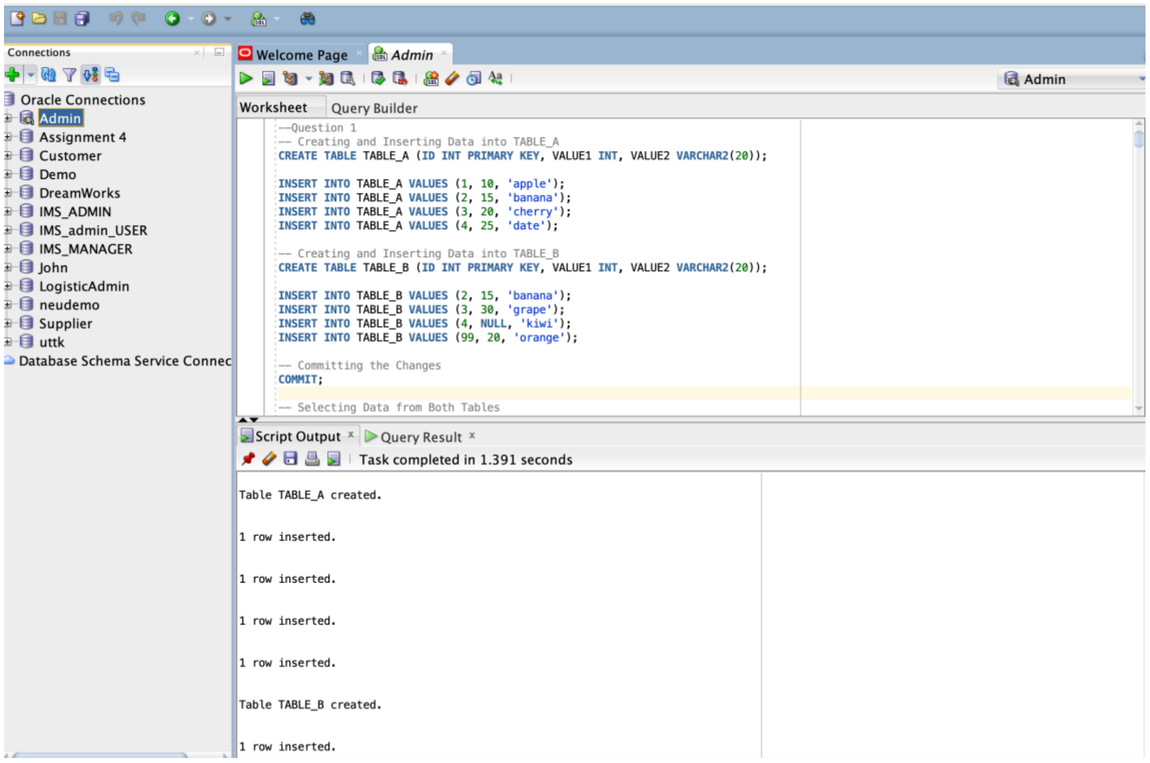
Assignment 6 - Individual submission - 100 points

1. ***Is it possible to insert, update and delete within one select statement? If so, how can we do that? Provide an example for your answer. (Hint: Use Merge) - 10 points***

Yes, utilizing the MERGE command in certain database systems like Microsoft SQL Server and Oracle, it is feasible to do insert, update, and delete actions all within one SQL query. You can conditionally execute an insert, update, or delete action depending on a given condition by using the MERGE statement.

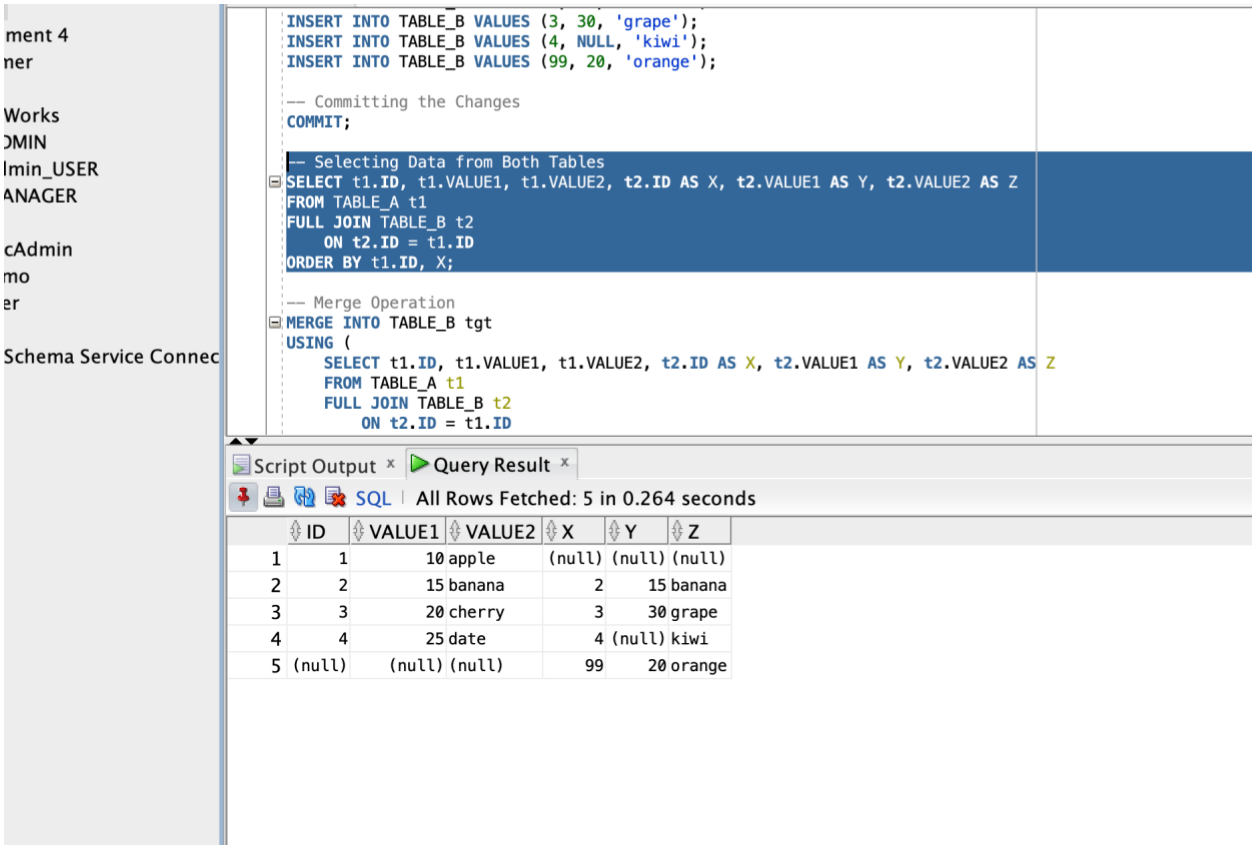
Creating table for implementing merge:



```
--Question 1
-- Creating and Inserting Data into TABLE_A
CREATE TABLE TABLE_A (ID INT PRIMARY KEY, VALUE1 INT, VALUE2 VARCHAR2(20));

INSERT INTO TABLE_A VALUES (1, 10, 'apple');
INSERT INTO TABLE_A VALUES (2, 15, 'banana');
INSERT INTO TABLE_A VALUES (3, 20, 'cherry');
INSERT INTO TABLE_A VALUES (4, 25, 'date');

-- Creating and Inserting Data into TABLE_B
CREATE TABLE TABLE_B (ID INT PRIMARY KEY, VALUE1 INT, VALUE2 VARCHAR2(20));

INSERT INTO TABLE_B VALUES (2, 15, 'banana');
INSERT INTO TABLE_B VALUES (3, 30, 'grape');
INSERT INTO TABLE_B VALUES (4, NULL, 'kiwi');
INSERT INTO TABLE_B VALUES (99, 20, 'orange');

-- Committing the Changes
COMMIT;

-- Selecting Data from Both Tables
```

Task completed in 1.391 seconds

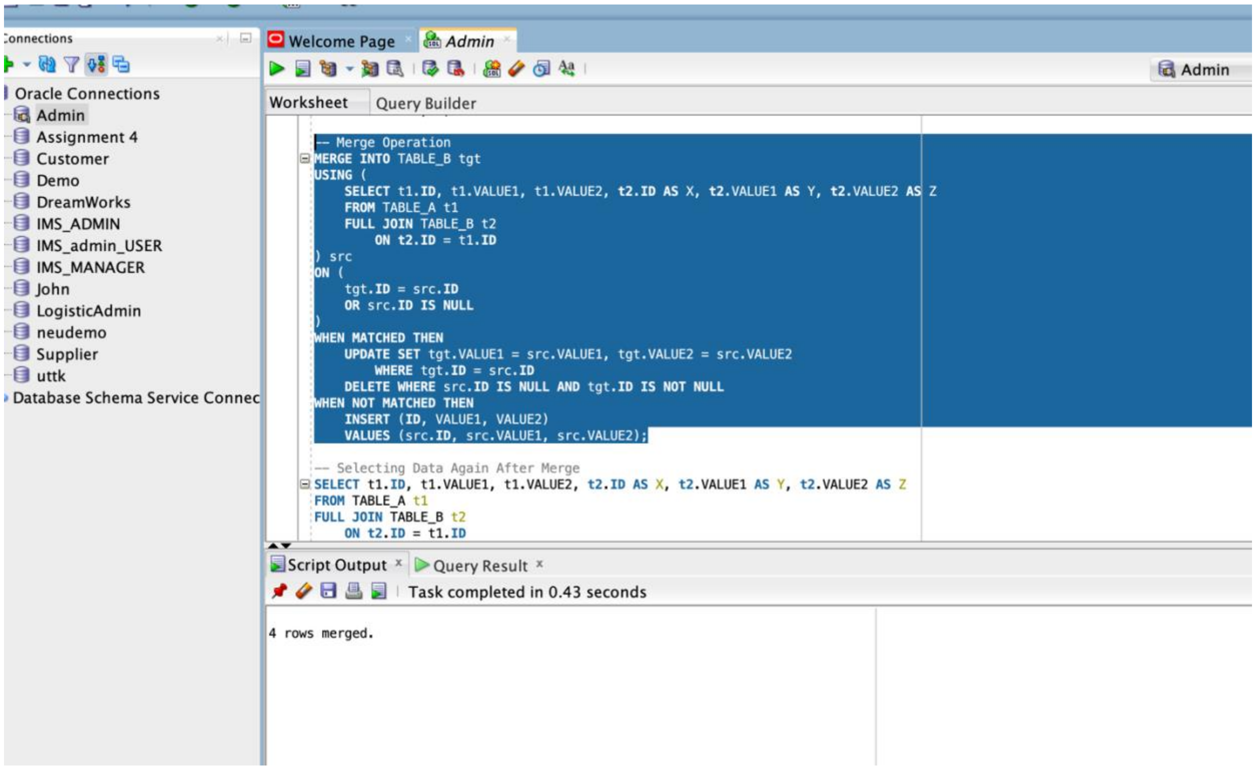
```
Table TABLE_A created.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


Table TABLE_B created.


1 row inserted.
```



```
INSERT INTO TABLE_B VALUES (3, 30, 'grape');
INSERT INTO TABLE_B VALUES (4, NULL, 'kiwi');
INSERT INTO TABLE_B VALUES (99, 20, 'orange');

-- Committing the Changes
COMMIT;

-- Selecting Data from Both Tables
SELECT t1.ID, t1.VALUE1, t1.VALUE2, t2.ID AS X, t2.VALUE1 AS Y, t2.VALUE2 AS Z
FROM TABLE_A t1
FULL JOIN TABLE_B t2
    ON t2.ID = t1.ID
ORDER BY t1.ID, X;

-- Merge Operation
MERGE INTO TABLE_B tgt
USING (
    SELECT t1.ID, t1.VALUE1, t1.VALUE2, t2.ID AS X, t2.VALUE1 AS Y, t2.VALUE2 AS Z
    FROM TABLE_A t1
    FULL JOIN TABLE_B t2
        ON t2.ID = t1.ID
```

