**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:**

DECLARE

    CURSOR senior\_customers IS

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

        FROM Loans l

        JOIN Customers c ON l.CustomerID = c.CustomerID

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

BEGIN

    FOR rec IN senior\_customers LOOP

        UPDATE Loans

        SET InterestRate = InterestRate - 1

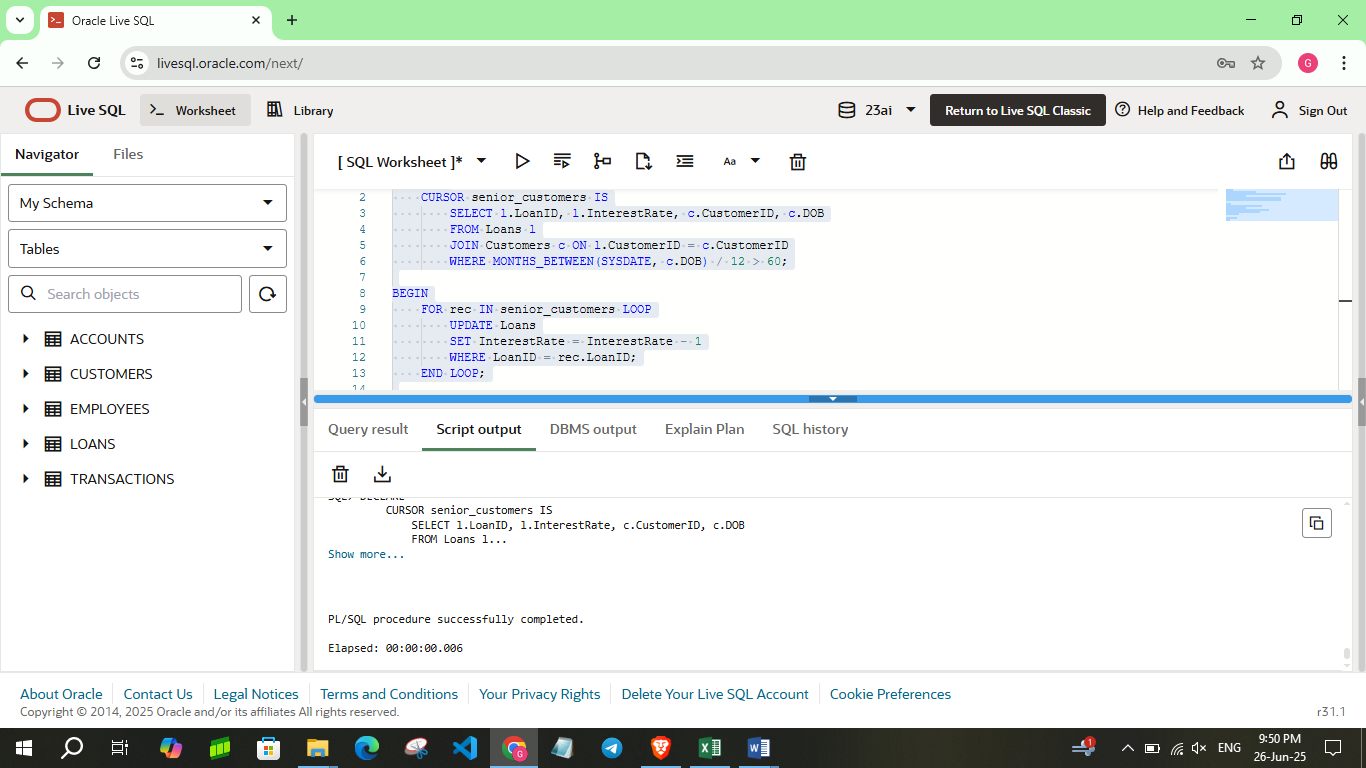
        WHERE LoanID = rec.LoanID;

    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:**

DECLARE

    CURSOR vip\_customers IS

        SELECT CustomerID, Balance

        FROM Customers

        WHERE Balance > 10000;

BEGIN

    FOR rec IN vip\_customers LOOP

        UPDATE Customers

        SET IsVIP = 'TRUE'

        WHERE CustomerID = rec.CustomerID;

