**PL/SQL**

**Exercise 1: Control Structures**

**Scenario 1**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Robert Grey', TO\_DATE('1945-09-10', 'YYYY-MM-DD'), 25000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Emily Clark', TO\_DATE('1988-12-25', 'YYYY-MM-DD'), 9500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (5, 'Kevin Patel', TO\_DATE('2000-03-03', 'YYYY-MM-DD'), 12000, SYSDATE);

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (101, 1, 10000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 12));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (102, 2, 8000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 20));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (103, 3, 12000, 5.5, SYSDATE, ADD\_MONTHS(SYSDATE, 10));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (104, 4, 9000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 8));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (105, 5, 11000, 7.0, SYSDATE, ADD\_MONTHS(SYSDATE, 15));

COMMIT;

BEGIN

    FOR cust IN (

        SELECT c.CustomerID, c.DOB, l.LoanID

        FROM Customers c

        JOIN Loans l ON c.CustomerID = l.CustomerID

    ) LOOP

        IF TRUNC(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12) > 60 THEN

            UPDATE Loans

            SET InterestRate = InterestRate - 1

            WHERE LoanID = cust.LoanID;

        END IF;

    END LOOP;

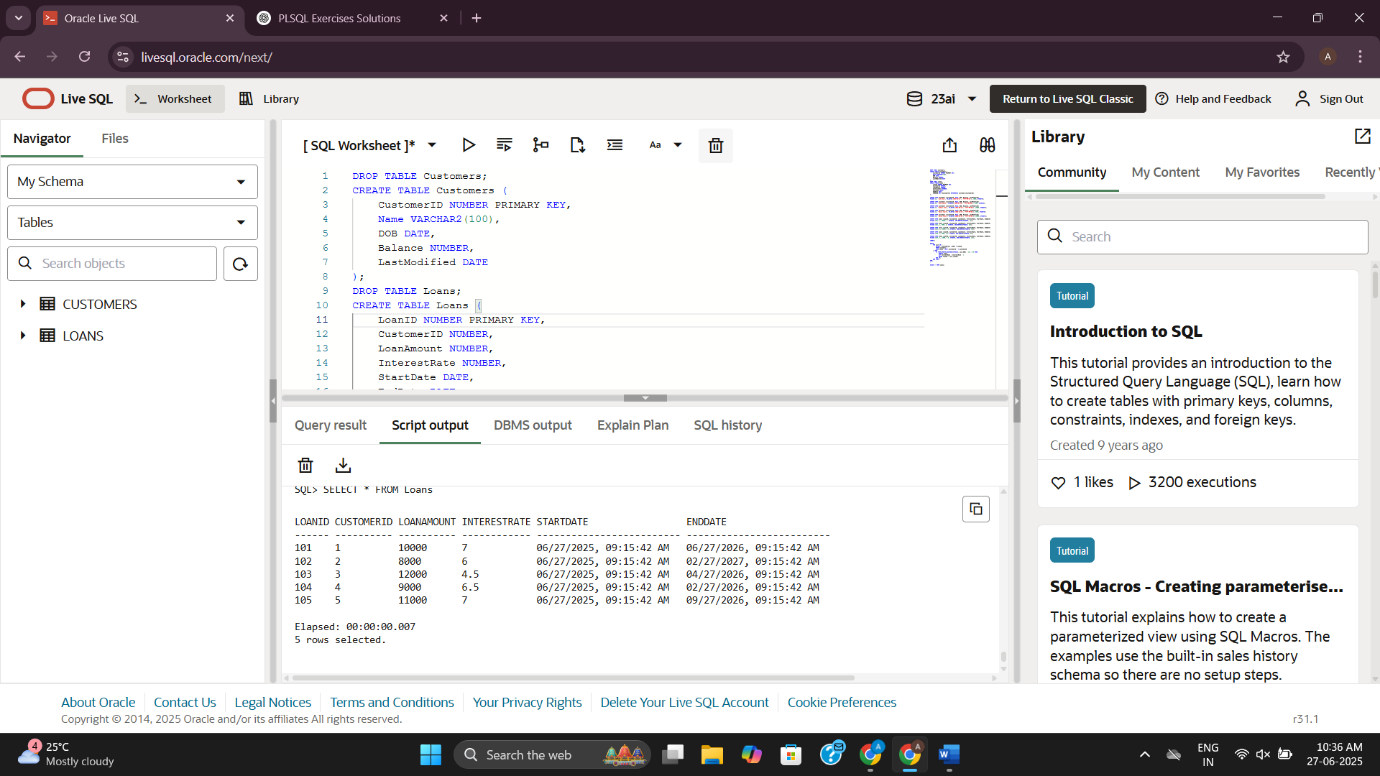
END;

/

SELECT \* FROM Loans;

**Output:**

LoanID 103 (CustomerID 3 – Robert Grey, born 1945) had the interest rate successfully reduced from 5.5% to 4.5% because he is over 60 years old, confirming that the PL/SQL block worked perfectly.

