Tech Mahindra

PLSQL Introduction

Language Categories

- To know what is PL/SQL, first we have to understand different types of programming languages.

- There are at least four ways to categorize popular languages.

- **Procedural programming languages**
  - Allow the programmer to define an ordered series of steps to follow in order to produce a result. Examples: PL/SQL, C, Visual Basic, Perl, Ada.
- **Object-oriented programming languages**
  - Based on the concept of an *object*, which is a data structure encapsulated with a set of routines, called *methods* that operate on the data. Examples: Java, C++, JavaScript, and sometimes Perl and Ada 95.

Language Categories

- **Declarative programming languages**
  - Allow the programmer to describe relationships between variables in terms of functions or rules; the language executor (interpreter or compiler) applies some fixed algorithm to these relations to produce a result. Examples: Logo, LISP, and Prolog.
- **Markup languages**
  - Define how to add information into a document that indicates its logical components or that provides layout instructions. Examples: HTML, XML.

- ***Structured Query Language***

  - *It* is a language based on set theory, so it is all about manipulating sets of data. SQL consists of a relatively small number of main commands such as SELECT, INSERT, CREATE, and GRANT; in fact, each statement accomplishes what might take hundreds of lines of procedural code to accomplish. That's one reason SQL-based databases are so widely used.

Learning Oracle PL/SQL

- PL/SQL, Oracle's programming language for stored procedures, delivers a world of possibilities for your database programs.

- PL/SQL supplements the standard relational database language, SQL, with a wide range of procedural features, including loops, IF-THEN statements, advanced data structures, and rich transactional control--all closely integrated with the Oracle database server.

- PL/SQL is a procedural structured Query Language, is an extension to SQL where we can write programs using all the SQL statements and procedural statements.

Procedural Statements – PL/SQL

- Assignment statements
- Conditional statements
- Loops
- Transactional processing statements

Assignment statements

- In C like programming language, we use = as assignment operator and == as comparison operator. Where as in PL/SQL we use = as comparison operator and := as assignment operator and the statement used with this operator is called as assignment statement. An example of assignment is shown below.

- c := a + b;

Conditional statements

- Generally, in any programming language, we use If statement as conditional statement.
- Various formats of ***IF*** statement can be used in programming.

Simple If:

```
IF <CONDITION>  THEN
    ST1;
    ST2;
ELSE
    ST3;
END IF;
```

ElseIf Ladder:

```
IF <CONDITION1> THEN
    Statements;
ELSIF <CONDITION2> THEN
    Statements;
ELSIF <CONDITION3> THEN
    Statements;
ELSE
    Statements;
END IF;
```

In addition to these IF statements, we can also use SQL functions DECODE () and CASE.

Loops

- In any programming language, we use tow different types of Loops
  - Conditional Loops
  - Iterative Loop
- ***For Loop***
- For is an iterative loop. This loop performs operation for a range of values.

```
FOR loop_counter IN [REVERSE] lower_bound .. upper_bound
LOOP
   Statements
END LOOP;
```

Example

```
FOR month_num IN 1 .. TO_NUMBER(TO_CHAR(SYSDATE, 'MM'))
LOOP
   Statements;
END LOOP;
```

Guidelines to use FOR Loop

- Use a FOR loop to shortcut the test for the number of iterations
- Do not declare the counter; it is declared implicitly
- lower bound .. upper bound is required
- Reference the counter within the loop only; it is undefined outside the loop.
- Do not reference the counter as the target of an assignment.

Conditional Loops

- ***Simple Loop***
  - The simple loop is the, well, simplest loop structure. It has the following syntax:

```
LOOP
   EXIT WHEN <condition>;
   Statements;
END LOOP;
```

We can use this exit condition any where within the loop.

While Loop

```
WHILE <condition>
LOOP
   statements;
End LOOP;
```

Transactional Processing Statements

- We use COMMIT and ROLLBACK as transactional processing statements. Once the transaction is over then you decide whether you save the data or not to save the data.

- **Note**: PL/SQL supports 4GL (object oriented programming language) features.

Programming structure

- PL/SQL also uses two different types of block structures in writing programs.
  - Unnamed PL/SQL block structure
  - Named PL/SQL block structure

Unnamed PL/SQL block structure

- They are also called as anonymous blocks; they do not have names in the database. Its code looks as follows:

```
DECLARE
   Declaration section  (optional)
BEGIN
   Executable section   (mandatory)
EXCEPTION
   Exception section    (optional)
END;
```

Explanation

Section	Description	Inclusion
Declarative	Contains all variables, constants, cursors, and user-defined exceptions that are referenced in the executable and declarative sections	Optional
Executable	Contains SQL statements in manipulate data in the database and PL/SQL statements to manipulate data in the block	Mandatory
Exception	Specifies the sections to perform when errors and abnormal conditions arise in the executable section	Optional

Errors in PL/SQL Programming

- The section between EXCEPTION and END is referred to as the exception section. The exception section traps error conditions.

