**Week 2**

**TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up JUnit**

**CODE:**

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalucaltorTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

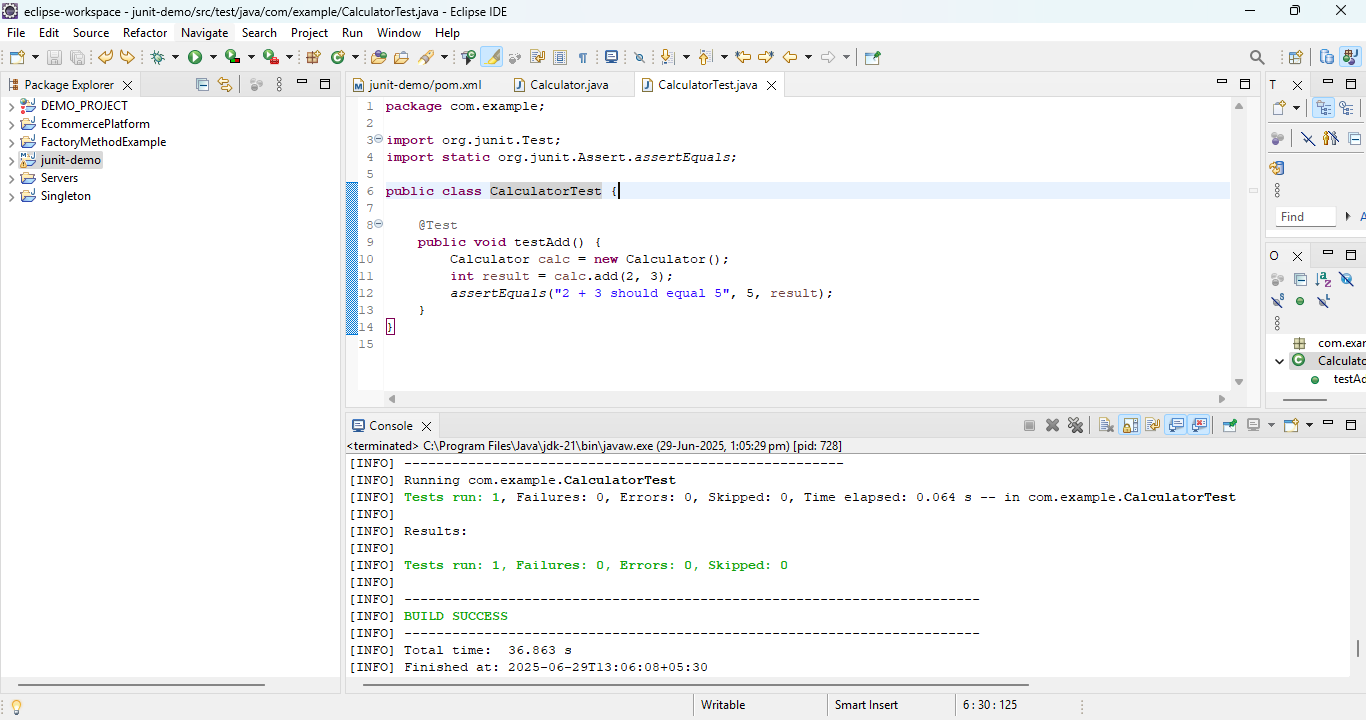
int result = calc.add(2, 3);

assertEquals("2 + 3 should equal 5", 5, result);

}

}

OUTPUT:



**Exercise 3: Assertions in JUnit**

CODE:

package com.example;

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);

