**Cognizant Deep Skilling - Digital Nurture 4.0**

**Exercise 1: Control Structures**

**Scenario 1:**

The bank wants to apply a discount to loan interest rates for customers above 60 years old.

**SQL query:**

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

InterestRate NUMBER

);

INSERT INTO Customers VALUES (1, 'Ravi', 65, 0.08);

INSERT INTO Customers VALUES (2, 'Meena', 45, 0.07);

INSERT INTO Customers VALUES (3, 'Anil', 70, 0.09);

INSERT INTO Customers VALUES (4, 'Divya', 59, 0.08);

COMMIT;

SELECT \* FROM Customers;

BEGIN

FOR cust IN (SELECT CustomerID, Age, InterestRate FROM Customers) LOOP

IF cust.Age > 60 THEN

UPDATE Customers

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = cust.CustomerID;

END IF;

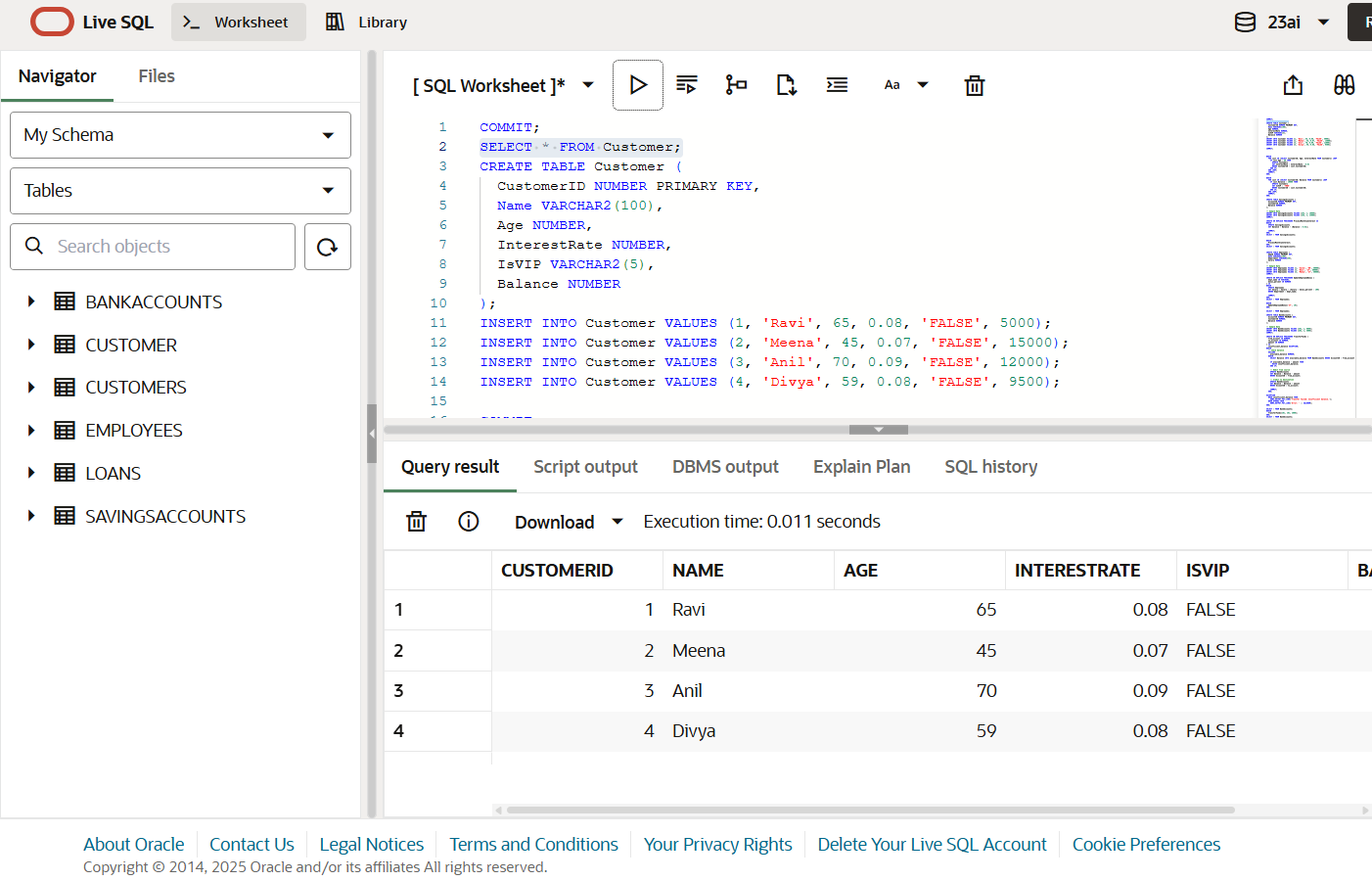
END LOOP;

COMMIT;

