**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.

DECLARE

CURSOR customer\_cursor IS

SELECT c.CustomerID, l.LoanID, l.InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE FLOOR(MONTHS\_BETWEEN(SYSDATE, c.DOB) / 12) > 60;

BEGIN

FOR rect IN customer\_cursor LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

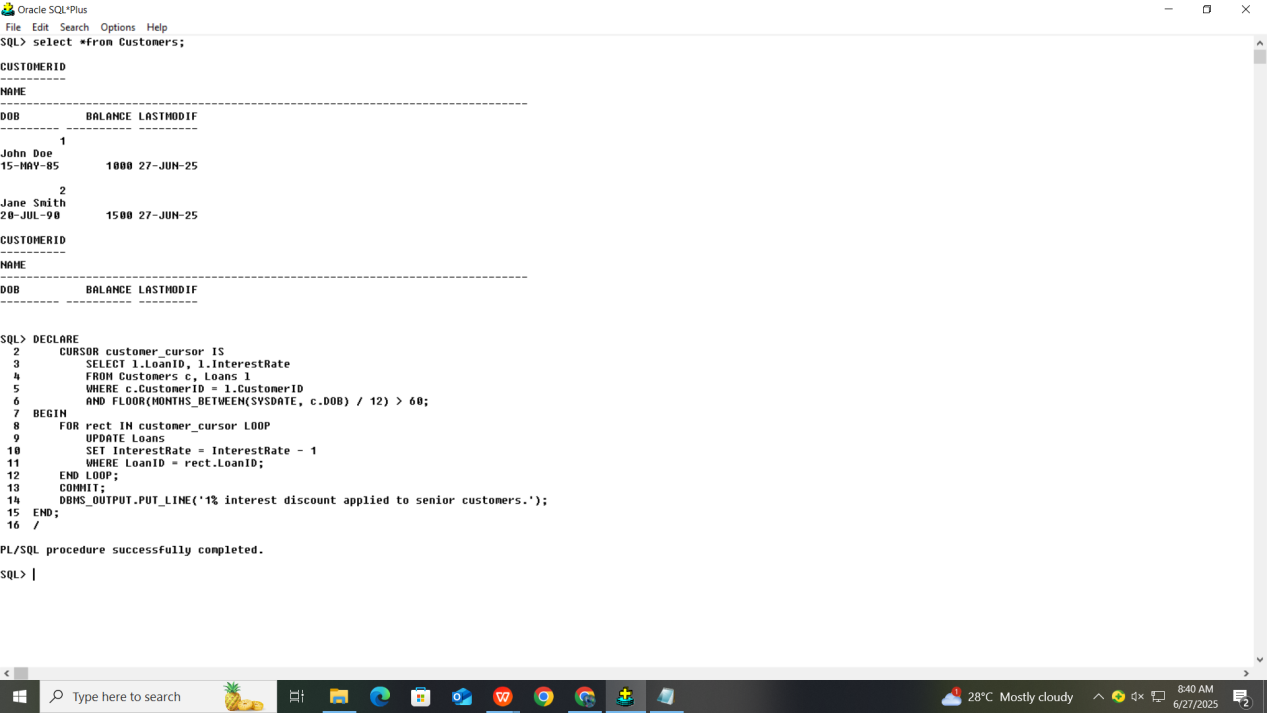
WHERE LoanID = rect.LoanID;

END LOOP;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' 1% interest discount applied to senior customers.');

END;  
**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.

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID

FROM Customers

WHERE Balance > 10000;

BEGIN

FOR rec IN vip\_cursor LOOP

UPDATE Customers

SET ISVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

END LOOP;

