Part A

Creation of simple PL/SQL program which includes declaration section, executable section and exception – Handling section.

Aim:

- To understand PL/SQL Block
- To understand PL/SQL identifiers
- To understand PL/SQL variable and their initialization
- To understand to take input from user in PL/SQL
- To understand displaying output
- To understand execution of PL/SQL program
- To understand single-line or multi-line comments in PL/SQL
- To understand raising user-defined exception in PL/SQL

Prerequisite: Oracle.

Outcome: Students will be able to write PL/SQL block.

Theory:

PL/SQL stands for "Procedural Language extensions to the Structured Query Language. PL/SQL adds many procedural constructs to SQL language to overcome some limitations of SQL. SQL is a popular language for both querying and updating data in the relational database management systems (RDBMS).

Disadvantages of SQL:

- SQL doesn't provide the programmers with a technique of condition checking, looping and branching.
- SQL statements are passed to Oracle engine one at a time which increases traffic and decreases speed.
- SQL has no facility of error checking during manipulation of data.

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.

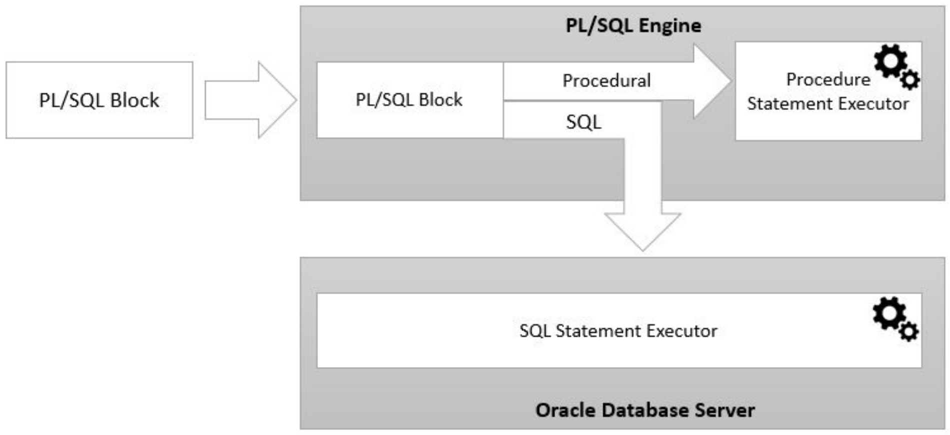
PL/SQL is a highly structured and readable language. Its constructs express the intent of the code clearly. Also, PL/SQL is a straightforward language to learn.

PL/SQL is an embedded language. PL/SQL only can execute in an Oracle Database. It was not designed to use as a standalone language like Java, C#, and C++. In other words, you cannot develop a PL/SQL program that runs on a system that does not have an Oracle Database.

PL/SQL is a high-performance and highly integrated database language. Besides PL/SQL, you can use other programming languages such as Java, C#, and C++. However, it is easier to write efficient code in PL/SQL than other programming languages when it comes to interacting with the Oracle Database.

PL/SQL architecture

The following picture illustrates the PL/SQL architecture:



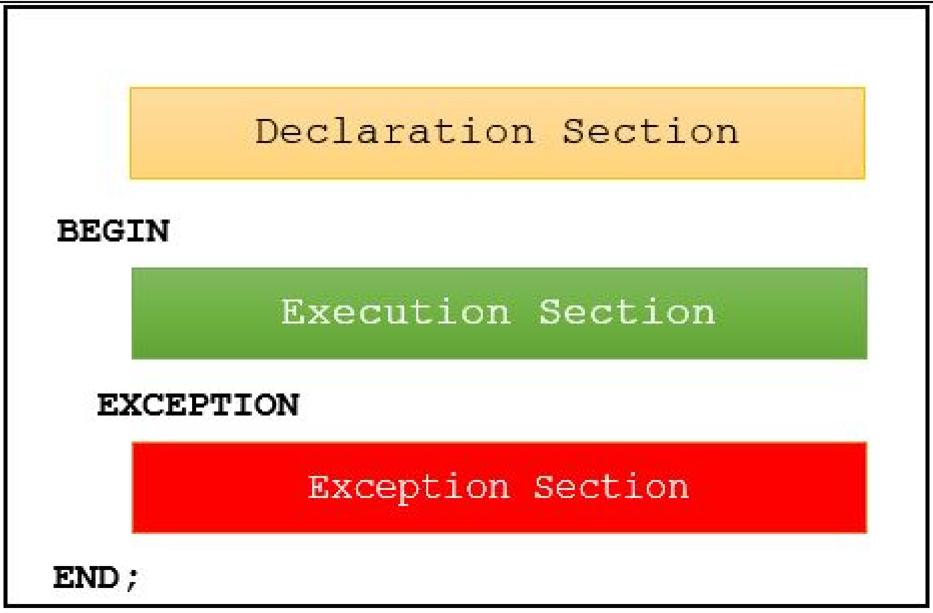
PL/SQL engine is in charge of compiling PL/SQL code into byte-code and executes the executable code. The PL/SQL engine can only be installed in an Oracle Database server or an application development tool such as Oracle Forms. Once you submit a PL/SQL block to the Oracle Database server, the PL/SQL engine collaborates with the SQL engine to compile and execute the code. PL/SQL engine runs the procedural elements while the SQL engine processes the SQL statements.
Now you should have a basic understanding of PL/SQL programming language and its architecture. Let's create the first working PL/SQL anonymous block.

