**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 customer\_rec IN (SELECT CustomerID, InterestRate, Age FROM Customers) LOOP

    IF customer\_rec.Age > 60 THEN

      UPDATE Customers

      SET InterestRate = InterestRate - 1

      WHERE CustomerID = customer\_rec.CustomerID;

    END IF;

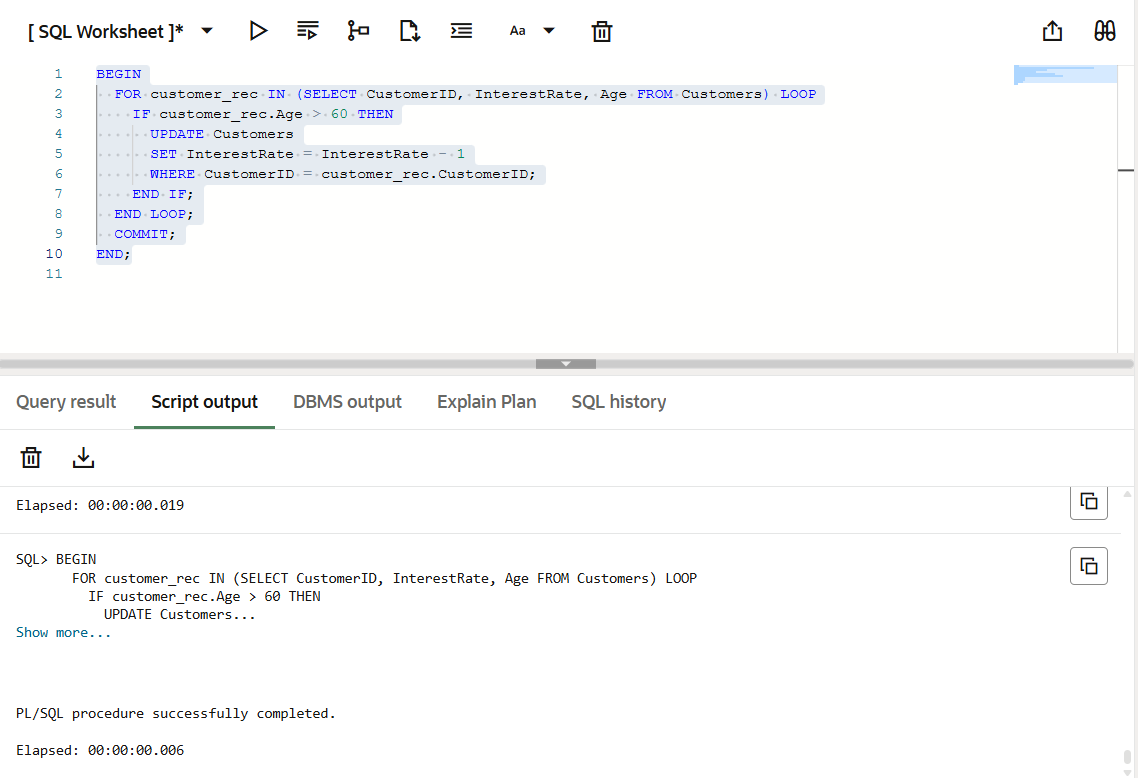
  END LOOP;

  COMMIT;

END;

select \* from CUSTOMERS;

**#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 customer\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF customer\_rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = customer\_rec.CustomerID;

