**SUPERSET ID: 6421553**

**WEEK 2**

**PL/SQL programming**

**Exercise 1: Control Structures**

**Scenario 1 :**

**Code :**  
  
DELIMITER //

CREATE PROCEDURE ApplyInterestDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_CustomerID INT;

DECLARE v\_DOB DATE;

DECLARE cur CURSOR FOR SELECT CustomerID, DOB FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_CustomerID, v\_DOB;

IF done THEN

LEAVE read\_loop;

END IF;

IF TIMESTAMPDIFF(YEAR, v\_DOB, CURDATE()) > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = v\_CustomerID;

END IF;

END LOOP;

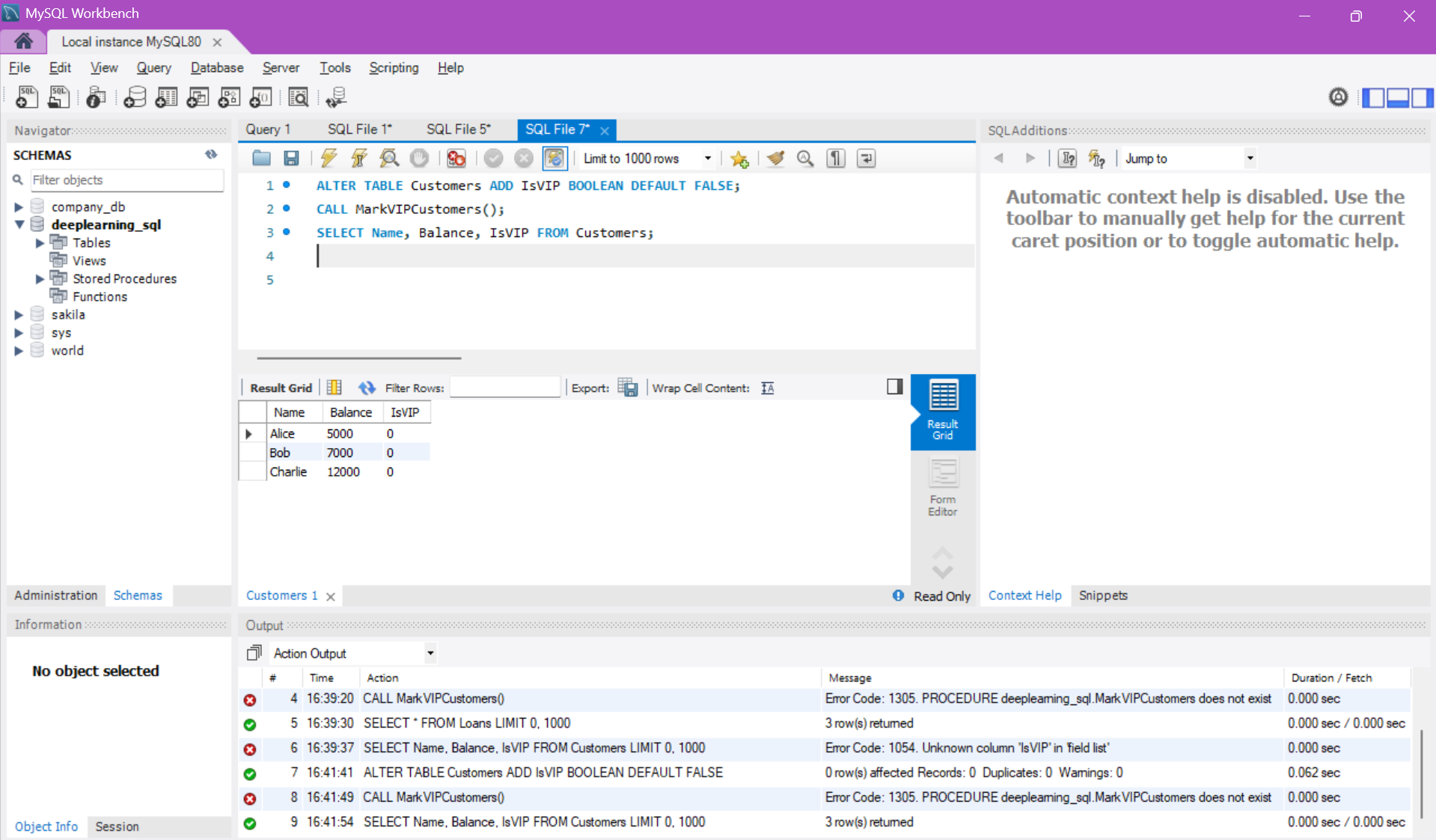
CLOSE cur;

END //

DELIMITER ;

-- Call the procedure:

CALL ApplyInterestDiscount();  
  
**Result :**



**Scenario 2 :**

**Code :**  
  
  
DELIMITER //

CREATE PROCEDURE MarkVIPCustomers()

BEGIN

UPDATE Customers

SET IsVIP = TRUE

WHERE Balance > 10000;

UPDATE Customers

SET IsVIP = FALSE

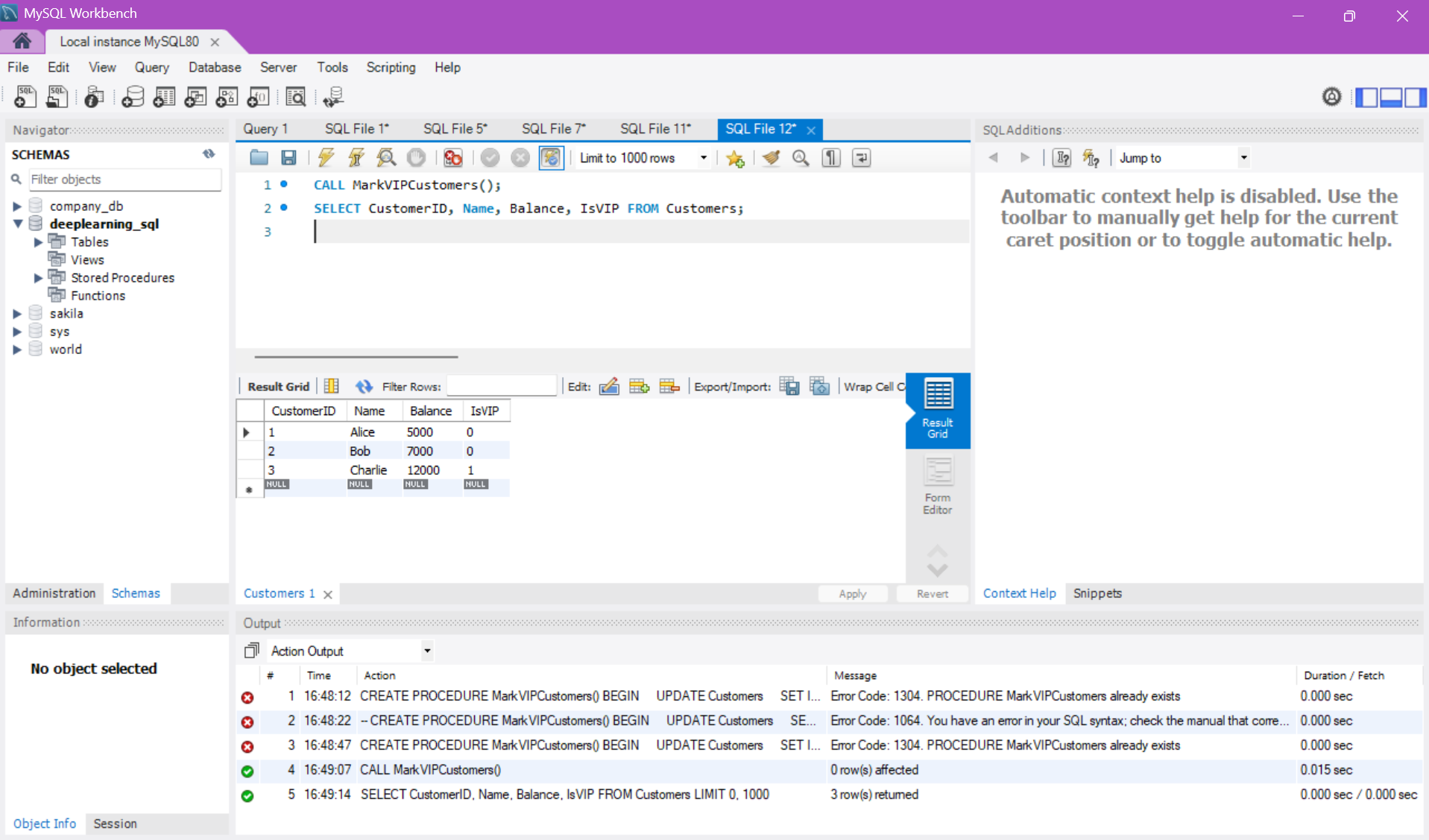
WHERE Balance <= 10000;

END //

DELIMITER ;

-- Call the procedure:

CALL MarkVIPCustomers();

**Result:** 

**Scenario 3 :**

**Code :**  
  
DELIMITER //

CREATE PROCEDURE LoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE v\_Name VARCHAR(100);

DECLARE v\_LoanID INT;

DECLARE v\_EndDate DATE;

DECLARE cur CURSOR FOR

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

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO v\_Name, v\_LoanID, v\_EndDate;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Reminder: Dear ', v\_Name,

', your loan ID ', v\_LoanID,

' is due on ', DATE\_FORMAT(v\_EndDate, '%d-%b-%Y')) AS Reminder;

END LOOP;

CLOSE cur;

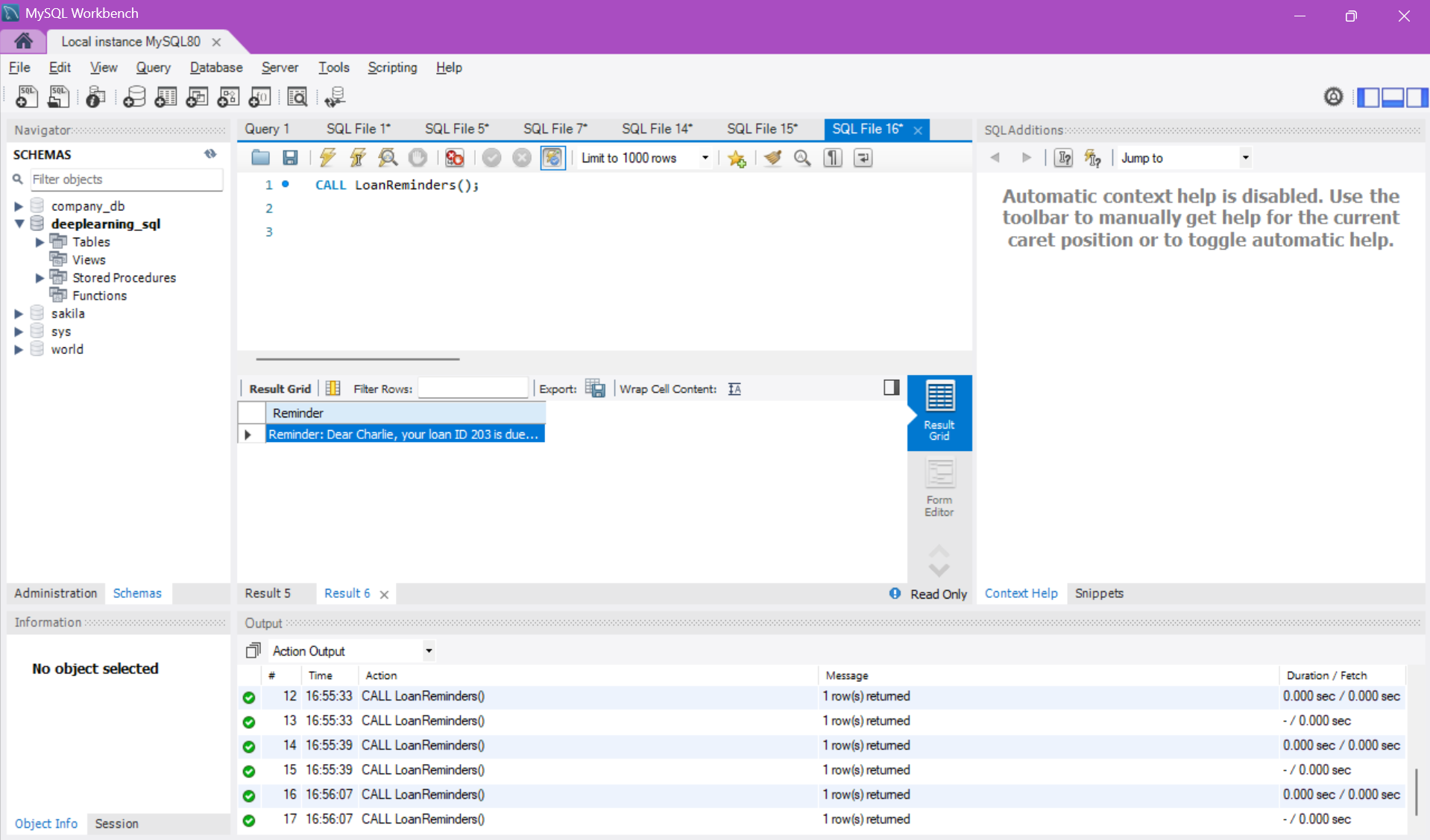
END //

DELIMITER ;

-- Call the procedure:

CALL LoanReminders();

**Result:**

****

**Exercise 2: Stored Procedures**

**Scenario 1 :**

**Code :**

DELIMITER //

CREATE PROCEDURE ProcessMonthlyInterest()

BEGIN

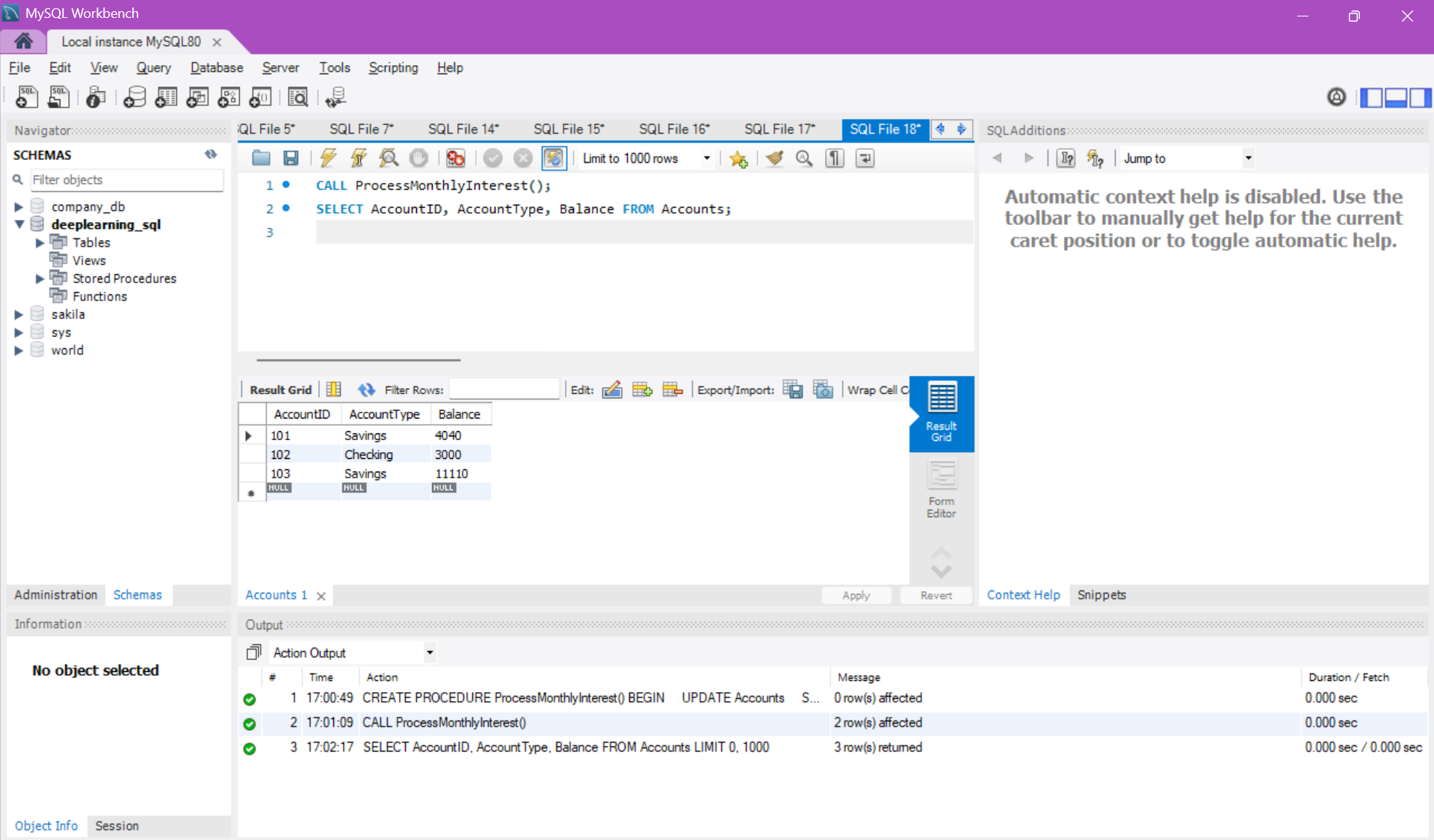
UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

END //

DELIMITER ;  
  
**Result:**



**Scenario 2 :**

**Code :**

DELIMITER //

CREATE PROCEDURE UpdateEmployeeBonus(

IN dept\_name VARCHAR(50),

IN bonus\_percent DECIMAL(5,2)

)

BEGIN

UPDATE Employees

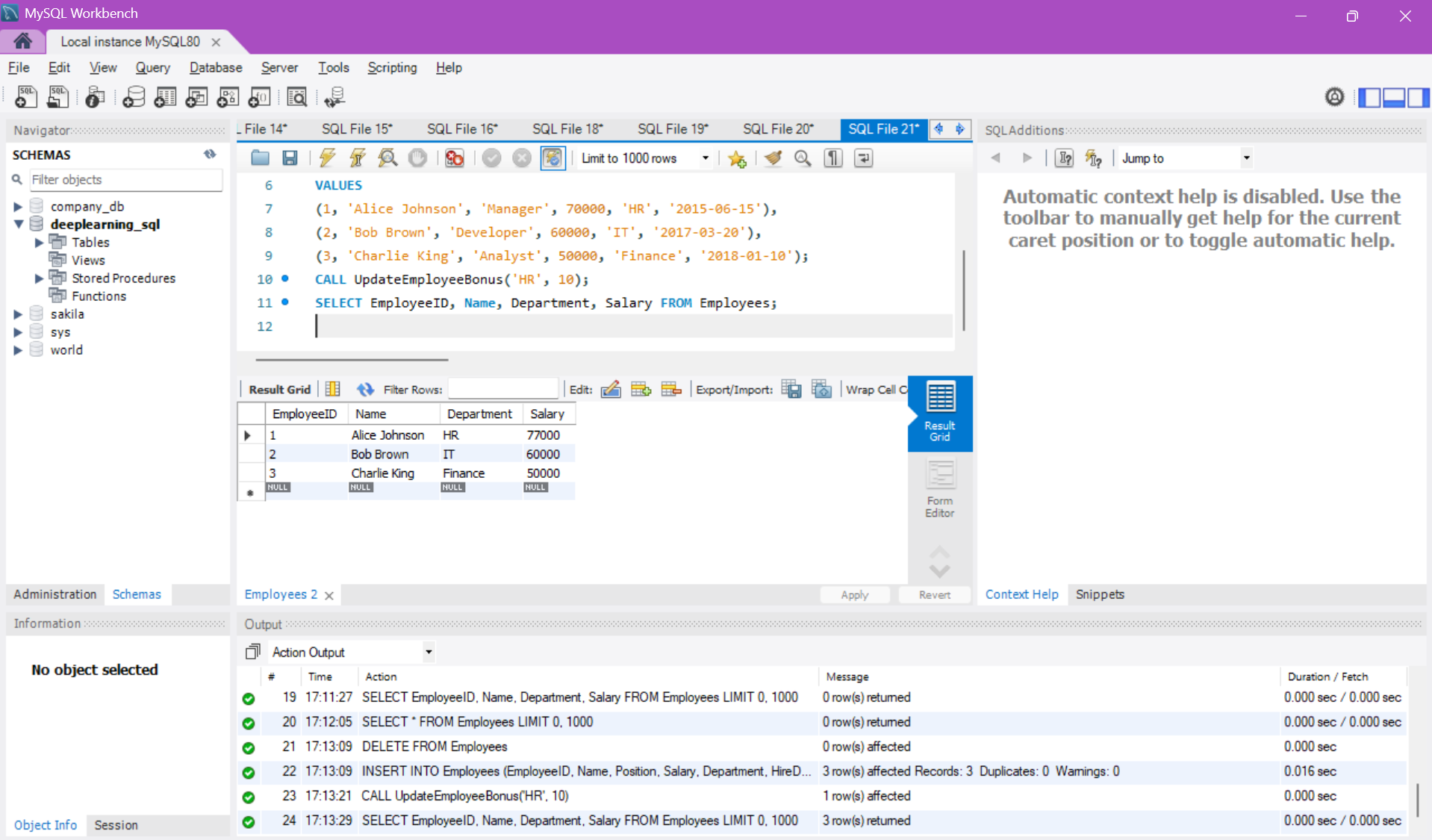
SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE Department = dept\_name;

END //

DELIMITER ;

**Result :**

****

**Scenario 3 :**

**Code :**

DROP PROCEDURE IF EXISTS TransferFunds;

DELIMITER //

CREATE PROCEDURE TransferFunds(

IN from\_account\_id INT,

IN to\_account\_id INT,

IN transfer\_amount DECIMAL(10,2)

)

BEGIN

DECLARE from\_balance DECIMAL(10,2);

-- Get the current balance of the source account

SELECT Balance INTO from\_balance

FROM Accounts

WHERE AccountID = from\_account\_id;

-- Check if there is enough balance

IF from\_balance >= transfer\_amount THEN

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - transfer\_amount

WHERE AccountID = from\_account\_id;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + transfer\_amount

WHERE AccountID = to\_account\_id;

ELSE

-- Raise error if insufficient balance

SIGNAL SQLSTATE '45000'

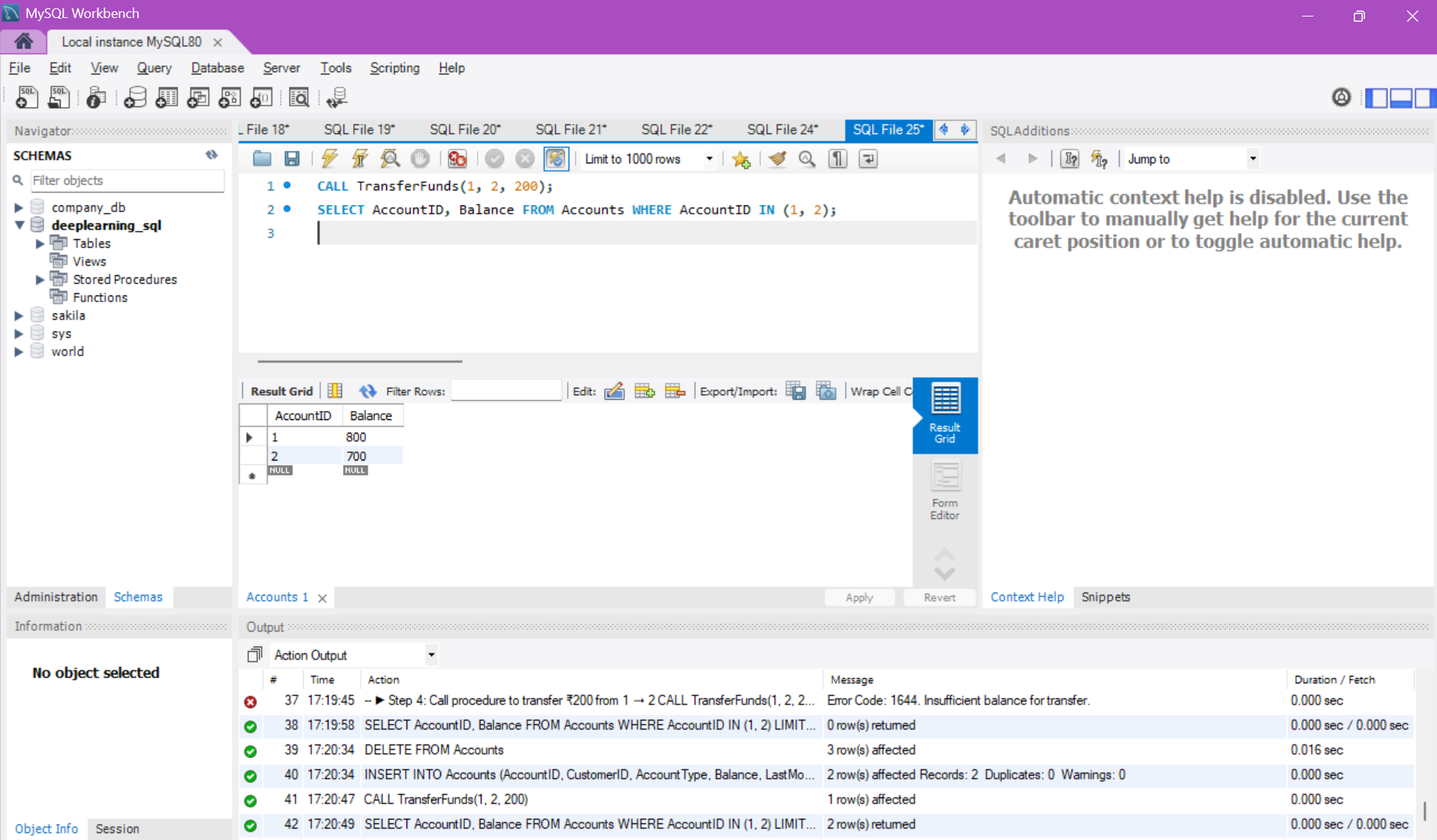
SET MESSAGE\_TEXT = 'Insufficient balance for transfer.';

END IF;

END //

DELIMITER ;

**Result:**

****

**TDD using JUnit5 and Mockito**

**Exercise 1: Setting Up Junit**

**pom.xml**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Code:**

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

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

assertEquals(5, result);

}

}

**OUTPUT:**

****

**EXERCISE 3: ASSERTIONS IN JUNIT**

**AssertionsTest.java**

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);

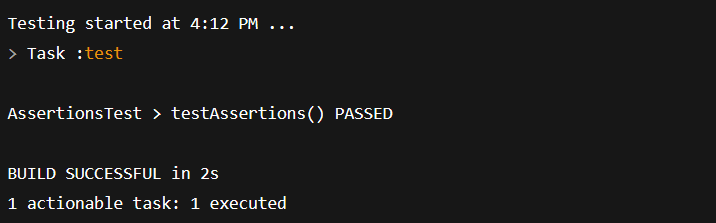
assertNull("Object should be null", null);

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

}

}

**OUTPUT:**

****

****

**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;

@Before

public void setUp() {

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

calculator = new Calculator();

}

@After

public void tearDown() {

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

calculator = null;

}

@Test

public void testAdd() {

int result = calculator.add(10, 5);

assertEquals("10 + 5 should be 15", 15, result);

}

@Test

public void testSubtract() {

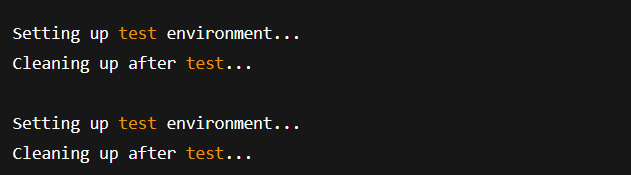
int result = calculator.subtract(10, 5);

assertEquals("10 - 5 should be 5", 5, result);

}

}

**OUTPUT:**

****

**3. MOCKITO EXERCISES**

**EXERCISE 1: MOCKING AND STUBBING**

public interface ExternalApi {

String getData();

}

**Service class**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Test class**

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);

}

}

**OUTPUT:**

****

**EXERCISE 2: VERIFYING INTERACTIONS**

public interface ExternalApi {

String getData();

}

**Service class**

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**Test class**

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();

}

}

**OUTPUT:**

****

**6.SL4J LOGGING EXERCISES**

**EXERCISE 1: LOGGING ERROR MESSAGES AND WARNING LEVELS TASK**

**pom.xml**

<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>

**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");

logger.info("This is an informational message");

logger.debug("This is a debug message (may not appear unless debug is enabled)");

}

}

**logback.xml**

<configuration>

<appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss} [%level] %logger - %msg%n</pattern>

</encoder>

</appender>

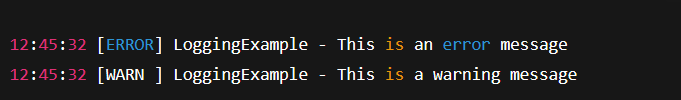
<root level="debug">

<appender-ref ref="STDOUT" />

</root>

</configuration>

**OUTPUT:**

****