

Practical 01

What is PL/SQL

THEORY: PL/SQL is a block structured language. The programs of PL/SQL are logical blocks that can

contain any number of nested sub-blocks. PL/SQL stands for "Procedural Language extension of SQL" that is used in Oracle. PL/SQL is integrated with Oracle database (since version 7).

The functionalities of PL/SQL usually extended after each release of Oracle database.

Although PL/SQL is closely integrated with SQL language, yet it adds some programming constraints that are not available in SQL.

PL/SQL Functionalities

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. It can support Array and handle exceptions (runtime errors). After the implementation of version 8 of Oracle database have included features associated with object orientation. You can create PL/SQL units like procedures, functions, packages, types and triggers, etc. which are stored in the database for reuse by applications.

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow of control statements to process the data.

The PL/SQL is known for its combination of data manipulating power of SQL with data processing power of procedural languages. It inherits the robustness, security, and portability of the Oracle Database.

PL/SQL is not case sensitive so you are free to use lower case letters or upper case letters except within string and character literals. A line of PL/SQL text contains groups of characters known as lexical units. It can be classified as follows:

- Delimiters
- Identifiers
- Literals
- Comments

Introducing PL/SQL block structure and anonymous block

PL/SQL program units organize the code into blocks. A block without a name is known as an anonymous block. The anonymous block is the simplest unit in PL/SQL. It is called anonymous block because it is not saved in the Oracle database.

An anonymous block is an only one-time use and useful in certain situations such as creating test units. The following illustrates anonymous block syntax:

```
[DECLARE]
Declaration statements;
BEGIN
Execution statements;
[EXCEPTION]
Exception handling statements;
END;
```

The anonymous block has three basic sections that are the declaration, execution, and exception handling. Only the execution section is mandatory and the others are optional.

- The declaration section allows you to define data types, structures, and [variables](#). You often declare variables in the declaration section by giving them names, data types, and initial values.
- The execution section is required in a block structure and it must have at least one statement. The execution section is the place where you put the execution code or business logic code. You can use both procedural and SQL statements inside the execution section.
- The exception handling section is starting with the [EXCEPTION](#) keyword. The exception section is the place that you put the code to handle exceptions. You can either catch or handle exceptions in the exception section.

1. Write a PL/SQL block to print "Hello World".

CODE:

1. BEGIN
2. dbms_output.put_line('hello world!');
3. END;
4. /

OUTPUT:

```
SQL> BEGIN
2  dbms_output.put_line('hello world!');
3  END;
4  /
hello world!
```

2. Write a PL/SQL block to declare a variable W and assign its value as 10, then print the value.

CODE:

1. DECLARE
2. W NUMBER;
3. BEGIN
4. W:=10;
5. dbms_output.put_line(W);
6. END;
7. /

OUTPUT:

```
SQL> DECLARE
  2  W NUMBER;
  3  BEGIN
  4  W:=10;
  5  dbms_output.put_line(W);
  6  END;
  7  /
10
```

3. Write a PL/SQL block to find the maximum of two numbers.

CODE:

1. DECLARE
2. A NUMBER;
3. B NUMBER;
4. BEGIN
5. A:=&A;
6. B:=&B;
7. IF A>B THEN
8. dbms_output.put_line('A is greater');
9. ELSE
10. dbms_output.put_line('B is greater');
11. END IF;
12. END;
13. /

OUTPUT:

```

Enter value for a: 5
old 5: A:=&A;
new 5: A:=5;
Enter value for b: 4
old 6: B:=&B;
new 6: B:=4;
A is greater

```

4. Write a PL/SQL block to calculate the factorial of a number.

CODE;

```

1. DECLARE
2. A NUMBER;
3. X NUMBER:=1;
4. BEGIN
5. A:=&A;
6. LOOP
7. X:=X*A;
8. A:=A-1;
9. EXIT WHEN A=0;
10. END LOOP;
11. dbms_output.put_line('factorial is: '||X);
12. END;
13. /

```

OUTPUT:

```

13 /
Enter value for a: 5
old 5: A:=&A;
new 5: A:=5;
factorial is: 120

```

5. Write a PL/SQL block to check if a number is prime.

CODE:

```

1. DECLARE
2. n NUMBER;
3. i NUMBER;
4. temp NUMBER;
5. BEGIN
6. n:=&n;
7. i:=2;
8. temp:=1;

```

```

9.  for i in 2..n/2 LOOP
10. IF MOD(n,i)=0
11. then
12. temp:=0;
13. exit;
14. END IF;
15. END LOOP;
16. if temp=1 THEN
17. dbms_output.put_line('is prime');
18. ELSE
19. dbms_output.put_line('is not a prime');
20. END IF;
21. END;
22. /

```

OUTPUT:

```

18  ELSE
19  dbms_output.put_l
20  END IF;
21  END;
22  /
Enter value for n: 5
old   6: n:=&n;
new   6: n:=5;
is prime

```

6. Write a PL/SQL block to find the nth Fibonacci number.

CODE:

```

1. DECLARE
2.   n NUMBER;
3.   a NUMBER := 0;
4.   b NUMBER := 1;
5.   c NUMBER;
6. BEGIN
7.   n := &n;
8.   dbms_output.put_line(a || ' ' || b || ' ');
9.   FOR i IN 3..n LOOP
10.    c := a + b;
11.    dbms_output.put_line(c || ' ');
12.    a := b;
13.    b := c;
14. END LOOP;

```

15. END;

16. /

OUTPUT:

```
Enter value for n: 8
old 7: n := &n;
new 7: n := 8;
0 1
1
2
3
5
8
13
```

7. Write a PL/SQL block to reverse a string.

CODE:

```
1. DECLARE
2.   a VARCHAR2(100);
3.   b VARCHAR2(100) := '';
4. BEGIN
5.   a := '&a';
6.   FOR i IN REVERSE 1..LENGTH(a) LOOP
7.     b := b || SUBSTR(a, i, 1);
8.   END LOOP;
9.   dbms_output.put_line('Reversed string is: ' || b);
10. END;
11. /
```

OUTPUT:

```
8   END LOOP;
9   dbms_output.put_line('Reversed string is: ' || b);
10  END;
11  /
Enter value for a: plsqr
old 5: a := '&a';
new 5: a := 'plsqr';
Reversed string is: rqspl
```

8. Write a PL/SQL block to check if num is even or odd.

CODE:

```
1. DECLARE
2.   num NUMBER;
3. BEGIN
4.   num := &num;
5.   IF MOD(num, 2) = 0 THEN
6.     dbms_output.put_line(num || ' is even');
7.   ELSE
8.     dbms_output.put_line(num || ' is odd');
9.   END IF;
10. END;
11. /
```

OUTPUT:

```
9      END IF;
10    END;
11    /
Enter value for num: 87543
old 4:   num := &num;
new 4:   num := 87543;
87543 is odd
```

9. Write a PL/SQL block to find the sum of all num from 1 to n.

CODE:

```
1. DECLARE
2.   n NUMBER;
3.   total NUMBER := 0;
4. BEGIN
5.   n := &n;
6.   FOR i IN 1..n LOOP
7.     total := total + i;
8.   END LOOP;
9.   DBMS_OUTPUT.PUT_LINE('Sum of numbers from 1 to ' || n || ' is ' || total);
10. END;
11. /
```

OUTPUT:

```

8      END LOOP;
9      DBMS_OUTPUT.PUT_LINE('Sum
10 END;
11 /
Enter value for n: 8
old   5:      n := &n;
new   5:      n := 8;
Sum of numbers from 1 to 8 is 36

```

10. Write a PL/SQL block to find the length of string.

CODE:

```

1. DECLARE
2.   A VARCHAR2(100);
3.   len NUMBER;
4. BEGIN
5.   A := '&A';
6.   len := LENGTH(A);
7.   dbms_output.put_line('Length of the string is ' || len);
8. END;
9. /

```

OUTPUT:

```

7      dbms_output.put_line('Length of th
8 END;
9 /
Enter value for a: bk birla college
old   5:      A := '&A';
new   5:      A := 'bk birla college';
Length of the string is 16

PL/SQL procedure successfully completed.

```


PRACTICAL NO: 2

AIM: To under the concept of Control structures with PL/SQL.

THEORY: Control structures are essential components of programming languages, allowing developers to direct the flow of execution based on conditions and repetitions.

- There are three types of control structures:

1. Conditional control
2. Iterative control
3. Sequential control

1. *CONDITIONAL CONTROL*: Condition statements checks multiple conditions at a time.

Syntax:

IF THEN ELSE STATEMENT -

- i. IF condition
THEN statement1;
ELSE statement2;
END IF;
- ii. IF condition1
THEN statement1;
ELSEIF condition2
THEN statement2;
ELSE statement3;
ENDIF;
- iii. IF condition1
THEN ELSE IF condition2
THEN statement1;
END IF;
ELSIF condition3;
THEN statement2;
END IF;

Case Statements (or switch statements) provide a streamlined way to execute different blocks of code based on the value of a variable, making them ideal for handling multiple potential cases. There are two types of case statements search case and simple case.

Syntax:

CASE < selector variable >

WHEN < expression 1 > THEN < commands >

WHEN < expression 2 > THEN < commands >

.

.

[ELSE < commands >]

END CASE;

2. ITERATIVE CONTROL Iterative control statements are meant for a sequence of statements that has to be repeated. In PL/SQL there are three ways for iterative statements:

Syntax:

1. LOOP and EXIT

LOOP

< statements >

[EXIT WHEN < condition >;]

END LOOP;

2. WHILE LOOP

WHILE < condition >

LOOP

< commands >

END LOOP;

3. FOR LOOP

FOR variable_name IN

[REVERSE] start..end

LOOP

< commands >

END LOOP;

3) SEQUENTIAL CONTROL

1. GOTO Statement:

In PL/SQL, the GOTO statement is used to branch to a labeled statement within a PL/SQL block. It is generally considered good practice to avoid using GOTO due to its impact on code readability and maintainability.

Syntax:

<label_name>

DECLARE

/* Variable declarations*/

BEGIN

GOTO label_name; -- Jumps to the label

/* More statements */

<label_name>

END;

2. CONTINUE statement: The CONTINUE statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

Syntax:

CONTINUE;

1. Write PL/SQL code for an IF statement that prints a message if a user-provided integer variable is greater than or equal to 10.

