Chapter 16

CONTROL STRUCTURES

- IF statement
- Loop…End Loop
- Exit command
- While Loop
- For loop
- Goto statement

IF statement

IF statement is used to check the condition and execute statements depending upon the result of the condition.

The following is the syntax of IF statement.

```
IF condition-1 THEN

      statements_set_1;

[ELSIF condition-2 THEN

       statements_set_2;] ...
[ELSE

       statements_set_3; ]

END IF;
```

Condition is formed using relational operators listed in table 1.

Operator	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equal to
<>, !=, ~=, ^=	Not equal to
LIKE	Returns true if the character pattern matches the given value.
BETWEEN..AND	Returns true if the value is in the given range.
IN	Returns true if the value is in the list.
IS NULL	Return true if the value is NULL.

Table 1: Relational Operators.

In order to combine two conditions, logical operators – AND and OR are used. When two conditions are combined with AND then both the conditions must be true to make the entire condition true. If conditions are combined with OR then if any one condition is true then the entire condition will be true.

The following are valid conditions:

```
    If amt > 5000 then

    If rate > 500 and qty < 10 then

    If rate between 100 and 200 then
```

Now, let us write a simple PL/SQL block that uses IF statement.

The following program will increase the course fee of Oracle by 10% if more than 100 students have joined for Oracle, otherwise it will decrease the course fee by 10%.

```
declare
   v_ns  number(5);
begin

    -- get no. of students of the course

    select  count(*) into v_ns
    from students
    where  bcode in ( select bcode from  batches
                      where  ccode = 'ora');
```

```
    if  v_ns > 100 then
           update courses set fee = fee * 1.1
           where  ccode = 'ora';
    else
           update courses set fee = fee * 0.9
           where  ccode = 'ora';
    end if;

    commit;

end;
/
```

The above block uses IF statement to check whether the variable V_NS is greater than 100. If the condition is satisfied it will increase the course fee by 10% otherwise it will decrease the course fee by 10%.

The above program will either increase the course fee by 10% or decrease it by 10%. That means it will take either of two possible actions. But in some cases we may have more than two actions. For instance, what if we have to change the course fee as follows based on the number of students joined in Oracle course:

No. of Students	**Percentage of change**
>100	Increase by 20
>50	Increase by 15
>10	Increase by 5
<=10	Decrease by 10

The following program will change course fee of Oracle according to the above table.

```
declare
   v_ns  number(5);
   v_fee courses.fee%type;

begin

    -- get no. of students of the course

    select  count(*) into v_ns
    from students
    where  bcode in ( select bcode from  batches
                      where  ccode = 'ora');

    select fee into v_fee
    from   courses
    where  ccode = 'ora';
```

```
    if  v_ns > 100 then
              v_fee := v_fee * 1.2;  -- 20%
    elsif  v_ns > 50 then
              v_fee := v_fee * 1.15;  -- 15%
    elsif  v_ns > 10 then
              v_fee := v_fee * 1.15;  -- 15%
    else
              v_fee := v_fee * 0.90;  -- 10% decrease

    end if;

    -- update fee in table

    update courses set fee = v_fee
    where  ccode = 'ora';


end;
/
```

The above program first checks whether the number of students is more than 100. If so, it increases course fee by 20%. If the first condition is not satisfied then it checks whether second condition is true (v_no > 50) and if so it executes the statement after that ELSIF.

The IF statement will be terminated once a condition is true and the corresponding statements are executed.

If none of the conditions is true then it executes statements given after ELSE.

Every IF must have a matching END IF. However, ELSIF that is used to check for a condition need not have corresponding END IF. Also note that ELSIF can be used only after an IF is used.

Apart from checking condition and executing statements depending on the result of the condition, PL/SQL also allows you to repeatedly execute a set of statements. Repeatedly executing a set of statements is called as looping structure.

In the next few sections we will discuss about looping structures.