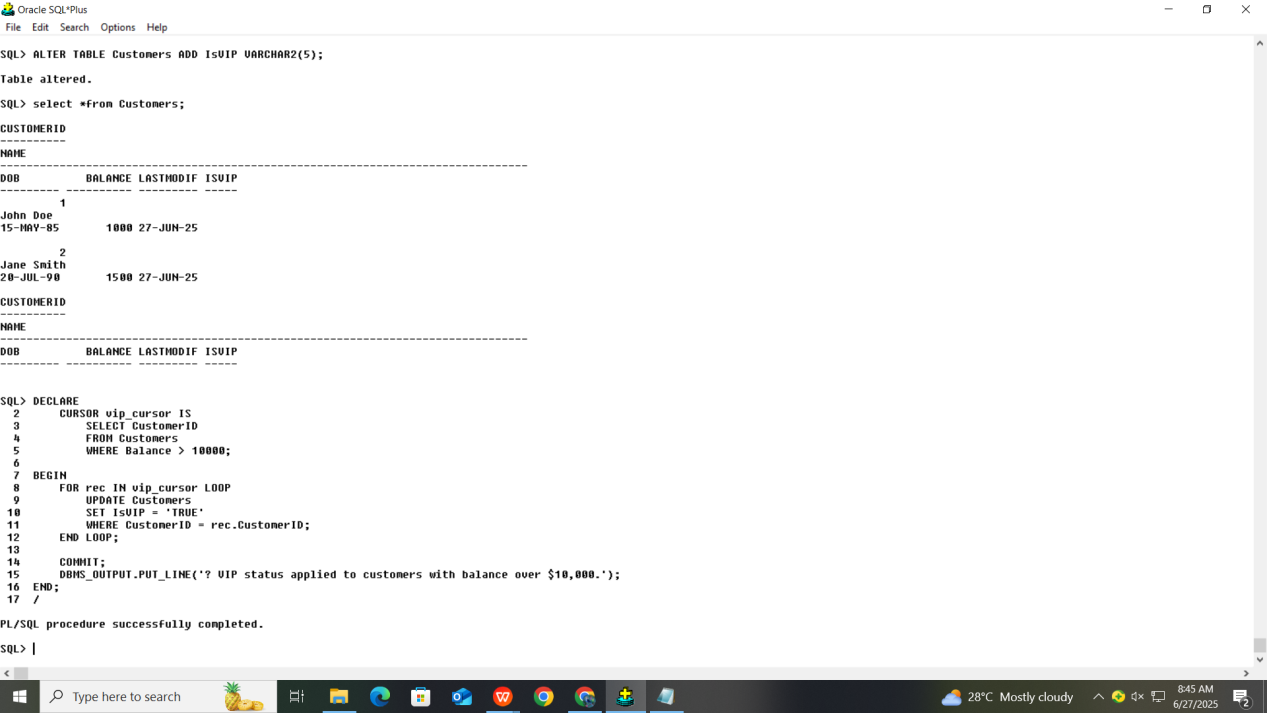
COMMIT;

DBMS\_OUTPUT.PUT\_LINE('? VIP status applied to customers with balance over $10,000.');

END;

/

**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.

DECLARE

CURSOR due\_loan IS

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

FROM Customers c, Loans l

WHERE c.CustomerID = l.CustomerID

AND l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR rec IN due\_loan LOOP

DBMS\_OUTPUT.PUT\_LINE(' Reminder:');

DBMS\_OUTPUT.PUT\_LINE(' Customer Name : ' || rec.Name);

