**DN 4.0 JAVA FSE SOLUTIONS – WEEK 2**

**SKILL: PL/SQL programming**

**Exercise 1:** Control Structures

**Creating Tables:**

**-- Create Customers Table**

CREATE TABLE Customers (

  CustomerID NUMBER PRIMARY KEY,

  CustomerName VARCHAR2(100),

  Age NUMBER,

  LoanInterestRate NUMBER(5,2),

  Balance NUMBER(10,2),

  IsVIP VARCHAR2(5)

);

**-- Create Loans Table**

CREATE TABLE Loans (

  LoanID NUMBER PRIMARY KEY,

  CustomerID NUMBER,

  DueDate DATE,

  FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**-- Insert Sample Data**

INSERT INTO Customers VALUES (1, 'Rahul', 67, 10.0, 16000, 'FALSE');

INSERT INTO Customers VALUES (2, 'Lasya', 48, 11.5, 9500, 'FALSE');

INSERT INTO Customers VALUES (3, 'Meenu', 70, 9.0, 10200, 'FALSE');

INSERT INTO Customers VALUES (4, 'Rishi', 59, 13.0, 6000, 'FALSE');

INSERT INTO Loans VALUES (101, 1, SYSDATE + 15);

INSERT INTO Loans VALUES (102, 2, SYSDATE + 35);

INSERT INTO Loans VALUES (103, 3, SYSDATE + 10);

INSERT INTO Loans VALUES (104, 4, SYSDATE + 5);

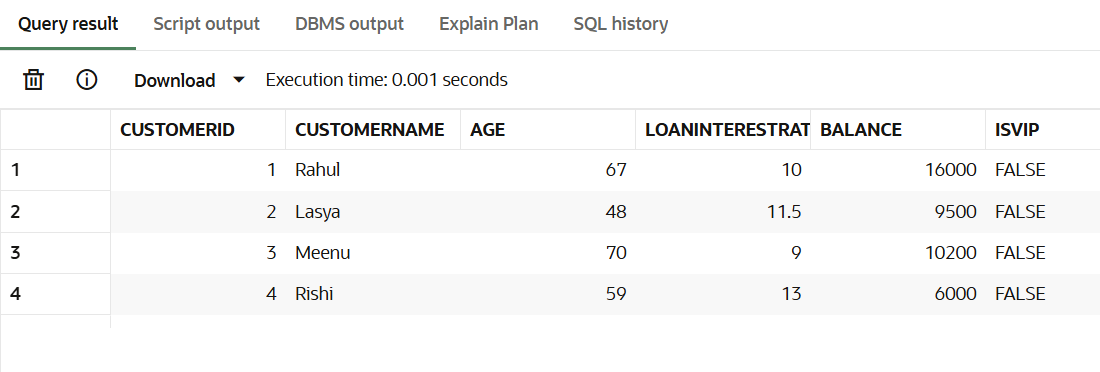
COMMIT;

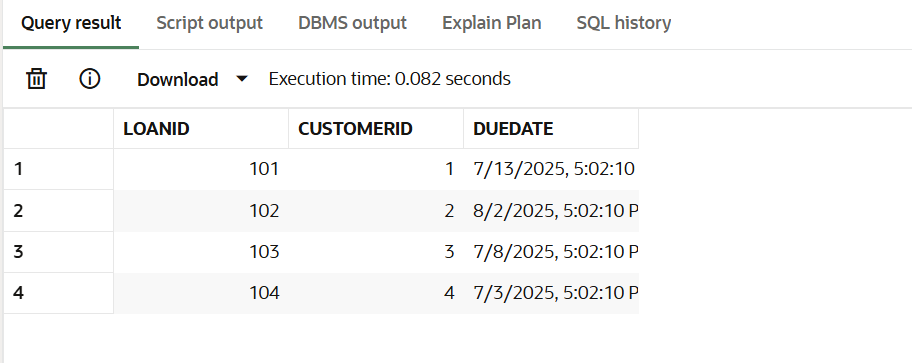
-**- Print data**

SELECT \* FROM Customers;

SELECT \* FROM Loans;

