DATABASE SYSTEMS (SW215)

PROCEDURAL LANGUAGE / SQL

By : HIRA NOMAN

INTRODUCTION

- PL/SQL is a powerful, yet straightforward database programming language.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.
- It is easy to both write and read and comes packed with lots of out-of-the-box optimizations and security features.
- Utilizing PL/SQL to perform database-specific operations, most notably SQL statement execution, offers several advantages, though, including tight integration with SQL, improved performance through reduced network traffic, and portability (PL/SQL programs can run on any Oracle Database instance).

BLOCK STRUCTURE

```
DECLARE
   <declarations section>
BEGIN
   <executable command(s)>
EXCEPTION
  <exception handling>
END ;
```

PL/SQL is a block-structured language ; this means that the PL/SQL programs are divided and written in logical blocks of code.

DECLARATIONS

This section starts with the keyword **DECLARE**. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

EXECUTABLE COMMANDS

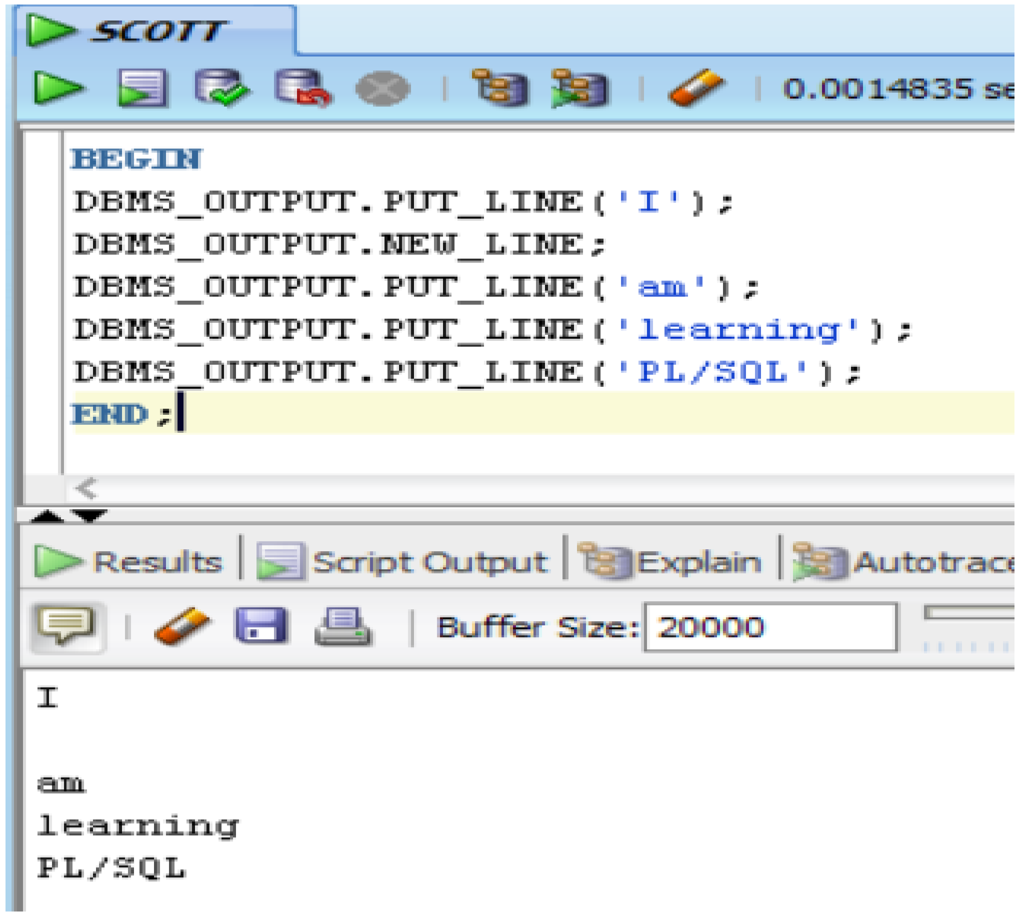
This section is enclosed between the keywords **BEGIN** and **END** and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

EXCEPTION HANDLING

This section starts with the keyword **EXCEPTION**. This optional section contains exception(s) that handle errors in the program

THE OUTPUT PACKAGE

- DBMS_OUTPUT is a package used for producing the output.
- PUT_LINE is the procedure .



```
BEGIN
DBMS_OUTPUT.PUT_LINE('I');
DBMS_OUTPUT.NEW_LINE;
DBMS_OUTPUT.PUT_LINE('am');
DBMS_OUTPUT.PUT_LINE('learning');
DBMS_OUTPUT.PUT_LINE('PL/SQL');
END;
```

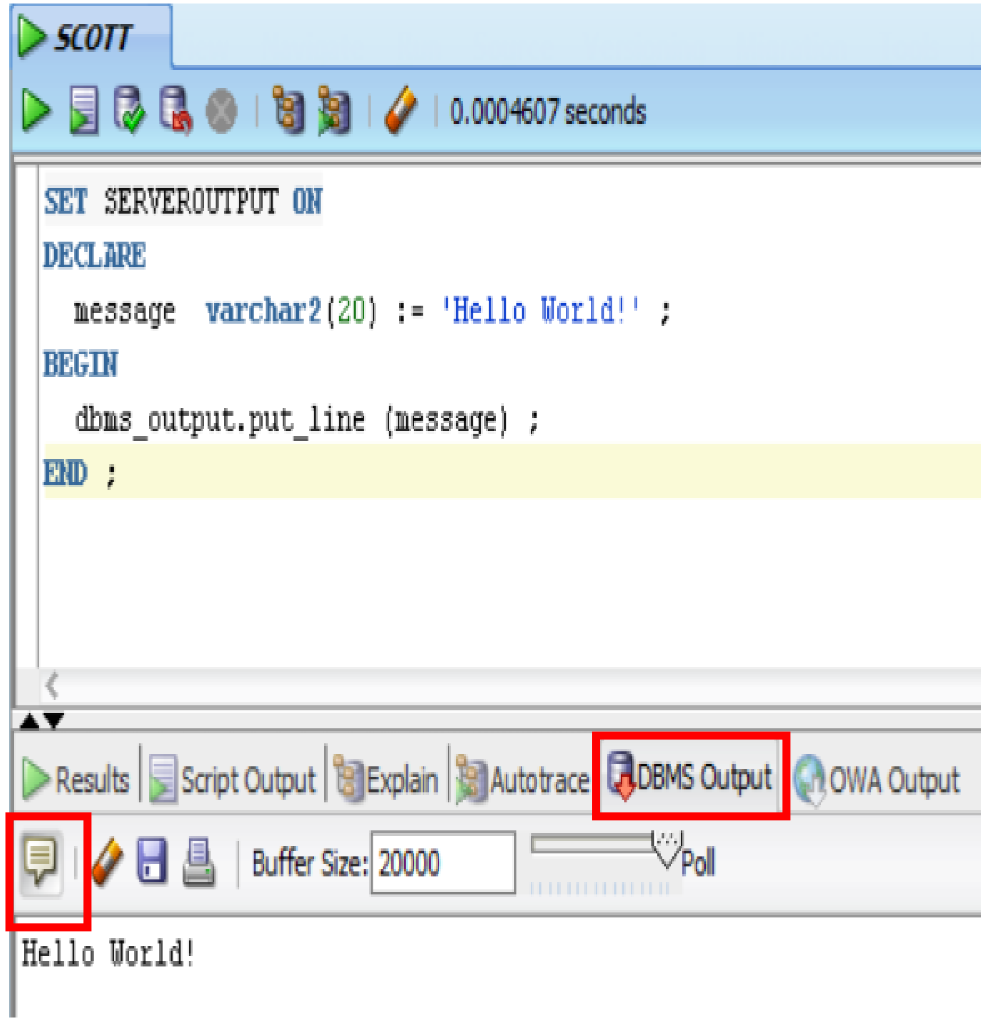
```
I

am
learning
PL/SQL
```

The 'Hello World' Example

```
DECLARE
        message  varchar2(20) := 'Hello World!' ;
BEGIN
        dbms_output.put_line (message) ;
END ;
/
```

- The **END;** line signals the end of the PL/SQL block.
- To run the code from the SQL command line, you may need to type **/** at the beginning of the first blank line after the last line of the code.



