Introduction

(Procedural Language / Structured Query Language)
PL/SQL is a block structured language that enables developers to combine the power of SQL with procedural statements.
All the statements of a block are passed to oracle engine all at once which increases processing speed and decreases the traffic.

Features of PL/SQL

PL/SQL has the following features −

- PL/SQL is tightly integrated with SQL.
- It offers extensive error checking.
- It offers numerous data types.
- It offers a variety of programming structures.
- It supports structured programming through functions and procedures.
- It supports object-oriented programming.
- It supports the development of web applications and server pages.

Advantages of PL/SQL

PL/SQL has the following advantages :-

1) PL/SQL allows sending an entire block of statements to the database at one time.
   This reduces network traffic and provides high performance for the applications.
2) PL/SQL gives high productivity to programmers as it can query, transform, and update data in a database.
3) PL/SQL saves time on design and debugging by strong features, such as exception handling, encapsulation, data hiding, and object-oriented data types.
4) Applications written in PL/SQL are fully portable.
5) PL/SQL provides high security level.
6) PL/SQL provides access to predefined SQL packages.
7) PL/SQL provides support for Object-Oriented Programming.
8) PL/SQL provides support for developing Web Applications and Server Pages.

TOPICS TO BE COVERED

1) BASIC
2) IF ELSE
3) CASE
4) LOOPS
5) STRINGS
6) ARRAYS
7) DATE AND TIME
8) TABLE CREATION
9) QUERIES
10) CLAUSES
11) OPERATORS
12) JOINS
13) FUNCTIONS

BASIC QUESTIONS

9) TO Perform Arithmetical Operation
```
DECLARE
            A NUMBER;
            B NUMBER;
            C NUMBER;
            D NUMBER;
            E NUMBER;
            F NUMBER;
BEGIN
            A:=&A;
            B:=&B;
            C := A+B;
            D := A-B;
            E := A*B;
            F := A/B;
            DBMS_OUTPUT.PUT_LINE('ADDITION : ' || C );
            DBMS_OUTPUT.PUT_LINE('SUBTRACTION : '|| D);
            DBMS_OUTPUT.PUT_LINE('MULTIPLICATION : '|| E);
            DBMS_OUTPUT.PUT_LINE('DIVISION : ' || F);
END;
/
```

Q2) TO Find Square and Cube of a Number;
```
DECLARE
            A NUMBER;
            SQ NUMBER;
            CU NUMBER;
BEGIN
            A:= &A;
            SQ:=A*A;
            CU:=SQ*A;
            DBMS_OUTPUT.PUT_LINE('SQUARE : ' || SQ );
            DBMS_OUTPUT.PUT_LINE('CUBE : '|| CU);
END;
/
```

Q3)TO Find Area of Square , Triangle , Rectange and Circle
```
DECLARE
            LENGTH NUMBER;
            BREADTH NUMBER;
            RADIUS NUMBER;
            BASE NUMBER;
            HEIGHT NUMBER;
            SIDE NUMBER;

            AREA_RECTANGLE NUMBER;
            AREA_TRIANGLE FLOAT;
            AREA_SQUARE NUMBER;
            AREA_CIRCLE FLOAT;
```

```
      BEGIN

                  LENGTH:=&LENGTH;
                  BREADTH:=&BREADTH;
                  AREA_RECTANGLE := LENGTH+BREADTH;

                  BASE:=&BASE;
                  HEIGHT:=&HEIGHT;
                  AREA_TRIANGLE:=0.5*BASE*HEIGHT;

                  SIDE:=&SIDE;
                  AREA_SQUARE:=SIDE*SIDE;

                  RADIUS:=&RADIUS;
                  AREA_CIRCLE:=3.142*RADIUS*RADIUS;

                  DBMS_OUTPUT.PUT_LINE('AREA OF RECTANGLE : ' ||
AREA_RECTANGLE);
                  DBMS_OUTPUT.PUT_LINE('AREA OF TRIANGLE :  ' ||
AREA_TRIANGLE);
                  DBMS_OUTPUT.PUT_LINE('AREA OF SQUARE :    ' ||
AREA_SQUARE);
                  DBMS_OUTPUT.PUT_LINE('AREA OF CIRCLE :    ' ||
AREA_CIRCLE);
      END;
      /
```

Q4)SWAPPING BETWEEN 2 NUMBERS;

```
      DECLARE
                  A NUMBER;
                  B NUMBER;
                  C NUMBER;
      BEGIN
                  A:=&A;
                  B:=&B;

                  DBMS_OUTPUT.PUT_LINE('BEFORE SWAPPING ');
                  DBMS_OUTPUT.PUT_LINE('A : ' || A );
                  DBMS_OUTPUT.PUT_LINE('B : ' || B );

                  C := A;
                  A := B;
                  B := C;
                  DBMS_OUTPUT.PUT_LINE('AFTER SWAPPING ');
                  DBMS_OUTPUT.PUT_LINE('A : ' || A );
                  DBMS_OUTPUT.PUT_LINE('B : ' || B );
      END;
      /
```

===
=============
IF ELSE
===
=============
Q1) Input week number(1-7) and print the corresponding day of week name

```
      DECLARE
                  OPT NUMBER;
      BEGIN
                  OPT := &OPT;
                  IF (OPT = 1) THEN
                              DBMS_OUTPUT.PUT_LINE('MONDAY');
                  ELSIF (OPT = 2) THEN
                              DBMS_OUTPUT.PUT_LINE('TUESDAY');
                  ELSIF (OPT = 3) THEN
                              DBMS_OUTPUT.PUT_LINE('WEDNESDAY');
                  ELSIF (OPT = 4) THEN
                              DBMS_OUTPUT.PUT_LINE('THURSDAY');
                  ELSIF (OPT = 5) THEN
                              DBMS_OUTPUT.PUT_LINE('FRIDAY');
                  ELSIF (OPT = 6) THEN
                              DBMS_OUTPUT.PUT_LINE('SATURDAY');
                  ELSIF (OPT = 7) THEN
                              DBMS_OUTPUT.PUT_LINE('SUNDAY');
                  ELSE
                              DBMS_OUTPUT.PUT_LINE('WRONG INPUT');

                  END IF;
      END;
      /
```

Q2) Input Month number(1-12) and print the corresponding Month name

```
      DECLARE
                  OPT NUMBER;
      BEGIN
                  DBMS_OUTPUT.PUT_LINE('ENTER VALUE BETWEEN 1 - 12 ');
                  OPT := &OPT;
                  IF (OPT = 1) THEN
                              DBMS_OUTPUT.PUT_LINE('JANUARY');
                  ELSIF (OPT = 2) THEN
                              DBMS_OUTPUT.PUT_LINE('FEBRARY');
                  ELSIF (OPT = 3) THEN
                              DBMS_OUTPUT.PUT_LINE('MARCH');
                  ELSIF (OPT = 4) THEN
                              DBMS_OUTPUT.PUT_LINE('APRIL');
                  ELSIF (OPT = 5) THEN
                              DBMS_OUTPUT.PUT_LINE('MAY');
                  ELSIF (OPT = 6) THEN
                              DBMS_OUTPUT.PUT_LINE('JUNE');
                  ELSIF (OPT = 7) THEN
                              DBMS_OUTPUT.PUT_LINE('JULY');
                  ELSIF (OPT = 8) THEN
                              DBMS_OUTPUT.PUT_LINE('AUGUST');
                  ELSIF (OPT = 9) THEN
```

```
                        DBMS_OUTPUT.PUT_LINE('SEPTEMBER');
            ELSIF (OPT = 10) THEN
                        DBMS_OUTPUT.PUT_LINE('OCTOBER');
            ELSIF (OPT = 11) THEN
                        DBMS_OUTPUT.PUT_LINE('NOVEMBER');
            ELSIF (OPT = 12) THEN
                        DBMS_OUTPUT.PUT_LINE('DECEMBER');
            ELSE
                        DBMS_OUTPUT.PUT_LINE('WRONG INPUT');

            END IF;
      END;
      /
```

Q3)To Find Maximum between 2 Numbers

```
      DECLARE
            A NUMBER;
            B NUMBER;
      BEGIN
            A:=&A;
            B:=&B;
            IF(A>B) THEN
                        DBMS_OUTPUT.PUT_LINE(A  || ' IS GREATER THAN
' || B);
            ELSE
                        DBMS_OUTPUT.PUT_LINE(B || ' IS GREATER THAN '
||A);
            END IF;
      END;
      /
```

Q4)Check Number is Even or Odd

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

Q5) TO CHECK THE CHARACTER IS UPPERCASE OF LOWERCASE

```
      DECLARE
            ALPHA CHAR(1);
      BEGIN
            ALPHA := '&ALPHA';

            IF ASCII(ALPHA) BETWEEN ASCII('A') AND ASCII('Z') THEN
                  DBMS_OUTPUT.PUT_LINE('The character is an Uppercase
alphabet.');
```

```
                  ELSIF ASCII(ALPHA) BETWEEN ASCII('a') AND ASCII('z') THEN
                        DBMS_OUTPUT.PUT_LINE('The character is a Lowercase
alphabet.');
                  ELSE
                        DBMS_OUTPUT.PUT_LINE('The character is not an
alphabet.');
                  END IF;
      END;
      /
```

```
==========================================================================
TOPIC :- CASE

==========================================================================
Case statements works like the if statement, only the keyword 'when' is
used

Syntax:-
      Case [Expression]
      when condition 1 then result 1
      when condition 2 result 2
      .
      .
      .
      .
      Else result
      End;


Q1) To Print The Day(1-7)
      DECLARE
                  OPT NUMBER;
      BEGIN
                  OPT := &OPT;
                  CASE OPT
                              WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('MONDAY');
                              WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('TUESDAY');
                              WHEN 3 THEN
DBMS_OUTPUT.PUT_LINE('WEDNESDAY');
                              WHEN 4 THEN DBMS_OUTPUT.PUT_LINE('THURSDAY');
                              WHEN 5 THEN DBMS_OUTPUT.PUT_LINE('FRIDAY');
                              WHEN 6 THEN DBMS_OUTPUT.PUT_LINE('SATURDAY');
                              WHEN 7 THEN DBMS_OUTPUT.PUT_LINE('SUNDAY');
                  ELSE DBMS_OUTPUT.PUT_LINE('WORONG INPUT');
      END;
      /


Q2) To Check Number is Even or Odd
      DECLARE
                  A NUMBER;
      BEGIN
                  A:=&A;
                  CASE MOD(A,2)
                              WHEN 0 THEN DBMS_OUTPUT.PUT_LINE('NUMBER IS
EVEN');
                              WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('NUMBER IS
ODD');
                  ELSE DBMS_OUTPUT.PUT_LINE('IT IS NOT AN NUMBER');
      END;
      /

Q3)To Find Maximum Between 2 Numbers
      DECLARE
```

```
                A NUMBER;
                B NUMBER;
        BEGIN
                A := &A;
                B := &B;

                CASE (A>B)
                        WHEN TRUE THEN  DBMS_OUTPUT.PUT_LINE( A || '
IS MAXIMUM ');
                        WHEN FALSE THEN DBMS_OUTPUT.PUT_LINE( B ||; '
IS MAXIMUM ');
                ELSE DBMS_OUTPUT.PUT_LINE('BOTH ARE SAME ');
        END;
        /
```

