**DN 4.0 WEEK 2 MANDATORY PROBLEMS**

**MODULE 3: PL/SQL PROGRAMMING**

**FULL TABLE SCHEMA AND ENTRIES FOR TABLES (MYSQL WORKBENCH 4.0)**

create database fse;

use fse;

-- Create Tables

CREATE TABLE Customers (

CustomerID int PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance int,

LastModified DATE

);

CREATE TABLE Accounts (

AccountID int PRIMARY KEY,

CustomerID int,

AccountType VARCHAR(20),

Balance int,

LastModified DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

TransactionID int PRIMARY KEY,

AccountID int,

TransactionDate DATE,

Amount int,

TransactionType VARCHAR(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

LoanID int PRIMARY KEY,

CustomerID int,

LoanAmount int,

InterestRate int,

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

EmployeeID int PRIMARY KEY,

Name VARCHAR(100),

Position VARCHAR(50),

Salary int,

Department VARCHAR(50),

HireDate DATE

);

-- Insert Data into Customers

INSERT INTO Customers VALUES (1, 'John Smith', '1958-06-15', 12000, CURDATE());

INSERT INTO Customers VALUES (2, 'Alice Johnson', '1975-03-22', 8000, CURDATE());

INSERT INTO Customers VALUES (3, 'Michael Brown', '1960-10-05', 15000, CURDATE());

INSERT INTO Customers VALUES (4, 'Emma Wilson', '1990-02-18', 5000, CURDATE());

INSERT INTO Customers VALUES (5, 'David Miller', '1950-08-30', 20000, CURDATE());

-- Insert Data into Accounts

INSERT INTO Accounts VALUES (1, 1, 'Savings', 7000, CURDATE());

INSERT INTO Accounts VALUES (2, 2, 'Checking', 3000, CURDATE());

INSERT INTO Accounts VALUES (3, 3, 'Savings', 9000, CURDATE());

INSERT INTO Accounts VALUES (4, 4, 'Savings', 4000, CURDATE());

INSERT INTO Accounts VALUES (5, 5, 'Checking', 10000, CURDATE());

-- Insert Data into Transactions

INSERT INTO Transactions VALUES (1, 1, CURDATE() - INTERVAL 20 DAY, 1000, 'Credit');

INSERT INTO Transactions VALUES (2, 2, CURDATE() - INTERVAL 10 DAY, 500, 'Debit');

INSERT INTO Transactions VALUES (3, 3, CURDATE() - INTERVAL 5 DAY, 2000, 'Credit');

INSERT INTO Transactions VALUES (4, 4, CURDATE() - INTERVAL 2 DAY, 1000, 'Debit');

INSERT INTO Transactions VALUES (5, 5, CURDATE() - INTERVAL 1 DAY, 3000, 'Credit');

-- Insert Data into Loans

INSERT INTO Loans VALUES (1, 1, 50000, 7.5, '2023-01-01', CURDATE() + INTERVAL 10 DAY);

INSERT INTO Loans VALUES (2, 2, 30000, 8.0, '2022-06-01', CURDATE() + INTERVAL 40 DAY);

INSERT INTO Loans VALUES (3, 3, 60000, 6.5, '2023-03-01', CURDATE() + INTERVAL 20 DAY);

INSERT INTO Loans VALUES (4, 4, 25000, 9.0, '2023-05-01', CURDATE() + INTERVAL 60 DAY);

INSERT INTO Loans VALUES (5, 5, 70000, 7.0, '2022-09-01', CURDATE() + INTERVAL 5 DAY);

-- Insert Data into Employees

INSERT INTO Employees VALUES (1, 'Robert King', 'Manager', 80000, 'Loans', '2015-04-10');

INSERT INTO Employees VALUES (2, 'Susan Green', 'Teller', 40000, 'Retail', '2018-06-15');

INSERT INTO Employees VALUES (3, 'William Davis', 'Analyst', 55000, 'Risk', '2019-09-20');

INSERT INTO Employees VALUES (4, 'Linda Moore', 'Clerk', 35000, 'Retail', '2020-01-05');