**Output:**

****

****

**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:**

SET SERVEROUTPUT ON;

BEGIN

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

    IF cust\_rec.Age > 60 THEN

      UPDATE Customers

      SET LoanInterestRate = LoanInterestRate - 1

      WHERE CustomerID = cust\_rec.CustomerID;

      DBMS\_OUTPUT.PUT\_LINE('1% discount applied to Customer ID ' || cust\_rec.CustomerID);

    END IF;

  END LOOP;

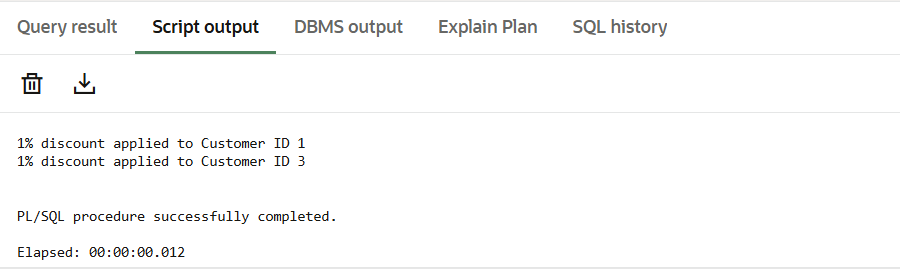
  COMMIT;

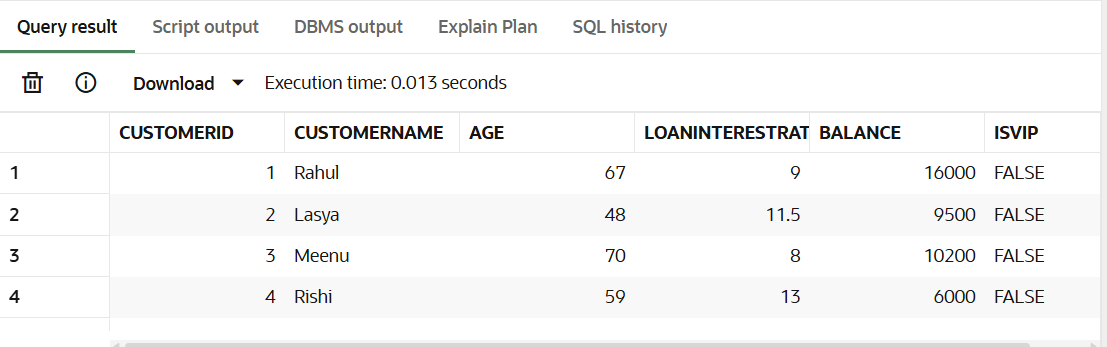
END;

/

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 cust\_rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

    IF cust\_rec.Balance > 10000 THEN

      UPDATE Customers

      SET IsVIP = 'TRUE'

      WHERE CustomerID = cust\_rec.CustomerID;

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

    END IF;

  END LOOP;

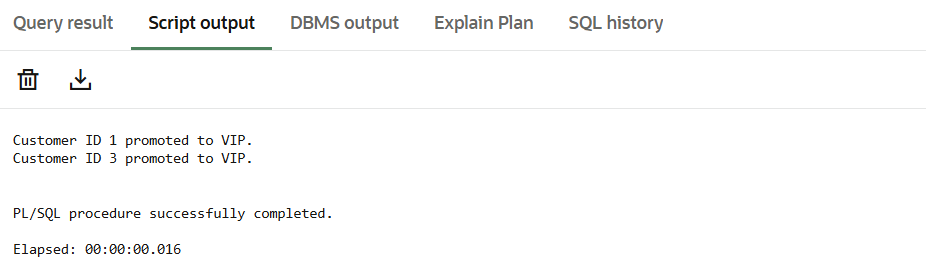
  COMMIT;