Q4) To Check Number is Positive , Negative or Zero

```
        DECLARE
                A NUMBER;
        BEGIN
                A := &A;
                CASE (A>0)
                        WHEN TRUE THEN DBMS_OUTPUT.PUT_LINE('POSITIVE
NUMBER');
                        WHEN FALSE THEN
DBMS_OUTPUT.PUT_LINE('NEGATIVE NUMBER');
                ELSE DBMS_OUTPUT.PUT_LINE('NUMBER IS ZERO');
        END;
        /
```

Q5) CREATE A CALCULATOR

```
        DECLARE
                OPERATION NUMBER;
                A NUMBER;
                B NUMBER;
                C NUMBER;
        BEGIN
                DBMS_OUTPUT.PUT_LINE('ENTER OPERATION ');
                DBMS_OUTPUT.PUT_LINE('1. FOR ADDITION');
                DBMS_OUTPUT.PUT_LINE('2. FOR SUBTRACTION');
                DBMS_OUTPUT.PUT_LINE('3. FOR MULTIPLICATION');
                DBMS_OUTPUT.PUT_LINE('4. FOR DIVISION');
                OPERATION := &OPERATION;

                A:=&A;
                B:=&B;

                CASE OPERATION
                        WHEN 1 THEN
                            C := A + B;
                            DBMS_OUTPUT.PUT_LINE('ADDITION : ' ||
C);

                        WHEN 1 THEN
```

```
                                C := A - B;
                                DBMS_OUTPUT.PUT_LINE('SUBTRACTION : '
|| C);

                          WHEN 1 THEN
                                C := A * B;
                                DBMS_OUTPUT.PUT_LINE('MULTIPLICATION :
' || C);

                          WHEN 1 THEN
                                C := A / B;
                                DBMS_OUTPUT.PUT_LINE('DIVISION : ' ||
C);

                          ELSE
                                DBMS_OUTPUT.PUT_LINE('WRONG INPUT');

                END;
                /
```

```
******************************************************************
Check Code
SQL> Declare
  2  Begin
  3  dbms_output.put_line('Hello Everyone This is an Check Code of all
Connections');
  4  End;
  5  /

******************************************************************
Types of PL/SQL Loops
There are 4 types of PL/SQL Loops.
      1. Basic Loop / Exit Loop
      2. While Loop
      3. For Loop
      4. Cursor For Loop
```

```
=============================
Q2) To Print Table of a Number
Code:-
SQL> Declare
  2  i number;
  3  n number;
  4  begin
  5  i:=1;
  6  n:=&n;
  7  dbms_output.put_line('Table of '||n||' is :');
  8  Loop
  9  exit when i>10;
 10  dbms_output.put_line(n||' x '||i||' = '||n*i);
 11  i:=i+1;
 12  end loop;
 13  end;
 14  /

===============================
Q3)Find first and last digit and sum of first and Last Digit using loop
Declare
n number;
l number;
f number;
begin
n:=&n;
dbms_output.put_line('First and Last Digit is : ');
l :=MOD(n,10);
Loop
```

```
exit when n<>0
f := MOD(n,10);
n := trunc(n/10);
exit loop;
dbms_output.put_line('First Digit is : '||f);
dbms_output.put_line('Last Digit is : '||l);
end;
/
```

```
================================
Q4)Swap First and Last Digit of a Number
SQL> Declare
  2  num NUMBER := 107868764;
  3
  4  first_digit NUMBER;
  5
  6  last_digit NUMBER;
  7
  8  temp NUMBER;
  9
 10  counter NUMBER := 0;
 11
 12  begin
 13
 14  last_digit := MOD(num,10);
 15
 16  temp := num;
 17
 18  while num<>0 loop
 19
 20  first_digit := MOD(num,10);
 21
 22  counter := counter + 1;
 23
 24  num := trunc(num/10);
 25
 26  end loop;
 27
 28  temp := temp - last_digit;
 29
 30  temp := temp - first_digit*POWER(10,counter-1);
 31
 32  temp := temp + first_digit;
 33
 34  temp := temp + last_digit*POWER(10,counter-1);
 35
 36  dbms_output.put_line(temp);
 37
 38  end;
 39
 40  /
```

```
====================================
While Loop
====================================
Q1) To Print  10 Numbers
SQL> Declare
  2  i INTEGER := 1;
  3  BEGIN
  4  WHILE i <= 10 LOOP
  5  DBMS_OUTPUT.PUT_LINE(i);
  6  i := i+1;
  7  END LOOP;
  8  END;
  9  /

====================================
Q2) To Find first and last digit using loop
SQL> Declare
  2  n number;
  3  l number;
  4  f number;
  5  begin
  6  n:=&n;
  7  dbms_output.put_line('First and Last Digit is : ');
  8  l :=MOD(n,10);
  9  while n!=0 Loop
 10  f := MOD(n,10);
 11  n := trunc(n/10);
 12  End loop;
 13  dbms_output.put_line('First Digit is : '||f);
 14  dbms_output.put_line('Last Digit is : '||l);
 15  end;
 16  /

===================================
Q3) To Print Reverse of a Number using Loop
SQL> Declare
  2  n number;
  3  d number;
  4  r number:=0;
  5  begin
  6  n:=&n;
  7  while n!=0 loop
  8  d:= MOD(n,10);
  9  r := (r*10)+d;
 10  n:= trunc(n/10);
 11  End Loop;
 12  dbms_output.put_line('Reverse is : ' ||r);
 13  End;
 14  /

======================================
Q4) To Find Factorial of a Number
```

```
SQL> Declare
  2  n number;
  3  f number:=1;
  4  i number:=1;
  5  begin
  6  n:=&n;
  7  while i<=n loop
  8  f :=f*i;
  9  i:=i+1;
 10  End loop;
 11  dbms_output.put_line('Factorial of '|| n || ' is : '||f);
 12  End;
 13  /
```

```
======================================
Q5)  To check Number is palindrome number or Not
SQL> Declare
  2  n number;
  3  r number:=0;
  4  d number;
  5  temp number;
  6  begin
  7  n:=&n;
  8  temp := n;
  9  while n!=0 loop
 10  d:= MOD(n,10);
 11  r := (r*10)+d;
 12  n:= trunc(n/10);
 13  End Loop;
 14  if(temp=r)
 15  then
 16  dbms_output.put_line( temp || ' is Palindrome ');
 17  else
 18  dbms_output.put_line( temp || ' is not Palindrome ');
 19  End if;
 20  End;
 21   /
```

```
======================================
Q6)  To Check Number is Prime or Not
Declare
n number;
i number:=2;
c number:=0;
begin
n:=&n;
while(i<n) loop
if Mod(n,i)= 0
then
c := c + 1;
end if;
i := i + 1;
end loop;
if( c = 0)
```

```
then
dbms_output.put_line(' Number is Prime ');
else
dbms_output.put_line(' Number is Not Prime');
End if;
End;
/
```

======================================

Q7) Armstrong Number or not

```
SQL> Declare
  2  n number;
  3  d number;
  4  cube number;
  5  temp number;
  6  fin number:=0;
  7  Begin
  8  n:=&n;
  9  temp:=n;
 10  while(n!=0) loop
 11  d:=Mod(n,10);
 12  cube:=d*d*d;
 13  fin:=fin+cube;
 14  n:=trunc(n/10);
 15  end loop;
 16  If(temp=fin) then
 17  Dbms_output.put_line(temp || ' is an Armstrong Number ');
 18  else
 19  Dbms_output.put_line(temp || ' is not an Armstrong Number ');
 20  End if;
 21  End;
 22  /
```

======================================

For Loop

======================================

Q1) To Print 1 To 10 Numbers

```
Declare
i number;
begin
for i in 1..10 Loop
dbms_output.put_line(i);
End loop;
End;
/
```

======================================

Q2) To Find First and Last Digit and Sum of 1 and Last Digit(It is Found uisnf if Else)

```
Declare
SQL> DECLARE
  2      a INTEGER := 14598;
  3      b INTEGER := 0;
  4      C INTEGER := 0;
```

```
    s INTEGER;
BEGIN
    IF a > 9 THEN
       c := Substr(a, 1, 1);
       b := Substr(a, Length(a), 1);
      s := b + c;
    ELSE
      s := a;
    END IF;
    dbms_output.Put_line('Sum of the first and last digit is ' ||s);
END;
/
```

==
Q3) To Print Numbers in Reverse Order in For Loop

```
SQL> Declare
  n1 number;
 begin
 n1 := 10;
 for k in  REVERSE 1..10  loop
 dbms_output.put_line(n1*k);
 End loop;
 End ;
 /
```

