**Name:** Paarthivi D

**Superset ID:** 6410961

**DN 4.0 - Java FSE Mandatory Hands-on**

**WEEK – 2**

**PL/SQL PROGRAMMING:**

**Exercise 1: Control Structures**

**Scenario 1:** The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**Question:** Write a PL/SQL block that loops through all customers, checks their age, and if they are above 60, apply a 1% discount to their current loan interest rates.

**Code:**

-- Step 1: Create the customers table

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    name VARCHAR2(100),

    age NUMBER,

    loan\_interest\_rate NUMBER

);

-- Step 2: Insert sample data

INSERT INTO customers VALUES (1, 'Alice', 65, 8.5);

INSERT INTO customers VALUES (2, 'Bob', 55, 7.9);

INSERT INTO customers VALUES (3, 'Carol', 70, 9.0);

-- Step 3: PL/SQL block to apply 1% discount for customers over 60

BEGIN

    FOR cust IN (SELECT \* FROM customers) LOOP

        IF cust.age > 60 THEN

            UPDATE customers

            SET loan\_interest\_rate = loan\_interest\_rate - 1

            WHERE customer\_id = cust.customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('1% discount applied to: ' || cust.name);

        END IF;

    END LOOP;

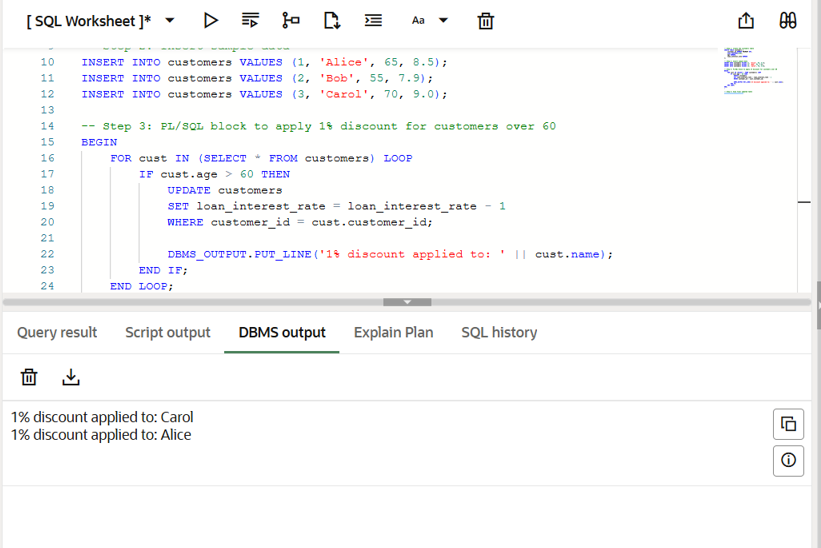
END;

/

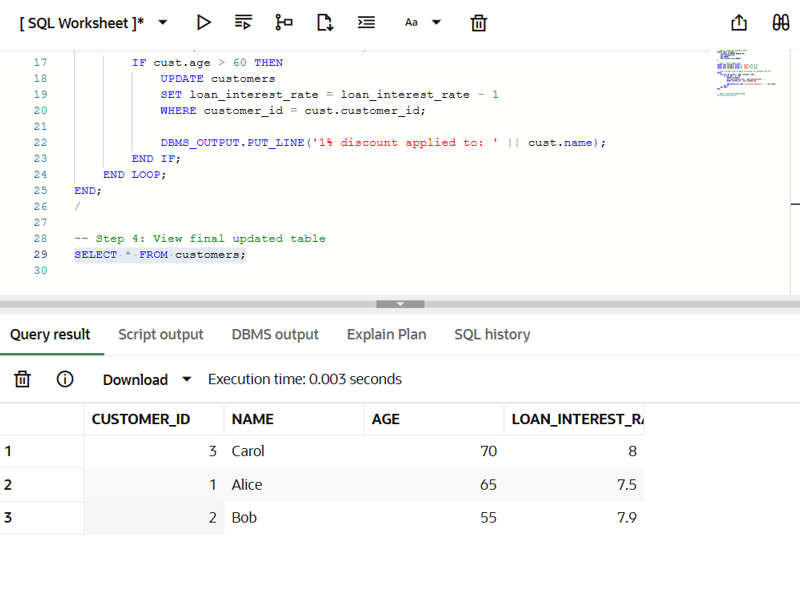
-- Step 4: View final updated table

SELECT \* FROM customers;

**DBMS output:**

****

**Query result:**

****

**Scenario 2:** A customer can be promoted to VIP status based on their balance.

**Question:** Write a PL/SQL block that iterates through all customers and sets a flag IsVIP to TRUE for those with a balance over $10,000.

