Name:=Shreyasi Gidmare

PRN:=23070521146

Practical 6

Aim: Write and execute basic PL/SQL (Procedural Language/Structured Query Language) programs - simple program, condition statements and loops.

PL/SQL Introduction

PL/SQL (Procedural Language/Structured Query Language) is a block-structured language developed by Oracle that allows developers to combine the power of SQL with procedural programming constructs. The PL/SQL language enables efficient data manipulation and control-flow logic, all within the Oracle Database.

Basics of PL/SQL

- PL/SQL stands for Procedural Language extensions to the Structured Query Language (SQL).
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- Oracle uses a PL/SQL engine to process the PL/SQL statements.
- PL/SQL includes procedural language elements like conditions and loops. It allows declaration of constants and variables, procedures and functions, types and variable of those types and triggers.

Features of PL/SQL

1. PL/SQL is basically a procedural language, which provides the functionality of decision-making, iteration, and many more features of procedural programming languages.
2. PL/SQL can execute a number of queries in one block using single command.
3. One can create a PL/SQL unit such as procedures, functions, packages, triggers, and types, which are stored in the database for reuse by applications.
4. PL/SQL provides a feature to handle the exception which occurs in PL/SQL block known as exception handling block.
5. Applications written in PL/SQL are portable to computer hardware or operating system where Oracle is operational.
6. PL/SQL Offers extensive error checking.

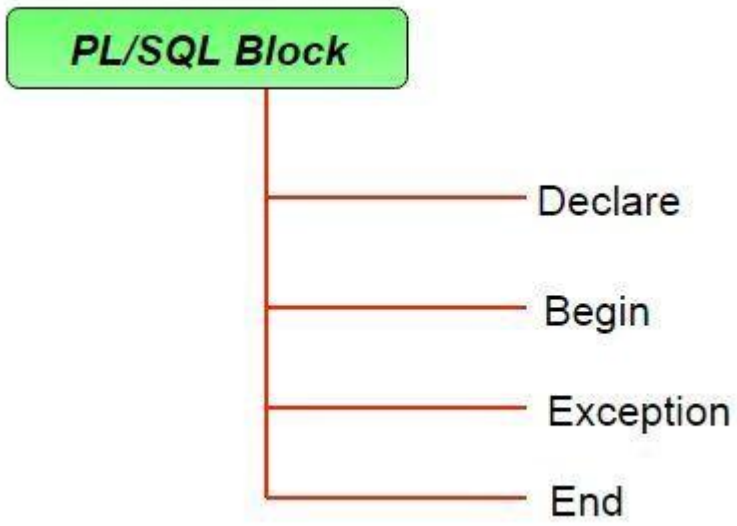
Differences Between SQL and PL/SQL

SQL	PL/SQL
SQL is a single query that is used to perform DML and DDL operations.	PL/SQL is a block of codes that used to write the entire program blocks/ procedure/ function, etc.
It is declarative, that defines what needs to be done, rather than how things need to be done.	PL/SQL is procedural that defines how the things needs to be done.
Execute as a single statement.	Execute as a whole block.

Mainly used to manipulate data.	Mainly used to create an application.
Cannot contain PL/SQL code in it.	It is an extension of SQL, so it can contain SQL inside it.

Structure of PL/SQL Block

PL/SQL extends SQL by adding constructs found in procedural languages, resulting in a structural language that is more powerful than SQL. The basic unit in PL/SQL is a block. All PL/SQL programs are made up of blocks, which can be nested within each other.



Typically, each block performs a logical action in the program. A block has the following structure:

```
DECLARE     declaration
statements;  BEGIN     executable
statements  EXCEPTIONS
exception handling statements

END;
```

- Declare section starts with DECLARE keyword in which variables, constants, records as cursors can be declared which stores data temporarily. It basically consists definition of PL/SQL identifiers. This part of the code is optional.
- Execution section starts with BEGIN and ends with END keyword.This is a mandatory section and here the program logic is written to perform any task like loops and conditional statements. It supports all DML commands, DDL commands and SQL*PLUS built-in functions as well.
- Exception section starts with EXCEPTION keyword.This section is optional which contains statements that are executed when a run-time error occurs. Any exceptions can be handled in this section.

PL/SQL Identifiers

There are several PL/SQL identifiers such as variables, constants, procedures, cursors, triggers etc.