==
Q4) To Check Prime Number or Not

```
 Declare
n number;
i number :=2;
c number:=0;
begin
n:=&n;
for k in 2..n-1 loop
if( Mod(n,k)= 0)
then
      c:=c+1;
End if;
End loop;
if(c=0)
then
      dbms_output.put_line('Number is Prime');
else
      dbms_output.put_line('Number is Not Prime');
End if;
End;
  /
```

```
======================================================
Strings
======================================================
Q1) To Print a Simple Text(Introduction to Datatypes)
SQL> Declare
  2  name varchar(20);
  3  company varchar2(30);
  4  introduction clob; -- Character Large Object Datatype
  5  choice char(1);
  6  Begin
  7  name:='Sajid Shaikh';
  8  company:='Itech Computer Education ';
  9  introduction:='I am a Professional software Developer';
 10  choice:='Y';
 11  if choice = 'Y' then
 12  dbms_output.put_line(name);
 13  dbms_output.put_line(company);
 14    dbms_output.put_line(introduction);
 15  End if;
 16  End;
 17  /

 =====================================================
 Q2) To Take User Input in String
SQL> declare
  2    name varchar2(10);
  3  begin
  4    name := '&k';
  5    dbms_output.put_line('Name is: ' || name);
  6  end;
  7  /


 ======================================================
Q3) To Find Length of String
SQL> Declare
  2  name varchar2(150);
  3  Begin
  4  name:='&name';
  5  dbms_output.put_line('Length of String is ' || Length(name));
  6  End;
  7  /


========================================================
Q4) To Convert Uppercase to LowerCase
SQL> Declare
  2  str varchar2(100);
  3  begin
  4  str :='&str';
  5  dbms_output.put_line('Before to Change Case :' ||str);
  6  dbms_output.put_line('LowerCase : ' || Lower(str));
  7  dbms_output.put_line('UpperCase : ' || Upper(str));
  8  End;
  9  /
```

==

Q5)To Concatenate Two Strings

Method-1

```
  SQL> Declare
  2     str1 varchar2(50):='Ram is Honest Boy';
  3     str2 varchar2(50):='Suraj is Briliant Boy';
  4     str3 varchar2(50) :='and';
  5     final_str varchar2(200);
  6     Begin
  7     final_str := str1 || ' ' || str3 || ' ' || str2;
  8     dbms_output.put_line('Concatenate String is : ' || final_str);
  9     End;
 10     /
```

Method-2

```
  SQL> DECLARE
  2     Test_String string(10) := 'Geeks';
  3     Test_String2 string(10) := 'For';
  4     Test_String3 string(10) := 'Geeks';
  5
  6  BEGIN
  7     dbms_output.put_line(CONCAT(CONCAT(Test_String, Test_String2),
Test_String3));
  8
  9  END;
 10  /
```

==

Q6)To Compare Two Strings

```
SQL> Declare
  2  str1 varchar2(50):='Welcome All';
  3  str2 varchar2(50):='Welcome All';
  4  str3 varchar2(50):='welcome all';
  5  begin
  6  dbms_output.put_line('Str1 :- ' || str1);
  7  dbms_output.put_line('Str2 :- ' || str2);
  8  dbms_output.put_line('Str3 :- ' || str3);
  9  dbms_output.put_line(' ');
 10    if(str1=str2) then
 11      dbms_output.put_line('Str1 Compares Str2 :- Equal ');
 12    else
 13      dbms_output.put_line('Str1 Compares Str2 :- Non Equal ');
 14    End if;
 15
 16    if(str1=str3) then
 17      dbms_output.put_line('Str1 Compares Str3 :- Equal ');
 18    else
 19      dbms_output.put_line('Str1 Compares Str3 :- Non Equal ');
 20    End if;
 21
 22    if(str2=str3) then
```

```
    dbms_output.put_line('Str2 Compares Str3 :- Equal ');
  else
    dbms_output.put_line('Str2 Compares Str3 :- Non Equal ');
  End if;

End;
/
```

==
Q7)To Toogle Case of Each Character of a String

```
SQL> DECLARE
    str1 VARCHAR2(32767);
    str2 VARCHAR2(32767) := '';
BEGIN
  str1 :='&str1';
    FOR i IN 1..LENGTH(str1) LOOP
        IF SUBSTR(str1, i, 1) = UPPER(SUBSTR(str1, i, 1)) THEN
            str2 := str2 || LOWER(SUBSTR(str1, i, 1));
        ELSE
            str2 := str2 || UPPER(SUBSTR(str1, i, 1));
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE(str2);
END;
/
```

==
Q8)To Count Total No of Alphabets , Digits and Symbols

```
SQL> Declare
    str varchar2(1500);
    alphabets number:=0;
    digits number:=0;
    symbols number:=0;

Begin
    str :='&str';
    For i in 1..Length(str) Loop
      If Ascii(substr(str , i , 1 )) Between 48 and 57 then
          digits := digits + 1 ;

      elsif Ascii(substr(str , i , 1)) Between 65 and 122 then
          alphabets := alphabets + 1 ;

       else
          symbols := symbols + 1 ;

      End If;
      End Loop;
    Dbms_output.put_line('Length of Str:'||Length(str));
    Dbms_output.put_line('Digits: '    ||    digits);
    Dbms_output.put_line('Alphabets: ' || alphabets);
    Dbms_output.put_line('Symbols: '   ||   symbols);         --
Symbols include spaces also
```

```

     End;
     /
```

```
========================================================
Q9)To Print Total Number of Consonants and Vowels
SQL> Declare
         str varchar2(1500);
         consonants number:= 0;
         vowels number:= 0;

     Begin
         str :='&str';

         str := Upper(str);
         For i in 1..Length(str) Loop
           If substr(str , i , 1 ) In ('A' , 'E' , 'I' , 'O' , 'U') Then
              vowels := vowels + 1;
           elsif substr(str , i , 1) Between 'B' and 'Z' then
             consonants := consonants + 1 ;
           End If;
         End Loop;

         Dbms_output.put_line('Length of Str:'||Length(str));
         Dbms_output.put_line('Vowels: '    ||    vowels);
         Dbms_output.put_line('Consonants: ' || consonants);    --Not
Counting Spaces in Consonants
       End;
     /
```

```
========================================================
Q10) Program To Count Total Number of Words
SQL> Declare
       str varchar2(1500);
       words number := 1;

     Begin
       str := '&str';
       For i in 1..Length(str) loop
         If substr(str , i , 1) = ' ' then
           words := words + 1 ;

           End if;
         End loop;

        Dbms_output.put_line('No Of Words : ' || words);
     End;
     /
```

```
========================================================
Q11) To Print String In Reverse Order
SQL> Declare
```

```
  str varchar2(1500);
Begin
  str := '&str';

  For i in Reverse 1..Length(str) loop
    Dbms_output.put(substr(str, i, 1));
  End loop;
  Dbms_output.New_Line;
End;
/
```

 ===

```
Q12) To Check String is palindrome or Not
SQL> Declare
  str varchar2(1500);
  new_str varchar2(1500);
Begin
  str := '&str';

  Dbms_output.put_line('Old String : ' || str);

  For i in Reverse 1..Length(str) loop
    new_str := new_str || substr(str, i, 1);
  End loop;
  Dbms_output.put_line('Reverse String : ' || new_str);
  Dbms_output.New_line;

  If new_str = str then
    Dbms_output.put_line('Strings are Equal ');
  Else
    Dbms_output.put_line('Strings are Not Equal');
  End If;
End;
/
```

==

```
****** SOME MORE FUNCTIONS ******
  i)  ASCII()
    The ASCII() function converts a Character to its ACSII Code
  Syntax:
      SELECT ASCII('A') FROM DUAL;

  ii)  CHR()
    The CHR() function converts an ASCII code, which is a numeric value
between 0 and 225,  to a character.
  Syntax:
      SELECT CHR('65') FROM DUAL;

  iii)  CONCAT()
    The CONCAT() concatenates two Strings
  Syntax:
      SELECT CONCAT('Happy',' Birthday') FROM DUAL;
  iv)  DUMP()
    The DUMP() function allows you to find the data type, length, and
internal representation of a value.
    Syntax:
      SELECT DUMP('HAPPY CODING') fron DUAL;

  v)  LENGTH()   or  VSIZE()
    Finds the Length of String
  Syntax:
      SELECT LENGTH('hello') FROM DUAL;
      SELECT VSIZE('hello') FROM DUAL;
      returns Same Output

  vi)  INSTRB()
    This function returns the location of substring using bytes(It is Case
Sensitive)
  Syntax:
    Select INSTRB('PLSQL ORACLE', 'O') from dual;

  vii)  LPAD()
    LPAD function Adds the left-side of a string and Returns with a
specific set of characters
  Syntax:
    SELECT LPAD(' HUMANS ' , 6 , 'HELLO') AS TEXT FROM DUAL;
    Here it Adds HEllo to Humans String and returns only 6 Characters
after Adding as 'HELLO HUMANS'

  viii)  LTRIM()
    This function returns the string by removing given characters from
left side(Case Sensitive)
  Syntax:
    SELECT LTRIM('JAVA','J') FROM DUAL;

  ix)  REPLACE()
    This function is used to replace the sequence of character with
another character in the given string.
  Syntax:
    REPLACE(string, to_replace , replace_string)
    SELECT REPLACE('Oracle Sql', 'Or','Ttt') FROM dual;

  x)  RPAD()
    RPAD() function returns the right-padded to the given length.
  Syntax:
    SELECT RPAD('HELLO' , 10 , ' WORLD' ) AS TEXT FROM DUAL;
    Here it Adds World To Hello String and returns only 10 Characters
after Adding as 'HELLO WORLD'

  xi)  RTRIM()
    This function returns the string by removing given characters from the
right side.(Case Sensitive LE)
  Syntax:
    SELECT RTRIM('ORACLE','LE') FROM DUAL;

  xii)  TRANSLATE()
```

This function is used to replace the character from the given character. This function replaces one character only

Syntax:

```
    SELECT TRANSLATE('%% HELLO WORLD' , '%','HH') FROM DUAL;
```

xiii) TRIM()

This function is used to remove the specified character from head of the string or tail of the string.

Syntax:

```
    SELECT TRIM(BOTH '*' FROM '********* HELLO WORLD *************') FROM
DUAL;
```

==

Array

An array is a part of collection type data and it stands for variable-size arrays.

The PL/SQL programming language provides a data structure called the VARRAY.

Creating a Varray Type

A varray type is created with the CREATE TYPE statement. You must specify the maximum size and the type of elements stored in the varray.

