**PL/SQL Exercise**

**Exercise 1: Control Structures**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Loans';

EXECUTE IMMEDIATE 'DROP TABLE Customers';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Customers (

CustomerID NUMBER PRIMARY KEY,

CustomerName VARCHAR2(100),

Age NUMBER,

Balance NUMBER,

LoanInterestRate NUMBER(5,2),

IsVIP VARCHAR2(5)

);

CREATE TABLE Loans (

LoanID VARCHAR2(10) PRIMARY KEY,

CustomerID NUMBER,

DueDate DATE,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

INSERT INTO Customers VALUES (101, 'John Doe', 65, 15000, 10.0, 'FALSE');

INSERT INTO Customers VALUES (102, 'Alice Ray', 45, 5000, 11.5, 'FALSE');

INSERT INTO Customers VALUES (103, 'Jane Smith', 70, 12000, 12.0, 'FALSE');

INSERT INTO Loans VALUES ('L001', 101, SYSDATE + 20);

INSERT INTO Loans VALUES ('L002', 102, SYSDATE + 40);

INSERT INTO Loans VALUES ('L003', 103, SYSDATE + 25);

COMMIT;

**Scenario 01**

BEGIN

FOR rec IN (SELECT CustomerID, Age FROM Customers) LOOP

IF rec.Age > 60 THEN

UPDATE Customers

SET LoanInterestRate = LoanInterestRate - 1

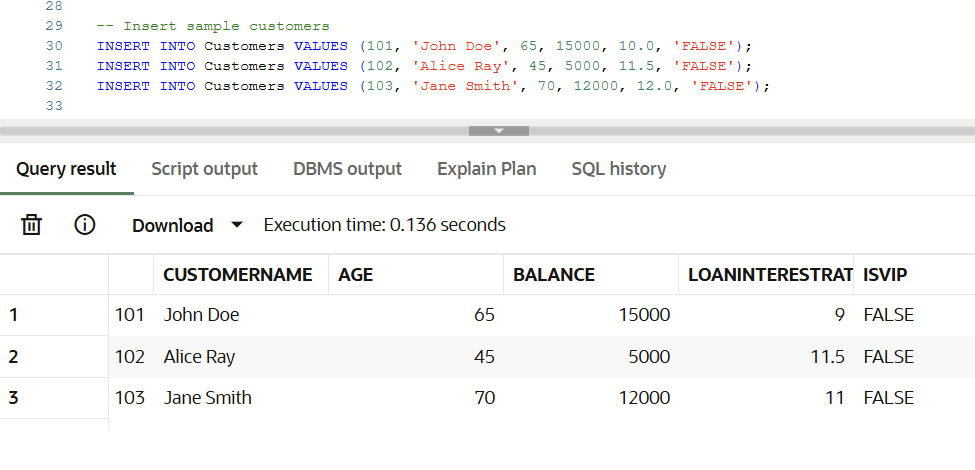
WHERE CustomerID = rec.CustomerID;

END IF;

END LOOP;

COMMIT;

END;



**Scenario 02**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers) LOOP

IF rec.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'TRUE'

WHERE CustomerID = rec.CustomerID;

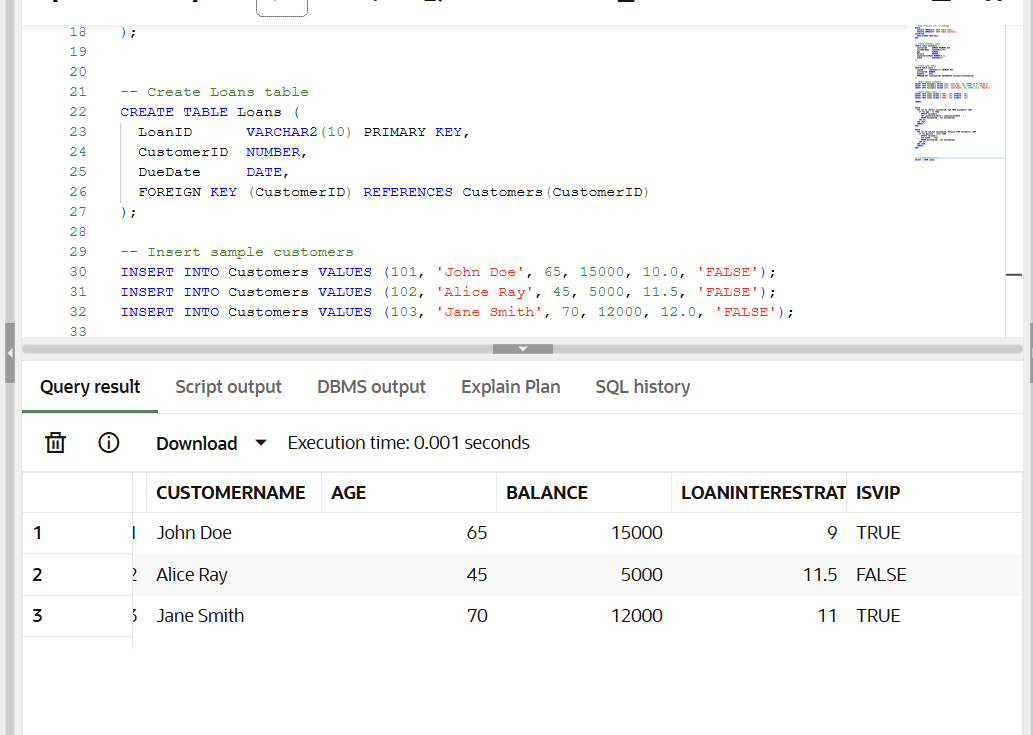
END IF;

END LOOP;

COMMIT;

END;

/



**Scenario 03**

BEGIN

FOR rec IN (

SELECT l.LoanID, l.DueDate, c.CustomerName

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.DueDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || rec.LoanID ||

' for customer ' || rec.CustomerName ||

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

);

END LOOP;

END;

/

SELECT \* FROM Customers;

SELECT \* FROM Loans;



**Exercise 3: Stored Procedures**

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE Accounts';

EXECUTE IMMEDIATE 'DROP TABLE Employees';

EXCEPTION

WHEN OTHERS THEN NULL;

END;

/

CREATE TABLE Accounts (

AccountID NUMBER PRIMARY KEY,

CustomerID NUMBER,

AccountType VARCHAR2(20), -- e.g., 'Savings', 'Checking'

Balance NUMBER(10,2)

);

CREATE TABLE Employees (

EmployeeID NUMBER PRIMARY KEY,

Name VARCHAR2(100),

Department VARCHAR2(50),

Salary NUMBER(10,2)

);

INSERT INTO Accounts VALUES (1, 101, 'Savings', 5000);

INSERT INTO Accounts VALUES (2, 102, 'Checking', 3000);

INSERT INTO Accounts VALUES (3, 103, 'Savings', 10000);

INSERT INTO Accounts VALUES (4, 104, 'Savings', 8000);

INSERT INTO Employees VALUES (201, 'Alice', 'IT', 60000);

INSERT INTO Employees VALUES (202, 'Bob', 'HR', 50000);

INSERT INTO Employees VALUES (203, 'Charlie', 'IT', 55000);

COMMIT;

**Scenario 01**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END;

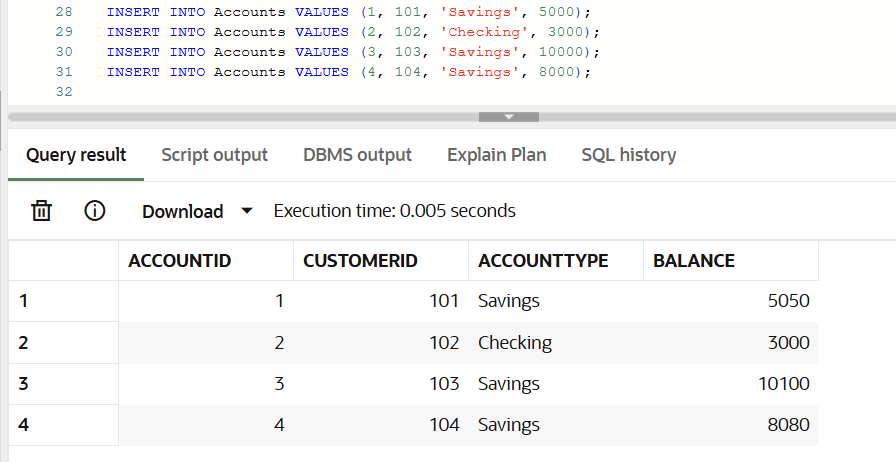
/

BEGIN

ProcessMonthlyInterest;

END;

/



**Scenario 02**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

deptName IN VARCHAR2,

bonusPercent IN NUMBER

) IS

BEGIN

UPDATE Employees

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

WHERE Department = deptName;

COMMIT;

END;

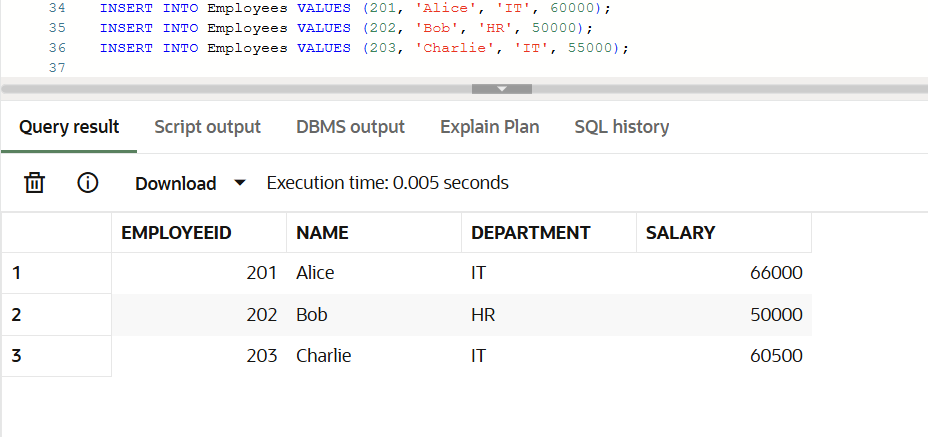
/

BEGIN

UpdateEmployeeBonus('IT', 10);

END;

/



**Scenario 03**

CREATE OR REPLACE PROCEDURE TransferFunds (

fromAccount IN NUMBER,

toAccount IN NUMBER,

amount IN NUMBER

) IS

insufficient\_funds EXCEPTION;

current\_balance NUMBER;

BEGIN

SELECT Balance INTO current\_balance

FROM Accounts

WHERE AccountID = fromAccount

FOR UPDATE;

IF current\_balance < amount THEN

RAISE insufficient\_funds;

END IF;

UPDATE Accounts

SET Balance = Balance - amount

WHERE AccountID = fromAccount;

UPDATE Accounts

SET Balance = Balance + amount

WHERE AccountID = toAccount;

COMMIT;

EXCEPTION

WHEN insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Transfer failed: Insufficient balance.');

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('An unexpected error occurred.');

END;

/

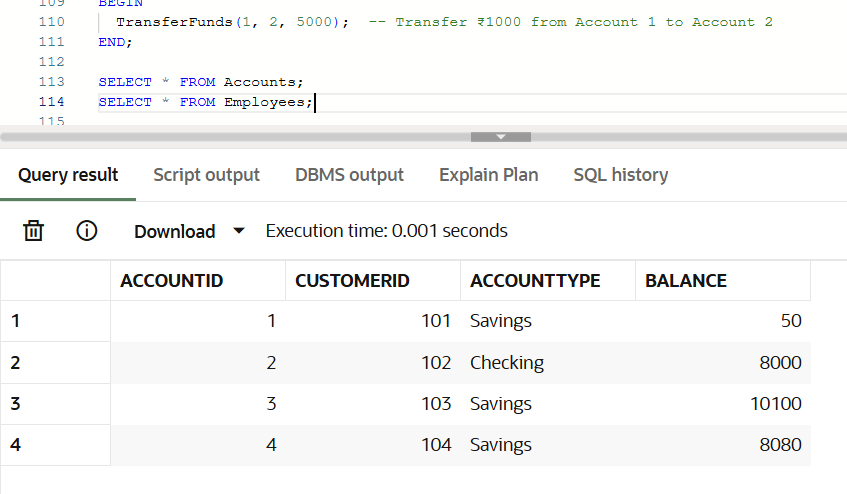
BEGIN

TransferFunds(1, 2, 5000);

END;

SELECT \* FROM Accounts;

SELECT \* FROM Employees;



**JUnit Testing Exercises**

**Exercise 1: Setting Up JUnit**

1. Create a new Java project in your IDE (e.g., IntelliJ IDEA, Eclipse).

mvn archetype:generate -DgroupId=com.example -DartifactId=JUnitDemo -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

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.

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.\*;

public class CalculatorTest {

@Test

public void testAddition() {

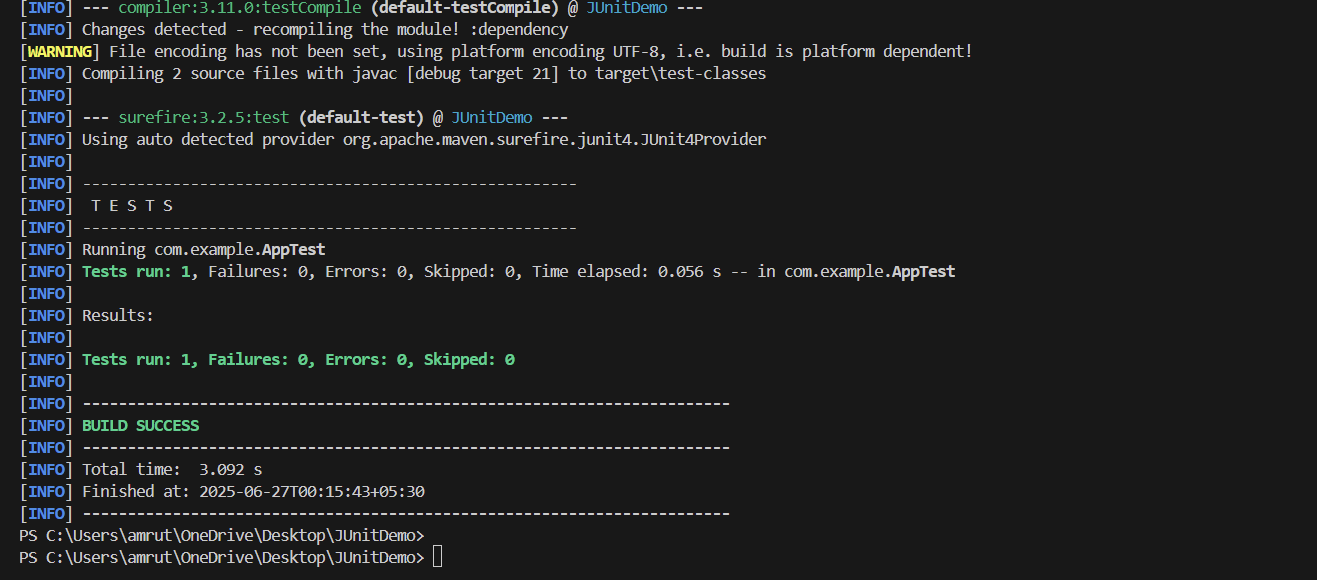
Calculator calc = new Calculator();

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

assertEquals(5, result);

}

}



**Exercise 3: Assertions in JUnit**

package com.example;

import org.junit.Test;

import static org.junit.Assert.\*;

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

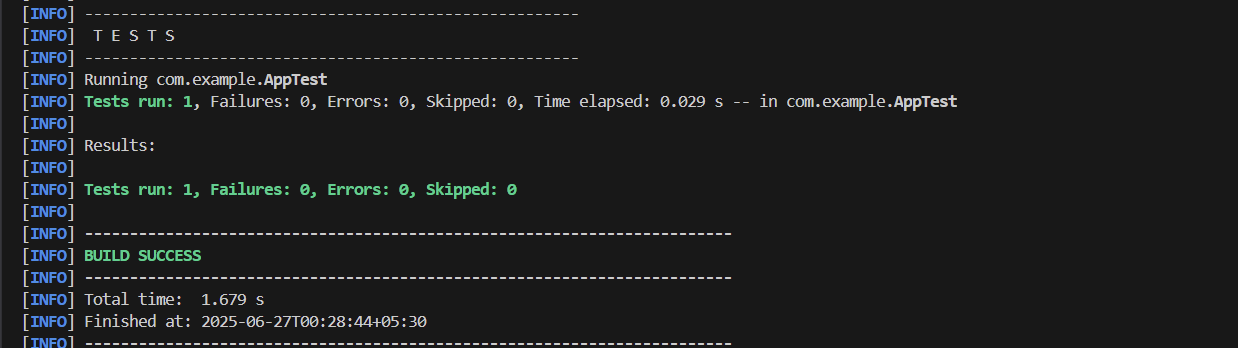
assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}



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

package com.example;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

import static org.junit.Assert.\*;

public class CalculatorTest {

private Calculator calculator;

// Arrange: Setup method runs before each test

@Before

public void setUp() {

calculator = new Calculator();

System.out.println("Setup done.");

}

// Teardown method runs after each test

@After

public void tearDown() {

System.out.println("Cleanup done.");

}

@Test

public void testAddition() {

// Act

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

// Assert

assertEquals(15, result);

}

@Test

public void testSubtraction() {

// Act

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

// Assert

assertEquals(5, result);

}

}

///calculator.java

package com.example;

public class Calculator {

public int add(int a, int b) {

return a + b;

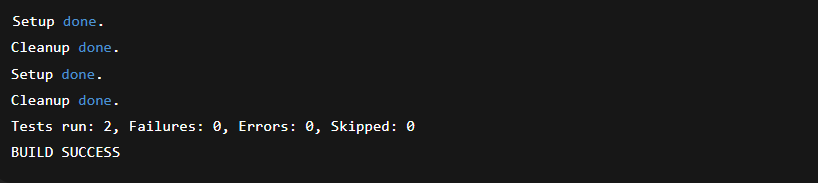
}

public int subtract(int a, int b) {

return a - b;

}

}



**Mockito Hands-On Exercises**

**Exercise 1: Mocking and Stubbing**

### **ExternalApi.java**

java

CopyEdit

package com.example;  
  
public interface ExternalApi {  
 String getData();  
}

### **MyService.java**

java

CopyEdit

package com.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData(); // Calls the external API  
 }  
}

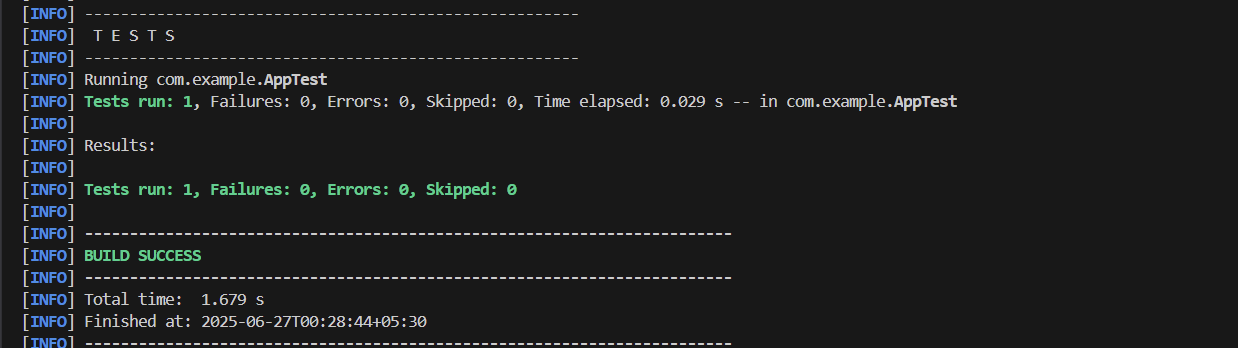
### **MyServiceTest.java**

📄 Save as: src/test/java/com/example/MyServiceTest.java

java

CopyEdit

package com.example;  
  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.assertEquals;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testExternalApi() {  
 // Step 1: Create mock  
 ExternalApi mockApi = mock(ExternalApi.class);  
  
 // Step 2: Stub the mock to return fixed value  
 when(mockApi.getData()).thenReturn("Mock Data");  
  
 // Step 3: Inject mock into service  
 MyService service = new MyService(mockApi);  
  
 // Step 4: Test  
 String result = service.fetchData();  
 assertEquals("Mock Data", result);  
 }  
}



### **ExternalApi.java (same as before)**

java

CopyEdit

package com.example;  
  
public interface ExternalApi {  
 String getData();  
}

package com.example;

public class MyService {

private final ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData(); // Method to be verified

}

}

package com.example;

import org.junit.jupiter.api.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

// Step 1: Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Step 2: Use service with mock

MyService service = new MyService(mockApi);

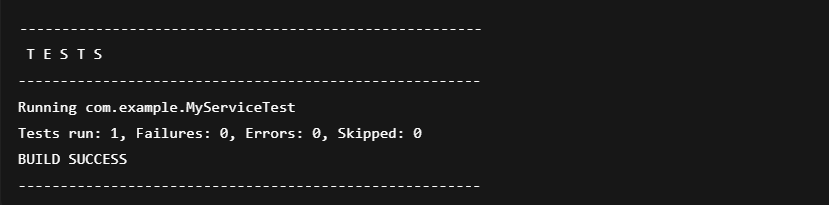
service.fetchData();

// Step 3: Verify that getData() was called

verify(mockApi).getData();

}

}



**Logging using SLF4J**

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

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");

}

}

