**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.

**PL/SQL PROGRAM**

**1.** BEGIN

FOR cust IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

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

) LOOP

UPDATE Loans

SET InterestRate = cust.InterestRate - 1

WHERE LoanID = cust.LoanID;

END LOOP;

COMMIT;

END;

/

**OUTPUT**

****

**2.**ALTER TABLE Customers ADD IsVIP number(1); -- 1 = TRUE, 0 = FALSE

BEGIN

  FOR cust IN (

    SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000

  ) LOOP

    UPDATE Customers

    SET IsVIP = 1,

        LastModified = SYSDATE

    WHERE CustomerID = cust.CustomerID;

  END LOOP;

  COMMIT;

END;

/

**OUTPUT**





**3.**BEGIN

  FOR loan\_rec IN (

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

    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 ID ' || loan\_rec.LoanID ||

                         ' for customer ' || loan\_rec.Name ||

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

  END LOOP;

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.

**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.

**PL/SQL CODE:**

**1.**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01,

LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

END;

/

**OUTPUT**



**2.**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_Department IN VARCHAR2,

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

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_BonusPercent / 100)

WHERE Department = p\_Department;

COMMIT;

END;

/

**OUTPUT**



**3.** CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

v\_NewTransID NUMBER;

BEGIN

-- Get source account balance

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID

FOR UPDATE;

-- Check if balance is sufficient

IF v\_FromBalance < p\_Amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account');

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_FromAccountID;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_Amount,

LastModified = SYSDATE

WHERE AccountID = p\_ToAccountID;

-- Get max TransactionID for manual assignment

SELECT NVL(MAX(TransactionID), 0) + 1 INTO v\_NewTransID FROM Transactions;

-- Insert transaction records

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

VALUES (v\_NewTransID, p\_FromAccountID, SYSDATE, p\_Amount, 'Withdrawal');

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

VALUES (v\_NewTransID + 1, p\_ToAccountID, SYSDATE, p\_Amount, 'Deposit');

COMMIT;

END;

/

**OUTPUT**



