**WEEK 2**

**PL/SQL programming**

**TABLE CREATION**

**CODE:**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    AccountType VARCHAR2(20),

    Balance NUMBER,

    LastModified DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Transactions (

    TransactionID NUMBER PRIMARY KEY,

    AccountID NUMBER,

    TransactionDate DATE,

    Amount NUMBER,

    TransactionType VARCHAR2(10),

    FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

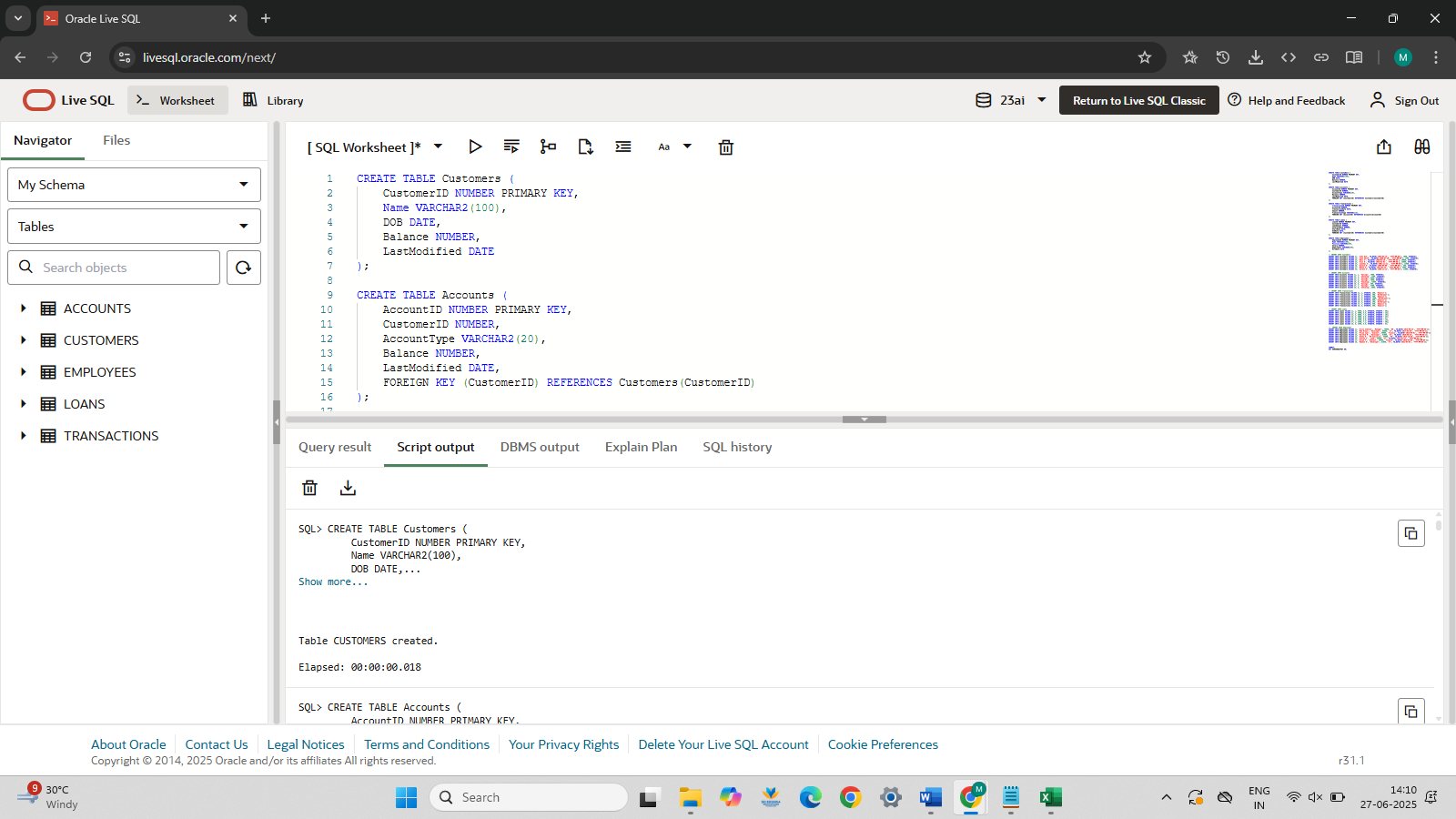
    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

**SCREENSHOT:**



**INSERT VALUES**

-- INSERT INTO Customers

INSERT INTO Customers VALUES (1, 'John Doe', TO\_DATE('1985-05-15', 'YYYY-MM-DD'), 1000, SYSDATE);

INSERT INTO Customers VALUES (2, 'Jane Smith', TO\_DATE('1990-07-20', 'YYYY-MM-DD'), 1500, SYSDATE);

INSERT INTO Customers VALUES (3, 'Ravi S', TO\_DATE('1950-03-10', 'YYYY-MM-DD'), 20000, SYSDATE);

INSERT INTO Customers VALUES (4, 'Arun K', TO\_DATE('1945-01-01', 'YYYY-MM-DD'), 15000, SYSDATE);

INSERT INTO Customers VALUES (5, 'Lakshmi V', TO\_DATE('2001-11-11', 'YYYY-MM-DD'), 12000, SYSDATE);

INSERT INTO Customers VALUES (6, 'Dinesh R', TO\_DATE('1998-09-09', 'YYYY-MM-DD'), 500, SYSDATE);

INSERT INTO Customers VALUES (7, 'Meena R', TO\_DATE('1975-03-25', 'YYYY-MM-DD'), 3000, SYSDATE);

INSERT INTO Customers VALUES (8, 'Suresh K', TO\_DATE('1960-12-12', 'YYYY-MM-DD'), 11000, SYSDATE);

-- INSERT INTO Accounts

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

INSERT INTO Accounts VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts VALUES (3, 3, 'Savings', 5000, SYSDATE);

INSERT INTO Accounts VALUES (4, 4, 'Savings', 7000, SYSDATE);

INSERT INTO Accounts VALUES (5, 5, 'Checking', 12000, SYSDATE);

INSERT INTO Accounts VALUES (6, 6, 'Savings', 400, SYSDATE);

INSERT INTO Accounts VALUES (7, 7, 'Savings', 3500, SYSDATE);

INSERT INTO Accounts VALUES (8, 8, 'Checking', 8000, SYSDATE);

-- INSERT INTO Transactions

INSERT INTO Transactions VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions VALUES (3, 3, SYSDATE, 500, 'Deposit');

INSERT INTO Transactions VALUES (4, 4, SYSDATE, 1000, 'Withdrawal');

INSERT INTO Transactions VALUES (5, 5, SYSDATE, 800, 'Deposit');

INSERT INTO Transactions VALUES (6, 6, SYSDATE, 100, 'Withdrawal');

INSERT INTO Transactions VALUES (7, 7, SYSDATE, 450, 'Deposit');

INSERT INTO Transactions VALUES (8, 8, SYSDATE, 300, 'Deposit');

-- INSERT INTO Loans

INSERT INTO Loans VALUES (1, 1, 5000, 5.0, SYSDATE, SYSDATE + 60);

INSERT INTO Loans VALUES (2, 3, 7000, 6.5, SYSDATE, SYSDATE + 25);

INSERT INTO Loans VALUES (3, 4, 6000, 7.5, SYSDATE, SYSDATE + 10);

INSERT INTO Loans VALUES (4, 5, 3000, 8.0, SYSDATE, SYSDATE + 35);

INSERT INTO Loans VALUES (5, 6, 2000, 9.0, SYSDATE, SYSDATE + 5);

INSERT INTO Loans VALUES (6, 7, 4500, 6.8, SYSDATE, SYSDATE + 15);

INSERT INTO Loans VALUES (7, 2, 5500, 7.2, SYSDATE, SYSDATE + 40);

INSERT INTO Loans VALUES (8, 8, 2500, 6.9, SYSDATE, SYSDATE + 8);

--  INSERT INTO Employees

INSERT INTO Employees VALUES (1, 'Alice Johnson', 'Manager', 70000, 'HR', TO\_DATE('2015-06-15', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (2, 'Bob Brown', 'Developer', 60000, 'IT', TO\_DATE('2017-03-20', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (3, 'Priya Iyer', 'Analyst', 50000, 'Finance', TO\_DATE('2018-04-01', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (4, 'Karthik M', 'Developer', 62000, 'IT', TO\_DATE('2020-08-25', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (5, 'Sneha R', 'HR Executive', 55000, 'HR', TO\_DATE('2016-10-30', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (6, 'Vikram S', 'Lead', 65000, 'IT', TO\_DATE('2014-01-12', 'YYYY-MM-DD'));

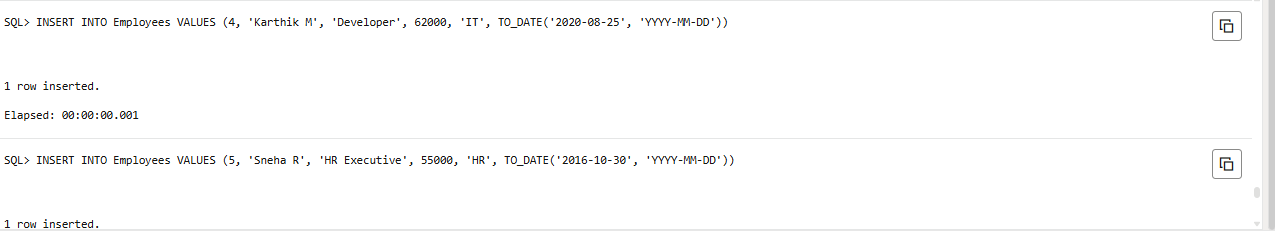
INSERT INTO Employees VALUES (7, 'Divya P', 'Analyst', 48000, 'Finance', TO\_DATE('2019-05-05', 'YYYY-MM-DD'));

INSERT INTO Employees VALUES (8, 'Ramesh N', 'Developer', 61000, 'IT', TO\_DATE('2021-03-01', 'YYYY-MM-DD'));

COMMIT;

SET SERVEROUTPUT ON;

**SCREENSHOT:**



**EXERCISE 1: CONTROL STRUCTURES**

**SCENARIO 1: APPLY DISCOUNT FOR SENIOR CITIZENS**

**CODE:**

BEGIN

    FOR cust IN (

        SELECT CustomerID, FLOOR(MONTHS\_BETWEEN(SYSDATE, DOB) / 12) AS age

        FROM Customers

    ) LOOP

        IF cust.age > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate - 1

            WHERE CustomerID = cust.CustomerID;

        END IF;

    END LOOP;