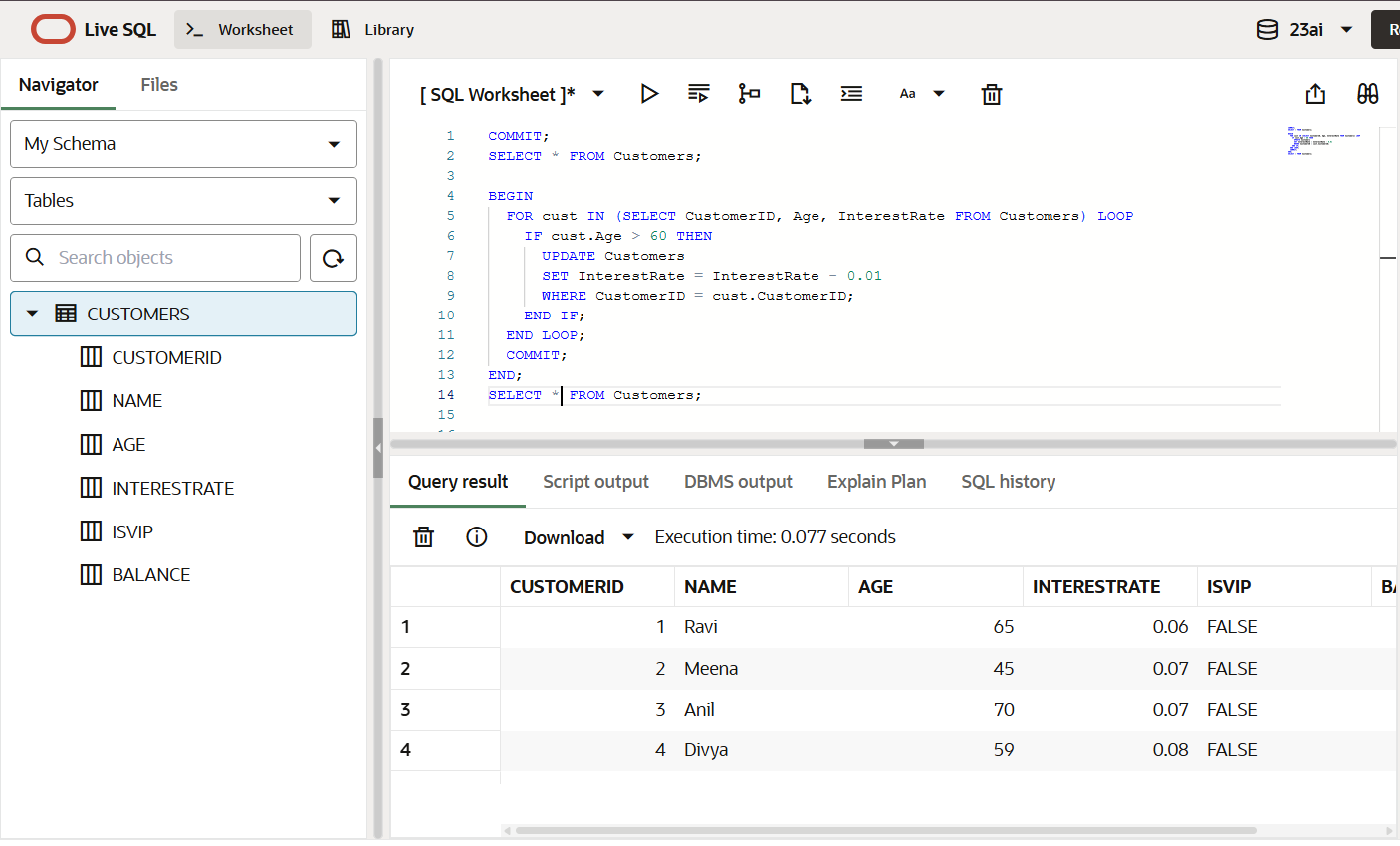
END;

SELECT \* FROM Customers;

**Table creation:**

****

**Output:**



**Scenario 2:**

A customer can be promoted to VIP status based on their balance.

**SQL query:**

BEGIN

  FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF cust.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = cust.CustomerID;

    END IF;

