**Week2\_PL/SQL Programming \_HandsOn**

**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 cust\_rec IN (

SELECT CustomerID, Age, InterestRate

FROM Customers

WHERE Age > 60

) LOOP

UPDATE Customers

SET InterestRate = InterestRate - 1

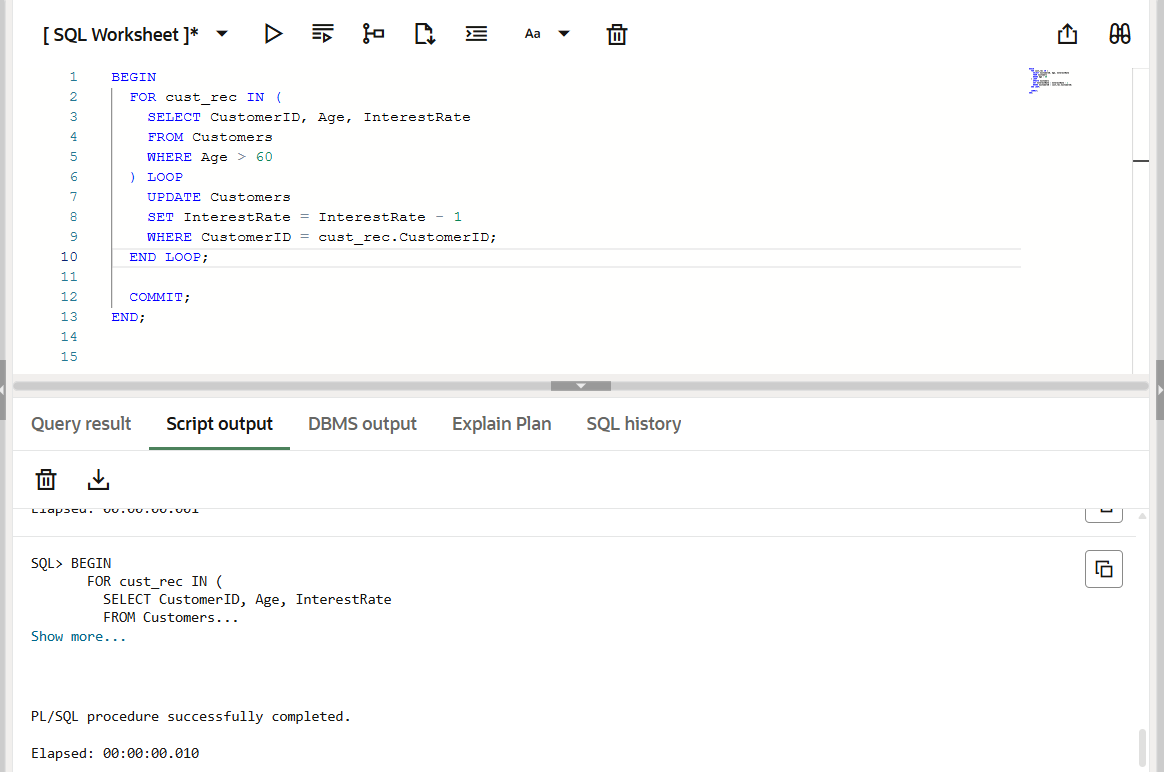
WHERE CustomerID = cust\_rec.CustomerID;

END LOOP;

COMMIT;