****

**Scenario 2**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Robert Grey', TO\_DATE('1945-09-10', 'YYYY-MM-DD'), 25000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Emily Clark', TO\_DATE('1988-12-25', 'YYYY-MM-DD'), 9500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (5, 'Kevin Patel', TO\_DATE('2000-03-03', 'YYYY-MM-DD'), 12000, SYSDATE);

COMMIT;

ALTER TABLE Customers ADD IsVIP VARCHAR2(1);

BEGIN

    FOR cust IN (SELECT CustomerID, Balance FROM Customers) LOOP

        IF cust.Balance > 10000 THEN

            UPDATE Customers

            SET IsVIP = 'Y',

                LastModified = SYSDATE

            WHERE CustomerID = cust.CustomerID;

        ELSE

            UPDATE Customers

            SET IsVIP = 'N',

                LastModified = SYSDATE

            WHERE CustomerID = cust.CustomerID;

        END IF;

    END LOOP;

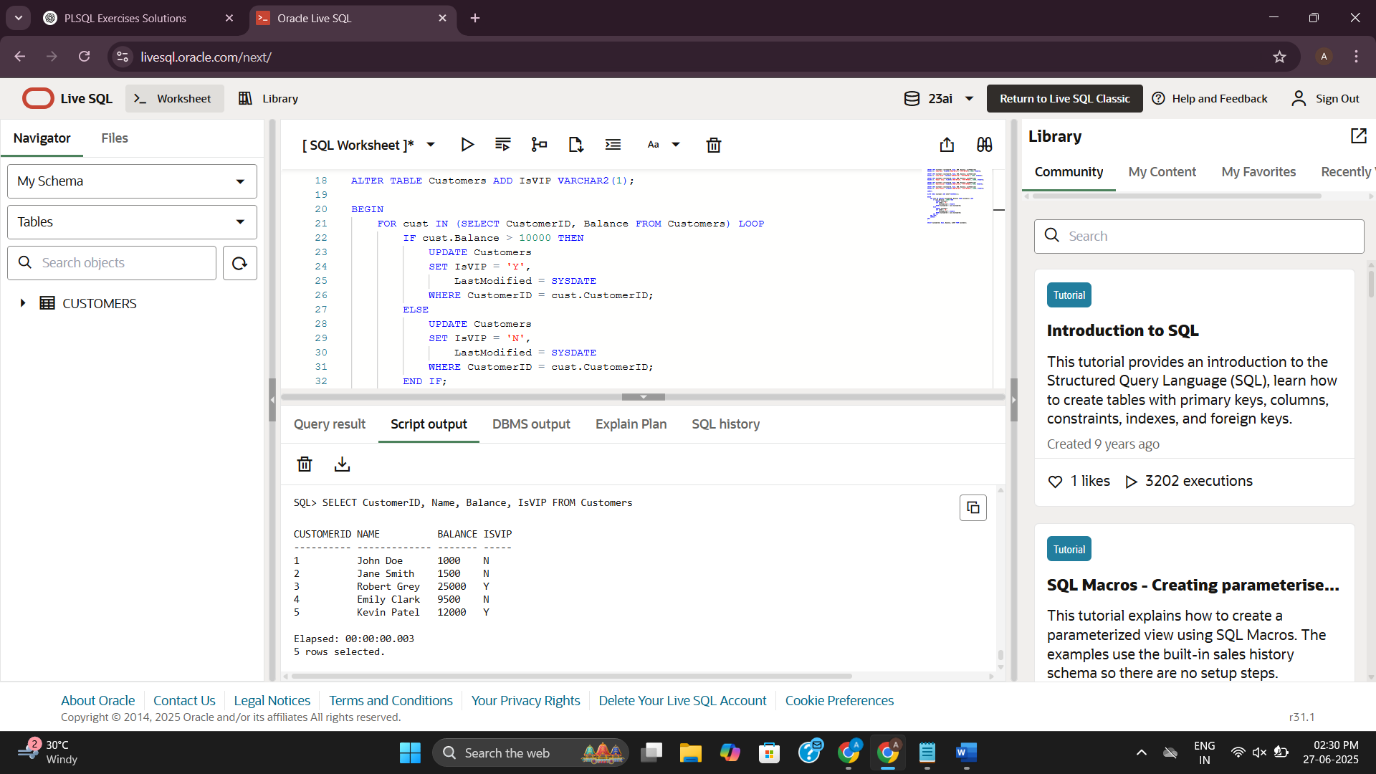
    COMMIT;

END;

/

SELECT CustomerID, Name, Balance, IsVIP FROM Customers;

**Output:**

****

**Scenario 3**

CREATE TABLE Customers (

    CustomerID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    DOB DATE,

    Balance NUMBER,

    LastModified DATE

);

CREATE TABLE Loans (

    LoanID NUMBER PRIMARY KEY,

    CustomerID NUMBER,

    LoanAmount NUMBER,

    InterestRate NUMBER,

    StartDate DATE,

    EndDate DATE,

    FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

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

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (3, 'Robert Grey', TO\_DATE('1945-09-10', 'YYYY-MM-DD'), 25000, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (4, 'Emily Clark', TO\_DATE('1988-12-25', 'YYYY-MM-DD'), 9500, SYSDATE);

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (5, 'Kevin Patel', TO\_DATE('2000-03-03', 'YYYY-MM-DD'), 12000, SYSDATE);

COMMIT;

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (101, 1, 10000, 7, SYSDATE, ADD\_MONTHS(SYSDATE, 12));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (102, 2, 8000, 6, SYSDATE, ADD\_MONTHS(SYSDATE, 20));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (103, 3, 12000, 5.5, SYSDATE, ADD\_MONTHS(SYSDATE, 10));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (104, 4, 9000, 6.5, SYSDATE, ADD\_MONTHS(SYSDATE, 8));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (105, 5, 11000, 7.0, SYSDATE, ADD\_MONTHS(SYSDATE, 15));

INSERT INTO Loans (LoanID, CustomerID, LoanAmount, InterestRate, StartDate, EndDate)

VALUES (106, 1, 5000, 5.5, SYSDATE, SYSDATE + 10);

COMMIT;

SET SERVEROUTPUT ON;

