**Exercise 1: Control Structures**

**Scenario 1:**

**Code:**

DECLARE

    CURSOR customer\_cursor IS

        SELECT c.customerid, c.dob

        FROM customers c;

    v\_customer\_id   customers.customerid%TYPE;

    v\_dob           customers.dob%TYPE;

    v\_age           NUMBER;

BEGIN

    FOR customer\_rec IN customer\_cursor LOOP

        v\_customer\_id := customer\_rec.customerid;

        v\_dob := customer\_rec.dob;

        v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, v\_dob) / 12);

        IF v\_age > 60 THEN

            UPDATE loans

            SET interestrate = interestrate - 1

            WHERE customerid = v\_customer\_id;

            DBMS\_OUTPUT.PUT\_LINE('1% interest applied to Customer ID ' || v\_customer\_id);

        END IF;

    END LOOP;

    COMMIT;

END;

/

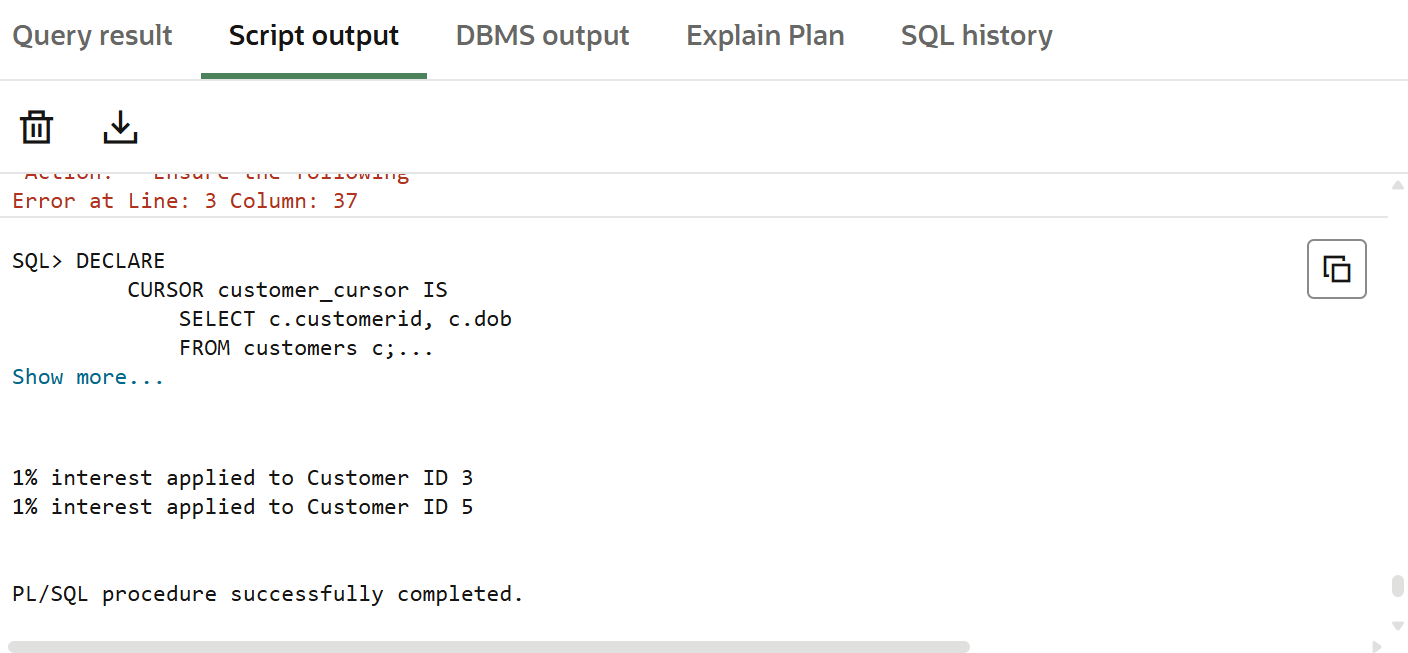
SELECT l.loanid, l.customerid, l.interestrate, c.name, c.dob