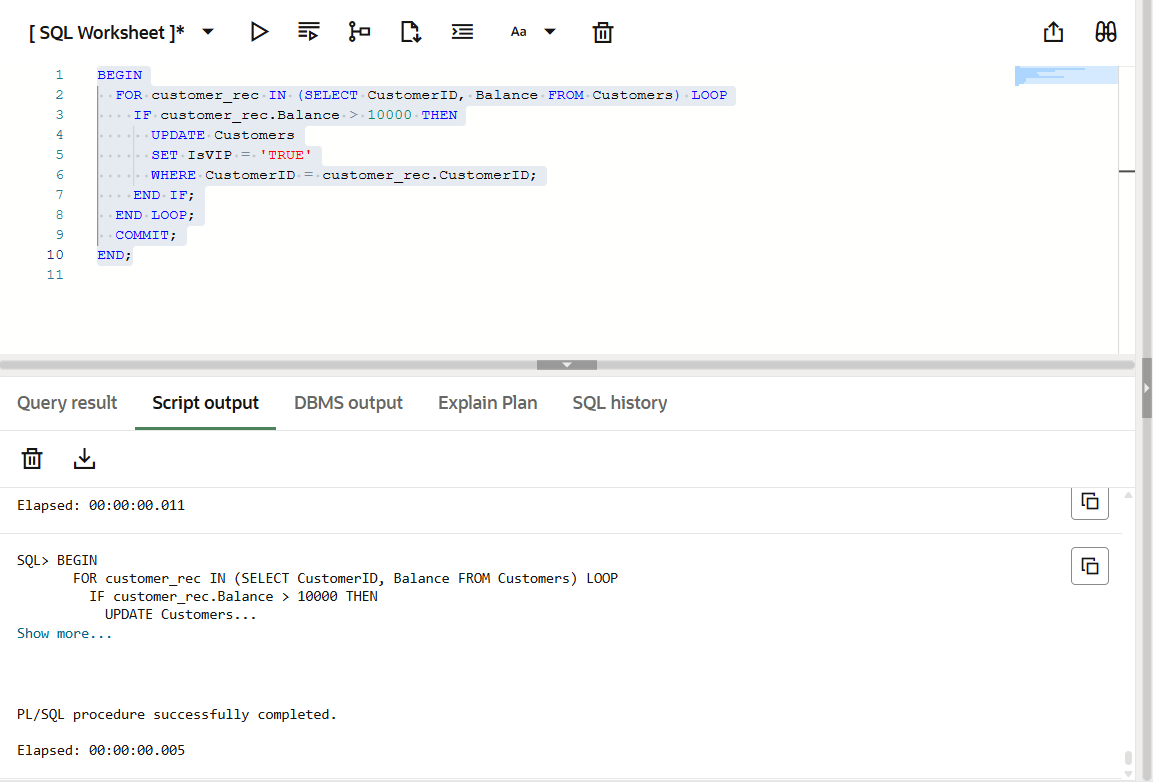
    END IF;

  END LOOP;

  COMMIT;

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 loan\_rec IN (

    SELECT CustomerID, DueDate

    FROM Loans

    WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

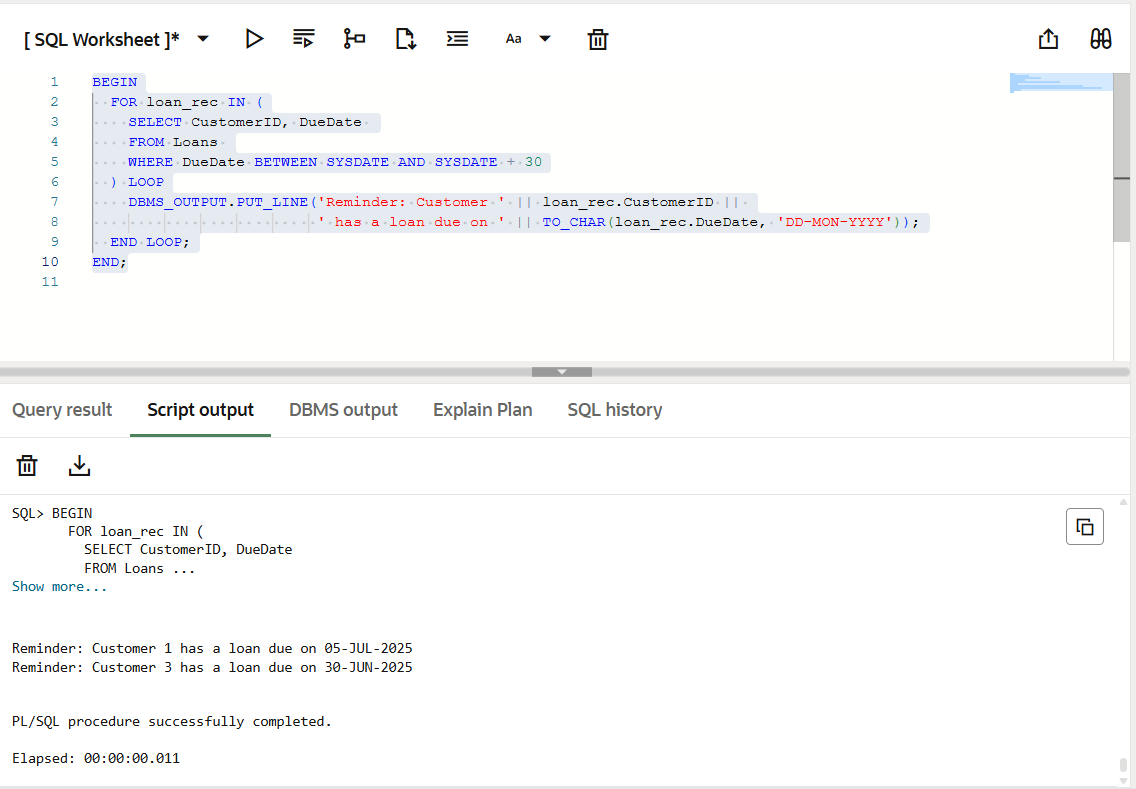
    DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || loan\_rec.CustomerID ||