1. **Variables:** Like several other programming languages, variables in PL/SQL must be declared prior to its use. They should have a valid name and data type as well. Syntax for declaration of variables:

```
variable_name datatype [NOT NULL := value ];
```

1. **Example to show how to declare variables in PL/SQL :**

```
SQL> SET SERVEROUTPUT ON;
  SQL> DECLARE     var1
INTEGER;     var2 REAL;
var3 varchar2(20) ;
BEGIN     null; END;
/
```

1. **Output:**

```
PL/SQL procedure successfully completed.
```

1. ***Explanation:***

- **SET SERVEROUTPUT ON:** It is used to display the buffer used by the dbms_output.
- **var1 INTEGER :** It is the declaration of variable, named *var1* which is of integer type. There are many other data types that can be used like float, int, real, smallint, long etc. It also supports variables used in SQL as well like NUMBER(prec, scale), varchar, varchar2 etc.

- **PL/SQL procedure successfully completed.:** It is displayed when the code is compiled and executed successfully.
- **Slash (/) after END;:** The slash (/) tells the SQL*Plus to execute the block.
- **Assignment operator (:=) :** It is used to assign a value to a variable.

2. **Displaying Output:** The outputs are displayed by using DBMS_OUTPUT which is a built-in package that enables the user to display output, debugging information, and send messages from PL/SQL blocks, subprograms, packages, and triggers. Let us see an example to see how to display a message using PL/SQL :

```
SQL> SET SERVEROUTPUT ON; SQL> DECLARE
var varchar2(40) := 'I love DBMS' ;
BEGIN      dbms_output.put_line(var);
  END;


  /
```

1. **Output:**

```
I love DBMS
PL/SQL procedure successfully completed.
```

1. ***Explanation:***
   - ***dbms_output.put_line* :** This command is used to direct the PL/SQL output to a screen.

2. **Using Comments:** Like in many other programming languages, in PL/SQL also, comments can be put within the code which has no effect in the code. There are two syntaxes to create comments in PL/SQL :

- **Single Line Comment:** To create a single line comment , the symbol – – is used.
- **Multi Line Comment:** To create comments that span over several lines, the symbol /* and */ is used.

3. **Taking input from user:** Just like in other programming languages, in PL/SQL also, we can take input from the user and store it in a variable. Let us see an example to show how to take input from users in PL/SQL:

A. Using Parameters in a Procedure

Instead of relying on user input inside PL/SQL, pass parameters when calling a procedure.

Example:

```
CREATE OR REPLACE PROCEDURE add_numbers(a IN NUMBER, b IN NUMBER)
AS     c NUMBER; BEGIN     c := a + b;

    DBMS_OUTPUT.PUT_LINE('Sum of ' || a || ' and ' || b || ' is = '
|| c);
END;
/
```

```
SQL> CREATE OR REPLACE PROCEDURE add_numbers(a IN NUMBER, b IN NUMBER) AS
  c NUMBER; BEGIN     c := a + b;
  2  DBMS_OUTPUT.PUT_LINE('Sum of ' || a || ' and ' || b || ' is = '
  3  || c);
  4  END;
  5  /

Procedure created.
```

How to Call It with Input?

```
BEGIN
    add_numbers(10, 20); END;
/
```

```
SQL> BEGIN
  2      add_numbers(10, 20); END;
  3  /

PL/SQL procedure successfully completed.
```

Why?

- ☑ **Works in PL/SQL procedures/functions**
- ☑ **No need for & variables**
- ☑ **Can be used in applications**

2. Using Bind Variables in SQL Developer

If you are using SQL Developer, Toad, or another IDE, you can use bind variables instead of substitution variables (&a).

Example:

```
VARIABLE a NUMBER;
VARIABLE b NUMBER;
BEGIN
    :a := 15; -- Assign user input
    :b := 25;
```

```
        DECLARE         c NUMBER;     BEGIN         c := :a + :b;
DBMS_OUTPUT.PUT_LINE('Sum of ' || :a || ' and ' || :b || ' is = '
|| c);     END;
END;
/
```

```
SQL> VARIABLE a NUMBER;
SQL> VARIABLE b NUMBER;
SQL> BEGIN
  2      :a := 15; -- Assign user input
  3      :b := 25;
  4       DECLARE         c NUMBER;     BEGIN         c := :a + :b;
DBMS_OUTPUT.PUT_LINE('Sum of ' || :a || ' and ' || :b || ' is = ' || c);
 END;
  5  END;
  6  /

PL/SQL procedure successfully completed.
```

Why?

