UNIT-3-PL/SQL and Conditional Statement

3.1 Introduction to PL/SQL

3.1.1 Basics of PL/SQL

- The PL/SQL programming language was developed by Oracle Corporation in the late 1980s as **Procedural Language extension of SQL** and the Oracle relational database.
- SQL + Programming
- PL/SQL is the combination of SQL with procedural feature of programming language.
- It is block structural language which has multiple blocks.
- It is the combination of “What to do?” (like SQL) and also “How to do?”
- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line **SQL*Plus interface**.

3.1.2 Features and advantages of PL/SQL

Features:

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of control structures.
- It supports structured programming through functions and procedures.

- It supports object-oriented programming.
- It supports the development of web applications and server pages.
- It offers extensive error checking.

Advantages:

- PL/SQL allows sending an entire block of statements to the database at one time. This reduces network traffic and provides high performance for the applications.
- It gives high productivity to programmers as it can query, transform and update data in a database.
- It saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
- Applications written in PL/SQL are fully portable.
- It provides high security level.
- It provides access to predefined SQL packages.
- It provides support for Object-Oriented Programming.
- It allows to develop web-applications and server pages.
- It gives better performance than SQL.

3.1.3 Difference between SQL and PL/SQL

SQL	PL/SQL
It is Structural Query Language for Database.	It is a programming language using SQL for a database.
It is single query based on DDL and DML statements.	It is a block of codes which includes entire program blocks, loops, control statements, procedures, functions etc.
It specifies “What to do?”	It specifies “What to do?” and “How to do?”
It is used to manipulate data.	It is used to create an application.
Direct interaction with database server.	No direct interaction with database server.
It can be embedded within PL/SQL.	It cannot be embedded within SQL.
It is declarative language.	It is Procedural Language.
It doesn’t allow variable, control statements, loop etc.	It allows variable, control statements, loop etc.
It is data-oriented language.	It is application-oriented language.
It can execute single operation at a time.	It can perform multiple operations at a time

3.1.4 Block Structure

- PL/SQL is a block structural language.
- PL/SQL programs are divided into logical blocks.
- Within each block, there is a need to follow the structure.
- The block structure:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
  EXCEPTION
  <exception handling>
END;
```

- The block structure is divided into three sections:

1. **DELCARE Section:**

- It is used to declare variables, cursor, sub-programs and other elements which are going to use within this program.
- Once they are declared, they can be used in SQL statements for data manipulation.
- It is optional. (If there is no need to declare any element then we can omit it.)
- **DECLARE** keyword is written first to start this section.

2. **BEGIN section**

- It consists of a set of SQL and PL/SQL statements, which describe processes that have to be applied to table data.
- It is used to write PL/SQL executable statements.
- Actual data manipulation, retrieval, looping and branching constructs are specified in this section.
- Minimum one statement is compulsorily required.
- It is compulsory to write this section.
- This section start with **BEGIN** keyword.

3. **EXCEPTION section**

- This section deals with handling of errors that arise during execution of the data manipulation statements.
- It is used to write the exceptions through which the errors of the program can be handled.
- It is optional.
- **EXCEPTION** keyword is written first to start this section.
- This section is written before END of executable command section. (Means within Executable command section.)
-

4. **END section**

- This marks the end of a PL/SQL block.
- Every PL/SQL statement ends with a semicolon (;).
- **END** keyword is used to end the PL/SQL block which is followed by ';'.

Example:

```
DECLARE
        a integer :=100;
BEGIN
        a:=a+10;
        dbms_output.put_line('a= ' || a);
END;
-----------------------------------------------------------------------------------------------------------------
(Output is:                a=110)
```

3.2 PL/SQL Programming basics:

3.2.1 Executing PL/SQL file:

- PL/SQL program can be directly executed from SQL Command line (SQL *PLUS).
- The program is normally saved with '.sql' extension. (Ex: p1.sql)
- To execute PL/SQL program, use following commands:
    a. SQL> @p1
    b. SQL>execute p1

3.2.2 Output on screen:

- DBMS_OUTPUT is a package that includes a number of procedures and functions that accumulate information in a buffer so that it can be retrieved later.
- PUT_LINE puts a piece of information in the package buffer followed by an end-of-line marker.
- A message is a character string to be displayed. To display data of other data type, they must be concatenated with some character string.
    - To display messages, the SERVEROUTPUT should be set to ON. SERVEROUTPUT environment parameter that displays the information passed as a parameter to the PUT LINE function.
    -
    - PL/SQL allows programmer to give output on monitor screen
    - Using PUT_LINE procedure which belongs to DBMS_OUTPUT package is used.
    - There is a need to 'SET SERVEROUTPUT ON' before executing program. It will show information which will store in buffer.
    - Write following command to get output on screen.

    DBMS_OUTPUT.PUT_LINE(String1 || string2 || …);

Example:

```
declare
        r int :=10;
        m int :=78;
begin
        DBMS_OUTPUT.PUT_LINE('RDBMS' );
        DBMS_OUTPUT.PUT_LINE('The rollno: ' || r);
        DBMS_OUTPUT.PUT_LINE('The rollno: ' || r || ' and the marks: ' || m );
end;
/
```

```
RDBMS
The rollno: 10
The rollno: 10 and the marks: 78

PL/SQL procedure successfully completed.
```

5. DBMS_OUTPUT.PUT_LINE procedure writes each message of program to the buffer of storage. Once the program is completed, the information of buffer is stored on screen.
6. Default buffer size is 2000 bytes. It can be in between 2000 to 10,00,000 bytes.
7. To change the buffer size, write 'SET SERVEROUTPUT ON SIZE 10000'.

3.2.3 Input from user:

8. PL/SQL allows programmer to take input from user.
9. & operator is used for input.
10. Syntax:
    var:=&var; (For number related data types)
    var:= '&var'; (For text and date related data types)
11. PL/SQL asks for input from user by writing statement (Enter value for var:). The inputted value is stored in respective variable.
12. With a varchar2() and date related input either single quotation needs to be written within program or user has to enter it before writing text or date.

Example: Take input from user

```
declare
        a varchar2(10);
        b number(5,2);
        c date;
begin
        a:='&a';
        b:=&b;
        c:='&c';
        dbms_output.put_line(a || ' '||b||' '||c);
end;
/
```

```
Enter value for a: RDBMS
old   6: a:='&a';
new   6: a:='RDBMS';
Enter value for b: 45.65
old   7: b:=&b;
new   7: b:=45.65;
Enter value for c: 12-Jan-21
old   8: c:='&c';
new   8: c:='12-Jan-21';
RDBMS 45.65 12-JAN-21

PL/SQL procedure successfully completed.
```

3.2.4 Comment

13. If you want to give comment in PL/SQL program then use one of the following:
14. Single line comment: Using -- (Two times desh)

Ex: -- Program definition

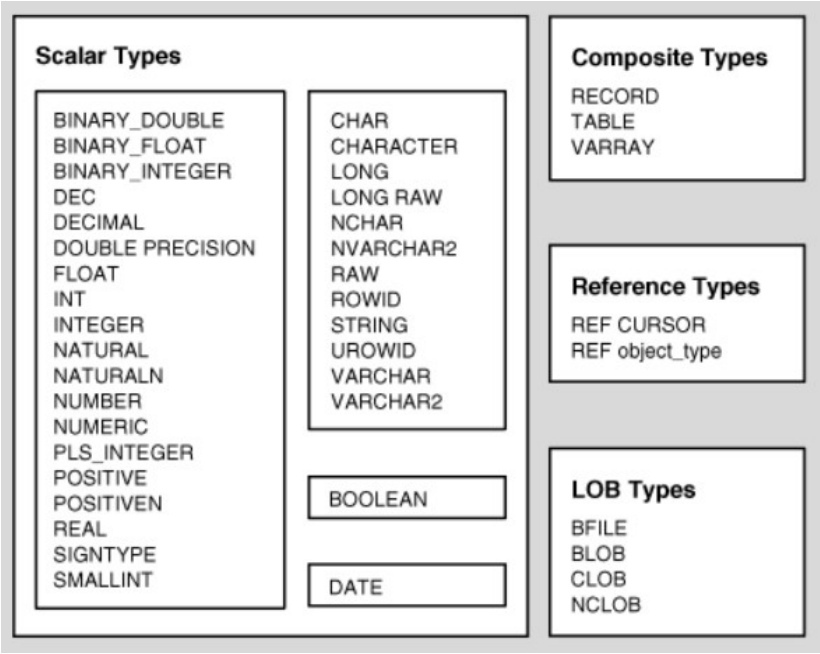
15. Multi lines comment: /*…….*/

Ex:

```
/* a:=5;
   B:=10; */
```

3.3 Variables, Constants and Data Type

3.3.1 Data type



Stud_info table:

RNO	NAME	MARKS	DID
1	Mayank	35	1
2	Maya	25	2
3	Mohit	45	1
4	Raghav	10	10
5	Ankit	49	3

3.3.2 Variables

- Variable is the name of the storage area where the data is being saved.
- Variable must be declared using data type with respective size within Declaration section first.
- Constant value can be assigned to variable and that can be changed within program.

- **Rules for variable name in PL/SQL:**
  - It can be combination of alphabet and numbers
  - It must start with alphabet.
  - Maximum 30 characters allowed.
  - Space not allowed.
  - It is case insensitive. (Capital and small letters are considered as same.)
  - Special characters are not allowed only ($, _ , #) are allowed.
  - Keywords are not allowed as a variable name.

- **Single value variable declaration:**
  syntax:
  variable_name datatype [NOT NULL] [:= |DEFAULT] value;
  Examples:
  - sum int :=0;
  - name varchar2(10);
  - gender varchar2(6) :='Female';
  - city varchar2(10) DEFAULT 'Surat';
- Dynamic declaration is also possible. If you don't know the exact data type of field then '%type' can be used which take the datatype of same field from table.
- Example:
  - a1 stud_info.name %type;
    Here a1 is the name of variable which is created as per stud_info table's name field and it takes the same data type of name field.

- **Assigning value to Variable:**
  By default, PL/SQL assigns NULL value to variable when it is declared within declaration section if user doesn't assign any value to variable.

Within Execution program section, user can assign value to the variable.

Syntax:

Variable_name := value|expression|…;

Examples:

- sum:= sum + 100;
- gender := 'Male';
- bdate:= '12-Jan-2001';
- select name into a1 from stud_info where rno=1;

Example:

```
DECLARE
        v_num integer;
        v_dy stud_info.name %type;
        v_dy1 stud_info.marks %type;
        v_date date;
        v_text varchar2(10);
BEGIN
        v_num:=15;
        v_date:='15-Jan-2021';
        v_text:='Manish';
        select name into v_dy from stud_info where rno=1;
        select marks into v_dy1 from stud_info where rno=&rno;      --Allow to take input
        dbms_output.put_line('v_num= '||v_num);
        dbms_output.put_line('v_date='||v_date);
        dbms_output.put_line('v_text= '||v_text);
        dbms_output.put_line('v_dy='||v_dy);
        dbms_output.put_line('v_dy1='||v_dy1);
END;
/
```

```
Enter value for rno: 3
old  12: select marks into v_dy1 from stud_info where rno=&rno;       --Allow to take input
new  12: select marks into v_dy1 from stud_info where rno=3;       --Allow to take input
v_num= 15
v_date=15-JAN-21
v_text= Manish
v_dy=Mayank
v_dy1=45

