**WEEK-2**

**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:**

DECLARE

CURSOR senior\_cursor IS

SELECT c.CustomerID, l.LoanID, l.InterestRate, c.DOB FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID;

v\_age NUMBER;

BEGIN

FOR rec IN senior\_cursor LOOP

v\_age := TRUNC(MONTHS\_BETWEEN(SYSDATE, rec.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = rec.LoanID;

END IF;

END LOOP;

COMMIT;

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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 rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers SET IsVIP = 'Y' WHERE CustomerID = rec.CustomerID;

ELSE

UPDATE Customers SET IsVIP = 'N' WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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

CURSOR due\_soon\_loans IS

SELECT c.Name, l.LoanID, l.EndDate FROM Loans l

JOIN Customers c ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN due\_soon\_loans LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || rec.LoanID || ' for customer ' || rec.Name || ' is due on ' || TO\_CHAR(rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

/

**OUTPUT:**

**A computer screen shot of a black screen

AI-generated content may be incorrect.**

**Exercise 2: Error Handling**

**Scenario 1:** Handle exceptions during fund transfers between accounts.

**Question:** Write a stored procedure **SafeTransferFunds** that transfers funds between two accounts. Ensure that if any error occurs (e.g., insufficient funds), an appropriate error message is logged and the transaction is rolled back.

**CODE:**

CREATING A PROCEDURE

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_SourceAccountID IN NUMBER,

p\_DestAccountID IN NUMBER,

p\_Amount IN NUMBER)

IS v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance FROM Accounts

WHERE AccountID = p\_SourceAccountID FOR UPDATE;

IF v\_SourceBalance < p\_Amount THEN

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

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_DestAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer successful: ' || p\_Amount || ' transferred from Account ' || p\_SourceAccountID || ' to Account ' || p\_DestAccountID);

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: ' || SQLERRM);

END;

/

CALLING THE PROCEDURE

BEGIN

SafeTransferFunds(101,102,500);

SafeTransferFunds(105,101,1000);

END;

/

**OUTPUT: A screenshot of a computer

AI-generated content may be incorrect.**

**Scenario 2:** Manage errors when updating employee salaries.

* + **Question:** Write a stored procedure **UpdateSalary** that increases the salary of an employee by a given percentage. If the employee ID does not exist, handle the exception and log an error message.

**CODE:**

CREATING PROCEDURE

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_EmployeeID IN NUMBER,

p\_Percentage IN NUMBER ) IS v\_CurrentSalary NUMBER;

BEGIN

SELECT Salary INTO v\_CurrentSalary

FROM Employees WHERE EmployeeID = p\_EmployeeID;

UPDATE Employees SET Salary = Salary + (v\_CurrentSalary \* p\_Percentage / 100),

HireDate = HireDate WHERE EmployeeID = p\_EmployeeID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Salary updated for Employee ID ' || p\_EmployeeID);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_EmployeeID || ' does not exist.');

WHEN OTHERS THEN

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

ROLLBACK;

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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:**

CREATING PROCEDURE

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts SET Balance = Balance + (Balance \* 0.01), LastModified = SYSDATE

WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all Savings accounts.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error processing interest: ' || SQLERRM);

END;

/

CALLING PROCEDURE

BEGIN

ProcessMonthlyInterest;

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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\_Department IN VARCHAR2, p\_BonusPercent IN NUMBER) IS v\_Count NUMBER;

BEGIN

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

WHERE Department = p\_Department;

SELECT COUNT(\*) INTO v\_Count FROM Employees

WHERE Department = p\_Department;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Bonus of ' || p\_BonusPercent || '% applied to ' || v\_Count ||

' employee(s) in department ' || p\_Department || '.');

EXCEPTION

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error applying bonus: ' || SQLERRM);

END;

/

CALLING PROCEDURE

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/

**OUTPUT:** A screenshot of a computer

AI-generated content may be incorrect.