All Rows Fetched: 5 in 0.264 seconds

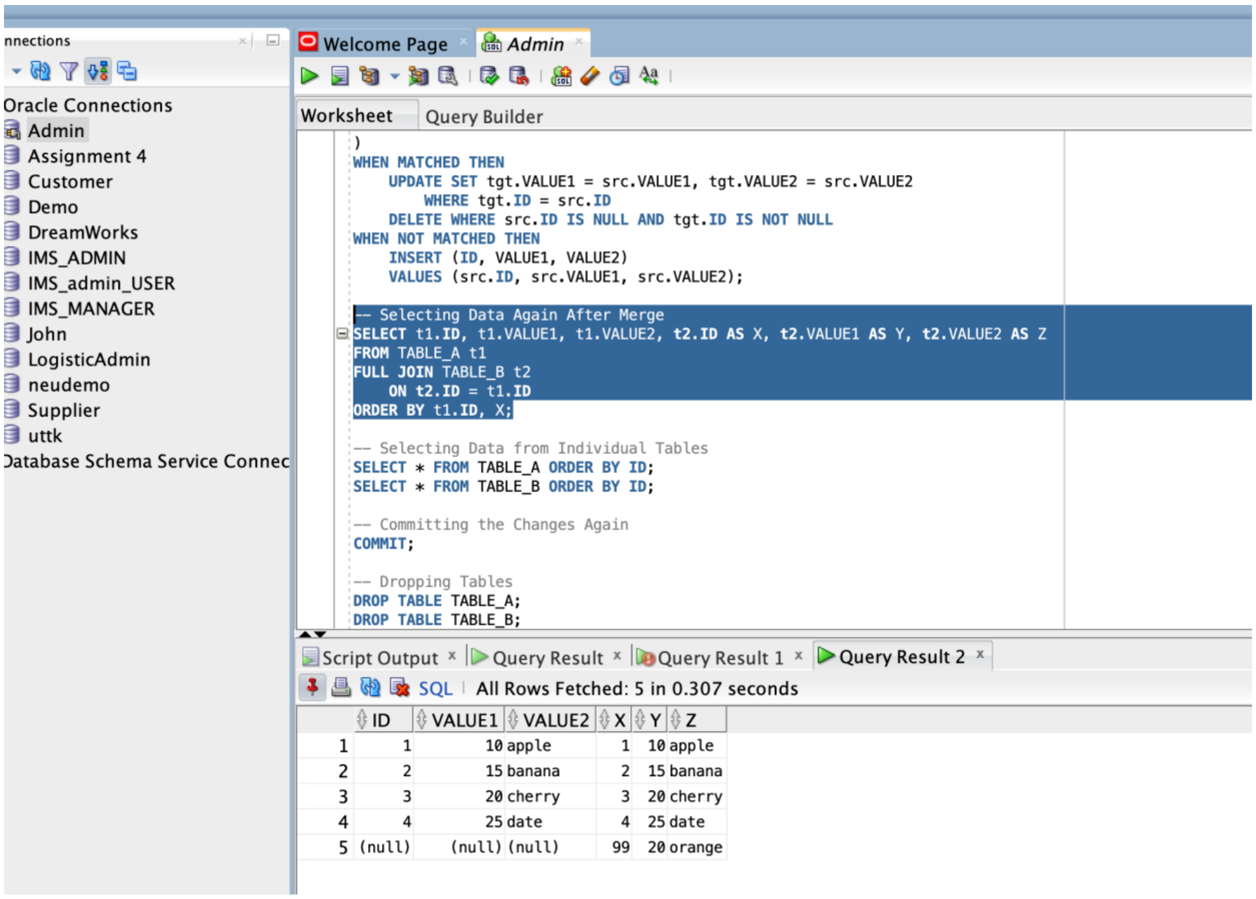
	ID	VALUE1	VALUE2	X	Y	Z
1	1	10	apple	(null)	(null)	(null)
2	2	15	banana	2	15	banana
3	3	20	cherry	3	30	grape
4	4	25	date	4	(null)	kiwi
5	(null)	(null)	(null)	99	20	orange



```
-- Merge Operation
MERGE INTO TABLE_B tgt
USING (
    SELECT t1.ID, t1.VALUE1, t1.VALUE2, t2.ID AS X, t2.VALUE1 AS Y, t2.VALUE2 AS Z
    FROM TABLE_A t1
    FULL JOIN TABLE_B t2
        ON t2.ID = t1.ID
) src
ON (
    tgt.ID = src.ID
    OR src.ID IS NULL
)
WHEN MATCHED THEN
    UPDATE SET tgt.VALUE1 = src.VALUE1, tgt.VALUE2 = src.VALUE2
        WHERE tgt.ID = src.ID
    DELETE WHERE src.ID IS NULL AND tgt.ID IS NOT NULL
WHEN NOT MATCHED THEN
    INSERT (ID, VALUE1, VALUE2)
    VALUES (src.ID, src.VALUE1, src.VALUE2);

-- Selecting Data Again After Merge
SELECT t1.ID, t1.VALUE1, t1.VALUE2, t2.ID AS X, t2.VALUE1 AS Y, t2.VALUE2 AS Z
FROM TABLE_A t1
FULL JOIN TABLE_B t2
    ON t2.ID = t1.ID
```

Task completed in 0.43 seconds

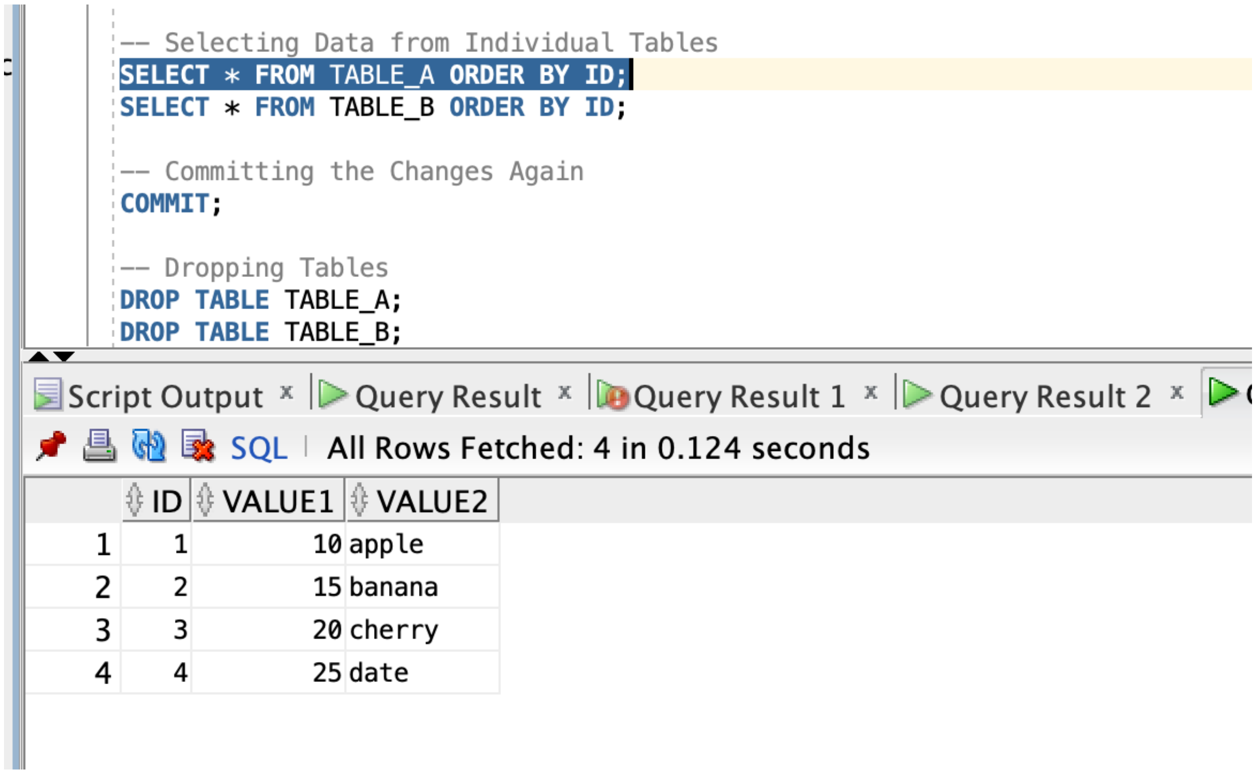
```
4 rows merged.
```



```
)
WHEN MATCHED THEN
    UPDATE SET tgt.VALUE1 = src.VALUE1, tgt.VALUE2 = src.VALUE2
        WHERE tgt.ID = src.ID
    DELETE WHERE src.ID IS NULL AND tgt.ID IS NOT NULL
WHEN NOT MATCHED THEN
    INSERT (ID, VALUE1, VALUE2)
    VALUES (src.ID, src.VALUE1, src.VALUE2);

-- Selecting Data Again After Merge
SELECT t1.ID, t1.VALUE1, t1.VALUE2, t2.ID AS X, t2.VALUE1 AS Y, t2.VALUE2 AS Z
FROM TABLE_A t1
FULL JOIN TABLE_B t2
    ON t2.ID = t1.ID
ORDER BY t1.ID, X;

-- Selecting Data from Individual Tables
SELECT * FROM TABLE_A ORDER BY ID;
SELECT * FROM TABLE_B ORDER BY ID;

-- Committing the Changes Again
COMMIT;

-- Dropping Tables
DROP TABLE TABLE_A;
DROP TABLE TABLE_B;
```

All Rows Fetched: 5 in 0.307 seconds

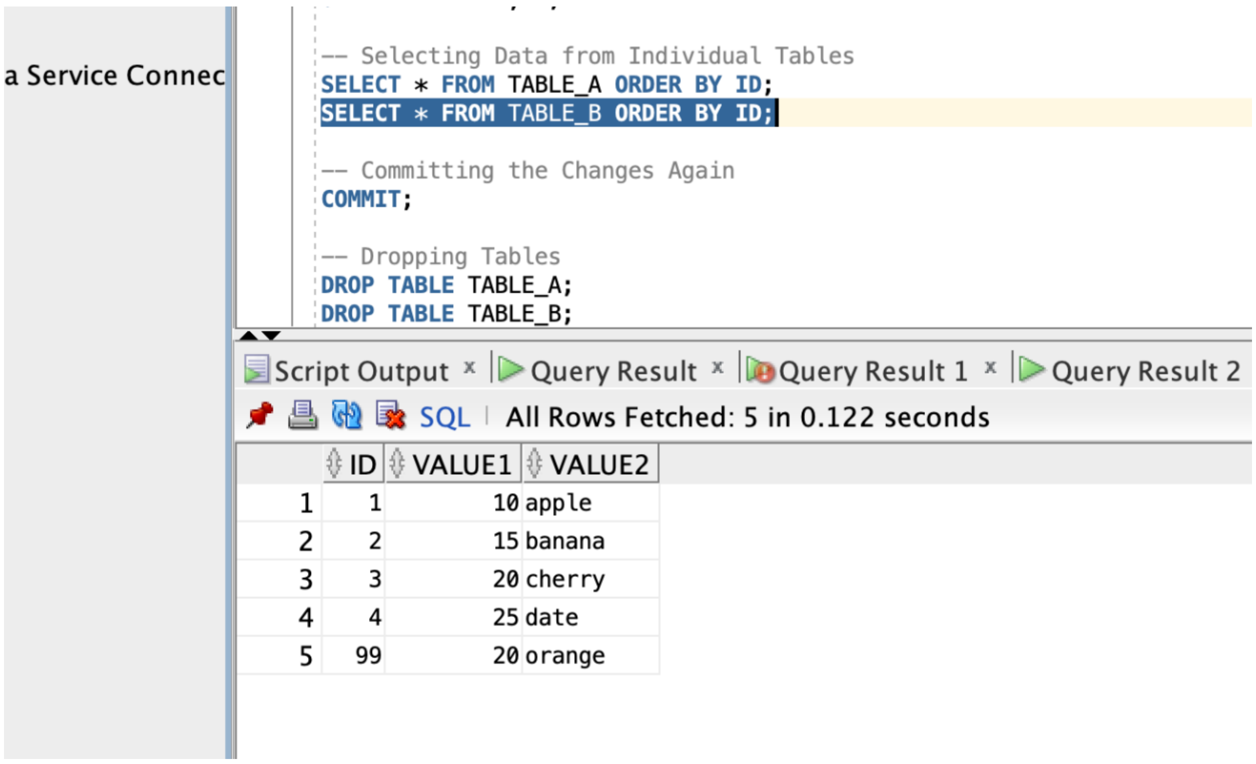
	ID	VALUE1	VALUE2	X	Y	Z
1	1	10	apple	1	10	apple
2	2	15	banana	2	15	banana
3	3	20	cherry	3	20	cherry
4	4	25	date	4	25	date
5	(null)	(null)	(null)	99	20	orange



```
-- Selecting Data from Individual Tables
SELECT * FROM TABLE_A ORDER BY ID;
SELECT * FROM TABLE_B ORDER BY ID;

-- Committing the Changes Again
COMMIT;

-- Dropping Tables
DROP TABLE TABLE_A;
DROP TABLE TABLE_B;
```

All Rows Fetched: 4 in 0.124 seconds

	ID	VALUE1	VALUE2
1	1	10	apple
2	2	15	banana
3	3	20	cherry
4	4	25	date



```
-- Selecting Data from Individual Tables
SELECT * FROM TABLE_A ORDER BY ID;
SELECT * FROM TABLE_B ORDER BY ID;

-- Committing the Changes Again
COMMIT;

-- Dropping Tables
DROP TABLE TABLE_A;
DROP TABLE TABLE_B;
```

All Rows Fetched: 5 in 0.122 seconds

	ID	VALUE1	VALUE2
1	1	10	apple
2	2	15	banana
3	3	20	cherry
4	4	25	date
5	99	20	orange

2. *What is the difference between LEAD and LAG functions? Explain different types of arguments that we can pass to these functions in detail. Using your example tables, show the usage of these functions. - 10 Points*

The analytical LEAD and LAG functions in Oracle SQL let you retrieve data from rows that are next to or before in the result set without the need for self-joins. Calculations and comparisons with data from adjacent rows are frequently carried out using these functions.

