**Implement a 2PC for distributed transaction management**

**Description**: A two-phase commit is a standardized protocol that ensures that a database commit is implementing in the situation where a commit operation must be broken into two separate parts.

In database management, saving data changes is known as a commit and undoing changes is known as a rollback. Both can be achieved easily using transaction logging when a single server is involved, but when the data is spread across geographically-diverse servers in distributed computing (i.e., each server being an independent entity with separate log records), the process can become more tricky.

A special object, known as a coordinator, is required in a distributed transaction. As its name implies, the coordinator arranges activities and synchronization between distributed servers. The two-phase commit is implemented as follows:

Phase 1 - Each server that needs to commit data writes its data records to the log. If a server is unsuccessful, it responds with a failure message. If successful, the server replies with an OK message.

Phase 2 - This phase begins after all participants respond OK. Then, the coordinator sends a signal to each server with commit instructions. After committing, each writes the commit as part of its log record for reference and sends the coordinator a message that its commit has been successfully implemented.

If a server fails, the coordinator sends instructions to all servers to roll back the transaction. After the servers roll back, each sends feedback that this has been completed.

**Source Code:**

**SOURCE CODE**:

Client.java

package TwoPC;

// Two Phase Commit protocol: Client Application

import java.io.\*;

import java.net.\*;

import java.awt.\*;

import java.awt.event.\*;

import javax.swing.\*;

import java.sql.\*;

class DBConnector {

public static Connection getDBConnection(String dsn) throws Exception {

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

return DriverManager.getConnection("jdbc:odbc:"+dsn);

}

}

public class Client extends JFrame implements ActionListener{

JButton b1,b2,b4,b5;

JPanel p1,p2;

JTextField t1;

JLabel l1;

ServerSocket ss;

Socket s;

DataOutputStream output;

DataInputStream input;

Connection con;

Statement stmt;

String serverMessage="Prepared";

int port = 8890;

String groupIP = "228.5.6.200";

Client(){

b1=new JButton("Prepared");

b2=new JButton("NotPrepared");

b4=new JButton("Execute");

b5=new JButton("Exit");

t1=new JTextField("",35);

l1=new JLabel("SQL");

p1=new JPanel();

p2=new JPanel();

p1.setLayout(new FlowLayout());

p1.add(l1);

p1.add(t1);

p2.add(b1);

p2.add(b2);

p2.add(b4);

p2.add(b5);

add(p1);

add(p2,"South");

setSize(600,300);

setTitle("Two Phase Commit Protocol: Client");

b1.addActionListener(this);

b2.addActionListener(this);

b4.addActionListener(this);

b5.addActionListener(this);

setVisible(true);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

MulticastSocketms =null;

InetAddress group ;

try

{

s = new Socket("localhost",8088);

System.out.println("Client Connected");

output=new DataOutputStream(s.getOutputStream());

input=new DataInputStream(s.getInputStream());

con = DBConnector.getDBConnection("my2pcdsn");

stmt = con.createStatement();

con.setAutoCommit(false);

ms = new MulticastSocket(port);

group= InetAddress.getByName(groupIP);

ms.joinGroup(group);

byte[] buffer = new byte[1024];

output.writeUTF("NotPrepared");

while (true)

{

DatagramPacketserMsg= new DatagramPacket(buffer, buffer.length);

ms.receive(serMsg);

String commitMsg = new String (serMsg.getData()).trim();

if (commitMsg.equals("commit"))

{

System.out.println("Received "+commitMsg);

con.commit();

t1.setText("Transactions Committed");

System.out.println("Transactions Committed");

}

}

}

catch (ConnectExceptionce)

{

ce.printStackTrace();

System.exit(0);

}

catch (Exception e)

{

e.printStackTrace();

}

}

public void actionPerformed(ActionEvent ae){

try

{

String str=ae.getActionCommand();

if(str.equals("Execute")){

String query = t1.getText();

stmt.executeUpdate(query);

t1.setText("Query Executed (NotPrepared)");

output.writeUTF("NotPrepared");

}

if(str.equals("Prepared")){

output.writeUTF("Prepared");

t1.setText(input.readUTF());

}

if(str.equals("NotPrepared")){

output.writeUTF("NotPrepared");

t1.setText("NotPrepared");

}

if(str.equals("Exit")){

output.writeUTF("Prepared");

stmt.close();

con.close();