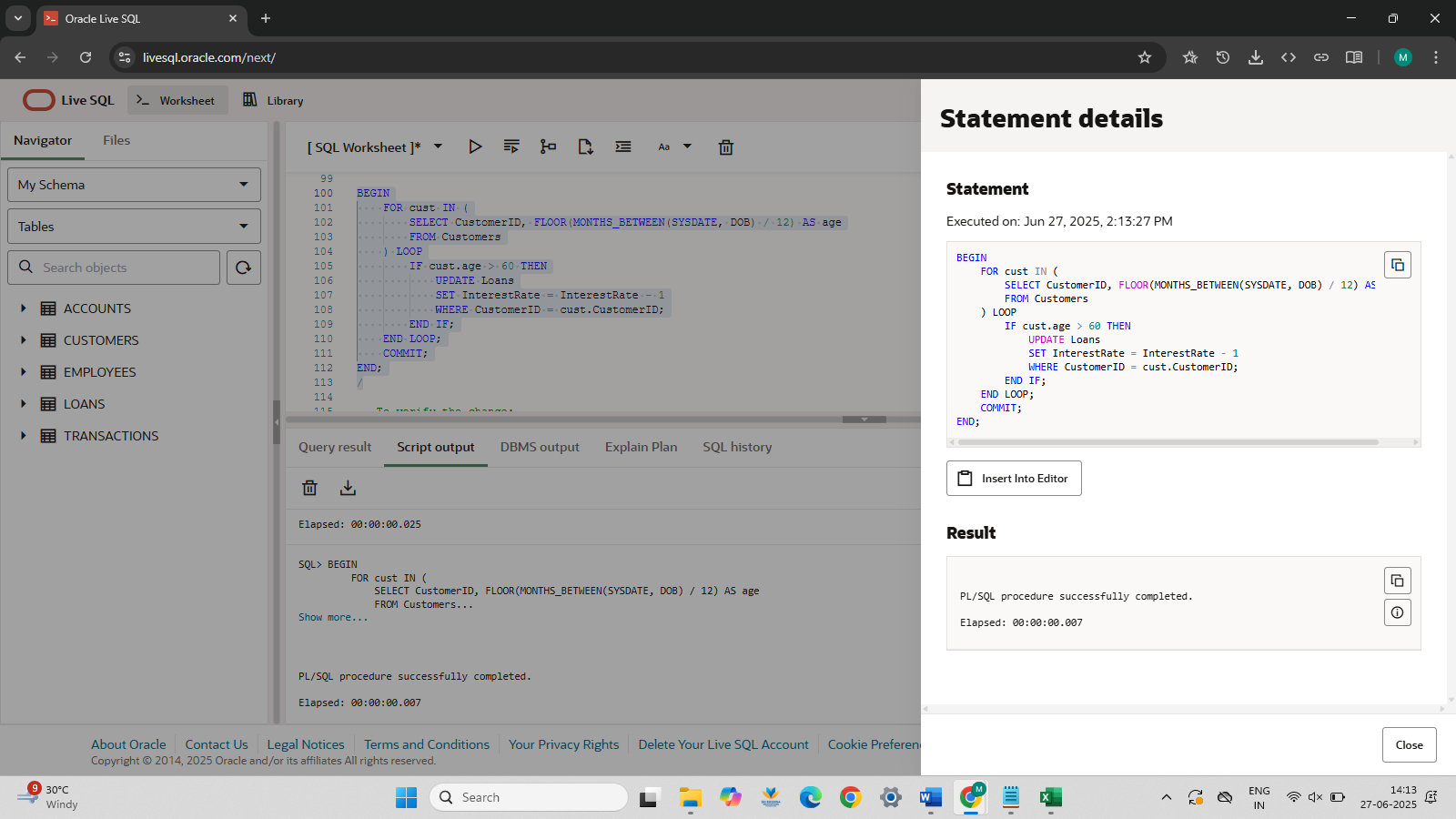
    COMMIT;

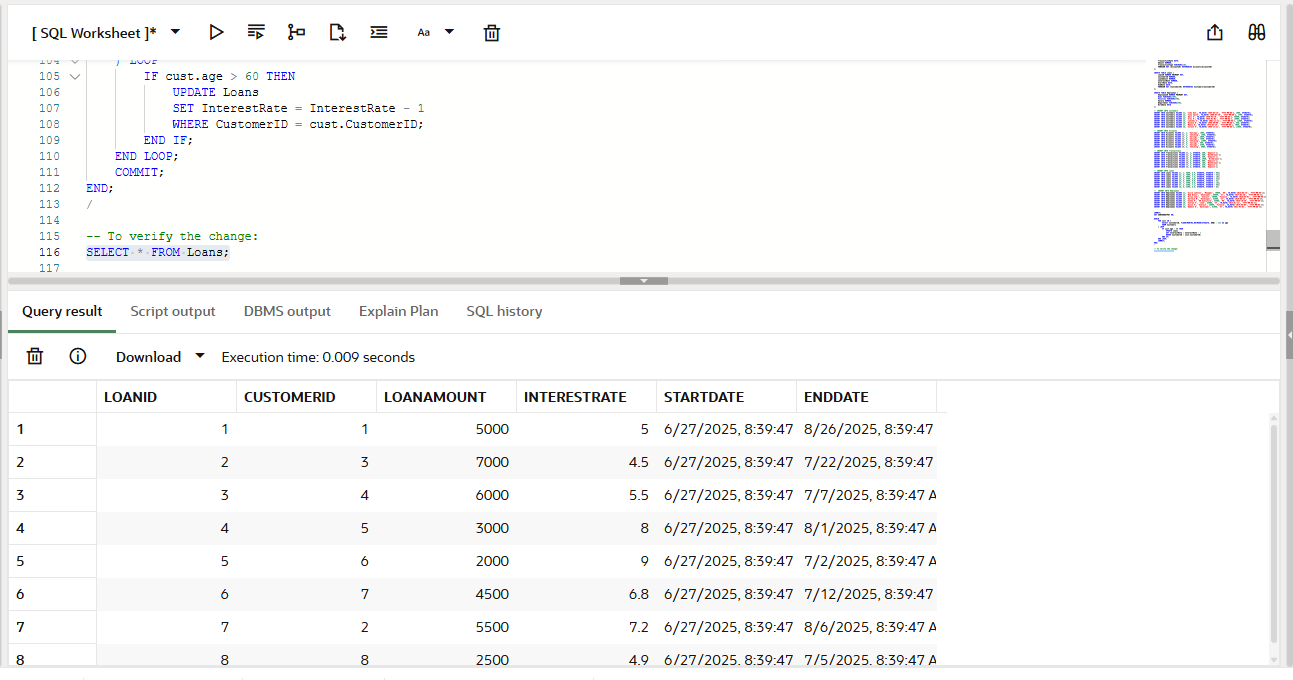
END;

-- To verify the change:

SELECT \* FROM Loans;

**OUTPUT SCREENSHOTS:**





**Scenario 2: Mark VIP Customers**

**CODE:**

--add IsVIP col

ALTER TABLE Customers ADD IsVIP VARCHAR2(5);

UPDATE Customers SET IsVIP = 'FALSE';

COMMIT;

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;

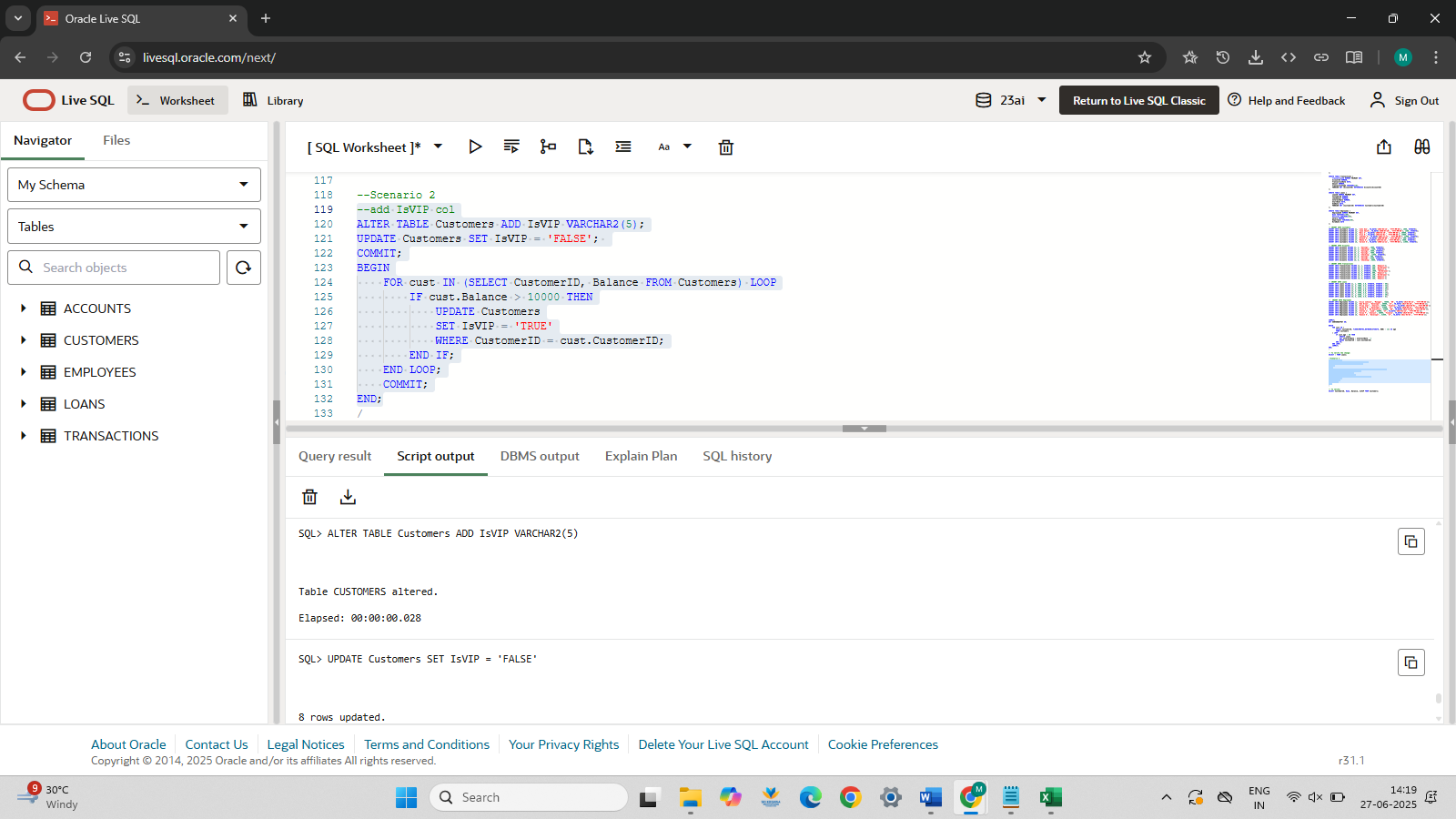
END;