BEGIN

    FOR loan\_rec IN (

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

        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') ||

            '. Amount due: $' || loan\_rec.LoanAmount

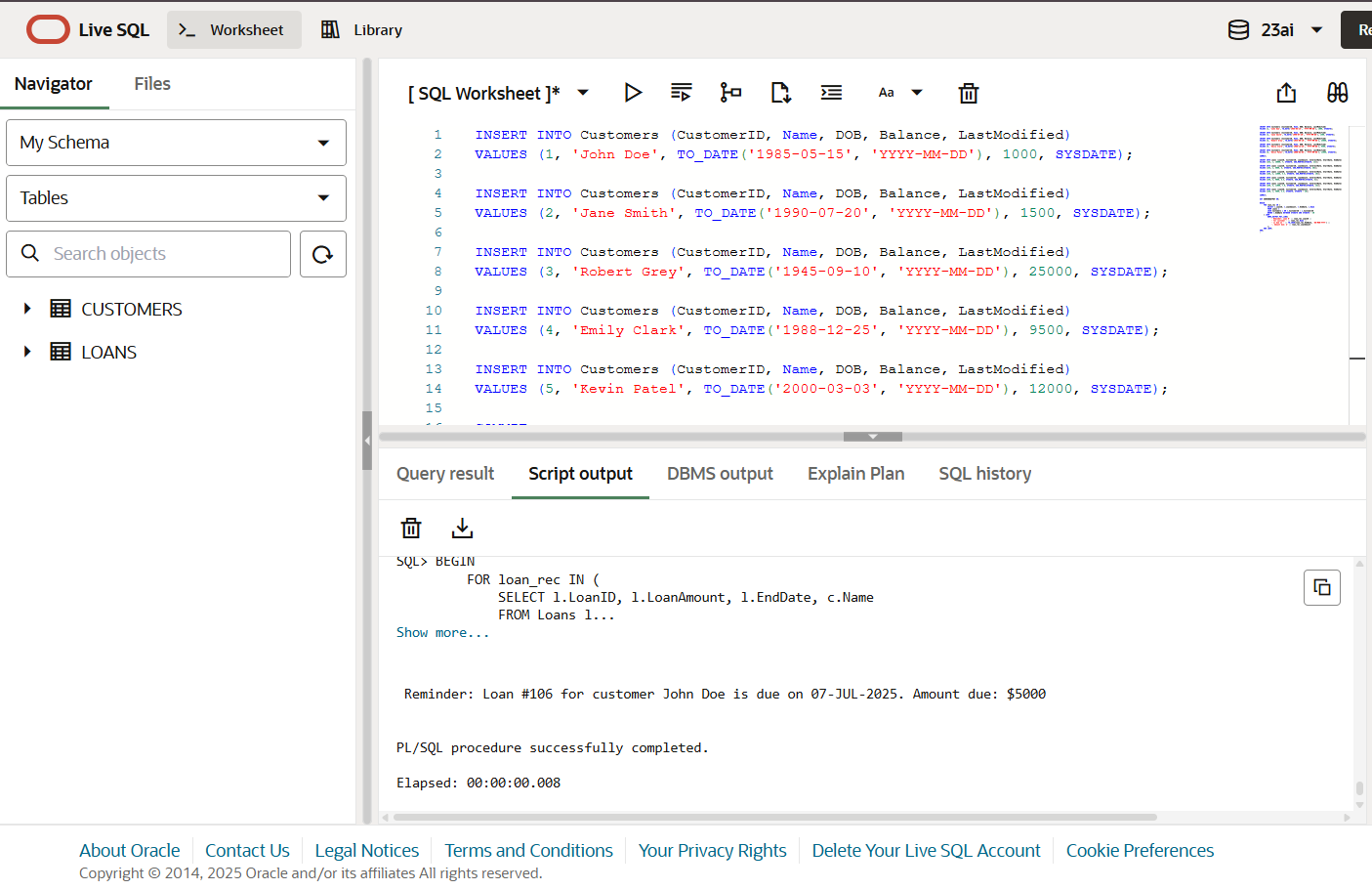
        );

    END LOOP;

END;

/

**Output:**



**Exercise 3: Stored Procedures**

**Scenario 1**

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)

);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (1, 1, 'Savings', 1000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (2, 2, 'Checking', 1500, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (3, 1, 'Savings', 2000, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (4, 2, 'Savings', 1800, SYSDATE);

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (5, 1, 'Checking', 1200, SYSDATE);

COMMIT;

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 3, SYSDATE, 500, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 4, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (5, 5, SYSDATE, 150, 'Deposit');

-- Optional: Create a sequence for generating unique TransactionIDs if not already present

CREATE SEQUENCE Transaction\_Seq START WITH 1000 INCREMENT BY 1;

-- Create the procedure to process monthly interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

    v\_interest NUMBER;

BEGIN

    -- Loop through all savings accounts

    FOR acc IN (

        SELECT AccountID, Balance

        FROM Accounts

        WHERE AccountType = 'Savings'

    ) LOOP

        -- Calculate 1% interest

        v\_interest := acc.Balance \* 0.01;

        -- Update the balance in the Accounts table

        UPDATE Accounts

        SET Balance = Balance + v\_interest,

            LastModified = SYSDATE

        WHERE AccountID = acc.AccountID;

        -- Insert a transaction record for the interest

        INSERT INTO Transactions (

            TransactionID,

            AccountID,

            TransactionDate,

            Amount,

            TransactionType

        ) VALUES (

            Transaction\_Seq.NEXTVAL,

            acc.AccountID,

            SYSDATE,

            v\_interest,

            'Interest'

        );

    END LOOP;

    COMMIT;

END;

/

EXEC ProcessMonthlyInterest;

SELECT AccountID, AccountType, Balance, LastModified

FROM Accounts

WHERE AccountType = 'Savings';

