**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.***

**CODE:**

CREATE TABLE customers\_new1 (

customer\_id INTEGER,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

age INTEGER,

country VARCHAR(100),

loan\_interest\_rate INTEGER);

INSERT INTO customers\_new1 VALUES (1, 'John', 'Doe', 65, 'USA', 10);

INSERT INTO customers\_new1 VALUES (2, 'Robert', 'Luna', 70, 'USA', 9);

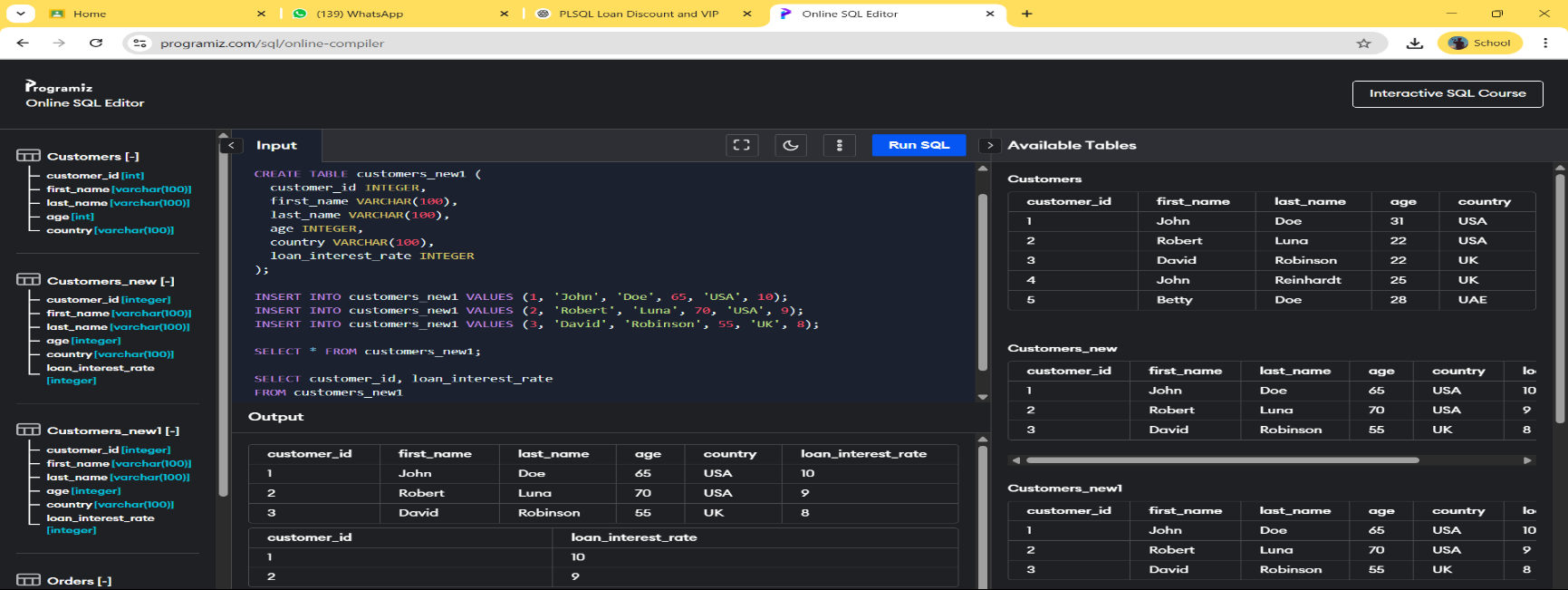
INSERT INTO customers\_new1 VALUES (3, 'David', 'Robinson', 55, 'UK', 8);

SELECT \* FROM customers\_new1;

SELECT customer\_id, loan\_interest\_rate

FROM customers\_new1

WHERE age > 60;

**OUTPUT:**

***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***.

**CODE:**

CREATE TABLE customers\_vip2 (

customer\_id INTEGER,

first\_name VARCHAR(100),

last\_name VARCHAR(100),

age INTEGER,

country VARCHAR(100),

balance INTEGER,

IsVIP VARCHAR(10));

INSERT INTO customers\_vip2 VALUES (1, 'John', 'Doe', 35, 'USA', 5000, 'FALSE');

INSERT INTO customers\_vip2 VALUES (2, 'Robert', 'Luna', 42, 'USA', 15000, 'FALSE');

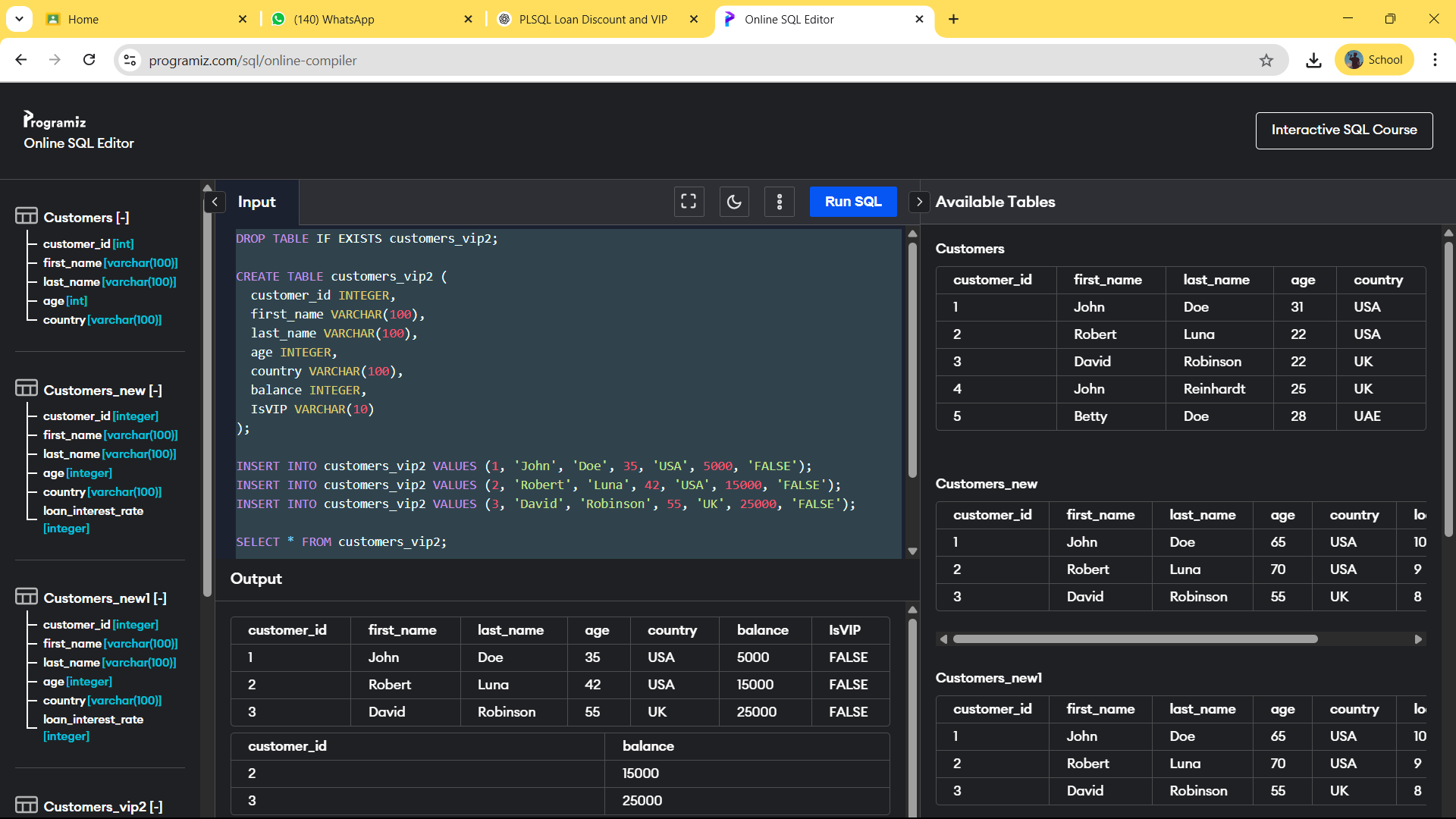
INSERT INTO customers\_vip2 VALUES (3, 'David', 'Robinson', 55, 'UK', 25000, 'FALSE');