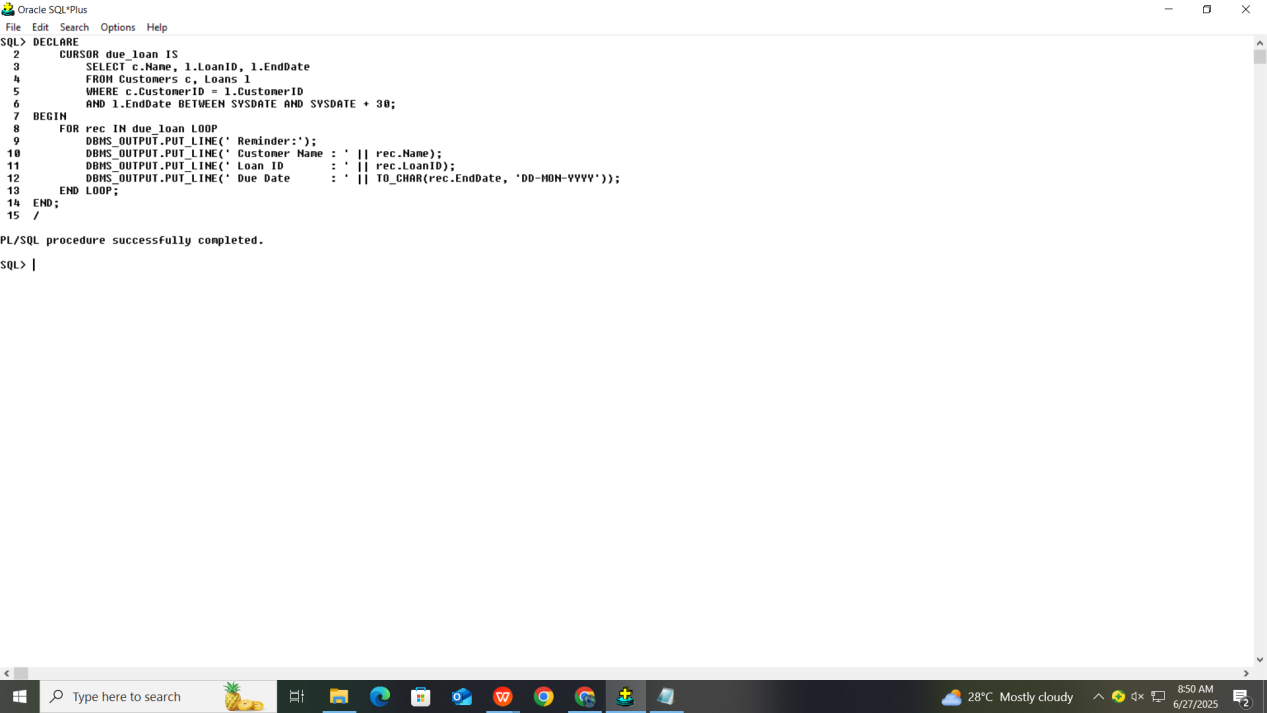
DBMS\_OUTPUT.PUT\_LINE(' Loan ID : ' || rec.LoanID);

DBMS\_OUTPUT.PUT\_LINE(' Due Date : ' || TO\_CHAR(rec.EndDate, 'DD-MON-YYYY'));

END LOOP;

END;

**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.

CREATE OR REPLACE PROCEDURE Process\_Monthly\_Interest IS

BEGIN

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

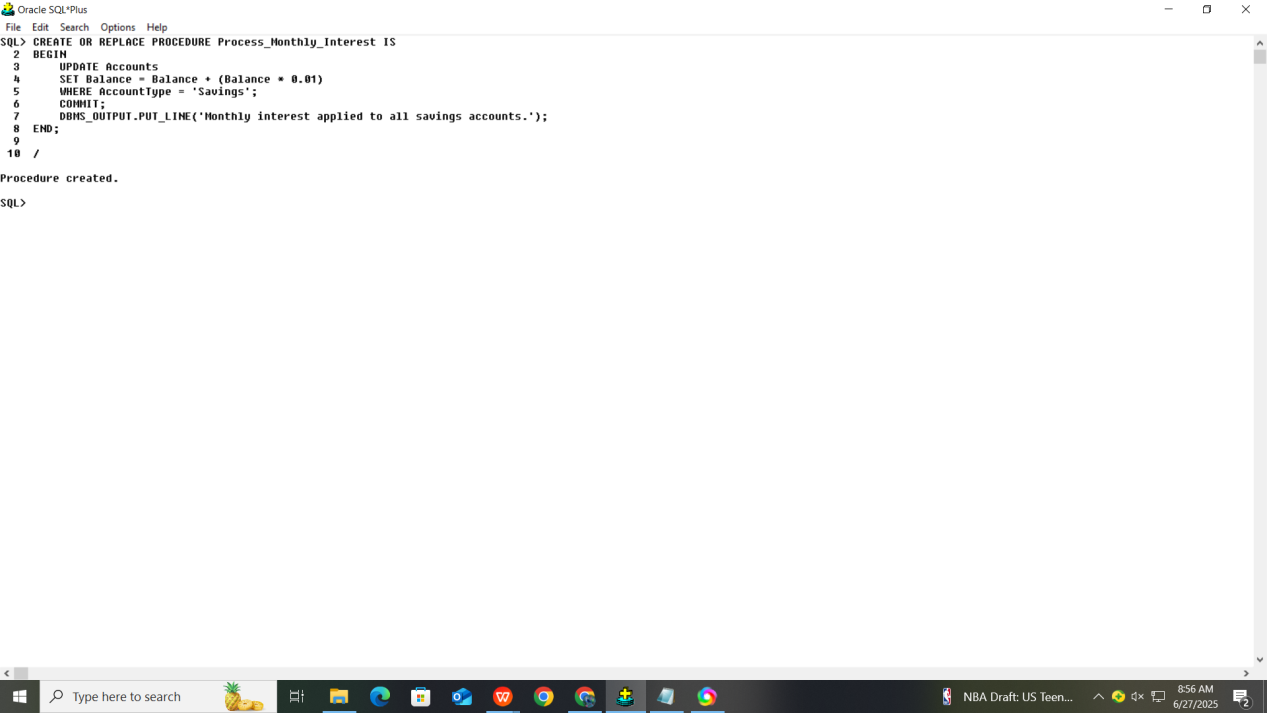
WHERE AccountType = 'Savings';

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Monthly interest applied to all savings accounts.');

END;

**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

CREATE OR REPLACE PROCEDURE Update\_Emp\_Bonus (

Departments IN VARCHAR2,

BonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* (BonusPercent / 100))

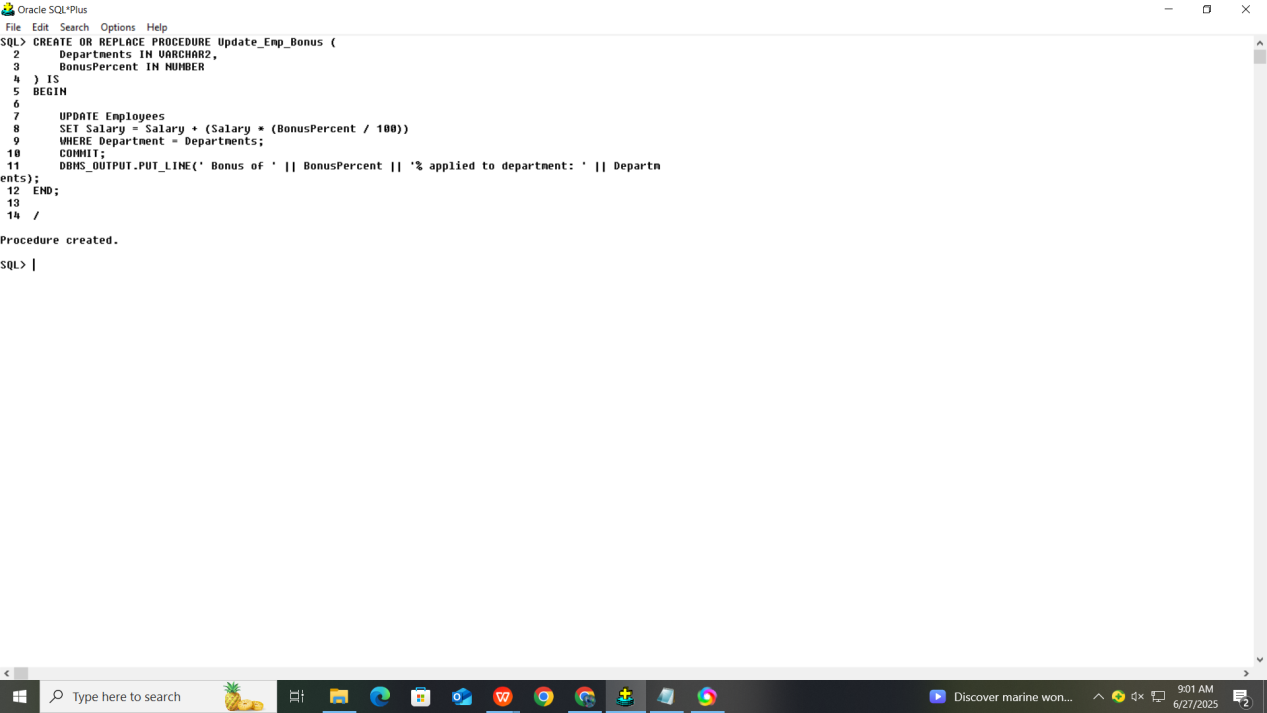
WHERE Department = Departments;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE(' Bonus of ' || BonusPercent || '% applied to department: ' || Departments);

END;

**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.

CREATE OR REPLACE PROCEDURE TransferFunds (

SourceAccountID IN NUMBER,

TargetAccountID IN NUMBER,

Amount IN NUMBER

) IS

SourceBalance NUMBER;

BEGIN

SELECT Balance INTO SourceBalance

FROM Accounts

WHERE AccountID = SourceAccountID

FOR UPDATE;

IF SourceBalance < Amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - Amount

WHERE AccountID = SourceAccountID;

UPDATE Accounts

SET Balance = Balance + Amount

WHERE AccountID = TargetAccountID;

COMMIT;

DBMS\_OUTPUT.PUT\_LINE('Successfully transferred ' || Amount ||

' from Account ID ' || SourceAccountID ||

' to Account ID ' || TargetAccountID);

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE(' One of the account IDs does not exist.');

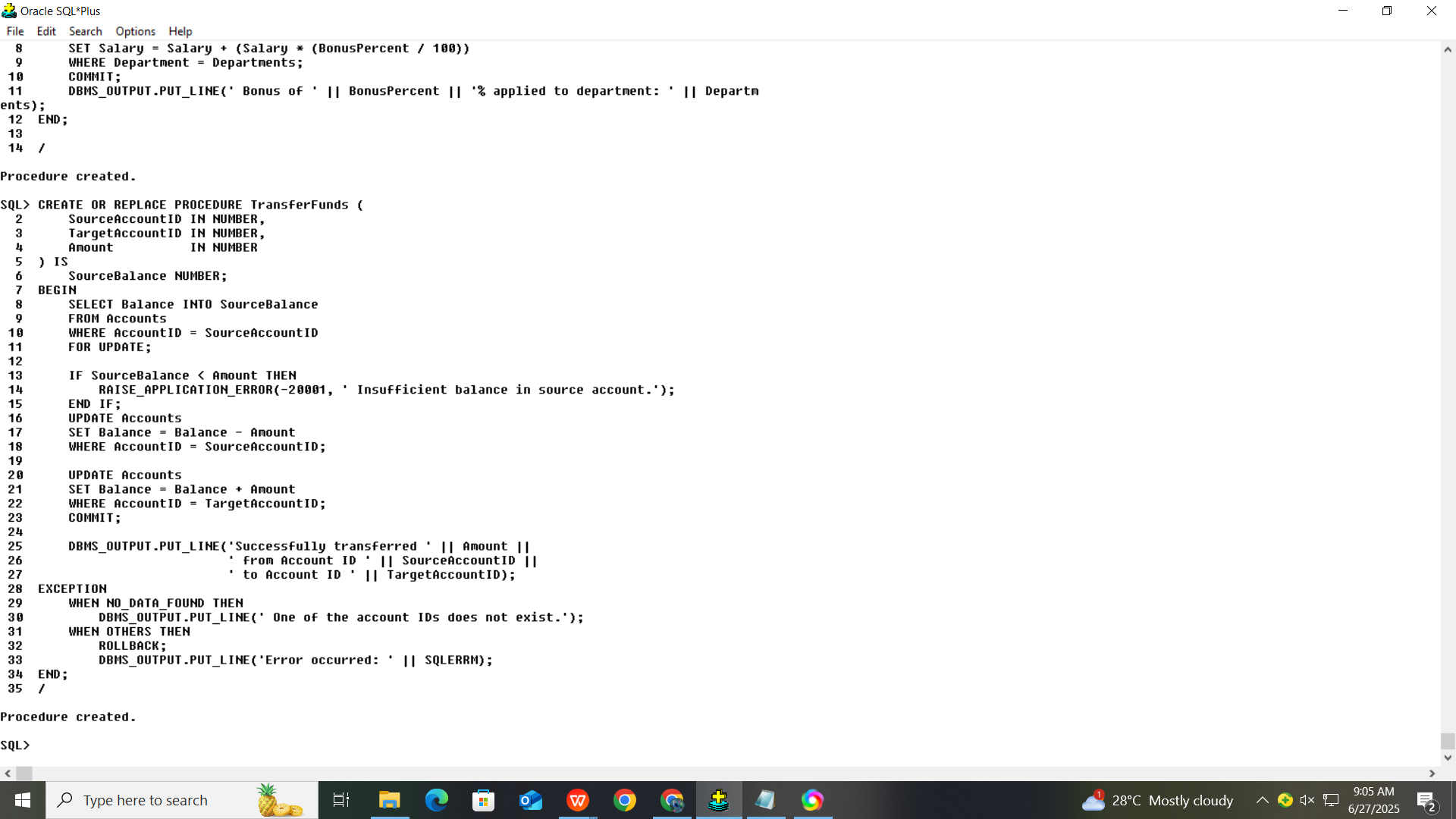
WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error occurred: ' || SQLERRM);

END;

**Output:**



**Junit Testing Excerises**

**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>

1. Create a new test class in your project.

**Solution Code:**

package com;

import org.junit.Test;

import static org.junit.Assert.\*;

public class SampleTest {

@Test

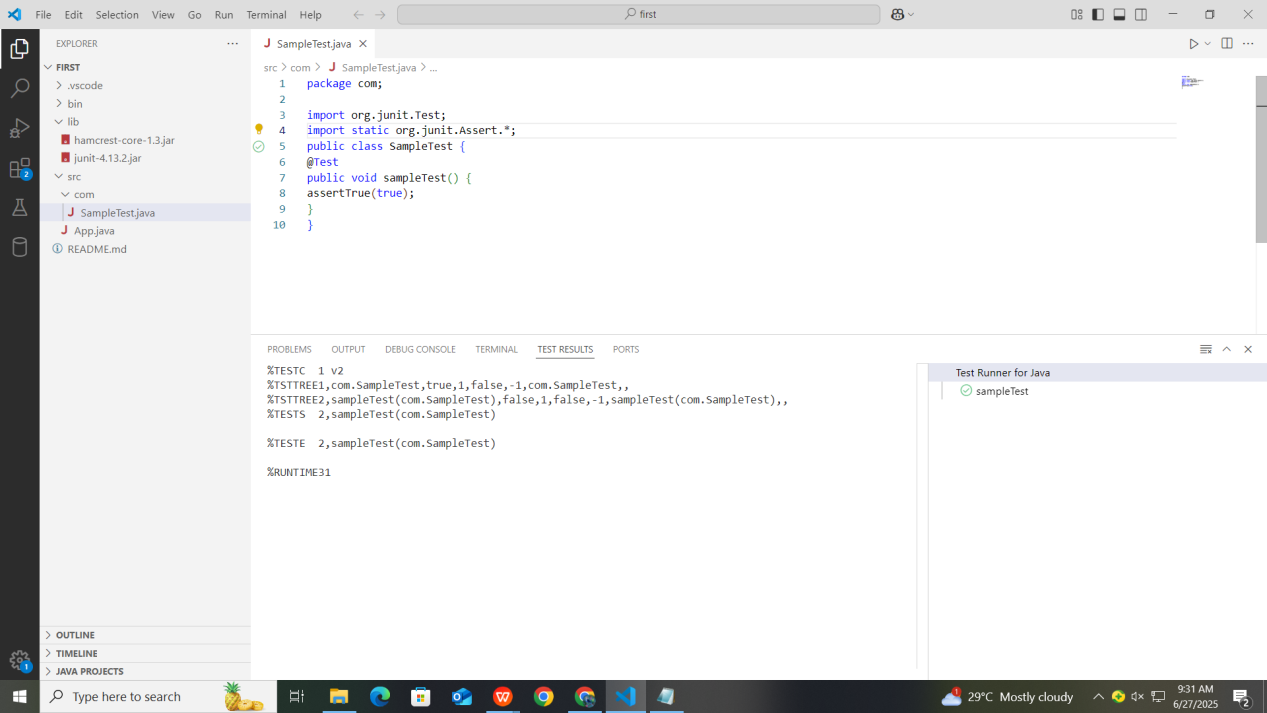
public void sampleTest() {

assertTrue(true);

}

}

**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:**

package com;

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);

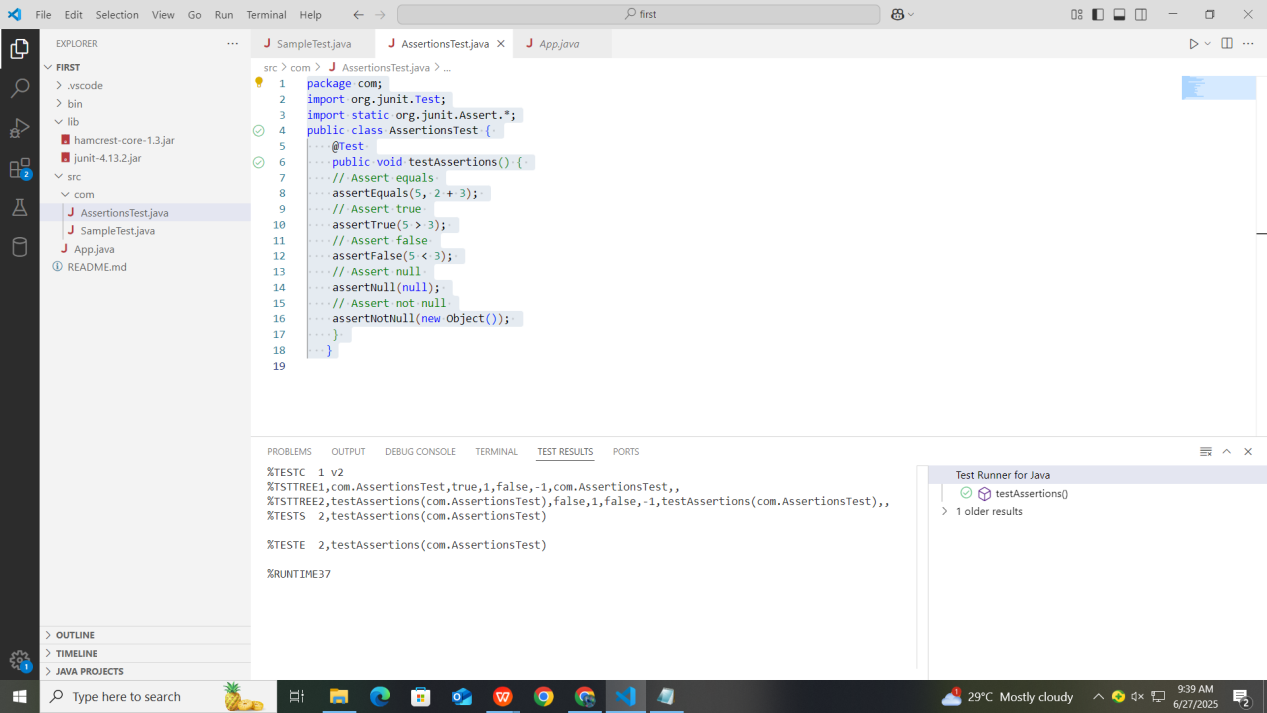
    // Assert not null

    assertNotNull(new Object());

    }

   }

**Output:**



**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 method

**Solution Code:**

**BankAccount.java**

package com;

public class BankAccount {

    private int balance;

    public BankAccount(int initialBalance) {

        this.balance = initialBalance;

    }

    public void deposit(int amount) {

        balance += amount;

    }

    public void withdraw(int amount) {

        balance -= amount;

    }

    public int getBalance() {

        return balance;

    }

}

**BankAccountTest.java**

package com;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class BankAccountTest {

    private BankAccount account;

    @Before

    public void setUp() {

                account = new BankAccount(1000);

    }

     @After

    public void tearDown() {

        account = null;

    }

    @Test

    public void testWithdraw() {

        account.withdraw(300);

        assertEquals(700, account.getBalance());

    }

    @Test

    public void testDeposit()

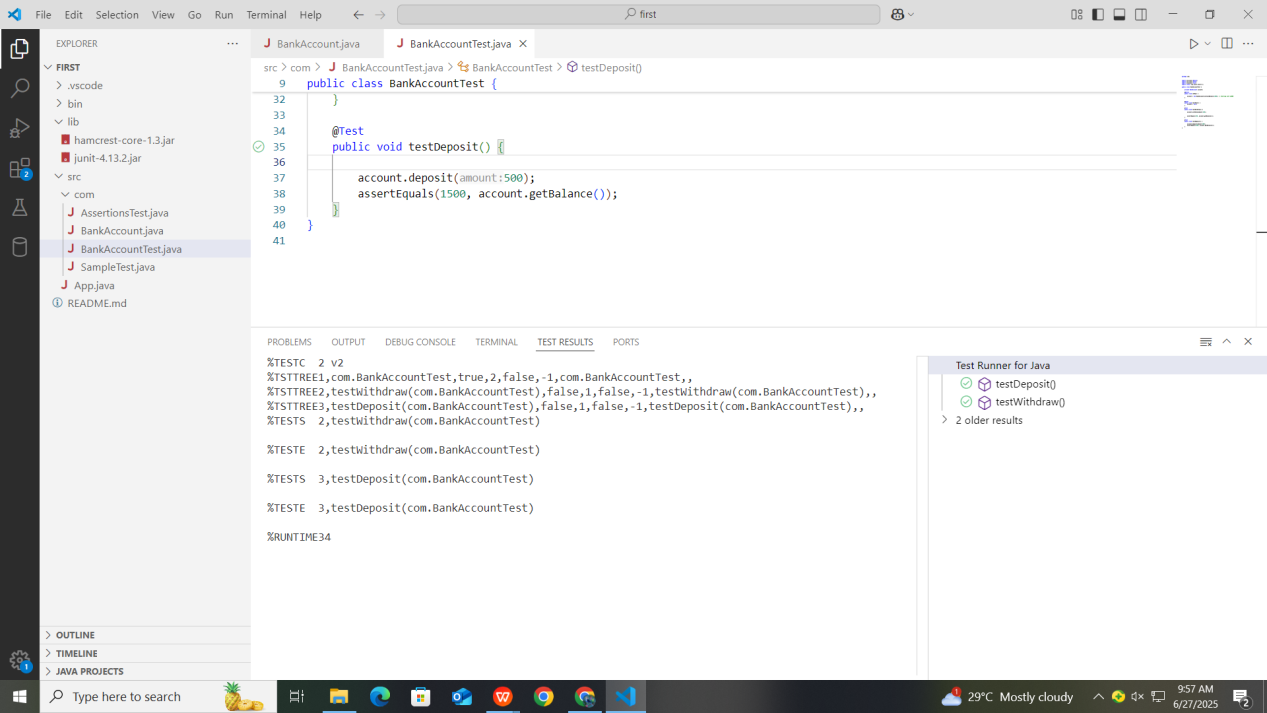
        account.deposit(500);

        assertEquals(1500, account.getBalance());

    }

}

**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);

}

}

**Solution Code:**

package com.example;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

class ExternalApi {

public String getData() {

return "Real Data";

}

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();}

}

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:**

**A computer screen shot of a program

AI-generated content may be incorrect.**

**Exercise 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();

}

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Logging using SLF4J**

**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");

}

}

**Solution Code:**package demo.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:**

**A screenshot of a computer

AI-generated content may be incorrect.**