LEAD Function:
The LEAD function allows you to retrieve information from a later row in the result set. Three reasons are necessary:

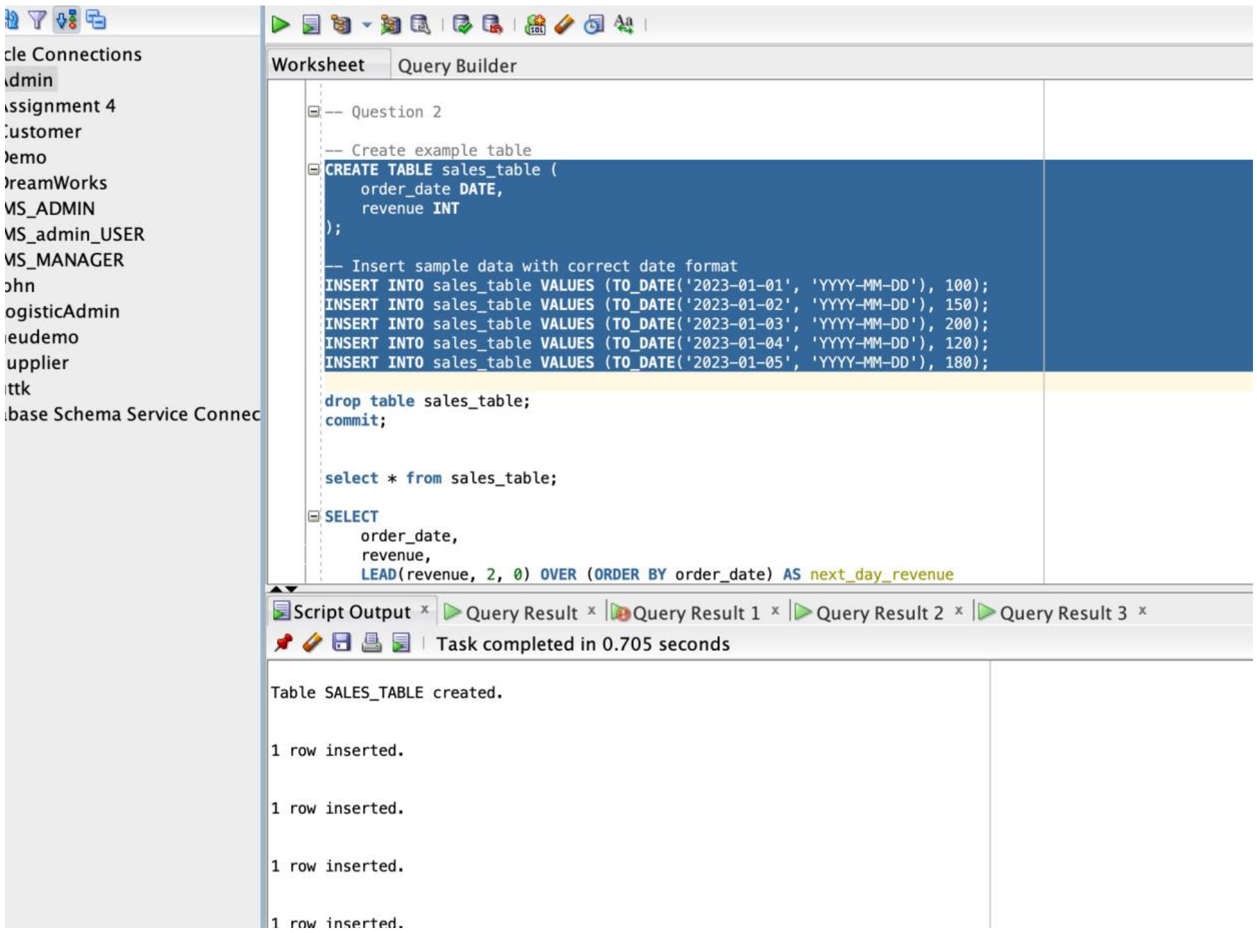
- *Expression: The expression or column whose value you wish to retrieve from the following row.*
- *Offset: The number of rows the value should be fetched from after the current row. One is the default.*
- *If the offset exceeds the end of the result set, the default value will be returned. Null is the default.*

LAG function:

To retrieve data from a previous row in the result set, use the LAG function. Additionally, three arguments are required:

- *Expression: The column or expression from the preceding row whose value you wish to retrieve.*
- *Offset: The number of rows that need to be fetched to go backward from the current row.*
- *Default: The value to return if the offset exceeds the start of the result set. The default value is 1. NULL is the default.*

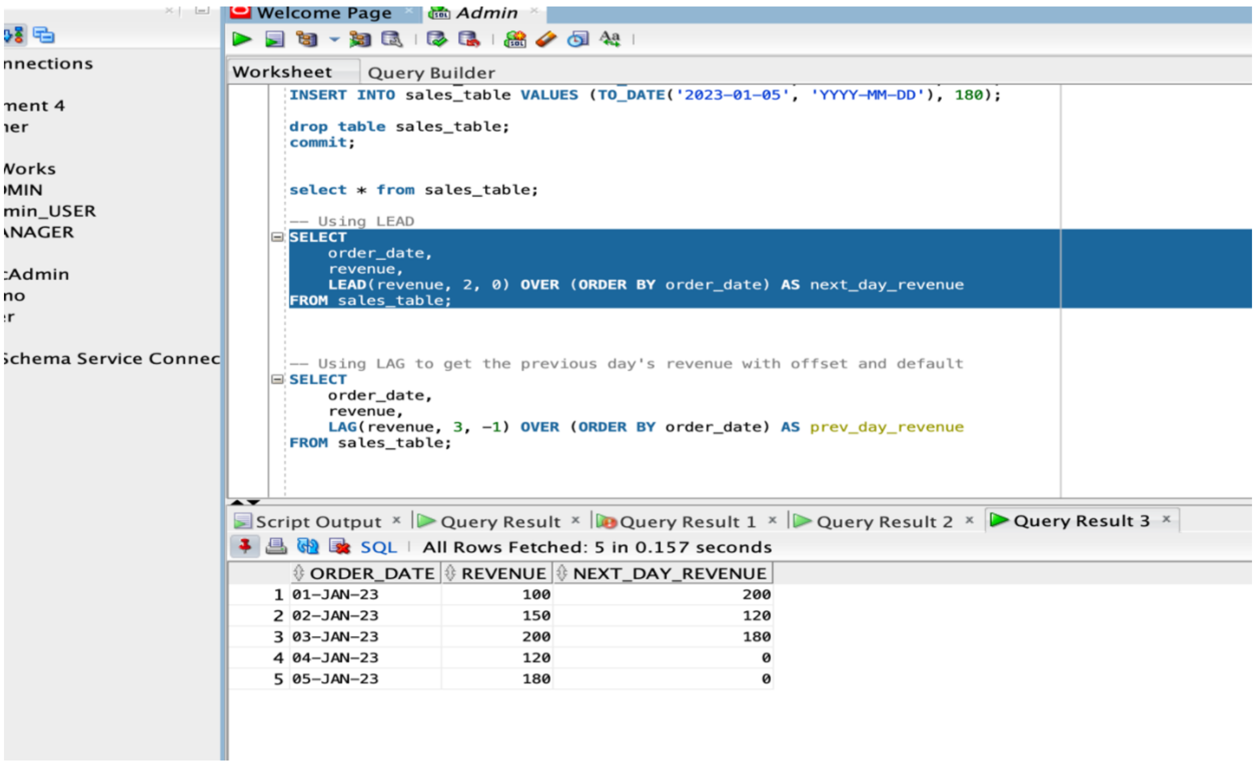
Creating and inserting records into Sale_table for example of lead () and lag ():



```
-- Question 2

-- Create example table
CREATE TABLE sales_table (
    order_date DATE,
    revenue INT
);

-- Insert sample data with correct date format
INSERT INTO sales_table VALUES (TO_DATE('2023-01-01', 'YYYY-MM-DD'), 100);
INSERT INTO sales_table VALUES (TO_DATE('2023-01-02', 'YYYY-MM-DD'), 150);
INSERT INTO sales_table VALUES (TO_DATE('2023-01-03', 'YYYY-MM-DD'), 200);
INSERT INTO sales_table VALUES (TO_DATE('2023-01-04', 'YYYY-MM-DD'), 120);
INSERT INTO sales_table VALUES (TO_DATE('2023-01-05', 'YYYY-MM-DD'), 180);

drop table sales_table;
commit;


select * from sales_table;

SELECT
    order_date,
    revenue,
    LEAD(revenue, 2, 0) OVER (ORDER BY order_date) AS next_day_revenue
```

```
Table SALES_TABLE created.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.
```

Using Lead ():

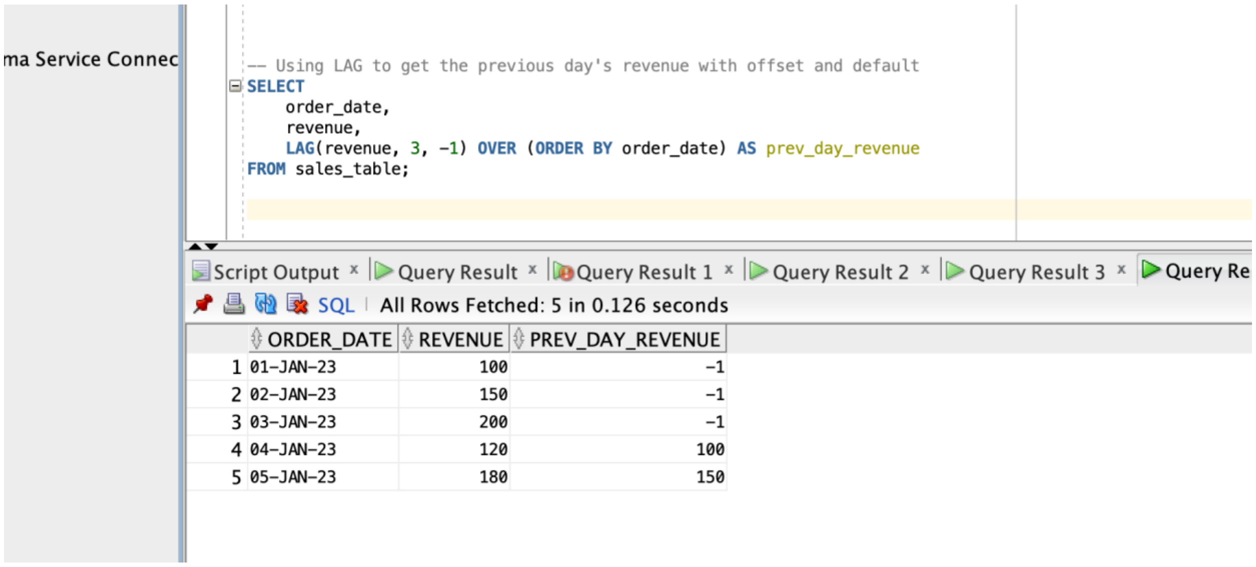


```
INSERT INTO sales_table VALUES (TO_DATE('2023-01-05', 'YYYY-MM-DD'), 180);

drop table sales_table;
commit;


select * from sales_table;

-- Using LEAD
SELECT
    order_date,
    revenue,
    LEAD(revenue, 2, 0) OVER (ORDER BY order_date) AS next_day_revenue
FROM sales_table;


-- Using LAG to get the previous day's revenue with offset and default
SELECT
    order_date,
    revenue,
    LAG(revenue, 3, -1) OVER (ORDER BY order_date) AS prev_day_revenue
FROM sales_table;
```

All Rows Fetched: 5 in 0.157 seconds

	ORDER_DATE	REVENUE	NEXT_DAY_REVENUE
1	01-JAN-23	100	200
2	02-JAN-23	150	120
3	03-JAN-23	200	180
4	04-JAN-23	120	0
5	05-JAN-23	180	0

Using Lag ():



```
-- Using LAG to get the previous day's revenue with offset and default
SELECT
    order_date,
    revenue,
    LAG(revenue, 3, -1) OVER (ORDER BY order_date) AS prev_day_revenue
FROM sales_table;
```

