**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 :-**

BEGIN

    FOR rec IN (

        SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate

        FROM Customers c

        JOIN Loans l ON c.CustomerID = l.CustomerID

    )

    LOOP

        IF MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12 > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate - 1

            WHERE LoanID = rec.LoanID;

        END IF;

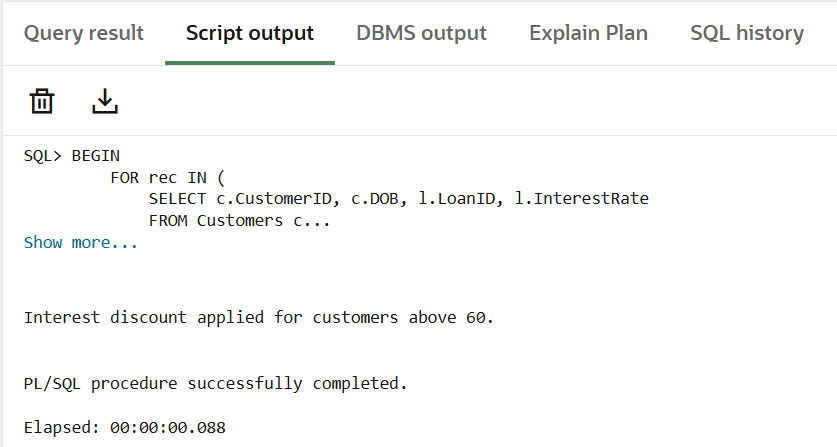
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('Interest discount applied for customers above 60.');

END;

/

**Output :-**



**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 :-**

BEGIN

    FOR rec IN (SELECT CustomerID, Balance FROM Customers)

    LOOP

        IF rec.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'TRUE'

            WHERE CustomerID = rec.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'FALSE'

            WHERE CustomerID = rec.CustomerID;

        END IF;

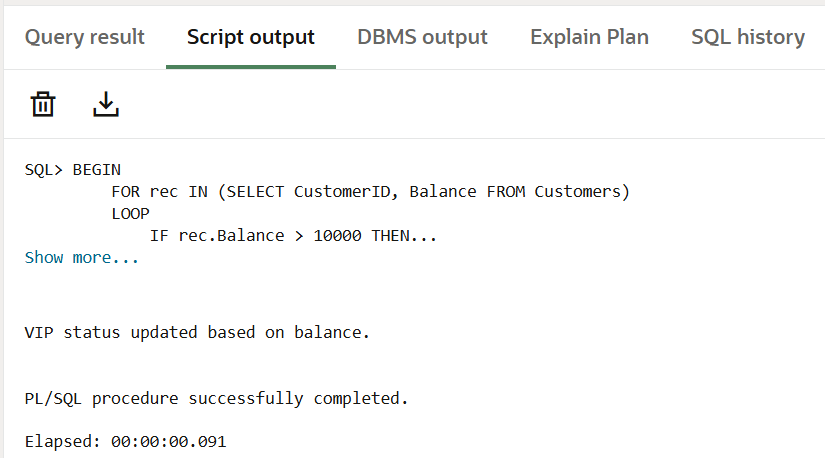
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('VIP status updated based on balance.');

END;

/

**Output :-**

****

**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 :-**

BEGIN

    FOR rec IN (

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

        FROM Loans l

        JOIN Customers c ON l.CustomerID = c.CustomerID

        WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

    )

    LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID ||

                             ' for customer ' || rec.Name ||

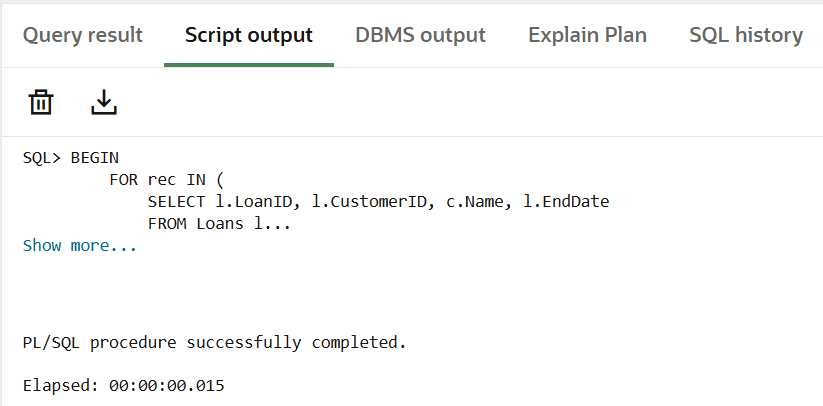
                             ' is due on ' || TO\_CHAR(rec.EndDate, 'YYYY-MM-DD'));

    END LOOP;

END;

/

**Output :-**

****

**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 :-**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    )

    LOOP

        UPDATE Accounts

        SET Balance = Balance + (acc.Balance \* 0.01),

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('Monthly interest processed for all savings accounts.');

END;

/

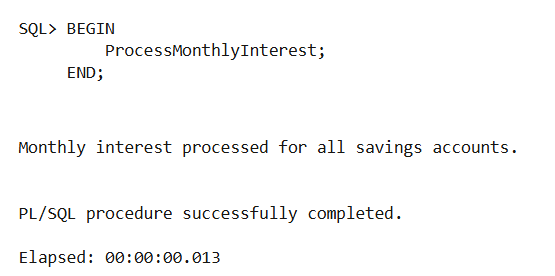
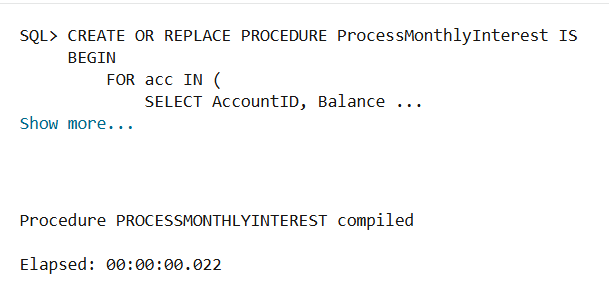
BEGIN

    ProcessMonthlyInterest;

END;

/

**Output :-**

****

**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 :-**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    p\_department IN VARCHAR2,

    p\_bonus\_percent IN NUMBER

) IS

BEGIN

    UPDATE Employees

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

    WHERE Department = p\_department;

    DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department: ' || p\_department);

END;

/

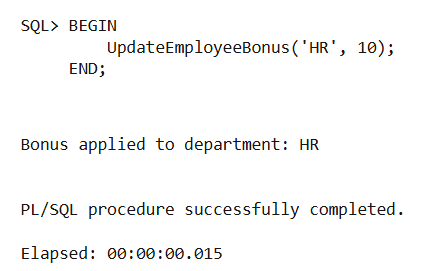
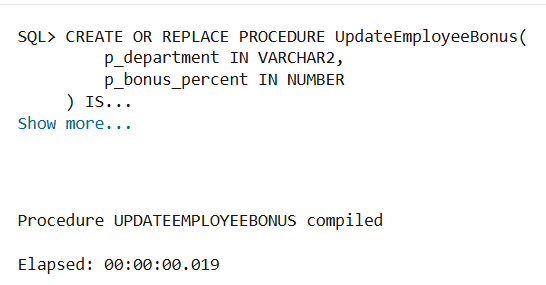
BEGIN

    UpdateEmployeeBonus('HR', 10);

END;

/

**Output :-**

****

**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 OR REPLACE PROCEDURE TransferFunds(

    p\_from\_account IN NUMBER,

    p\_to\_account IN NUMBER,

    p\_amount IN NUMBER

) IS

    v\_balance NUMBER;

BEGIN

    -- Get source account balance

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account

    FOR UPDATE;

    -- Check for sufficient balance

    IF v\_balance >= p\_amount THEN

        -- Deduct from source

        UPDATE Accounts

        SET Balance = Balance - p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_from\_account;

        -- Add to destination

        UPDATE Accounts

        SET Balance = Balance + p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_to\_account;

        DBMS\_OUTPUT.PUT\_LINE('Transfer successful.');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in source account.');

    END IF;

EXCEPTION

    WHEN NO\_DATA\_FOUND THEN

        DBMS\_OUTPUT.PUT\_LINE('Invalid account number(s).');

    WHEN OTHERS THEN

        DBMS\_OUTPUT.PUT\_LINE('Error during transfer: ' || SQLERRM);

END;

/

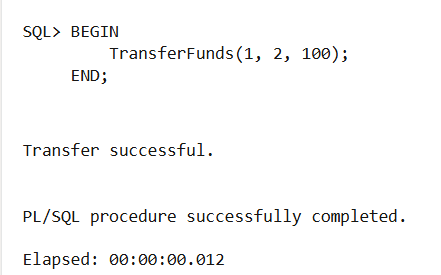
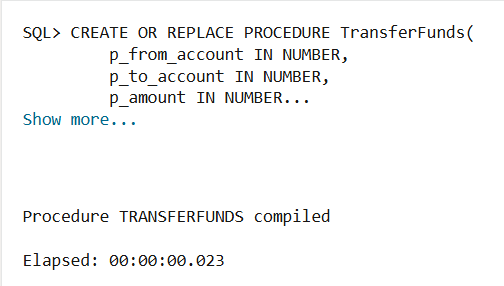
BEGIN

    TransferFunds(1, 2, 100);

END;

/

**Output :-**

****