CHAPTER: 5 PL/SQL, DATABASE OBJECTS & SECURITY (24 Marks)

5.1 What is PL/SQL?

- **PL/SQL** stands for **Procedural Language** extension of SQL.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- Oracle uses a **PL/SQL** engine to processes the PL/SQL statements. A PL/SQL language code can be stored in the client system (client-side) or in the database (server-side).

5.1.1 Advantages of PL/SQL

- ***Block Structures:*** PL SQL consists of blocks of code, which can be nested within each other. Each block forms a unit of a task or a logical module. PL/SQL Blocks can be stored in the database and reused.
- ***Procedural Language Capability*:** PL SQL consists of procedural language constructs such as conditional statements (if else statements) and loops like (FOR loops).
- ***Better Performance*:** PL SQL engine processes multiple SQL statements simultaneously as a single block, thereby reducing network traffic.
- ***Error Handling*:** PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program. Once an exception is caught, specific actions can be taken depending upon the type of the exception or it can be displayed to the user with a message.

5.1.2 PL/SQL Block Structures

- PL/SQL Block consists of three sections:
  1. The Declaration section (optional).
  2. The Execution section (mandatory).
  3. The Exception Handling (or Error) section (optional).
- **How a Sample PL/SQL Block Looks**

```
DECLARE
    Variable declaration
BEGIN
    Program Execution
EXCEPTION
    Exception handling
END;
```

1. **Declaration Section:**

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

2. **Execution Section:**

- The Execution section of a PL/SQL Block starts with the reserved keyword BEGIN and ends with END.
- This is a mandatory section and is the section where the program logic is written to perform any task.
- The programmatic constructs like loops, conditional statement and SQL statements form the part of execution section.

3. **Exception Section:**

- The Exception section of a PL/SQL Block starts with the reserved keyword EXCEPTION.
- This section is optional.
- Any errors in the program can be handled in this section, so that the PL/SQL Blocks terminates gracefully.

Note:- Every statement in the above three sections must end with a semicolon ;

5.1.3 PL/SQL identifiers

- PL/SQL identifiers are constants, variables, exceptions, procedures, cursers, & reserved words.

5.1.4 PL/SQL Comments

- The PL/SQL supports single line & multi-line comments.
- ***Single line comments*** start with delimiter
  -- (double hyphen)
- ***Multi-line comments*** are enclosed by /*____*/.

5.1.5 PL/SQL Data types

Category	*Data Type*
Number	**INT** ANSI specific integer type with maximum precision of 38 decimal digits
	FLOAT ANSI and IBM specific floating-point type with maximum precision of 126 binary digits (approximately 38 decimal digits)

	NUMERIC(pre, scale) Floating type with maximum precision of 38 decimal digits
	NUMBER(prec., scale) Fixed-point or floating-point number with absolute value in range 1E-130 to (but not including) 1.0E126. A NUMBER variable can also represent 0
	DECIMAL(prec., scale) IBM specific fixed-point type with maximum precision of 38 decimal digits
Character	**CHAR** Fixed-length character string with maximum size of 32,767 bytes. 255 character
	VARCHAR/VARCHAR2 Variable-length character string with maximum size of 32,767 bytes. 4000 char.
	RAW This data type is used to store binary data such as digitized, picture or image.
	LONG Variable-length character string with maximum size of 32,760 bytes
	LONG RAW Variable-length binary or byte string with maximum size of 32,760 bytes, not interpreted by PL/SQL
Character	**NCHAR** Fixed-length national character string with maximum size of 32,767 bytes
Boolean	**Boolean** Logical values on which logical operations are performed.
Date-time	**Date()** Dates and times. Standard format is DD-month-YY.
Large Object (LOB)	**BLOB** This data type is used to store binary character like such as text, graphic images, video clips, and sound waveforms. And hold up to 4GB.
	CLOB This data type is used to store character variable & hold upto 4GB.

5.1.6 PL/SQL Variables

- General **Syntax** to declare a variable is

  Variable_name datatype [NOT NULL: = value];

- **For example:**
  1. dept varchar2(10) NOT NULL := “HR Dept”;
  2. Sal number:=10000;

5.1.7 PL/SQL Constants

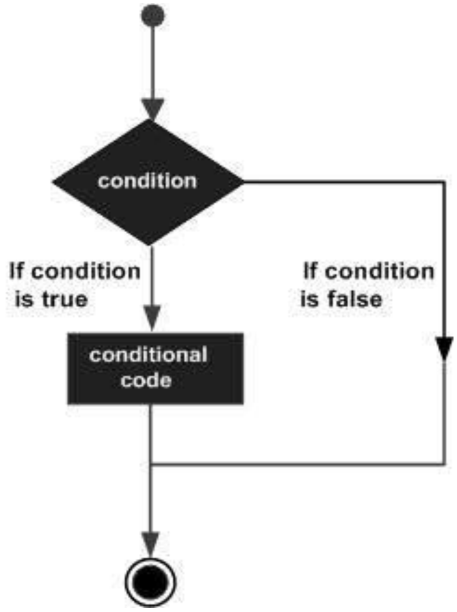
- A *constant* is a value used in a PL/SQL Block that remains unchanged throughout the program.
- General **Syntax** to declare a constant is:
  *Constant_name **CONSTANT** datatype: = VALUE;*
- **For example**:- *pi **CONSTANT** number:=3.14;*

How to display messages on screen?

