Oracle for Developers (PL/SQL)

Introduction to PL/SQL

Lesson Objectives

To understand the following topics:

- Introduction to PL/SQL
- PL/SQL Block structure
- Handling variables in PL/SQL
- Declaring a PL/SQL table
- Variable scope and Visibility
- SQL in PL/SQL
- Programmatic Constructs

Overview

PL/SQL is a procedural extension to SQL.

- The “data manipulation” capabilities of “SQL” are combined with the “processing capabilities” of a “procedural language”.
- PL/SQL provides features like conditional execution, looping and branching.
  - PL/SQL supports subroutines, as well.
- PL/SQL program is of block type, which can be “sequential” or “nested” (one inside the other).

DML

- Insert, Update, Delate, Merge, Select

DDL

Create, Alter, Rename, Drop, Truncate, and Comment

TCL

Commit, Rollback, and Savepoint

DCL

Grant and Revoke

Salient Features

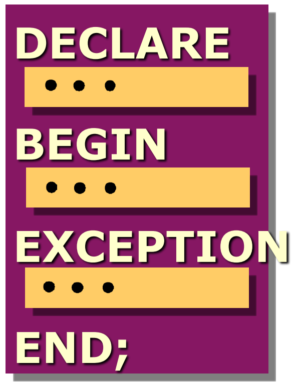
PL/SQL provides the following features:

- Tight Integration with SQL
- Better performance
  - Several SQL statements can be bundled together into one PL/SQL block and sent to the server as a single unit.
- Standard and portable language
  - Although there are a number of alternatives when it comes to writing software to run against the Oracle Database, it is easier to run highly efficient code in PL/SQL, to access the Oracle Database, than in any other language.

PL/SQL Block Structure

A PL/SQL block comprises of the following structures:

- DECLARE – Optional
  - Variables, cursors, user-defined exceptions
- BEGIN – Mandatory
  - SQL statements;
  - PL/SQL statements;
- EXCEPTION – Optional
  - Actions to perform when errors occur;
- END; – Mandatory



Block Types

There are two types of blocks in PL/SQL:

- Anonymous
- Named:
  - Procedure
  - Function

Anonymous

```
[DECLARE]

BEGIN
 --statements

[EXCEPTION]

END;
```

Procedure

```
PROCEDURE name
IS | AS

BEGIN
 --statements

[EXCEPTION]

END;
```

Function

```
FUNCTION name
RETURN datatype
IS | AS
BEGIN
 --statements
 RETURN value;
[EXCEPTION]

END;
```

Points to Remember

While handling variables in PL/SQL:

- declare and initialize variables within the declaration section
- assign new values to variables within the executable section
- pass values into PL/SQL blocks through parameters
- view results through output variables

Guidelines for declaring variables

Given below are a few guidelines for declaring variables:

- follow the naming conventions
- initialize the variables designated as NOT NULL
- initialize the identifiers by using the assignment operator (:=) or by using the DEFAULT reserved word
- Declare at most one Identifier per line

Types of Variables

PL/SQL variables

- Scalar
- Composite
- Reference
- LOB (large objects)

Non-PL/SQL variables

- Bind, Host , Environment or Session variables

Declaring PL/SQL variables

Syntax

```
identifier [CONSTANT] datatype [NOT NULL]
                [:= | DEFAULT expr];
```

Example

```
DECLARE
    v_hiredate      DATE;
    v_deptno        NUMBER(2) NOT NULL := 10;
    v_location      VARCHAR2(13) := 'Atlanta';
    c_comm  CONSTANT NUMBER := 1400;
```

Base Scalar Data Types

Base Scalar Datatypes:

- Given below is a list of Base Scalar Datatypes:
  - VARCHAR2 (maximum_length)
  - NUMBER [(precision, scale)]
  - DATE
  - Timestamp
  - CHAR [(maximum_length)]
  - LONG
  - LONG RAW $\Big\}$ LOB
  - BOOLEAN
  - BINARY_INTEGER
  - PLS_INTEGER

