Department of Software Engineering
Mehran University of Engineering and Technology, Jamshoro

Course: SW215 – Database System			
Instructor	Ms Shafiya Qadeer	**Practical/Lab No.**	11
Date	26-02-2021	**CLOs**	3
Signature		**Assessment Score**	2 Marks

Topic	Introduction to PL/SQL, control structure and data types
Objectives	- To become familiar with PL/SQL Environment and Program

Lab Discussion: Theoretical concepts and Procedural steps

PL/SQL

- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.

BASIS FOR COMPARISON	SQL	PL/SQL
Basic	In SQL you can execute a single query or a command at a time.	In PL/SQL you can execute a block of code at a time.
Full form	Structured Query Language	Procedural Language, extension of SQL.
Purpose	It is like a source of data that is to be displayed.	It is language that creates an application that display's the data acquired by SQL.
Writes	In SQL you can write queries and command using DDL, DML statements.	In PL/SQL you can write block of code that has procedures, functions, packages or variables, etc.
Use	Using SQL, you can retrieve, modify, add, delete, or manipulate the data in the	Using PL/SQL, you can create applications or server pages that display's the information obtained from SQL in a proper format.

PL/SQL offers the following advantages:

- Reduces network traffic This one is **great advantages** of PL/SQL. Because PL/SQL nature is **entire block** of SQL statements execute into **oracle engine** all at once so it's main benefit is **reducing** the **network traffic**.

- Procedural language support PL/SQL is a **development tools** not only for data manipulation futures but also provide the conditional checking, looping or branching operations same as like **other programming language**.
- Error handling PL/SQL is dealing with **error handling**, It's permits the smart way **handling the errors** and giving **user friendly** error messages, when the errors are encountered.
- Declare variable PL/SQL gives you control to **declare variables** and access them **within the block**. The declared variables can be used at the time of **query processing**.
- Intermediate Calculation Calculations in PL/SQL done quickly and efficiently without using Oracle engines. This **improves** the transaction performance.
- PL/SQL Block consists of three sections:

1. The Declaration section (optional).
2. The Execution section (mandatory).
3. The Exception Handling (or Error) section (optional).

Declaration Section:

- The Declaration section of a PL/SQL Block starts with the reserved keyword DECLARE.
- This section is optional.
- It is used to declare variables, constants, records and cursors, which are used to manipulate data in the execution section.
- Variables, Constants and Records, which stores data temporarily.
- Cursors are also declared in this section.

Execution Section:

- The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END.
- Note: Execution section must have one statement.

- This is a mandatory section and is the section where the program logic is written to perform any task.
- The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

Exception Section:

- The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION.
- This section is optional.
- Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully.
- If the PL/SQL Block contains exceptions that cannot be handled, the Block terminates abruptly with errors.

SYNTAX:

DECLARE (optional)
<declarations section>
BEGIN (mandatory)
<executable command(s)>
EXCEPTION (optional)
<exception handling>
END;

Important points:

- Every PL/SQL statement must end with a semicolon ;
- Comments can be used to document code.
- The PL/SQL single-line comments start with the delimiter -- (double hyphen)
- Multi-line comments are enclosed by /* and */

```
DECLARE
--variable declaration
msg varchar(20):= 'Hello F-16SW';
BEGIN
DBMS_OUTPUT.PUT_LINE(msg);
/* This statement prints the
output: Hello F-16SW  */
END;
```

Results Explain Describe Saved SQL History

```
Hello F-16SW

Statement processed.
```

SELECT Statement in PL/SQL

- Data projection/fetching means to retrieve the required data from the database table. This can be achieved by using the command 'SELECT' with 'INTO' clause.
- Syntax:

SELECT column_name [,column_name,….] INTO variable_name [, variable_name,..] FROM table_name [WHERE condition];

```
DECLARE
e_salincrease CONSTANT number(20):=200;

v_ename varchar2(55);
v_job emp.job%type;
v_empno number(5);

BEGIN
Select job, ename, sal+e_salincrease INTO v_job, v_ename, v_empno from emp where ename='SMITH';
DBMS_OUTPUT.Put_line('The new salary for '||v_ename||' serving as '||v_job||' is '||v_empno);
END;
```

Results Explain Describe Saved SQL History

```
The new salary for SMITH serving as CLERK is 1000

Statement processed.
```

- PL/SQL has three categories of control statements:

1. Conditional selection statements
2. Loop statements

3. Sequential control statements

1. Conditional selection statements

- Programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- The conditional selection statements are IF and ELSE statements and CASE.