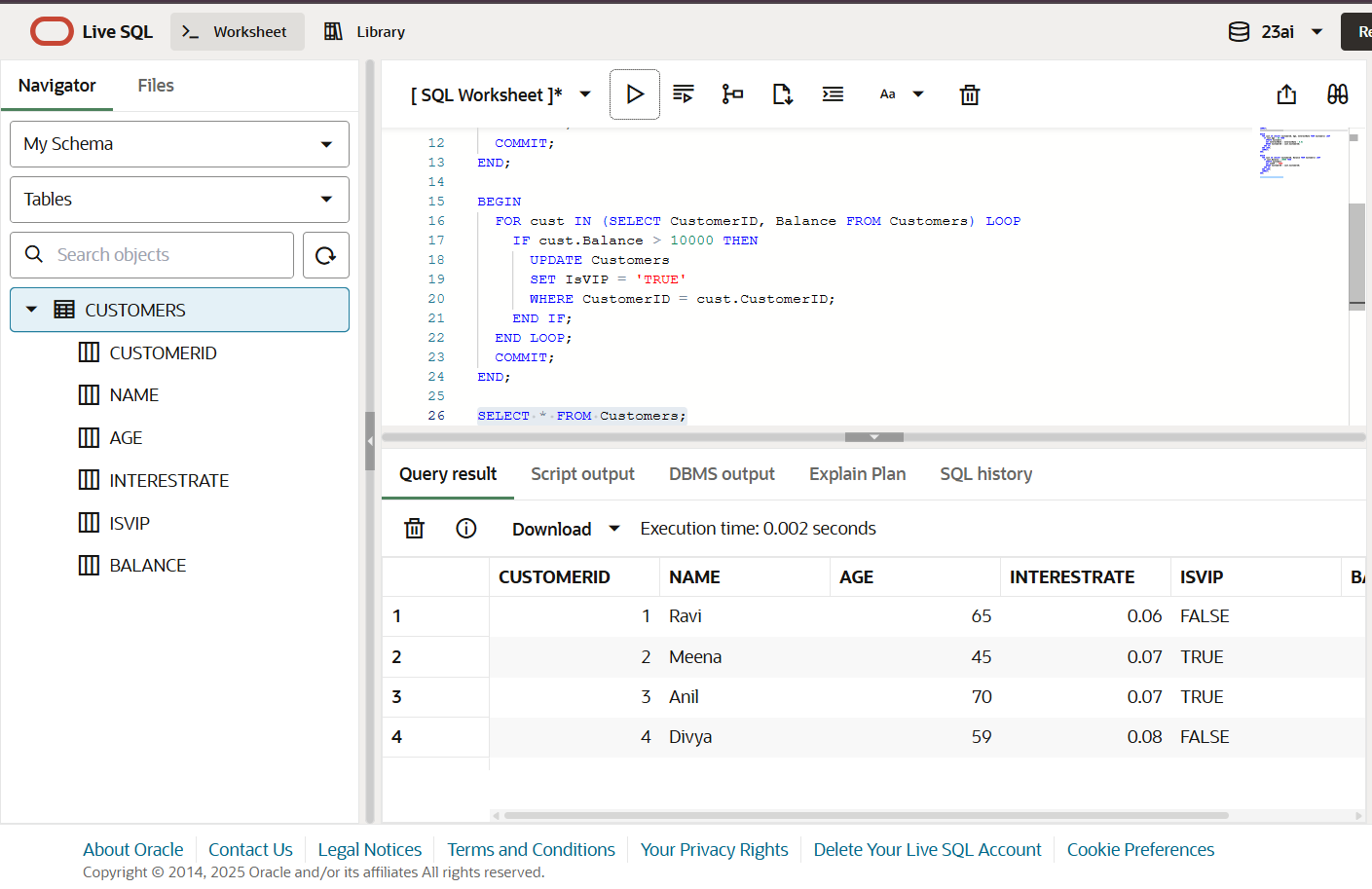
  END LOOP;

  COMMIT;

END;

SELECT \* FROM Customers;

**Output:**



**Scenario 3:**

The bank wants to send reminders to customers whose loans are due within the next 30 days.

**SQL query:**

CREATE TABLE Loans (

  LoanID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  DueDate DATE,

  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10); -- Due soon

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40);

COMMIT;