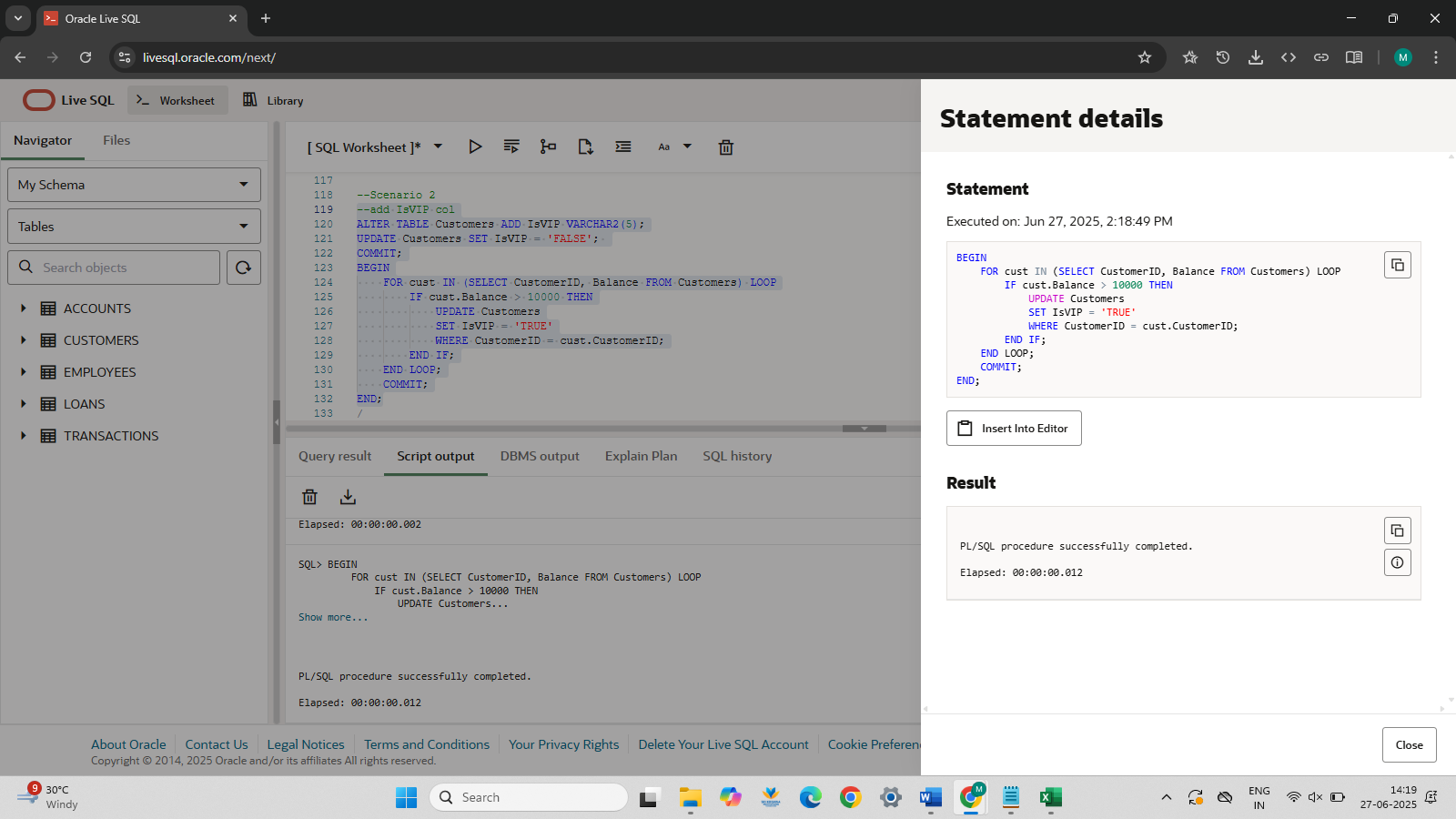
/

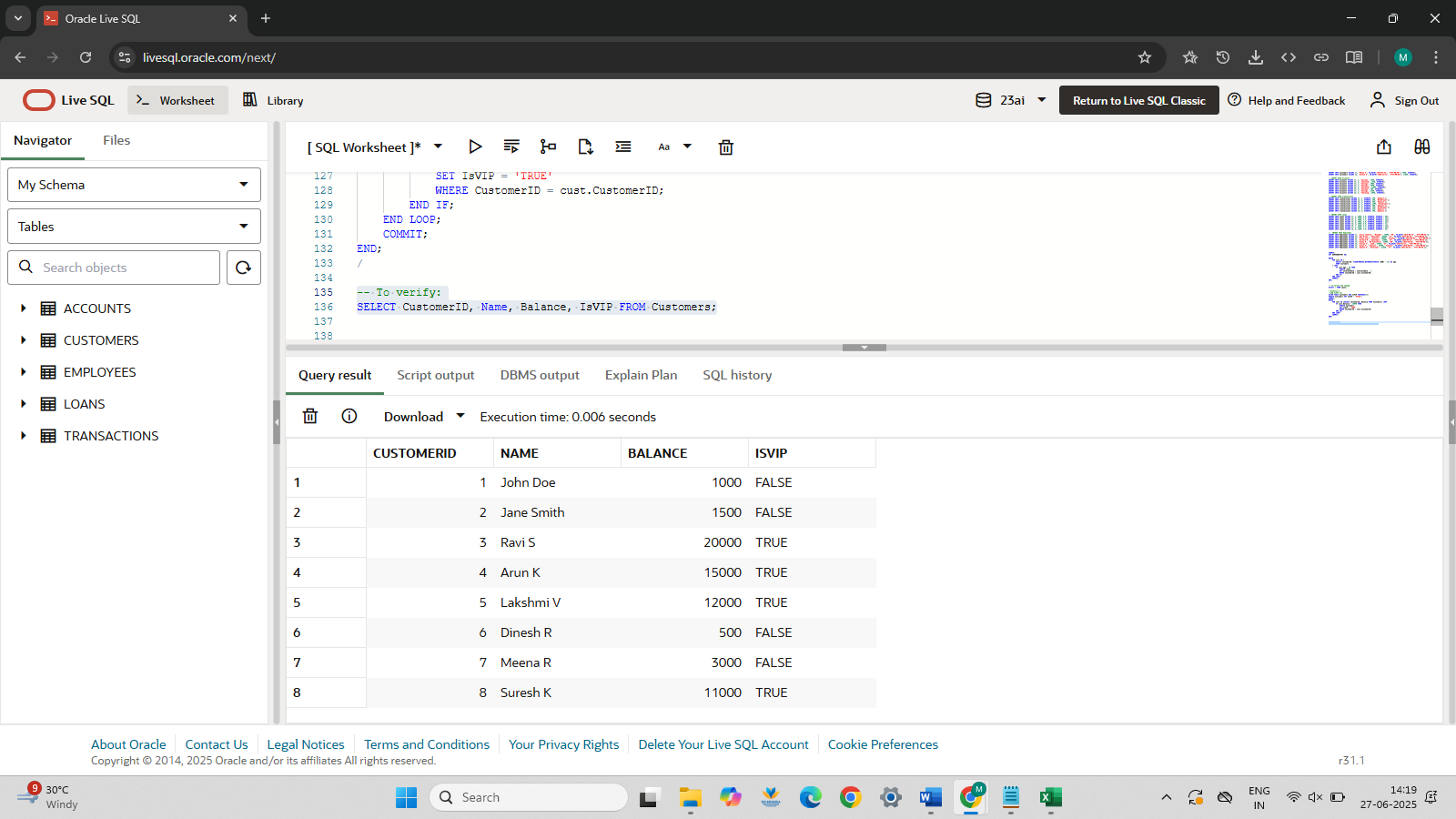
-- To verify:

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**OUTPUT SCREENSHOTS:**







**Scenario 3:**

**CODE:**

--Scenario 3

BEGIN

    FOR loan\_rec IN (

        SELECT L.LoanID, C.Name, L.EndDate

        FROM Loans L

        JOIN Customers C ON L.CustomerID = C.CustomerID

        WHERE L.EndDate BETWEEN SYSDATE AND SYSDATE + 30

    ) LOOP

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

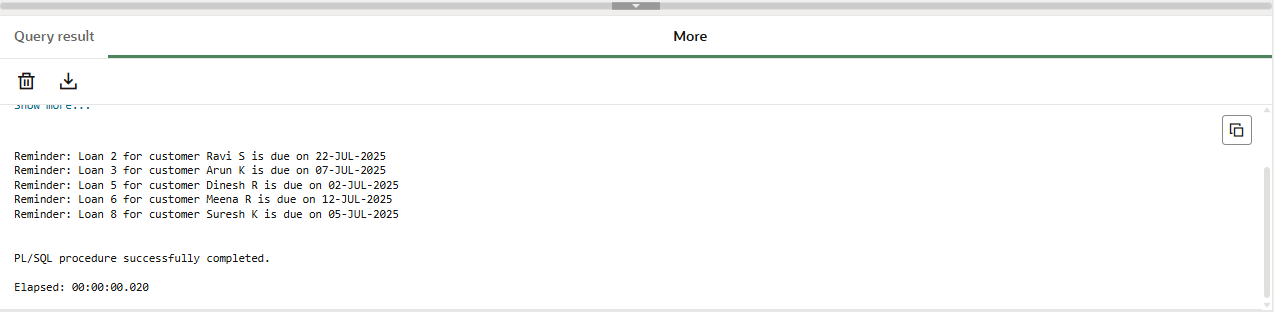
                             ' for customer ' || loan\_rec.Name ||

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

    END LOOP;

END;

**OUTPUT SCREENSHOTS:**



**EXERCISE 3: STORED PROCEDURES**

**SCENARIO 1**

**CODE:**

--SCENARIO 1

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    ) LOOP

        UPDATE Accounts

        SET Balance = acc.Balance + (acc.Balance \* 0.01),

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

    END LOOP;

    COMMIT;

END;

--CALL

/

BEGIN

    ProcessMonthlyInterest;

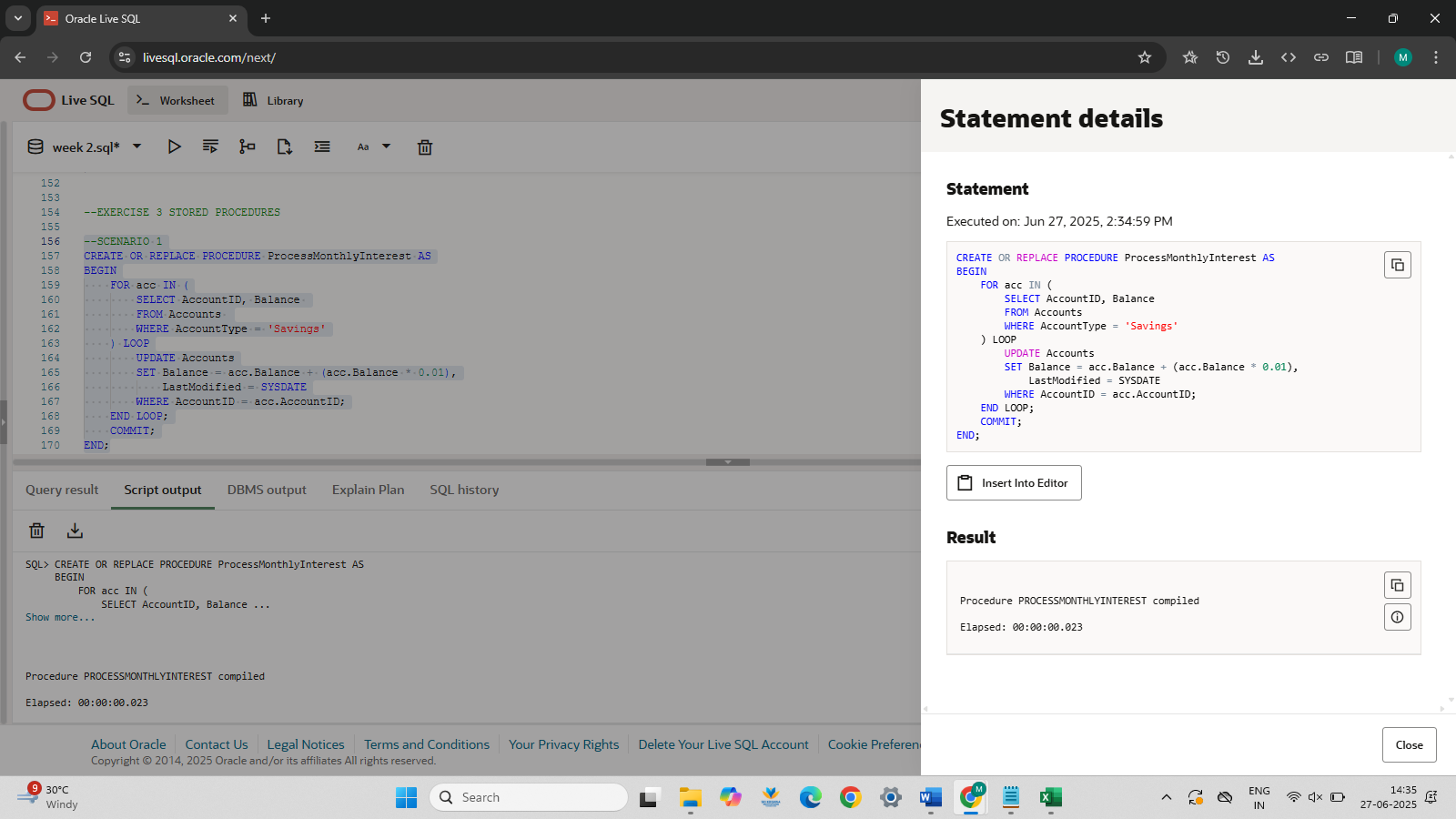
END;