SELECT \* FROM customers\_vip2;

SELECT customer\_id, balance

FROM customers\_vip2

WHERE balance > 10000;

**OUTPUT:**

***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.***

**CODE:**

CREATE TABLE loans3 (

loan\_id INTEGER,

customer\_id INTEGER,

due\_date DATE

);

INSERT INTO loans3 VALUES (201, 1, DATE('now', '+10 days'));

INSERT INTO loans3 VALUES (202, 2, DATE('now', '+20 days'));

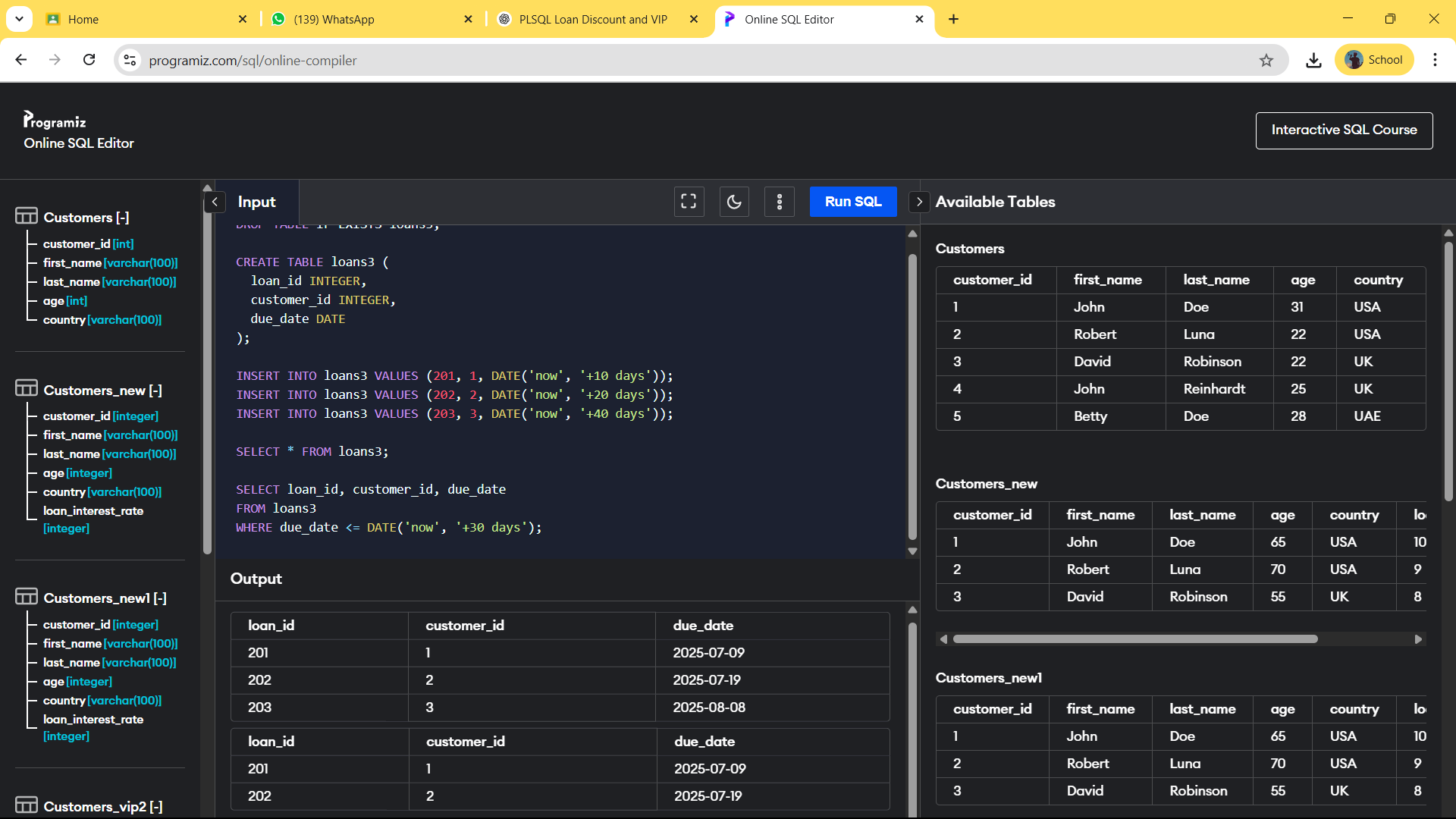
INSERT INTO loans3 VALUES (203, 3, DATE('now', '+40 days'));

SELECT \* FROM loans3;

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

FROM loans3

WHERE due\_date <= DATE('now', '+30 days');

**OUTPUT:**

***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.***

**CODE:**

CREATE TABLE savings\_accounts (

account\_id INTEGER,

customer\_name TEXT,

balance REAL

);

INSERT INTO savings\_accounts VALUES (1, 'John Doe', 1000);

INSERT INTO savings\_accounts VALUES (2, 'Jane Smith', 2000);

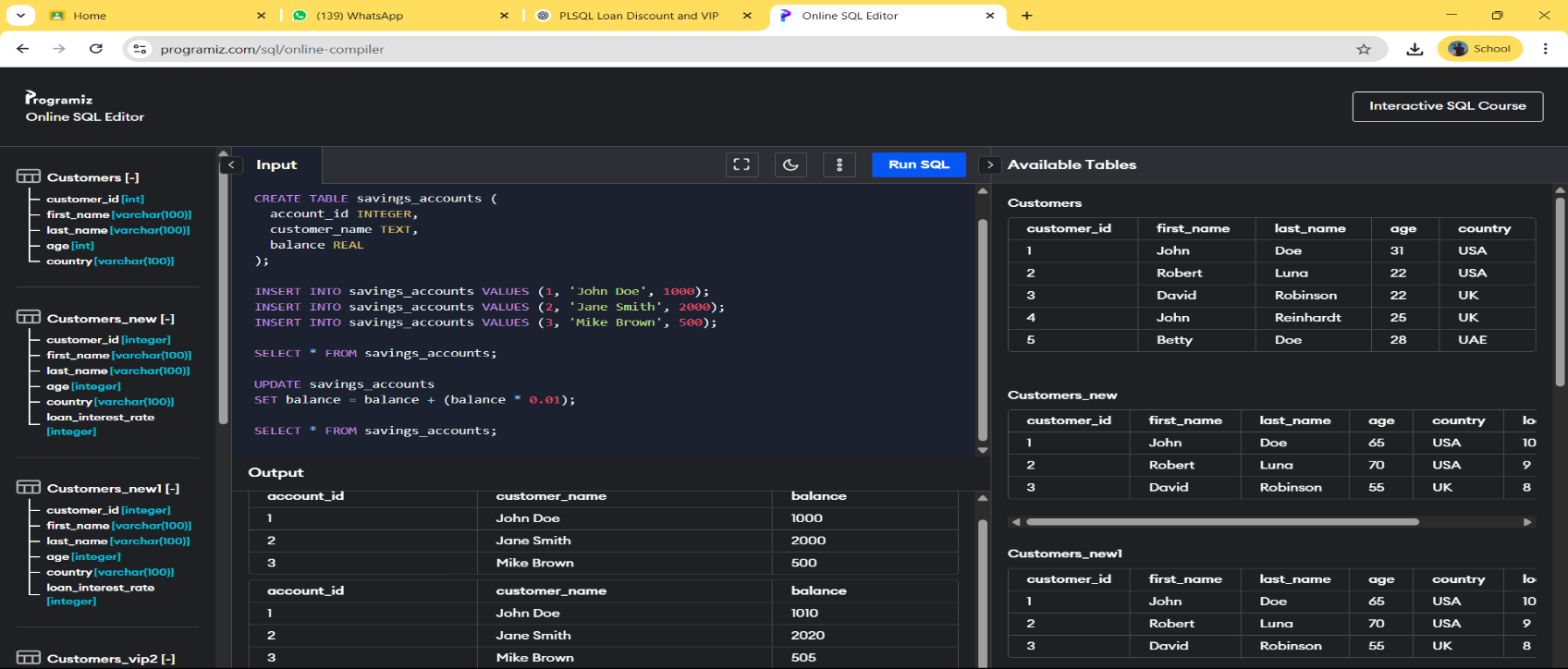
INSERT INTO savings\_accounts VALUES (3, 'Mike Brown', 500);

SELECT \* FROM savings\_accounts;

UPDATE savings\_accounts

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

SELECT \* FROM savings\_accounts;

**OUTPUT:**

***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.***

**CODE:**

CREATE TABLE employees (

emp\_id INTEGER,

emp\_name TEXT,

department\_id INTEGER,

salary REAL

);

INSERT INTO employees VALUES (1, 'Alice', 10, 5000);

INSERT INTO employees VALUES (2, 'Bob', 10, 6000);

INSERT INTO employees VALUES (3, 'Charlie', 20, 7000);

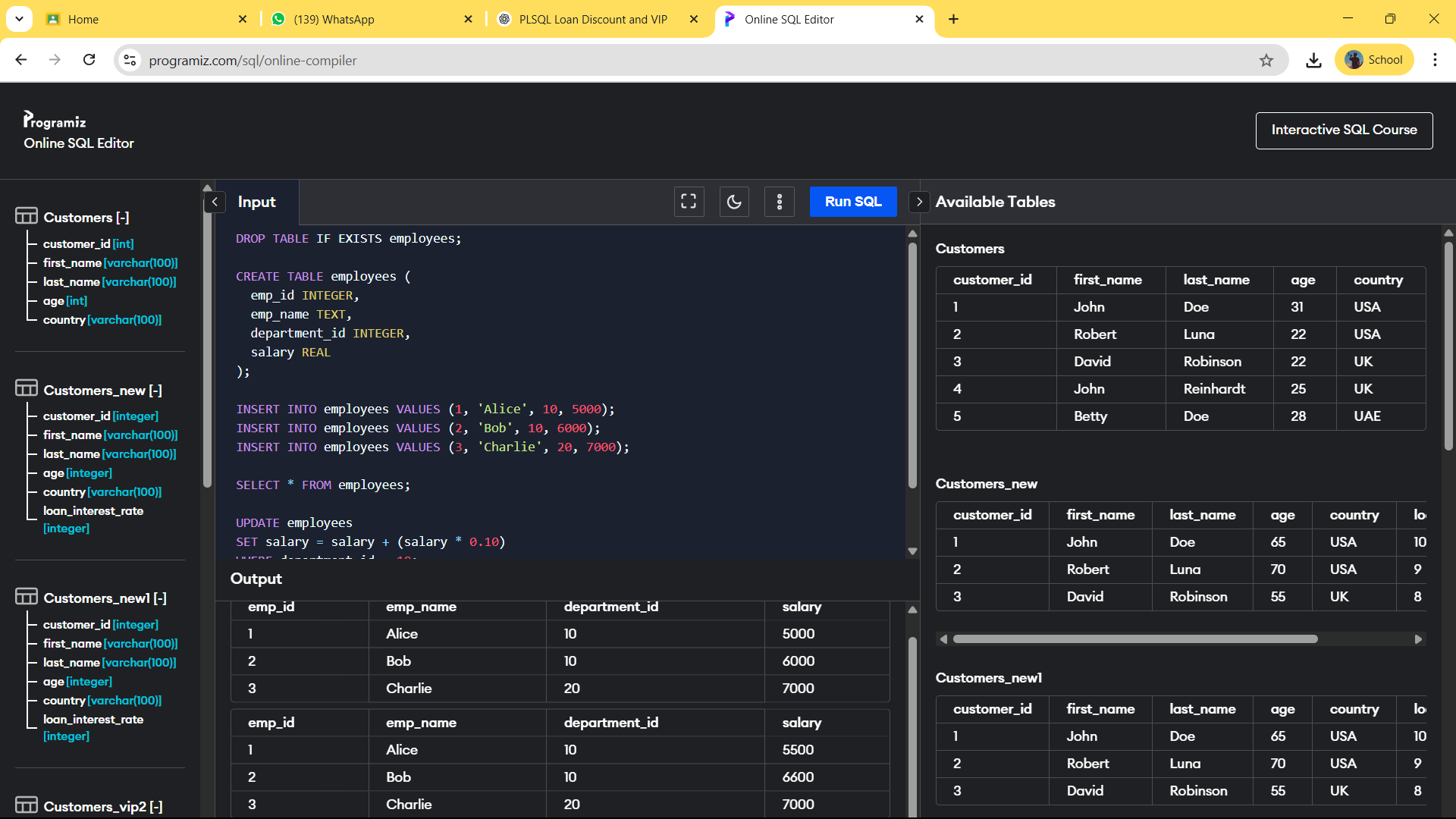
SELECT \* FROM employees;

