DMDD ASSIGNMENT 4

Aishwarya Wagdarkar - 002964461

Ans 1: It is not possible to update, insert, delete in one select statement. But this can be done using MERGE statement.

This type of situation is required in scenarios where you want to check if the data exists or not then UPDATE, DELETE, INSERT.

For example : if record exists than UPDATE if not then INSERT and after updating if row fails to meet certain criterion then DELETE it.

Oracle is the only database with the solution for these type of problems. The MERGE operation merges data from a source result set into a target table by using JOIN conditions depending on a criterion you define and whether or not the data from the source already exists in the target. Conditional INSERT, UPDATE, DELETE are used in new SQL statement based on no. of rows fetched.

Syntax :
```
MERGE <target_table> [AS TARGET]
USING <table_source> [AS SOURCE]
ON <search_condition>
[WHEN MATCHED
THEN <merge_matched> ]
[WHEN NOT MATCHED [BY TARGET]
THEN <merge_not_matched> ]
[WHEN NOT MATCHED BY SOURCE
THEN <merge_matched> ];
```

Example :
```
MERGE INTO COMM ep
USING (SELECT * FROM EMP) emp
ON (ep.EMPNO = emp.EMP)
WHEN MATCHED
THEN UPDATE SET ep.COMM = 1000
DELETE WHERE (SAL < 2000)
WHEN NOT MATCHED THEN
INSERT (ep.EMPNO, ep.NAME, ep.DEPTNO, ep.COMM)
VALUES('10', 'JOHN', 'HEALTH', 1500);
```

JOIN determines which row already exists and have to be updated. JOIN between commission table (aliased as ep) and subquery aliased as emp table. When the operation join is performed and successfully executed the values are matched and UPDATE task is performed when data is MATCHED. If JOIN query is successfully and no data is found INSERT operation is performed on NOT MATCHED execution. Thus, if table EMP does not have corresponding EMPNO in COMMISSION table then data will be INSETED into commission table.

Ans 2: LEAD and LAG displays the previous and following values in comparison with to current value you are looking at. They give completely opposite results. Like windows functions we need to specify the column in which we have to run the function over and column we are portioning by and column we are ordering by. Generally ORDER BY is used while using these functions. These functions are called as non-aggregate Windows Function

These functions use OVER() clause and PARTITION BY and ORDER BY. PARTITION BY is not mandatory but ORDER BY is almost always necessary.

The LEAD function is used to access data from SUBSEQUENT rows along with data from the current row.

Syntax : LEAD(scalar_expression, N, default) OVER ([partition_by/order_by)

Scalar_expression : The value that will be returned based on the offset.

N : The number of rows to get a value from backwards from the current row. The default value is 1 if none is supplied.

Default : If offset exceeds the partition's scope, the default value will be returned. NULL is returned if no default value is supplied.

Over(partition_by/ order_by) - Partition by splits the FROM clause's result set into divisions to which the function can be applied. If the PARTITION BY clause is not specified, the function interprets the whole result set as a single group. The order by clause sorts in ascending order by default.

Example :

```
-----create department table----

--DEPARTMENT TABLE CREATION

CREATE TABLE DEPARTMENT

(DEPT_ID NUMBER NOT NULL,

DEPT_NAME VARCHAR(50) NOT NULL,

CONSTRAINT DEPARTMENT_PK PRIMARY KEY(DEPT_ID));


--- EMPLOYEE TABLE CREATION

CREATE TABLE EMPLOYEE

(EMP_NO NUMBER(4,0) NOT NULL,
```

NAME VARCHAR (50) NOT NULL,

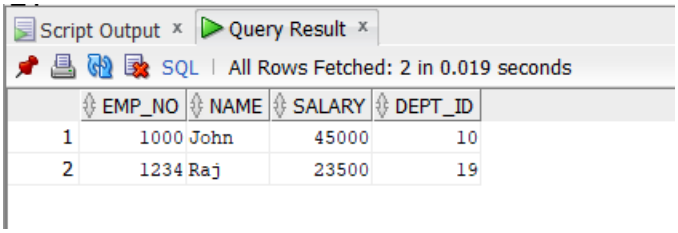
SALARY NUMBER(7,2),

DEPT_ID NUMBER(4,0),

CONSTRAINT EMP_PK PRIMARY KEY(EMP_NO));

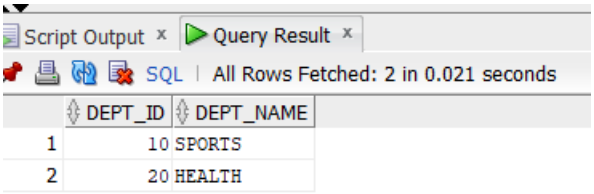
Now we have created 2 tables DEPARTMENT and EMPLOYEE and added 2 records for each table.

EMPLOYEE TABLES LOOK LIKE BELOW



	EMP_NO	NAME	SALARY	DEPT_ID
1	1000	John	45000	10
2	1234	Raj	23500	19

DEPARTMENT TABLE LOOKS LIKE BELOW

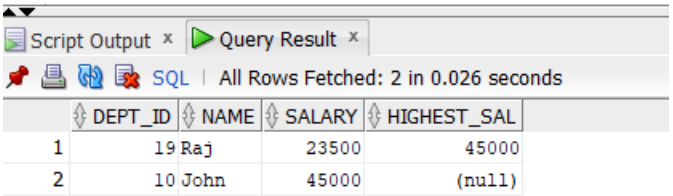


	DEPT_ID	DEPT_NAME
1	10	SPORTS
2	20	HEALTH

EXAMPLE FOR LEAD:

SELECT DEPT_ID, NAME, SALARY,

LEAD(SALARY,1) OVER (ORDER BY SALARY) AS HIGHEST_SAL FROM EMPLOYEE;



	DEPT_ID	NAME	SALARY	HIGHEST_SAL
1	19	Raj	23500	45000
2	10	John	45000	(null)

LEAD Function will sort in ascending order all the values of salary from employee table and return highest value since we have used an offset as 1.