Code:

declare

num number:=#

begin

if num>=10 then

dbms_output.put_line('the number is greater or equal to 10.');

end if;

end;

output:

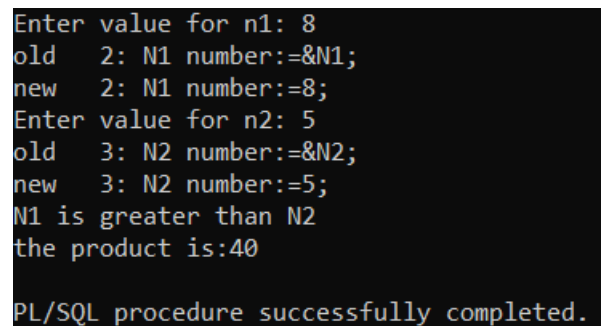
```
Enter value for num: 75
old  2: num number:=&num;
new  2: num number:=75;
the number is greater or equal to 10.

PL/SQL procedure successfully completed.
```

2. Write PL/SQL code that prompts the user for two integer inputs, N1 and N2. If N1 is greater than N2, print "N1 is greater than N2" along with their product. Otherwise, print "N1 is not greater than N2" along with their sum.

Code:

```
DECLARE
N1 number:=&N1;
N2 number:=&N2;
BEGIN
IF N1>N2 THEN
DBMS_OUTPUT.PUT_LINE('N1 is greater than N2');
DBMS_OUTPUT.PUT_LINE('the product is:| |N1*N2);
ELSE
DBMS_OUTPUT.PUT_LINE('N1 is not greater than N2');
DBMS_OUTPUT.PUT_LINE('the sum is:| |N1+N2);
END IF;
END;
OUTPUT:
```

A screenshot of a PL/SQL execution window with a black background and white text. It shows the execution of a PL/SQL block. The first prompt is 'Enter value for n1: 8'. The second line shows 'old 2: N1 number:=&N1;' followed by 'new 2: N1 number:=8;'. The next prompt is 'Enter value for n2: 5'. This is followed by 'old 3: N2 number:=&N2;' and 'new 3: N2 number:=5;'. The output then shows 'N1 is greater than N2' and 'the product is:40'. The final line is 'PL/SQL procedure successfully completed.'

```
Enter value for n1: 8
old 2: N1 number:=&N1;
new 2: N1 number:=8;
Enter value for n2: 5
old 3: N2 number:=&N2;
new 3: N2 number:=5;
N1 is greater than N2
the product is:40
PL/SQL procedure successfully completed.
```

3. Write PL/SQL code that prompts the user for two integer inputs, num1 and num2. Compare the two numbers and print one of the following messages based on their comparison:"NUM1 SMALLER THAN NUM2" if num1 is less than num2"EQUAL" if num1 is equal to num2"NUM2 SMALLER THAN NUM1" if num1 is greater than num2.

CODE:

```
DECLARE
num1 number;
num2 number;
```

```

BEGIN
num1:=&num1;
num2:=&num2;
IF num1<num2 THEN
DBMS_OUTPUT.PUT_LINE('num1 is smaller than num2');
ELSIF num1=num2 THEN
DBMS_OUTPUT.PUT_LINE('both are equal');
ELSE
DBMS_OUTPUT.PUT_LINE('num2 is smaller than num1');
END IF;
END;

```

Output:

```

Enter value for num1: 6
old 5: num1:=&num1;
new 5: num1:=6;
Enter value for num2: 8
old 6: num2:=&num2;
new 6: num2:=8;
num1 is smaller than num2

PL/SQL procedure successfully completed.

```

4. Write PL/SQL code that assigns an age to a variable and uses a CASE statement to classify the age into one of the following categories:

- "Child" if the age is less than 13,
- "Teenager" if the age is between 13 and 19,
- "Young Adult" if the age is between 20 and 35,
- "Middle-Aged Adult" if the age is between 36 and 55,
- "Senior Adult" if the age is above 55,
- "Unknown Age Group" for any other cases.

Code:

```

declare
age number;
begin

```

```

age:=25;
case
when age<13 then
dbms_output.put_line('child');
when age between 13 and 19 then
dbms_output.put_line('teenage');
when age between 20 and 35 then
dbms_output.put_line('young adult');
when age between 36 and 55 then
dbms_output.put_line('middle age adult');
when age>55 then
dbms_output.put_line('senior adult');
else
dbms_output.put_line('Unknown Age Group');
end case;
end;
output:

```

```

young adult
PL/SQL procedure successfully completed.

```

5. Write PL/SQL code that assigns a grade to a variable and uses a CASE statement to classify the grade into one of the following categories:

- If A prints “EXCELLENT”,
- If N prints “VERY GOOD”,
- If C prints “GOOD”,
- If D prints “AVERAGE”,
- If E prints “PASSED WITH GRACE MARKS”.

Code:

```

declare
grade char(1);
begin
grade:='B';

```

```
case grade
when 'A' then
dbms_output.put_line('excellent');
when 'B' then
dbms_output.put_line('very good');
when 'C' then
dbms_output.put_line('good');
when 'D' then
dbms_output.put_line('average');
when 'E' then
dbms_output.put_line('passed with grace marks');
end case;
end;
OUTPUT:
```

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

6. Write a PL/SQL Code Using the GOTO Statement

CODE:

DECLARE

num NUMBER;

fact NUMBER;

BEGIN

num := 5;

fact := 1;

IF num < 10 THEN

GOTO loopstart;

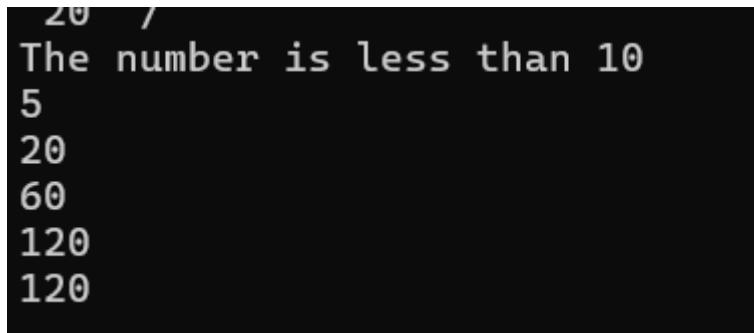
END IF;

```

<<loopstart>>
DBMS_OUTPUT.PUT_LINE('The number is less than 10');

FOR i IN REVERSE 1..num LOOP
    fact := fact * i;
    DBMS_OUTPUT.PUT_LINE(fact);
END LOOP;
END;
/
OUTPUT:

```



```

20 /
The number is less than 10
5
20
60
120
120

```

7. What does the GOTO end_of_program; statement do in the following PL/SQL block?

CODE:

DECLARE

num NUMBER;

BEGIN

DBMS_OUTPUT.PUT_LINE('Welcome to the Program');

num := -2;

IF num > 0 THEN

DBMS_OUTPUT.PUT_LINE('You entered a positive number.');

GOTO end_of_program;

ELSE

DBMS_OUTPUT.PUT_LINE('You entered either zero or a negative number.');

END IF;

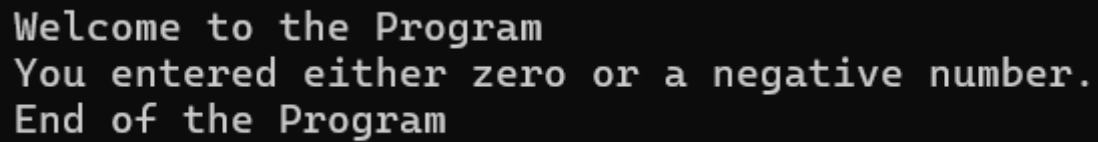
<<end_of_program>>


```

DBMS_OUTPUT.PUT_LINE('End of the Program');
END;
/

```

OUTPUT:



```

Welcome to the Program
You entered either zero or a negative number.
End of the Program

```

Since the num is negative. The program directly jump to the label '<<end_of_program>>'

8. Write PL/SQL code to print odd number from 1 to 10 using CONTINUE statement.

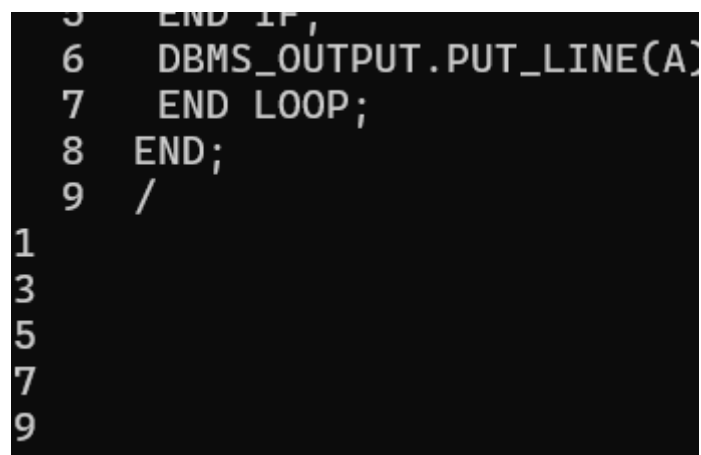
CODE:

```

BEGIN
FOR A IN 1 .. 10 LOOP
IF MOD(A, 2) = 0 THEN
CONTINUE;
END IF;
DBMS_OUTPUT.PUT_LINE(A);
END LOOP;
END;
/

```

OUTPUT:



```

5      END IF;
6      DBMS_OUTPUT.PUT_LINE(A);
7      END LOOP;
8      END;
9      /

1
3
5
7
9

```

9.) Write a PL/SQL code Loop Exit query. (WHILE LOOP)

CODE:

DECLARE

A NUMBER := 1;

BEGIN

WHILE A < 10

LOOP

DBMS_OUTPUT.PUT_LINE('Counter : ' || A);

A := A + 1 ;

EXIT WHEN A = 7;

END LOOP;

END;

OUTPUT:

```
8      EXIT WHEN A = 7;
9      END LOOP;
10     END;
11    /
Counter : 1
Counter : 2
Counter : 3
Counter : 4
Counter : 5
Counter : 6
```

10. Write pl/sql program to define NOT NULL constraint

CODE:

DECLARE

rollno number(4);

dept varchar2(100) NOT NULL := 'IT DEPARTMENT';

BEGIN

rollno := 200;

dbms_output.put_line(' The salary= ' || rollno || ' and Dept Name= ' || dept);

END;

/

OUTPUT:

```
2  rollno number(4);
3  dept varchar2(100) NOT NULL := 'IT DEPARTME
4  BEGIN
5  rollno := 200;
6  dbms_output.put_line(' The salary= '||rolln
7  END;
8  /
The salary= 200 and Dept Name= IT DEPARTMENT
```

Practical 03

AIM: To understand the concept of Procedures in PLSQL

THEORY: The PL/SQL stored procedure or simply a procedure is a PL/SQL block which performs one or more specific tasks. It is just like procedures in other programming languages.

