**Exercise 1: Control Structures**

**Setup.sql:**

**Code-**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Emily Davis', TO\_DATE('1988-09-10', 'YYYY-MM-DD'), 2000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Soham Ghosh', TO\_DATE('1964-12-05', 'YYYY-MM-DD'), 2500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (3, 3, 'Savings', 2000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (4, 4, 'Checking', 2500, SYSDATE);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 3, SYSDATE, 500, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 4, SYSDATE, 700, 'Withdrawal');

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (2, 2, 20000, 4.5, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (3, 4, 15000, 8, SYSDATE, ADD\_MONTHS(SYSDATE, 36));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (3, 'Carol White', 'Analyst', 50000, 'Finance', TO\_DATE('2018-11-01', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (4, 'David King', 'Support', 45000, 'Customer Service', TO\_DATE('2020-02-10', 'YYYY-MM-DD'));

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (5, 'Aritra Roy', TO\_DATE('1950-12-05', 'YYYY-MM-DD'), 12500, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (4, 5, 30000, 9, SYSDATE, ADD\_MONTHS(SYSDATE, 48));

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (6, 'Asmit Ghosh', TO\_DATE('1950-12-05', 'YYYY-MM-DD'), 20500, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (5, 6, 30000, 9, SYSDATE, SYSDATE + 20);

**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.

**Control Structures 1.sql:**

DECLARE

    CURSOR CUST\_CUR IS

        SELECT C.CUSTOMERID, C.NAME, C.DOB, L.LOANID, L.LOANAMOUNT, L.INTERESTRATE

        FROM CUSTOMERS C

        JOIN LOANS L ON L.CUSTOMERID = C.CUSTOMERID;

    age NUMBER;

    new\_interest LOANS.INTERESTRATE%TYPE;

BEGIN

    FOR CUST IN CUST\_CUR LOOP

        age := CEIL(MONTHS\_BETWEEN(SYSDATE, CUST.DOB) / 12);

        IF age > 60 THEN

            new\_interest := CUST.INTERESTRATE - 1;

            IF new\_interest < 0 THEN

                DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || CUST.CUSTOMERID || ' - Age: ' || age ||

                                     ' - Discount skipped to prevent negative interest rate.');

            ELSE

                UPDATE LOANS

                SET INTERESTRATE = new\_interest

                WHERE LOANID = CUST.LOANID;

                DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || CUST.CUSTOMERID || ' - Age: ' || age ||

                                     ' - Interest rate reduced from ' || CUST.INTERESTRATE ||

                                     '% to ' || new\_interest || '%');

            END IF;

        ELSE

            DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || CUST.CUSTOMERID || ' - Age: ' || age || ' - Not Eligible for Discount');

        END IF;

    END LOOP;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Changes Successful');