**Code:**

-- Create customers table

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    name VARCHAR2(100),

    balance NUMBER,

    is\_vip VARCHAR2(5) DEFAULT 'FALSE'

);

-- Insert fresh data

INSERT INTO customers VALUES (1, 'Alice', 12000, 'FALSE');

INSERT INTO customers VALUES (2, 'Bob', 8000, 'FALSE');

INSERT INTO customers VALUES (3, 'Carol', 15000, 'FALSE');

-- Promote VIP customers with balance > 10000

BEGIN

    FOR cust IN (SELECT \* FROM customers) LOOP

        IF cust.balance > 10000 THEN

            UPDATE customers

            SET is\_vip = 'TRUE'

            WHERE customer\_id = cust.customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('VIP status applied to: ' || cust.name);

        END IF;

    END LOOP;

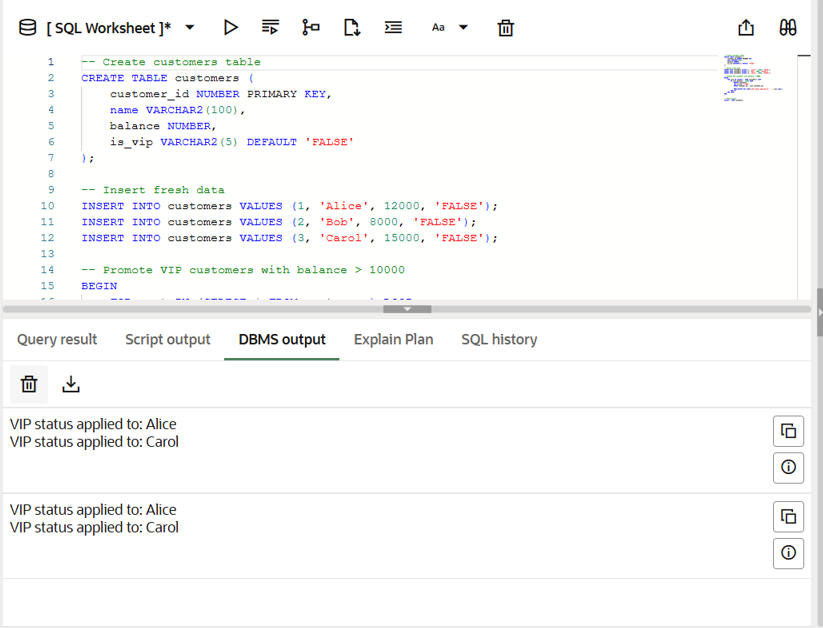
END;

/

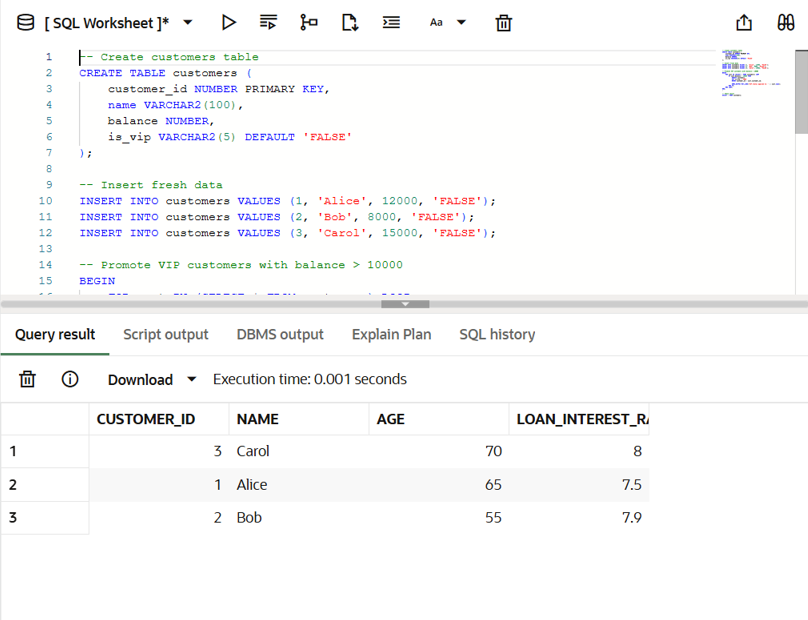
-- Check result

SELECT \* FROM customers;

**DBMS output:**



**Query result:**



**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**Question:** Write a PL/SQL block that fetches all loans due in the next 30 days and prints a reminder message for each customer.

