PL/SQL Control Structure

Dr. Abu Raihan Mostofa Kamal

December 7, 2021

Conditional statements

IF Statements. IF statements evaluate a condition. The condition can be any comparison expression, or set of comparison expressions that evaluates to a logical true or false.

```
DECLARE
X NUMBER;
BEGIN
X:=10;
IF (X = 0) THEN
dbms_output.put_line('The value of x is 0 ');
ELSIF(X between 1 and 10) THEN
dbms_output.put_line('The value of x is between 1 and 10 ');
ELSE
dbms_output.put_line('The value of x is greater than 10 ');
END IF;
END;


```

IF statement: Application

```
 DECLARE
CGPA NUMBER;
X NUMBER :=034403;
BEGIN

SELECT MAX(CGPA) INTO CGPA
FROM STUDENTS
WHERE ID=X;

IF (CGPA >3.78) THEN
dbms_output.put_line('Brilliant');
ELSIF(CGPA between 3.5 and 3.78) THEN
dbms_output.put_line('Mid Level');
ELSE
dbms_output.put_line('Poor');
END IF;
END;
```

Simple CASE Statements

The simple CASE statement sets a selector that is any PL/SQL datatype except a BLOB, BFILE, or composite type.

```

DECLARE
selector NUMBER := 1;
BEGIN
CASE selector
WHEN 0 THEN
dbms_output.put_line('Case 0!');
WHEN 1 THEN
dbms_output.put_line('Case 1!');
ELSE
dbms_output.put_line('No match!');
END CASE;
END;
/
```

Simple CASE Statements: Application

Normally it is used in aid of function or procedure.

```

CASE employee_type
WHEN 'P' THEN  --permanent
award_salary_bonus(employee_id);
WHEN 'C' THEN  --contractual
award_hourly_bonus(employee_id);
WHEN 'T' THEN  ---temporary
award_commissioned_bonus(employee_id);
ELSE
NULL; ---do nothing
END CASE;
```

Searched CASE Statements

It enables to apply logic on the selected value in SQL, similarly it can be used inside your PL SQL code with **into** clause.

```

 SELECT name, ID,
(CASE
WHEN salary < 1000 THEN 'Low'
WHEN salary BETWEEN 1000 AND 3000 THEN 'Medium'
WHEN salary > 3000 THEN 'High'
ELSE 'N/A'
END) salary
FROM emp
ORDER BY name;


```

LOOP

LOOP and EXIT Statements

```
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;
/
```

```
SQL> /
10
20
30
40
50
After Exit x is: 60

PL/SQL procedure successfully
  completed.

```

FOR LOOP

A simple Example:

```
DECLARE
a number(2);
BEGIN
FOR a in 10 .. 15 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

```

FOR LOOP: Application in DBMS

In general, many applications need a **scheduled data feeding mechanism** which involves such loops.

- A loan in bank can be pre-scheduled
- A student's tuition may be pre-scheduled