PL/SQL procedure successfully completed.
```

3.3.3 Constant

- CONSTANT keyword is used to declare a variable as constant value throughout the program execution.
- Syntax:

  Variable_name [CONSTANT] datatype := value;
- Example: pi CONSTANT double precision :=3.14;

3.4 Table defined Record (%ROWTYPE) (Not in syllabus)

- PL/SQL allows to create user defined record for a row of table.
- It allows to declare one record whose structure is similar to table structure.
- Using %rowtype, user can create table-based record. It automatically assigns the data type similar to the fields of table.
- When we want to write block in which we want to work with single record of table, then the respective object can be created based on table.
- **Declaration:**

  Object_name Table_name %rowtype;

Example:

```
DECLARE
    stud1 stud_info %rowtype;
BEGIN
    select * into stud1 from stud_info where rno=1;     --single row output only
    dbms_output.put_line('Rollno= ' || stud1.rno);
    dbms_output.put_line('Name= ' || stud1.name);
    dbms_output.put_line('Marks= ' || stud1.marks);
    dbms_output.put_line('did= ' || stud1.did);
end;
/
```

```
Rollno= 1
Name= Mayank
Marks= 35
did= 1

PL/SQL procedure successfully completed.
```

3.5 User Defined Record

- User defined record allows us to create multiple fields' object of single table or using multiple tables.
- It allows to create user defined record structure.
- When we want to write block in which we don't want all fields' information from table, then in place of table defined record, we can use user defined record.
- Declaration:

```
TYPE type_name IS RECORD
 ( field_name1  datatype1  [NOT NULL]  [:= DEFAULT EXPRESSION],
  field_name2   datatype2   [NOT NULL]  [:= DEFAULT EXPRESSION],
  ...
  field_nameN  datatypeN  [NOT NULL]  [:= DEFAULT EXPRESSION);
record-name type_name;
```

Example:

```
DECLARE
TYPE stud_dept IS RECORD
(rno stud_info.rno %type,
name stud_info.name %type,
dname dept.dname %type);
s1 stud_dept;
s2 stud_dept;
BEGIN
select rno,name,dname into s1 from stud_info, dept where stud_info.did=dept.did and
rno=1;
select rno,name,dname into s2 from stud_info, dept where stud_info.did=dept.did and
dept.did=3;
dbms_output.put_line('Roll no= ' || s1.rno);
dbms_output.put_line('Name= ' || s1.name);
dbms_output.put_line('Department Name= ' || s1.dname);
dbms_output.put_line('-----------------------------------------------------');
dbms_output.put_line('Roll no= ' || s2.rno);
dbms_output.put_line('Name= ' || s2.name);
dbms_output.put_line('Department Name= ' || s2.dname);
end;
/
```

```
Roll no= 1
Name= Mayank
Department Name= IT
-----------------------------------------------------
Roll no= 5
Name= Ankit
Department Name= Science

PL/SQL procedure successfully completed.
```

3.6 Conditional Statements

- Conditional statements are used to perform statement or statements as per the condition. If the condition becomes true then evaluates one type of statements and if the condition is false then evaluates another type of statements.
- Following are the different conditional statements provided by PL/SQL.

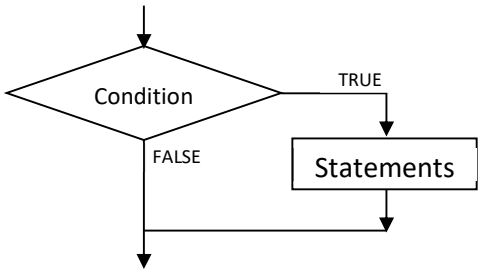
3.6.1 IF…THEN statement

- It is used to execute statements if the given condition is true. If the condition is false or NULL then it does nothing.

Syntax:

```
IF <condition> THEN
        Statement(s);
END IF;
```

Flowchart:



Example: Check first number is smaller than second number.