EXCEPTION

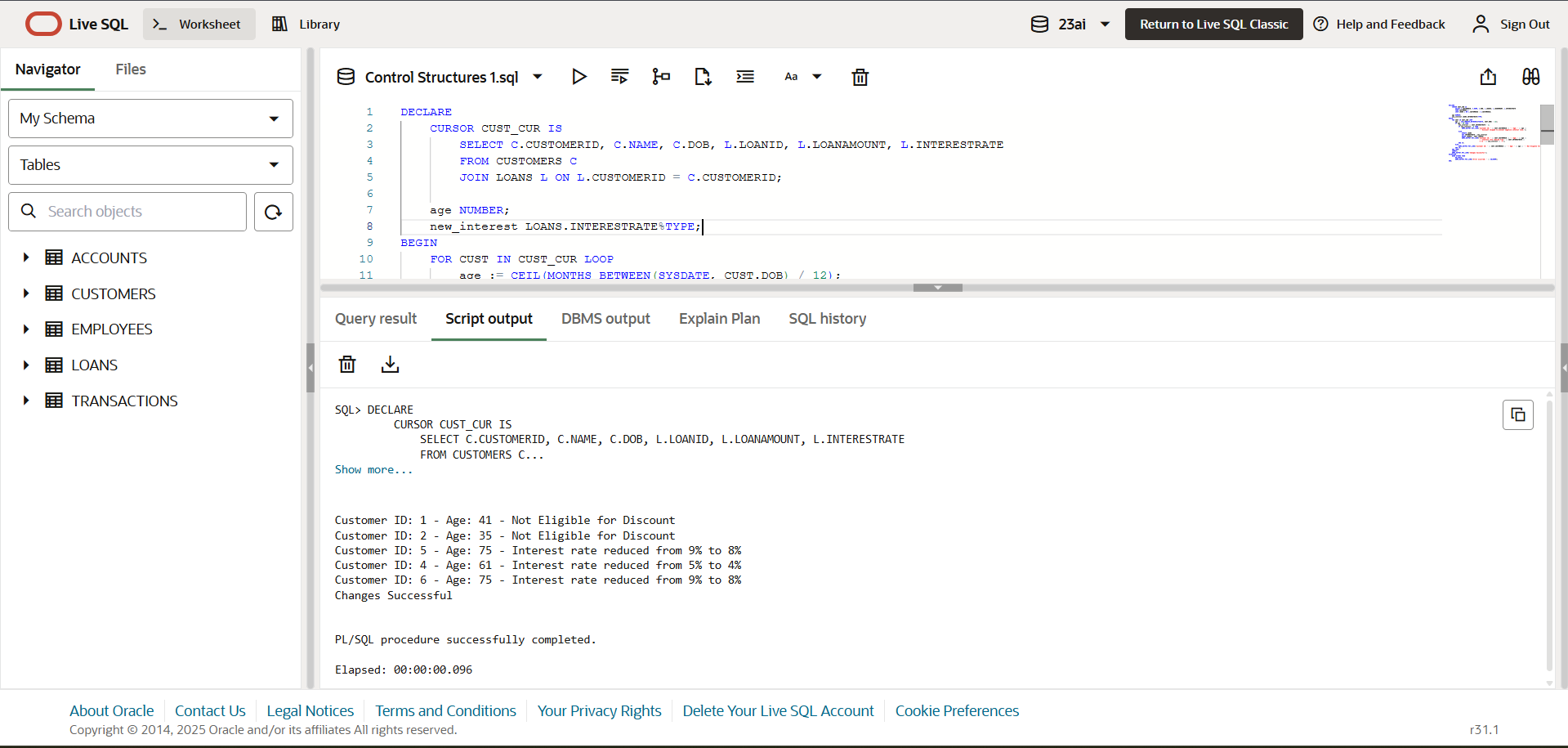
    WHEN OTHERS THEN

        ROLLBACK;

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

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.

**Control Strctures 2.sql:**

ALTER TABLE CUSTOMERS

ADD IS\_VIP BOOLEAN DEFAULT FALSE;

DECLARE

    CURSOR CUST\_CUR IS

        SELECT CUSTOMERID,NAME,BALANCE

        FROM CUSTOMERS;

    updated\_count NUMBER := 0;

BEGIN

    FOR CUST IN CUST\_CUR LOOP

        IF CUST.BALANCE > 10000 THEN

            UPDATE CUSTOMERS

            SET IS\_VIP = TRUE

            WHERE CUSTOMERID = CUST.CUSTOMERID;

            DBMS\_OUTPUT.PUT\_LINE('Customer Id:' || CUST.CUSTOMERID || '- Customer Name:' || CUST.NAME || ' got promoted');

            updated\_count := updated\_count + 1;

        END IF;

