

Practical-9

- **PL/SQL Block Syntax and DML Operation through PL/SQL Block.**

- In PL/SQL, the code is not executed in single line format, but it is always executed by grouping the code into a single element called Blocks.
- Blocks contain both PL/SQL as well as SQL instruction. All these instructions will be executed as a whole rather than executing a single instruction at a time.

❖ BLOCK STRUCTURE:

- PL/SQL blocks have a pre-defined structure in which the code is to be grouped.
- The different sections of PL/SQL block are as follows:
 1. Declaration section
 2. Execution section
 3. Termination section

1) Declaration section:

- This is the first section of the PL/SQL blocks.
- This section is an optional part.
- This is the section in which the declaration of variables, cursors, exceptions, subprograms, pragma instructions and collections that are needed in the block will be declared.
- This section starts with the keyword 'DECLARE' for triggers and anonymous block. For other subprograms, this keyword will not be present. Instead, the part after the subprogram name definition marks the declaration section.

2) Execution section:

- Execution part is the main and mandatory part which actually executes the code that is written inside it. Since the PL/SQL expects the executable statements from this block this cannot be an empty block, i.e., it should have at least one valid executable code line in it.
- This can contain one or many blocks inside it as a nested block.
- This section starts with the keyword 'BEGIN'.

3) Termination section:

- This is the last section of the PL/SQL blocks.
- This section is a mandatory part.
- This section should always be followed by the keyword 'END'.
- The Keyword 'END' marks the end of PL/SQL block.

❖ PL/SQL Block syntax:

declare
(Declaration of variable must be done here)

begin
(Logic of the program must be written here)

end;
/
(Termination of program takes place here)

❖ DML operation through PL/SQL block:

- DML stands for Data Manipulation Language.
- These statements are mainly used to perform the manipulation activity.
- It includes the below given operations:
 - Data insertion
 - Data update
 - Data deletion
 - Data selection
- In PL/SQL, we can do the data manipulation only by using the SQL commands.

1) Data insertion:

In PL/SQL, we can insert the data into any table using the SQL command INSERT INTO. This command will take the table name, table column and column values as the input and insert the value in the base table.

The INSERT command can also take the values directly from another table using 'SELECT' statement rather than giving the values for each column. Through 'SELECT' statement, we can insert as many rows as the base table contains.

- Syntax:

(i)
BEGIN
INSERT INTO <table_name>(<column1 >,<column2>,...<column_n>)
VALUES(<value1><value2>,...:<value_n>);
END;
/

- The above syntax shows the INSERT INTO command. The table name and values are mandatory fields, whereas column names are not mandatory if the insert statements have values for all the column of the table.
- The keyword 'VALUES' is mandatory if the values are given separately as shown above.

OR

```
(ii)
BEGIN
INSERT INTO <table_name>(<column1>,<column2>,...,<column_n>)
SELECT <column1>,<column2>,... <column_n> FROM <table_name2>;
END;
/
```

- The above syntax shows the INSERT INTO command that takes the values directly from the <table_name2> using the SELECT command.
- The keyword 'VALUES' should not be present in this case as the values are not given separately.

- **Block:**

```
begin
insert into PERSON values('&p_id','&p_name','&age','&salary');
end;
/
```

```
SQL> set serveroutput on
SQL> begin
  2 insert into PERSON values('&p_id','&p_name','&age','&salary');
  3 end;
  4 /
Enter value for p_id: 6
Enter value for p_name: HARSHIL
Enter value for age: 20
Enter value for salary: 89000
old 2: insert into PERSON values('&p_id','&p_name','&age','&salary');
new 2: insert into PERSON values('6','HARSHIL','20','89000');

PL/SQL procedure successfully completed.
```

2) Data update:

Data update simply means an update of the value of any column in the table. This can be done using 'UPDATE' statement. This statement takes the table name, column name and value as the input and updates the data.