**Code:**

-- Step 1: Create the customers table

CREATE TABLE customers (

    customer\_id NUMBER PRIMARY KEY,

    name VARCHAR2(100),

    balance NUMBER,

    loan\_interest\_rate NUMBER

);

-- Step 2: Insert sample data

INSERT INTO customers VALUES (1, 'Alice', 12000, 7.5);

INSERT INTO customers VALUES (2, 'Bob', 4000, 8.0);

INSERT INTO customers VALUES (3, 'Carol', 3000, 7.8);

-- Step 3: PL/SQL block to apply penalty for low-balance customers

BEGIN

    FOR cust IN (SELECT \* FROM customers) LOOP

        IF cust.balance < 5000 THEN

            UPDATE customers

            SET loan\_interest\_rate = loan\_interest\_rate + 1

            WHERE customer\_id = cust.customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('1% penalty applied to: ' || cust.name);

        END IF;

    END LOOP;

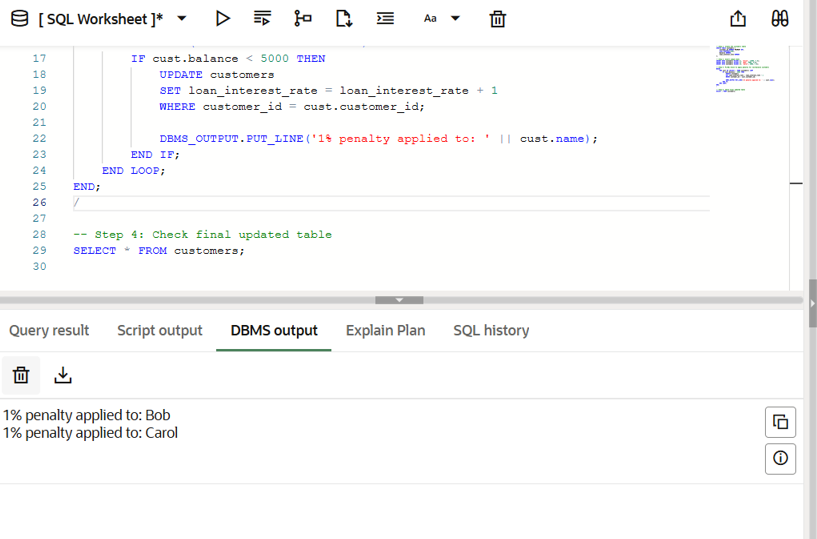
END;

/

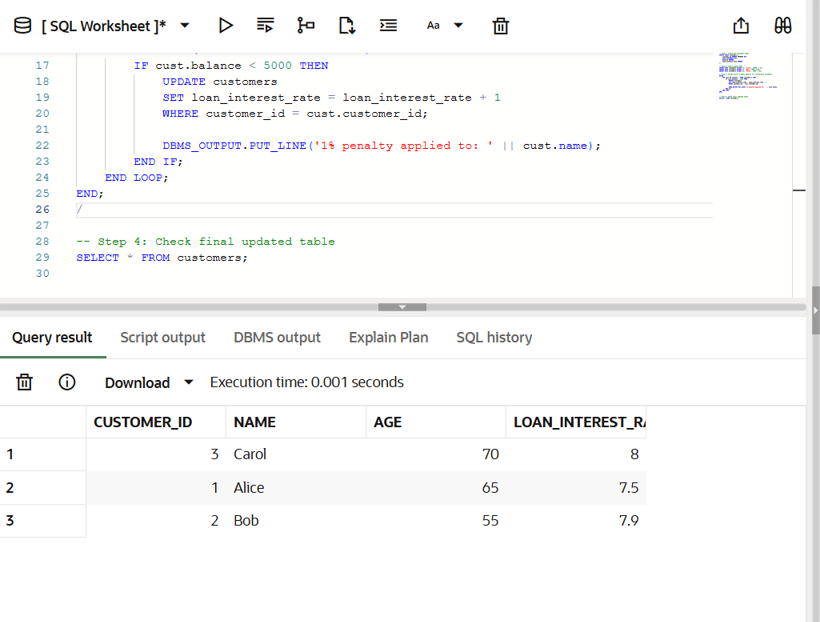
-- Step 4: Check final updated table

SELECT \* FROM customers;

**DBMS output:**



**Query result:**



**Exercise 3: Stored Procedures**

**Scenario 1:** The bank needs to process monthly interest for all savings accounts.

**Question:** Write a stored procedure **ProcessMonthlyInterest** that calculates and updates the balance of all savings accounts by applying an interest rate of 1% to the current balance.