PL/SQL anonymous block overview

PL/SQL is a block-structured language whose code is organized into blocks. A PL/SQL block consists of three sections: declaration, executable, and exception-handling sections. In a block, the executable section is mandatory while the declaration and exception-handling sections are optional.

A PL/SQL block has a name. Functions or Procedures is an example of a named block. A named block is stored into the Oracle Database server and can be reused later.

A block without a name is an anonymous block. An anonymous block is not saved in the Oracle Database server, so it is just for one-time use. However, PL/SQL anonymous blocks can be useful for testing purposes. The following picture illustrates the structure of a PL/SQL block:



1) Declaration section

A PL/SQL block has a declaration section where you declare variables, allocate memory for cursors, and define data types.

2) Executable section

A PL/SQL block has an executable section. An executable section starts with the keyword BEGIN and ends with the keyword END. The executable section must have a least one executable statement, even if it is the NULL statement which does nothing.

3) Exception-handling section

A PL/SQL block has an exception-handling section that starts with the keyword EXCEPTION. The exception-handling section is where you catch and handle exceptions raised by the code in the execution section. Note a block itself is an executable statement, therefore you can nest a block within other blocks.

Syntax to write an exception

```
WHEN exception THEN
  statement;
```

Basic structure of PL/SQL Program

```
DECLARE
declarations section;

BEGIN
executable command(s);

EXCEPTION
WHEN exception1 THEN
statement1;
WHEN exception2 THEN
statement2;
[WHEN others THEN]
/* default exception handling code */

END;
```

PL/SQL anonymous block example

The following example shows a simple PL/SQL anonymous block with one executable section.

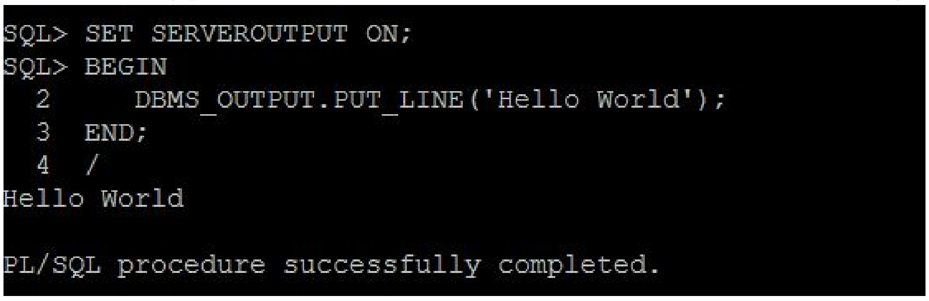
```
1 BEGIN
2    DBMS_OUTPUT.put_line ('Hello World!');
3 END;
```

The executable section calls the DMBS_OUTPUT.PUT_LINE procedure to display the "Hello World" message on the screen.

Execute a PL/SQL anonymous block using SQL*Plus

Once you have the code of an anonymous block, you can execute it using SQL*Plus, which is a command-line interface for executing SQL statement and PL/SQL blocks provided by Oracle Database.

