**Exercise 1: Control Structures**

**Code:**

**Table Creation:**

-- Customers table

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER(10,2),

IsVIP CHAR(1) DEFAULT 'N'

);

-- Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

InterestRate NUMBER(5,2),

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

**Input:**

INSERT INTO Customers VALUES (1, 'Alice', 65, 12000, 'N');

INSERT INTO Customers VALUES (2, 'Bob', 45, 8000, 'N');

INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'N');

INSERT INTO Loans VALUES (101, 1, 8.5, SYSDATE + 10);

INSERT INTO Loans VALUES (102, 2, 7.0, SYSDATE + 40);

INSERT INTO Loans VALUES (103, 3, 9.0, SYSDATE + 25);

COMMIT;

**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 IN (SELECT CustomerID FROM Customers WHERE Age > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

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

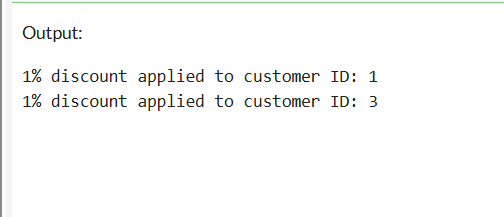
END LOOP;

COMMIT;

END;

/

**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:**

SET SERVEROUTPUT ON;

BEGIN

FOR cust IN (SELECT CustomerID FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

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

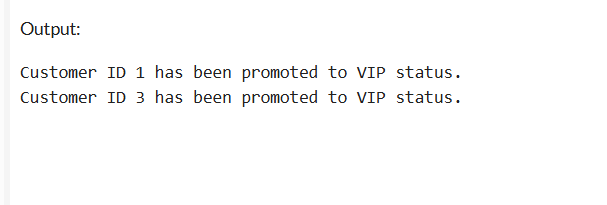
END LOOP;

COMMIT;

END;

/

**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:**

SET SERVEROUTPUT ON;

DECLARE

CURSOR loan\_cursor IS

SELECT l.LoanID, 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\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || loan\_rec.Name ||

', your loan (ID: ' || loan\_rec.LoanID ||

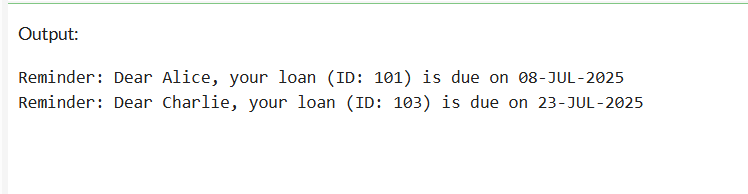
') is due on ' || TO\_CHAR(loan\_rec.DueDate, 'DD-MON-YYYY'));

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.

**CODE:**

CREATE TABLE SavingsAccounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(12,2)

);

INSERT INTO SavingsAccounts VALUES (1, 101, 10000);

INSERT INTO SavingsAccounts VALUES (2, 102, 20000);

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

SET SERVEROUTPUT ON;

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM SavingsAccounts) LOOP

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

DBMS\_OUTPUT.PUT\_LINE('1% interest applied to Account ID: ' || acc.AccountID);

END LOOP;

COMMIT;

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.

**CODE:**

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

DepartmentID NUMBER,

Salary NUMBER(10,2)

);

INSERT INTO Employees VALUES (1, 'Alice', 101, 5000);

INSERT INTO Employees VALUES (2, 'Bob', 101, 5500);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) IS

BEGIN

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

UPDATE Employees

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

WHERE EmpID = emp.EmpID;

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

END LOOP;

COMMIT;

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.

**CODE:**

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER(12,2)

);

INSERT INTO Accounts VALUES (1001, 101, 5000);

INSERT INTO Accounts VALUES (1002, 102, 3000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_FromAccountID IN NUMBER,

p\_ToAccountID IN NUMBER,

p\_Amount IN NUMBER

) IS

v\_FromBalance NUMBER;

BEGIN

SELECT Balance INTO v\_FromBalance

FROM Accounts

WHERE AccountID = p\_FromAccountID;

IF v\_FromBalance < p\_Amount THEN

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

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_FromAccountID;

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);

COMMIT;

END;

/