Conditional Statements in PL/SQL

- PL/SQL supports programming language features like conditional statements, iterative statements.
- The programming constructs are similar to how you use in programming languages like Java and C++.

PL/SQL IF Statement

- The ***PL/SQL IF statement*** allows you to execute a sequence of statements conditionally.
- The IF statements evaluate a condition.
- The condition can be anything that evaluates to a logical true or false
- such as comparison expression or combination of multiple comparison expressions.

PL/SQL IF Statement

- You can compare two variables of the same type or different types but they are convertible to each other.
- You can compare two literals.
- In addition, a Boolean variable can be used as a condition.
- The PL/SQL IF statement has three forms:
- IF-THEN, IF-THEN-ELSE and IF-THEN-ELSIF

PL/SQL IF-THEN Statement

- The following is the syntax of the IF-THEN statement:
- IF condition THEN
  - sequence_of_statements;
- END IF;

PL/SQL IF-THEN Statement

- If the condition evaluates to true, the sequence of statements will execute.
- If the condition is false or NULL,
- the IF statement does nothing.
- Note that END IF is used to close the IF statement, not ENDIF.

PL/SQL IF-THEN Statement

```
1.    DECLARE
2.      n_min_salary NUMBER(6,0);
3.      n_max_salary NUMBER(6,0);
4.      n_mid_salary NUMBER(6,2);
5.      n_salary     EMPLOYEES.SALARY%TYPE;
6.      n_emp_id     EMPLOYEES.EMPLOYEE_ID%TYPE := 200;
```

PL/SQL IF-THEN Statement

```
1.      BEGIN
2.        -- get salary range of the employee
3.        -- based on job
4.        SELECT min_salary,
5.               max_salary
6.        INTO n_min_salary,
7.             n_max_salary
8.        FROM JOBS
9.        WHERE JOB_ID = (SELECT JOB_ID
10.                       FROM EMPLOYEES
11.                       WHERE EMPLOYEE_ID = n_emp_id);
```

PL/SQL IF-THEN Statement

```
1.      -- calculate mid-range
2.        n_mid_salary := (n_min_salary + n_max_salary) / 2;
3.        -- get salary of the given employee
4.        SELECT salary
5.        INTO n_salary
6.        FROM employees
7.        WHERE employee_id = n_emp_id;
8.
```

PL/SQL IF-THEN Statement

```
1.      -- update employee's salary if it is lower than
2.        -- the mid range
3.        IF n_salary < n_mid_salary THEN
4.          UPDATE employees
5.          SET salary = n_mid_salary
6.          WHERE employee_id = n_emp_id;
7.        END IF;
8.      END;
```

PL/SQL IF-THEN-ELSE Statement

- This is the second form of the IF statement.
- The ELSE keyword is added with the alternative sequence of statements.
- Below is the syntax of the IF-ELSE statement.

PL/SQL IF-THEN-ELSE Statement

- IF condition THEN
  - sequence_of_if_statements;
- ELSE
  - sequence_of_else_statements;
- END IF;
- If the condition is NULL or false, the sequence of else statements will execute.

PL/SQL IF-THEN-ELSE Statement

```
1.      -- update employee's salary if it is lower than
2.        -- the mid range, otherwise increase 5%
3.        IF n_salary < n_mid_salary THEN
4.          UPDATE employees
5.          SET salary = n_mid_salary
6.          WHERE employee_id = n_emp_id;
7.        ELSE
8.          UPDATE employees
9.          SET salary = salary + salary * 5 /100
10.         WHERE employee_id = n_emp_id;
11.       END IF;
```

PL/SQL IF-THEN-ELSIF Statement

- PL/SQL supports IF-THEN-ELSIF statement to allow you to execute a sequence of statements based on multiple conditions.
- The syntax of PL/SQL IF-THEN-ELSIF is as follows:

PL/SQL IF-THEN-ELSIF Statement

- IF condition1 THEN
  - sequence_of_statements1
- ELSIF condition2 THEN
  - sequence_of_statements2
- ELSE
  - sequence_of_statements3
- END IF;

PL/SQL IF-THEN-ELSIF Statement

- Note that an IF statement can have any number of ELSIF clauses.
- IF the first condition is false or NULL, the ELSIF clause checks second condition and so on.
- If all conditions are NULL or false, the sequence of statements in the ELSE clause will execute.

PL/SQL IF-THEN-ELSIF Statement

- Note that the final ELSE clause is optional so you can omit it.
- If any condition from top to bottom is true, the corresponding sequence of statements will execute.

PL/SQL IF-THEN-ELSIF Statement

```
1.     -- update employee's salary if it is lower than
2.       -- the mid range, otherwise increase 5%
3.       IF n_salary > n_mid_salary THEN
4.         DBMS_OUTPUT.PUT_LINE('Employee ' || TO_CHAR(n_emp_id) ||
5.                              ' has salary $' || TO_CHAR(n_salary) ||
6.                              ' higher than mid-range $' ||
   TO_CHAR(n_mid_salary));
```

PL/SQL IF-THEN-ELSIF Statement

```
1.     ELSIF n_salary < n_mid_salary THEN
2.         DBMS_OUTPUT.PUT_LINE('Employee ' || TO_CHAR(n_emp_id) ||
3.                              ' has salary $' || TO_CHAR(n_salary) ||
4.                              ' lower than mid-range $' ||
   TO_CHAR(n_mid_salary));
```

PL/SQL IF-THEN-ELSIF Statement

```
1.      ELSE
2.          DBMS_OUTPUT.PUT_LINE('Employee ' || TO_CHAR(n_emp_id) ||
3.                               ' has salary $' || TO_CHAR(n_salary) ||
4.                               ' equal to mid-range $' ||
    TO_CHAR(n_mid_salary));
5.        END IF;
6.      END;
```

PL/SQL FOR Loop

- PL/SQL FOR loop is an iterative statement that allows you to execute a sequence of statements a fixed number of times.
- Unlike the PL/SQL WHILE loop, the number of iterations of the PL/SQL FOR loop is known before the loop starts.
- The following illustrates the PL/SQL FOR loop statement syntax:

PL/SQL FOR Loop

- FOR loop_counter IN [REVERSE] lower_bound .. higher_bound
- LOOP
  - sequence_of_statements;
- END LOOP;

PL/SQL FOR Loop

- SET SERVEROUTPUT ON SIZE 1000000;
- DECLARE
  - n_times NUMBER := 10;
- BEGIN
- FOR n_i IN 1..n_times LOOP
  - DBMS_OUTPUT.PUT_LINE(n_i);
- END LOOP;
- END;
- /

PL/SQL FOR Loop

- SET SERVEROUTPUT ON SIZE 1000000;
- DECLARE
  - n_times NUMBER := 10;
- BEGIN
- FOR n_i IN REVERSE 1..n_times LOOP
  - DBMS_OUTPUT.PUT_LINE(n_i);
- END LOOP;
- END;

PL/SQL WHILE Loop

- If you don't know in advance how many times to execute a sequence of statements because the execution depends on a condition that is not fixed.
- In such cases, you should use PL/SQL WHILE loop statement.
- The following illustrates the PL/SQL WHILE LOOP syntax:

PL/SQL WHILE Loop

- WHILE condition
- LOOP
  - sequence_of_statements;
- END LOOP;

PL/SQL CASE Statement

- The PL/SQL CASE statement allows you to execute a sequence of statements based on a selector.
- A selector can be anything such as variable, function, or expression that the CASE statement evaluates to a Boolean value.
- You can use almost any PL/SQL data types as a selector except BLOB, BFILE and composite types.
- syntax:

- Unlike the PL/SQL IF statement, PL/SQL CASE statement uses a selector instead of combination of multiple Boolean expressions.
- The following illustrates the PL/SQL CASE statement

```
1.    [<<label_name>>]
2.    CASE [TRUE | selector]
3.       WHEN expression1 THEN
4.            sequence_of_statements1;
5.       WHEN expression2 THEN
6.            sequence_of_statements2;
7.       ...
8.       WHEN expressionN THEN
9.            sequence_of_statementsN;
10.      [ELSE sequence_of_statementsN+1;]
11.   END CASE [label_name];
```

- Followed by the keyword CASE is a selector.
- The PL/SQL CASE statement evaluates the selector only once to decide which sequence of statements to execute.
- Followed by the selector is any number of the WHEN clause.
- If the selector value is equal to *expression* in the WHEN clause,
- the corresponding sequence of statement after the THEN keyword will be executed.

- If the selector's value is not one of the choices covered by WHEN clause,
- the sequence of statements in the ELSE clause is executed.
- The ELSE clause is optional so if you omit the ESLE clause, PL/SQL will add the following implicit ELSE clause:
  - ELSE RAISE CASE_NOT_FOUND;
- The keywords END CASE are used to terminate the CASE statement.

Example of Using PL/SQL CASE Statement

- The following code snippet demonstrates the PL/SQL CASE statement. We'll use table *employees* for demonstration

```
1.    SET SERVEROUTPUT ON SIZE 1000000;
2.    DECLARE
3.      n_pct    employees.commission_pct%TYPE;
4.      v_eval   VARCHAR2(10);
5.      n_emp_id employees.employee_id%TYPE := 145;
6.    BEGIN
7.      -- get commission percentage
8.      SELECT commission_pct
9.      INTO n_pct
10.     FROM employees
11.     WHERE employee_id = n_emp_id;
```

```
1.    -- evalutate commission percentage
2.      CASE n_pct
3.        WHEN 0 THEN
4.          v_eval := 'N/A';
5.        WHEN 0.1 THEN
6.          v_eval := 'Low';
7.        WHEN 0.4 THEN
8.          v_eval := 'High';
9.        ELSE
10.         v_eval := 'Fair';
11.     END CASE;
12.     -- print commission evaluation
13.     DBMS_OUTPUT.PUT_LINE('Employee ' || n_emp_id ||
14.                          ' commission ' || TO_CHAR(n_pct) ||
15.                          ' which is '   || v_eval);
16.   END;
```

PL/SQL LOOP Statement

- PL/SQL LOOP is an iterative control structure that allows you to execute a sequence of statements repeatedly.
- The simplest of LOOP consists of
  - the LOOP keyword,
  - the sequence of statements and
  - the END LOOP keywords

```
1.    LOOP
2.      sequence_of_statements;
3.    END LOOP;
```

- Note that there must be at least one executable statement between LOOP and END LOOP keywords.
- The sequence of statements is executed repeatedly until it reaches a loop exits.
- PL/SQL provides you EXIT and EXIT-WHEN statements to allow you to terminate a loop.

- The EXIT forces the loop halt execution unconditionally and passes control to the next statement after keyword END LOOP.
- The EXIT-WHEN statement allows the loop complete conditionally.
- When the EXIT-WHEN statement is reached, the condition in the WHEN clause is checked.

- If the condition is true, the loop is terminated and pass control to the next statement after keyword END LOOP.
- If condition is false, the loop will continue repeatedly until the condition is evaluated to true.
- Therefore if you don't want to have a infinite loop you must change variable's value inside loop to make condition true.

- The following illustrates PL/SQL LOOP with EXIT and EXIT-WHEN statements:

```
1.    LOOP
2.      ...
3.      EXIT;
4.    END LOOP;
```

```
1.    LOOP
2.      ...
3.      EXIT WHEN condition;
4.    END LOOP;
```

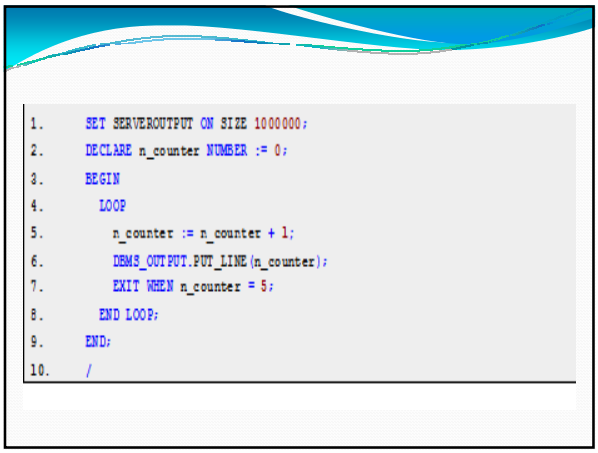
Example of PL/SQL LOOP with EXIT Statement

- In this example, we declare a counter. Inside the loop we add 1 to the counter and print it out.
- If the counter is 5, we use EXIT statement to terminate the loop.
- Below is the code example of PL/SQL LOOP statement with EXIT:

```
1.    SET SERVEROUTPUT ON SIZE 1000000;
2.    DECLARE n_counter NUMBER := 0;
3.    BEGIN
4.      LOOP
5.        n_counter := n_counter + 1;
6.        DBMS_OUTPUT.PUT_LINE(n_counter);
7.        IF n_counter = 5 THEN
8.          EXIT;
9.        END IF;
10.     END LOOP;
11.   END;
12.   /
```

Example of PL/SQL LOOP with EXIT-WHEN Statement

- We'll use the same counter example above. However instead of using the IF-THEN and EXIT statements,
- we use EXIT-WHEN to terminate the loop.
- The code example is as follows:



```
1.    SET SERVEROUTPUT ON SIZE 1000000;
2.    DECLARE n_counter NUMBER := 0;
3.    BEGIN
4.      LOOP
5.        n_counter := n_counter + 1;
6.        DBMS_OUTPUT.PUT_LINE(n_counter);
7.        EXIT WHEN n_counter = 5;
8.      END LOOP;
9.    END;
10.   /
```