The procedure contains a header and a body.

1. **Header:** The header contains the name of the procedure and the parameters or variables passed to the procedure.
2. **Body:** The body contains a declaration section, execution section and exception section similar to a general PL/SQL block.

When you want to create a procedure or function, you have to define parameters .There is three ways to pass parameters in procedure:

1. **IN parameters:** The IN parameter can be referenced by the procedure or function. The value of the parameter cannot be overwritten by the procedure or the function.
2. **OUT parameters:** The OUT parameter cannot be referenced by the procedure or function, but the value of the parameter can be overwritten by the procedure or function.
3. **INOUT parameters:** The INOUT parameter can be referenced by the procedure or function and the value of the parameter can be overwritten by the procedure or function.

Syntax for creating procedure:

```
CREATE [OR REPLACE] PROCEDURE  
procedure_name [ (parameter [,parameter])  
]  
IS  
  
[declaration_section]  
  
BEGIN  
  
executable_section  
  
[EXCEPTION  
  
exception_section]  
  
END [procedure_name];
```

Calling Procedures:

Begin

Procedure_name(value

s);

*dbms_output.put_line('record inserted successfully'); **END;***

1. Write a PL/SQL Procedure (greeting) for Hello world?

CODE:

1. CREATE OR REPLACE PROCEDURE greetings(
2. a OUT VARCHAR2
3.) AS
4. BEGIN
5. a:='Hello World';
6. END;
7. /
8. DECLARE
9. x VARCHAR2(100);
10. BEGIN
11. greetings(x);
12. dbms_output.put_line(x);
13. END;
14. /

OUTPUT:

```
SQL> set serveroutput on;  
SQL> DECLARE  
2  x VARCHAR2(100);  
3  BEGIN  
4  greetings(x);  
5  dbms_output.put_line(x);  
6  END;  
7  /  
Hello World
```

2. Write a PL/SQL code for procedure with an IN parameter(for user_name)?

CODE:

1. CREATE OR REPLACE PROCEDURE secondtask(
2. user_name IN VARCHAR2
3.) AS
4. BEGIN
5. dbms_output.put_line('Hello ' || user_name);
6. END;
7. /
8. DECLARE
9. username VARCHAR2(100);

```

10. BEGIN
11.  username := '&username';
12.  secondtask(username);
13. END;
14. /

```

OUTPUT:

```

SQL> DECLARE
  2  username VARCHAR2(100);
  3  BEGIN
  4  username := '&username';
  5  secondtask(username);
  6  END;
  7  /
Enter value for username: STUDENTT
old  4:  username := '&username';
new  4:  username := 'STUDENTT';
Hello STUDENTT

PL/SQL procedure successfully completed.

SQL> |

```

3. Write a PL/SQL block that defines a procedure findMin to find the minimum of two numbers using an OUT parameter and then call this procedure to determine the minimum of 23 and 45.

CODE:

```

1. CREATE OR REPLACE PROCEDURE findmin(a IN NUMBER, b IN NUMBER, result OUT
   VARCHAR2 ) AS
2. BEGIN
3. IF a<b THEN
4. result:='A(23) is minimum';
5. ELSE
6. result:='A(23) is minimum';
7. END IF;
8. dbms_output.put_line(result);
9. END;
10. /
11. DECLARE
12. x NUMBER:=23;
13. y NUMBER:=45;

```

```

14. final VARCHAR2(100);
15. BEGIN
16. findmin(x,y,final);
17. dbms_output.put_line(final);
18. END;
19. /

```

OUTPUT:

```

SQL> DECLARE
  2  x NUMBER:=23;
  3  y NUMBER:=45;
  4  final VARCHAR2(100);
  5  BEGIN
  6  findmin(x,y,final);
  7  dbms_output.put_line(final);
  8  END;
  9  /
A(23) is minimum

PL/SQL procedure successfully completed.

```

4. Write a PL/SQL block that defines a procedure squareNum using an IN OUT parameter to compute the square of a number and then call this procedure to calculate the square of 23.

CODE:

```

1. CREATE OR REPLACE PROCEDURE squarenum( result IN OUT NUMBER ) AS
2. BEGIN
3. result:= result*result;
4. dbms_output.put_line(result);
5. END;
6. /
7. DECLARE
8. x NUMBER;
9. BEGIN
10. x:=23;
11. squarenum(x);
12. END;
13. /

```


OUTPUT:

```
SQL> DECLARE
  2  x NUMBER;
  3  BEGIN
  4  x:=23;
  5  squarenum(x);
  6  END;
  7  /
529
```

5. Write a PL/SQL procedure named `increase_salary` that takes an employee's salary as an IN OUT parameter and a percentage increase as an IN parameter, and increases the salary by the specified percentage.

CODE:

1. CREATE OR REPLACE PROCEDURE `increase_salary`(`employees_salary` IN OUT NUMBER, `percentage` IN NUMBER, `result` OUT NUMBER) AS
2. BEGIN
3. `result:= employees_salary + ((employees_salary*percentage)/100);`
4. `employees_salary:=result;`
5. `dbms_output.put_line('The salary increased by ' || percentage || '% is' || result);`
6. END;
7. /
8. DECLARE
9. `employee_salary NUMBER:=&employee_salary;`
10. `percentage NUMBER:=&percentage;`
11. `new_sal NUMBER;`
12. BEGIN
13. `increase_salary(employee_salary,percentage,new_sal);`
14. END;
15. /

OUTPUT:

```
SQL> DECLARE
  2  employee_salary NUMBER:=&employee_salary;
  3  percentage NUMBER:=&percentage;
  4  new_sal NUMBER;
  5  BEGIN
  6  increase_salary(employee_salary,percentage,new_sal);
  7  END;
  8  /
Enter value for employee_salary: 10000
old  2: employee_salary NUMBER:=&employee_salary;
new  2: employee_salary NUMBER:=10000;
Enter value for percentage: 5
old  3: percentage NUMBER:=&percentage;
new  3: percentage NUMBER:=5;
The salary increased by 5% is10500
```

6. How do you create a procedure with IN, OUT, and IN OUT parameters?/ "Create a PL/SQL procedure calculate_discount that calculates a discount and final price using IN, OUT, and IN OUT parameters.

CODE:

1. CREATE OR REPLACE PROCEDURE calculate_discount(final_price IN OUT NUMBER, original_price IN NUMBER, discount IN NUMBER) AS
2. BEGIN
3. final_price:=original_price-((original_price * discount)/100);
4. dbms_output.put_line('The Final Price After giving discount of '||discount||' is: ' || final_price);
5. END;
6. /
7. DECLARE
8. price NUMBER:=&price;
9. percent_discount NUMBER:=&percent_discount;
10. result NUMBER;
11. BEGIN
12. calculate_discount(result,price,percent_discount);
13. END;
14. /

OUTPUT:

```
7. END;
8. /
Enter value for price: 4000
old 2: price NUMBER:=&price;
new 2: price NUMBER:=4000;
Enter value for percent_discount: 10
old 3: percent_discount NUMBER:=&percent_discount;
new 3: percent_discount NUMBER:=10;
The Final Price After giving discount of 10 is: 3600

PL/SQL procedure successfully completed.
```

7. Create a PL/SQL procedure greet_user_default with a default parameter value that prints a welcome message, and demonstrate calling it with and without providing a parameter.

CODE:

1. CREATE OR REPLACE PROCEDURE greet_user_default (
2. p_name IN VARCHAR2 DEFAULT 'Guesst'
3.) IS
4. BEGIN
5. DBMS_OUTPUT.PUT_LINE('Welcome, ' || p_name || '!');
6. END greet_user_default;
7. /

```

8. BEGIN
9.   greet_user_default;
10. END;
11. /
12. BEGIN
13.   greet_user_default('Student of Birla College');
14. END;
15. /

```

OUTPUT:

```

SQL> SET SERVEROUTPUT ON;
SQL>   BEGIN
      2   greet_user_default;
      3   END;
      4   /
Welcome, Guesst!

PL/SQL procedure successfully completed.

SQL> BEGIN
      2   greet_user_default('Student of Birla College');
      3   END;
      4   /
Welcome, Student of Birla College!

PL/SQL procedure successfully completed.

```

8. Write a PL/SQL procedure add_student to insert a new student's details into the student_enrollment table and demonstrate calling it to add a student with specified attributes.

CODE:

```

1. CREATE OR REPLACE PROCEDURE add_student (
2.   p_student_id IN NUMBER,
3.   p_fname IN VARCHAR2,
4.   p_lname IN VARCHAR2
5. ) IS
6. BEGIN
7.   INSERT INTO student (
8.     student_id,
9.     fname,
10.    lname
11.  ) VALUES (
12.    p_student_id,
13.    p_fname,
14.    p_lname
15.  );

```

```

16. DBMS_OUTPUT.PUT_LINE('Student added successfully.');
```

```

17. END add_student;
18. /
19. BEGIN
20. add_student(
21.     p_student_id => 1,
22.     p_fname => 'Student1 of birla institutete',
23.     p_lname => 'sarname of student1'
24. );
25. END;
26. /
```

OUTPUT:

```

SQL> BEGIN
  2     add_student(
  3         p_student_id => 1,
  4         p_fname => 'Student1 of birla institutete',
  5         p_lname => 'sarname of student1'
  6     );
  7 END;
  8 /
Student added successfully.

PL/SQL procedure successfully completed.

SQL>
SQL> select * from student;

STUDENT_ID FNAME
-----
LNAME
-----
          1 Student1 of birla institutete
sarname of student1
```

9. . How can you use procedures to update data in a table?