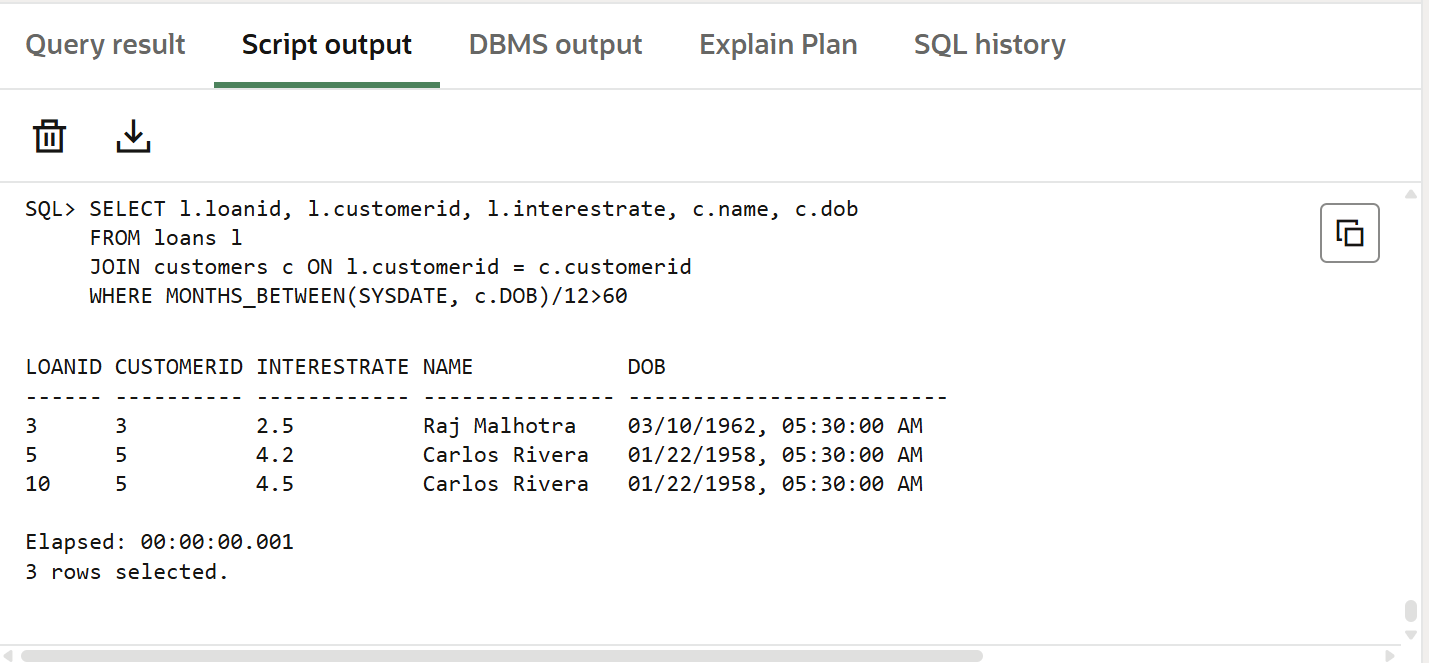
FROM loans l

JOIN customers c ON l.customer\_id = c.customer\_id

WHERE MONTHS\_BETWEEN(SYSDATE, c.DOB)/12>60;

**Output:**

****



**Scenario 2:**

**Code:**

ALTER TABLE customers

ADD IsVIP CHAR(5) DEFAULT 'False';

DECLARE

    CURSOR customer\_cursor IS

        SELECT customerid, balance

        FROM customers;

    v\_customer\_id   customers.customerid%TYPE;

    v\_balance       customers.balance%TYPE;

BEGIN

    FOR customer\_rec IN customer\_cursor LOOP

        v\_customer\_id := customer\_rec.customerid;

        v\_balance := customer\_rec.balance;

        IF v\_balance > 10000 THEN

            UPDATE customers

            SET IsVIP = 'True'

            WHERE customerid = v\_customer\_id;

        END IF;