```
declare
        x int;
        y int;
begin
        x:=&x;          --input from user
        y:=&y;          --input from user
        if x<y then
                dbms_output.put_line(x || ' is less than '||y);
        end if;
end;
/
```

```
Enter value for x: 4
old   5: x:=&x;
new   5: x:=4;
Enter value for y: 5
old   6: y:=&y;
new   6: y:=5;
4 is less than 5

PL/SQL procedure successfully completed.
```

```
Enter value for x: 5
old   5: x:=&x;
new   5: x:=5;
Enter value for y: 2
old   6: y:=&y;
new   6: y:=2;

PL/SQL procedure successfully completed.
```

Database Example: Display the marks if given student's mark is greater than 30.

```
declare
        stud1 stud_info %rowtype;
begin
        select * into stud1 from stud_info where rno=&rno;
        if (stud1.marks>30) then
                dbms_output.put_line('The marks of Rollno: '||stud1.rno || ' is '||stud1.marks);
        end if;
end;
/
```

```
Enter value for rno: 1
old   4: select * into stud1 from stud_info where rno=&rno;
new   4: select * into stud1 from stud_info where rno=1;
The marks of Rollno: 1 is 35

PL/SQL procedure successfully completed.
```

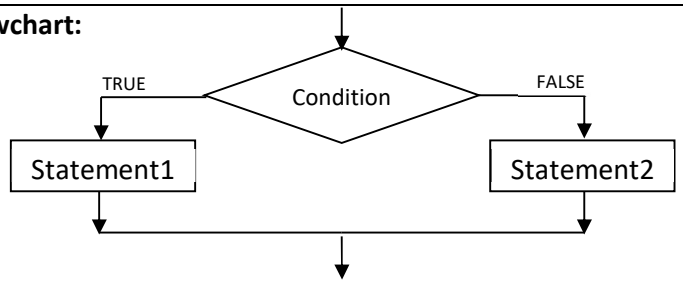
3.6.2 IF-ELSE statements

- It is used to execute statements as the condition. If the condition is true then it executes statements1 else it executes statements 2.

Syntax:

```
IF <Condition> THEN
        Statements1;
ELSE
        Statements2;
END IF;
```

Flowchart:



Example: Display the smallest number from given two numbers.

```
declare
        x int;
        y int;
begin
        x:=&x;
        y:=&y;
        if x<y then
                dbms_output.put_line(x || ' is less than '||y);
        else
                dbms_output.put_line(y || ' is less than '||x);
        end if;
end;
/
```

```
Enter value for x: 4
old   5: x:=&x;
new   5: x:=4;
Enter value for y: 5
old   6: y:=&y;
new   6: y:=5;
4 is less than 5

PL/SQL procedure successfully completed.
```

```
Enter value for x: 5
old   5: x:=&x;
new   5: x:=5;
Enter value for y: 2
old   6: y:=&y;
new   6: y:=2;
2 is less than 5

PL/SQL procedure successfully completed.
```

Database Example: Display whether the student is pass or fail.

```
declare
        stud1 stud_info %rowtype;
begin
        select * into stud1 from stud_info where rno=&rno;
        if (stud1.marks>25) then
                dbms_output.put_line('Student is Pass.');
        else
                dbms_output.put_line('Student is Fail.');
        end if;
end;
/
```

```
Enter value for rno: 3
old   4: select marks into m from stud_info where rno=&rno;
new   4: select marks into m from stud_info where rno=3;
Student is Pass.

PL/SQL procedure successfully completed.
```

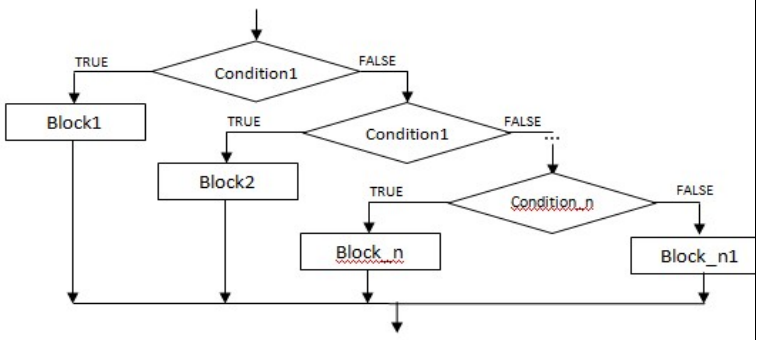
3.6.3 Multiple conditions (IF-THEN-ELSEIF-ELSE statements)

- It is used to check the multiple conditions and gives the output accordingly.
- If condition1 is true then block1 is executed else condition2 is checked. If it is true then block2 is executed and so on. If all mentioned conditions are false then Block_n1 is executed.

Syntax:

```
IF <Condition1> THEN
        block1;
ELSIF <Condition2> THEN
        block2;
...
ELSIF <Condition-n> THEN
        block_n;
ELSE
        block_n1;
END IF;
```

Flowchart:



Example: Display the smallest number and if both the numbers are same then display the message accordingly.

```
declare
        x int :=10;
        y int :=10;
begin
        if x<y then
                dbms_output.put_line(x || ' is less than '||y);
        elsif x>y then
                dbms_output.put_line(y || ' is less than '||x);
        else
                dbms_output.put_line(x ||' and '||y || ' both are equal.');
        end if;
end;
/
```

```
10 and 10 both are equal.

PL/SQL procedure successfully completed.
```

Database Example: Display the class as per the marks of given student.

```
declare
        m stud_info.marks %type;
begin
        select marks into m from stud_info where rno=&rno;
        if (m>=70) then
                dbms_output.put_line('Distinction');
        elsif (m>=60) then
                dbms_output.put_line('First class');
        elsif (m>=50) then
                dbms_output.put_line('Second class');
        elsif (m>=40) then
                dbms_output.put_line('Third class');
        elsif (m>=25) then
                dbms_output.put_line('Pass class');
        else
                dbms_output.put_line('Fail');
        end if;
end;
/
```

```
Enter value for rno: 5
old   4: select marks into m from stud_info where rno=&rno;
new   4: select marks into m from stud_info where rno=5;
Third class

PL/SQL procedure successfully completed.
```

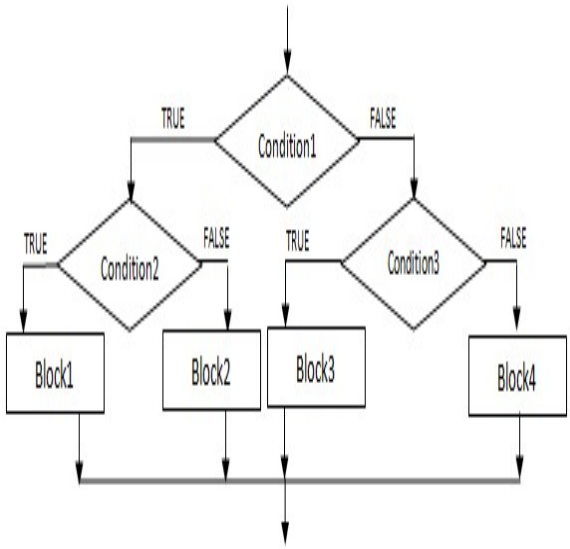
3.6.4 Nested IF statements

- Nested if is used to write one if condition within another if condition.
- First Condition1 is checked
- If it is true then condition2 is checked. If the condition2 is true then block1 is executed else block2 is executed.
- If it is false then condition3 is checked. If the condition3 is true then block3 is executed else block4 is executed.

Syntax:

```
IF <Condition1> THEN
        IF <Condition2> THEN
                block1;
        ELSE
                block2;
        END IF;
ELSE
        IF<Condition3> THEN
                block3;
        ELSE
                block4;
        END IF;
END IF;
```

Flowchart:



Example: Display the smallest number among three numbers.