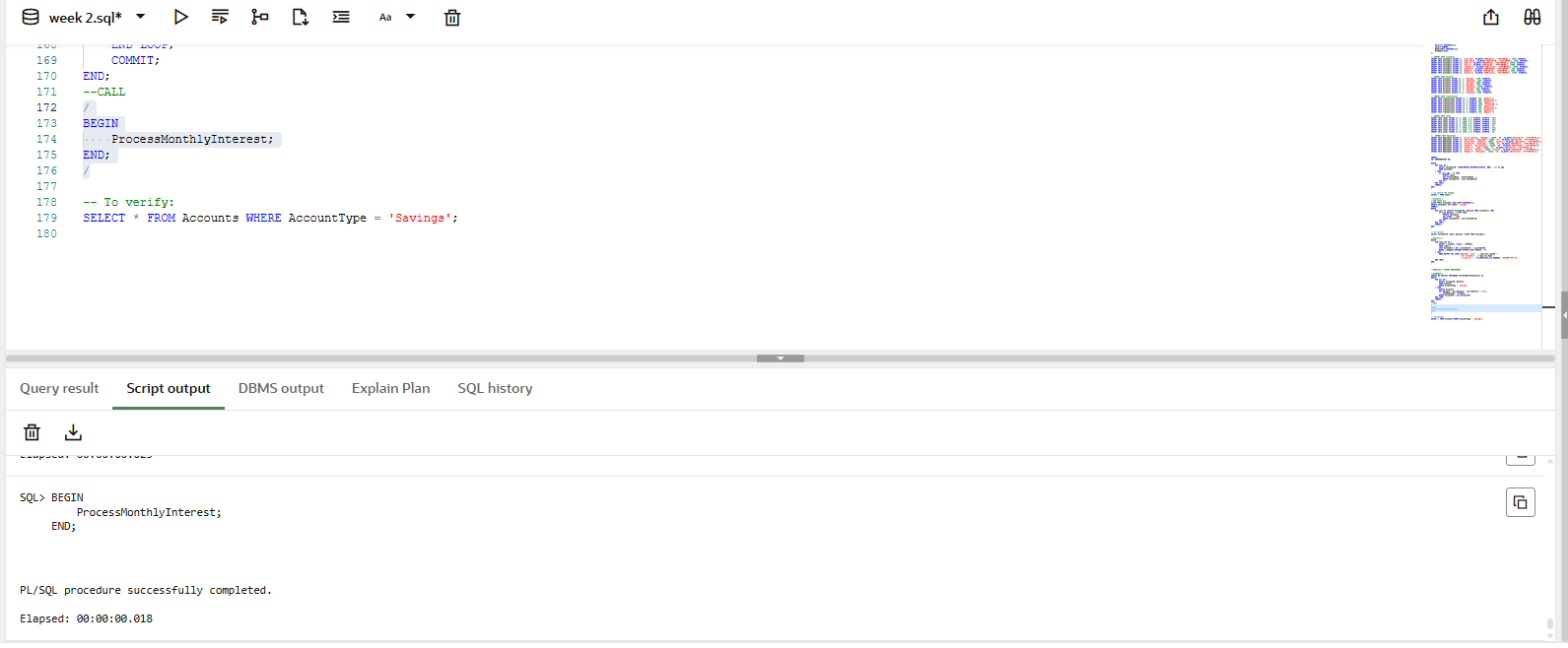
/

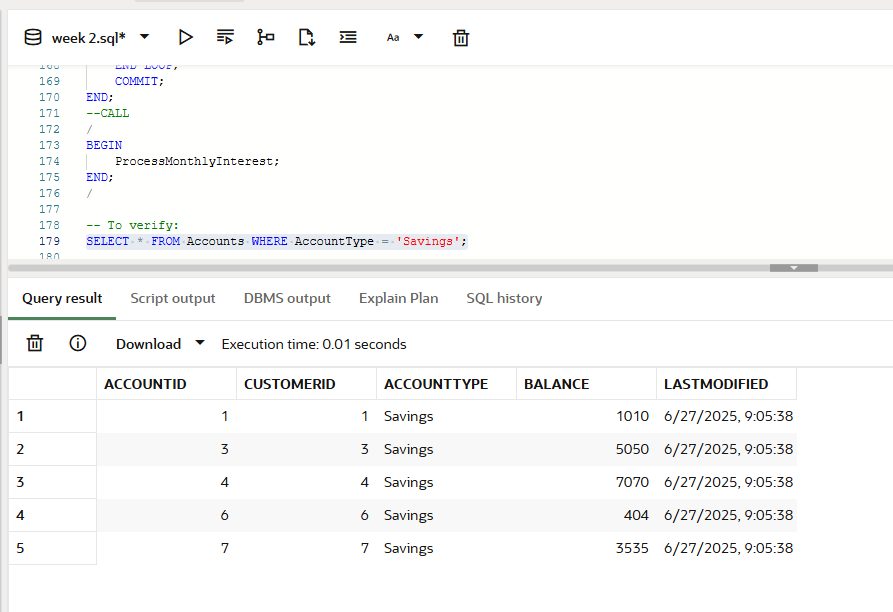
-- To verify:

SELECT \* FROM Accounts WHERE AccountType = 'Savings';

**OUTPUT SCREENSHOTS:**







**SCENARIO 2**

**CODE:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_pct IN NUMBER

) AS

BEGIN

    UPDATE Employees

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

    WHERE Department = p\_department;

    COMMIT;

END;

/

BEGIN

    UpdateEmployeeBonus('IT', 10);  -- Apply 10% bonus to IT dept

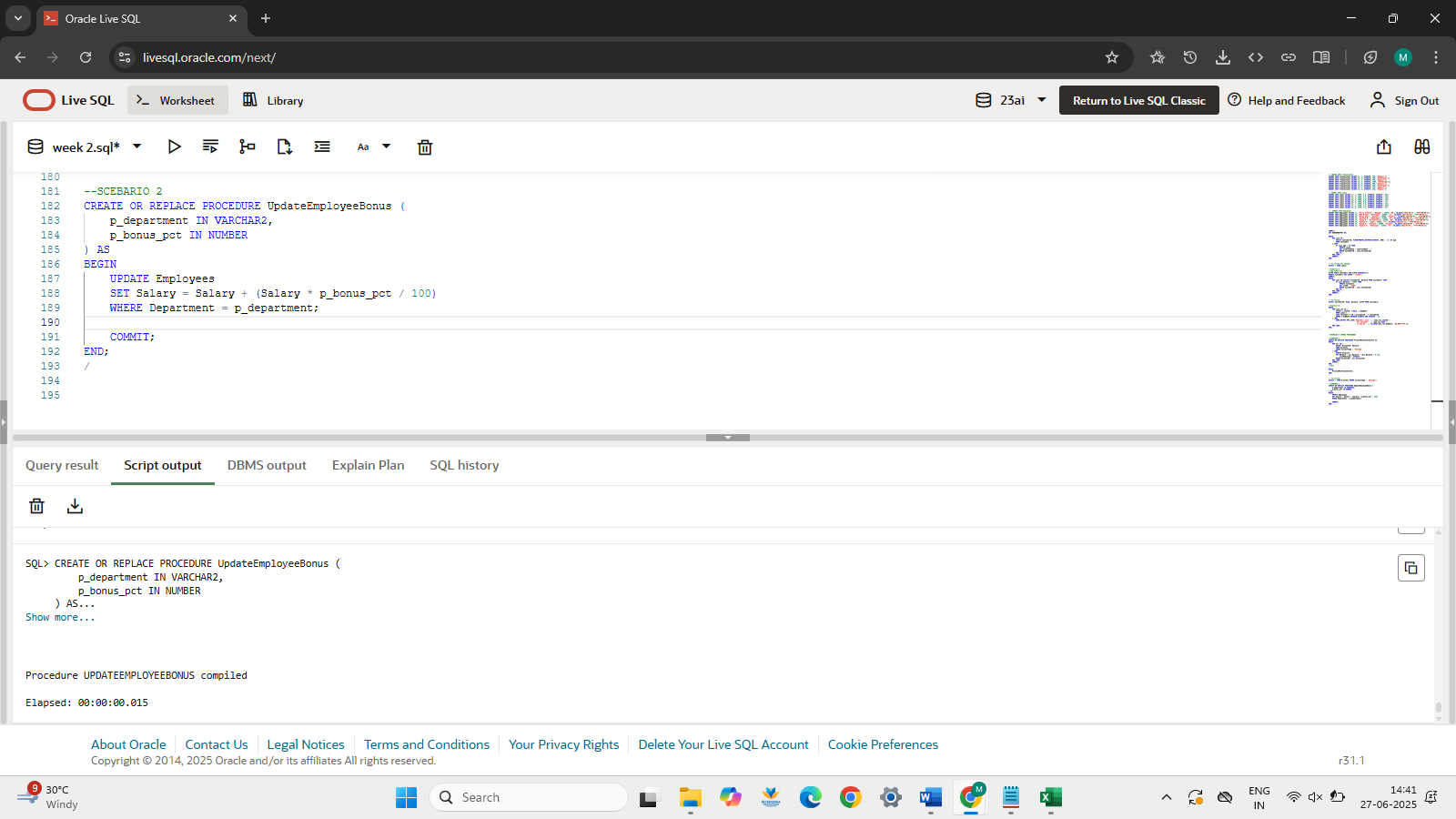
END;

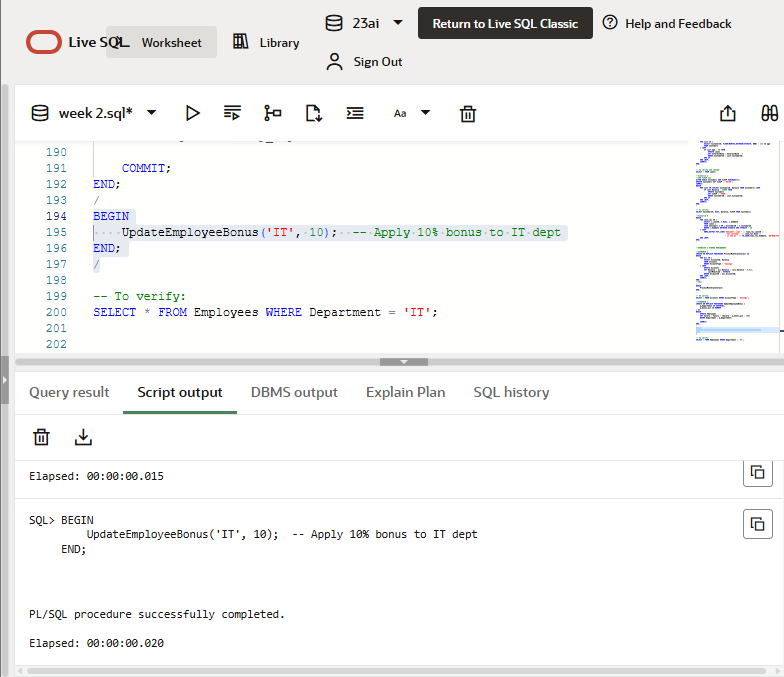
/

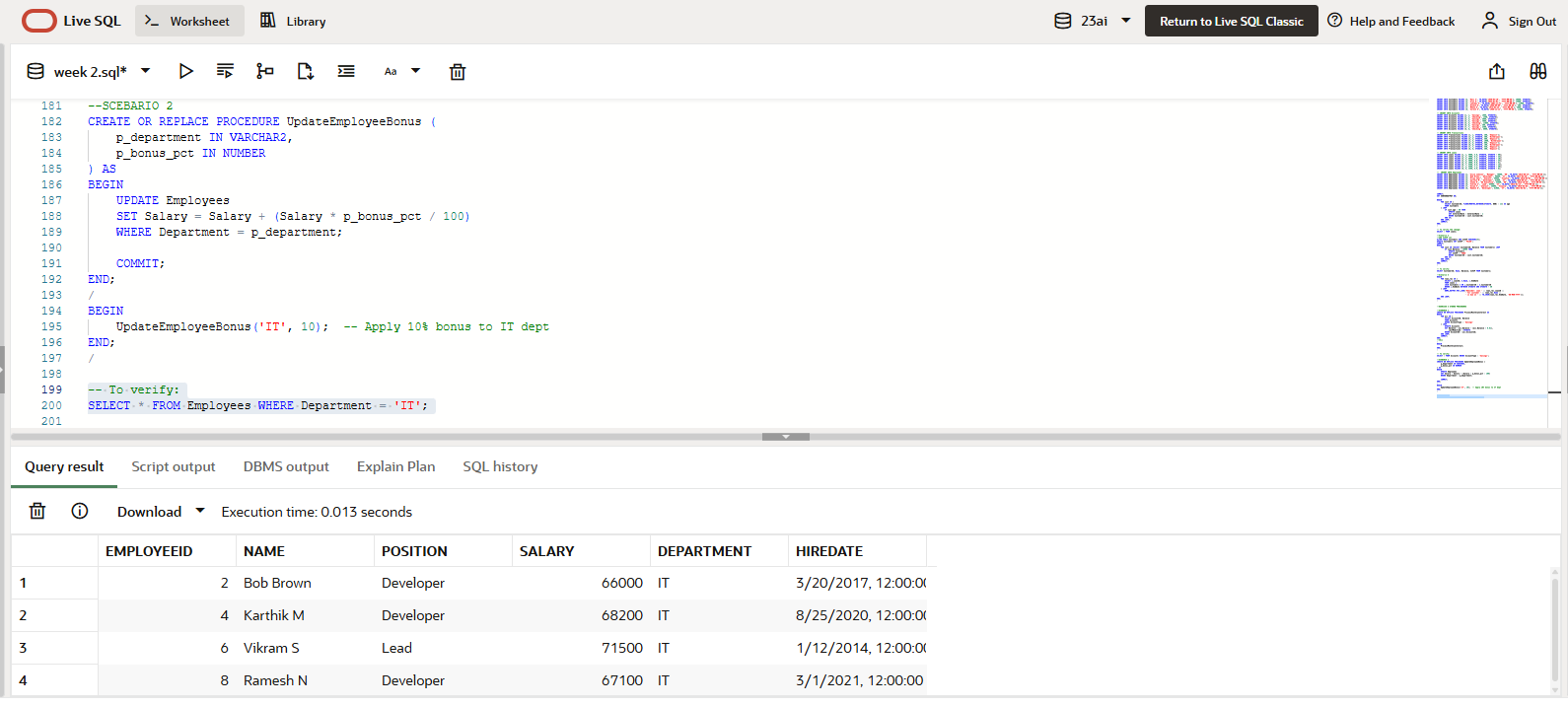
-- To verify:

SELECT \* FROM Employees WHERE Department = 'IT';

**OUTPUT SCREENSHOTS:**







**SCENARIO 3**

**CODE:**

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_source\_acc IN NUMBER,

    p\_dest\_acc IN NUMBER,

    p\_amount IN NUMBER

) AS

    v\_source\_balance NUMBER;

BEGIN

    SELECT Balance INTO v\_source\_balance

    FROM Accounts

    WHERE AccountID = p\_source\_acc;

    IF v\_source\_balance >= p\_amount THEN

        UPDATE Accounts

        SET Balance = Balance - p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_source\_acc;

        UPDATE Accounts

        SET Balance = Balance + p\_amount,

            LastModified = SYSDATE

        WHERE AccountID = p\_dest\_acc;

        COMMIT;

        DBMS\_OUTPUT.PUT\_LINE('Transfer successful!');

    ELSE

        DBMS\_OUTPUT.PUT\_LINE('Insufficient funds in source account.');

    END IF;

END;

/

BEGIN

    TransferFunds(1, 2, 200);

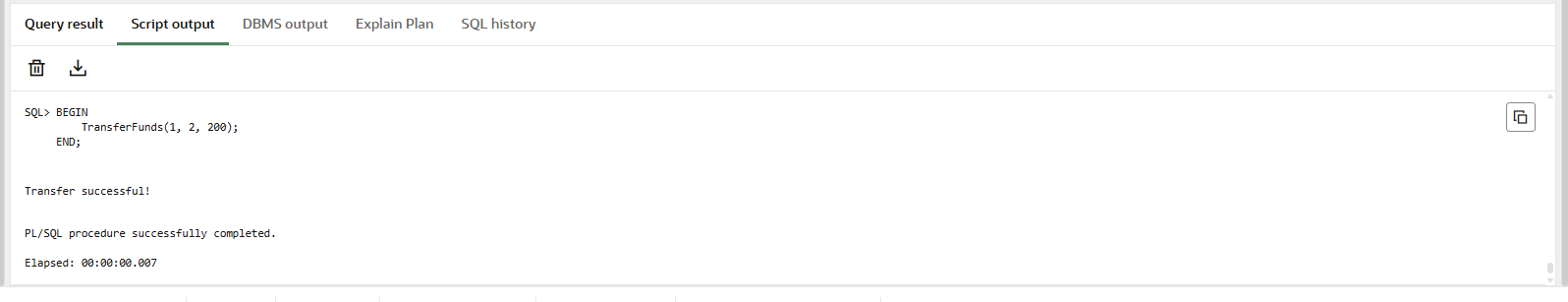
END;

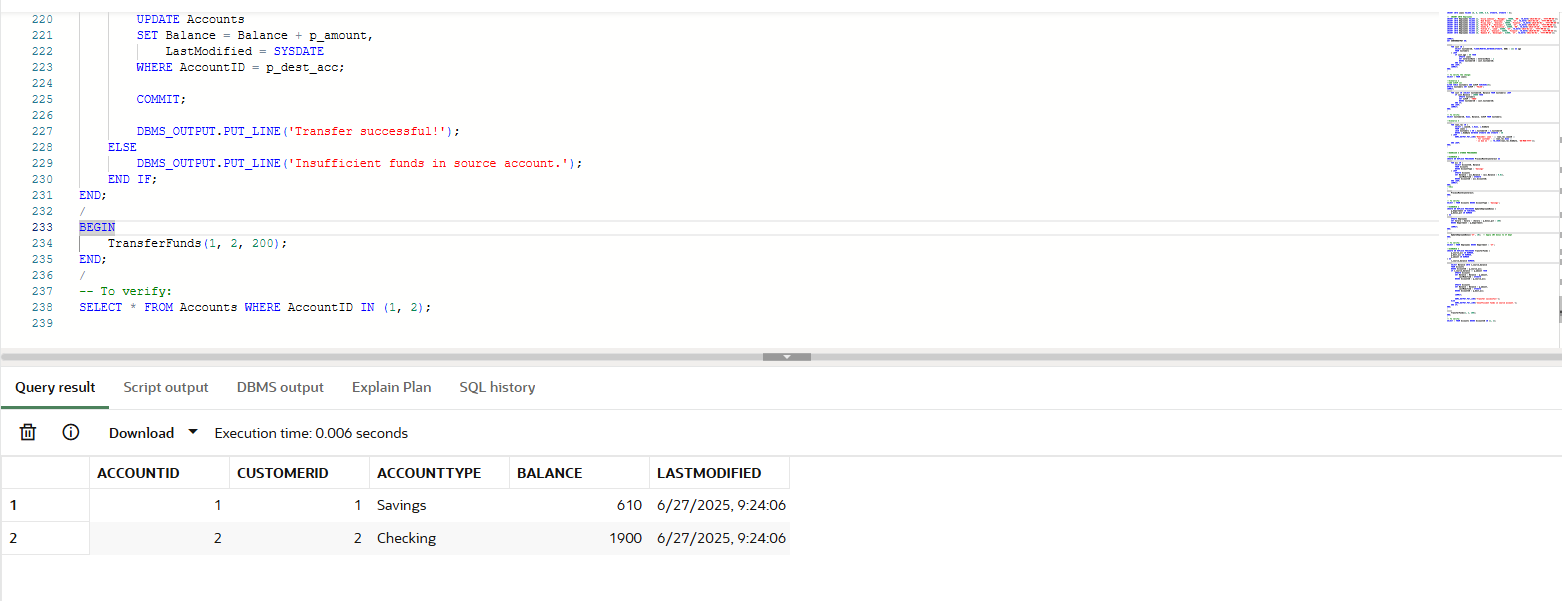
/

-- To verify:

SELECT \* FROM Accounts WHERE AccountID IN (1, 2);

**OUTPUT SCREENSHOTS:**





**TDD using JUnit5 and Mockito**

**JUnit Basic Testing Exercises**

**Exercise 1: Setting Up JUnit**

Basic Unit Test Setup with JUnit 4.13.2 in Maven

**Scenario:**

You need to set up JUnit in your Java project to start writing unit tests.

**pom.xml Code With Dependency:**

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.maha.junit</groupId>  
 <artifactId>junitdemo</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**DESCRIPTION:**

This exercise demonstrates the process of configuring JUnit in a Java project using Maven and writing a basic unit test. The Calculator class contains a simple add method, and the test class CalculatorTest uses JUnit 4.13.2 to verify the correctness of this method using the assertEquals assertion. This forms the foundation for Test-Driven Development (TDD) by enabling reliable, automated testing from the beginning of the development lifecycle.

