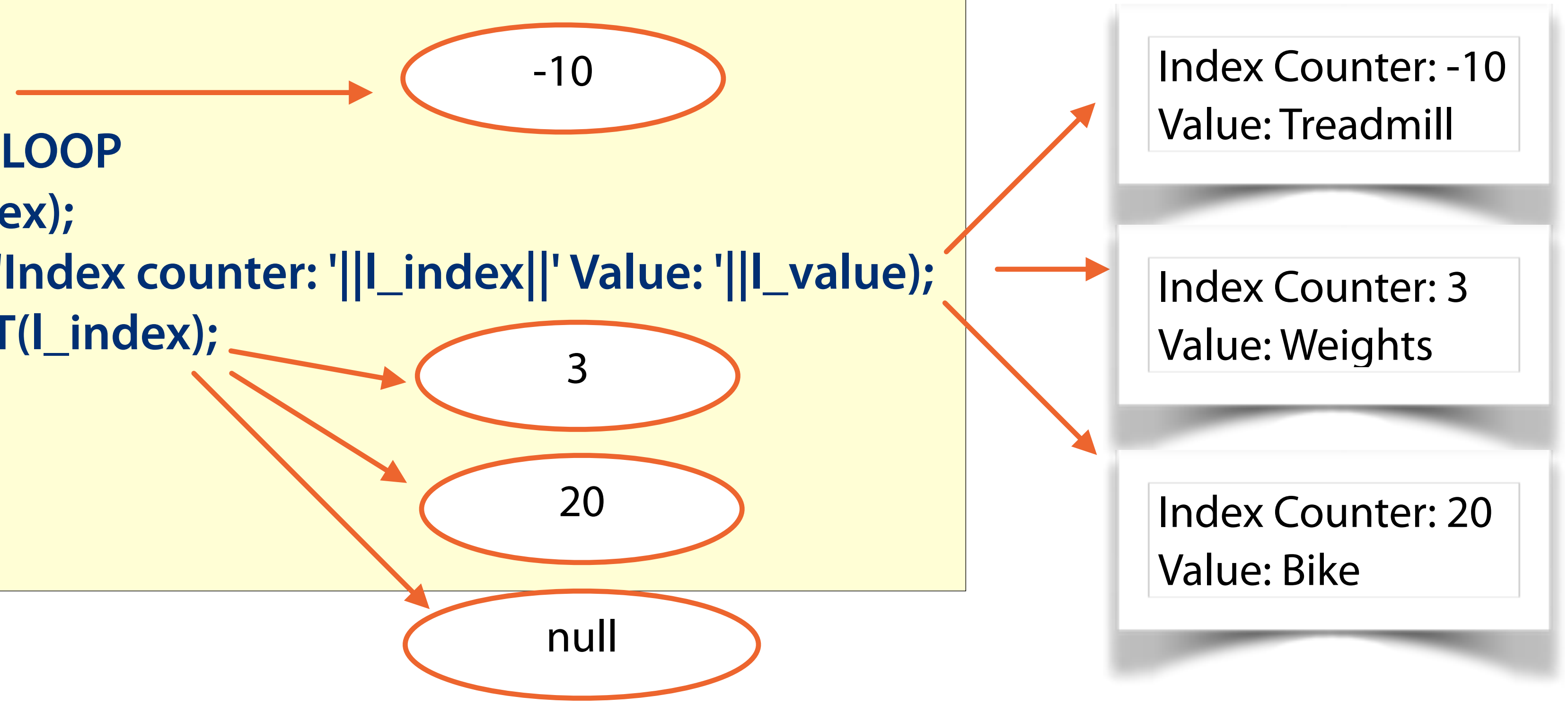
Associative Array Sorting

Numeric	Integer values
String	Character string

Associative Array Sorting

Integer Sorting

```
DECLARE
  TYPE items_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa  items_aa;
  l_index       BINARY_INTEGER;
  l_value       VARCHAR2(60);
BEGIN
  l_Items_aa(-10) := 'Treadmill';
  l_Items_aa(20)  := 'Bike';
  l_items_aa(3)    := 'Weights';
  l_index := l_items_aa.FIRST;
  WHILE  l_index IS NOT NULL LOOP
    l_value := l_items_aa(l_index);
    DBMS_OUTPUT.PUT_LINE('Index counter: '||l_index||' Value: '||l_value);
    L_index := l_items_aa.NEXT(l_index);
  END LOOP;
END;
```

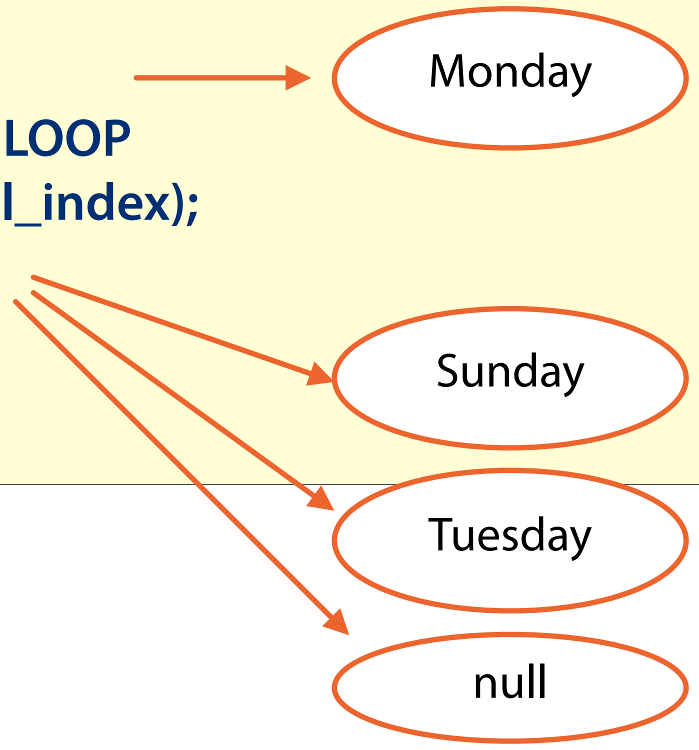


```
Index Counter: -10
Value: Treadmill

Index Counter: 3
Value: Weights

Index Counter: 20
Value: Bike
```

Associative Array Sorting

Character Sorting

```
DECLARE
  TYPE days_aa IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
  l_days_aa    days_aa;
  l_index        VARCHAR2(20);
BEGIN
  l_days_aa('Sunday')  := 1;
  l_days_aa('Monday') := 2;
  l_days_aa('Tuesday') := 3;
  l_index := l_days_aa.FIRST;
  WHILE  l_index IS NOT NULL LOOP
    l_index := l_days_aa.NEXT(l_index);
  END LOOP;
END;
```



Where Can They Be Declared?

Anonymous blocks

Stored programs units

Visibility

Local declaration → Local visibility

```
CREATE OR REPLACE FUNCTION get_order_counts ....
  TYPE items_aa IS TABLE OF VARCHAR2(60)
    INDEX BY PLS_INTEGER;
  l_items_aa    items_aa;
BEGIN
  l_items_aa(1)  := 'Treadmill';
  ...
END get_order_counts;
```

Package specification → Global visibility

```
CREATE OR REPLACE PACKAGE globals IS ....
  TYPE  items_aa IS TABLE OF VARCHAR2(60)
    INDEX BY PLS_INTEGER;
  ...
END globals;
```

```
CREATE OR REPLACE FUNCTION get_order_counts ....
  l_items_aa    globals.items_aa;
BEGIN
  l_items_aa(1)  := 'Treadmill';
  ...
END get_order_counts;
```

Session Persistance

Package specification

```
CREATE OR REPLACE PACKAGE globals IS
  TYPE items_aa IS TABLE OF VARCHAR2(60)
    INDEX BY PLS_INTEGER;
  g_items_aa  items_aa;
  …
END globals;
```

```
CREATE OR REPLACE PROCEDURE set_items  IS
  BEGIN
  globals.g_items_aa(1)  := 'Treadmill';
  …
END set_items;
```

```
CREATE OR REPLACE PROCEDURE get_items  IS
 …
 BEGIN
  …
  l_value :=  globals.g_items_aa(1);
  …
END get_items;
```

Data Exchange

```
CREATE OR REPLACE PACKAGE globals IS
  TYPE items_aa IS TABLE OF VARCHAR2(60)
    INDEX BY PLS_INTEGER;
  …
END globals;
```

```
CREATE OR REPLACE FUNCTION init RETURN globals.items_aa  IS
 ….
  l_items_aa globals.items_aa;
  BEGIN
  l_items_aa(1)  := 'Treadmill';
  l_items_aa(2)  := 'Bike';
  …
  RETURN l_items_aa;
END init;
```

```
DECLARE
   l_items_aa globals.items_aa := init;
  BEGIN
   …
   DBMS_OUTPUT.PUT_LINE(l_items_aa(2));
   …
END ;
```

Oracle-Supplied Arrays

DBMS_UTILITY

NAME_ARRAY

```
TYPE name_array IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

NUMBER_ARRAY

```
TYPE number_array IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
```

DBMS_SQL

VARCHAR2A

```
TYPE varchar2a IS TABLE OF VARCHAR2(32767) INDEX BY BINARY_INTEGER;
```

DATE_TABLE

```
TYPE date_table IS TABLE OF DATE INDEX BY BINARY_INTEGER;
```

Oracle-Supplied Arrays

```
DECLARE

  l_numbers_aa   dbms_utility. number_array;

  l_date_aa        dbms_sql. date_table;

BEGIN

  l_date_aa(1) := SYSDATE;
  …

END;
```

Comparing Associative Arrays

Cannot directly compare

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;

 l_items_first_aa        items_aa;
 l_items_second_aa   items_aa;

BEGIN

 l_Items_first_aa(1) := 'Treadmill';
 l_Items_first_aa(2) := 'Bike';

 l_items_second_aa(1)  := 'Treadmill';
 l_items_second_aa(2)  := 'Bike';
 IF l_Items_first_aa = l_items_second_aa THEN          ✗
 ...
 END IF;
END;
```

Usage Guidelines

Small Lookup tables

Passing collection to and from database server

Summary

What is an associative array?

Define and use

Associative array index

Where can they be declared?

Usage guidelines

Next up.. Collection Methods

Collection Methods

Pankaj Jain

@twit_pankajj

Module Overview

COUNT

DELETE

EXISTS

LAST

Module Overview

Iterating

EXTEND

TRIM

PRIOR

Collection Methods

```
<collection_variable>.<collection_method>
```

LAST

Last index in collection

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;
 l_last_index            BINARY_INTEGER;
BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 l_last_index := l_items_aa.LAST;    ->    20

END;
```

EXISTS(n)

Check for existence of index counter

Returns boolean value

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;

BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 IF  l_items_aa.EXISTS(1)  THEN
    DBMS_OUTPUT.PUT_LINE(l_items_aa(1));
 END IF;
END;
```

FALSE

ORA-01403:
no data found

COUNT

Returns count of elements

```
DECLARE
  TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa            items_aa;


BEGIN
  l_Items_aa(-10) := 'Treadmill';
  l_Items_aa(20) := 'Bike';
  l_items_aa(3) := 'Elliptical';

  DBMS_OUTPUT.PUT_LINE(l_items_aa.COUNT);        →  3

  DBMS_OUTPUT.PUT_LINE(l_items_aa.LAST);         →  20

END;
```

DELETE

Removes all elements in a collection

VARRAY allows delete without any arguments

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa                items_aa;

BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 l_items_aa.DELETE;

DBMS_OUTPUT.PUT_LINE(l_items_aa.COUNT);

END;
```

$\xrightarrow{}$ $\boxed{0}$

DELETE(n)

Removes Elements at Index n

Null n Does Nothing

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa            items_aa;

BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20) := 'Bike';
 l_items_aa(3) := 'Elliptical';

 l_items_aa.DELETE(20);

 IF l_items_aa.EXISTS(20) THEN        → FALSE
  DBMS_OUTPUT.PUT_LINE(l_items_aa(20));
 END IF;
END;
```

DELETE(m,n)

Removes elements from m to n

m and n inclusive

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;

BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20)  := 'Bike';
 l_items_aa(25)  := 'Elliptical';
 l_items_aa(27)  := 'Weights';

 l_items_aa.DELETE(20,27);

 DBMS_OUTPUT.PUT_LINE(l_items_aa.COUNT);

END;
```

$\longrightarrow$ 1

PRIOR(n)

Gets prior index counter

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;
 l_prior_index           BINARY_INTEGER;
BEGIN
 l_Items_aa(-10) := 'Treadmill';
 l_Items_aa(20)  := 'Bike';
 l_items_aa(25)  := 'Elliptical';
 l_items_aa(27)  := 'Weights';

 l_prior_index:=  l_items_aa.PRIOR(25);

 DBMS_OUTPUT.PUT_LINE(l_prior_index);

END;
```

$\xrightarrow{\quad}$ 20

TRIM

Nested Tables	Varrays

TRIM

Removes one element from the end

TRIM(n)

Removes n elements from the end

Can raise subscript beyond count exception

EXTEND

Adds one null element at the end of collection

EXTEND(n)

Adds n null elements at the end of collection

EXTEND(n,i)

Adds n elements with copy of element i at the end of collection

Iterating a Collection

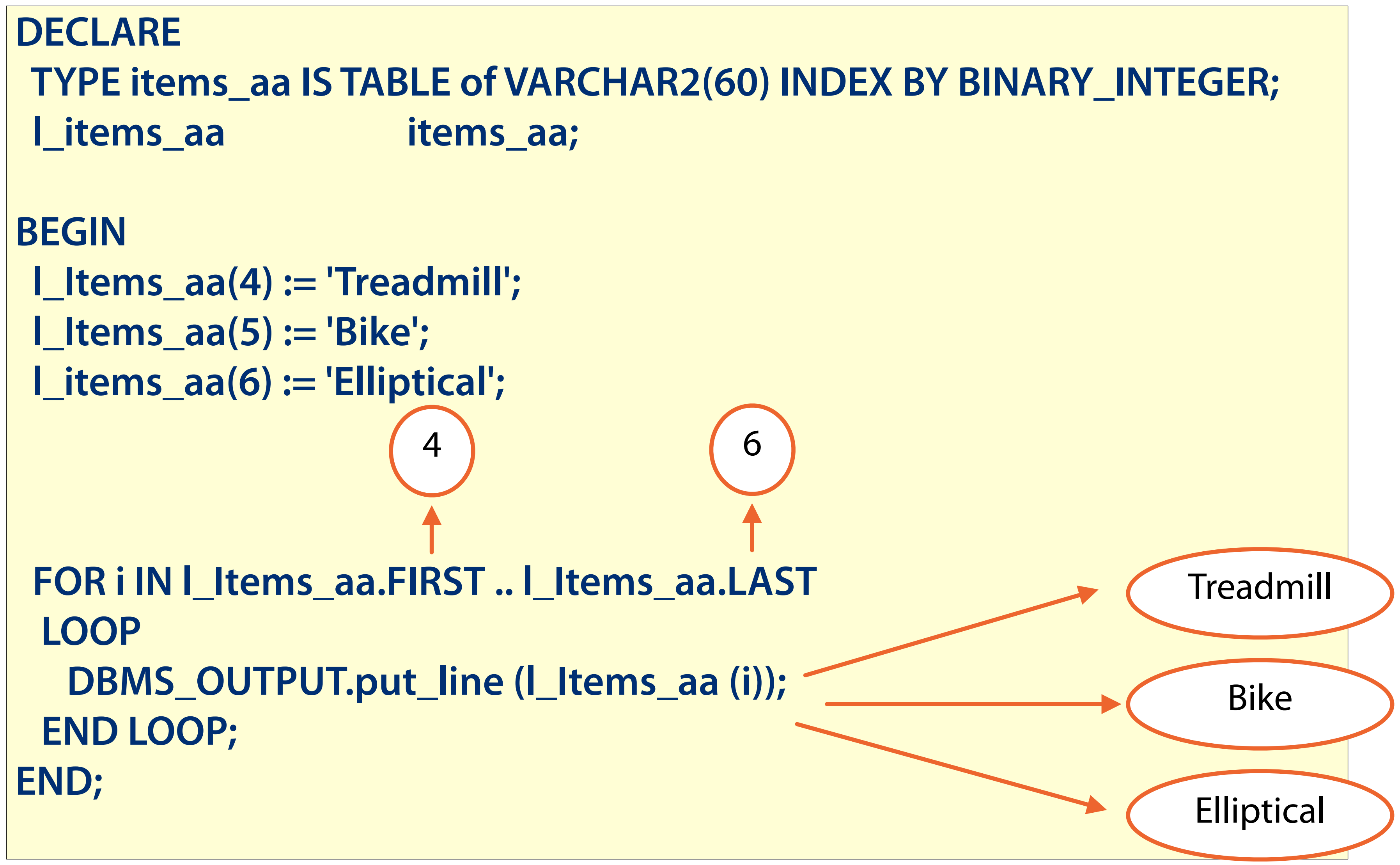
FOR loop

WHILE loop

FOR loop

Dense collection

Access all elements



```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;

BEGIN
 l_Items_aa(4) := 'Treadmill';
 l_Items_aa(5) := 'Bike';
 l_items_aa(6) := 'Elliptical';

 FOR i IN l_Items_aa.FIRST .. l_Items_aa.LAST
  LOOP
    DBMS_OUTPUT.put_line (l_Items_aa (i));
  END LOOP;
END;
```