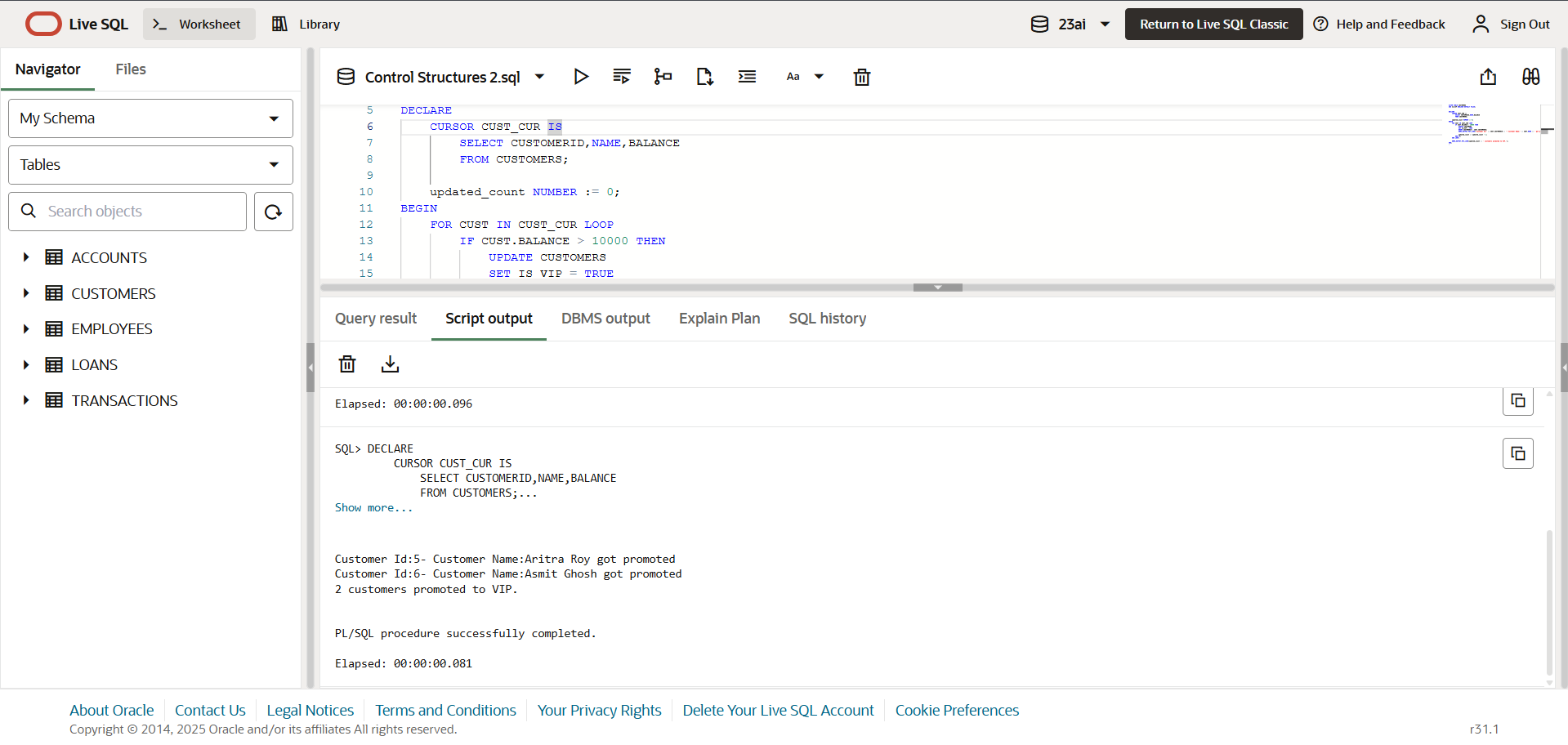
    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE(updated\_count || ' customers promoted to VIP.');

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.

**Control Structures 3.sql:**

DECLARE

   CURSOR LOAN\_CUR IS

      SELECT l.LOANID, l.CUSTOMERID, l.LOANAMOUNT, 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\_DUE IN LOAN\_CUR LOOP

      DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || LOAN\_DUE.NAME ||

                           ', your loan (ID: ' || LOAN\_DUE.LOANID ||

                           ') of amount $' || LOAN\_DUE.LOANAMOUNT ||

                           ' is due on ' || TO\_CHAR(LOAN\_DUE.ENDDATE, 'DD-MON-YYYY') ||

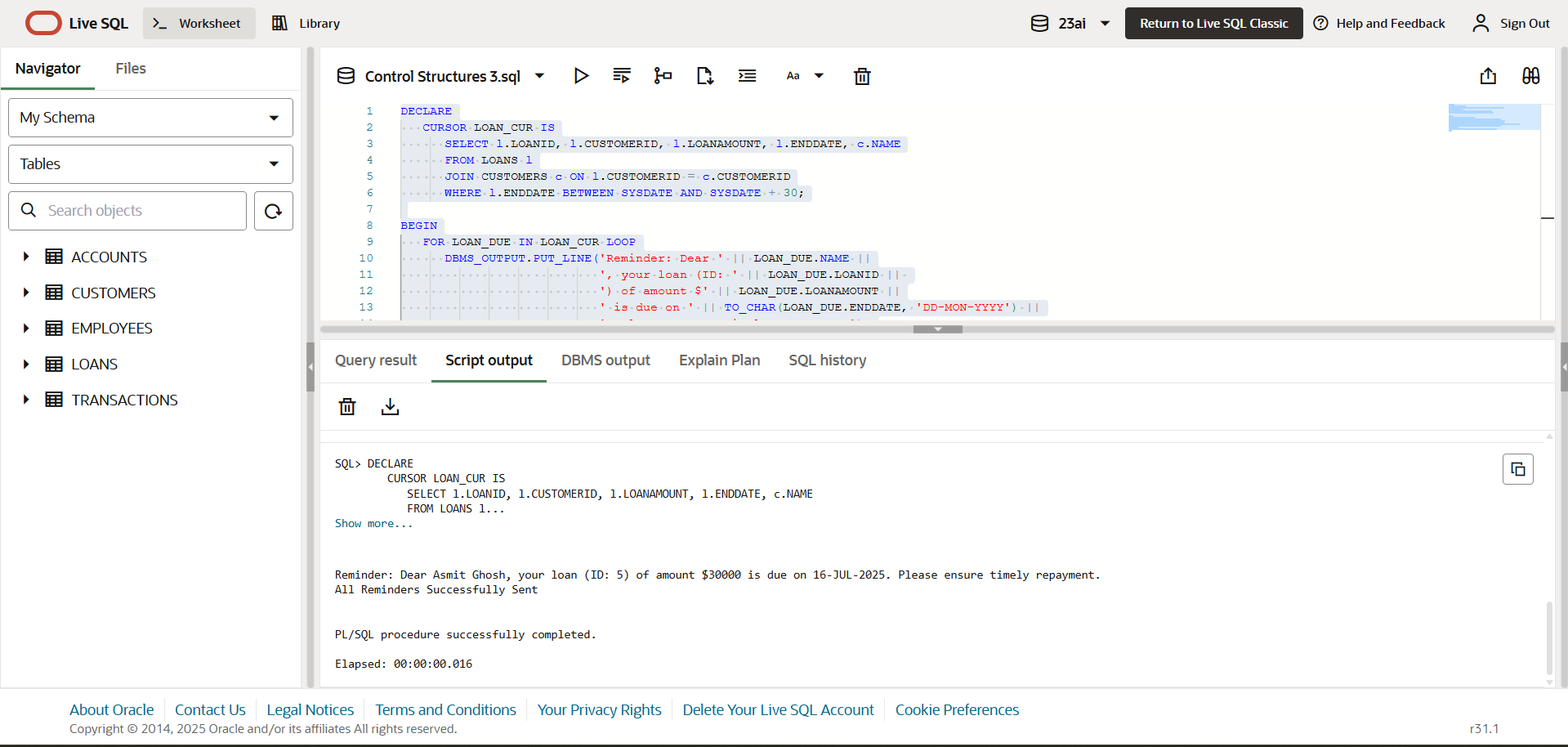
                           '. Please ensure timely repayment.');

   END LOOP;

   DBMS\_OUTPUT.PUT\_LINE('All Reminders Successfully Sent');

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.