- Errors are of two types.
  - Primitive errors
  - Logical or runtime errors
    - Primitive (syntax) errors are handled at the time of compiling the program itself.
    - Logical or runtime errors are handled only at runtime.
    - These logical errors are handled in the EXCEPTION SECTION.
    - This section is optional.

Identifiers/Variables

- **Naming Rules for Identifiers**
  - Identifiers are the names given to PL/SQL elements such as variables, procedures, variables, and user-defined types.
  - These names must follow certain rules of the road; namely, they:

    - Are no more than 30 characters in length. Start with a letter.
    - Consist of any of the following: letters, numerals, the dollar sign ($), the hash sign (#), and the underscore (_).
    - Cannot be the same as a PL/SQL reserved word.
    - Are unique within its scope. You cannot, in other words, declare two variables with the same name in the same block of code.

Scope of Variables

- The *scope* of a variable is the portion of PL/SQL code in which that variable can be referenced (i.e., its value read or modified). Most of the variables you define will have as their scope the block in which they were defined.

- Example:

```
DECLARE
   v_book_title VARCHAR2 (100);
BEGIN
   v_book_title := 'Learning Oracle PL/SQL';
END;
/
```

Scope of variable

- The variable **v_*book*_*title*** can be referenced within this block of code, but nowhere else. So if it happens to write another separate block of code, any attempt to read or change the value of **v_*book*_*title*** will result in a compilation error. Consider the block below that is assumed to be the continuation of the above block.

```
SQL> BEGIN
2    IF v_book_title LIKE '%PL/SQL%'
3    THEN
4       buy_it;
5    END IF;
6 END;
```

In this case the above block will produce a compilation because it has a variable **v_book_title**, that is not defined in that block. We can not access the variables in another block here.

Types of Variables

- PL/SQL variables
  - Scalar
  - Composite
  - Reference
  - LOB(large objects)
- Non-PL/SQL Variables
  - They include host language variables declared in precompiler programs, screen fields in Forms applications, and SQL*Plus or iSQL*Plus host variables

Declaration of variables

- A variable in PL/SQL can be declared in two different ways
  - By using the predefined/user defined data types and sizes
  - By referencing to a predefined column of a database object

Declaring variable by giving its data type and size

  - **Syntax**: Identifier [CONSTANT] DATATYPE [NOT NULL] [:= | DEFAULT expr];

- Example:

```
v_joindate    DATE;
v_empno       NUMBER(3) NOT NULL := 1001;
v_location    VARCHAR2(10)             := ‘sec-bad’;
c_comm.       CONSTANT NUMBER    := 1400;
```

Note

- **=** Used as comparison operator and **:=** as assignment operator.
- Naming convention for variable, variable name starts with “v” and constant start with “c”.
- We can also give integrity constraints at the time of declaring variableVename varchar2 (10) not null.
- Only one variable is allowed to declare in each line.
- Follow naming conventions
  - Variable Name starts with V_
  - Constant Name starts with C_
  - Global variable starts with G_

Declaring variable by using *%TYPE* Attribute

- A variable declared by referring column of a table. We can also use %TYPE attribute to declare a variable according to another previously declared variable.

```
v_empno     emp.empno%TYPE;
v_salary    NUMBER(8,2);
v_comm.     v_salary%TYPE;
```

A NOT NULL database column constraint does not apply to variables that are declared using %TYPE. Therefore, if you declare a variable using the %TYPE attribute that uses a database column defined as NOT NULL, you can assign the NULL value to the variable.

Scalar Data type variables

- Scalar Variable
  - Hold a single value
  - Have no internal components
- ***Base Scalar Data Types***

Data Type	Description
CHAR [(max-Len)]	Base type for fixed-length character data up to 32,767 bytes. If you do not specify a maximum length, the default length is set to 1.
VARCHAR2 (max-Len)	Base type for variable-length character data up o 32,767 bytes. There is no default size for VARCHAR2 variables and constants.
LONG	Base type for variable-length character data up to 32,760 bytes. Use the LONG data type to store variable-length character strings. You can insert any LONG value into a LONG database column because the maximum width of a LONG column is 2 ** 31 bytes. However, you cannot retrieve a value longer than 32760 bytes from a LONG column into a LONG variable.
LONG RAW	Base type for binary data and byte strings up to 32,760 bytes.
NUMBER(precision, scale)	Number having precision p and scale s. the precision p can range from 1 to 38.
BINARY_INTEGER	Base type for integers between –2,147,483,647 and 2,147,483,647
PLS_INTEGER	BASE type for signed integers between –2,147,483,647 and 2,147,483,647. PLS_INTEGER values require less storage and are faster than NUMBER and BINARY_INTEGER values.
BOOLEAN	Base type that stores one of three possible values
DATE	Base type for date and time

Composite Data type variables

- A scalar type has no internal components. A composite type has internal components that can be manipulated individually. Composite data types (Also known as collections) are TABLE, RECORD, NESTED.