CODE:

```

1. CREATE OR REPLACE PROCEDURE update_student (
2.     p_student_id IN NUMBER,
3.     p_new_lname IN VARCHAR2
4. ) IS
5. BEGIN
6.     UPDATE student
7.     SET lname = p_new_lname
```

```

8.   WHERE student_id = p_student_id;
9.   DBMS_OUTPUT.PUT_LINE('Student last name updated successfully. ');
10. END update_student;
11. /
12. BEGIN
13.   update_student(
14.     p_student_id => 1,
15.     p_new_lname => 'UPDATED lastname'
16.   );
17. END;
18. /

```

OUTPUT:

```

SQL> BEGIN
  2      update_student(
  3          p_student_id => 1,
  4          p_new_lname => 'UPDATED lastname'
  5      );
  6  END;
  7  /
Student last name updated successfully.

PL/SQL procedure successfully completed.

SQL> select * from student;

STUDENT_ID FNAME
-----
LNAME
-----
          1 Student1 of birla institute
UPDATED lastname

```

10. How can you delete data using a PL/SQL procedure?

CODE;

```
1. CREATE OR REPLACE PROCEDURE delete_student (  
2.   p_student_id IN NUMBER  
3. ) IS  
4. BEGIN  
5.   DELETE FROM student  
6.   WHERE student_id = p_student_id;  
7.   DBMS_OUTPUT.PUT_LINE('Student deleted successfully.');
```

8. END delete_student;
9. /

```
10. BEGIN  
11.   delete_student(  
12.     p_student_id => 1  
13.   );  
14. END;  
15. /
```

OUTPUT:

```
SQL> BEGIN  
  2      delete_student(  
  3          p_student_id => 1  
  4      );  
  5  END;  
  6  /  
Student deleted successfully.  
  
PL/SQL procedure successfully completed.  
  
SQL> SELECT * FROM student  
  2  /  
  
no rows selected
```

Practical 04

Aim: To understand FUNCTIONS in plsql

THEORY: The PL/SQL Function is very similar to PL/SQL Procedure. The main difference between procedure and a function is, a function must always return a value, and on the other hand a procedure may or may not return a value. Except this, all the other things of PL/SQL procedure are true for PL/SQL function too.

Syntax to create a function:

1. CREATE [OR REPLACE] FUNCTION function_name [parameters]
2. [(parameter_name [IN | OUT | IN OUT] type [, ...])]
3. RETURN return_datatype
4. {IS | AS}
5. BEGIN
6. < function_body >
7. END [function_name];

The function must contain a return statement.

- RETURN clause specifies that data type you are going to return from the function.
- Function_body contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

PL/SQL Drop Function

Syntax for removing your created function:

If you want to remove your created function from the database, you should use the following syntax.

DROP FUNCTION function_name;

1. Create a function to calculate the sum of two numbers.

CODE:

1. CREATE OR REPLACE FUNCTION sum_two_numbers(a NUMBER, b NUMBER)
2. RETURN NUMBER
3. IS
4. result NUMBER;
5. BEGIN
6. result := a + b;
7. RETURN result;

```

8. END;
9. /
10.
11. DECLARE
12.   x NUMBER := &x;
13.   y NUMBER := &y;
14. BEGIN
15.   dbms_output.put_line('The sum of ' || x || ' and ' || y || ' is ' ||
      sum_two_numbers(x, y));
16. END;
17. /

```

OUTPUT:

```

Enter value for x: 10
old 2:      x NUMBER := &x;
new 2:      x NUMBER := 10;
Enter value for y: 20
old 3:      y NUMBER := &y;
new 3:      y NUMBER := 20;
The sum of 10 and 20 is 30

```

2. Create a function to return the greater of two numbers.

CODE:

```

1. CREATE OR REPLACE FUNCTION greaterthan(a NUMBER, b NUMBER)
2. RETURN VARCHAR2
3. IS
4.   c VARCHAR2(100);
5. BEGIN
6.   IF a > b THEN
7.     c := 'The first number is greater';
8.   ELSE
9.     c := 'The second number is larger';
10.  END IF;
11.  RETURN c;
12. END;
13. /
14.
15. DECLARE
16.   x NUMBER := &x;
17.   y NUMBER := &y;
18. BEGIN
19.   dbms_output.put_line(greaterthan(x, y));

```


20. END;

21. /

OUTPUT:

```
Enter value for x: 14
old 2:      x NUMBER := &x;
new 2:      x NUMBER := 14;
Enter value for y: 12
old 3:      y NUMBER := &y;
new 3:      y NUMBER := 12;
The first number is greater

PL/SQL procedure successfully completed.
```

3. Create a function to return 'Even' if a number is even, and 'Odd' if a number is odd.

CODE:

```
1. CREATE OR REPLACE FUNCTION evenodd(a NUMBER)
2. RETURN VARCHAR2
3. IS
4.   x VARCHAR2(100);
5. BEGIN
6.   IF a MOD 2 = 0 THEN
7.     x := 'even';
8.   ELSE
9.     x := 'odd';
10.  END IF;
11.  RETURN x;
12. END;
13. /
14.
15. DECLARE
16.   num NUMBER := &num;
17. BEGIN
18.   dbms_output.put_line(evenodd(num));
19. END;
20. /
```

OUTPUT:

```

Enter value for num: 25
old 2:      num NUMBER := &num;
new 2:      num NUMBER := 25;
odd

```

```

6 /
Enter value for num: 12
old 2:      num NUMBER := &num;
new 2:      num NUMBER := 12;
even

```

4. Write PL/SQL code for a function named `get_square` that calculates the square of a given number.

CODE:

1. CREATE OR REPLACE FUNCTION get_square(a NUMBER)
2. RETURN NUMBER
3. IS
4. result NUMBER;
5. BEGIN
6. result := a * a;
7. RETURN result;
8. END;
9. /
- 10.
11. DECLARE
12. num NUMBER := #
13. BEGIN
14. dbms_output.put_line('The square of ' || num || ' is ' || get_square(num));
15. END;
16. /

OUTPUT:

```

6 /
Enter value for num: 13
old 2:      num NUMBER := &num;
new 2:      num NUMBER := 13;
The square of 13 is 169

```

5. Write PL/SQL code for a function named `get_square` that calculates the square of a given number.

CODE:

1. CREATE OR REPLACE FUNCTION temperature(a NUMBER)
2. RETURN NUMBER
3. IS
4. RESULT NUMBER;
5. BEGIN
6. RESULT := (9/5 * a) + 32;
7. RETURN RESULT;

```

8. END;
9. /
10.
11. DECLARE
12.   temp NUMBER := &temp;
13. BEGIN
14.   dbms_output.put_line(temperature(temp));
15. END;
16. /

```

OUTPUT:

```

6 /
Enter value for temp: 35
old 2:   temp NUMBER := &temp;
new 2:   temp NUMBER := 35;
95

```

6. Create a function to calculate the factorial of a number.

CODE:

```

1. CREATE OR REPLACE FUNCTION factorial(n NUMBER)
2. RETURN NUMBER
3. IS
4.   result NUMBER := 1;
5. BEGIN
6.   IF n = 0 THEN
7.     RETURN 1;
8.   ELSE
9.     FOR i IN 1..n LOOP
10.      result := result * i;
11.    END LOOP;
12.   END IF;
13.   RETURN result;
14. END;
15. /
16.
17. DECLARE
18.   num NUMBER := &num;
19. BEGIN
20.   dbms_output.put_line('The factorial of ' || num || ' is ' || factorial(num));
21. END;
22. /

```

OUTPUT:

```

6 /
Enter value for num: 6
old 2: num NUMBER := &num;
new 2: num NUMBER := 6;
The factorial of 6 is 720

```

7. Write PL/SQL for function which calculates the nth Fibonacci number using recursion.

CODE:

```

1. CREATE OR REPLACE FUNCTION fibonacci(n NUMBER)
2. RETURN NUMBER
3. IS
4.     result NUMBER;
5. BEGIN
6.     IF n = 0 THEN
7.         result := 0;
8.     ELSIF n = 1 THEN
9.         result := 1;
10.    ELSE
11.        result := fibonacci(n - 1) + fibonacci(n - 2);
12.    END IF;
13.    RETURN result;
14. END;
15. /
16.
17. DECLARE
18.     num NUMBER := &num;
19.     i NUMBER;
20. BEGIN
21.     FOR i IN 0..num LOOP
22.         dbms_output.put_line(fibonacci(i));
23.     END LOOP;
24. END;
25. /

```

OUTPUT:

```

Enter value for num: 10
old 2:      num NUMBER := &num;
new 2:      num NUMBER := 10;
0
1
1
2
3
5
8
13
21
34
55

PL/SQL procedure successfully completed.

```

8. Write a plsql program for function which calculates the area and perimeter of circle radius given by user

CODE:

```

1. CREATE OR REPLACE FUNCTION circle_area_perimeter(r NUMBER)
2. RETURN VARCHAR2
3. IS
4.   area NUMBER;
5.   perimeter NUMBER;
6. BEGIN
7.   area := 3.14159 * r * r;
8.   perimeter := 2 * 3.14159 * r;
9.   RETURN 'Area: ' || area || ', Perimeter: ' || perimeter;
10. END;
11. /
12.
13. DECLARE
14.   r NUMBER := &r;
15. BEGIN
16.   dbms_output.put_line(circle_area_perimeter(r));
17. END;
18. /

```

OUTPUT:

```

Enter value for r: 14
old 2:      r NUMBER := &r;
new 2:      r NUMBER := 14;
Area: 615.75164, Perimeter: 87.96452

PL/SQL procedure successfully completed.

