

ANUBHAV SINGH

IC 2K16 54

ADBMS ASSIGNMENTS

ASSIGNMENT 0

1. Hello world

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

```
Hello, World!
```

```
PL/SQL procedure successfully completed.
```

2. User Subtype

```
DECLARE
SUBTYPE name IS char(20);
SUBTYPE message IS varchar2(100);
salutation name;
greetings message;
BEGIN salutation := 'Reader ';
greetings := 'Welcome to the World of PL/SQL';
dbms_output.put_line('Hello ' || salutation || greetings);
END;
```

```
SQL> @USERSUBTYPE
Hello Reader                               Welcome to the World of PL/SQL

PL/SQL procedure successfully completed.
```

3. Initializing variables in PL/SQL

```
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;
```

[illegible]

4. Scope of variables.

```
DECLARE
num1 number := 95;
num2 number := 85;
BEGIN
dbms_output.put_line('Outer Variable num1: ' || num1);
dbms_output.put_line('Outer Variable num2: ' || num2);
DECLARE
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;
/
```

```
SQL> @varscope
Outer Variable num1: 95
Outer Variable num2: 85
Inner Variable num1: 195
Inner Variable num2: 185

PL/SQL procedure successfully completed.
```

5. Declaration of constant using measurements of circle.

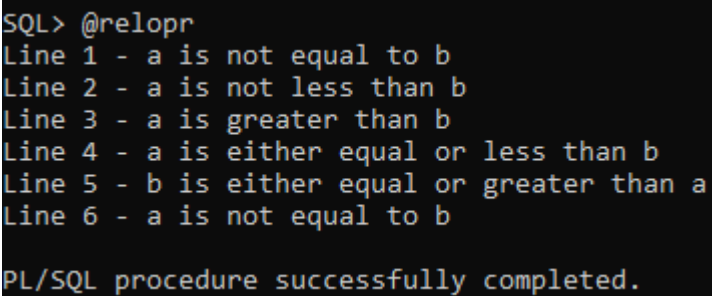
```
DECLARE
pi constant number := 3.141592654;
radius number(5,2);
dia number(5,2);
circumference number(7, 2);
area number (10, 2);
BEGIN
radius := 9.5;
dia := radius * 2;
circumference := 2.0 * pi * radius;
area := pi * radius * radius;
dbms_output.put_line('Radius: ' || radius);
dbms_output.put_line('Diameter: ' || dia);
dbms_output.put_line('Circumference: ' || circumference);
dbms_output.put_line('Area: ' || area);
END;
/
```

```
SQL> @circle
Radius: 9.5
Diameter: 19
Circumference: 59.69
Area: 283.53

PL/SQL procedure successfully completed.
```

6. Relational operator.

```
DECLARE
a number (2) := 21;
b number (2) := 10;
BEGIN
IF (a = b)
then
dbms_output.put_line('Line 1 - a is equal to b');
ELSE
dbms_output.put_line('Line 1 - a is not equal to b');
END IF;
IF (a < b)
then
dbms_output.put_line('Line 2 - a is less than b');
ELSE
dbms_output.put_line('Line 2 - a is not less than b');
END IF;
IF ( a > b )
THEN
dbms_output.put_line('Line 3 - a is greater than b');
ELSE
dbms_output.put_line('Line 3 - a is not greater than b');
END IF;
a := 5;
b := 20;
IF ( a <= b )
THEN
dbms_output.put_line('Line 4 - a is either equal or less than b');
END IF;
IF ( b >= a )
THEN
dbms_output.put_line('Line 5 - b is either equal or greater than a');
END IF;
IF ( a <> b )
THEN dbms_output.put_line('Line 6 - a is not equal to b');
ELSE
dbms_output.put_line('Line 6 - a is equal to b');
END IF;
END;
/
```

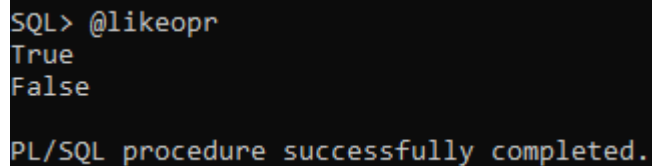


```
SQL> @relopr
Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either equal or less than b
Line 5 - b is either equal or greater than a
Line 6 - a is not equal to b

PL/SQL procedure successfully completed.
```

7. Like operator.

```
DECLARE
PROCEDURE compare (value varchar2, pattern varchar2 ) is
BEGIN
IF value LIKE pattern
THEN
dbms_output.put_line ('True');
ELSE
dbms_output.put_line ('False');
END IF;
END;
BEGIN
compare('Zara Ali', 'Z%A_i');
compare('Nuha Ali', 'Z%A_i');
END;
/
```

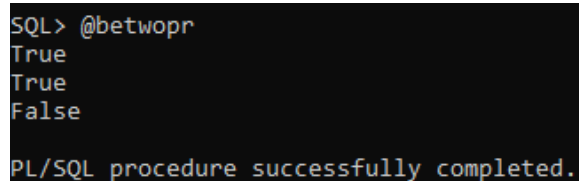


```
SQL> @likeopr
True
False

PL/SQL procedure successfully completed.
```

8. Between operator

```
DECLARE
x number(2) := 10;
BEGIN
IF (x between 5 and 20)
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
IF (x BETWEEN 5 AND 10)
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
IF (x BETWEEN 11 AND 20)
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
END;
/
```

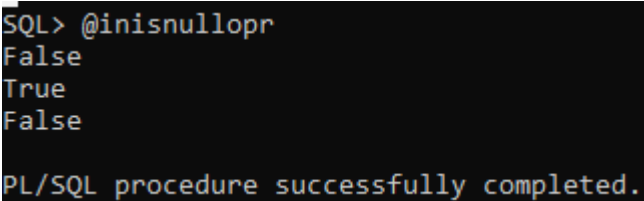


```
SQL> @betwopr
True
True
False

PL/SQL procedure successfully completed.
```

9. In and Is Null operator

```
DECLARE
letter varchar2(1) := 'm';
BEGIN
IF (letter in ('a', 'b', 'c'))
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
IF (letter in ('m', 'n', 'o'))
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
IF (letter is null)
THEN
dbms_output.put_line('True');
ELSE
dbms_output.put_line('False');
END IF;
END;
/
```

A screenshot of a SQL command window showing the execution of the PL/SQL block. The prompt is 'SQL> @inisinullopr'. The output consists of three lines: 'False', 'True', and 'False'. At the bottom, a message states 'PL/SQL procedure successfully completed.'

```
SQL> @inisinullopr
False
True
False

PL/SQL procedure successfully completed.
```

10. Logical operator.

```
DECLARE
a boolean := true;
b boolean := false;
BEGIN
IF (a AND b)
THEN
dbms_output.put_line('Line 1 - Condition is true');
END IF;
IF (a OR b)
THEN
dbms_output.put_line('Line 2 - Condition is true');
END IF;
IF (NOT a)
THEN
dbms_output.put_line('Line 3 - a is not true');
ELSE
dbms_output.put_line('Line 3 - a is true');
END IF;
IF (NOT b)
THEN
```

```

dbms_output.put_line('Line 4 - b is not true');
ELSE
dbms_output.put_line('Line 4 - b is true');
END IF;
END;
/

```

```

SQL> @logopr
Line 2 - Condition is true
Line 3 - a is true
Line 4 - b is not true

PL/SQL procedure successfully completed.

```

11. Operator precedence.

```

DECLARE
a number(2) := 20;
b number(2) := 10;
c number(2) := 15;
d number(2) := 5;
e number(2) ;
BEGIN
e := (a + b) * c / d;
dbms_output.put_line('Value of (a + b) * c / d is : ' || e);
e := ((a + b) * c) / d;
dbms_output.put_line('Value of ((a + b) * c) / d is : ' || e);
e := (a + b) * (c / d);
dbms_output.put_line('Value of (a + b) * (c / d) is : ' || e);
e := a + (b * c) / d;
dbms_output.put_line('Value of a + (b * c) / d is : ' || e);
END;
/

```

```

SQL> @oprpre
Value of (a + b) * c / d is : 90
Value of ((a + b) * c) / d is : 90
Value of (a + b) * (c / d) is : 90
Value of a + (b * c) / d is : 50

PL/SQL procedure successfully completed.

```

12. IF ELSE.

```

DECLARE
a number(2) := 10;
BEGIN
a:= 10;
IF( a < 20 )
THEN
dbms_output.put_line('a is less than 20 ');
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/

```

```
SQL> @ifelse
a is less than 20
value of a is : 10

PL/SQL procedure successfully completed.
```

13. IF THEN ELSE

```
DECLARE
a number(3) := 100;
BEGIN
IF( a < 20 )
THEN
dbms_output.put_line('a is less than 20 ');
ELSE
dbms_output.put_line('a is not less than 20 ');
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```

```
SQL> @ifthenelse
a is not less than 20
value of a is : 100

PL/SQL procedure successfully completed.
```

14. IF THEN ELSEIF

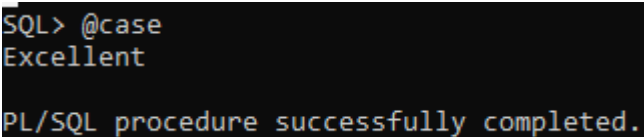
```
DECLARE
a number(3) := 100;
BEGIN
IF ( a = 10 )
THEN
dbms_output.put_line('Value of a is 10' );
ELSIF ( a = 20 )
THEN
dbms_output.put_line('Value of a is 20' );
ELSIF ( a = 30 )
THEN
dbms_output.put_line('Value of a is 30' );
ELSE
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: ' || a );
END;
/
```

```
SQL> @ifthenelseif
None of the values is matching
Exact value of a is: 100

PL/SQL procedure successfully completed.
```


15. CASE Statement.

```
DECLARE
grade char(1) := 'A';
BEGIN
CASE grade
when 'A' then
dbms_output.put_line('Excellent');
when 'B' then
dbms_output.put_line('Very good');
when 'C' then
dbms_output.put_line('Well done');
when 'D' then
dbms_output.put_line('You passed');
when 'F' then
dbms_output.put_line('Better try again');
else
dbms_output.put_line('No such grade');
END CASE;
END;
```



```
SQL> @case
Excellent

PL/SQL procedure successfully completed.
```

16. Searched CASE Statement.