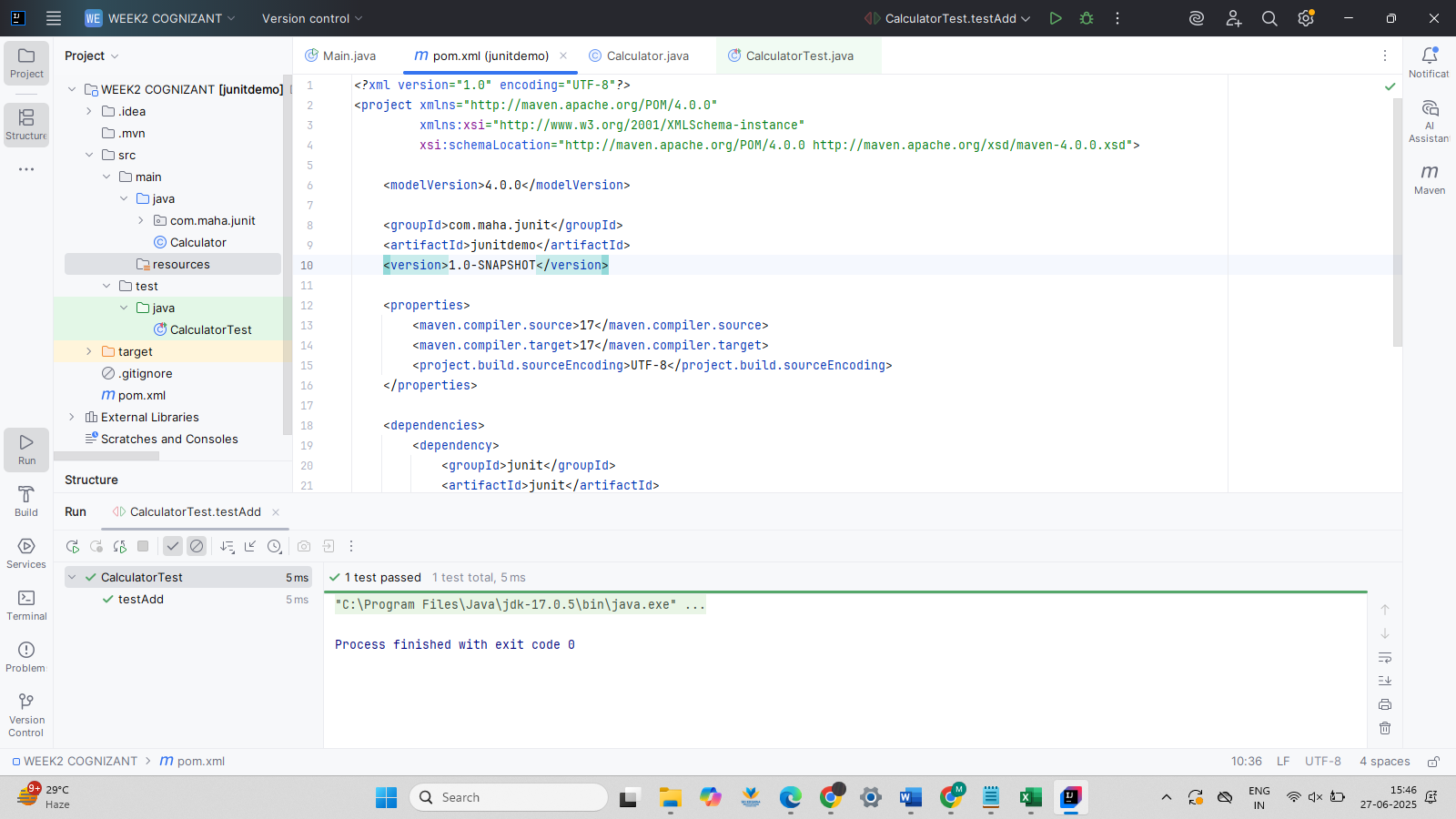
**Java Class: Calculator.java**

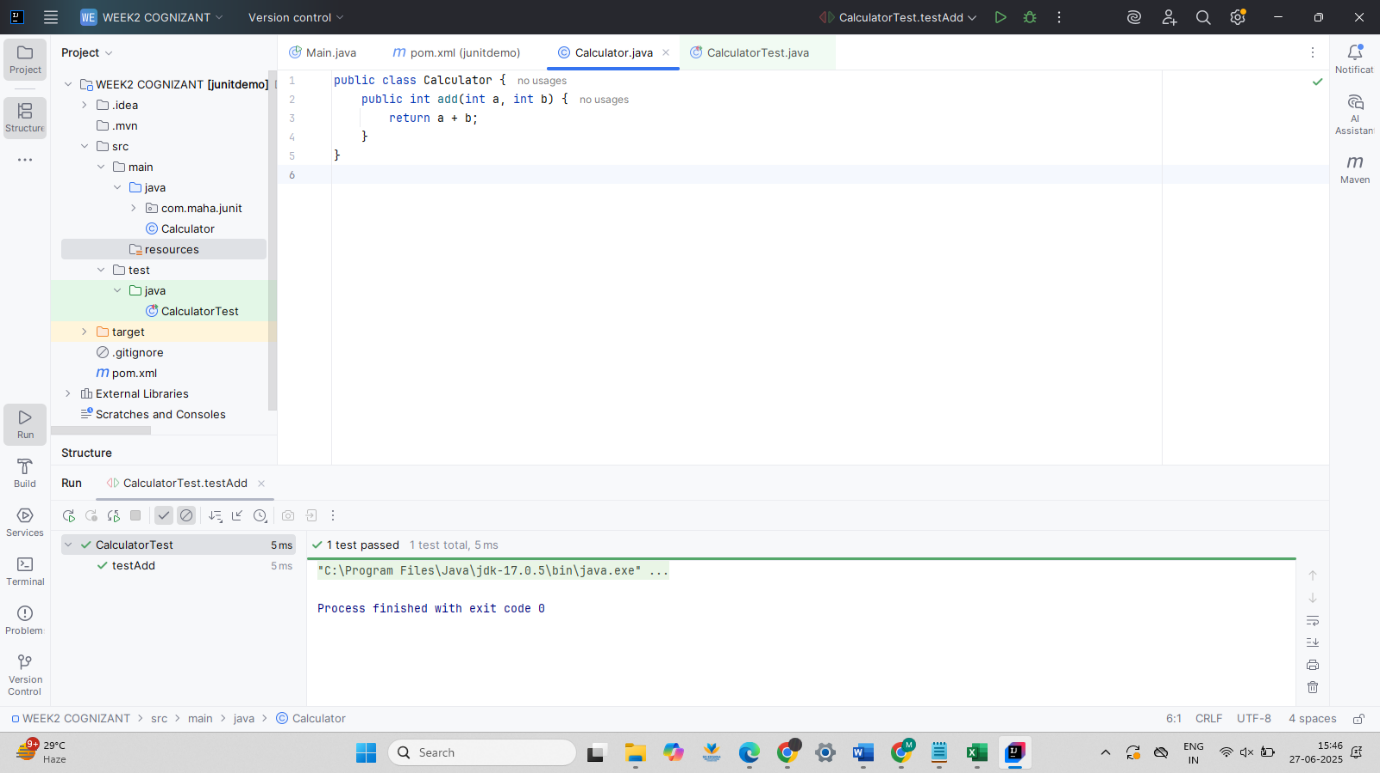
public class Calculator {  
 public int add(int a, int b) {  
 return a + b;  
 }  
}

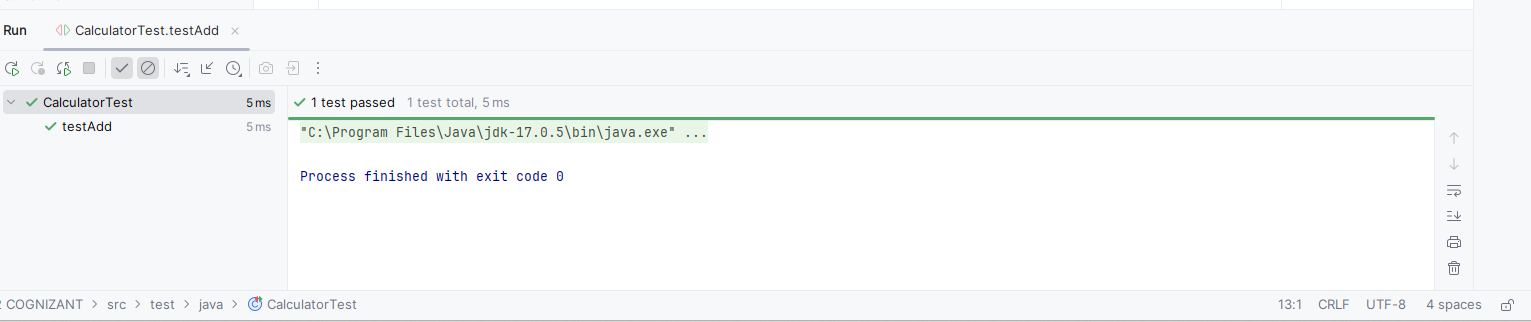
**Test Class: CalculatorTest.java**

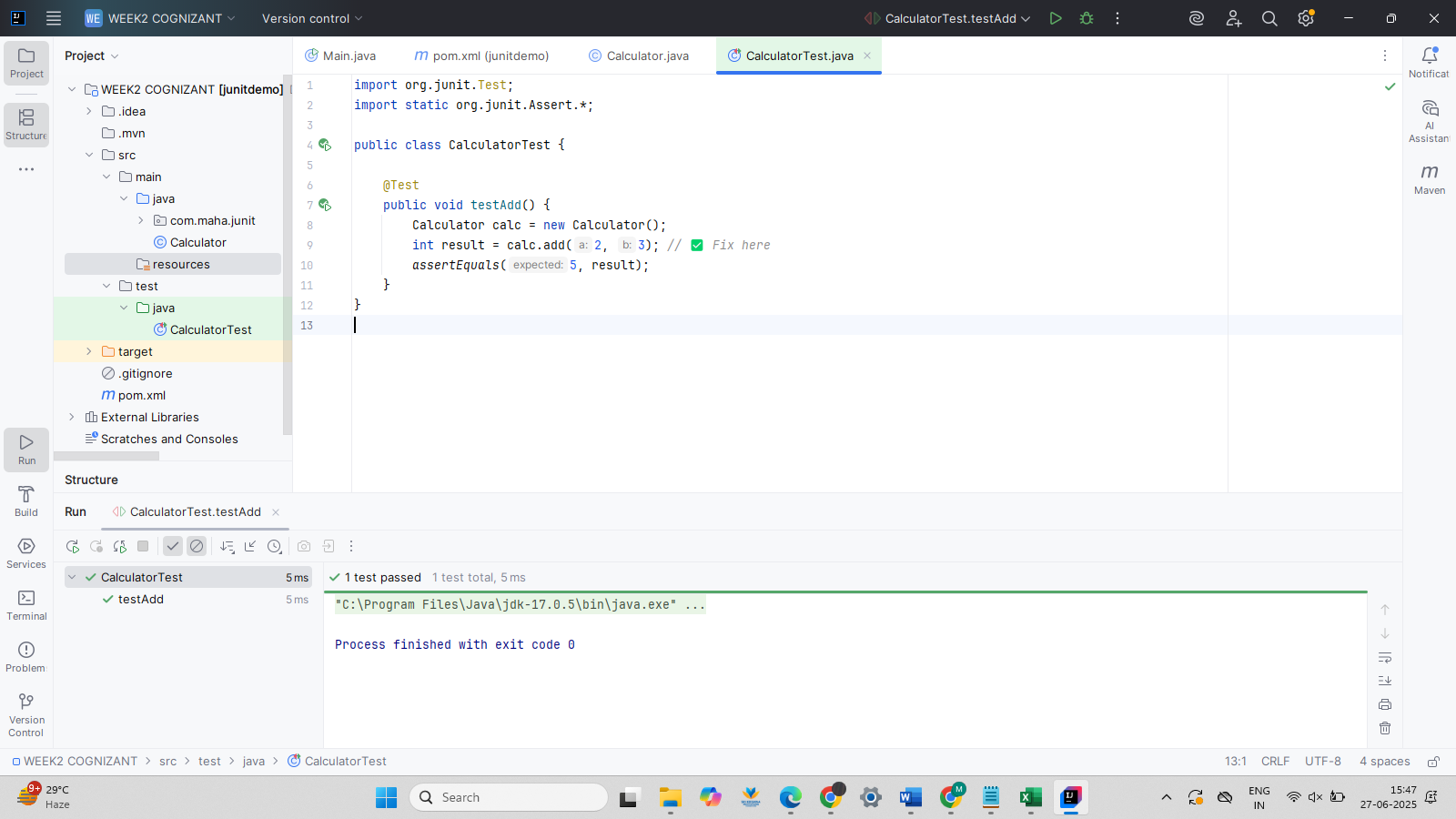
import org.junit.Test;  
import static org.junit.Assert.\*;  
  
public class CalculatorTest {  
  
 @Test  
 public void testAdd() {  
 Calculator calc = new Calculator();  
 int result = calc.add(2, 3);  *assertEquals*(5, result);  
 }  
}

