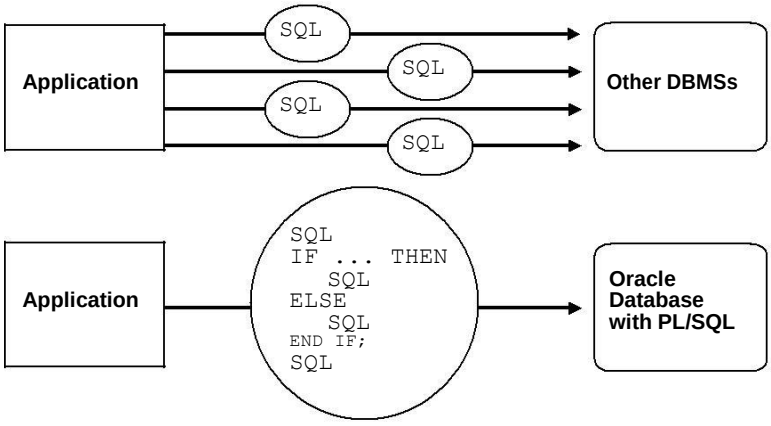
Why PL/SQL?

Better Performance

Without PL/SQL, Oracle must process SQL statements one at a time. Programs that issue many SQL statements require multiple calls to the database, resulting in significant network and performance overhead.

With PL/SQL, an entire block of statements can be sent to Oracle at one time. This can drastically reduce network traffic between the database and an application. As in the following Figure, you can use PL/SQL blocks and subprograms to group SQL statements before sending them to the database for execution. PL/SQL also has language features to further speed up SQL statements that are issued inside a loop.



PL SQL Block Structure

```
[DECLARE
 --  declarations]
 --
 BEGIN
 --  statements
 --
 [EXCEPTION
 --
 --  handlers]
 END;
```

Declaring Variables

Variables are declared in DECLARE secion. Variables can have any SQL datatype, such as `CHAR`, `DATE`, or `NUMBER`, or a PL/SQL-only datatype, such as `BOOLEAN`.

Example:

```
Employee_number number(3);
Employee_name varchar2(10);
Employee_salary number(9,2);
```

Declaring constant

```
Comm_limit CONSTANT number := 5000.00;
```

Declaring Boolean variable

pass_fail Boolean;

Declaring Date Variable

Emp_Birth_Date DATE;

Declaring Datatypes for PL/SQL Variables

%TYPE

The **%TYPE** attribute provides the datatype of a variable or database column. This is particularly useful when declaring variables that will hold column values. For example, assume that there is a column named Ename in a table EMP and we have to declare variable **V_Ename** to hold value of **Ename** column. Rather than finding data type and width of **Ename** column and declaring variable, we can use following type declaration-

V_Ename EMP.Ename%Type;

PL/SQL Control Structures

Different forms of IF-THEN Statement

(i) **IF** *condition* **THEN**

.....

END IF;

Example:

```
DECLARE
        sales NUMBER(8,2) := 10100;
        Discount number (5,2);
BEGIN
        IF sales > 10000 THEN
            Discount=sales * 0.3;
        END IF;
END;
```

(ii) **IF** *condition* **THEN**

.....

ELSE

```
                ..........................
        END IF;
```

The statements in the ELSE clause are executed only if the condition is FALSE or NULL.

Example:

```
        DECLARE
                sales NUMBER(8,2) := 10100;
                Discount number(5,2);
        BEGIN
                IF sales > 10000 THEN
                    Discount=sales * 0.3;
                ELSE
                    Discount=sales * 0.2;
                END IF;
       END;
```

(iii) **IF** *condition* **THEN**

```
                ..........................
        ELSIF condition

                ..........................
        ELSE

                ..........................
        END IF;
```

Example:

```
        DECLARE
                sales NUMBER(8,2) := 10100;
                Discount NUMBER (5,2);
        BEGIN
                IF sales > 10000 THEN
                    Discount=sales * 0.3;
                ELSIF sales > 5000 THEN
                    Discount=sales * 0.2;
                ELSIF sales > 3000 THEN
                    Discount=sales * 0.1;
                ELSE
                    Discount=0;
                END IF;
       END;
```

You can use IF .. ELSE .. END IF; statements in nested form also.

Printing on the screen

DBMS_OUTPUT is the package , using PUT_LINE(string variable) , string variable value can be displayed on the screen.

Before using DBMS_OUTPUT.PUT_LINE(..) , use SET SERVEROUTPUT ON at SQL prompt.

Example:

```
DBMS_OUTPUT.PUT_LINE(' HELLO ….');
```

DBMS_OUTPUT.PUT_LINE('MY Register Number ' || to_char(123456));

|| symbol concatenates two strings.

CASE Statements

Like the IF statement, the CASE statement selects one sequence of statements to execute. However, to select the sequence, the CASE statement uses a selector rather than multiple Boolean expressions.