- **DBMS_OUTPUT**: Is package that include a number of procedure & functions that accumulate information in a buffer so that it can be retrieved later. These functions can also be used to display messages to the user.
- **PUT_LINE**:- put a piece of information in the package buffer followed by an end-of-line marker. It can also be used to display messages to the user. If used to display a message, it is the message 'String'.
- For example: **dbms_output.put_line (x);**
- To display messages to the user the **SERVEROUTPUT** should be set to **ON.**
- For example: **SET SERVEROUTPUT ON.**

5.2 Conditional Statements in PL/SQL

- Decision-making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.
- The programming constructs are similar to how you use in programming languages like Java and C++.
- Following is the general form of a typical conditional (i.e., decision making) structure found in most of the programming languages −



- **Conditional Control: -** PL/SQL programming language provides following types of decision-making statements **i.e. IF-THEN, IF-THEN-ELSE, IF-THEN-ELSEIF.**

1. IF-THEN Statement:-

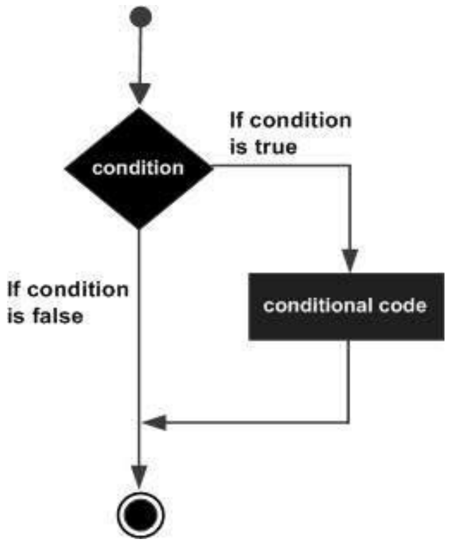
- The IF statement associates a condition with a sequence of statements enclosed by the **keywords THEN and END IF**.
- If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.
- **Syntax for IF-THEN statement**

```
IF <condition> THEN
Statement1;
END IF;
```

- **For Example:-**

```
IF (a <= 20) THEN
c:= c+1;
END IF;
```

- **Flow Diagram**



Sample Example 1:-

```
SET SERVEROUTPUT ON;
DECLARE
a number(2) := 10;
BEGIN
a:= 10;
            - - check the Boolean condition using if
statement
IF ( a < 20 ) THEN
            -- if condition is true then print the following
dbms_output.put_line('a is less than 20 ' );
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```

2. IF-THEN-ELSE Statement:-

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

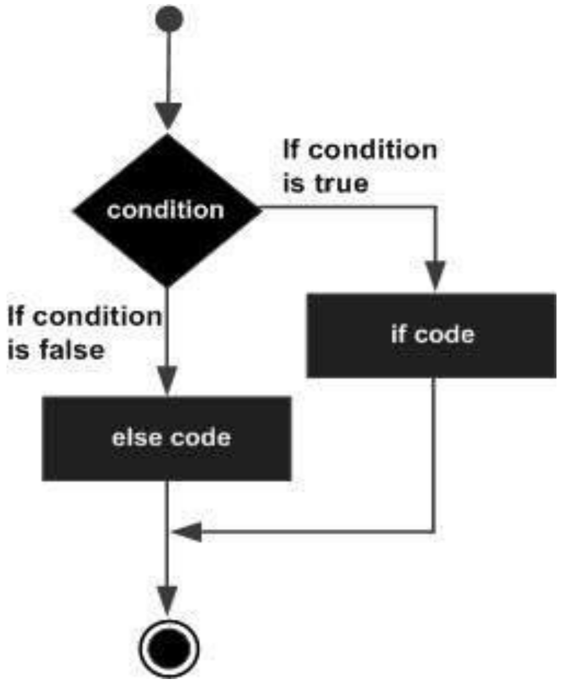
- Syntax for **IF-THEN-ELSE** statement

```
IF <condition> THEN
Statement1;
ELSE
Statement2;
END IF;
```

- **For Example:-**

```
IF color = red THEN
dbms_output.put_line (‘you have chosen a red car');
ELSE
dbms_output.put_line (‘please choose a color for your car');
END IF;
```

- **Flow Diagram**



Sample Example 1:-

```
SET SERVEROUTPUT ON;
DECLARE
a number(3) := 100;
BEGIN
        -- check the boolean condition using if statement
IF( a < 20 ) THEN
         -- if condition is true then print the following
dbms_output.put_line('a is less than 20 ' );
ELSE
dbms_output.put_line('a is not less than 20 ' );
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```

Output:-

```
a is not less than 20
value of a is : 100
PL/SQL procedure successfully completed.
```

3. IF-THEN-ELSEIF Statement:-

- The **IF-THEN-ELSIF** statement allows you to choose between several alternatives. An **IF-THEN** statement can be followed by an optional **ELSIF...ELSE** statement. The **ELSIF** clause lets you add additional conditions.
- Syntax for **IF-THEN-ELSIF** statement

```
IF (boolean_expression 1) THEN
      S1; - - Executes when the Boolean expression 1 is true
ELSIF (boolean_expression 2) THEN
       S2; - - Executes when the Boolean expression 2 is true
ELSIF (boolean_expression 3) THEN
       S3; - - Executes when the Boolean expression 3 is true
ELSE
       S4; -- executes when the none of the above condition is true
END IF;
```