**OUTPUT SCREENSHOTS:**









**Exercise 3: Assertions in Junit**

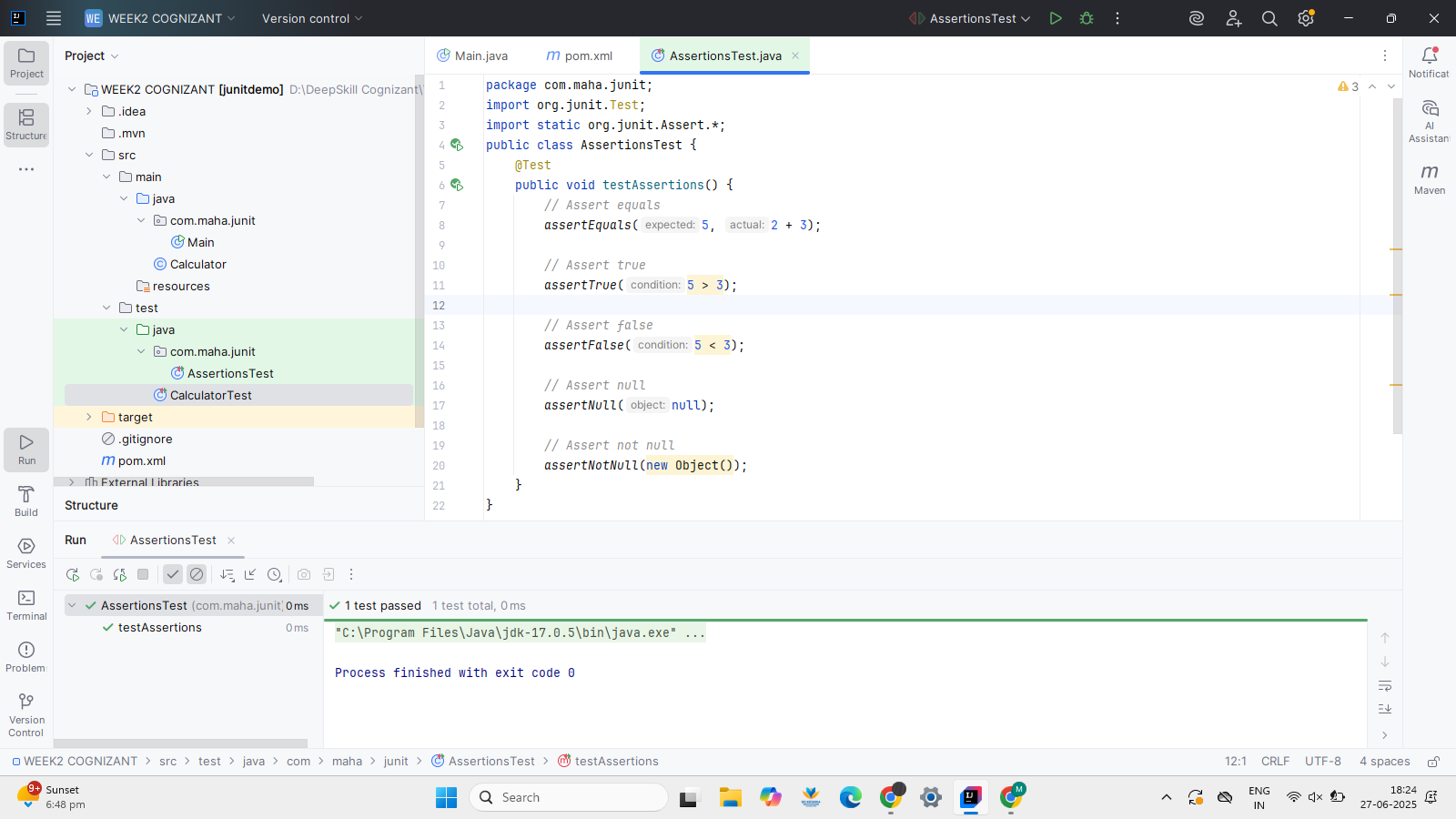
**Scenario:**

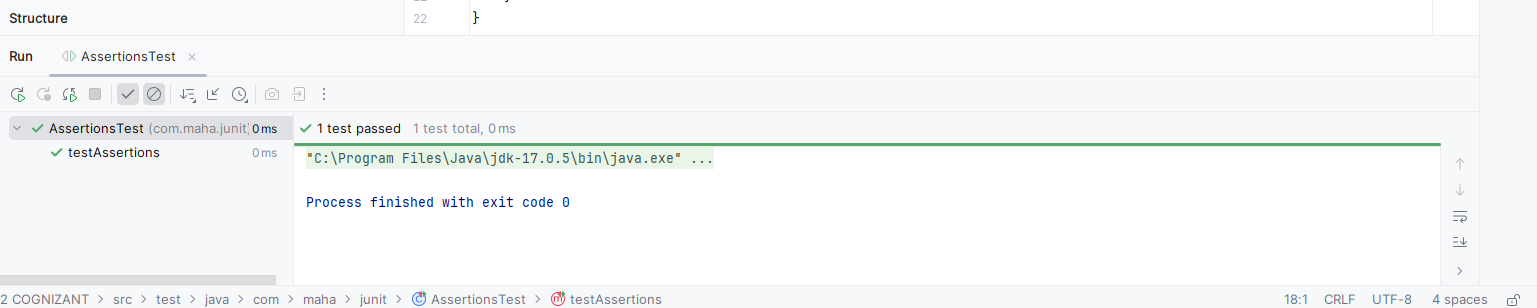
You need to use different assertions in JUnit to validate your test results.

**CODE:**

package com.maha.junit;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
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());  
 }  
}

**OUTPUT SCREENSHOTS:**





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

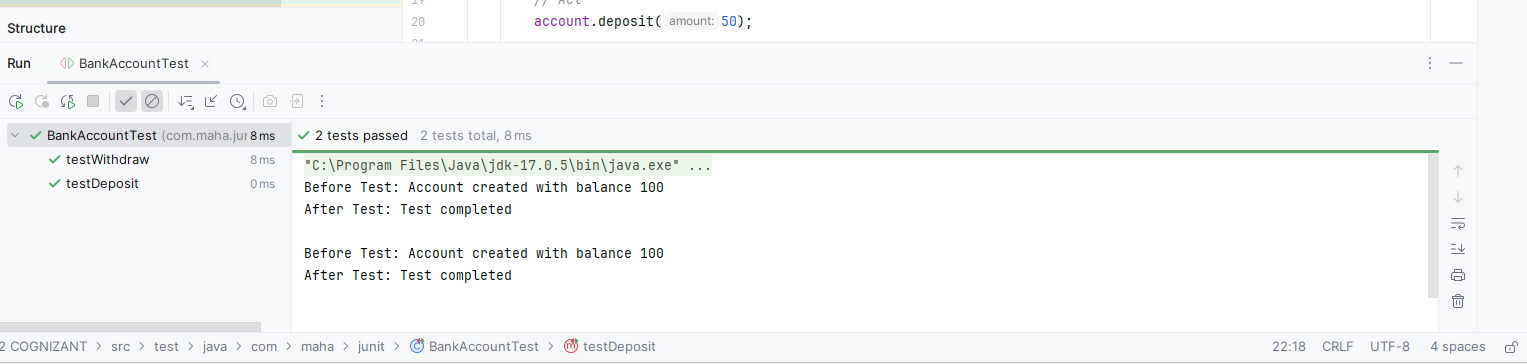
**Scenario:**

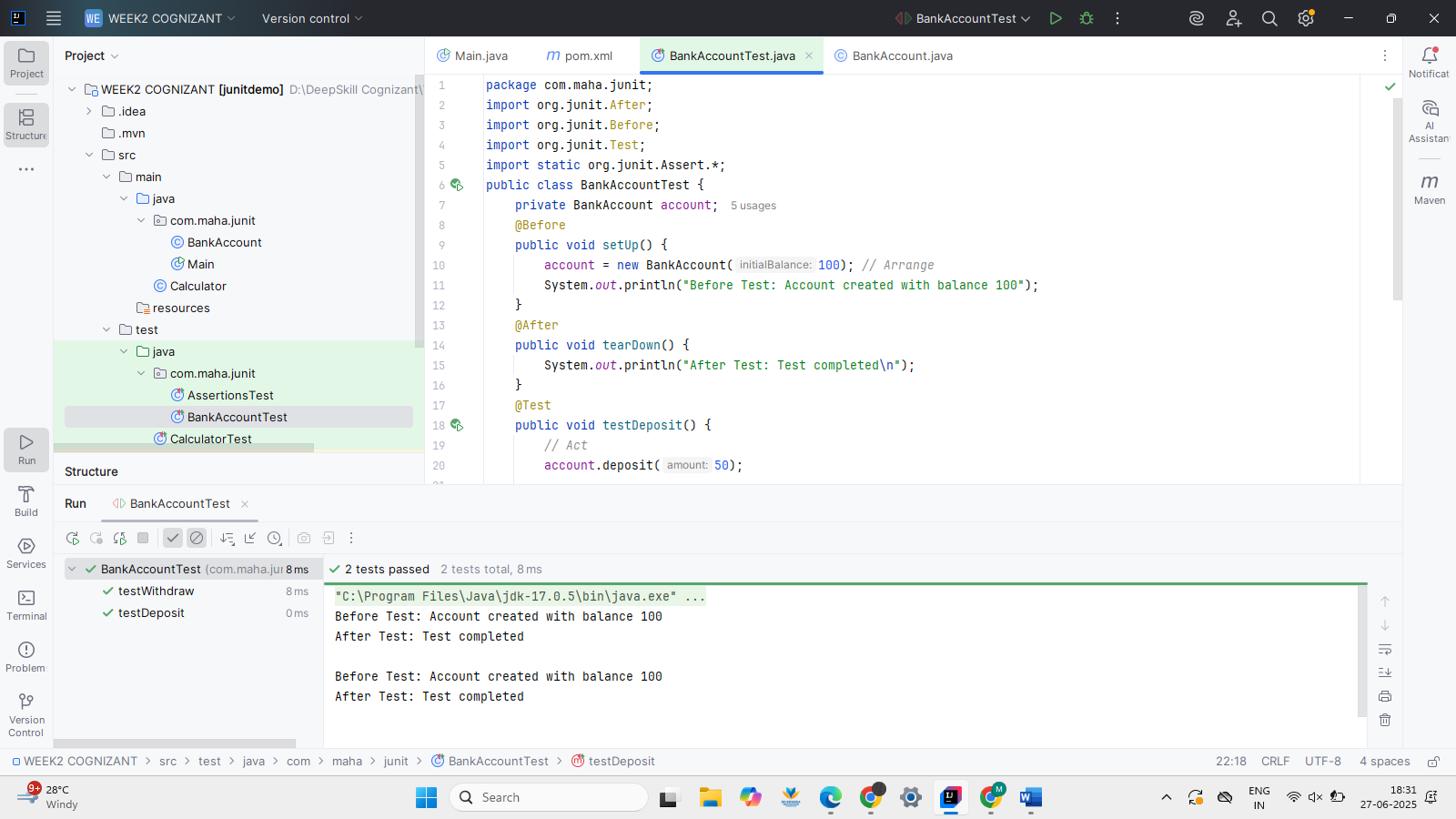
You need to organize your tests using the Arrange-Act-Assert (AAA) pattern and use setup and teardown methods.

**CODE:**

package com.maha.junit;  
import org.junit.After;  
import org.junit.Before;  
import org.junit.Test;  
import static org.junit.Assert.\*;  
public class BankAccountTest {  
 private BankAccount account;  
 @Before  
 public void setUp() {  
 account = new BankAccount(100); *// Arrange* System.*out*.println("Before Test: Account created with balance 100");  
 }  
 @After  
 public void tearDown() {  
 System.*out*.println("After Test: Test completed\n");  
 }  
 @Test  
 public void testDeposit() {  
 *// Act* account.deposit(50);  
  
 *// Assert  
 assertEquals*(150, account.getBalance());  
 }  
 @Test  
 public void testWithdraw() {  
 *// Act* account.withdraw(30);  
  
 *// Assert  
 assertEquals*(70, account.getBalance());  
 }  
}

**OUTPUT SCREENSHOTS:**





**MOCKITO EXERCISES**

**Exercise 1: Mocking and Stubbing**

Using Mockito to Stub External API Call

You need to test a service that depends on an external API...