System.exit(0);

}

}

catch(Exception e){

JLabelerrorFields = new JLabel("<HTML><FONT COLOR = BLUE>"+e.getMessage()+"</FONT></HTML>");

JOptionPane.showMessageDialog(null,errorFields);

e.printStackTrace();

}

}

public static void main(String args[]){

Client c=new Client();

}

}

Server.java

package TwoPC;

// Two Phase Commit protocol: Server Application

import java.io.\*;

import java.net.\*;

public class Server {

public static ServerSocket ss;

public Server() {

}

public static void main(String args[]) throws Exception {

ss = new ServerSocket(8088);

System.out.println("Two Phase Commit Protocol: Server");

new Clients(2);

while (true) {

System.out.println("Server waiting: ");

Socket s = ss.accept();

new Coordinator(s);

}

}

}

class Clients {

static int n;

static String[] status;

Clients(int num) {

n = num;

status = new String[n];

for (int j = 0; j < n; j++) {

status[j] = new String("");

}

}

}

class Coordinator implements Runnable {

public static int i = -1;

int flag = 1;

Socket s;

Thread t;

MulticastSocketms = null;

InetAddress group;

DataInputStream input;

DataOutputStream output;

int port = 8890;

String groupIP = "228.5.6.200";

Coordinator(Socket c) {

s = c;

try {

input = new DataInputStream(s.getInputStream());

output = new DataOutputStream(s.getOutputStream());

ms = new MulticastSocket(port);

group = InetAddress.getByName(groupIP);

ms.joinGroup(group);

} catch (Exception e) {

e.printStackTrace();

}

t = new Thread(this);

t.start();

i++;

}

public void run() {

int index = i;

String clientSattus;

try {

while (true) {

clientSattus = input.readUTF();

if (clientSattus.equalsIgnoreCase("Prepared")) {

output.writeUTF("Wait for others to prepare");

}

Clients.status[index] = new String(clientSattus);

for (int k = 0; k <Clients.n; k++) {

System.out.println("Client " + (k + 1) + " " + Clients.status[k]);

if (Clients.status[k].equalsIgnoreCase("Prepared")) {

continue;

} else {

flag = 0;

}

}

if (flag == 1) {

byte[] msg = new String("commit").getBytes();

DatagramPacketmsgpack = new DatagramPacket(msg, msg.length, group, port);

ms.send(msgpack);

System.out.println("Commit message sent to clients: " + new String(msg));

}

flag = 1;

}

} catch (Exception e) {

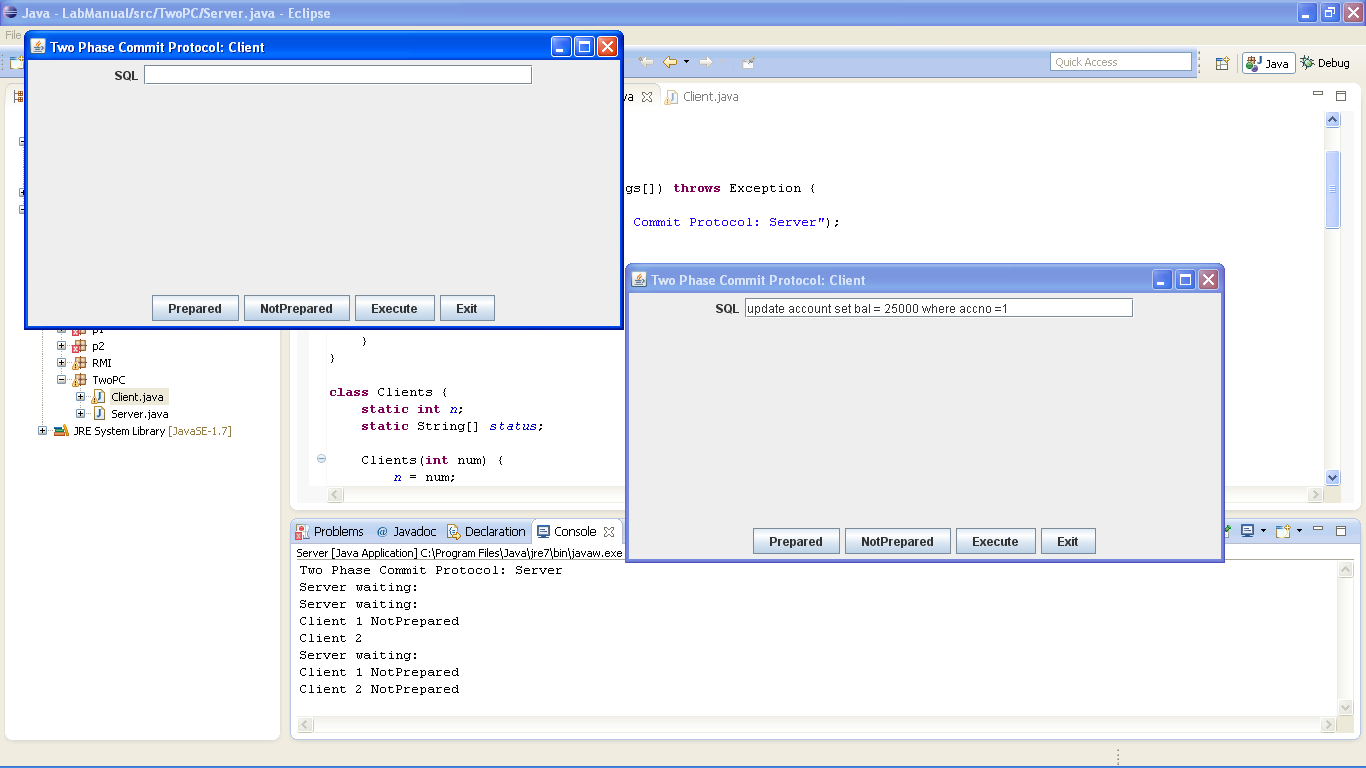
e.printStackTrace();