**Code:**

-- Drop table if it exists

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE savings\_accounts';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create the table

CREATE TABLE savings\_accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_name VARCHAR2(100),

    balance NUMBER

);

-- Insert sample data

INSERT INTO savings\_accounts VALUES (101, 'Alice', 10000);

INSERT INTO savings\_accounts VALUES (102, 'Bob', 5000);

INSERT INTO savings\_accounts VALUES (103, 'Carol', 7500);

-- Create the procedure

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

    FOR acc IN (SELECT \* FROM savings\_accounts) LOOP

        UPDATE savings\_accounts

        SET balance = balance + (balance \* 0.01)

        WHERE account\_id = acc.account\_id;

        DBMS\_OUTPUT.PUT\_LINE('Interest added for account: ' || acc.account\_id);

    END LOOP;

END;

/

-- Enable DBMS output

SET SERVEROUTPUT ON;

-- Call the procedure

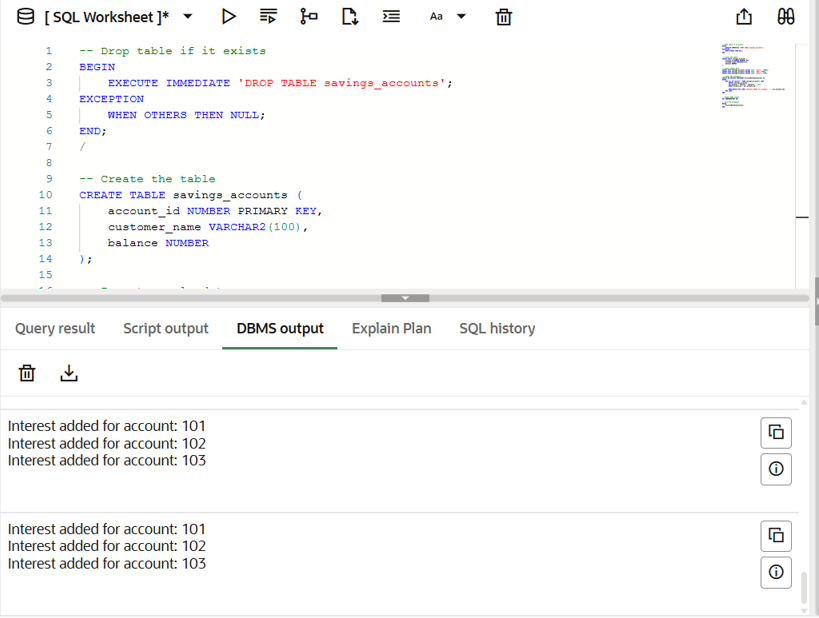
BEGIN

    ProcessMonthlyInterest;

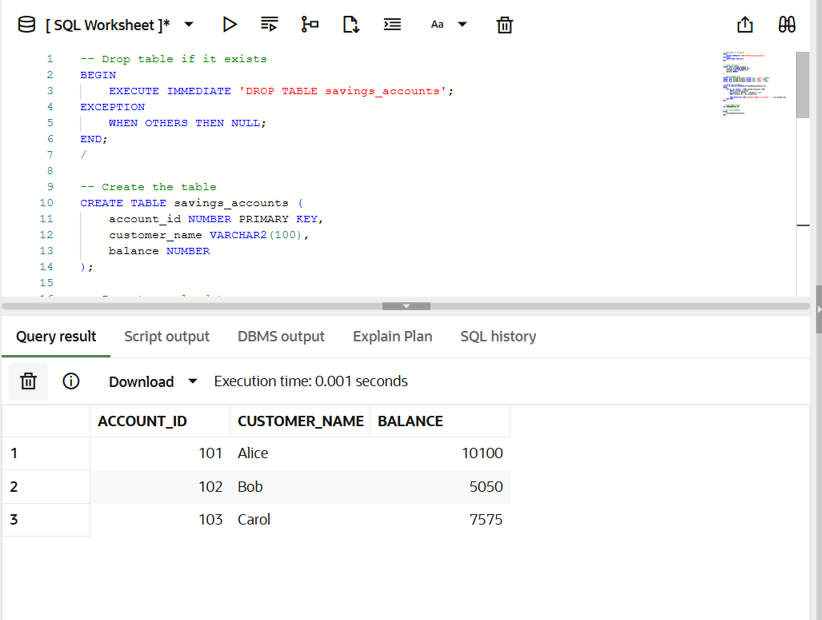
END;

/

**DBMS output:**

****

**Query result:**

****

**Scenario 2:** The bank wants to implement a bonus scheme for employees based on their performance.

**Question:** Write a stored procedure **UpdateEmployeeBonus** that updates the salary of employees in a given department by adding a bonus percentage passed as a parameter.

**Code:**

-- Step 1: Create the employees table

CREATE TABLE employees (

    emp\_id NUMBER PRIMARY KEY,

    name VARCHAR2(100),

    department VARCHAR2(50),

    salary NUMBER

);

-- Step 2: Insert sample employee data

INSERT INTO employees VALUES (1, 'Alice', 'HR', 30000);

INSERT INTO employees VALUES (2, 'Bob', 'IT', 40000);

INSERT INTO employees VALUES (3, 'Carol', 'HR', 32000);

INSERT INTO employees VALUES (4, 'David', 'IT', 42000);

INSERT INTO employees VALUES (5, 'Eve', 'Finance', 35000);

-- Step 3: Create the stored procedure

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_percent IN NUMBER

) AS

BEGIN

    FOR emp IN (

        SELECT \* FROM employees WHERE department = p\_department

    ) LOOP

        UPDATE employees

        SET salary = salary + (salary \* p\_bonus\_percent / 100)

        WHERE emp\_id = emp.emp\_id;

        DBMS\_OUTPUT.PUT\_LINE(

            'Bonus applied to: ' || emp.name || ', New Salary: ' ||

            TO\_CHAR(emp.salary + (emp.salary \* p\_bonus\_percent / 100))

        );

    END LOOP;

END;

/

-- Step 4: Enable DBMS output

SET SERVEROUTPUT ON;

-- Step 5: Call the procedure (Example: give 10% bonus to IT department)

BEGIN

    UpdateEmployeeBonus('IT', 10);

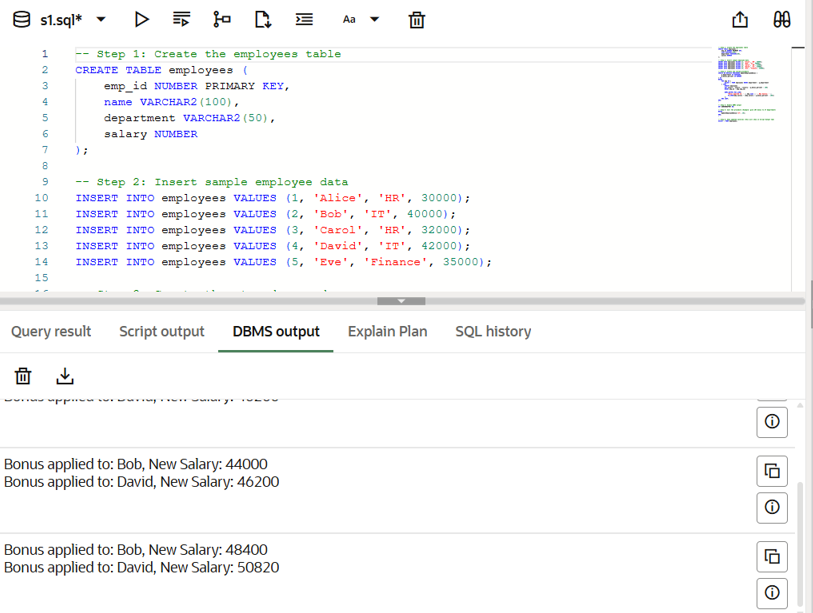
END;

/

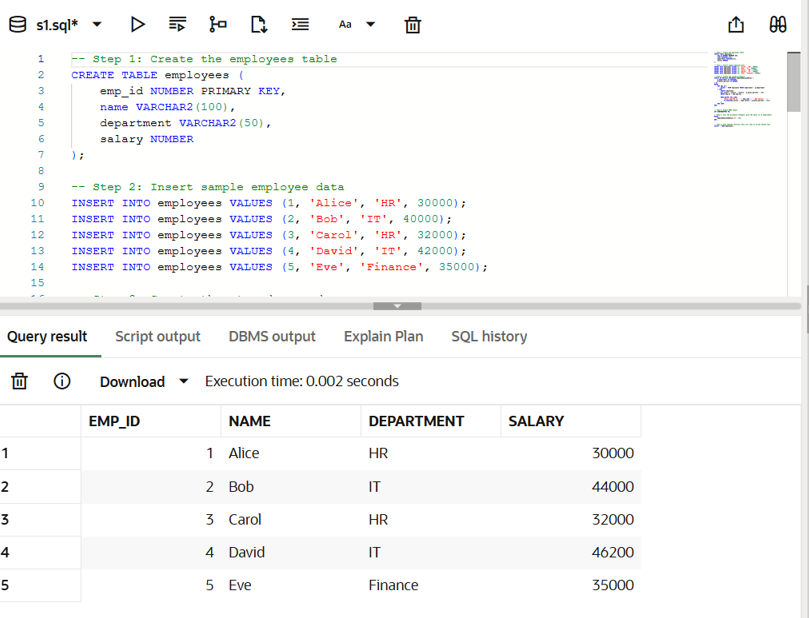
-- Step 6: Show updated salaries (this will show in Script Output tab)

SELECT \* FROM employees;

**DBMS output:**



**Query result:**



**Scenario 3:** Customers should be able to transfer funds between their accounts.

**Question:** Write a stored procedure **TransferFunds** that transfers a specified amount from one account to another, checking that the source account has sufficient balance before making the transfer.

**Code:**

CREATE TABLE accounts (

    account\_id NUMBER PRIMARY KEY,

    customer\_name VARCHAR2(100),

    balance NUMBER

);

INSERT INTO accounts VALUES (1, 'Alice', 10000);

INSERT INTO accounts VALUES (2, 'Bob', 6000);

INSERT INTO accounts VALUES (3, 'Carol', 8000);

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_from\_account IN NUMBER,

    p\_to\_account IN NUMBER,

    p\_amount IN NUMBER

) AS

    v\_from\_balance NUMBER;

BEGIN

    SELECT balance INTO v\_from\_balance

    FROM accounts

    WHERE account\_id = p\_from\_account;

    IF v\_from\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient balance in account ' || p\_from\_account);

    ELSE

        UPDATE accounts

        SET balance = balance - p\_amount

        WHERE account\_id = p\_from\_account;

        UPDATE accounts

        SET balance = balance + p\_amount

        WHERE account\_id = p\_to\_account;

        DBMS\_OUTPUT.PUT\_LINE('Transferred ₹' || p\_amount || ' from account ' || p\_from\_account || ' to account ' || p\_to\_account);

    END IF;

END;

/

SET SERVEROUTPUT ON;

BEGIN

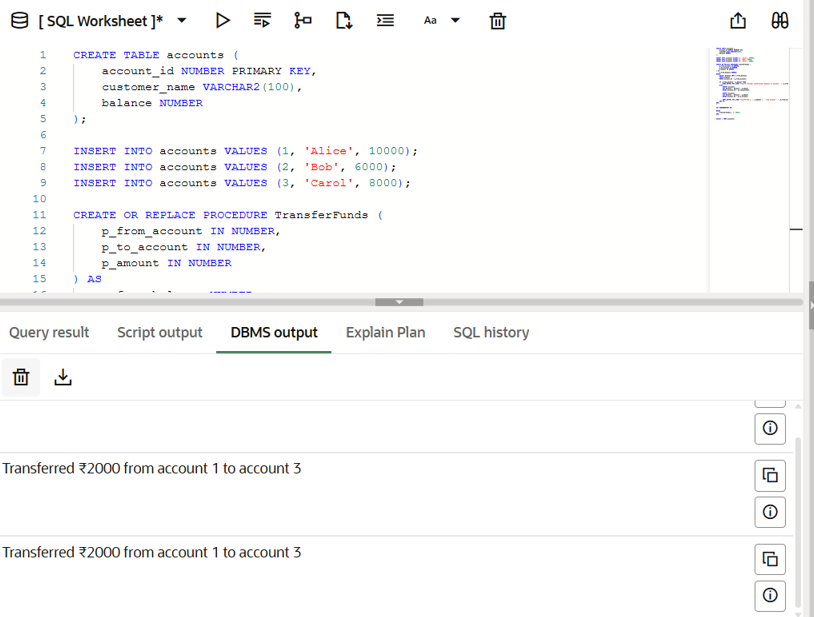
    TransferFunds(1, 3, 2000);

END;

/

SELECT \* FROM accounts;

**DBMS output:**



**Query result:**