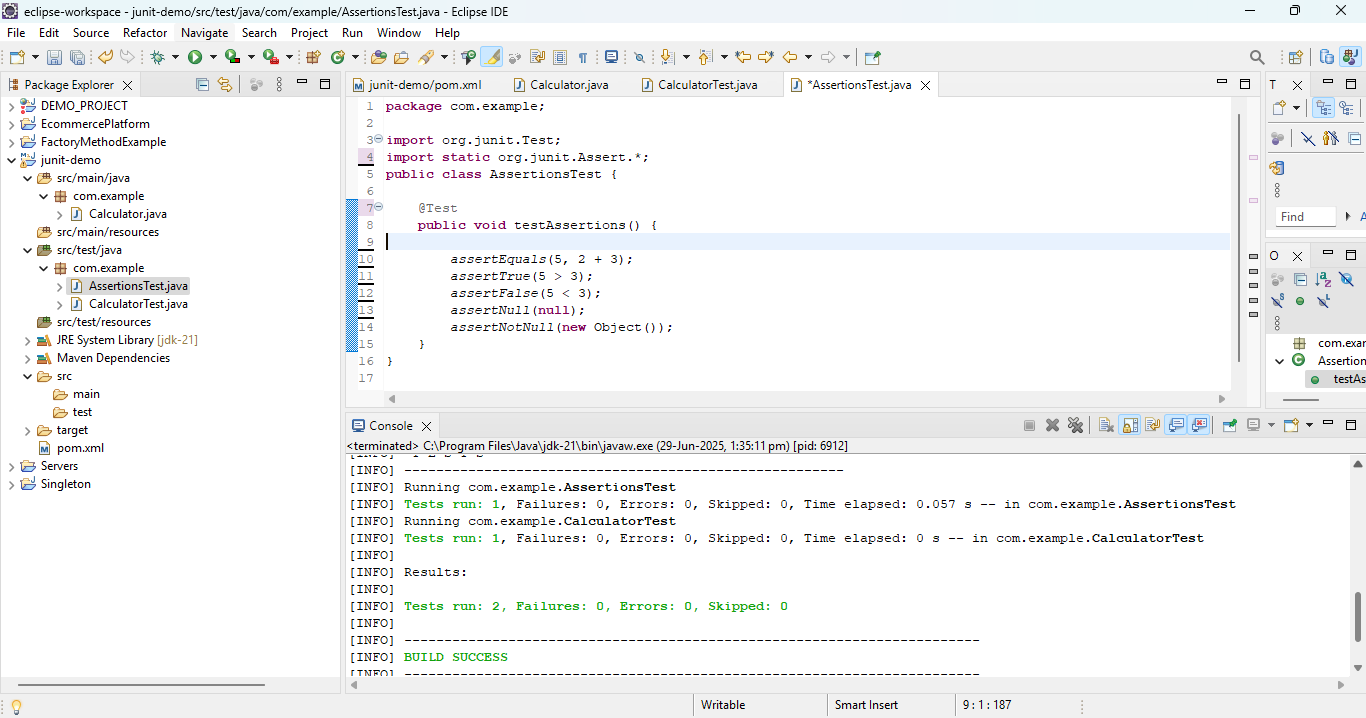
assertNull(null);

assertNotNull(new Object());

}

}

OUTPUT:



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

**CODE:**

**Arithmetic.java**

package com.example;

public class Arithmetic {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

public void close() {

}

}

ArithmeticOperationTest.java

package com.example;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

import static org.junit.Assert.\*;

public class ArithmeticOperationsTest {

private Arithmetic arithmetic;

@Before

public void setUp() {

arithmetic = new Arithmetic();

}

@After

public void tearDown() {

arithmetic.close();

arithmetic = null;

}

@Test

public void add\_twoPositiveNumbers\_returnsSum() {

int a = 2, b = 3;

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

assertEquals("2 + 3 should equal 5", 5, result);

}

@Test

public void subtract\_smallerFromBigger\_returnsPositive() {

int a = 10, b = 4;

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

assertEquals("10 - 4 should equal 6", 6, result);

}

@Test

public void subtract\_biggerFromSmaller\_returnsNegative() {

int a = 4, b = 10;