```
DECLARE
grade char(1) := 'B';
BEGIN
case
when grade = 'A'
then
dbms_output.put_line('Excellent');
when grade = 'B'
then
dbms_output.put_line('Very good');
when grade = 'C'
then
dbms_output.put_line('Well done');
when grade = 'D'
then
dbms_output.put_line('You passed');
when grade = 'F' then
dbms_output.put_line('Better try again');
else
dbms_output.put_line('No such grade');
end case;
end;
```

```
SQL> @searchedcase
Very good

PL/SQL procedure successfully completed.
```

17. Nested IF THEN ELSE

```
DECLARE
a number(3) := 100;
b number(3) := 200;
BEGIN
IF( a = 100 )
THEN
IF( b = 200 )
THEN
dbms_output.put_line('Value of a is 100 and b is 200' );
END IF;
END IF;
dbms_output.put_line('Exact value of a is : ' || a );
dbms_output.put_line('Exact value of b is : ' || b );
END;
/
```

```
SQL> @nestedifthenelse
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200

PL/SQL procedure successfully completed.
```

18. LOOPS

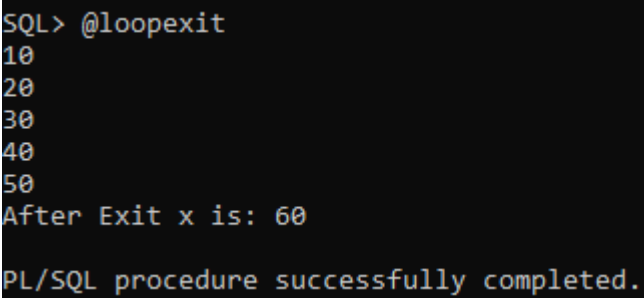
```
DECLARE
x number := 10;
BEGIN
LOOP
dbms_output.put_line(x);
x := x + 10;
IF x > 50
THEN exit;
END IF;
END LOOP;
dbms_output.put_line('After Exit x is: ' || x);
END;
/
```

```
SQL> @loops
10
20
30
40
50
After Exit x is: 60

PL/SQL procedure successfully completed.
```

19. LOOP EXIT WHEN Statement

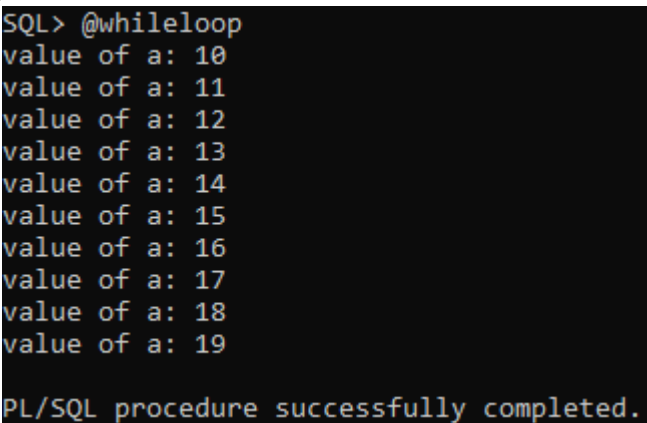
```
DECLARE
x number := 10;
BEGIN
LOOP
dbms_output.put_line(x);
x := x + 10;
exit
WHEN x > 50;
END LOOP;
dbms_output.put_line('After Exit x is: ' || x);
END;
/
```



```
SQL> @loopexit
10
20
30
40
50
After Exit x is: 60
PL/SQL procedure successfully completed.
```

20. WHILE LOOP

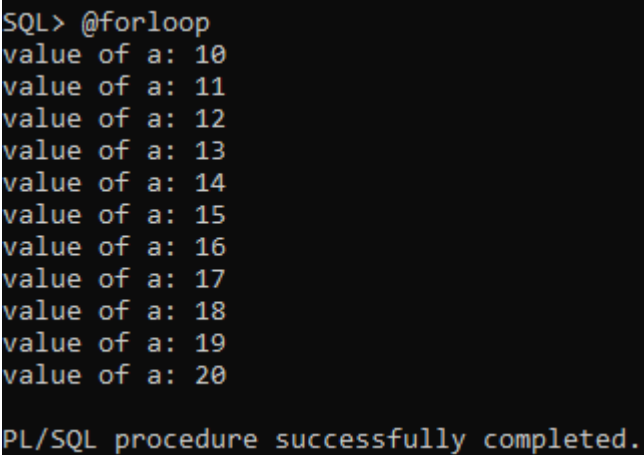
```
DECLARE
a number(2) := 10;
BEGIN
WHILE a < 20 LOOP
dbms_output.put_line('value of a: ' || a);
a := a + 1;
END LOOP;
END;
/
```



```
SQL> @whileloop
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
PL/SQL procedure successfully completed.
```

21. FOR LOOP.

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



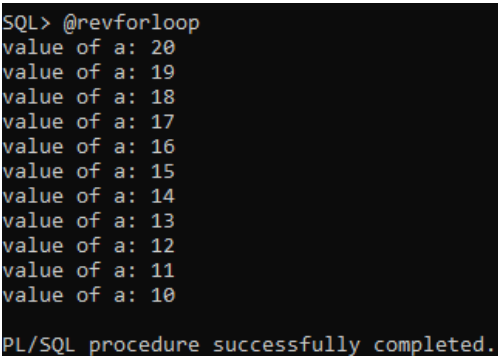
A screenshot of a SQL*Plus session showing the execution of a PL/SQL FOR loop. The prompt 'SQL>' is followed by '@forloop'. The output displays 'value of a:' followed by numbers 10 through 20 on separate lines. At the bottom, it says 'PL/SQL procedure successfully completed.'

```
SQL> @forloop
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
value of a: 20

PL/SQL procedure successfully completed.
```

22. REVERSE FOR LOOP.

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



A screenshot of a SQL*Plus session showing the execution of a PL/SQL REVERSE FOR loop. The prompt 'SQL>' is followed by '@revforloop'. The output displays 'value of a:' followed by numbers 20 down to 10 on separate lines. At the bottom, it says 'PL/SQL procedure successfully completed.'

```
SQL> @revforloop
value of a: 20
value of a: 19
value of a: 18
value of a: 17
value of a: 16
value of a: 15
value of a: 14
value of a: 13
value of a: 12
value of a: 11
value of a: 10

PL/SQL procedure successfully completed.
```

23. NESTED LOOPS

```
DECLARE
i number(3);
j number(3);
BEGIN
i := 2;
LOOP j:= 2;
LOOP exit WHEN ((mod(i, j) = 0) or (j = i));
```

```

j := j + 1;
END LOOP;
IF (j = i)
THEN
dbms_output.put_line(i || ' is prime');
END IF;
i := i + 1;
exit WHEN i = 50;
END LOOP;
END;
/

```

```

SQL> @nestedloop
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime
23 is prime
29 is prime
31 is prime
37 is prime
41 is prime
43 is prime
47 is prime

PL/SQL procedure successfully completed.

```

24. LABELING A 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;
/

```

```

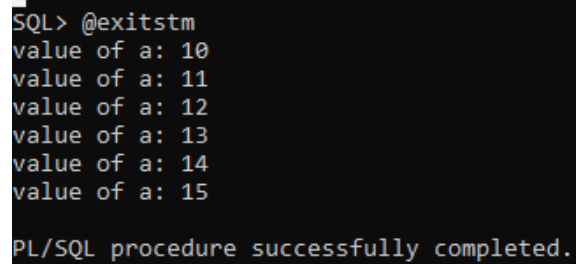
SQL> @looplabel
i is: 1 and j is: 1
i is: 1 and j is: 2
i is: 1 and j is: 3
i is: 2 and j is: 1
i is: 2 and j is: 2
i is: 2 and j is: 3
i is: 3 and j is: 1
i is: 3 and j is: 2
i is: 3 and j is: 3

PL/SQL procedure successfully completed.

```

25. Exit Statement

```
DECLARE
a number(2) := 10;
BEGIN
WHILE a < 20 LOOP
dbms_output.put_line ('value of a: ' || a);
a := a + 1;
IF a > 15 THEN
EXIT;
END IF;
END LOOP;
END;
/
```



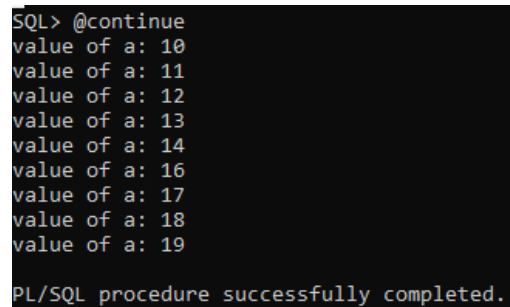
A screenshot of a SQL*Plus session showing the execution of the PL/SQL code from the previous block. The prompt is SQL> @exitstm. The output displays the values of 'a' from 10 to 15, each on a new line. The message 'PL/SQL procedure successfully completed.' appears at the bottom.

```
SQL> @exitstm
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15

PL/SQL procedure successfully completed.
```

26. CONTINUE Statement

```
DECLARE
a number(2) := 10;
BEGIN
WHILE a < 20 LOOP
dbms_output.put_line ('value of a: ' || a);
a := a + 1;
IF a = 15 THEN
a := a + 1;
CONTINUE;
END IF;
END LOOP;
END;
/
```



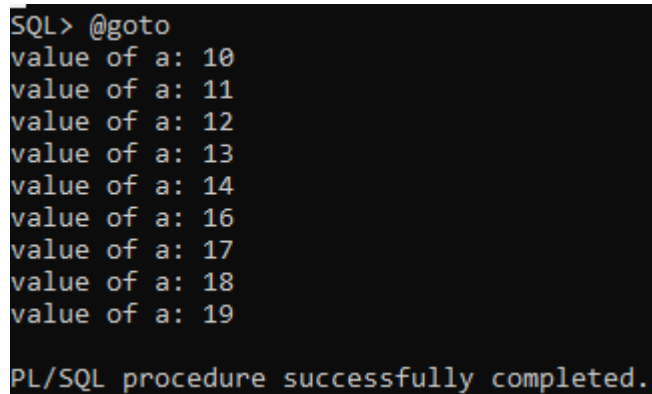
A screenshot of a SQL*Plus session showing the execution of the PL/SQL code from the previous block. The prompt is SQL> @continue. The output displays the values of 'a' from 10 to 19, each on a new line. The message 'PL/SQL procedure successfully completed.' appears at the bottom.

```
SQL> @continue
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19

PL/SQL procedure successfully completed.
```

27. GOTO Statement.

```
DECLARE
a number(2) := 10;
BEGIN
<<loopstart>>
WHILE a < 20 LOOP
dbms_output.put_line ('value of a: ' || a);
a := a + 1;
IF a = 15 THEN
a := a + 1;
GOTO loopstart;
END IF;
END LOOP;
END;
/
```

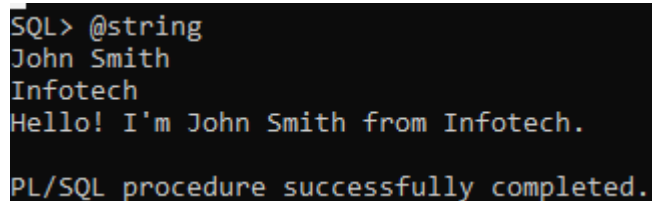


```
SQL> @goto
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19

PL/SQL procedure successfully completed.
```

28. String Variables

```
DECLARE
name varchar2(20);
company varchar2(30);
introduction clob;
choice char(1);
BEGIN
name := 'John Smith';
company := 'Infotech';
introduction := ' Hello! I'm John Smith from Infotech.';
choice := 'y';
IF choice = 'y' THEN
dbms_output.put_line(name);
dbms_output.put_line(company);
dbms_output.put_line(introduction);
END IF;
END;
/
```

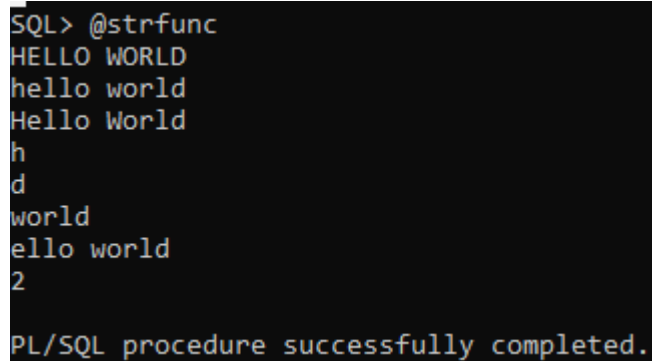


```
SQL> @string
John Smith
Infotech
Hello! I'm John Smith from Infotech.

PL/SQL procedure successfully completed.
```

29. String Functions

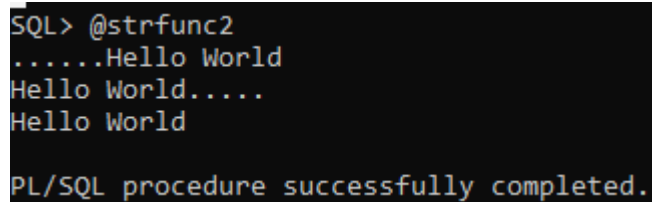
```
DECLARE
greetings varchar2(11) := 'hello world';
BEGIN
dbms_output.put_line(UPPER(greetings));
dbms_output.put_line(LOWER(greetings));
dbms_output.put_line(INITCAP(greetings));
dbms_output.put_line ( SUBSTR (greetings, 1, 1));
dbms_output.put_line ( SUBSTR (greetings, -1, 1));
dbms_output.put_line ( SUBSTR (greetings, 7, 5));
dbms_output.put_line ( SUBSTR (greetings, 2));
dbms_output.put_line ( INSTR (greetings, 'e'));
END;
/
```



```
SQL> @strfunc
HELLO WORLD
hello world
Hello World
h
d
world
ello world
2
PL/SQL procedure successfully completed.
```

30. Trim in string

```
DECLARE
greetings varchar2(30) := '.....Hello World.....';
BEGIN
dbms_output.put_line(RTRIM(greetings, '.'));
dbms_output.put_line(LTRIM(greetings, '.'));
dbms_output.put_line(TRIM( '.' from greetings));
END;
/
```



```
SQL> @strfunc2
.....Hello World
Hello World.....
Hello World
PL/SQL procedure successfully completed.
```


PL/SQL Based Assignment 0.1

1. What is PL/SQL?

PL/SQL is designed for seamless processing of SQL statements enhancing the security, portability, and robustness of the database. Similar to other database languages, it gives more control to the programmers by the use of loops, conditions and object-oriented concepts. Full form of PL/SQL is "Procedural Language extensions to SQL".

2. Differentiate between % ROWTYPE and TYPE RECORD.

The %ROWTYPE provides a record type that represents a row in a table. and TYPE RECORD provide the data type of column record type.

3. Explain uses of cursor.

The major function of a cursor is to retrieve data, one row at a time, from a result set, unlike the SQL commands which operate on all the rows in the result set at one time. Cursors are used when the user needs to update records in a singleton fashion or in a row by row manner, in a database table.

4. Show code of a cursor for loop.

```
DECLARE
CURSOR c_product
IS SELECT
product_name,
list_price
FROM products
ORDER BY list_price DESC;
BEGIN
FOR r_product
IN c_product LOOP
dbms_output.put_line( r_product.product_name || ': $' || r_product.list_price );
END LOOP;
END;
/
```

5. Explain the uses of database trigger.

A database trigger is special stored procedure that is run when specific actions occur within a database. Most triggers are defined to run when changes are made to a table's data. Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.

6. What are the two types of exceptions?

There are three types of exceptions: Predefined exceptions are error conditions that are defined by PL/SQL. Non-predefined exceptions include any standard TimesTen errors. User-defined exceptions are exceptions specific to your application.

7. Show some predefined exceptions.

Predefined exceptions are errors which occur during the execution of the program. The predefined exceptions are internally defined exceptions that PL/SQL has given names e.g., NO_DATA_FOUND , TOO_MANY_ROWS . User-defined exceptions are custom exception defined by users.

8. Explain Raise_application_error.

The raise_application_error is actually a procedure defined by Oracle that allows the developer to raise an exception and associate an error number and message with the procedure. ... Oracle provides the raise_application_error procedure to allow you to raise custom error numbers within your applications.

9. Show how functions and procedures are called in a PL/SQL block.

A procedure is created with the CREATE OR REPLACE PROCEDURE statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows –

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
{IS | AS}
BEGIN
    < procedure_body >
END procedure_name;
```

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

10. Explain two virtual tables available at the time of database trigger execution.
The table columns are referred as OLD.column_name and NEW.column_name. For triggers related to INSERT only NEW.column_name values only available. For triggers related to UPDATE only OLD.column_name NEW.column_name values only available. For triggers related to DELETE only OLD.column_name values only available.

11. What are the rules to be applied to NULLs whilst doing comparisons?

Rule #1:

Use NULLs to indicate unknown/missing information only. Do not use NULLs in place of zeroes, zero-length strings or other "known" blank values. Update your NULLs with proper information as soon as possible.

Rule #2:

In ANSI SQL, NULL is not equal to anything, even other NULLs! Comparisons with NULL always result in UNKNOWN.

Rule #3:

Use SET ANSI_NULLS ON, and always use ANSI Standard SQL Syntax for NULLs. Straying from the standard can cause problems including portability issues, incompatibility with existing code and databases and returning incorrect results.

Rule #4:

The ANSI Standard COALESCE() and CASE syntaxes are preferred over ISNULL() or other proprietary syntax.

12. How is a process of PL/SQL compiled?

When PL/SQL is loaded into the server it is compiled to byte code before execution. The process of native compilation converts PL/SQL stored procedures to native code shared libraries which are linked into the kernel resulting in performance increases for the procedural code.

13. Differentiate between Syntax and runtime errors.

A program with a syntax error cannot be executed. The program with a runtime error can be executed but dumps under certain conditions. ... Syntax errors are static error that can be detected by the compiler. Runtime errors are dynamic error that cannot be detected by the compiler.

14. Explain Commit, Rollback and Savepoint..

The following commands are used to control transactions.

COMMIT – to save the changes.

ROLLBACK – to roll back the changes.

SAVEPOINT – creates points within the groups of transactions in which to ROLLBACK.

15. Define Implicit and Explicit Cursors.

Implicit cursors are automatically created when select statements are executed. Explicit cursors need to be defined explicitly by the user by providing a name. They are capable of fetching a single row at a time. Explicit cursors can fetch multiple rows. They are more vulnerable to errors such as Data errors, etc.

16. Explain mutating table error.

A mutating table error occurs when a row-level trigger tries to examine or change a table that is already undergoing change (via an INSERT, UPDATE, or DELETE statement). In particular, this error occurs when a row-level trigger attempts to read or write the table from which the trigger was fired.

17. When is a declare statement required?

DECLARE is an optional section of a PL/SQL block. It is used for the declaration of local memory variables, localized subprograms (procedures or functions) to be used within the PL/SQL block. The scope and visibility of the variables within the DECLARE section is limited only to that anonymous block.

The DECLARE keyword is not required in PL/SQL subprograms (procedures and functions). The section contained within IS and BEGIN is known as the Declarative section. Note that TRIGGER requires the DECLARE section to be declared explicitly.

18. How many triggers can be applied to a table?

Triggers are implicitly fired by Oracle when a triggering event occurs, no matter which user is connected or which application is being used. There are 12 types of triggers that can exist in a table in Oracle: 3 before statement, 3 after statement, 3 before each row and 3 after each row.

19. What is the importance of SQLCODE and SQLERRM?

SQLCODE: It returns the error number for the last encountered error.

SQLERRM: It returns the actual error message of the last encountered error.

20. If a cursor is open, how can we find in a PL/SQL Block?

OPEN cursor_name; In this syntax, the cursor_name is the name of the cursor declared in the declaration section. When you open a cursor, Oracle parses the query, binds variables, and executes the associated SQL statement.

21. Show the two PL/SQL cursor exceptions.

Exception Name	Reason
INVALID_CURSOR	When you perform an invalid operation on a cursor like closing a cursor, fetch data from a cursor that is not opened.
NO_DATA_FOUND	When a SELECT...INTO clause does not return any row from a table.

22. What operators deal with NULL?

In SQL Server, NULL value indicates an unavailable or unassigned value. The value NULL does not equal zero (0), nor does it equal a space (' '). Because the NULL value cannot be equal or unequal to any value, you cannot perform any comparison on this value by using operators such as '=' or '<>'.

23. Does SQL*Plus also have a PL/SQL Engine?

No. Unlike Oracle Forms, SQL*Plus does not have an embedded PL/SQL engine. Thus, all your PL/SQL code is sent directly to the database engine for execution. This makes it much more efficient as SQL statements are not stripped off and sent to the database individually.

24. What packages are available to PL/SQL developers?

- DBMS_ALERT Package
- DBMS_OUTPUT Package
- DBMS_PIPE Package
- HTF and HTP Packages
- UTL_FILE Package
- UTL_HTTP Package
- UTL_SMTP Package.

25. Explain 3 basic parts of a trigger.

A trigger has three basic parts:

- A triggering event or statement
- A trigger restriction
- A trigger action

The Triggering Event or Statement

A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:

- An INSERT, UPDATE, or DELETE statement on a specific table (or view, in some cases)
- A CREATE, ALTER, or DROP statement on any schema object
- A database startup or instance shutdown
- A specific error message or any error message
- A user logon or logoff

Trigger Restriction

A trigger restriction specifies a Boolean expression that must be true for the trigger to fire. The trigger action is not run if the trigger restriction evaluates to false or unknown.

Trigger Action

A trigger action is the procedure (PL/SQL block, Java program, or C callout) that contains the SQL statements and code to be run when the following events occur:

- A triggering statement is issued.
- The trigger restriction evaluates to true.

26. What are character functions?

A character function is a function that takes one or more character values as parameters and returns either a character value or a number value. When a character function returns a character value, that value is always of type VARCHAR2 (variable length), with the following two exceptions: UPPER and LOWER.

27. Explain TTITLE and BTITLE.

The TTITLE command defines the top title; the BTITLE command defines the bottom title. You can also set a header and footer for each report.

28. Show the cursor attributes of PL/SQL.

Cursor attributes (PL/SQL) Each cursor has a set of attributes that enables an application program to test the state of the cursor. These attributes are %ISOPEN, %FOUND, %NOTFOUND, and %ROWCOUNT. This attribute is used to determine whether a cursor is in the open state.

29. What is an Intersect?

The Oracle INTERSECT operator is used to return the results of 2 or more SELECT statements. However, it only returns the rows selected by all queries or data sets. If a record exists in one query and not in the other, it will be omitted from the INTERSECT results.

30. What are sequences?

A sequence is an object in Oracle that is used to generate a number sequence. This can be useful when you need to create a unique number to act as a primary key.

31. How would you reference column values BEFORE and AFTER you have inserted and deleted triggers?

A database event is a delete, *insert*, or update operation. For example, *if you define a trigger for a delete on a particular table, the before triggers can be either row or statement triggers. For update, you can specify columns*. *Referencing old and new values: the referencing clause.*

32. What are the uses of SYSDATE and USER keywords?

The SYSDATE() function returns the current date and time.

In Oracle, a user is someone that can connect to a database, and optionally (depending on the assigned privileges) can own objects (such as [tables](#)) in the database.

33. How does ROWID help in running a query faster?

ROWID's are the fastest way to access a row of data, but if you can do an operation in a single DML statement, that is faster than selecting the data first, then supplying the ROWID to the DML statement. If rows are moved, the ROWID will change. Rows can move due to maintenance operations like shrinks and table moves.

34. What are database links used for?

A database link is a schema object in one database that enables you to access objects on another database. The other database need not be an Oracle Database system. However, to access non-Oracle systems you must use Oracle Heterogeneous Services.

35. What does fetching a cursor do?

FETCH a cursor advances the cursor to the next row.

36. What does closing a cursor do?

Closing a cursor releases the context area.

37. Explain the uses of Control File.

When an instance of an ORACLE database is started, its control file is used to identify the database and redo log files that must be opened for database operation to proceed. It is also used in database recovery.

38. Explain Consistency.

An operation (a query or a transaction) is transaction set consistent if all its reads return data written by the same set of committed transactions. An operation is not transaction set consistent if some reads reflect the changes of one set of transactions and other reads reflect changes made by other transactions

39. Differ between Anonymous blocks and sub-programs.

Anonymous block are created on client and subprograms are stored on server. Subprograms called functions must return values. 10. Anonymous blocks cannot take parameters.

40. Differ between DECODE and CASE.

CASE is a statement while DECODE is a function.

CASE can work with logical operators other than '=' : DECODE performs an equality check only.

CASE is capable of other logical comparisons such as < ,> ,BETWEEN , LIKE etc.

41. Explain autonomous transaction.

An autonomous transaction is an independent transaction that is initiated by another transaction, and executes without interfering with the parent transaction.

When an autonomous transaction is called, the originating transaction gets suspended.

42. Differentiate between SGA and PGA.

The basic difference between SGA and PGA is that PGA cannot be shared between multiple processes, in the sense, that it is used only for requirements of a particular process whereas the SGA is used for the whole instance and it is shared.

43. What is the location of Pre_defined_functions?

Pre_defined_functions package specification within the file *stdspec.sql* located in the *ORACLE_HOME/rdbms/admin* directory on the database server.

44. Explain polymorphism in PL/SQL.

Polymorphism is a feature of object-oriented programming, is the ability to create a variable, a function, or an object that has more than one form.

In Oracle procedural programming also supports polymorphism in the form of program unit overloading inside a package, member function type etc.

45. What are the uses of MERGE?

The Oracle MERGE statement selects data from one or more source tables and updates or inserts it into a target table. The MERGE statement allows you to specify a condition to determine whether to update data from or insert data into the target table.

46. Can 2 queries be executed simultaneously in a Distributed Database System?

Its depend upon the query. If the two queries are firing on the table it is not possible. If the queries are firing on different tables then it is possible.

47. Explain Raise_application_error.

The raise_application_error is actually a procedure defined by Oracle that allows the developer to raise an exception and associate an error number and message with the procedure. ... Oracle provides the raise_application_error procedure to allow you to raise custom error numbers within your applications.

48. What is out parameter used for eventhough return statement can also be used in pl/sql?

Returned values from functions that has no out parameters can be used from SQL queries (inside a select or dml statement). If you have a code that *returns* a simple primitive type, i reccomend function. ... Functions *can be out parameters* too, but that functions, like procedures, cannot be called from SQL queries.

49. How would you convert date into Julian date format?

Julian days are the number of days since January 1, 4712 BC. It is represents as a number. So every date since January 1, 4712 BC can be represented as a number, which is called Julian Date.

Eg:-

```
SQL > select to_char(to_date('24-Jan-2013','dd-mon-yyyy'),'J') as julian from dual;
```

50. Explain SPOOL.

The "spool" command is used within SQL*Plus to direct the output of any query to a server-side flat file. SQL> spool /tmp/myfile.lst. Becuse the spool command interfaces with the OS layer, the spool command is commonly used within Oracle shell scripts.

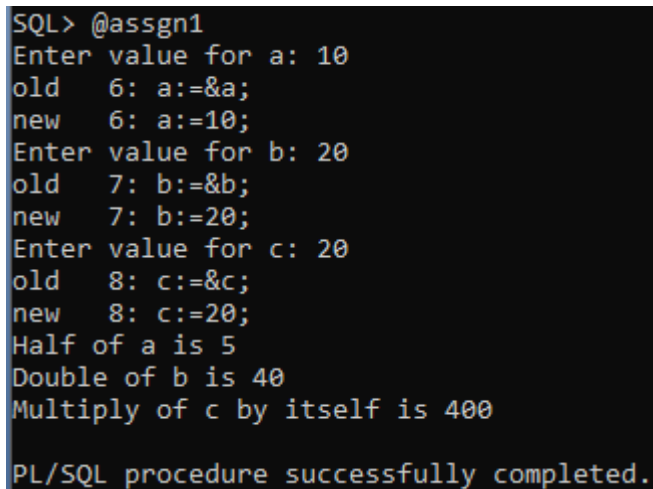
PLSQL ASSIGNMENT 1

1. Write a program that declares and assigns values to the variables a, b, and c, and then does the following:-

Halves the value of a, doubles b, multiplies c by itself. Display the output of the program on the screen using dbms_output.put_line.

```
declare
a int;
b int;
c int;
begin
a:=&a;
b:=&b;
c:=&c;
dbms_output.put_line('Half of a is '||a*0.5);
dbms_output.put_line('Double of b is '||2*b);
dbms_output.put_line('Multiply of c by itself is '||c*c);
end;
/
```

OUTPUT



```
SQL> @assgn1
Enter value for a: 10
old 6: a:=&a;
new 6: a:=10;
Enter value for b: 20
old 7: b:=&b;
new 7: b:=20;
Enter value for c: 20
old 8: c:=&c;
new 8: c:=20;
Half of a is 5
Double of b is 40
Multiply of c by itself is 400

PL/SQL procedure successfully completed.
```

2. Write a program that computes the perimeter and the area of a rectangle. Define your own values for the length and width. (Assuming that L and W are the length and width of the rectangle, Perimeter = 2*(L+W) and Area = L*W. Display the output on the screen using dbms_output.put_line.

```
declare
l int;
w int;
begin
l:=&l;
w:=&w;
```

```

dbms_output.put_line('Perimeter of Rectangle is ' || 2*(l+w));
dbms_output.put_line('Area of Rectangle is ' || l*w);
end;
/

```

OUTPUT

```

SQL> @assgn2
Enter value for l: 10
old 5: l:=&l;
new 5: l:=10;
Enter value for w: 20
old 6: w:=&w;
new 6: w:=20;
Perimeter of Rectangle is 60
Area of Rectangle is 200

PL/SQL procedure successfully completed.

```

- Suppose you had to write a block to compute the volume of a cube. The values you would need are the three dimensions of the cube. Think up four appropriate variable names to be used in the program – three variables to hold the three dimensions, and one for the result. (Assuming that L, W and H are the dimensions of a cube, $\text{Volume} = L * W * H$). Display the output on the screen using dbms_output.put_line. L, W and H are to be input by the user.

```

declare
l real;
w real;
b real;
volume real;
begin
l:=&l;
w:=&w;
b:=&b;
volume:=l*b*w;
dbms_output.put_line('volume of cube is ' || volume);
end;
/

```

OUTPUT

```

SQL> @assgn3
Enter value for l: 10
old 7: l:=&l;
new 7: l:=10;
Enter value for w: 20
old 8: w:=&w;
new 8: w:=20;
Enter value for b: 30
old 9: b:=&b;
new 9: b:=30;
volume of cube is 6000

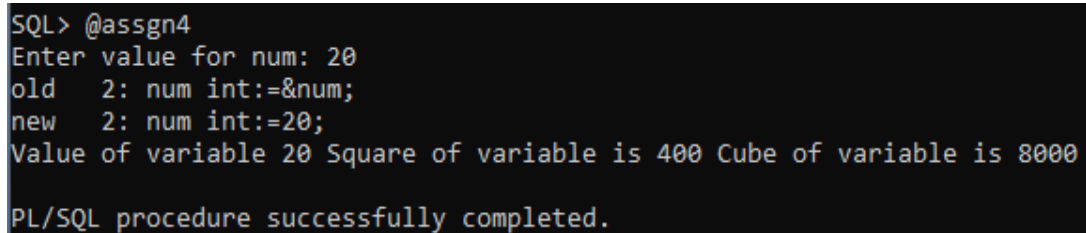
PL/SQL procedure successfully completed.

```

4. Write a program that declares an integer variable called num, assigns a value to it, and computes and inserts into the temp table the value of the variable itself, its square and its cube

```
declare
num int:=&num;
temp int;
begin
temp:=num;
dbms_output.put_line('Value of variable '||temp||' Square of variable is
'||temp*temp||' Cube of variable is '||temp*temp*temp);
end;
/
```

OUTPUT



```
SQL> @assgn4
Enter value for num: 20
old 2: num int:=&num;
new 2: num int:=20;
Value of variable 20 Square of variable is 400 Cube of variable is 8000
PL/SQL procedure successfully completed.
```

5. Convert a temperature in Fahrenheit (F) to its equivalent in Celsius (C) and vice versa.

The required formulae are:-

$$C = (F - 32) * 5 / 9$$

$$F = 9 / 5 * C + 32$$

Display the output on the screen using dbms_output.put_line. Data has to be input by the user.

```
declare
f number:=&f;
c number:=&c;
total number;
begin
total:=(f-32)*5/9;
dbms_output.put_line('Fahrenheit entered '||f||' equals to celsius '||total);
total:=9/5*c+32;
dbms_output.put_line('Celsius entered '||f||' equals to Fahrenheit '||total);
end;
/
```

OUTPUT

```
SQL> @assgn5
Enter value for f: 98.6
old 2: f number:=&f;
new 2: f number:=98.6;
Enter value for c: 37
old 3: c number:=&c;
new 3: c number:=37;
Fahrenheit entered 98.6 equals to celsius 37
Celsius entered 98.6 equals to Fahrenheit 98.6

PL/SQL procedure successfully completed.
```

6. Convert a given number of days to a measure of time given in years, weeks, and days. For example, 375 days equals 1 year, 1 week and 3 days. (Ignore leap year Display the output on the screen using dbms_output.put_line. Data has to be input by the user.

```
declare
num int:=&num;
year int;
week int;
day int;
temp int;
begin
temp:=MOD(num,365);
year:=num/365;
week:=temp/7;
day:=MOD(temp,7);
dbms_output.put_line(num || ' is equivalent to ' || year || ' year ' || week || ' week ' || day || '
days');
end;
/
```

OUTPUT

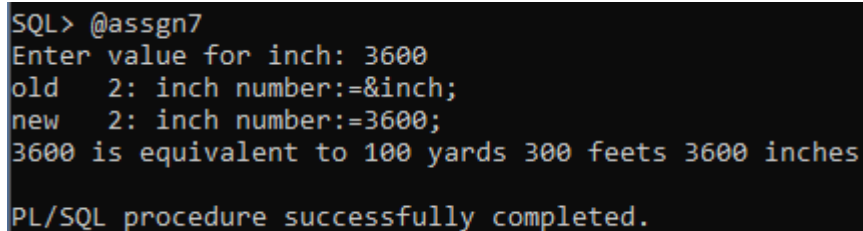
```
SQL> @assgn6
Enter value for num: 1000
old 2: num int:=&num;
new 2: num int:=1000;
1000 is equivalent to 3 year 39 week 4 days

PL/SQL procedure successfully completed.
```

7. Convert a number of inches into yards, feet, and inches. For example, 124 inches equals 3 yards, 1 foot, and 4 inches. Display the output on the screen using `dbms_output.put_line`. Data has to be input by the user.

```
declare
inch number:=&inch;
yard number;
feet number;
begin
yard:=inch/36;
feet:=inch/12;
dbms_output.put_line(inch||' is equivalent to '||yard||' yards '||feet||' feet '||inch||'
inches');
end;
/
```

OUTPUT



```
SQL> @assgn7
Enter value for inch: 3600
old 2: inch number:=&inch;
new 2: inch number:=3600;
3600 is equivalent to 100 yards 300 feet 3600 inches

PL/SQL procedure successfully completed.
```

8. Add up five amounts of money (Rs. and paise) represented as float numbers, and print the result as a truncated integer value. Display the output on the screen using `dbms_output.put_line`. Data has to be input by the user.

```
declare
n1 number;
n2 number;
n3 number;
n4 number;
n5 number;
total number;
begin
n1:=&n1;
n2:=&n2;
n3:=&n3;
n4:=&n4;
n5:=&n5;
total:=n1+n2+n3+n4+n5;
dbms_output.put_line('Entered amount is in Rs. and paise');
dbms_output.put_line('Total amount is '||total||' rs. ');
dbms_output.put_line('Truncated amount is '||TRUNC(total,0)||' rs. ');
end;
/
```

OUTPUT

```
SQL> @assgn8
Enter value for n1: 10
old 9: n1:=&n1;
new 9: n1:=10;
Enter value for n2: 20
old 10: n2:=&n2;
new 10: n2:=20;
Enter value for n3: 30
old 11: n3:=&n3;
new 11: n3:=30;
Enter value for n4: 40
old 12: n4:=&n4;
new 12: n4:=40;
Enter value for n5: 50
old 13: n5:=&n5;
new 13: n5:=50;
Entered amount is in Rs. and paise
Total amount is 150 rs.
Truncated amount is 150 rs.

PL/SQL procedure successfully completed.
```

9. Write a program that enables a user to input an integer. The program should then state whether the integer is evenly divisible by 5. (Use decode instead of IF statement where required). Display the output on the screen using dbms_output.put_line. Data has to be input by the user.

```
declare
num number:=&num;
total varchar2(50);
begin
select decode(MOD (num,5),0,'It is divisible','It is not divisible')
into total
from dual;
dbms_output.put_line(total);
end;
/
```

OUTPUT

```
SQL> @assgn9
Enter value for num: 20
old 2: num number:=&num;
new 2: num number:=20;
It is divisible

PL/SQL procedure successfully completed.
```

10. Your block should read in two real numbers and tell whether the product of the two numbers is equal to or greater than 100. Display the output on the screen using `dbms_output.put_line`. (Use `decode` instead of IF statement where required). Data has to be input by the user.

```
declare
firstno number:=&firstno;
secondno number:=&secondno;
total varchar2(50);
begin
select decode(trunc(firstno*secondno/100),0,'less then 100','greater then or equal to 100')
into total from dual;
dbms_output.put_line(total);
end;
/
```

OUTPUT

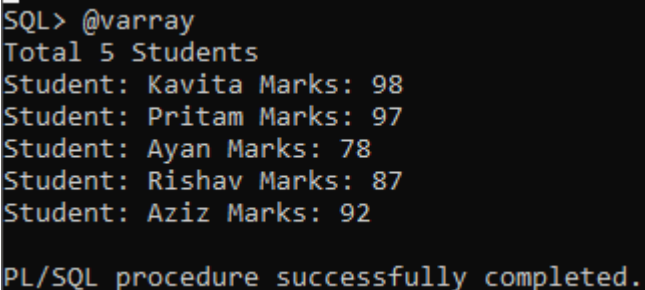
```
SQL> @assgn10
Enter value for firstno: 50
old 2: firstno number:=&firstno;
new 2: firstno number:=50;
Enter value for secondno: 60
old 3: secondno number:=&secondno;
new 3: secondno number:=60;
greater then or equal to 100

PL/SQL procedure successfully completed.
```


PLSQL Assignment 2

1. VARRAY

```
DECLARE
type namesarray
IS VARRAY(5) OF VARCHAR2(10);
type grades
IS VARRAY(5) OF INTEGER;
names namesarray;
marks grades;
total integer;
BEGIN
names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
marks:= grades(98, 97, 78, 87, 92);
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;
/
```

A screenshot of a SQL command window showing the execution of a PL/SQL procedure. The output displays the total number of students (5) and lists each student's name and marks. The procedure completed successfully.

```
SQL> @varray
Total 5 Students
Student: Kavita Marks: 98
Student: Pritam Marks: 97
Student: Ayan Marks: 78
Student: Rishav Marks: 87
Student: Aziz Marks: 92

PL/SQL procedure successfully completed.
```

2. Create function

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN
  number IS total number(2) := 0;
BEGIN
SELECT count(*) into total FROM customers;
RETURN total;
END;
/
```

```
SQL> @createfun  
Function created.
```

3. Call function

```
DECLARE  
c number(2);  
BEGIN  
c := totalCustomers();  
dbms_output.put_line('Total no. of Customers: ' || c);  
END;  
/
```

```
SQL> @callfunc1  
Total no. of Customers: 6  
  
PL/SQL procedure successfully completed.
```

4. PLSQL FUNCTION

```
DECLARE  
a number;  
b number;  
c number;  
FUNCTION findMax(x IN number, y IN number)  
RETURN number IS z number;  
BEGIN  
IF x > y  
THEN z:= x;  
ELSE  
Z:= y;  
END IF;  
RETURN z;  
END;  
BEGIN  
a:= 23;  
b:= 45;  
c := findMax(a, b);  
dbms_output.put_line(' Maximum of (23,45): ' || c);  
END;  
/
```

```
SQL> @callfun
Maximum of (23,45): 45

PL/SQL procedure successfully completed.
```

5. FACTORIAL

```
DECLARE
num number;
factorial number;
FUNCTION fact(x number)
RETURN number IS
f number;
BEGIN
IF x=0
THEN
f := 1;
ELSE
f := x * fact(x-1);
END IF;
RETURN f;
END;
BEGIN num:= 6;
factorial := fact(num);
dbms_output.put_line(' Factorial ' || num || ' is ' || factorial);
END;
/
```

```
SQL> @factorial
Factorial 6 is 720

PL/SQL procedure successfully completed.
```

6. Implicit cursor

```
DECLARE
total_rows number(2);
BEGIN
UPDATE customers SET salary = salary + 500;
IF sql%notfound
THEN
dbms_output.put_line('no customers selected');
ELSIF sql%found
THEN
total_rows := sql%rowcount;
dbms_output.put_line( total_rows || ' customers selected ');
```

```
END IF;  
END;  
/
```

```
SQL> @implicitcursor  
6 customers selected  
  
PL/SQL procedure successfully completed.
```

7. Explicit cursor

```
DECLARE  
c_id customers.id%type;  
c_name customers.name%type;  
c_addr customers.address%type;  
CURSOR  
c_customers is SELECT id, name,  
address FROM customers;  
BEGIN  
OPEN c_customers;  
LOOP  
FETCH c_customers into c_id, c_name, c_addr;  
dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);  
EXIT  
WHEN c_customers%notfound;  
END LOOP;  
CLOSE c_customers;  
END;  
/
```

```
SQL> @explicitcursor  
1 Ramesh Ahmedabad  
2 Khilan Delhi  
3 kaushik Kota  
4 Chaitali Mumbai  
5 Hardik Bhopal  
6 Komal MP  
6 Komal MP  
  
PL/SQL procedure successfully completed.
```

8. Table based records

```
DECLARE  
customer_rec customers%rowtype;  
BEGIN  
SELECT * into customer_rec FROM customers WHERE id = 5;  
dbms_output.put_line('Customer ID: ' || customer_rec.id);  
dbms_output.put_line('Customer Name: ' || customer_rec.name);  
dbms_output.put_line('Customer Address: ' || customer_rec.address);  
dbms_output.put_line('Customer Salary: ' || customer_rec.salary);
```

```
end;  
/
```

```
SQL> @tablerec1  
Customer ID: 5  
Customer Name: Hardik  
Customer Address: Bhopal  
Customer Salary: 9000  
  
PL/SQL procedure successfully completed.
```

9. Cursor based records

```
DECLARE  
CURSOR customer_cur is SELECT id,  
name,  
address FROM customers;  
customer_rec customer_cur%rowtype;  
BEGIN  
OPEN customer_cur;  
LOOP FETCH customer_cur into customer_rec;  
EXIT  
WHEN customer_cur%notfound;  
DBMS_OUTPUT.put_line(customer_rec.id || ' ' || customer_rec.name);  
END LOOP;  
END;  
/
```

```
SQL> @cursorrec  
1 Ramesh  
2 Khilan  
3 kaushik  
4 Chaitali  
5 Hardik  
6 Komal  
  
PL/SQL procedure successfully completed.
```

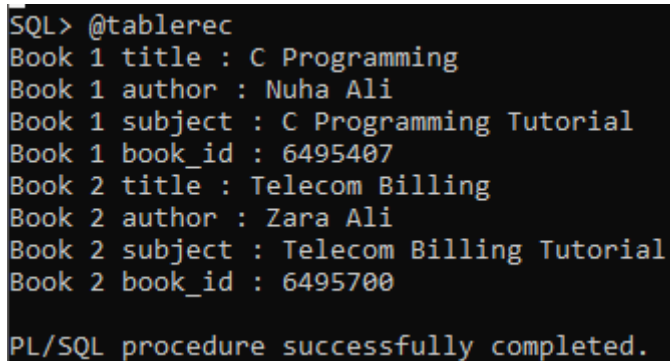
10. DEFINING A RECORD.

```
DECLARE  
type books is record  
(title varchar(50),  
author varchar(50),  
subject varchar(100),  
book_id number);  
book1 books;  
book2 books;  
BEGIN
```

```

book1.title := 'C Programming';
book1.author := 'Nuha Ali ';
book1.subject := 'C Programming Tutorial';
book1.book_id := 6495407;
book2.title := 'Telecom Billing';
book2.author := 'Zara Ali';
book2.subject := 'Telecom Billing Tutorial';
book2.book_id := 6495700;
dbms_output.put_line('Book 1 title : ' || book1.title);
dbms_output.put_line('Book 1 author : ' || book1.author);
dbms_output.put_line('Book 1 subject : ' || book1.subject);
dbms_output.put_line('Book 1 book_id : ' || book1.book_id);
dbms_output.put_line('Book 2 title : ' || book2.title);
dbms_output.put_line('Book 2 author : ' || book2.author);
dbms_output.put_line('Book 2 subject : ' || book2.subject);
dbms_output.put_line('Book 2 book_id : ' || book2.book_id);
END;
/

```



```

SQL> @tablerec
Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700

PL/SQL procedure successfully completed.

```

11. RECORD AS SUBPROGRAM PARAMETERS

```

DECLARE
type books is record
(title varchar(50),
author varchar(50),
subject varchar(100),
book_id number);
book1 books;
book2 books;
PROCEDURE printbook (book books) IS
BEGIN
dbms_output.put_line ('Book title : ' || book.title);
dbms_output.put_line('Book author : ' || book.author);
dbms_output.put_line('Book subject : ' || book.subject);
dbms_output.put_line('Book book_id : ' || book.book_id);
END;
BEGIN

```

```

book1.title := 'C Programming';
book1.author := 'Nuha Ali ';
book1.subject := 'C Programming Tutorial';
book1.book_id := 6495407;
book2.title := 'Telecom Billing';
book2.author := 'Zara Ali';
book2.subject := 'Telecom Billing Tutorial';
book2.book_id := 6495700;
printbook(book1);
printbook(book2);
END;
/

```

```

SQL> @subprog
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700

PL/SQL procedure successfully completed.

```

12. Exception handling

```

DECLARE
c_id customers.id%type := 8;
c_name customers.name%type;
c_addr customers.address%type;
BEGIN
SELECT name, address INTO c_name, c_addr FROM customers WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION WHEN no_data_found
THEN
dbms_output.put_line('No such customer!');
WHEN others
THEN
dbms_output.put_line('Error!');
END;
/

```

```

SQL> @exception
No such customer!

PL/SQL procedure successfully completed.

```

13. User defined Exception

```
DECLARE
c_id customers.id%type := &cc_id;
c_name customers.name%type;
c_addr customers.address%type;
ex_invalid_id EXCEPTION;
BEGIN
IF c_id <= 0 THEN RAISE ex_invalid_id;
ELSE
SELECT name, address INTO c_name, c_addr FROM customers WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: ' || c_name);
DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
END IF;
EXCEPTION
WHEN ex_invalid_id
THEN dbms_output.put_line('ID must be greater than zero!');
WHEN no_data_found
THEN dbms_output.put_line('No such customer!');
WHEN others
THEN dbms_output.put_line('Error!');
END;
/
```

```
SQL> @userexcep
Enter value for cc_id: -2
old 2: c_id customers.id%type := &cc_id;
new 2: c_id customers.id%type := -2;
ID must be greater than zero!

PL/SQL procedure successfully completed.
```

14. Creating Triggers

```
CREATE OR REPLACE TRIGGER
display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON
customers
FOR EACH ROW WHEN (NEW.ID > 0) DECLARE
sal_diff number;
BEGIN
sal_diff := :NEW.salary - :OLD.salary;
dbms_output.put_line('Old salary: ' || :OLD.salary);
dbms_output.put_line('New salary: ' || :NEW.salary);
dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

```
SQL> @trigger

Trigger created.
```


15. Triggering a Trigger

```
SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (7, 'Kriti', 22, 'HP', 7500.00 );  
Old salary:  
New salary: 7500  
Salary difference:
```

```
SQL> UPDATE customers SET salary = salary + 500 WHERE id = 2;  
Old salary: 2000  
New salary: 2500  
Salary difference: 500  
  
1 row updated.
```

PLSQL Assignment 3

1. PACKAGE

```
CREATE PACKAGE
cust_sal AS PROCEDURE
find_sal(c_id customers.id%type);
END cust_sal;
/
```

```
SQL> @package;

Package created.
```

2. Package Body

```
CREATE OR REPLACE PACKAGE BODY
cust_sal AS PROCEDURE
find_sal(c_id customers.id%TYPE) IS c_sal customers.salary%TYPE;
BEGIN
SELECT salary INTO c_sal FROM customers WHERE id = c_id;
dbms_output.put_line('Salary: ' || c_sal);
END find_sal;
END cust_sal;
/
```

```
SQL> @packagebody;

Package body created.
```

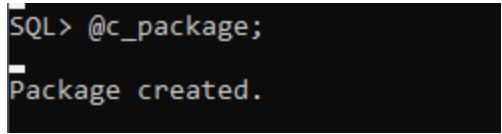
3. Using Package Elements

```
SQL> declare
  2  code customers.id%type := &cc_id;
  3  BEGIN
  4  cust_sal.find_sal(code);
  5  END;
  6  /
Enter value for cc_id: 1
old  2: code customers.id%type := &cc_id;
new  2: code customers.id%type := 1;
Salary: 2500

PL/SQL procedure successfully completed.
```

4. Creating package

```
CREATE OR REPLACE PACKAGE c_package AS
PROCEDURE addCustomer(c_id customers.id%type,
c_name customers.name%type,
c_age customers.age%type,
c_addr customers.address%type,
c_sal customers.salary%type);
PROCEDURE delCustomer(c_id customers.id%TYPE);
PROCEDURE listCustomer;
END c_package;
/
```

A screenshot of a terminal window with a black background. It shows a SQL prompt 'SQL>' followed by the command '@c_package;' and the output 'Package created.' on the next line.

```
SQL> @c_package;
Package created.
```

5. Creating Package body;

```
CREATE OR REPLACE PACKAGE BODY c_package AS
PROCEDURE addCustomer(c_id customers.id%type,
c_name customers.name%type,
c_age customers.age%type,
c_addr customers.address%type,
c_sal customers.salary%type)
IS BEGIN
INSERT INTO customers (id,name,age,address,salary)
VALUES
(c_id, c_name, c_age, c_addr, c_sal);
END addCustomer;
PROCEDURE delCustomer(c_id customers.id%type)
IS BEGIN
DELETE FROM customers WHERE id = c_id;
END delCustomer;
PROCEDURE
listCustomer IS CURSOR c_customers is SELECT name FROM customers;
TYPE c_list is TABLE OF customers.name%type;
name_list c_list := c_list();
counter integer :=0;
BEGIN
FOR n IN c_customers LOOP counter := counter +1;
name_list.extend;
name_list(counter) := n.name;
dbms_output.put_line('Customer(' || counter || ') ' || name_list(counter));
```

```

END LOOP;
END listCustomer;
END c_package;
/

```

```

SQL> @c_packagebody;
Package body created.

```

6. Using Package

```

DECLARE
code customers.id%type:= 8;
BEGIN
c_package.addcustomer(7, 'Rajnish', 25, 'Chennai', 3500);
c_package.addcustomer(8, 'Subham', 32, 'Delhi', 7500);
c_package.listcustomer;
c_package.delcustomer(code);
c_package.listcustomer;
END;
/

```

```

SQL> @c_packageuse;
Old salary:
New salary: 3500
Salary difference:
Old salary:
New salary: 7500
Salary difference:
Customer(1)Ramesh
Customer(2)Khilan
Customer(3)kaushik
Customer(4)Chaitali
Customer(5)Hardik
Customer(6)Komal
Customer(7)Rajnish
Customer(8)Subham
Customer(1)Ramesh
Customer(2)Khilan
Customer(3)kaushik
Customer(4)Chaitali
Customer(5)Hardik
Customer(6)Komal
Customer(7)Rajnish

PL/SQL procedure successfully completed.

```

Collections

7. Index By table

```
DECLARE TYPE salary IS TABLE OF NUMBER INDEX BY VARCHAR2(20);
salary_list salary;
name VARCHAR2(20);
BEGIN
salary_list('Rajnish') := 62000;
salary_list('Minakshi') := 75000;
salary_list('Martin') := 100000;
salary_list('James') := 78000;
name := salary_list.FIRST;
WHILE name IS NOT null LOOP
dbms_output.put_line ('Salary of ' || name || ' is ' || TO_CHAR(salary_list(name)));
name := salary_list.NEXT(name);
END LOOP;
END;
```

```
/
SQL> @indexbytable;
Salary of James is 78000
Salary of Martin is 100000
Salary of Minakshi is 75000
Salary of Rajnish is 62000

PL/SQL procedure successfully completed.
```

8. Elements of Index By Table

```
DECLARE CURSOR
c_customers is select name from customers;
TYPE c_list IS TABLE of customers.name%type INDEX BY binary_integer;
name_list c_list;
counter integer :=0;
BEGIN
FOR n IN c_customers LOOP counter := counter +1;
name_list(counter) := n.name;
dbms_output.put_line('Customer(' || counter || '):' || name_list(counter));
END LOOP;
END;
```

```
/
SQL> @eleindexbytable;
Customer(1):Ramesh
Customer(2):Khilan
Customer(3):kaushik
Customer(4):Chaitali
Customer(5):Hardik
Customer(6):Komal

PL/SQL procedure successfully completed.
```

9. Nested Tables

```
DECLARE TYPE names_table IS TABLE OF VARCHAR2(10);
TYPE grades IS TABLE OF INTEGER;
names names_table;
marks grades;
total integer;
BEGIN
names := names_table('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz');
marks:= grades(98, 97, 78, 87, 92);
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;
/
```

```
SQL> @nestedtable;
Total 5 Students
Student:Kavita, Marks:98
Student:Pritam, Marks:97
Student:Ayan, Marks:78
Student:Rishav, Marks:87
Student:Aziz, Marks:92

PL/SQL procedure successfully completed.
```

10.Elements of Nested Tables

```
DECLARE
CURSOR c_customers is
SELECT name FROM customers;
TYPE c_list IS TABLE of customers.name%type;
name_list c_list := c_list();
counter integer :=0;
BEGIN
FOR n IN c_customers
LOOP
counter := counter +1;
name_list.extend;
name_list(counter) := n.name;
dbms_output.put_line('Customer(' || counter || '):' || name_list(counter));
END LOOP;
END;
/
```

```
SQL> @elenested;
Customer(1):Ramesh
Customer(2):Khilan
Customer(3):kaushik
Customer(4):Chaitali
Customer(5):Hardik
Customer(6):Komal

PL/SQL procedure successfully completed.
```

Commit

11.Commit a Table

```
SQL> CREATE TABLE CUSTOMERS( ID INT NOT NULL, NAME VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25), SALARY DECIMAL (18, 2), PRIMARY KEY (ID) );
Table created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (2, 'Khilan', 25, 'Delhi', 1500.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (3, 'kaushik', 23, 'Kota', 2000.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (5, 'Hardik', 27, 'Bhopal', 8500.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (6, 'Komal', 22, 'MP', 4500.00 );
1 row created.

SQL> COMMIT;
```

12.Rollback and Savepoints

```
SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (7, 'Rajnish', 27, 'HP', 9500.00 );
1 row created.

SQL> INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS,SALARY) VALUES (8, 'Riddhi', 21, 'WB', 4500.00 );
1 row created.

SQL> SAVEPOINT sav1;
Savepoint created.

SQL> UPDATE CUSTOMERS SET SALARY = SALARY + 1000;
8 rows updated.

SQL> ROLLBACK TO sav1;
Rollback complete.

SQL> UPDATE CUSTOMERS SET SALARY = SALARY + 1000 WHERE ID = 7;
1 row updated.

SQL> UPDATE CUSTOMERS SET SALARY = SALARY + 1000 WHERE ID = 8;
1 row updated.

SQL> COMMIT;
Commit complete.
```

PLSQL Assignment 4

Date and Time Function

1.

```
SQL> SELECT SYSDATE FROM DUAL;  
  
SYSDATE  
-----  
11-MAY-20
```

2.

```
SQL> SELECT TO_CHAR(CURRENT_DATE, 'DD-MM-YYYY HH:MI:SS') FROM DUAL;  
  
TO_CHAR(CURRENT_DATE,  
-----  
11-05-2020 10:42:50
```

3.

```
SQL> SELECT ADD_MONTHS(SYSDATE, 5) FROM DUAL;  
  
ADD_MONTHS  
-----  
11-OCT-20
```

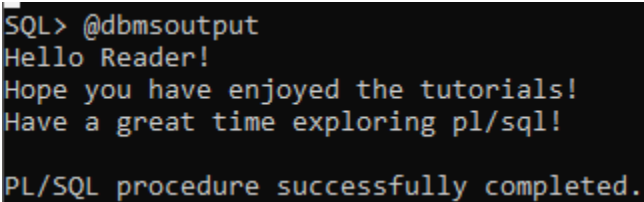
4.

```
SQL> SELECT LOCALTIMESTAMP FROM DUAL;  
  
LOCALTIMESTAMP  
-----  
11-MAY-20 10.44.30.845000 AM
```


DBMS OUTPUT

1. Using DBMS_OUTPUT function

```
DECLARE
lines dbms_output.chararr;
num_lines number;
BEGIN
dbms_output.enable;
dbms_output.put_line('Hello Reader!');
dbms_output.put_line('Hope you have enjoyed the tutorials!');
dbms_output.put_line('Have a great time exploring pl/sql!');
num_lines := 3;
dbms_output.get_lines(lines, num_lines);
FOR i IN 1..num_lines
LOOP
dbms_output.put_line(lines(i));
END LOOP;
END;
/
```

A screenshot of a SQL command window with a black background and white text. It shows the execution of a PL/SQL block that uses DBMS_OUTPUT. The output consists of three lines of text: 'Hello Reader!', 'Hope you have enjoyed the tutorials!', and 'Have a great time exploring pl/sql!'. Below these, it says 'PL/SQL procedure successfully completed.'.

```
SQL> @dbmsoutput
Hello Reader!
Hope you have enjoyed the tutorials!
Have a great time exploring pl/sql!

PL/SQL procedure successfully completed.
```

OBJECT ORIENTED

1. Type Creation

```
SQL> set serveroutput on;
SQL> CREATE OR REPLACE TYPE address AS OBJECT
  2  (house_no varchar2(10), street varchar2(30), city varchar2(20), state varchar2(10), pincode varchar2(10) );
  3  /

Type created.
```

2. Type Creation

```
SQL> CREATE OR REPLACE TYPE customer AS OBJECT
  2  (code number(5), name varchar2(30), contact_no varchar2(12), addr address, member procedure display );
  3  /

Type created.
```

3. Instantiating Object

```
SQL> edit objins;

SQL> @objins;
House No: 103A
Street: M.G.Road
City: Jaipur
State: Rajasthan
Pincode: 201301

PL/SQL procedure successfully completed.
```

4. Map method

```
SQL> CREATE OR REPLACE TYPE rectangle AS OBJECT
  2  (length number, width number, member function enlarge( inc number) return rectangle, member procedure display, map member function measure return number );
  3  /

Type created.
```

5. Type creation using Map

```
SQL> CREATE OR REPLACE TYPE BODY rectangle AS
  2  MEMBER FUNCTION enlarge(inc number) return rectangle IS
  3  BEGIN
  4  return rectangle(self.length + inc, self.width + inc);
  5  END enlarge;
  6  MEMBER PROCEDURE display IS
  7  BEGIN
  8  dbms_output.put_line('Length: ' || length); dbms_output.put_line('Width: ' || width);
  9  END display;
 10  MAP MEMBER FUNCTION measure return number IS
 11  BEGIN
 12  return (sqrt(length*length + width*width));
 13  END measure;
 14  END;
 15  /

Type body created.
```

6. Using Type body

```
SQL> DECLARE
  2  r1 rectangle; r2 rectangle; r3 rectangle; inc_factor number := 5;
  3  BEGIN
  4  r1 := rectangle(3, 4); r2 := rectangle(5, 7); r3 := r1.enlarge(inc_factor); r3.display;
  5  IF (r1 > r2) THEN -- calling measure function
  6  r1.display;
  7  ELSE r2.display;
  8  END IF;
  9  END;
 10 /
Length: 8
Width: 9
Length: 5
Width: 7
PL/SQL procedure successfully completed.
```

7. Order method

```
SQL> CREATE OR REPLACE TYPE rectangle AS OBJECT (length number, width number, member procedure display, order member function measure(r rectangle) return number );
  2 /
Type created.
```

8. Body of Procedure

```
CREATE OR REPLACE TYPE BODY rectangle AS
MEMBER PROCEDURE display IS
BEGIN
  dbms_output.put_line('Length: ' || length);
  dbms_output.put_line('Width: ' || width);
END display;
ORDER MEMBER FUNCTION measure(r rectangle) return number IS
BEGIN
  IF(sqrt(self.length*self.length + self.width*self.width)> sqrt(r.length*r.length +
  r.width*r.width))
  then return(1);
  ELSE
  return(-1);
  END IF;
  END measure;
END;
/
SQL> @typeorder;
Type body created.
```

9. Use Type order

```
DECLARE
r1 rectangle;
r2 rectangle;
BEGIN
r1 := rectangle(23, 44);
```

```

r2 := rectangle(15, 17);
r1.display;
r2.display;
IF (r1 > r2)
THEN
r1.display;
ELSE r2.display;
END IF;
END;
/

```

```

SQL> edit typeorderuse;
SQL> @typeorderuse;
Length: 23
Width: 44
Length: 15
Width: 17
Length: 23
Width: 44

PL/SQL procedure successfully completed.

```

10. Inheritance

```

SQL> CREATE OR REPLACE TYPE rectangle AS OBJECT
  2  (length number, width number, member function enlarge( inc number) return rectangle, NOT FINAL member procedure display) NOT FINAL
  3  /
Type created.

```

11. Type Creation

```

SQL> CREATE OR REPLACE TYPE BODY rectangle AS
  2  MEMBER FUNCTION enlarge(inc number) return rectangle IS
  3  BEGIN
  4  return rectangle(self.length + inc, self.width + inc);
  5  END enlarge;
  6  MEMBER PROCEDURE display IS
  7  BEGIN
  8  dbms_output.put_line('Length: ' || length);
  9  dbms_output.put_line('Width: ' || width);
 10  dbms_output.put_line('Width: ' || width);
 11  END display;
 12  END;
 13  /
Type body created.

```

12. Child creation

```
SQL> CREATE TYPE tabletop UNDER rectangle
 2  (
 3  material varchar2(20),
 4  OVERRIDING MEMBER PROCEDURE display);
 5  /

Type created.
```

13. Child Body

```
SQL> CREATE OR REPLACE TYPE BODY tabletop AS
 2  OVERRIDING MEMBER PROCEDURE display IS
 3  BEGIN
 4  dbms_output.put_line('Length: ' || length);
 5  dbms_output.put_line('Width: ' || width);
 6  dbms_output.put_line('Material: ' || material);
 7  END display;
 8  END;
 9  /

Type body created.
```

14. Performing Inheritance

```
SQL> DECLARE
 2  t1 tabletop; t2 tabletop;
 3  BEGIN
 4  t1:= tabletop(20, 10, 'Wood'); t2 := tabletop(50, 30, 'Steel'); t1.display;
 5  t2.display;
 6  END;
 7  /

Length: 20
Width: 10
Material: Wood
Length: 50
Width: 30
Material: Steel

PL/SQL procedure successfully completed.
```

15. Abstract Object

```
SQL> CREATE OR REPLACE TYPE rectangles AS OBJECT
 2  (length number, width number, NOT INSTANTIABLE NOT FINAL MEMBER PROCEDURE display)
 3  NOT INSTANTIABLE NOT FINAL
 4  /

Type created.
```

PL/SQL ASSIGNMENT 5

True or False:

1. When you use DML in a PL/SQL block, Oracle uses explicit cursors to track the data changes.

TRUE

2. EXPLICIT cursors are created by the programmer.
3. IMPLICIT cursors are created by the Oracle server.

4. The following code is supposed to display the lowest and highest elevations for a country name entered by the user. However, the code does not work. Fix the code by following the guidelines for retrieving data that you learned in this lesson.

```
DECLARE
v_country_name wf_countries.country_name%TYPE
:= 'United States of America';
v_lowest_elevation wf_countries.lowest_elevation%TYPE;
v_highest_elevation wf_countries.highest_elevation%TYPE;
BEGIN
SELECT lowest_elevation, highest_elevation
INTO v_lowest_elevation, v_highest_elevation
FROM wf_countries;
DBMS_OUTPUT.PUT_LINE('The lowest elevation in
'||country_name||' is '||v_lowest_elevation
||' and the highest elevation is '||
v_highest_elevation||'.');
END;
```

```
SQL> create table countries(country_name varchar2(30),lowest_elevation varchar2(30),highest_elevation varchar2(30));
Table created.

SQL> insert into countries(country_name,lowest_elevation,highest_elevation)
  2  values
  3  ('USA','BOSTON','CHICAGO');

1 row created.

SQL> DECLARE
  2 v_country_name countries.country_name%TYPE := 'USA';
  3 v_highest_elevation countries.highest_elevation%TYPE;
  4 v_lowest_elevation countries.lowest_elevation%TYPE;
  5 BEGIN
  6 SELECT lowest_elevation, highest_elevation INTO v_lowest_elevation, v_highest_elevation FROM countries;
  7 DBMS_OUTPUT.PUT_LINE('The lowest elevation in '||v_country_name||' is '||v_lowest_elevation ||' and the highest elevation is '|| v_highest_elevation||'.');
  8 end;
  9 /
The lowest elevation in USA is BOSTON and the highest elevation is CHICAGO.
PL/SQL procedure successfully completed.
```

5. How many transactions are shown in the following code? Explain your reasoning.

```
BEGIN
INSERT INTO my_savings (account_id, amount)
      VALUES (10377,200);
INSERT INTO my_checking(account_id, amount)
      VALUES (10378,100);
END;
```

2 transaction states

1st one is to insert data into my_saving and

2nd one is to insert data into my_checking.

6. Examine the following block. If you were to run this block, what data do you think would be saved in the database?

```
BEGIN
INSERT INTO endangered_species
      VALUES (100, 'Polar Bear','Ursus maritimus');
SAVEPOINT sp_100;
INSERT INTO endangered_species
      VALUES (200, 'Spotted Owl','Strix occidentalis');
SAVEPOINT sp_200;
INSERT INTO endangered_species
      VALUES (300, 'Asiatic Black Bear','Ursus thibetanus');
ROLLBACK TO sp_100;
COMMIT;
END;
```

/

```
SQL> create table endangered_species(specie_id int,specie_name varchar(20),specie_bio varchar(20));
Table created.

SQL> BEGIN
  2  INSERT INTO endangered_species VALUES (100, 'Polar Bear','Ursus maritimus');
  3  SAVEPOINT sp_100;
  4  INSERT INTO endangered_species VALUES (200, 'Spotted Owl','Strix occidentalis');
  5  SAVEPOINT sp_200;
  6  INSERT INTO endangered_species VALUES (300, 'Asiatic Black Bear','Ursus thibetanus');
  7  ROLLBACK TO sp_100;
  8  COMMIT;
  9  END;
10  /

PL/SQL procedure successfully completed.

SQL> select * from endangered_species;

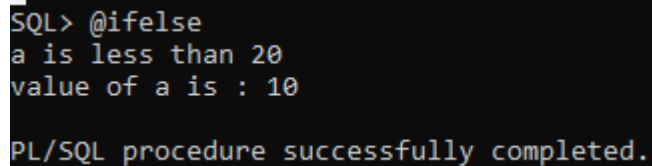
SPECIE_ID SPECIE_NAME      SPECIE_BIO
-----
100 Polar Bear      Ursus maritimus
```

7. List the three categories of control structures in PL/SQL with example.

Transaction control structure,
Conditional control structure,
Iterative control structure

1. IF ELSE.

```
DECLARE
a number(2) := 10;
BEGIN
a:= 10;
IF( a < 20 )
THEN
dbms_output.put_line('a is less than 20 ');
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```

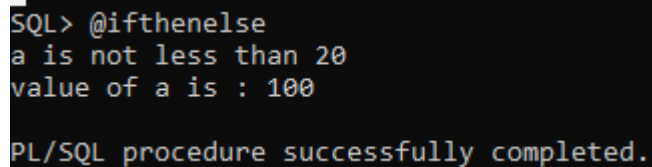


SQL> @ifelse
a is less than 20
value of a is : 10

PL/SQL procedure successfully completed.

2. IF THEN ELSE

```
DECLARE
a number(3) := 100;
BEGIN
IF( a < 20 )
THEN
dbms_output.put_line('a is less than 20 ');
ELSE
dbms_output.put_line('a is not less than 20 ');
END IF;
dbms_output.put_line('value of a is : ' || a);
END;
/
```



SQL> @ifthenelse
a is not less than 20
value of a is : 100

PL/SQL procedure successfully completed.

3. IF THEN ELSEIF

```
DECLARE
a number(3) := 100;
BEGIN
IF ( a = 10 )
THEN
dbms_output.put_line('Value of a is 10' );
```



```

ELSIF ( a = 20 )
THEN
dbms_output.put_line('Value of a is 20' );
ELSIF ( a = 30 )
THEN
dbms_output.put_line('Value of a is 30' );
ELSE
dbms_output.put_line('None of the values is matching');
END IF;
dbms_output.put_line('Exact value of a is: ' || a );
END;
/

```

```

SQL> @ifthenelseif
None of the values is matching
Exact value of a is: 100

PL/SQL procedure successfully completed.

```

4. CASE Statement.

```

DECLARE
grade char(1) := 'A';
BEGIN
CASE grade
when 'A' then
dbms_output.put_line('Excellent');
when 'B' then
dbms_output.put_line('Very good');
when 'C' then
dbms_output.put_line('Well done');
when 'D' then
dbms_output.put_line('You passed');
when 'F' then
dbms_output.put_line('Better try again');
else
dbms_output.put_line('No such grade');
END CASE;
END;
/

```

```

SQL> @case
Excellent

PL/SQL procedure successfully completed.

```

5. Searched CASE Statement.

```

DECLARE
grade char(1) := 'B';
BEGIN
case
when grade = 'A'
then
dbms_output.put_line('Excellent');
when grade = 'B'

```

```

then
dbms_output.put_line('Very good');
when grade = 'C'
then
dbms_output.put_line('Well done');
when grade = 'D'
then
dbms_output.put_line('You passed');
when grade = 'F' then
dbms_output.put_line('Better try again');
else
dbms_output.put_line('No such grade');
end case;
end;
/

```

```

SQL> @searchedcase
Very good

PL/SQL procedure successfully completed.

```

6. LOOPS

```

DECLARE
x number := 10;
BEGIN
LOOP
dbms_output.put_line(x);
x := x + 10;
IF x > 50
THEN exit;
END IF;
END LOOP;
dbms_output.put_line('After Exit x is: ' || x);
END;
/

```

```

SQL> @loops
10
20
30
40
50
After Exit x is: 60

PL/SQL procedure successfully completed.

```

8. List the keywords that can be part of an IF statement with one example each.

IF, THEN, ELSE, ELIF, END IF

```

DECLARE
a number(3) := 100;
b number(3) := 200;
BEGIN
IF( a = 100 )
THEN

```

```

IF( b = 200 )
THEN
dbms_output.put_line('Value of a is 100 and b is 200' );
END IF;
END IF;
dbms_output.put_line('Exact value of a is : ' || a );
dbms_output.put_line('Exact value of b is : ' || b );
END;
/

```

```

SQL> @nestedifthenelse
Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200

PL/SQL procedure successfully completed.

```

9. List the keywords that are a required part of an IF statement.

IF condition

THEN statement

END IF

- 11. Write a PL/SQL block to find the total monthly salary paid by the company for a given department number from the employees table. Display a message indicating whether the total salary is greater than or less than \$19,000. Test your block twice using the Administration department (department_id =10) and the IT department (department_id =60). The IT department should be greater than \$19,000, while the Administration department's total should be less than \$19,000.**

```

SQL> CREATE TABLE employees(SALARY INTEGER NOT NULL,department_id INTEGER NOT NULL PRIMARY KEY);
Table created.

```

```

SQL> INSERT INTO employees VALUES (230000,80);

```

```

1 row created.

```

```

SQL> INSERT INTO employees VALUES (13000,40);

```

```

1 row created.

```

```

SQL> select * from employees;

```

SALARY	DEPARTMENT_ID
230000	80
13000	40

```

DECLARE
v_salary employees.salary%TYPE;
BEGIN
SELECT SUM(salary)
INTO v_salary FROM employees WHERE department_id = 60;
DBMS_OUTPUT.PUT_LINE('Total salary is : '||v_salary);
IF v_salary < 19000 THEN DBMS_OUTPUT.PUT_LINE('Total salary in IT
department is smaller than $19K ');
ELSE DBMS_OUTPUT.PUT_LINE('Total salary in IT department is greater than
$19K ');
END IF;
END;

```

```

SQL> edit dept1;
_
SQL> @dept1;
11 /
Total salary is : 230000
Total salary in IT department is greater than $19K

PL/SQL procedure successfully completed.

```

11. What happens if we use the Marketing department (department_id=20) in the previous script?

```

DECLARE
v_salary employees.salary%TYPE;
BEGIN
SELECT SUM(salary)
INTO v_salary FROM employees WHERE department_id = 40;
DBMS_OUTPUT.PUT_LINE('Total salary is : '||v_salary);
IF v_salary < 19000 THEN DBMS_OUTPUT.PUT_LINE('Total salary in IT
department is smaller than $19K ');
ELSE DBMS_OUTPUT.PUT_LINE('Total salary in IT department is greater than
$19K ');
END IF;
END;
/

```

```

SQL> edit dept;
_
SQL> @dept;
Total salary is : 13000
Total salary in IT department is smaller than $19K

PL/SQL procedure successfully completed.

```

12. Alter the PL/SQL code to include an ELSIF to handle this situation.

```
DECLARE
v_salary employees.salary%TYPE;
BEGIN
SELECT SUM(salary)
INTO v_salary FROM employees WHERE department_id = 40;
DBMS_OUTPUT.PUT_LINE('Total salary is : '||v_salary);
IF v_salary < 19000 THEN
DBMS_OUTPUT.PUT_LINE('Total salary in Marketing department is smaller than
$19K ');
ELSIF v_salary = 19000 THEN
DBMS_OUTPUT.PUT_LINE('Total salary in Marketing department is equal $19K ');
ELSE
DBMS_OUTPUT.PUT_LINE('Total salary in Marketing department is greater than
$19K ');
END IF;
END;
```

```
SQL> edit deptelseif;
_
SQL> @deptelseif;
Total salary is : 13000
Total salary in Marketing department is smaller than $19K

PL/SQL procedure successfully completed.
```

13. Write a PL/SQL block to select the number of countries using a supplied currency name. If the number of countries is greater than 20, display “More than 20 countries”. If the number of countries is between 10 and 20, display “Between 10 and 20 countries”. If the number of countries is less than 10, display “Fewer than 10 countries”. Use a CASE statement.

```
DECLARE
v_no_of_countries countries.currency_code%TYPE;
BEGIN
SELECT COUNT(currency_code) INTO v_no_of_countries
FROM countries WHERE currency_code =001;
CASE
WHEN v_no_of_countries <10 THEN
DBMS_OUTPUT.PUT_LINE('Fewer than 10 countries');
WHEN v_no_of_countries BETWEEN 10 AND 20
THEN DBMS_OUTPUT.PUT_LINE('Between 10 and 20 countries');
ELSE DBMS_OUTPUT.PUT_LINE('More than 20 countries');
END CASE;
DBMS_OUTPUT.PUT_LINE('No of countries is : '||v_no_of_countries);
END;
/
```

```
SQL> create table countries(country_name varchar(3),country_code int,currency_code int);
```

```
Table created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('a',10,001);
```

```
1 row created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('b',11,001);
```

```
1 row created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('c',12,001);
```

```
1 row created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('d',13,001);
```

```
1 row created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('e',14,001);
```

```
1 row created.
```

```
SQL> insert into countries(country_name,country_code,currency_code)
2 values('f',15,001);
```

```
1 row created.
```

```
SQL> select * from countries;
```

COU	COUNTRY_CODE	CURRENCY_CODE
a	10	1
b	11	1
c	12	1
d	13	1
e	14	1
f	15	1

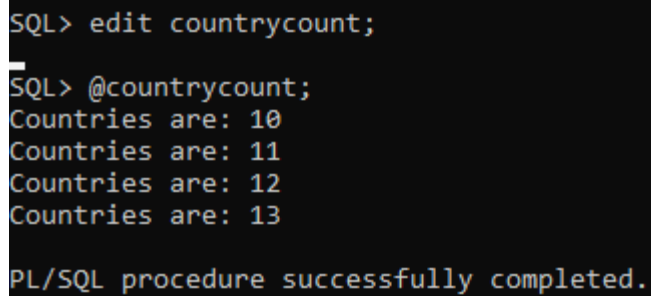
```
SQL> edit countrycode;
```

```
SQL> @countrycode;
Fewer than 10 countries
No of countries is : 6
```

```
PL/SQL procedure successfully completed.
```

14. Write a PL/SQL block to display the country_id and country_name values from the WF_COUNTRIES table for country_id whose values range from 1 through 3. Use a basic loop. Increment a variable from 1 through 3. Use an IF statement to test your variable and EXIT the loop after you have displayed the first 3 countries.

```
DECLARE
v_countryid countries.country_code%TYPE := 10;
v_countryname countries.country_name%TYPE;
v_counter NUMBER(2) := 10;
BEGIN
LOOP
SELECT country_code INTO v_countryid FROM countries
WHERE country_code = v_counter;
v_counter := v_counter + 1;
DBMS_OUTPUT.PUT_LINE('Countries are: '||v_countryid);
EXIT
WHEN v_counter > 13;
END LOOP;
END;
/
```



```
SQL> edit countrycount;
_
SQL> @countrycount;
Countries are: 10
Countries are: 11
Countries are: 12
Countries are: 13

PL/SQL procedure successfully completed.
```

15. Write a PL/SQL block to produce a list of available vehicle license plate numbers. These numbers must be in the following format: NN-MMM, where NN is between 60 and 65, and MMM is between 100 and 110. Use nested FOR loops. The outer loop should choose numbers between 60 and 65. The inner loop should choose numbers between 100 and 110, and concatenate the two numbers together.

```
SQL> begin
  2  FOR v_out IN 60..65 LOOP
  3  FOR v_inn IN 100..110 LOOP
  4  DBMS_OUTPUT.PUT_LINE(v_out||'-'||v_inn);
  5  END LOOP;
  6  END LOOP;
  7  END;
  8  /
60->100
60->101
60->102
60->103
60->104
60->105
60->106
60->107
60->108
60->109
60->110
61->100
61->101
61->102
61->103
61->104
61->105
61->106
61->107
61->108
61->109
61->110
62->100
62->101
62->102
62->103
62->104
62->105
62->106
62->107
62->108
62->109
62->110
63->100
63->101
63->102
63->103
63->104
63->105
63->106
63->107
63->108
63->109
63->110
64->100
64->101
64->102
64->103
64->104
64->105
64->106
64->107
64->108
64->109
64->110
65->100
65->101
65->102
65->103
65->104
65->105
65->106
65->107
65->108
65->109
65->110
PL/SQL procedure successfully completed.
```