INSERT INTO Employees VALUES (5, 'James Wilson', 'Executive', 65000, 'Operations', '2017-07-25');

**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.**

**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.**

**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.**

**PROGRAM:**

ALTER TABLE Customers ADD COLUMN IsVIP BOOLEAN DEFAULT FALSE;

SET SQL\_SAFE\_UPDATES = 0;

-- Scenario 1:

UPDATE Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

SET l.InterestRate = l.InterestRate - 1

WHERE TIMESTAMPDIFF(YEAR, c.DOB, CURDATE()) > 60;

-- Scenario 2:

UPDATE Customers

SET IsVIP = TRUE

WHERE Balance > 10000;

-- Scenario 3:

SELECT CONCAT('Reminder: Dear ', c.Name, ', your loan ID ', l.LoanID,

' is due on ', DATE\_FORMAT(l.EndDate, '%d-%b-%Y')) AS Reminder

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

**AS Scenario 1 and 2 will no produce any output those will just say how many rows are affected and scenario 3 will produce an output as follows**

'Reminder: Dear David Miller, your loan ID 5 is due on 04-Jul-2025'

**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.**

**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.**

**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.**

**PROGRAM:**

--Scenario 1:

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

END;

--Scenario 2:

CREATE PROCEDURE UpdateEmployeeBonus(IN dept VARCHAR(50), IN bonus\_percent DECIMAL(5,2))

BEGIN

UPDATE Employees

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

WHERE Department = dept;

END;

--Scenario 3:

CREATE PROCEDURE TransferFunds(

IN sourceAccount INT,

IN targetAccount INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE source\_balance DECIMAL(10,2);

SELECT Balance INTO source\_balance

FROM Accounts

WHERE AccountID = sourceAccount;

IF source\_balance >= amount THEN

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = sourceAccount;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = targetAccount;

ELSE

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Insufficient funds in source account';

END IF;

END

**None of above scenario codes will produce output until we write a update statement separately to execute the above procedure.**

**MODULE 4: TDD USING JUNIT5 AND MOCKITO**

**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**

**3. Create a new test class in your project.**

PROGRAM:

CalculatorTest.java:

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS junit:junit:4.13.2

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

public int add(int a, int b) {

return a + b;

}

@Test

public void testAdd() {

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

}

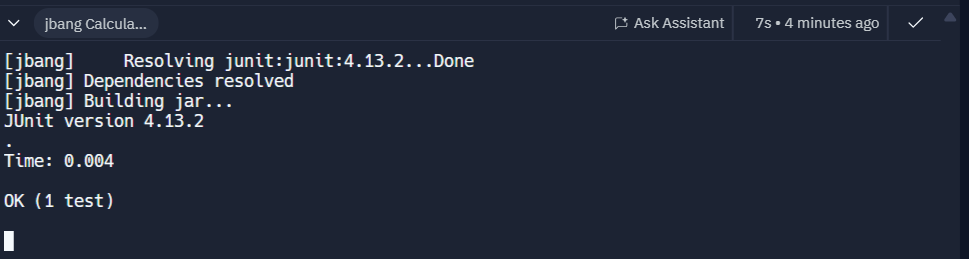
public static void main(String[] args) {

org.junit.runner.JUnitCore.main("CalculatorTest");

}

}

**OUTPUT:**

****

**HERE I AM NOT USING ANY IDE LIKE ECPLISE OR ANY OTHER I AM USING REPLIT WHERE I CRETED A REPL CALL JBANG WHICH WILL CONTAIN ALL THE REQURIED DEPENDENCIES SO I CREATED A SIMPLE TEST CALCULATOR CODE WITH ONE TEST CASE AND THE ABOVE OUTPUT SAYS THAT 1 TEST CASE EXECUTED SUCCESSFULLY.**

**Exercise 3: Assertions in JUnit Scenario:**

**You need to use different assertions in JUnit to validate your test results.**

**PROGRAM:**

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS junit:junit:4.13.2

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertEquals() {

assertEquals(5, 2 + 3);