**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:**

CREATING PROCEDURE

CREATE OR REPLACE PROCEDURE TransferFunds ( p\_SourceAccountID IN NUMBER,

p\_DestAccountID IN NUMBER, p\_Amount IN NUMBER)

IS v\_SourceBalance NUMBER;

BEGIN

SELECT Balance INTO v\_SourceBalance FROM Accounts

WHERE AccountID = p\_SourceAccountID FOR UPDATE;

IF v\_SourceBalance < p\_Amount THEN

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient funds in Account ' || p\_SourceAccountID);

RETURN;

END IF;

UPDATE Accounts SET Balance = Balance - p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_SourceAccountID;

UPDATE Accounts SET Balance = Balance + p\_Amount, LastModified = SYSDATE

WHERE AccountID = p\_DestAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Transfer of ' || p\_Amount ||

' successful from Account ' || p\_SourceAccountID || ' to Account ' || p\_DestAccountID);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: One or both account IDs are invalid.');

WHEN OTHERS THEN

ROLLBACK;

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

END;

/

CALLING PROCEDURE:

BEGIN

TransferFunds(101, 102, 500);

END;

/

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**JUnit, Mockito and SL4J**

**JUnit\_Basic Testing Exercises**

**Exercise 1: Setting Up JUnit**

**Scenario:** You need to set up JUnit in your Java project to start writing unit tests.

**Steps:**1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

2. Add JUnit dependency to your project. If you are using Maven, add the following to your pom.xml: junit junit 4.13.2 test

3. Create a new test class in your project.

**CODE:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class SampleTest {

@Test

public void sampleTest() {

assertTrue(true);

}

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 2:** Writing Basic JUnit Tests

**Scenario:** You need to write basic JUnit tests for a simple Java class.

**Steps:**

1. Create a new Java class with some methods to test.

2. Write JUnit tests for these methods.

**CODE:**

// Calculator.java

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public int multiply(int a, int b) {

return a \* b;

}

public int divide(int a, int b) {

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

}}

// CalculatorTest.java

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

Calculator calc = new Calculator();

@Test

public void testAdd() {

assertEquals(5, calc.add(2, 3));

}

@Test

public void testSubtract() {

assertEquals(1, calc.subtract(3, 2));

}

@Test

public void testMultiply() {

assertEquals(6, calc.multiply(2, 3));

}

@Test(expected = ArithmeticException.class)

public void testDivideByZero() {

calc.divide(5, 0);

}

@Test

public void testDivide() {

assertEquals(2, calc.divide(6, 3));

}}

**OUTPUT:** A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3:** Assertions in JUnit

**Scenario:** You need to use different assertions in JUnit to validate your test results. **Steps:** Write tests using various JUnit assertions.

**CODE:**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(10 > 5);

assertFalse(2 > 3);

Object obj = null;

assertNull(obj);

Object obj2 = new Object();

assertNotNull(obj2);

} }

**OUTPUT: A screenshot of a computer

AI-generated content may be incorrect.Exercise 4:** Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in JUnit

**Scenario:** You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods**.**

**Steps:** 1. Write tests using the AAA pattern.

1. Use @Before and @After annotations for setup and teardown methods.

**CODE:**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class FixtureTest {

private Calculator calc;

@Before

public void setUp() {

// Arrange

calc = new Calculator();

System.out.println("Setting up before test...");

}

@After

public void tearDown() {

System.out.println("Cleaning up after test...");

}

@Test

public void testAddition() {

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

int result = calc.subtract(5, 2);

assertEquals(3, result);

}

}

**OUTPUT:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**Mockito exercises**

**Exercise 1: Mocking and Stubbing**

**Scenario:** You need to test a service that depends on an external API. Use Mockito to mock the external API and stub its methods.

**Steps:**

1. Create a mock object for the external API.

2. Stub the methods to return predefined values.

3. Write a test case that uses the mock object.

Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class); when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

} }

**CODE:**

package com.example;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

class ExternalApi {

public String getData() {

return "Real Data";

} }

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

} }

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

when(mockApi.getData()).thenReturn("Mock Data");

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

} }

**OUTPUT: A computer screen shot of a program

AI-generated content may be incorrect.**

**Exercise 2:** Verifying Interactions

**Scenario:** You need to ensure that a method is called with specific arguments. **Steps**:

1. Create a mock object.

2. Call the method with specific arguments.

3. Verify the interaction.

**CODE:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class VerifyInteractionTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

} }

**OUTPUT: A screenshot of a computer

AI-generated content may be incorrect.**

**SL4J Logging exercises**

**Exercise 1**: Logging Error Messages and Warning Levels

**Task:** Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**Step-by-Step Solution:**

1. Add SLF4J and Logback dependencies to your `pom.xml` file: org.slf4j slf4j-api 1.7.30 ch.qos.logback logback-classic 1.2.3

2. Create a Java class that uses SLF4J for logging

**CODE:**

package demo.example;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class LoggingExample {

    private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

    public static void main(String[] args) {

        logger.error("This is an error message");

        logger.warn("This is a warning message");

    }}

**OUTPUT:**

**A screenshot of a computer

AI-generated content may be incorrect.**