WEEK 2

PL/SQL programming

*Exercise 1: Control Structures*

**Scenario 1: Apply Interest Discount for Customers Aged Above 60**

This PL/SQL block loops through all customers in the database. If a customer's age is greater than 60, it applies a 1% discount to their current loan interest rate.

**Code for Scenario 1:**

**Table Creation: Customers**

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Age INT,

Balance DECIMAL(10,2),

InterestRate DECIMAL(5,2),

IsVIP VARCHAR(5)

);

**Sample Data Insertion**

INSERT INTO Customers VALUES

(1, 65, 8000.00, 7.5, 'FALSE'),

(2, 58, 12000.00, 6.0, 'FALSE'),

(3, 70, 20000.00, 8.2, 'FALSE'),

(4, 45, 3000.00, 6.9, 'FALSE');

**Stored Procedure: ApplySeniorDiscount**

DELIMITER //

CREATE PROCEDURE ApplySeniorDiscount()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE c\_id INT;

DECLARE c\_age INT;

DECLARE cur CURSOR FOR SELECT CustomerID, Age FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO c\_id, c\_age;

IF done THEN

LEAVE read\_loop;

END IF;

IF c\_age > 60 THEN

UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE CustomerID = c\_id;

END IF;

END LOOP;

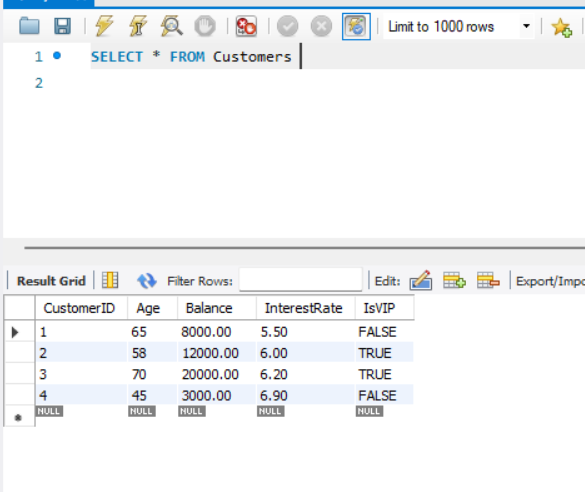
CLOSE cur;

END //

DELIMITER ;

**Output for Scenario 1:**

Interest rate successfully updated by 1% for customers aged above 60.



*Exercise 1: Control Structures*

**Scenario 2: Promote to VIP Based on Balance Over $10,000**

This PL/SQL procedure loops through all customers using a cursor. If a customer's balance is greater than $10,000, their IsVIP flag is set to TRUE. This logic uses conditional checking and updates customer status accordingly.

**Code for Scenario 2:**

**Table Used: Customers**

**Stored Procedure: PromoteVIP**

DELIMITER //

CREATE PROCEDURE PromoteVIP()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE c\_id INT;

DECLARE c\_balance DECIMAL(10,2);

DECLARE cur CURSOR FOR SELECT CustomerID, Balance FROM Customers;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO c\_id, c\_balance;

IF done THEN

LEAVE read\_loop;

END IF;

IF c\_balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = c\_id;

END IF;

END LOOP;

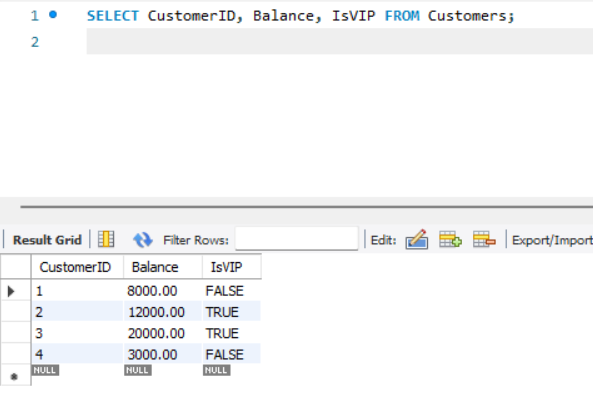
CLOSE cur;

END //

DELIMITER ;

**Output for Scenario 2:**

IsVIP flag successfully updated to TRUE for all customers with balance over $10,000.



*Exercise 1: Control Structures*

**Scenario 3: Send Loan Reminders for Loans Due Within the Next 30 Days**

This PL/SQL procedure checks for loans that are due within the next 30 days and sends reminder messages for each eligible customer. It uses a cursor to iterate through loan records from a Loans table and prints reminder messages using a SELECT statement. This logic applies date-based filtering and demonstrates effective use of loops and conditions.

**Code for Scenario 3:**

**Table Creation: Loans**

CREATE TABLE Loans (

LoanID INT PRIMARY KEY,

CustomerID INT,

DueDate DATE

);

**Sample Data Insertion**

INSERT INTO Loans VALUES

(101, 1, CURDATE() + INTERVAL 10 DAY),

(102, 2, CURDATE() + INTERVAL 40 DAY),

(103, 3, CURDATE() + INTERVAL 25 DAY),

(104, 4, CURDATE() - INTERVAL 5 DAY);

**Stored Procedure: SendLoanReminders**

DELIMITER //

CREATE PROCEDURE SendLoanReminders()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE c\_id INT;

DECLARE due\_date DATE;

DECLARE cur CURSOR FOR

SELECT CustomerID, DueDate FROM Loans

WHERE DueDate BETWEEN CURDATE() AND CURDATE() + INTERVAL 30 DAY;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN cur;

read\_loop: LOOP

FETCH cur INTO c\_id, due\_date;

IF done THEN

LEAVE read\_loop;

END IF;

SELECT CONCAT('Reminder: Customer ', c\_id,

' has a loan due on ', due\_date) AS ReminderMessage;

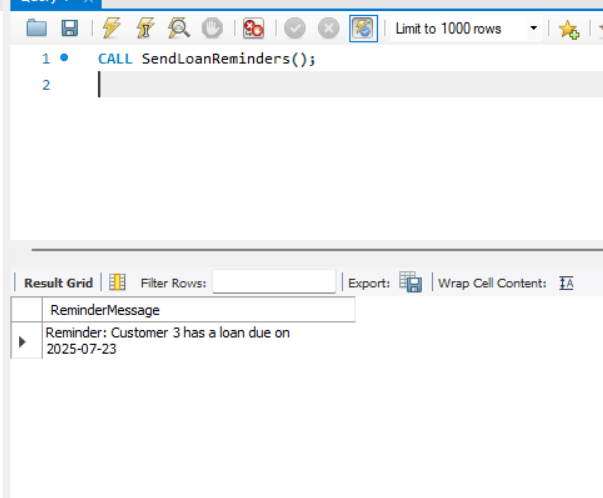
END LOOP;

CLOSE cur;

END //

DELIMITER ;

**Output for Scenario 3:**

****

TDD using JUnit5 and Mockito

1. JUnit\_Basic Testing Exercises

*Exercise 1: Setting up JUnit*

**Code for Exercise 1:**

**pom.xml Dependency**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

**Calculator.java**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

**CalculatorTest.java**

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

Calculator calc = new Calculator();

int result = calc.add(5, 3);

assertEquals(8, result);

}

}

*Exercise 3: Assertions in JUnit*

**Code for Exercise 3:**

**Java Code (AssertionsTest.java)**

package com.example;

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);

assertNull(null);

assertNotNull(new Object());

}

}

*Exercise 4: Arrange-Act-Assert (AAA) Pattern with Setup and Teardown*

**Code for Exercise 4 :**

**Java Code (AAATest.java)**

package com.example;

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class AAATest {

private Calculator calculator;

@Before

public void setUp() {

calculator = new Calculator();

}

@After

public void tearDown() {

calculator = null;

}

@Test

public void testAddition() {

int result = calculator.add(4, 2);

assertEquals(6, result);

}

@Test

public void testAdditionWithNegative() {

int result = calculator.add(-3, 5);

assertEquals(2, result);

}

}

TDD using JUnit5 and Mockito

3. Mockito exercises

*Exercise 1: Mocking and Stubbing*

**Code for Exercise 1:**

**Java Code (MyServiceTest.java)**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

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

public class MyServiceTest {

@Test

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);

}

**Supporting Class 1: ExternalApi.java**

package com.example;

public interface ExternalApi {

String getData();

}

**Supporting Class 2: MyService.java**

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

*Exercise 2: Verifying Junit5 and Mockito*

**Code for Exercise 2:**

**Java Code (MyServiceTest.java)**

package com.example;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

SLF4J logging framework

6. SL4J Logging exercises

*Exercise 1: Logging Error Messages and Warning Levels*

**Code for Exercise 1:**

**Maven Dependencies (pom.xml)**

<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>

**Java Code (LoggingExample.java)**

package com.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");

}

}