UPDATE employees

SET salary = salary + (salary \* 0.10)

WHERE department\_id = 10;

SELECT \* FROM employees;

**OUTPUT:**

***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.***

**CODE:**

CREATE TABLE accounts (

account\_id INTEGER,

customer\_name TEXT,

balance REAL

);

INSERT INTO accounts VALUES (1, 'John Doe', 1000);

INSERT INTO accounts VALUES (2, 'Jane Smith', 500);

SELECT \* FROM accounts;

SELECT balance FROM accounts WHERE account\_id = 1;

UPDATE accounts

SET balance = balance - 300

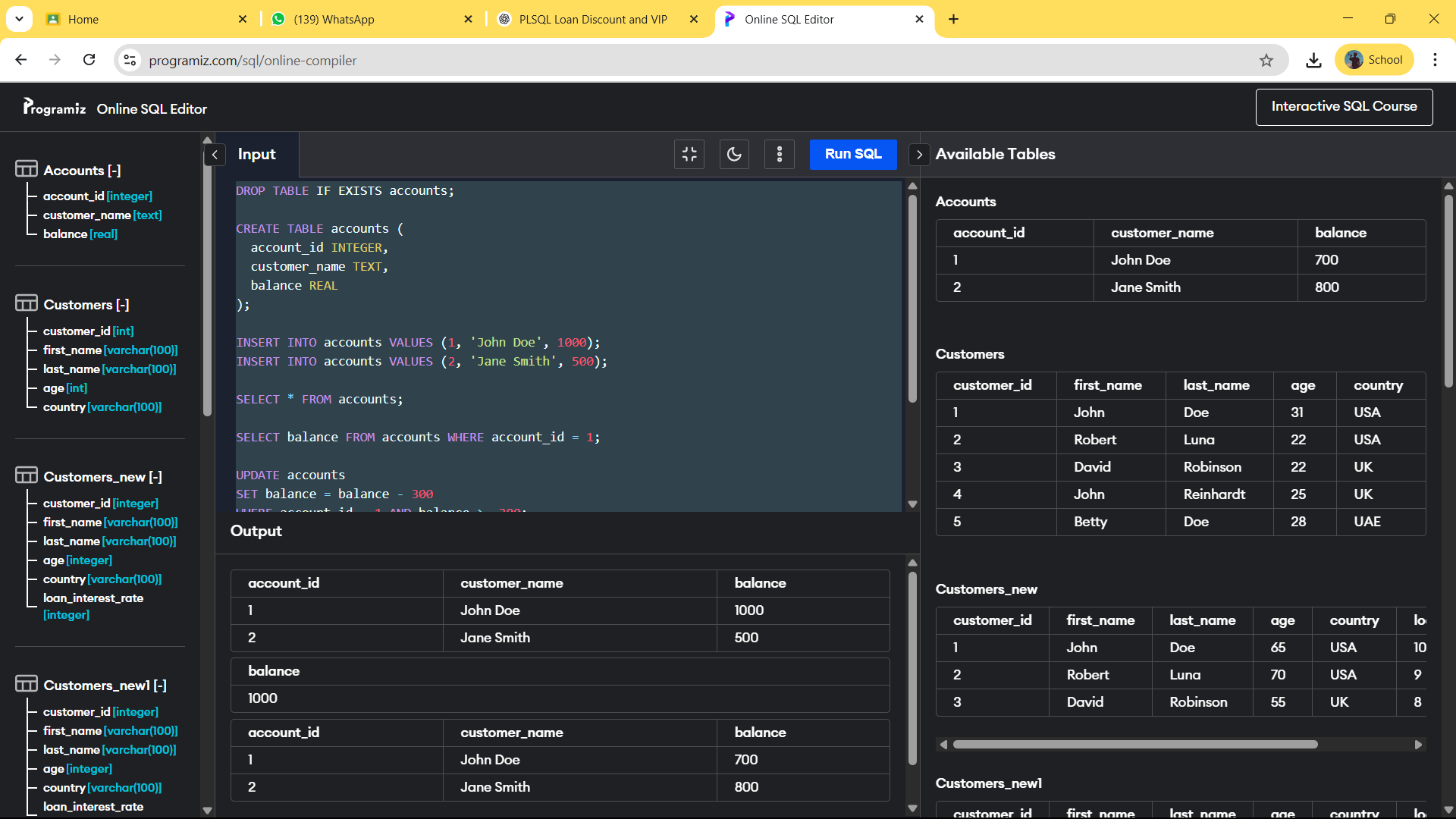
WHERE account\_id = 1 AND balance >= 300;

UPDATE accounts

SET balance = balance + 300

WHERE account\_id = 2;

SELECT \* FROM accounts;

**OUTPUT:**

***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:***

***<dependency>***

***<groupId>junit</groupId>***

***<artifactId>junit</artifactId>***

***<version>4.13.2</version>***

***<scope>test</scope>***

***</dependency>***

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

**CODE:**

import java.util.Scanner;

public class CalculatorTest {

public int add(int a, int b) {

return a + b;

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number: ");

int num1 = sc.nextInt();

System.out.print("Enter second number: ");

int num2 = sc.nextInt();

System.out.print("Enter expected sum: ");

int expected = sc.nextInt();

CalculatorTest test = new CalculatorTest();

int result = test.add(num1, num2);

if (result == expected) {

System.out.println("Test Passed");

} else {

System.out.println("Test Failed");

