**Exercise 1: Control Structures**

**Scenario 1: Discount for Senior Citizens**

**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:**

DECLARE

CURSOR customer\_cursor IS

SELECT CustomerID, Age, LoanInterestRate

FROM Customers;

BEGIN

FOR customer IN customer\_cursor LOOP

IF customer.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = customer.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 2: Promote Customers to VIP**

**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:**

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID, Balance

FROM Customers;

BEGIN

FOR cust IN vip\_cursor LOOP

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**Scenario 3: Loan Due Reminders**

**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:**

DECLARE

CURSOR loan\_cursor IS

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30;

customer\_name VARCHAR2(100);

BEGIN

FOR loan IN loan\_cursor LOOP

SELECT Name INTO customer\_name

FROM Customers

WHERE CustomerID = loan.CustomerID;

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear ' || customer\_name ||

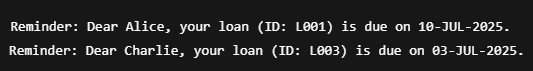
', your loan (ID: ' || loan.LoanID ||

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

END LOOP;

END;

**Output:**

****

**Exercise 3: Stored Procedures**

**Scenario 1: Monthly Interest for 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 OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

COMMIT;

END;

**Scenario 2: Bonus Scheme for Employees**

**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 OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE DepartmentID = dept\_id;

COMMIT;

END;

**Scenario 3: Fund Transfer Between 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 OR REPLACE PROCEDURE TransferFunds(

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) IS

from\_balance NUMBER;

BEGIN

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account;

IF from\_balance < amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds.');

ELSE

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

COMMIT;

END IF;

END;