- **For Example 1**

Sample Example 1:-

```
SET SERVEROUTPUT ON;
DECLARE
a number(3) := 100;
BEGIN
IF ( a = 10 ) THEN
dbms_output.put_line('Value of a is 10' );
ELSIF ( a = 20 ) THEN
dbms_output.put_line('Value of a is 20' );
ELSIF ( a = 30 ) THEN
dbms_output.put_line('Value of a is 30' );
ELSE
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: '|| a );
END;
/
```

```
-- OUTPUT
None of the values is matching
Exact value of a is: 100
PL/SQL procedure successfully completed.
```

Sample Program:-

1. Write PL/SQL program to find largest number of two numbers

Sample Example 1:-

```
SET SERVEROUTPUT ON;
DECLARE
a number(3);
b number(3);
BEGIN
dbms_output.put_line(Enter value of A:-' );
a:=&a;
dbms_output.put_line(Enter value of B:-' );
b:=&b;
IF (a>b) THEN
dbms_output.put_line(‘ A is Largest' );
ELSE
dbms_output.put_line(‘ B is Largest' );
END IF;
END;
/
```

2. Write PL/SQL program to find maximum number of three numbers

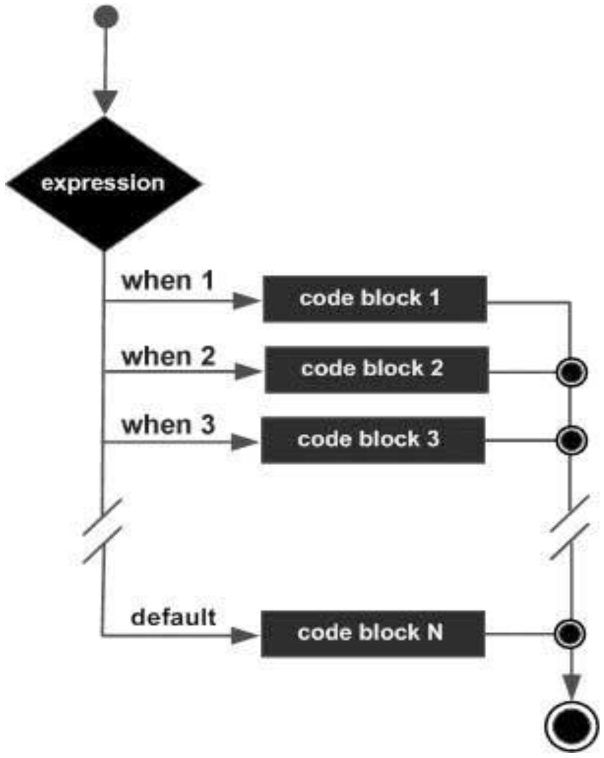
```
DECLARE
      a number(3);
      b number(3);
      c number(3);
BEGIN
      dbms_output.put_line(Enter value of A:-' );
      a:=&a;
      dbms_output.put_line(Enter value of B:-' );
      b:=&b;
      dbms_output.put_line(Enter value of C:-' );
      c:=&c;
IF (a>b) and (a>c) THEN
      dbms_output.put_line(‘ A is maximum' );
ELSIF ( b>a) and (b>c) THEN
      dbms_output.put_line(‘ B is maximum' );
ELSE
      dbms_output.put_line(‘ C is maximum' );
END IF;
END;
/
```

CASE statement: -

- The CASE statement selects one sequence of statements to execute.
- Syntax

```
CASE [expression]
WHEN condition_1 THEN result_1;

WHEN condition_2 THEN result_2;

WHEN condition_n THEN result_N;

……………

 ELSE result     - - default case

 END CASE;
```

- CASE statement Example
- Flow Diagram

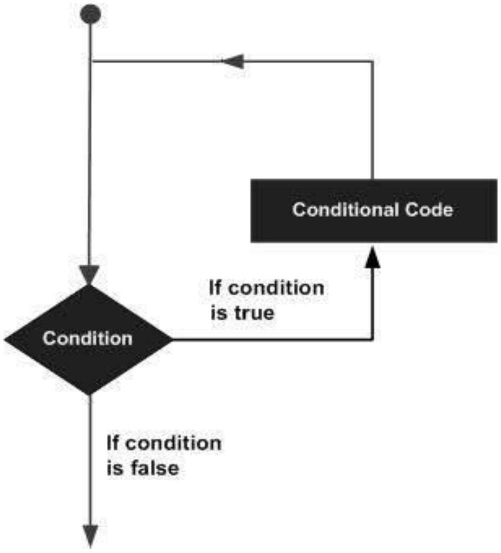


Sample Example 1:-

```
SET SERVEROUTPUT ON;
DECLARE
grade char(1) := 'A';
BEGIN
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('Well done');
when 'D' then
dbms_output.put_line('You passed');
when 'F' then
dbms_output.put_line('Better try again');
else
dbms_output.put_line('No such grade');
END CASE;
END;
 /
```

5.3 Iterative Control

- A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages −



1. Basic or simple LOOP: -

- Basic loop structure encloses sequence of statements in between the **LOOP** and **END LOOP** statements.
- Syntax:-

```
LOOP
    Sequence of statements;
END LOOP;
```