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 cust\_rec IN (

SELECT CustomerID, Balance

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

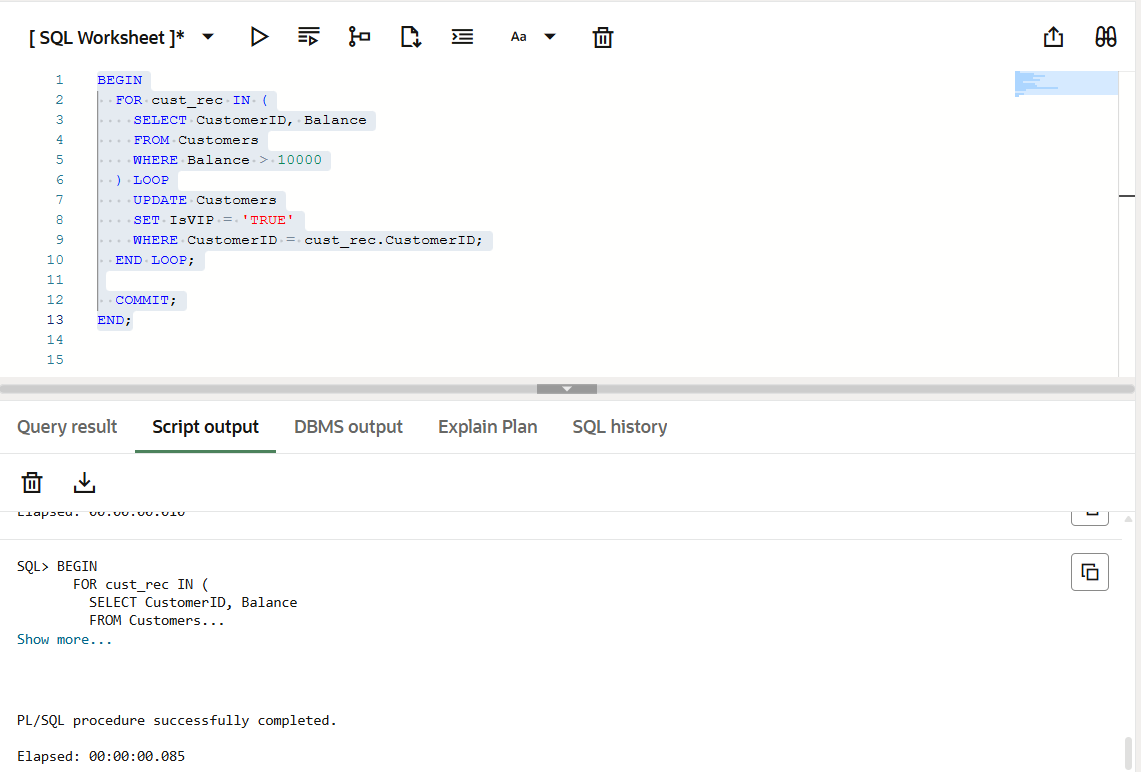
WHERE CustomerID = cust\_rec.CustomerID;

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 :**

DECLARE

v\_due\_date DATE := SYSDATE + 30;

BEGIN

FOR loan\_rec IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate <= v\_due\_date

) LOOP

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

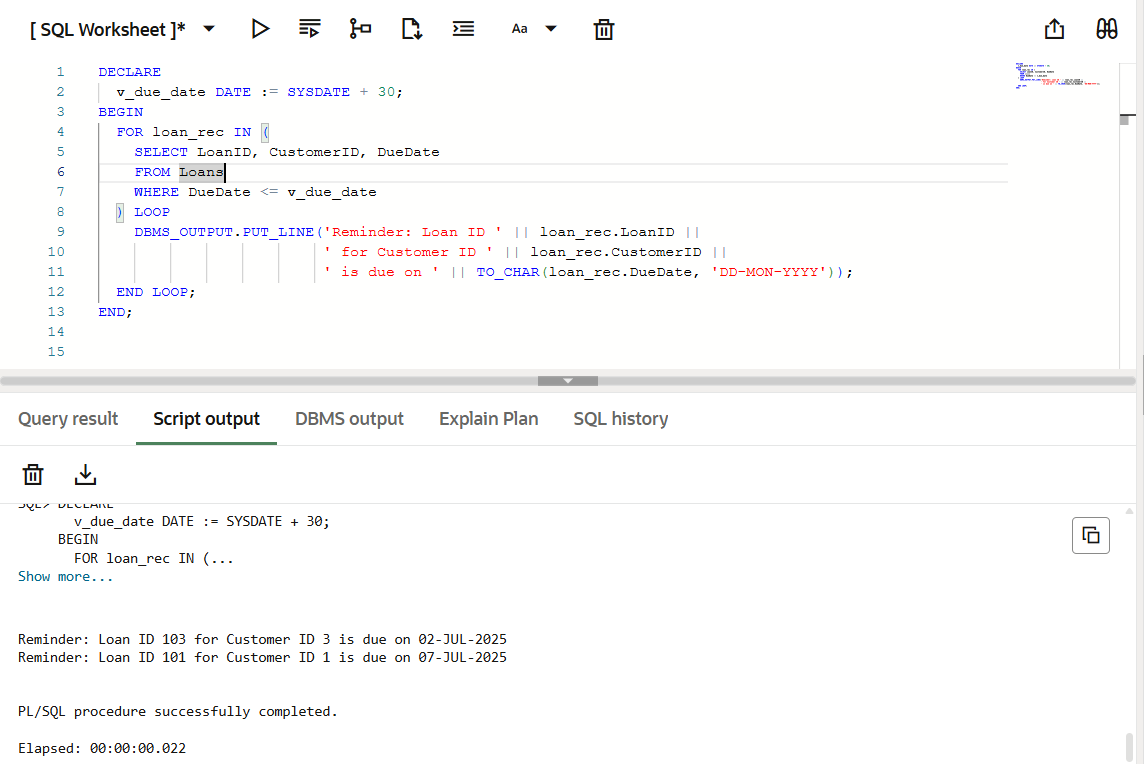
' for Customer ID ' || loan\_rec.CustomerID ||