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

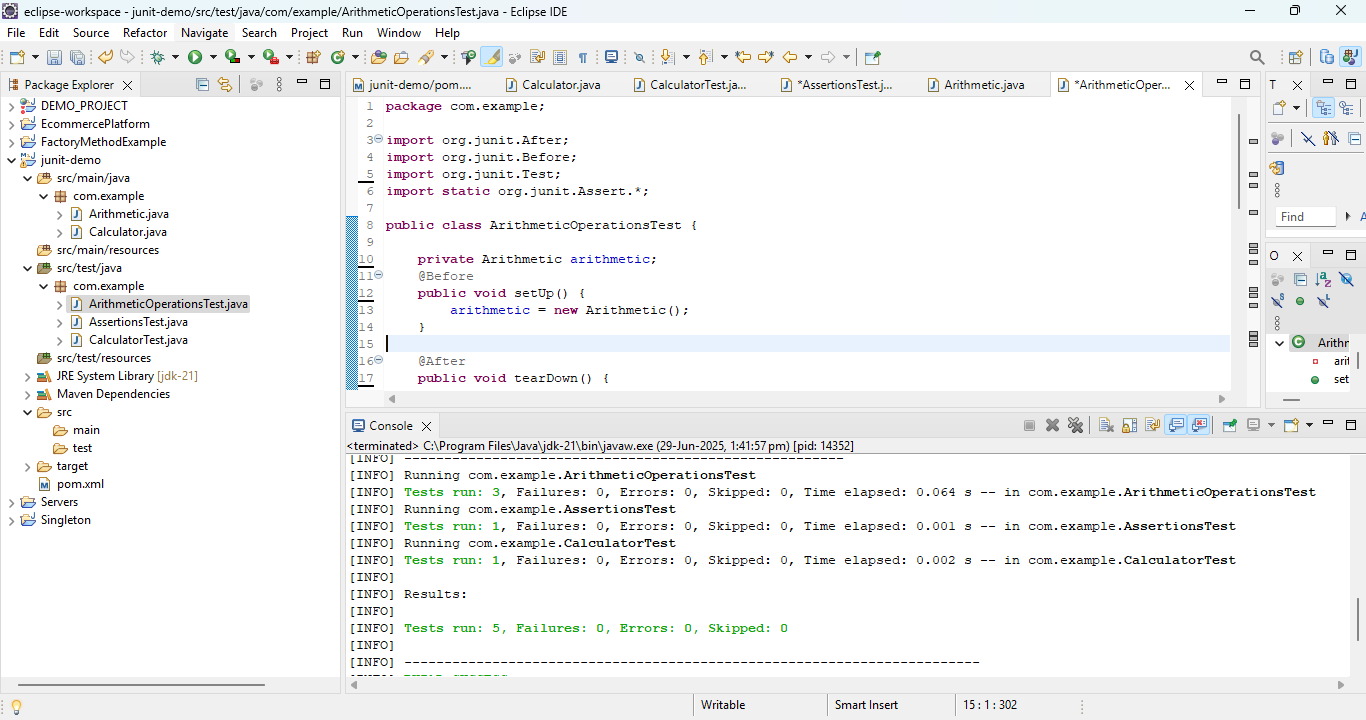
assertEquals(-6, result);

assertTrue("Result should be negative", result < 0);

}

}

**OUTPUT:**



**Exercise 1: Mocking and Stubbing**

**CODE:**

**ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

MyService.java

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

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

public class MyServiceTest {

@Test

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

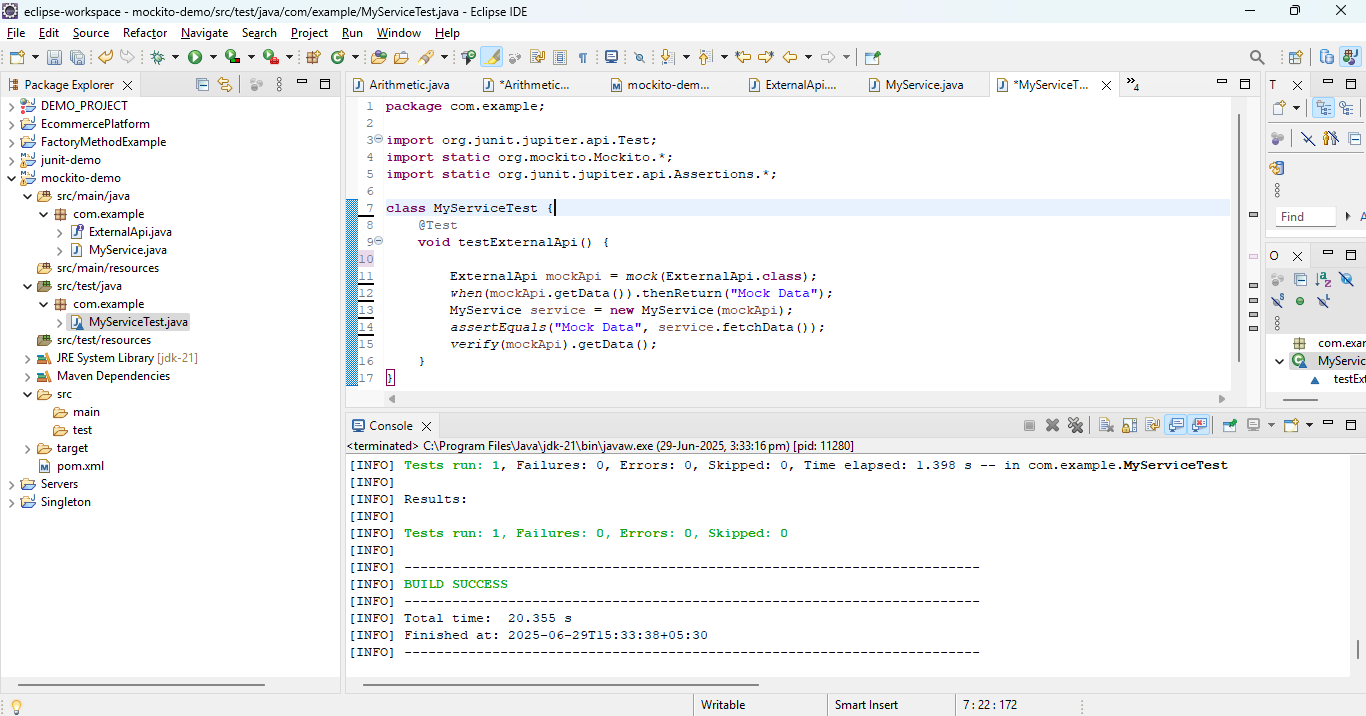
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**CODE:**

**ArithmeticApi.java**

**package** com.example;

**public** **interface** ArithmeticApi {

String getData();

}

**ArithmeticService.java**

package com.example;

public class ArithmeticService {

private final ArithmeticApi api;

public ArithmeticService(ArithmeticApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**ArithmeticServiceTest.java**

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

class ArithmeticServiceTest {

@Test

void testVerifyInteraction() {

ArithmeticApi mockApi = mock(ArithmeticApi.class);

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

ArithmeticService service = new ArithmeticService(mockApi);

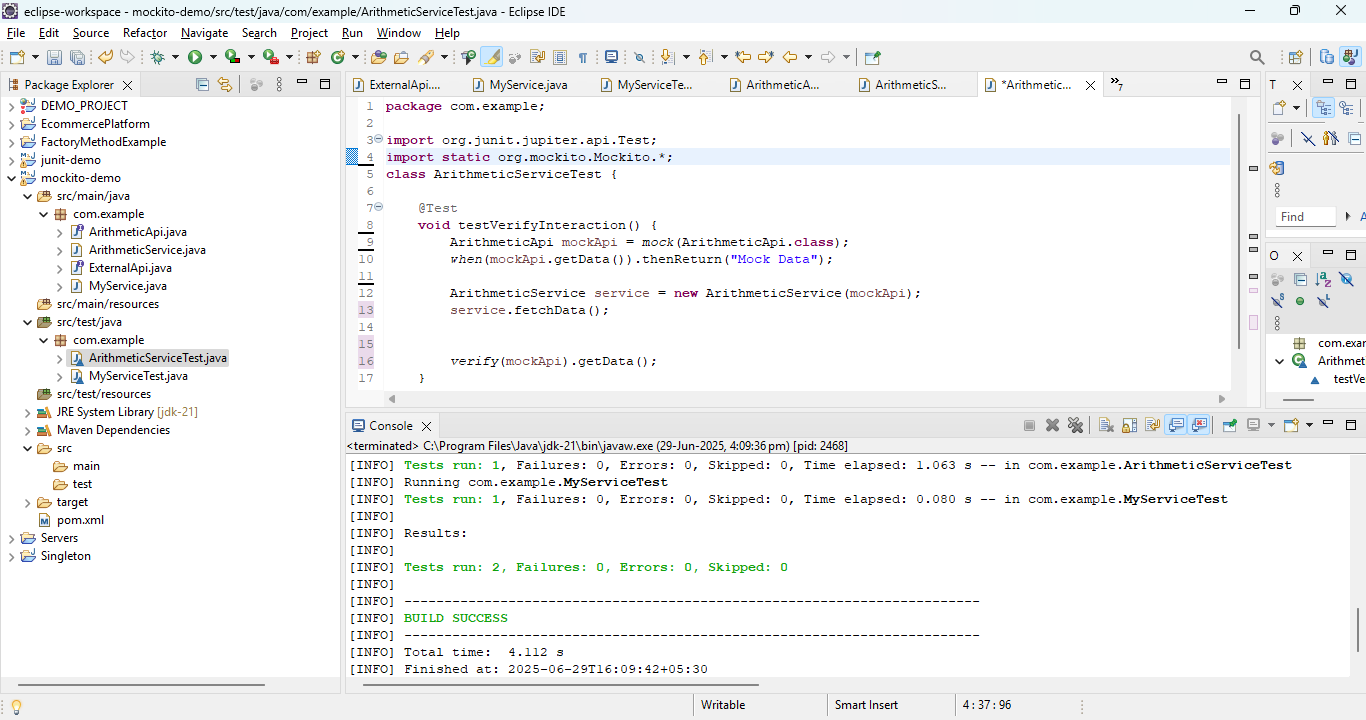
service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**



**SLF4J logging framework**

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

**CODE:**

**LoggingExample.java**

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) {

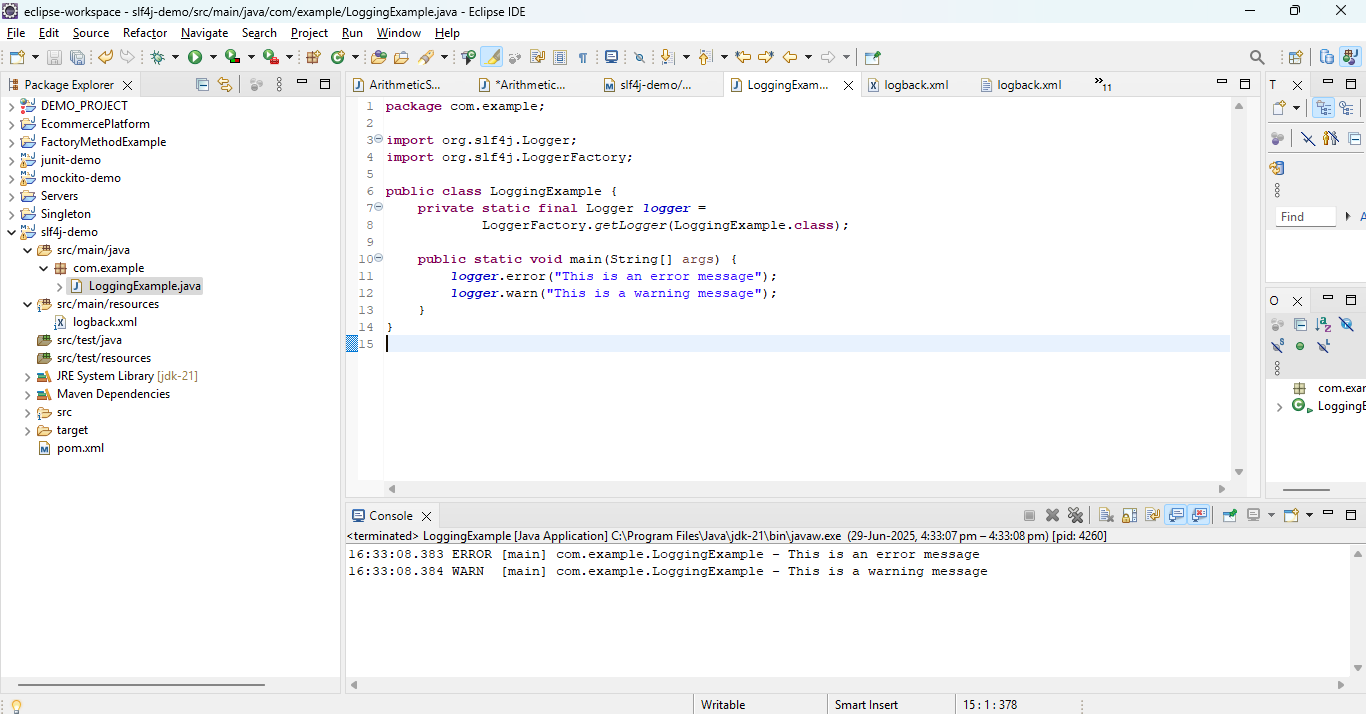
logger.error("This is an error message");

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

}

}

**OUTPUT:**



**PL/SQL programming**

**Exercise 1: Control Structures**

**CODE:**

CREATE TABLE customers (

customer\_id INT PRIMARY KEY,

age INT,

interest\_rate DECIMAL(5,3),

balance DECIMAL(10,2),

isvip BOOLEAN DEFAULT FALSE

);

CREATE TABLE loans (

loan\_id INT PRIMARY KEY,

customer\_id INT,

due\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES customers(customer\_id)

);

INSERT INTO customers VALUES

(101, 58, 0.075, 8000, FALSE),

(102, 62, 0.080, 12000, FALSE),

(103, 66, 0.065, 9500, FALSE),

(104, 45, 0.070, 15000, FALSE);

INSERT INTO loans VALUES

(201, 101, CURDATE() + INTERVAL 10 DAY),

(202, 102, CURDATE() + INTERVAL 28 DAY),

(203, 103, CURDATE() + INTERVAL 40 DAY);

**1.Scenario:**

UPDATE customers

SET interest\_rate = interest\_rate - 0.01

WHERE age > 60;

**2.Scanerio:**

UPDATE customers

SET isvip = TRUE

WHERE balance > 10000;

**3.Scenario:**

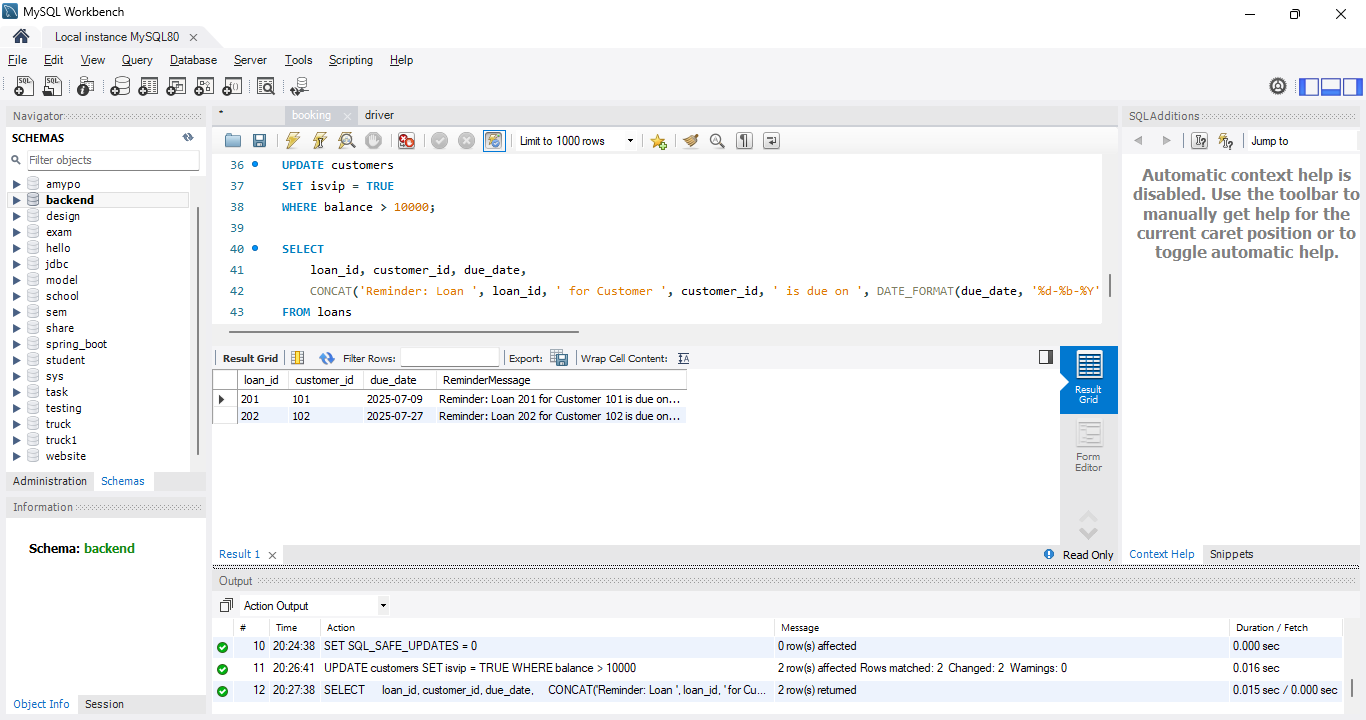
SELECT loan\_id, customer\_id, due\_date,

CONCAT('Reminder: Loan ', loan\_id, ' for Customer ', customer\_id, ' is due on ', DATE\_FORMAT(due\_date, '%d-%b-%Y')) AS ReminderMessage

FROM loans

WHERE due\_date BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

**OUTPUT:**



**Exercise 3: Stored Procedures**

**CODE:**

CREATE TABLE savings\_accounts (

account\_id INT PRIMARY KEY,

balance DECIMAL(12,2)

);

INSERT INTO savings\_accounts VALUES (1, 1000), (2, 1500), (3, 2000);

CREATE TABLE employees (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

department\_id INT,

salary DECIMAL(10,2)

);

INSERT INTO employees VALUES

(1, 'Alice', 10, 50000),

(2, 'Bob', 10, 55000),

(3, 'Charlie', 20, 60000);

CREATE TABLE accounts (

account\_id INT PRIMARY KEY,

customer\_id INT,

balance DECIMAL(12,2)

);

INSERT INTO accounts VALUES

(1, 101, 5000),

(2, 101, 2000),

(3, 102, 1000);

1.Scenario:

DELIMITER $$

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

UPDATE savings\_accounts

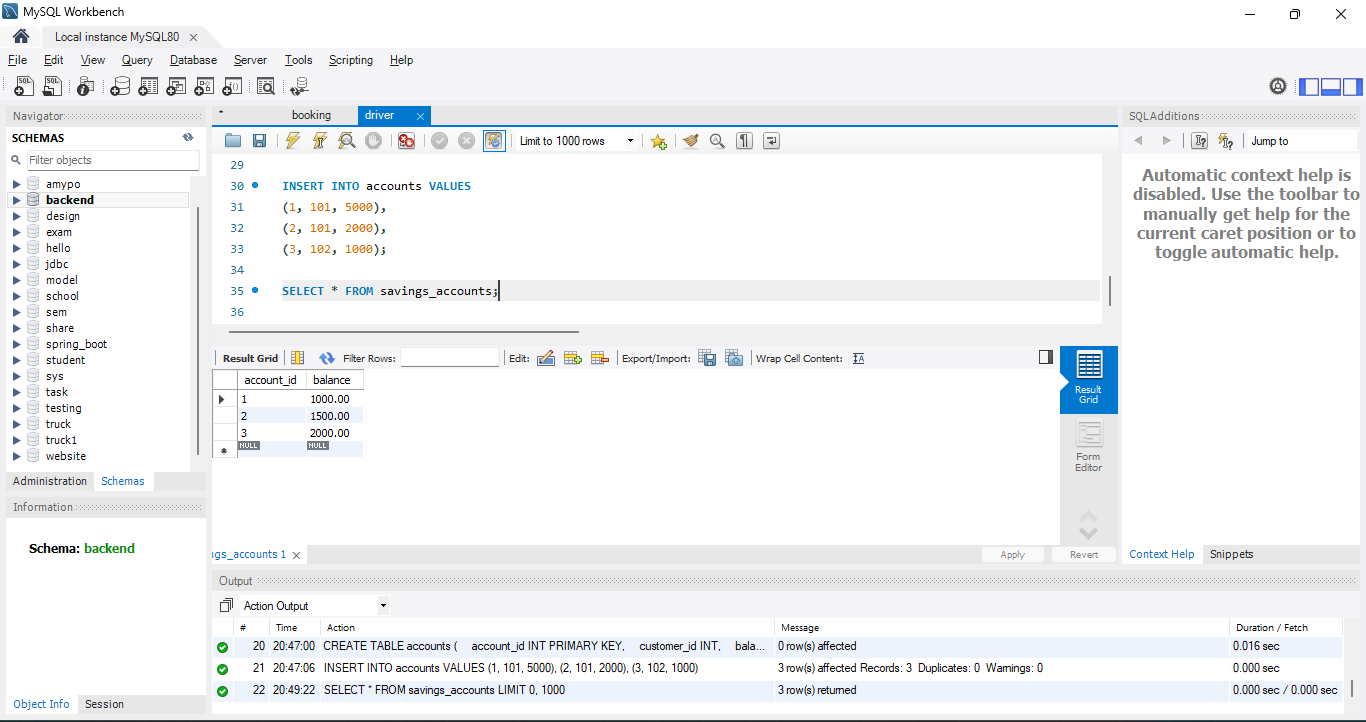
SET balance = balance + (balance \* 0.01);

END$$

DELIMITER ;