```
declare
        x int ;
        y int ;
        z int ;
begin
        x:=&x;
        y:=&y;
        z:=&z;
        if x<y then
          if x<z then
             dbms_output.put_line(x || ' is less than '||y|| ' and '||z);
          else
             dbms_output.put_line(z || ' is less than '||y|| ' and '||x);
          end if;
      else
          if y<z then
             dbms_output.put_line(y || ' is less than '||x|| ' and '||z);
          else
              dbms_output.put_line(z || ' is less than '||y|| ' and '||x);
          end if;
      end if;
end;
/
```

```
Enter value for x: 23
old   6:          x:=&x;
new   6:          x:=23;
Enter value for y: 43
old   7:          y:=&y;
new   7:          y:=43;
Enter value for z: 12
old   8:          z:=&z;
new   8:          z:=12;
12 is less than 43 and 23

PL/SQL procedure successfully completed.
```

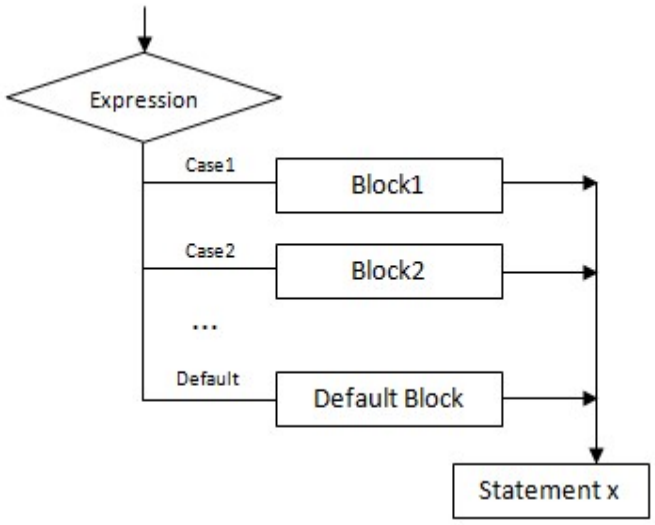
3.6.5 CASE statements

- CASE is used to compare one value from given many values.
- It compares a variable with many values and evaluates the block where it finds equality.

Syntax:

```
CASE Expression
    WHEN value1 THEN Block1;
    WHEN value2 THEN Block2;
    ….
    ….
    ELSE Default block;
END CASE;
```

Flowchart:



Example: Display the week day as per the given number.

```
declare
        no int;
begin
     no:= &no;
     case no
        when 1 then dbms_output.put_line('Monday');
        when 2 then dbms_output.put_line('Tuesday');
        when 3 then dbms_output.put_line('Wednesday');
        when 4 then dbms_output.put_line('Thursday');
        when 5 then dbms_output.put_line('Friday');
        when 6 then dbms_output.put_line('Saturday');
        when 7 then dbms_output.put_line('Sunday');
        else dbms_output.put_line('Wrong input’);
      end case;
end;
/
```

```
Enter value for no: 4
old   4:          no:= &no;
new   4:          no:= 4;
Thursday

PL/SQL procedure successfully completed.
```

Database Example: Update the marks of given student as per following condition:

If did=1 then marks=marks + 10;

If did=2 then marks=marks + 5;

Otherwise, marks=marks +20;

```
declare
        stud1 stud_info1 %rowtype;
begin
        select * into stud1 from stud_info1 where rno =&rno;
        case stud1.did
                when 1 then
                        update stud_info1 set stud_info1.marks=stud_info1.marks+10 where rno=stud1.rno;
                when 2 then
                        update stud_info1 set stud_info1.marks=stud_info1.marks+5 where rno=stud1.rno;
                else
                        update stud_info1 set stud_info1.marks=stud_info1.marks+20 where rno=stud1.rno;
        end case;
end;
/
```

```
Enter value for rno: 1
old   4: select * into stud1 from stud_info1 where rno =&rno;
new   4: select * into stud1 from stud_info1 where rno =1;

PL/SQL procedure successfully completed.
```