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

END LOOP;

END;

**OUTPUT:**

****

**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

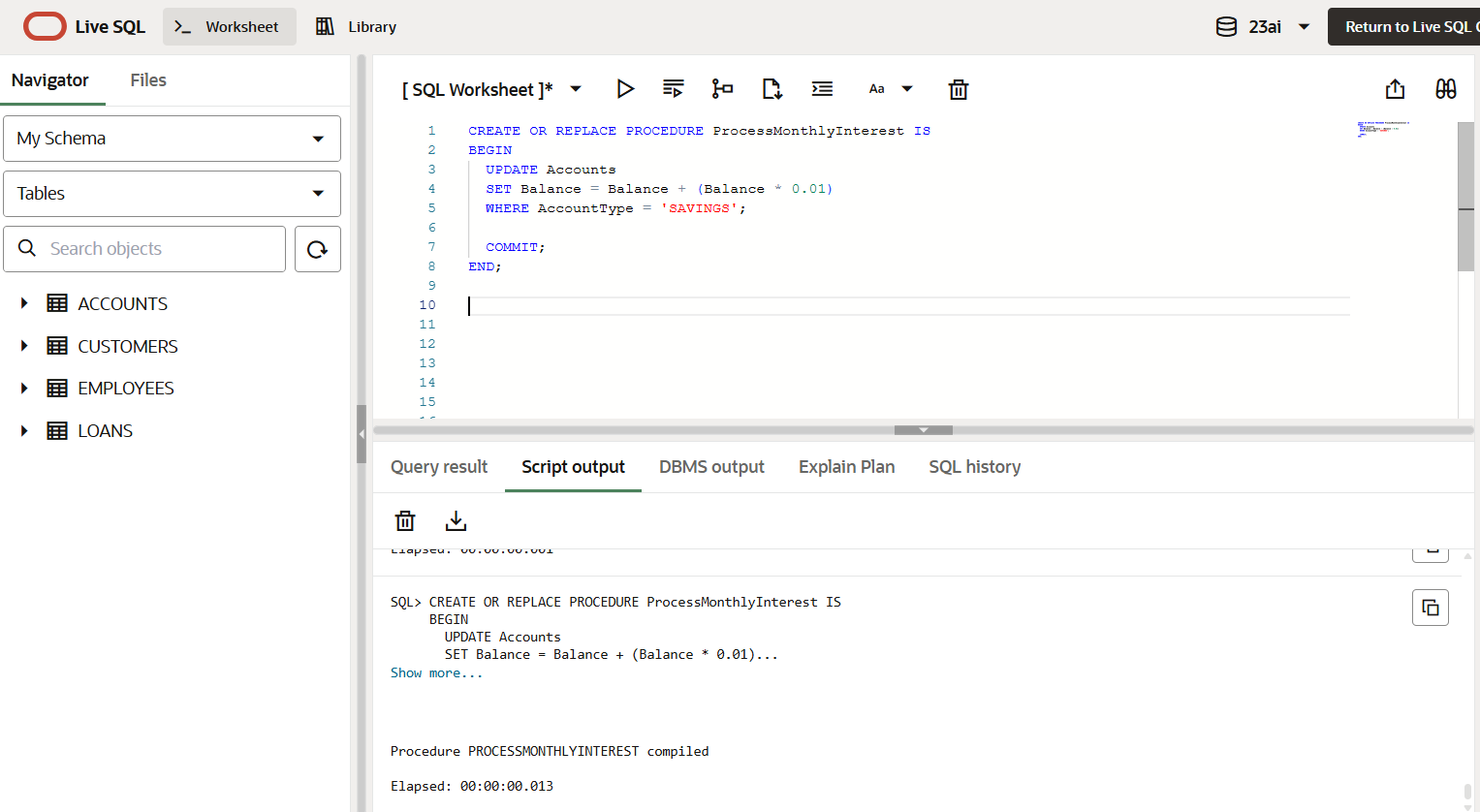
SET Balance = Balance + (Balance \* 0.01)