```

9. Write a plsql program to create function that prints average marks of student in 3 subject.. Get input from user

CODE:

1. CREATE OR REPLACE FUNCTION avgmarks(sub1 NUMBER, sub2 NUMBER, sub3 NUMBER)
2. RETURN NUMBER
3. IS
4. avg_marks NUMBER;
5. BEGIN
6. avg_marks := (sub1 + sub2 + sub3) / 3;
7. RETURN avg_marks;
8. END;
9. /
- 10.
11. DECLARE
12. maths NUMBER := &maths;
13. physics NUMBER := &physics;
14. chemistry NUMBER := &chemistry;
15. BEGIN
16. dbms_output.put_line('The average marks of the student are: ' || avgmarks(maths,physics,chemistry));
17. END;
18. /

OUTPUT:

```

Enter value for maths: 20
old 2:      maths NUMBER := &maths;
new 2:      maths NUMBER := 20;
Enter value for physics: 16
old 3:      physics NUMBER := &physics;
new 3:      physics NUMBER := 16;
Enter value for chemistry: 12
old 4:      chemistry NUMBER := &chemistry;
new 4:      chemistry NUMBER := 12;
The average marks of the student are: 16

```

10. Write a plsql program to create a function which check whether the user input is a vowel or not

CODE:

```
1. CREATE OR REPLACE FUNCTION isvowel(c CHAR)
2. RETURN VARCHAR2
3. IS
4.     result VARCHAR2(50);
5. BEGIN
6.     IF LOWER(c) IN ('a', 'e', 'i', 'o', 'u') THEN
7.         result := 'The character is a vowel.';
8.     ELSE
9.         result := 'The character isnt a vowel.';
10.    END IF;
11.    RETURN result;
12. END;
13. /
14. DECLARE
15.     alphabat CHAR := '&alphabat';
16. BEGIN
17.     dbms_output.put_line(isvowel(alphabat));
18. END;
19. /
```

OUTPUT:

```
Enter value for alphabat: A
old 2:      alphabat CHAR := '&alphabat';
new 2:      alphabat CHAR := 'A';
The character is a vowel.

PL/SQL procedure successfully completed.

SQL> DECLARE
2      alphabat CHAR := '&alphabat';
3 BEGIN
4      dbms_output.put_line(isvowel(alphabat));
5 END;
6 /
Enter value for alphabat: z
old 2:      alphabat CHAR := '&alphabat';
new 2:      alphabat CHAR := 'z';
The character isnt a vowel.
```

Practical 05

Aim: To understand CURSORS in plsql

THEORY: When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement. In PL/SQL, the context area is controlled by Cursor. A cursor contains

information on a select statement and the rows of data accessed by it.

A cursor is used to referred to a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors:

- Implicit Cursors
- Explicit Cursors

1) PL/SQL Implicit Cursors

The implicit cursors are automatically generated by Oracle while an SQL statement is executed, if you don't use an explicit cursor for the statement.

These are created by default to process the statements when DML statements like INSERT, UPDATE, DELETE etc. are executed.

Oracle provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are: %FOUND, %NOTFOUND, %ROWCOUNT and %ISOPEN.

Attribute	Description
%FOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect at least one row or more rows or a SELECT INTO statement returned one or more rows. Otherwise it returns FALSE.
%NOTFOUND	Its return value is TRUE if DML statements like INSERT, DELETE and UPDATE affect no row, or a SELECT INTO statement return no rows. Otherwise it returns FALSE. It is a just opposite of %FOUND.
%ISOPEN	It always returns FALSE for implicit cursors, because the SQL cursor is automatically closed after executing its associated SQL statements.
%ROWCOUNT	It returns the number of rows affected by DML statements like INSERT, DELETE, and UPDATE or returned by a SELECT INTO statement.

2) PL/SQL Explicit Cursors

The Explicit cursors are defined by the programmers to gain more control over the context area. These cursors should be defined in the declaration section of the PL/SQL block. It is created on a SELECT statement which returns more than one row.

Following is the syntax to create an explicit cursor:

Syntax of explicit cursor

Following is the syntax to create an explicit cursor:

1. `CURSOR cursor_name IS select_statement;;`

Steps:

You must follow these steps while working with an explicit cursor.

1. Declare the cursor to initialize in the memory.
 2. Open the cursor to allocate memory.
 3. Fetch the cursor to retrieve data.
 4. Close the cursor to release allocated memory.
1. Write a plsql block to delete employoess who have left company(indicated by NULL value in end_date) from emp table. Display the number of rows deleted

CODE:

```
1. BEGIN
2. DELETE FROM emp
3. WHERE end_date IS NOT NULL;
4.
5. IF SQL%ROWCOUNT>0 THEN
6.     dbms_output.put_line(SQL%ROWCOUNT || ' employess are deleted');
7. ELSE
8.     dbms_output.put_line('NO EMPLOYESS DETECTED');
9. END IF;
10. END;
11. /
```

OUTPUT:

```

SQL> BEGIN
2  DELETE FROM emp
3  WHERE end_date IS NOT NULL;
4
5  IF SQL%ROWCOUNT>0 THEN
6      dbms_output.put_line(SQL%ROWCOUNT || ' employess are deleted');
7  ELSE
8      dbms_output.put_line('NO EMPLOYESS DETECTED');
9  END IF;
10 END;
11 /
2 employess are deleted

```

2. Write a PL/SQL block that updates the salary of employees in the emp table by adding 1000 to their current salary for those who belong to a department with the department number D001. The block should handle three scenarios: If the update is successful, it should output "Salary Updated." If no records are found for the given department, it should output "Record not found..." If no rows are updated, it should output "No records to be updated."

CODE:

```

1. BEGIN
2. UPDATE emp2
3. SET emp_sal =emp_sal+1
4. WHERE dept_no = 'D002';
5. IF SQL%FOUND THEN
6. dbms_output.put_line('salary UPDATED');
7. END IF;
8. IF SQL%NOTFOUND THEN
9. dbms_output.put_line('record not FOUND');
10. END IF;
11. IF SQL%ROWCOUNT>0 THEN
12. dbms_output.put_line(SQL%ROWCOUNT || ' ROWS are updated');
13. ELSE
14. dbms_output.put_line('nothing to update');
15. END IF;
16. END;
17. /

```

OUTPUT:

```

SQL> BEGIN
  2  UPDATE emp2
  3  SET emp_sal =emp_sal+1
  4  WHERE dept_no = 'D002';
  5  IF SQL%FOUND THEN
  6  dbms_output.put_line('salary UPDATED');
  7  END IF;
  8  IF SQL%NOTFOUND THEN
  9  dbms_output.put_line('record not FOUND');
 10  END IF;
 11  IF SQL%ROWCOUNT>0 THEN
 12  dbms_output.put_line(SQL%ROWCOUNT || ' ROWS are updated');
 13  ELSE
 14  dbms_output.put_line('nothing to update');
 15  END IF;
 16  END;
 17  /
salary UPDATED
1 ROWS are updated

```

3. Explain how a cursor FOR loop works in PL/SQL

CODE:

```

1. DECLARE
2.   CURSOR e_emp_cursor IS
3.     SELECT EMP_NO, EMP_SAL FROM emp2
4.     WHERE DEPT_NO= 'D001';
5. BEGIN
6.   FOR emp_record IN e_emp_cursor
7.   LOOP
8.     DBMS_OUTPUT.PUT_LINE('record founded: ' || emp_record.EMP_NO || ' ' ||
emp_record.EMP_SAL);
9.   END LOOP;
10. END;
11. /

```

OUTPUT:

```

  9      END LOOP;
 10  END;
 11  /
record founded: E001      25000

PL/SQL procedure successfully completed.

```

4. Write a PL/SQL block that inserts a new department into the Departments table and displays a confirmation message showing the number of rows inserted.

CODE:

```

1. BEGIN
2.   INSERT INTO emp2 (EMP_NO, DEPT_NO)

```

```

3.  VALUES ('E006', 'D066');
4.
5.  IF SQL%ROWCOUNT > 0 THEN
6.      DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' Value inserted. ');
7.  ELSE
8.      DBMS_OUTPUT.PUT_LINE('No rows inserted. ');
9.  END IF;
10. END;
11. /

```

OUTPUT:

```

10  END;
11  /
1 Value inserted.

PL/SQL procedure successfully completed.

SQL> SELECT * FROM emp2
2  /

```

EMP_NO	EMP_NAME	EMP_SAL	DEPT_NO
E001	TABISH	25000	D001
E002	FATIMA	400	D002
E006			D066
E004	AYSHA	1000	D004
E005	NAZMIN	5000	D005

5. Describe what happens when the cursor c_emp_cursor is opened, fetched, and closed in the PL/SQL block. If the employees table has 3 employees in department 30, what output would this block produce?

CODE:

```

1. DECLARE
2.     CURSOR c_emp_cursor IS
3.         SELECT EMP_NAME, EMP_SAL FROM emp2
4.         WHERE EMP_NO= 'E002';
5.     v_empno emp2.EMP_NAME%TYPE;
6.     v_lname emp2.EMP_SAL%TYPE;
7. BEGIN
8.     OPEN c_emp_cursor;
9.     LOOP
10.        FETCH c_emp_cursor INTO v_empno, v_lname;
11.        EXIT WHEN c_emp_cursor%NOTFOUND;
12.        DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_lname);
13.    END LOOP;
14.    CLOSE c_emp_cursor;
15. END;
16. /

```

OUTPUT:

```
SQL> DECLARE
2     CURSOR c_emp_cursor IS
3         SELECT EMP_NAME,EMP_SAL FROM emp2
4         WHERE EMP_NO= 'E001';
5     v_empno emp2.EMP_NAME%TYPE;
6     v_lname emp2.EMP_SAL%TYPE;
7 BEGIN
8     OPEN c_emp_cursor;
9     LOOP
10        FETCH c_emp_cursor INTO v_empno, v_lname;
11        EXIT WHEN c_emp_cursor%NOTFOUND;
12        DBMS_OUTPUT.PUT_LINE(v_empno || ' ' || v_lname);
13    END LOOP;
14    CLOSE c_emp_cursor;
15 END;
16 /
TABISH 25000
```

6. Write a PL/SQL block to update the salary of employees in the Employees table by 10% and print the number of rows affected using an implicit cursor.