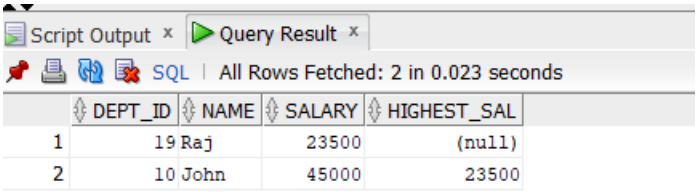
The **LAG function** is used to access data from **PREVIOUS** rows along with data from the current row. Without a self join, the prior value may be returned on the same record, making comparisons simple.

Syntax : LAG(scalar_expression[N][default]) OVER ([partition_by/order_by)

EXAMPLE :

SELECT DEPT_ID, NAME, SALARY,

LEAD(SALARY,1) OVER (ORDER BY SALARY) AS HIGHEST_SAL FROM EMPLOYEE;



	DEPT_ID	NAME	SALARY	HIGHEST_SAL
1	19	Raj	23500	(null)
2	10	John	45000	23500

Ans 3.

a) PL/SQL Anonymous Block

PL/SQL is a block-structured programming language, which means that the code is divided into blocks. There are three components to a PL/SQL block: declaration, executable, and exception-handling. The executable part of a block is required, while the declaration and exception-handling sections are optional. Named block like functions or procedures are PL/SQL blocks which can be stored in the database and can be used later.

Whereas **anonymous PL/SQL** are the unnamed blocks statements that include SQL statements as well as PL/SQL control statements. Data of anonymous block is not kept in the Oracle Database server so it is used only once. Therefore, anonymous blocks in PL/SQL might be handy for testing.

SYNTAX:

Declare

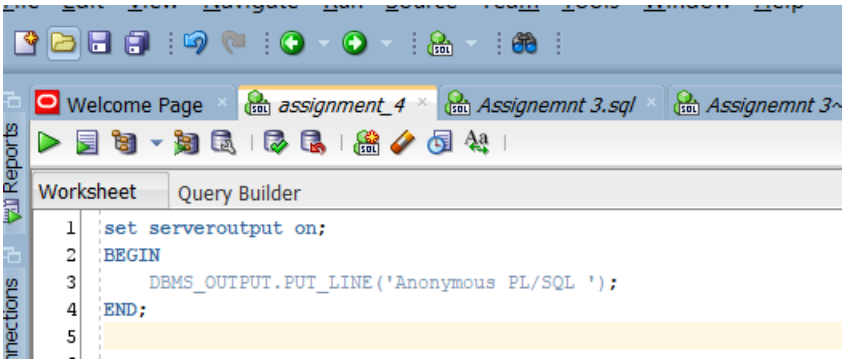
Begin

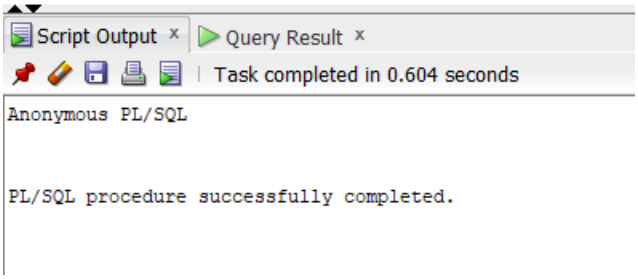
Exception

End

Example :

```
BEGIN
        DBMS_OUTPUT.PUT_LINE('Hello SQL');
END;
```





b) Cursor - The Oracle engine uses a work area for internal processing and storage of content in order to execute SQL commands. SQL's operations have exclusive access to this work space. The PL/SQL construct 'Cursor' allows the user to name the work area and retrieve the information contained in it.

A cursor's primary role is to obtain data from a result set one row at a time, as opposed to SQL commands, which work on the entire result set at once. Cursors are utilized when a user needs to edit records in a database table in a singleton or row by row way.

There are 2 types of Cursor:

Implicit Cursor:

When we run a SQL statement inside a PL/SQL block, the Oracle server automatically creates and destroys implicit cursors. The Oracle server automatically opens, fetches, processes, and shuts the implicit cursor without any programmer interaction, which is why implicit cursors are significantly quicker than explicit cursors, resulting in a simple and elegant code. When a DML statement (INSERT, UPDATE, or DELETE) is issued, it is accompanied by an implicit cursor. The

cursor contains the data that needs to be inserted for INSERT operations. The cursor indicates the rows that will be impacted by UPDATE and DELETE actions.

Explicit Cursor:

Explicit cursors are cursors that have been programmed to provide the user additional control over the context area. In the PL/SQL Block's declaration section, an explicit cursor should be defined. It's based on a SELECT statement that produces several rows.

Syntax : CURSOR cursor_name IS select_statement;

The steps for working with an explicit cursor are as follows:

- Declaring the cursor for memory initialization.
- Allocating memory via opening the cursor.
- Obtaining the cursor in order to retrieve the data.
- To free up the allotted memory, close the cursor.

c) PL/Sql parameter modes

IN : Unless it was explicitly initialized with a default value, an IN formal parameter is initialized to the actual argument with which it was called. The IN parameter can be accessed within the called program, but it cannot be assigned a new value by the called program. The real parameter always holds the value that was set previous to the call when control returns to the caller application.