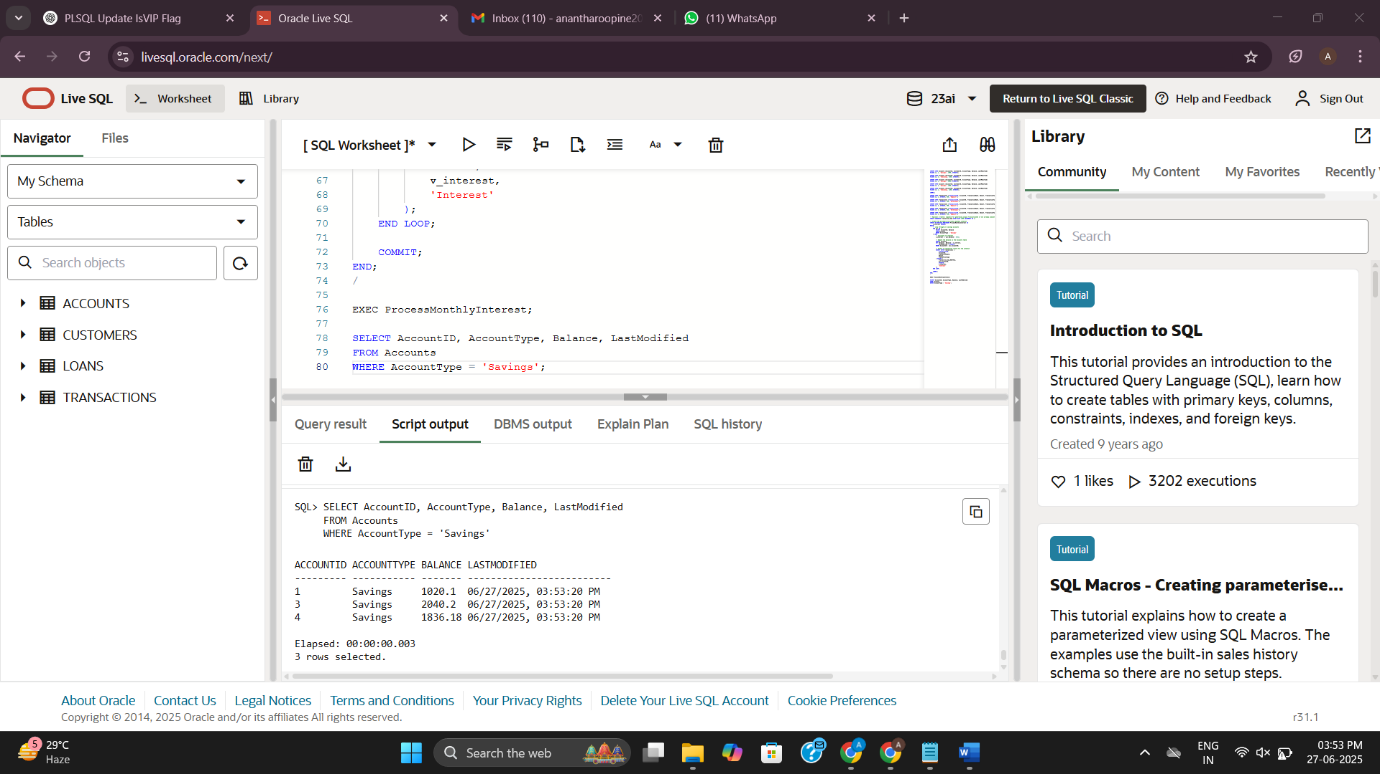
SELECT TransactionID, AccountID, TransactionDate, Amount, TransactionType