- **Syntax**: <variable> <tablename>%ROWTYPE;
  - **v_rec emp%ROWTYPE;**

LOB Data Type Variables

- With LOB (large object) data types you can store blocks of unstructured data (such as text, graphic, images, video clips and sound wave forms) up to 4GB in size.
- LOB data types allow efficient, random, piecewise access to the data and can also be used as attributes of an object type. LOBs also support random access to data.
  - **CLOB** (Character large object) data type is used to store large blocks of single-byte character data in the database.
  - **BLOB** (Binary large object) data type is used to store large binary objects in the database.
  - **BFILE** (Binary file) data type is used to store large binary objects in operating system files outside the database.
  - **NCLOB** (National language character large object) data type is used to store large blocks of single-byte or fixed-width multi-byte NCHAR Unicode data in the database, in line or out of line.

Use of variables

- Variables can be used for:
  - Temporary storage of data
  - Manipulation of stored values
  - Reusability
  - Ease of maintenance

Executable Section

- BEGIN
- Signals the start of the executable part of a PL/SQL block, which contains executable statements. A PL/SQL block must contain at least one executable statement (even just the NULL statement).

- In this section user is going to write the program logic using all the
  - SQL statements
  - Assignment statements
  - Conditional Statements
  - Loops
  - Transactional Processing statements

DBMS_OUTPUT Built-in

- It is a Built in package object, used to display output messages.
- The DBMS_OUTPUT package enables you to send messages from stored procedures, packages, and triggers. The package is especially useful for displaying PL/SQL debugging information

- This package comes with different procedures
  - 1. PUT_LINE()
    - This procedure places a line in the buffer.
  - 2. PUT()
    - This procedure places a partial line in the buffer.
    - Used to place the information adjacent to each other
  - 3. NEW_LINE()
    - This procedure puts an end-of-line marker.

The SELECT INTO Clause

- The SELECT INTO is actually a standard SQL query where the SELECT INTO clause is used to place the returned data into predefined variables.

- Example
  - SELECT SAL INTO V_SAL FROM EMP WHERE EMPNO = 7521;
  - SELECT ENAME INTO V_ENAME FROM EMP WHERE EMPNO = 7521
  - SELECT EMPNO,ENAME INTO V_EMPNO,V_ENAME FROM EMP WHERE EMPNO = 7521;

Program

- Write a program to maintain students information with Rollno, Student_name, sub1, sub2, total, average where sub1, sub2 are the marks in different subjects.

- Solution
  - Step 1 create student table with above structure
  - Step 2 In SQL*plus environment, open the system editor using a command
    - SQL> EDIT student

      Or
    - In iSQL*plus environment, write the code in Executable window and save the script into a file.

Solution

```
DECLARE
   v_stu student%ROWTYPE;
BEGIN
   -- generate rollno automatically
   SELECT NVL(MAX(rollno),0) + 1 INTO v_stu.rollno FROM student;
   -- input name and marks in two subjects
   v_stu.sname    := ‘&name’;
   v_stu.sub1     := &marks1;
   v_stu.sub2     := &marks2;
   v_stu.total    := v_stu.sub1 + v_stu.sub2;
   v_stu.average  := v_stu.total/2;
   INSERT INTO STUDENT VALUES v_stu;
   COMMIT;
END;
/
```

To run PL/SQL program From SQL*Plus Environment

SQL> START student

Or

SQL> @student or use Execute button from ISQL*plus

Bind Variables

- A bind variable is a variable that you declare in a host environment. Bind variables can be used to pass run-time values, either number or character, into or out of one or more PL/SQL programs.

- The PL/SQL program uses bind variables as they would use any other variable. You can reference variables declared in the host or calling environment in PL/SQL statements, unless the statement is in a procedure, function, or package.

Creating Bind Variables

- To declare a bind variable in the iSQL*Plus environment, use the command VARIABLE.