SELECT \* FROM savings\_accounts;

**OUTPUT:**



2.Scenario:

DELIMITER $$

CREATE PROCEDURE UpdateEmployeeBonus(

IN dept\_id INT,

IN bonus\_percent DECIMAL(5,2)

)

BEGIN

UPDATE employees

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

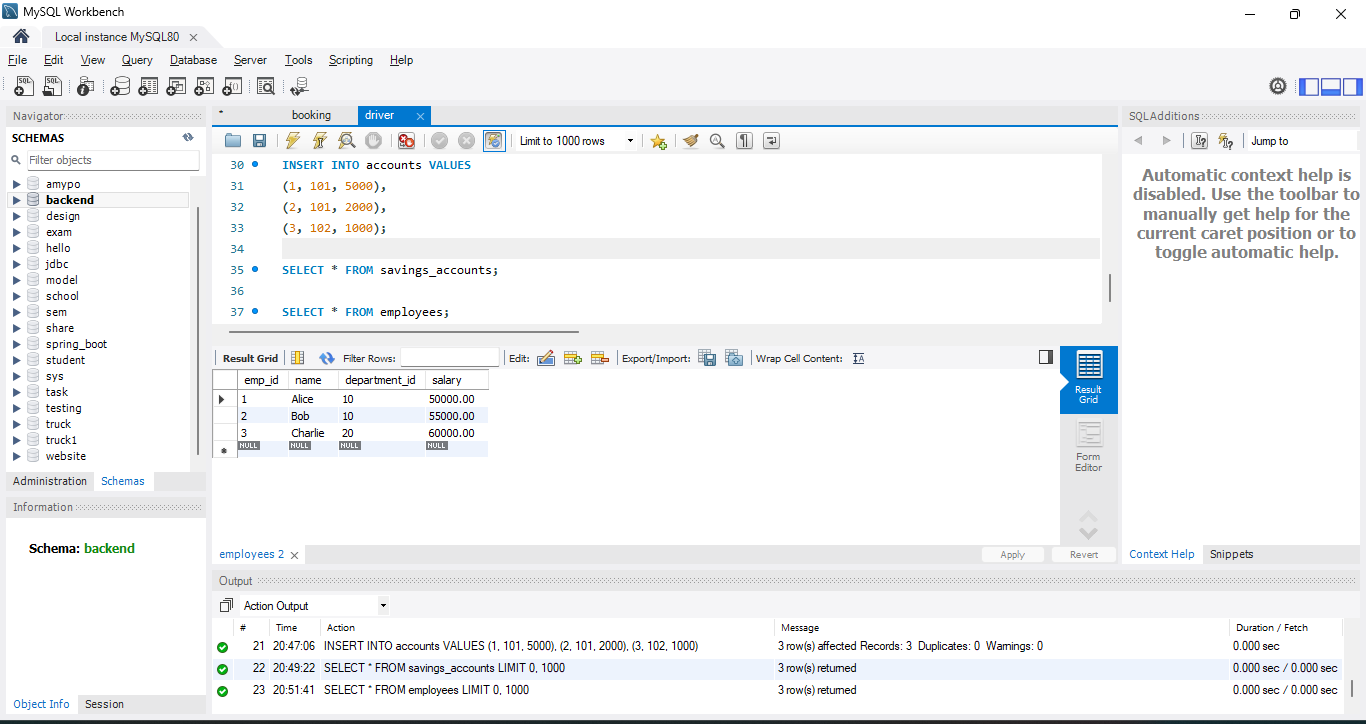
WHERE department\_id = dept\_id;

END$$

DELIMITER ;

SELECT \* FROM employees;

OUTPUT:



3.Scenario:

DELIMITER $$

CREATE PROCEDURE TransferFunds(

IN from\_account INT,

IN to\_account INT,

IN amount DECIMAL(10,2)

)

BEGIN

DECLARE from\_balance DECIMAL(12,2);

SELECT balance INTO from\_balance

FROM accounts

WHERE account\_id = from\_account;

IF from\_balance >= amount THEN

UPDATE accounts

SET balance = balance - amount

WHERE account\_id = from\_account;

UPDATE accounts

SET balance = balance + amount

WHERE account\_id = to\_account;

ELSE

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Insufficient balance for transfer';

END IF;

END$$

DELIMITER ;

SELECT \* FROM accounts;

OUTPUT:

