**WEEK 2 PLSQL EXCERCISES**

**1.Control Structures**

**-- CREATE TABLE Customers (**

**-- CustomerID NUMBER PRIMARY KEY,**

**-- Name VARCHAR2(100),**

**-- DOB DATE,**

**-- Balance NUMBER,**

**-- LastModified DATE**

**-- );**

**--**

**-- CREATE TABLE Loans (**

**-- LoanID NUMBER PRIMARY KEY,**

**-- CustomerID NUMBER,**

**-- LoanAmount NUMBER,**

**-- InterestRate NUMBER,**

**-- StartDate DATE,**

**-- EndDate DATE,**

**-- FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)**

**-- );**

**--**

**-- INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (1, 'John Doe', TO\_DATE('1950-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);**

**-- INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (2, 'Jane Smith', TO\_DATE('1985-07-22', 'YYYY-MM-DD'), 300, SYSDATE);**

**-- INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified) VALUES (3, 'Bob Brown', TO\_DATE('1990-01-10', 'YYYY-MM-DD'), 2500, SYSDATE);**

**--**

**-- INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (1, 1, 5000, 5, SYSDATE, ADD\_MONTHS(SYSDATE, 60));**

**-- INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (2, 2, 3000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 36));**

**-- INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate) VALUES (3, 3, 7000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 72));**

**--**

**-- COMMIT;**

**-- Scenario 1: We want to reduce interest rate by 1 if customer is older than 60 years**

**SET SERVEROUTPUT ON;**

**BEGIN**

**FOR cust IN (**

**SELECT c.CustomerID, c.DOB, l.LoanID, l.InterestRate**

**FROM Customers c**

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

**) LOOP**

**-- Calculate age by dividing months between today and DOB by 12**

**IF MONTHS\_BETWEEN(SYSDATE, cust.DOB)/12 > 60 THEN**

**UPDATE Loans**

**SET InterestRate = InterestRate - 1**

**WHERE LoanID = cust.LoanID;**

**DBMS\_OUTPUT.PUT\_LINE('Interest discount given to Customer ID: ' || cust.CustomerID);**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**A screenshot of a computer

AI-generated content may be incorrect.**

**-- Scenario 2: Add bonus money of 200 if balance is less than 500**

**BEGIN**

**FOR cust IN (**

**SELECT CustomerID, Balance FROM Customers**

**) LOOP**

**IF cust.Balance < 500 THEN**

**UPDATE Customers**

**SET Balance = Balance + 200,**

**LastModified = SYSDATE**

**WHERE CustomerID = cust.CustomerID;**

**DBMS\_OUTPUT.PUT\_LINE('Bonus 200 added for Customer ID: ' || cust.CustomerID);**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('No bonus for Customer ID: ' || cust.CustomerID || ', current balance: ' || cust.Balance);**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**/**

**A screenshot of a computer

AI-generated content may be incorrect.**

**-- Scenario 3: Deduct 100 as penalty if balance is more than 2000**

**BEGIN**

**FOR cust IN (**

**SELECT CustomerID, Balance FROM Customers**

**) LOOP**

**IF cust.Balance > 2000 THEN**

**UPDATE Customers**

**SET Balance = Balance - 100,**

**LastModified = SYSDATE**

**WHERE CustomerID = cust.CustomerID;**

**DBMS\_OUTPUT.PUT\_LINE('Penalty of 100 applied to Customer ID: ' || cust.CustomerID);**

**ELSE**

**DBMS\_OUTPUT.PUT\_LINE('No penalty for Customer ID: ' || cust.CustomerID || ', balance: ' || cust.Balance);**

**END IF;**

**END LOOP;**

**COMMIT;**

**END;**

**/**/

**A screenshot of a computer

AI-generated content may be incorrect.**

**Exercise 3: Stored Procedures**

**-- Scenario 1:Process Monthly Interest for all savings accounts**

**CREATE TABLE Accounts (**

**AccountID NUMBER PRIMARY KEY,**

**CustomerID NUMBER,**

**AccountType VARCHAR2(20),**

**Balance NUMBER**

**);**

**-- i have inserted values**

**INSERT INTO Accounts VALUES (1, 101, 'Savings', 1000);**

**INSERT INTO Accounts VALUES (2, 102, 'Savings', 2000);**

**INSERT INTO Accounts VALUES (3, 103, 'Current', 3000);**

**COMMIT;**

**-- I Created the procedure for processing monthly interest**

**CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Scenario 1: Process Monthly Interest');**

**FOR acc IN (**

**SELECT AccountID, Balance FROM Accounts**

**WHERE AccountType = 'Savings'**

**) LOOP**

**UPDATE Accounts**

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

**WHERE AccountID = acc.AccountID;**

**DBMS\_OUTPUT.PUT\_LINE('Interest applied to Account ID: ' || acc.AccountID);**

**END LOOP;**

**END;**

**/**

**-- I Executed the procedure**

**SET SERVEROUTPUT ON;**

**BEGIN**

**ProcessMonthlyInterest;**

**END;**

**/**

**A screenshot of a computer

AI-generated content may be incorrect.**

**-- Scenario 2: Update Employee Bonus by department**

**-- i Created Employees table**

**CREATE TABLE Employees (**

**EmployeeID NUMBER PRIMARY KEY,**

**Name VARCHAR2(100),**

**Department VARCHAR2(50),**

**Salary NUMBER**

**);**

**INSERT INTO Employees VALUES (1, 'Alice', 'HR', 50000);**

**INSERT INTO Employees VALUES (2, 'Bob', 'IT', 60000);**

**INSERT INTO Employees VALUES (3, 'Charlie', 'IT', 55000);**

**COMMIT;**

**-- Creating the procedure to update employee bonus**

**CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (**

**deptName IN VARCHAR2,**

**bonusPercent IN NUMBER**

**) AS**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Scenario 2: Update Employee Bonus');**

**FOR emp IN (**

**SELECT EmployeeID, Salary FROM Employees**

**WHERE Department = deptName**

**) LOOP**

**UPDATE Employees**

**SET Salary = Salary + (Salary \* bonusPercent / 100)**

**WHERE EmployeeID = emp.EmployeeID;**

**DBMS\_OUTPUT.PUT\_LINE('Bonus applied to Employee ID: ' || emp.EmployeeID);**

**END LOOP;**

**END;**

**/**

**BEGIN**

**UpdateEmployeeBonus('IT', 10);**

**END;**

**/**

**A screenshot of a computer

AI-generated content may be incorrect.**

**-- Scenario 3: Transfer funds between accounts**

**CREATE OR REPLACE PROCEDURE TransferFunds (**

**fromAccID IN NUMBER,**

**toAccID IN NUMBER,**

**amount IN NUMBER**

**) AS**

**fromBalance NUMBER;**

**BEGIN**

**DBMS\_OUTPUT.PUT\_LINE('Scenario 3: Transfer Funds');**

**SELECT Balance INTO fromBalance FROM Accounts WHERE AccountID = fromAccID;**

**IF fromBalance >= amount THEN**

**UPDATE Accounts SET Balance = Balance - amount WHERE AccountID = fromAccID;**

**UPDATE Accounts SET Balance = Balance + amount WHERE AccountID = toAccID;**

**DBMS\_OUTPUT.PUT\_LINE('Transferred '||amount ||' from Account '||fromAccID||' to Account '||toAccID);**

**ELSE**

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

**END IF;**

**END;**

**/**

**-- Execute fund transfer**

**BEGIN**

**TransferFunds(2, 3, 500);**

**END;**

**/A screenshot of a computer

AI-generated content may be incorrect.**

**JUNIT BASIC TESTING 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:

3. Create a new test class in your project.

**CODE:**

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.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

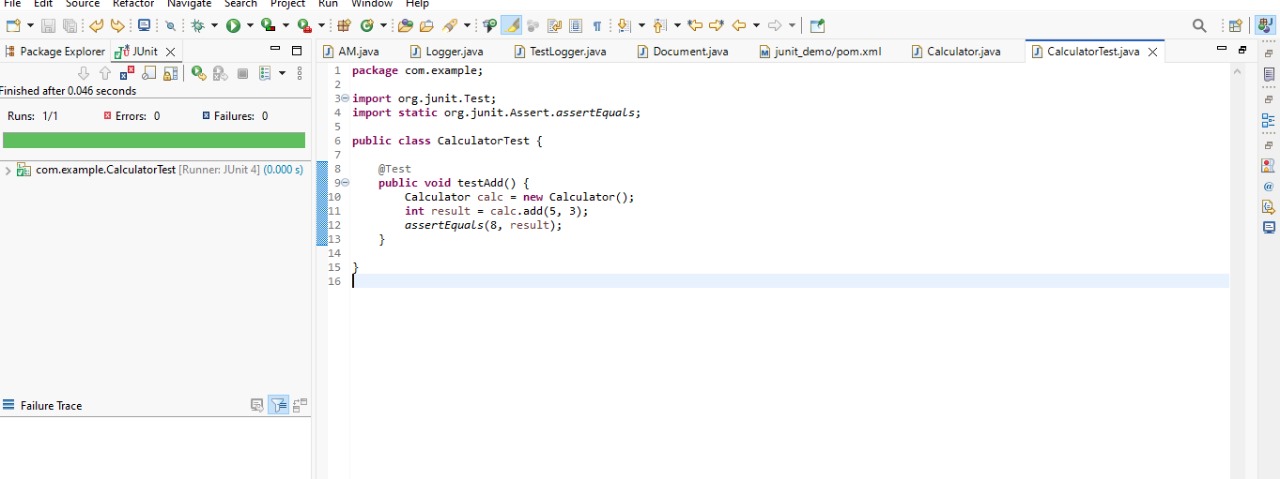
Calculator calc = new Calculator();

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

assertEquals(8, result);

}

}



2. Assertions in JUnit Scenario: You need to use different assertions in JUnit to validate your test results.

Steps:

1. Write tests using various JUnit assertions.

**CODE:**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

public class AssertionsTest {

@Test

public void testAssertions() {

// Assert equals

assertEquals(5, 2 + 3);

// Assert true

assertTrue(5 > 3);

// Assert false

assertFalse(5 < 3);

// Assert null

assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}

A computer screen shot of a computer screen

AI-generated content may be incorrect.

3. 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.

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

**CODE:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

package com.example;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

// Arrange: setup before each test

calculator = new Calculator();

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

}

@After

public void tearDown() {

// Cleanup after each test

calculator = null;

System.out.println("Teardown: Calculator reset");

}

@Test

public void testAddition() {

// Arrange is already done in setUp()

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

// Act

int result = calculator.subtract(10, 4);

// Assert

assertEquals(6, result);

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**MOCKITO EXERCISES**

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:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

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);

}

}

A computer screen shot of a computer code

AI-generated content may be incorrect.

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.

package com.example;

**CODE:**

import static org.mockito.Mockito.\*;

import org.junit.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

// Verify that getData() was called once

verify(mockApi).getData();

}

}

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**SL4J LOGGING EXERCISES**

1. Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**CODE:**

package com.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");

}

}