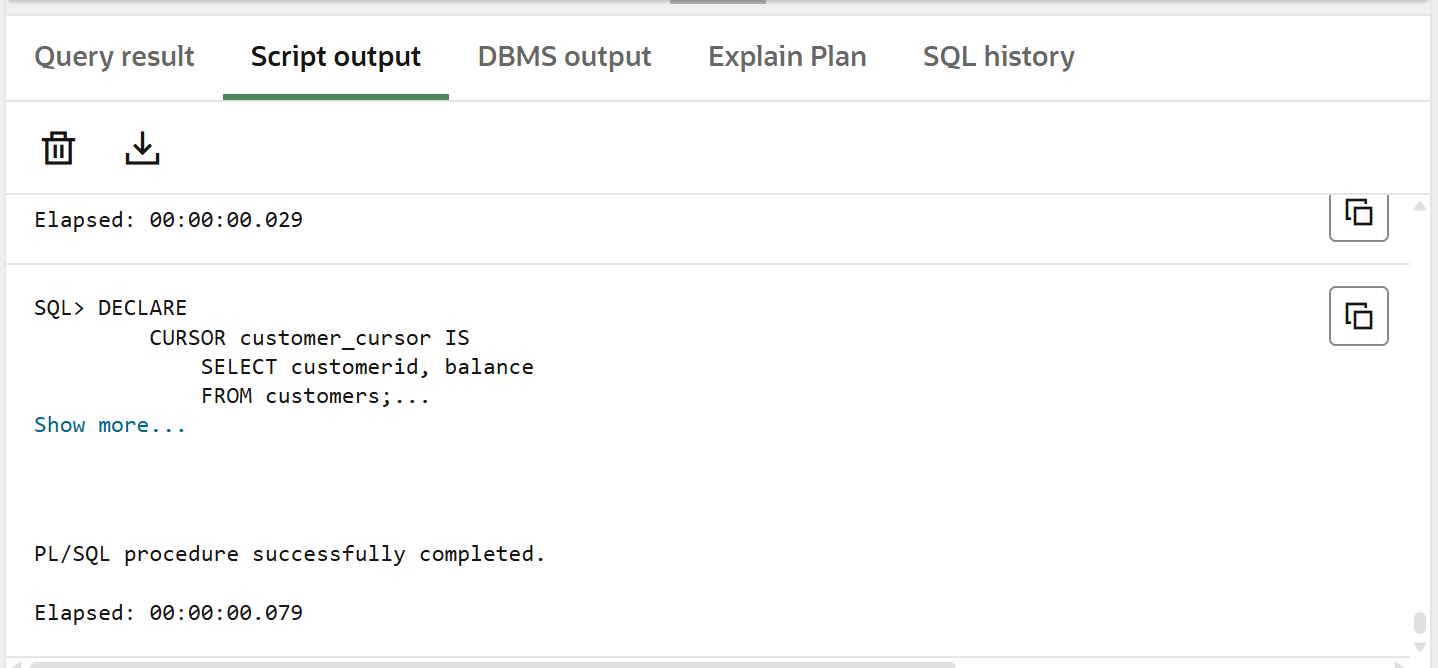
    END LOOP;