}

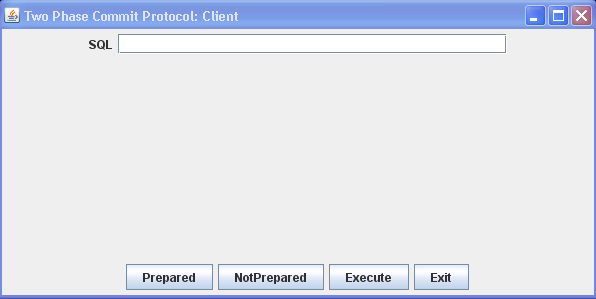
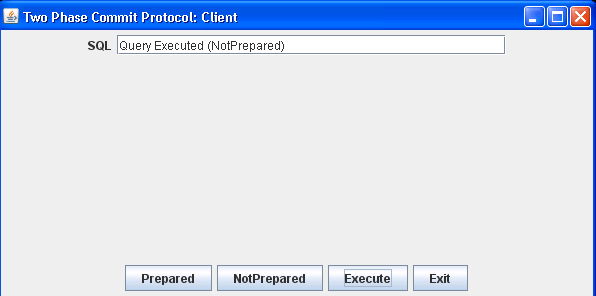
}

}

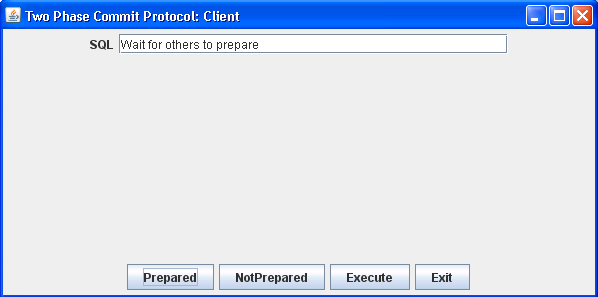
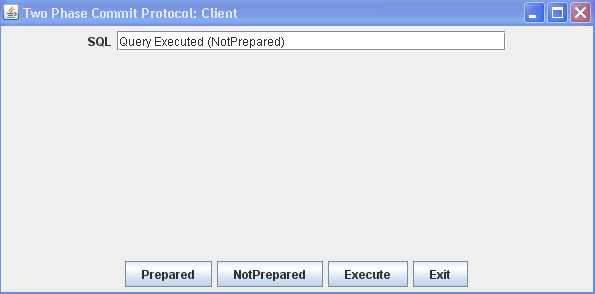
**OUTPUT**:



**Client 1 executes Query**



**Client 2 says Prepared**



**Client 1 says Prepared and since both clients said Prepared, the transaction Commits**

