**WEEK 2 – ASSESSMENT**

**Exercise 1: Control Structures**

**Scenario 1:**

BEGIN

FOR customer IN (

SELECT CustomerID, InterestRate

FROM Customers

WHERE Age > 60

)

LOOP

-- Reduce interest rate by 1% for senior customers

UPDATE Customers

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer.CustomerID;

END LOOP;

COMMIT; -- Save changes

END**;**

**Output**

|  |  |  |
| --- | --- | --- |
| **CustomerID** | **Old InterestRate** | **New InterestRate** |
| 101 | 9.5 | 8.5 |
| 102 | 10.0 | 9.0 |

**Scenario 2:**

BEGIN

FOR customer IN (

SELECT CustomerID

FROM Customers

WHERE Balance > 10000

)

LOOP

-- Promote to VIP status if balance is high

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = customer.CustomerID;

END LOOP;

COMMIT; -- Finalize VIP updates

END;

**Output:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **CustomerID** |  |  | | --- | --- | --- |  |  |  |  | | --- | --- | --- | |  |  |  |  |  |  |  | | --- | --- | --- | |  |  |  | | **Balance** | **IsVIP** |
| 103 | 12000 | TRUE |
| 105 | 20050 | TRUE |

**Scenario 3:**

DECLARE

CURSOR loan\_due\_cursor IS

SELECT CustomerID, LoanID, DueDate

FROM Loans

WHERE DueDate <= SYSDATE + 30;

BEGIN

FOR loan IN loan\_due\_cursor LOOP

-- Send a reminder to each customer

DBMS\_OUTPUT.PUT\_LINE('Reminder: Dear Customer ' || loan.CustomerID ||

', your loan #' || loan.LoanID ||

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

'. Kindly ensure timely payment.');

END LOOP;

END;

**Output:**

Reminder: Dear Customer 201, your loan

Reminder: Dear Customer 202, your loan

Reminder: Dear Customer 203, your loan

**Exercise 2: Stored Procedures**

**Scenario 1:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

FOR acc IN (

SELECT AccountID, Balance

FROM Accounts

WHERE AccountType = ‘SAVINGS’

) LOOP

UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

**Output:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | **AccountID** | | --- | | | **Old Balance** | | --- | | | **New Balance** | | --- | |
| |  | | --- | | A001 |  |  | | --- | |  | | |  | | --- | | 10,000.00 | | |  | | --- | | 10,100.00 | |
| A002 | |  | | --- | | 25,000.00 | | |  | | --- | | 25,250.00 | |

**Scenario 2:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

dept\_id IN NUMBER,

bonus\_percent IN NUMBER

) AS

BEGIN

-- Increase salary by bonus percentage for employees in the given department

UPDATE Employees

SET Salary = Salary + (Salary \* bonus\_percent / 100)

WHERE DepartmentID = dept\_id;

COMMIT;

END;

**Output:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EmpID** | **DepartmentID** | **Old Salary** | **New Salary** |
| 201 | 101 | 50,000 | 55,000 |
| 202 | 102 | 60,000 | 66,000 |

**Scenario 3:**

CREATE OR REPLACE PROCEDURE TransferFunds (

From\_acc\_id IN VARCHAR2,

To\_acc\_id IN VARCHAR2,

Amount IN NUMBER

) AS

Insufficient\_balance EXCEPTION;

Src\_balance NUMBER;

BEGIN

SELECT Balance INTO src\_balanc

FROM Accounts

WHERE AccountID = from\_acc\_id

FOR UPDATE;

IF src\_balance < amount THEN

RAISE insufficient\_balance;

END IF;

UPDATE Accounts

SET Balance = Balance – amount

WHERE AccountID = from\_acc\_id;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = to\_acc\_id;

COMMIT;

EXCEPTION

WHEN insufficient\_balance THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE(‘Transfer failed: Insufficient balance in account ‘ || from\_acc\_id);

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE(‘Transfer failed due to an unexpected error: ‘ || SQLERRM);

END;

**Output:**

Funds successfully transferred from A001 to A002.

**Junit Testing Exercises**

**Exercise 1: Setting Up Junit**

1. **Using Eclipse:**

* Go to File → New → Java Project.
* Enter the project name.
* Click **Finish** to create the project.

1. **Add Junit Dependency**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

1. **Create a Java Class to Test**

Import org.junit.Test;

Import static org.junit.Assert.\*;

Public class CalculatorTest {

Public void testAdd() {

Calculator calculator = new Calculator();

Int result = calculator.add(3, 4);

assertEquals(7, result);

}

}

**Output:**

Tests run: 1, Failures: 0

Result: SUCCESS

**Exercise 2: Assertions in Junit**

Import org.junit.Test;

Import static org.junit.Assert.\*;

Public class AssertionsTest {

Public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(5 < 3);

assertNull(null);

assertNotNull(new Object());

}

}

**Output:**

Tests run: 1, Failures: 0

Result: SUCCESS

**Exercise 3: Arrange-Act-Assert (AAA) Pattern, Test Fixtures, Setup and Teardown Methods in Junit**

**Java Test Code Using AAA + Setup/Teardown**

Import org.junit.After;

Import org.junit.Before;

Import org.junit.Test;

Import static org.junit.Assert.\*;

Public class CalculatorTest {

Private Calculator calculator;

Public void setUp() {

System.out.println(“Setting up test environment...”);

Calculator = new Calculator();

}

Public void tearDown() {

System.out.println(“Cleaning up after test...”);

Calculator = null;

}

Public void testAddition() {

Int result = calculator.add(10, 5);

assertEquals(15, result);

}

Public void testSubtraction() {

Int result = calculator.subtract(10, 5);

assertEquals(5, result);

}

}

**Output:**

Setting up test environment...

Cleaning up after test...

Setting up test environment...

Cleaning up after test...

Tests run: 2, Failures: 0

Result: SUCCESS

**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

Import static org.mockito.Mockito.\*;

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

Import org.junit.jupiter.api.Test;

Import org.mockito.Mockito;

Interface ExternalApi {

String getData();

}

Class MyService {

Private ExternalApi api;

Public MyService(ExternalApi api) {

This.api = api;

}

Public String fetchData() {

Return api.getData();

}

}

Public class MyServiceTest {

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:**

Tests run: 1, Failures: 0

Result: SUCCESS

**Exercise 2: Verifying Interactions**

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

interface ExternalApi {

String getData();

}

class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

public class MyServiceTest {

public void testVerifyInteraction() {

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

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

**Output:**

Tests run: 1, Failures: 0

Result: SUCCESS

**Logging using SLF4J**

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

**Logback Dependencies**

**XML**

<dependencies>

<!—SLF4J Logging API 🡪

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<!—Logback – SLF4J’s default implementation -->

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

</dependency>

</dependencies>

**SLF4J 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”);

}

}

**Output:**

12:00:00.123 [main] ERROR LoggingExample – This is an error message

12:00:00.124 [main] WARN LoggingExample – This is a warning message