**PL/SQL PROGRAMMING**

**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.

**CODE:**

BEGIN

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

IF cust\_rec.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = cust\_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.

**CODE:**

BEGIN

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

IF cust\_rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust\_rec.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.

**CODE:**

DECLARE

v\_due\_date\_threshold DATE := SYSDATE + 30;

BEGIN

FOR loan\_rec IN (

SELECT L.LoanID, L.CustomerID, L.DueDate, C.Name

FROM Loans L

JOIN Customers C ON L.CustomerID = C.CustomerID

WHERE L.DueDate <= v\_due\_date\_threshold

) 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;

**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 OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

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

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acct.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 OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_DepartmentID IN NUMBER,

p\_BonusPercent IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE DepartmentID = p\_DepartmentID;

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 OR REPLACE PROCEDURE TransferFunds(

p\_SourceAccountID IN NUMBER,

p\_TargetAccountID IN NUMBER,

p\_Amount IN NUMBER

) AS

v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance

FROM Accounts

WHERE AccountID = p\_SourceAccountID;

IF v\_SourceBalance >= p\_Amount THEN

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_Amount

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts

SET Balance = Balance + p\_Amount

WHERE AccountID = p\_TargetAccountID;

COMMIT;

ELSE

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

END IF;

END;

/

**JUnit testing**

**Exercise 1: Setting Up Junit**

pom.xml

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

(Maven)

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M5</version>

</plugin>

</plugins>

</build>

JUnit test class

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(2, 3);

assertEquals(5, result);

}

}

**Exercise 3: Assertions in JUnit**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

assertNull(null);

assertNotNull(new Object());

}

}

**Exercise 4: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit**

public class Calculator {

public int add(int a, int b) { return a + b; }

public int subtract(int a, int b) { return a - b; }

}

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

public void setUp() {

calculator = new Calculator();

System.out.println("Setup: Calculator instance created.");

}

public void tearDown() {

calculator = null;

System.out.println("Teardown: Calculator instance cleared.");

}

@Test

public void testAdd() {

int a = 5;

int b = 3;

int result = calculator.add(a, b);

assertEquals(8, result);

}

@Test

public void testSubtract() {

int a = 10;

int b = 4;

int result = calculator.subtract(a, b);

assertEquals(6, result);

}

}