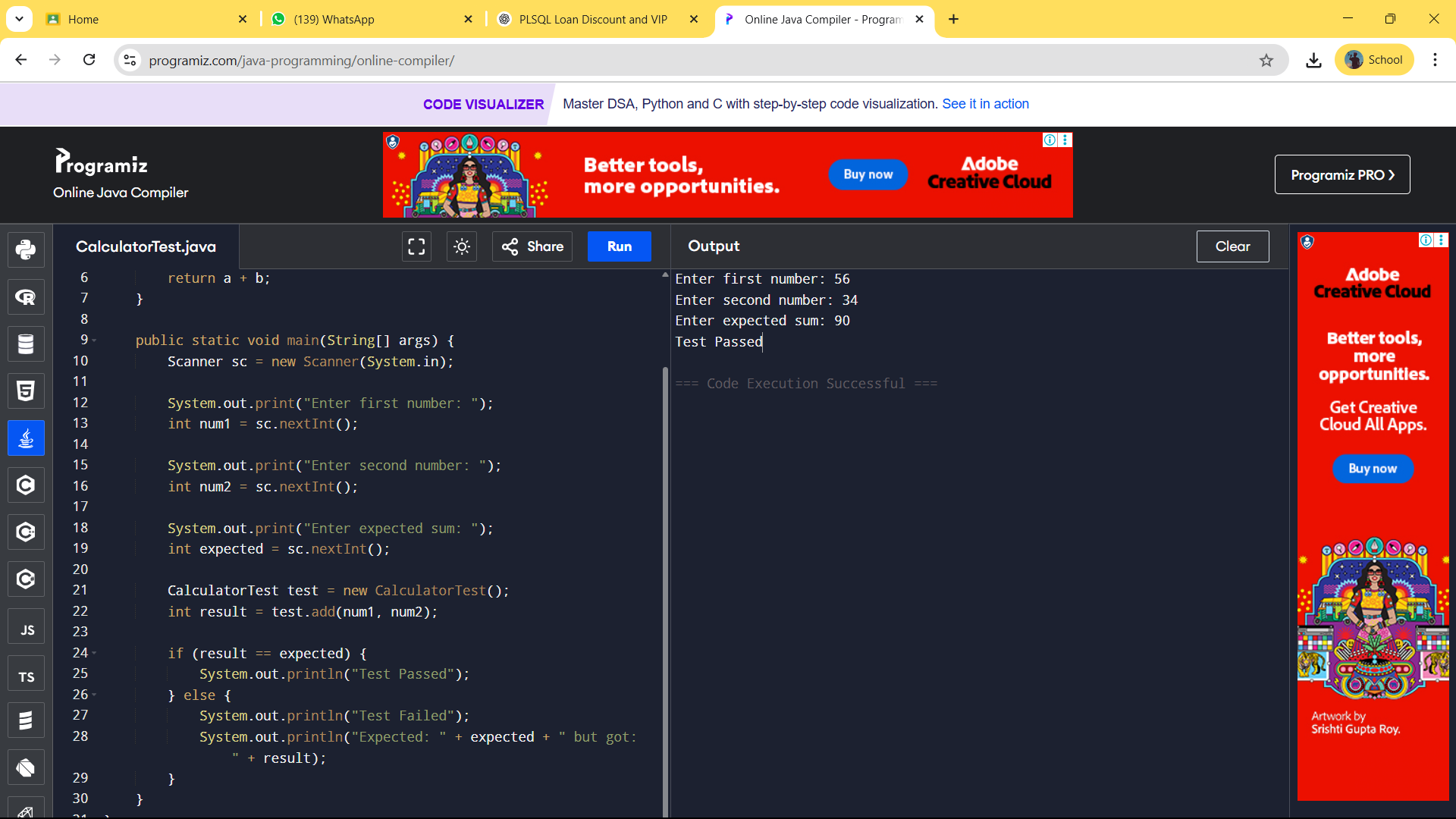
System.out.println("Expected: " + expected + " but got: " + result);

}

}

}

**OUTPUT:**



***Exercise 3: 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.***

***Solution Code:***

***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());***

***}***

***}***

**CODE:**

import java.util.Scanner;

public class AssertionsTest {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter first number for equals: ");

int a = sc.nextInt();

System.out.print("Enter second number for equals: ");

int b = sc.nextInt();

System.out.print("Enter expected sum: ");

int expectedSum = sc.nextInt();

if (a + b == expectedSum) {

System.out.println("assertEquals passed");

} else {

System.out.println("assertEquals failed");

}

System.out.print("Enter a number to check if greater than 3: ");

int num = sc.nextInt();

if (num > 3) {

System.out.println("assertTrue passed");

} else {

System.out.println("assertTrue failed");

}

if (num < 3) {

System.out.println("assertFalse failed");

} else {

System.out.println("assertFalse passed");

}

Object obj1 = null;

if (obj1 == null) {

System.out.println("assertNull passed");

} else {

System.out.println("assertNull failed");

}

Object obj2 = new Object();

if (obj2 != null) {

System.out.println("assertNotNull passed");