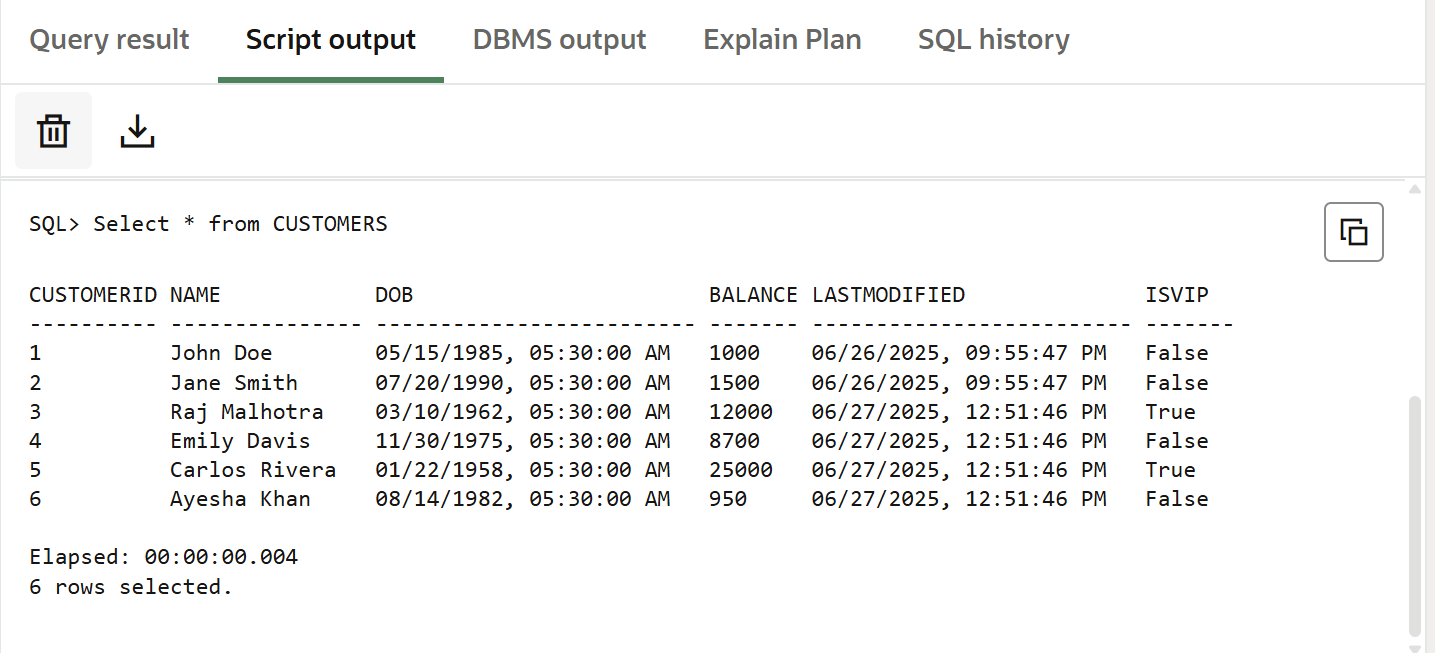
COMMIT;

END;

Select \* From CUSTOMERS;

**Output:**

****



**Scenario 3:**

**Code:**

DECLARE

CURSOR loan\_cursor IS

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

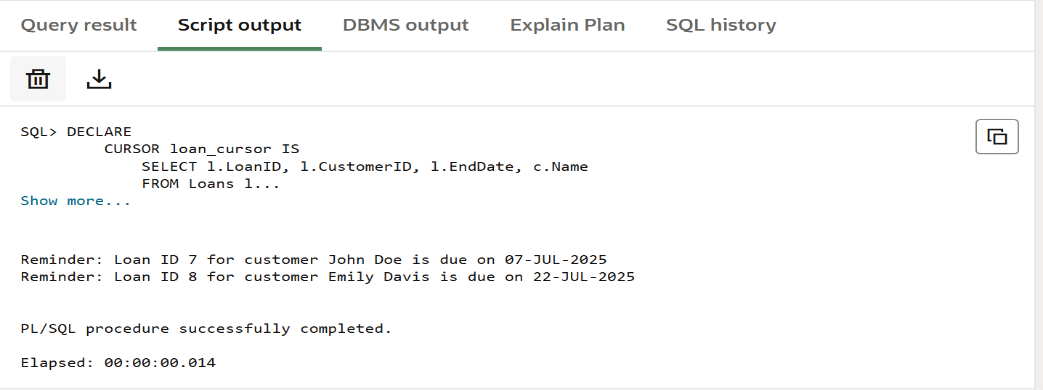
' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'DD-MON-YYYY')

);

END LOOP;

END;

/



**EXERCISE 3: STORED PROCEDURES**

**Scenario 1:**

**Code:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

    UPDATE Accounts

    SET

        Balance = Balance + (Balance \* 0.01),

        LastModified = SYSDATE

    WHERE

        UPPER(AccountType) = 'SAVINGS';

    COMMIT;

END;

/

BEGIN

    ProcessMonthlyInterest;