- Syntax:

```
BEGIN
UPDATE <table_name>
SET <column1>=<VALUE1>,<column2>=<value2>,<column_n>=<value_n>
WHERE <condition that uniquely identifies the record that needs to be update>;
END;
/
```

- The above syntax shows the UPDATE. The keyword 'SET' instruct that PL/SQL engine to update the value of the column with the value given.
- 'WHERE' clause is optional. If this clause is not given, then the value of the mentioned column in the entire table will be updated.

- Block:

```
set serveroutput on begin
update PERSON
set p_name = 'DEV' where p_id = '6';
end;
/
```

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2  UPDATE PERSON
  3  SET p_name = 'DEV' WHERE p_id = '6';
  4  end;
  5  /

PL/SQL procedure successfully completed.

SQL> select * from person;

   P_ID P_NAME          AGE     SALARY
-----
    6 DEV              20      89000
    1 HUNAID           20     100000
    2 HITENDRA         19     150000
    3 HARSHAD          18     175000
    4 AESHWARY         19     155000
    5 DHURVAL          20     120000

6 rows selected.
```

3) Data deletion:

Data deletion means to delete one full record from the database table. The 'DELETE' command is used for this purpose.

- **Syntax:**

```
BEGIN
DELETE FROM <table_name> WHERE <condition that uniquely identifies the record that needs
to be update>;
END;
/
```

- The above syntax shows the DELETE command. The keyword 'FROM' is optional and with or without 'FROM' clause the command behaves in the same way.
- 'WHERE' clause is optional. If this clause is not given, then the entire table will be deleted.

- **Block:**

```
set serveroutput on begin
delete from PERSON where p_id = '10005';
end;
/
```

```
SQL> SET SERVEROUTPUT ON
SQL> BEGIN
  2 delete from person where p_id='6';
  3 end;
  4 /

PL/SQL procedure successfully completed.

SQL> select * from person;
```

P_ID	P_NAME	AGE	SALARY
1	HUNAID	20	100000
2	HITENDRA	19	150000
3	HARSHAD	18	175000
4	AESHWARY	19	155000
5	DHRUVAL	20	120000

4) Data selection:

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. The 'SELECT' command will fetch the values from the database, and 'INTO' clause will assign these values to the local variable of the PL/SQL block.

- Below are the points that need to be considered in 'SELECT' statement.
 - 'SELECT' statement should return only one record while using 'INTO' clause as one variable can hold only one value. If the 'SELECT' statement returns more than one value than 'TOO_MANY_ROWS' exception will be raised.
 - 'SELECT' statement will assign the value to the variable in the 'INTO' clause, so it needs to get at least one record from the table to populate the value. If it didn't get any record, then the exception 'NO_DATA_FOUND' is raised.
 - The number of columns and their datatype in 'SELECT' clause should match with the number of variables and their datatypes in the 'INTO' clause.
 - The values are fetched and populated in the same order as mentioned in the statement.
 - 'WHERE' clause is optional that allows to having more restriction on the records that are going to be fetched.
 - 'SELECT' statement can be used in the 'WHERE' condition of other DML statements to define the values of the conditions.
 - The 'SELECT' statement when using 'INSERT', 'UPDATE', 'DELETE' statements should not have 'INTO' clause as it will not populate any variable in these cases.
- **Syntax:**

BEGIN

SELECT <column 1>, . . . , <column_n> INTO <variable 1>, . . . , <variable_n> FROM <table_name>

WHERE <condition to fetch the required records>;

END;

/

- The above syntax shows the SELECT-INTO command. The keyword 'FROM' is mandatory that identifies the table name from which the data needs to be fetched.
- 'WHERE' clause is optional. If this clause is not given, then the data from the entire table will be fetched.

Practical-10

- Control Structures in PL/SQL

1) Write a PL/SQL block to check whether the given number is even or odd.