IF - THEN statement

- The IF statement associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. If the condition is true, the statements get executed and if the condition is false then the IF statement does nothing.

IF condition THEN

statements

END IF;

If the condition is true, the statements run; otherwise, the IF statement does nothing.

```
DECLARE
a INTEGER(10):= 100 ;
BEGIN
IF (a = 100) THEN
a:=a+1 ;
dbms_output.put_line('Value is: ' || a);
END IF;
END;
```

Results Explain Describe Saved SQL History

```
Value is: 101

Statement processed.
```

```
DECLARE
a INTEGER(10):= 100 ;
BEGIN
IF (a = 100) THEN
a:=a+1 ;
dbms_output.put_line('Value is: ' || a);
END IF;
END;
```

Results Explain Describe Saved SQL History

```
Value is: 101

Statement processed.
```

IF-THEN-ELSE statement

- IF statement adds the keyword ELSE followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.

IF condition THEN

statements

ELSE

else_statements

END IF;

```
DECLARE
a INTEGER(10):= 100 ;
BEGIN
IF (a > 100) THEN
dbms_output.put_line('a is greater than 100');
ELSE
dbms_output.put_line('Value of a is: ' || a );
END IF;
END;
```

Results Explain Describe Saved SQL History

```
Value of a is: 100

Statement processed.
```

IF THEN ELSEIF Statement

- The IF THEN ELSIF statement runs the first statements for which condition is true. Remaining conditions are not evaluated. If

no condition is true, theelse_statements run, if they exist; otherwise, the IF THEN ELSIF statement does nothing.

IF condition_1 THEN

statements_1

ELSIF condition_2 THEN

statements_2

ELSIF condition_3 THEN

statements_3

...

ELSE

else_statements

END IF;

```
DECLARE
grade VARCHAR(10):= 'A+' ;
BEGIN
IF(grade = 'A+') THEN
dbms_output.put_line('Grade is A+');
ELSIF (grade = 'A') THEN
dbms_output.put_line('Grade is A');
ELSIF(grade = 'B') THEN
dbms_output.put_line('Grade is B');
ELSE
dbms_output.put_line('Fail');
END IF;
END;
```

Results Explain Describe Saved SQL History

```
Grade is A+

Statement processed.
```

CASE Statement

- Like the **IF** statement, the **CASE statement** selects one sequence of statements to execute.
- However, to select the sequence, the **CASE** statement uses a selector rather than multiple Boolean expressions.
- A selector is an expression, the value of which is used to select one of several alternatives.

Simple CASE Statement

CASE selector

WHEN selector_value_1 THEN statements_1

WHEN selector_value_2 THEN statements_2

...

WHEN selector_value_n THEN statements_n

ELSE

else_statements -- default case

END CASE;

- The selector is an expression (typically a single variable).
  Each selector_value can be either a literal or an expression.
- The simple CASE statement runs the first statements for which selector_value equals selector. Remaining conditions are not evaluated. If no selector_value equals selector, the CASE statement runs else_statements if they exist and raises the predefined exception CASE_NOT_FOUND otherwise.

```
DECLARE
color VARCHAR(10):= 'yellow' ;
BEGIN
CASE color
WHEN 'green' THEN dbms_output.put_line('Color is green');
WHEN 'red' THEN dbms_output.put_line('Color is red');
WHEN 'black' THEN dbms_output.put_line('Color is Black');
ELSE
dbms_output.put_line('none of the color is matching');
END CASE;
END;
```

Results Explain Describe Saved SQL History

```
none of the color is matching

Statement processed.
```

GOTO Statement

- A **GOTO** statement in PL/SQL programming language provides an unconditional jump from the GOTO to a labeled statement in the same subprogram.
- **NOTE** – The use of GOTO statement is not recommended in any programming language because it makes it difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a GOTO can be rewritten so that it doesn't need the GOTO.

Syntax

GOTO label;

..

..