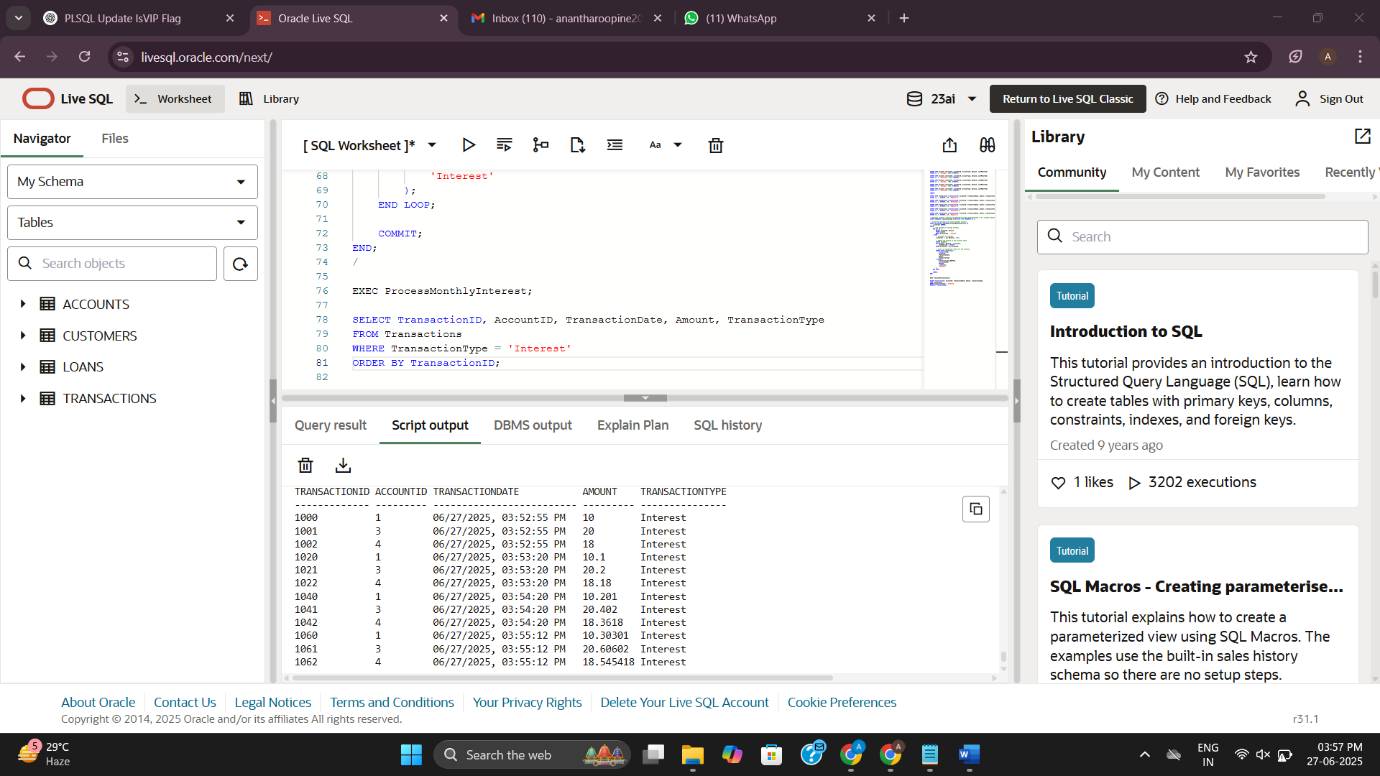
FROM Transactions

WHERE TransactionType = 'Interest'

ORDER BY TransactionID;

**Output:**

****

****

**Scenario 2**

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(100),

    Position VARCHAR2(50),

    Salary NUMBER,

    Department VARCHAR2(50),

    HireDate DATE

);

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

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

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

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

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (3, 'Carol Smith', 'Analyst', 55000, 'Finance', TO\_DATE('2018-11-05', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (4, 'David Lee', 'Support Engineer', 48000, 'IT', TO\_DATE('2019-07-12', 'YYYY-MM-DD'));

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (5, 'Eva Martin', 'HR Assistant', 42000, 'HR', TO\_DATE('2020-02-28', 'YYYY-MM-DD'));

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

    p\_department IN VARCHAR2,

    p\_bonus\_percent IN NUMBER

) AS

BEGIN

    -- Update salaries of employees in the specified department

    UPDATE Employees

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

    WHERE Department = p\_department;

    COMMIT;

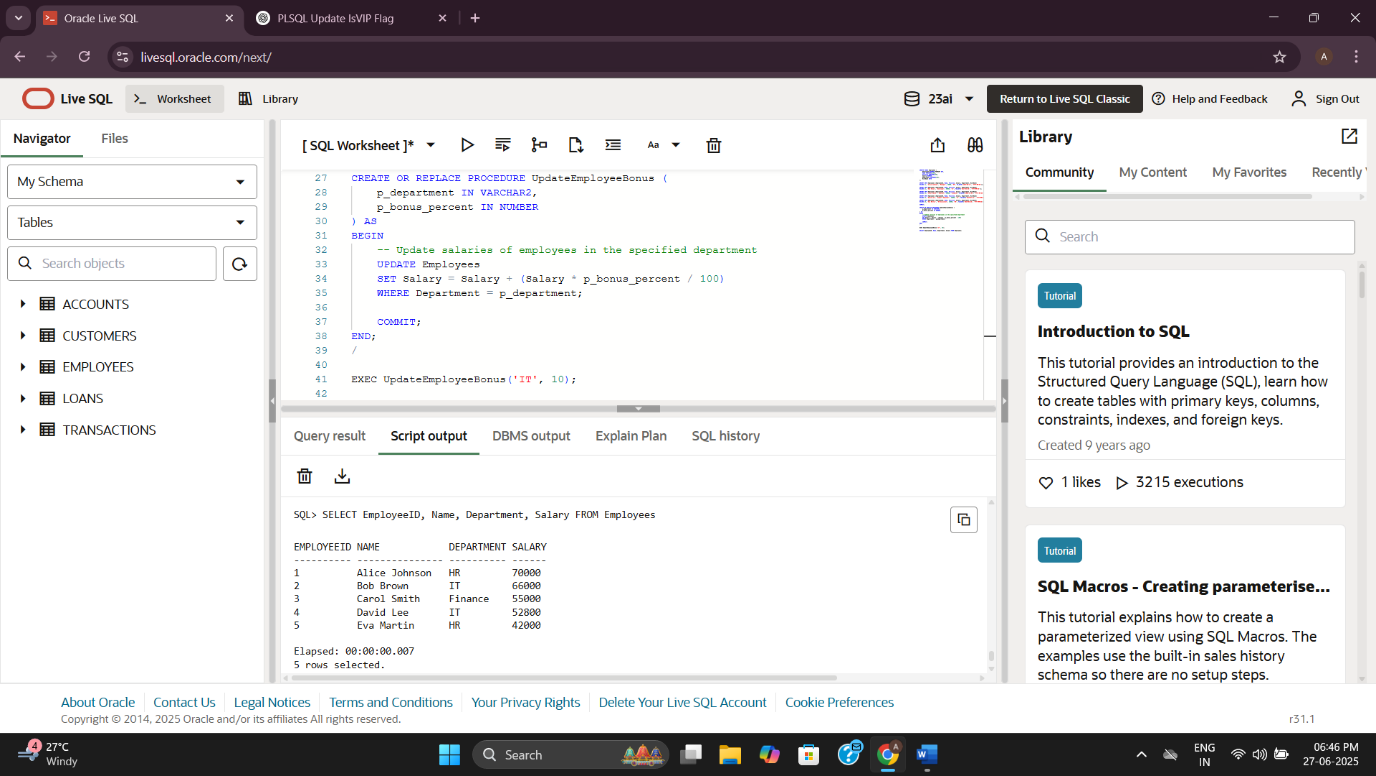
END;