```
set serveroutput on
declare
n number:=&n;
begin
if mod(n,2)=0
then
dbms_output.put_line('Number is even.');
```

```
else
dbms_output.put_line('Number is odd.');
```

```
end if;
end;
/
```

```
SQL> set serveroutput on
SQL> declare
  2  n number:=&n;
  3  begin
  4  if mod(n,2)=0
  5  then
  6  dbms_output.put_line('Number is even.');
```

```
  7  else
  8  dbms_output.put_line('Number is odd.');
```

```
  9  end if;
 10 end;
 11 /
```

```
Enter value for n: 6
```

```
old  2: n number:=&n;
```

```
new  2: n number:=6;
```

```
Number is even.
```

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

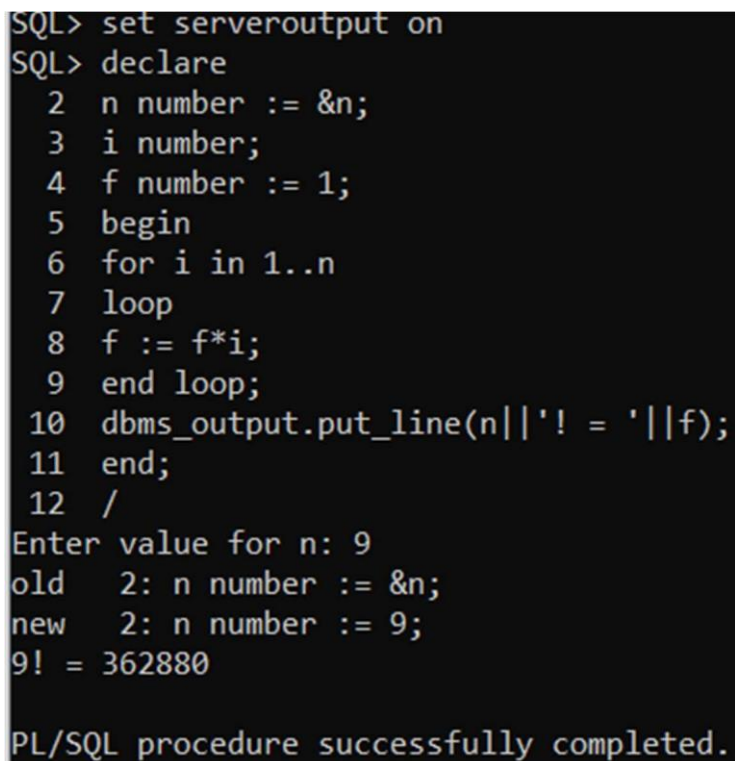
2) Write a PL/SQL block to find the sum of odd numbers between 1 to 100.

```
set serveroutput on;
declare
n number := 1;
n1 number;
s number := 0;
begin
loop
n1 := n + 2;
s := s + n1;
exit when n1 = 99;
end loop;
dbms_output.put_line('Sum of odd numbers between 1 to 100 is ' || s);
end;
/
```

```
SQL> set serveroutput on;
SQL> declare
  2  n number := 1;
  3  n1 number;
  4  s number := 0;
  5  begin
  6  loop
  7  n1 := n + 2;
  8  s := s + n1;
  9  exit when n1 = 99;
10  end loop;
11  dbms_output.put_line('Sum of odd numbers between 1 to 100 is ' || s);
12  end;
13  /
```


3) Write the PL/SQL block to find the factorial of the given number.

```
set serveroutput on
declare
n number := &n;
i number;
f number := 1;
begin
for i in 1..n
loop
f := f*i;
end loop;
dbms_output.put_line(n||'! = '||f);
end;
/
```



```
SQL> set serveroutput on
SQL> declare
  2  n number := &n;
  3  i number;
  4  f number := 1;
  5  begin
  6  for i in 1..n
  7  loop
  8  f := f*i;
  9  end loop;
 10  dbms_output.put_line(n||'! = '||f);
 11  end;
 12  /
Enter value for n: 9
old   2: n number := &n;
new   2: n number := 9;
9! = 362880

PL/SQL procedure successfully completed.
```

4) Write a PL/SQL block to generate Fibonacci series.