The following picture illustrates how to execute a PL/SQL block using SQL*Plus:



```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
  2     DBMS_OUTPUT.PUT_LINE('Hello World');
  3  END;
  4  /
Hello World

PL/SQL procedure successfully completed.
```

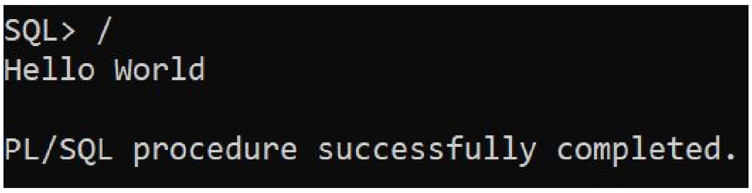
First, connect to the Oracle Database server using a username and password.

Second, turn on the server output using the SET SERVEROUTPUT ON command so that the DBMS_OUTPUT.PUT_LINE procedure will display text on the screen.

Third, type the code of the block and enter a forward slash (/) to instruct SQL*Plus to execute the block. Once you type the forward-slash (/), SQL*Plus will execute the block and display the Hello World message on the screen as shown in the illustrations.

Note that you must execute SET SERVEROUTPUT ON command in every session that you connect to the Oracle Database in order to show the message using the DBMS_OUTPUT.PUT_LINE procedure.

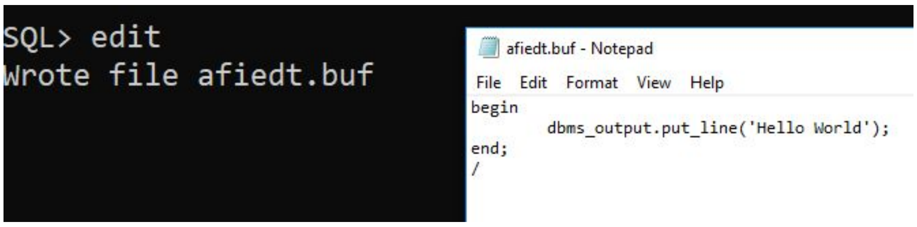
To execute the block that you have entered again, you use / command instead of typing everything from the scratch:



```
SQL> /
Hello World

PL/SQL procedure successfully completed.
```

If you want to edit the code block, use the edit command. SQL*Plus will write the code block to a file and open it in a text editor as shown in the following picture:



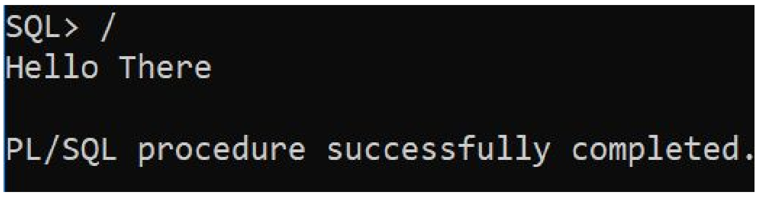
```
SQL> edit
Wrote file afiedt.buf
```

```
begin
        dbms_output.put_line('Hello World');
end;
/
```

You can change the contents of the file like the following:

1 begin
2 dbms_output.put_line('Hello There');
3 end;
4 /

And save and close the file. The contents of the file will be written to the buffer and recompiled. After that, you can execute the code block again, it will use the new code:



```
SQL> /
Hello There

PL/SQL procedure successfully completed.
```

The next anonymous block example adds an exception-handling section which catches ZERO_DIVIDE exception raised in the executable section and displays an error message.

```
DECLARE
    v_result NUMBER;
BEGIN
  v_result := 1 / 0;
  EXCEPTION
    WHEN ZERO_DIVIDE THEN
      DBMS_OUTPUT.PUT_LINE( 'divisor cannot be zero' );
END;
/
```

Output:
 divisor cannot be zero

Now, you should know how to create PL/SQL anonymous blocks and execute them using SQL*Plus and Oracle SQL Developer tools.

The PL/SQL Identifiers

PL/SQL identifiers are name given to constants, variables, exceptions, procedures, cursors, and reserved words. The identifiers consist of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.

By default, identifiers are not case-sensitive. So you can use integer or INTEGER to represent a numeric value. You cannot use a reserved keyword as an identifier.

The PL/SQL Comments

Program comments are explanatory statements that can be included in the PL/SQL code that you write and helps anyone reading its source code. All programming languages allow some form of comments.

The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler. The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

The PL/SQL 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 ];
```

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;
/
```

Output:

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

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.

INITIALISING VARIABLES:

The variables can also be initialised just like in other programming languages. Let us see an example for the same:

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
    var1 INTEGER := 2 ;
    var3 varchar2(20) := 'I Love Programming' ;

  BEGIN
    null;

  END;
 /
```

Output:

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

Explanation:

- **Assignment operator (:=)**: It is used to assign a value to a variable.

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 programming ;

   BEGIN
     dbms_output.put_line(var);

   END;
   /
```

Output:

```
I love programming

PL/SQL procedure successfully completed.
```

Explanation:

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

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:

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
      -- taking input for variable a
      a number := &a;
      -- taking input for variable b
      b varchar2(30) := &b;
  BEGIN
      null;
  END;
  /
```

Output:

```
Enter value for a: 24
old   2: a number := &a;
new   2: a number := 24;
Enter value for b: 'Programming'
old   3: b varchar2(30) := &b;
new   3: b varchar2(30) := 'Programming';

PL/SQL procedure successfully completed.
```

Raising User-defined Exceptions:

Users can create their own exceptions according to the need and to raise these exceptions explicitly *raise* command is used.

```
DECLARE
x int:=&x; /*taking value at run time*/
y int:=&y;
div_r float;
exp1 EXCEPTION;
exp2 EXCEPTION;
BEGIN
IF y=0 then
        raise exp1;
ELSEIF y > x then
        raise exp2;
ELSE
        div_r:= x / y;
        dbms_output.put_line('the result is '||div_r);
END IF;
EXCEPTION
WHEN exp1 THEN
        dbms_output.put_line('Error');
        dbms_output.put_line('division by zero not allowed');
```

```
WHEN exp2 THEN
        dbms_output.put_line('Error');
        dbms_output.put_line('y is greater than x please check the input');
END;
```