CODE:

1. BEGIN
2. UPDATE emp2
3. SET EMP_SAL = EMP_SAL * 1.10;
4. IF SQL%ROWCOUNT > 0 THEN
5. DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
6. ELSE
7. DBMS_OUTPUT.PUT_LINE('No rows updated.');
8. END IF;
9. END;
10. /

OUTPUT:

```
SQL> BEGIN
2     UPDATE emp2
3     SET EMP_SAL = EMP_SAL * 1.10;
4     IF SQL%ROWCOUNT > 0 THEN
5         DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT || ' rows updated.');
```

7. If the Employees table initially has 5 rows and all employees have their salaries increased by 5000, what output will this block produce? Explain why.

CODE:

1. DECLARE

```

2.  total_rows NUMBER(2);
3.  BEGIN
4.    UPDATE emp2
5.    SET EMP_SAL = EMP_SAL + 5000;
6.
7.    IF sql%notfound THEN
8.      DBMS_OUTPUT.PUT_LINE('No employees updated');
9.    ELSIF sql%found THEN
10.     total_rows := sql%rowcount;
11.     DBMS_OUTPUT.PUT_LINE(total_rows || ' employees updated');
12.  END IF;
13. END;
14. /

```

OUTPUT:

```

5 employees updated

PL/SQL procedure successfully completed.

SQL> SELECT * FROM EMP2;

```

EMP_NO	EMP_NAME	EMP_SAL	DEPT_NO
E001	TABISH	32500	D001
E002	FATIMA	5440	D002
E006			D066
E004	AYSHA	6100	D004
E005	NAZMIN	10500	D005

8. What will be the output of this block if there are 5 rows in the customers table before the update? Explain why

CODE:

```

1. declare
2.  rowCount NUMBER;
3.  begin
4.    update EMP2 set DEPT_NO = 'D010'
5.    WHERE EMP_NO = 'E001';
6.    if sql%found then
7.      rowCount := sql%rowcount;
8.      dbms_output.put_line(rowCount || 'is replaced');
9.    end if;
10. end;
11. /

```

OUTPUT:

```

11 /
11s replaced

PL/SQL procedure successfully completed.

SQL> SELECT * FROM emp2;

EMP_NO      EMP_NAME      EMP_SAL  DEPT_NO
-----
E001        TABISH        32500    D010
E002        FATIMA        5440     D002
E006                D066
E004        AYSHA        6100     D004
E005        NAZMIN        10500    D005

```

9. Write a plsql block to FETCH more than one record and single column from a table

CODE:

```

1. DECLARE
2.   CURSOR c3 IS
3.     SELECT EMP_NAME FROM emp2;
4.   name emp2.EMP_NAME%TYPE;
5. BEGIN
6.   OPEN c3;
7.   LOOP
8.     FETCH c3 INTO name;
9.     EXIT WHEN c3%NOTFOUND;
10.    DBMS_OUTPUT.PUT_LINE(name);
11.  END LOOP;
12.  CLOSE c3;
13. END;
14. /

```

OUTPUT:

```

SQL> DECLARE
2   CURSOR c3 IS
3   SELECT EMP_NAME FROM emp2;
4   name emp2.EMP_NAME%TYPE;
5 BEGIN
6   OPEN c3;
7   LOOP
8     FETCH c3 INTO name;
9     EXIT WHEN c3%NOTFOUND;
10    DBMS_OUTPUT.PUT_LINE(name);
11  END LOOP;
12  CLOSE c3;
13 END;
14 /
TABISH
FATIMA
AYSHA
NAZMIN

```


10. Write a PL/SQL code to calculate the total and the percentage of marks of the students in four subjects from the table STUDENT with the schema given below.
STUDENT (RNO, S1, S2, S3, S4, total, percentage) The roll numbers and the marks in each subject are stored in the database. Also update your database

CODE:

```
1. DECLARE
2. T NUMBER;
3. PER NUMBER;
4. CURSOR C1 IS
5. SELECT * FROM student;
6. REC C1%ROWTYPE;
7. BEGIN
8. OPEN C1;
9. LOOP
10. FETCH C1 INTO REC;
11. EXIT WHEN C1%NOTFOUND;
12. T:=REC.s1+REC.s2+REC.s3+REC.s4;
13. PER:=T/4;
14. UPDATE student SET TOTAL = T,
15. PERCENTAGE = PER
16. WHERE rno = REC.rno;
17. END LOOP;
18. CLOSE C1;
19. END;
20. /
```

OUTPUT:

```
SQL> DECLARE
2      CURSOR c3 IS
3          SELECT EMP_NAME FROM emp2;
4      name emp2.EMP_NAME%TYPE;
5  BEGIN
6      OPEN c3;
7      LOOP
8          FETCH c3 INTO name;
9          EXIT WHEN c3%NOTFOUND;
10         DBMS_OUTPUT.PUT_LINE(name);
11     END LOOP;
12     CLOSE c3;
13 END;
14 /
TABISH
FATIMA
AYSHA
NAZMIN
```

Practical 06

AIM: To understand Exception Handling in PLSQL

THEORY: PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

- System-defined Exceptions
- User-defined Exceptions

Syntax for exception handling:

Following is a general syntax for exception handling:

1. **DECLARE**
2. <declarations **section**>
3. **BEGIN**
4. <executable command(s)>
5. **EXCEPTION**
6. <exception handling goes here >
7. **WHEN** exception1 **THEN**
8. exception1-handling-statements
9. **WHEN** exception2 **THEN**
10. exception2-handling-statements
11. **WHEN** exception3 **THEN**
12. exception3-handling-statements
13.
14. **WHEN** others **THEN**
15. exception3-handling-statements
16. **END;**

1. Write a PL/SQL block to handle the NO_DATA_FOUND exception, which occurs when a SELECT INTO statement returns no rows.

CODE:

1. DECLARE
2. vempname empexception.empname%TYPE;
3. BEGIN
4. SELECT empname INTO vempname FROM empexception
5. WHERE empno='E999';
6. dbms_output.put_line('Emp name is ' || vempname);
7. EXCEPTION
8. WHEN NO_DATA_FOUND THEN
9. dbms_output.put_line('Data not found, please enter a valid info');
10. END;
11. /

OUTPUT:

```

7  EXCEPTION
8  WHEN NO_DATA_FOUND THEN
9  dbms_output.put_line('Data not found, please enter a valid info');
10 END;
11 /
Data not found, please enter a valid info
PL/SQL procedure successfully completed.

```

2. What will happen if a number is divided by zero in the following PL/SQL block, and how does the block handle this exception?

CODE:

1. DECLARE
2. num NUMBER;
3. BEGIN
4. num:=756/0;
5. EXCEPTION
6. WHEN zero_divide THEN
7. dbms_output.put_line('Error! Cannot divide by ZERO');
8. END;

OUTPUT:

```

SQL> DECLARE
2  num NUMBER;
3  BEGIN
4  num:=756/0;
5  EXCEPTION
6  WHEN zero_divide THEN
7  dbms_output.put_line('Error! Cannot divide by ZERO');
8  END;
9  /
Error! Cannot divide by ZERO

```

3. Write a PL/SQL block to handle the TOO_MANY_ROWS exception, which occurs when a SELECT INTO statement returns more than one row.

CODE:

1. DECLARE
2. temp VARCHAR2(20);
3. BEGIN
4. SELECT empname INTO temp
5. FROM empexception;
6. dbms_output.put_line(temp);
7. EXCEPTION
8. WHEN too_many_rows THEN
9. dbms_output.put_line('Tooo many rows have been selected');
10. END;
11. /

OUTPUT:

```
4 SELECT empname INTO temp
5 FROM empexception;
6 dbms_output.put_line(temp);
7 EXCEPTION
8 WHEN too_many_rows THEN
9 dbms_output.put_line('Tooo many rows ha
10 END;
11 /
Tooo many rows have been selected
```

4. Write a PL/SQL block to handle a specific unnamed exception by assigning it a name using PRAGMA EXCEPTION_INIT

CODE:

1. DECLARE
2. child_record_exception EXCEPTION;
3. PRAGMA EXCEPTION_INIT(child_record_exception, -1517);
4. BEGIN
5. DELETE FROM empexception WHERE empname = 'Nazmin11';
6. EXCEPTION
7. WHEN child_record_exception THEN
8. dbms_output.put_line('CHILD R4CORDS FOR THIS DEPARTMENT EXIST IN EMPLOYEE TABLE');
9. END;

OUTPUT:

```

SQL> DECLARE
  2  child_record_exception EXCEPTION;
  3  PRAGMA EXCEPTION_INIT(child_record_exception
  4  BEGIN
  5  DELETE FROM empexception WHERE empname = 'Na
  6  EXCEPTION
  7  WHEN child_record_exception THEN
  8  dbms_output.put_line('CHILD R4ECORDS FOR THI
  9  END;
 10  /

PL/SQL procedure successfully completed.

```

5. Create a PLSQL procedure that raises a custom exception if employees salary is less than certain threshold.

CODE:

```

1. CREATE OR REPLACE PROCEDURE checksal(a IN NUMBER) IS
2.  b EXCEPTION;
3.  c employee.salary%TYPE;
4.  BEGIN
5.  SELECT salary INTO c FROM employee WHERE id = a;
6.  IF c < 3000 THEN
7.    RAISE b;
8.  END IF;
9.  DBMS_OUTPUT.PUT_LINE('Salary is sufficient. ');
10. EXCEPTION
11. WHEN b THEN
12.  DBMS_OUTPUT.PUT_LINE('Error: Salary is below the minimum threshold. ');
13. END checksal;
14. /
15. BEGIN
16. checksal(5);
17. END;
18. /

```

OUTPUT:

```

PL/SQL procedure successfully completed.

SQL> BEGIN
  2  checksal(5);
  3  END;
  4  /
Error: Salary is below the minimum threshold.

```

6. Write a PL/SQL block to handle the situation when no CASE statement matches any condition and there is no ELSE clause.