FOR loop

Sparse collection

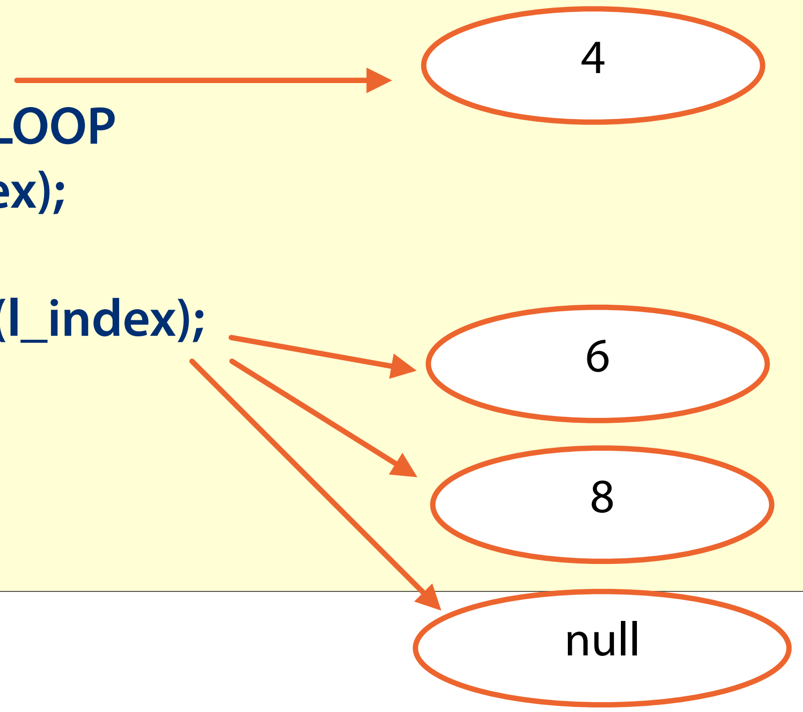
```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa              items_aa;

BEGIN
  l_Items_aa(4) := 'Treadmill';
  l_items_aa(6) := 'Bike';
  l_items_aa(8) := 'Elliptical';
                       4              8                  5
                       ↑              ↑                  ↑

  FOR i IN l_Items_aa.FIRST .. l_Items_aa.LAST
   LOOP
      IF l_items_aa.EXISTS(i) THEN
        DBMS_OUTPUT.put_line (l_Items_aa (i));
      END IF;
   END LOOP;
END;
```

ORA-01403:
no data found

WHILE loop

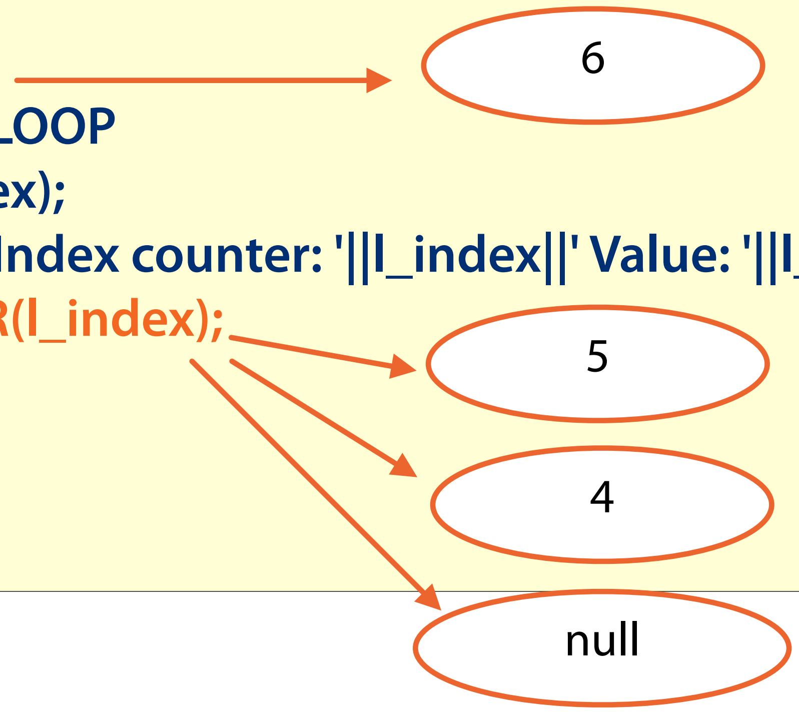
Sparse collection

```
DECLARE
  TYPE items_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa  items_aa;
  l_index       BINARY_INTEGER;
  l_value       VARCHAR2(60);
BEGIN
  l_Items_aa(4) := 'Treadmill';
  l_items_aa(6) := 'Bike';
  l_items_aa(8) := 'Elliptical';
  l_index := l_items_aa.FIRST;
  WHILE  l_index IS NOT NULL LOOP
    l_value := l_items_aa(l_index);

    l_index := l_items_aa.NEXT(l_index);

  END LOOP;
END;
```



WHILE REVERSE loop

```
DECLARE
  TYPE items_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;
  l_items_aa  items_aa;
  l_index       BINARY_INTEGER;
  l_value       VARCHAR2(60);
BEGIN
  l_Items_aa(4) := 'Treadmill';
  l_Items_aa(5) := 'Bike';
  l_items_aa(6) := 'Elliptical';
  l_index := l_items_aa.LAST;
  WHILE  l_index IS NOT NULL LOOP
    l_value := l_items_aa(l_index);
    DBMS_OUTPUT.PUT_LINE('Index counter: '||l_index||' Value: '||l_value);
    l_index := l_items_aa.PRIOR(l_index);
  END LOOP;
END;
```



FOR REVERSE loop

```
DECLARE
 TYPE items_aa IS TABLE of VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_items_aa              items_aa;

BEGIN
 l_Items_aa(4) := 'Treadmill';
 l_Items_aa(5) := 'Bike';
 l_items_aa(6) := 'Elliptical';
                          4               6


 FOR i IN REVERSE l_Items_aa.FIRST .. l_Items_aa.LAST
  LOOP
    DBMS_OUTPUT.put_line (i);
    DBMS_OUTPUT.put_line (l_Items_aa (i));
  END LOOP;
END;
```

```
6
5
4
```

Summary

Collection Methods

Iterating Collections

Next up.. Nested Tables

Nested Tables

Pankaj Jain

@twit_pankajj

Module Overview

Define & Use

EXTEND & TRIM methods

Exceptions

Schema nested tables

Where Can They Be Declared?

PL/SQL

- Anonymous blocks
- Stored subprograms

Database Level

Defining Nested Tables

PL/SQL

```
TYPE <type_name> IS TABLE OF <element_type> [NOT NULL] ;
```

```
TYPE mytype_nt IS TABLE OF NUMBER;

TYPE mytype_nt IS TABLE OF VARCHAR2(60) NOT NULL;

TYPE mytype_nt IS TABLE OF customers%ROWTYPE;
```

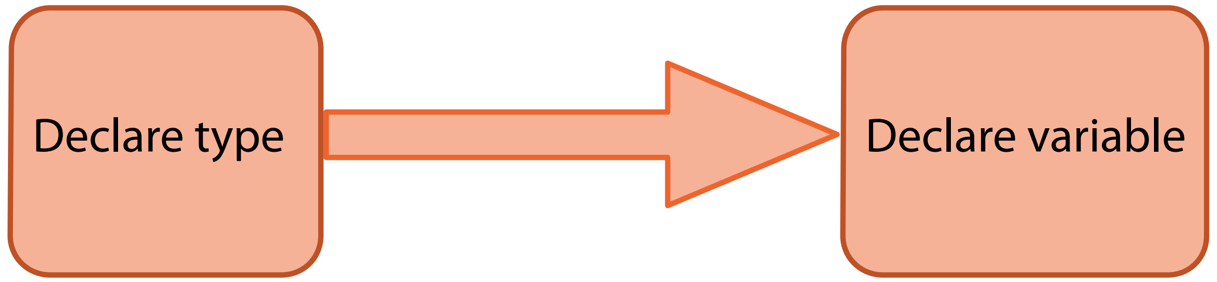
Defining Nested Tables

SQL

```
CREATE [OR REPLACE] TYPE <type_name> IS/AS TABLE OF <element_type> [NOT NULL] ;
```

```
CREATE OR REPLACE TYPE mytype_nt IS TABLE OF NUMBER;

CREATE OR REPLACE TYPE  mytype_nt IS TABLE OF VARCHAR2(60) NOT NULL;
```

Declaring Variables



```
CREATE OR REPLACE TYPE  items_nt IS TABLE OF VARCHAR2(60) NOT NULL;
```

```
DECLARE
  TYPE  items_nt IS TABLE OF VARCHAR2(60) NOT NULL;
  l_items_nt  items_nt;
```

Atomically null array

```
DECLARE

  l_items_nt  items_nt;
```

Atomically null array

Initializing Nested Tables

```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60) ;
  l_items_nt      items_nt;
BEGIN
  DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
END;
```

ORA-06531: Reference to uninitialized collection

Initializing Nested Tables

Constructor

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt;

BEGIN
 l_items_nt := items_nt('Bike', 'Treadmill');
 DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
END;
```

2

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt('Bike', 'Treadmill');

BEGIN

 DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
END;
```

Initializing Nested Tables

Constructor without arguments

```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60) ;
  l_items_nt     items_nt := items_nt();

BEGIN

  DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
END;
```

Initialized as an empty array

0

Nested Table Index

Integer

Starts with 1

Upper limit 2147483647

Adding Elements

Extend method

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(l_items_nt.LAST) := 'Bike';

  l_items_nt.EXTEND;
  l_items_nt(l_items_nt.LAST) := 'Treadmill';

 DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
END;
```

```
2
```

Adding Elements

Extend method

```
DECLARE
 TYPE items_rec IS RECORD( item_name  items.item_name%TYPE,
                                         count NUMBER);
 TYPE items_nt IS TABLE of items_rec ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(l_items_nt.LAST).item_name := 'Bike';
  l_items_nt(l_items_nt.LAST).count := 1;


  l_items_nt.EXTEND;
  l_items_nt(l_items_nt.LAST).item_name := 'Treadmill';
  l_items_nt(l_items_nt.LAST).count := 2;

DBMS_OUTPUT.PUT_LINE( l_items_nt(1).item_name);
END;
```

Adding Elements

Extend method

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt      items_nt := items_nt();
CURSOR get_items IS
    SELECT *
       FROM items;
BEGIN
  FOR get_items_var IN get_items LOOP
    l_items_nt.EXTEND;
    l_items_nt(l_items_nt.LAST) := get_items_var.item_name;
  END LOOP;
END;
```

Adding Elements

Extend(n)

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND(2);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';

END;
```

Adding Elements

Extend(n,i)

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND(2);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';
 l_items_nt.EXTEND(2,1);

 DBMS_OUTPUT.PUT_LINE( l_items_nt(3));
 DBMS_OUTPUT.PUT_LINE( l_items_nt(4));
END;
```

```
Bike
Bike
```

Deleting Elements

Can Be Sparse

DELETE(n)

DELETE

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND(3);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';
  l_items_nt(3) := 'Elliptical';

 l_items_nt.DELETE(2);
 DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);

 l_items_nt.DELETE;
 DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);

END;
```

2

0

Nested Table Index

Reassigning Overwrites Previous Value at that Index

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) ;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND(3);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';
  l_items_nt(1) := 'Elliptical';

 DBMS_OUTPUT.PUT_LINE(l_items_nt(1));

END;
```

Elliptical

Assigning Value to Nested Tables

Assigning Another Nested Table

```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_items_nt      items_nt := items_nt();
  l_copy_nt       items_nt;
BEGIN
  l_Items_nt.EXTEND(3);
  l_Items_nt(1) := 'Treadmill';
  l_Items_nt(2) := 'Bike';
  l_items_nt(3) := 'Elliptical';

  l_copy_nt := l_items_nt;

 dbms_output.put_line(l_copy_nt(2));
END;
```

Bike

Assigning Value to Nested Tables

Same Type

```
DECLARE
  TYPE items_nt    IS TABLE of VARCHAR2(60);
  TYPE dup_nt      IS TABLE of VARCHAR2(60) ;
  l_items_nt        items_nt;
  l_dup_nt          dup_nt;
BEGIN
  l_Items_nt.EXTEND(3);
  l_Items_nt(1) := 'Treadmill';
  l_Items_nt(2) := 'Bike';
  l_items_nt(3) := 'Elliptical';

  l_dup_nt := l_items_nt;  ✗

END;
```

Assigning Value to Nested Tables

Assigning Empty Array

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt      items_nt := items_nt();
 l_copy_nt       items_nt;
BEGIN
 l_Items_nt.EXTEND(3);
 l_Items_nt(1) := 'Treadmill';
 l_Items_nt(2) := 'Bike';
 l_items_nt(3) := 'Elliptical';

 l_items_nt := l_copy_nt ;

END;
```

Exceptions During Assignment

Not null constraint

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60) NOT NULL;
 l_items_nt     items_nt := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := NULL;
END;
```

PLS-00382: expression is of wrong type

Exceptions During Assignment

Uninitialized collection

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt ;

BEGIN
  l_items_nt(1) := 'Bike';
END;
```

ORA-06531: Reference to uninitialized collection

Exceptions During Assignment

Subscript beyond count

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt(1) := 'Bike';
EXCEPTION
  WHEN SUBSCRIPT_BEYOND_COUNT THEN
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
 END;
```

ORA-06533: Subscript beyond count

Exceptions During Assignment

No data found

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Treadmill';
  l_items_nt.DELETE(1);
  DBMS_OUTPUT.PUT_LINE(l_items_nt(1));
EXCEPTION
 WHEN  NO_DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
     RAISE;
END;
```

ORA-01403: no data found

Exceptions During Assignment

Value Error

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(4);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Treadmill';
EXCEPTION
 WHEN VALUE_ERROR THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
END;
```

ORA-06502: PL/SQL numeric or value error

Exceptions During Assignment

Value Error

```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_items_nt      items_nt  := items_nt();

BEGIN
   l_items_nt.EXTEND;
   l_items_nt('A') := 'Treadmill';
EXCEPTION
  WHEN VALUE_ERROR THEN
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
      RAISE;
END;
```

ORA-6502: PL/SQL numeric or value error

Exceptions During Assignment

Subscript outside of limit

1 to 2147483647

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(0) := 'Treadmill';
EXCEPTION
 WHEN SUBSCRIPT_OUTSIDE_LIMIT THEN
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
END;
```

ORA-6532:
Subscript outside
of limit

Reducing size

TRIM

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Bike';
  l_items_nt.EXTEND;
  l_items_nt(2) := 'Treadmill';
  l_items_nt.TRIM;
  DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);
END;
```

```
1
```

Reducing size

Works on inner collection size

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Bike';
  l_items_nt.EXTEND;
  l_items_nt(2) := 'Treadmill';
  l_items_nt.DELETE(2);
  l_items_nt.TRIM;
  DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);
END;
```

1

Reducing size

TRIM(n)

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Bike';
  l_items_nt.EXTEND;
  l_items_nt(2) := 'Treadmill';
  l_items_nt.TRIM(2);
  DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);
END;
```

0

Reducing size

Subscript beyond count

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt     items_nt  := items_nt();

BEGIN
  l_items_nt.EXTEND;
  l_items_nt(1) := 'Bike';
  l_items_nt.EXTEND;
  l_items_nt(2) := 'Treadmill';
  l_items_nt.TRIM(3);
  DBMS_OUTPUT.PUT_LINE(l_items_nt.COUNT);
END;
```

ORA-06533: Subscript beyond count

Schema Level Nested Tables

- Available throughout the system
- Columns of database tables
- Easier information retrieval

Interacting with Schema Nested Tables

```
CREATE OR REPLACE TYPE items_nt AS TABLE OF VARCHAR2(60);
```

```
CREATE TYPE orders_ot AS OBJECT (order_id NUMBER, order_item_id NUMBER);

CREATE OR REPLACE TYPE  orders_nt IS TABLE OF orders_ot;
```

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt    DEFAULT items_nt(),
  orderslist    orders_nt  DEFAULT orders_nt())
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
```

Dropping Schema Nested Tables

```
DROP  TYPE  <type_name> [FORCE | VALIDATE];
```

```
DROP TYPE mytype_nt;
```

ORA-2303: cannot drop or replace a type with type or table dependents

```
DROP TYPE mytype_nt FORCE;
```

Altering the size of a Schema Nested Table

```
ALTER TYPE <NESTED_TABLE>  MODIFY ELEMENT TYPE  <new_datatype_size> CASCADE | INVALIDATE;
```

```
CREATE TYPE items_nt AS TABLE OF VARCHAR2(60);
```

```
ALTER TYPE items_nt MODIFY ELEMENT TYPE VARCHAR2(100) CASCADE;
```

```
ALTER TYPE items_nt MODIFY ELEMENT TYPE VARCHAR2(10) CASCADE;
```

PLS-00729: only widening of the collection element type is allowed

Inserting

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
INSERT INTO
    account_orders (act_id,
                    act_month,
                    itemslist,
                    orderslist)
     VALUES        ( 1,
                   'JANUARY',
                   items_nt('Bike', 'Treadmill'),
                   orders_nt(orders_ot(1,1),orders_ot(2,2));
```

PL/SQL

```
DECLARE
  l_items_nt     items_nt := items_nt();
  l_orders_nt    orders_nt:= orders_nt();
  l_orders_ot    orders_ot := orders_ot(1,1);

BEGIN
  l_items_nt.EXTEND(2);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';

  l_orders_nt.EXTEND(2);
  l_orders_nt(1) := l_orders_ot;
  l_orders_nt(2) := orders_ot(2,2);
  INSERT INTO  account_orders (act_id, act_month,itemslist,     orderslist)
    VALUES                            (  1, 'JANUARY',     l_items_nt, l_orders_nt);
  COMMIT;
END;
```

Updating

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
UPDATE account_orders
     SET itemslist =  items_nt('Elliptical'),
          orderslist =  orders_nt(orders_ot(1,1),orders_ot(3,3))
      WHERE  act_id = 1
      AND       act_month =  'JANUARY';
```

PL/SQL

```
DECLARE
  l_items_nt      items_nt := items_nt();
  l_orders_nt    orders_nt:= orders_nt();
  l_orders_ot    orders_ot := orders_ot(1,1);

BEGIN
  l_items_nt.EXTEND(1);
  l_items_nt(1) := 'Elliptical';

  l_orders_nt.EXTEND(2);
  l_orders_nt(1) := l_orders_ot;
  l_orders_nt(2) := orders_ot(3,3);
  UPDATE account_orders SET itemslist = l_items_nt,
                                            orderslist = l_orders_nt
     WHERE  act_id = 1
     AND       act_month =  'JANUARY';
  COMMIT;
END;
```

Deleting

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
UPDATE account_orders
     SET itemslist =  NULL,
          orderslist =  NULL
     WHERE  act_id = 1
     AND       act_month =  'JANUARY';
```

PL/SQL

```
BEGIN
  DELETE FROM account_orders
     WHERE  act_id = 1
     AND       act_month =  'JANUARY';
  COMMIT;
END;
```

```
BEGIN
  UPDATE account_orders SET itemslist = NULL,
                                          orderslist = NULL
     WHERE  act_id = 1
     AND       act_month =  'JANUARY';
  COMMIT;
END;
```

Selecting

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
SELECT * FROM account_orders
WHERE  act_id = 1  AND       act_month =  'JANUARY';

ACT_ID   ACT_MONTH  ITEMSLIST   ORDERSLIST
-------- ---------- ----------- -----------
   1         JANUARY     DEMO.ITEMS_NT('Bike','Treadmill')
```