LOOP

This is used to repeatedly execute a set of statements. This is the simplest form of looping structures.

```
LOOP
     Statements;
END LOOP;
```

***Note**: Loop… End Loop has no termination point. So unless you terminate loop using EXIT command (discussed next) it becomes an infinite loop.*

Statements are the statements that are to be repeatedly executed. In case of LOOP, these statements must make sure they exit the loop at one point or other. Otherwise the statements will be executed indefinitely.

EXIT

This is used to exit out of a Loop. This is mainly used with LOOP statement, as there is no other way of terminating the LOOP.

The following is the syntax of EXIT command.

```
EXIT  [WHEN condition];
```

If EXIT is used alone, it will terminate the current loop as and when it is executed.

If EXIT is used with WHEN clause, then the current loop is terminated only when the *condition* given after WHEN is satisfied.

The following examples show how you can use EXIT and EXIT WHEN to exit a Loop.

Example 1:

```
LOOP
        ...

        IF count > 10 THEN
            EXIT;    --  terminates  loop
        END IF;

        ...

END  LOOP;
```

Example 2:

```
LOOP
        ...

        EXIT WHEN count >10; -- terminates loop

        ...

END LOOP;
```

The following program will display numbers 1 to 10 using LOOP.

```
declare
     i number(2) := 1;
begin

     loop
         dbms_output.put_line(i);
         i := i + 1;
         exit when i > 10;
     end loop;

end;
/
```

In the above program, we first initialized variable **I** to 1. Then LOOP starts and displays the value of **I** on the screen. Then **I** is incremented by one. EXIT statement checks whether **I** is greater than 10. If the condition is true then EXIT is executed and LOOP is terminated otherwise it enters into loop again and redisplays the value of **I**.

Nested Loops

It is possible to have a loop within another loop. When a loop is placed within another loop it is called as nested loop. The inner loop is executed for each iteration of outer loop.

The following example will display table up to 10 for numbers from 1 to 5.

```
declare

    i number(2);
    j number(2);

begin
```

```
    i := 1;

    loop

       j:= 1;

       loop

          dbms_output.put_line(i || '*' || j || '=' || i * j);
          j := j + 1;
          exit when j > 10;

       end loop;

       i := i + 1;

       exit when i > 5;

    end loop;

end;
/
```

Exiting from nested loop

It is possible to exit current loop using EXIT statement. It is also possible to use EXIT statement to exit any enclosing loop. This is achieved using **loop label**.

A loop label is a label that is assigned to a loop. By using this label that is assigned to a loop, EXIT can exit a specific loop instead of the current loop.

The following example uses EXIT to exit the outer loop.

```
<<outerloop>>
LOOP
   ...
   LOOP
      ...
      EXIT outerloop WHEN ...  - exits outer loop

   END LOOP;
   ...
END LOOP;
```

EXIT statement uses label to specify which enclosing loop is to be terminated. EXIT uses *outerloop*, which is the label given to outer loop, to terminate outer loop.

WHILE

Executes a series of statements as long as the given condition is true.

```
  WHILE condition LOOP
        Statements;
  END LOOP;
```

As long as the *condition* is true then statements will be repeatedly executed. Once the condition is false then loop is terminated.

The following example will display numbers from 1 to 10.

```
declare
     i number(2) := 1;
begin

     while  i <= 10
     loop
         dbms_output.put_line(i);
         i := i + 1;
     end loop;

end;
/
```

As long as the condition (I<=10) is true statements given within LOOP and END LOOP are executed repeatedly.

The condition is checked at the beginning of iteration. Statements are executed only when the condition is true otherwise loop is terminated.

FOR

This looping structure is best suited to cases where we have to repeatedly execute a set of statements by varying a variable from one value to another.

```
FOR counter IN [REVERSE] lowerrange .. upperrange   LOOP
          Statements;
END LOOP;
```

lowerrange and *upperrange* may also be expressions.