```
Input 1: x = 20
         y = 10

Output: the result is 2
```

```
Input 2: x = 20
         y = 0

Output:
Error
division by zero not allowed
```

```
Input 3: x=20
         y = 30

Output:<.em>
Error
y is greater than x please check the input
```

Practice Exercise

1	Declare a integer variable and define string value for variable in execution section, it will generate the Value_Error exception, catch this exception in the exception section, and prompt a appropriate error message to the user**.**
2	Read two integers from user, add them and store result in total. Also declare a sum1 variable in declaration section of PL/SQL block and initialize it with 30. Now, if the total is less than sum then raise twoSmallNumbers exception. If the total is greater than sum, then raise twoLargeNumbers exception. If the total is equal to sum, then display “perfect numbers” message. Print appropriate error message for each raised exception.

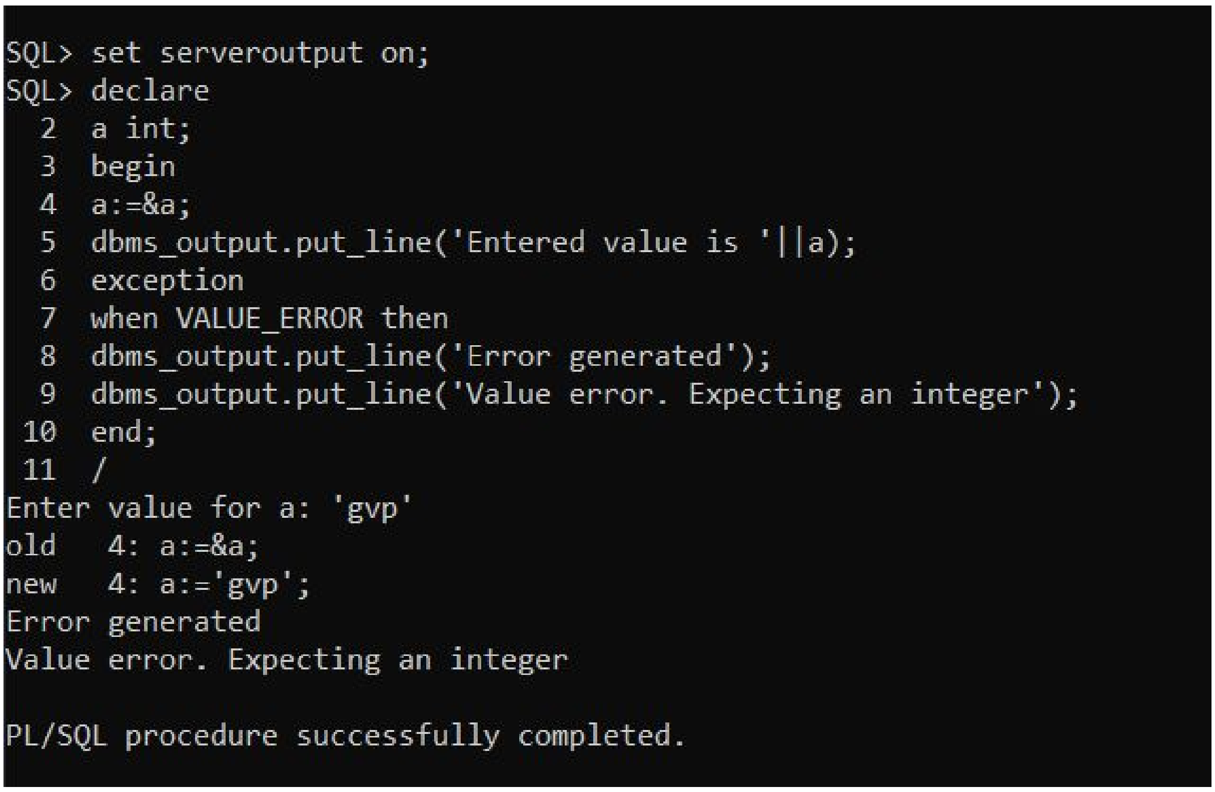
Instructions:

1. Write and execute the query in Oracle SQL server/ SQL* Plus.
2. Paste the snapshot of the output in input & output section.

Part B

Code and Output:

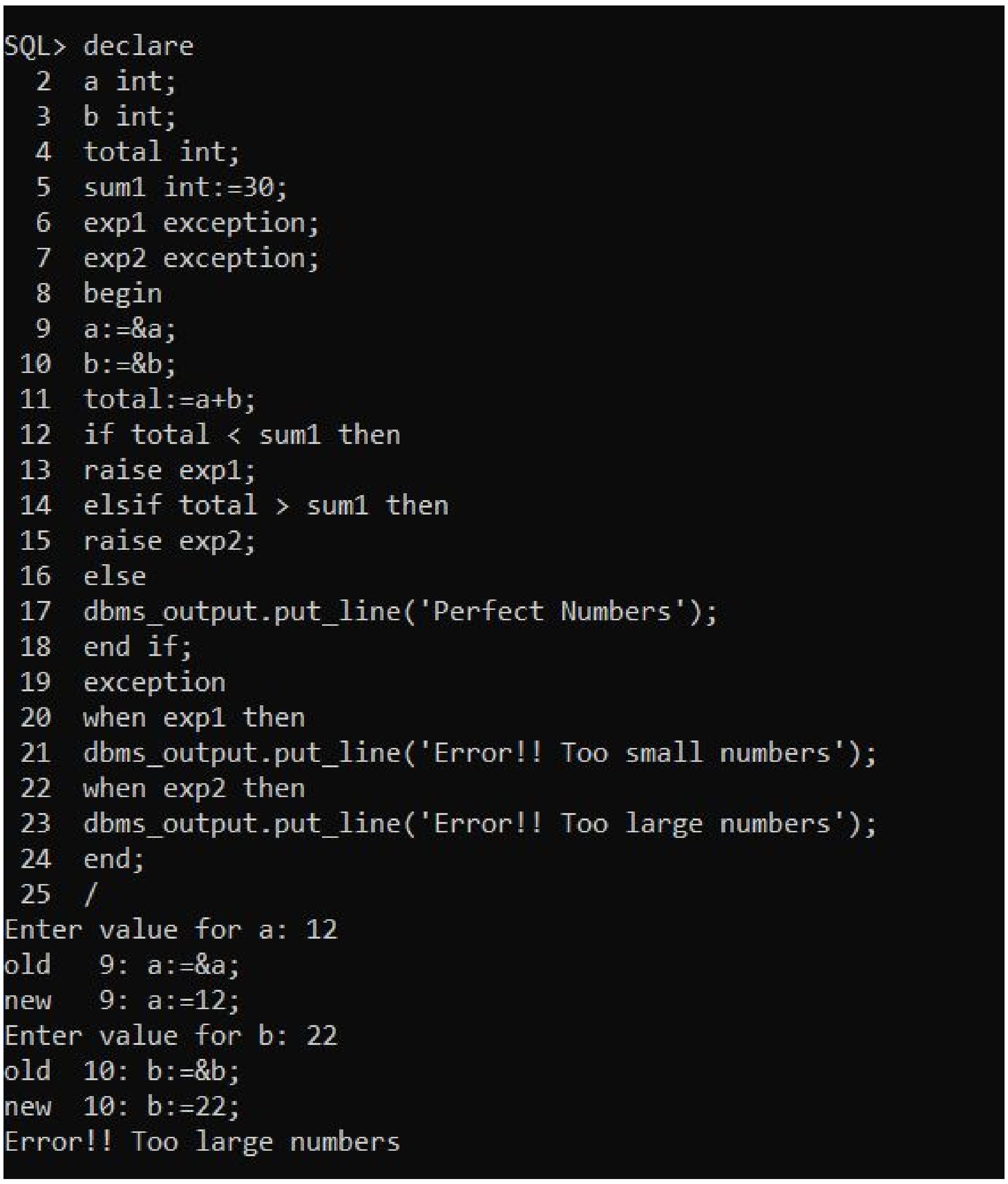
1.



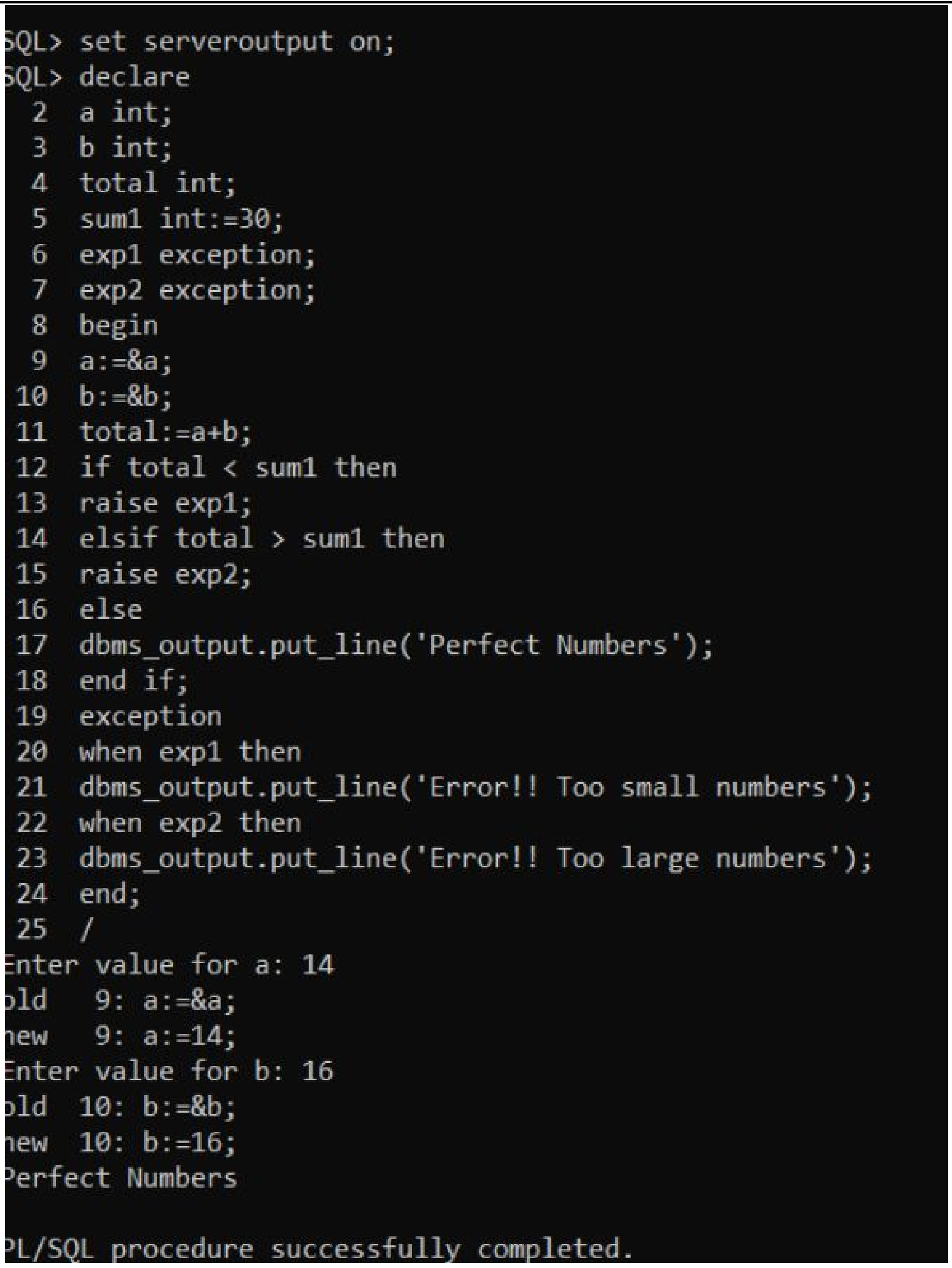
```
SQL> set serveroutput on;
SQL> declare
  2  a int;
  3  begin
  4  a:=&a;
  5  dbms_output.put_line('Entered value is '||a);
  6  exception
  7  when VALUE_ERROR then
  8  dbms_output.put_line('Error generated');
  9  dbms_output.put_line('Value error. Expecting an integer');
 10  end;
 11  /
Enter value for a: 'gvp'
old   4: a:=&a;
new   4: a:='gvp';
Error generated
Value error. Expecting an integer