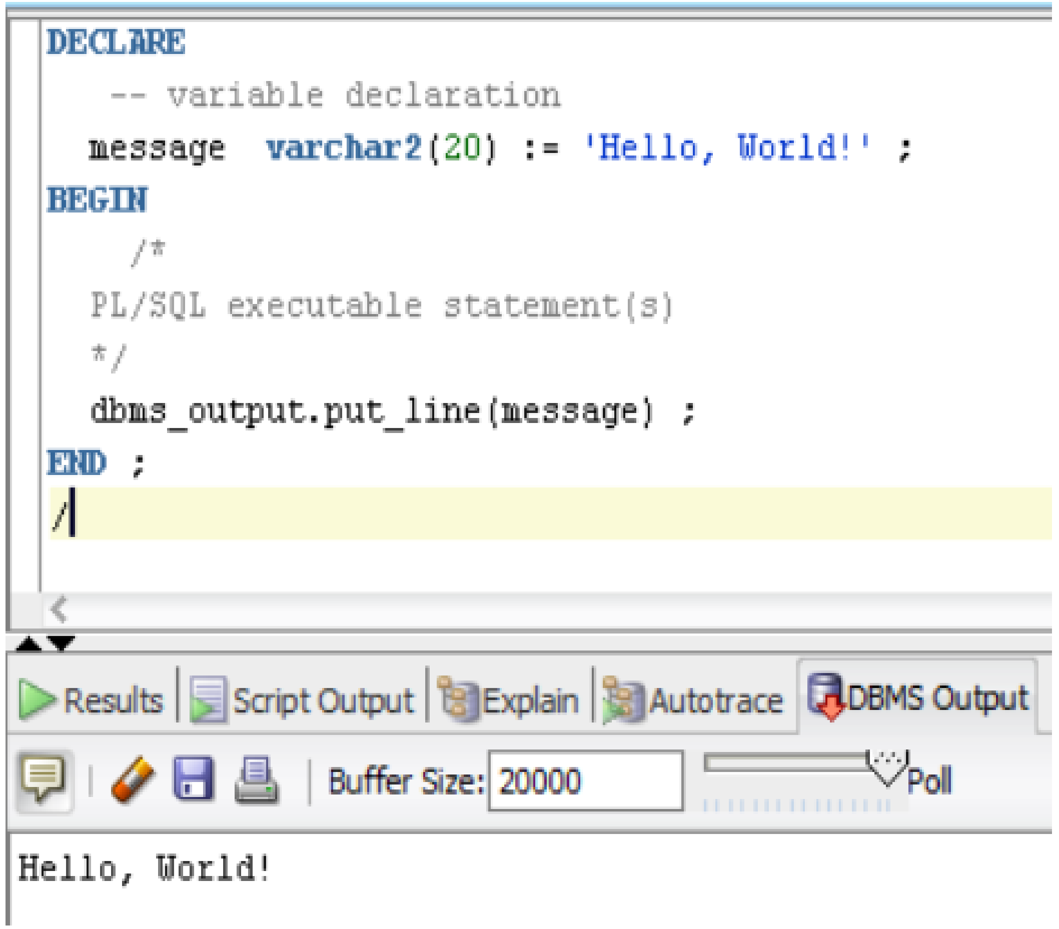
```
SET SERVEROUTPUT ON
DECLARE
  message  varchar2(20) := 'Hello World!' ;
BEGIN
  dbms_output.put_line (message) ;
END ;
```

```
Hello World!
```

THE PL/SQL COMMENTS

- The PL/SQL supports single-line and multi-line comments. All characters available inside any comment are ignored by the PL/SQL compiler.
- The PL/SQL single-line comments start with the delimiter -- (double hyphen) and multi-line comments are enclosed by /* and */.

```
DECLARE
         -- variable declaration
        message  varchar2(20) := 'Hello, World!' ;
BEGIN
          /*
          PL/SQL executable statement(s)
        */
        dbms_output.put_line(message) ;
END ;
/
```



```
DECLARE
   -- variable declaration
  message  varchar2(20) := 'Hello, World!' ;
BEGIN
    /*
  PL/SQL executable statement(s)
  */
  dbms_output.put_line(message) ;
END ;
/
```

```
Hello, World!
```

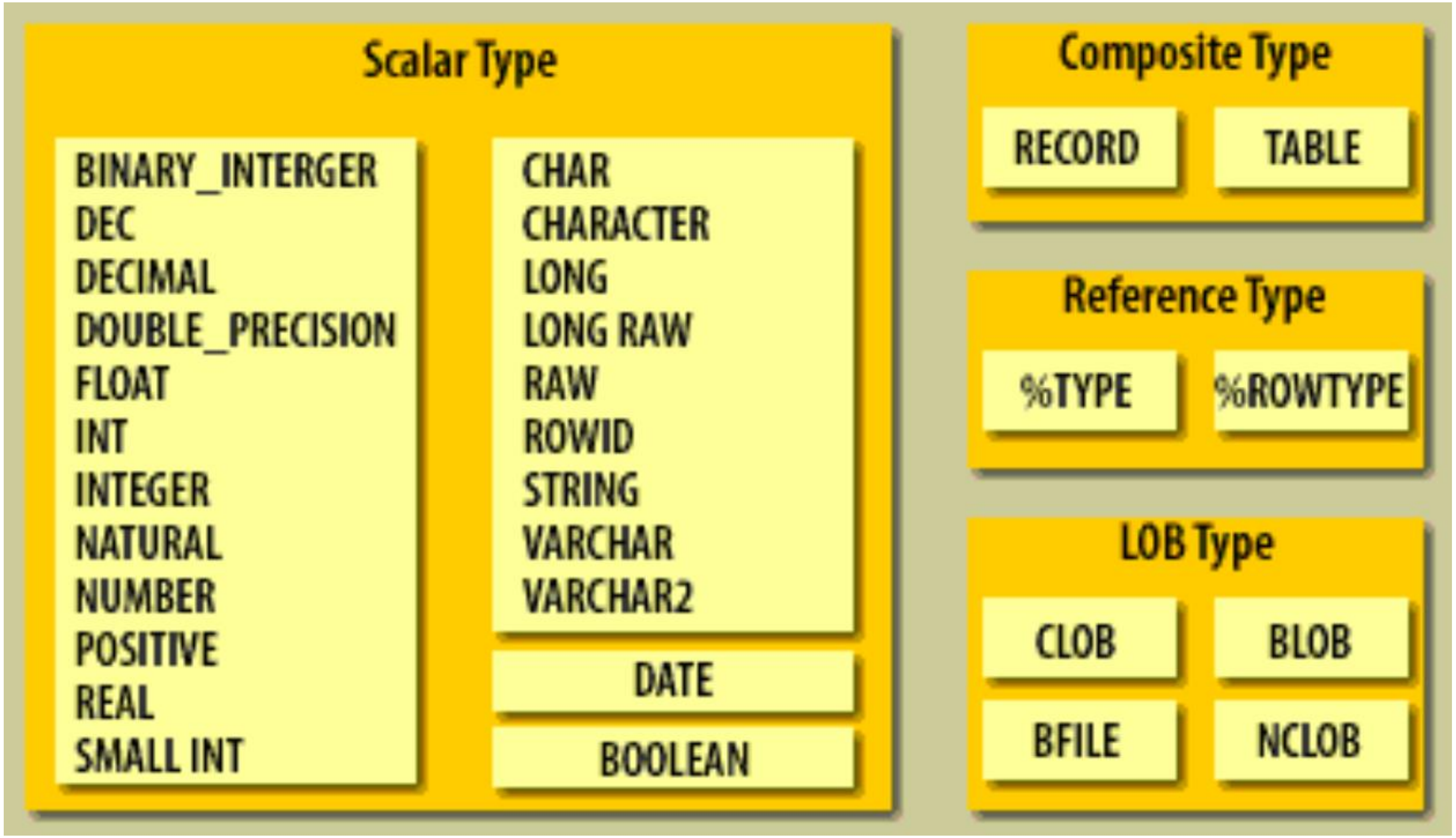
PL/SQL DATA TYPES

1. SCALAR: Single values with no internal components, such as a NUMBER, DATE, or BOOLEAN.

2. COMPOSITE: Data items that have internal components that can be accessed individually.

3. LARGE OBJECT (LOB): Pointers to large objects that are stored separately from other data items, such as text, graphic images, video clips, and sound waveforms.