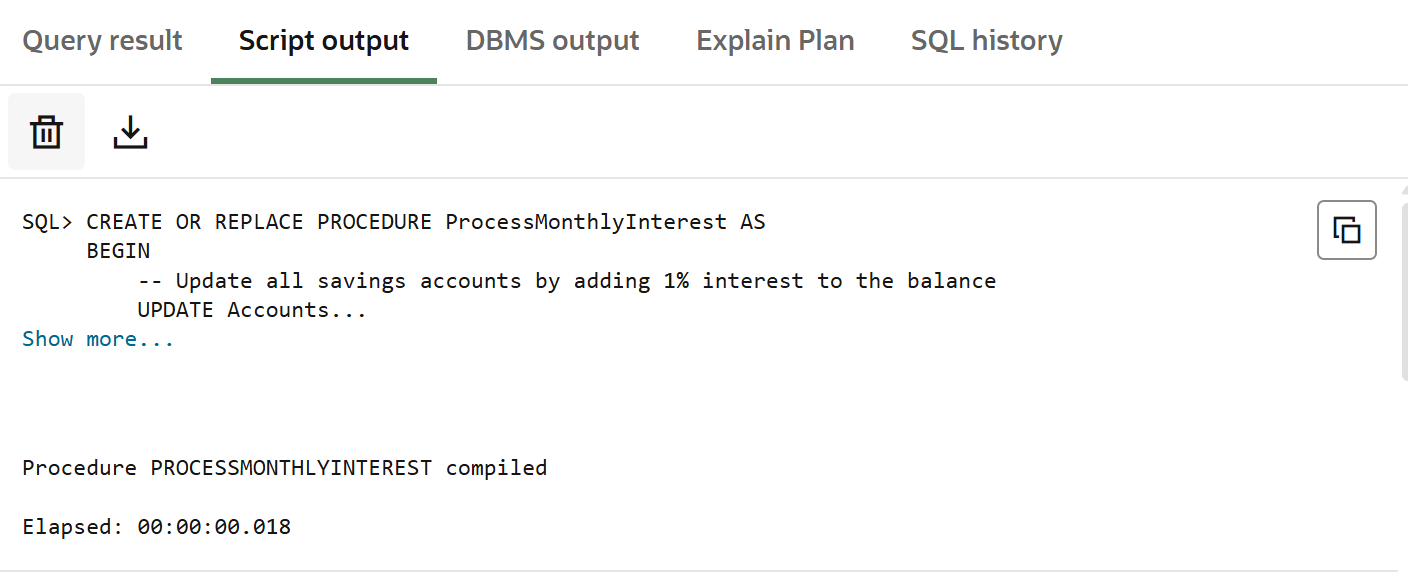
END;