                         ' has a loan due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

  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

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

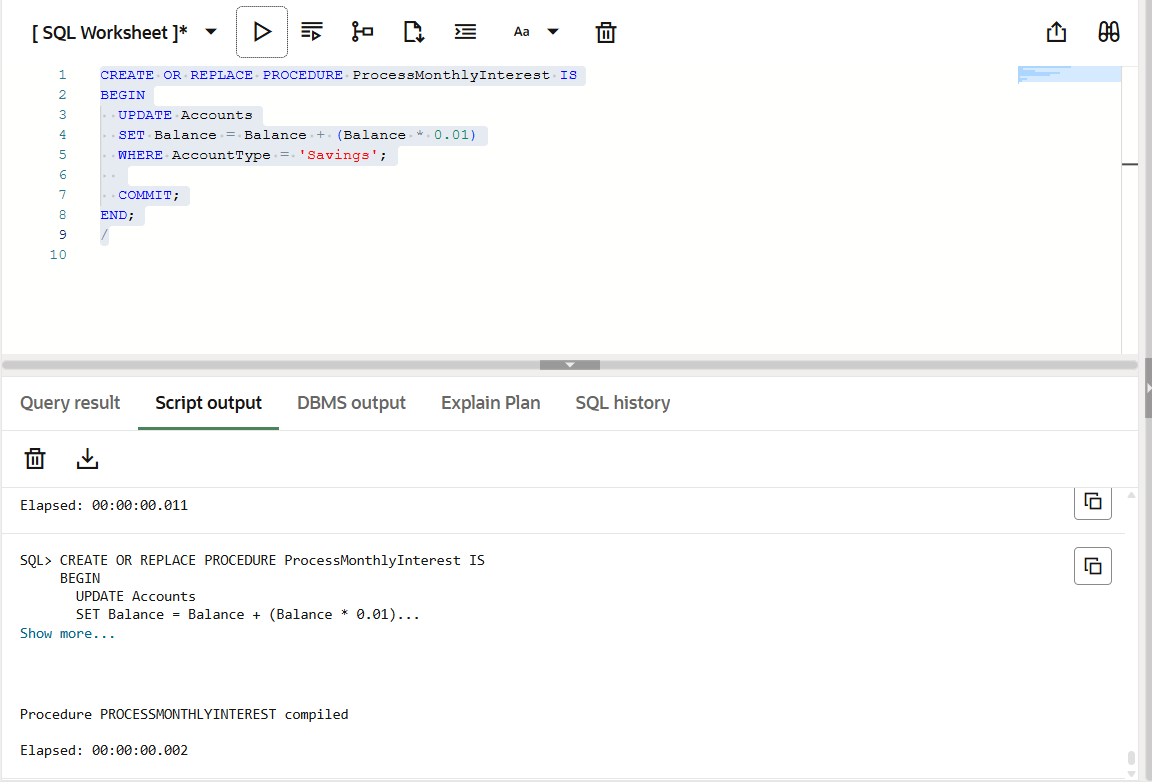
WHERE AccountType = 'Savings';