- Example

Sample Example

```
SET SERVEROUTPUT ON;
DECLARE
        x  number := 10;
BEGIN
LOOP
        dbms_output.put_line(x);
        x := x + 10;
IF x > 50 THEN
        exit;
END IF;
END LOOP;     - - after exit, control resumes here
dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

Output:-

```
10
20
30
40
50
After Exit x is: 60
PL/SQL procedure successfully completed
```

2. While Loop: -

- A **WHILE LOOP** statement in PL/SQL programming language repeatedly executes a target statement as long as a given condition is true.
- Syntax

```
WHILE condition LOOP
        sequence_of_statements;
 END LOOP;
```

- Example

Sample Example

```
SET SERVEROUTPUT ON;
DECLARE
      a number(2) := 10;
BEGIN
WHILE a < 20 LOOP
dbms_output.put_line('value of a: ' || a);
a := a + 1;
END LOOP;
END;
/
```

Output:-

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
PL/SQL procedure successfully completed.
```

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

- Syntax

```
FOR counter IN initial_value. . final_value LOOP
sequence_of_statements;
END LOOP;
```

Sample Example

```
SET SERVEROUTPUT ON;
DECLARE
       a number(2);
BEGIN
FOR a IN 10 . . 20 LOOP
     dbms_output.put_line('value of a: ' || a);
END LOOP;
END;
/
```

Output:-

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20
PL/SQL procedure successfully completed.
```

Sample Program:-

1. **Write a PL/SQL program to print even or odd number from given range (Accept number range from user).**

Ans: **- NOTE: -** In above program it specified that given range so that we used FOR LOOP

PL/SQL code to display **EVEN numbers**

```
SET SERVEROUTPUT ON;

DECLARE
        A Number;
        B Number;
BEGIN
a: =&A;
b: =&B;
        FOR i IN a . . b LOOP
        IF (mod (i, 2):=0) THEN
        Dbms_output.Put_line (i);
        END If;
        END Loop;
END;
```

PL/SQL code to display **ODD numbers**

```
SET SERVEROUTPUT ON;

DECLARE
        A Number;
        B Number;
BEGIN
a: =&A;
b: =&B;
        FOR i IN a . . b LOOP
        IF (mod (i, 2):=1) THEN
        Dbms_output.Put_line (i);
        END If;
        END Loop;
END;
```

2. Write PL/SQL program to display factorial of any number.

```
SET SERVEROUTPUT ON;

DECLARE
    f number: =1;
    n number := &n;
    i number;
BEGIN
          FOR i IN 1..n LOOP
          f := f * i;
          END LOOP;
          dbms_output.put_line (f);
END;
/
```

3. Write a PL/SQL program to find the square of a number given by user using WHILE….LOOP. (accept the number from user dynamically)

```
SET SERVEROUTPUT ON;

DECLARE
          n number:= &n;
          sqr number: = 0;
          n_cntr number: =0;
BEGIN
          Dbms_Output.Put_Line (N);
          WHILE n_cntr < n LOOP
          sqr: = sqr + n;
          n_cntr:= n_cntr + 1;
END LOOP;
          Dbms_Output.Put_Line (‘square of ' || n || ' is ' || sqr);
END;
/
```

5.4 Exception Handling

Errors:-

- Two types of errors can be found in a program: compilation errors and runtime errors.
- There is a special section in a PL/SQL block that handles the runtime errors.
- This section is called the *exception-handling section*, and in it, runtime errors are referred to as *exceptions*.
- The exception-handling section allows programmers to specify what actions should be taken when a specific exception occurs.

Exception:-

- In order to handle run time errors in the program, an exception handler must be added.
- Exception handling is nothing but a code block in memory that will attempt to resolve current error condition.
- The **exception-handling section has the following structure**:

```
EXCEPTION
    WHEN Exception_name THEN
    Error-processing Statements;
```

- **Note: -** The exception-handling section is placed after the executable section of the block.

- **Exception Handling Structure**

```
DECLARE
        --Declaration section
BEGIN
        -- Program section
--Exception section
EXCEPTION

    WHEN ex_name1 THEN

        --Error handling statements

    WHEN ex_name2 THEN
        --Error handling statements
END;
```

- **Exception Handling Example:**

```
DECLARE
  num1 number := &sv_num1;
  num2 number := &sv_num2;
  result number;
BEGIN
  result := num1 / num2;
  DBMS_OUTPUT.PUT_LINE ( ‘the is result: ’|| result);
EXCEPTION
  WHEN ZERO_DIVIDE
  THEN
    DBMS_OUTPUT.PUT_LINE (‘A number cannot be divided
by zero.’);
END;
/
```

Output:-

```
Enter value for sv_num1: 4
old 2: v_num1 integer := &sv_num1;
new 2: v_num1 integer := 4;

Enter value for sv_num2: 0
old 3: v_num2 integer := &sv_num2;
new 3: v_num2 integer := 0;

A number cannot be divided by zero.