Base Scalar Data Types - Example

Here are a few examples of Base Scalar Datatypes:

```
v_job          VARCHAR2(9);
v_count        BINARY_INTEGER := 0;
v_total_sal    NUMBER(9,2) := 0;
v_orderdate    DATE := SYSDATE + 7;
c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
v_valid        BOOLEAN NOT NULL := TRUE;
```

Declaring Datatype by using %TYPE Attribute

While using the %TYPE Attribute:

- Declare a variable according to:
  - a database column definition
  - another previously declared variable
- Prefix %TYPE with:
  - the database table and column
  - the previously declared variable name

Declaring Datatype by using %TYPE Attribute

Example:

```
…
v_name                    staff_master.staff_name%TYPE;
v_balance                 NUMBER(7,2);
v_min_balance  v_balance%TYPE := 10;
…
```

Declaring Datatype by using %ROWTYPE

Example:

```
DECLARE
      nRecord                staff_master%rowtype;
BEGIN
      SELECT * into nrecord
             FROM staff_master
             WHERE staff_code = 100001;

             UPDATE staff_master
             SET staff_sal = staff_sal + 101
             WHERE emp_code = 100001;
END;
```

1.3: Handling Variables in PL/SQL

Composite Data Types

Composite Datatypes in PL/SQL:

- Three composite datatypes are available in PL/SQL:
  - Records (only 1 row) e.g. **%ROWTYPE**, **User Defined Record Type**
  - tables (multiple record)
  - Varray
- A composite type contains components within it. A variable of a composite type contains one or more scalar variables.

- To process multiple record in PLSQL you need either table type variable or Explicit Cursor.

Record Data Types

Record Datatype:

- A record is a collection of individual fields that represents a row in the table.
- They are unique and each has its own name and datatype.
- The record as a whole does not have value.

Defining and declaring records:

- %ROWTYPE.
- Decalre User Defined RECORD type, then declare records of that type.
- Define in the declarative part of any block, subprogram, or package.

Record Data Types (User Defined Record Type)

Syntax:

```
TYPE type_name IS RECORD (field_declaration [,field_ declaration] ...);
```

Record Data Types - Example

Here is an example for declaring Record datatype:

```
DECLARE
TYPE DeptRec IS RECORD (
Dept_id        department_master.dept_code%TYPE,
Dept_name      varchar2(15),
```

Record Data Types - Example

Here is an example for declaring and using Record datatype:

```
DECLARE
        TYPE recname is RECORD
        (customer_id number,
         customer_name varchar2(20));
        var_rec    recname;
BEGIN
        var_rec.customer_id:=20;
        var_rec.customer_name:=‘Smith';
        dbms_output.put_line(var_rec.customer_id||'
'||var_rec.customer_name);
END;
```

Table Data Type

A PL/SQL table is:

- a one-dimensional, unbounded, sparse collection of homogeneous elements
- indexed by integers
- In technical terms, a PL/SQL table:
  - is like an array
  - is like a SQL table; yet it is not precisely the same as either of those data structures
  - is one type of collection structure
  - is PL/SQL's way of providing 2D arrays

Table Data Type

Declaring a PL/SQL table:

- There are two steps to declare a PL/SQL table:
  - Declare a TABLE type structure.
  - Declare PL/SQL tables variable of that type.

```
TYPE type_name is TABLE OF (
{data_type | table.column%type} [NOT NULL] | table_name%ROWTYPE
INDEX BY BINARY_INTEGER | varchar2(2));
```

- If the column is defined as NOT NULL, then PL/SQL table will reject NULLs.

Table Data Type - Examples

Example 1:

- To create a PL/SQL table named as "student_table" of char column.

```
DECLARE
TYPE student_table is table of char(10)
INDEX BY BINARY_INTEGER;
```

Example

- To create "student_table" based on the existing column of "student_name" of EMP table.

```
DECLARE
TYPE student_table is table of student_master.student_name%type
INDEX BY BINARY_INTEGER;
```

Table Data Type - Examples

After defining type emp_table, define the PL/SQL tables of that type.