The index of an array (including VARRAYs) starts from 1. The first element of the array has an index of 1, the second element has an index of 2, and so on.

basic syntax :-

```
      CREATE Or REPLACETYPE namearray IS VARRAY(5) OF VARCHAR2(10);
Example:-
                  Type sgrades IS VARRAY(5) OF INTEGER;
```

Where,

```
                  varray_type_name is a valid attribute name,
                  n is the number of elements (maximum) in the varray,
                  element_type is the data type of the elements of the
array.
                  Maximum size of a varray can be changed using the ALTER
TYPE statement.
```

==

Q1)Print Elements

```
      DECLARE
                  TYPE arr is VARRAY(5) OF NUMBER;
                  a arr:= arr(5,2,3,4,1);
                  total NUMBER;
      BEGIN
                  total := a.COUNT;
                  DBMS_OUTPUT.PUT_LINE('VARRAY IS ');
                  FOR i in 1..total LOOP
                              DBMS_OUTPUT.PUT_LINE('ELEMENT ' || i ||
a(i));
                  END LOOP;
      END;
      /
```

==

Q2)Program to Find Sum of all Elements

```
      DECLARE
                  TYPE arr is VARRAY(10) of NUMBER;
                  a arr := arr(1,2,3,4,5,6,7,8,9,10);
                  sum_value INTEGER := 0 ;
                  temp INTEGER;
                  n INTEGER;
       BEGIN
                  a := arr(1,2,3,4,5,6,7,8,9,10);
                  n := a.count;
                  FOR i in 1..n LOOP
                             DBMS_OUTPUT.PUT_LINE('Element ' || a(i));
                             temp := a(i);
                             sum_value := sum_value + a(i);
                  END LOOP;
                  DBMS_OUTPUT.PUT_LINE('SUM OF ALL ELEMENTS : ' ||
 SUM_VALUE);
       END;
       /
```

==
Q3)To Find Max and Min Between The Elements

```
      DECLARE
               TYPE arr IS VARRAY(10) OF NUMBER;
               a arr := arr(5,4,15,25,85,74,-45,98,100);
               max_val NUMBER;
               min_val NUMBER;
               n INTEGER;
      BEGIN
               n := a.COUNT;

               IF n > 0 THEN
                        max_val := a(1);
                        min_val := a(1);

                        FOR i IN 2..n LOOP
                            IF a(i) > max_val THEN
                                     max_val := a(i);
                            ELSIF a(i) < min_val THEN
                                     min_val := a(i);
                            END IF;
                        END LOOP;

                        DBMS_OUTPUT.PUT_LINE('Maximum Element: ' ||
max_val);
                        DBMS_OUTPUT.PUT_LINE('Minimum Element: ' ||
min_val);
               ELSE
                        DBMS_OUTPUT.PUT_LINE('Array is empty.');
               END IF;
      END;
      /
```

==
Q3)To Sort the Elements of Array

```
      **************** DESCENDING ORDER ***********************

      DECLARE
```

```
                TYPE arr is VARRAY(10) OF NUMBER;
                a arr := arr(5,4,15,25,85,74,-45,98,100,150);
                total NUMBER;
                temp NUMBER;
        BEGIN
                total := a.COUNT;
                FOR i in 1..total LOOP
                        FOR j in 1..total LOOP
                            IF a(i) > a(j) THEN
                                    temp := a(i);
                                    a(i) := a(j);
                                    a(j) := temp;
                            END IF;
                        END LOOP;
                END LOOP;

                DBMS_OUTPUT.PUT_LINE('SORTED VARRAY IS ');
                FOR i in 1..total LOOP
                        DBMS_OUTPUT.PUT_LINE('ELEMENT ' || i || ' : '
|| a(i));
                END LOOP;
        END;
        /

        **************** ASCENDING ORDER ***********************

        DECLARE
                TYPE arr is VARRAY(10) OF NUMBER;
                a arr := arr(5,10,-5,14,32,18,2,3,4,1);
                total NUMBER;
                temp NUMBER;
        BEGIN
                total := a.COUNT;
                FOR i in 1..total LOOP
                        FOR j in 1..total LOOP
                            IF a(i) < a(j) THEN
                                    temp := a(i);
                                    a(i) := a(j);
                                    a(j) := temp;
                            END IF;
                        END LOOP;
                END LOOP;

                DBMS_OUTPUT.PUT_LINE('SORTED VARRAY IS ');
                FOR i in 1..total LOOP
                        DBMS_OUTPUT.PUT_LINE('ELEMENT ' || i || ' : '
|| a(i));
                END LOOP;
        END;
        /

=================================================================
Q5) Program to Change Any Element of VARRAY
        DECLARE
```

```
                 TYPE arr is VARRAY(10) of NUMBER;
                 a arr := arr(5,4,14,87,-5,78,13,48,95,10);
                 index_no NUMBER;
                 new_Value NUMBER;
                 n INTEGER;
      BEGIN
                 n := a.count;
                 DBMS_OUTPUT.PUT_LINE('VARRAY IS ');
                 FOR i in 1..n LOOP
                            DBMS_OUTPUT.PUT_LINE('Element ' || i || ' : '
|| a(i));
                 END LOOP;

                 index_no := &index_no;

                 IF index_no > 0 AND index_no < 11 THEN
                          new_value := &new_value;
                          a(index_no) := new_value;

                          DBMS_OUTPUT.PUT_LINE('UPDATED VARRAY IS ');
                          FOR i in 1..n LOOP
                                 DBMS_OUTPUT.PUT_LINE('Element ' || i ||
' : ' || a(i));
                            END LOOP;

                 ELSE
                          DBMS_OUTPUT.PUT_LINE('Error: This Index No Does
Not Exist!!!');
                 END IF;
      END;
      /
```

NOTE :In SQL*Plus, substitution variables (e.g., &index_no and &new_value) are replaced before the code is sent to the database for execution. This means that all input prompts are processed first,because after all inputs the entire block, including the loops and logic, is executed and after and Sent to Database .

==

Q6) To Print VARRAY In Reverse Order

```
      DECLARE
                 TYPE arr is VARRAY(10) OF NUMBER;
                 a arr:= arr(5,10,15,20,25,30,35,40,45,50);
                 total NUMBER;
      BEGIN
                 total := a.COUNT;

                 DBMS_OUTPUT.PUT_LINE('VARRAY IS ');
                 FOR i in  1..total LOOP
                            DBMS_OUTPUT.PUT_LINE('ELEMENT ' || i || ' : '
|| a(i));
                 END LOOP;

                 DBMS_OUTPUT.PUT_LINE('REVERSED VARRAY IS ');
                 FOR i in REVERSE 1..total LOOP
```

```
                             DBMS_OUTPUT.PUT_LINE('ELEMENT '|| i || ' : '
|| a(i));
                 END LOOP;
      END;
      /
```

Q7)Print Marksheet Using Varray

```
                 SQL> Declare
                           TYPE sname is Varray(5) Of varchar2(50);
                           TYPE sstd is varray(5) of INTEGER;
                           TYPE sdiv is Varray(5) Of char(1);
                           TYPE srolls is varray(5) of INTEGER;
                           names sname;
                           std sstd;
                           div sdiv;
                           roll srolls;
                    total INTEGER;
                    BEGIN
                             names := sname('Ayush', 'Suraj', 'Ayan',
'Rishabh', 'Rohit');
                             std   := sstd( 5 , 6 , 5 , 6 , 1 );
                             div   := sdiv('A' , 'B' , 'C' , 'D' , 'A');
                             roll  := srolls(15,16,17,18,19);
                             total := names.count;
                             DBMS_OUTPUT.PUT_LINE('Total '|| total || '
Students');
                             FOR i in 1..total LOOP
                             DBMS_OUTPUT.PUT_LINE('Student ' || (i));
                                DBMS_OUTPUT.PUT_LINE('' || 'Name :' ||
names(i));

                                DBMS_OUTPUT.PUT_LINE('' || 'Std  :' ||
std(i));

                                DBMS_OUTPUT.PUT_LINE('' || 'Div  :' ||
div(i));

                                DBMS_OUTPUT.PUT_LINE('' || 'Roll :' ||
roll(i));

                                DBMS_OUTPUT.NEW_LINE;
                             END LOOP;
                    END;
                    /
```