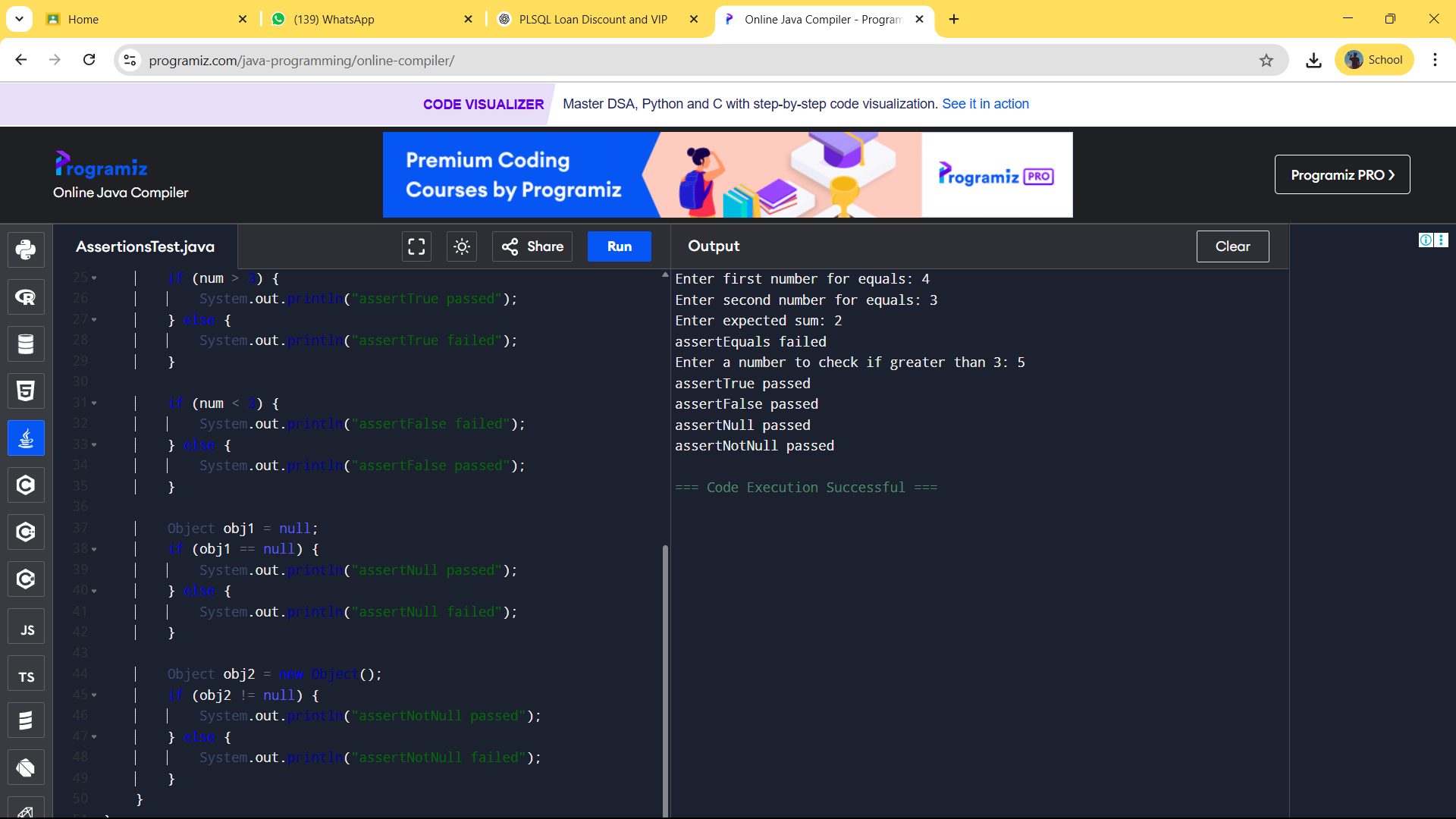
} else {

System.out.println("assertNotNull failed");

}

}

}

**OUTPUT:**

***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.***

***Steps:***

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

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

**CODE:**

import java.util.Scanner;

public class CalculatorTest {

private Calculator calculator;

public void setUp() {

System.out.println("Setting up Calculator instance");

calculator = new Calculator();

}

public void tearDown() {

System.out.println("Cleaning up Calculator instance");

calculator = null; }

public void testAdd(int a, int b, int expected) {

setUp();

System.out.println("Arrange: inputs are " + a + " and " + b + ", expected sum is " + expected);

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

System.out.println("Act: calculated sum is " + result);

if (result == expected) {

System.out.println("Assert: Add Test Passed ");

} else {

System.out.println("Assert: Add Test Failed . Expected " + expected + " but got " + result);

}

tearDown();

}

public void testSubtract(int a, int b, int expected) {

setUp();

System.out.println("Arrange: inputs are " + a + " and " + b + ", expected difference is " + expected);

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

System.out.println("Act: calculated difference is " + result);

if (result == expected) {

System.out.println("Assert: Subtract Test Passed ");

} else {

System.out.println("Assert: Subtract Test Failed . Expected " + expected + " but got " + result);

}

tearDown();

}

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

CalculatorTest test = new CalculatorTest();

System.out.print("Enter first number for addition: ");

int a1 = sc.nextInt();

System.out.print("Enter second number for addition: ");

int b1 = sc.nextInt();

System.out.print("Enter expected sum: ");

int expectedAdd = sc.nextInt();

test.testAdd(a1, b1, expectedAdd);

System.out.print("\nEnter first number for subtraction: ");

int a2 = sc.nextInt();

System.out.print("Enter second number for subtraction: ");

int b2 = sc.nextInt();

System.out.print("Enter expected difference: ");

int expectedSub = sc.nextInt();

test.testSubtract(a2, b2, expectedSub);

}

}

class Calculator {

public int add(int a, int b) {

return a + b;

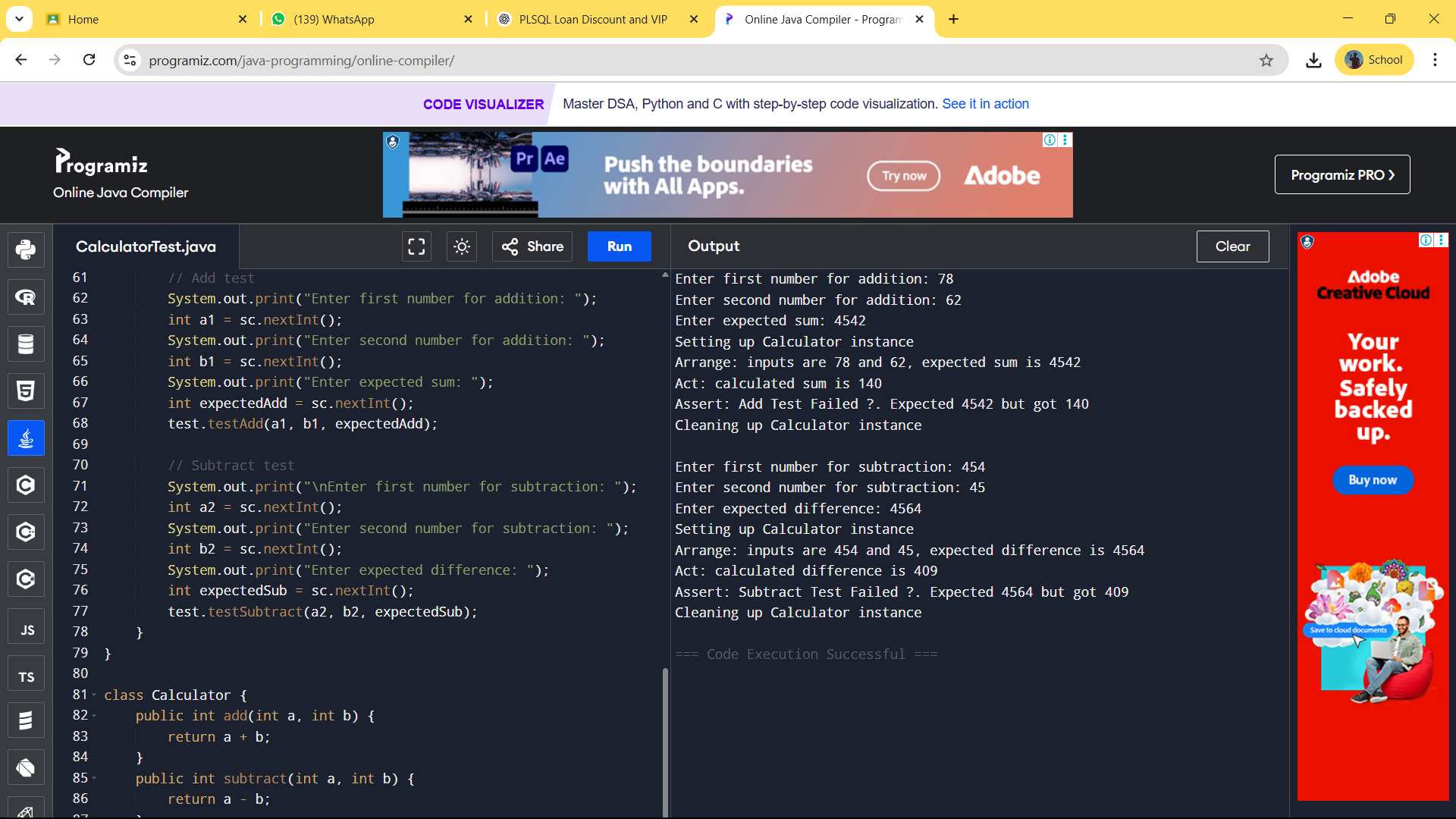
}

public int subtract(int a, int b) {

return a - b;