PL/SQL

```
DECLARE
  l_items_nt     items_nt := items_nt();
  l_orders_nt    orders_nt:= orders_nt();
  CURSOR get_details_cur IS
    SELECT  itemslist, orderslist
      FROM  account_orders
      WHERE  act_id = 1  AND       act_month =  'JANUARY';
 BEGIN
   OPEN  get_details_cur;
   FETCH get_details_cur INTO l_items_nt, l_orders_nt;
   CLOSE get_details_cur;
   IF  l_items_nt IS NOT NULL THEN
     FOR i IN l_items_nt.FIRST .. l_items_nt.LAST  LOOP
       DBMS_OUTPUT.PUT_LINE('Item name '||l_items_nt(i));
     END LOOP;
   END IF;
   IF  l_items_nt IS NOT NULL THEN
    FOR i IN l_orders_nt.FIRST .. l_orders_nt.LAST  LOOP
      DBMS_OUTPUT.PUT_LINE('Item id '||l_orders_nt(i).order_id);
    END LOOP;
   END IF;
END;
```

Selecting

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
SELECT * FROM account_orders
WHERE  act_id = 1  AND       act_month =  'JANUARY';

ACT_ID  ACT_MONTH     ITEMSLIST                              ORDERSLIST
------- ------------        --------- -------------
   1        JANUARY        DEMO.ITEMS_NT('Bike', 'Treadmill')   DEMO.ORDERS_NT(DEMO.ORDERS_OT(1,1),DEMO.ORDERS_OT(2,2))
```

Summary

Declare and initialize

Add and remove elements

Exceptions

Schema level nested tables

DML on nested table columns

Next up.. Comparing nested tables & nested table operators

Nested Tables: Comparison, TABLE & MULTISET Operators

Pankaj Jain

@twit_pankajj

Module Overview

TABLE Expression

Piecewise Operations

MULTISET

Comparing Nested Tables

Other Operators

Selecting

```
CREATE TABLE account_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_nt,
  orderslist    orders_nt)
  NESTED TABLE  itemslist   STORE AS itemlist_store
  NESTED TABLE  orderslist  STORE AS orderslist_store;
)
```

SQL

```
SELECT * FROM account_orders
WHERE  act_id = 1  AND      act_month =  'JANUARY';

ACT_ID  ACT_MONTH     ITEMSLIST                          ORDERSLIST
------- -----------      ------- -----------
   1       JANUARY       DEMO.ITEMS_NT('Bike', 'Treadmill')   DEMO.ORDERS_NT(DEMO.ORDERS_OT(1,1),DEMO.ORDERS_OT(2,2))
```

Unnesting Using TABLE Expression

TABLE(collection column)

```
SELECT a.act_id, b.COLUMN_VALUE FROM  account_orders  a, TABLE(a.itemslist) b WHERE a.act_id =  1;
```

```
ACT_ID  COLUMN_VALUE
------  ------------
  1        Bike
  1        Treadmill
```

```
SELECT a.act_id, b.order_id, b.order_item_id FROM  account_orders  a, TABLE(a.orderslist) b WHERE a.act_id =  1;
```

```
ACT_ID  ORDER_ID  ORDER_ITEM_ID
------  -----------------------
  1      1          1
  1      2          2
```

Unnesting Using TABLE Expression

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
2	MARCH			

```
SELECT a.act_id, b.COLUMN_VALUE FROM  account_orders  a, TABLE(a.itemslist) b;
```

```
ACT_ID  COLUMN_VALUE
------- ------------
  1        Bike
  1        Treadmill
```

Unnesting Using TABLE Expression

Outer join

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
2	MARCH			

```
SELECT a.act_id, b.COLUMN_VALUE FROM  account_orders  a, TABLE(a.itemslist) (+) b;
```

```
ACT_ID  COLUMN_VALUE
------- ------------
  1        Bike
  1        Treadmill
  2        null
```

Unnesting Using TABLE Expression

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
2	MARCH			

```
SELECT a.act_id, b.order_id,b.order_item_id FROM  account_orders  a, TABLE(a.orderslist) b;
```

```
ACT_ID  ORDER_ID      ORDER_ITEM_ID
———— - —————-         ———————————-
  1        1             1
  1        2             2
```

Unnesting Using TABLE Expression

Outer join

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
2	MARCH			

```
SELECT a.act_id, b.order_id,b.order_item_id FROM  account_orders  a, TABLE(a.orderslist) (+) b;
```

```
ACT_ID  ORDER_ID     ORDER_ITEM_ID
------- ---------    -------------
  1       1            1
  1       2            2
  2       null         null
```

TABLE Expression

TABLE(subquery)

```
SELECT a.order_id, a.order_item_id FROM TABLE(SELECT orderslist FROM account_orders WHERE act_id =1) a;
```

Restrictions

- Return a collection type
- Select list should have only one item
- Return a single collection

Piecewise Insert

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical		

PL/SQL

```
BEGIN

  INSERT INTO  TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1)
     VALUES  (  'Elliptical');
  COMMIT;
END;
```

SQL

```
INSERT INTO  TABLE (SELECT itemslist FROM account_orders  WHERE act_id = 1)
     VALUES  (  'Elliptical');
```

Piecewise Insert

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical	3	3

PL/SQL

```
BEGIN

  INSERT INTO  TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)
     VALUES  (3,3);
  COMMIT;
END;
```

SQL

```
INSERT INTO  TABLE (SELECT orderslist FROM account_orders  WHERE act_id = 1)
     VALUES  ( 3,3);
```

Piecewise Update

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
		Bike	1	1
1	JANUARY	Treadmill	2	2
		Elliptical Weights	3	3

PL/SQL

```
BEGIN

  UPDATE TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1)
     SET COLUMN_VALUE = 'Weights'
      WHERE COLUMN_VALUE = 'Elliptical';
  COMMIT;
END;
```

SQL

```
UPDATE TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1)
     SET COLUMN_VALUE = 'Weights'
      WHERE COLUMN_VALUE = 'Elliptical';
```

Piecewise Update

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical	3 4	3 4

PL/SQL

```
BEGIN

  UPDATE TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)
     SET order_id = 4, order_item_id = 4
      WHERE order_id = 3 AND order_item_id = 3;
  COMMIT;
END;
```

SQL

```
UPDATE TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)
     SET order_id = 4, order_item_id = 4
      WHERE order_id = 3 AND order_item_id = 3;
```

Piecewise Update

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
		Bike	1	1
1	JANUARY	Treadmill	2	2
		Elliptical	3 4	3 4

PL/SQL

```
BEGIN

  UPDATE TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)  a
     SET VALUE(a) = orders_ot(4,4)
      WHERE a.order_id = 3 AND a.order_item_id = 3;
  COMMIT;
END;
```

SQL

```
UPDATE TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)  a
     SET VALUE(a) = orders_ot(4,4)
      WHERE a.order_id = 3 AND a.order_item_id = 3;
```

Piecewise Update

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical Weights	3	3

PL/SQL

```
BEGIN

  UPDATE TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1) a
     SET VALUE(a) = 'Weights'
      WHERE COLUMN_VALUE = 'Elliptical';
  COMMIT;
END;
```

SQL

```
UPDATE TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1) a
     SET VALUE(a) = 'Weights'
      WHERE COLUMN_VALUE = 'Elliptical';
```

Piecewise Delete

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical	3	3

PL/SQL

```
BEGIN

  DELETE FROM TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1)
  WHERE COLUMN_VALUE = 'Elliptical';
  COMMIT;
END;
```

SQL

```
DELETE FROM TABLE (SELECT itemslist FROM account_orders WHERE act_id = 1)
  WHERE COLUMN_VALUE = 'Elliptical';
```

Piecewise Delete

ACT_ID	ACT_MONTH	ITEMSLIST	ORDERSLIST	
1	JANUARY	Bike	1	1
		Treadmill	2	2
		Elliptical	3	3

PL/SQL

```
BEGIN

  DELETE FROM TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)
  WHERE order_id = 3;
  COMMIT;
END;
```

SQL

```
DELETE FROM TABLE (SELECT orderslist FROM account_orders WHERE act_id = 1)
  WHERE order_id = 3;
```

MULTISET

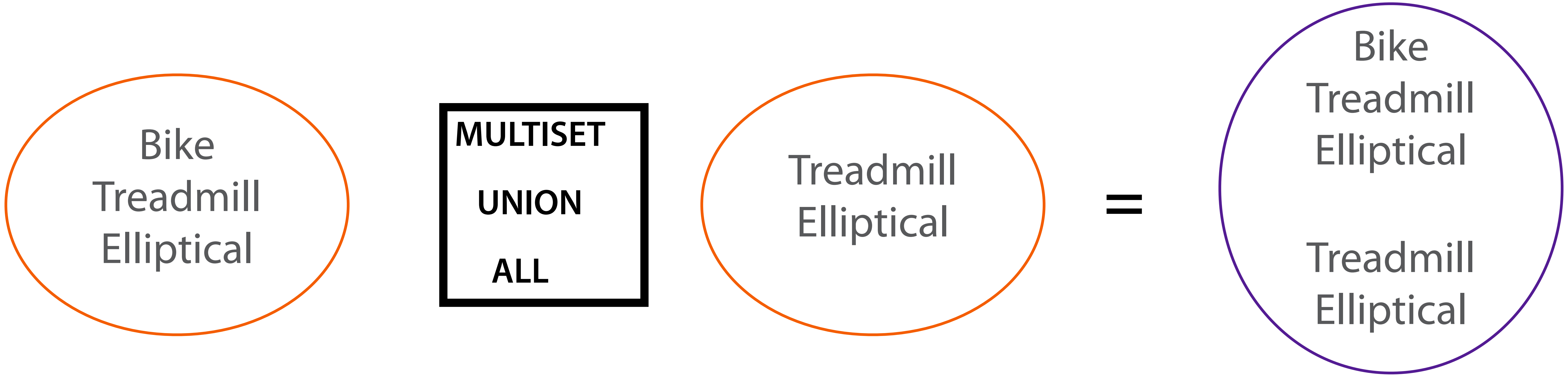
Transforming Nested Tables

Same Nested Table Type

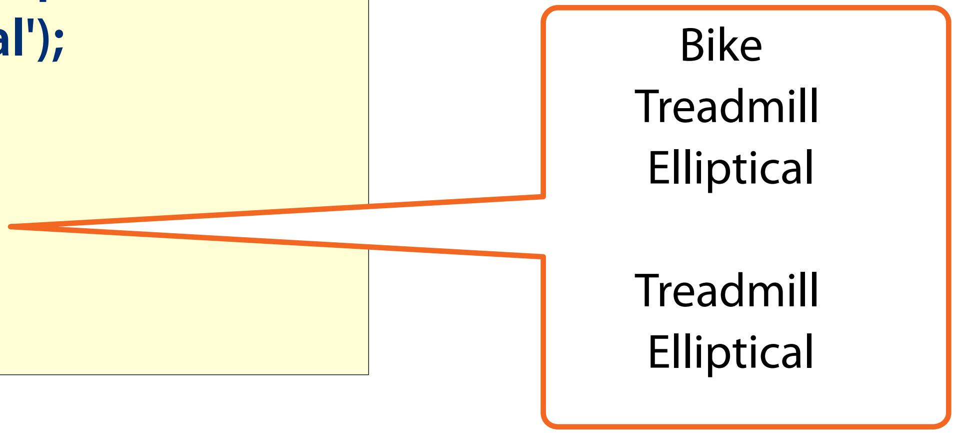
MULTISET Operator	**Eqivalent SQL Operator**
MULTISET UNION	UNION ALL
MULTISET UNION DISTINCT	UNION
MULTISET INTERSECT	INTERSECT
MULTISET EXCEPT	MINUS

MULTISET UNION

```
NESTED_TABLE1  MULTISET  UNION [ ALL | DISTINCT ]  NESTED_TABLE2
```

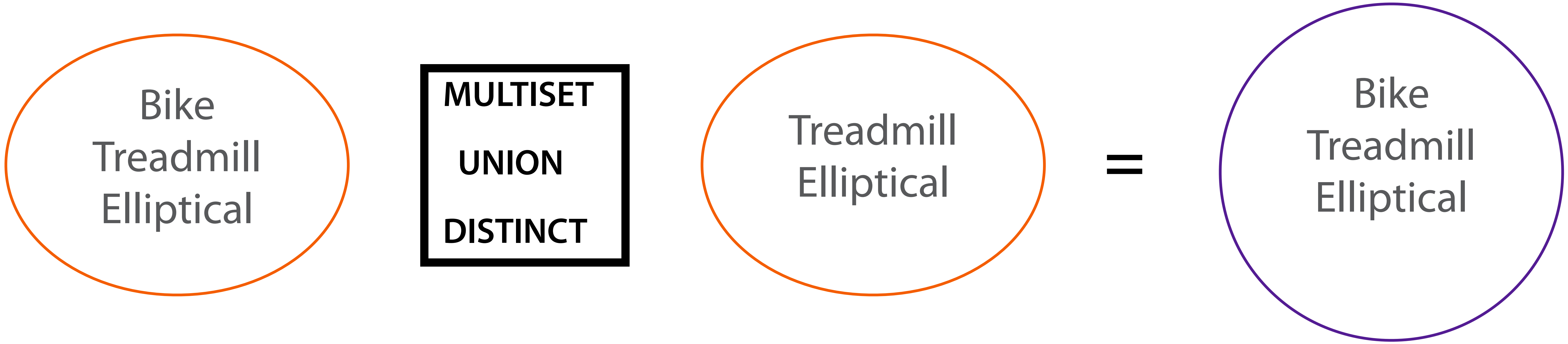


```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt        items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical');
 l_second_nt       items_nt  := items_nt('Treadmill', 'Elliptical');
 l_final_nt        items_nt;
BEGIN
  l_final_nt := l_first_nt MULTISET UNION l_second_nt;
END;
```



MULTISET UNION DISTINCT

NESTED_TABLE1 MULTISET UNION [ALL | DISTINCT] NESTED_TABLE2



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt          items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical');
 l_second_nt         items_nt  := items_nt('Treadmill', 'Elliptical');
 l_final_nt          items_nt;
BEGIN
  l_final_nt := l_first_nt MULTISET UNION DISTINCT l_second_nt;
END;
```

Bike
Treadmill
Elliptical

Interacting with Schema Nested Tables

```
CREATE OR REPLACE TYPE items_nt AS TABLE OF VARCHAR2(60);
```

```
CREATE TABLE item_orders (
  act_month VARCHAR2(8),
  store1_items     items_nt   DEFAULT items_nt(),
  store2_items     items_nt   DEFAULT items_nt())
  NESTED TABLE  store1_items   STORE AS store1
  NESTED TABLE  store2_items   STORE AS store2;
```

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Treadmill	Elliptical
	Elliptical	

MULTISET UNION

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Treadmill	Elliptical
	Elliptical	

SQL

```
SELECT store1_items
          MULTISET UNION
          store2_items
FROM item_orders
 WHERE act_month = 'JANUARY';
```

PL/SQL

```
DECLARE
 l_final_nt        items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET UNION store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

```
Bike
Treadmill
Elliptical

Treadmill
Elliptical
```

MULTISET UNION DISTINCT

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Treadmill	Elliptical
	Elliptical	

SQL

```
SELECT store1_items
          MULTISET UNION DISTINCT
          store2_items
FROM item_orders
 WHERE act_month = 'JANUARY';
```

PL/SQL

```
DECLARE
 l_final_nt          items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET UNION DISTINCT store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

```
Bike
Treadmill
Elliptical
```

MULTISET UNION DISTINCT

SQL

```
SELECT store1_items
           MULTISET UNION DISTINCT
           store2_items
FROM item_orders
 WHERE act_month = 'JANUARY';
```

```
————————————————————————-
DEMO.ITEMS_NT(Bike,Treadmill,Elliptical)
```

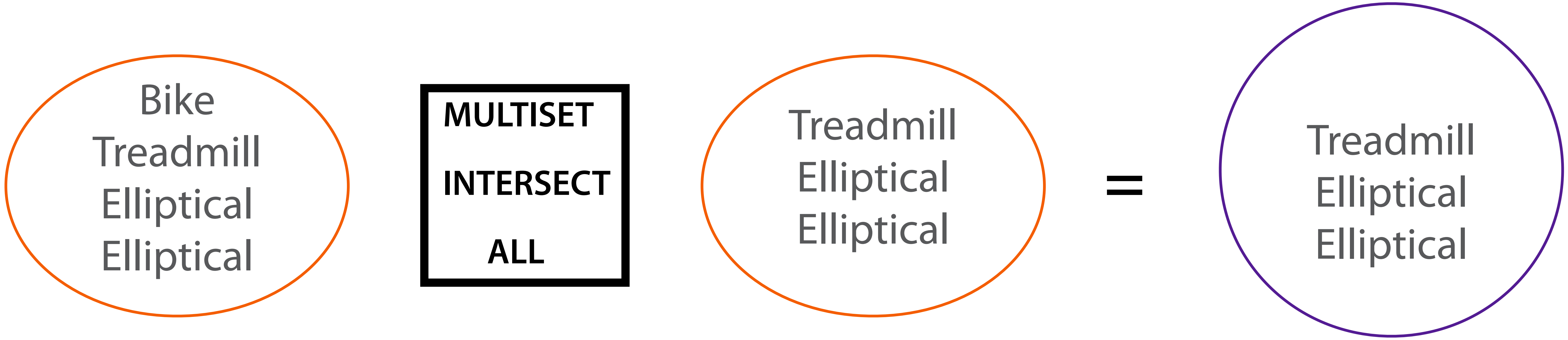
```
 SELECT COLUMN_VALUE
   FROM item_orders a,
   TABLE( a.store1_items
             MULTISET UNION DISTINCT
            a.store2_items) b
WHERE a.act_month = 'JANUARY';
```

```
SELECT COLUMN_VALUE
  FROM  TABLE(SELECT  store1_items
                                   MULTISET UNION DISTINCT
                                 store2_items
                       FROM   item_orders
 WHERE act_month = 'JANUARY');
```

```
———————
Bike
Treadmill
Elliptical
```

MULTISET INTERSECT ALL

NESTED_TABLE1 MULTISET INTERSECT [ALL | DISTINCT] NESTED_TABLE2



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt         items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical', 'Elliptical');
 l_second_nt      items_nt  := items_nt('Treadmill', 'Elliptical', 'Elliptical');
 l_final_nt         items_nt;
BEGIN
  l_final_nt := l_first_nt MULTISET INTERSECT l_second_nt;
END;
```

Treadmill
Elliptical
Elliptical

MULTISET INTERSECT ALL

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Treadmill	Treadmill
	Elliptical	Elliptical
	Elliptical	Elliptical
	Bike	

SQL