PL/SQL procedure successfully completed.
```

2.

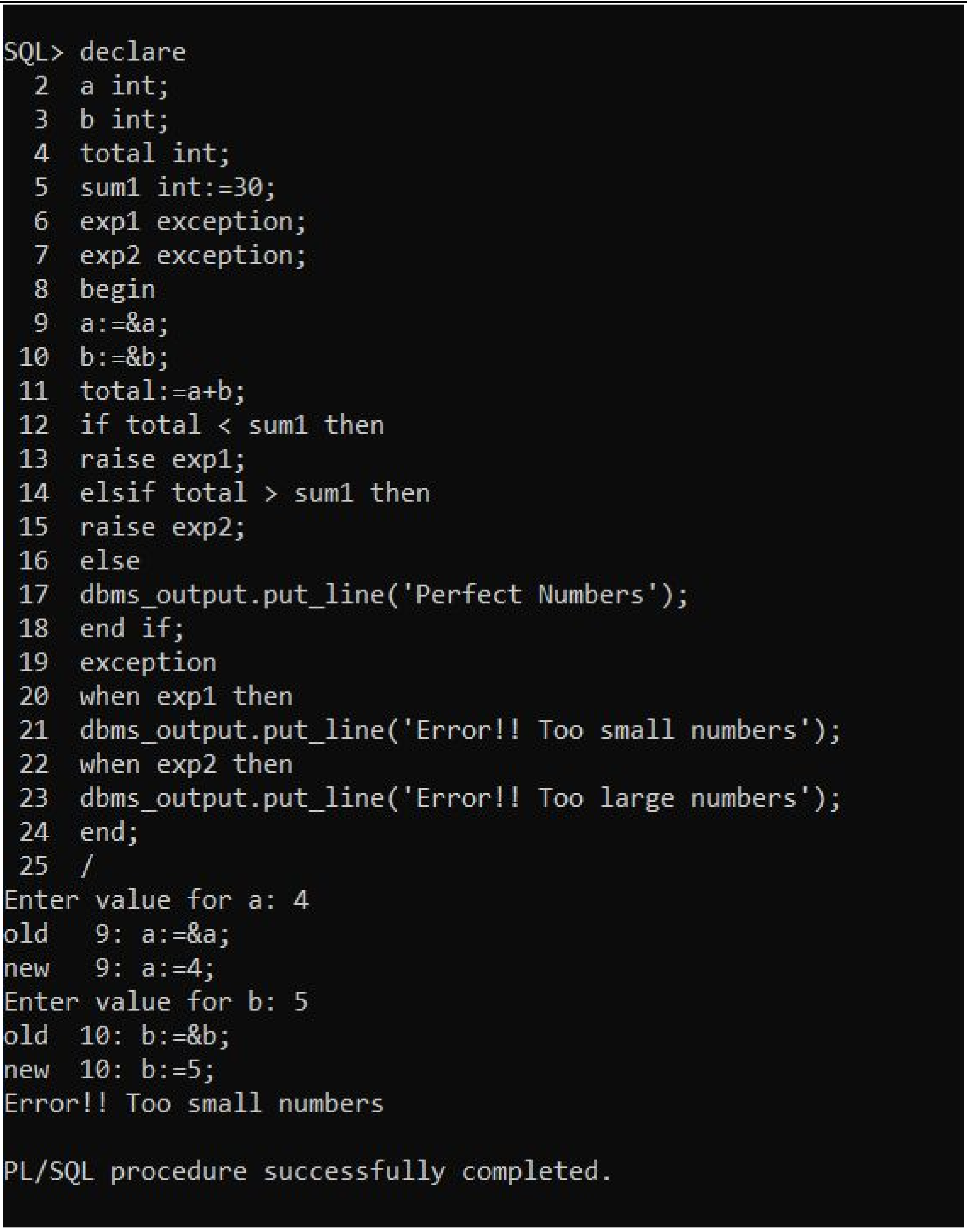


