**TWO PHASE COMMIT**

Sure, let's walk through a simple example of implementing a two-phase commit protocol between two databases using Java in Eclipse and MSSQL as the database. In this example, we'll consider a scenario where we need to transfer funds from one account to another atomically.

Step 1: Set Up MSSQL Databases

Create Databases: Create two databases named source\_db and destination\_db.

Create Tables: Create a table named accounts in each database to store account information.

sql

Copy code

-- Source Database

CREATE TABLE source\_db.accounts (

id INT PRIMARY KEY,

balance DECIMAL(10, 2)

);

-- Destination Database

CREATE TABLE destination\_db.accounts (

id INT PRIMARY KEY,

balance DECIMAL(10, 2)

);

Step 2: Set Up Eclipse Project

Create New Java Project: Open Eclipse and create a new Java project.

Add MSSQL JDBC Driver: Download the MSSQL JDBC driver (e.g., sqljdbc42.jar) and add it to your project's build path.

Step 3: Write Java Code for Two-Phase Commit

java

Copy code

import java.sql.\*;

public class TwoPhaseCommitExample {

private static final String SOURCE\_DB\_URL = "jdbc:sqlserver://source\_server:source\_port;databaseName=source\_db";

private static final String DESTINATION\_DB\_URL = "jdbc:sqlserver://destination\_server:destination\_port;databaseName=destination\_db";

private static final String USER = "your\_username";

private static final String PASSWORD = "your\_password";

public static void main(String[] args) {

Connection conn1 = null;

Connection conn2 = null;

try {

// Register JDBC driver

Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");

// Open connections

conn1 = DriverManager.getConnection(SOURCE\_DB\_URL, USER, PASSWORD);

conn2 = DriverManager.getConnection(DESTINATION\_DB\_URL, USER, PASSWORD);

// Set auto-commit to false

conn1.setAutoCommit(false);

conn2.setAutoCommit(false);

// Update balances

updateBalance(conn1, 1, -100); // Withdraw $100 from account 1

updateBalance(conn2, 2, 100); // Deposit $100 to account 2

// Commit the transaction

conn1.commit();

conn2.commit();

System.out.println("Transaction committed successfully.");

} catch (SQLException se) {

// Handle errors for JDBC

se.printStackTrace();

try {

if (conn1 != null) conn1.rollback();

if (conn2 != null) conn2.rollback();

} catch (SQLException re) {

re.printStackTrace();

}

} catch (Exception e) {

// Handle errors for Class.forName

e.printStackTrace();

} finally {

// Finally block to close resources

try {

if (conn1 != null) conn1.close();

if (conn2 != null) conn2.close();

} catch (SQLException se) {

se.printStackTrace();

}

}

}

private static void updateBalance(Connection conn, int accountId, double amount) throws SQLException {

PreparedStatement pstmt = null;

String sql = "UPDATE accounts SET balance = balance + ? WHERE id = ?";

try {

pstmt = conn.prepareStatement(sql);

pstmt.setDouble(1, amount);

pstmt.setInt(2, accountId);

pstmt.executeUpdate();

} finally {

if (pstmt != null) pstmt.close();

}

}

}

Replace "source\_server", "source\_port", "destination\_server", "destination\_port", "your\_username", and "your\_password" with your actual server details.

Step 4: Testing

Compile and Run: Compile and run the Java program in Eclipse.

Verify Database: Check the balances in the accounts table of both databases to verify that the transaction was successful.

Step 5: Deployment

Deploy your application to your production environment after thorough testing.

This example demonstrates a simple two-phase commit protocol between two databases using Java in Eclipse with MSSQL. You can extend this example to handle more complex transactions and error scenarios as needed.