4. REFERENCE: Pointers to other data items.


Scalar Type
BINARY_INTERGER
DEC
DECIMAL
DOUBLE_PRECISION
FLOAT
INT
INTEGER
NATURAL
NUMBER
POSITIVE
REAL
SMALL INT
CHAR
CHARACTER
LONG
LONG RAW
RAW
ROWID
STRING
VARCHAR
VARCHAR2
DATE
BOOLEAN
Composite Type
RECORD
TABLE
Reference Type
%TYPE
%ROWTYPE
LOB Type
CLOB
BLOB
BFILE
NCLOB

SCALAR DATA TYPES

1. Numeric: Numeric values on which arithmetic operations are performed. It includes sub types such as number, decimal, real, float, etc.

2. Character: Character values on which character operations such as strings are performed. It includes sub types such as char, varchar, varchar2, nvarchar2, etc.

3. Data and Time: This data type is used to store fixed data type which displays and saves time and date values. The default data format saved into the database might be ‘DD-MM-YY’. However, you can change and alter the position of the terms accordingly.

4. Boolean: These include logical values on which logical operations are performed. The logical values are the Boolean values TRUE and FALSE and the value NULL. However, SQL has no data type equivalent to BOOLEAN. It cannot be used in SQL Statements.

5. Rowid: Physical row identifier, the address of a row in an ordinary table.

LOB DATA TYPE

1. **Binary File (Bfile):** Used to store large binary objects in operating system files outside the database System-dependent. Cannot exceed 4GB.

2. **Binary Large Object (Blob):** Used to store large binary objects in the database. Memory Capacity: 8 to 128 TB.

3. **Character Large Object (Clob):** Used to store large blocks of character data in the database. Memory Capacity: 8 to 128 TB.

4. **National Language Character Large Object (Nclob):** Used to store large blocks of NCHAR data in the database. Memory Capacity: 8 to 128 TB.

VARIABLE DECLARATION

- The name of a PL/SQL variable consists of a letter optionally followed by more letters, numerals, dollar signs, underscores, and number signs and should not exceed 30 characters.
- By default, variable names are not case-sensitive.
- You cannot use a reserved PL/SQL keyword as a variable name.
- PL/SQL variables must be declared in the declaration section.
- When you declare a variable, PL/SQL allocates memory for the variable's value and the storage location is identified by the variable name.

SYNTAX:

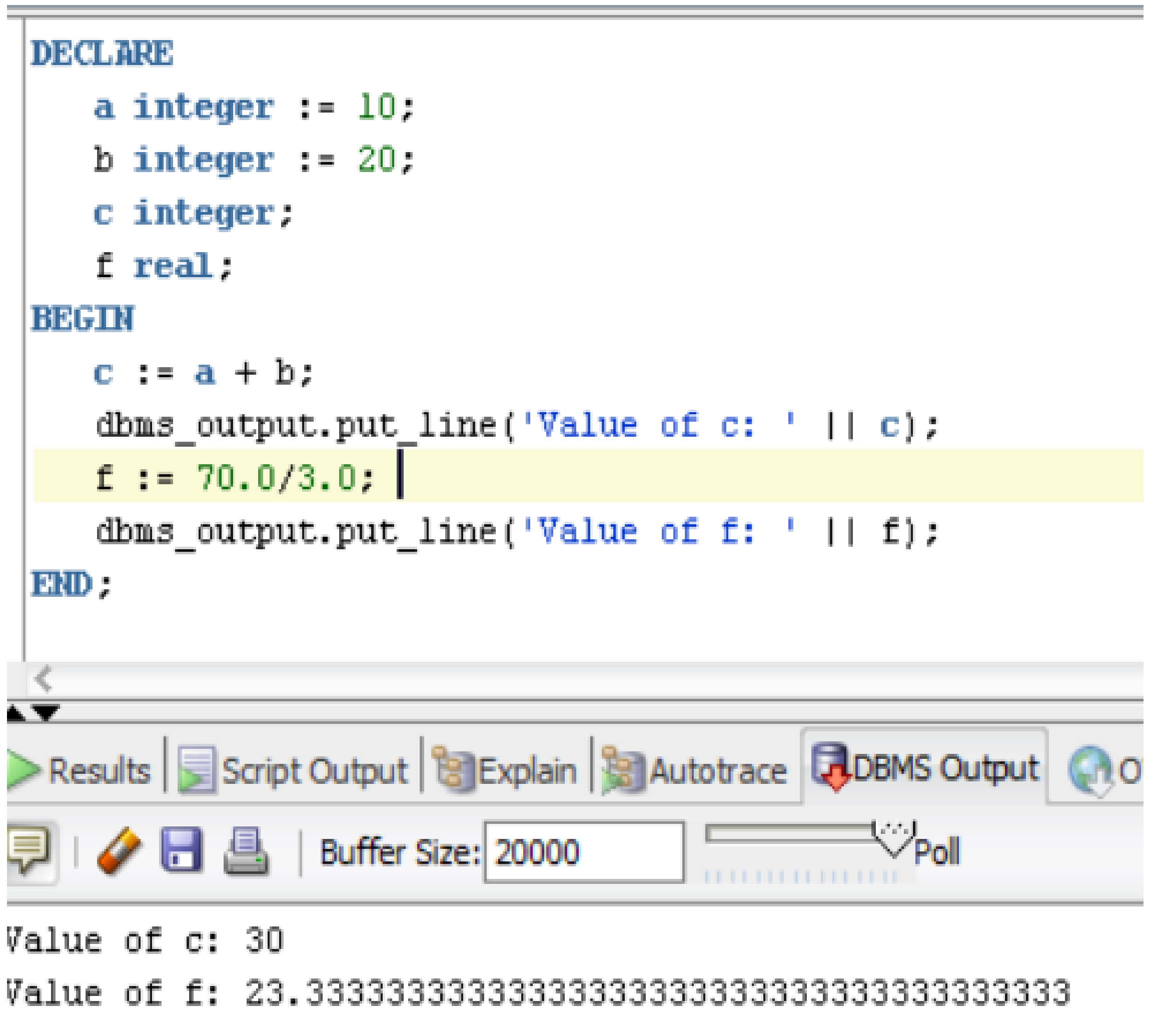
```
variable_name [CONSTANT] datatype [ NOT NULL ] [:= | DEFAULT initial_value ] ;
```

EXAMPLES:

```
sales number(10, 2) ;
pi CONSTANT float := 3.1415 ;
name varchar2(25) ;
address varchar2(100) ;
counter binary_integer := 0;
greetings varchar2(20) DEFAULT 'Have a Good Day';
```

- When you provide a size, scale or precision limit with the data type, it is called a constrained declaration.
- Constrained declarations require less memory than unconstrained declarations.

```
DECLARE
   a integer := 10;
   b integer := 20;
   c integer;
   f real;
BEGIN
   c := a + b;
   dbms_output.put_line('Value of c: ' || c);
   f := 70.0/3.0;
   dbms_output.put_line('Value of f: ' || f);
END;
/
```