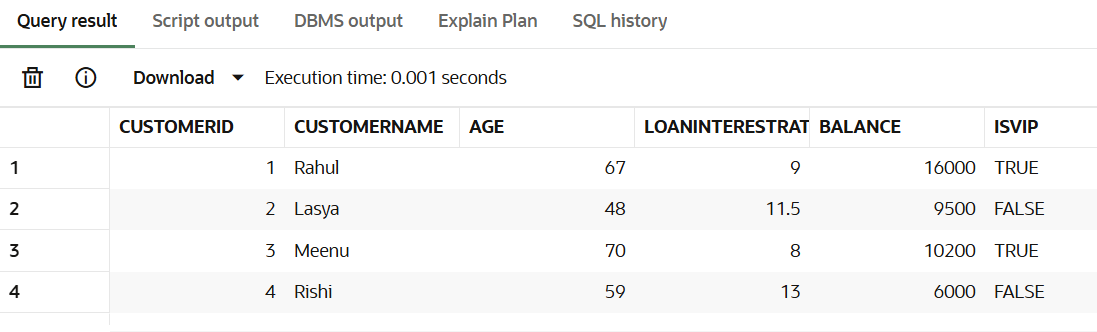
END;

/

SELECT \* FROM Customers;

**Output:**

****

****

**Scenario 3:** The bank wants to send reminders to customers whose loans are due within the next 30 days.

**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 l.LoanID, l.CustomerID, c.CustomerName, 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\_rec.LoanID ||

                         ' for ' || loan\_rec.CustomerName ||

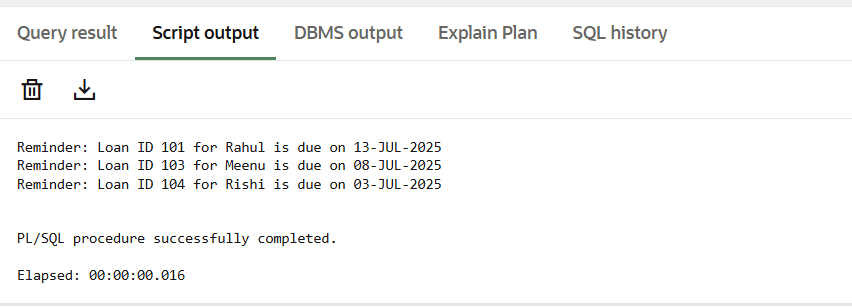
                         ' is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

  END LOOP;

END;

/

**Output:**

****

**Exercise 3:** Stored Procedures

**Creating Tables:**

**-- Create Accounts Table**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

AccountType VARCHAR2(20),

Balance NUMBER(10, 2)

);

**-- Create Employees Table**

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

DepartmentID NUMBER,

Salary NUMBER(10, 2)

);

**-- Insert sample accounts**

INSERT INTO Accounts VALUES (1, 'SAVINGS', 10000);

INSERT INTO Accounts VALUES (2, 'CURRENT', 15000);

INSERT INTO Accounts VALUES (3, 'SAVINGS', 20000);

INSERT INTO Accounts VALUES (4, 'SAVINGS', 5000);

**-- Insert sample employees**

INSERT INTO Employees VALUES (101, 10, 50000);

INSERT INTO Employees VALUES (102, 10, 55000);

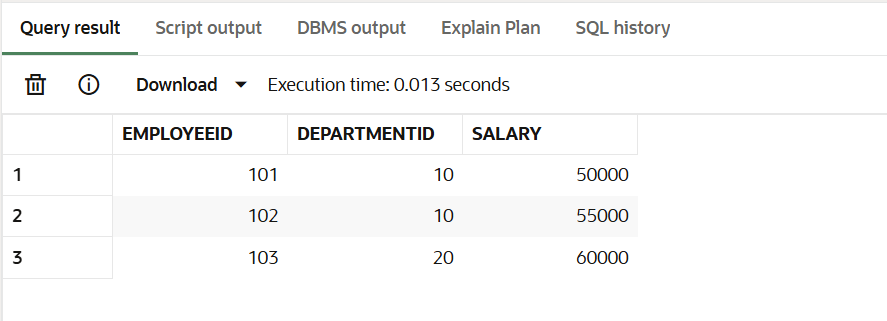
INSERT INTO Employees VALUES (103, 20, 60000);

COMMIT;

SELECT\* FROM Accounts;

SELECT\* FROM Employees;