```
set serveroutput on
declare
a number := 0;
b number := 1;
c number;
begin
dbms_output.put(a||' '||b||' ');
for i in 3..10 loop
c := a+b;
dbms_output.put(c||' ');
a :=b;
b :=c;
end loop;
dbms_output.put_line(' ');
end;
/
```

```
SQL> set serveroutput on
SQL> declare
  2 a number := 0;
  3 b number := 1;
  4 c number;
  5 begin
  6 dbms_output.put(a||' '||b||' ');
  7 for i in 3..10 loop
  8 c := a+b;
  9 dbms_output.put(c||' ');
 10 a :=b;
 11 b :=c;
 12 end loop;
 13 dbms_output.put_line(' ');
 14 end;
 15 /
0 1 1 2 3 5 8 13 21 34

PL/SQL procedure successfully completed.
```

5) Write a PL/SQL block to reverse a number.

```
set serveroutput on
declare
n number;
s number := 0;
r number;
k number;
begin
n :=&n;
k :=n;
loop
exit when n=0;
s :=s*10;
r :=mod(n,10);
s :=s+r;
n :=trunc(n/10);
end loop;
dbms_output.put_line('The reversed digits of '||K||' = '||S);
end;
/
```

```
SQL> set serveroutput on
SQL> declare
  2  n number;
  3  s number := 0;
  4  r number;
  5  k number;
  6  begin
  7  n :=&n;
  8  k :=n;
  9  loop
 10  exit when n=0;
 11  s :=s*10;
 12  r :=mod(n,10);
 13  s :=s+r;
 14  n :=trunc(n/10);
 15  end loop;
 16  dbms_output.put_line('The reversed digits of '||K||' = '||S);
 17  end;
 18  /
Enter value for n: 2345
old   7: n :=&n;
new   7: n :=2345;
The reversed digits of 2345 = 5432

PL/SQL procedure successfully completed.
```

6) Write a PL/SQL block to find the greatest of three numbers.

```
set serveroutput on
declare
a number := &a;
b number := &b;
c number := &c;
begin
if a>b and a>c then
dbms_output.put_line(a||' is greatest. ');
elsif b>a and b>c then
dbms_output.put_line(b||' is greatest. ');
else
dbms_output.put_line(c||' is greatest. ');
end if;
end;
/
```

```
SQL> set serveroutput on
SQL> declare
  2  a number := &a;
  3  b number := &b;
  4  c number := &c;
  5  begin
  6  if a>b and a>c then
  7  dbms_output.put_line(a||' is greatest. ');
  8  elsif b>a and b>c then
  9  dbms_output.put_line(b||' is greatest. ');
 10  else
 11  dbms_output.put_line(c||' is greatest. ');
 12  end if;
 13  end;
 14  /
Enter value for a: 5
old  2: a number := &a;
new  2: a number := 5;
Enter value for b: 4
old  3: b number := &b;
new  3: b number := 4;
Enter value for c: 6
old  4: c number := &c;
new  4: c number := 6;
6 is greatest.

PL/SQL procedure successfully completed.
```

7) Write a PL/SQL block to find area of circles with radius greater than 3 and less than equal to 7 and store the result in a table with attributes radius and area.

```
set serveroutput on
declare
area number(5,2);
radius number(1):=3;
pi constant number(3,2):=3.14;
begin
while radius<=7
loop
area:=pi*radius*radius;
insert into areas values(radius,area);
radius:=radius+1;
end loop;
end;
/
```

```
SQL> create table areas(radius number(1),area number(5,2));
```

Table created.

```
SQL> set serveroutput on
```

```
SQL> declare
```

```
2 area number(5,2);
3 radius number(1):=3;
4 pi constant number(3,2):=3.14;
5 begin
6 while radius<=7
7 loop
8 area:=pi*radius*radius;
9 insert into areas values(radius,area);
10 radius:=radius+1;
11 end loop;
12 end;
13 /
```

PL/SQL procedure successfully completed.

```
SQL> select * from areas;
```

RADIUS	AREA
3	28.26
4	50.24
5	78.5
6	113.04
7	153.86