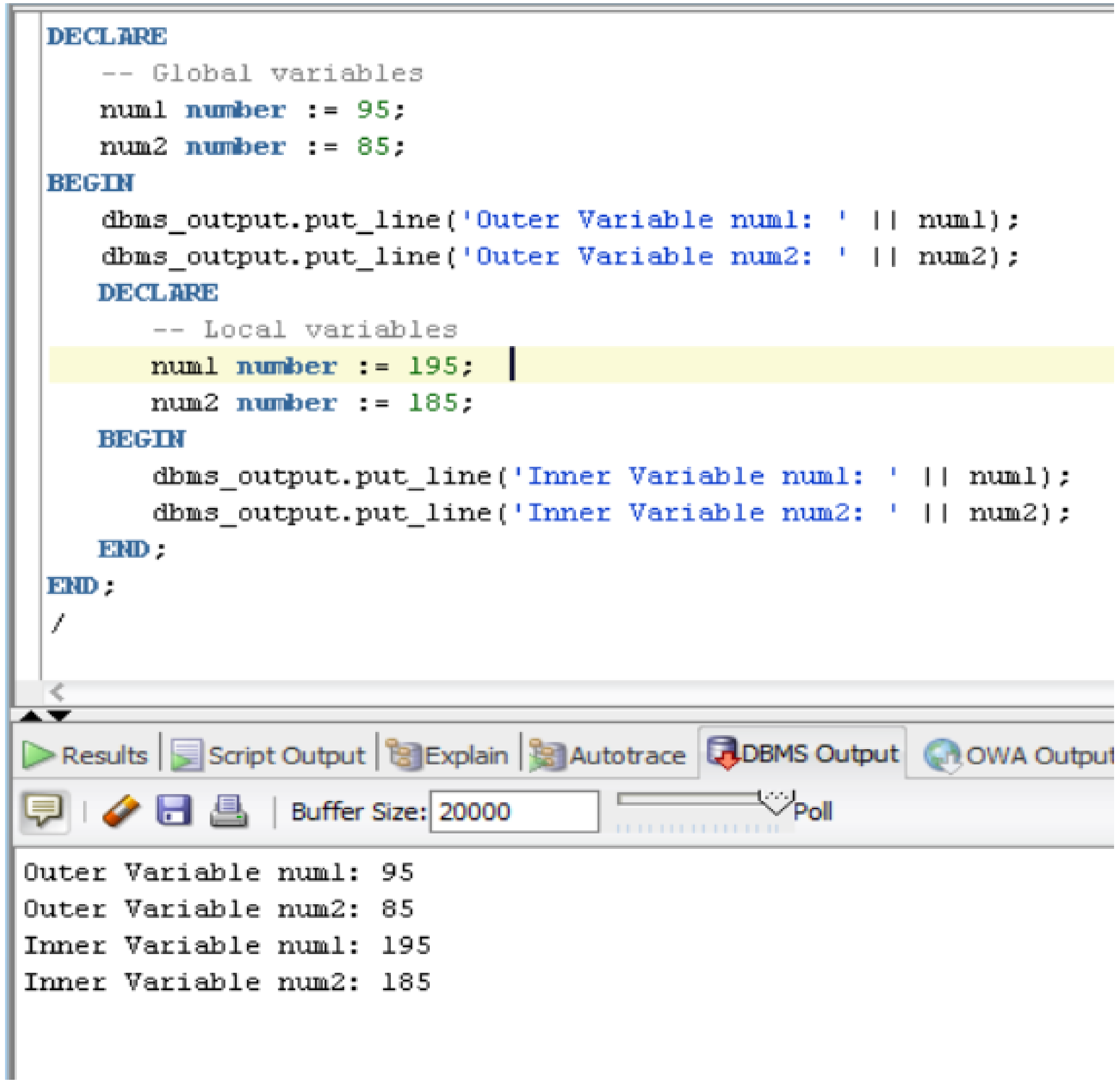


```
DECLARE
    a integer := 10;
    b integer := 20;
    c integer;
    f real;
BEGIN
    c := a + b;
    dbms_output.put_line('Value of c: ' || c);
    f := 70.0/3.0;
    dbms_output.put_line('Value of f: ' || f);
END;
```

```
Value of c: 30
Value of f: 23.33333333333333333333333333333333333333
```

GLOBAL AND LOCAL VARIABLES

```
DECLARE
  -- Global variables
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Outer Variable num1: ' || num1);
  dbms_output.put_line('Outer Variable num2: ' || num2);
  DECLARE
    -- Local variables
    num1 number := 195;
    num2 number := 185;
  BEGIN
    dbms_output.put_line('Inner Variable num1: ' || num1);
    dbms_output.put_line('Inner Variable num2: ' || num2);
  END;
END;
/
```



```
DECLARE
   -- Global variables
   num1 number := 95;
   num2 number := 85;
BEGIN
   dbms_output.put_line('Outer Variable num1: ' || num1);
   dbms_output.put_line('Outer Variable num2: ' || num2);
   DECLARE
      -- Local variables
      num1 number := 195;
      num2 number := 185;
   BEGIN
      dbms_output.put_line('Inner Variable num1: ' || num1);
      dbms_output.put_line('Inner Variable num2: ' || num2);
   END;
END;
/
```

```
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185
```

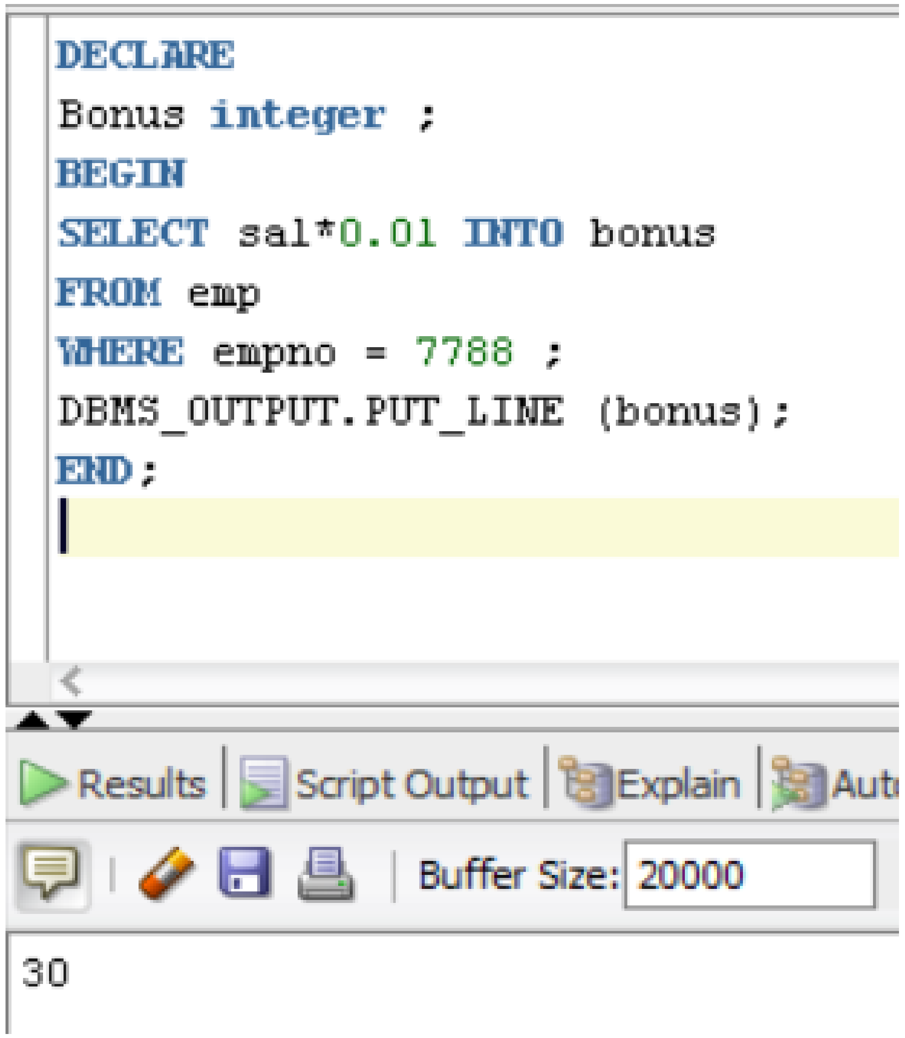
PL/SQL SELECT STATEMENT

SYNTAX:

```
SELECT col1,col2,col3 INTO var1,var2,var3
FROM table_name
[ WHERE condition ] ;
```