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\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

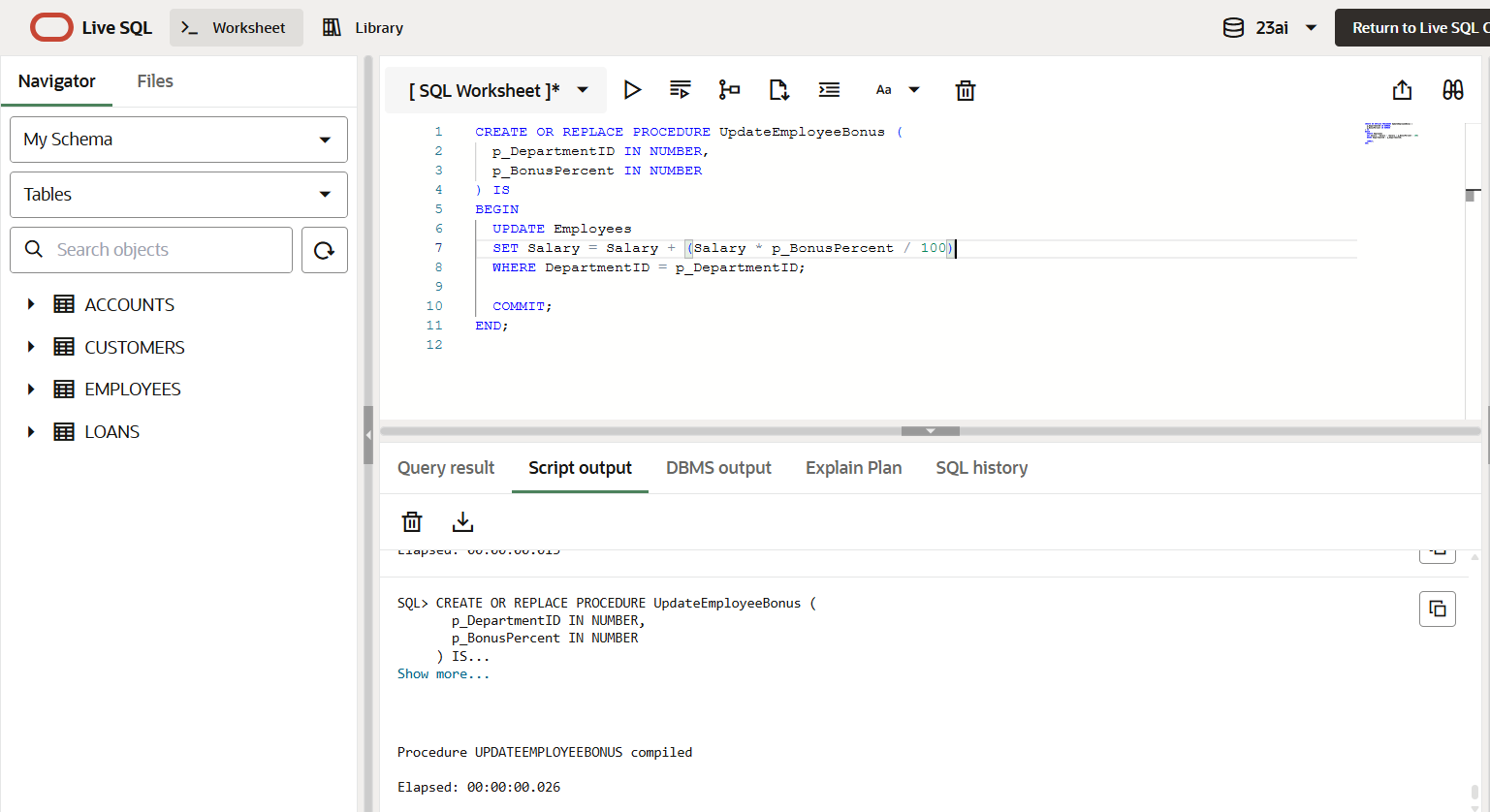
SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE DepartmentID = p\_DepartmentID;

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\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

-- Check balance of source account

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

IF v\_FromBalance < p\_Amount THEN

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

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

COMMIT;

END;

**OUTPUT :**