BEGIN

  FOR loan IN (

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

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.DueDate <= SYSDATE + 30

  ) LOOP

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

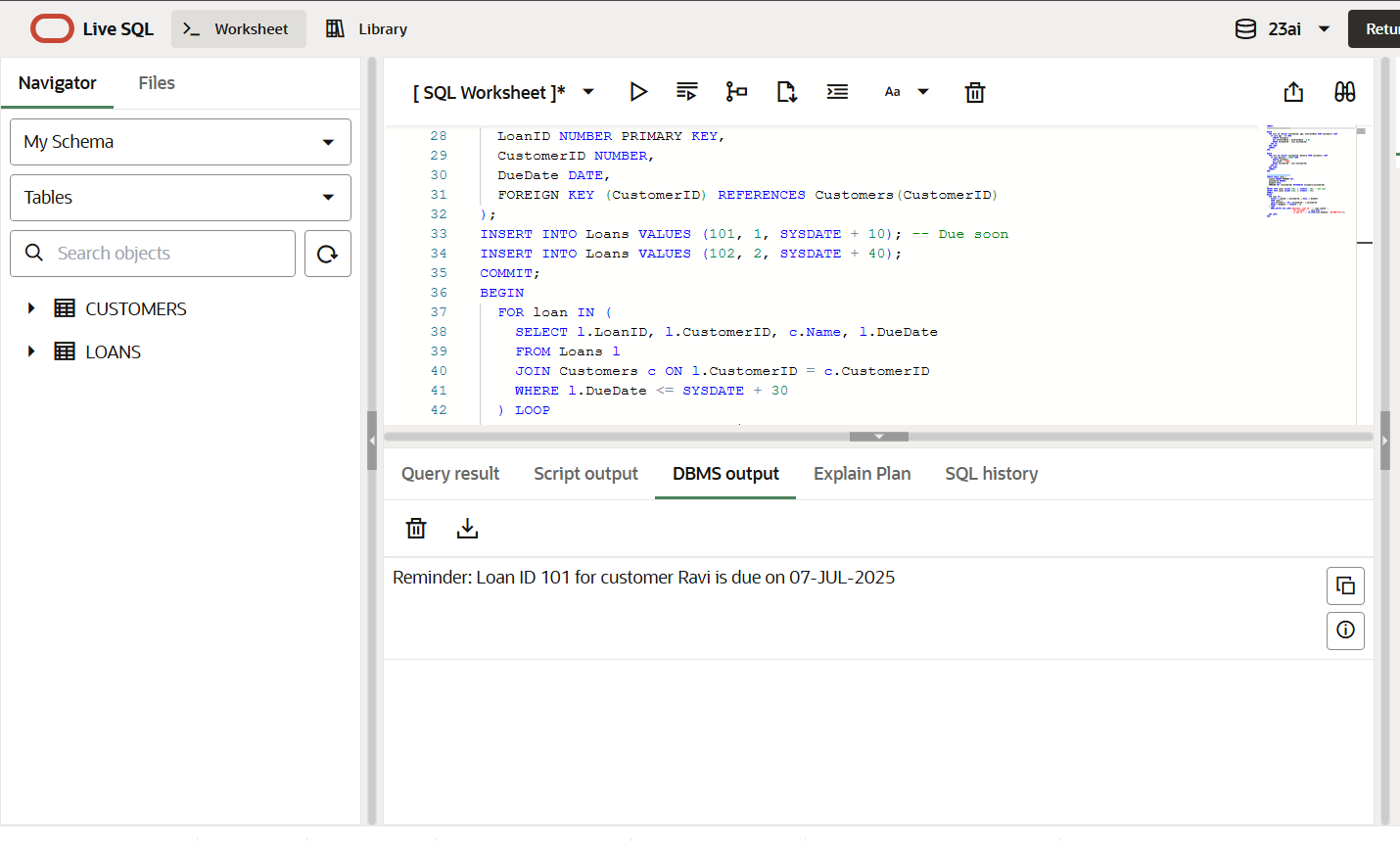
                         ' for customer ' || loan.Name ||

                         ' is due on ' || TO\_CHAR(loan.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.

**SQL query:**

CREATE TABLE SavingsAccounts (

  AccountID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  Balance NUMBER

);

INSERT INTO SavingsAccounts VALUES (101, 1, 10000);

INSERT INTO SavingsAccounts VALUES (102, 2, 15000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

  UPDATE SavingsAccounts

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

COMMIT;

END;

SELECT \* FROM SavingsAccounts;