OUT : The actual argument with which an OUT formal parameter was called is used to initialize it. The calling program can use the formal parameter to reference and assign new values. If the calling program exits without throwing an exception, the real parameter gets the formal parameter's final value. If a handled exception occurs, the real parameter takes the formal parameter's most recent value. The value of the actual argument remains the same as it was before the call if an unhandled exception occurs.

INOUT : An IN OUT formal parameter, like an IN parameter, is set to the real parameter with which it was called. An IN OUT formal parameter, like an OUT parameter, is changeable by the called program, and the formal parameter's last value is transferred to the calling program's real parameter if the called program ends without an error. If a handled exception occurs, the real parameter is set to the formal parameter's most recent value. If an unhandled exception occurs, the real parameter's value stays the same as it was before the call.

d) Different types of triggers

SQL Server triggers are stored procedures that are run automatically in reaction to events in the database object, database, or server. Trigger is automatically called and the PL/SQL trigger block is executed when the triggering SQL statement is executed.

BEFORE TRIGGER : Before the triggering DML statement (INSERT, UPDATE, DELETE) executes, the BEFORE trigger executes. Depending on the BEFORE trigger conditions block, the triggering SQL query may or may not run.

Syntax : CREATE TRIGGER **trigger_name**
BEFORE INSERT
ON **table_name** FOR EACH ROW
trigger_body;

AFTER TRIGGER : After the triggering DML statement (INSERT, UPDATE, DELETE) has completed, the AFTER trigger will fire. Before executing Database operations, the trigger SQL statement is executed as soon as it is accompanied by the trigger code.

Syntax : CREATE TRIGGER trigger_name
AFTER INSERT
ON table_name FOR EACH ROW
trigger_body;

ROW TRIGGER : The ROW trigger fires for each and every record that is INSERTING, UPDATEING, or DELETING data from a database table. If row deletion is set as a trigger event, the trigger file deletes five rows from the database each time it is run.

STATEMENT TRIGGER: For each statement, the statement trigger fires just once. If row deletion is set as a trigger event, the trigger file will remove all five rows from the database at once.

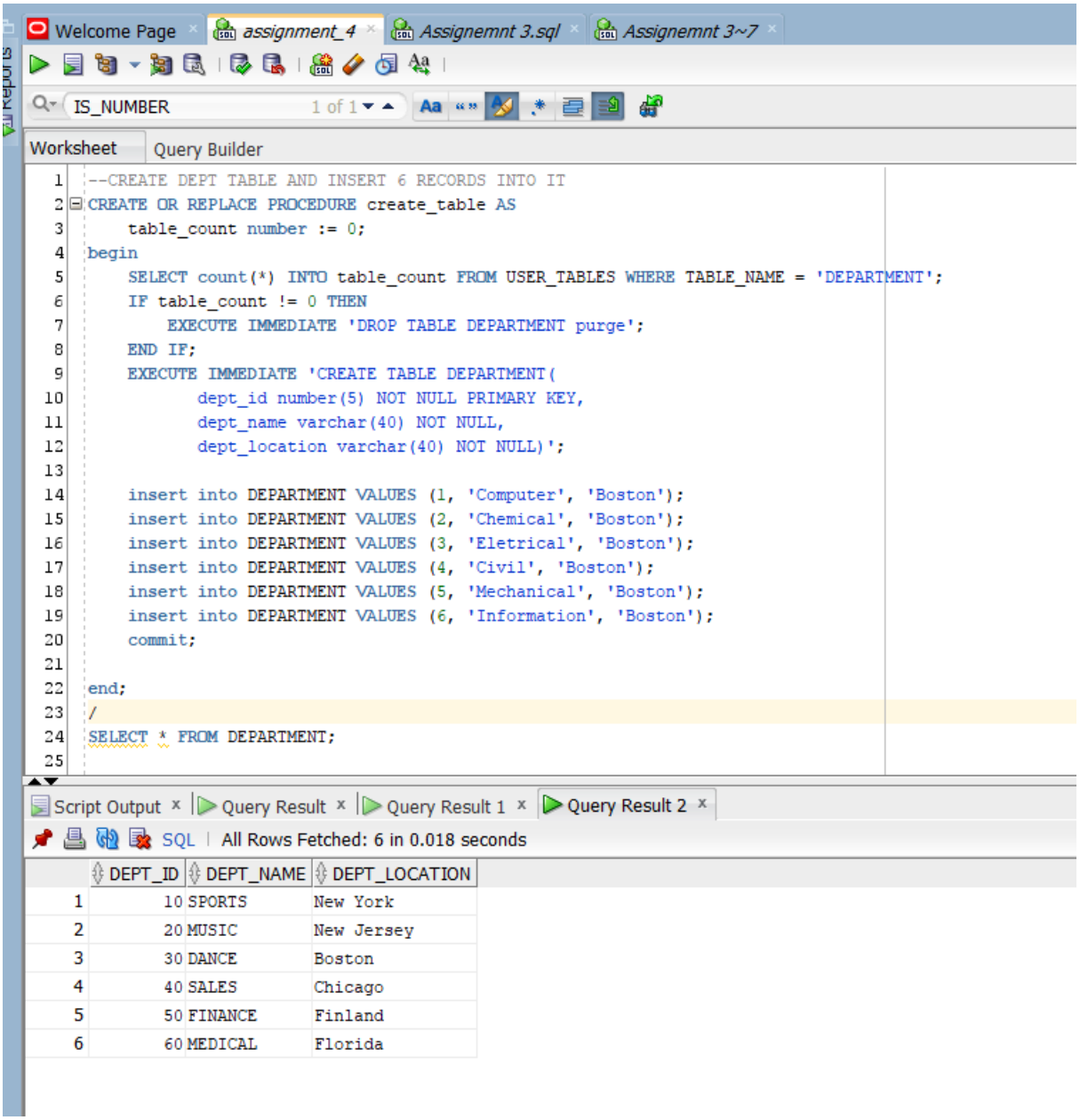
e) Cartesian join: The CARTESIAN JOIN which is also called as CROSS JOIN returns two or more connected tables' sets of all records in form of Cartesian product. In the presence of where condition cartesian join behaves like a INNER JOIN. Cross joins are identical to inner joins that means join-condition is always True.