**Description**:

This exercise demonstrates how to isolate and test a service class by mocking its dependency using Mockito. The MyService class depends on an ExternalApi interface. Instead of calling the real implementation, we use a mock object to simulate the behavior of the external API. The method getData() is stubbed to return a predefined value ("Mock Data"), and the test verifies that MyService.fetchData() returns this value correctly. This approach helps ensure unit tests are fast, reliable, and do not depend on external systems.

**pom.xml:**

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.maha.junit</groupId>  
 <artifactId>MockitoExercise</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.junit.jupiter</groupId>  
 <artifactId>junit-jupiter</artifactId>  
 <version>5.9.3</version>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.mockito</groupId>  
 <artifactId>mockito-core</artifactId>  
 <version>5.10.0</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>3.0.0-M8</version>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**ExternalApi.java:**

public interface ExternalApi {  
 String getData();  
}

**MyService.java**

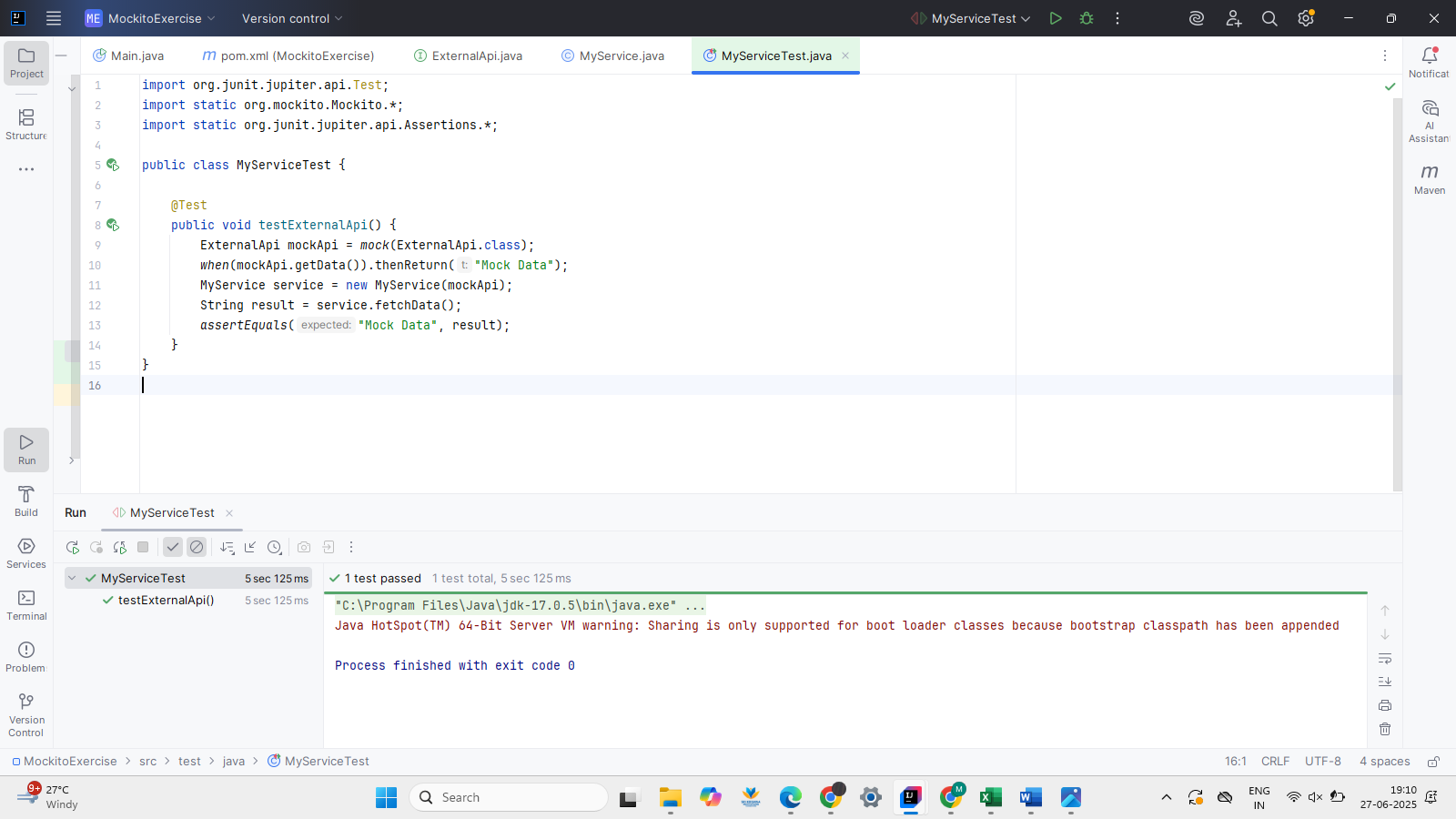
public class MyService {  
 private ExternalApi api;  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.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 SCREENSHOTS:**



**Exercise 2: Verifying Interactions**

Using Mockito to Verify Method Calls with Specific Arguments

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

**Description:**  
This test ensures that fetchData() calls the method getData() on the mocked ExternalApi object. Mockito’s verify() function confirms that the interaction occurred exactly as expected.

**ExternalApi.java**

public interface ExternalApi {  
 String getData();  
}

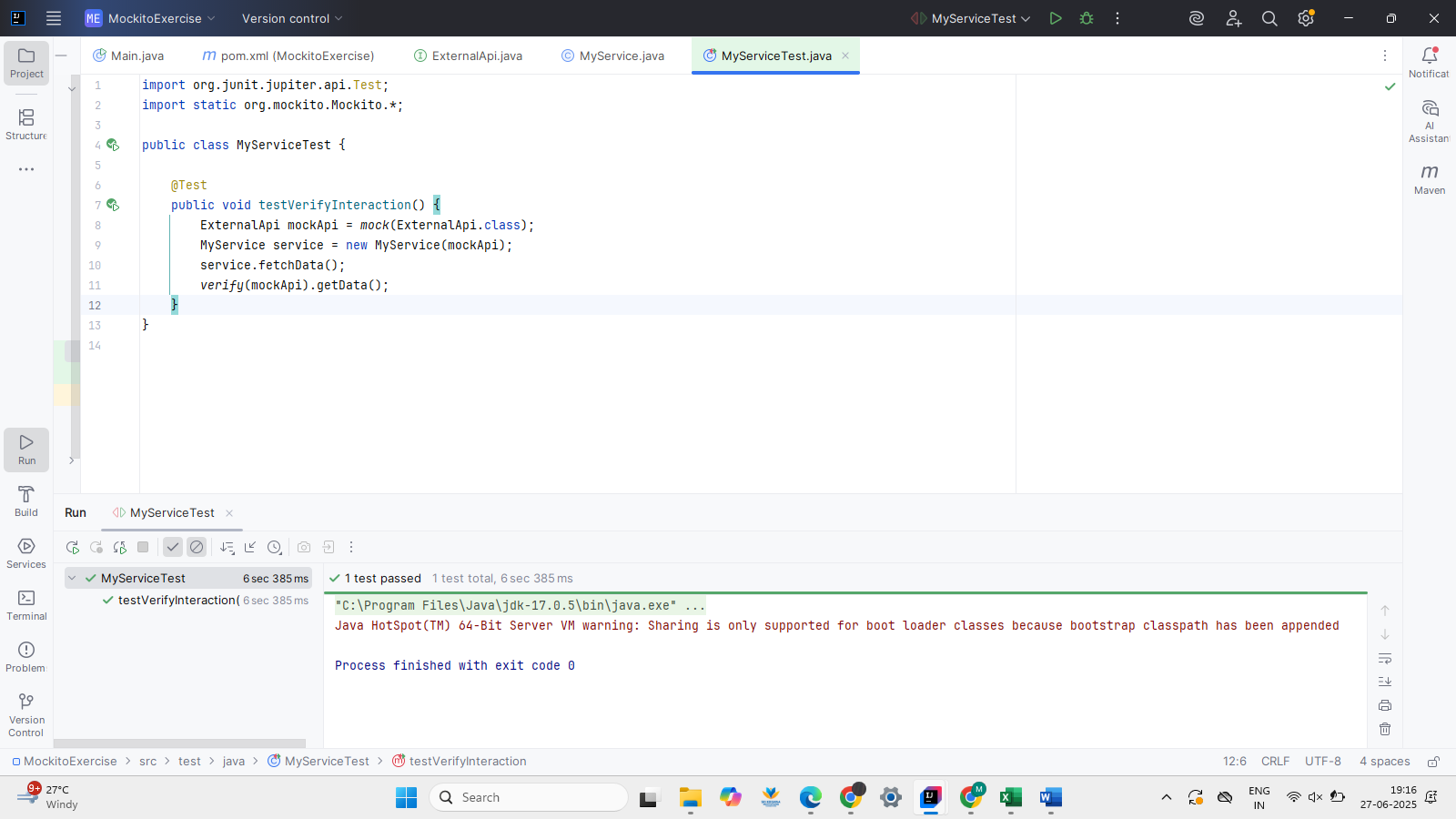
**MyService.java**

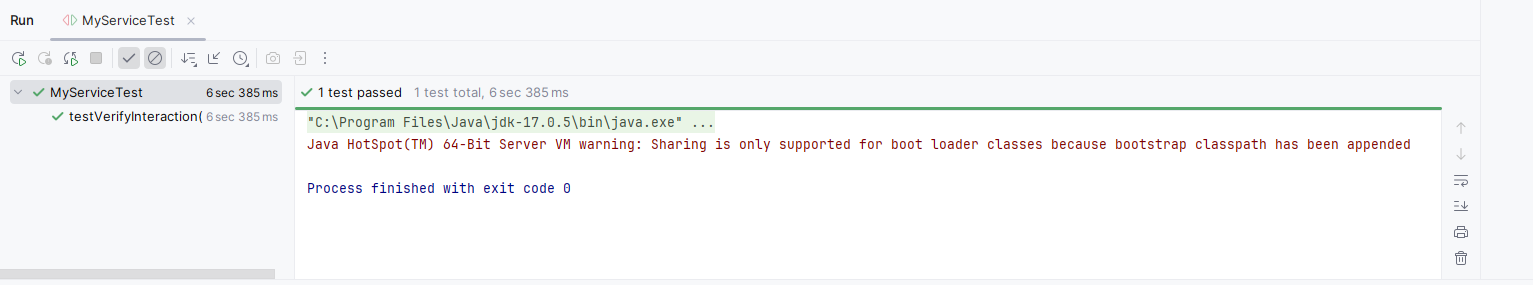
public class MyService {  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
}

**MyServiceTest.java**

import org.junit.jupiter.api.Test;  
import static org.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 SCREENSHOTS:**





SLF4J logging framework

SL4J Logging exercises

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

Demonstrating SLF4J Logging with Logback in a Java Application

**Task:**

Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**Description:**

This exercise showcases how to integrate SLF4J logging in a Java application using the Logback implementation. The LoggingExample class uses LoggerFactory to create a logger and logs messages at error and warning levels. This setup helps track and debug runtime issues effectively without using System.out.println() statements, promoting better logging practices.

**pom.xml:**

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <groupId>com.maha.junit</groupId>  
 <artifactId>SLF4Jlogging</artifactId>  
 <version>1.0-SNAPSHOT</version>  
 <properties>  
 <maven.compiler.source>17</maven.compiler.source>  
 <maven.compiler.target>17</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <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>  
</project>

**LoggingExample.java:**

package com.maha.junit;  
  
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 SCREENSHOTS:**

