**Exercise 1: Control Structures**

**Scenario 1: Apply 1% Discount for Customers Above 60**

**Assumptions:**

* Table: Customers
* Columns: CustomerID, Age, LoanInterestRate

**Code:**

BEGIN

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

IF cust.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Interest rate updated for Customer ID: ' || cust.CustomerID);

END IF;

END LOOP;

END;

**Scenario 2: Promote Customers to VIP Based on Balance**

**Assumptions:**

* Table: Customers
* Columns: CustomerID, Balance, IsVIP (assumed to be CHAR(1) or BOOLEAN equivalent)

**Code:**

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('VIP status set for Customer ID: ' || cust.CustomerID);

END IF;

END LOOP;

END;

**Scenario 3: Send Reminders for Loans Due in Next 30 Days**

**Assumptions:**

* Table: Loans
* Columns: LoanID, CustomerID, DueDate
* Table: Customers
* Columns: CustomerID, Name

**Code:**

DECLARE

CURSOR due\_loans IS

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan IN due\_loans LOOP

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

' for customer ' || loan.Name ||

' (Customer ID: ' || loan.CustomerID ||

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

END LOOP;

END;

**Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest**

**Requirement:** Apply **1% interest** to all **savings accounts**.

### Assumptions:

* Table: Accounts
* Columns: AccountID, AccountType, Balance
* Savings accounts have AccountType = 'SAVINGS'

### Code:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'SAVINGS') LOOP

UPDATE Accounts

SET Balance = Balance + (acc.Balance \* 0.01)

WHERE AccountID = acc.AccountID;

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

END LOOP;

END;

## **Scenario 2: Update Employee Bonus**

### Requirement: Update **salary** by a **bonus %** for employees in a specific **department**.

### Assumptions:

* Table: Employees
* Columns: EmployeeID, DepartmentID, Salary

### Code:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

FOR emp IN (SELECT EmployeeID, Salary FROM Employees WHERE DepartmentID = p\_DepartmentID) LOOP

UPDATE Employees

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

WHERE EmployeeID = emp.EmployeeID;

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

END LOOP;

END;

## **Scenario 3: Transfer Funds Between Accounts**

### Requirement: Transfer funds from one account to another **after checking balance**.

### Assumptions:

* Table: Accounts
* Columns: AccountID, Balance

### Code:

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

-- Get balance of source account

SELECT Balance INTO v\_FromBalance FROM Accounts WHERE AccountID = p\_FromAccountID FOR UPDATE;

-- Check if sufficient balance

IF v\_FromBalance < p\_Amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient balance in Account ID: ' || p\_FromAccountID);

RETURN;

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_ToAccountID;

DBMS\_OUTPUT.PUT\_LINE('Transferred ' || p\_Amount || ' from Account ' ||

p\_FromAccountID || ' to Account ' || p\_ToAccountID);

END;