PL/SQL procedure successfully completed.
```

5.4.1 Types of Exception

- There are 3 types of Exceptions.
  a) Predefined exceptions or Named System Exceptions
  b) User-defined Exceptions
  c) Raising Exceptions

a) Predefined Exceptions:-

- System exceptions are automatically raised by Oracle, when a program violates a RDBMS rule.
- There are some system exceptions which are raised frequently, so they are pre-defined and given a name in Oracle which are known as Named System Exceptions.
- **For example: NO_DATA_FOUND** and **ZERO_DIVIDE** are called Named System exceptions.
- Named system exceptions are:
  1) not declared explicitly,
  2) Raised implicitly when a predefined Oracle error occurs.
  3) Caught by referencing the standard name within an exception-handling routine.

Exception Name	Reason
CURSOR_ALREADY_OPEN	When you open a cursor that is already open.
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.

TOO_MANY_ROWS	When you SELECT or fetch more than one row into a record or variable.
ZERO_DIVIDE	When you attempt to divide a number by zero.

- **Predefined Exceptions Example**

For Example: Suppose a NO_DATA_FOUND exception is raised in a proc. we can write a code to handle the exception as given below.

```
DECLARE
        - -
BEGIN
      - -
Execution section
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line ('A SELECT...INTO did not return any row.');
END;
```

b) User-defined Exceptions

- This type of an exception is called a *user-defined exception* because it is defined by the programmer.
- Before the exception can be used, it must be declared.
- It must be declare by the user in the declaration part of the block where the exception is used.
- A user-defined exception is **declared in the declarative part of a PL/SQL block as shown below:**

```
DECLARE
   Exception_name   EXCEPTION;
```

- Once an exception has been declared, the executable statements associated with this exception are specified in the exception-handling section of the block.
- **User-defined Exceptions Example**

```
DECLARE
         E_invalid_id EXCEPTION;
BEGIN
       .........
EXCEPTION
WHEN E_invalid_id THEN
Dbms_output.Put_line (‘An Id Cannot Be Negative‘ );
END;
/
```

c) Raising Exceptions

- Exceptions are raised by the database server automatically whenever there is any internal database errors, but exceptions can be raised explicitly by the programmer by using the command **RAISE.**

- Example

```
DECLARE
        Exception_name EXCEPTION
BEGIN
IF (condition) THEN
RAISE Exception_name;
END IF;
EXCEPTION
WHEN Exception_name THEN
dbms_output.put_line (' Raising Exceptions ');
END;
```

5.5 Cursor

- A cursor is a temporary work area created in the system memory when a SQL statement is executed.
- A cursor contains information on a select statement and the rows of data accessed by it.
- This temporary work area is used to store the data retrieved from the database, and manipulate this data.
- A cursor can hold more than one row, but can process only one row at a time. The set of rows the cursor holds is called the *active* set.
- **A cursor is a pointer to this context area.**
- **PL/SQL controls the context area through a cursor.**

5.5.1 Types of Cursor

- There are two types of cursors in PL/SQL:
  1. Implicit cursors
  2. Explicit cursors

1. Implicit cursors

- If database engine opens a cursor for internal processing, it is called as implicit cursor.
- These are created by default when DML statements like, INSERT, UPDATE, and DELETE statements are executed. They are also created when a SELECT statement that returns just one row is executed.
- When you execute DML statements like INSERT, UPDATE, DELETE & SELECT statements, implicit statements are created to process these statements.
- Oracle provides few attributes called as implicit cursor attributes to check the status of DML operations.

- **The cursor attributes available are:-**

Attributes	Return Value	Example
%FOUND	The return value is TRUE at least one row was processed.	SQL%FOUND
%NOTFOUND	The return value is TRUE if no rows were processed.	SQL%NOTFOUND
%ROWCOUNT	Return the number of rows affected by the DML operations INSERT, DELETE, UPDATE, SELECT	SQL%ROWCOUNT
%ISOPEN	True if cursor is open or false if cursor has not been opened or has been closed.	SQL%ISOPEN

- **Example of implicit cursor:**

```
DECLARE
        total_rows number (2);
BEGIN
Update EMP set salary= salary +1500 where empno =10;
If SQL%FOUND then
        Dbms_out.put_line ('Emp table modified');
Else
        Dbms_out.put_line ('Emp table not modified');
End if;
END;
```

2. Explicit Cursors

- A user can open a cursor for processing data as required. Such user defined cursors are known as explicit cursors.
- It should be defined in the declaration section of the PL/SQL block.
- They must be created when you are executing a SELECT statement that returns more than one row.
- Both implicit and explicit cursors have the same functionality, but they differ in the way they are accessed.
- **General Syntax** for creating a cursor is as given below:

```
CURSOR cursor_name IS select_statement;
```

Where,

- **cursor_name** – A suitable name for the cursor.
- **select_statement** – A select query which returns multiple rows.

- **How to use Explicit Cursor?**

- **There are four steps in using an Explicit Cursor.**
  1. **DECLARE** the cursor in the declaration section.
  2. **OPEN** the cursor in the Execution Section.
  3. **FETCH** the data from cursor into PL/SQL variables or records in the Execution Section.
  4. **CLOSE** the cursor in the Execution Section before you end the PL/SQL Block.

- **Step1: DECLARE the cursor: -** Define cursor with a name & the associated **SELECT** statement.

- For example:-

```
CURSOR c_customers IS SELECT id, name, address FROM
customers;
```

- **Step2:** Opening the cursor in the Execution Section.

- For example:-

```
OPEN c_customers;
```

- **Step3:** FETCH the data from cursor into PL/SQL variables or records in the Execution Section.

- For example:-

```
FETCH c_customers INTO C_id, C_name, C_ address;
```

- **Step4:** CLOSE the cursor in the Execution Section before you end the PL/SQL Block.

- For example:-

```
CLOSE c_customers;
```

- **Example of Explicit cursor:**

```
DECLARE
            emp_rec emp_table %rowtype;
CURSOR emp_cur IS   SELECT *FROM WHERE salary > 10K;
BEGIN
  OPEN emp_cur;
  FETCH emp_cur INTO emp_rec;
    dbms_output.put_line (emp_rec.first_name || '  ' || emp_rec.last_name);
  CLOSE emp_cur;
END;
```

5.6 Procedures

- A **stored procedure** or in simple a **proc** is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.
- A procedure has a header and a body.
- The header consists of the name of the procedure and the parameters or variables passed to the procedure.
- The body consists or declaration section, execution section and exception section similar to a general PL/SQL Block.
- A procedure is a module performing one or more actions; it does not need to return any values.
- The syntax for creating a procedure is as follows:

```
CREATE [OR REPLACE] PROCEDURE
procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN < procedure_body >
END procedure_name;
```

- Where,

- *Procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and **OUT** represents the parameter that will be used to return a value outside of the procedure.
- *Procedure-body* contains the executable part.
- The **AS** keyword is used instead of the **IS** keyword for creating a standalone procedure.

- Example

```
CREATE OR REPLACE PROCEDURE greetings AS
BEGIN
dbms_output.put_line('Hello World!');
END;
/
```

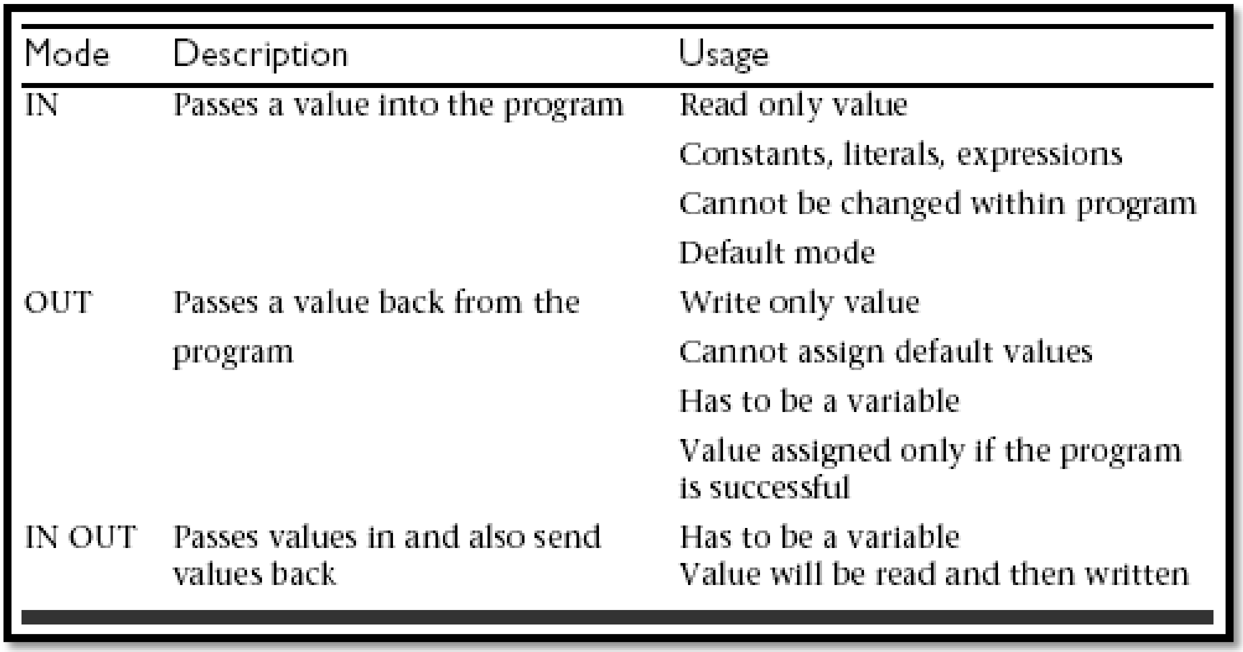
Output:-

```
SQL> Procedure Created.
```

- **Executing a Standalone Procedure**
  - A standalone procedure can be called in two ways −
  - Using the **EXECUTE** keyword
  - Calling the name of the procedure from a PL/SQL block
  - The above procedure named **'greetings'** can be called with the **EXECUTE** keyword as –

```
SQL> EXECUTE greetings;

--The above call will display this output−
 Hello World
 PL/SQL procedure successfully completed.
```

- **Types of Parameters**



Mode	Description	Usage
IN	Passes a value into the program	Read only value Constants, literals, expressions Cannot be changed within program Default mode
OUT	Passes a value back from the program	Write only value Cannot assign default values Has to be a variable Value assigned only if the program is successful
IN OUT	Passes values in and also send values back	Has to be a variable Value will be read and then written

- **Example of Procedures (**Write PL/SQL Program to finds the minimum number of two values)

```
DECLARE
        a number;
        b number;
        c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number)
IS
BEGIN
        IF x < y THEN
                z: = x;
        ELSE
                z:= y;
        END IF;
END;
BEGIN
        a:= 23;
        b:= 45;
        FindMin (a, b, c);
dbms_output.put_line (' minimum of (23, 45): ' || c);
END;
 /
```

5.7 Functions

- Functions are a type of stored code and are very similar to procedures.
- The significant difference is that a function is a PL/SQL block that ***returns*** **a single value.**
- Functions can accept one, many, or no parameters, but a function must have a return clause in the executable section of the function.
- The datatype of the return value must be declared in the header of the function.
- A function has output that needs to be assigned to a variable, or it can be used in a SELECT statement.
- The **syntax for creating a function** is as follows:

```
CREATE [OR REPLACE] FUNCTION function_name (parameter list)
RETURN datatype
IS
BEGIN
  <function body>
RETURN (return_value);
END
```

- **Functions Example**
- **SQL> Select * from customers;**

Id	Name	Age	Address	Salary
1	John	25	Kota	30000.00
2	Susan	33	Delhi	24000.00
3	David	50	Mumbai	12000.00
4	Ann	45	Pune	12000.00
5	Mary	36	Chennai	9000.00

Table 1.

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number
IS
        total number (2):= 0;
BEGIN
        SELECT count(*) into total FROM customers;
RETURN total;
END;
/
```

- When the above code is executed using the SQL prompt, it will produce the following result −

```
SQL> Function created.
```

- **Functions Example(calling function)**

```
DECLARE
        c number(2);
BEGIN
        c := totalCustomers(); -- function call
dbms_output.put_line ('Total no. of Customers: ' || c);
END;
/
```

- When the above code is executed at the SQL prompt, it produces the following result –

```
Output:-
SQL> Total no. of Customers: 5
        PL/SQL procedure successfully completed.
```

- **Example of Function**

```
DECLARE
      a number;
      b number;
      c number;
FUNCTION findMax(x IN number, y IN number)
RETURN number
IS
      z number;
BEGIN
      IF x > y THEN
      z:= x;
      ELSE
      z:= y;
END IF;
      RETURN z;
END;
BEGIN
      a:= 23;
      b:= 45;
      c := findMax(a, b);
      dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
/
```

Output

```
SQL> Maximum of (23, 45): 45
           PL/SQL procedure successfully completed.
```

Deleting a procedures:-

- PL/SQL procedure is remove from the database by using the drop procedure command.
- Syntax:- **DROP PROCEDURE** procedure_name;
- Example:- **DROP PROCEDURE greetings**

Deleting a Function:-

- PL/SQL procedure is remove from the database by using the drop function command.
- Syntax:- **DROP FUNCTION** function_name;
- Example:- **DROP FUNCTION** findMin;

Q. Difference between procedure & function

Procedure	Function
It may return a value or not.	It must return a value.
It is used for actions.	It is used for computations.
It cannot be used in select statement.	It can be used in select statement.
It is executed in the programs	It is called in the programs

5.8 Database Trigger

- Triggers are stored programs, which are automatically executed or fired when some events occur.
- A trigger is a PL/SQL block structure which is fired when a DML statements like Insert, Delete, and Update is executed on a database table.
- Triggers are, in fact, 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).

Use of database Triggers

- Triggers can be written for the following purposes −
  - Generating some derived column values automatically
  - Enforcing referential integrity
  - Event logging and storing information on table access
  - Auditing
  - Synchronous replication of tables
  - Imposing security authorizations
  - Preventing invalid transactions

- **SYNTAX:**

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE|AFTER | INSTEAD OF } triggering_event ON
 table_name
[FOR EACH ROW]
[WHEN condition]
DECLARE
Declaration statements
BEGIN
Executable statements
EXCEPTION
Exception-handling statements
END;
```

Where,

- The trigger_name references the name of the trigger.
- BEFORE or AFTER specify when the trigger is fired (before or after the triggering event).
- The triggering_event references a DML statement issued against the table (e.g., INSERT, DELETE, and UPDATE).
- The table_name is the name of the table associated with the trigger.
- The clause, FOR EACH ROW, specifies a trigger is a row trigger and fires once for each modified row.
- A WHEN clause specifies the condition for a trigger to be fired.

Trigger Example

- The following program creates a row-level trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. **(referred above table 1)**

```
CREATE OR REPLACE TRIGGER display_salary_changes BEFORE
DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
        sal_diff number;
 BEGIN
        sal_diff:=:NEW.salary - :OLD.salary;
        dbms_output.put_line ('Old salary: ' || OLD.salary);
        dbms_output.put_line (‘New salary: ' || NEW.salary);
        dbms_output.put_line ('Salary difference: ' || sal_diff);
END;
/
```

```
Output:-
 SQL> Trigger created.
```

Triggering a Trigger

- Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table −

```
SQL> INSERT INTO CUSTOMERS VALUES (7, 'Kriti', 22, 'HP', 7500.00);
```

- When a record is created in the CUSTOMERS table, the above create trigger, display_salary_changes will be fired and it will display the following result −

```
Old salary:
New salary: 7500
Salary difference:
```

Types of Database Trigger

- A trigger may be a **ROW** or **STATEMENT** type.

1. **Row Level Trigger:-**

- An event is triggered for each row updated, inserted or deleted.
- A row trigger is fired each time the table is affected by the triggering statement.
- E.g. if an Update statement updates multiple rows of a table, a row trigger is fired once for each row affected by update statement.

2. Statement Level trigger:-

- An event is triggered for each SQL statement executed.
- A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows affected by the triggering statement.

5.9 PL/SQL Security

Locks:

- Locks are mechanism used to ensure data integrity while allowing maximum concurrent access to data.
- In multi-user systems, many users may update the same information at the same time.
- Locking allows only one user to update a particular data block; another person cannot modify the same data.
- The oracle engine automatically locks table data while executing SQL statements like Select /insert/ Update/ Delete.
- Syntax:
  LOCK TABLE *TABLE-NAME* **IN** {SHARED | EXCLUSIVE} MODE;

- There are two types of Locks
  1. Shared lock
  2. Exclusive lock

1. <u>**Shared lock:-**</u>

- If a transaction ***Ti*** has obtained a **shared-mode lock** (denoted by S) on item *Q*, then ***Ti*** can read, but cannot write, *Q*.
- Shared Lock is provided to the **readers** of the data.
- These locks enable all the users to **read the concurrent data at the same time**, but they are **not allowed to change/ write the data** or obtain exclusive lock on the object.
- It could be set for table or table row. Lock is released or unlocked at the end of transaction.

2. <u>**Exclusive lock:-**</u>

- If a transaction *Ti* has obtained an **exclusive-mode lock** (denoted by X) on item *Q*, then *Ti* can both **read and write *Q***.
- Exclusive Lock is provided to the **writers** of the data.
- When this lock is set on object or transaction, it means that only writer, who has **set the lock can change the data**, and if other users cannot access the locked object.
  Lock is released at the end of change in transaction.

<u>**Locking Strategies**</u>

1. Implicit Locking:

- Oracle engine automatically locks table data while executing SQL statements
- It is fully automatic & requires no user intervention.
- Oracle automatically locks the rows whenever user performs DML operations such as Insert, Delete, and Update.

2. Explicit Locking:

- The technique of lock taken on **a table or its resources** by a user is called Explicit Locking.
- Users can lock tables they own or any tables on which they have been granted table privileges (such as select, insert, update, delete).
- Explicit locking done by two ways as:-

  1) Row Level Locks

  - It is used to lock selected rows of table. It is imposed by “for update” clause in select.

  2) Table level lock

  - Used to lock complete table.
  - To manually override Oracle’s default locking strategy by creating a data lock in a specific mode.

- **Syntax:**

  LOCK TABLE <TableName> **IN** {SHARE|EXCLUSIVE} [WAIT| NOWAIT]
- **Example:**

  LOCK TABLE Emp IN EXCLUSIVE Mode

SUMMER-16 (34 Marks)

1. Give any four advantages of using PL/SQL. *(Any four advantages - 1 mark each)*
2. What are Predefined exceptions and User defined exceptions? *(Predefined exception - 2 marks; User Defined exception - 2 marks)*
3. What is lock? Explain types of locks. *(Lock - 2 marks; Description - 1 mark each type)*
4. Write a PL/SQL program to find the square of a number given by user using WHILE….LOOP.(accept the number from user dynamically) *(Correct Program - 4 marks)*
5. Write a PL/SQL program using while loop to display n even numbers. *(Correct logic - 2 marks; correct syntax - 2 marks)*
6. List out any four statements of PL/SQL. *(Any four statement list/syntax - 1 mark each)*
7. What is database trigger? Compare database triggers and procedures and explain the use of database trigger. *(Definition - 1 mark; Comparison - 2 marks; Uses - 1 mark)*
8. Explain PL/SQL block structure. *(Correct Explanation - 4 marks)*
9. List types of cursor and explain each with example. *(Description - 1 mark; example - 1 mark for each type; 1 mark shall be awarded if only list is given)*

WINTER– 16

1. Define cursor? List the two types of cursor.
2. Explain the exception handling with its two type.
3. Explain PL/SQL Block structure.
4. What is database Trigger? How to create Trigger?
5. Write a PL/SQL program to print even or odd number from given range (Accept number range from user).
6. Explain function in PL/SQL with suitable example
7. Explain two locking strategies.
8. Explain loop control structure used in PL/SQL.

WINTER – 15

1. What statement is used in PL/SQL to display the output? (Correct statement – 2 Marks)

   Ans:

   dbms_output.put_line (var/msg);

OR

```
    set serveroutput on;
   dbms_output.put_line (var/msg);
```

2. What is Cursor?
3. Describe exception handling in brief.(Description of exception – 2 Marks; syntax – 2 Marks)
4. Explain implicit and explicit locking strategy. (Implicit Locking Strategy - 1 Mark; Explicit Locking Strategy - 3 Marks)
5. Write PL/SQL program to print even or odd numbers from given range. (Accept number from user.) (Correct logic - 2 Marks, correct syntax – 2 Marks)
6. Explain conditional control structure in PL/SQL. (Explanation of each structure - 1 Mark, syntax of each - 1 Mark)
7. What is lock based protocol? Explain share and exclusive mode protocols in lock based protocol. (Lock based protocol explanation – 2 Marks, shared and exclusive lock explanation – 2 Marks)
8. What are adv. of PL/SQL? (Any four advantages – 1 Mark each)
9. Write PL/SQL program to display factorial of any number. (Correct logic – 2 Marks, correct syntax – 2 Marks)
10. Explain implicit and explicit cursors. (Implicit cursor – 2 Marks, Explicit cursor – 2 Marks)
11. Explain parameterized cursor with example. (Explanation - 2 Marks, Any relevant Example - 2 Marks)
12. Give block structure of PL/SQL and explain main components. (Block structure - 2 Marks, Explanation - 2 Marks)