Syntax : SELECT table1.column1, table2.column2…

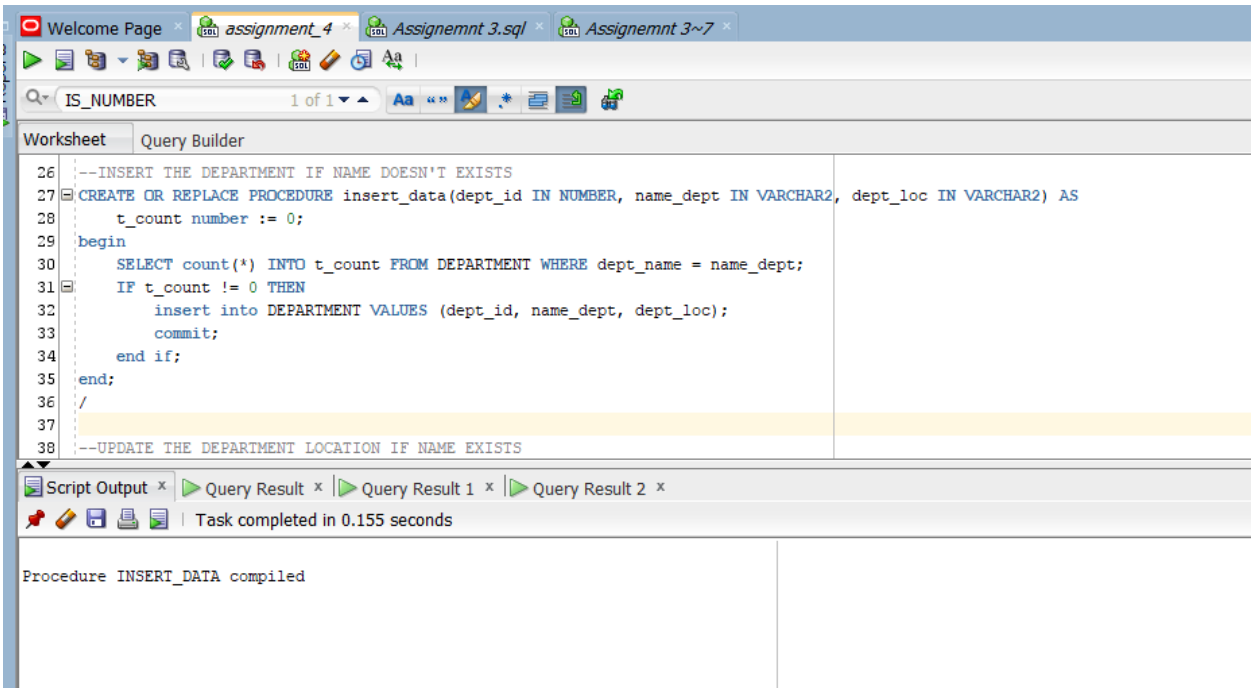
FROM TABLE1,TABLE2,TABLE3

Example : Select employee.Name, employee.dept, department.dept from employee CROSS JOIN department