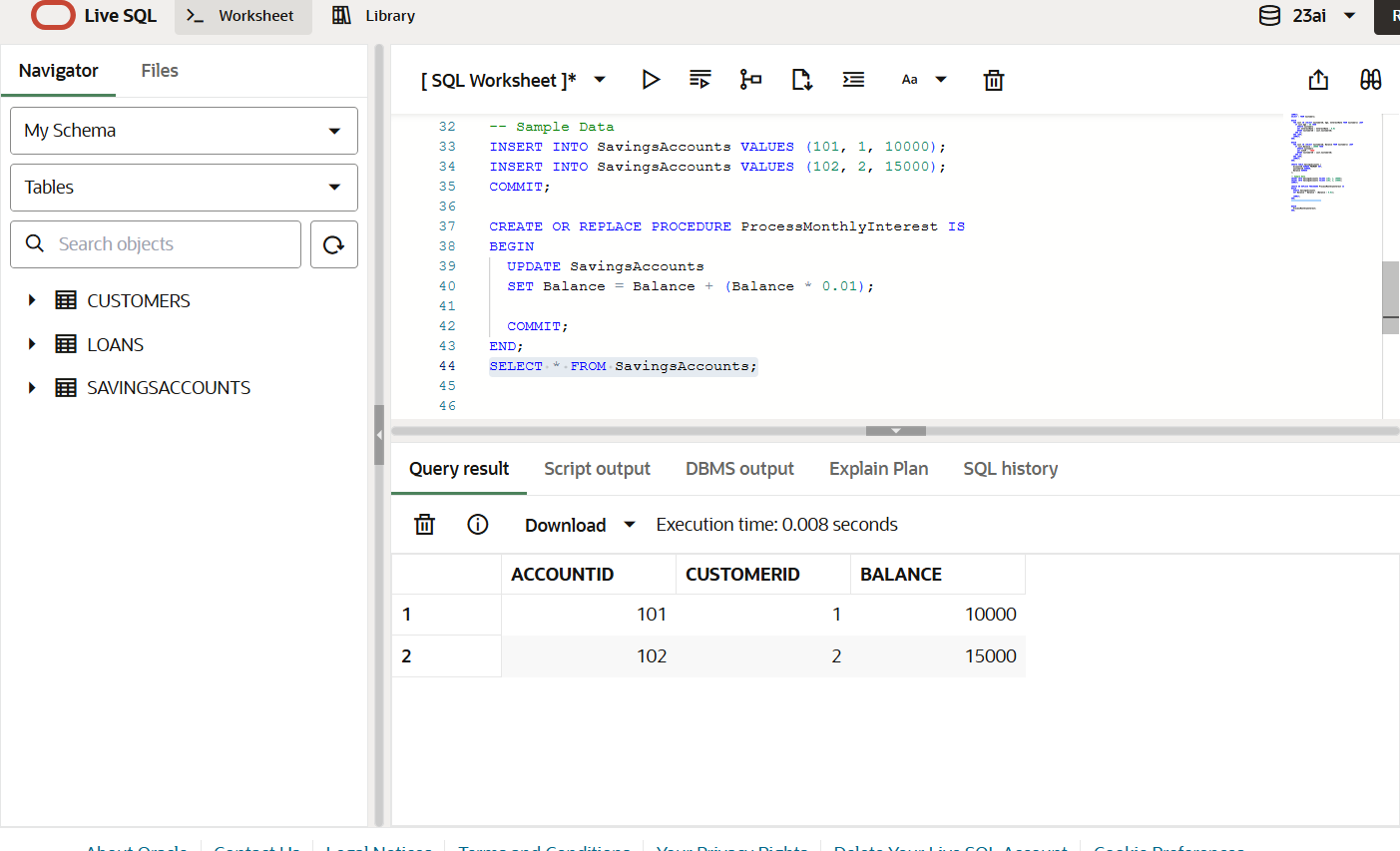
BEGIN

  ProcessMonthlyInterest;

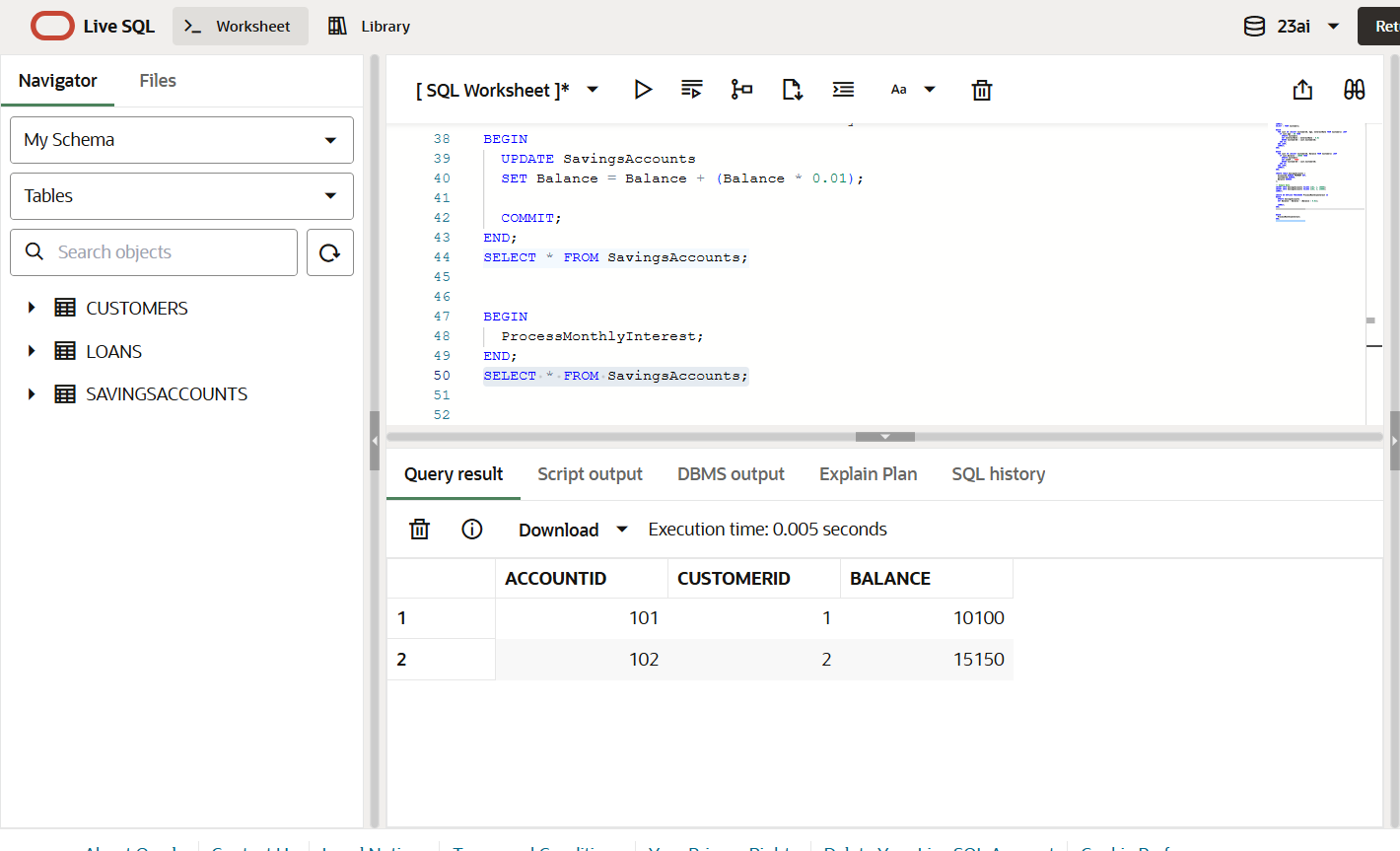
END;

SELECT \* FROM SavingsAccounts;

**Table creation:**



**Output:**



**Scenario 2:**

The bank wants to implement a bonus scheme for employees based on their performance.

**SQL query:**

CREATE TABLE Employees (

  EmpID NUMBER PRIMARY KEY,

  Name VARCHAR2(100),

  Department VARCHAR2(50),

  Salary NUMBER

);

-- Sample Data

INSERT INTO Employees VALUES (1, 'Anita', 'HR', 40000);

INSERT INTO Employees VALUES (2, 'Ravi', 'IT', 60000);

INSERT INTO Employees VALUES (3, 'Meena', 'IT', 50000);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  dept\_name IN VARCHAR2,

  bonus\_percent IN NUMBER

) IS

BEGIN

  UPDATE Employees

  SET Salary = Salary + (Salary \* bonus\_percent / 100)

  WHERE Department = dept\_name;

COMMIT;

END;

SELECT \* FROM Employees;

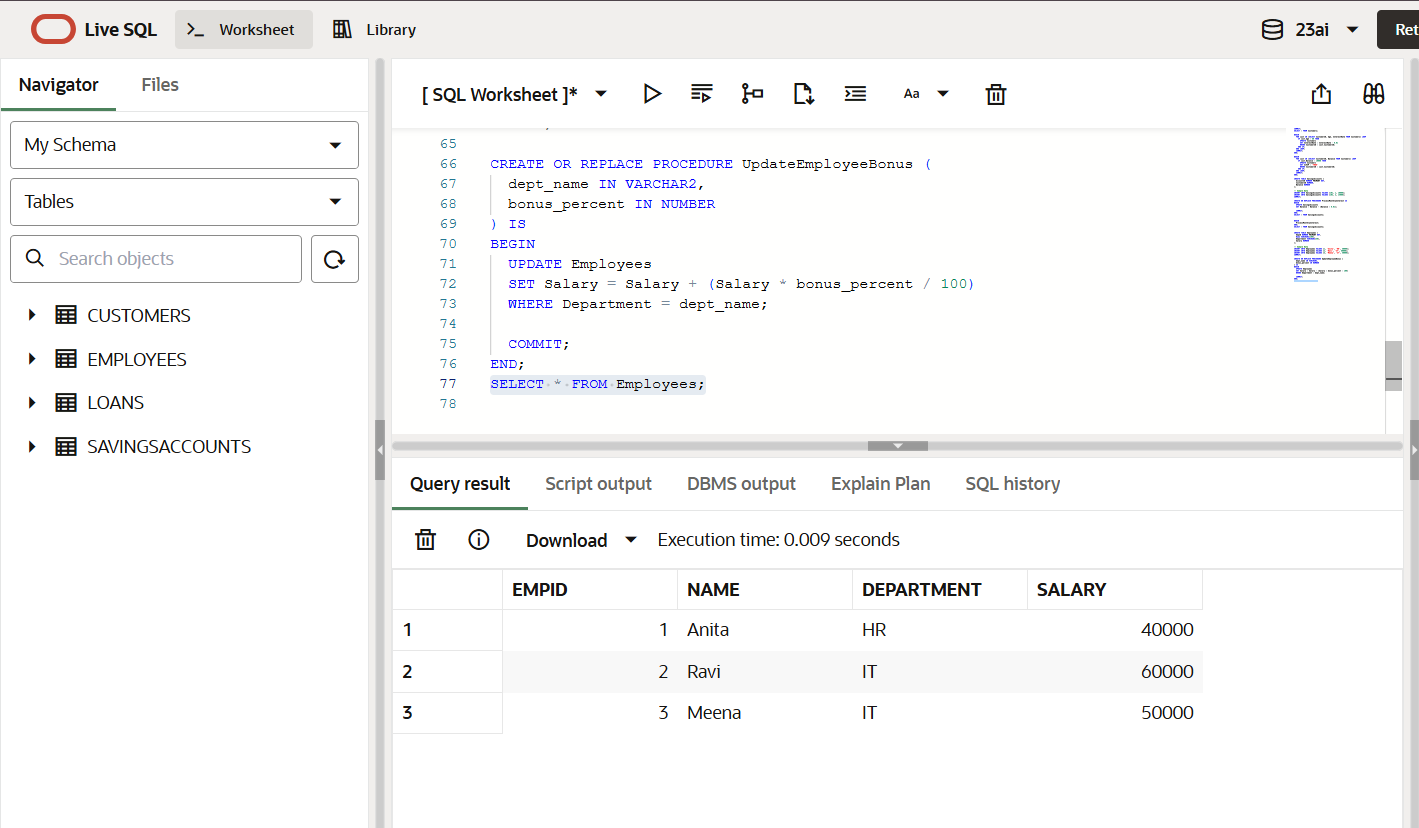
BEGIN

  UpdateEmployeeBonus('IT', 10);

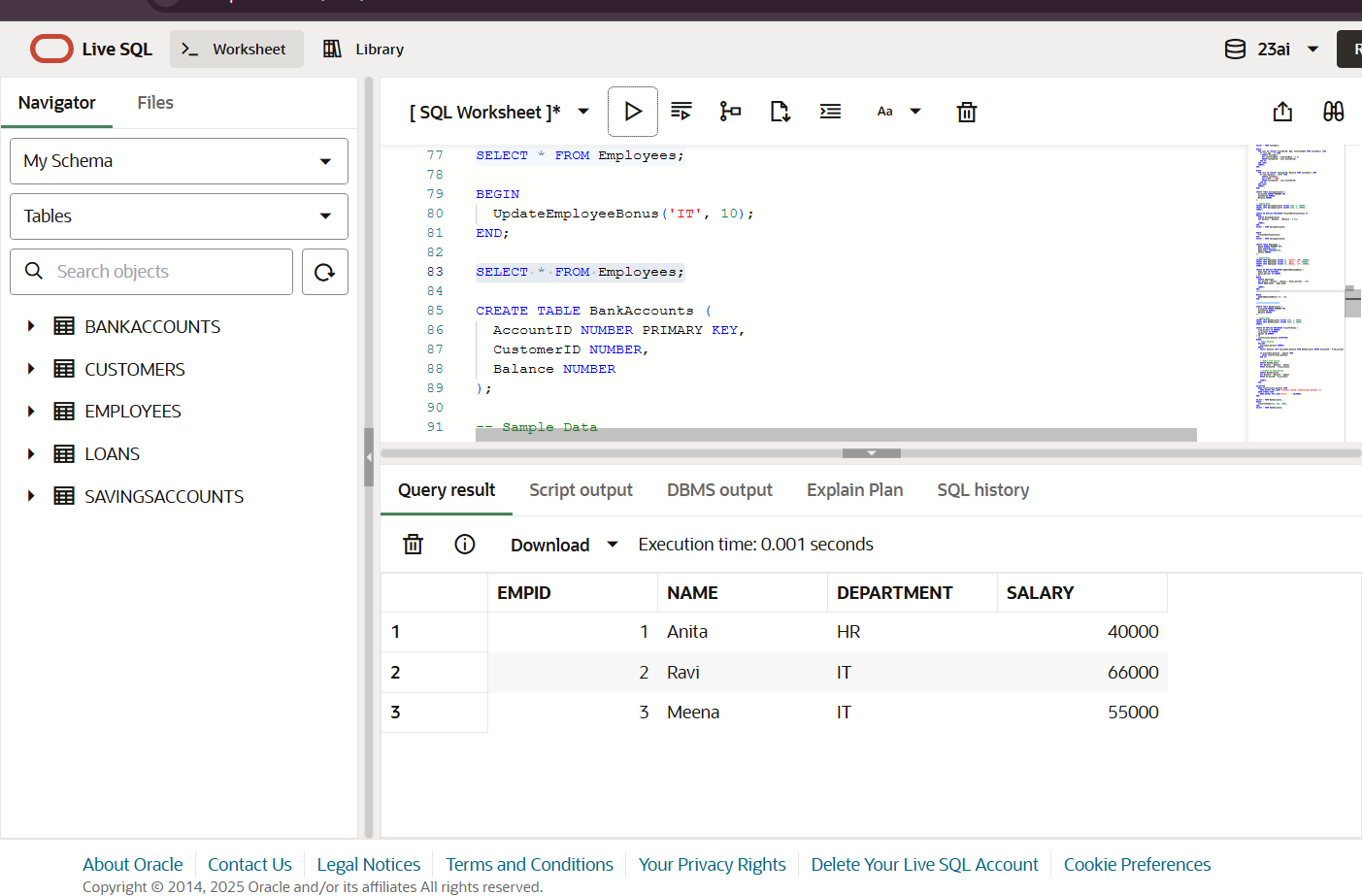
END;

SELECT \* FROM Employees;

**Table creation:**



**Output:**



**Scenario 3:**

Customers should be able to transfer funds between their accounts.

**SQL query:**

CREATE TABLE BankAccounts (

  AccountID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  Balance NUMBER

);

-- Sample Data

INSERT INTO BankAccounts VALUES (201, 1, 8000);

INSERT INTO BankAccounts VALUES (202, 2, 3000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

  from\_account IN NUMBER,

  to\_account IN NUMBER,

  amount IN NUMBER

) IS

  insufficient\_balance EXCEPTION;

BEGIN

  -- Check balance

  DECLARE

    available\_balance NUMBER;

  BEGIN

    SELECT Balance INTO available\_balance FROM BankAccounts WHERE AccountID = from\_account;

   IF available\_balance < amount THEN

      RAISE insufficient\_balance;

END IF;

-- Debit from source

    UPDATE BankAccounts

    SET Balance = Balance - amount

    WHERE AccountID = from\_account;

-- Credit to destination

    UPDATE BankAccounts

    SET Balance = Balance + amount

    WHERE AccountID = to\_account;

COMMIT;

  END;

EXCEPTION

  WHEN insufficient\_balance THEN

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

  WHEN OTHERS THEN

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

END;

SELECT \* FROM BankAccounts;

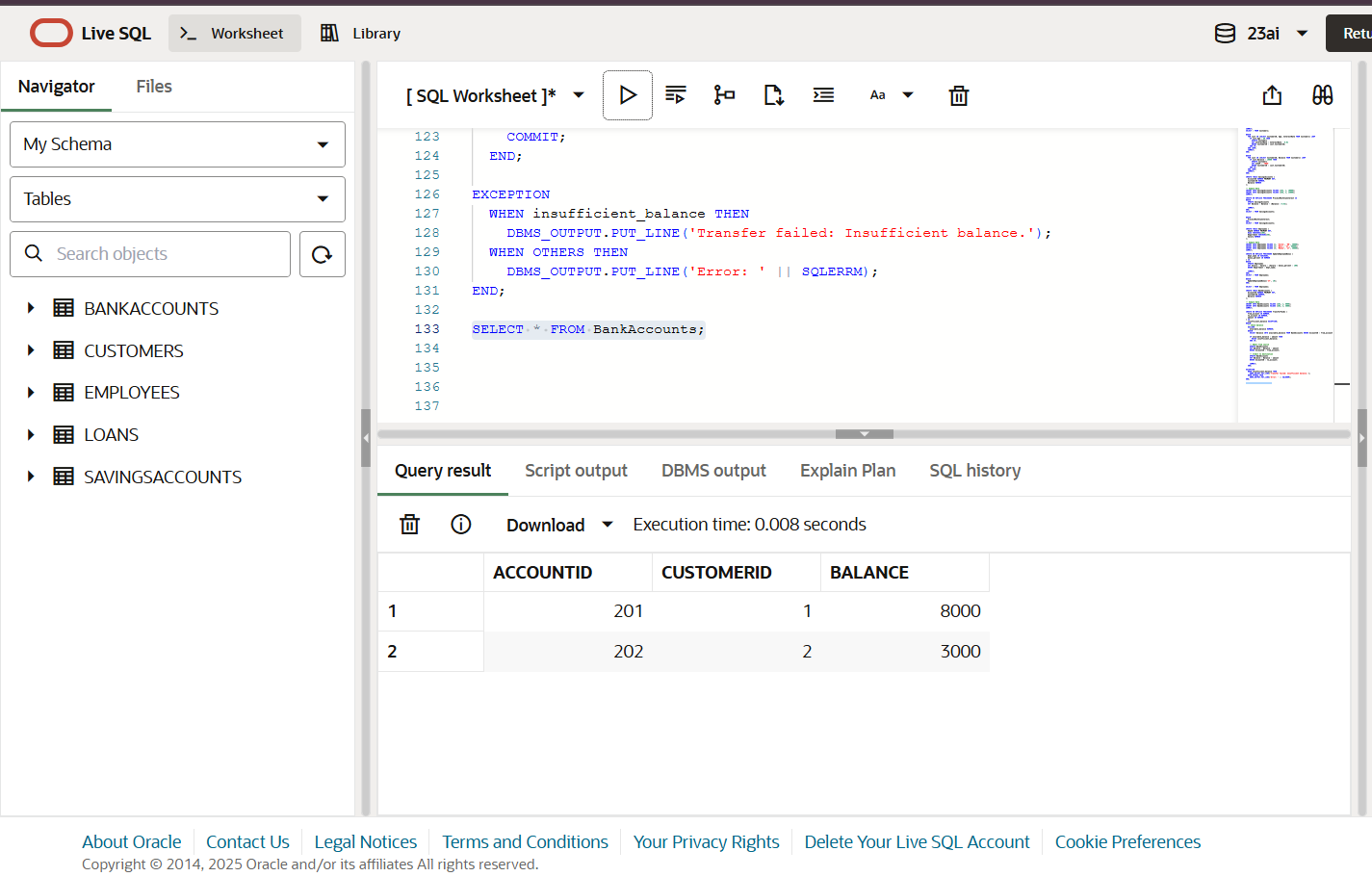
BEGIN

  TransferFunds(201, 202, 2000);

END;

SELECT \* FROM BankAccounts;

**Table creation:**



**Output:**