/

EXEC UpdateEmployeeBonus('IT', 10);

SELECT EmployeeID, Name, Department, Salary FROM Employees;

**Output:**

****

**Scenario 3**

CREATE TABLE Transactions (

TransactionID NUMBER PRIMARY KEY,

AccountID NUMBER,

TransactionDate DATE,

Amount NUMBER,

TransactionType VARCHAR2(10),

FOREIGN KEY (AccountID) REFERENCES Accounts(AccountID)

);

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (1, 1, SYSDATE, 200, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (2, 2, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (3, 3, SYSDATE, 500, 'Deposit');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (4, 4, SYSDATE, 300, 'Withdrawal');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (5, 5, SYSDATE, 150, 'Deposit');

ALTER TABLE Transactions

MODIFY TransactionType VARCHAR2(20);

CREATE OR REPLACE PROCEDURE TransferFunds (

    p\_from\_account\_id IN NUMBER,

    p\_to\_account\_id IN NUMBER,

    p\_amount IN NUMBER

) AS

    v\_from\_balance NUMBER;

BEGIN

    -- Get current balance of source account

    SELECT Balance INTO v\_from\_balance

    FROM Accounts

    WHERE AccountID = p\_from\_account\_id

    FOR UPDATE;

    -- Check if balance is sufficient

    IF v\_from\_balance < p\_amount THEN

        RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient balance in source account.');

    END IF;

    -- Deduct amount from source account

    UPDATE Accounts

    SET Balance = Balance - p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_from\_account\_id;

    -- Add amount to destination account

    UPDATE Accounts

    SET Balance = Balance + p\_amount,

        LastModified = SYSDATE

    WHERE AccountID = p\_to\_account\_id;

    -- Insert transaction for source account

    INSERT INTO Transactions (

        TransactionID, AccountID, TransactionDate, Amount, TransactionType

    ) VALUES (

        Transaction\_Seq.NEXTVAL, p\_from\_account\_id, SYSDATE, p\_amount, 'Transfer Out'

    );

    -- Insert transaction for destination account

    INSERT INTO Transactions (

        TransactionID, AccountID, TransactionDate, Amount, TransactionType

    ) VALUES (

        Transaction\_Seq.NEXTVAL, p\_to\_account\_id, SYSDATE, p\_amount, 'Transfer In'

    );

    COMMIT;

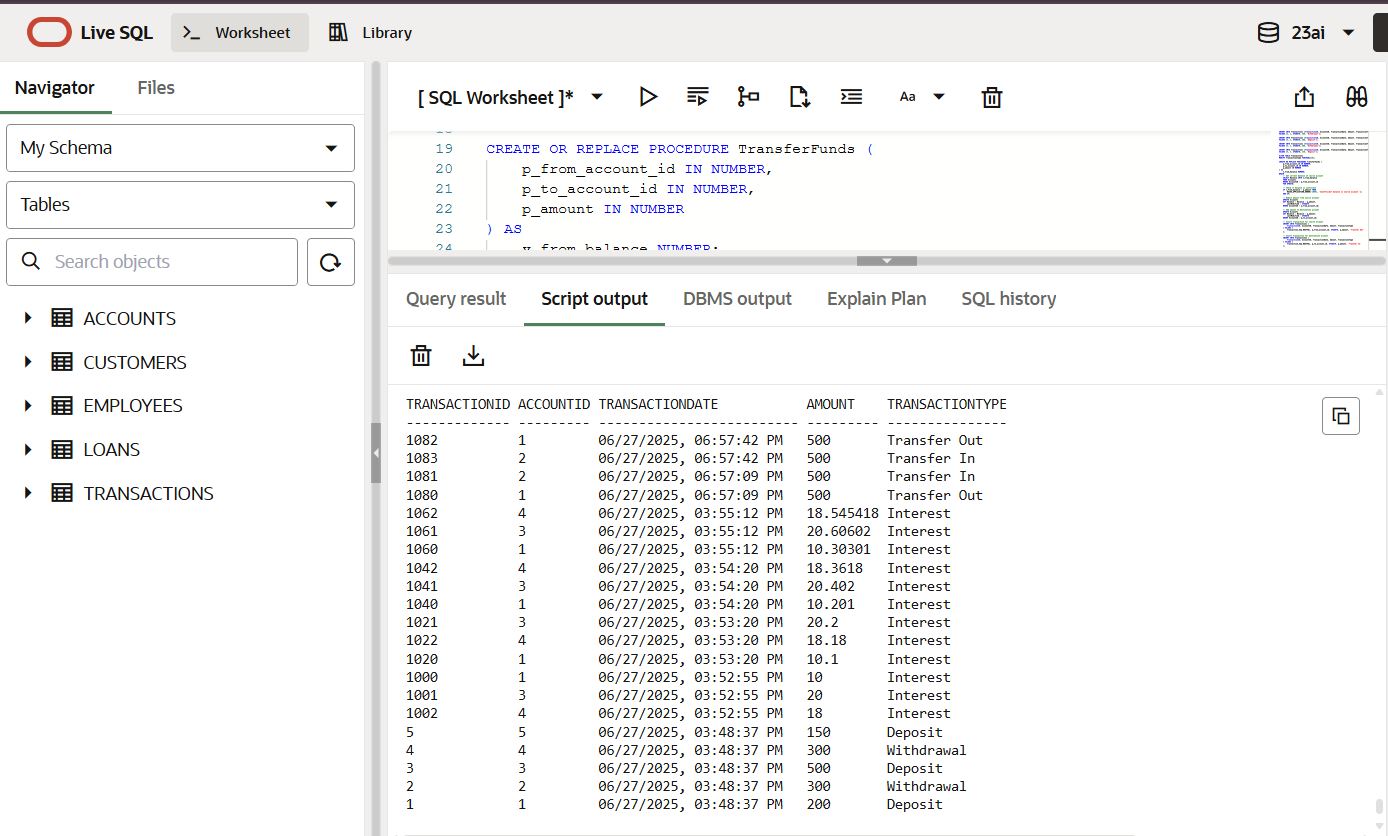
END;