**Output:**

****

**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:**

-- Enable output

SET SERVEROUTPUT ON;

-- **Procedure 1:** Apply monthly interest to savings accounts

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)

    WHERE AccountID = acc.AccountID;

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

  END LOOP;

  COMMIT;

END;

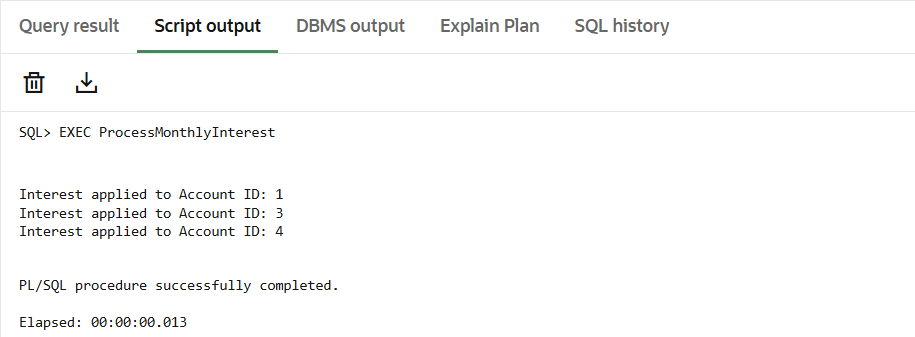
/

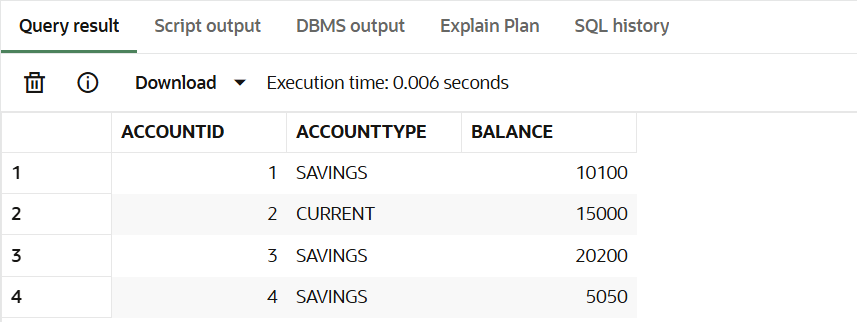
-- **Call Procedure 1:** Process Monthly Interest

EXEC ProcessMonthlyInterest;

SELECT \* FROM Accounts;

**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:**

-- **Procedure 2:** Update employee salary with bonus

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

  dept\_id IN NUMBER,

  bonus\_pct IN NUMBER

) IS

BEGIN

  FOR emp IN (

    SELECT EmployeeID, Salary

    FROM Employees

    WHERE DepartmentID = dept\_id

  ) LOOP

    UPDATE Employees

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

    WHERE EmployeeID = emp.EmployeeID;

    DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID);

  END LOOP;

  COMMIT;

END;

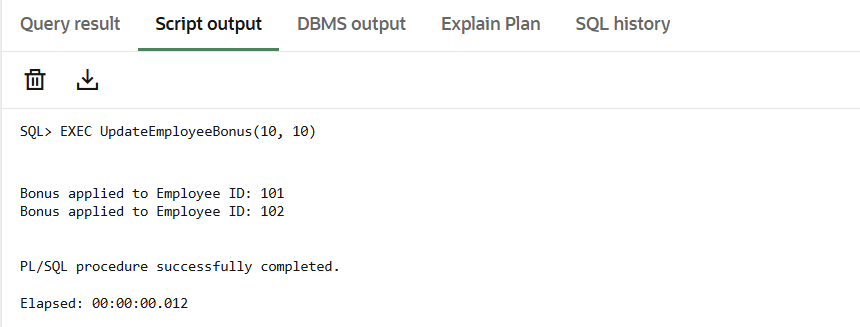
/

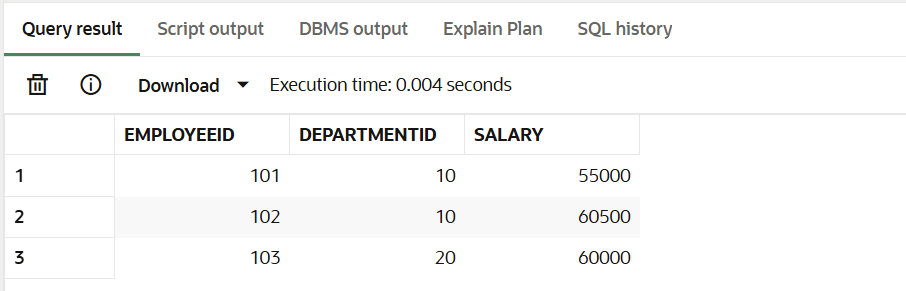
-- **Call Procedure 2:** Add 10% bonus to employees in department 10

EXEC UpdateEmployeeBonus(10, 10);

SELECT \* FROM Employees;

**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:**

-- **Procedure 3:** Transfer funds between accounts

CREATE OR REPLACE PROCEDURE TransferFunds(

  from\_acc IN NUMBER,

  to\_acc IN NUMBER,

  amount IN NUMBER

) IS

  insufficient\_funds EXCEPTION;

BEGIN

  DECLARE

    source\_balance NUMBER;

  BEGIN

    SELECT Balance INTO source\_balance FROM Accounts WHERE AccountID = from\_acc;

    IF source\_balance < amount THEN

      RAISE insufficient\_funds;

    END IF;

    UPDATE Accounts

    SET Balance = Balance - amount

    WHERE AccountID = from\_acc;

    UPDATE Accounts

    SET Balance = Balance + amount

    WHERE AccountID = to\_acc;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Account ' || from\_acc || ' to Account ' || to\_acc);

  END;

EXCEPTION

  WHEN insufficient\_funds THEN

    DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient funds in Account ' || from\_acc);

  WHEN OTHERS THEN

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

END;

/

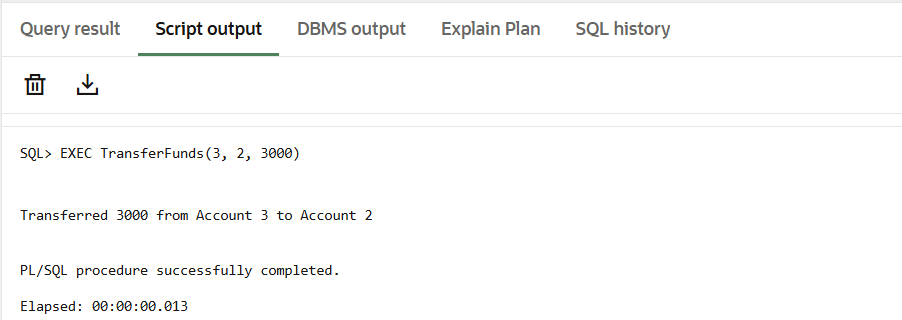
-- **Call Procedure 3:** Transfer 3000 from Account 3 to Account 2

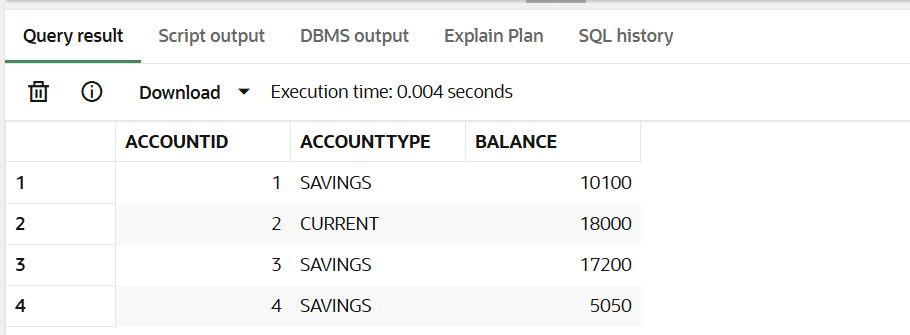
EXEC TransferFunds(3, 2, 3000);

-- Show updated results

SELECT \* FROM Accounts;

**Output:**

****

****