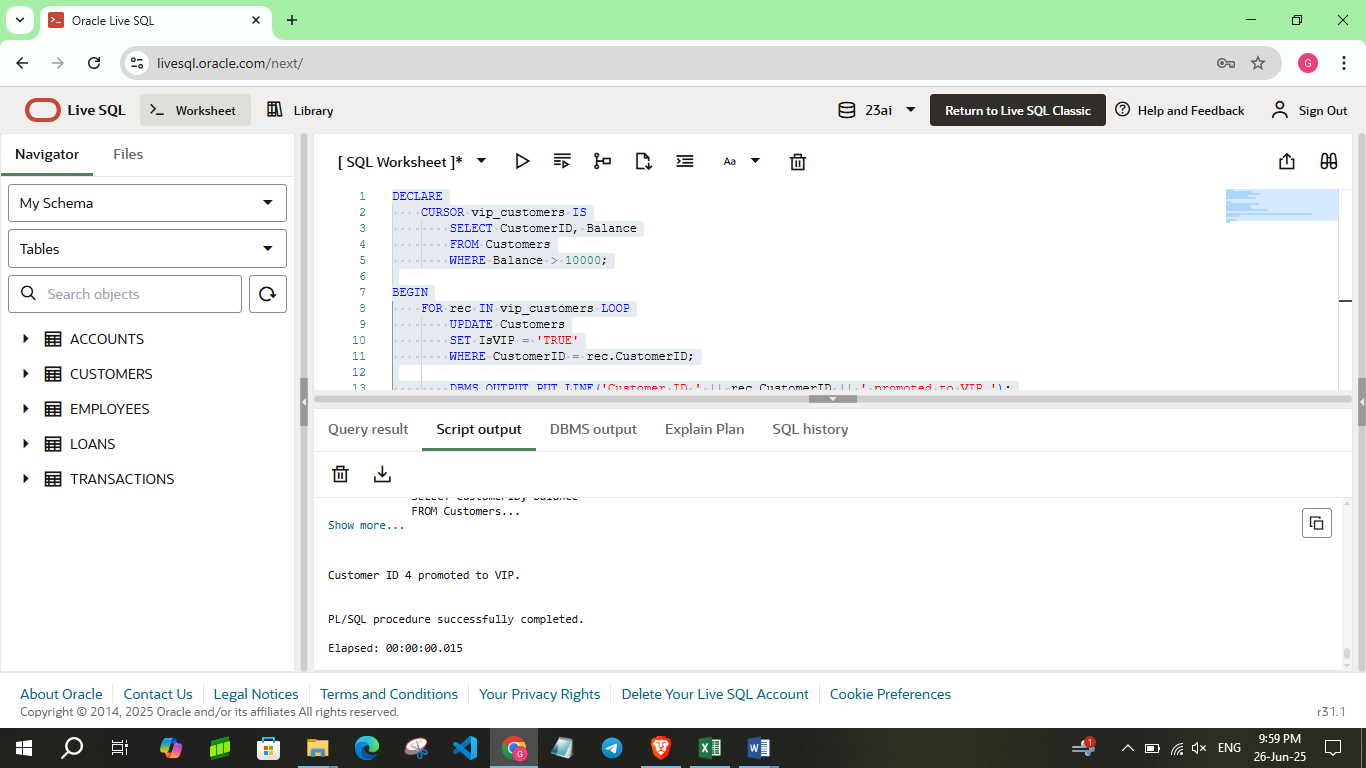
        DBMS\_OUTPUT.PUT\_LINE('Customer ID ' || rec.CustomerID || ' promoted to VIP.');

