**PL/SQL**

**Exercise 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.

**Solution:**

BEGIN

FOR rec IN (

SELECT cid.CustomerID, lid.LoanID, lr.InterestRate

FROM Customers cid

JOIN Loans l ON cid.CustomerID = lid.CustomerID

WHERE MONTHS\_BETWEEN(SYSDATE, cid.DOB) / 12 > 60

) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

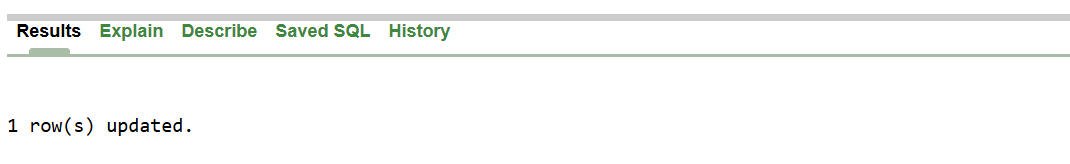
WHERE LoanID = rec.LoanID;

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.

**Solution:**

BEGIN

FOR rec IN (

SELECT CustomerID, Balance

FROM Customers

WHERE Balance > 10000

) LOOP

UPDATE Customers

SET IsVIP = 'TRUE'

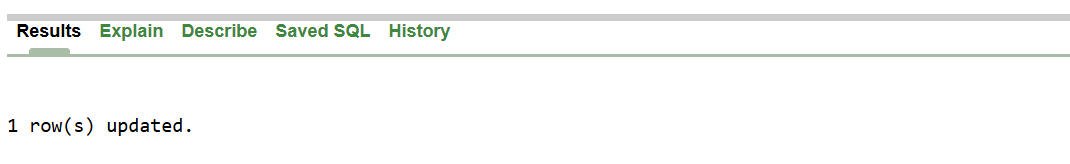
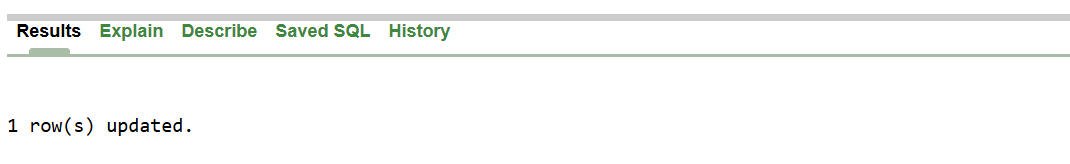
WHERE CustomerID = rec.CustomerID;

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.

**Solution:**

BEGIN

FOR rec IN (

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

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE l.DueDate BETWEEN SYSDATE AND SYSDATE + 30

) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Customer ' || rec.Name ||

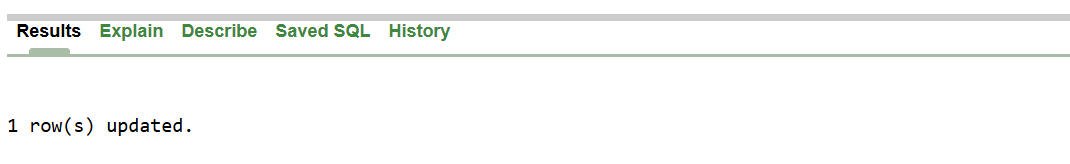
' (ID: ' || rec.CustomerID || ') has a loan (Loan ID: ' || rec.LoanID ||

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

END LOOP;

END;

/



**Exercise 3: 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.

**Solution:**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

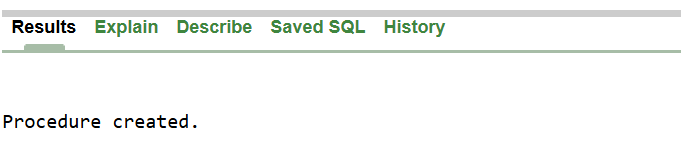
UPDATE Accounts

SET Balance = Balance + (Balance \* 0.01)

WHERE AccountType = 'SAVINGS';

COMMIT;

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.

**Solution:**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department\_id IN NUMBER,

p\_bonus\_percent IN NUMBER

) AS

BEGIN

UPDATE Employees

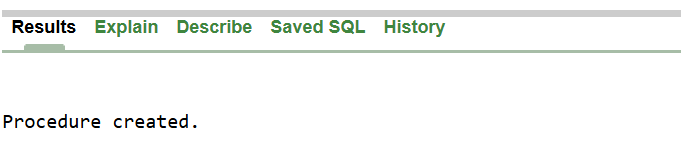
SET Salary = Salary + (Salary \* p\_bonus\_percent / 100)

WHERE DepartmentID = p\_department\_id;

COMMIT;

END;

/



**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.

**Solution:**

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) AS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account\_id;

IF v\_balance < p\_amount THEN

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

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

UPDATE Accounts

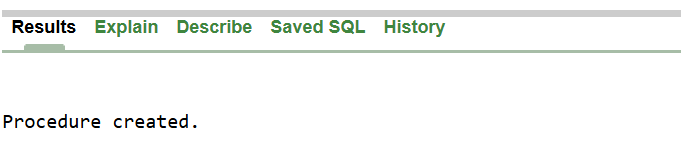
SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

END;

/



**JUnit Testing Exercises**

**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:

<dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
</dependency>

1. Create a new test class in your project.

**Exercise 3: 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.  
  
Solution Code:

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**

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.  
  
Solution Code:

import static org.junit.Assert.\*;  
import org.junit.Before;  
import org.junit.After;  
import org.junit.Test;  
  
public class BankAccountTest {  
 private BankAccount account;  
  
 @Before  
 public void setUp() {  
 // Arrange  
 account = new BankAccount();  
 account.deposit(100);  
 }  
  
 @Test  
 public void testWithdraw() {  
 // Act  
 account.withdraw(40);  
  
 // Assert  
 assertEquals(60, account.getBalance());  
 }  
  
 @After  
 public void tearDown() {  
 account = null;  
 }  
}  
  
class BankAccount {  
 private int balance = 0;  
  
 public void deposit(int amount) {  
 balance += amount;  
 }  
  
 public void withdraw(int amount) {  
 balance -= amount;  
 }  
  
 public int getBalance() {  
 return balance;  
 }  
}

**MOCKITO EXERCISES**

**Exercise 1: Mocking and Stubbing**

**Question:**  
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.

**Solution:**

java

CopyEdit

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

@Test

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);

}

}

**Exercise 2: Verifying Interactions**

**Question:**  
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.

**Solution:**

java

CopyEdit

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.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();

}

}

**SL4J Logging Exercises**

Exercise 1: Logging Error Messages and Warning Levels

Task:  
Write a Java application that demonstrates logging error messages and warning levels using SLF4J.

Solution:

Step 1: Add SLF4J and Logback dependencies to your pom.xml

xml

CopyEdit

<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>

Step 2: Create a Java class that uses SLF4J for logging

java

CopyEdit

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");

}

}