EXAMPLE

```
DECLARE
Bonus INTEGER ;
BEGIN
SELECT sal*0.01 INTO bonus
FROM emp
WHERE empno = 7788 ;
DBMS_OUTPUT.PUT_LINE (bonus) ;
END ;
/
```



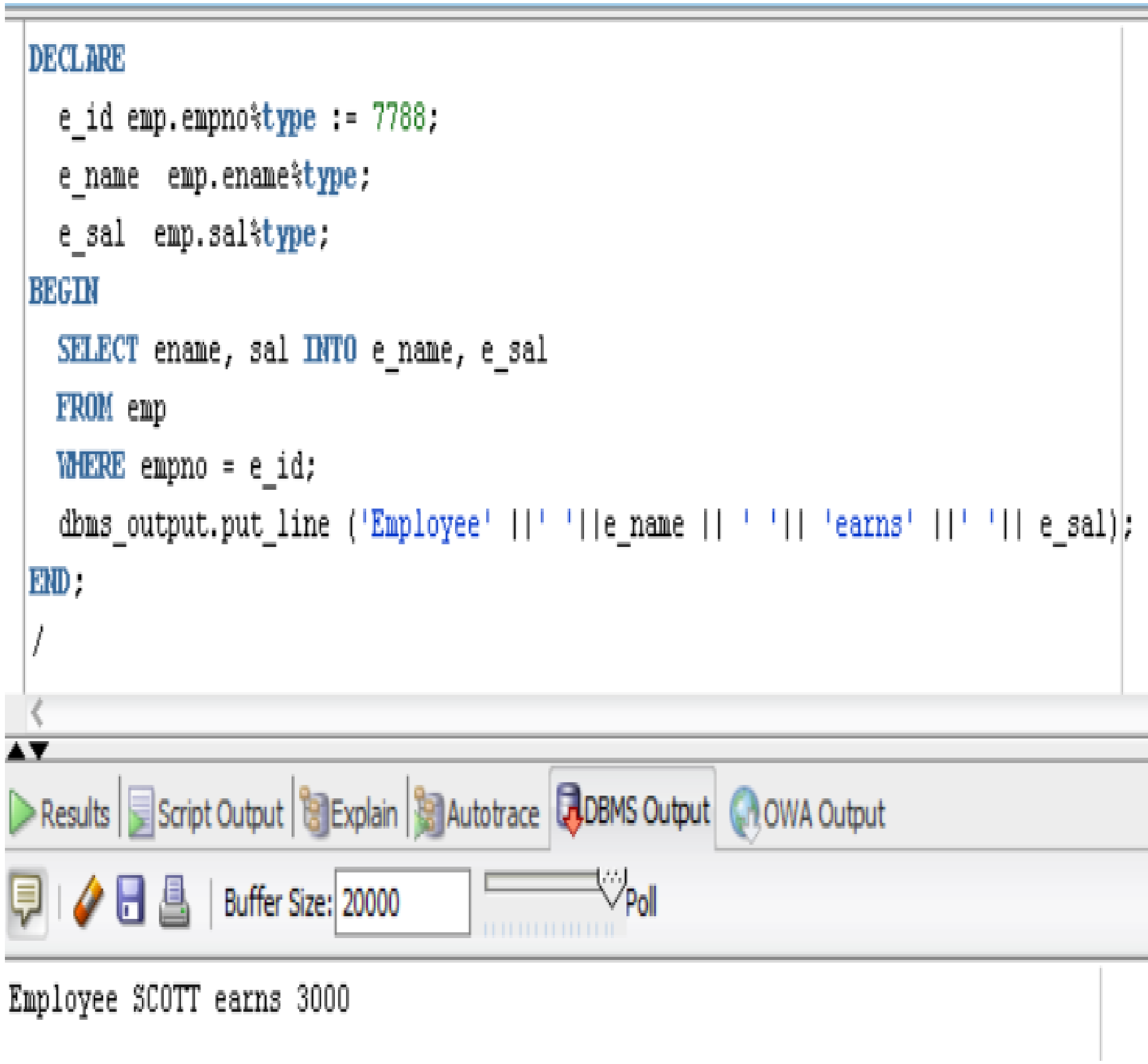
```
DECLARE
Bonus integer ;
BEGIN
SELECT sal*0.01 INTO bonus
FROM emp
WHERE empno = 7788 ;
DBMS_OUTPUT.PUT_LINE (bonus);
END;
```

Buffer Size: 20000

```
30
```

REFERENCE TYPE VARIABLES

```
DECLARE
        e_id emp.empno %type := 7788;
        e_name  emp.ename %type;
        e_sal  emp.sal %type;
BEGIN
        SELECT ename, sal INTO e_name,
        e_sal
        FROM emp
        WHERE empno = e_id;
        dbms_output.put_line (‘Employee
        ‘ ||e_name ||          ' earns ' ||
        e_sal);
END;
/
```

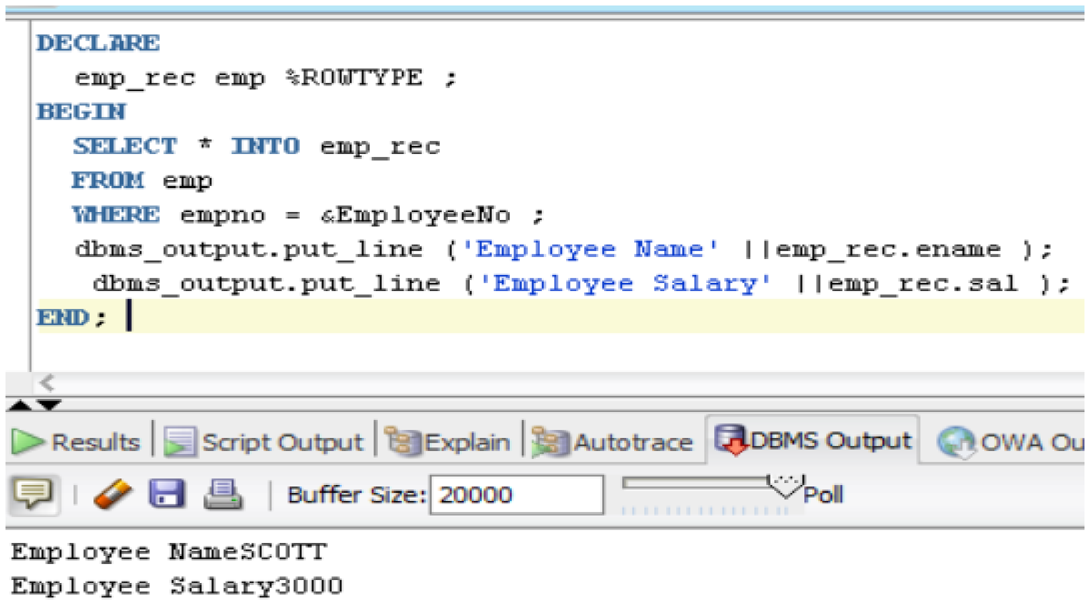


```
DECLARE
  e_id emp.empno%type := 7788;
  e_name  emp.ename%type;
  e_sal  emp.sal%type;
BEGIN
  SELECT ename, sal INTO e_name, e_sal
  FROM emp
  WHERE empno = e_id;
  dbms_output.put_line ('Employee' ||' '||e_name || ' '|| 'earns' ||' '|| e_sal);
END;
/
```

```
Employee SCOTT earns 3000
```