- For example, we can declare a variable of type number and VARCHAR2 as follows:
  - **VARIABLE v_sal NUMBER**
  - **VARIABLE v_getmsg AS VARCHAR2(30)**

- Both SQL and iSQL*Plus environments can reference the bind variable, and iSQL*Plus can display its value through PRINT command

Using Bind Variables

- To reference a bind variable in PL/SQL, you must prefix its name with a colon (:)

```
VARIABLE g_salary number
BEGIN
   SELECT sal INTO :g_salary FROM EMP WHERE empno = 7521;
END;
/
PRINT g_salary
```

Referencing Non-PL/SQL Variables

- To reference host-variables, you must prefix the references with an ampersand (&) to distinguish them from declared PL/SQL variables

```
VARIABLE g_month_sal NUMBER
DEFINE   p_annual_sal=50000
SET VERIFY OFF

DECLARE
   v_sal  NUMBER(9,2) := &p_annual_sal;  -- reads the value from host variable
BEGIN
    :g_monthly_sal := v_sal/12;
END;
/
PRINT  g_monthly_sal
```

Note: The DEFINE command specifies a user variable and assigns it a CHAR value. Even though you enter the number 50000, iSQL*Plus assigns a CHAR value to p_annual_sal consisting of the characters 5,0,0,0 and 0.

Embedding Single Quotes inside a String

- The trickiest part of working with string literals comes when you need to include a single quote inside a string literal (as part of the literal itself). Generally, the rule is that you write two single quotes next to each other inside a string if you want the literal to contain a single quote in that position.

Literal	Actual Value
‘There’ ‘s no business like show business’	There’s no business like show business
‘ “INSTITUTE FOR TECHNOLOGY” ‘	“INSTITUTE FOR TECHNOLOGY”
‘NLS LANGUAGE= ‘ ‘ENGLISH”’	NLS LANGUAGE = ‘ENGLISH’
‘ ’ ’ ’	‘
‘ ’ ‘HELLO’ ‘ ‘	‘HELLO’

```
SQL> set serveroutput on;
SQL> BEGIN
  2      DBMS_OUTPUT.PUT_LINE('There''s no business like show business');
  3  END;
  4  /
There's no business like show business

PL/SQL procedure successfully completed.
```

Programming Guidelines

- Make code maintenance easier by:
  - Documenting code with comments
  - Developing a case convention for the code
  - Developing naming conventions for identifiers and other objects
  - Enhancing readability by Indenting

Code Conventions

Category	Case Convention	Examples
SQL statements	Uppercase	SELECT, INSERT
PL/SQL keywords	Uppercase	DECLARE,BEGIN,IF
Data types	Uppercase	VARCHAR2, BOOLEAN
Identifiers and parameters	Lowercase	v_sal_emp_cursor, p_sal
Database tables and columns	Lowercase	emp, empno

Exercise 1

- Evaluate each of the following declarations. Determine which of them are not legal and explain why.
  - v_id NUMBER(4);
  - v_x,v_y,v_z VARCHAR2(10);
  - V_birthdate DATE NOT NULL;
  - v_in_stock BOOLEAN :=1;

Exercise 2

- Identify the values at the arrow positions

```
DECLARE
      v_weight   NUMBER(3) := 600;
      v_message  VARCHAR2(255) := 'Product 10012';
BEGIN
      DECLARE
         v_weight    NUMBER(3) := 1;
         v_message   VARCHAR2(255):='Product 11001';
         v_new_locn  VARCHAR2(50) := 'Europe';
       BEGIN
         v_weight    := v_weight  + 1;
         v_new_locn  :=  'Western ' || v_new_locn;
1   <--------------------------------

       END;
       v_weight    := v_weight + 1;
       v_message   := v_message || ' is in stock ' ;
2   <--------------------------------
       v_new_locn  := 'Western ' || v_new_locn;

END;
/
```

Evaluate the PL/SQL block above and determine the data type and value of each of the following variables according to the rules of scooping.

- The value of V_WEIGHT at position 1 is :
- The value of V_NEW_LOCN at position 1 is :
- The value of V_WEIGHT at position 2 is
- The value of V_MESSAGE at position 2 is
- The value of V_NEW_LOCN at position 2 is

Exercise 3

- Write a PL/SQL program to maintain Employee Payroll data with:

Column	datatype	Integrity Constraint
EMPNO	NUMBER(4)	PRIMARY KEY
ENAME	VARCHAR2(20)	
BASIC	NUMBER(6)	NOT NULL
DA	NUMBER(8,2)	
HRA	NUMBER(7,2)	
GROSS	NUMBER(10,2)	
PF	NUMBER(7,2)	
NET	NUMBER(10,2)	

Business Logic:

1. Generate EMPNO automatically
2. Input name and basic
3. Calculate other particulars
   - DA is % on basic
   - HRA is % on basic
   - GROSS = BASIC + DA = HRA
   - PF = % ON BASIC
   - NET = GROSS – PF
4. Store the data into the table