```
SELECT store1_items
          MULTISET INTERSECT ALL
          store2_items
 FROM item_orders
  WHERE act_month = 'JANUARY';
```

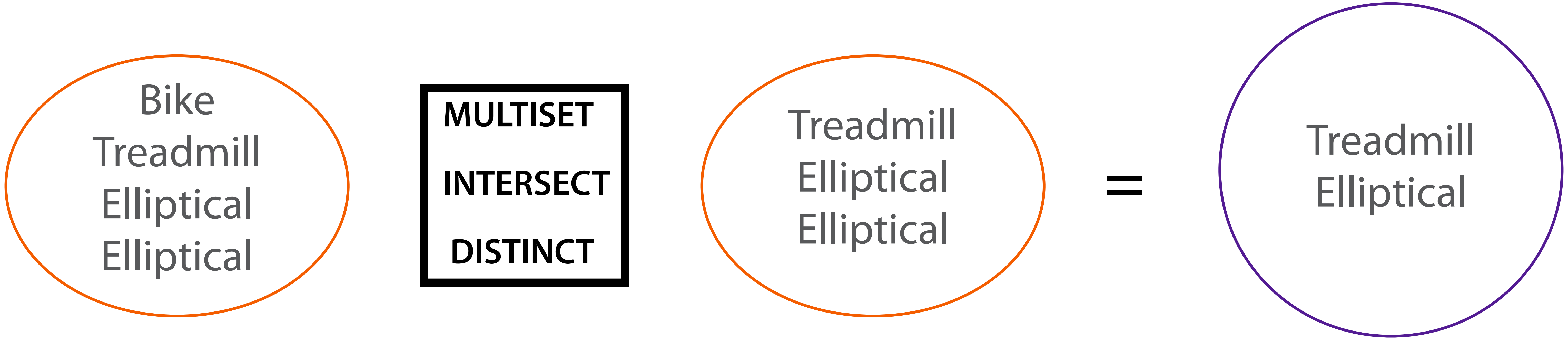
PL/SQL

```
DECLARE
 l_final_nt          items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET INTERSECT ALL store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

```
Treadmill
Elliptical
Elliptical
```

MULTISET INTERSECT DISTINCT

NESTED_TABLE1 MULTISET INTERSECT [ALL | DISTINCT] NESTED_TABLE2



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt        items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical', 'Elliptical');
 l_second_nt     items_nt  := items_nt('Treadmill', 'Elliptical', 'Elliptical');
 l_final_nt         items_nt;
BEGIN
  l_final_nt := l_first_nt MULTISET INTERSECT DISTINCT l_second_nt;
END;
```

Treadmill
Elliptical

MULTISET INTERSECT DISTINCT

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Treadmill	Treadmill
	Elliptical	Elliptical
	Elliptical	Elliptical
	Bike	

SQL

```
SELECT store1_items
          MULTISET INTERSECT DISTINCT
          store2_items
 FROM item_orders
  WHERE act_month = 'JANUARY';
```

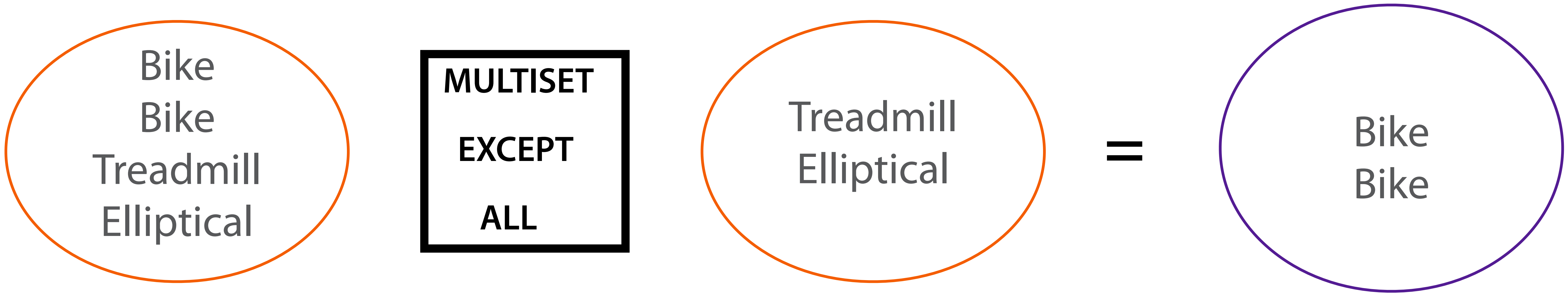
PL/SQL

```
DECLARE
 l_final_nt        items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET INTERSECT DISTINCT store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

Treadmill
Elliptical

MULTISET EXCEPT ALL

NESTED_TABLE1 MULTISET EXCEPT [ALL | DISTINCT] NESTED_TABLE2



```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_first_nt          items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
  l_second_nt      items_nt  := items_nt('Treadmill', 'Elliptical');
  l_final_nt          items_nt;
BEGIN
   l_final_nt := l_first_nt MULTISET EXCEPT l_second_nt;
END;
```

Bike
Bike

MULTISET EXCEPT ALL

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

```
SELECT store1_items
          MULTISET EXCEPT ALL
          store2_items
 FROM item_orders
  WHERE act_month = 'JANUARY';
```

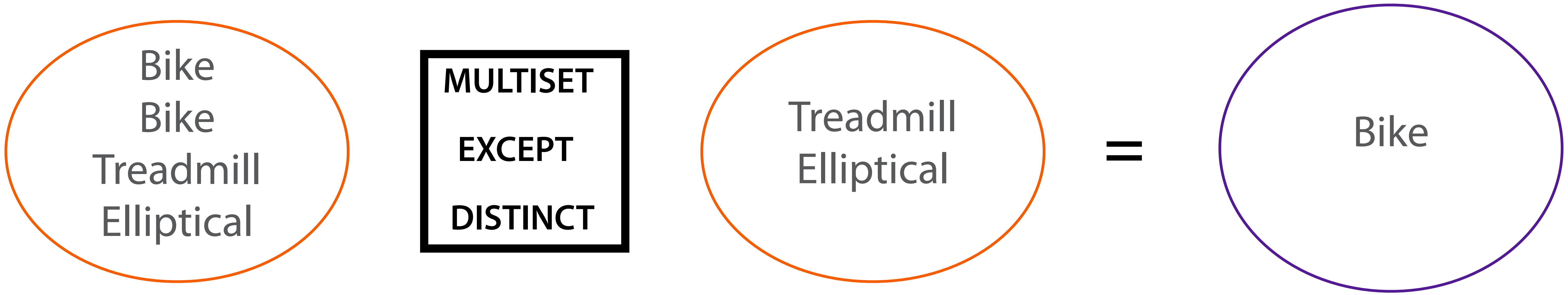
PL/SQL

```
DECLARE
 l_final_nt       items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET EXCEPT ALL store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

```
Bike
Bike
```

MULTISET EXCEPT DISTINCT

```
NESTED_TABLE1  MULTISET  EXCEPT [ ALL | DISTINCT ]  NESTED_TABLE2
```



```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_first_nt        items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
  l_second_nt       items_nt  := items_nt('Treadmill', 'Elliptical');
  l_final_nt        items_nt;
BEGIN
   l_final_nt := l_first_nt MULTISET EXCEPT  DISTINCT l_second_nt;
END;
```

Bike

MULTISET EXCEPT DISTINCT

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

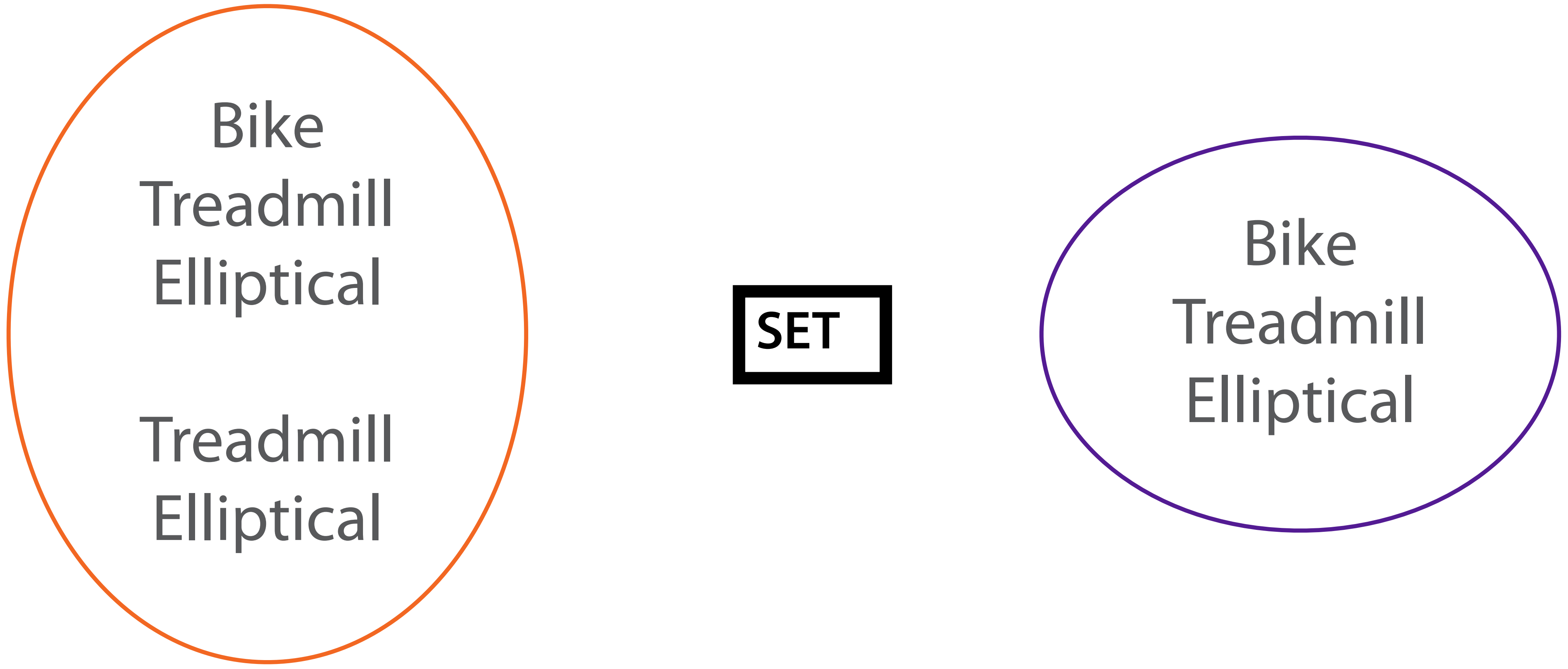
```
SELECT store1_items
          MULTISET EXCEPT DISTINCT
          store2_items
 FROM item_orders
 WHERE act_month = 'JANUARY';
```

PL/SQL

```
DECLARE
 l_final_nt         items_nt;
 CURSOR cur_get_items IS
  SELECT store1_items MULTISET EXCEPT DISTINCT store2_items
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

Bike

SET



```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_first_nt          items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical', 'Treadmill', 'Elliptical');
  l_final_nt          items_nt;
BEGIN
  l_final_nt :=  SET(l_first_nt);
END;
```

Bike
Treadmill
Elliptical

SET

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

```
SELECT SET(store1_items)
 FROM item_orders
  WHERE act_month = 'JANUARY';
```

PL/SQL

```
DECLARE
 l_final_nt          items_nt;
 CURSOR cur_get_items IS
  SELECT SET(store1_items)
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
END;
```

```
Bike
Treadmill
Elliptical
```

Comparing Nested Tables

Compare for equality or inequality

IS NULL

IS EMPTY

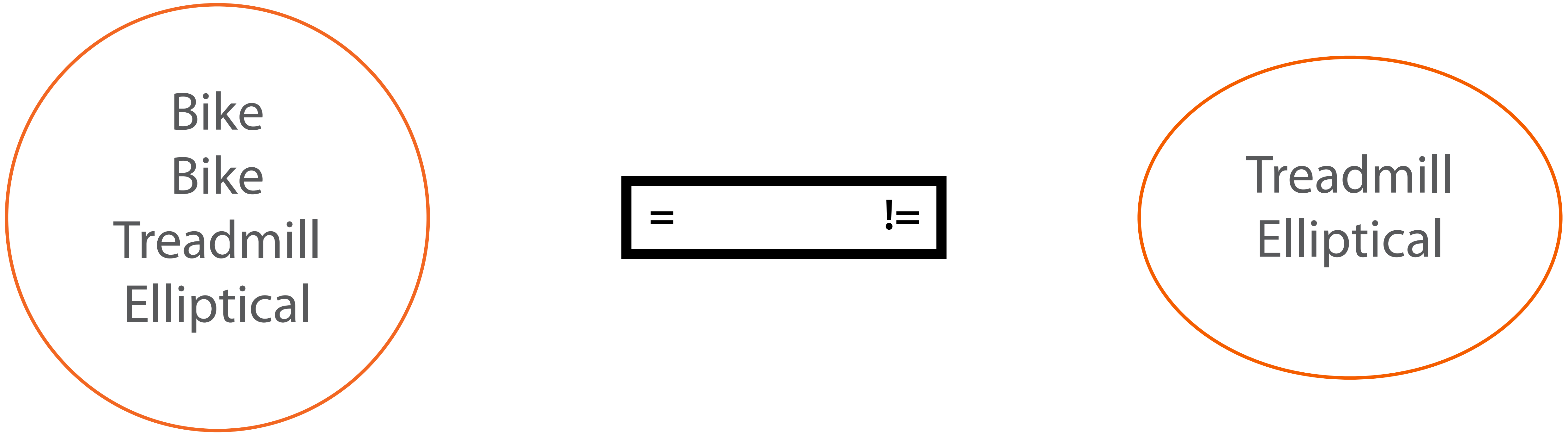
IS [NOT] A SET

CARDINALITY

MEMBER OF

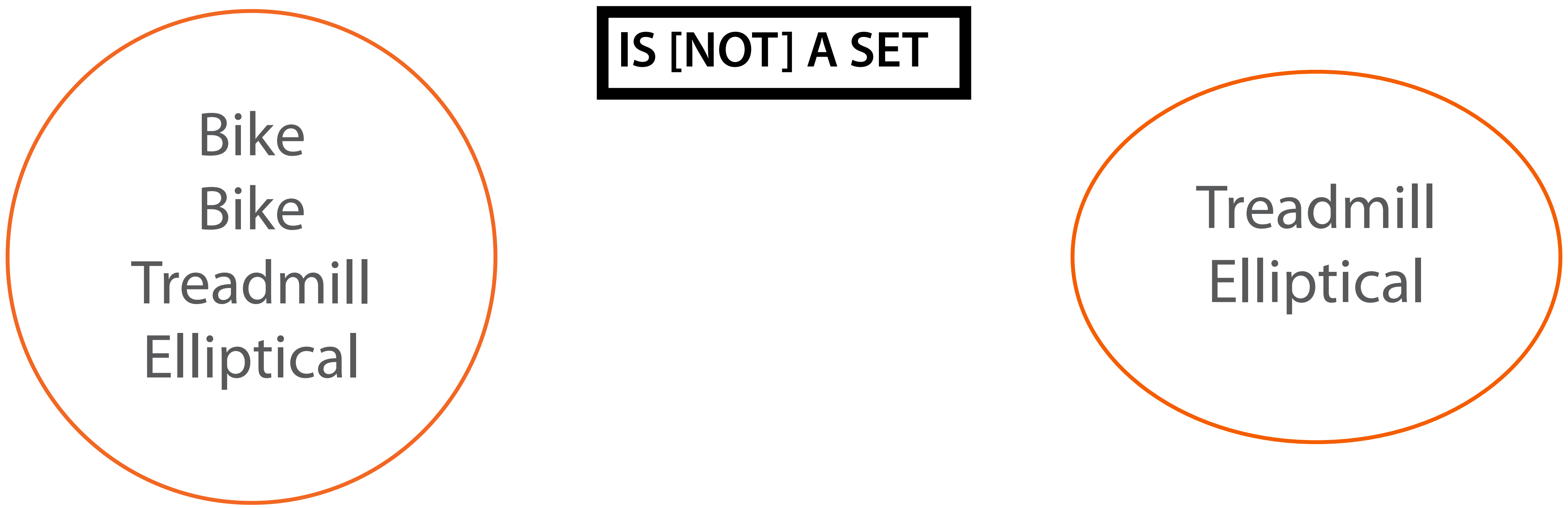
SUBMULTISET

Compare for (In)Equality



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt        items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
 l_second_nt      items_nt  := items_nt('Treadmill', 'Elliptical');
BEGIN
  IF l_first_nt  =  l_second_nt THEN
     dbms_output.put_line('Equal');
  END IF;
  IF l_first_nt  !=  l_second_nt THEN
     dbms_output.put_line('Not Equal');
  END IF;
END;
```

IS [NOT] A SET



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt         items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
 l_second_nt        items_nt  := items_nt('Treadmill', 'Elliptical');
 isaset             boolean        items_nt;
BEGIN
  isaset := l_first_nt       IS NOT A SET;      TRUE

  isaset := l_second_nt  IS A SET;              TRUE
END;
```