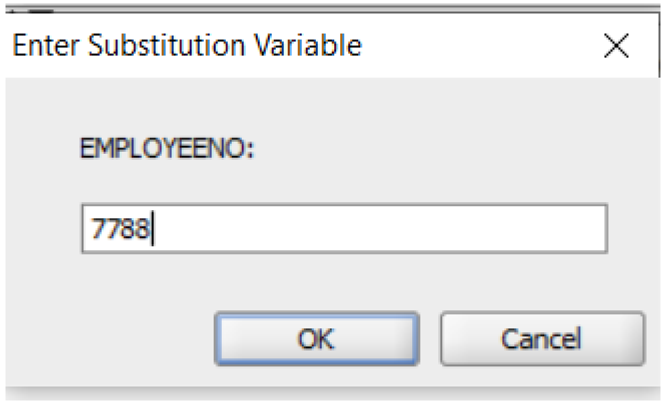
REFERENCE TYPE VARIABLES

```
DECLARE
        emp_rec emp %ROWTYPE ;
BEGIN
        SELECT * INTO emp_rec
        FROM emp
        WHERE empno = &EmployeeNo ;
        dbms_output.put_line (‘Employee Name’ ||emp_rec.ename );
         dbms_output.put_line (‘Employee Salary’ ||emp_rec.sal );
END;
/
```



```
DECLARE
  emp_rec emp %ROWTYPE ;
BEGIN
  SELECT * INTO emp_rec
  FROM emp
  WHERE empno = &EmployeeNo ;
  dbms_output.put_line ('Employee Name' ||emp_rec.ename );
   dbms_output.put_line ('Employee Salary' ||emp_rec.sal );
END;
```

```
Employee NameSCOTT
Employee Salary3000
```



Enter Substitution Variable

EMPLOYEENO: 7788

OPERATORS

LIKE	The LIKE operator compares a character, string, or CLOB value to a pattern and returns TRUE if the value matches the pattern and FALSE if it does not.	If ‘Ahmed Ali' like ‘A% A_i' returns a Boolean true, whereas 'Nuha Ali' like ‘A% A_i' returns a Boolean false.
BETWEEN	The BETWEEN operator tests whether a value lies in a specified range. x BETWEEN a AND b means that x >= a and x <= b.	If x = 10 then, x between 5 and 20 returns true, x between 5 and 10 returns true, but x between 11 and 20 returns false.
IN	The IN-operator tests set membership. x IN (set) means that x is equal to any member of set.	If x = 'm' then, x in ('a', 'b', 'c') returns Boolean false but x in ('m', 'n', 'o') returns Boolean true.
IS NULL	The IS NULL operator returns the BOOLEAN value TRUE if its operand is NULL or FALSE if it is not NULL. Comparisons involving NULL values always yield NULL.	If x = 'm', then 'x is null' returns Boolean false.

PL/SQL CONTROL STRUCTURES

1. CONDITIONAL CONTROL STRUCTURES.
2. ITERATIVE CONTROL STRUCTURES.
3. SEQUENTIAL CONTROL STRUCTURES.

CONDITIONAL CONTROL STRUCTURES

S.No	Statement & Description
1	IF-THEN statement: The **IF statement** associates a condition with a sequence of statements enclosed by the keywords, **THEN** and **END IF**. If the condition is true, the statements get executed and if the condition is false or NULL then the IF statement does nothing.
2	IF-THEN-ELSE statement: **IF statement** adds the keyword **ELSE** followed by an alternative sequence of statement. If the condition is false or NULL, then only the alternative sequence of statements get executed. It ensures that either of the sequence of statements is executed.
3	IF-THEN-ELSIF statement: It allows you to choose between several alternatives.
4	Case statement: 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. A selector is an expression whose value is used to select one of several alternatives.
5	Searched CASE statement: The searched CASE statement **has no selector**, and it's WHEN clauses contain search conditions that yield Boolean values.
6	nested IF-THEN-ELSE: You can use one **IF-THEN** or **IF-THEN-ELSIF** statement inside another **IF-THEN** or **IF-THEN-ELSIF** statement(s).

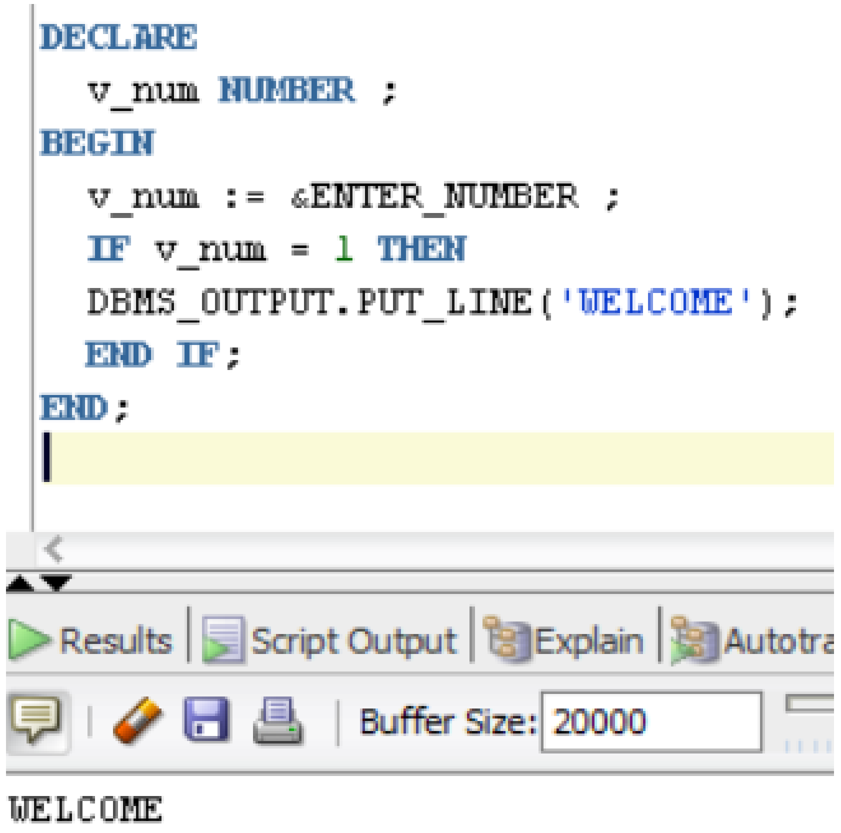
1. IF-THEN statement:

SYNTAX:

```
IF condition THEN
        block_code;
END IF ;
```

EXAMPLE:

```
DECLARE
        v_num NUMBER ;
BEGIN
        v_num :=  &Enter_number;
        IF v_num = 1 THEN
        DBMS_OUTPUT.PUT_LINE (‘WELCOME’);
        END IF;
END ;
/
```



```
DECLARE
  v_num NUMBER ;
BEGIN
  v_num := &ENTER_NUMBER ;
  IF v_num = 1 THEN
  DBMS_OUTPUT.PUT_LINE('WELCOME');
  END IF;
END;
```

```
WELCOME
```

2. IF-THEN-ELSE statement:

SYNTAX:

```
IF condition THEN
        block_code_1;
ELSE
        block_code_2;
END IF ;
```

EXAMPLE:

```
DECLARE
        v_num NUMBER ;
BEGIN
        v_num :=  &Enter_number;
        IF v_num = 1 THEN
        DBMS_OUTPUT.PUT_LINE
        (‘WELCOME’);
        ELSE
        DBMS_OUTPUT.PUT_LINE (‘BYE’);
        END IF;
END ;
/
```

3. NESTED IF-THEN-ELSE statement:

SYNTAX:

```
IF condition THEN
        block_code_1;
        IF condition THEN
                block_code_2;
        ELSE
                block_code_3;
        END IF ;
ELSE
        block_code_4;
END IF ;
```

ITERATIVE CONTROL STRUCTURES

S.No	Loop Type & Description
1	**PL/SQL Basic LOOP:** In this loop structure, sequence of statements is enclosed between the LOOP and the END LOOP statements. At each iteration, the sequence of statements is executed and then control resumes at the top of the loop.
2	**PL/SQL WHILE LOOP:** Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	**PL/SQL FOR LOOP:** Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
4	**Nested loops in PL/SQL:** You can use one or more loop inside any another basic loop, while, or for loop.