Example 2

```
                 SQL> DECLARE
                   2     TYPE snames IS VARRAY(5) OF VARCHAR2(10);
                   3     TYPE sgrades IS VARRAY(5) OF INTEGER;
                   4     names snames;
                   5     marks sgrades;
                   6     total INTEGER;
                   7  BEGIN
                   8     names := snames('Kavita', 'Pritam', 'Ayan',
'Rishav', 'Aziz');
                   9     marks:= sgrades(98, 97, 78, 87, 92);
                  10     total := names.count;
```

```
   DBMS_OUTPUT.PUT_LINE('Total '|| total || ' Students');
   FOR i in 1 .. total LOOP
      DBMS_OUTPUT.PUT_LINE('Student: ' || names(i) || ' Marks: ' || marks(i));
   END LOOP;
END;
/
```

```
===========================================================================
====
Topic :- 7) DATE AND TIME

---------------------------------------------------------------------------
------------------------------------------------------------
10) SELECT SYSDATE FROM DUAL;
      OUTPUT:-  14-NOV-23
---------------------------------------------------------------------------
------------------------------------------------------------
11) SELECT TO_CHAR(CURRENT_DATE , 'DD-MM-YYYY HH:MI:SS') FROM DUAL;
      OUTPUT:-14-11-2023 07:12:58       -- By using Current Date
---------------------------------------------------------------------------
------------------------------------------------------------
12) SELECT TO_CHAR(SYSDATE , 'DD-MM-YYYY HH:MI:SS') FROM DUAL;
      OUTPUT:-14-11-2023 07:13:53       -- By using Sysdate

---------------------------------------------------------------------------
------------------------------------------------------------
13) Print Month with Day
      BEGIN
               DBMS_OUTPUT.put_line(TO_CHAR (SYSDATE, 'Day, DDth Month
YYYY'));
      END;
      /

      OUTPUT:- Tuesday  , 14TH November  2023
      This Code Will Work in BEGIN and END Statement Only
---------------------------------------------------------------------------
------------------------------------------------------------
14)   SELECT ADD_MONTHS(SYSDATE , 5) FROM DUAL;
      OUTPUT:- 14-APR-24                -- Adds No of Months Given and
Returns New Date and Used Sysdate
---------------------------------------------------------------------------
------------------------------------------------------------
15)   SELECT ADD_MONTHS(CURRENT_DATE , 5) FROM DUAL;
      OUTPUT:- 14-APR-24                -- Adds No of Months Given and
Returns New Date and Used Current_Date
---------------------------------------------------------------------------
------------------------------------------------------------
16)   SELECT SYSDATE AS CURRENT_DATE_TIME, EXTRACT(Month FROM SYSDATE) AS
    ONLY_CURRENT_MONTH FROM DUAL;
      OUTPUT:- 14-NOV-23   11                       --Gives Current Date
and Only Current Month ( Used Sysdate)
---------------------------------------------------------------------------
------------------------------------------------------------
17)   SELECT Current_Date AS CURRENT_DATE_TIME, EXTRACT(Month FROM
    SYSDATE) AS ONLY_CURRENT_MONTH FROM DUAL;
      OUTPUT:- 14-NOV-23   11                       --Gives Current Date
and Only Current Month ( Used Current_Date)
---------------------------------------------------------------------------
------------------------------------------------------------
18)   SELECT LOCALTIMESTAMP FROM DUAL;
      OUTPUT:- 14-NOV-23 07.18.54.127000 PM
```

```
-----------------------------------------------------------------------------
-------------------------------------------------------------
19) SYSTIMSTAMP := It is a function that returns the current date and time
    including fractional seconds and time zone. It is more precise
      than the CURRENT_TIMESTAMP function because it includes fractional
    seconds.

      SELECT SYSTIMSTAMP FROM dual;
      OUTPUT:-

                  NAME                         CODE
                              OUTPUT
                  YEAR                   TO_CHAR(SYSTIMESTAMP, 'YYYY')
            2023
                  MONTH                  TO_CHAR(SYSTIMESTAMP, 'MM')
      DECEMBER
                  WEEK                   TO_CHAR(SYSTIMESTAMP, 'WW')
      52
                  DAY                    TO_CHAR(SYSTIMESTAMP, 'DD')
      23
                  DAY OF YEAR      TO_CHAR(SYSTIMESTAMP, 'DDD')
      296
                  WEEK DAY         TO_CHAR(SYSTIMESTAMP, 'Day')
      MONDAY
                  HOUR OF DAY      TO_CHAR(SYSTIMESTAMP, 'HH24')
      18
                  MINUTE                 TO_CHAR(SYSTIMESTAMP, 'MI')
      24
                  SECOND                 TO_CHAR(SYSTIMESTAMP, 'SS')
      30
                  MILLISECONDS     TO_CHAR(SYSTIMESTAMP, 'FF3')
      777

-----------------------------------------------------------------------------
-------------------------------------------------------------
20)  CURRENT_TIMESTAMP: This function returns the current date and time of
     the database server. It is similar to SYSTIMESTAMP but does not
      include fractional seconds.

      SELECT CURRENT_TIMESTAMP FROM DUAL;
      OUTPUT :- 14-NOV-23 07.24.48.031000 PM +05:30

      *********** Code - 2 ********************

      SELECT TO_CHAR(CURRENT_TIMESTAMP, 'YYYY-MM-DD HH24:MI:SS') AS
current_timestamp,
       TO_CHAR(SYSTIMESTAMP, 'YYYY-MM-DD HH24:MI:SS.FF9 TZD') AS
systimestamp
      FROM DUAL;

      OUTPUT:- 2023-11-16 18:30:57  2023-11-16 18:30:57.629000000
-----------------------------------------------------------------------------
-------------------------------------------------------------
21)  TO_DATE
```

Convert a date which is in the character string to a DATE value.

```
SELECT TO_DATE( '01 Jan 2017', 'DD MON YYYY' ) FROM DUAL;
OUTPUT:- 01-JAN-17
```

Topic :- Table Creation and CRUD Operations

In Oracle, CREATE TABLE statement is used to create a new table in the database.
To create a table, you have to name that table and define its columns and datatype for each column.

1)Create Table

Syntax:

```
CREATE TABLE table_name
(
  column1 datatype [ NULL | NOT NULL ],
  column2 datatype [ NULL | NOT NULL ],
  ...
  column_n datatype [ NULL | NOT NULL ]
);
```

2)Create Table As

The CREATE TABLE AS statement is used to create a table from an existing table by copying the columns of existing table.

Syntax:

```
CREATE TABLE new_table
AS (SELECT * FROM old_table);
```

3)Alter Table

It is Used to add, modify, drop or delete columns in a table. It is also used to rename a table.

A)Add Column in Table

Syntax:

```
ALTER TABLE table_name
  ADD column_name column-definition;
```

B)Modify Column in Table

Syntax:

```
  ALTER TABLE table_name
  MODIFY (column_1 column_type,
      column_2 column_type,
      ...
      column_n column_type);
```

C) Drop Column in Table

Syntax:

```
Alter Table table_name
  Drop Column Column_name
```

D)Rename Column

Syntax:

```
ALTER TABLE table_name
  RENAME COLUMN old_name to new_name;
```

E)Rename Table

Syntax:

```
ALTER TABLE table_name
  RENAME TO new_table_name;
```

```
4)Drop Table
  SynTax:
    Drop Table Table_name
```

```
==========================================================================
=============
Topic :- 9) Types OF Queries

     Table:
          Create Table table1(Name Varchar2(50) ,Age Number ,Salary
Integer);
          Insert into Employees Values('Mohan'  , 21 ,  , 25000);
          Insert into Employees Values('Suraj'  , 22 ,  , 30000);
          Insert into Employees Values('Ravish' , 24 ,  , 40000);
          Insert into Employees Values('Nitish' , 25 ,  , 50000);
--------------------------------------------------------------------------
--------------
     1)Select Query
          Select * from Table_name;

          Example:-
               Select * from table1;
--------------------------------------------------------------------------
--------------
     2)Insert Query
          A)Insert
               Insert into Table_name Values(....);

          B)Insert All
          Syntax:
               Insert All
                    Into table_name(Column1, column2,...) Values(....)
                    Into table_name(Column1, column2,...) Values(....)
                    Into table_name(Column1, column2,...) Values(....)
                    Into table_name(Column1, column2,...) Values(....)
               Select * From table_name;
--------------------------------------------------------------------------
--------------
     3)Update Query
          Update table_name set column_name  = ' ' .... where condition;

          Example:
               Update table1 set name='Prayas',  where Age = 22;

--------------------------------------------------------------------------
--------------
     4)Delete Query
          Delete from Table_name where Condition;

     Example:-
          Delete table Table1 where age = 22;

--------------------------------------------------------------------------
--------------
```

```
Topic :- Types OF clauses

*****************************************
Tables For Reference
      Table_name : Employees
      Columns:= (Name varchar2(50) Varchar2(50) ,Age Number ,Address
varchar2(200) ,Salary Integer)

****************************************

1)Distinct Clause
      DISTINCT clause is used to remove the duplicate records from the
result set
      Syntax:
            Select Distinct .....
            From table_name
            Where conditions;

            Example:
                  SELECT DISTINCT name, age, salary
                  FROM  Employees
                  WHERE age >22;
--------------------------------------------------------------------------
--------------

2)Order By Clause
      ORDER BY Clause is used to sort or re-arrange the records in the
result set.
      Syntax:
            Select column_name(s) ,  .....
            From table_name
            Where conditions
            Order by column_name Asc/Desc;

            Example:
                  SELECT name, age, salary
                  FROM  Employees
                  WHERE age >= 22;
                  Order by Age ASC;
--------------------------------------------------------------------------
--------------

3)GROUP BY Clause
      Group By Clause is Used to Collect Data From Multiple Records and
Group the Results accordingly

      Syntax:
            Select Column_name(s), Function(column_name)
            from Table_name
            Where Conditions
            Order by Column_name;

      Example:
            Select name , Sum(Salary)
            From Employees
            Group By Name;
--------------------------------------------------------------------------
--------------

4)Having Clause
```

```
It is a Condition Clause which is used with Group BY
Syntax:
      Select Column_name(s), Function(column_name)
      from Table_name
      Where Conditions
      Order by Column_name;
      Having Condition;

Example:
      Select name , Sum(Salary) as total_Salary
      From Employees
      Group By Name
      Having Sum(Salary)<50000;
```

```
---------------------------------------------------------------------------
--------------
```

```
Topic :- Types OF Operators


*****************************************
Tables For Reference
      Table_name : Supplier
      Columns:= (Id Number, FName varchar2(20), LNameVarchar2(20))

      Table_Name : Customers
      Columns:=(Name Varchar2(20) , Age Number , Amount Nmber)

*************************************************************************
************************
1)Union Operator
UNION operator is used to combine the result sets of two or more Oracle
SELECT statements. It combines the both SELECT statement
and removes duplicate rows between them.Each SELECT statement within the
UNION operator must have the same number of fields
in the result sets with similar data types.
Syntax :
      SELECT expression1, expression2, ... expression_n
      FROM table1
      WHERE conditions
      UNION
      SELECT expression1, expression2, ... expression_n
      FROM table2
      WHERE conditions;

      Ex:
      Select Id ,  Name from Customers Union Select Id , FNAME from
Supplier;
      /* It Should Contain Same Datatype on Both Select Statements (int ,
varchar ==> Int , varchar) not varchar , Int */

=========================================================================
========================================================

2)UNION ALL
This operator is used to combine the result sets of 2 or more SELECT
statements. It is different from UNION operator in a way
that it does not remove duplicate rows between the various SELECT
statements. It returns all of the rows.
Each SELECT statement within the UNION ALL must have the same number of
fields in the result sets with similar data types.
      Syntax :
      SELECT expression1, expression2, ... expression_n
      FROM table1
      WHERE conditions
      UNION
      SELECT expression1, expression2, ... expression_n
      FROM table2
      WHERE conditions;

      Ex:
      Select Id ,  Name from Customers Union All Select Id , FNAME from
Supplier;
      /* It Should Contain Same Datatype on Both Select Statements (int ,
varchar ==> Int , varchar) not varchar , Int */
```

==
==

3)Intersect Operator
It Returns the Results of 2 otr more Select Statemen and Picks the Common Or Intersecting Records

```
      Syntax:-
            SELECT expression1, expression2, ... expression_n
            FROM table1
            WHERE conditions
            INTERSECT
            SELECT expression1, expression2, ... expression_n
            FROM table2
            WHERE conditions;

      Ex:
      Select Id from Customers Intersect Select Id from Supplier;
```

==
==

4)Minus Operator
It is used to return all rows in the First Select Statement and Not by Second Select Satatement
Each SELECT statement has a dataset and the MINUS operator returns all documents from the first dataset and then
removes all documents from the second dataset.

```
      Syntax:
      Select Id From Supplier Minus Select Id From Customers;
```

Explanation : result of this query would be a list of Id values from the "Supplier" table that are not found in the "Customers" table.