For Example:

- These t
- You can

For Ex

```
Student_tab   student_table;
```

```
Student_tab :=('SMITH','JONES','BLAKE'); --Illegal
```

Referencing PL/SQL Tables

Here is an example of referencing PL/SQL tables:

```
DECLARE
   TYPE staff_table is table of
    staff_master.staff_name%type
    INDEX BY BINARY_INTEGER;
staff_tab staff_table;
BEGIN
    staff_tab(1) := ‘Smith'; --update Smith's salary
    UPDATE staff_master
    SET staff_sal = 1.1 * staff_sal
    WHERE staff_name = staff_tab(1);
END;
```

Referencing PL/SQL Tables - Examples

- To assign values to specific rows, the following syntax is used:

```
PLSQL_table_name(index_key_value) := PLSQL expression;
```

- From ORACLE 7.3, the PL/SQL tables allow records as their columns.

System Defined row type

System Defined row type

v_rec emp%ROWTYPE

V_rec

empno	Ename	job	mgr	hiredate	Sal	Comm	deptno

v_rec.ename
v_rec.sal
v_rec.deptno

User Defined row type

```
TYPE  emp_row_type   IS RECORD (ename   emp.ename% TYPE,  job
emp.job% TYPE,  sal   emp.sal% TYPE, deptno  emp.deptno% TYPE);
v_rec      emp_row_type  ;
```

Ename	Job	Sal	Deptno

User Defined Table Type

```
TYPE  emp_tbl_type   IS TABLE OF  emp%ROWTYPE  INDEX  BY
binary_integer  ;
V_rec                 emp_tbl_type;
```

INDEX	em pno	Ename	job	mgr	hiredat e	Sal	Com m	deptno
1								
2								
3								
4								
5								
6								
7								
8								

User Defined Table Type

```
TYPE  emp_tbl_type   IS TABLE OF  emp.ename%YPE  INDEX
BY binary_integer  ;
V_rec                  emp_tbl_type;
```

INDEX	ENAME
1	
2	
3	
4	

Scope and Visibility of Variables

Scope of Variables:

- The scope of a variable is the portion of a program in which the variable can be accessed.
- The scope of the variable is from the “variable declaration” in the block till the “end” of the block.
- When the variable goes out of scope, the PL/SQL engine will free the memory used to store the variable, as it can no longer be referenced.

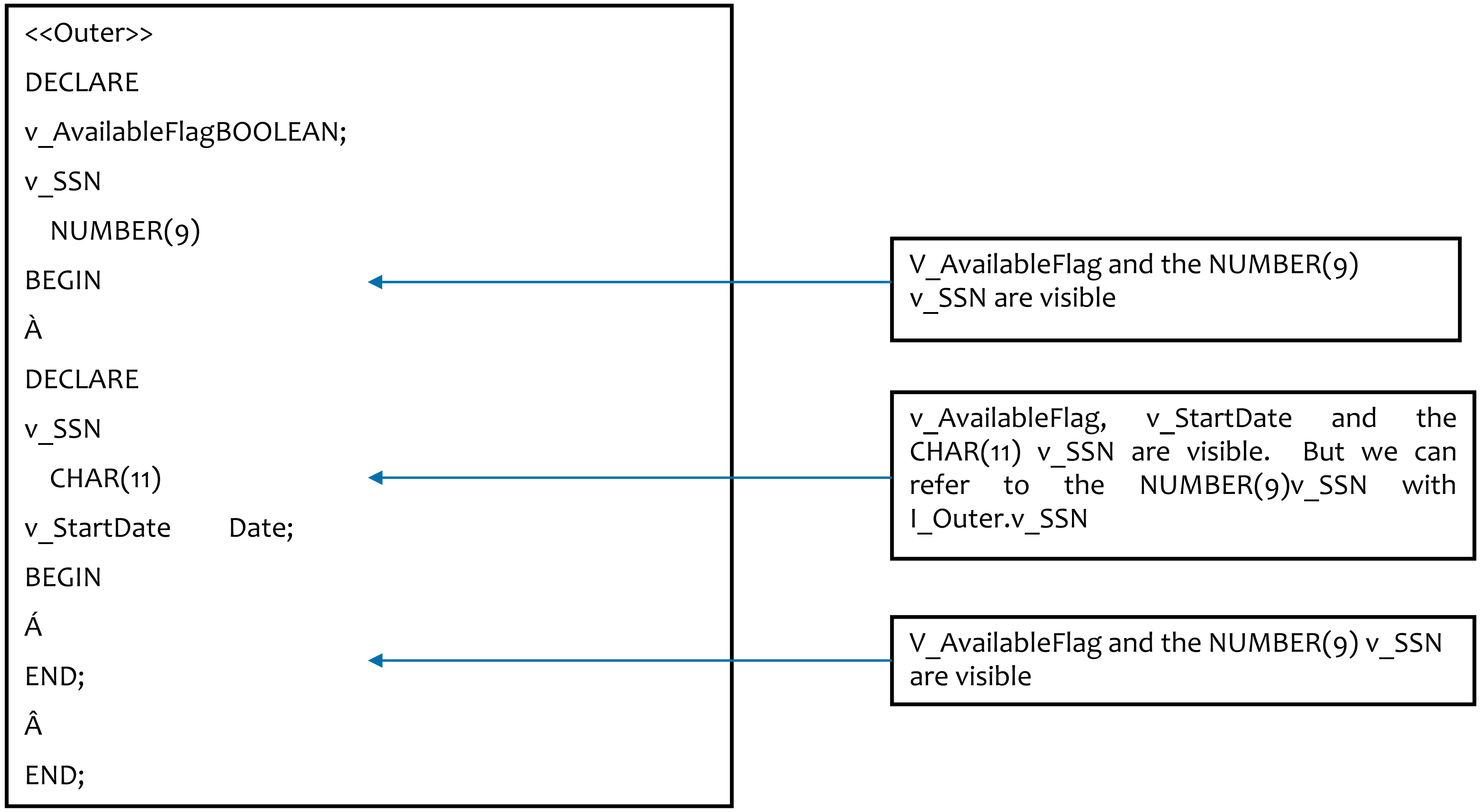
Scope and Visibility of Variables

Visibility of Variables:

- The visibility of a variable is the portion of the program, where the variable can be accessed without having to qualify the reference. The visibility is always within the scope, it is not visible.

Scope and Visibility of Variables

Pictorial representation of visibility of a variable:



```
<<Outer>>
DECLARE
v_AvailableFlagBOOLEAN;
v_SSN
   NUMBER(9)
BEGIN
À
DECLARE
v_SSN
   CHAR(11)
v_StartDate      Date;
BEGIN
Á
END;
Â
END;
```

V_AvailableFlag and the NUMBER(9) v_SSN are visible

v_AvailableFlag, v_StartDate and the CHAR(11) v_SSN are visible. But we can refer to the NUMBER(9)v_SSN with l_Outer.v_SSN

V_AvailableFlag and the NUMBER(9) v_SSN are visible

Scope and Visibility of Variables

```
<<OUTER>>
DECLARE
V_Flag  BOOLEAN ;
V_Var1  CHAR(9);
BEGIN
<<INNER>>
DECLARE
V_Var1  NUMBER(9);
V_Date  DATE;
BEGIN
NULL;
END;
NULL;
END;
```

1.6: SQL in PL/SQL

Types of Statements

Given below are some of the SQL statements that are used in PL/SQL:

- INSERT statement
  - The syntax for the INSERT statement remains the same as in SQL-INSERT.
  - For example:

```
DECLARE
        v_dname varchar2(15) := ‘Accounts';
BEGIN
        INSERT into department_master
      VALUES (50, v_dname);
END;
```

Types of Statements

- DELETE statement

For Example:

```
DECLARE
        v_sal_cutoff  number  := 2000;
BEGIN
        DELETE FROM staff_master
     WHERE  staff_sal < v_sal_cutoff;
END;
```

Types of Statements

- UPDATE statement

For Example:

```
DECLARE
       v_sal_incr  number(5) := 1000;
BEGIN
       UPDATE staff_master
     SET staff_sal = staff_sal + v_sal_incr
     WHERE staff_name='Smith';
END;
```

Types of Statements

- SELECT statement
  - Syntax:

```
SELECT Column_List INTO Variable_List
        FROM Table_List
        [WHERE expr1]
        CONNECT BY expr2 [START WITH expr3]]
        GROUP BY expr4] [HAVING expr5]
        [UNION | INTERSECT | MINUS SELECT ...]
        [ORDER BY expr | ASC | DESC]
        [FOR UPDATE [OF Col1,...] [NOWAIT]]
        INTO Variable_List;
```

Types of Statements

The column values returned by the SELECT command must be stored in variables.
The Variable_List should match Column_List in both COUNT and DATATYPE.
Here the variable lists are PL/SQL (Host) variables. They should be defined before use.

