**PL/SQL EXERCISES**

**Control Structures**

1. Scenario 1

DECLARE

v\_age NUMBER;

BEGIN

FOR cust IN (SELECT CustomerID, DOB FROM Customers) LOOP

v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, cust.DOB) / 12);

IF v\_age > 60 THEN

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

1. Scenario 2

ALTER TABLE Customers ADD (IsVIP CHAR(1));

-- 'Y' for VIP, NULL or 'N' for non-VIP

BEGIN

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

IF cust.Balance > 10000 THEN

UPDATE Customers

SET IsVIP = 'Y'

WHERE CustomerID = cust.CustomerID;

ELSE

UPDATE Customers

SET IsVIP = 'N'

WHERE CustomerID = cust.CustomerID;

END IF;

END LOOP;

COMMIT;

END;

/

1. Scenario 3

DECLARE

v\_name Customers.Name%TYPE;

BEGIN

FOR loan\_rec IN (

SELECT l.LoanID, l.CustomerID, l.EndDate, c.Name

FROM Loans l

JOIN Customers c ON l.CustomerID = c.CustomerID

WHERE l.EndDate <= SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE(

'Reminder: Loan ID ' || loan\_rec.LoanID ||

' for customer ' || loan\_rec.Name ||

' is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD')

);

END LOOP;

END;

**Stored Procedures**

1. Scenario 1

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

FOR acc IN (SELECT AccountID, Balance FROM Accounts WHERE AccountType = 'Savings') LOOP

UPDATE Accounts

SET Balance = Balance + (acc.Balance \* 0.01),

LastModified = SYSDATE

WHERE AccountID = acc.AccountID;

END LOOP;

COMMIT;

END;

/

1. Scenario 2

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN Employees.Department%TYPE,

p\_bonus\_percent IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE Department = p\_department;

COMMIT;

END;

/

1. Scenario 3

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) IS

v\_balance NUMBER;

BEGIN

-- Check if the source account has sufficient balance

SELECT Balance INTO v\_balance FROM Accounts WHERE AccountID = p\_from\_account FOR UPDATE;

IF v\_balance < p\_amount THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds in the source account.');

END IF;

-- Deduct from source

UPDATE Accounts

SET Balance = Balance - p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_from\_account;

-- Add to destination

UPDATE Accounts

SET Balance = Balance + p\_amount,

LastModified = SYSDATE

WHERE AccountID = p\_to\_account;

-- Log the transactions (optional)

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_SEQ.NEXTVAL, p\_from\_account, SYSDATE, p\_amount, 'Transfer Out');

INSERT INTO Transactions (TransactionID, AccountID, TransactionDate, Amount, TransactionType)

VALUES (Transactions\_SEQ.NEXTVAL, p\_to\_account, SYSDATE, p\_amount, 'Transfer In');

COMMIT;

END;

/

**JUnit, Mockito Exercises**

**Exercise 1**:

Code:

Greet.java

public class Greet {

public String says() {

return "hello";

}

}

greetTest.java

import org.junit.Test;

import static org.junit.Assert.\*;

public class GreetTest {

@Test

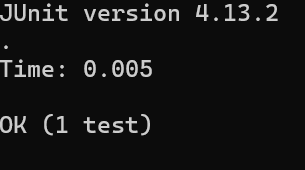
public void test() {

Greet g = new Greet();

assertEquals("hello", g.says());

}

}



**Exercise 3: Assertions in JUnit**

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

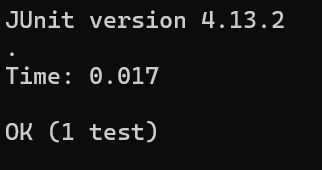
assertNull(null);

// Assert not null

assertNotNull(new Object());

}

}



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

***Calculator.java***

public class Calculator {

public int add(int a, int b) { return a + b; }

public int subtract(int a, int b) { return a - b; }

}

***CalculatorTest.java***

import static org.junit.Assert.\*;

import org.junit.Before;

import org.junit.After;

import org.junit.Test;

public class CalculatorTest {

private Calculator calculator;

@Before

public void setUp() {

// Arrange (Setup)

calculator = new Calculator();

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

}

@After

public void tearDown() {

// Cleanup

calculator = null;

System.out.println("Tearing down...");

}

@Test

public void testAdd() {

// Act

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

// Assert

assertEquals(5, result);

}

@Test

public void testSubtract() {

// Act

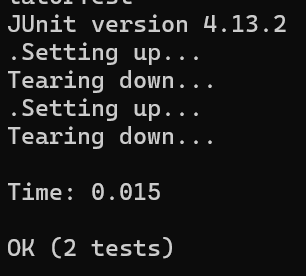
int result = calculator.subtract(5, 2);

// Assert

assertEquals(3, result);

}

}



**Exercise 1: Mocking and Stubbing**

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 static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

import org.junit.Test;

public class MyServiceTest {

@Test

public void testExternalApi() {

// Create mock

ExternalApi mockApi = mock(ExternalApi.class);

// Stub method

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

// Inject mock into service

MyService service = new MyService(mockApi);

// Act

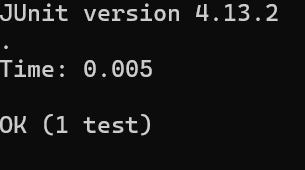
String result = service.fetchData();

// Assert

assertEquals("Mock Data", result);

}

}



**Exercise 2: Verifying Interactions**

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 static org.mockito.Mockito.\*;

import static org.junit.Assert.\*;

import org.junit.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

}

