Built-in Functions

PL/SQL provide many powerful functions to enable easy data manipulation. The built-in-functions fall into the following categories:

1) Error reporting functions
2) Single row functions
3) Single row character functions
4) Datatype conversion functions
5) Date functions
6) Miscellaneous functions

Except **error reporting functions,** all built-in functions are available in SQL statements. The SQL Group functions (AVG,MAX,MIN,SUM,COUNT) are not valid in PL/SQL block as they apply to groups of rows in a table, and are therefore available only within SQL statements.

Error Reporting Functions :- It gives information about PL/SQL execution errors. These functions are used for error trapping. These functions are not allowed in SQL statements. These functions are :-

	Function	**Description**
1)	SQLCODE	returns the numeric value for the error code
2)	SQLERRM	returns character data containing the message associated with the error number.

Conditional and iterative control

As the name implies, PL/SQL supports programming language features like conditional statements, iterative statements. Pl/SQL control structure allows you to control the behaviour of the block as it runs. You can change the logical flow of statements within the PL/SQL block with a number of control statements.

Four types of PL/SQL control structures are

1) Conditional constructs with the IF statement
2) CASE expressions
3) LOOP Control structures
4) Sequential Control

Conditional Constructs with the If statement:-

Structure of the PL?SQL IF statements is similar to the structure of IF statements in other procedural languages. It allows PL?SQL to perform actions selectively based on conditions.

There are three forms of IF statement :-

(a) IF-THEN-END IF

Syntax :-

```
        IF condition THEN
                statements;
        END IF;
```

EXAMPLE :-

```
DECLARE
        A NUMBER;
        B NUMBER;
BEGIN
        A:=10;
        B:=5;
        IF A>B THEN
                dbms_Output.Put_Line('A is BIGGER THAN B');
        END IF;
END;
/
```

(b) IF-THEN-ELSE-END IF

Syntax :-

```
IF condition THEN
        Statements;
ELSE
        Statements;
END IF;
```

Example :-

```
DECLARE
        A NUMBER;
        B NUMBER;
BEGIN
        A:=22;
        B:=33;
        IF A>B THEN
                dbms_output.put_line('A is bigger than b');
ELSE
```

```
          dbms_output.put_line(‘B is bigger than A’);
     END IF;
END;
/
```

© IF-THEN-ELSEIF-END IF
Syntax :-

```
IF condition THEN
     Statements;
[ELSEIF condition THEN
     Statements;]
[ELSE
     Statements;]
END IF;
```

EXAMPLE:-

```
DECLARE
     A NUMBER;
     B NUMBER;
     C NUMBER;
BEGIN
     A:=10;
     B:=20;
     C:=30;
     IF A>B THEN
          IF A>C THEN
               DBMS_OUTPUT.PUT_LINE(A||’ is biggest’);
          ELSE
               DBMS_OUTPUT.PUT_LINE(C|| ‘ IS BIGGEST’);
          END IF;
          ELSEIF B>C THEN
               dbms_output.put_line(B||’ is biggest’);
          ELSE
          dbms_output.put_line(C||’ is biggest’);
     END IF;
END;
/
```

CASE EXPRESSIONS

A CASE expression selects a result and returns it. To select the result, the CASE expression uses a selector, an expression whose value is used to select one of several alternatives. The selector is followed by one or more WHEN clauses, which are checked sequentially. The value of the selector determines which clause is executed. If the value of the selector equals the value of a WHEN-clause expression, that WHEN clause is executed.
PL/SQL also provides a searched CASE expression, which has the form:

```
CASE
        WHEN search_condition1 THEN result1
        WHEN search_condition2 THEN result2
        ……
        WHEN  search_conditionN THEN resultN
        [ELSE resultN+1;]
END;
```

Example :-

LOOP CONTROL STRUCTURES

Iterative control statements or LOOP control structures are used when we want to repeat the execution of one or more statements for specified number of times. PL/SQL provides the following types of loops:

(1) Basic loop/Simple loop
(2) WHILe loop
(3) FOR loop

(i) Basic loop/Simple loop :- The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP. each time the flow of execution reaches the END LOOP statement, control is returned to the corresponding LOOP statement above it. A basic loop allows execution of its statement at least once, even if the condition is already met upon entering the loop. Without EXIT statement, the loop would be infinite or endless.

(a) The EXIT Statement

You can use the exit statement to terminate a loop. Control passes to the next statement after the END LOOP statement. You can issue EXIT either as an action within an IF statement or as a stand-alone statement within the loop. The EXIT statement must be placed inside a loop.

Syntax :-

```
LOOP
Statements;
……..
EXIT [WHEN condition];
END LOOP;
```

Example :-

```
Declare
          N number;
          F number;
          I number;
BEGIN
          F:= 1;
          I:= 1;
          N:= &Number;
   LOOP
          IF I>N THEN
                    EXIT;
      Else
          F:=F*I;
          I:= I+1;
      END IF;
  END LOOP;
  Dbms_output.Put_Line (' factorial of ' || N ||' is: ' ||F);
END;
/
```

(b) WHILE LOOP

You can use the WHILE loop to repeat a sequence of statements until the controlling condition is TRUE. The condition is evaluated at the start of each iteration. The loop terminates when condition is FALSE. If the condition is FALSE at the start of the loop, then no further iterations are performed.