CODE:

```
1. DECLARE
2.   v_grade CHAR(1) := 'D';
3.   v_result VARCHAR2(20);
4. BEGIN
5.   CASE v_grade
6.     WHEN 'A' THEN v_result := 'Excellent';
7.     WHEN 'B' THEN v_result := 'Good';
8.     WHEN 'C' THEN v_result := 'Average';
9.   END CASE;
10.
11.  DBMS_OUTPUT.PUT_LINE('Result: ' || v_result);
12. EXCEPTION
13.  WHEN CASE_NOT_FOUND THEN
14.    DBMS_OUTPUT.PUT_LINE('No matching case found.');
```

OUTPUT:

```
9      END CASE;
10
11      DBMS_OUTPUT.PUT_LINE('Result:
12 EXCEPTION
13      WHEN CASE_NOT_FOUND THEN
14          DBMS_OUTPUT.PUT_LINE('No
15 END;
16 /
No matching case found.
```

7. Write a plsql code that raise an exception if employee phone number is NULL.

CODE:

```
1. CREATE OR REPLACE PROCEDURE new_emp(e_no IN VARCHAR2, e_phone IN
   NUMBER) IS
2.   no_phone EXCEPTION;
3. BEGIN
4.   IF e_phone IS NULL THEN
5.     RAISE no_phone;
6.   ELSE
7.     INSERT INTO emp5(emp_no, emp_phone) VALUES (e_no, e_phone);
8.   END IF;
9. EXCEPTION
10.  WHEN DUP_VAL_ON_INDEX THEN
```

```

11.     raise_application_error(-20001, 'Employee number cannot be duplicated. ');
12. WHEN no_phone THEN
13.     raise_application_error(-20002, 'Phone number cannot be left blank. ');
14. WHEN OTHERS THEN
15.     raise_application_error(-20003, 'Error while inserting the employee record. ');
16. END;
17. BEGIN
18. new_emp('01', NULL);
19. END;
20. /

```

OUTPUT:

```

SQL> BEGIN
      2  new_emp('01', NULL);
      3  END;
      4  /
BEGIN
*
ERROR at line 1:
ORA-20002: Phone number cannot be left blank.
ORA-06512: at "SYSTEM.NEW_EMP", line 18

```

8. Write a PLSQL code to raise an user defined exception if denominator is less than numerator or zero.

CODE:

```

1. DECLARE
2.     num1    int := &num1;
3.     num2    int := &num2;
4.     result  float;
5.     divByZero EXCEPTION;
6.     num2Larger EXCEPTION;
7. BEGIN
8.     IF num2 = 0 THEN
9.         RAISE divByZero;
10.    ELSIF num2 > num1 THEN
11.        RAISE num2Larger;
12.    ELSE
13.        result := num1 / num2;
14.        dbms_output.put_line('The result is: ' || result);
15.    END IF;
16. EXCEPTION
17.    WHEN divByZero THEN
18.        dbms_output.put_line('Error');
19.        dbms_output.put_line('Division by zero is not allowed. ');
20.    WHEN num2Larger THEN
21.        dbms_output.put_line('Error');

```

```

22.     dbms_output.put_line('num2 is greater than num1, please check the input. ');
23. END;
24. /

```

OUTPUT:

```

26         dbms_output.put_line('Error');
27         dbms_output.put_line('num2 is greater than num1, please check the input. ');
28     END;
29     /
Enter value for num1: 8
old   2:      num1      int := &num1;
new   2:      num1      int := 8;
Enter value for num2: 0
old   3:      num2      int := &num2;
new   3:      num2      int := 0;
Error
Division by zero is not allowed.

```

9. Write a PLSQL code for NO_DATA_FOUND.

CODE:

```

1. DECLARE
2.     courseName varchar(20);
3. BEGIN
4.     SELECT course_id
5.     INTO courseName
6.     FROM course_info
7.     WHERE course_name = 'Advanced_Database_Management_System';
8. EXCEPTION
9.     WHEN no_data_found THEN
10.        dbms_output.put_line('ERROR');
11.        dbms_output.put_line('There is no course with that name. ');
12.        dbms_output.put_line('Advanced_Database_Management_System not found in
        course_info table. ');
13. END;

```

OUTPUT:

```

11         WHEN no_data_found THEN
12             dbms_output.put_line('ERROR');
13             dbms_output.put_line('There is no course with that name. ');
14             dbms_output.put_line('Advanced_Database_Management_System not found in
        course_info table. ');
15     END;
16     /
ERROR
There is no course with that name.
Advanced_Database_Management_System not found in course_info table.

```


10. Write a PL/SQL block to handle the VALUE_ERROR exception, which occurs when there is an error in converting a data type.

CODE:

```
1. DECLARE
2.   v_number NUMBER;
3. BEGIN
4.   v_number := TO_NUMBER('abc');
5. EXCEPTION
6.   WHEN VALUE_ERROR THEN
7.     DBMS_OUTPUT.PUT_LINE('Error in data type conversion. ');
8. END;
9. /
```

OUTPUT:

```
5  EXCEPTION
6    WHEN VALUE_ERROR THEN
7      DBMS_OUTPUT.PUT_LINE('Error
8  END;
9  /
Error in data type conversion.
```

Practical 07

AIM: To understand Packages in PLSQL.

THEORY: PL/SQL packages are a way to organize and encapsulate related **procedures, functions, variables, triggers**, and other PL/SQL items into a single item. Packages provide a modular approach to write and maintain the code. It makes it easy to manage large codes.

A package is compiled and then stored in the database, which then can be shared with many applications. The package also has specifications, which declare an item to be public or private. Public items can be referenced from outside of the package.

A PL/SQL package is a collection of related **Procedures, Functions, Variables**, and other elements that are grouped for **Modularity** and **Reusability**.

SYNTAX:

```
CREATE OR REPLACE PACKAGE package_name IS
```

```
-- Public declarations
```

```
    FUNCTION function_name(parameter1 datatype, parameter2 datatype) RETURN  
    return_datatype;
```

```
    PROCEDURE procedure_name(parameter1 datatype);
```

```
-- Optionally, you can declare variables or types
```

```
END package_name;
```

```
/
```

```
CREATE OR REPLACE PACKAGE BODY package_name IS
```

```
-- Private declarations (optional)
```

```
    FUNCTION function_name(parameter1 datatype, parameter2 datatype) RETURN  
    return_datatype IS
```

```
    BEGIN
```

```
        RETURN result;
```

```
    END function_name;
```

```
    PROCEDURE procedure_name(parameter1 datatype) IS
```

```
    BEGIN
```

```
        END procedure_name;
```

```
END package_name;
```

1. Create a Package for Calculating Rectangle Area and Perimeter.

CODE:

```
1. CREATE OR REPLACE PACKAGE rectpackage AS
2.     PROCEDURE calculaterect (len IN NUMBER, wid IN NUMBER, area OUT
   NUMBER, perimeter OUT NUMBER);
3. END rectpackage;
4. CREATE OR REPLACE PACKAGE BODY rectpackage AS
5.     PROCEDURE calculaterect (len IN NUMBER, wid IN NUMBER, area OUT
   NUMBER, perimeter OUT NUMBER) IS
6. BEGIN
7. area:= len* wid;
8. perimeter:=2*(len+wid);
9. END calculaterect;
10. END rectpackage;
11. DECLARE
12. length NUMBER;
13. width NUMBER;
14. result1 NUMBER;
15. result2 NUMBER;
16. BEGIN
17. length:=&length;
18. width:=&width;
19. rectpackage.calculaterect(length,width,result1,result2);
20. dbms_output.put_line('The area is '||result1||' and perimeter is: '||result2);
21. END;
22. /
```

OUTPUT:

```
12 /
Enter value for length: 5
old 7: length:=&length;
new 7: length:=5;
Enter value for width: 7
old 8: width:=&width;
new 8: width:=7;
The area is 35 and perimeter is: 24

PL/SQL procedure successfully completed.
```

2. Temperature conversion package in PLSQL.

CODE:

1. CREATE OR REPLACE PACKAGE tempconversion AS
2. FUNCTION celtofer(cel IN NUMBER) RETURN NUMBER;
3. FUNCTION fertocel(fer IN NUMBER) RETURN NUMBER;
4. END tempconversion;
5. CREATE OR REPLACE PACKAGE BODY tempconversion AS
6. FUNCTION celtofer(cel IN NUMBER) RETURN NUMBER IS
7. BEGIN
8. RETURN (cel*9/5)+32;
9. END celtofer;
10. FUNCTION fertocel(fer IN NUMBER) RETURN NUMBER IS
11. BEGIN
12. RETURN (fer-32)*5/9;
13. END fertocel;
14. END tempconversion;
15. DECLARE
16. celtemp NUMBER:=&celtemp;
17. ferhenittemp NUMBER;
18. celciustemp NUMBER;
19. BEGIN
20. ferhenittemp:= tempconversion.celtofer(celtemp);
21. dbms_output.put_line('The value for ferhenit is: '||ferhenittemp);
22. celciustemp:=tempconversion.fertocel(ferhenittemp);
23. dbms_output.put_line('The value for converted celcius is: '||celciustemp);
24. END;
25. /

OUTPUT:

```

11 /
Enter value for celtemp: 45
old 2: celtemp NUMBER:=&celtemp;
new 2: celtemp NUMBER:=45;
The value for ferhenit is: 113
The value for converted celcius is: 45

```

3. Create a package named cust_sal that includes a procedure find_sal. The procedure should take a customer ID as input and print the salary of the customer with that ID.