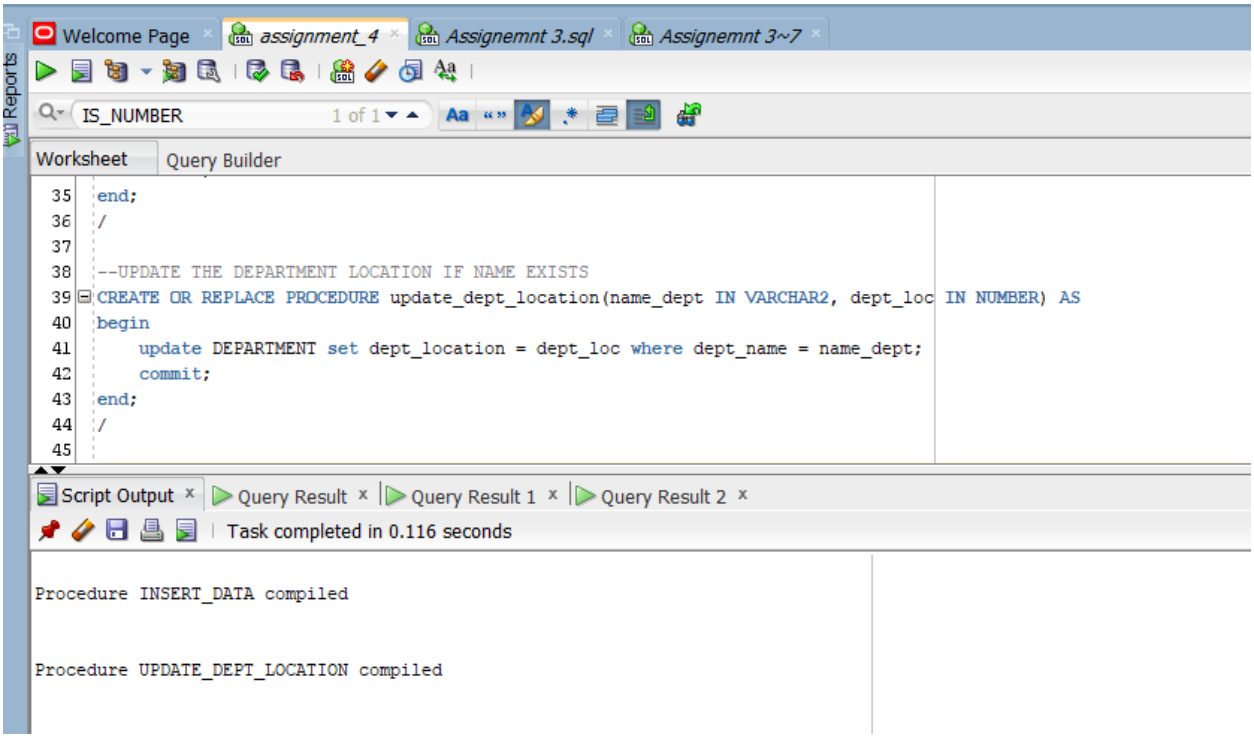
4.A. CREATE DEPT TABLE AND INSERT 6 RECORDS INTO IT



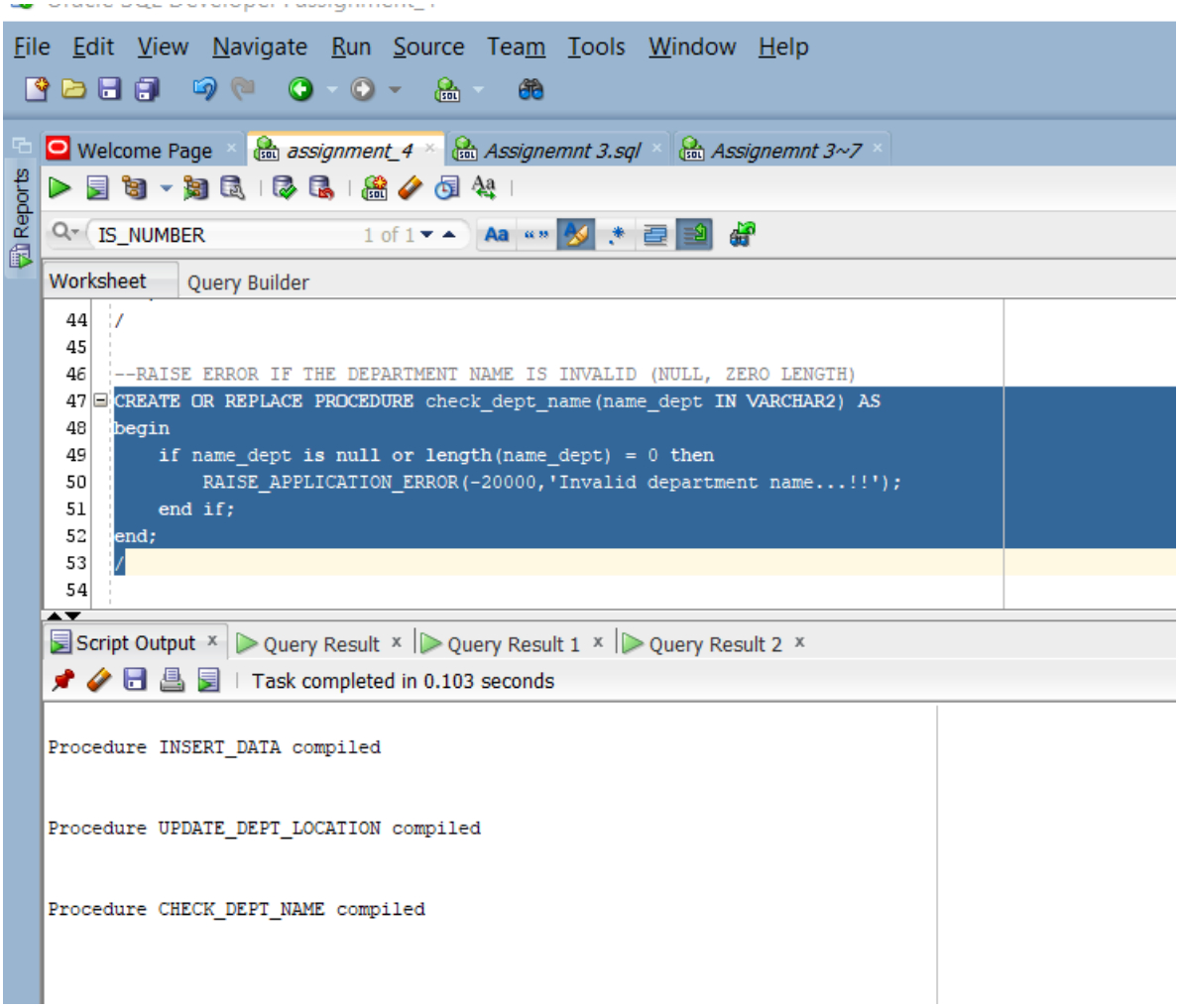
B. INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS



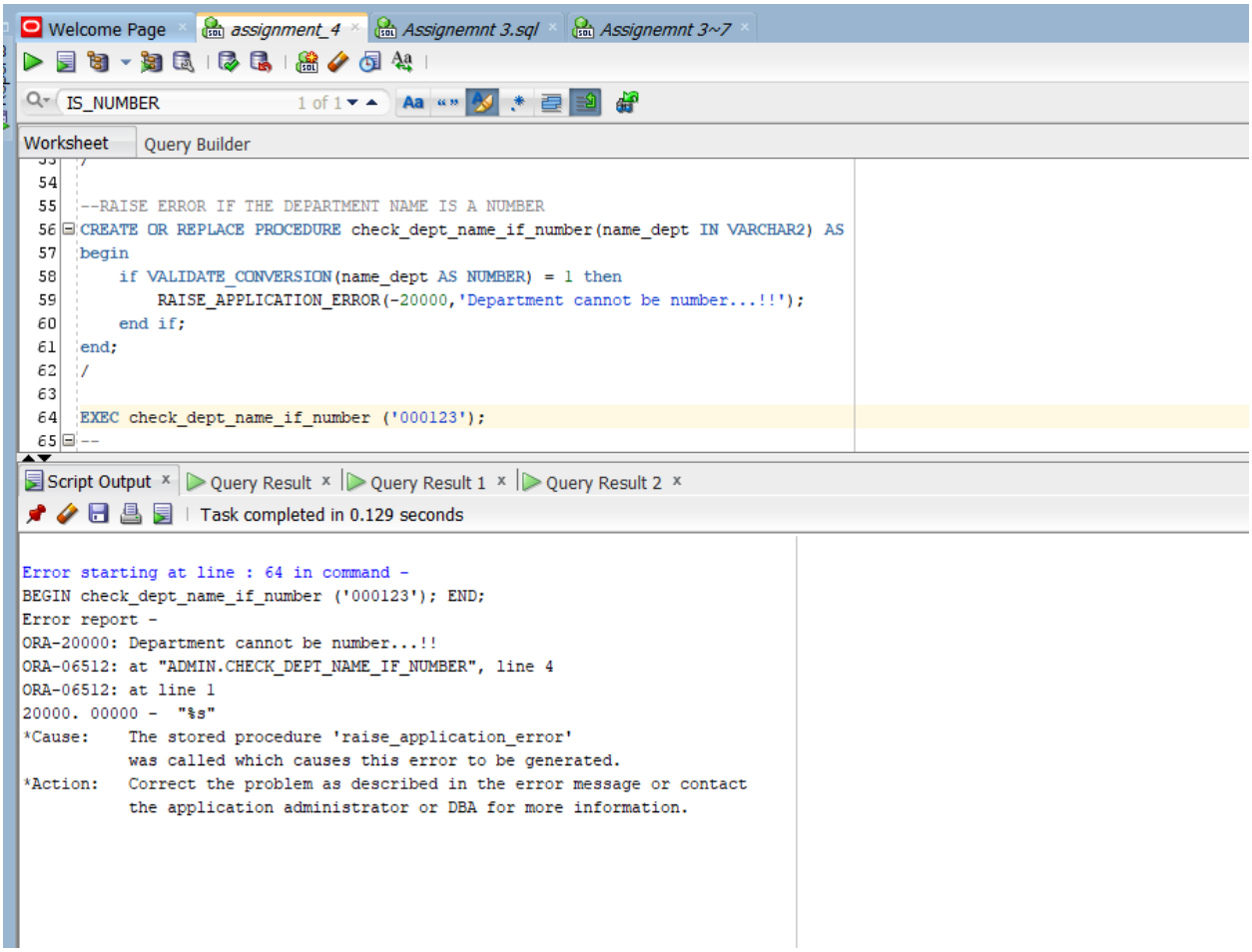
C. UPDATE THE DEPARTMENT LOCATION IF NAME EXISTS



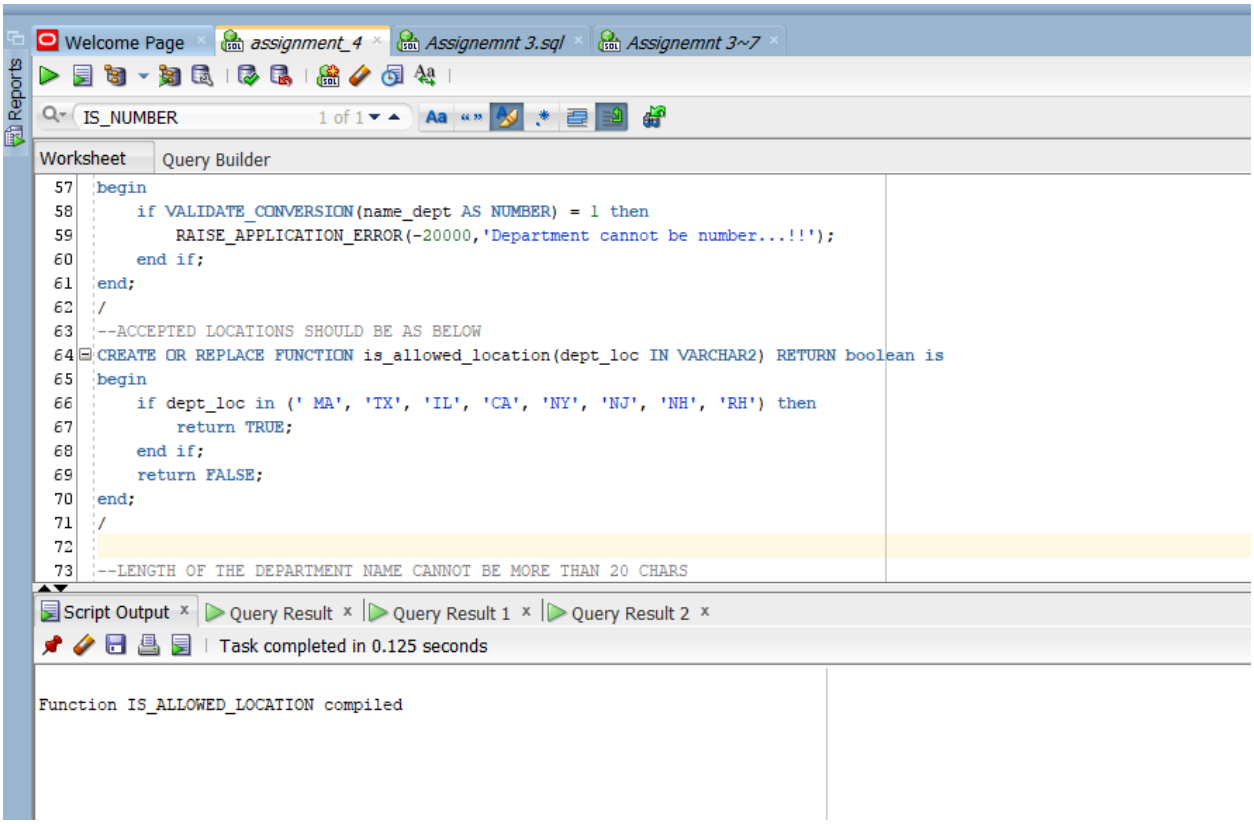
D. RAISE ERROR IF THE DEPARTMENT NAME IS INVALID (NULL, ZERO LENGTH)



