**WEEK 2 PLSQL EXCERCISES**

**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.

-- Enable output (required to see DBMS\_OUTPUT messages)

**CODE:**

-- Create tables

CREATE TABLE Customers (

CustomerID NUMBER,

LoanID NUMBER,

InterestRate NUMBER,

Age NUMBER

);

CREATE TABLE Loans (

LoanID NUMBER,

InterestRate NUMBER

);

-- Insert data

INSERT INTO Customers VALUES (101, 201, 10.5, 65);

INSERT INTO Customers VALUES (102, 202, 11.0, 58); -- not eligible

INSERT INTO Loans VALUES (201, 10.5);

INSERT INTO Loans VALUES (202, 11.0);

COMMIT;

BEGIN

FOR cust\_rec IN (

SELECT CustomerID, LoanID

FROM Customers

WHERE Age > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE LoanID = cust\_rec.LoanID;

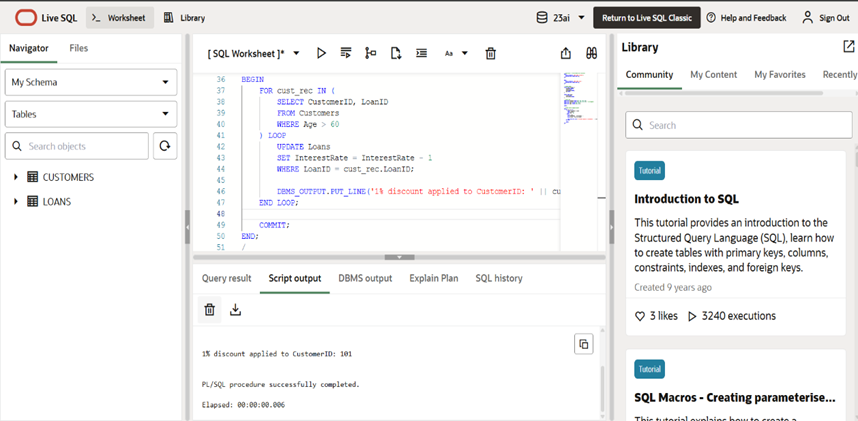
DBMS\_OUTPUT.PUT\_LINE('1% discount applied to CustomerID: ' || cust\_rec.CustomerID);

END LOOP;

COMMIT;

END;

/



**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.

**CODE:**

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

    CustomerID NUMBER,

    Balance NUMBER,

    IsVIP VARCHAR2(5)

);

INSERT INTO Customers VALUES (101, 15000, 'FALSE');

INSERT INTO Customers VALUES (102, 8000, 'FALSE');

INSERT INTO Customers VALUES (103, 20000, 'FALSE');

COMMIT;

BEGIN

    FOR vip\_rec IN (

        SELECT CustomerID, Balance

        FROM Customers

        WHERE Balance > 10000

    ) LOOP

        UPDATE Customers

        SET IsVIP = 'TRUE'

        WHERE CustomerID = vip\_rec.CustomerID;

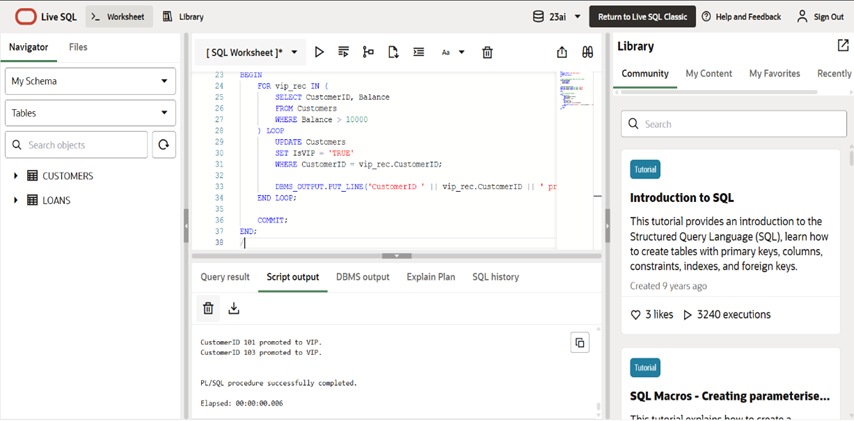
        DBMS\_OUTPUT.PUT\_LINE('CustomerID ' || vip\_rec.CustomerID || ' promoted to VIP.');

    END LOOP;

    COMMIT;

END;

/



**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.

**CODE:**

-- Drop the Loans table if it already exists

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

-- Create Loans table with correct columns

CREATE TABLE Loans (

    LoanID NUMBER,

    CustomerID NUMBER,

    DueDate DATE

);

-- Insert test data

INSERT INTO Loans VALUES (201, 101, SYSDATE + 10); -- Due soon

INSERT INTO Loans VALUES (202, 102, SYSDATE + 35); -- Too late

INSERT INTO Loans VALUES (203, 103, SYSDATE + 5);  -- Due soon

COMMIT;

-- Show reminders for loans due in next 30 days

BEGIN

    FOR due\_rec IN (

        SELECT LoanID, CustomerID, DueDate

        FROM Loans

        WHERE DueDate <= SYSDATE + 30

    ) LOOP

        DBMS\_OUTPUT.PUT\_LINE(

            'Reminder: Loan ' || due\_rec.LoanID ||

            ' for Customer ' || due\_rec.CustomerID ||

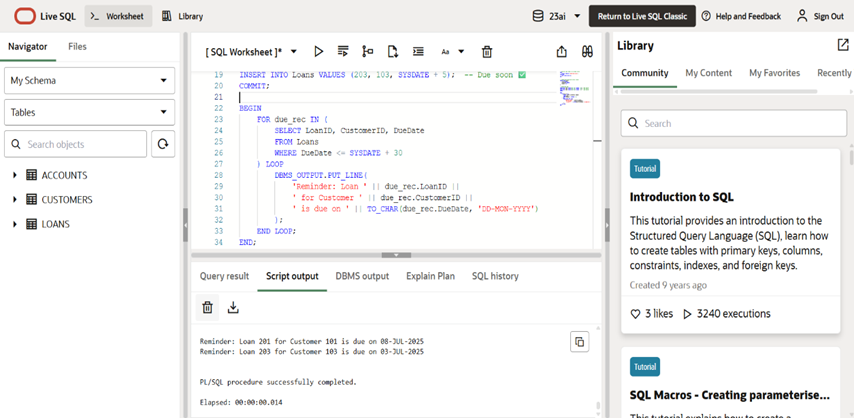
            ' is due on ' || TO\_CHAR(due\_rec.DueDate, 'DD-MON-YYYY')

        );

    END LOOP;

END;

/



**2. 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.

**CODE:**

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    Balance NUMBER,

    AccountType VARCHAR2(20)

);

INSERT INTO Accounts VALUES (101, 1000, 'SAVINGS');

INSERT INTO Accounts VALUES (102, 2000, 'CHECKING');

INSERT INTO Accounts VALUES (103, 3000, 'SAVINGS');

INSERT INTO Accounts VALUES (104, 3000, 'SAVINGS');

COMMIT;

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

    UPDATE Accounts

    SET Balance = Balance + (Balance \* 0.01)

    WHERE AccountType = 'SAVINGS';

    -- Optional: count updated rows

    DBMS\_OUTPUT.PUT\_LINE(SQL%ROWCOUNT || ' savings accounts updated.');

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

END;

/

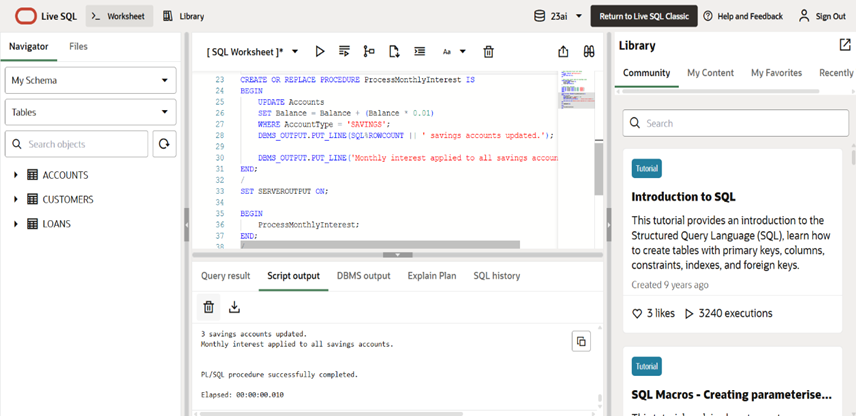
SET SERVEROUTPUT ON;

BEGIN

    ProcessMonthlyInterest;

END;

/



**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.

**CODE:**

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Employees (

    EmployeeID NUMBER PRIMARY KEY,

    Name VARCHAR2(50),

    Salary NUMBER,

    DepartmentID NUMBER

);

INSERT INTO Employees VALUES (1, 'Alice', 50000, 10);

INSERT INTO Employees VALUES (2, 'Bob', 60000, 20);

INSERT INTO Employees VALUES (3, 'Charlie', 55000, 10);

COMMIT;

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

    dept\_id IN NUMBER,

    bonus\_percent IN NUMBER

) IS

BEGIN

    UPDATE Employees

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

    WHERE DepartmentID = dept\_id;

    DBMS\_OUTPUT.PUT\_LINE('Bonus applied to department ID: ' || dept\_id);

END;

/

SET SERVEROUTPUT ON;

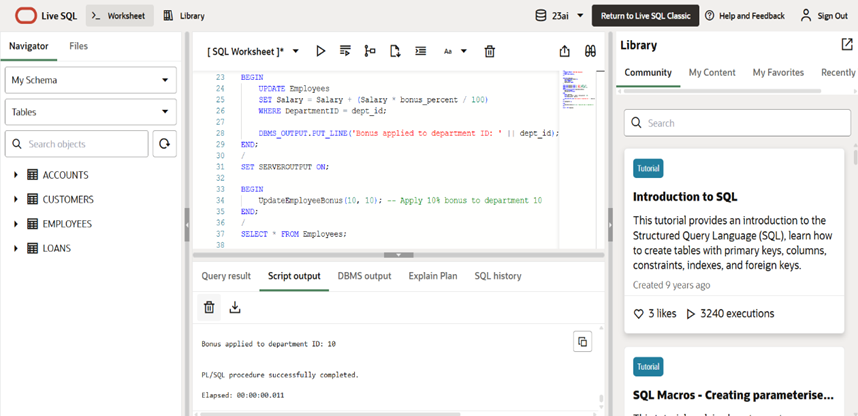
BEGIN

    UpdateEmployeeBonus(10, 10); -- Apply 10% bonus to department 10

END;

/

SELECT \* FROM Employees;



**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.

**CODE:**

BEGIN

    EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXCEPTION

    WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

    AccountID NUMBER PRIMARY KEY,

    Balance NUMBER

);

INSERT INTO Accounts VALUES (101, 5000);

INSERT INTO Accounts VALUES (102, 3000);

COMMIT;

CREATE OR REPLACE PROCEDURE TransferFunds(

    from\_account IN NUMBER,

    to\_account IN NUMBER,

    amount IN NUMBER

) IS

    insufficient\_funds EXCEPTION;

    v\_balance NUMBER;

BEGIN

    -- Lock and get balance from sender

    SELECT Balance INTO v\_balance

    FROM Accounts

    WHERE AccountID = from\_account

    FOR UPDATE;

    IF v\_balance < amount THEN

        RAISE insufficient\_funds;

    END IF;

    -- Deduct from sender

    UPDATE Accounts

    SET Balance = Balance - amount

    WHERE AccountID = from\_account;

    -- Add to receiver

    UPDATE Accounts

    SET Balance = Balance + amount

    WHERE AccountID = to\_account;

    COMMIT;

    DBMS\_OUTPUT.PUT\_LINE('Transferred ' || amount || ' from Account ' || from\_account || ' to Account ' || to\_account);

EXCEPTION

    WHEN insufficient\_funds THEN

        DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance in Account ' || from\_account);

        ROLLBACK;

    WHEN OTHERS THEN

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

        ROLLBACK;

END;

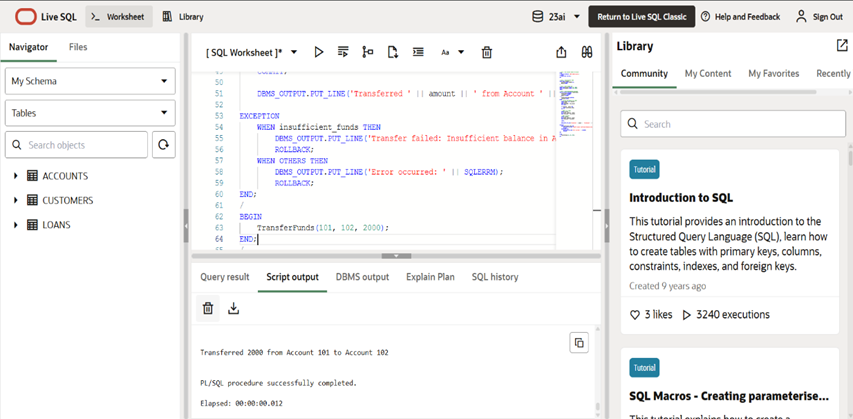
/

BEGIN

    TransferFunds(101, 102, 2000);

END;

/



**JUNIT BASIC TESTING 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:

3. Create a new test class in your project.

**CODE:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.assertEquals;

public class CalculatorTest {

@Test

public void testAdd() {

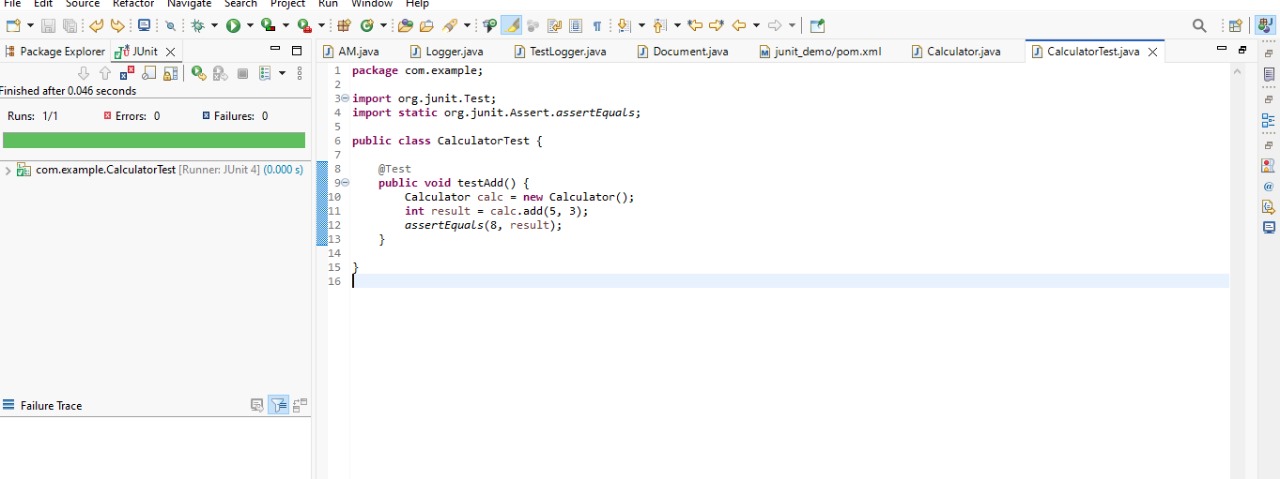
Calculator calc = new Calculator();

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

assertEquals(8, result);

}

}



2. 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.

**CODE:**

package com.example;

import static org.junit.Assert.\*;

import org.junit.Test;

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());

}

}

A computer screen shot of a computer screen

AI-generated content may be incorrect.

3. 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 methods.

**CODE:**

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public int subtract(int a, int b) {

return a - b;

}

}

package com.example;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Before;

import org.junit.Test;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

// Arrange: setup before each test

calculator = new Calculator();

System.out.println("Setup: Calculator created");

}

@After

public void tearDown() {

// Cleanup after each test

calculator = null;

System.out.println("Teardown: Calculator reset");

}

@Test

public void testAddition() {

// Arrange is already done in setUp()

// Act

int result = calculator.add(2, 3);

// Assert

assertEquals(5, result);

}

@Test

public void testSubtraction() {

// Act

int result = calculator.subtract(10, 4);

// Assert

assertEquals(6, result);

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**MOCKITO EXERCISES**

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.

**CODE:**

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

import static org.mockito.Mockito.\*;

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);

}

}

A computer screen shot of a computer code

AI-generated content may be incorrect.

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.

package com.example;

**CODE:**

import static org.mockito.Mockito.\*;

import org.junit.Test;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

// Verify that getData() was called once

verify(mockApi).getData();

}

}

package com.example;

public interface ExternalApi {

String getData();

}

package com.example;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**SL4J LOGGING EXERCISES**

1. Logging Error Messages and Warning Levels Task: Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

**CODE:**

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");

}

}