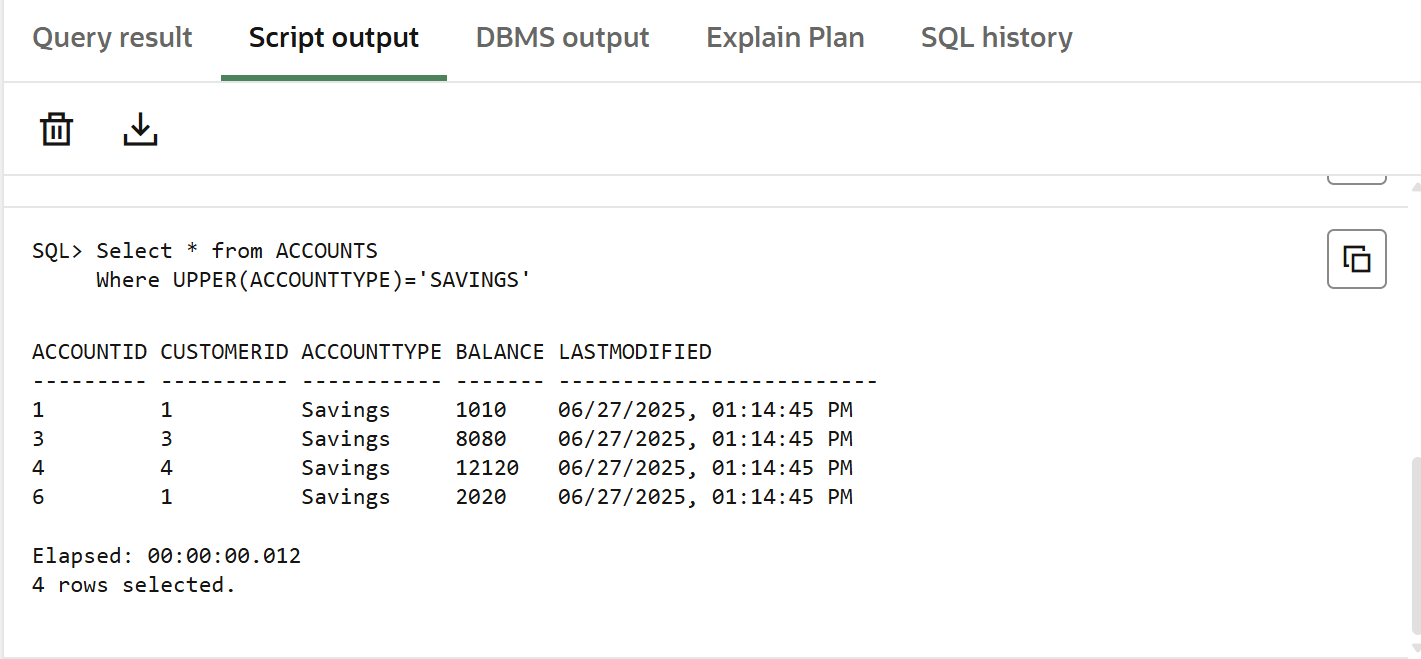
/

Select \* from ACCOUNTS

Where UPPER(ACCOUNTTYPE)='SAVINGS';

**OUTPUT:**

****

****

**Scenario 2:**

**Code:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_DepartmentName IN VARCHAR2,

    p\_BonusPercent IN NUMBER

)

AS

    CURSOR emp\_cursor IS

        SELECT EmployeeID, Name, Position, Salary

        FROM Employees

        WHERE UPPER(Department) = UPPER(p\_DepartmentName);

    v\_NewSalary Employees.Salary%TYPE;

BEGIN

    FOR emp\_rec IN emp\_cursor LOOP

        -- Calculate new salary

        v\_NewSalary := emp\_rec.Salary + (emp\_rec.Salary \* (p\_BonusPercent / 100));

        DBMS\_OUTPUT.PUT\_LINE('BEFORE -> ID: ' || emp\_rec.EmployeeID ||

                             ', Name: ' || emp\_rec.Name ||

                             ', Position: ' || emp\_rec.Position ||

                             ', Salary: ' || emp\_rec.Salary);

        UPDATE Employees

        SET Salary = v\_NewSalary

        WHERE EmployeeID = emp\_rec.EmployeeID;

        DBMS\_OUTPUT.PUT\_LINE('AFTER  -> ID: ' || emp\_rec.EmployeeID ||

                             ', Name: ' || emp\_rec.Name ||

                             ', Position: ' || emp\_rec.Position ||

                             ', Salary: ' || v\_NewSalary);

        DBMS\_OUTPUT.PUT\_LINE('-----------------------------------');

    END LOOP;

    COMMIT;

END;