E. RAISE ERROR IF THE DEPARTMENT NAME IS A NUMBER

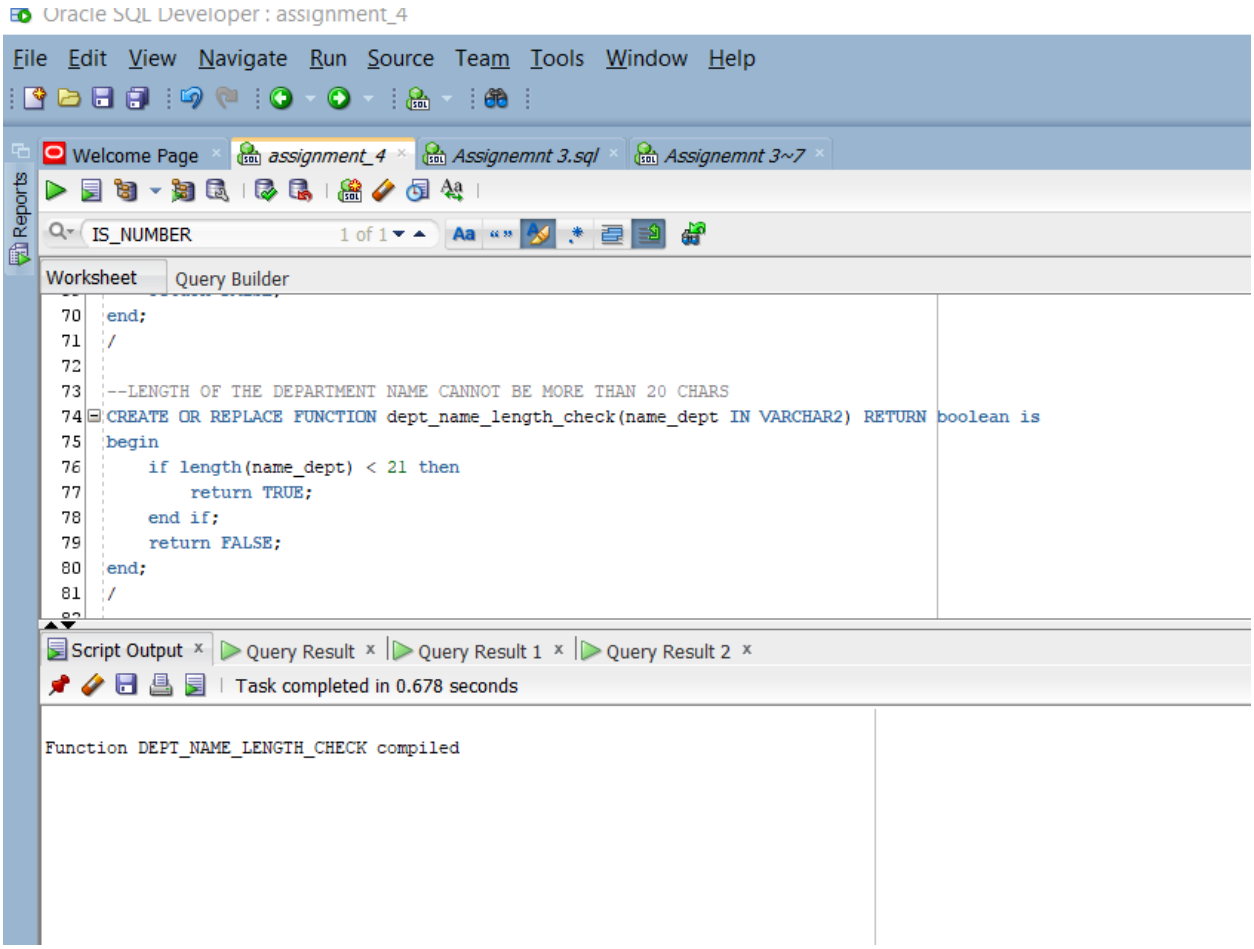


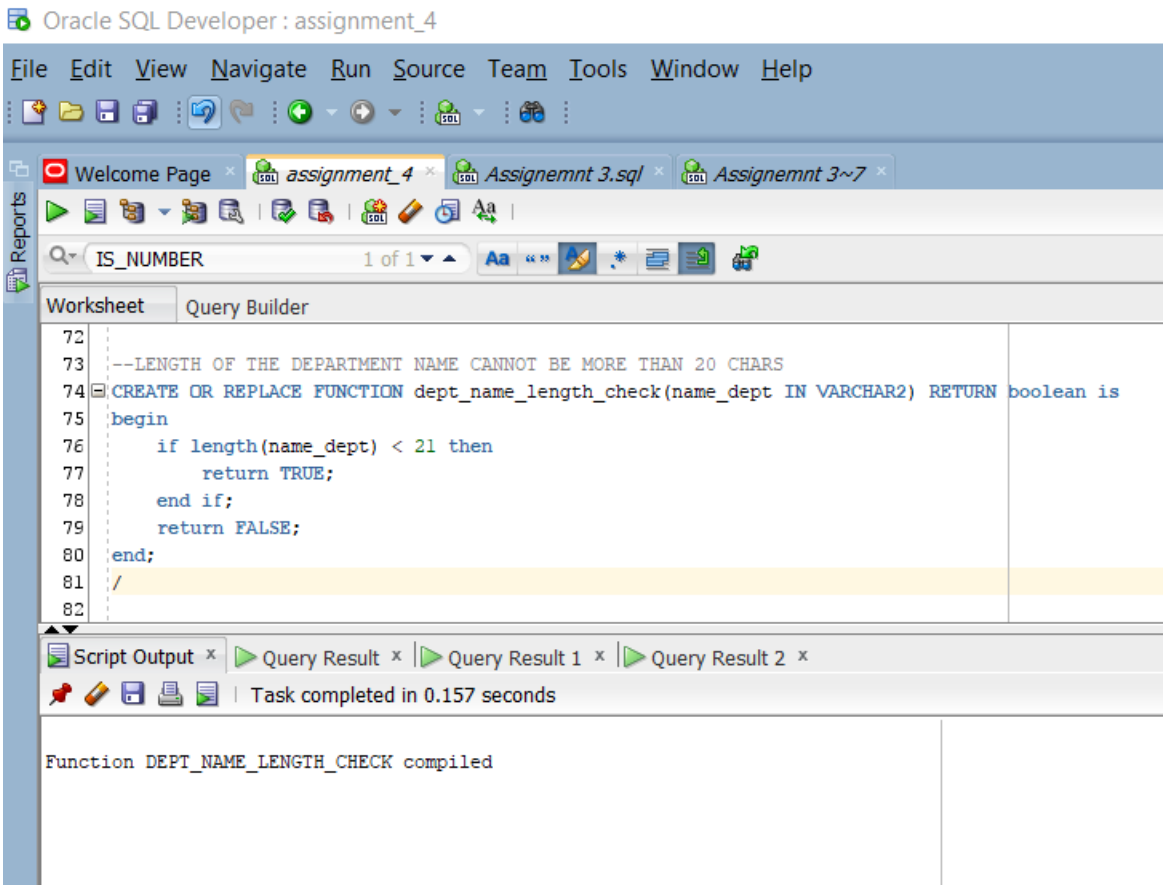
F. ACCEPTED LOCATIONS SHOULD BE AS BELOW MA, TX, IL, CA, NY, NJ, NH, RH



G. DEPARTMENT ID SHOULD BE AUTO-GENERATED

H. LENGTH OF THE DEPARTMENT NAME CANNOT BE MORE THAN 20


Oracle SQL Developer : assignment_4
File Edit View Navigate Run Source Team Tools Window Help
Welcome Page
assignment_4
Assignemnt 3.sql
Assignemnt 3~7
Reports
IS_NUMBER
1 of 1
Worksheet
Query Builder
70 end;
71 /
72
73 --LENGTH OF THE DEPARTMENT NAME CANNOT BE MORE THAN 20 CHARS
74 CREATE OR REPLACE FUNCTION dept_name_length_check(name_dept IN VARCHAR2) RETURN boolean is
75 begin
76 if length(name_dept) < 21 then
77 return TRUE;
78 end if;
79 return FALSE;
80 end;
81 /
Script Output
Query Result
Query Result 1
Query Result 2
Task completed in 0.678 seconds
Function DEPT_NAME_LENGTH_CHECK compiled



I. WHILE INSERTING THE DEPARTMENT NAME CONVERT EVERYTHING TO CAMEL CASE

```
/*
CREATE SEQUENCE dept_seq
        MINVALUE 1
        START WITH 1
        INCREMENT BY 1
        CACHE 10
/*


--INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS
CREATE OR REPLACE PROCEDURE insert_data(dept_id IN NUMBER, name_dept IN VARCHAR2, dept_loc IN VARCHAR2) AS
```

```
    t_count number := 0;
begin
    --MAKE SURE DEPARTMENT NAME IS UNIQUE