- ☑ **Works in SQL Developer / Toad**
- ☑ **Avoids old/new message**
- ☑ **Can be assigned values dynamically**

3. Using an Anonymous Block with ACCEPT

If you still want to use SQL*Plus but avoid the "old/new" messages, use ACCEPT with bind variables.

Example:

```
ACCEPT a NUMBER PROMPT 'Enter value for a: ';
```

```
ACCEPT b NUMBER PROMPT 'Enter value for b: ';

DECLARE     a NUMBER := &a;     b NUMBER :=

&b;

    c NUMBER;

BEGIN     c := a + b;     DBMS_OUTPUT.PUT_LINE('Sum of ' || a || '

and ' || b || ' is = '

|| c);

END;

/
```

```
SQL> ACCEPT a NUMBER PROMPT 'Enter value for a: ';
Enter value for a: ACCEPT b NUMBER PROMPT 'Enter value for b: '; DECLARE
 a NUMBER := &a;     b NUMBER := &b;
SP2-0425: "ACCEPT b NUMBER PROMPT 'Enter value for b: '; DECLARE          a
 NUMBER := &a;     b NUMBER := &b; " is not a valid NUMBER
Enter value for a: c NUMBER;
SP2-0425: "c NUMBER; " is not a valid NUMBER
Enter value for a: BEGIN     c := a + b;     DBMS_OUTPUT.PUT_LINE('Sum of '
|| a || ' and ' || b || ' is = '
SP2-0425: "BEGIN          c := a + b;     DBMS_OUTPUT.PUT_LINE('Sum of ' ||
 a || ' and ' || b || ' is = ' " is not a valid NUMBER
Enter value for a: || c);
SP2-0425: "|| c); " is not a valid NUMBER
Enter value for a: END;
SP2-0425: "END; " is not a valid NUMBER
Enter value for a: /
SP2-0425: "/ " is not a valid NUMBER
Enter value for a: 5
SQL> /

PL/SQL procedure successfully completed.
```

Why?

- ☑ **Works in SQL*Plus**
- ☑ **No "old/new" message**
- ☑ **Simpler for command-line input**

4. Using a Table for Input (Real-World Use Case)

In real applications, user input is stored in a table and read in PL/SQL.

Example:

```
-- Create table to store input values
CREATE TABLE input_values (a NUMBER, b NUMBER);
-- Insert user input
INSERT INTO input_values VALUES (30, 40);
COMMIT;
-- Read values and process
DECLARE
    a NUMBER;
b NUMBER;     c
NUMBER; BEGIN
    -- Fetch values from table
    SELECT a, b INTO a, b FROM input_values WHERE ROWNUM = 1;
c := a + b;     DBMS_OUTPUT.PUT_LINE('Sum of ' || a || ' and ' || b
|| ' is = '
|| c);
END;
/
```

```
SQL> -- Read values and process
SQL> DECLARE
  2  a NUMBER;     b NUMBER;     c NUMBER; BEGIN
  3      -- Fetch values from table
  4      SELECT a, b INTO a, b FROM input_values WHERE ROWNUM = 1;          c := a + b;     DBMS_OUTPUT.PUT_LINE('Sum of
' || a || ' and ' || b || ' is = '
  5  || c);
  6  END;
  7  /

PL/SQL procedure successfully completed.
```

Why?

- ☑ **Works inside PL/SQL**
- ☑ **Allows persistent storage**
- ☑ **Used in real-world applications**

1. (***) *Let us see an example on PL/SQL to demonstrate all above concepts in one single block of code.*

```
SET SERVEROUTPUT ON;

 DECLARE
 variable a
 a integer := &a ;

 BEGIN      c := a + b ;
dbms_output.put_line('Sum of '||a||' and '||b||' is =
'||c);

  END;

  /

Enter value for a: 2
Enter value for b: 3

Sum of 2 and 3 is = 5
 PL/SQL procedure successfully completed
```

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2  a integer := &a ;
  3  b integer := &b ;
  4  c integer ;
  5  BEGIN
  6  c := a + b ;
  7  dbms_output.put_line('Sum of '||a||' and '||b||' is = '||c);
  8  END;
  9  /
old   2: a integer := &a ;
new   2: a integer :=         15 ;
old   3: b integer := &b ;
new   3: b integer :=         15 ;
Sum of 15 and 15 is = 30

PL/SQL procedure successfully completed.
```

.