Types of Statements

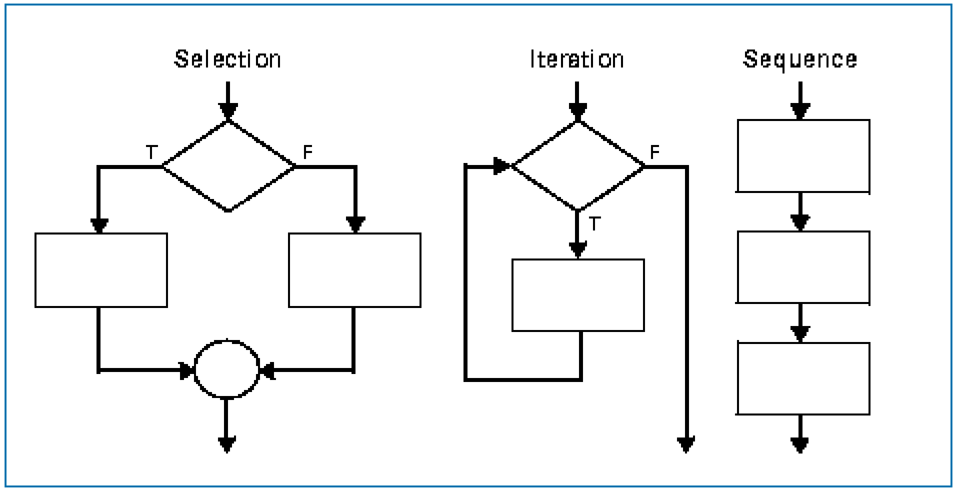
Example: <<BLOCK1>>

```
DECLARE
        deptno   number(10)  := 30;
        dname   varchar2(15) ;
BEGIN
        SELECT dept_name INTO dname FROM department_master
WHERE dept_code = Block1. deptno;
        DELETE FROM department_master
                                WHERE dept_code = Block1. deptno ;
END;
```

Programmatic Constructs in PL/SQL

Programmatic Constructs are of the following types:

- Selection structure
- Iteration structure
- Sequence structure



IF Construct

Given below is a list of Programmatic Constructs which are used in PL/SQL:

- Conditional Execution:
  - This construct is used to execute a set of statements only if a particular condition is TRUE or FALSE.
  - Syntax:

```
IF Condition_Expr  THEN
        PL/SQL_Statements
END IF;
```

IF Construct - Example

For Example:

```
IF v_staffno = 100003
THEN
      UPDATE staff_master
      SET staff_sal = staff_sal + 100
      WHERE staff_code = 100003 ;
END IF;
```

IF Construct - Example

To take alternate action if condition is FALSE, use the following syntax:

```
IF Condition_Expr THEN
        PL/SQL_Statements_1 ;
ELSE
        PL/SQL_Statements_2 ;
END IF;
```

IF Construct - Example

To check for multiple conditions, use the following syntax.

```
IF Condition_Expr_1 THEN
                PL/SQL_Statements_1 ;
     ELSIF Condition_Expr_2  THEN
                PL/SQL_Statements_2 ;
     ELSIF Condition_Expr_3  THEN
                PL/SQL_Statements_3 ;
     ELSE
                PL/SQL_Statements_n ;
END IF;
```

- Note: Conditions for NULL are checked through IS NULL and IS NOT NULL predicates.