**Stored Procedures 1.sql:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

   UPDATE ACCOUNTS

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

       LASTMODIFIED = SYSDATE

   WHERE UPPER(ACCOUNTTYPE) = 'SAVINGS';

   COMMIT;

   DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to Savings accounts.');

   DBMS\_OUTPUT.PUT\_LINE('Updated Savings Account Details:');

   FOR acc IN (

       SELECT ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED

       FROM ACCOUNTS

       WHERE UPPER(ACCOUNTTYPE) = 'SAVINGS'

   ) LOOP

       DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || acc.ACCOUNTID ||

                            ', CustomerID: ' || acc.CUSTOMERID ||

                            ', Balance: $' || TO\_CHAR(acc.BALANCE, '9999.99') ||

                            ', LastModified: ' || TO\_CHAR(acc.LASTMODIFIED, 'DD-MON-YYYY HH24:MI:SS'));

   END LOOP;

EXCEPTION

   WHEN OTHERS THEN

      ROLLBACK;

      DBMS\_OUTPUT.PUT\_LINE('Error processing interest: ' || SQLERRM);

END;

/

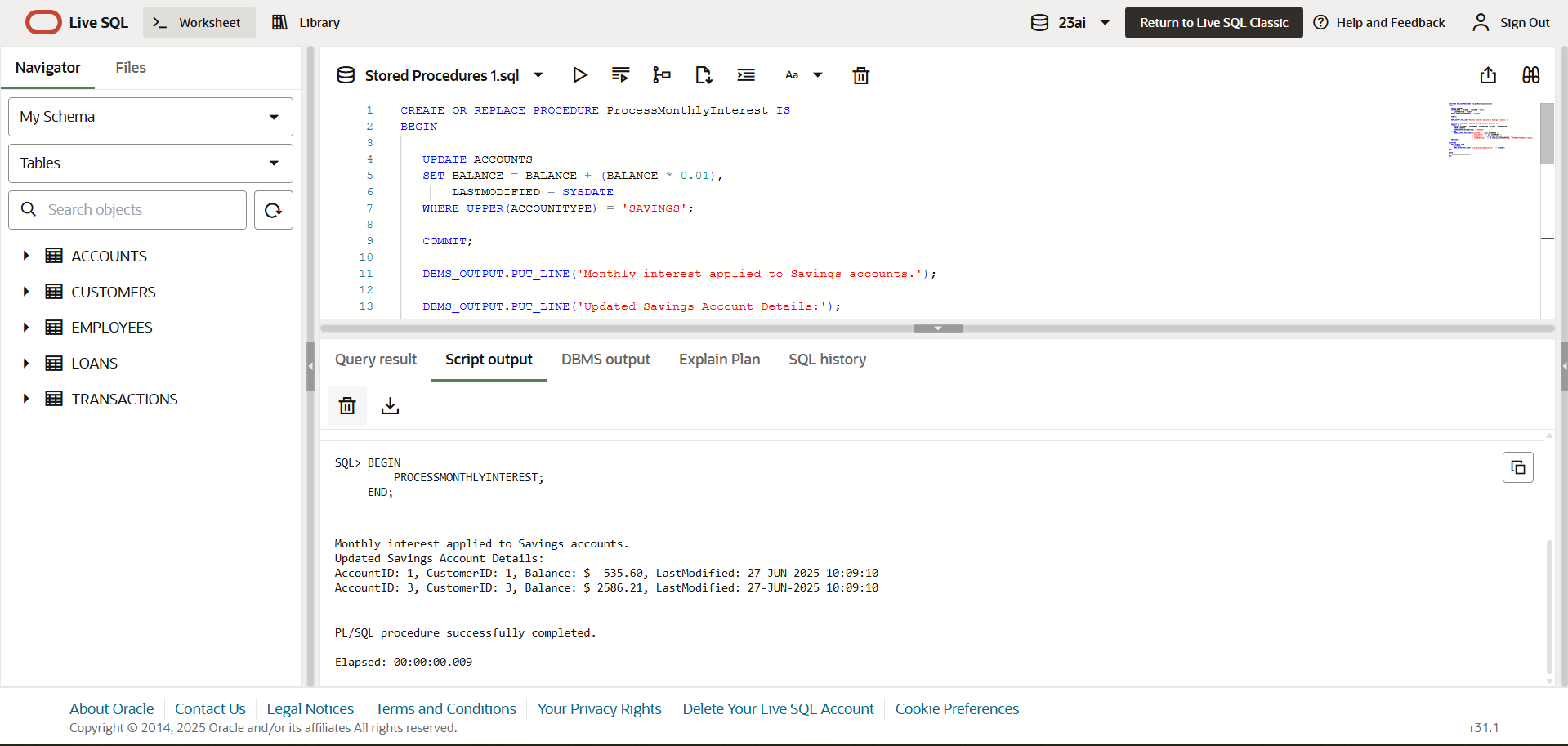
BEGIN

    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.

**Stored Procedures 2.sql:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

   p\_department      IN EMPLOYEES.DEPARTMENT%TYPE,

   p\_bonus\_percent   IN NUMBER

) IS

BEGIN

   UPDATE EMPLOYEES

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

   WHERE UPPER(DEPARTMENT) = UPPER(p\_department);

   COMMIT;

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

   FOR emp IN (

      SELECT EMPLOYEEID, NAME, SALARY

      FROM EMPLOYEES

      WHERE UPPER(DEPARTMENT) = UPPER(p\_department)

   ) LOOP

      DBMS\_OUTPUT.PUT\_LINE('EmployeeID: ' || emp.EMPLOYEEID ||

                           ', Name: ' || emp.NAME ||

                           ', New Salary: ' || TO\_CHAR(emp.SALARY, '99999.99'));

   END LOOP;

EXCEPTION

   WHEN OTHERS THEN

      ROLLBACK;

      DBMS\_OUTPUT.PUT\_LINE('Error applying bonus: ' || SQLERRM);

END;