==
==
=
==
==
=

```
Topic :- Types OF Joins

Tables For Reference
      Table_name : Supplier(9 Records)
      Columns:= (Id Number, FName varchar2(20), Address Varchar2(20));

      Table_Name : Orders(5 Records)
      Columns:=(Id Number ,Order_no Number,  City Varchar2(20))

      Table_Name : Customers(3 Records)
      Columns:=(Name Varchar2(20) , Age Number , Amount Nmber)

===========================================================================
=========================================================
1)Inner Join
Inner Join Returns all rows from multiple tables where the join condition
is True.
      Syntax:
            Select Columns
            From Table1
            Inner Join Table2
            On Table1.column_name = Table2.column_name;

            Select Supplier.id , Supplier.Fname , orders.Order_no ,
Orders.City
            from Supplier
            Inner  Join Orders
            on supplier.Id = Orders.id;


===========================================================================
=========================================================

2)Left Outer Join
It returns all the rows of the first table and specified rows of the
second table where the ON condition is true
      Syntax:
            Select Columns
            From Table1
            Left outer Join Table2
            On Table1.colum_name=table2.column_name;

            Select Supplier.id , Supplier.Fname , orders.Order_no ,
Orders.City
            from Supplier
            Left Outer Join Orders
            on supplier.Id = Orders.id;

===========================================================================
=========================================================

3)Right Outer Join
It returns all the rows of the Second table and specified rows of the
first table where the ON condition is true
      Syntax:
            Select Columns
            From Table1
            Right outer Join Table2
            On Table1.colum_name=table2.column_name;
```

```
           Select Supplier.id , Supplier.Fname , orders.Order_no ,
Orders.City
           from Supplier
           Right Outer Join Orders
           on Orders.Id = Supplier.id;
```

==
==

4)Full Outer Join
It returns all the rows of the first table and all rows of the second table where the ON condition is true or False(Not Seen On Condition but it is in Syntax)

```
      Syntax:
           Select Columns
           From Table1
           Full outer Join Table2
           On Table1.colum_name=table2.column_name;

           Select Supplier.id , Supplier.Fname , orders.Order_no ,
Orders.City
           from Supplier
           Full Outer Join Orders
           on Orders.Id = Supplier.id;
```

==
==

5)Cross Join
The CROSS JOIN specifies that all rows from first table join with all of the rows of second table. If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have x*y rows. It normally happens when no matching join columns are specified.

Syntax:
```
      Select * from Table1 , Table2
```

Example:
```
      Select * from Supplier , Orders;
      Select * from Supplier , Customers;          -- 9*3 = 27 Rows
Returned
```

==
==

6)Anti Join
Anti-join is used to make the queries run faster. It is a very powerful SQL construct Oracle offers for faster queries.
Anti-join between two tables returns rows from the first table where no matches are found in the second table.
It is opposite of a semi-join. An anti-join returns one copy of each row in the first table for which no match is found .
Anti-joins are written using the NOT EXISTS constructs.

Syntax:
```
      Select * From Supplier Where Not Exists ( Select * from Customers
where customers.id = supplier.id);
```

==
==

7)Semi Join

A semi-join between two tables returns rows that match an EXISTS subquery without duplicating rows from the left side of the predicate when multiple rows on the right side satisfy the criteria of the subquery.

While a semi-join returns one copy of each row in the first table for which at least one match is found, an anti-join returns one copy of each row in the first table for which no match is found.

Syntax:

```
      Select * From Supplier Where Exists ( Select * from Customers where
customers.id = supplier.id);

      Select * From Customers Where Exists ( Select * from Supplier where
customers.id = supplier.id);
```

==
==

8)Self Join

A self join is a join in which a table is joined with itself.

To join a table itself means that each row of the table is combined with itself and with every other row of the table.

The table appears twice in the FROM clause and is followed by table aliases that qualify column names in the join condition.

The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were.

Syntax:

```
      SELECT a.FName, b.Price FROM Supplier a, Supplier b WHERE a.Id =
b.Id;
```

==
==
==
==
==
=======================================

****

Topic :- 13)User Defined Functions   :-
      A function is a subprogram or Subroutine that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block.

```
--Use of Dual
-- SELECT 5 + 3 FROM DUAL; -- Returns 8
```

Method::

```
      Syntax:
            Create or replace Function Function_Name(parameter , ...)
            Return return_Datatype_name
            Is
                  Declare_Section
            Begin
                  <|function Body|>
            End;
            /

      Example:
            CREATE OR REPLACE FUNCTION ADDER(num1  NUMBER , num2 NUMBER)
            RETURN NUMBER
            IS
                  num3 NUMBER;
            BEGIN
                  num3:= num1+num2;
                  RETURN num3;
            END;
            /

            DECLARE
                  n NUMBER;
            BEGIN
                  n:= ADDER(11,22);
                  DBMS_OUTPUT.PUT_LINE('Addition is ' || n);
            END;
            /
```

      Questions on Functions
----

1)To Find Maximum Number

```
      CREATE OR REPLACE FUNCTION FINDMAX(X IN NUMBER , Y IN NUMBER)
      RETURN NUMBER
      IS
      BEGIN
            IF X>Y THEN
                  RETURN X;
            ELSE
                  RETURN Y;
            END IF;
      END;
      /
      CODE:-
            DECLARE
                  A NUMBER;
                  B NUMBER;
                  C NUMBER;
            BEGIN
                  A:=&A;
                  B:=&B;

               C := FINDMAX(A, B);
               DBMS_OUTPUT.PUT_LINE(' MAXIMUM of ( ' || A || ',' || B||')
IS: ' || C);
            END;
            /

      -----------OR---------------

            SELECT FINDMAX(10,2) AS MAXIMUM FROM DUAL;

--------------------------------------------------------------------------
--------
2)To Check Even or Odd
      CREATE FUNCTION EVENODD (num IN NUMBER)
      RETURN VARCHAR2
      IS
      BEGIN
          IF MOD(num, 2) = 0 THEN
              RETURN 'Even';
          ELSE
              RETURN 'Odd';
          END IF;
      END;
      /

      CODE:-
            DECLARE
                  N NUMBER;
            BEGIN
                  N:=&N;
                  DBMS_OUTPUT.PUT_LINE('NUMBER IS  ' || EVENODD(N));
            END;
            /

      -----------OR---------------

      SELECT CheckEvenOdd(7) AS result FROM DUAL;

--------------------------------------------------------------------------
--------
Q3)   To Find Table of a Number
      CREATE OR REPLACE  FUNCTION TABLES(n in NUMBER)
      RETURN NUMBER
      Is
            t NUMBER;
            i NUMBER;
      BEGIN
            i:=1;
            WHILE i<=10 LOOP
                  t := n * i;
```

```
                DBMS_OUTPUT.PUT_LINE(n || ' X ' || i || ' = ' || t);
                i := i + 1;
          END LOOP;
          RETURN NULL;
     END;
     /

     CODE:-
          DECLARE
                N NUMBER;
          BEGIN
                N:=&N;
                DBMS_OUTPUT.PUT_LINE('TABLE OF ' || N || TABLES(N));
          END;
          /

     -----------OR---------------

     SELECT TABLES(5) AS TABLE  FROM DUAL;
```


Q4) To find Factorial of Number

```
     CREATE OR REPLACE FUNCTION FACTORIAL(N IN NUMBER)
     RETURN NUMBER
     IS
          i NUMBER := 1;
          F NUMBER := 1;
     BEGIN
         IF N = 0 THEN
             RETURN 1;
         ELSE
             WHILE I<=N LOOP
                F := F * i;
                END LOOP;
         END IF;
         RETURN F;
     END;
     /

     CODE:-
          DECLARE
                F number;
                N NUMBER;
          BEGIN
                N := &N;
                F := FACTORIAL(N);
                DBMS_OUTPUT.PUT_LINE('Factorial is '||F);
         END;
         /

        ----------------------OR-------------------

        SELECT FACTORIAL(5) AS FACTORIAL FROM DUAL;
```


Q5)To Find Reverse of a Number

```
CREATE OR REPLACE FUNCTION REVNUM(N Number)
RETURN NUMBER
IS
      r NUMBER := 0;
      temp NUMBER := n;
BEGIN
      WHILE temp > 0 LOOP
            r := (r*10) + Mod(temp,10);
            temp := trunc(temp/10);
      End Loop;
      Return r;
End;
/


CODE:-
      DECLARE
            REV NUMBER;
            N NUMBER;
      BEGIN
            N := &NUMBER;
            REV := REVNUM(N);
            DBMS_OUTPUT.PUT_LINE('Reverse  is '|| REV);
    END;
    /

  ----------------------OR-------------------

      SELECT REVNUM(12345) FROM DUAL;
```

****
                              END

****

```
Tables For Reference
      Table_name : Supplier
      Columns:= (Id Number, FName varchar2(20), Address Varchar2(20));

      Table_Name : Orders
      Columns:=(Id Number ,Order_no Number,  City Varchar2(20))
```

```
===========================================================================
==
```

14)CURSORS:-

Cursors in Oracle are like pointers or iterators that help you navigate through the result sets of SQL queries

In simple terms, think of a cursor as a virtual finger that points to a specific row in a table, and you can use it to move through the rows to read or manipulate the data. Cursors are especially useful when you need to work with multiple rows of data returned by a database query in a program or script

Syntax:
```
      Declare
            variable declare;
            Cursor cursor_name IS Select statement;
      Begin
            OPEN cursor_name;
            loop
            Fetch cursor_name into variables_you declared;
            Dbms_output...
            Exit when Cursor_name%NOTFOUND;
            CLOSE cursor_name;
      End;
      /
```

