**Mandatory hands-on**

**PLSQL\_Exercises**

**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.

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.

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.

**Scenario 1: Code**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance DECIMAL(10,2),

LastModified DATE

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10,2),

InterestRate DECIMAL(5,2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

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

(1, 'Harinee', '1949-03-27', 12000, CURRENT\_DATE),

(2, 'Kavi', '1975-03-18', 1500, CURRENT\_DATE);

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

(1, 1, 5000, 5.00, CURRENT\_DATE, DATE\_ADD(CURRENT\_DATE, INTERVAL 12 MONTH)),

(2, 2, 3000, 4.50, CURRENT\_DATE, DATE\_ADD(CURRENT\_DATE, INTERVAL 12 MONTH));

UPDATE Loans

SET InterestRate = InterestRate - (InterestRate \* 0.01)

WHERE CustomerID IN (

SELECT CustomerID

FROM Customers

WHERE TIMESTAMPDIFF(YEAR, DOB, CURDATE()) > 60

);

SELECT

l.LoanID,

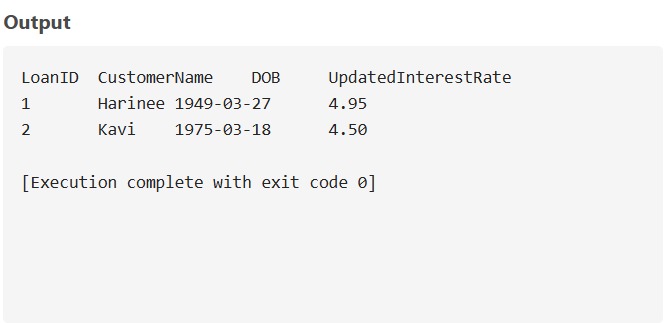
c.Name AS CustomerName,

c.DOB,

l.InterestRate AS UpdatedInterestRate

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID;



**Exercise 1: Control Structures**

**Scenario 2 Code**

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE,

Balance DECIMAL(10,2),

LastModified DATE,

IsVIP CHAR(5) DEFAULT 'FALSE'

);

INSERT INTO Customers VALUES

(1, 'Ananthi', '1975-11-04', 9000, CURRENT\_DATE, 'FALSE'),

(2, 'Balu', '1980-02-27', 18000, CURRENT\_DATE, 'FALSE'),

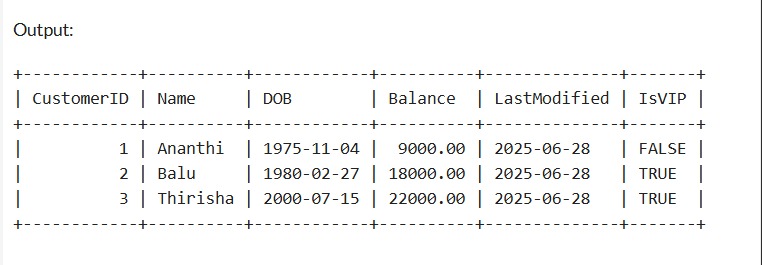
(3, 'Thirisha', '2000-07-15', 22000, CURRENT\_DATE, 'FALSE');

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE Balance > 10000;

SELECT \* FROM Customers;



**Exercise 1: Control Structures**

**Scenario 3 Code**

DROP TABLE IF EXISTS Loans;

DROP TABLE IF EXISTS Customers;

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Name VARCHAR(100),

DOB DATE

);

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

LoanAmount DECIMAL(10,2),

InterestRate DECIMAL(5,2),

StartDate DATE,

EndDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES

(1, 'Nathu', '1975-11-04'),

(2, 'Bala', '1980-02-27'),

(3, 'Navya', '2000-07-15');

INSERT INTO Loans VALUES

(1, 1, 5000, 5, '2022-01-01', DATE\_ADD(CURDATE(), INTERVAL 10 DAY)),

(2, 2, 10000, 4.5, '2023-01-01', DATE\_ADD(CURDATE(), INTERVAL 40 DAY)),

(3, 3, 15000, 6, '2021-01-01', DATE\_ADD(CURDATE(), INTERVAL 5 DAY));

SELECT

c.Name AS CustomerName,

l.LoanID,

l.LoanAmount,

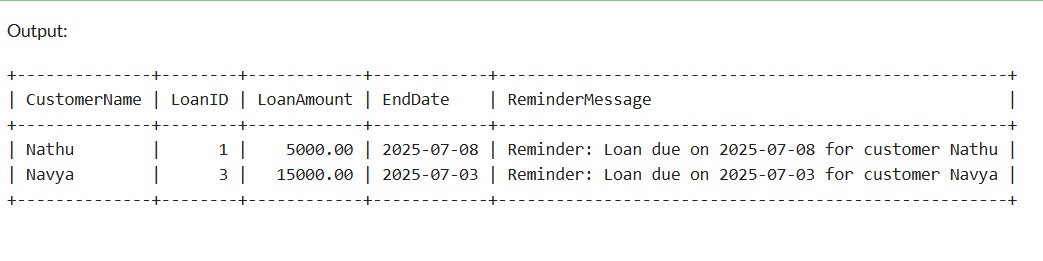
l.EndDate,

CONCAT('Reminder: Loan due on ', l.EndDate, ' for customer ', c.Name) AS ReminderMessage

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate BETWEEN CURDATE() AND DATE\_ADD(CURDATE(), INTERVAL 30 DAY);



**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.

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.

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.

**Exercise 3: Stored procedures**

**Scenario 1 Code**

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

AccountType TEXT,

Balance REAL,

LastModified DATE

);

INSERT INTO Accounts VALUES

(1, 101, 'Savings', 2000.00, CURRENT\_DATE),

(2, 102, 'Savings', 5000.00, CURRENT\_DATE),

(3, 103, 'Checking', 1500.00, CURRENT\_DATE);

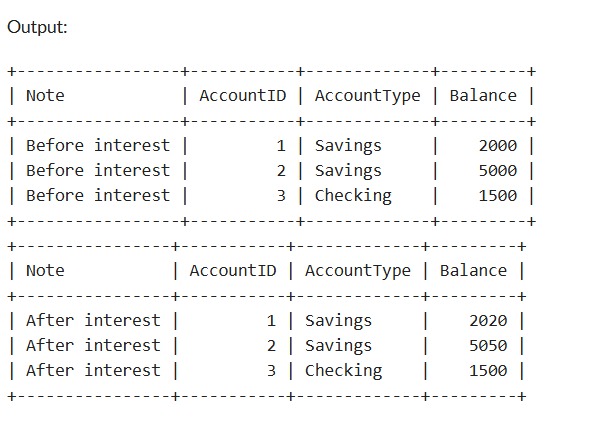
SELECT 'Before interest' AS Note, AccountID, AccountType, Balance FROM Accounts;

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'Savings';

SELECT 'After interest' AS Note, AccountID, AccountType, Balance FROM Accounts;



**Exercise 3: Stored procedures**

**Scenario 2 Code**

DROP TABLE IF EXISTS Employees;

CREATE TABLE Employees (

EmployeeID INTEGER PRIMARY KEY,

Name TEXT,

Position TEXT,

Salary REAL,

Department TEXT,

HireDate DATE

);

INSERT INTO Employees VALUES

(1, 'Nive', 'Manager', 70000, 'HR', '2015-06-15'),

(2, 'Afrena', 'Developer', 60000, 'IT', '2017-03-20'),

(3, 'Thirisha', 'Developer', 62000, 'IT', '2018-08-10');

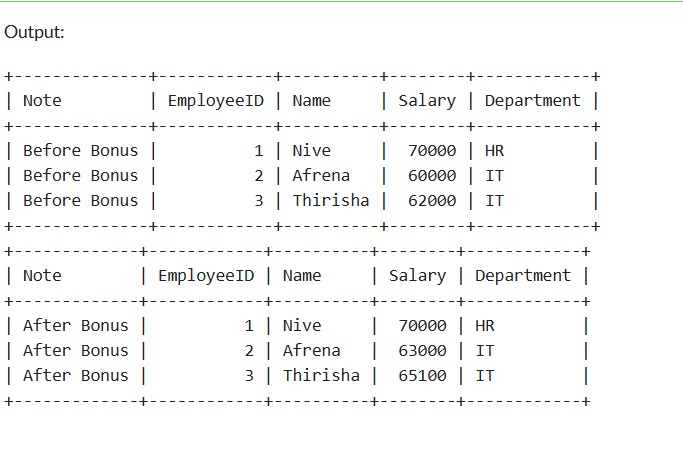
SELECT 'Before Bonus' AS Note, EmployeeID, Name, Salary, Department FROM Employees;

UPDATE Employees

SET Salary = Salary + (Salary \* 5.0 / 100)

WHERE Department = 'IT';

SELECT 'After Bonus' AS Note, EmployeeID, Name, Salary, Department FROM Employees;



**Exercise 3: Stored procedures**

**Scenario 3 Code**

DROP TABLE IF EXISTS Accounts;

CREATE TABLE Accounts (

AccountID INTEGER PRIMARY KEY,

CustomerID INTEGER,

AccountType TEXT,

Balance REAL,

LastModified DATE

);

INSERT INTO Accounts VALUES

(1, 101, 'Savings', 2000.00, CURDATE()),

(2, 101, 'Checking', 500.00, CURDATE());

SELECT 'Before Transfer' AS Note, AccountID, Balance FROM Accounts;

START TRANSACTION;

SELECT Balance FROM Accounts WHERE AccountID = 1;

UPDATE Accounts

SET Balance = Balance - 300

WHERE AccountID = 1 AND Balance >= 300;

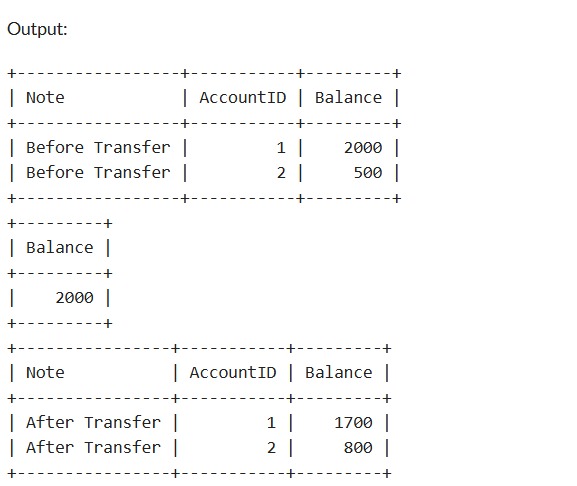
UPDATE Accounts

SET Balance = Balance + 300

WHERE AccountID = 2;

COMMIT;

SELECT 'After Transfer' AS Note, AccountID, Balance FROM Accounts;



**Exercise 1: Setting Up JUnit**

import org.junit.Test;

import static org.junit.Assert.\*;

public class AssertionsTest {

@Test

public void testAssertions()

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

Object nullObject = null;

assertNull(nullObject);

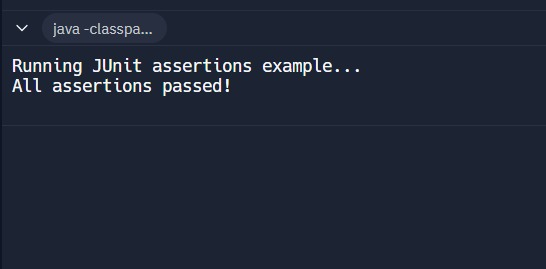
Object notNullObject = new Object();

assertNotNull(notNullObject);

}

}

**OUTPUT**



**Exercise 3: Assertions in Junit**

**Code :**

import org.junit.jupiter.api.Test;

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

public class Main {

public static void main(String[] args) {

System.out.println("Running JUnit tests...");

AssertionsTest test = new AssertionsTest();

try {

test.testAssertions();

System.out.println("All assertions passed!");

} catch (AssertionError e) {

System.out.println("Assertion failed: " + e.getMessage());

} catch (Exception e) {

System.out.println("Test failed with exception: " + e.getMessage());

}

}

}

class AssertionsTest {

@Test

void testAssertions() {

assertEquals(5, 2 + 3);

System.out.println("assertEquals(5, 2 + 3) - PASSED");

assertTrue(5 > 3);

System.out.println("assertTrue(5 > 3) - PASSED");

assertFalse(5 < 3);

System.out.println("assertFalse(5 < 3) - PASSED");

assertNull(null);

System.out.println("assertNull(null) - PASSED");

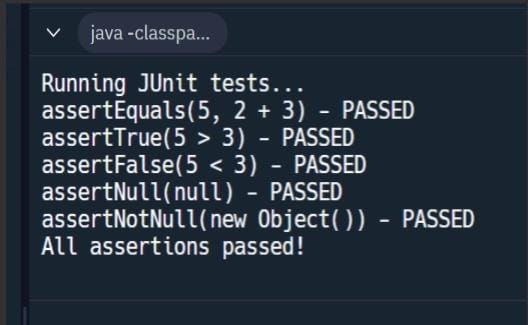
assertNotNull(new Object());

System.out.println("assertNotNull(new Object()) - PASSED");

}

}

**OUTPUT:**



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

**Code:**

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

// Class to test

class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

System.out.println("Setting up test fixture...");

calculator = new Calculator();

}

@After

public void tearDown() {

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

calculator = null;

}

@Test

public void testAddition() {

// Arrange

int a = 5;

int b = 3;

// Act

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

// Assert

assertEquals(8, result);

}

@Test

public void testSubtraction() {

// Arrange

int a = 10;

int b = 4;

// Act

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

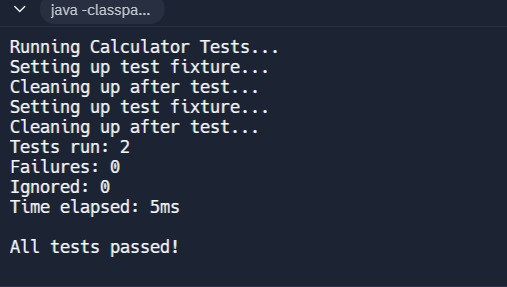
// Assert

assertEquals(6, result);

}

}

**OUTPUT :**



**Mockito Exercises**

**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.

**Code:**

Main.java :

public class Main {

public static void main(String[] args) {

System.out.println("Application started successfully!");

ExternalApi api = new ExternalApi() {

public String getData() {

return "Real API Data";

}

};

MyService service = new MyService(api);

String result = service.fetchData();

System.out.println("Fetched data: " + result);

}

}

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.junit.jupiter.api.Assertions.assertEquals;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

public void testExternalApi() {

ExternalApi mockApi = mock(ExternalApi.class);

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

MyService service = new MyService(mockApi);

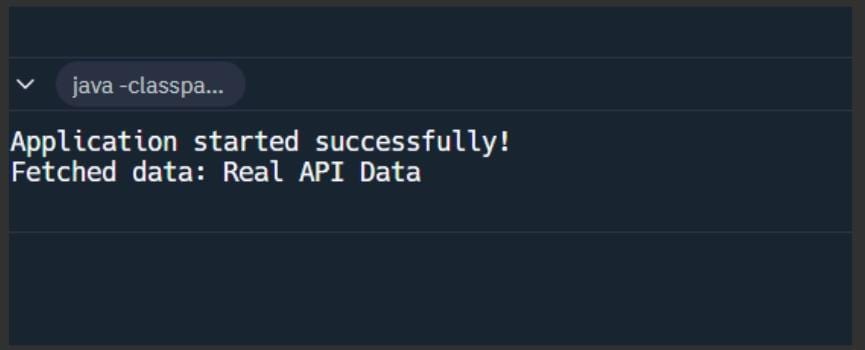
String result = service.fetchData();

assertEquals("Mock Data", result);

}

}

**OUTPUT:**



**Exercise 2: Verifying Interactions**

**Code:**

ExternalApi.java :

public interface ExternalApi {

String getData();

}

MyService.java:

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public void fetchData() {

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 {

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

Main.java :

public class Main {

public static void main(String[] args) {

System.out.println("Running Java Application with Mockito Testing");

MyServiceTest test = new MyServiceTest();

try {

test.testVerifyInteraction();

System.out.println(" Test passed: Mock interaction verified successfully!");

} catch (Exception e) {

System.out.println(" Test failed: " + e.getMessage());

}

System.out.println("\nDemonstrating service with real implementation:");

ExternalApi realApi = new ExternalApi() {

public String getData() {

return "Real data from external API";

}

};

MyService service = new MyService(realApi);

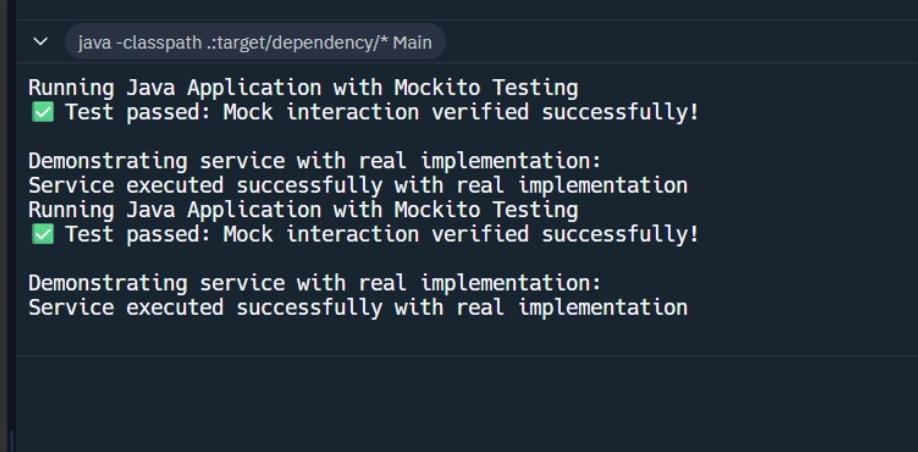
service.fetchData();

System.out.println("Service executed successfully with real implementation");

}

}

**OUTPUT:**



**SLF4J Exercise**

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

**Code**

Main.java :

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Main {

private static final Logger logger = LoggerFactory.getLogger(Main.class);

public static void main(String[] args) {

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

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

}

}

**OUTPUT:**