    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

    CURSOR due\_loans IS

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

        FROM Loans l

        JOIN Customers c ON l.CustomerID = c.CustomerID

        WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

    FOR rec IN due\_loans LOOP

        DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || rec.Name ||

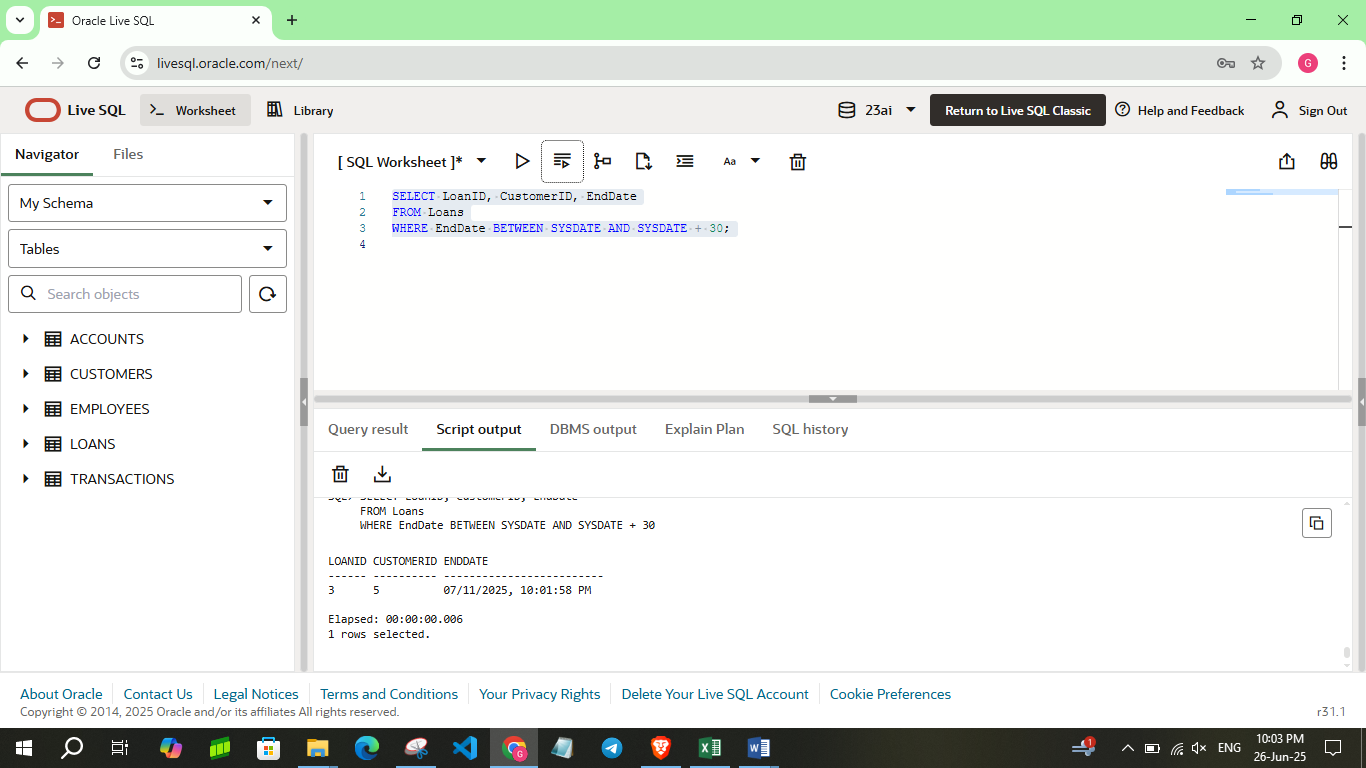
                             ', your loan (ID: ' || rec.LoanID ||

                             ') is due on ' || TO\_CHAR(rec.EndDate, '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

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    ) LOOP

        UPDATE Accounts

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

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

        DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);

    END LOOP;

    COMMIT;

END;

BEGIN

    ProcessMonthlyInterest;

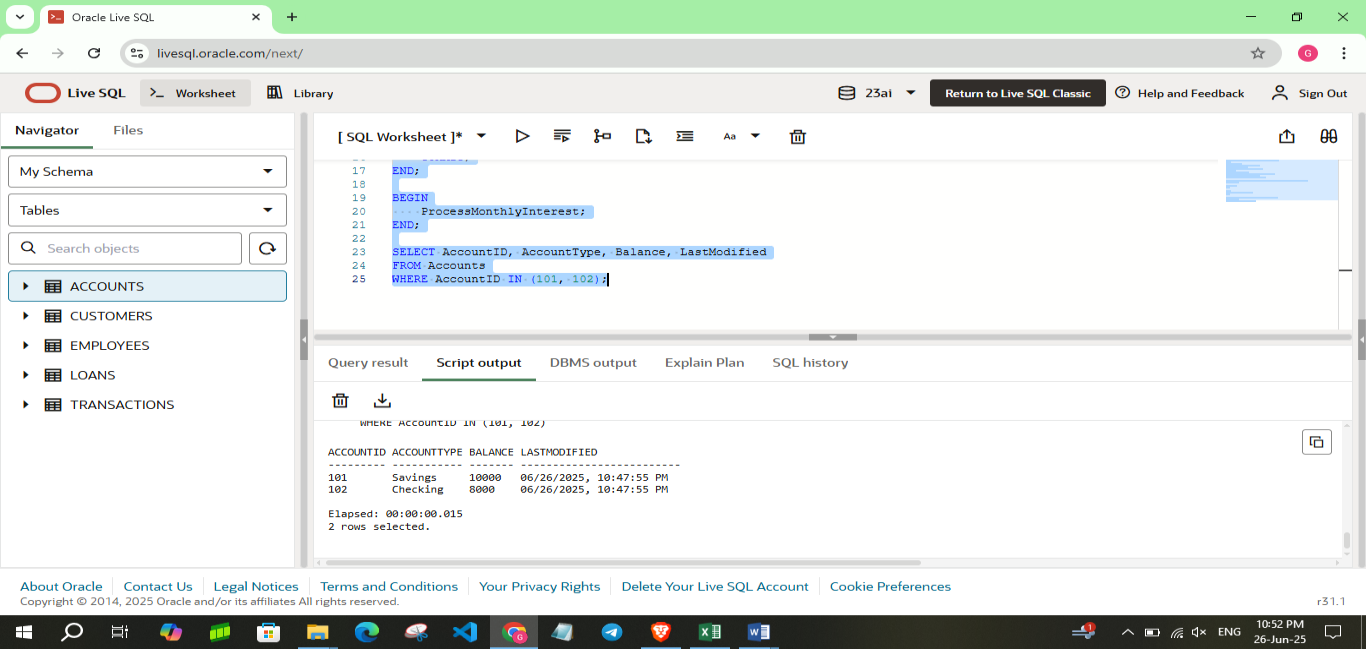
END;