IS [NOT] A SET

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

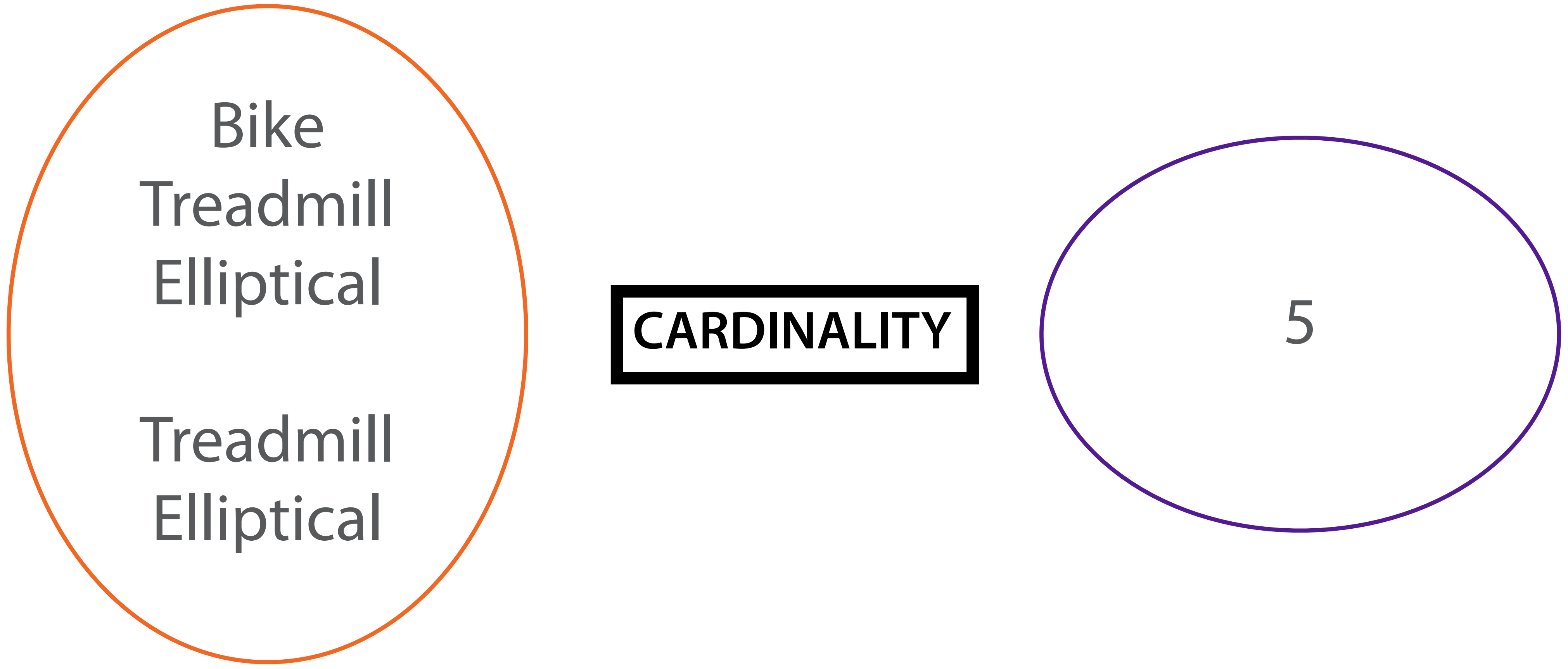
PL/SQL

```
DECLARE
 l_final_nt         items_nt;
 CURSOR cur_get_items IS
  SELECT  store1_items
  FROM     item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_final_nt;
  CLOSE cur_get_items;
   IF l_final_nt IS A SET THEN
    …
  END IF;
END;
```

SQL

```
SELECT  act_month,
           CASE
             WHEN store1_items IS A SET THEN 'It is a set'
             ELSE  'It is not a set'
           END  dup_items
 FROM  item_orders;
```

CARDINALITY



```
DECLARE
  TYPE items_nt IS TABLE of VARCHAR2(60);
  l_first_nt          items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical', 'Treadmill', 'Elliptical');
  l_count         NUMBER;
BEGIN
   l_count :=  CARDINALITY(l_first_nt);
END;
```

5

CARDINALITY

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

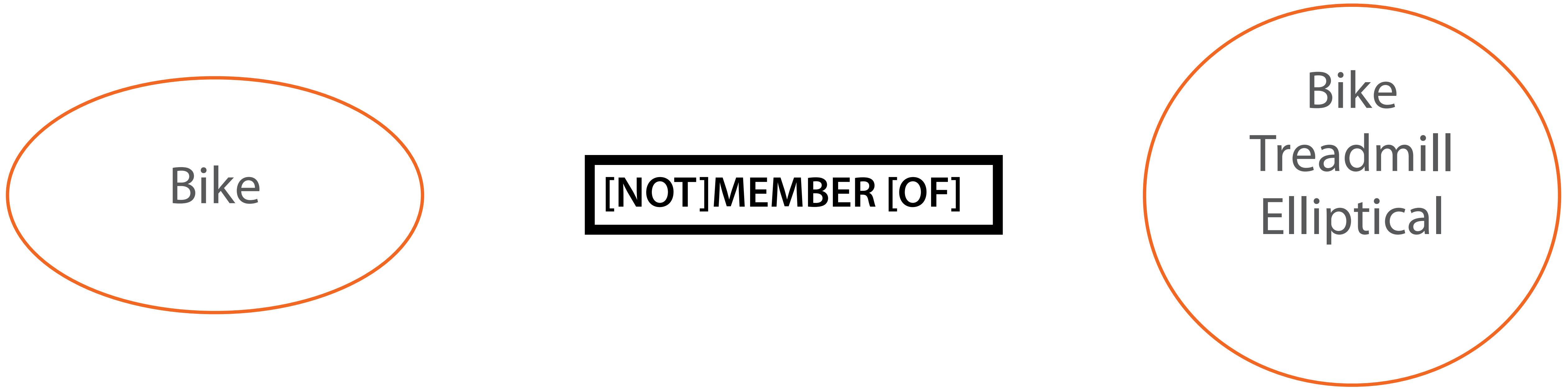
```
SELECT CARDINALITY(store1_items)
 FROM item_orders
  WHERE act_month = 'JANUARY';
```

PL/SQL

```
DECLARE
 l_count          NUMBER;
 CURSOR cur_get_items IS
  SELECT CARDINALITY(store1_items)
  FROM item_orders
   WHERE act_month = 'JANUARY';
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_count;
  CLOSE cur_get_items;
END;
```

5

MEMBER OF



```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_items_nt        items_nt  := items_nt('Bike', 'Treadmill', 'Elliptical');
 l_present       Boolean;
BEGIN
  l_present :=  'Bike' MEMBER OF (l_items_nt);              TRUE

 l_present :=  'Weights' MEMBER OF (l_items_nt);            FALSE
END;
```

MEMBER OF

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

```
SELECT    act_month
  FROM     item_orders
 WHERE  'Bike' MEMBER OF store1_items;
```

PL/SQL

```
DECLARE
 l_month    item_orders.act_month%TYPE;
 CURSOR    cur_get_items IS
  SELECT     act_month
    FROM      item_orders
   WHERE  'Bike' MEMBER OF store1_items;
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_month;
  CLOSE cur_get_items;
END;
```

JANUARY

IS [NOT] EMPTY

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt          items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
 l_second_nt      items_nt ;
BEGIN
  IF l_first_nt  IS EMPTY THEN
     DBMS_OUTPUT.PUT_LINE('Collection is empty');
  END IF;
  IF l_second_nt  IS EMPTY THEN
     DBMS_OUTPUT.PUT_LINE('Collection is empty');
  END IF;
  IF l_second_nt  IS NOT EMPTY THEN
     DBMS_OUTPUT.PUT_LINE('Collection is not empty');
  END IF;
END;
```

FALSE

FALSE

FALSE

IS [NOT] EMPTY

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

```
SELECT    act_month
  FROM     item_orders
 WHERE  store1_items IS NOT EMPTY;
```

PL/SQL

```
DECLARE
 l_month    item_orders.act_month%TYPE;
 CURSOR    cur_get_items IS
  SELECT     act_month
    FROM      item_orders
   WHERE  store1_items IS NOT EMPTY;
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_month;
  CLOSE cur_get_items;
END;
```

JANUARY

SUBMULTISET [OF]

```
DECLARE
 TYPE items_nt IS TABLE of VARCHAR2(60);
 l_first_nt          items_nt  := items_nt('Bike', 'Bike', 'Treadmill', 'Elliptical');
 l_second_nt     items_nt   := items_nt('Bike');
BEGIN
  IF  l_second_nt  SUBMULTISET OF  l_first_nt THEN
     DBMS_OUTPUT.PUT_LINE('It is Submultiset');
  END IF;
END;
```

SUBMULTISET [OF]

ACT_MONTH	STORE1_ITEMS	STORE2_ITEMS
JANUARY	Bike	Treadmill
	Bike	Elliptical
	Treadmill	
	Elliptical	

SQL

```
SELECT    act_month
  FROM     item_orders
 WHERE  store2_items SUBMULTISET OF store1_items;
```

PL/SQL

```
DECLARE
 l_month    item_orders.act_month%TYPE;
 CURSOR    cur_get_items IS
  SELECT    act_month
    FROM     item_orders
   WHERE  store2_items SUBMULTISET OF store1_items;
BEGIN
  OPEN cur_get_items;
   FETCH cur_get_items INTO l_month;
  CLOSE cur_get_items;
END;
```

JANUARY

Summary

Unnesting using TABLE Expression

Piecewise DML

MULTISET operator

SET operator

Comparing nested tables

Next up.. Varrays

Varrays

Pankaj Jain

@twit_pankajj

Module Overview

Define & Use

Adding & Removing Elements

Exceptions

Schema Level Varrays

What is a varray?

Variable size array

Maximum size specified at declaration

Dense

Database level

Piecewise operations are not allowed

Equivalent to Array type in other languages

Where Can They Be Declared?

PL/SQL	Schema Level
Anonymous blocks	Schema type
Stored subprograms	Table column

Usage Guidelines

Max number of elements is known

Elements are accessed sequentially

Fewer number of rows

Maintaining order of elements is important

Defining Varrays

PL/SQL

```
TYPE <type_name> IS VARRAY(size_limit)  OF <element_type> [NOT NULL] ;
```

```
TYPE mytype_va IS VARRAY(5) OF NUMBER;

TYPE mytype_va IS VARRAY(5) OF VARCHAR2(60) NOT NULL;

TYPE mytype_va IS VARRAY(5) OF customers%ROWTYPE;
```

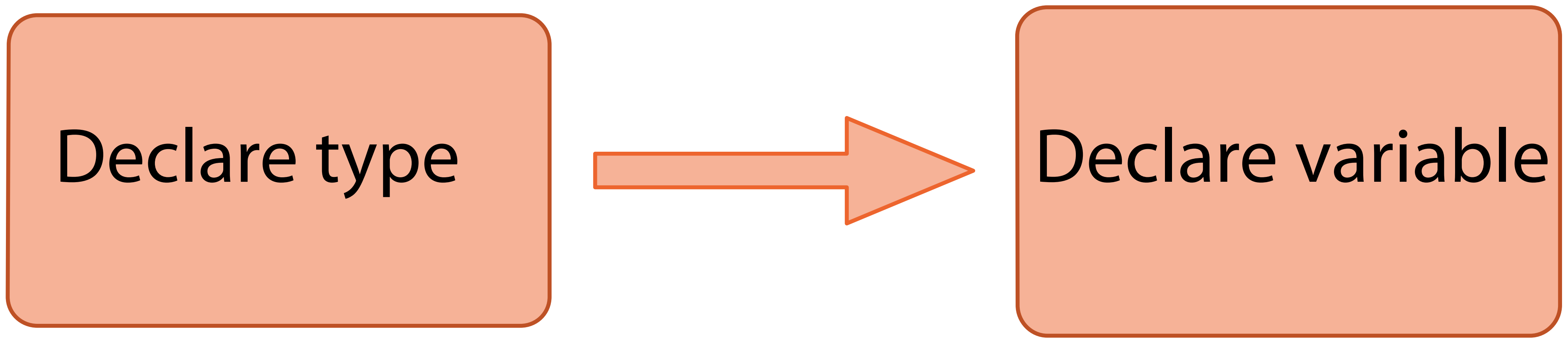
Defining Varrays

SQL

```
CREATE [OR REPLACE] TYPE <type_name> IS/AS VARRAY(size_limit) OF <element_type> [NOT NULL] ;
```

```
CREATE OR REPLACE TYPE mytype_va IS VARRAY(5) OF NUMBER;

CREATE OR REPLACE TYPE  mytype_va IS VARRAY(5) OF VARCHAR2(60) NOT NULL;
```

Declaring Variables



```
CREATE OR REPLACE TYPE  items_va IS VARRAY(5) OF
VARCHAR2(60) NOT NULL;
```

```
DECLARE
 TYPE  items_va IS VARRAY(5) OF VARCHAR2(60) NOT NULL;
 l_items_va  items_va;
 …
```

Atomically null array

```
DECLARE
 l_items_va  items_va;
 …
```

Atomically null array

Initializing Varrays

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60) ;
 l_items_va     items_va;
BEGIN
 DBMS_OUTPUT.PUT_LINE( l_items_va.COUNT);
END;
```

ORA-06531: Reference to uninitialized collection

Initializing Varrays

Constructor

```
DECLARE
  TYPE items_va IS VARRAY(5) of VARCHAR2(60);
  l_items_va      items_va;

BEGIN
  l_items_va := items_va('Bike', 'Treadmill');
  DBMS_OUTPUT.PUT_LINE( l_items_va.COUNT);
END;
```

2

```
DECLARE
  TYPE items_va IS TABLE of VARCHAR2(60) ;
  l_items_va      items_va := items_va('Bike', 'Treadmill');

BEGIN
 DBMS_OUTPUT.PUT_LINE( l_items_va.COUNT);
END;
```

2

Initializing Varrays

Constructor without arguments

```
DECLARE
  TYPE items_va IS VARRAY(5) of VARCHAR2(60) ;
  l_items_va      items_va := items_va();

BEGIN

  DBMS_OUTPUT.PUT_LINE( l_items_va.COUNT);
END;
```

Initialized as an empty array

0

Varray Index

Integer

Starts with 1

Adding Elements

Extend

Extend(n)

Extend(n,i)

```
DECLARE
  TYPE items_va IS TABLE of VARCHAR2(60) ;
  l_items_va     items_va := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(1) := 'Bike';
  l_items_va.EXTEND(2);
  l_items_va(2) := 'Bike';
  l_items_va(3) := 'Treadmill';
  l_items_va.EXTEND(2,1);

 DBMS_OUTPUT.PUT_LINE( l_items_va(4));
 DBMS_OUTPUT.PUT_LINE( l_items_va(5));
END;
```

```
Bike
Bike
```

Adding Elements

Extend method

```
DECLARE
 TYPE items_rec IS RECORD( item_name  items.item_name%TYPE,
                                        count NUMBER);
 TYPE items_va IS VARRAY(5) of items_rec ;
 l_items_va      items_va := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(l_items_va.LAST).item_name := 'Bike';
  l_items_va(l_items_va.LAST).count := 1;


  l_items_va.EXTEND;
  l_items_va(l_items_va.LAST).item_name := 'Treadmill';
  l_items_va(l_items_va.LAST).count := 2;

DBMS_OUTPUT.PUT_LINE( l_items_va(1).item_name);
END;
```

Adding Elements

Extend method

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60) ;
 l_items_va      items_va := items_va();
 CURSOR get_items IS
    SELECT *
       FROM items
    WHERE ROWNUM < 6;
BEGIN
  FOR get_items_var IN get_items LOOP
    l_items_va.EXTEND;
    l_items_va(l_items_va.LAST) := get_items_var.item_name;
  END LOOP;
END;
```

Deleting Elements

Cannot be Sparse

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60) ;
 l_items_va     items_va := items_va();

BEGIN
  l_items_va.EXTEND(3);
  l_items_va(1) := 'Bike';
  l_items_va(2) := 'Treadmill';
  l_items_va(3) := 'Elliptical';

 l_items_va.DELETE(2);

 l_items_va.DELETE;

 DBMS_OUTPUT.PUT_LINE(l_items_va.COUNT);

END;
```

PLS-00306: wrong number of arguments in call to DELETE

0

Reducing Size

TRIM

TRIM(n)

```
DECLARE
 TYPE items_va IS TABLE of VARCHAR2(60);
 l_items_va     items_va  := items_va();

BEGIN
  l_items_va.EXTEND(3);
  l_items_va(1) := 'Bike';
  l_items_va(2) := 'Treadmill';
  l_items_va(3) := 'Elliptical';
  l_items_va.TRIM;
  DBMS_OUTPUT.PUT_LINE(l_items_va.COUNT);      2
  l_items_va.TRIM(2);
  DBMS_OUTPUT.PUT_LINE(l_items_va.COUNT);      0
END;
```

Reassignment

Reassigning overwrites previous value at that index

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60) ;
 l_items_va     items_va := items_va();

BEGIN
  l_items_va.EXTEND(3);
  l_items_va(1) := 'Bike';
  l_items_va(2) := 'Treadmill';
  l_items_va(1) := 'Elliptical';

 DBMS_OUTPUT.PUT_LINE(l_items_va(1));

END;
```

Elliptical

Assigning Value to Varrays

Assigning another varray

```
DECLARE
  TYPE items_va IS VARRAY(5) of VARCHAR2(60);
  l_items_va      items_va := items_va();
  l_copy_va       items_va;
BEGIN
  l_items_va.EXTEND(3);
  l_items_va(1) := 'Treadmill';
  l_items_va(2) := 'Bike';
  l_items_va(3) := 'Elliptical';

  l_copy_va := l_items_va;

 dbms_output.put_line(l_copy_va(2));
END;
```

Bike

Assigning Value to Varrays

Same type

```
DECLARE
 TYPE items_va   IS VARRAY(5) of VARCHAR2(60);
 TYPE dup_va     IS VARRAY(5) of VARCHAR2(60) ;
 l_items_va      items_va;
 l_dup_va        dup_va;
BEGIN
 l_items_va.EXTEND(3);
 l_items_va(1) := 'Treadmill';
 l_items_va(2) := 'Bike';
 l_items_va(3) := 'Elliptical';

 l_dup_va := l_items_va; ✗

END;
```

Assigning Value to Varrays

Assigning empty array

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60);
 l_items_va      items_va := items_va();
 l_copy_va       items_va := items_va();
BEGIN
 l_items_va.EXTEND(3);
 l_items_va(1) := 'Treadmill';
 l_items_va(2) := 'Bike';
 l_items_va(3) := 'Elliptical';

 l_items_va := l_copy_va ;

END;
```

Exceptions During Assignment

Subscript outside of limit

```
DECLARE
  TYPE items_va IS VARRAY(2) of VARCHAR2(60);
  l_items_va     items_va  := items_va();

BEGIN
   l_items_va.EXTEND(3);
   …
EXCEPTION
  WHEN SUBSCRIPT_OUTSIDE_LIMIT THEN
     DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
      RAISE;
END;
```

ORA-6532:
Subscript
outside of limit

Exceptions During Assignment

Subscript outside of limit

```
DECLARE
 TYPE items_va IS TABLE of VARCHAR2(60);
 l_items_va     items_va  := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(0) := 'Treadmill';
EXCEPTION
 WHEN SUBSCRIPT_OUTSIDE_LIMIT THEN
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
END;
```

ORA-6532: Subscript outside of limit

Exceptions During Assignment

Value Error

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(4);
 l_items_va     items_va  := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(1) := 'Treadmill';
EXCEPTION
 WHEN VALUE_ERROR THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
END;
```

ORA-06502: PL/SQL numeric or value error

Exceptions During Assignment

Value Error

```
DECLARE
  TYPE items_va IS VARRAY(5) of VARCHAR2(60);
  l_items_va     items_va  := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va('A') := 'Treadmill';
EXCEPTION
  WHEN VALUE_ERROR THEN
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
END;
```

ORA-6502: PL/SQL numeric or value error

Exceptions During Assignment

Uninitialized collection

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60);
 l_items_va     items_va ;

BEGIN
  l_items_va(1) := 'Bike';
END;
```

ORA-06531: Reference to uninitialized collection

Exceptions During Assignment

Not null constraint

```
DECLARE
  TYPE items_va IS VARRAY(5) of VARCHAR2(60) NOT NULL;
  l_items_va      items_va := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(1) := NULL;
END;
```

PLS-00382: expression is of wrong type

Exceptions During Assignment

Subscript beyond count

```
DECLARE
 TYPE items_va IS VARRAY(5) of VARCHAR2(60);
 l_items_va     items_va  := items_va();

BEGIN
  l_items_va(1) := 'Bike';
EXCEPTION
  WHEN SUBSCRIPT_BEYOND_COUNT THEN
    DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    RAISE;
 END;
```

ORA-06533: Subscript beyond count

Reducing Size

Subscript beyond count

```
DECLARE
 TYPE items_va IS VARRAY(3) of VARCHAR2(60);
 l_items_va     items_va  := items_va();

BEGIN
  l_items_va.EXTEND;
  l_items_va(1) := 'Bike';
  l_items_va.EXTEND;
  l_items_va(2) := 'Treadmill';
  l_items_va.TRIM(4);
  DBMS_OUTPUT.PUT_LINE(l_items_va.COUNT);
END;
```

ORA-06533: Subscript beyond count

Schema Level Varrays

Available throughout the system

Columns of database tables

Easier information retrieval

Interacting with Schema Level Varrays

```
CREATE OR REPLACE TYPE items_va AS VARRAY(5) OF VARCHAR2(60);
```

```
CREATE TYPE orders_ot AS OBJECT (order_id NUMBER, order_item_id NUMBER);

CREATE OR REPLACE TYPE  orders_va IS VARRAY(5) OF orders_ot;
```

```
CREATE TABLE act_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va(),
  orderslist    orders_va  DEFAULT orders_va());
```

Dropping Schema Level Varrays

```
DROP  TYPE  <type_name> [FORCE | VALIDATE];
```

```
DROP TYPE mytype_va;
```

ORA-2303: cannot drop or replace a type with type or table dependents

```
DROP TYPE mytype_va FORCE;
```

Altering Schema Level Varray size

```
ALTER TYPE <varray_name>  MODIFY ELEMENT TYPE  <new_datatype_size> CASCADE | INVALIDATE;
```

```
CREATE TYPE items_va AS VARRAY(5) OF VARCHAR2(60);
```

```
ALTER TYPE items_va MODIFY ELEMENT TYPE VARCHAR2(100) CASCADE;
```

```
ALTER TYPE items_va MODIFY ELEMENT TYPE VARCHAR2(10) CASCADE;
```

DML Operations on Varrays

No piecewise operations allowed

Varray columns inserted/updated as atomic unit

Inserting

```
CREATE TABLE act_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va(),
  orderslist    orders_va  DEFAULT orders_va());
```

SQL

```
INSERT INTO
  act_orders (act_id,
                act_month,
                itemslist,
                orderslist)
    VALUES ( 1,
             'JANUARY',
             items_va('Bike', 'Treadmill'),
             orders_va(orders_ot(1,1),orders_ot(2,2)));
```

PL/SQL

```
DECLARE
  l_items_va      items_va := items_va();
  l_orders_va     orders_va:= orders_va();
  l_orders_ot    orders_ot := orders_ot(1,1);

BEGIN
  l_items_va.EXTEND(2);
  l_items_va(1) := 'Bike';
  l_items_va(2) := 'Treadmill';

  l_orders_va.EXTEND(2);
  l_orders_va(1) := l_orders_ot;
  l_orders_va(2) := orders_ot(2,2);
  INSERT INTO  act_orders  (act_id, act_month,  itemslist,   orderslist)
                            VALUES(  1,       'JANUARY',  l_items_va, l_orders_va);
  COMMIT;
END;
```

Updating

```
CREATE TABLE act_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va(),
  orderslist    orders_va  DEFAULT orders_va());
```

SQL

```
UPDATE act_orders
     SET
      itemslist =  items_va('Elliptical'),
    orderslist = orders_va(orders_ot(1,1),orders_ot(3,3))
       WHERE  act_id = 1
           AND  act_month =  'JANUARY';
```

PL/SQL

```
DECLARE
  l_items_va      items_va := items_va();
  l_orders_va    orders_va:= orders_va();
  l_orders_ot    orders_ot := orders_ot(1,1);

BEGIN
  l_items_va.EXTEND(1);
  l_items_va(1) := 'Elliptical';

  l_orders_va.EXTEND(2);
  l_orders_va(1) := l_orders_ot;
  l_orders_va(2) := orders_ot(3,3);
  UPDATE act_orders SET itemslist = l_items_va,
                                  orderslist = l_orders_va
                           WHERE  act_id = 1
                               AND  act_month =  'JANUARY';
  COMMIT;
END;
```

Deleting

```
CREATE TABLE act_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va(),
  orderslist    orders_va  DEFAULT orders_va());
```

SQL

```
UPDATE act_orders
     SET itemslist =  NULL,
          orderslist =  NULL
     WHERE  act_id = 1
     AND       act_month =  'JANUARY';
```

PL/SQL

```
BEGIN
 DELETE FROM act_orders
      WHERE  act_id = 1
      AND       act_month =  'JANUARY';
  COMMIT;
END;
```

```
BEGIN
  UPDATE act_orders SET itemslist = NULL,
                                              orderslist = NULL
      WHERE  act_id = 1
      AND       act_month =  'JANUARY';
  COMMIT;
END;
```

Selecting

```
CREATE TABLE act_orders (
   act_id NUMBER,
   act_month VARCHAR2(8),
   itemslist      items_va    DEFAULT items_va(),
   orderslist    orders_va  DEFAULT orders_va());
```

SQL

```
SELECT * FROM act_orders
WHERE  act_id = 1  AND       act_month =  'JANUARY';

ACT_ID   ACT_MONTH  ITEMSLIST   ORDERSLIST
———-  ———————— ———————-  ——————————
   1          JANUARY      DEMO.ITEMS_VA('Bike','Treadmill')
```

PL/SQL

```
DECLARE
  l_items_va      items_va := items_va();
 l_orders_va    orders_va:= orders_va();
 CURSOR get_details_cur IS
    SELECT  itemslist, orderslist
       FROM  act_orders
      WHERE  act_id = 1  AND       act_month =  'JANUARY';
 BEGIN
   OPEN  get_details_cur;
   FETCH get_details_cur INTO l_items_va, l_orders_va;
   CLOSE get_details_cur;
   IF  l_items_va IS NOT NULL THEN
     FOR i IN l_items_va.FIRST .. l_items_va.LAST  LOOP
       DBMS_OUTPUT.PUT_LINE('Item name '||l_items_va(i));
     END LOOP;
   END IF;
   IF  l_items_va IS NOT NULL THEN
    FOR i IN l_orders_va.FIRST .. l_orders_va.LAST  LOOP
      DBMS_OUTPUT.PUT_LINE('Item id '||l_orders_va(i).order_id);
    END LOOP;
   END IF;
END;
```

Selecting

```
CREATE TABLE act_orders (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va(),
  orderslist    orders_va  DEFAULT orders_va());
```

SQL

```
SELECT * FROM act_orders
WHERE  act_id = 1  AND       act_month =  'JANUARY';

ACT_ID   ACT_MONTH     ITEMSLIST                              ORDERSLIST
———-  ——————         ———— ——————
   1         JANUARY          DEMO.ITEMS_VA('Bike', 'Treadmill')    DEMO.ORDERS_VA(DEMO.ORDERS_OT(1,1),DEMO.ORDERS_OT(2,2))
```

Unnesting

TABLE Expression

SQL

```
SELECT  order_id, order_item_id FROM  TABLE(SELECT orderslist FROM act_orders
                                              WHERE  act_id = 1
                                                AND  act_month =  'JANUARY');

ORDER_ID   ORDER_ITEM_ID
————————-   ——————————— ——-
  1                1
  3                3
```

```
SELECT  b.COLUMN_VALUE FROM  act_orders a, TABLE(itemslist) b
                             WHERE  act_id = 1
                               AND  act_month =  'JANUARY';

COLUMN_VALUE
——————————————
  Elliptical
```

Summary

Working with Varrays in PL/SQL

EXTEND and TRIM methods

Exceptions

Schema level varrays

DML on varray table columns

Next up.. Multilevel collections & converting collections

Multilevel Collections & Converting Collections

Pankaj Jain

@twit_pankajj

Module Overview

Nesting Collections

CAST Function

Multiset Function

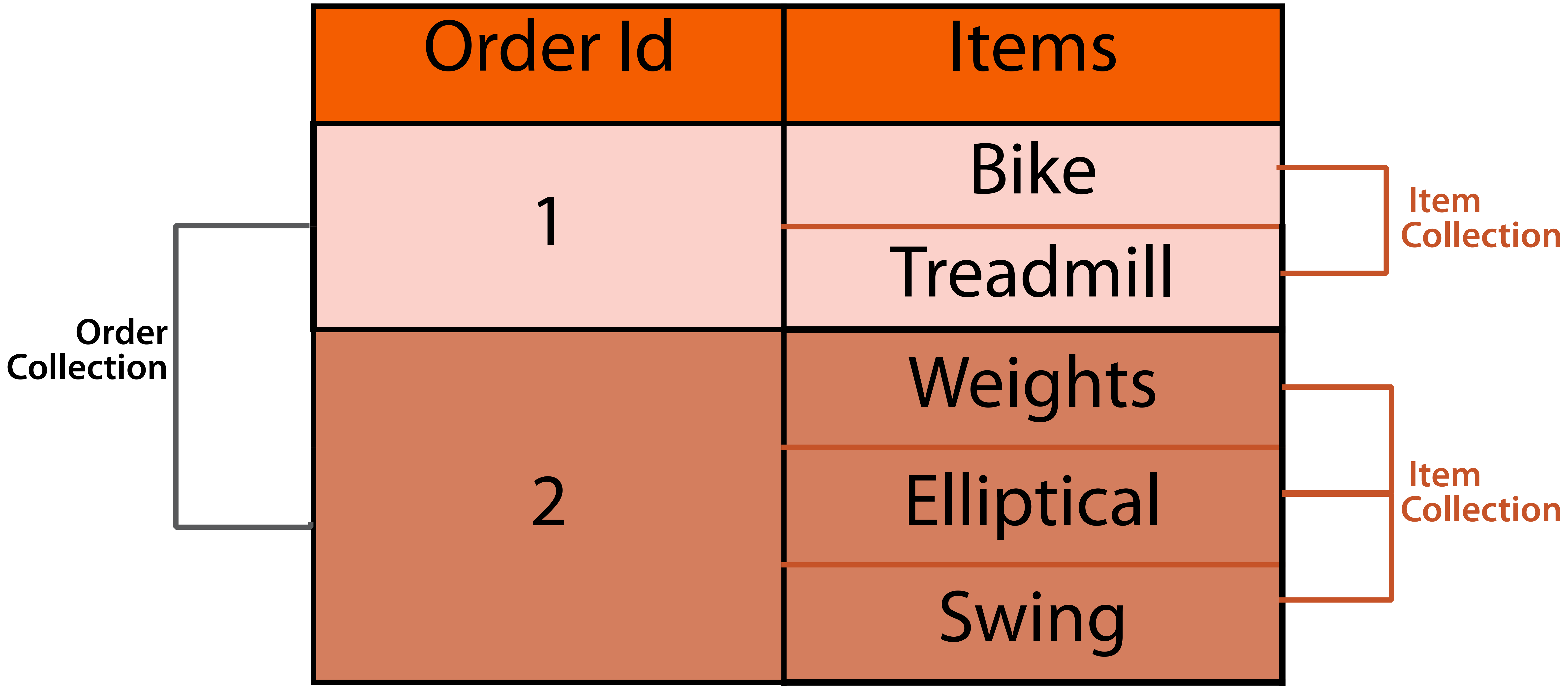
Collect Function

Multilevel Collections

Collection Within Collection

- Nested table within nested table
- Varrays within varrays
- Associative arrays within associative arrays
- Varrays within nested table
- ......

Multilevel Collections



Order Id	Items
1	Bike
	Treadmill
2	Weights
	Elliptical
	Swing

Associative Array Within Another Associative Array

Order Id	Items
1	Bike
	Treadmill
2	Weights
	Elliptical

```
DECLARE
   TYPE items_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;
   TYPE orders_rec IS RECORD(order_id NUMBER, items items_aa);
   TYPE orders_aa IS TABLE OF orders_rec INDEX BY BINARY_INTEGER;
   l_items_aa items_aa;
   l_orders_aa orders_aa;
  BEGIN
   l_items_aa(1) := 'Bike';
   l_items_aa(2) := 'Treadmill';
   l_orders_aa(1).order_id := 1;
   l_orders_aa(1).items := l_items_aa;

   l_items_aa(1) := 'Weights';
   l_items_aa(2) := 'Elliptical';
   l_orders_aa(2).order_id := 2;
   l_orders_aa(2).items := l_items_aa;

   ....
   ....
END;
```

Associative Array Within Another Associative Array

Accessing & Replacing Elements

Order Id	Items
1	Bike
	Treadmill
2	Weights
	Elliptical

```
DECLARE
  …
  …
BEGIN
 …
 …

  --Access 2nd element of the items collection from the first order
   DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                    l_orders_aa(1).items(2));          Treadmill

 --Replace 2nd element of the items collection from the first order
   l_orders_aa(1).items(2):= 'Weights';
  DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                    l_orders_aa(1).items(2));          Weights
 …
 …
END;
```

Associative Array Within Another Associative Array

Adding & Deleting Elements

Order Id	Items
1	Bike
	Weights
2	Weights
	Elliptical
	Swing

```
DECLARE
  …
  …
BEGIN
 …
  --Add a third item to the second order
    l_orders_aa(2).items(3):= 'Swing';

   DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
    l_orders_aa(2).items.COUNT);          3

  --Remove the second item of the second order

    l_orders_aa(2).items.DELETE(2);
    DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
 …l_orders_aa(2).items.COUNT);          2
 …
END;
```

Nested Table Within a Varray

Order Id	Items
1	Bike
	Treadmill
2	Weights
	Elliptical

```
DECLARE
   TYPE items_nt IS TABLE OF VARCHAR2(60);
   TYPE orders_ot IS RECORD(order_id NUMBER, items items_nt);
   TYPE orders_va IS VARRAY(5) OF orders_ot;
    l_items_nt  items_nt := items_nt();
    l_orders_va orders_va := orders_va();
  BEGIN
    l_items_nt.EXTEND(2);
    l_items_nt(1) := 'Bike';
    l_items_nt(2) := 'Treadmill';
    l_orders_va.EXTEND;
    l_orders_va(1).order_id := 1;
    l_orders_va(1).items := l_items_nt;

    l_items_nt(1) := 'Weights';
    l_items_nt(2) := 'Elliptical';
    l_orders_va.EXTEND;
    l_orders_va(2).order_id := 2;
    l_orders_va(2).items := l_items_nt;

    ....
    ....
END;
```

Nested Table Within a Varray

Accessing & Replacing Elements

Order Id	Items
1	Bike Treadmill
2	Weights Elliptical

```
DECLARE
  …
  …
BEGIN
 …
 …

  --Access 2nd element of the items collection from the first order
    DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                  l_orders_va(1).items(2));        Treadmill

 --Replace 2nd element of the items collection from the first order
   l_orders_va(1).items(2):= 'Weights';
   DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                 l_orders_va(1).items(2));         Weights
 …
 …
END;
```

Nested Table Within a Varray

Adding & Deleting Elements

Order Id	Items
1	Bike
	Weights
2	Weights
	Elliptical
	Swing

```
DECLARE
  …
  …
BEGIN
 …
 --Add a third item to the second order
   l_orders_va(2).items.EXTEND;
   l_orders_va(2).items(3):= 'Swing';
   DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
   l_orders_va(2).items.COUNT);          3

 --Remove the second item of the second order
   l_orders_va(2).items.DELETE(2);
   DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
   l_orders_va(2).items.COUNT);          2
 …
 …
END;
```

Nested Table Within Another Nested Table

Order Id	Items
1	Bike Treadmill
2	Weights Elliptical

```
CREATE OR REPLACE TYPE items_nt IS TABLE OF VARCHAR2(60);
CREATE OR REPLACE TYPE orders_ot IS OBJECT(order_id NUMBER, items items_nt);


 CREATE OR REPLACE TYPE  orders_nt IS TABLE OF orders_ot;
```

```
 DECLARE
  l_items_nt items_nt := items_nt();
  l_orders_nt orders_nt := orders_nt();
BEGIN
  l_items_nt.EXTEND(2);
  l_items_nt(1) := 'Bike';
  l_items_nt(2) := 'Treadmill';
  l_orders_nt.EXTEND;
  l_orders_nt(1) := orders_ot(1,l_items_nt);


   l_orders_nt.EXTEND;
   l_orders_nt(2) := orders_ot(2,items_nt('Weights','Elliptical'));
  ...
  END;
```

Nested Table Within Another Nested Table

Accessing & Replacing Elements

Order Id	Items
1	Bike
	Treadmill
2	Weights
	Elliptical

```
DECLARE
  …
  …
BEGIN
 …
 …

  --Access 2nd element of the items collection from the first order
    DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                        l_orders_nt(1).items(2));        Treadmill

 --Replace 2nd element of the items collection from the first order
    l_orders_nt(1).items(2):= 'Weights';
   DBMS_OUTPUT.PUT_LINE('First order''s 2nd item is '||
                                        l_orders_nt(1).items(2));        Weights
 …
 …
END;
```

Nested Table Within Another Nested Table

Adding & Deleting Elements

Order Id	Items
1	Bike
	Weights
2	Weights
	Elliptical
	Swing

```
DECLARE
  …
  …
BEGIN
 …
  --Add a third item to the second order
    l_orders_nt(2).items.EXTEND;
    l_orders_nt(2).items(3):= 'Swing';
   DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
    l_orders_nt(2).items.COUNT);                3

  --Remove the second item of the second order
    l_orders_nt(2).items.DELETE(2);
    DBMS_OUTPUT.PUT_LINE('Count of items for the second order is '||
    l_orders_nt(2).items.COUNT);                2
 …
 …
END;
```

