**PL/SQL Exercises**

**Exercise 1: Control Structures**

For table creation :

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5),

InterestRate NUMBER(5,3)

);

-- Insert sample data

INSERT INTO Customers VALUES (1, 'Alice', 45, 8000, 'FALSE', 0.075);

INSERT INTO Customers VALUES (2, 'Bob', 63, 12000, 'FALSE', 0.080);

INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'FALSE', 0.085);

INSERT INTO Customers VALUES (4, 'David', 58, 9500, 'FALSE', 0.078);

INSERT INTO Customers VALUES (5, 'Eve', 66, 11000, 'FALSE', 0.082);

COMMIT;

1) code:

BEGIN

FOR cust IN (

SELECT CustomerID, Age, InterestRate

FROM Customers

WHERE Age > 60

) LOOP

UPDATE Customers

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = cust.CustomerID;

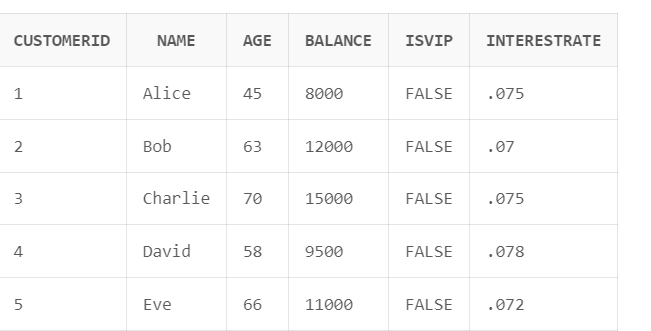
END LOOP;

COMMIT;

END;

/

SELECT \* FROM Customers;



2) code:

BEGIN

FOR cust IN (

SELECT CustomerID, Balance

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = cust.CustomerID;

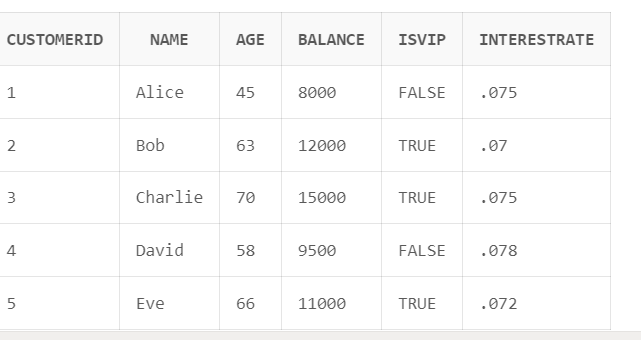
END LOOP;

COMMIT;

END;

/

SELECT \* FROM Customers;



3) code for table creation:

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

IsVIP VARCHAR2(5),

InterestRate NUMBER(5,3)

);

-- Create Loans table

CREATE TABLE Loans (

LoanID NUMBER PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Insert sample customers

BEGIN

INSERT INTO Customers VALUES (1, 'Alice', 45, 8000, 'FALSE', 0.075);

INSERT INTO Customers VALUES (2, 'Bob', 63, 12000, 'FALSE', 0.080);

INSERT INTO Customers VALUES (3, 'Charlie', 70, 15000, 'FALSE', 0.085);

INSERT INTO Customers VALUES (4, 'David', 58, 9500, 'FALSE', 0.078);

INSERT INTO Customers VALUES (5, 'Eve', 66, 11000, 'FALSE', 0.082);

COMMIT;

END;

/

-- Insert sample loans

BEGIN

INSERT INTO Loans VALUES (101, 1, SYSDATE + 10); -- due soon

INSERT INTO Loans VALUES (102, 2, SYSDATE + 40); -- not due soon

INSERT INTO Loans VALUES (103, 3, SYSDATE + 25); -- due soon

INSERT INTO Loans VALUES (104, 4, SYSDATE + 5); -- due soon

INSERT INTO Loans VALUES (105, 5, SYSDATE - 3); -- overdue

COMMIT;

END;

/

PL/SQL code:

BEGIN

FOR rec IN (

SELECT LoanID, CustomerID, DueDate

FROM Loans

WHERE DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ID ' || rec.CustomerID ||

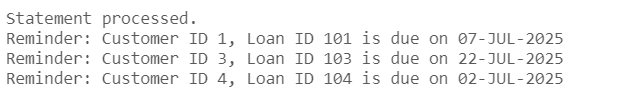
', Loan ID ' || rec.LoanID ||

' is due on ' || TO\_CHAR(rec.DueDate, 'DD-MON-YYYY'));

END LOOP;

END;

/



**Exercise 3: Stored Procedures**

1) code:

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Insert sample accounts

BEGIN

INSERT INTO Accounts VALUES (1, 101, 1000);

INSERT INTO Accounts VALUES (2, 102, 2000);

INSERT INTO Accounts VALUES (3, 103, 5000);

INSERT INTO Accounts VALUES (4, 104, 800);

INSERT INTO Accounts VALUES (5, 105, 12000);

COMMIT;

END;

/

-- Create procedure to apply 1% interest

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

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

COMMIT;

END;

/

-- Call the procedure

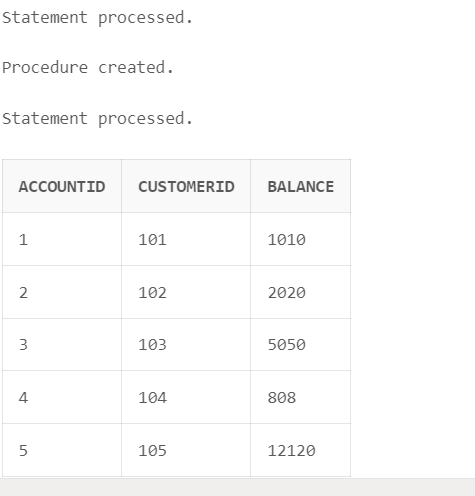
BEGIN

ProcessMonthlyInterest;

END;

/

SELECT \* FROM Accounts;



2) code:

-- Drop and recreate Employees table

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

EmpID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER

);

-- Insert sample employees

BEGIN

INSERT INTO Employees VALUES (1, 'Alice', 'HR', 30000);

INSERT INTO Employees VALUES (2, 'Bob', 'Sales', 40000);

INSERT INTO Employees VALUES (3, 'Charlie', 'Sales', 45000);

INSERT INTO Employees VALUES (4, 'David', 'IT', 50000);

INSERT INTO Employees VALUES (5, 'Eve', 'HR', 35000);

COMMIT;

END;

/

-- Create procedure to add bonus to department

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_dept IN VARCHAR2,

p\_bonus IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = p\_dept;

COMMIT;

END;

/

-- Call the procedure with 10% bonus for Sales

BEGIN

UpdateEmployeeBonus('Sales', 10);

END;

/

SELECT \* FROM Employees;



3) code:

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

Balance NUMBER

);

-- Insert sample data

BEGIN

INSERT INTO Accounts VALUES (1, 101, 1000);

INSERT INTO Accounts VALUES (2, 102, 2000);

INSERT INTO Accounts VALUES (3, 103, 5000);

INSERT INTO Accounts VALUES (4, 104, 800);

INSERT INTO Accounts VALUES (5, 105, 12000);

COMMIT;

END;

/

-- Create TransferFunds procedure

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from IN NUMBER,

p\_to IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Fetch source account balance

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from;

-- Check sufficient funds

IF v\_balance < p\_amount THEN

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

END IF;

-- Deduct from source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from;

-- Add to destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to;

COMMIT;

END;

/

-- Call the procedure: Transfer ₹2000 from Account 5 to Account 1

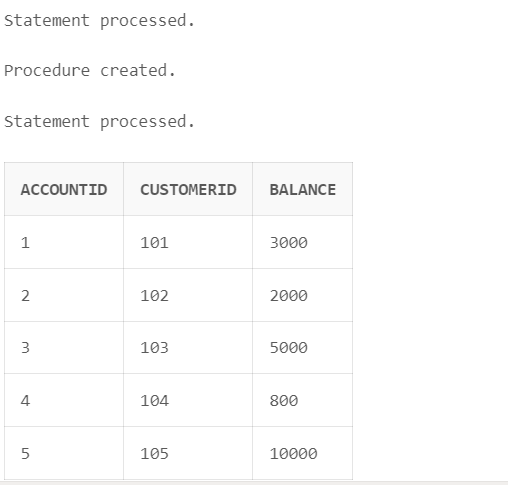
BEGIN

TransferFunds(5, 1, 2000);

END;

/

SELECT \* FROM BankAccounts;



**JUnit\_Basic Testing Exercises**

**Exercise 1: Setting Up Junit**

Done

**Exercise 3: Assertions in Junit**

Code:

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

    @Test

    public void testAssertions() {

        assertEquals(5, 2 + 3);         // Equal values

        assertTrue(5 > 3);              // True condition

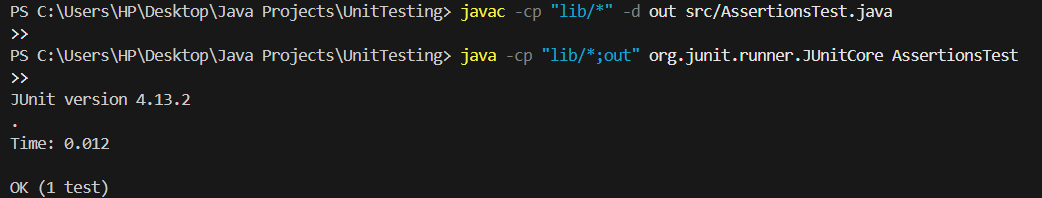
        assertFalse(5 < 3);             // False condition

        assertNull(null);              // Null check

        assertNotNull(new Object());    // Not null

    }

}



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

Code:

public class Calculator {

    public int add(int a, int b) {

        return a + b;

    }

    public int subtract(int a, int b) {

        return a - b;

    }

}

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

    private Calculator calc;

    @Before

    public void setUp() {

        calc = new Calculator();  // Arrange

        System.out.println("Setting up before each test");

    }

    @After

    public void tearDown() {

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

    }

    @Test

    public void testAdd() {

        int result = calc.add(10, 5);  // Act

        assertEquals(15, result);      // Assert

    }

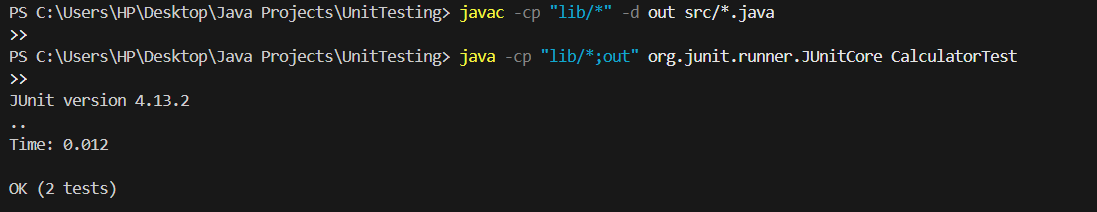
    @Test

    public void testSubtract() {

        int result = calc.subtract(10, 5);  // Act

        assertEquals(5, result);            // Assert

    }



**3. Mockito exercises**

**Exercise 1: Mocking and Stubbing**

Code:

public interface ExternalApi {

String getData();

}

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testExternalApi() {

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

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

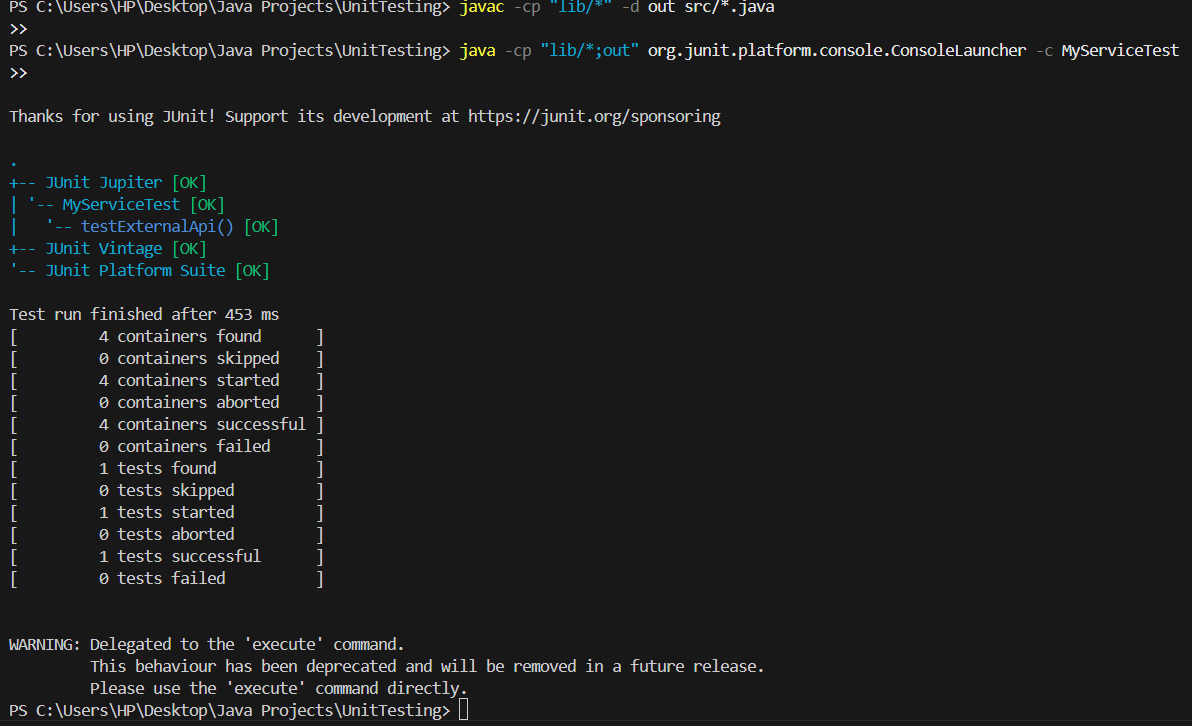
MyService service = new MyService(mockApi);

String result = service.fetchData();

assertEquals("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions**

Code:

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

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);

    }

    @Test

public void testVerifyInteraction() {

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

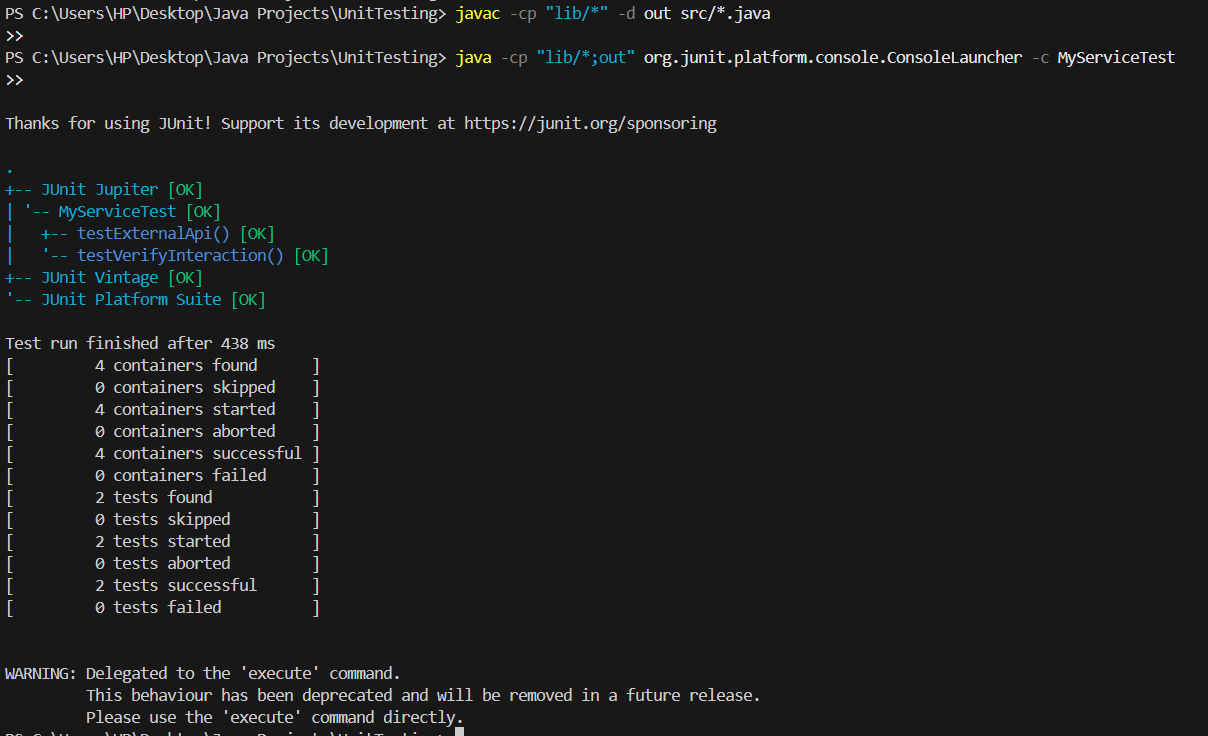
    MyService service = new MyService(mockApi);

    service.fetchData();

    verify(mockApi).getData(); // confirms getData() was called

}

}



**6. SL4J Logging exercises**

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

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");

    }

}