/

EXEC TransferFunds(1, 2, 500);

SELECT \* FROM Transactions ORDER BY TransactionDate DESC;

**Output:**



**JUnit\_Basic Testing**

Exercise 1: Setting Up JUnit

Project Name: MyJUnitProject

Calculator.java Code:

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** a + b;

}

**public** **int** multiply(**int** a, **int** b) {

**return** a \* b;

}

}

CalculatorTest.java Code:

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**public** **class** CalculatorTest {

**private** Calculator;

**public** **void** setUp() {

calculator = **new** Calculator();

}

**public** **void** tearDown() {

calculator = **null**;

}

**public** **void** testAdd() {

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

*assertEquals*(5, result);

}

**public** **void** testMultiply() {

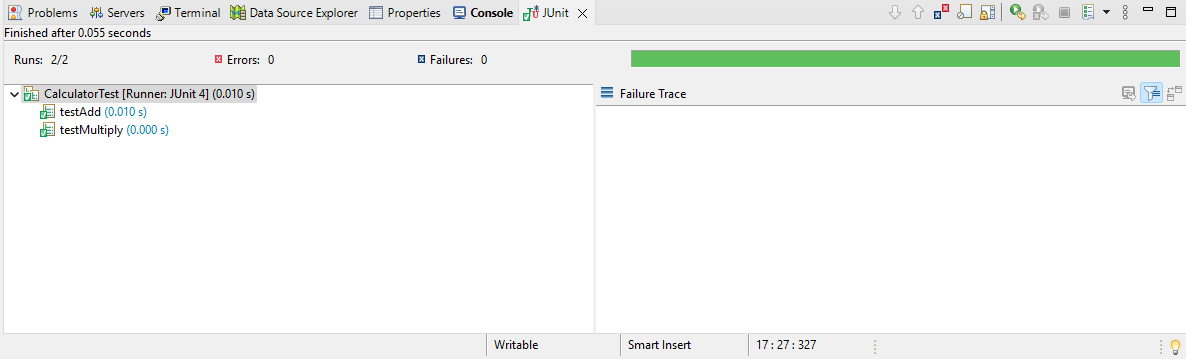
**int** result = calculator.multiply(2, 3);

*assertEquals*(6, result);

}

}

Output:



Exercise 3: Assertions in JUnit

Project Name: MyJUnitProject

AssertionsTest.java Code:

**import** **static** org.junit.Assert.\*;

**import** org.junit.Test;

**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

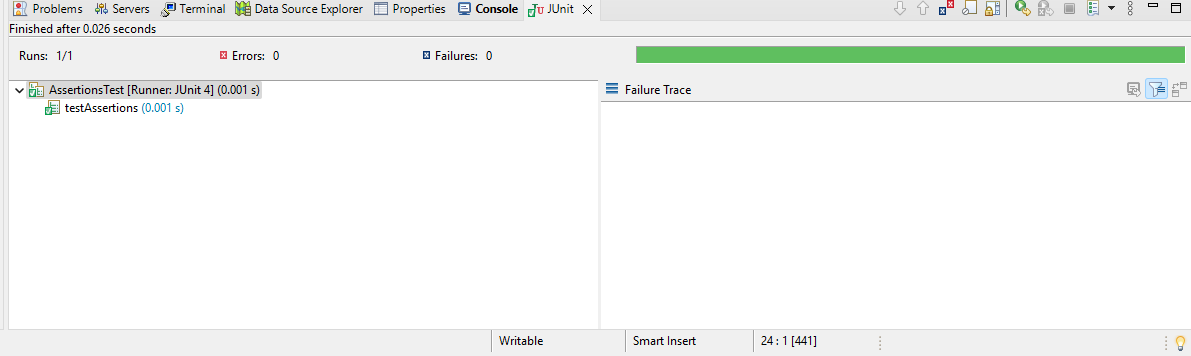
*assertNull*(**null**);

// Assert not null

*assertNotNull*(**new** Object());

}

}  
  
Output:



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

Project Name: MyJUnitProject

CalculatorAAATest.java Code:

**import** **static** org.junit.Assert.\*;

**import** org.junit.Before;

**import** org.junit.After;

**import** org.junit.Test;

**public** **class** CalculatorAAATest {

**private** Calculator;

// Will run BEFORE every test

@Before

**public** **void** setUp() {

// Arrange for every test

calculator = **new** Calculator();

}

// Will run AFTER every test

@After

**public** **void** tearDown() {

// Cleanup after every test

calculator = **null**;

}

// TEST METHOD using the AAA Pattern

@Test

**public** **void** testAdd() {

// Arrange

**int** a = 2;

**int** b = 3;

// Act

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

// Assert

*assertEquals*(5, result);