FOR loop sets *counter* to lower range and checks whether it is greater than upper range. If *counter* is less than or equal to upper range then statements given between LOOP and END LOOP will be executed. At the end of execution of statements, counter will be incremented by one and the same process will repeat.

The following is the sequence of steps in FOR LOOP.

Steps

The following is the sequence in which FOR will take the steps.

1. Counter is set to lowerrange.
2. If counter is less than or equal to upperrange then statements are executed otherwise loop is terminated.
3. Counter is incremented by one and only one. It is not possible to increment counter by more than one.
4. Repeats step2.

The following example will display numbers from 1 to 10.

```
begin

    for i in 1..10
    loop
         dbms_output.put_line(i);
    end loop;

end;
/
```

If REVERSE option is used the following steps will take place:

1. Counter is set to upper range.
2. If counter is greater than or equal to lower range then statements are executed otherwise loop is terminated.
3. Counter is decremented by one.
4. Go to step 2.

The following FOR loop uses REVERSE option to display number from 10 to 1.

```
begin

    for i in REVERSE 1..10
    loop
         dbms_output.put_line(i);
    end loop;

end;
/
```

Note: *It is not possible to change the step value for FOR loop.*

Sample program using FOR loop

The following example will display the missing roll numbers. The program starts at lowest available roll number and goes up to largest roll number. It will display the roll numbers that are within in the range and not in the STUDENTS table.

```
declare

  v_minrollno  students.rollno%type;
  v_maxrollno  students.rollno%type;
  v_count      number(2);

begin

  -- get min and max roll numbers

  select  min(rollno) , max(rollno) into  v_minrollno, v_maxrollno
  from  students;

  for i  in v_minrollno .. v_maxrollno
  loop

      select count(*) into v_count
      from students
      where rollno = i;

      -- display roll number if count is 0
      if  v_count = 0 then
            dbms_output.put_line(i);
      end if;

  end loop;
end;
/
```

The above program takes minimum and maximum roll numbers using MIN and MAX functions. Then it sets a loop that starts at minimum roll number and goes up to maximum roll number. In each iteration it checks whether there is any student with the current roll number (represented by variable **i**). If no row is found then COUNT (*) will be 0. So it displays the roll number if count is zero.

GOTO statement

Transfers control to the named label. The label must be unique and should precede an executable PL/SQL statement or PL/SQL block.

The following example shows how to create label and how to transfer control to the label using GOTO statement.

```
BEGIN

            ...
            GOTO   change_details;
            ...

          <<change_details>>
          update  students  ... ;

END;
```

GOTO statement transfers control to UPDATE statement that is given after the label ***change_details***

Label is created by enclosing a name within two sets of angle brackets (<< >>). The label must be given either before an executable PL/SQL statement or a block.

Restrictions

The following are the restrictions on the usage of GOTO statement.

- Cannot branch into an IF statement
- Cannot branch into a LOOP
- Cannot branch into a Sub block.
- Cannot branch out of a subprogram – a procedure or function.
- Cannot branch from exception handler into current block.

The following is an **invalid** usage of GOTO statement as it tries to enter into an IF block:

```
BEGIN
          ...
         /* following  is  invalid because GOTO can not
            branch into  an IF statement */

           GOTO change_details;
           ...
           IF  condition  THEN
                 ...
                <<change_details>>
                update  students  ... ;
                 ...
           END IF;
END;
```

Summary

Any programming language has basic control structures like IF statement, looping structures etc. These control structures are an important part of PL/SQL, because they allow data processing. In this chapter, we have covered basic control structure, but the practical usage of these control constructs will be better understood as you write more programs.

Exercises

1. Write a PL/SQL block to decrease the duration of the course C++ if more than 2 batches have started in the last two months.
2. Write a PL/SQL block to insert a new row into PAYMENTS table with the following data:
   - Roll number is the number of student with the name *George Micheal*.
   - Date of payment is previous Monday.
   - Amount is the balance amount to be paid by the student.
3. Display how many students have joined in each month in the current year.