

CSE 4409: Database Management Systems II

Abu Raihan Mostofa Kamal

Professor, CSE Department
Islamic University of Technology (IUT)
Gazipur-1704 Bangladesh
✉ raihan.kamal@iut-dhaka.edu
☎ +8801843925543

January 30, 2024

Chapter Outline

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.

```
1 DECLARE
2 X NUMBER;
3 BEGIN
4 X:=10;
5 IF (X = 0) THEN
6 dbms_output.put_line('The value of x is 0 ');
7 ELSIF(X between 1 and 10) THEN
8 dbms_output.put_line('The value of x is between 1 and 10 ');
9 ELSE
10 dbms_output.put_line('The value of x is greater than 10 ');
11 END IF;
12 END;
```



IF statement: Application

```
1 DECLARE
2 CGPA NUMBER;
3 X NUMBER :=034403;
4 BEGIN
5
6
7 SELECT MAX(CGPA) INTO CGPA
8 FROM STUDENTS
9 WHERE ID=X;
10
11 IF (CGPA >3.78) THEN
12 dbms_output.put_line('Brilliant');
13 ELSIF(CGPA between 3.5 and 3.78) THEN
14 dbms_output.put_line('Mid Level');
15 ELSE
16 dbms_output.put_line('Poor');
17 END IF;
18 END;
```



Simple CASE Statements

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

```
1  
2 DECLARE  
3 selector NUMBER := 1;  
4 BEGIN  
5 CASE selector  
6 WHEN 0 THEN  
7 dbms_output.put_line('Case 0!');  
8 WHEN 1 THEN  
9 dbms_output.put_line('Case 1!');  
10 ELSE  
11 dbms_output.put_line('No match!');  
12 END CASE;  
13 END;  
14 /
```

Simple CASE Statements: Application

Normally it is used in aid of function or procedure.

```
1
2  CASE employee_type
3  WHEN 'P' THEN --permanent
4  award_salary_bonus(employee_id);
5  WHEN 'C' THEN --contractual
6  award_hourly_bonus(employee_id);
7  WHEN 'T' THEN ---temporary
8  award_commissioned_bonus(employee_id);
9  ELSE
10 NULL; ---do nothing
11 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.

```
1
2      SELECT name, ID,
3      (CASE
4      WHEN salary < 1000 THEN 'Low'
5      WHEN salary BETWEEN 1000 AND 3000 THEN 'Medium'
6      WHEN salary > 3000 THEN 'High'
7      ELSE 'N/A'
8      END) salary
9      FROM emp
10     ORDER BY name;
```

LOOP

LOOP and EXIT Statements

```
1      DECLARE
2      x number := 10;
3      BEGIN
4      LOOP
5      dbms_output.put_line(x);
6      x := x + 10;
7      IF x > 50 THEN
8      exit;
9      END IF;
10     END LOOP;
11     -- after exit, control
12       resumes here
13     dbms_output.put_line('After
14       Exit x is: ' || x);
15     END;
16     /
```

```
1      SQL> /
2      10
3      20
4      30
5      40
6      50
7      After Exit x is: 60
8
9      PL/SQL procedure
      successfully completed.
```



FOR LOOP

A simple Example:

```
1 DECLARE
2   a number(2);
3 BEGIN
4   FOR a in 10 .. 15 LOOP
5     dbms_output.put_line('value
6       of a: ' || a);
7   END LOOP;
8 END;
```

```
1 OUTPUT:
2
3 value of a: 10
4 value of a: 11
5 value of a: 12
6 value of a: 13
7 value of a: 14
8 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

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