COMMIT;

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))

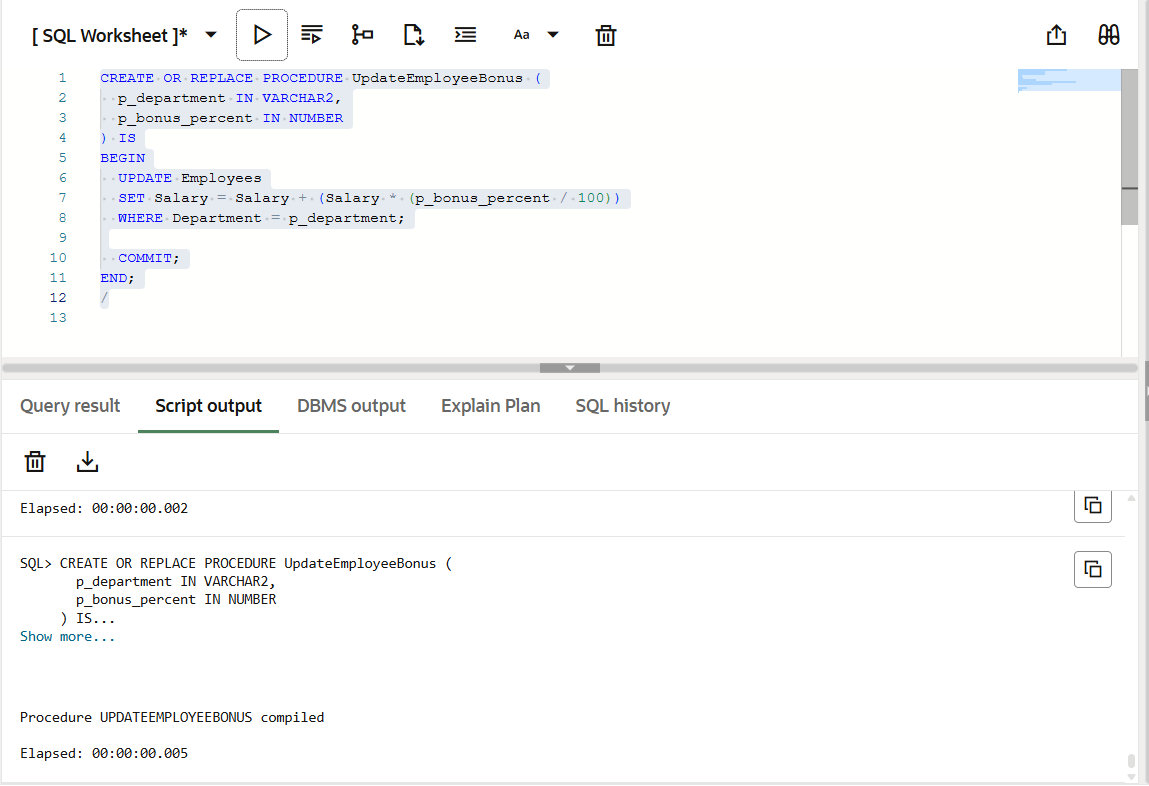
WHERE Department = p\_department;

COMMIT;

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

  SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account;

  IF v\_balance >= p\_amount THEN

    -- Deduct from source

    UPDATE Accounts

    SET Balance = Balance - p\_amount

    WHERE AccountID = p\_from\_account;

    UPDATE Accounts

    SET Balance = Balance + p\_amount

    WHERE AccountID = p\_to\_account;

    COMMIT;

  ELSE

    RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance');

  END IF;

END;

/

**#Output**