CODE:

1. create or replace package cust_sal AS
2. procedure find_sal(id customer.cust_id%type);
3. end cust_sal;
4. create or replace package body cust_sal AS
5. procedure find_sal(id customer.cust_id%type) IS
6. salary customer.cust_salary%type;
7. begin
8. select cust_salary into salary from customer
9. where cust_id=id;
10. dbms_output.put_line('the salary is:'||salary);

```

11. end find_sal;
12. end cust_sal;
13. declare
14. code customer.cust_id%type:=&code;
15. begin
16. cust_sal.find_sal(code);
17. end;
18. /

```

OUTPUT:

```

5 end,
6 /
Enter value for code: 2
old 2: code customer.cust_id%type:=&code
new 2: code customer.cust_id%type:=2;
the salary is:150

```

4. Create a PL/SQL package to calculate the compound interest for a given principal, rate, and time.

CODE:

```

1. CREATE OR REPLACE PACKAGE interest_calculator IS
2.   FUNCTION calculate_compound_interest(principal NUMBER, rate NUMBER, time
   NUMBER) RETURN NUMBER;
3. END interest_calculator;
4. CREATE OR REPLACE PACKAGE BODY interest_calculator IS
5. FUNCTION calculate_compound_interest(principal NUMBER, rate NUMBER, time NUMBER)
   RETURN NUMBER IS
6. BEGIN
7. RETURN principal * POWER((1 + rate / 100), time);
8. END calculate_compound_interest;
9. END interest_calculator;
10. /
11. DECLARE
12.   v_principal NUMBER := 1000;
13.   v_rate NUMBER := 5;
14.   v_time NUMBER := 10;
15.   v_result NUMBER;
16. BEGIN
17.   v_result := interest_calculator.calculate_compound_interest(v_principal, v_rate, v_time);
18.   DBMS_OUTPUT.PUT_LINE('Compound Interest: ' || v_result);
19. END;
20. /

```

OUTPUT:

```
SQL> DECLARE
  2     v_principal NUMBER := 1000;
  3     v_rate NUMBER := 5;
  4     v_time NUMBER := 10;
  5     v_result NUMBER;
  6 BEGIN
  7     v_result := interest_calculator.calculat
  8     DBMS_OUTPUT.PUT_LINE('Compound Interest:
  9 END;
 10 /
Compound Interest: 1628.89462677744140625

PL/SQL procedure successfully completed.
```

5. Create a PL/SQL package to count the number of vowels in a string.

CODE:

```
1. CREATE OR REPLACE PACKAGE vowel_counter IS
2.     FUNCTION count_vowels(input_str VARCHAR2) RETURN NUMBER;
3. END vowel_counter;
4. /
5. CREATE OR REPLACE PACKAGE BODY vowel_counter IS
6.     FUNCTION count_vowels(input_str VARCHAR2) RETURN NUMBER IS
7.         vowel_count NUMBER := 0;
8.     BEGIN
9.         FOR i IN 1..LENGTH(input_str) LOOP
10.            IF SUBSTR(UPPER(input_str), i, 1) IN ('A', 'E', 'I', 'O', 'U') THEN
11.                vowel_count := vowel_count + 1;
12.            END IF;
13.        END LOOP;
14.        RETURN vowel_count;
15.    END count_vowels;
16. END vowel_counter;
17. /
18. DECLARE
19.     v_result NUMBER;
20. BEGIN
21.     v_result := vowel_counter.count_vowels('Birla college is in kalyan');
22.     DBMS_OUTPUT.PUT_LINE('Number of vowels: ' || v_result);
23. END;
24. /
```

OUTPUT:

```
Package body created.

SQL> DECLARE
  2     v_result NUMBER;
  3 BEGIN
  4     v_result := vowel_counter.count_vowels('Birla college is in kalyan');
  5     DBMS_OUTPUT.PUT_LINE('Number of vowels: ' || v_result);
  6 END;
  7 /
Number of vowels: 9
```

Practical 08

AIM: To understand Triggers in PL SQL.

THEORY: Trigger is invoked by Oracle engine automatically whenever a specified event occurs. Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are stored programs, which are automatically executed or fired when some event occurs.

Triggers are written to be executed in response to any of the following events.

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATE, ALTER, or DROP).
- A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Syntax for creating trigger:

1. **CREATE** [OR REPLACE] **TRIGGER** trigger_name
2. { **BEFORE** | **AFTER** | **INSTEAD OF** }
3. { **INSERT** [OR] | **UPDATE** [OR] | **DELETE** }
4. [**OF** col_name]
5. **ON** table_name
6. [**REFERENCING** OLD **AS** o NEW **AS** n]
7. [**FOR EACH ROW**]
8. **WHEN** (condition)
9. **DECLARE**
10. Declaration-statements
11. **BEGIN**
12. Executable-statements
13. **EXCEPTION**
14. Exception-handling-statements
15. **END;**

1. What will be inserted into the Affect table when the following statement is executed:
INSERT INTO Adbms (Id, Name, Score) VALUES (6, 'Arjun', 500); using trigger?

CODE:

1. CREATE OR REPLACE TRIGGER BEFORE_INSERT
2. BEFORE INSERT ON Adbms
3. FOR EACH ROW
4. BEGIN
5. INSERT INTO Affect (Id, Name, Score)
6. VALUES (:NEW.Id, :NEW.Name, :NEW.Score);
7. END;
8. /

OUTPUT:

```
8 /
Trigger created.
SQL> INSERT INTO Adbms (Id, Name, Score) VALUES (6, 'XYZ', 500);
1 row created.
SQL> Select * from Affect;
      ID NAME                SCORE
-----
6 XYZ                500
```

2. For the BEFORE_DELETE trigger, what will be inserted into the Affect table when the following statement is executed: DELETE FROM Adbms WHERE Id = 3;?

CODE:

1. CREATE OR REPLACE TRIGGER BEFORE_DELETE
2. BEFORE DELETE ON Adbms
3. FOR EACH ROW
4. BEGIN
5. INSERT INTO Affect (Id, Name, Score)
6. VALUES (:OLD.Id, :OLD.Name, :OLD.Score);
7. END;
8. /
9. DELETE FROM Adbms WHERE Id = 3;

OUTPUT:

```
1 row deleted.
SQL> SELECT * FROM AFFECT;
      ID NAME                SCORE
-----
TIME
-----
      6 XYZ                  500
      3 Ayesha              150
```

3. For the BEFORE_UPDATE trigger, what will be inserted into the Affect table when the following statement is executed: UPDATE Adbms SET Score = 900 WHERE Id = 5;?

CODE:

1. CREATE OR REPLACE TRIGGER BEFORE_UPDATE
2. BEFORE UPDATE ON Adbms
3. FOR EACH ROW
4. BEGIN
5. INSERT INTO Affect (Id, Name, Score)
6. VALUES (:OLD.Id, :OLD.Name, :OLD.Score);
7. END;
8. /

OUTPUT:

```
Trigger created.
SQL> UPDATE Adbms SET Score = 900 WHERE Id = 5;
1 row updated.
SQL> SELECT * FROM Affect;
      ID NAME                SCORE
-----
TIME
-----
      6 XYZ                  500
      3 Ayesha              150
      5 Aziaf                550

SQL> SELECT * FROM Adbms;
      ID NAME                SCORE
-----
      1 Tabish                750
      2 Fatima                668
      5 Aziaf                900
      6 XYZ                  500
```

4. Explain the functionality of the AFTER_DELETE trigger in the given PL/SQL code and describe what happens when a row is deleted from the Practical table, particularly focusing on how the trigger affects the Affect table.

CODE;

1. CREATE OR REPLACE TRIGGER AFTER_DELETE
2. AFTER DELETE ON Practical
3. FOR EACH ROW
4. BEGIN
5. INSERT INTO Affect (Id, Name, Score)
6. VALUES (:OLD.Id, :OLD.Name, :OLD.Score);
7. END;
8. /

OUTPUT:

```
SQL> DELETE FROM Adbms WHERE Id = 5;

1 row deleted.

SQL> SELECT * FROM affect;

      ID NAME                SCORE
-----
      5 Aziaf                900
```

5. How does the after_change_users trigger ensure that changes made to the Users table are logged in the AuditLog table, and what information is captured for each type of operation (INSERT, UPDATE, DELETE)?

CODE:

1. CREATE OR REPLACE TRIGGER after_change_users
2. AFTER INSERT OR UPDATE OR DELETE ON Users
3. FOR EACH ROW
4. BEGIN
5. IF INSERTING THEN
6. INSERT INTO AuditLog (UserId, Operation, Timestamp, NewUsername, NewEmail)
7. VALUES (:NEW.UserId, 'INSERT', SYSDATE, :NEW.Username, :NEW.Email);
- 8.
9. ELSIF UPDATING THEN
10. INSERT INTO AuditLog (UserId, Operation, Timestamp, OldUsername, OldEmail, NewUsername, NewEmail)
11. VALUES (:OLD.UserId, 'UPDATE', SYSDATE, :OLD.Username, :OLD.Email, :NEW.Username, :NEW.Email);
- 12.

```

13.  ELSIF DELETING THEN
14.      INSERT INTO AuditLog (UserId, Operation, Timestamp, OldUsername, OldEmail)
15.      VALUES (:OLD.UserId, 'DELETE', SYSDATE, :OLD.Username, :OLD.Email);
16.  END IF;
17. END;
18. /

```

OUTPUT:

```
SQL> /
```

USERID	OPERATION	TIMESTAMP	OLDUSERNAM	OLDEMAIL	NEWUSERNAM	NEWEMAIL
1	INSERT	25-SEP-24			Tabish	tabish@example.com
3	INSERT	25-SEP-24			Fatima	Fatima@example.com
1	UPDATE	25-SEP-24	Tabish	tabish@example.com	MohdTabish	mohdTabish@example.com
1	DELETE	25-SEP-24	MohdTabish	mohdTabish		
				@example.com		

6. Craete a Trigger for user name with same id as in original table.

CODE:

```

1. CREATE OR REPLACE TRIGGER welcome
2.  AFTER INSERT ON Adbms
3.  FOR EACH ROW
4.  BEGIN
5.  INSERT INTO Affect (Id, "time", Name)
6.  VALUES (:NEW.Id, SYSDATE, USER);
7.  END;
8.  /
9. insert into Adbms (Id, Name, Score) VALUES (6, 'aziaf', 500);

```

OUTPUT:

```
SQL> /
```

ID	NAME	SCORE	time
6	SYSTEM		25-SEP-24 11.05.30.000000 PM

7. Create a PL/SQL trigger to store date when values is inserted in table.

CODE:

1. CREATE OR REPLACE TRIGGER before_insert_users
2. BEFORE INSERT ON Users
3. FOR EACH ROW
4. BEGIN
5. :NEW.CreatedDate := SYSDATE;
6. :NEW.UpdatedDate := SYSDATE;
7. END;
8. /

OUTPUT:

```
Trigger created.
```

```
SQL> INSERT INTO users (Id, Name, Score)
2  VALUES (1, 'Tabish', 100);
```

```
1 row created.
```

```
SQL> INSERT INTO users (Id, Name, Score)
2  VALUES (2, 'Fatima', 200);
```

```
1 row created.
```

```
SQL> select * from users;
```

ID	NAME	SCORE	CREATEDDA	UPDATEDDA
1	Tabish	100	26-SEP-24	26-SEP-24
2	Fatima	200	26-SEP-24	26-SEP-24

```
SQL> |
```