Example:

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := & grade;
   CASE grade
    WHEN 'A' THEN DBMS_OUTPUT.PUT_LINE('Excellent');
    WHEN 'B' THEN DBMS_OUTPUT.PUT_LINE('Very Good');
    WHEN 'C' THEN DBMS_OUTPUT.PUT_LINE('Good');
    WHEN 'D' THEN DBMS_OUTPUT.PUT_LINE('Fair');
    WHEN 'F' THEN DBMS_OUTPUT.PUT_LINE('Poor'); ELSE
    DBMS_OUTPUT.PUT_LINE('No such grade');
  END CASE;
END;
/
```

These CASE and WHEN statements are equivalent to series of IF statements.

LOOP Statement

The simplest form of LOOP statement is the basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP, as follows:

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

This will be an infinite loop, therefore exit condition need to be specified using EXIT or EXIT WHEN command.

Example: Counting numbers from 1 to 10 and exiting from loop when I value becomes more than 10.

```
DECLARE
        I number(2);

BEGIN
        I:=1;
        LOOP
                DBMS_OUTPUT.PUT_LINE(" Counter "||I);
                I:=I+1;
                IF I > 10  THEN
                        EXIT;
                END IF:
         END LOOP;
END;
/
```

Same program can be written using EXIT WHEN as below

Example

```
DECLARE
        I number(2);
```

```
BEGIN
        I:=1;
        LOOP
                DBMS_OUTPUT.PUT_LINE(" Counter "||I);
                I:=I+1;
                EXIT WHEN I>10;
         END LOOP;
END;
/
```

WHILE-LOOP Statement

The WHILE-LOOP statement executes the statements in the loop body as long as a condition is true:

```
WHILE condition LOOP
        ……..
        sequence_of_statements
        ……..
END LOOP;
```

Example

```
DECLARE
        I number(2);

BEGIN
        I:=1;
        WHILE I<11 LOOP
                DBMS_OUTPUT.PUT_LINE ("Counter "||to_char(I));
                I:=I+1;
        END LOOP;
END;
/
```

FOR-LOOP Statement

```
FOR counter IN initial_value .. final_value
 LOOP
   ……..
   sequence_of_statements;
  ……..
 END LOOP;
```

Example- Print numbers starting from 10 to 20.

```
DECLARE
        a number(2);
BEGIN
        FOR a in 10 .. 20
        LOOP
                dbms_output.put_line ('value of a: ' || a);
        END LOOP;
END;
/
```

Reverse FOR LOOP

By default, iteration proceeds from the initial value to the final value, generally upward from the lower bound to the higher bound. You can reverse this order by using the REVERSE keyword. In such case, iteration proceeds the other way. After each iteration, the loop counter is decremented.

However, you must write the range bounds in ascending (not descending) order.

```
FOR counter IN REVERSE initial_value .. final_value
 LOOP
    ……
    sequence_of_statements;
  ……..
 END LOOP;
```

Example- Print numbers starting from 20 to 10.

```
        DECLARE
                a number(2);
        BEGIN
                FOR a IN REVERSE 10 .. 20
                LOOP
                        dbms_output.put_line ('value of a: ' || a);
                END LOOP;
        END;
/
```

Example: Check a given string is palindrome or not.

```
DECLARE
                Given_String varchar2(5);
                cnt number(2);
                Reverse_string varchar(5);

        BEGIN
                Given_String:='& Given_String';
                cnt:=length(Given_String);
                while cnt>0
                loop
                        Reverse_string :=Reverse_string || substr(Given_String,cnt,1);
                        cnt:=cnt-1;
                end loop;

                DBMS_OUTPUT.PUT_LINE   ('given string is  '|| Given_String);
                DBMS_OUTPUT.PUT_LINE   ('inverted number is : '|| Reverse_string);
        END;
/
```

SELECTING Columns Value Into Variables

```
SELECT Column1,Column2, .. INTO Variabl1, Variabl2,.. FROM table..  ;
```

Example: Display employee name and salary of a given employee from EMP table. Assume empno,ename,sal,deptno are the columns in EMP table.

```
 DECLARE
      V_empno EMP.Empno%type;
      V_Ename EMP.Ename%type;
      V_Sal  EMP.sal%type;
BEGIN
      V_empno:=&V_empno;
      SELECT Ename,Sal INTO V_Ename , V_Sal  FROM EMP WHERE Empno= V_empno ;
```

```
      DBMS_OUTPUT.PUT_LINE('Employee Name '||V_Ename||'---'||'Salary '||
      to_char(V_Sal));
END;
/
```

Example: Assume that there is a table **STUDENT** (Regno, Name, Marks1, Mark2,Total). Accept a register number and display marks and total. Update Total of corresponding STUDENT.

```
SET SERVEROUTPUT ON;
DECLARE
      V_Regno STUDENT.Regno%type;
      V_Mark1 STUDENT.Mark1%type;
      V_Mark2  STUDENT.Mark2%type;
      V_Total Number(3);
BEGIN
      V_Regno:=&V_Regno;
      SELECT Mark1,Mark2 INTO V_Mark1, V_Mark2 FROM EMP WHERE Regno= V_Regno;
      V_Total:= V_Mark1+ V_Mark1;
      UPDATE STUDENT SET Total=V_Total WHERE Regno= V_Regno;
      DBMS_OUTPUT.PUT_LINE('Mark1 '||to_char(V_Mark1)||'---'||'Mark2
      '||to_char(V_Mark1)||'---'||'Total '||to_char(V_Total));
END;
/
```