• **AIM**:

To write a PL/SQL block using different control (if, if else, for loop, while loop,…) statements.

• **OBJECTIVE:**

PL/SQL Control Structure provides conditional tests, loops, flow control and branches that let to produce well-structured programs

• **PL/SQL**

PL/SQL is Oracle's procedural language extension to SQL. PL/SQL allows you to mix SQL statements with procedural statements like IF statements, Looping structures etc.

It is an extension of SQL, the following are advantages of PL/SQL.

1. We can use programming features like if statement loops etc.
2. PL/SQL helps in reducing network traffic.
3. We can have user defined error messages by using the concept of exception handling.
4. We can perform related actions by using the concept of Triggers.
5. We can save the source code permanently for repeated execution.

• **PL/SQL Block:**

DECLARE
Declaration of variable
Declaration of cursor----------(OPTIONAL)
Declaration of exception

BEGIN
Executable commands--------(MANDATORY)

EXCEPTION
Exception handlers----------(OPTIONAL)

END;
/----- To execute the program / command

• **DECLARE** :

This section is used to declare local variables, cursors, Exceptions etc. This section is optional.

• **EXECUTABLE SECTION:**

This section contains lines of code which are used to complete the table. It is mandatory.

• **EXCEPTION SECTION:**

This section contains lines of code which will be executed only when an exception is raised. This section is optional.

• **SIMPLEST PL/SQL BLOCK:**

BEGIN

END;

• **SERVEROUTPUT:**

This will be used to display the output of the PL/SQL programs. By default this will be off.
SYNTAX:
SET SERVEROUTPUT ON | OFF
Ex:
SQL> SET SERVEROUTPUT ON

• **TRIGGERS:**

Triggers consist of a PL/SQL block that is associated with an event that occurs in the database.

• **IDENTIFIERS**

Identifiers are used to name PL/SQL objects, such as variables, cursors, types and subprograms. Identifiers consist of a letter, optionally followed by any sequence of characters, including letters, numbers, dollar signs, underscores, and pound signs only. The maximum length for an identifier is 30 characters.

EXAMPLE :

```
DECLARE
    "a" number := 5;
    "A" number := 6;
BEGIN
    dbms_output.put_line('a = ' || a);
    dbms_output.put_line('A = ' || A);
END;
/
```

OUTPUT:

a = 6
A = 6

• COMMENTS

Comments improve readability and make your program more understandable. They are ignored by the PL/SQL compiler. There are two types of comments available.

SINGLE LINE COMMENTS
MULTILINE COMMENTS

• SINGLE LINE COMMENTS

A single-line comment can start any point on a line with two dashes and continues until the end of the line.

EXAMPLE:

```
BEGIN
    Dbms_output.put_line('hello'); ----- sample program
END;
/
```

• MULTILINE COMMENTS

Multiline comments start with the /* delimiter and ends with */ delimiter.

EXAMPLE:

```
BEGIN
    Dbms_output.put_line('hello'); /* sample program */
END;
/
```

• **VARIABLE DECLARATIONS:**

Variables can be declared in declarative section of the block;

EXAMPLE:

```
DECLARE
   a number;
   b number := 17;
   c number default 21;
```

• **CONSTANT DECLARATIONS:**

To declare a constant, you include the CONSTANT keyword, and you must supply a default value.

EXAMPLE:

```
DECLARE
   B constant number := 17;
   C constant number default 19;
```

• **ANCHORED DECLARATIONS:**

PL/SQL offers two kinds of anchoring.

SCALAR ANCHORING
RECORD ANCHORING

• **SCALAR ANCHORING**

Use the %TYPE attribute to define your variable based on table's column of some other PL/SQL scalar variable.

EXAMPLE:

```
DECLARE
   dno dept.deptno%type;
   Subtype t_number is number;
   a t_number;
   Subtype t_sno is student.sno%type;
   V_sno t_sno;
```

• **RECORD ANCHORING**

Use the %ROWTYPE attribute to define your record structure based on a table.

EXAMPLE:

```
DECLARE
   V_dept dept%rowtype;
```

• BENEFITS OF ANCHORED DECLARATIONS

1 . Synchronization with database columns.
2 . Normalization of local variables.

• PL/SQL CONTROL STRUCTURES

PL/SQL has a variety of control structures that allow you to control the behaviour of the block as it runs. These structures include conditional statements and loops.

1.If-then else
2.Case
3.Case with no else
4.Labeled case
5.Searched Case
6.Simple loop
7.While loop
8.For loop
9.Goto and Labels

• IF-THEN-ELSE

Syntax:

```
IF <condition1> THEN
   Sequence of statements;
ELSEIF <condition1>THEN
   Sequence of statements;
   ……
ELSE
   Sequence of statements;
END IF;
```

EXAMPLE:

```
DECLARE
  dno number(2);
BEGIN
  select deptno into dno from dept where dname ='ACCOUNTING';
  IF dno = 10 THEN
     dbms_output.put_line('Location is NEW YORK');
  ELSEIF dno = 20 THEN
     dbms_output.put_line('Location is DALLAS');
  ELSEIF dno = 30 THEN
     dbms_output.put_line('Location is CHICAGO');
  ELSE
     dbms_output.put_line('Location is BOSTON');
  END IF;
END;
```

OUTPUT:
Location is NEW YORK

• CASE:

Syntax:

```
Case test-variable
   When value-1 then sequence of statements;
   When value-2 then sequence of statements;
   ....
   When value-n then sequence of statements;
Else sequence of statements;
End case;
```

EXAMPLE:

```
DECLARE
   dno number(2);
BEGIN
   select deptno into dno from dept where dname ='ACCOUNTING';
   case dno when 10 then
      dbms_output.put_line('Location is NEW YORK'); when 20 then
      dbms_output.put_line('Location is DALLAS'); when 30 then
      dbms_output.put_line('Location is CHICAGO');
   else
      dbms_output.put_line('Location is BOSTON');
   end case;
END;
```

Output:
Location is NEW YORK

• CASE WITHOUT ELSE:

Syntax:

```
  Case test-variable
     When value-1 then sequence of statements;
     When value-2 then sequence of statements;
     ……
     When value-n then sequence of statements;
  End case;
```

EXAMPLE:

```
DECLARE
   dno number(2);
BEGIN
   select deptno into dno from dept where dname ='ACCOUNTING';
   case dno when 10 then
      dbms_output.put_line('Location is NEW YORK'); when 20 then
      dbms_output.put_line('Location is DALLAS'); when 30 then
      dbms_output.put_line('Location is CHICAGO'); when 40 then
      dbms_output.put_line('Location is BOSTON');
   end case;
END;
```

Output:
Location is NEW YORK

• **SIMPLE LOOP**

Syntax:

```
 Loop
   Sequence of statements;
   Exit when <condition>;
 End loop;
```

In the syntax exit when <condition> is equivalent to

```
 If <condition> then
   Exit;
 End if;
```

EXAMPLE:

```
DECLARE
   i number := 1;
BEGIN
   loop
      dbms_output.put_line('i = ' ||i);
       i := i + 1;
      EXIT when i > 2;
end loop;
END;
```

Output:
i = 1
i = 2

• **WHILE LOOP:**

Syntax:

```
While <condition> loop
   Sequence of statements;
End loop;
```

EXAMPLE:

```
DECLARE
   i number := 1;
BEGIN
   While i <= 3 loop
      dbms_output.put_line('i = ' || i);
      i := i + 1;
   end loop;
END;
```

Output:

```
i = 1
i = 2
i = 3
```

• **FOR LOOP:**

Syntax:

```
For <loop_counter_variable> in low_bound..high_bound loop
   Sequence of statements;
End loop;
```

EXAMPLE-1:

```
BEGIN
   For i in 1..5 loop
      dbms_output.put_line('i = ' || i);
   end loop;
END;
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

EXAMPLE-2:

```
BEGIN
   For i in reverse 1..3 loop
      dbms_output.put_line('i = ' || i);
   end loop;
END;
```

Output:
i = 3
i = 2
i = 1

• GOTO AND LABELS:

Where label is a label defined in the PL/SQL block. Labels are enclosed in double angle brackets. When a goto statement is evaluated, control immediately passes to the statement identified by the label.

Syntax:

```
Goto label;
```

EXAMPLE:

```
BEGIN
  For i in 1..5 loop
     dbms_output.put_line('i = ' || i);
     if i = 2 then
       goto exit_loop;
     end if;
  end loop;
<<exit_loop>
Null;
END;
```

Output:
i = 1
i = 2

• **RESULT :**