SELECT AccountID, AccountType, Balance, LastModified

FROM Accounts

WHERE AccountID IN (101, 102);

**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 of ' || p\_bonus\_percent || '% applied to employees in ' || p\_department || ' department.');

    COMMIT;

END;

BEGIN

    UpdateEmployeeBonus('IT', 10);

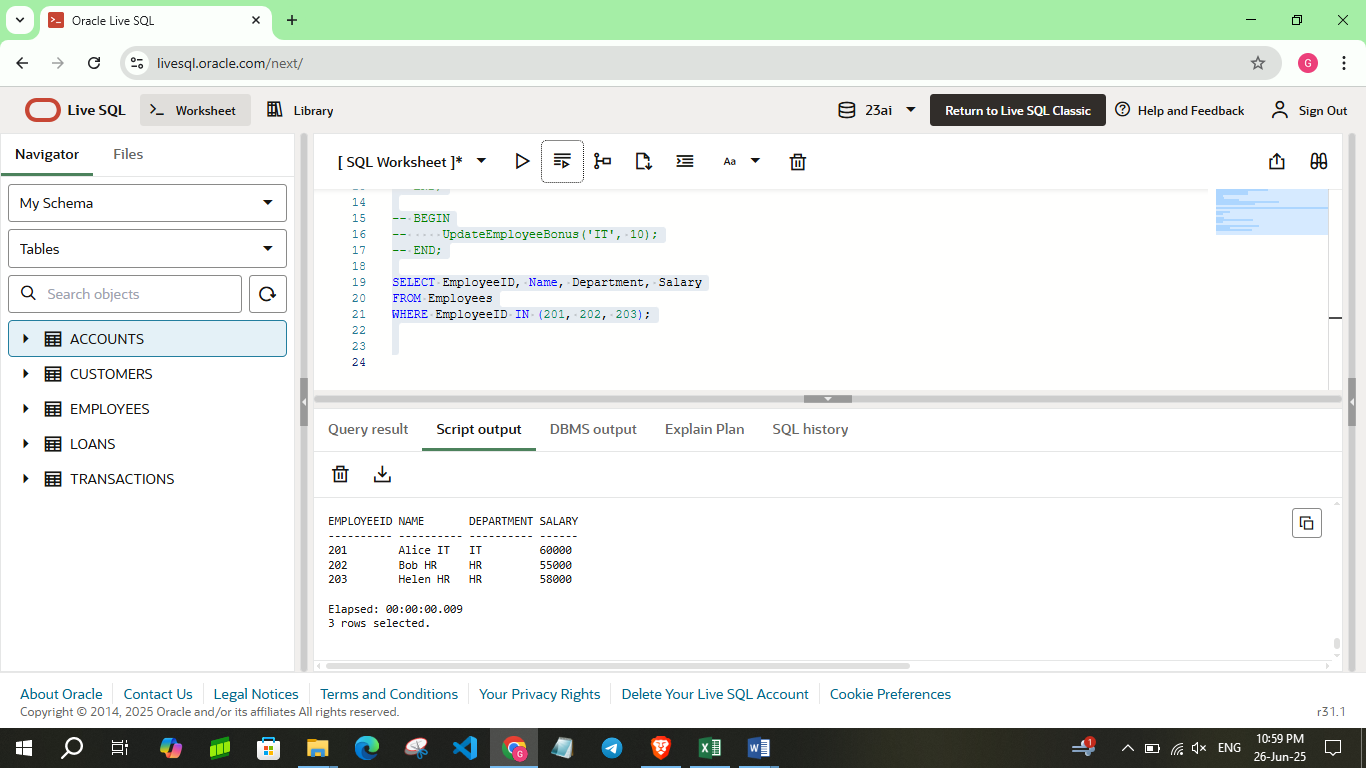
END;

SELECT EmployeeID, Name, Department, Salary

FROM Employees

WHERE EmployeeID IN (201, 202, 203);

**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 balance of source account

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account;

    IF v\_balance < p\_amount THEN

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');

    ELSE

        -- Deduct from source account

        UPDATE Accounts

        SET Balance = Balance - p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_from\_account;

        -- Add to destination account

        UPDATE Accounts

        SET Balance = Balance + p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_to\_account;

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

        COMMIT;

    END IF;

END;

BEGIN

    TransferFunds(103, 104, 1000);

END;

BEGIN

    TransferFunds(105, 104, 100);

END;

SELECT AccountID, Balance

FROM Accounts

WHERE AccountID IN (103, 104, 105);

**OUTPUT:**