Nested Table Within Another Nested Table

Database Table Column

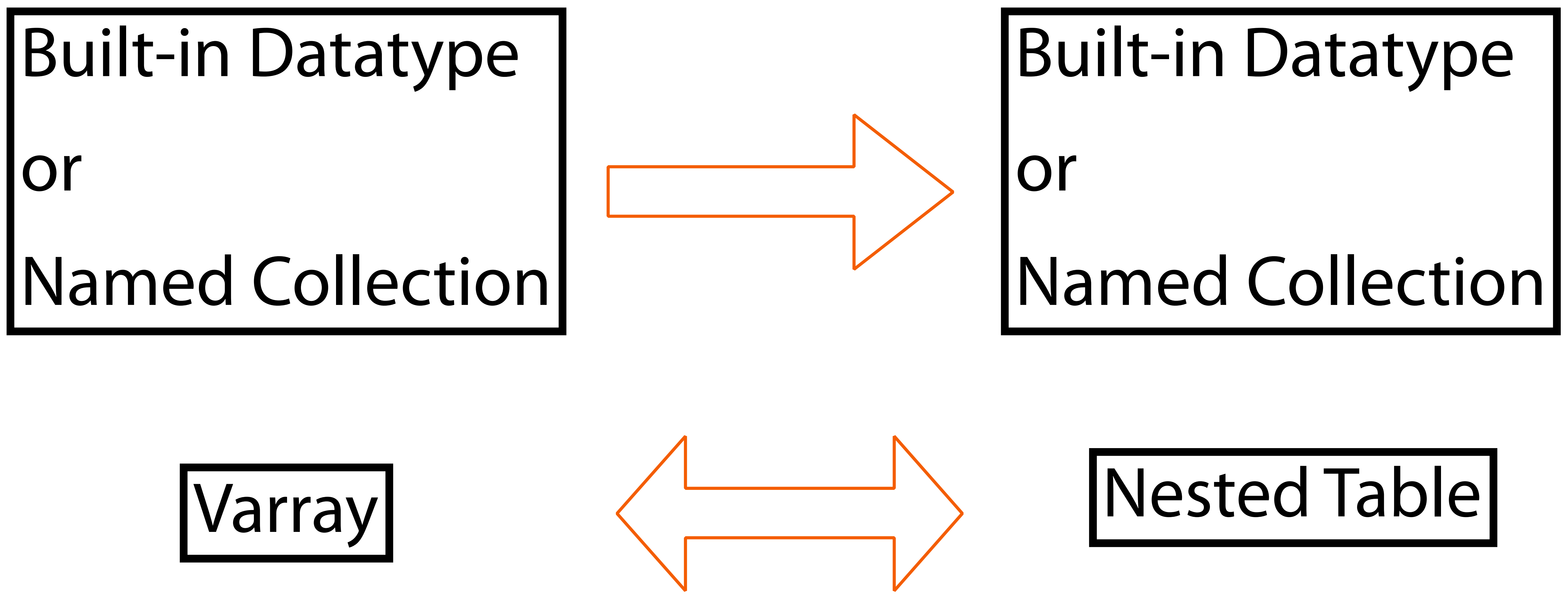
```
CREATE TABLE monthly_orders
   (act_id NUMBER,
    act_month VARCHAR2(8),
    order_info  orders_nt)
   NESTED TABLE order_info STORE AS order_store
     (NESTED TABLE items STORE AS items_store);
```

```
CREATE OR REPLACE TYPE items_nt IS TABLE OF VARCHAR2(60);
CREATE OR REPLACE TYPE orders_ot IS OBJECT(order_id NUMBER, items items_nt);

CREATE OR REPLACE TYPE  orders_nt IS TABLE OF orders_ot;
```

```
INSERT INTO monthly_orders
   (act_id,
    act_month,
    order_info )
   VALUES
   (1,
     'JANUARY',
     orders_nt(
       orders_ot(1, items_nt('Bike', 'Treadmill') ) ,
       orders_ot(2, items_nt('Weights')  )
      )
   )
```

```
DECLARE
  CURSOR order_info_cur IS
     SELECT act_id,
                 order_info
       FROM monthly_orders
       WHERE act_month = 'JANUARY';
  l_act_id monthly_orders.act_id%TYPE;
  l_order_info orders_nt;
BEGIN
  OPEN order_info_cur;
   FETCH order_info_cur
     INTO l_act_id,
            l_order_info;
   CLOSE order_info_cur;
   ....
```

CAST Function



Same internal elements

CAST Function

```
CAST (
        (expr) /
        MULTISET (subquery)
        AS <type_name>
      )
```

- Expr : Built-in datatype / collection type / ANYDATA
- MULTISET used with subqueries
- COLLECT used with scalar columns

CAST Function

```
CREATE OR REPLACE TYPE items_va AS VARRAY(5) OF VARCHAR2(60);
CREATE OR REPLACE TYPE items_nt AS TABLE OF VARCHAR2(60);
```

```
CREATE TYPE order_info_ot AS OBJECT (order_id NUMBER, item_name VARCHAR2(60));

CREATE OR REPLACE TYPE  order_info_nt IS TABLE OF order_info_ot;
```

```
CREATE TABLE items_ordered (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va());
```

CAST Function

```
CREATE TABLE items_ordered (
  act_id NUMBER,
  act_month VARCHAR2(8),
  itemslist     items_va    DEFAULT items_va());
```

```
SELECT CAST( itemslist AS items_nt)
  FROM  items_ordered;
```

```
DECLARE
  CURSOR get_items_cur IS
    SELECT CAST( itemslist AS items_nt)
      FROM  items_ordered;
  l_items_nt  items_nt;
BEGIN
  OPEN  get_items_cur ;
  FETCH get_items_cur INTO l_items_nt;
  CLOSE get_items_cur;
  l_items_nt := SET(l_items_nt);
  ….
END;
```

```
DECLARE
  CURSOR get_items_cur IS
    SELECT itemslist
      FROM  items_ordered;
  l_items_nt  items_nt;
  l_items_va  items_va;
BEGIN
  OPEN  get_items_cur ;
  FETCH get_items_cur INTO l_items_va;
  CLOSE get_items_cur;
  SELECT CAST(l_items_va AS items_nt) INTO l_items_nt
    FROM dual;
  ….
END;
```

CAST Function

```
CREATE TYPE order_info_ot AS
OBJECT (order_id NUMBER, item_name VARCHAR2(60));

CREATE OR REPLACE TYPE  order_info_nt IS TABLE OF
order_info_ot;
```

```
SELECT CAST(
           MULTISET(SELECT order_id, item_name FROM
                           orders, items
                           WHERE order_item_id = item_id
                           AND   order_act_id = 1
                        )   AS order_info_nt
           )
    FROM  dual;
```

```
CREATE OR REPLACE FUNCTION get_order_info(p_act NUMBER)
                                           RETURN order_info_nt IS
  CURSOR get_order_cur IS
  SELECT CAST(
           MULTISET(SELECT order_id, item_name FROM
                           orders, items
                           WHERE order_item_id = item_id
                             AND order_act_id = p_act
                        )   AS order_info_nt
           )
    FROM  dual;
  l_order_info_nt  order_info_nt;
BEGIN
  OPEN  get_order_cur ;
  FETCH get_order_cur INTO l_order_info_nt;
  CLOSE get_order_cur;
   ….
  RETURN l_order_info_nt;
END  get_order_info;
```

COLLECT Function

```
CREATE OR REPLACE TYPE items_nt AS TABLE OF VARCHAR2(60);
```

```
SELECT CAST(
            COLLECT(item_name)
            AS items_nt
           )
  FROM  items;
```

```
SELECT CAST(
            MULTISET(SELECT item_name FROM   items)
            AS items_nt
           )
  FROM  dual;
```

```
DECLARE
  CURSOR get_items_cur IS
    SELECT CAST(
                COLLECT(item_name)
                AS items_nt
                )
      FROM  items;
  l_items_nt  items_nt;
BEGIN
  OPEN  get_items_cur ;
  FETCH get_items_cur INTO l_items_nt;
  CLOSE get_items_cur;
  l_items_nt := SET(l_items_nt);
  ….
END;
```

Summary

Nesting collections

Adding and removing elements

CAST function with MULTISET

CAST function with COLLECT

Next up.. Bulk Collect

Bulk Operations: Bulk Collect

Pankaj Jain

@twit_pankajj

Module Overview

Benefit

Usage

LIMIT Clause

Performance Comparison

Bulk Operations

Bulk Collect

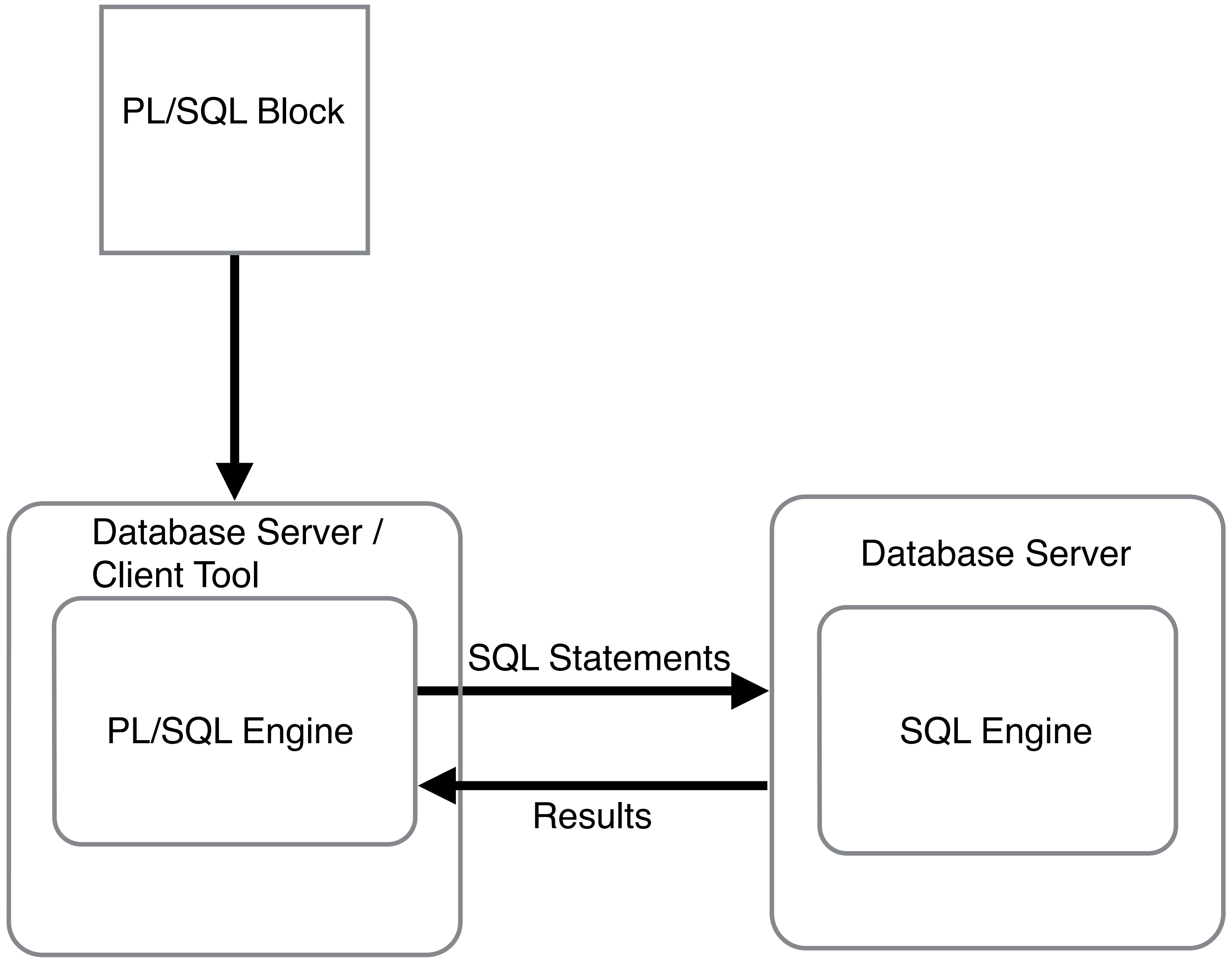
Fetching data from database in bulk

FORALL

Inserting / Updating /Deleting data in the database in bulk

Context Switches

PL/SQL Processing



Benefits

Performance optimization

Reducing network roundtrips

Bulk Collect

```
BULK COLLECT INTO <collection_name>
```

Where Can It Appear?

- SELECT INTO clause
- FETCH INTO clause
- RETURNING INTO clause
- Dynamic SQL statements

Bulk Collect

Fetch can be done in all three collection types	**Fetched collection is dense**	**Erases previous values**

Memory Considerations

SELECT INTO Clause

```
DECLARE
 TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
 l_itemid_nt     itemid_nt;

BEGIN
  SELECT   item_id
     BULK COLLECT INTO l_itemid_nt
   FROM items;
  DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

SELECT INTO Clause

Does not raise exception

```
DECLARE
 TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
 l_itemid_nt     itemid_nt;

BEGIN
  SELECT   item_id
     BULK COLLECT INTO l_itemid_nt
   FROM items
  WHERE item_name LIKE 'D%';
  DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

```
0
```

SELECT INTO Clause

Multiple Columns

Individual Collections

```
DECLARE
 TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
 l_itemid_nt     itemid_nt;
 TYPE item_name_aa IS TABLE OF VARCHAR2(60) INDEX BY BINARY_INTEGER;
 l_item_name_aa     item_name_aa;

BEGIN
  SELECT  item_id,
          item_name
    BULK COLLECT INTO l_itemid_nt,
                      l_item_name_aa
   FROM items;
   DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
   DBMS_OUTPUT.PUT_LINE( l_item_name_aa.COUNT);
END;
```

SELECT INTO Clause

Multiple Columns

Collection of Records

```
DECLARE
  TYPE item_info IS RECORD(  item_id NUMBER, item_name items.item_name%TYPE);
  TYPE item_info_nt IS TABLE OF item_info;
  l_item_info_nt      item_info_nt;

BEGIN
   SELECT   item_id,
            item_name
     BULK COLLECT INTO l_item_info_nt
    FROM items;
    DBMS_OUTPUT.PUT_LINE( l_item_info_nt.COUNT);
END;
```

Limiting Rows Fetched

ROWNUM

```
DECLARE
 TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
 l_itemid_nt     itemid_nt;

BEGIN
  SELECT   item_id
   BULK COLLECT INTO l_itemid_nt
   FROM items
   WHERE item_value > 500
   AND ROWNUM   < 101;
  DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

SAMPLE

```
DECLARE
 TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
 l_itemid_nt     itemid_nt;

BEGIN
  SELECT   item_id
   BULK COLLECT INTO l_itemid_nt
   FROM items
   SAMPLE(60)
    WHERE item_value > 500;
  DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

FETCH INTO Clause

```
DECLARE
  TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
  l_itemid_nt     itemid_nt;
  CURSOR get_item_info_cur IS
     SELECT  item_id
       FROM  items
     WHERE   item_value > 500;
BEGIN
   OPEN   get_item_info_cur;
    FETCH get_item_info_cur BULK COLLECT INTO l_itemid_nt;
    CLOSE   get_item_info_cur;
    DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

LIMIT Clause

```
DECLARE
   TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
   l_itemid_nt      itemid_nt;
   CURSOR get_item_info_cur IS
     SELECT  item_id
       FROM  items
     WHERE   item_value > 500;
BEGIN
   OPEN   get_item_info_cur;
     LOOP
       FETCH get_item_info_cur BULK COLLECT INTO l_itemid_nt LIMIT 100;
       EXIT WHEN l_itemid_nt.COUNT = 0;
       DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
       …
     END LOOP;
    CLOSE   get_item_info_cur;
END;
```

LIMIT Clause

```
DECLARE
   TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
   l_itemid_nt      itemid_nt;
   CURSOR get_item_info_cur IS
     SELECT  item_id
       FROM  items
     WHERE   ROWNUM <= 212;
BEGIN
   OPEN   get_item_info_cur;
     LOOP
       FETCH get_item_info_cur BULK COLLECT INTO l_itemid_nt LIMIT 100;
       EXIT WHEN get_item_info_cur%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
        …
     END LOOP;
    CLOSE   get_item_info_cur;
END;
```

```
100    100    12
FALSE  FALSE  TRUE
```

LIMIT Clause

```
DECLARE
   TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
   l_itemid_nt      itemid_nt;
   CURSOR get_item_info_cur IS
     SELECT  item_id
       FROM  items
     WHERE   ROWNUM <= 212;
BEGIN
   OPEN   get_item_info_cur;
     LOOP
       FETCH get_item_info_cur BULK COLLECT INTO l_itemid_nt LIMIT 100;
       EXIT WHEN l_itemid_nt.COUNT = 0;
       DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
        ...
     END LOOP;
    CLOSE   get_item_info_cur;
END;
```

```
100     100     12      0
FALSE   FALSE   FALSE   TRUE
```

RETURNING INTO Clause

```
DECLARE
  TYPE itemid_nt  IS TABLE OF PLS_INTEGER;
  l_itemid_nt     itemid_nt;
BEGIN
   UPDATE items
   SET item_value = item_value * 1.10
    WHERE item_value < 550
    RETURNING item_id  BULK COLLECT INTO l_itemid_nt;
    DBMS_OUTPUT.PUT_LINE( l_itemid_nt.COUNT);
END;
```

Dynamic SQL Statements

```
CREATE TYPE items_nt IS TABLE OF VARCHAR2(60);
```

```
CREATE OR REPLACE FUNCTION get_item_ids(p_where VARCHAR2) RETURN  items_nt IS
   l_items_nt     items_nt;
BEGIN
  EXECUTE IMMEDIATE
     'SELECT   item_name
       FROM    items '||
       p_where
     BULK COLLECT INTO l_items_nt;
    DBMS_OUTPUT.PUT_LINE( l_items_nt.COUNT);
     RETURN l_items_nt;
END;
```

```
DECLARE
   l_items_nt     items_nt;
BEGIN
  l_items_nt :=  get_item_ids('WHERE item_value > 500');
END;
```

Summary

Benefits

Usage

- SELECT INTO
- FETCH   INTO
- RETURNING INTO
- DYNAMIC SQL

LIMIT

Next up.. FORALL Statement for Bulk DML

Bulk Operations: FORALL

Pankaj Jain

@twit_pankajj

Module Overview

Usage

INDICES OF Clause

VALUES OF Clause

Module Overview

SQL%BULK_ROWCOUNT

SQL%BULK_EXCEPTIONS

Performance

FORALL

