**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.

-- DROP TABLES if they already exist (optional cleanup)

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

-- Step 2: Create Customers and Loans tables

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

);

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

LoanAmount NUMBER,

InterestRate NUMBER,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Step 3: Insert Sample Data

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

VALUES (2, 'Jane Smith', TO\_DATE('1950-07-20', 'YYYY-MM-DD'), 12000, SYSDATE);

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

VALUES (2, 2, 10000, 8, SYSDATE, SYSDATE + 20);

-- Step 4: Run PL/SQL Block to Apply 1% Discount

BEGIN

FOR c IN (

SELECT CustomerID

FROM Customers

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

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = c.CustomerID;

END LOOP;

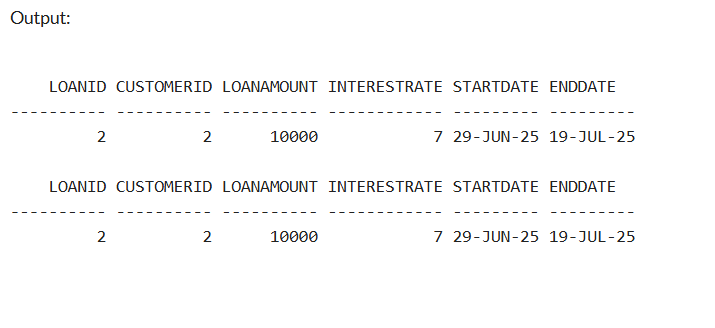
COMMIT;

END;

/

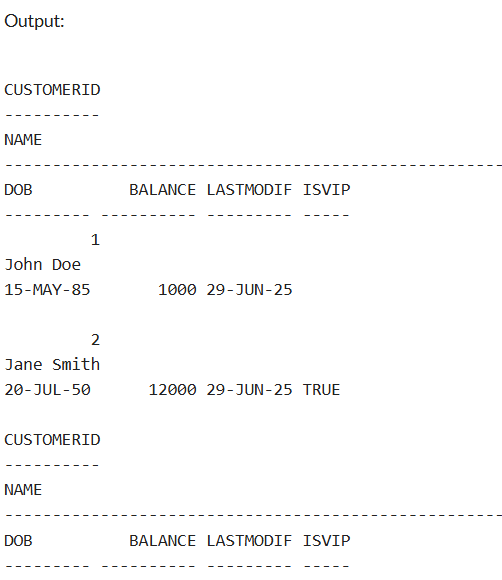
-- Step 5: Check Output

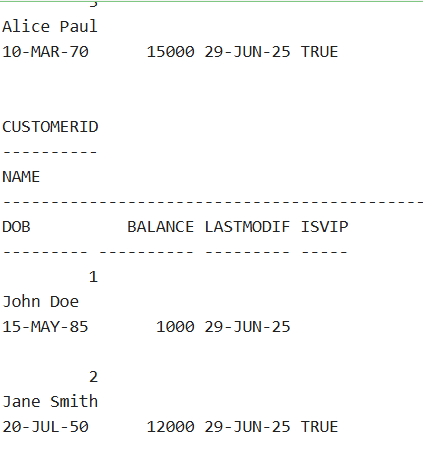
SELECT \* FROM Loans;

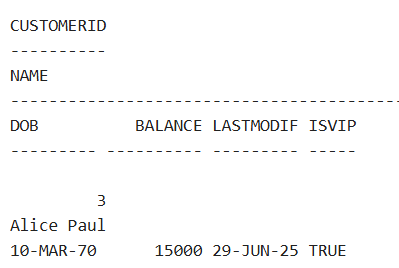


**Scenario 2:** A customer can be promoted to VIP status based on their balance.

* + **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.
  + -- Drop the Customers table if it exists (clean run)
  + BEGIN
  + 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
  + );
  + -- Add IsVIP column
  + ALTER TABLE Customers ADD (IsVIP VARCHAR2(5));
  + -- Insert sample data
  + 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('1950-07-20', 'YYYY-MM-DD'), 12000, SYSDATE);
  + INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)
  + VALUES (3, 'Alice Paul', TO\_DATE('1970-03-10', 'YYYY-MM-DD'), 15000, SYSDATE);
  + -- PL/SQL block to promote VIPs
  + BEGIN
  + FOR c IN (
  + SELECT CustomerID
  + FROM Customers
  + WHERE Balance > 10000
  + ) LOOP
  + UPDATE Customers
  + SET IsVIP = 'TRUE'
  + WHERE CustomerID = c.CustomerID;
  + END LOOP;
  + COMMIT;
  + END;
  + /
  + -- Check final result
  + SELECT \* FROM Customers;







**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.

SET SERVEROUTPUT ON;

BEGIN

-- Drop tables if they already exist

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Loans'; EXCEPTION WHEN OTHERS THEN NULL; END;

BEGIN EXECUTE IMMEDIATE 'DROP TABLE Customers'; EXCEPTION WHEN OTHERS THEN NULL; END;

-- Create Customers table

EXECUTE IMMEDIATE '

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DOB DATE,

Balance NUMBER,

LastModified DATE

)';

-- Create Loans table

EXECUTE IMMEDIATE '

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

EXECUTE IMMEDIATE q'[INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15','YYYY-MM-DD'), 1000, SYSDATE)]';

EXECUTE IMMEDIATE q'[INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1950-07-20','YYYY-MM-DD'), 12000, SYSDATE)]';

EXECUTE IMMEDIATE q'[INSERT INTO Customers VALUES (3, 'Alice Paul', TO\_DATE('1970-03-10','YYYY-MM-DD'), 15000, SYSDATE)]';

-- Insert Loans

EXECUTE IMMEDIATE q'[INSERT INTO Loans VALUES (1, 1, 5000, 5, SYSDATE, SYSDATE + 15)]'; -- due in 15 days

EXECUTE IMMEDIATE q'[INSERT INTO Loans VALUES (2, 2, 10000, 7, SYSDATE, SYSDATE + 45)]'; -- not due

EXECUTE IMMEDIATE q'[INSERT INTO Loans VALUES (3, 3, 12000, 8, SYSDATE, SYSDATE + 25)]'; -- due in 25 days

COMMIT;

-- Output Header

DBMS\_OUTPUT.PUT\_LINE('--- Loan Due Reminders (Next 30 Days) ---');

-- Use dynamic cursor to access tables created in EXECUTE IMMEDIATE

FOR rec IN (

SELECT Name, EndDate

FROM (

SELECT c.Name AS Name, l.EndDate AS EndDate

FROM Customers c, Loans l

WHERE c.CustomerID = l.CustomerID

AND l.EndDate BETWEEN SYSDATE AND SYSDATE + 30

)

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Dear ' || rec.Name ||

', your loan 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.

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Monthly interest processed for all savings accounts.');

END;

/

**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.

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percent IN NUMBER -- e.g., 10 for 10%

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

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

END;

/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check source account balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

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

RETURN;

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

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

END;

/