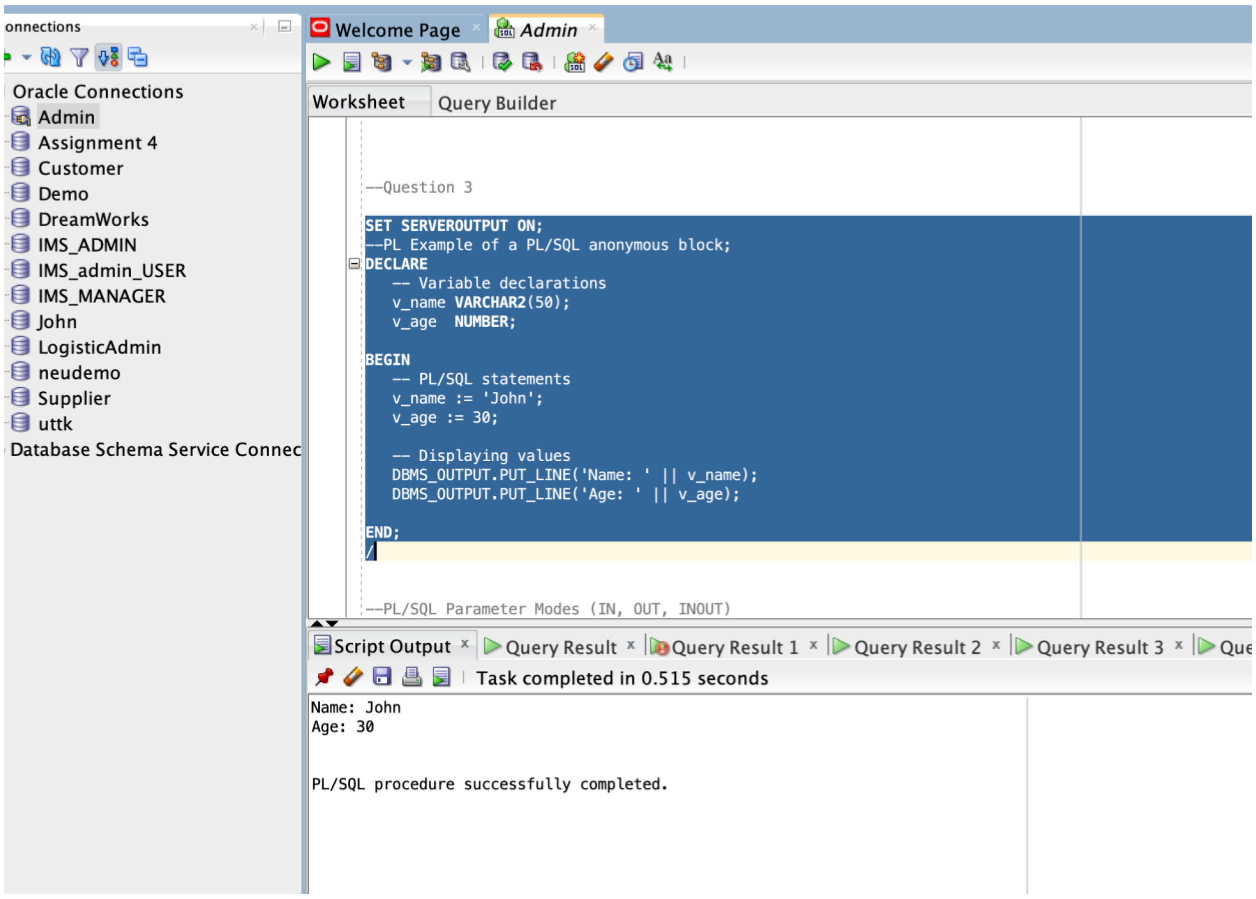
All Rows Fetched: 5 in 0.126 seconds

	ORDER_DATE	REVENUE	PREV_DAY_REVENUE
1	01-JAN-23	100	-1
2	02-JAN-23	150	-1
3	03-JAN-23	200	-1
4	04-JAN-23	120	100
5	05-JAN-23	180	150

3. *Explain the below terms with an example–*

   - *PL/SQL Anonymous Block - 8 points*

A procedural extension of SQL created specifically for Oracle databases is called PL/SQL (Procedural Language/Structured Query Language). A collection of PL/SQL statements that are run collectively is known as an Anonymous Block in the language. Anonymous blocks are defined inline within the code and are unnamed, unlike stored *procedures or functions. They serve several functions, including exception handling, transaction control, and data manipulation.*



```
--Question 3

SET SERVEROUTPUT ON;
--PL Example of a PL/SQL anonymous block;
DECLARE
   -- Variable declarations
   v_name VARCHAR2(50);
   v_age  NUMBER;

BEGIN
   -- PL/SQL statements
   v_name := 'John';
   v_age := 30;

   -- Displaying values
   DBMS_OUTPUT.PUT_LINE('Name: ' || v_name);
   DBMS_OUTPUT.PUT_LINE('Age: ' || v_age);

END;
/

--PL/SQL Parameter Modes (IN, OUT, INOUT)
```

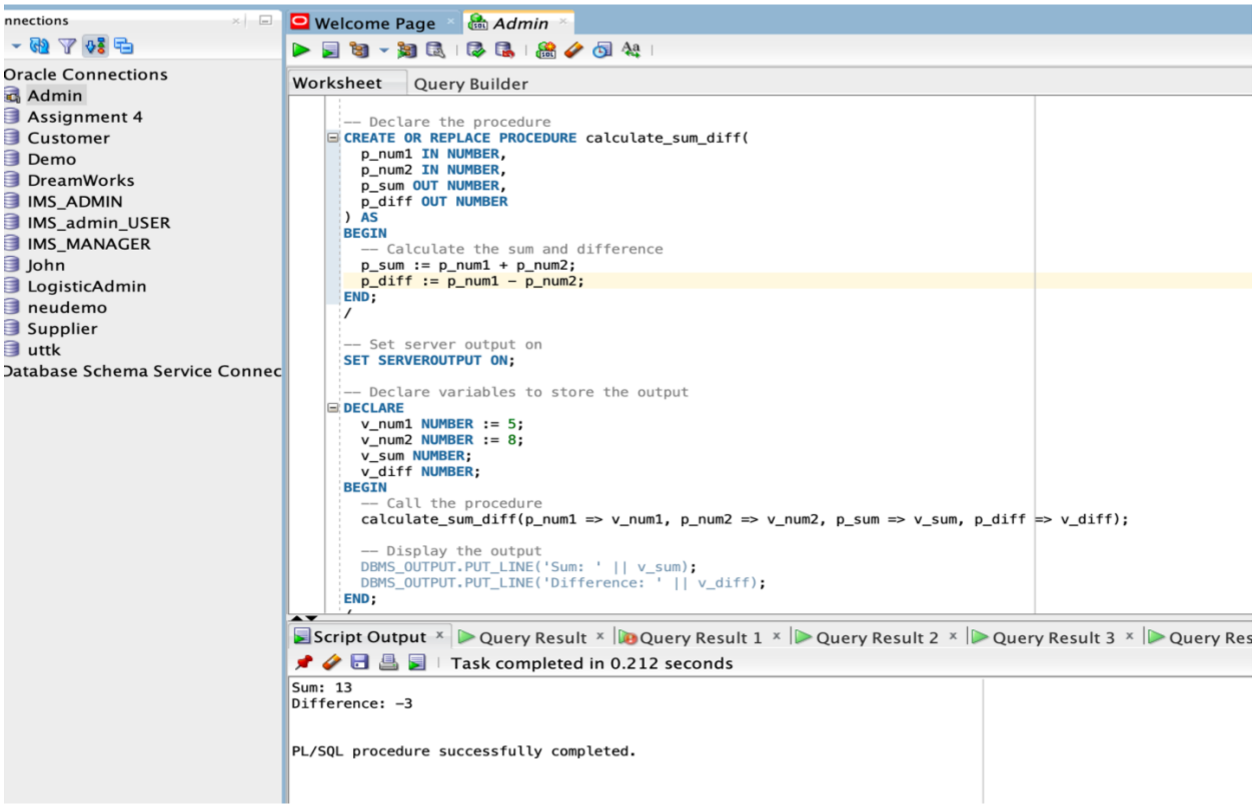
Task completed in 0.515 seconds

```
Name: John
Age: 30


PL/SQL procedure successfully completed.
```

- *Pl/Sql parameter modes (IN, OUT, INOUT) - 8 points*

To specify how a parameter is passed, you can use parameter modes in PL/SQL when defining a procedure or function. Three modes of parameters exist:

- *IN: This is the standard mode. It means that although the parameter can be used in the function or procedure, its value cannot be changed.*
- *OUT: The modified value of a parameter that can be changed within a procedure or function is returned to the caller when this mode is utilized.*
- *INOUT: It integrates the functionalities of the IN and OUT modes. The caller receives the modified value of the parameter, which they can utilize within the procedure or function.*



```
-- Declare the procedure
CREATE OR REPLACE PROCEDURE calculate_sum_diff(
  p_num1 IN NUMBER,
  p_num2 IN NUMBER,
  p_sum OUT NUMBER,
  p_diff OUT NUMBER
) AS
BEGIN
  -- Calculate the sum and difference
  p_sum := p_num1 + p_num2;
  p_diff := p_num1 - p_num2;
END;
/

-- Set server output on
SET SERVEROUTPUT ON;

-- Declare variables to store the output
DECLARE
  v_num1 NUMBER := 5;
  v_num2 NUMBER := 8;
  v_sum NUMBER;
  v_diff NUMBER;
BEGIN
  -- Call the procedure
  calculate_sum_diff(p_num1 => v_num1, p_num2 => v_num2, p_sum => v_sum, p_diff => v_diff);

  -- Display the output
  DBMS_OUTPUT.PUT_LINE('Sum: ' || v_sum);
  DBMS_OUTPUT.PUT_LINE('Difference: ' || v_diff);
END;
```

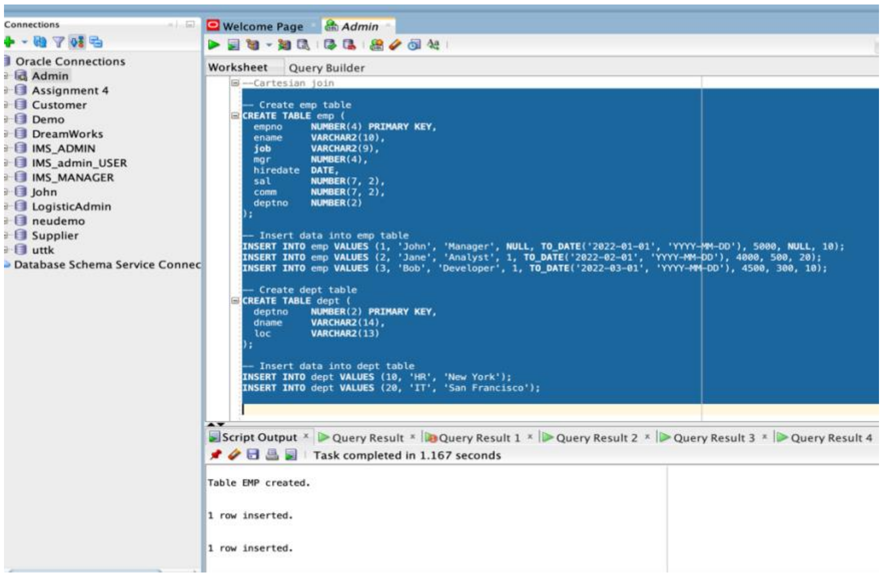
Task completed in 0.212 seconds

```
Sum: 13
Difference: -3


PL/SQL procedure successfully completed.
```

- *Cartesian join - 4 points*

Every row from the first table is combined with every row from the second table in a Cartesian Join (also known as a Cross Join) sort of join operation in a relational database. The two tables are then produced as a Cartesian product.

Creating tables for cartesian join:

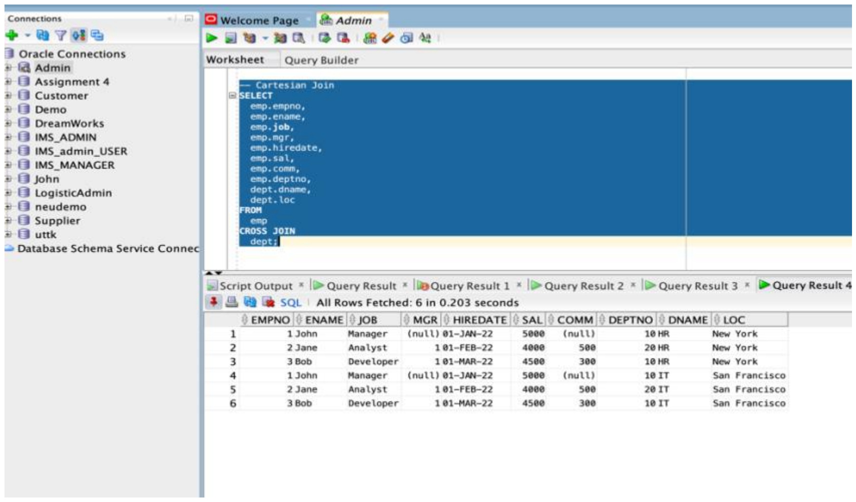


```
--Cartesian join

-- Create emp table
CREATE TABLE emp (
  empno      NUMBER(4) PRIMARY KEY,
  ename      VARCHAR2(10),
  job        VARCHAR2(9),
  mgr        NUMBER(4),
  hiredate   DATE,
  sal        NUMBER(7, 2),
  comm       NUMBER(7, 2),
  deptno     NUMBER(2)
);

-- Insert data into emp table
INSERT INTO emp VALUES (1, 'John', 'Manager', NULL, TO_DATE('2022-01-01', 'YYYY-MM-DD'), 5000, NULL, 10);
INSERT INTO emp VALUES (2, 'Jane', 'Analyst', 1, TO_DATE('2022-02-01', 'YYYY-MM-DD'), 4000, 500, 20);
INSERT INTO emp VALUES (3, 'Bob', 'Developer', 1, TO_DATE('2022-03-01', 'YYYY-MM-DD'), 4500, 300, 10);

-- Create dept table
CREATE TABLE dept (
  deptno     NUMBER(2) PRIMARY KEY,
  dname      VARCHAR2(14),
  loc        VARCHAR2(13)
);

-- Insert data into dept table
INSERT INTO dept VALUES (10, 'HR', 'New York');
INSERT INTO dept VALUES (20, 'IT', 'San Francisco');
```

Task completed in 1.167 seconds

```
Table EMP created.


1 row inserted.


1 row inserted.
```

Cross Join / Cartesian Join:



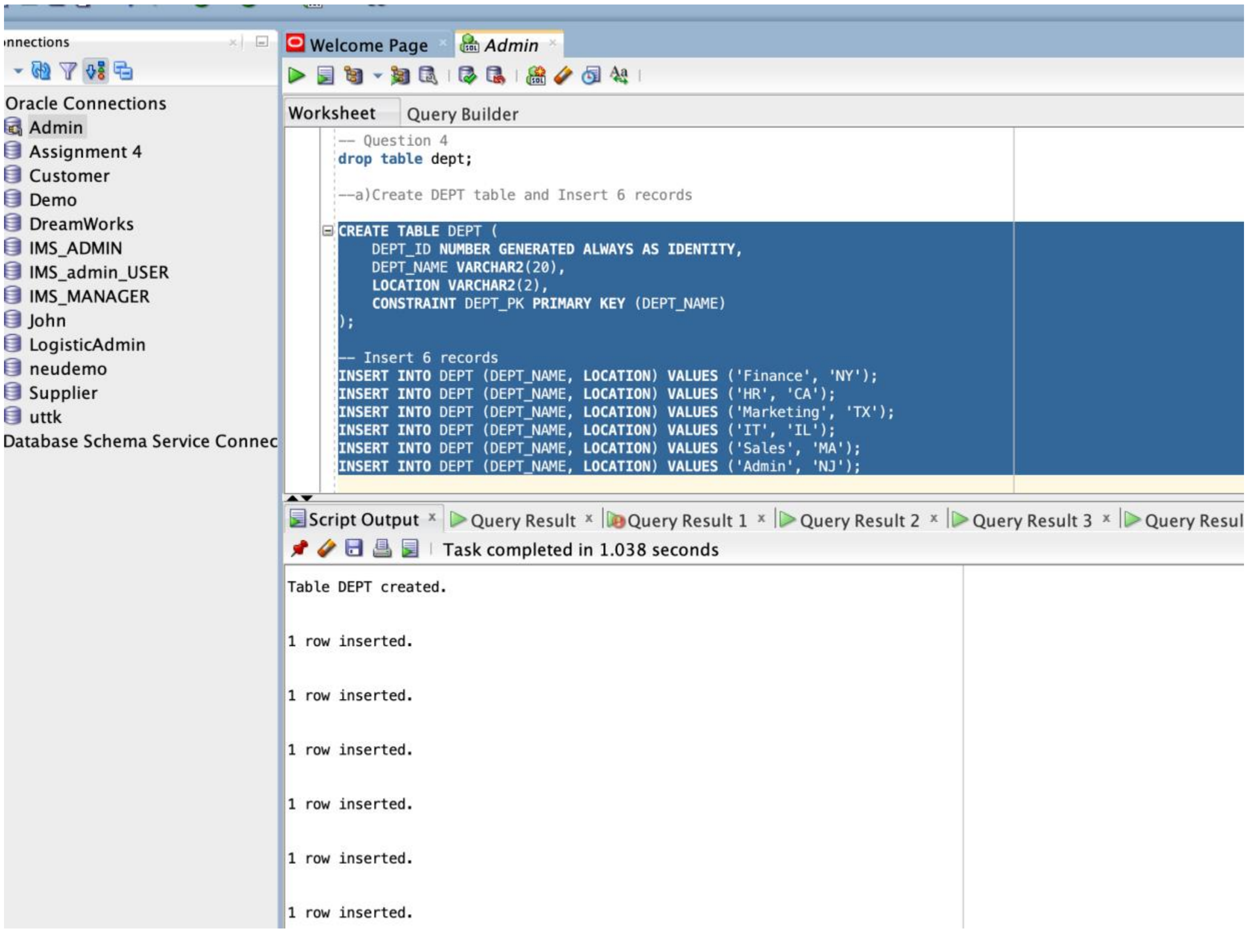
```
-- Cartesian Join
SELECT
  emp.empno,
  emp.ename,
  emp.job,
  emp.mgr,
  emp.hiredate,
  emp.sal,
  emp.comm,
  emp.deptno,
  dept.dname,
  dept.loc
FROM
  emp
CROSS JOIN
  dept;
```

All Rows Fetched: 6 in 0.203 seconds

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	DNAME	LOC
1	1	John	Manager	(null)	01-JAN-22	5000	(null)	10	HR	New York
2	2	Jane	Analyst	1	01-FEB-22	4000	500	20	HR	New York
3	3	Bob	Developer	1	01-MAR-22	4500	300	10	HR	New York
4	1	John	Manager	(null)	01-JAN-22	5000	(null)	10	IT	San Francisco
5	2	Jane	Analyst	1	01-FEB-22	4000	500	20	IT	San Francisco
6	3	Bob	Developer	1	01-MAR-22	4500	300	10	IT	San Francisco

4. *Create procedure with appropriate arguments to perform updates and inserts on department table (Department name will be unique). Make sure to upload script execution test cases for all the combinations to prove the validation is successfully working and upload screenshots for each question proving the test cases.*

   A. *CREATE DEPT TABLE AND INSERT 6 RECORDS (refer to Instructions for Schema) - 6 points.*

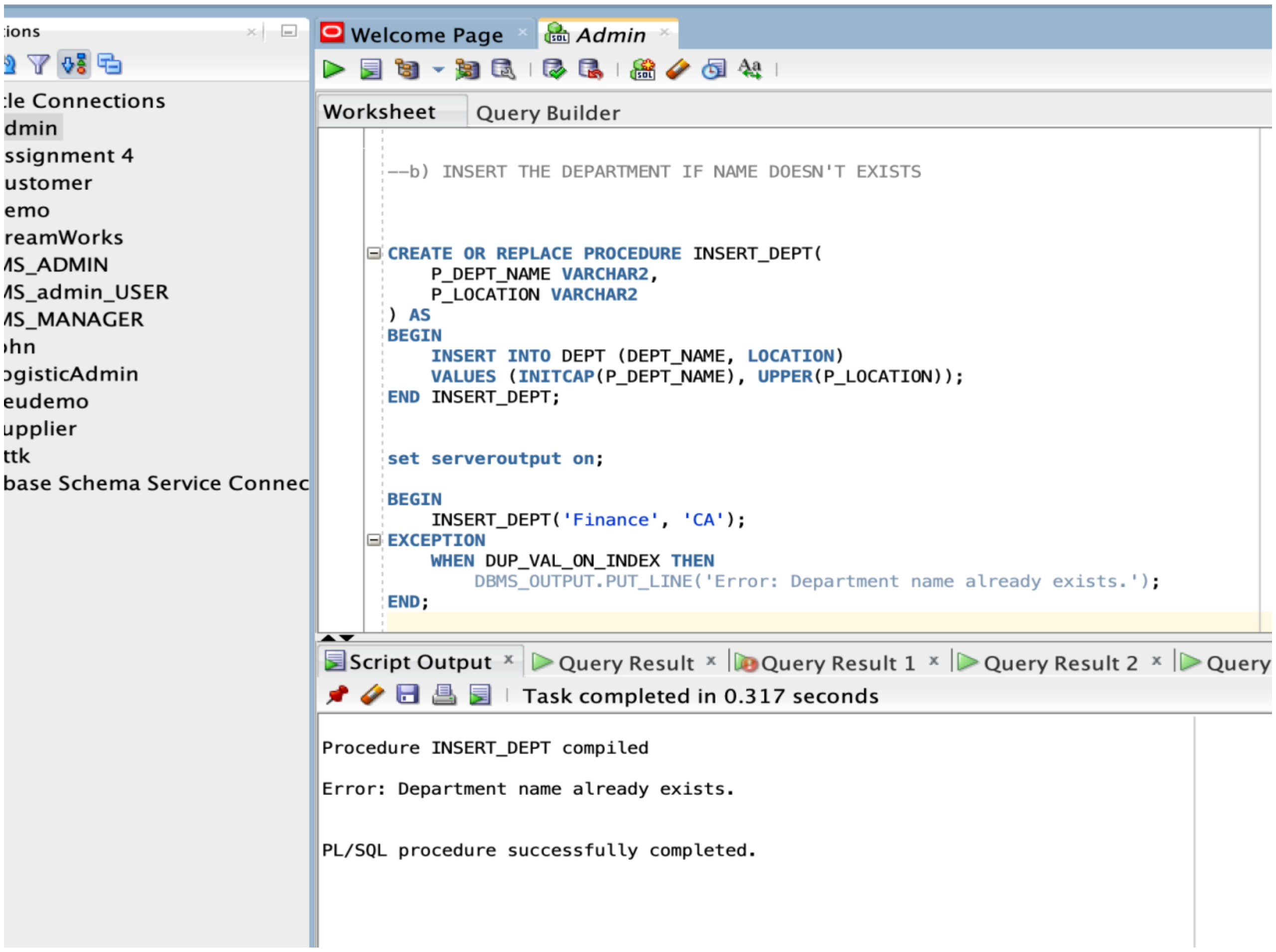


```
-- Question 4
drop table dept;

--a)Create DEPT table and Insert 6 records

CREATE TABLE DEPT (
    DEPT_ID NUMBER GENERATED ALWAYS AS IDENTITY,
    DEPT_NAME VARCHAR2(20),
    LOCATION VARCHAR2(2),
    CONSTRAINT DEPT_PK PRIMARY KEY (DEPT_NAME)
);

-- Insert 6 records
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('Finance', 'NY');
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('HR', 'CA');
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('Marketing', 'TX');
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('IT', 'IL');
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('Sales', 'MA');
INSERT INTO DEPT (DEPT_NAME, LOCATION) VALUES ('Admin', 'NJ');
```

Task completed in 1.038 seconds

```
Table DEPT created.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.


1 row inserted.
```

B. INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS - 6 points.

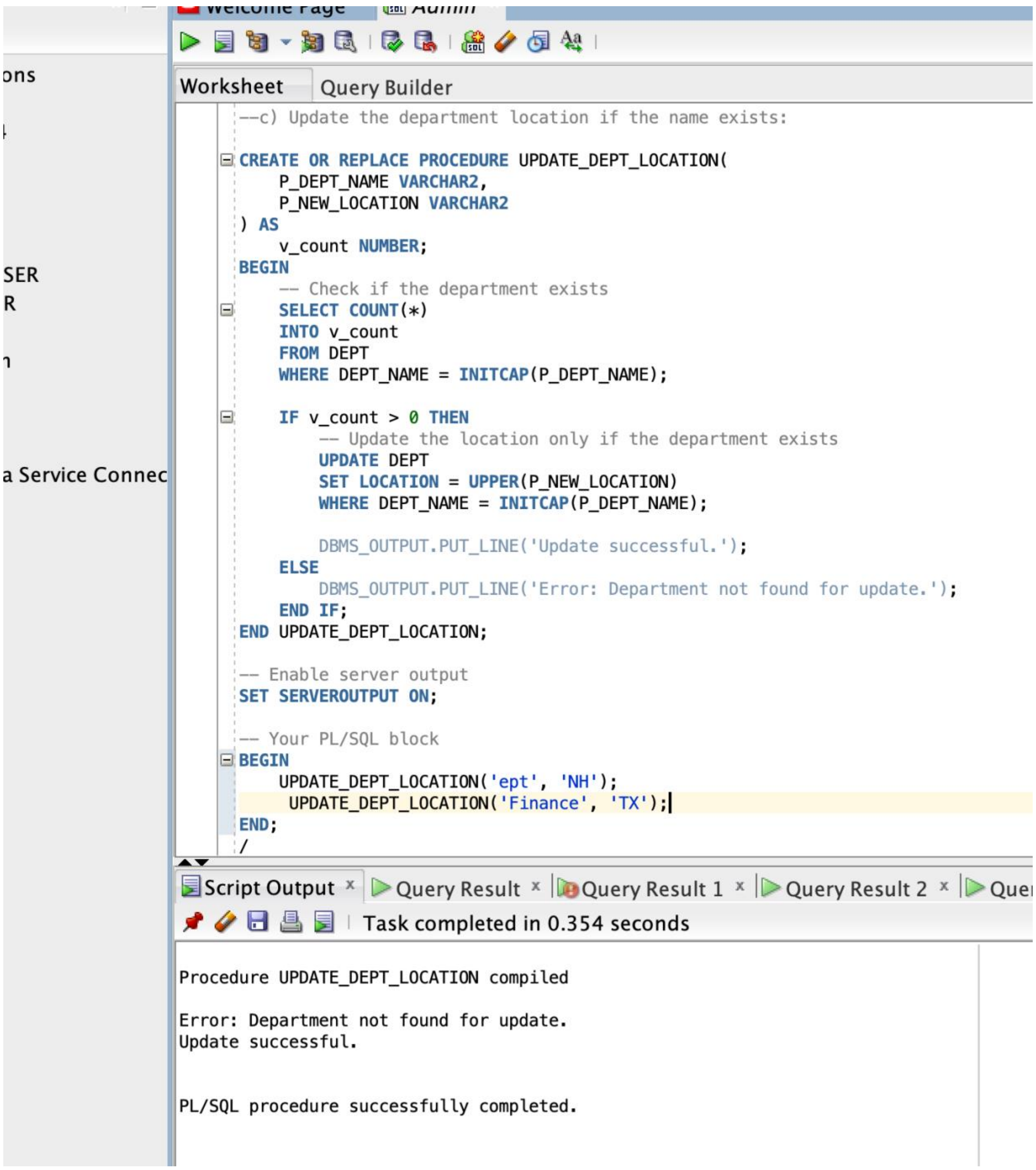


```
--b) INSERT THE DEPARTMENT IF NAME DOESN'T EXISTS


CREATE OR REPLACE PROCEDURE INSERT_DEPT(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
BEGIN
    INSERT INTO DEPT (DEPT_NAME, LOCATION)
    VALUES (INITCAP(P_DEPT_NAME), UPPER(P_LOCATION));
END INSERT_DEPT;


set serveroutput on;

BEGIN
    INSERT_DEPT('Finance', 'CA');
EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
        DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.');
END;
```

Task completed in 0.317 seconds

```
Procedure INSERT_DEPT compiled

Error: Department name already exists.


PL/SQL procedure successfully completed.
```

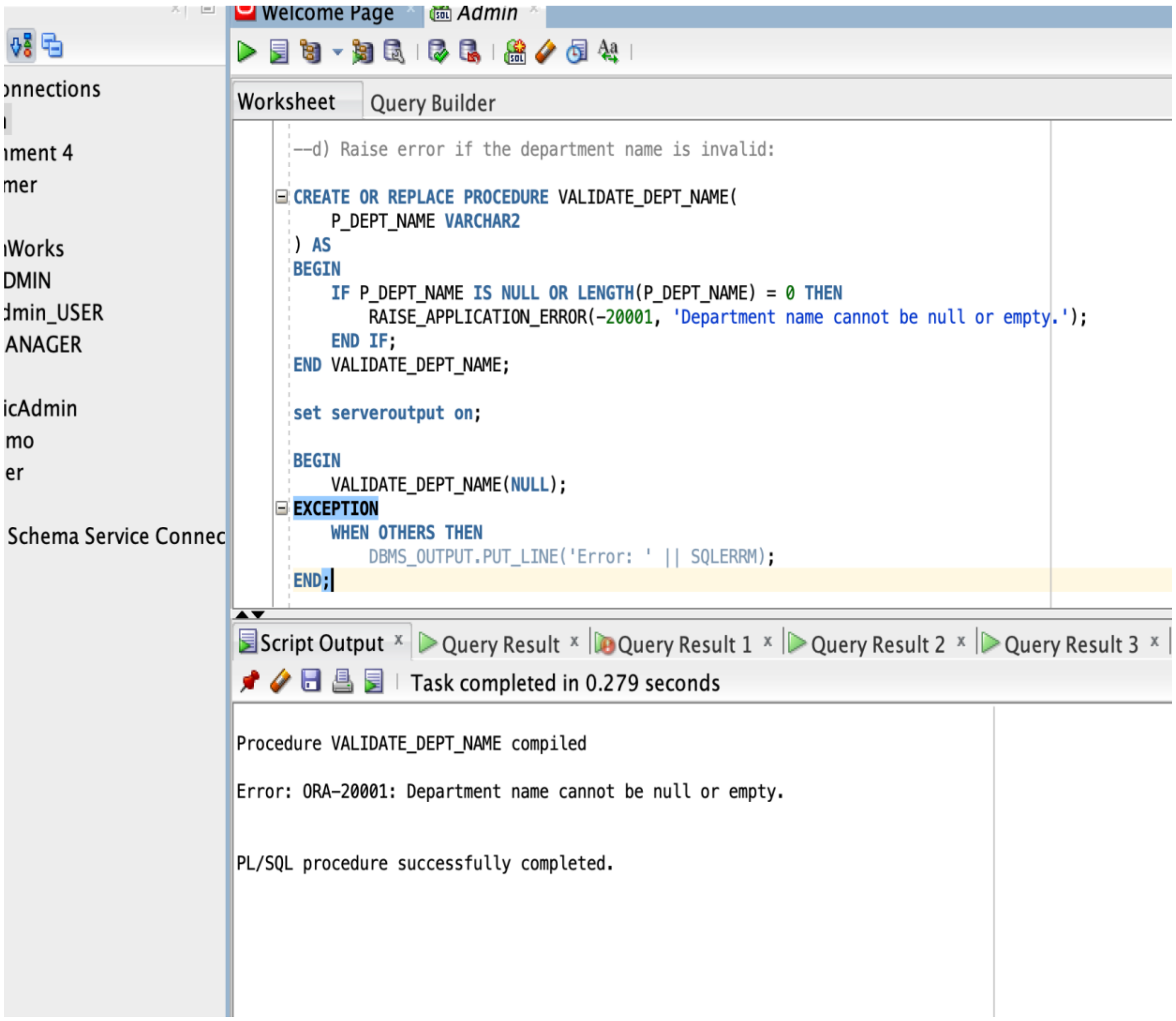
C. UPDATE THE DEPARTMENT LOCATION IF NAME EXISTS - 6 points.



```
--c) Update the department location if the name exists:

CREATE OR REPLACE PROCEDURE UPDATE_DEPT_LOCATION(
    P_DEPT_NAME VARCHAR2,
    P_NEW_LOCATION VARCHAR2
) AS
    v_count NUMBER;
BEGIN
    -- Check if the department exists
    SELECT COUNT(*)
    INTO v_count
    FROM DEPT
    WHERE DEPT_NAME = INITCAP(P_DEPT_NAME);

    IF v_count > 0 THEN
        -- Update the location only if the department exists
        UPDATE DEPT
        SET LOCATION = UPPER(P_NEW_LOCATION)
        WHERE DEPT_NAME = INITCAP(P_DEPT_NAME);

        DBMS_OUTPUT.PUT_LINE('Update successful.');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Error: Department not found for update.');
    END IF;
END UPDATE_DEPT_LOCATION;

-- Enable server output
SET SERVEROUTPUT ON;

-- Your PL/SQL block
BEGIN
    UPDATE_DEPT_LOCATION('ept', 'NH');
     UPDATE_DEPT_LOCATION('Finance', 'TX');
END;
/
```

```
Procedure UPDATE_DEPT_LOCATION compiled

Error: Department not found for update.
Update successful.


PL/SQL procedure successfully completed.
```

D. *RAISE ERROR IF THE DEPARTMENT NAME IS INVALID (NULL, ZERO LENGTH) - 6 points.*

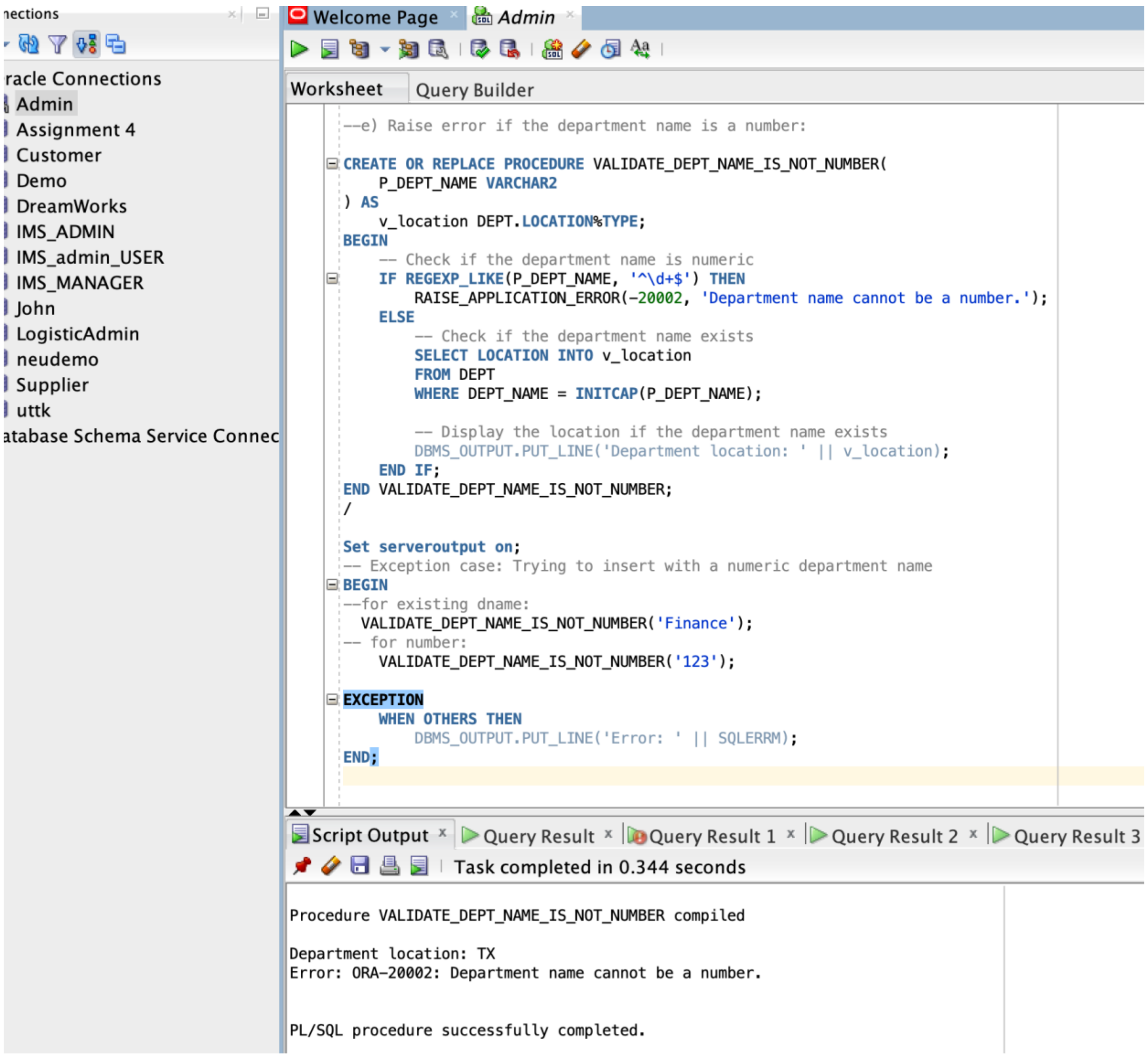


```
--d) Raise error if the department name is invalid:

CREATE OR REPLACE PROCEDURE VALIDATE_DEPT_NAME(
    P_DEPT_NAME VARCHAR2
) AS
BEGIN
    IF P_DEPT_NAME IS NULL OR LENGTH(P_DEPT_NAME) = 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Department name cannot be null or empty.');
    END IF;
END VALIDATE_DEPT_NAME;

set serveroutput on;

BEGIN
    VALIDATE_DEPT_NAME(NULL);
EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Task completed in 0.279 seconds

```
Procedure VALIDATE_DEPT_NAME compiled

Error: ORA-20001: Department name cannot be null or empty.