```
SQL> declare
  2  a int;
  3  b int;
  4  total int;
  5  sum1 int:=30;
  6  exp1 exception;
  7  exp2 exception;
  8  begin
  9  a:=&a;
 10  b:=&b;
 11  total:=a+b;
 12  if total < sum1 then
 13  raise exp1;
 14  elsif total > sum1 then
 15  raise exp2;
 16  else
 17  dbms_output.put_line('Perfect Numbers');
 18  end if;
 19  exception
 20  when exp1 then
 21  dbms_output.put_line('Error!! Too small numbers');
 22  when exp2 then
 23  dbms_output.put_line('Error!! Too large numbers');
 24  end;
 25  /
Enter value for a: 12
old   9: a:=&a;
new   9: a:=12;
Enter value for b: 22
old  10: b:=&b;
new  10: b:=22;
Error!! Too large numbers
```



```
SQL> set serveroutput on;
SQL> declare
  2  a int;
  3  b int;
  4  total int;
  5  sum1 int:=30;
  6  exp1 exception;
  7  exp2 exception;
  8  begin
  9  a:=&a;
 10  b:=&b;
 11  total:=a+b;
 12  if total < sum1 then
 13  raise exp1;
 14  elsif total > sum1 then
 15  raise exp2;
 16  else
 17  dbms_output.put_line('Perfect Numbers');
 18  end if;
 19  exception
 20  when exp1 then
 21  dbms_output.put_line('Error!! Too small numbers');
 22  when exp2 then
 23  dbms_output.put_line('Error!! Too large numbers');
 24  end;
 25  /
Enter value for a: 14
old   9: a:=&a;
new   9: a:=14;
Enter value for b: 16
old  10: b:=&b;
new  10: b:=16;
Perfect Numbers

PL/SQL procedure successfully completed.
```



```
SQL> declare
  2  a int;
  3  b int;
  4  total int;
  5  sum1 int:=30;
  6  exp1 exception;
  7  exp2 exception;
  8  begin
  9  a:=&a;
 10  b:=&b;
 11  total:=a+b;
 12  if total < sum1 then
 13  raise exp1;
 14  elsif total > sum1 then
 15  raise exp2;
 16  else
 17  dbms_output.put_line('Perfect Numbers');
 18  end if;
 19  exception
 20  when exp1 then
 21  dbms_output.put_line('Error!! Too small numbers');
 22  when exp2 then
 23  dbms_output.put_line('Error!! Too large numbers');
 24  end;
 25  /
Enter value for a: 4
old   9: a:=&a;
new   9: a:=4;
Enter value for b: 5
old  10: b:=&b;
new  10: b:=5;
Error!! Too small numbers

PL/SQL procedure successfully completed.
```

Observation & Learning:
Understood different sections of pl/sql block , how to declare identifiers ,variables and their initialization, how to take take user input to variables using & operator,how to display output in pl/sql, how to use comments in pl/sql program and raise a user-defined exception .

Conclusion:

Learned and practiced how to write simple pl/sql blocks with declarative, executable and exception sections.

Questions:

1. What happen if **Executable section** is not written into PL/SQL block?
2. Which sections in PL/SQL block are optional?

Answers:

1.An error will occur if executable section of pl/sql block is empty. There should be atleast 1 statement inside pl/sql executable section.

2.Declarative section and exception section of pl/sql block are optional.