}

}

**OUTPUT:**

***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.***

***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.***

***Solution Code:***

***import static org.mockito.Mockito.\*;***

***import org.junit.jupiter.api.Test;***

***import org.mockito.Mockito;***

***public class MyServiceTest {***

***@Test***

***public void testExternalApi() {***

***ExternalApi mockApi = Mockito.mock(ExternalApi.class);***

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

***MyService service = new MyService(mockApi);***

***String result = service.fetchData();***

***assertEquals("Mock Data", result);***

***}***

***}***

**CODE:**

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

public void testExternalApi() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

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

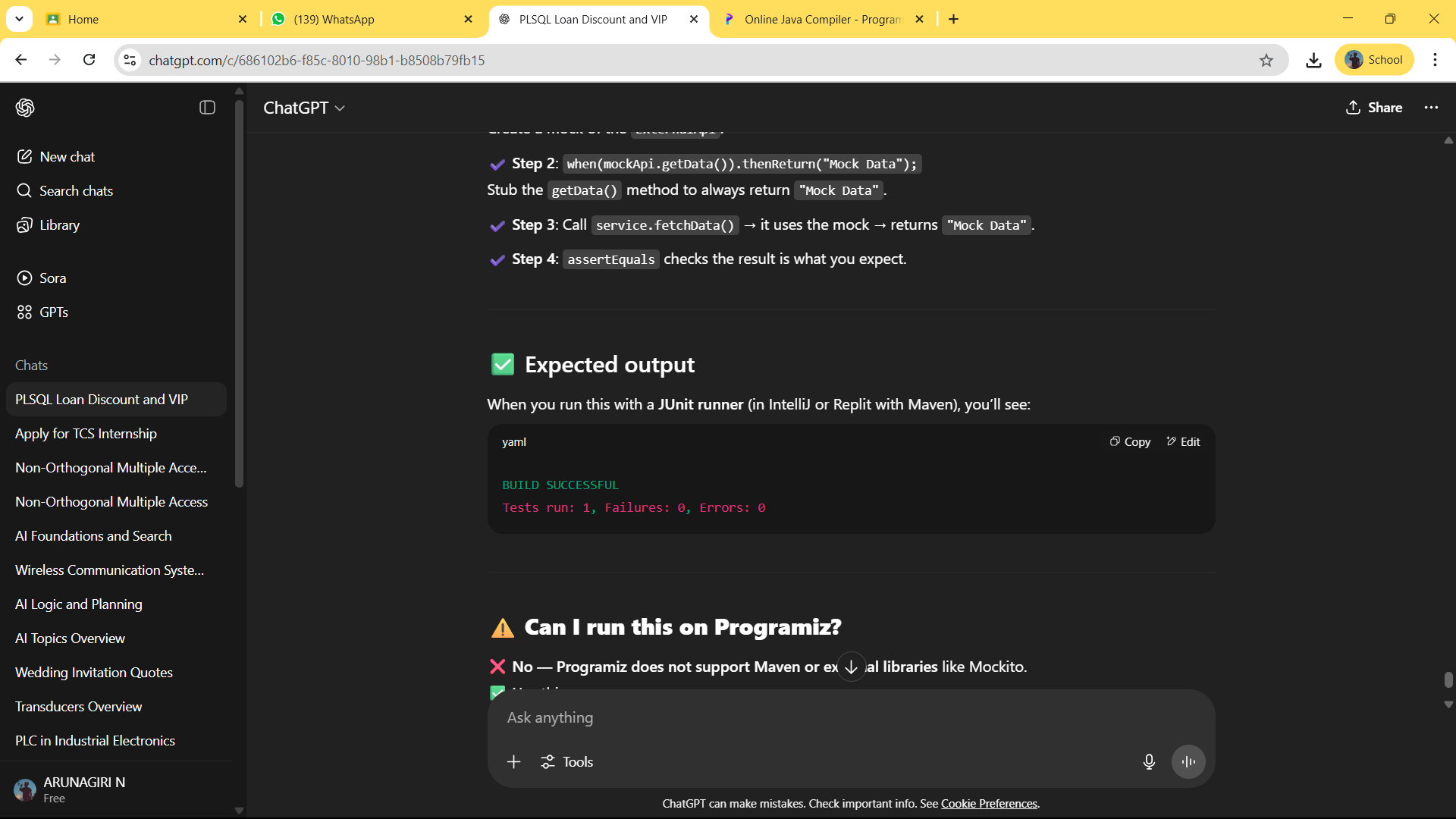
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**

***xercise 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.***

***Solution Code:***

***import static org.mockito.Mockito.\*; import org.junit.jupiter.api.Test;***

***import org.mockito.Mockito;***

***public class MyServiceTest {***

***@Test***

***public void testVerifyInteraction() {***

***ExternalApi mockApi = Mockito.mock(ExternalApi.class);***

***MyService service = new MyService(mockApi);***

***service.fetchData();***

***verify(mockApi).getData();***

***}***

***}***

**CODE:**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

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

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.mock(ExternalApi.class);