1. BASIC LOOP :

SYNTAX:

```
LOOP
        statements ;
        EXIT [ WHEN condition] ;
END LOOP ;
```

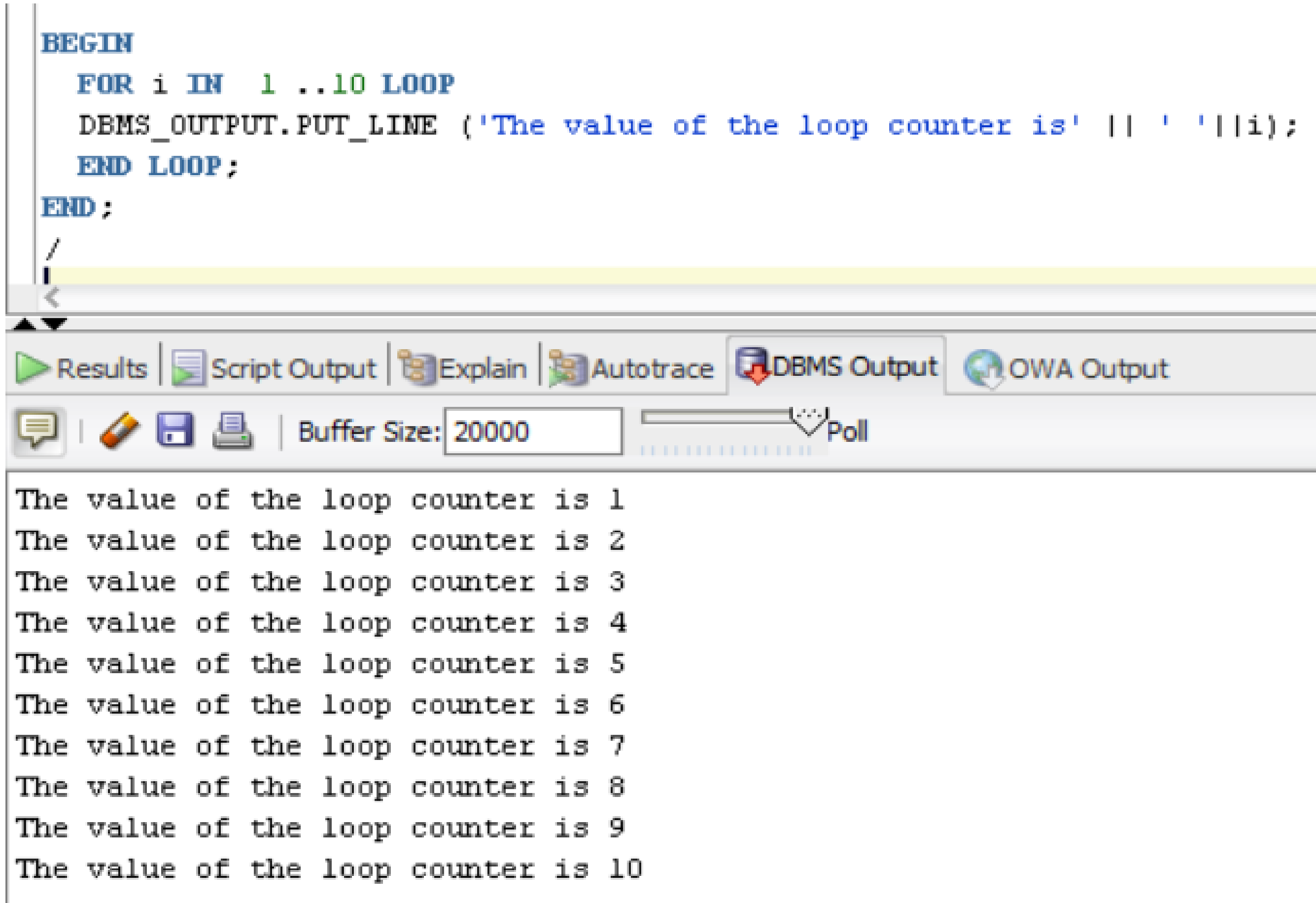
- If the EXIT WHEN condition is listed after the statements to be executed, it implies that any statements within the loop will automatically be executed at least once. This is referred to as POST-TEST.
- However, if the EXIT WHEN condition is placed before the statements to be executed it is called the PRE-TEST.
- If the increment to the counter is skipped, then the loop will become infinite.

2. FOR LOOP :

SYNTAX:

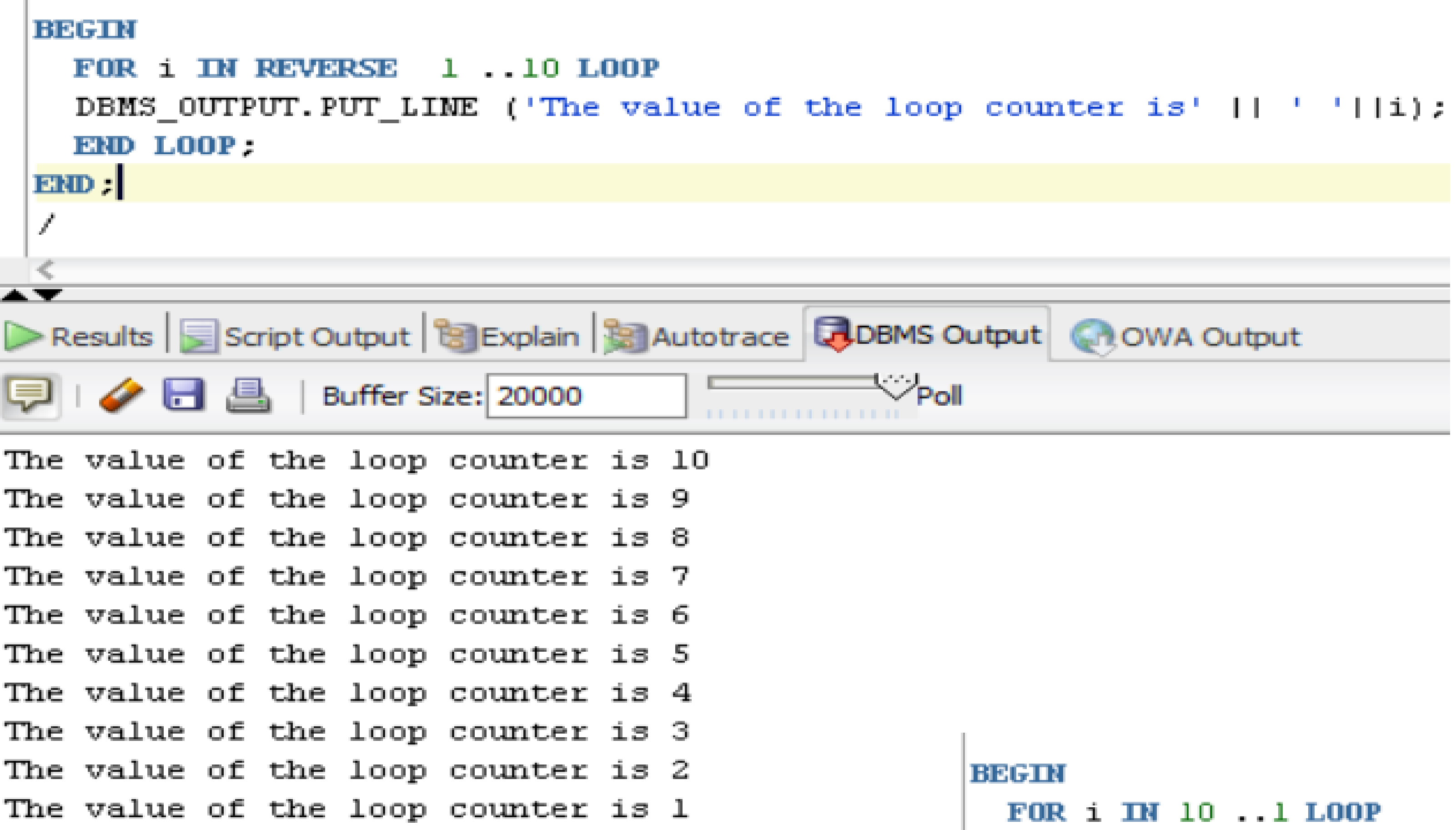
```
FOR counter IN [ REVERSE ] lowerlimit .. upperlimit LOOP
        statements ;
END LOOP ;
```

