**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.

**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.

**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:**

-- Drop if already exists (safe reset)

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Loans';

  EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Create Customers table

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

-- Create Loans table

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert Customers

INSERT INTO Customers VALUES (1, 'John Elder', TO\_DATE('1950-06-15', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Young', TO\_DATE('1990-03-22', 'YYYY-MM-DD'), 8000, SYSDATE);

INSERT INTO Customers VALUES (3, 'Robert Rich', TO\_DATE('1962-01-01', 'YYYY-MM-DD'), 20000, SYSDATE);

INSERT INTO Customers VALUES (4, 'Sara Saver', TO\_DATE('1985-10-10', 'YYYY-MM-DD'), 3000, SYSDATE);

INSERT INTO Customers VALUES (5, 'Tom Senior', TO\_DATE('1948-12-30', 'YYYY-MM-DD'), 11000, SYSDATE);

-- Insert Loans

INSERT INTO Loans VALUES (101, 1, 50000, 8.5, SYSDATE - 200, SYSDATE + 10);  -- due in 10 days

INSERT INTO Loans VALUES (102, 2, 10000, 7.5, SYSDATE - 100, SYSDATE + 40);  -- due in 40 days

INSERT INTO Loans VALUES (103, 3, 75000, 6.0, SYSDATE - 300, SYSDATE + 5);   -- due in 5 days

INSERT INTO Loans VALUES (104, 4, 15000, 9.0, SYSDATE - 150, SYSDATE - 5);   -- already due

INSERT INTO Loans VALUES (105, 5, 60000, 7.2, SYSDATE - 400, SYSDATE + 20);  -- due in 20 days

COMMIT;

SET SERVEROUTPUT ON;

**--Scenario 1: Apply 1% Discount for Seniors**

BEGIN

  FOR rec IN (

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

           TRUNC(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) AS Age

    FROM Customers c

    JOIN Loans l ON c.CustomerID = l.CustomerID

  ) LOOP

    IF rec.Age > 60 THEN

      UPDATE Loans

      SET InterestRate = InterestRate - 1

      WHERE LoanID = rec.LoanID;

      DBMS\_OUTPUT.PUT\_LINE('Applied 1% discount for Customer ID ' || rec.CustomerID || ' (Age: ' || rec.Age || ')');

    END IF;

  END LOOP;

  COMMIT;

END;

/

****

**-- Scenario 2: Promote VIP Customers**

-- First, alter table to add IsVIP column

ALTER TABLE Customers ADD IsVIP VARCHAR2(5) DEFAULT 'FALSE';

-- Now the PL/SQL block

BEGIN

  FOR rec IN (

    SELECT CustomerID, Balance

    FROM Customers

    WHERE Balance > 10000

  ) LOOP

    UPDATE Customers

    SET IsVIP = 'TRUE', LastModified = SYSDATE

    WHERE CustomerID = rec.CustomerID;

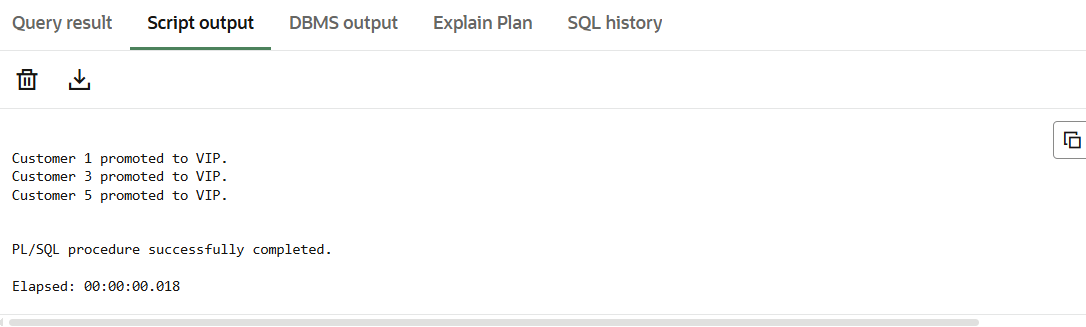
    DBMS\_OUTPUT.PUT\_LINE('Customer ' || rec.CustomerID || ' promoted to VIP.');

  END LOOP;

  COMMIT;

END;

/

****

**-- Scenario 3: Send Loan Due Reminders**

BEGIN

  FOR rec IN (

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

    FROM Loans l

    JOIN Customers c ON l.CustomerID = c.CustomerID

    WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

  ) LOOP

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

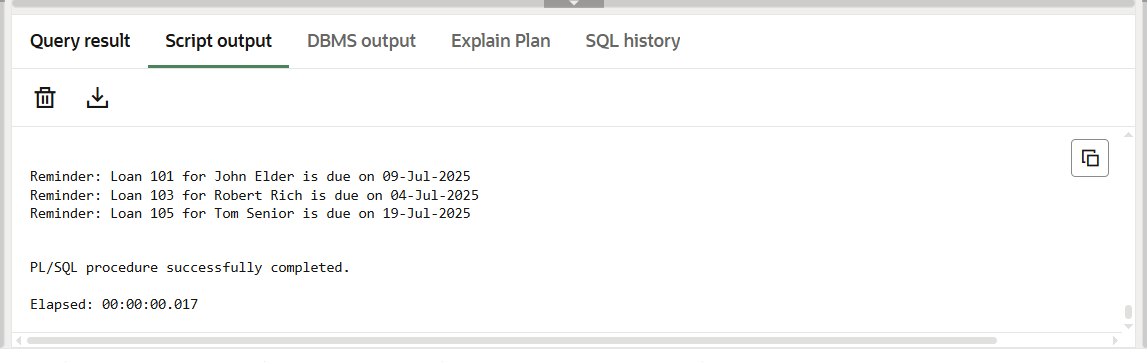
                         ' for ' || rec.Name ||

                         ' is due on ' || TO\_CHAR(rec.EndDate, 'DD-Mon-YYYY'));

  END LOOP;

END;

/

****

**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.

**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.

**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.

-- Drop tables if they exist

BEGIN

  EXECUTE IMMEDIATE 'DROP TABLE Accounts';

  EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

  WHEN OTHERS THEN NULL;

END;

/

-- Create Employees table

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

-- Create Accounts table

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert into Employees

INSERT INTO Employees VALUES (1, 'Alice', 'Manager', 70000, 'Finance', SYSDATE - 1000);

INSERT INTO Employees VALUES (2, 'Bob', 'Analyst', 50000, 'IT', SYSDATE - 800);

INSERT INTO Employees VALUES (3, 'Carol', 'Clerk', 30000, 'HR', SYSDATE - 600);

INSERT INTO Employees VALUES (4, 'Dave', 'Analyst', 48000, 'IT', SYSDATE - 400);

INSERT INTO Employees VALUES (5, 'Eve', 'Manager', 72000, 'Finance', SYSDATE- 300);

-- Insert into Accounts

INSERT INTO Accounts VALUES (201, 1, 'Savings', 10000, SYSDATE);

INSERT INTO Accounts VALUES (202, 2, 'Checking', 4000, SYSDATE);

INSERT INTO Accounts VALUES (203, 3, 'Savings', 20000, SYSDATE);

INSERT INTO Accounts VALUES (204, 4, 'Savings', 1500, SYSDATE);

INSERT INTO Accounts VALUES (205, 5, 'Checking', 1000, SYSDATE);

COMMIT;

**-- Scenario 1: ProcessMonthlyInterest (1% on savings)**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

  FOR rec IN (

    SELECT AccountID, Balance

    FROM Accounts

    WHERE AccountType = 'Savings'

  ) LOOP

    UPDATE Accounts

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

        LastModified = SYSDATE

    WHERE AccountID = rec.AccountID;

    DBMS\_OUTPUT.PUT\_LINE('Applied 1% interest to AccountID: ' || rec.AccountID);

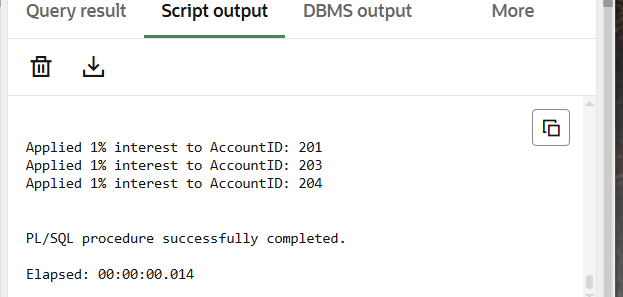
  END LOOP;

  COMMIT;

END;

/

EXEC ProcessMonthlyInterest;

****

**-- Scenario 2: UpdateEmployeeBonus (param-based bonus by department)**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

  deptName IN VARCHAR2,

  bonusPercent IN NUMBER

) AS

BEGIN

  FOR rec IN (

    SELECT EmployeeID, Salary

    FROM Employees

    WHERE Department = deptName

  ) LOOP

    UPDATE Employees

    SET Salary = Salary + (Salary \* bonusPercent / 100)

    WHERE EmployeeID = rec.EmployeeID;

    DBMS\_OUTPUT.PUT\_LINE('Updated bonus for EmployeeID: ' || rec.EmployeeID);

  END LOOP;

  COMMIT;

END;

/

EXEC UpdateEmployeeBonus('IT', 10);

****

**--  Scenario 3: TransferFunds (with balance check)**

CREATE OR REPLACE PROCEDURE TransferFunds (

  sourceAccID IN NUMBER,

  targetAccID IN NUMBER,

  amount IN NUMBER

) AS

  insufficient\_balance EXCEPTION;

BEGIN

  -- Check balance

  DECLARE

    srcBalance NUMBER;

  BEGIN

    SELECT Balance INTO srcBalance FROM Accounts WHERE AccountID = sourceAccID;

    IF srcBalance < amount THEN

      RAISE insufficient\_balance;

    END IF;

  END;

  -- Perform transfer

  UPDATE Accounts

  SET Balance = Balance - amount, LastModified = SYSDATE

  WHERE AccountID = sourceAccID;

  UPDATE Accounts

  SET Balance = Balance + amount, LastModified = SYSDATE

  WHERE AccountID = targetAccID;

  DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount ||

                       ' from AccountID ' || sourceAccID ||

                       ' to AccountID ' || targetAccID);

  COMMIT;

EXCEPTION

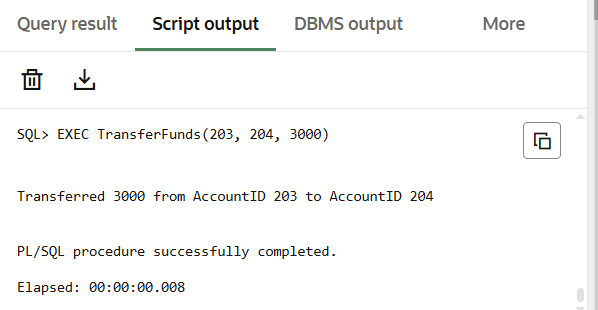
  WHEN insufficient\_balance THEN

    DBMS\_OUTPUT.PUT\_LINE('Transfer failed: insufficient balance in AccountID ' || sourceAccID);

END;

/

EXEC TransferFunds(203, 204, 3000);  -- Should succeed



EXEC TransferFunds(205, 204, 3000);  -- Should fail due to low balance