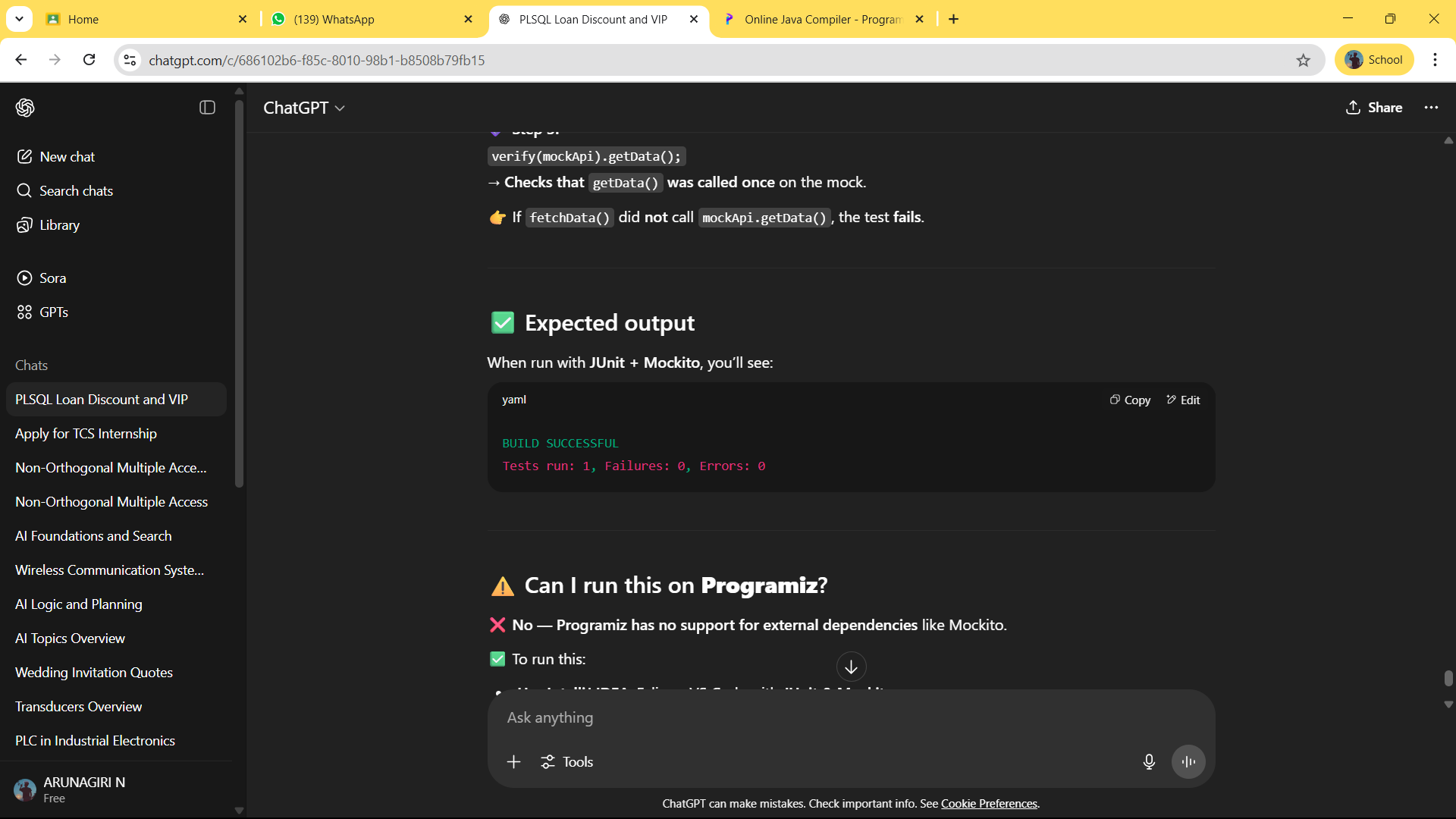
MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**OUTPUT:**

***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:***

***<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>***

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

***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");***

***}***

***}***

**CODE:**

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");

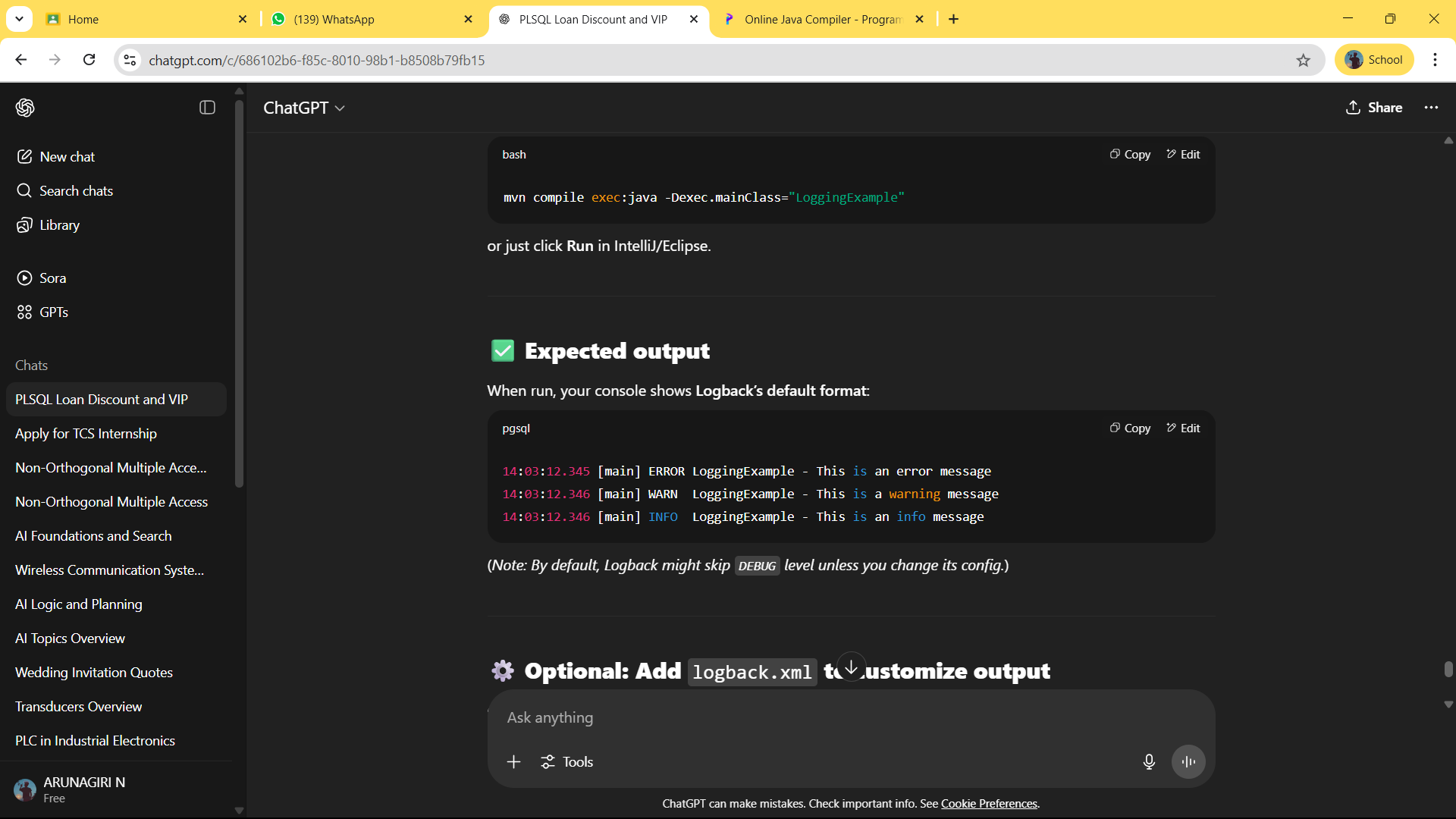
logger.warn("This is a warning message");

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

logger.debug("This is a debug message");

}

}

**OUTPUT:**