/

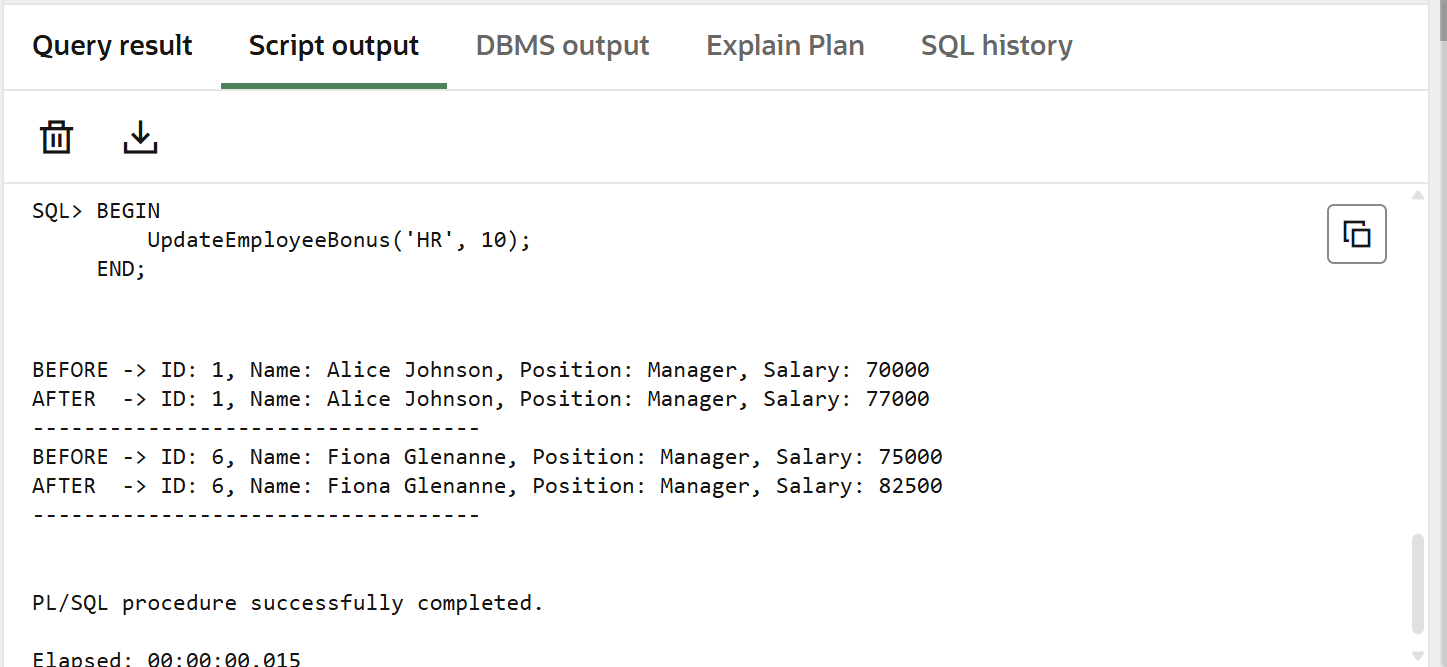
BEGIN

    UpdateEmployeeBonus('HR', 10);

END;

/

**Output:**

****

**Scenario 3:**

**Code:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_FromAccountID IN NUMBER,

    p\_ToAccountID IN NUMBER,

    p\_Amount IN NUMBER

)

AS

    v\_FromBalanceBefore Accounts.Balance%TYPE;

    v\_ToBalanceBefore   Accounts.Balance%TYPE;

    v\_FromBalanceAfter  Accounts.Balance%TYPE;

    v\_ToBalanceAfter    Accounts.Balance%TYPE;

BEGIN

    SELECT Balance INTO v\_FromBalanceBefore

    FROM Accounts

    WHERE AccountID = p\_FromAccountID

    FOR UPDATE;

    SELECT Balance INTO v\_ToBalanceBefore

    FROM Accounts

    WHERE AccountID = p\_ToAccountID

    FOR UPDATE;

    IF v\_FromBalanceBefore < p\_Amount THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

    END IF;

    UPDATE Accounts

    SET Balance = Balance - p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_FromAccountID;

    UPDATE Accounts

    SET Balance = Balance + p\_Amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_ToAccountID;

    SELECT Balance INTO v\_FromBalanceAfter

    FROM Accounts

    WHERE AccountID = p\_FromAccountID;

    SELECT Balance INTO v\_ToBalanceAfter

    FROM Accounts

    WHERE AccountID = p\_ToAccountID;

    DBMS\_OUTPUT.PUT\_LINE('--- Transfer Summary ---');

    DBMS\_OUTPUT.PUT\_LINE('From Account ID: ' || p\_FromAccountID);

    DBMS\_OUTPUT.PUT\_LINE('  Balance BEFORE: ' || v\_FromBalanceBefore);

    DBMS\_OUTPUT.PUT\_LINE('  Balance AFTER : ' || v\_FromBalanceAfter);

    DBMS\_OUTPUT.PUT\_LINE('To Account ID  : ' || p\_ToAccountID);

    DBMS\_OUTPUT.PUT\_LINE('  Balance BEFORE: ' || v\_ToBalanceBefore);

    DBMS\_OUTPUT.PUT\_LINE('  Balance AFTER : ' || v\_ToBalanceAfter);

    COMMIT;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        RAISE\_APPLICATION\_ERROR(-20002, 'One or both account IDs are invalid.');

    WHEN OTHERS THEN

        ROLLBACK;

        RAISE;

END;

/

BEGIN

    TransferFunds(3, 6, 500);

END;

/

**OUTPUT:**