PL/SQL procedure successfully completed.
```

E. *RAISE ERROR IF THE DEPARTMENT NAME IS A NUMBER -6 points.*

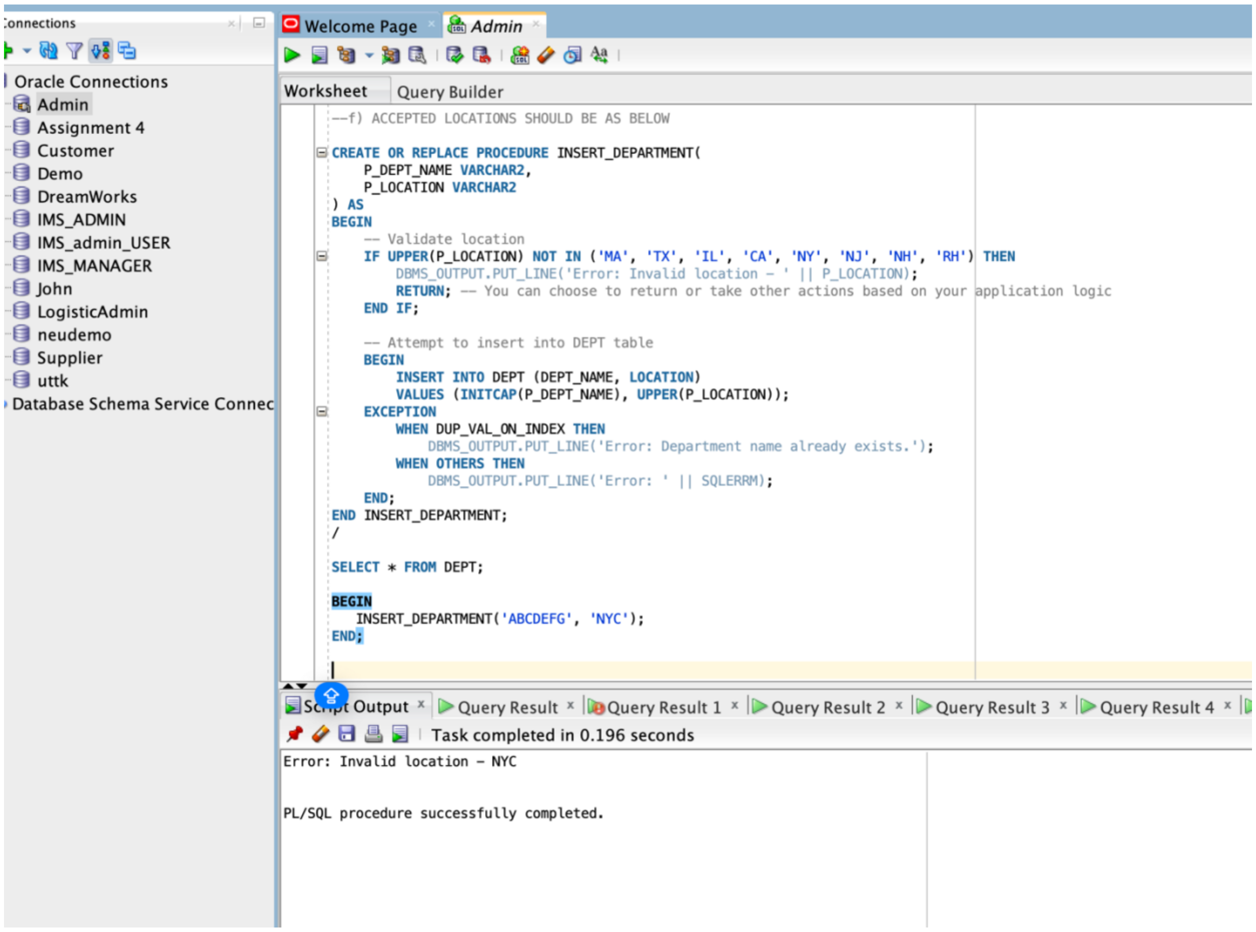


```
--e) Raise error if the department name is a number:

CREATE OR REPLACE PROCEDURE VALIDATE_DEPT_NAME_IS_NOT_NUMBER(
    P_DEPT_NAME VARCHAR2
) AS
    v_location DEPT.LOCATION%TYPE;
BEGIN
    -- Check if the department name is numeric
    IF REGEXP_LIKE(P_DEPT_NAME, '^\d+$') THEN
        RAISE_APPLICATION_ERROR(-20002, 'Department name cannot be a number.');
    ELSE
        -- Check if the department name exists
        SELECT LOCATION INTO v_location
        FROM DEPT
        WHERE DEPT_NAME = INITCAP(P_DEPT_NAME);

        -- Display the location if the department name exists
        DBMS_OUTPUT.PUT_LINE('Department location: ' || v_location);
    END IF;
END VALIDATE_DEPT_NAME_IS_NOT_NUMBER;
/

Set serveroutput on;
-- Exception case: Trying to insert with a numeric department name
BEGIN
--for existing dname:
  VALIDATE_DEPT_NAME_IS_NOT_NUMBER('Finance');
-- for number:
    VALIDATE_DEPT_NAME_IS_NOT_NUMBER('123');

EXCEPTION
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
END;
```

Task completed in 0.344 seconds

```
Procedure VALIDATE_DEPT_NAME_IS_NOT_NUMBER compiled

Department location: TX
Error: ORA-20002: Department name cannot be a number.


PL/SQL procedure successfully completed.
```

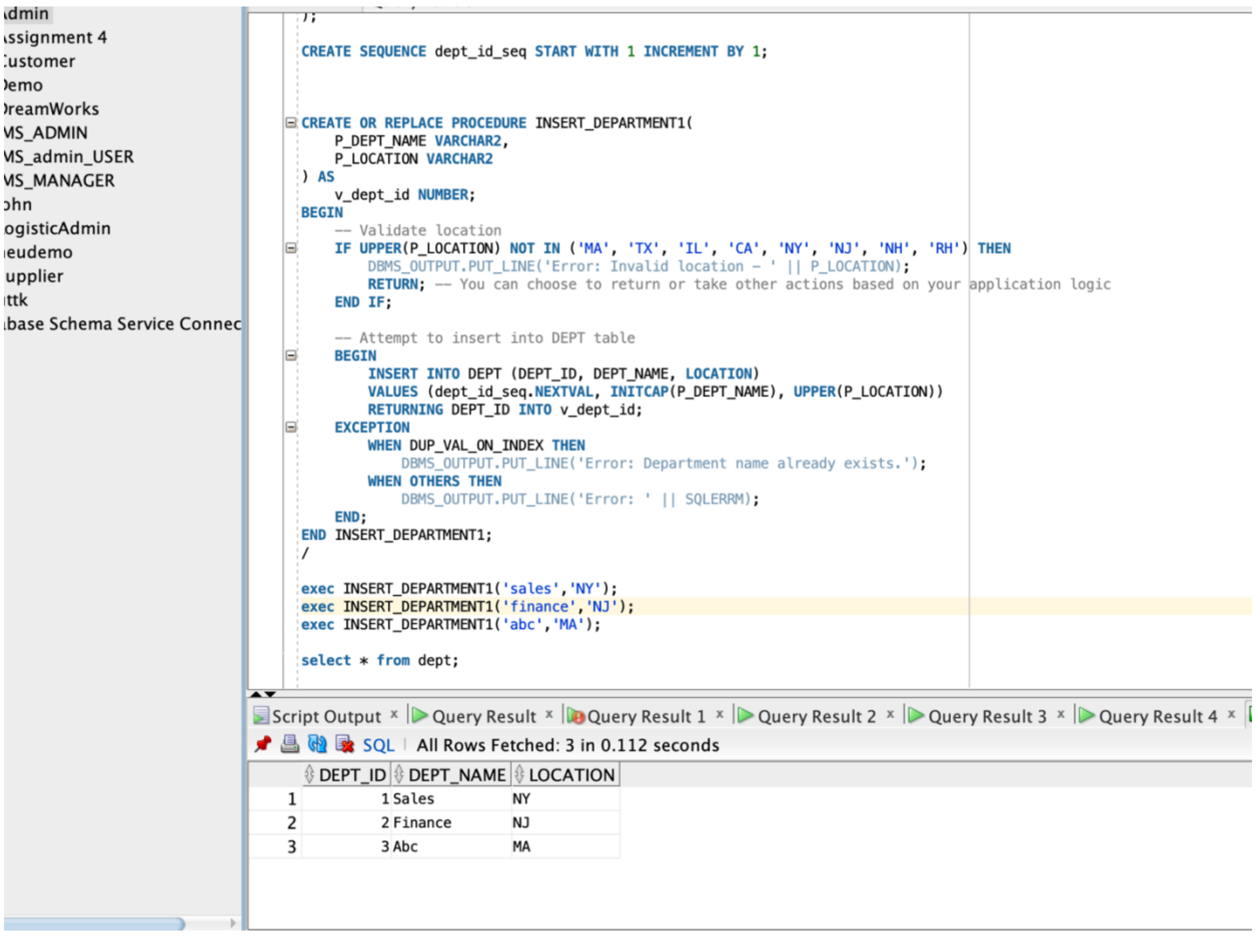
F. ACCEPTED LOCATIONS SHOULD BE AS BELOW - 6 points.
MA, TX, IL, CA, NY, NJ, NH, RH



```
--f) ACCEPTED LOCATIONS SHOULD BE AS BELOW

CREATE OR REPLACE PROCEDURE INSERT_DEPARTMENT(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
BEGIN
    -- Validate location
    IF UPPER(P_LOCATION) NOT IN ('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH') THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid location - ' || P_LOCATION);
        RETURN; -- You can choose to return or take other actions based on your application logic
    END IF;

    -- Attempt to insert into DEPT table
    BEGIN
        INSERT INTO DEPT (DEPT_NAME, LOCATION)
        VALUES (INITCAP(P_DEPT_NAME), UPPER(P_LOCATION));
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END INSERT_DEPARTMENT;
/

SELECT * FROM DEPT;

BEGIN
   INSERT_DEPARTMENT('ABCDEFG', 'NYC');
END;
```

```
Error: Invalid location - NYC


PL/SQL procedure successfully completed.
```

G. DEPARTMENT ID SHOULD BE AUTO-GENERATED - 6 points.



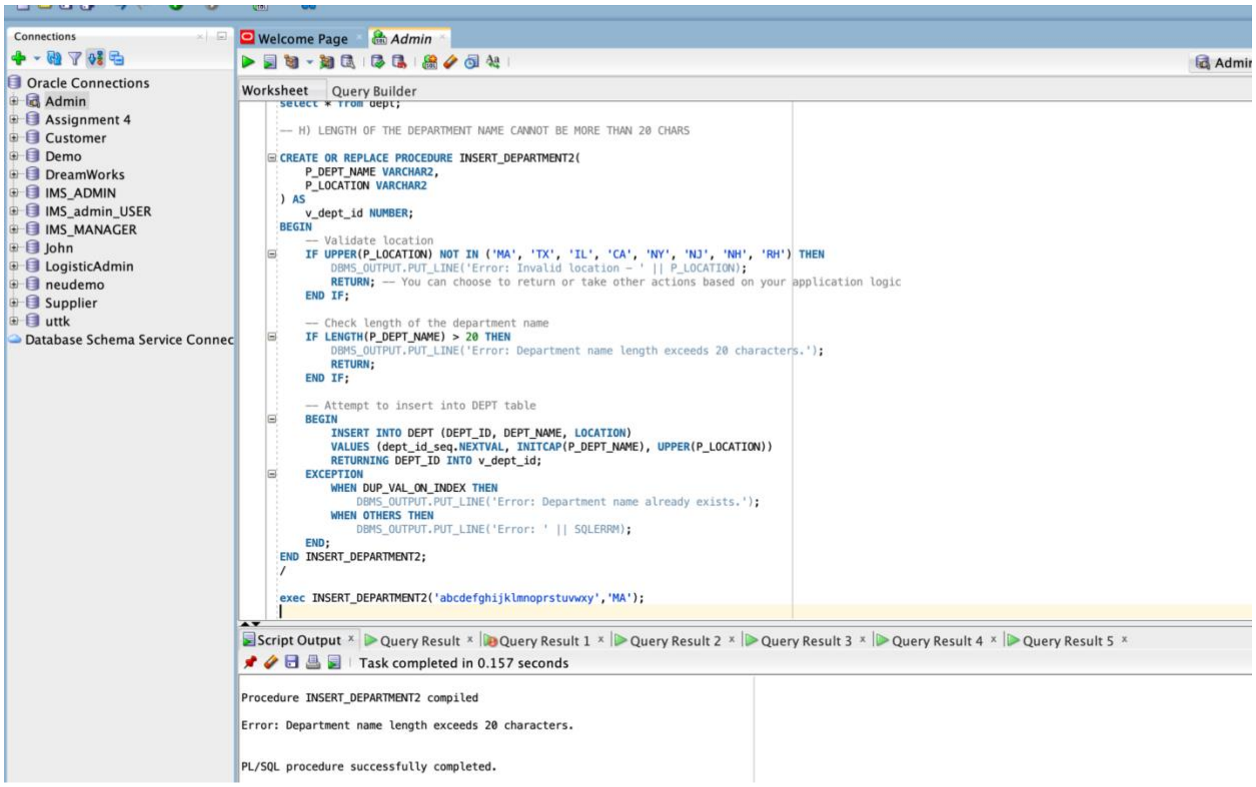
```
dmin
ssignment 4
ustomer
emo
reamWorks
MS_ADMIN
MS_admin_USER
MS_MANAGER
hn
ogisticAdmin
eudemo
upplier
ttk
base Schema Service Connec
```

