**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.

**Solution:**BEGIN

FOR customer\_rec IN (SELECT CustomerID, Age FROM Customers) LOOP

IF customer\_rec.Age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = customer\_rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**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.

**Solution:**

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**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.

**Solution:**

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, c.CustomerName, l.DueDate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate <= SYSDATE + 30 AND l.DueDate >= SYSDATE

) LOOP

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

' for ' || loan\_rec.CustomerName ||

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

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.

**Solution:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01);

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.

**Solution:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE DepartmentID = dept\_id;

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.

**Solution:**

CREATE OR REPLACE PROCEDURE TransferFunds(

from\_account IN NUMBER,

to\_account IN NUMBER,

amount IN NUMBER

) AS

insufficient\_balance EXCEPTION;

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = from\_account FOR UPDATE;

IF v\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = from\_account;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_account;

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient balance in source account.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Unexpected error occurred: ' || SQLERRM);

END;sS