<< label >> --Label declaration

statement;

```
DECLARE
ctr number(5):=6;
BEGIN
IF ctr<5 then
GOTO small_number;
GOTO END_PROGRAM;
ELSE
GOTO large_number;
END IF;
<<small_number>>
DBMS_OUTPUT.PUT_LINE('Small');
GOTO END_PROGRAM;
<<large_number>>
DBMS_OUTPUT.PUT_LINE('large');
<<END_PROGRAM>>
NULL;
END;
```

Results Explain Describe Saved SQL History

```
large

Statement processed.
```

Controlling Loop Iterations

- LOOP statements execute a sequence of statements multiple times.
- There are three formof LOOP statements:

1. SIMPLE LOOP
2. WHILE-LOOP

3. FOR-LOOP

1.SIMPLE LOOP

- The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP.
- With each iteration, the sequence of statements is executed and then control resumes at the top of the loop.

LOOP

Sequence of statements;

END LOOP;

- You use an EXIT statement to stop looping and prevent an infinite loop.
- An EXIT statement or an EXIT WHEN statement is required to break the loop.
- You can place one or more EXIT statements anywhere inside a loop, but not outside a loop.
- There are two forms of EXIT statements:

i. EXIT

ii. EXIT-WHEN.

i. EXIT STATEMENT

- When an EXIT statement is encountered, the loop completes immediately and control passes to the next statement.

Example Using an EXIT Statement

```
DECLARE
   x INTEGER:= 0;
BEGIN
   dbms_output.put_line('Print Even Numbers');
LOOP
      dbms_output.put_line(x);
      x := x + 2;
      IF x >= 40 THEN
         exit;
      END IF;
   END LOOP;
dbms_output.put_line(x);
END;
```

```
Print Even Numbers
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

Statement processed.
```

ii. Using the EXIT-WHEN Statement

- When the EXIT statement is encountered, the condition in the WHEN clause is evaluated. If the condition is true, the loop completes and control passes to the next statement after the loop.
- The EXIT-WHEN statement replaces simple IF statement.

```
DECLARE
   x INTEGER:= 0;
BEGIN
   dbms_output.put_line('Print Even Numbers');
LOOP
      dbms_output.put_line(x);
      x := x + 2;
         EXIT WHEN x >=40 ;
   END LOOP;
dbms_output.put_line(x);
END;
```

```
Print Even Numbers
0
2
4
6
8
10
12
14
16
18
20
22
24
26
28
30
32
34
36
38
40

Statement processed.
```

WHILE-LOOP Statement

- The WHILE-LOOP statement executes the statements in the loop body as long as a condition is true.
- It tests the condition before executing the loop body.

WHILE condition LOOP

sequence_of_statements

EXIT WHEN boolean_expression;

END LOOP;

```
DECLARE
   a INTEGER(2) := 10;
BEGIN
   WHILE (a < 20) LOOP
      dbms_output.put_line('value of a: ' || a);
      a := a + 1;
   END LOOP;
  dbms_output.put_line('Loop ended');
END;
```

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
Loop ended
```

Using the FOR-LOOP Statement

- A FOR LOOP is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

FOR counter IN initial_value .. final_value LOOP

sequence_of_statements;

END LOOP;

- Iteration occurs between the start(initial_value) and end(final_value) integer values.
- The counter is always incremented by 1.
- The loop exits when the counter reachs the value of the end integer(final_value).

```
DECLARE
   a INTEGER(2); --declare variable a
BEGIN
   FOR a IN 10 .. 20 LOOP  /* initial value is a and final value is 20 */
      dbms_output.put_line('value of a: ' || a);
  END LOOP;
END;
```

Results Explain Describe Saved SQL History

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
```

Take User Input in PL/SQL

```
DECLARE
a INTEGER(6);
b VARCHAR(16) ;


BEGIN
a:= :x ;
b:= :y ;
dbms_output.put_line('Inputed Number is: ' || a);
dbms_output.put_line('Inputed name is: ' || b);

end;
```

Results Explain Describe Saved SQL

```
Inputed Number is: 10
Inputed name is: abc

Statement processed.
```

Submit

:X 10

:Y abc

Lab Tasks

1. Write a program in PL/SQL that displays the deptno, and location of all employees in the Sales department
2. Display the sum of salary of all employees in deptno 30
3. Write a PL/SQL Block to display the following CGPA:

   for grade 4 display Excellent

   for grade 3 display Very good

for grade 2 display Well done

for grade 1 display You passed

Note: Use Case statement.

4. Print Odd Numbers upto 61. After 61, the loop should be ended.

Note: Use While or Simple Loop

5. Print Natural Numbers upto 50 by using FOR Loop or Simple Loop.