Example:
```
      Declare
            name Varchar2(30);
            ids Number;
            Cursor cur_supp Is
            Select Id , FName from Supplier Order by ID;
      Begin
            Open cur_supp;
            Loop
            Fetch cur_supp into ids , name;
            DBMS_OUTPUT.PUT_LINE('ID: '||ids||',  Name: '|| name);
            Exit When cur_supp%NOTFOUND;
            End Loop;
            Close cur_supp;
      End;
      /
```

Q1) To Show Whole Data of Row of a Table
```
      DECLARE
          row_data SUPPLIER%ROWTYPE;
          CURSOR c1 IS SELECT * FROM SUPPLIER;
      BEGIN
          OPEN c1;
```

```
        LOOP
            FETCH c1 INTO row_data;
            EXIT WHEN c1%NOTFOUND;

            DBMS_OUTPUT.PUT_LINE('Id      : ' || row_data.id);
            DBMS_OUTPUT.PUT_LINE('Name    : ' || row_data.fname);
            DBMS_OUTPUT.PUT_LINE('Address : ' || row_data.address);
            DBMS_OUTPUT.PUT_LINE('Price   : ' || row_data.price);

DBMS_OUTPUT.PUT_LINE('==========================================');
        END LOOP;

        CLOSE c1;
    END;
    /
```

Q2) To Fetch Data Using Cursor For Loop

```
    DECLARE
        -- Declare variables
        row_data SUPPLIER%ROWTYPE;

        -- Declare cursor
        CURSOR c1 IS SELECT * FROM SUPPLIER;
    BEGIN
        -- Open cursor and loop through the results and closes
automatically
        FOR row_data IN c1
        LOOP
            -- Output data
            DBMS_OUTPUT.PUT_LINE('Id      : ' || row_data.id);
            DBMS_OUTPUT.PUT_LINE('Name    : ' || row_data.fname);
            DBMS_OUTPUT.PUT_LINE('Address : ' || row_data.address);
            DBMS_OUTPUT.PUT_LINE('Price   : ' || row_data.price);

DBMS_OUTPUT.PUT_LINE('==========================================');
        END LOOP;
    END;
    /
```

Q3) To Fetch Data of a Specific Row using parameterized Cursor

```
    DECLARE
        row_data SUPPLIER%ROWTYPE;
        CURSOR c1(check_Id NUMBER) IS SELECT * FROM SUPPLIER where ID =
check_Id;
        Input_Id NUMBER;
    BEGIN

        Input_Id := &Id;
        FOR row_data IN c1(Input_ID)
        LOOP
            -- Output data
            DBMS_OUTPUT.PUT_LINE('Id      : ' || row_data.id);
            DBMS_OUTPUT.PUT_LINE('Name    : ' || row_data.fname);
            DBMS_OUTPUT.PUT_LINE('Address : ' || row_data.address);
            DBMS_OUTPUT.PUT_LINE('Price   : ' || row_data.price);

DBMS_OUTPUT.PUT_LINE('==========================================');
        END LOOP;
    END;
```

```
      /
================================================================================
====================
                              END
================================================================================
====================
```

15)TRIGGERS:
An SQL trigger is a database object that is associated with a table and automatically executes a set of SQL
statements when a specific event occurs on that table. Triggers are used to enforce business rules, maintain data integrity,
and automate certain actions within a database.

Reference TABLE NAME : CARS(NAME VARCHAR2(15) , NO_LEFT NUMBER);

```
            CREATE TABLE CARS(NAME VARCHAR2(20) , LEFT NUMBER);
            SELECT * FROM CARS;
            INSERT INTO CARS VALUES('AUDI' , 10);
            INSERT INTO CARS VALUES('FORD MUSTANG' , 5);
            INSERT INTO CARS VALUES('BMW', 15);
```

==
================================
==
================================
TOPIC - 1 :- DATA MANIPULATION LANGUAGE TRIGGER(INSERT , DELETE , UPDATE)

1)BEFORE INSERT
```
      CREATE OR REPLACE TRIGGER bi_cars
      BEFORE INSERT ON CARS
      FOR EACH ROW
      DECLARE
            v_user VARCHAR2(20);
      BEGIN
            SELECT USER INTO v_user FROM DUAL;
            DBMS_OUTPUT.PUT_LINE('A ROW WAS INSERTED BY '|| v_user);
      END;
      /

      EX:- SQL> INSERT INTO CARS VALUES('NISSAN GTR' , 18);
            THIS ROW WAS INSERTED BY SYSTEM
```

2)BEFORE UPDATE
```
      CREATE OR REPLACE TRIGGER bu_cars
      BEFORE UPDATE ON CARS
      FOR EACH ROW
      DECLARE
            v_user VARCHAR2(20);
      BEGIN
            SELECT USER INTO v_user FROM DUAL;
            DBMS_OUTPUT.PUT_LINE('A ROW WAS UPDATED BY '|| v_user);
      END;
      /

      EX:-  SQL> UPDATE CARS SET LEFT = 7 WHERE NAME = 'MERCEDES';
        A ROW WAS UPDATED BY SYSTEM
```

3)BEFORE DELETE
```
      CREATE OR REPLACE TRIGGER bd_cars
      BEFORE DELETE ON CARS
      FOR EACH ROW
      DECLARE
            v_user VARCHAR2(20);
      BEGIN
            SELECT USER INTO v_user FROM DUAL;
            DBMS_OUTPUT.PUT_LINE('A ROW WAS DELETED BY '|| v_user);
      END;
      /

      EX:-  SQL> DELETE CARS  WHERE NAME = 'MERCEDES';
        A ROW WAS DELETED BY SYSTEM
```

4)BEFORE (INSERT , UPDATE AND DELETE) TOGETHER

```
      CREATE OR REPLACE TRIGGER tr_cars
      BEFORE INSERT OR UPDATE OR DELETE ON CARS
      FOR EACH ROW
      DECLARE
            v_user VARCHAR2(20);
      BEGIN
            SELECT USER INTO v_user FROM DUAL;
            IF INSERTING THEN
                  DBMS_OUTPUT.PUT_LINE('A ROW WAS INSERTED BY '|| v_user);

            ELSIF UPDATING THEN
                  DBMS_OUTPUT.PUT_LINE('A ROW WAS UPDATED BY '|| v_user);

            ELSIF DELETING THEN
                  DBMS_OUTPUT.PUT_LINE('A ROW WAS DELETED BY '|| v_user);

            END IF;
      END;
      /
```

```
==========================================================================
===================
                            END
==========================================================================
===================
16) EXCEPTION HANDLING

Types of Exceptions:
      1)System-Defined
      2)User Defined

      1)System Defined Exceptions
            System Defined Exceptions are defined and maintained
implicitly by the Oracle Server.
            These Eceptions are Mainly Defined in the Oracle Standard
Package


      2)User Defined Exceptions
            These Exceptions are Raised Explicitly in the PL/SQL blocks

      We can declare user Defined PL/SQL exceptions in 3 Ways.

      a)Using Variable Exception Type
            Exception_name Exception ;

      b)Using PRAGMA EXCEPTION_INIT function
            Using PRAGMA EXCEPTION_INIT function you can map a non-
predifined error number with the variable of Exception datatype

      c)Using RAISE_APPLICATION_ERROR method.
            Using this Method you can declare a user defined Exception
With Your own Customized Error Number and Message




Q1) To Raise a Exception of Not to Divide By Zero
      DECLARE
            dividend NUMBER;
            divisor  NUMBER;
            result   NUMBER;
            ex_DivZero Exception;
      BEGIN
            dividend := &dividend;
            divisor := &divisor;
            IF divisor = 0 THEN
                  RAISE ex_DivZero;
            END If;
            result := (dividend/divisor);
            DBMS_OUTPUT.PUT_LINE('RESULT : ' || result);

            EXCEPTION
            WHEN ex_DivZero THEN
                  DBMS_OUTPUT.PUT_LINE('ERROR :- YOUR DIVISOR IS ZERO ');
            WHEN OTHERS THEN
                  DBMS_OUTPUT.PUT_LINE('Error');
      END;
      /


Q2) Handle the No_DATA_Found Exception

      DECLARE
            name VARCHAR(20);
      BEGIN
            BEGIN
                  SELECT FNAME INTO NAME FROM SUPPLIER WHERE ID = 12 ;
                  DBMS_OUTPUT.PUT_LINE('COMPANY NAME ' ||  NAME );
            EXCEPTION
                  WHEN NO_DATA_FOUND THEN
                  DBMS_OUTPUT.PUT_LINE('ERROR : NO MATCHING RECORD 
FOUND');
            END;
      END;
      /
```

Q3) TO Handle Invalid_NUmber Exception

```
      DECLARE
            v_input Varchar2(20) := 'abc';
            v_number NUMBER;
      BEGIN
            BEGIN
                   v_number := TO_NUMBER(v_input);
                   DBMS_OUTPUT.PUT_LINE('CONVERSION SUCCESSFUL');
            EXCEPTION
                  WHEN INVALID_NUMBER THEN
                        DBMS_OUTPUT.PUT_LINE('ERROR : INVALID NUMNER');
            END;
      EXCEPTION
            WHEN OTHERS THEN
                  DBMS_OUTPUT.PUT_LINE('ERROR IS : ' || SQLERRM);
      END;
      /
```

Q4) Handle The Program_Error Exception When PL SQL Encounters an Internal Error

```
      DECLARE
            program_error EXCEPTION;
            error_message Varchar2(4000);
            x number:= 15;
      BEGIN
            IF x>10 THEN
                  RAISE program_error;

            END IF;

            EXCEPTION
            WHEN program_error THEN
                  error_message := 'ERROR IS ' || SQLERRM;
                  DBMS_OUTPUT.PUT_LINE(error_message);
      END;
      /
```

==
===================
END
==
===================

17)COLLECTIONS:-
 A collection is an ordered group of elements having the same data type. Each element is identified by a unique subscript that represents its position in the collection.

```
 ************************ FUNCTIONS ****************************
      1) COUNT
      2) EXISTS
      3) FIRST , LAST
      4) LIMIT
      5) PRIOR, NEXT

      ************************ PROCEDURES *****************************
      1)DELETE
      2)EXTEND
      3)TRIM




      ==============================================================
      1)Count
           Gives the total count of the elements present in a collection
           Code :
           DECLARE
                TYPE my_nested_table IS TABLE OF NUMBER;
                var_nt my_nested_table :=
my_nested_table(2,4,6,8,10,12,14,16,18,20);

           BEGIN
                DBMS_OUTPUT.PUT_LINE('The Size of Nested Table is ' ||
var_nt.COUNT);
           END;
           /

      2)Exists and EXTEND
           Exists: This method will return Boolean results. It will
return 'TRUE' if the nth element exists in that collection, else it will
return FALSE

           Extend : Extends one element in a collection at the end

           Code :
           DECLARE
                TYPE my_nested_table IS TABLE OF NUMBER;
                var_nt my_nested_table :=
my_nested_table(5,10,15,20,25,30);
                I NUMBER;
                val NUMBER;
           BEGIN
                I := &INDEX;
                IF var_nt.EXISTS(I) THEN
                     DBMS_OUTPUT.PUT_LINE('VALUE STORED AT INDEX 1 IS '
|| var_nt(I));
                ELSE
                     DBMS_OUTPUT.PUT_LINE('SORRY ! NO DATA AT THIS
INDEX INSERTING NEW DATA USING EXTEND');
                     var_nt.EXTEND;
                     val := &insert;
                     var_nt(I) := val;
```

```
                    DBMS_OUTPUT.PUT_LINE('VALUE STORED AT INDEX ' || I
|| ' IS ' || var_nt(I));
                END IF;
           END;
           /
```

      3) Limit
           It returns the maximum size of the collection. For Varray, it will return the fixed size that has been defined. For Nested table and Index-by-table, it gives NULL

```
           Code :
           DECLARE
                TYPE students_names IS VARRAY(6) OF VARCHAR2(20);
                num NUMBER;

                names students_names;
           BEGIN
                names := students_names('Kavita', 'Pritam', 'Ayan',
'Rishav', 'Aziz');
                num := names.LIMIT;
                DBMS_OUTPUT.PUT_LINE('SIZE OF ARRAY ' || num );
                FOR i in 1 .. total LOOP
                  DBMS_OUT.put_line('Student: ' || names(i) || ' Marks:
' || marks(i));
               END LOOP;
           END;
           /
```

      4) Prior And Next
           Prior: Returns precedes index variable in a collection of the nth element. If there is no precedes index value NULL is returned

           Next : Returns succeeds index variable in a collection of the nth element. If there is no succeeds index value NULL is returned

```
           Code :
           DECLARE
                Type my_nested_table IS TABLE OF NUMBER;
                var_nt my_nested_table :=
my_nested_table(6,12,18,24,30,36,42,48,54,60);
           BEGIN
                DBMS_OUTPUT.PUT_LINE('Previous Index to Index 3 : ' ||
var_nt.PRIOR(3));
                DBMS_OUTPUT.PUT_LINE('Value at Previous Index before
Index 3 : ' || var_nt(var_nt.PRIOR(3)));

                DBMS_OUTPUT.PUT_LINE('Next Higher Index to Index 3 : '
|| var_nt.NEXT(3));
                DBMS_OUTPUT.PUT_LINE('Value at Index after Index 3 : '
|| var_nt(var_nt.NEXT(3)));
           END;
           /
```

PL/SQL provides three collection types –

1)Index-by tables or Associative array
2)Nested table

3)Variable-size array or Varray

1) Associative Array
     An index-by table (also called an associative array) is a set of key-value pairs. Each key is unique and is used to locate the corresponding value. The key can be either an integer or a string.

```
     TYPE type_name IS TABLE OF element_type
     INDEX BY datatype;
     table_name type_name;
```

Q1)To Store Names of Countries and Capitals and Print it using Loop

```
     DECLARE

           -- declaration of Assocative Array
           TYPE country IS TABLE OF VARCHAR2(20)
                 INDEX BY VARCHAR2(20);
           capitals country;

           flag varchar2(20);

     BEGIN
           capitals('India') := 'Delhi';
           capitals('Japan') := 'Tokyo';
           capitals('Afghanistan')     := 'Kabul';
           capitals('Brazil') := 'Brasilia';

           flag := capitals.FIRST;
           WHILE flag IS NOT NULL
           LOOP
                 DBMS_OUTPUT.PUT_LINE('Key --> ' || flag || '  , Value --
> '|| capitals(flag));
                 flag := capitals.NEXT(flag);
           END LOOP;
     END;
     /
```

Q2)To Store EMployees Names and Salaries and Print it using Loop

```
     DECLARE
           TYPE workers IS TABLE OF NUMBER
                 INDEX BY VARCHAR2(20);

           employ workers;

           flag varchar2(20);

     BEGIN
           employ('Suresh') := 15000;
           employ('Ramesh') := 17200;
           employ('Prachi') := 19000;
           employ('AJit')   := 19200;
           employ('Suraj')  := 25000;

           flag := employ.FIRST;
           WHILE flag IS NOT NULL
           LOOP
```

```
                DBMS_OUTPUT.PUT_LINE('Employee Name = ' || flag || '  ,
Salary = '|| employ(flag));
                flag := employ.NEXT(flag);
          END LOOP;
     END;
     /
```

==
====================================
2) Nested Table
     A nested table is like a one-dimensional array with an arbitrary number of elements

     a)An array has a declared number of elements, but a nested table does not. The size of a nested table can increase dynamically.
     b)An array is always dense, i.e., it always has consecutive subscripts. A nested array is dense initially, but it can become sparse when elements are deleted from it.

```
     syntax :-
          TYPE type_name IS TABLE OF element_type [NOT NULL];
          table_name type_name;
```

Q1)To Print a Nested Table

```
          DECLARE
          TYPE my_nested_table IS TABLE OF NUMBER;
          var_nt my_nested_table :=
my_nested_table(9,18,27,36,45,54,63,72,81,90);
     BEGIN
          DBMS_OUTPUT.PUT_LINE('NESTED TABLE FIRST ITEM(GIVES INDEX
NUMBER) ' || var_nt.FIRST);
          DBMS_OUTPUT.PUT_LINE('NESTED TABLE LAST ITEM(GIVES INDEX
NUMBER)  ' || var_nt.LAST);

          FOR I IN 1..var_nt.COUNT
          LOOP
                DBMS_OUTPUT.PUT_LINE('VALUE STORED AT INDEX ' || I || '
IS ' || var_nt(I));
          END LOOP;
     END;
     /
```

Q2)TO Store Data of Students Name , Roll No using one than more Nested Table

```
     DECLARE
          TYPE names_table IS TABLE OF varchar2(20);
          TYPE roll_no_table IS TABLE OF NUMBER;
          var_nt1 names_table;
          var_nt2 roll_no_table;
     BEGIN

          var_nt1 := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav',
'Aziz' , 'Mayuresh' , 'aditya' , 'Vedant' , 'Soham');

          var_nt2 := roll_no_table(1,2,3,4,5,6,7,8,9,10);

          DBMS_OUTPUT.PUT_LINE('TOTAL STUDENTS : ' || var_nt1.COUNT);
```

```
           FOR I IN 1..var_nt1.COUNT
           LOOP
                 DBMS_OUTPUT.PUT_LINE('Name: ' || var_nt1(i) || ' ,  Roll
no ' || var_nt2(I));
           END LOOP;
     END;
     /
```

3) Varray (3rd Type of Collection) is Completed in Module 6 of Varrays
Its Same only

==
===================

END

==
===================

===
===========
18) BULK COLLECT CLAUSE:
These are SELECT statements that retrieve multiple rows with a single fetch, thereby improving the speed of data retrieval.

===
==
Information

The main purpose of using “BULK COLLECT” is to increase the performance of the process by reducing the interaction between database and PL/SQL engine. Which helps us when we have lots of data in that case, we can use BULK COLLECT to reduces context switches between SQL and PL/SQL engine which will allows SQL engine to fetch all the records at once.

It can be used with all three types of collections: associative arrays, nested tables, and arrays. You can fetch into individual collections (one for each expression in the SELECT list) or a single collection of records.

1) TO Use BULK COLLECT With SELECT INTO and Nested Table
```
      DECLARE
            TYPE nt_table1 IS TABLE OF VARCHAR2(20);
            fname nt_table1;

            TYPE nt_table2 IS TABLE OF NUMBER;
            Fid nt_table2;
      BEGIN
            SELECT ID , NAME BULK COLLECT INTO Fid , fname FROM EMPLOYEES;

            FOR I IN 1..FNAME.COUNT
            LOOP
                  DBMS_OUTPUT.PUT_LINE('ID : ' || Fid(I) || ' ----> NAME
:' || FNAME(I));
            END LOOP;
      END;
      /
```

2)TO Use BULK COLLECT With Cursor and Nested Table
```
      DECLARE
            CURSOR EXp_CUR IS
            SELECT NAME FROM EMPLOYEES;

            TYPE nt_name IS TABLE OF VARCHAR2(20);
            fname nt_name;
            I NUMBER;
      BEGIN
            OPEN EXP_CUR;
            LOOP
                  FETCH EXP_CUR BULK COLLECT INTO Fname;
                  EXIT WHEN FNAME.COUNT = 0;

                  FOR I IN FNAME.FIRST .. FNAME.LAST
                  LOOP
                        DBMS_OUTPUT.PUT_LINE('ID : ' || I || '------> NAME
: ' || FNAME(I));
                  END LOOP;
            END LOOP;
            CLOSE EXP_CUR;
      END;
      /
```

3)To Use Bulk Collect Using Limit Clause

Limit Clause Can be Used With FETCH Into

Limit Clause Cannot be used with select-into

```
      DECLARE
            CURSOR EXp_CUR IS
            SELECT NAME FROM EMPLOYEES;

            TYPE nt_name IS TABLE OF VARCHAR2(20);
            fname nt_name;
            I NUMBER;
      BEGIN
            OPEN EXP_CUR;
            FETCH EXP_CUR BULK COLLECT INTO Fname LIMIT 10;
            CLOSE EXP_CUR;

            FOR I IN 1 .. FNAME.COUNT
            LOOP
                  DBMS_OUTPUT.PUT_LINE('NAME : ' || FNAME(I));
            END LOOP;
      END;
      /
```

==
==

END

==
==