WEEK 2

PLSQL\_Exercises

**Exercise 1: Control Structures**

**SCENARIO 1:**

BEGIN

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

IF cust.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

**SCENARIO 2:**

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:**

DECLARE

v\_today DATE := SYSDATE;

BEGIN

FOR loan IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

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

' for Customer ID ' || loan.CustomerID ||

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

END LOOP;

END;

**Exercise 3: Stored Procedures**

**SCENARIO 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

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

UPDATE SavingsAccounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

**SCENARIO 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE DepartmentID = p\_department\_id;

COMMIT;

END;

**SCENARIO 3:**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check if the source account has enough balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance >= p\_amount THEN

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

ELSE

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

END IF;

END;

***JUnit\_Basic Testing Exercises***

**Exercise 1: Setting Up JUnit**

**POM.xml:**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Calculator.java:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java:**

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(5, 3);

assertEquals(8, result);

}

}

**Exercise 3: Assertions in JUnit**

**class AssertionsTest :**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions() {

assertEquals("Sum should be 5", 5, 2 + 3);

assertTrue("5 is greater than 3", 5 > 3);

assertFalse("5 is not less than 3", 5 < 3);

assertNull("Should be null", null);

assertNotNull("Should not be null", new Object());

}

}

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

**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;

}

}

**CalculatorTest.java**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Setup method: runs before each test

@Before

public void setUp() {

calculator = new Calculator(); // Arrange

System.out.println("Setup complete");

}

// Teardown method: runs after each test

@After

public void tearDown() {

calculator = null;

System.out.println("Teardown complete");

}

@Test

public void testAdd() {

// Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

assertEquals(8, result);

}

@Test

public void testSubtract() {

// Arrange

int a = 10;

int b = 4;

// Act

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

// Assert

assertEquals(6, result);

}

}

***Mockito exercises***

**Exercise 1: Mocking and Stubbing:**

**ExternalApi.java**public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java:**

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**Pom.xml:**

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

**Exercise 2: Verifying Interactions**

**ExternalApi.java:**

public interface ExternalApi {

String getData();

}

**MyService.java:**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData(); // We will verify this method call

}

}

**MyServiceTest.java:**

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Pom.xml:**

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.10.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.mockito</groupId>

<artifactId>mockito-core</artifactId>

<version>5.12.0</version>

<scope>test</scope>

</dependency>

***SL4J Logging exercises***

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

**Pom.xml:**

<dependencies>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

**LoggingExample.java:**

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");

}

}