```
);

CREATE SEQUENCE dept_id_seq START WITH 1 INCREMENT BY 1;


CREATE OR REPLACE PROCEDURE INSERT_DEPARTMENT1(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
    v_dept_id NUMBER;
BEGIN
    -- Validate location
    IF UPPER(P_LOCATION) NOT IN ('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH') THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid location - ' || P_LOCATION);
        RETURN; -- You can choose to return or take other actions based on your application logic
    END IF;

    -- Attempt to insert into DEPT table
    BEGIN
        INSERT INTO DEPT (DEPT_ID, DEPT_NAME, LOCATION)
        VALUES (dept_id_seq.NEXTVAL, INITCAP(P_DEPT_NAME), UPPER(P_LOCATION))
        RETURNING DEPT_ID INTO v_dept_id;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END INSERT_DEPARTMENT1;
/

exec INSERT_DEPARTMENT1('sales','NY');
exec INSERT_DEPARTMENT1('finance','NJ');
exec INSERT_DEPARTMENT1('abc','MA');

select * from dept;
```

All Rows Fetched: 3 in 0.112 seconds

	DEPT_ID	DEPT_NAME	LOCATION
1	1	Sales	NY
2	2	Finance	NJ
3	3	Abc	MA

H. LENGTH OF THE DEPARTMENT NAME CANNOT BE MORE THAN 20 CHARS - 6 points.

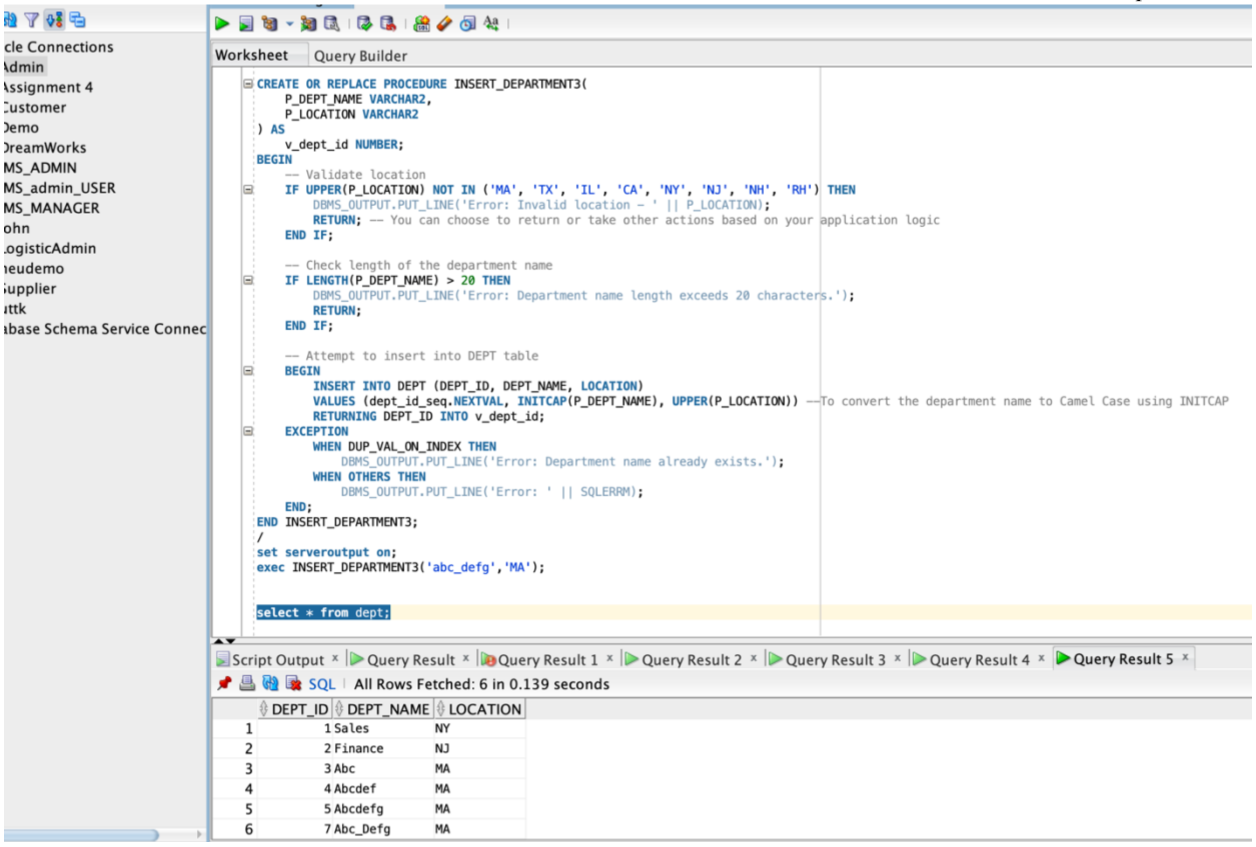


```
select * from dept;

-- H) LENGTH OF THE DEPARTMENT NAME CANNOT BE MORE THAN 20 CHARS

CREATE OR REPLACE PROCEDURE INSERT_DEPARTMENT2(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
    v_dept_id NUMBER;
BEGIN
    -- Validate location
    IF UPPER(P_LOCATION) NOT IN ('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH') THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid location - ' || P_LOCATION);
        RETURN; -- You can choose to return or take other actions based on your application logic
    END IF;

    -- Check length of the department name
    IF LENGTH(P_DEPT_NAME) > 20 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Department name length exceeds 20 characters.');
        RETURN;
    END IF;

    -- Attempt to insert into DEPT table
    BEGIN
        INSERT INTO DEPT (DEPT_ID, DEPT_NAME, LOCATION)
        VALUES (dept_id_seq.NEXTVAL, INITCAP(P_DEPT_NAME), UPPER(P_LOCATION))
        RETURNING DEPT_ID INTO v_dept_id;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END INSERT_DEPARTMENT2;
/

exec INSERT_DEPARTMENT2('abcdefghijklmnoprstuvwxy','MA');
```

Task completed in 0.157 seconds

```
Procedure INSERT_DEPARTMENT2 compiled

Error: Department name length exceeds 20 characters.


PL/SQL procedure successfully completed.
```

I. WHILE INSERTING THE DEPARTMENT NAME CONVERT EVERYTHING TO CAMEL CASE - 6 points

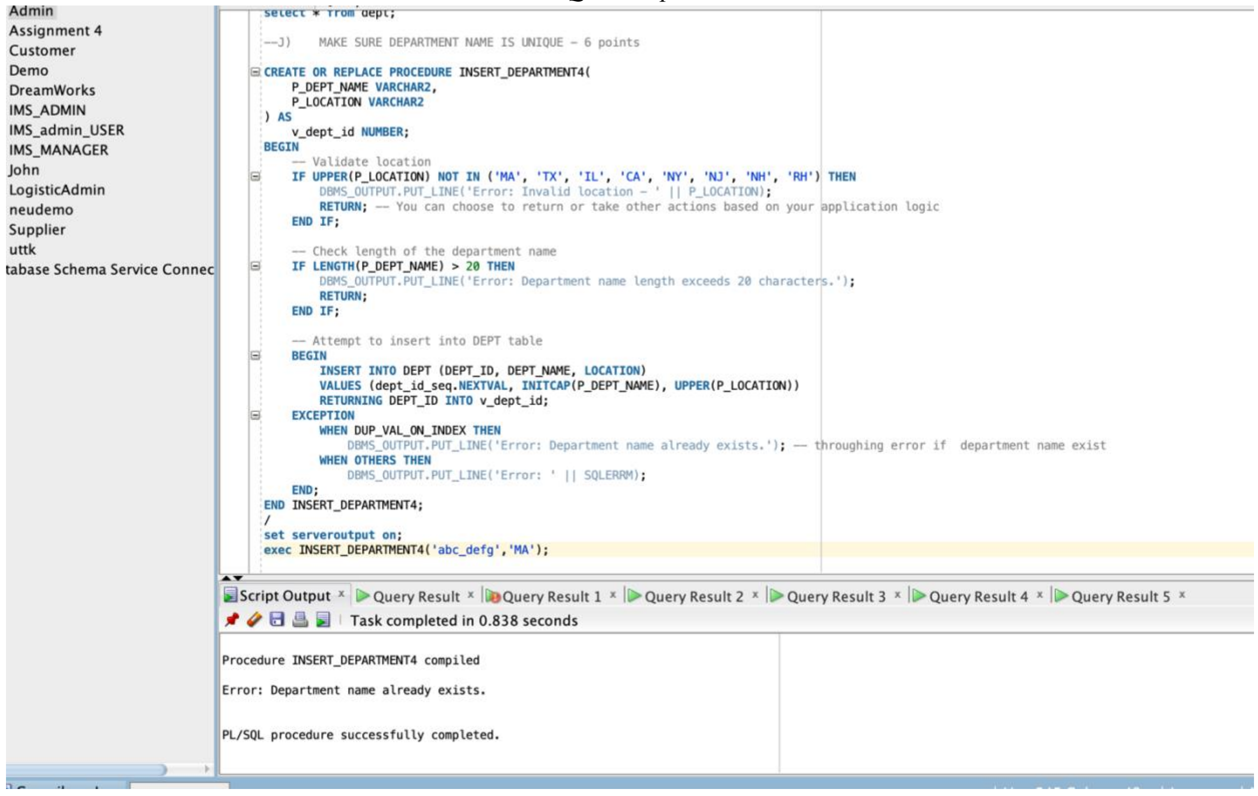


```
CREATE OR REPLACE PROCEDURE INSERT_DEPARTMENT3(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
    v_dept_id NUMBER;
BEGIN
    -- Validate location
    IF UPPER(P_LOCATION) NOT IN ('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH') THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid location - ' || P_LOCATION);
        RETURN; -- You can choose to return or take other actions based on your application logic
    END IF;

    -- Check length of the department name
    IF LENGTH(P_DEPT_NAME) > 20 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Department name length exceeds 20 characters.');
        RETURN;
    END IF;

    -- Attempt to insert into DEPT table
    BEGIN
        INSERT INTO DEPT (DEPT_ID, DEPT_NAME, LOCATION)
        VALUES (dept_id_seq.NEXTVAL, INITCAP(P_DEPT_NAME), UPPER(P_LOCATION)) --To convert the department name to Camel Case using INITCAP
        RETURNING DEPT_ID INTO v_dept_id;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.');
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END INSERT_DEPARTMENT3;
/
set serveroutput on;
exec INSERT_DEPARTMENT3('abc_defg','MA');


select * from dept;
```

All Rows Fetched: 6 in 0.139 seconds

	DEPT_ID	DEPT_NAME	LOCATION
1	1	Sales	NY
2	2	Finance	NJ
3	3	Abc	MA
4	4	Abcdef	MA
5	5	Abcdefg	MA
6	7	Abc_Defg	MA

J. MAKE SURE DEPARTMENT NAME IS UNIQUE - 6 points.



```
select * from dept;

--J)    MAKE SURE DEPARTMENT NAME IS UNIQUE - 6 points

CREATE OR REPLACE PROCEDURE INSERT_DEPARTMENT4(
    P_DEPT_NAME VARCHAR2,
    P_LOCATION VARCHAR2
) AS
    v_dept_id NUMBER;
BEGIN
    -- Validate location
    IF UPPER(P_LOCATION) NOT IN ('MA', 'TX', 'IL', 'CA', 'NY', 'NJ', 'NH', 'RH') THEN
        DBMS_OUTPUT.PUT_LINE('Error: Invalid location - ' || P_LOCATION);
        RETURN; -- You can choose to return or take other actions based on your application logic
    END IF;

    -- Check length of the department name
    IF LENGTH(P_DEPT_NAME) > 20 THEN
        DBMS_OUTPUT.PUT_LINE('Error: Department name length exceeds 20 characters.');
        RETURN;
    END IF;

    -- Attempt to insert into DEPT table
    BEGIN
        INSERT INTO DEPT (DEPT_ID, DEPT_NAME, LOCATION)
        VALUES (dept_id_seq.NEXTVAL, INITCAP(P_DEPT_NAME), UPPER(P_LOCATION))
        RETURNING DEPT_ID INTO v_dept_id;
    EXCEPTION
        WHEN DUP_VAL_ON_INDEX THEN
            DBMS_OUTPUT.PUT_LINE('Error: Department name already exists.'); -- throughing error if  department name exist
        WHEN OTHERS THEN
            DBMS_OUTPUT.PUT_LINE('Error: ' || SQLERRM);
    END;
END INSERT_DEPARTMENT4;
/
set serveroutput on;
exec INSERT_DEPARTMENT4('abc_defg','MA');
```

```
Procedure INSERT_DEPARTMENT4 compiled

Error: Department name already exists.


PL/SQL procedure successfully completed.
```