/

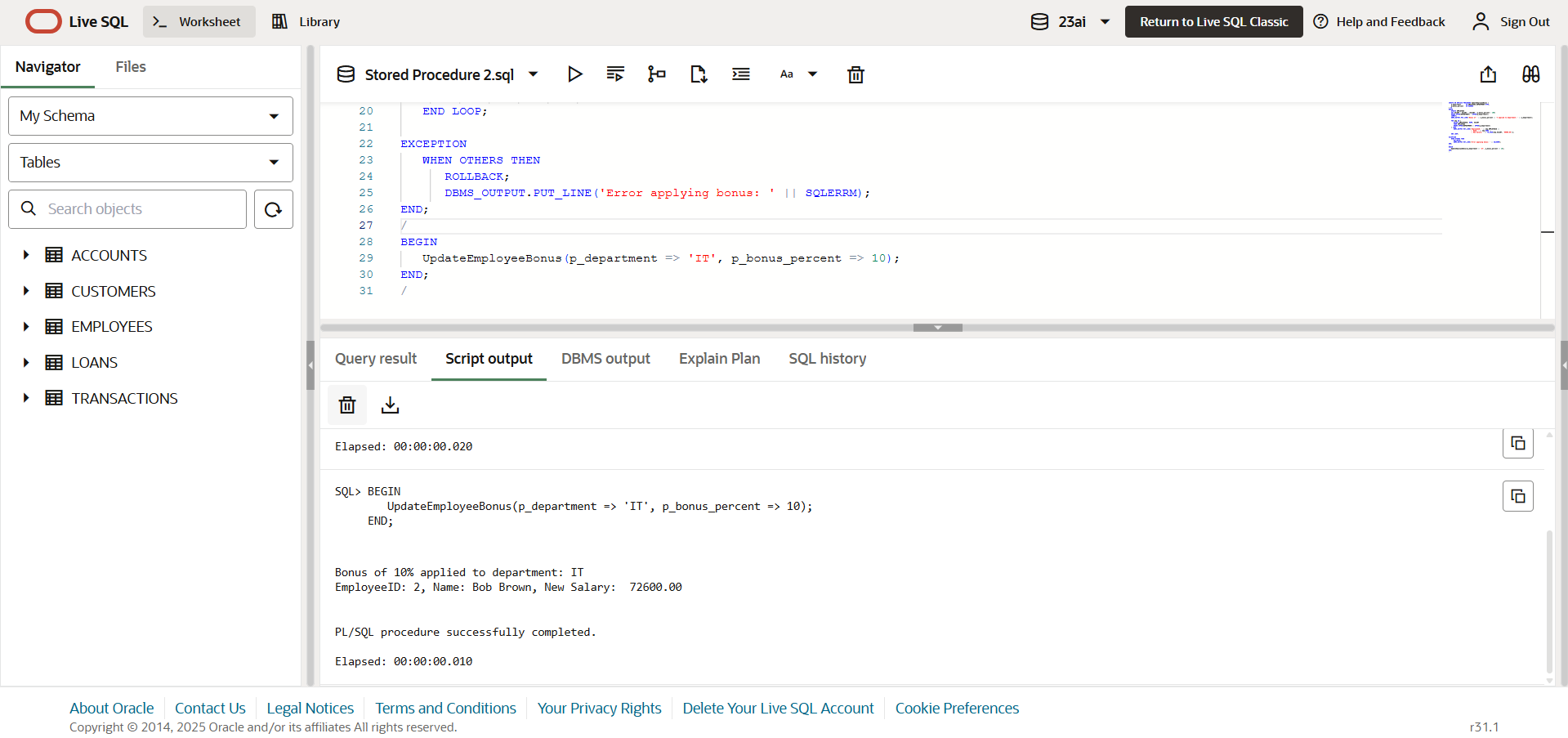
BEGIN

   UpdateEmployeeBonus(p\_department => 'IT', p\_bonus\_percent => 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.

**Stored Procedures 3.sql:**

CREATE OR REPLACE PROCEDURE TransferFunds (

   p\_from\_account\_id IN ACCOUNTS.ACCOUNTID%TYPE,

   p\_to\_account\_id   IN ACCOUNTS.ACCOUNTID%TYPE,

   p\_amount          IN NUMBER

) IS

   v\_from\_balance ACCOUNTS.BALANCE%TYPE;

BEGIN

   SELECT BALANCE INTO v\_from\_balance

   FROM ACCOUNTS

   WHERE ACCOUNTID = p\_from\_account\_id

   FOR UPDATE;

   IF v\_from\_balance < p\_amount THEN

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

      RETURN;

   END IF;

   UPDATE ACCOUNTS

   SET BALANCE = BALANCE - p\_amount,

       LASTMODIFIED = SYSDATE

   WHERE ACCOUNTID = p\_from\_account\_id;

   UPDATE ACCOUNTS

   SET BALANCE = BALANCE + p\_amount,

       LASTMODIFIED = SYSDATE

   WHERE ACCOUNTID = p\_to\_account\_id;

   COMMIT;

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

   DBMS\_OUTPUT.PUT\_LINE('Updated account details:');

   FOR acc IN (

      SELECT ACCOUNTID, CUSTOMERID, ACCOUNTTYPE, BALANCE, LASTMODIFIED

      FROM ACCOUNTS

      WHERE ACCOUNTID = p\_from\_account\_id OR ACCOUNTID = p\_to\_account\_id

   ) LOOP

      DBMS\_OUTPUT.PUT\_LINE('AccountID: ' || acc.ACCOUNTID ||

                           ', CustomerID: ' || acc.CUSTOMERID ||

                           ', Type: ' || acc.ACCOUNTTYPE ||

                           ', Balance: $' || TO\_CHAR(acc.BALANCE, '999999.99') ||

                           ', LastModified: ' || TO\_CHAR(acc.LASTMODIFIED, 'DD-MON-YYYY HH24:MI:SS'));

   END LOOP;

EXCEPTION

   WHEN OTHERS THEN

      ROLLBACK;

      DBMS\_OUTPUT.PUT\_LINE('Transfer failed due to an error.');

END;

/

BEGIN

   TransferFunds(3, 1, 500);

END;

/

**OUTPUT:**