Syantax :-

```
WHILE condition LOOP
Statement1;
Statement2;
……..
END LOOP;
```

Example :-

```
 DECLARE
          N number;
          F number;
          I number;
 BEGIN
          F:=1;
          I:=1;
          N:=&Number;
          While (I<=N)
          LOOP
                    F:= F*I;
                    I:=I+1;
          END LOOP;
     Dbms_Output.put_line ( ' Factorial of ' || N || '  IS : ' || F);
 END;
 /
```

© FOR LOOP

FOR loops have the same general structure as the basic loop. In addition, they have a control statement before the LOOP keyword to determine the number of iterations the PL/SQL performs.

Syntax :-

```
FOR counter IN [REVERSE]
          <lower_bound>..<super_bound> LOOP
Statement1;
Statement2;
…….
END LOOP
;
```

Example :-

```
 DECLARE
            N NUMBER;
            F NUMBER;
 BEGIN
                 F:=1;
                 N:= &NUMBER;
        FOR I IN 1..N
                 LOOP
                 F:=F*I;
        END LOOP;
                 dbms_output.Put_line (' Factorial of '||N||' is: ' ||F);
 END;
 /
```

In the syntax :-

(a) counter :- is an implicitly declared integer whose value automatically increases or decreases by 1 on each iteration of the loop until the upper or lower bound is reached.

(b) REVERSE :- causes the counter to decrement with each iteration from the upper bound to lower bound.

© lower_bound :- specifies the lower bound for the range of counter values.

(d) upper_bound:- specifies the upper bound for the range of counter values.

SEQUENTIAL CONTROL

By default, all PL/SQL blocks are executed in a top down sequential process.
The process begins with a BEGIN statement and terminates with an END statement.
So, to change the sequence of execution of statements, you can use, following unconditional statements:

1. **GOTO Statement**

   The PL/SQL GOTO statement is a sequence control structure available in oracle. The GOTO statement immediately transfers program control(called branching) unconditionally to a named statement label or block label. The statement or label name must be unique in the block.

   Syntax:
   GOTO << LABEL_NAME>>;
   Where LABLE_NAME is the name of the label identifying the target statements.

Example :-

```
BEGIN
dbms_output.Put_Line (' This is the first line.');
GOTO FOUR;
Dbms_Output.Put_Line (' This is the Second Line.');
dbms_output.Put_Line (' this is the Third Line.');
<<FOUR>>
Dbms_Output.Put_Line (' this is the FOURTH LINE.');
END;
/
```

2. **NULL Statement**
Generally when you write a statement in a program, you want it to do something, but in some cases, you want to tell PL/SQL to do nothing and in such cases, NULL statement can be used. The NULL statement does nothing other than pass control to the next statement.

Syntax:
NULL;

Example :-

```
DECLARE
        A NUMBER;
BEGIN
      A:=&A;
   IF A MOD 2!=0 THEN
        DBMS_OUTPUT.PUT_LINE (' NUMBER IS ODD');
   ELSE
      NULL;
   END IF;
END;
/
```

NOTE :- Dbms_Output.Put_Line() is a built-in PL/SQL procedure that will produce the output on the screen. It accepts only one argument. Hence , the different variables are concatenated with double pipe(||) symbol. To enable the server output, the SET SERVEROUTPUT ON command must be given at the

SQL* Plus Prompt, prior to the execution of the Dbms_output.Put_Line() procedure.

SQL WITHIN PL/SQL

Whenever any sort of data manipulation has to be done using database, SQL commands are used. The only SQL commands allowed in PL/SQL program are DML (Data Manipulation Language) and tCL (transaction control language) commands.

Points to remember while using SQL statements within PL/SQL;

(1) DDL statements are illegal in PL/SQL.

(2) SELECT statements which do not return a single row will cause an exception to be raised.

(3) DML commands can process multiple rows.

DQL and DML in PL/SQL

The allowable DQL and DML statements are SELECT, INSERT, UPDATE and DELETE. These statements are used without any restriction in PL/SQL.

When SELECT statements are used,queries must return one, and only one row, otherwise an error will be generated.therefore, SELECT statements which return no rows or more than one row will cause errors.

THE INTO CLAUSE

The INTO clause is used with SELECT, to store values from the table into variables. The INTO clause occurs between the SELECT and FROM clauses. This clause specifies the names of variables that will be populated by the items being selected in the SELECT clause. For each item selected a separate variable must be used, and their order corresponds to the items selected.

WRITING PL/SQL CODE

PL/SQL code is written using any text editor like NOTEPAD. The PL/SQL program is compiled and executed using the command **@<filename>** or **START<filename>.**

COMPOSITE DATATYPES: