DBMS
Practical #7

What is PL/SQL?

As the name suggests, PL/SQL is a **superset** of SQL.

PL/SQL is a block-structured language that enables developers to combine the power of SQL with procedural statements.

PL/SQL bridges the gap between database technology and procedural programming languages.

PL/SQL Block Structure

Syntax of PL/SQL Block Structure:

```
DECLARE        --optional
               <declarations>
               .
               .
               .
BEGIN          --mandatory
               <executable statements. At least one executable statement is mandatory
               .
               .
               .
EXCEPTION      --optional
               <exception handler>
               .
               .
               .
END;           --mandatory
/
```

Declaring Variables

```
a int;
```

```
sum number(4);
```

```
In_stock boolean;
```

Assigning Values to a variable

Three ways to assign values to a variable

- Using Assignment Operator(:=)
- Eg. a int := 10;
- bonus := empsal*0.10;
- 2. The second way to assign values to a variable is by selecting database values into it.
- select empsal * 0.10 into bonus from emp where eid=2;
- 3. Using IN and OUT Parameter

Basic Examples

Display values of variables using PLSQL block.

```
--
Set Serveroutput on
declare
   i int:=10;
   j int:=20;
begin
 DBMS_OUTPUT.PUT_LINE('value of i=' || (i));
 DBMS_OUTPUT.PUT_LINE('value of j=' || (j));
end;
/
```

Display addition of two numbers using PLSQL block.

```
--
Set Serveroutput on
declare
   i int:= 5;
   j int:=10;
   k int:= i + j;
begin
DBMS_OUTPUT.PUT_LINE('value of k=' || (k));
end;
/
```

Performing Arithmetic Operations accepting value from user

```
--
set serveroutput on
declare
 i int;  j int; k int;
begin
i:=&i;
j:=&j;
 k:=i + j;  DBMS_OUTPUT.PUT_LINE('Addition value of k=' || (k));
 k:= i - j;  DBMS_OUTPUT.PUT_LINE('Subtraction value of k=' || (k));
 k:= i *j;  DBMS_OUTPUT.PUT_LINE('Multiplication value of k=' || (k));
 k := i/j;  DBMS_OUTPUT.PUT_LINE('Division value of k=' || (k));
end;
```

IF Statements

- The structure of the IF statement is similar to the structure of IF statement in other languages
- It allows PL/SQL to perform actions selectively based on conditions. The syntax of IF statement is

```
IF <condition> THEN
        statements;
[ELSIF <condition> THEN
        statements;]
[ELSE
        statements;]
END IF;
```

- condition is the Boolean expression or variable that returns TRUE, FALSE or NULL
- THEN introduces a clause that associates the Boolean expression with the sequence of statements that follows it
- Statements can be one or more PL/SQL or SQL statements. The statements in the THEN clause are executed only if the condition in the associated IF clause evaluates to TRUE

IF Statements Contd..

- ELSIF Is a keyword that introduces a Boolean expression
- ELSE Introduces a default clause that is executed if and only if none of the earlier conditions are TRUE
- END IF Marks the end of the IF statement
- ELSIF and ELSE are optional in an IF statement.
- You can have any number of ELSIF keywords but only one ELSE keyword in IF statement
- END IF marks the end of an IF statement and must be terminated by semicolon

IF Example – Find the smallest number

Direct/Static Input

--

```
declare
a int:=5;
b int:=6;
begin
if a < b then
DBMS_OUTPUT.PUT_LINE(a);
end if;
end;
```

User Input

--

```
Set serveroutput on
declare
a int;
b int;
begin
a:=&a;
b:=&b;
if a < b then
DBMS_OUTPUT.PUT_LINE(a);
else
DBMS_OUTPUT.PUT_LINE(b);
end if;
end;
```

IF ELSE Example- Find the number is even or odd

```
Set serveroutput on
DECLARE
   NO NUMBER;
BEGIN
   NO:=&no;
   IF MOD(NO,2) = 0 THEN
      DBMS_OUTPUT.PUT_LINE('EVEN NUMBER');
   ELSE
      DBMS_OUTPUT.PUT_LINE('ODD NUMBER');
   END IF;
END;
```

IF-ELSIF-ELSE Check number is positive or negative

```
Set serveroutput on
DECLARE
   NO NUMBER;
BEGIN
   NO:=&NO;
   IF NO < 0 THEN
      DBMS_OUTPUT.PUT_LINE('NEGATIVE NUMBER');
   ELSIF NO > 0 THEN
      DBMS_OUTPUT.PUT_LINE('POSITIVE NUMBER');
   ELSE
      DBMS_OUTPUT.PUT_LINE('EQUAL TO ZERO');
   END IF;
END;
```

IF ELSE EXAMPLE –Find the grade of Student

```
--
Set serveroutput on
declare
per number;
begin
per:=&per;
if per>=75 then
DBMS_OUTPUT.PUT_LINE('Your grade: A');
elsif per>=60 then
DBMS_OUTPUT.PUT_LINE('Your grade: B');
elsif per>=50 then
DBMS_OUTPUT.PUT_LINE('Your grade: C');
elsif per>=40 then
DBMS_OUTPUT.PUT_LINE('Your grade: D');
else
DBMS_OUTPUT.PUT_LINE('Your grade: F');
end if;
end;
```

CASE Expressions

- A CASE expression returns a result based on one or more alternatives.
- To return the result, the CASE expression uses a selector which is an expression whose value is used to return one of the several alternatives
- The selector is followed by one or more WHEN clauses that are checked sequentially
- The value of the selector determines which result is returned
- If the value of the selector equals the value of a WHEN clause expression that WHEN clause is executed, and the result is returned. The syntax of CASE expression is

```
CASE selector
        WHEN expression1 THEN result1
        WHEN expression2 THEN result2
        ….
        WHEN expressionN THEN resultN
        [ELSE resultN+1]
END;
```

Searched Case Expression

- PL/SQL also provides a searched CASE expression which has the following form

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

- A searched case expression has no selector
- Its WHEN clause contains search conditions that yield a Boolean value rather than expression that can yield a value of any type
- In searched CASE statements you do not have to test expression . Instead the WHEN clause contains an expression that results in a Boolean value

Case Expression example

```
--
Set serveroutput on
declare
per number;
Begin
per:=&per;
case per
when 8 then DBMS_OUTPUT.PUT_LINE('Your grade: A');
when 7 then DBMS_OUTPUT.PUT_LINE('Your grade: B');
when 6 then DBMS_OUTPUT.PUT_LINE('Your grade: C');
when 5 then DBMS_OUTPUT.PUT_LINE('Your grade: D');
else DBMS_OUTPUT.PUT_LINE('Your grade: F');
end case;
end;
```

Iterative Control : Loop Statements

- Loops are mainly used to execute statements or a sequence of instructions multiple times repeatedly until an exit condition occurs
- It is mandatory to have an exit condition in a loop else it becomes infinite
- Looping constructs are the second type of control structure
- PL/SQL provides the following types of loops
  - Basic loop that performs repetitive actions without overall conditions
  - FOR loops that perform iterative actions based on a count
  - WHILE loops that perform iterative action based on a condition

Iterative control

Simple loop

FOR Loop

The WHILE loop

Create a simple loop such that a message is displayed when a loop exceeds a particular value.

--

```
Set serveroutput on
declare
i number:=0;
begin
Loop
i:=i+2;
exit when i>10;
end loop;
DBMS_OUTPUT.PUT_LINE('the value of i has reached '|| (i));
End;
```

Basic Loop Examples

Print number between 1 to 10

```
--
DECLARE
        CTR NUMBER:=1;
BEGIN
        LOOP

        DBMS_OUTPUT.PUT_LINE(CTR);
                CTR:=CTR+1;
                EXIT WHEN CTR>10;
        END LOOP;
END;
```

Print factorial of a number using Loop.

```
--
DECLARE
        FACT NUMBER:=1;
        NO NUMBER:=&NO;
BEGIN
        LOOP
                FACT :=FACT * NO;
                NO:=NO-1;
                EXIT WHEN NO<1;
        END LOOP;
DBMS_OUTPUT.PUT_LINE(FACT);
END;
```

While Loops

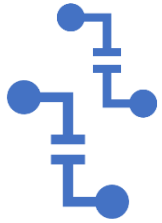
- WHILE loop can be used to repeat a sequence of statements until the controlling condition is TRUE.
- The condition is evaluated at the start of each iteration and if it evaluates to FALSE the loop terminates
- If the condition is FALSE at the beginning of the loop the loop may not execute at all
- The syntax for the WHILE loop is

```
WHILE <condition>
LOOP
        statements
END LOOP;
```

While Loop Example

- Display 1 to 10 using while loop

```
--
declare
a int:= 0;
begin
while a < 10
loop
a := a + 1;
DBMS_OUTPUT.PUT_LINE( a);
end loop;
end;
/
```



FOR Loops

- FOR loops have a control statement before the loop keyword to set the number of iterations that PL/SQL performs
- It has the same structure as that of the basic loop
- The syntax of the FOR loop is

```
FOR <variable> IN [REVERSE] lower_bound..upper_bound
LOOP
        statements;
END LOOP;
```

- In the above syntax the variable for the loop need not be declared as it is declared implicitly as integer
- The lower bound must be always less than upper bound
- REVERSE causes the loop to decrement with each iteration from the upper bound to lower bound

For Loop Examples

- Display 1 to 10 using for loop

```
--
begin
for i in 1..10
loop
dbms_output.put_line (i);
end loop;
end;
/
```

- Display 1 to 10 in descending order using for loop

```
--
begin
for i in reverse 1..10
loop
dbms_output.put_line (i);
end loop;
end;
/
```

Thank You