Name: Dhruv Bheda **SAPID:** 60004200102

DIV: B/B1

$\frac{\textbf{ADBMS}}{\textbf{Exp6}}$

	Dhauv Bheda
	ADBMS 60004200102
	Exp-6.
	Aim: Implementation of 2-Phase Protocol
1.	Theory:
	-> The two-phase commet protocol breaks a
	database commit into two phases to ensure
	correctness & town tolerence in at
The latest of	convections & fault tolerence in af
	-> Phase 1: Pocparie Phase
	-> After each slave has locally completed it's
	. transaction, et sende a "Done" message to
19.00	the controlling site.
-	-> When the controlling sit has received "DONE"
- / 1	"message from all slaves, it sends a "Prepare"
	mesage to slaves
	-> The slaves vote on whether they still want
	to commit or not.
- No	-> It a slowe wan'ts to commit it sends
	"Ready" mesoge
	-> Otherwise, it sends "Not Ready" masage.
	> Ohoo 2: Omet/Ahm Phage
	> Phase 2: Commit/Aboxt Phase. > After the controlling site has received "Roady"
	mesage from all slaves:
	-1) The controlling site sends "Alobal Commit" may to
	slaves.
	11) The slaves apply the transaction & send a
	"Commit Ack" message to controlling site.
	FOR FOLKS
Sundaram	FOR EDUCATIONAL USE
A STATE OF THE PARTY OF	

Disadvantages: a) The major disadvantage of the Two-phase commit protocol is faced when Coordinator site failure may result in blocking, so a decre con either to commit or about Transaction (T) may have to be postponed until appointer recovere b) Consider a scenario, if a Transaction (T) holds lock on data items of active sites, but. amid the execution sif coordinator talls & the active sites keep no additional log-second except < read t T> like < about > or < commits. go, it becomes impossible to determine what deasion has been made to xcommit >/Kabons Conclusion :-Thus, we successfully studied and implemented the 2-Phase Protocol.

Code:

Client

```
import java.io.*;
import java.net.*;
public class Client implements Runnable
    static Socket clientSocket = null;
    static PrintStream os = null;
    static DataInputStream is = null;
    static BufferedReader inputLine = null;
    static boolean closed = false;
    public static void main(String[] args)
    {
        int port number=1111;
        String host="localhost";
        try {
            clientSocket = new Socket(host, port_number);
            inputLine = new BufferedReader(new
InputStreamReader(System.in));
            os = new PrintStream(clientSocket.getOutputStream());
            is = new DataInputStream(clientSocket.getInputStream());
        } catch (Exception e)
            System.out.println("Exception occurred : "+e.getMessage());
        if (clientSocket != null && os != null && is != null)
            try
            {
                new Thread(new Client()).start();
                while (!closed)
                {
                    os.println(inputLine.readLine());
                os.close();
                is.close();
                clientSocket.close();
            } catch (IOException e)
                System.err.println("IOException: " + e);
```

```
@SuppressWarnings("deprecation")
    public void run()
        String responseLine;
        try
        {
           while ((responseLine = is.readLine()) != null)
            {
                System.out.println("\n"+responseLine);
                if (responseLine.equalsIgnoreCase("GLOBAL COMMIT")==true
| responseLine.equalsIgnoreCase("GLOBAL ABORT")==true )
                {
                    break;
            closed=true;
        catch (IOException e)
            System.err.println("IOException: " + e);
    }
```

Server

```
import java.io.*;
import java.net.*;
import java.util.*;

public class Server {
    boolean closed = false, inputFromAll = false;
    List<ClientThread> thread;
    List<String> data;
    List<String> decision;

Server() {
```

```
thread = new ArrayList<ClientThread>();
        data = new ArrayList<String>();
        decision= new ArrayList<String>();
    }
    public static void main(String args[])
        Socket clientSocket = null;
        ServerSocket = null;
        int port number = 1111;
        Server server = new Server();
        {
            serverSocket = new ServerSocket(port number);
        } catch (IOException e) {
            System.out.println(e);
        while (!server.closed)
            try {
                    clientSocket = serverSocket.accept();
                    ClientThread clientThread = new ClientThread(server,
clientSocket);
                    (server.thread).add(clientThread);
                    System.out.println("\nNow Total clients are : " +
(server.thread).size());
                    (server.data).add("NOT_SENT");
                    (server.decision).add("NOT_SENT");
                    clientThread.start();
            } catch (IOException e) { }
        try {
            serverSocket.close();
        } catch (Exception e1) { }
class ClientThread extends Thread
    DataInputStream is = null;
    String line;
    String destClient = "";
```

```
String name;
    PrintStream os = null;
    Socket clientSocket = null;
    String clientIdentity;
    Server server;
    public ClientThread(Server server, Socket clientSocket)
        this.clientSocket = clientSocket;
        this.server = server;
    }
    @SuppressWarnings("deprecation")
    public void run()
        try {
            is = new DataInputStream(clientSocket.getInputStream());
            os = new PrintStream(clientSocket.getOutputStream());
            os.println("Enter your name.");
            name = is.readLine();
            clientIdentity = name;
            os.println("Welcome " + name + " to this 2 Phase
Application.\nYou will receive a vote Request now...");
            os.println("Send Ready or Not Ready after local
transaction..");
            while (true)
            {
                line = is.readLine();
                if (line.equalsIgnoreCase("NOT READY"))
                    System.out.println("\nFrom '" + clientIdentity
                            + "' : NOT READY\n\nSince NOT READY we will
not wait for inputs from other clients.");
                    System.out.println("\nAborted....");
                    for (int i = 0; i < (server.thread).size(); i++) {</pre>
                        ((server.thread).get(i)).os.println("GLOBAL_ABORT"
);
                        ((server.thread).get(i)).os.close();
                        ((server.thread).get(i)).is.close();
                    break;
```

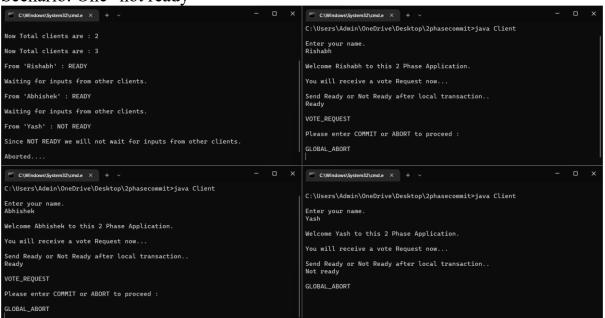
```
if (line.equalsIgnoreCase("READY"))
                {
                    System.out.println("\nFrom '" + clientIdentity + "' :
READY");
                    if ((server.thread).contains(this))
                    {
                         (server.data).set((server.thread).indexOf(this),
"READY");
                        for (int j = 0; j < (server.data).size(); j++)</pre>
(!(((server.data).get(j)).equalsIgnoreCase("NOT_SENT")))
                                 server.inputFromAll = true;
                                 continue;
                             else{
                                 server.inputFromAll = false;
                                 System.out.println("\nWaiting for inputs
from other clients.");
                                 break;
                        }
                        if (server.inputFromAll)
                             System.out.println("All Ready..");
                    }
                }
                os.println("VOTE_REQUEST\nPlease enter COMMIT or ABORT to
proceed : ");
                line = is.readLine();
                if (line.equalsIgnoreCase("ABORT"))
                {
                    System.out.println("\nFrom '" + clientIdentity
                             + "' : ABORT\n\nSince ABORT we will not wait
for inputs from other clients.");
                    System.out.println("\nAborted....");
                    for (int i = 0; i < (server.thread).size(); i++) {</pre>
                        ((server.thread).get(i)).os.println("GLOBAL_ABORT"
```

```
((server.thread).get(i)).os.close();
                         ((server.thread).get(i)).is.close();
                    break;
                if (line.equalsIgnoreCase("COMMIT")){
                    System.out.println("\nFrom '" + clientIdentity + "' :
COMMIT");
                     if ((server.thread).contains(this))
                     {
                         (server.decision).set((server.thread).indexOf(this
), "COMMIT");
                         for (int j = 0; j < (server.decision).size(); j++)</pre>
                             if
(!(((server.decision).get(j)).equalsIgnoreCase("NOT_SENT")))
                                 server.inputFromAll = true;
                                 continue;
                             else{
                                 server.inputFromAll = false;
                                 System.out.println("\nWaiting for inputs
from other clients.");
                                 break;
                         if (server.inputFromAll){
                             System.out.println("\n\nCommited....");
                             for (int i = 0; i < (server.thread).size();</pre>
i++)
                             {
                                 ((server.thread).get(i)).os.println("GLOBA
L COMMIT");
                                 ((server.thread).get(i)).os.close();
                                 ((server.thread).get(i)).is.close();
                             break;
                         }
                    }
                }
            server.closed = true;
```

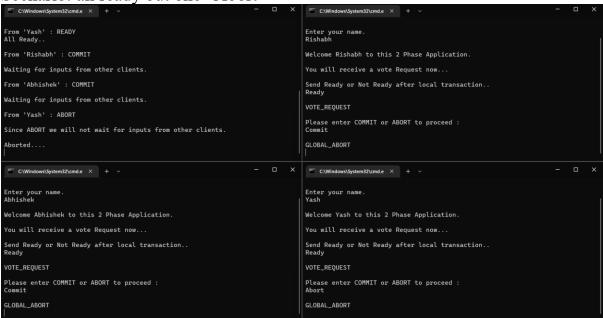
```
clientSocket.close();
} catch (IOException e) { }
}
}
```

Output:

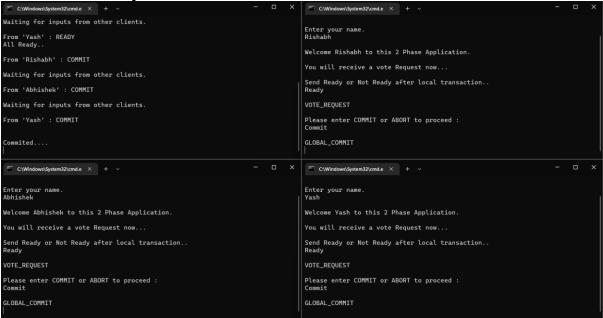
Scenario: One "not ready"



Scenario: all ready but one "Abort"



Scenario: all ready and all commit



Conclusion:

Thus, we successfully studied and implemented 2PL protocol.