    SELECT count(*) INTO t_count FROM DEPARTMENT WHERE dept_name =
name_dept;
    IF t_count != 0 and check_dept_name(name_dept) and
check_dept_name_if_number(name_dept) and is_allowed_location(dept_loc) and
dept_name_length_check(name_dept) THEN
        --WHILE INSERTING THE DEPARTMENT NAME CONVERT EVERYTHING TO
CAMEL CASE
        insert into DEPARTMENT VALUES (dept_id, replace(initcap(name_dept),' '),
dept_loc);
        commit;
    end if;
end;
/
```

J. MAKE SURE DEPARTMENT NAME IS UNIQUE

```
/*
CREATE SEQUENCE dept_seq
            MINVALUE 1
            START WITH 1
            INCREMENT BY 1
            CACHE 10
/*
```

```
--INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS
CREATE OR REPLACE PROCEDURE insert_data(dept_id IN NUMBER, name_dept IN
VARCHAR2, dept_loc IN VARCHAR2) AS
    t_count number := 0;
```

```
begin
    --MAKE SURE DEPARTMENT NAME IS UNIQUE
    SELECT count(*) INTO t_count FROM DEPARTMENT WHERE dept_name =
name_dept;
    IF t_count != 0 and check_dept_name(name_dept) and
check_dept_name_if_number(name_dept) and is_allowed_location(dept_loc) and
dept_name_length_check(name_dept) THEN
        --WHILE INSERTING THE DEPARTMENT NAME CONVERT EVERYTHING TO
CAMEL CASE
        insert into DEPARTMENT VALUES (dept_id, replace(initcap(name_dept),' '),
dept_loc);
        commit;
    end if;
end;
/
```