```
FORALL index_counter IN <bounds> [SAVE EXCEPTIONS] sql_statement
```

Where Can It Appear?

- INSERT statement
- UPDATE statement
- DELETE statement
- Dynamic SQL statements

Use with all three collection types

FORALL Statement

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

Considerations

FORALL iterator declared implicitly as integer

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

Considerations

Refer to a collection in the DML

Use iterator as the index value

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

Considerations

One DML per FORALL

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);
 l_itemid_upd_aa      itemid_aa := itemid_aa(10,11,17);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
   DELETE FROM items
    WHERE item_id IN l_itemid_aa(i);
    UPDATE items
        SET      item_value = item_value*1.10
     WHERE   item_id = l_itemid_upd_aa(i);
END;
```

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);
 l_itemid_upd_aa      itemid_aa := itemid_aa(10,11,17);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
   DELETE FROM items
    WHERE item_id IN l_itemid_aa(i);
  FORALL i IN l_itemid_upd_aa.FIRST .. l_itemid_upd_aa.LAST
    UPDATE items
        SET      item_value = item_value*1.10
     WHERE   item_id = l_itemid_upd_aa(i);
END;
```

Considerations

IN low_value..high_value usage needs dense collection

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa      itemid_aa := itemid_aa(4,6,8);

BEGIN
  FORALL i IN l_itemid_aa.FIRST .. l_itemid_aa.LAST
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

Considerations

You can refer to individual record fields in a collection

You can refer to the same collection in the SET and WHERE clause

```
DECLARE
 TYPE item_rec IS RECORD(item_id NUMBER, inc_factor NUMBER);
 TYPE items_aa IS TABLE OF item_rec INDEX BY BINARY_INTEGER;
 l_items_aa     items_aa;
BEGIN
  l_items_aa(1).item_id := 22;
  l_items_aa(1). inc_factor := 1.10;
  l_items_aa(2).item_id := 26;
  l_items_aa(2). inc_factor := 1.15;

  FORALL i IN l_items_aa.FIRST .. l_items_aa.LAST
   UPDATE items
         SET  item_value = item_value  * l_items_aa(i). inc_factor
    WHERE  item_id IN     l_items_aa(i).item_id;
  DBMS_OUTPUT.PUT_LINE('Rows updated  '||SQL%ROWCOUNT);
END;
```

Considerations

You can access just a part of the collection

```
DECLARE
 TYPE itemid_aa  IS TABLE OF PLS_INTEGER;
 l_itemid_aa     itemid_aa := itemid_aa(4,6,8,9,11);

BEGIN
  FORALL i IN 3 .. 5
   DELETE FROM items
    WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

Considerations

Inserting composite collection

```
DECLARE
   TYPE items_aa IS TABLE OF items%ROWTYPE INDEX BY BINARY_INTEGER;
   l_items_aa items_aa;
BEGIN

    l_items_aa(1).item_id:= 1;
    l_items_aa(1).item_name := 'Weights';
    l_items_aa(1).item_value:= 600;
    l_items_aa(2).item_id:= 2;
    l_items_aa(2).item_name := 'Bike';
    l_items_aa(2).item_value:= 600;
    FORALL i in l_items_aa.FIRST..l_items_aa.LAST
      INSERT INTO items
      VALUES l_items_aa(i);
END;
```

INDICES OF Clause

Sparse Collection

```
DECLARE
  TYPE itemid_aa  IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  l_itemid_aa     itemid_aa ;
BEGIN
  l_itemid_aa(1) := 3;
  l_itemid_aa(3) := 5;
  l_itemid_aa(7) := 5;
  FORALL i IN l_itemid_aa.FIRST ..l_itemid_aa.LAST
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

ORA-22160: Element at index[2] does not exist

INDICES OF Clause

Sparse Collection

```
DECLARE
  TYPE itemid_aa  IS TABLE OF NUMBER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
BEGIN
  l_itemid_aa(1) := 3;
  l_itemid_aa(3) := 5;
  l_itemid_aa(7) := 5;
  FORALL i IN INDICES OF l_itemid_aa
    DELETE FROM items
     WHERE item_id IN l_itemid_aa(i);
  DBMS_OUTPUT.PUT_LINE('Rows Deleted  '||SQL%ROWCOUNT);
END;
```

VALUES OF Clause

Iterate over specific elements	**Iterate in a specific order**	**Iterate over certain elements more than once**

Involves another reference collection

VALUES OF Clause

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa     itemid_aa ;
  l_second_aa     itemid_aa;
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 23;
  l_itemid_aa(3) := 25;
  l_itemid_aa(4) := 27;

  l_second_aa(1) := 3;
  l_second_aa(2) := 3;
  l_second_aa(3) := 2;
  l_second_aa(4) := 1;                                       3
  FORALL i IN VALUES OF l_second_aa          ------>         3
    UPDATE  items                                            2
          SET  item_value = item_value * 1.10                1
     WHERE  item_id = l_itemid_aa(i);
END;
```

FORALL & BULK COLLECT Together

```
DECLARE
  TYPE number_aa  IS TABLE OF PLS_INTEGER;
  l_orderid_aa      number_aa := number_aa(4,6,8);
  l_itemid_aa       number_aa ;

BEGIN
   FORALL i IN l_orderid_aa.FIRST .. l_orderid_aa.LAST
     DELETE FROM orders
      WHERE order_id IN l_orderid_aa(i)
      RETURNING item_id BULK COLLECT INTO l_itemid_aa;
   DBMS_OUTPUT.PUT_LINE('Items  Deleted  '||l_itemid_aa.COUNT);
END;
```

SQL%BULK_ROWCOUNT

Stores rows affected by each iteration of the DML statement

FORALL and SQL%BULK_ROWCOUNT use the same subscript

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa       itemid_aa ;
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 23;
  l_itemid_aa(3) := 25;
  FORALL i IN l_itemid_aa.FIRST..l_itemid_aa.LAST
    DELETE FROM orders
     WHERE  order_item_id = l_itemid_aa(i);
  FOR i IN l_itemid_aa.FIRST..l_itemid_aa.LAST LOOP
     DBMS_OUTPUT.PUT_LINE('Orders deleted for item_id '||l_itemid_aa(i)||' is '||SQL%BULK_ROWCOUNT(i));
  END LOOP;
END;
```

SQL%BULK_ROWCOUNT

FORALL and SQL%BULK_ROWCOUNT use the same subscript

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 23;
  l_itemid_aa(3) := 25;
  FORALL i IN 1..2
    DELETE FROM orders
     WHERE  order_item_id = l_itemid_aa(i);
  FOR i IN 1..2 LOOP
     DBMS_OUTPUT.PUT_LINE('Orders deleted for item_id '||l_itemid_aa(i)||' is '||SQL%BULK_ROWCOUNT(i));
  END LOOP;
END;
```

SQL%BULK_ROWCOUNT

FORALL and SQL%BULK_ROWCOUNT use the same subscript

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa     itemid_aa ;
  l_index         NUMBER;
BEGIN
  l_itemid_aa(1) := 3;          1  4  5
  l_itemid_aa(4) := 2;
  l_itemid_aa(5) := 1;
  FORALL i IN INDICES OF l_itemid_aa
     DELETE FROM orders
      WHERE  order_item_id = l_itemid_aa(i);
   l_index := l_itemid_aa.FIRST;   <----------  1
   WHILE l_index IS NOT NULL LOOP
      DBMS_OUTPUT.PUT_LINE('Orders deleted for item_id '||l_itemid_aa(l_index)||' is '||SQL%BULK_ROWCOUNT(l_index));
      l_index := l_itemid_aa.NEXT(l_index);   <----------  4   5
  END LOOP;
END;
```

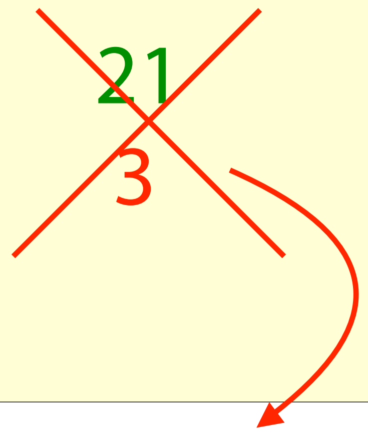
SQL%BULK_ROWCOUNT

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
  l_second_aa      itemid_aa;
  l_index          NUMBER;
  l_value          NUMBER;
BEGIN
  l_itemid_aa(2) := 21;
  l_itemid_aa(4) := 23;
  …                                          2 4
  l_second_aa(1) := 2;
  l_second_aa(3) := 4;
  FORALL i IN VALUES OF l_second_aa
    DELETE FROM orders
     WHERE  order_item_id = l_itemid_aa(i);
   l_index := l_second_aa.FIRST;             1                   21 23          2 4
   WHILE l_index IS NOT NULL LOOP
     l_value := l_second_aa(l_index);        2 4
      DBMS_OUTPUT.PUT_LINE('Orders deleted for item_id '||l_itemid_aa(l_value)||' is '||SQL%BULK_ROWCOUNT(l_value));
      l_index := l_second_aa.NEXT(l_index);          3
   END LOOP;
END;
```

EXCEPTIONS

Unhandled exception rolls back all previous changes

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 3;
  l_itemid_aa(3) := 25;
  l_itemid_aa(4) := 2;

  FORALL i IN l_itemid_aa.FIRST..l_itemid_aa.LAST
    DELETE FROM items
     WHERE  order_item_id = l_itemid_aa(i);
END;
```



EXCEPTIONS

Handling exception does not rolls back all previous changes

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 3;
  l_itemid_aa(3) := 25;
  l_itemid_aa(4) := 2;
  FORALL i IN l_itemid_aa.FIRST..l_itemid_aa.LAST
    DELETE FROM items
     WHERE  order_item_id = l_itemid_aa(i);
EXCEPTION
  WHEN OTHERS THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
    COMMIT;
END;
```

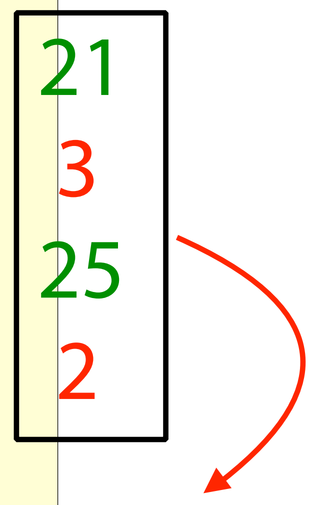
21 ✓

3

SAVE EXCEPTIONS

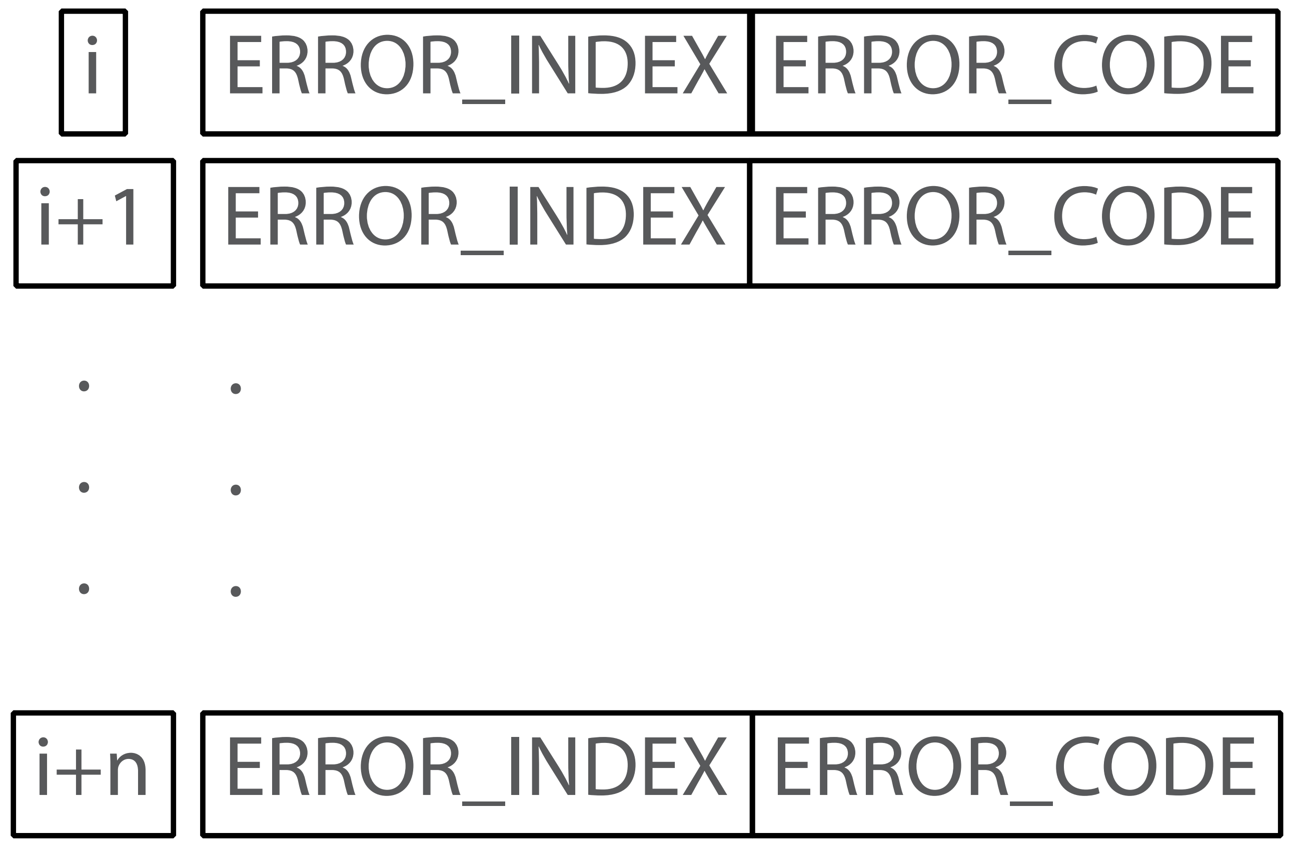
Saves exceptions and continues processing

```
DECLARE
  TYPE itemid_aa  IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa      itemid_aa ;
  array_dml_exception EXCEPTION;
  PRAGMA EXCEPTION_INIT (array_dml_exception, -24381);
BEGIN
  l_itemid_aa(1) := 21;
  l_itemid_aa(2) := 3;
  l_itemid_aa(3) := 25;
  l_itemid_aa(4) := 2;
  FORALL i IN l_itemid_aa.FIRST..l_itemid_aa.LAST  SAVE EXCEPTIONS
    DELETE FROM items
     WHERE  order_item_id = l_itemid_aa(i);
EXCEPTION
  WHEN array_dml_exception THEN
   DBMS_OUTPUT.PUT_LINE(DBMS_UTILITY.FORMAT_ERROR_STACK);
END;
```



SQL%BULK_EXCEPTIONS

Collection of records



Total Errors

SQL%BULK_EXCEPTIONS.COUNT

SQL%BULK_EXCEPTIONS

```
DECLARE
  TYPE itemid_aa IS TABLE OF PLS_INTEGER INDEX BY PLS_INTEGER;
  l_itemid_aa itemid_aa ;
  array_dml_exception EXCEPTION;
  PRAGMA EXCEPTION_INIT (array_dml_exception, -24381);
  l_error_count NUMBER;
BEGIN
  l_itemid_aa(1) := 4;
  l_itemid_aa(2) := 3;
  l_itemid_aa(3) := 25;
  l_itemid_aa(4) := 2;
  FORALL i IN l_itemid_aa.FIRST..l_itemid_aa.LAST SAVE EXCEPTIONS
   DELETE FROM items
   WHERE item_id = l_itemid_aa(i);
EXCEPTION
  WHEN array_dml_exception THEN
    l_error_count := SQL%BULK_EXCEPTIONS.COUNT;
    FOR i in 1..l_error_count LOOP
      DBMS_OUTPUT.PUT_LINE('Error occured at index '||SQL%BULK_EXCEPTIONS(i).ERROR_INDEX||' for error message '||
                                           SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
    END LOOP;
END;
```

SQL%BULK_EXCEPTIONS

```
l_itemid_aa(1) := 21;
l_itemid_aa(3) := 3;
l_itemid_aa(6) := 25;
l_itemid_aa(8) := 2;
```

1 3 6 8

2 4

```
 …
 …
 FORALL i IN INDICES OF l_itemid_aa SAVE EXCEPTIONS
   DELETE FROM items
   WHERE item_id = l_itemid_aa(i);
EXCEPTION
 WHEN array_dml_exception THEN
  l_error_count := SQL%BULK_EXCEPTIONS.COUNT;
  FOR i in 1..l_error_count LOOP
    l_counter := 0;
    l_index := l_itemid_aa.FIRST;
    WHILE l_index IS NOT NULL LOOP
      l_counter := l_counter + 1;
      IF l_counter = SQL%BULK_EXCEPTIONS(i).ERROR_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error occured at index '||l_index||' for error message '||SQLERRM(-SQL
                                        %BULK_EXCEPTIONS(i).ERROR_CODE));
      END IF;
      l_index := l_itemid_aa.NEXT(l_index);
    END LOOP;
  END LOOP;
END;
```

SQL%BULK_EXCEPTIONS

```
l_itemid_aa(1) := 21;
l_itemid_aa(3) := 3;
l_itemid_aa(6) := 25;
l_itemid_aa(8) := 2;
```

```
l_second_aa(1) := 1;
l_second_aa(2) := 6;
l_second_aa(3) := 3;
l_second_aa(4) := 8;
```

1 6 3 8

3 4

```
 …
 …
 FORALL i IN VALUES OF l_second_aa SAVE EXCEPTIONS
   DELETE FROM items
   WHERE item_id = l_itemid_aa(i);
EXCEPTION
 WHEN array_dml_exception THEN
  l_error_count := SQL%BULK_EXCEPTIONS.COUNT;
  FOR i in 1..l_error_count LOOP
      l_index := l_second_aa(SQL%BULK_EXCEPTIONS(i).ERROR_INDEX );
      DBMS_OUTPUT.PUT_LINE('Error occured at index '||l_index||' for error message '||
                                             SQLERRM(-SQL%BULK_EXCEPTIONS(i).ERROR_CODE));
  END LOOP;
END;
```

Summary

Usage

Sparse Collections

SQL%BULK_ROWCOUNT

Exceptions