- The counter of the FOR LOOP is not a variable that must be declared in the declarative section.
- The counter is implicitly declared when the loop is executed the first time.



```
BEGIN
  FOR i IN  1 ..10 LOOP
  DBMS_OUTPUT.PUT_LINE ('The value of the loop counter is' || ' '||i);
  END LOOP;
END;
/
```

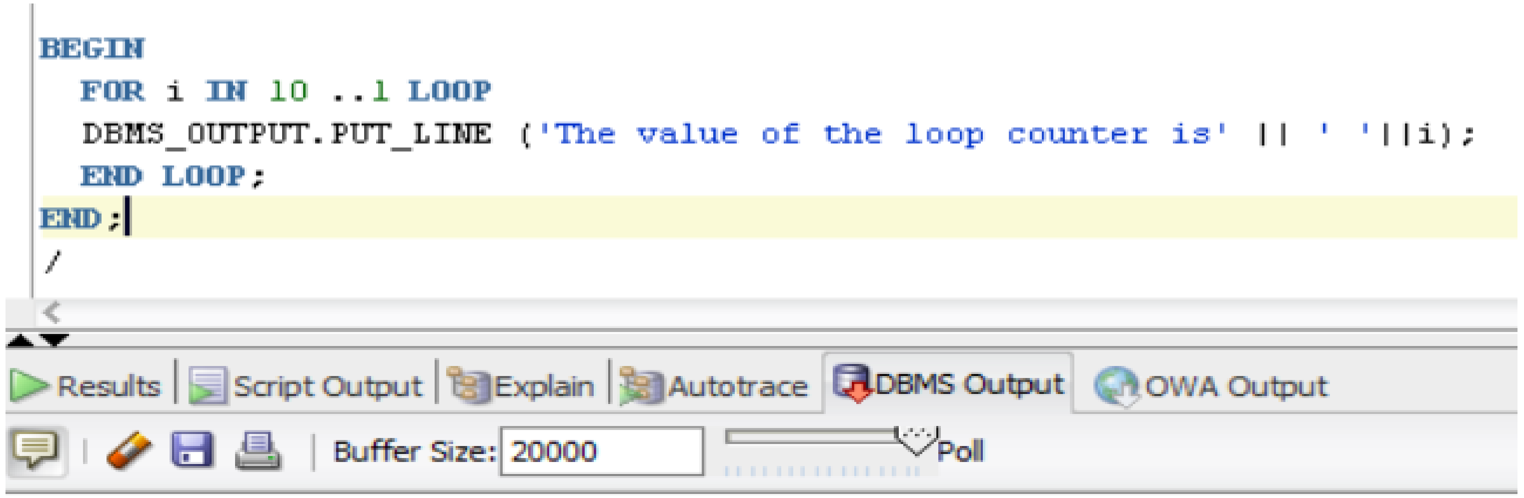
```
The value of the loop counter is 1
The value of the loop counter is 2
The value of the loop counter is 3
The value of the loop counter is 4
The value of the loop counter is 5
The value of the loop counter is 6
The value of the loop counter is 7
The value of the loop counter is 8
The value of the loop counter is 9
The value of the loop counter is 10
```



```
BEGIN
  FOR i IN REVERSE  1 ..10 LOOP
  DBMS_OUTPUT.PUT_LINE ('The value of the loop counter is' || ' '||i);
  END LOOP;
END;
/
```

```
The value of the loop counter is 10
The value of the loop counter is 9
The value of the loop counter is 8
The value of the loop counter is 7
The value of the loop counter is 6
The value of the loop counter is 5
The value of the loop counter is 4
The value of the loop counter is 3
The value of the loop counter is 2
The value of the loop counter is 1
```

- Default increment of the FOR LOOP is 1.
- Custom increments must be accommodated through code.



```
BEGIN
  FOR i IN 10 ..1 LOOP
  DBMS_OUTPUT.PUT_LINE ('The value of the loop counter is' || ' '||i);
  END LOOP;
END;
/
```

LOOP CONTROL STATEMENTS

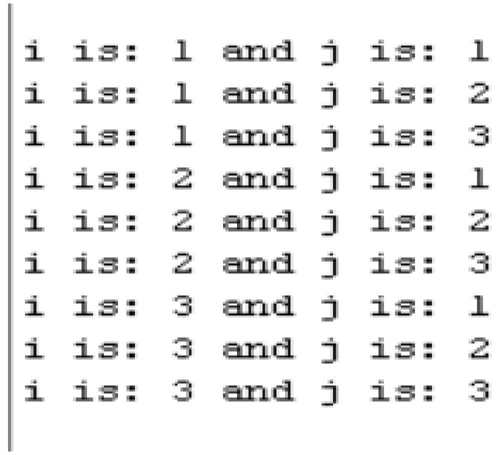
- Loop control statements change execution from its normal sequence.
- PL/SQL supports the following control statements. Labeling loops also help in taking the control outside a loop.

S.No	Control Statement & Description
1	**EXIT statement:** The Exit statement completes the loop and control passes to the statement immediately after the END LOOP.
2	**CONTINUE statement:** Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	**GOTO statement:** Transfers control to the labeled statement. Though it is not advised to use the GOTO statement in your program

LOOP LABELS

- PL/SQL loops can be labeled.
- The label should be enclosed by double angle brackets (<< and >>) and appear at the beginning of the LOOP statement.
- The label name can also appear at the end of the LOOP statement.
- You may use the label in the EXIT statement to exit from the loop.

```
DECLARE
            i number(1);
            j number(1);
BEGIN
                        << outer_loop >>
             FOR i IN 1..3 LOOP
                        << inner_loop >>
            FOR j IN 1..3 LOOP
            dbms_output.put_line('i is: '|| i || ' and j is: ' || j);
            END loop inner_loop;
            END loop outer_loop;
END ;
/
```



```
i is: 1 and j is: 1
i is: 1 and j is: 2
i is: 1 and j is: 3
i is: 2 and j is: 1
i is: 2 and j is: 2
i is: 2 and j is: 3
i is: 3 and j is: 1
i is: 3 and j is: 2
i is: 3 and j is: 3
```

TASK A

1. Write a PL/SQL code that prints EVEN number between 1 to 10.
2. Write a PL/SQL code that takes a digit between 1 to 9 as input from user and if the digit is < 5 then displays it's a small number otherwise displays that it is a large number(make use of loop labels and loop control statements).
3. Write a PL/SQL code that displays the following output:

```
1
12
123
1234
………
12345678910
```

4. Write a PL/SQL that displays the reverse of your roll number.