}

@Test

public void testAssertTrue() {

assertTrue(5 > 3);

}

@Test

public void testAssertFalse() {

assertFalse(5 < 3);

}

@Test

public void testAssertNull() {

assertNull(null);

}

@Test

public void testAssertNotNull() {

assertNotNull(new Object());

}

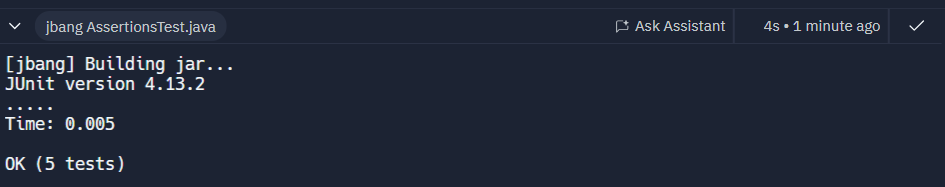
public static void main(String[] args) {

org.junit.runner.JUnitCore.main("AssertionsTest");

}

}

**OUTPUT:**

****

**HERE I CREATED 5 DIFFERENT TEST CASE WITH ASSERTIONS SO HERE THE OUT TELLS THAT ALL THE 5 TEST CASES CREATED ARE SUCCESSFULLY EXECUTED.**

**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.**

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

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

**PROGRAM:**

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS junit:junit:4.13.2

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

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

}

@After

public void tearDown() {

calculator = null;

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

}

@Test

public void testAdd() {

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

assertEquals(5, result);

}

@Test

public void testSubtract() {

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

assertEquals(3, result);

}

public static void main(String[] args) {

org.junit.runner.JUnitCore.main("CalculatorTest");

}

}

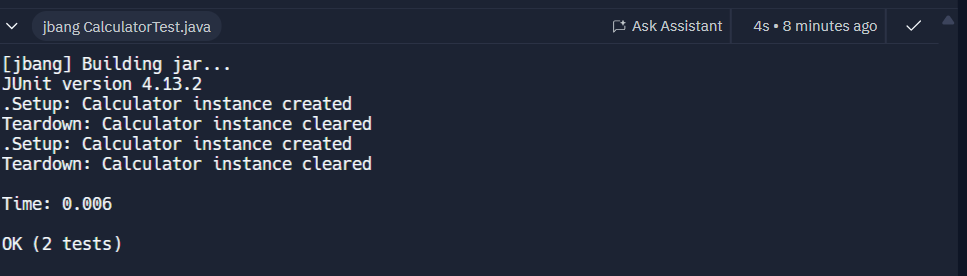
class Calculator {

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

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

}

**OUTPUT:**

****

**HERE ACCORDING AAA WHICH MEANS ARRANGE-ACT-ASSERT HERE I CLEARLY DID GAVE EXPLANTION ABOUT THOSE ACTIONS DOWN**

@Test

public void testAdd() {

// Arrange

Calculator calc = new Calculator();

// Act

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

// Assert

assertEquals(5, result);

}

Arrange: Calculator calc = new Calculator();

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

Assert: assertEquals(5, result);

**SO, THIS PATTERN KEEPS THE TEST CASE CLEAR AND UNDERSTANDABLE.**

**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.**

**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.**

**PROGRAM:**

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS junit:junit:4.13.2

//DEPS org.mockito:mockito-core:4.11.0

import org.junit.Test;

import org.junit.runner.JUnitCore;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

@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);

}

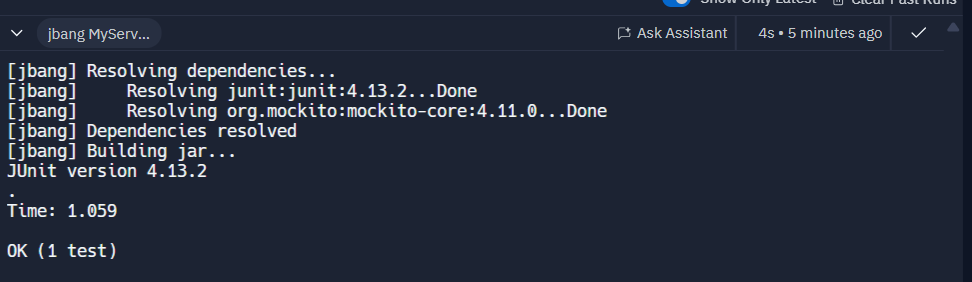
public static void main(String[] args) {

JUnitCore.main("MyServiceTest");

}

}

**OUTPUT:**

****

**HERE I WROTE THE CODE USING JUnit 4 AND Mockito**

**STEP 1: Create a mock object:**

ExternalApi mockApi = mock(ExternalApi.class);

**STEP 2: Stub methods:**

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

**STEP 3: Write test using mock:**

MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

**Exercise 2: Verifying Interactions**

**Scenario:**

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

**1. Create a mock object.**

**2. Call the method with specific arguments.**

**3. Verify the interaction.**

**PROGRAM:**

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS junit:junit:4.13.2

//DEPS org.mockito:mockito-core:4.11.0

import org.junit.Test;

import org.junit.runner.JUnitCore;

import static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

public class MyServiceVerifyTest {

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

api.getData();

}

}

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

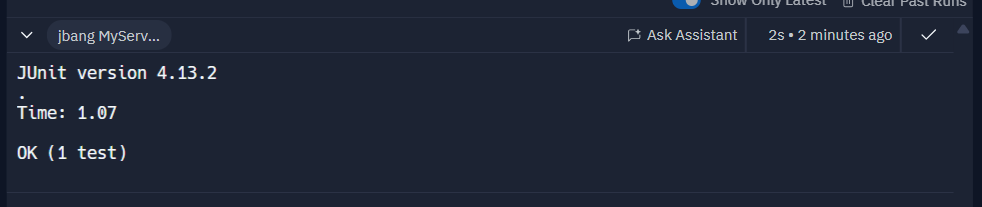
public static void main(String[] args) {

JUnitCore.main("MyServiceVerifyTest");

}

}

**OUTPUT:**

****

**THESE ARE THE STEPS THAT I USED**

**STEP 1: Create mock object:**

ExternalApi mockApi = mock(ExternalApi.class);

**STEP 2: Call method with specific arguments:**

service.fetchData();

fetchData() calls mockApi.getData()

**STEP 3: Verify interaction:**

verify(mockApi).getData();

**MODULE 5: SLF4J LOGGING FRAMEWORK**

**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:**
    2. **Create a Java class that uses SLF4J for logging**

**PROGRAM:**

///usr/bin/env jbang "$0" "$@" ; exit $?

//DEPS org.slf4j:slf4j-api:1.7.30

//DEPS ch.qos.logback:logback-classic:1.2.3

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) {

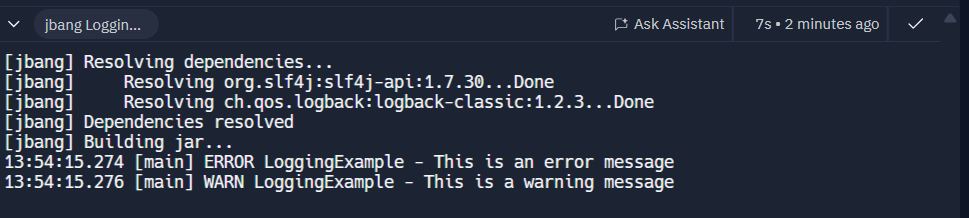
logger.error("This is an error message");

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

}

}

**OUTPUT:**

****

**HERE I AM USING REPLIT SO THERE IS NO NEED OF XML FILE SO THERE IS NO NEED OF PON.XML AND NEXT CREATING A JAVA CLASSES THAT USES SLF4J FOR LOGGING.**

* **I created a Logger instance using LoggerFactory.**
* **Then I called methods like error() and warn() to write logs.**
* **SLF4J delegates to Logback (added in step 1) to actually print logs to the console.**

**NOW, HERE Java code uses SLF4J API and then SLF4J delegates to Logback after that logs get printed.**