Basic Loop

Looping

- A LOOP is used to execute a set of statements more than once.
- Syntax:

```
LOOP
        PL/SQL_Statements;
        EXIT WHEN <condition>;
END LOOP ;
```

Basic Loop

For example:

```
DECLARE
        v_counter  number := 50 ;
BEGIN
LOOP
        INSERT INTO department_master
                        VALUES(v_counter,'new dept');
        v_counter := v_counter + 10 ;
END LOOP;
        COMMIT ;
END ;
/
```

Basic Loop – EXIT statement

EXIT

- Exit path is provided by using EXIT or EXIT WHEN commands.
- EXIT is an unconditional exit. Control is transferred to the statement following END LOOP, when the execution flow reaches the EXIT statement.

contd.

Basic Loop – EXIT statement

Syntax:

```
BEGIN
        .....
        .....
        LOOP
                IF <Condition> THEN
        . . . . .
                EXIT ;                  -- Exits loop immediately
                END IF ;

        END LOOP;
        LOOP
                ... ... ... .
                ... ... ... .
                EXIT WHEN <condition>
        END LOOP;

        . . . . .                               -- Control resumes here
        COMMIT ;
        END ;
```

Basic Loop – EXIT statement

For example:

```
DECLARE
        v_counter number := 50 ;
BEGIN
        LOOP
            INSERT INTO department_master
           VALUES(v_counter,'NEWDEPT');
        DELETE  FROM emp WHERE deptno = v_counter;
                         v_counter := v_counter + 10 ;
                        EXIT WHEN v_counter >100 ;
        END LOOP;
        COMMIT ;
END ;
```

- Note: As long as v_counter has a value less than or equal to 100, the loop continues.

For Loop

FOR Loop:

- Syntax:

```
FOR Variable IN [REVERSE] Lower_Bound . . Upper_Bound
LOOP
        PL/SQL_Statements
END LOOP ;
```

While Loop

WHILE Loop

- The WHILE loop is used as shown below.
- Syntax:

```
WHILE Condition
LOOP
        PL/SQL  Statements;
END LOOP;
```

- EXIT OR EXIT WHEN can be used inside the WHILE loop to prematurely exit the loop.

Sequential statement

```
declare
v_design  varchar2(20) := '&designation';
v_salary   number(10,2) := &salary;
v_revised_sal          number(10,2);
begin
v_revised_sal := CASE v_design
                                                WHEN 'MANAGER' THEN v_salary + v_salary*.3
                                                WHEN 'ANALYST' THEN v_salary + v_salary*.2
                                                WHEN 'CLERK'   THEN v_salary + v_salary*.1
                                                ELSE v_salary
              END;
  dbms_output.put_line('Current salary is ' || v_salary || ' and revised salary is ' ||
v_revised_sal);
end;
/
```

Labeling of Loops

Labeling of Loops:

- The label can be used with the EXIT statement to exit out of a particular loop.

```
BEGIN
        <<Outer_Loop>>
        LOOP
                PL/SQL
                << Inner_Loop>>
                LOOP
                        PL/SQL  Statements ;
                        EXIT Outer_Loop WHEN <Condition Met>
                END LOOP  Inner_Loop
        END LOOP  Outer_Loop
END ;
```

Summary

In this lesson, you have learnt:

- PL/SQL is a procedural extension to SQL.
- PL/SQL exhibits a block structure, different block types being: Anonymous, Procedure, and Function.
- While declaring variables in PL/SQL:
  - declare and initialize variables within the declaration section
  - assign new values to variables within the executable section
  - pass values into PL/SQL blocks through parameters
  - view results through output variables

Summary

- Different types of PL/SQL Variables are: Scalar, Composite, Reference, LOB
- Scope of a variable: It is the portion of a program in which the variable can be accessed.
- Visibility of a variable: It is the portion of the program, where the variable can be accessed without having to qualify the reference.
- Different programmatic constructs in PL/SQL are Selection structure, Iteration structure, Sequence structure

Review – Questions

Question 1: A record is a collection of individual fields that represents a row in the table.

- True/ False

Question 2: %ROWTYPE is used to declare a variable with the same datatype as a column of a specific table.

- True / False

Question 3: PL/SQL tables use a primary key to give you array-like access to rows.

- True / False

Review – Questions

Question 4: While using FOR loop, Upper_Bound, and Lower_Bound must be integers.

- True / False