}

@Test

**public** **void** testSubtract() {

// Arrange

**int** a = 5;

**int** b = 3;

// Act

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

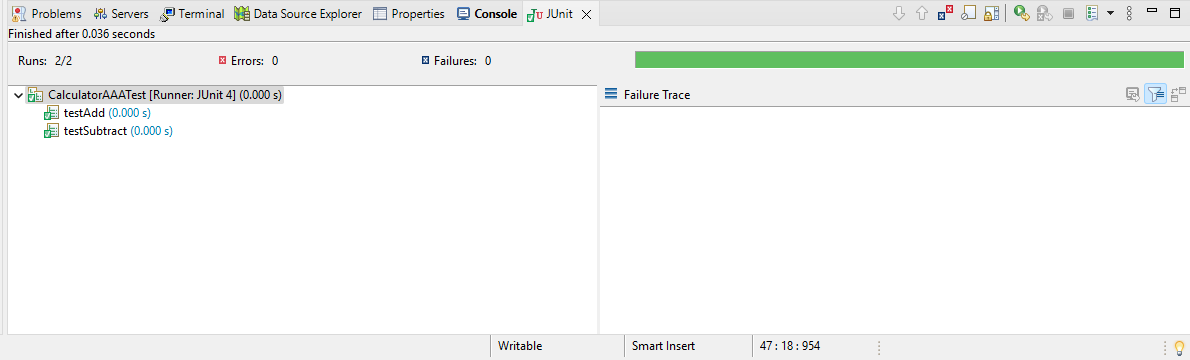
// Assert

*assertEquals*(2, result);

}

}

Output:



**Mockito Exercises**

Exercise 1: Mocking and Stubbing

Project Name:Mockito TestProject

ExternalApi.java Code:

**public** **interface** ExternalApi {

String getData();

}

MyService.java Code:

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceTest.java Code:

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

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

**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);

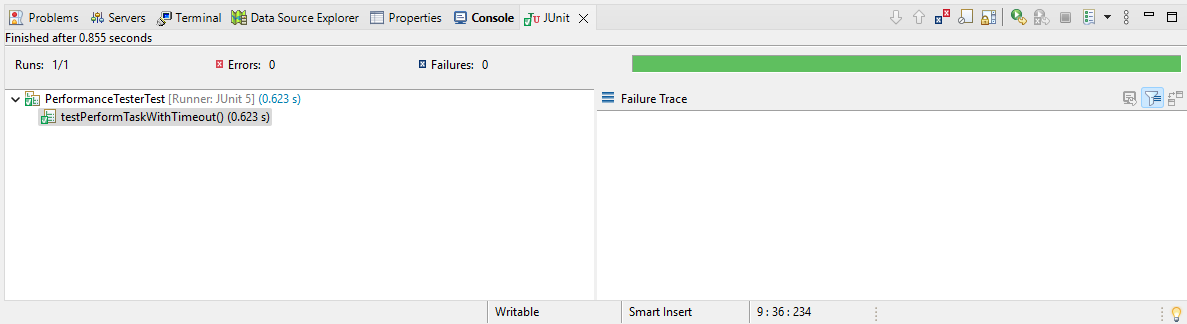
String result = service.fetchData();

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

}

}

Output:



Exercise 2: Verifying Interactions

Project Name:Mockito TestProject

ExternalApi.java Code:

**public** **interface** ExternalApi {

String getData();

}

MyService.java Code:

**public** **class** MyService {

**private** ExternalApi api;

**public** MyService(ExternalApi api) {

**this**.api = api;

}

**public** String fetchData() {

**return** api.getData();

}

}

MyServiceInteractionTest.java Code:

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

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

**import** org.mockito.Mockito;

**public** **class** MyServiceInteractionTest {

@Test

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

// 1. Create a mock of ExternalApi

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

// 2. Pass the mock into MyService

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

// 3. Call the method

service.fetchData();

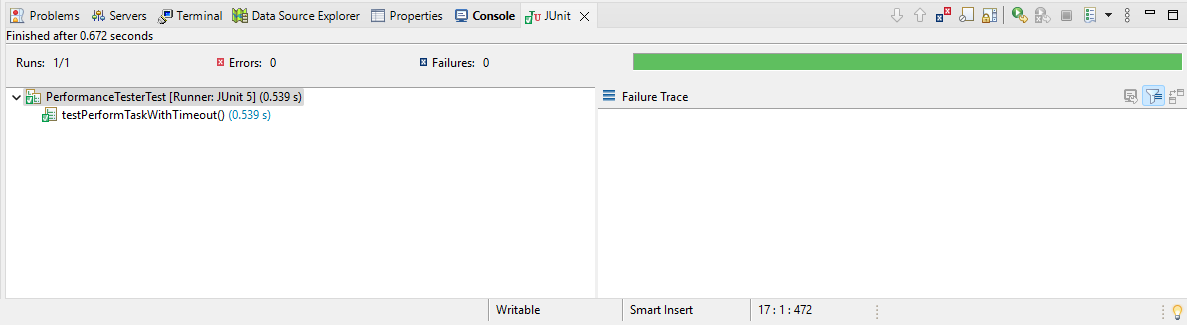
// 4. Verify the interaction

*verify*(mockApi).getData();

}

}

Output:



Logging using SLF4J

Exercise 1: Logging Error Messages and Warning Levels

Project name:LoggingExample

Pom.xl

*<?*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>org.example</groupId>  
 <artifactId>LoggingExample</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 *<!-- SLF4J API -->* <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>1.7.30</version>  
 </dependency>  
  
 *<!-- Logback Implementation -->* <dependency>  
 <groupId>ch.qos.logback</groupId>  
 <artifactId>logback-classic</artifactId>  
 <version>1.2.3</version>  
 </dependency>  
 </dependencies>  
  
</project>

Package:com.example

Class:LoggingExample

package com.example;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class LoggingExample {  
 private static final 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:

