Banking System

(Experiment 5 - Data Consistency)

Yash Brid 2019130008
Abhishek Chopra 2019130009
Sumeet Haldipur 2019130018

Aim

To implement data consistency for a distributed banking system.

Objective

- To implement Data Consistency in a distributed system using Java.
- To apply data consistency to keep data consistent over different databases during replication.

Theory

Consistency in database systems refers to the requirement that any given database transactions must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined database constraints. Data Consistency refers to the usability of data. Data Consistency problems may arise even in a single-sit environment during recovery situations when backup copies of the production data are used in place of the original data. In order to ensure that your backup data is usable, it is necessary to understand the backup methodologies that are in place as well as how the primary data is created and accessed. Another very important consideration is the consistency of the data once the recovery has been completed and the application is ready to begin processing.

Data Replication

- In the distributed system, data is duplicated mainly for “reliability” and “performance”. ● Replication is required, especially if the distributed system needs to grow in quantity and geographically. Replication is one of the scaling techniques. Also, it can cope with data corruption and replica crashes.
- At this time, it is necessary to maintain the consistency of the state of the data and the replica. However, this directly leads to scalability problems.

Data replication is done with an aim to:

- Increase the availability of data.
- Speed up the query evaluation.

There are two types of data replication:

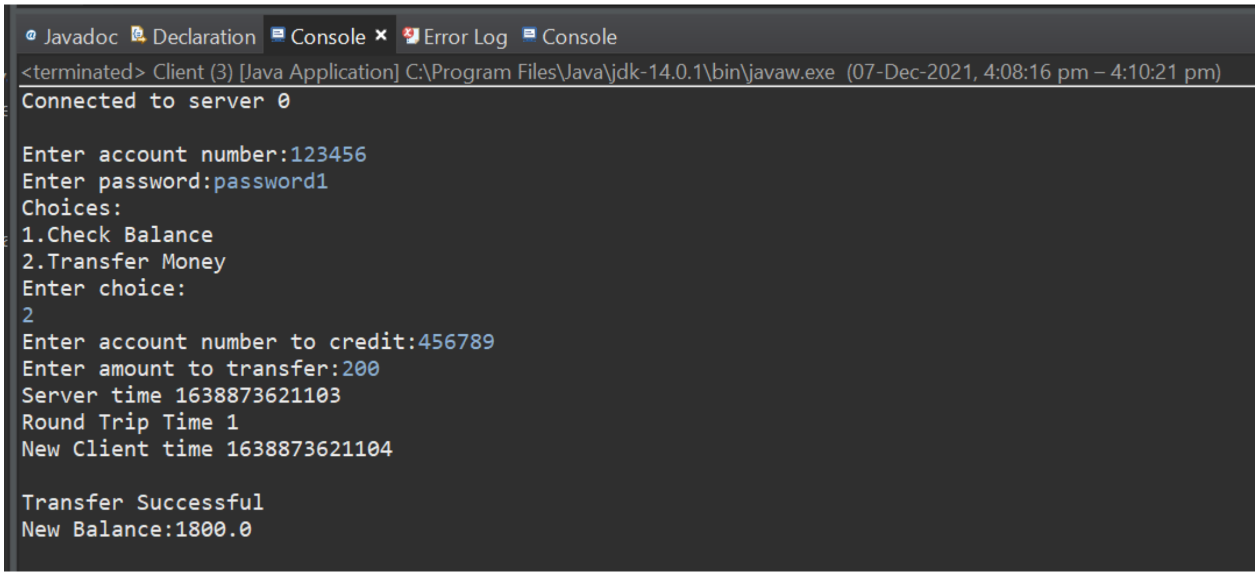
- **Synchronous Replication**: In synchronous replication, the replica will be modified immediately after some changes are made in the relation table. So there is no difference between original data and replica.
- **Asynchronous replication**: In asynchronous replication, the replica will be modified after commit is fired on to the database.

Code

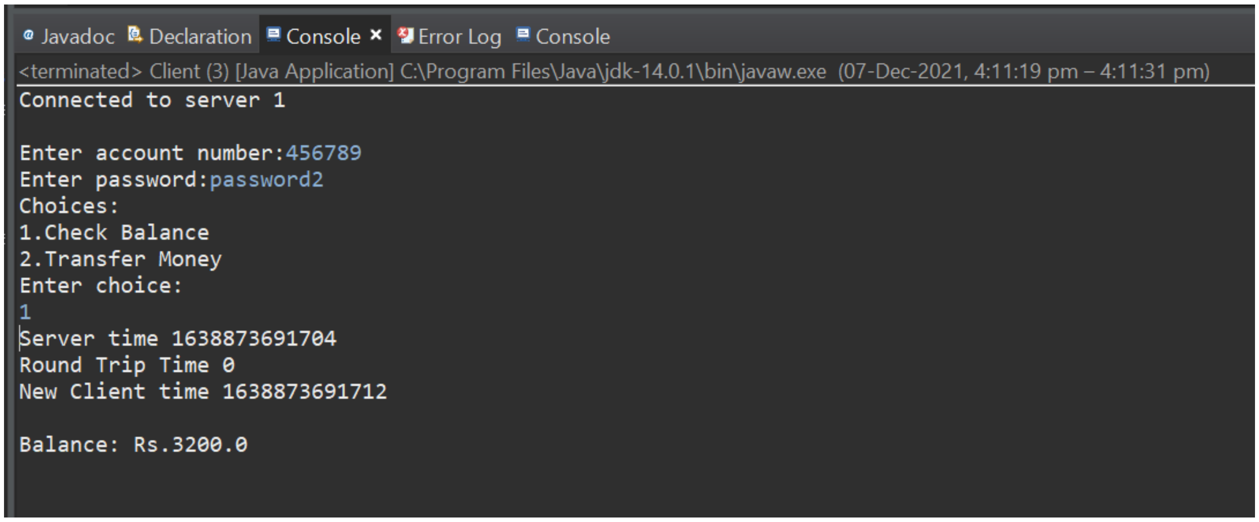
Server.java

```
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import java.util.ArrayList;

public class Server extends UnicastRemoteObject implements checkBal {
    public Server(int serverNo) throws RemoteException {
        super();
        RN = new int[3];
        no_of_requests = 0;
        exec=1;
        critical = false;
        this.serverNo = serverNo;
        try {
            Registry reg = LocateRegistry.getRegistry("localhost", 8082);
            TokenInterface token = (TokenInterface)
reg.lookup("tokenServer");
            this.token = token;
        } catch (Exception e) {
            System.out.println("Exception occurred : " + e.getMessage());
        }
    }

    static ArrayList<Account> a = new ArrayList<Account>() {
        {
            add(new Account("123456", "password1", 2000.0));
            add(new Account("456789", "password2", 3000.0));
            add(new Account("234567", "password3", 4000.0));
            add(new Account("345678", "password4", 5000.0));
        }

    };

    static ArrayList<Account> b = new ArrayList<Account>() {
        {
            add(new Account("123456", "password1", 2000.0));
            add(new Account("456789", "password2", 3000.0));
            add(new Account("234567", "password3", 4000.0));
            add(new Account("345678", "password4", 5000.0));
        }

    };



    int RN[],exec;
    boolean critical;
    int no_of_requests;
    TokenInterface token;
    int serverNo;

    public double checkBalance(String acc_no, String password) throws
RemoteException {

        try{
            if(exec==1){
            throw new Exception("Datastore not accessible");
            }
            System.out.println("Balance request received for account
number " + acc_no);
            for (int i = 0; i < a.size(); i++) {
                double bal = a.get(i).checkBalance(acc_no, password);
                if (bal != -1)
                    return bal;
            }
            return -1.0;
        }
        catch(Exception e){
            System.out.println(e.getMessage()+"\nCannot access datastore
1\nTrying to access backup datastore");

            for (int i = 0; i < b.size(); i++) {
                double bal = b.get(i).checkBalance(acc_no, password);
                if (bal != -1)
                    return bal;
            }
            return -1.0;


        }


    }

    public boolean transfer(String d_acc_no, String cred_acc_no, String
password, double amt) throws RemoteException {
        System.out.println("Transfer request received for account number "
+ d_acc_no);
        System.out.println("Transfer to credit account number " +
cred_acc_no);
        boolean isValid = false;

        try{
            if(exec==1){
                throw new Exception("Datastore not accessible");
                }
            for (int i = 0; i < a.size(); i++) {
                isValid = a.get(i).checkValid(d_acc_no, password);
                if (isValid) {
                    break;
                }
            }

        }
        catch(Exception e){
            // System.out.println(e.getMessage()+"\nCannot access
datastore 1\nTrying to access datastore 2");
            for (int i = 0; i < b.size(); i++) {
                isValid = b.get(i).checkValid(d_acc_no, password);
                if (isValid) {
                    break;
                }
            }

        }

        if (!isValid) {
            return false;
        } else {
            if (token.getOwner() == -1) {
                token.setOwner(serverNo);
                System.out.println("No owner");
                no_of_requests++;
                RN[serverNo]++;
            } else {
                sendRequest();
            }
            while (token.getOwner() != serverNo)
                ;
            System.out.println("Got token");
            critical = true;
            boolean b = critical_section(d_acc_no, cred_acc_no, password,
amt);
            critical = false;
            releaseToken();
            return b;
        }

    }

    public void sendRequest() throws RemoteException {
        no_of_requests++;
        for (int i = 0; i < 3; i++) {
            try {
                Registry reg = LocateRegistry.getRegistry("localhost",
8000+i);
                checkBal server = (checkBal) reg.lookup("bankServer"+i);
                server.receiveRequest(serverNo, no_of_requests);
            } catch (Exception e) {
                System.out.println("Exception occurred : " +
e.getMessage());
            }
        }
    }

    public boolean critical_section(String d_acc_no, String cred_acc_no,
String password, double amt) {
        int deb_ind = 0;
        int cred_ind = 0;
        try{
            if(exec==1){
                throw new Exception("Datastore not accessible");
                }
            for (int i = 0; i < a.size(); i++) {
                if (a.get(i).acc_no.equals(d_acc_no) &&
a.get(i).password.equals(password)) {
                    deb_ind = i;
                }
                if (a.get(i).acc_no.equals(cred_acc_no)) {
                    cred_ind = i;
                }
            }
            if (a.get(deb_ind).balance < amt)
                return false;
            else {
                a.get(deb_ind).balance -= amt;
                a.get(cred_ind).balance += amt;
                b.get(deb_ind).balance -= amt;
                b.get(cred_ind).balance += amt;
                return true;
            }
        }
        catch(Exception e){
            System.out.println(e.getMessage()+"\nCannot access datastore
1\nTrying to access backup datastore");
            for (int i = 0; i < b.size(); i++) {
                if (b.get(i).acc_no.equals(d_acc_no) &&
b.get(i).password.equals(password)) {
                    deb_ind = i;
                }
                if (b.get(i).acc_no.equals(cred_acc_no)) {
                    cred_ind = i;
                }
            }
            if (b.get(deb_ind).balance < amt)
                return false;
            else {
                b.get(deb_ind).balance -= amt;
                b.get(cred_ind).balance += amt;
                return true;
            }
        }
    }

    public void receiveRequest(int i, int n) throws RemoteException {
            System.out.println("Recieved request from " + i);
        if (RN[i] <= n) {
            RN[i] = n;
            if (token.getToken()[i] + 1 == RN[i]) {
                if (token.getOwner() == serverNo) {
                    if (critical) {
                        System.out.println("Add to queue");
                        token.getQueue()[token.getTail()] = i;
                        token.setTail(token.getTail() + 1);
                    } else {
                        System.out.println("Queue empty, setting owner");
                        token.setOwner(i);
                    }
                }
            }
        }
    }

    public void releaseToken() throws RemoteException {
        token.setToken(serverNo, RN[serverNo]);
        if (token.getHead() != token.getTail()) {
            System.out.println("Release token");
            token.setOwner(token.getQueue()[token.getHead()]);
           System.out.println("New owner" + token.getOwner());
            token.setHead(token.getHead() + 1);
        }
    }

    public static void main(String[] args) {
        try {
            Registry reg = LocateRegistry.createRegistry(8000);
            reg.rebind("bankServer0", new Server(0));

            Registry reg1 = LocateRegistry.createRegistry(8001);
            reg1.rebind("bankServer1", new Server(1));

            Registry reg2 = LocateRegistry.createRegistry(8002);
            reg2.rebind("bankServer2", new Server(2));

            System.out.println("3 servers are running now ");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

class Account {
    String acc_no;
    String password;
    double balance;

    Account(String acc_no, String password, double balance) {
        this.acc_no = acc_no;
        this.password = password;
        this.balance = balance;
    }

    public double checkBalance(String acc_no, String password) { if
        (this.acc_no.equals(acc_no) && this.password.equals(password))
        return this.balance;
        else
            return -1.0;
    }
    public boolean checkValid(String acc_no, String password) { if
        (this.acc_no.equals(acc_no) && this.password.equals(password))
        return true;
        else
            return false;
    }
}
```

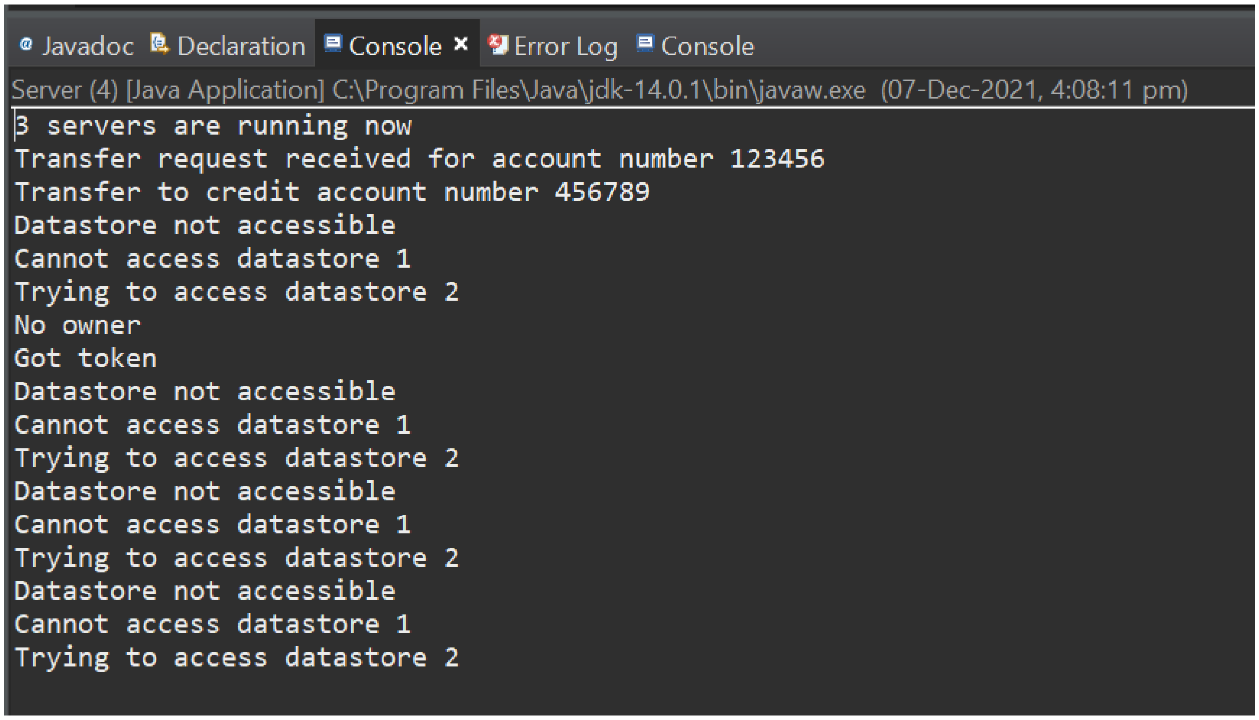
Output :



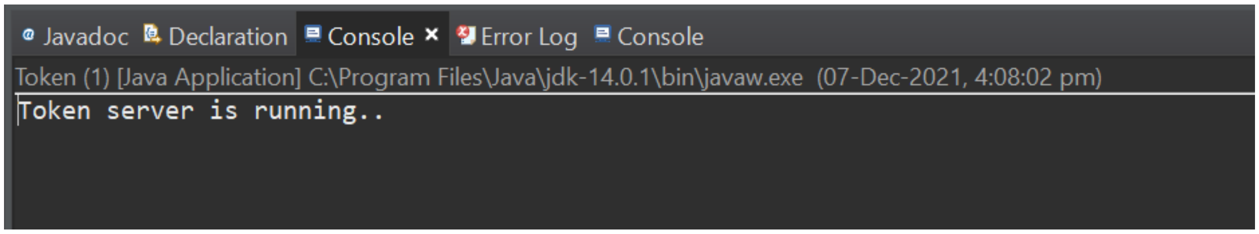
```
<terminated> Client (3) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:08:16 pm – 4:10:21 pm)
Connected to server 0

Enter account number:123456
Enter password:password1
Choices:
1.Check Balance
2.Transfer Money
Enter choice:
2
Enter account number to credit:456789
Enter amount to transfer:200
Server time 1638873621103
Round Trip Time 1
New Client time 1638873621104

Transfer Successful
New Balance:1800.0
```



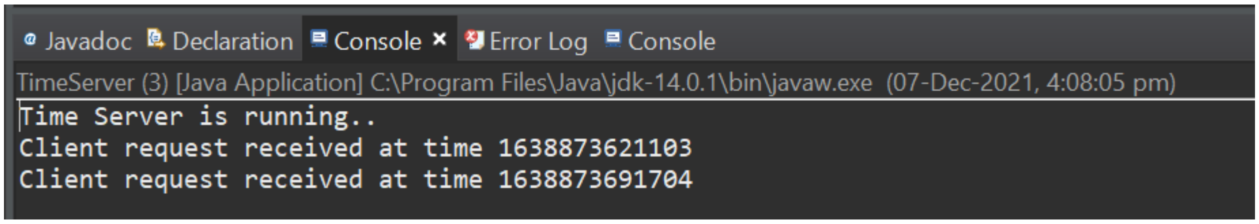
```
<terminated> Client (3) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:11:19 pm – 4:11:31 pm)
Connected to server 1

Enter account number:456789
Enter password:password2
Choices:
1.Check Balance
2.Transfer Money
Enter choice:
1
Server time 1638873691704
Round Trip Time 0
New Client time 1638873691712

Balance: Rs.3200.0
```



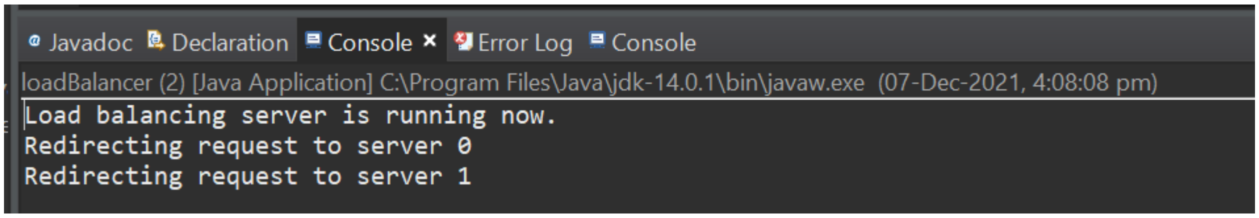
```
Server (4) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:08:11 pm)
3 servers are running now
Transfer request received for account number 123456
Transfer to credit account number 456789
Datastore not accessible
Cannot access datastore 1
Trying to access datastore 2
No owner
Got token
Datastore not accessible
Cannot access datastore 1
Trying to access datastore 2
Datastore not accessible
Cannot access datastore 1
Trying to access datastore 2
Datastore not accessible
Cannot access datastore 1
Trying to access datastore 2
```



```
Token (1) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:08:02 pm)
Token server is running..
```



```
TimeServer (3) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:08:05 pm)
Time Server is running..
Client request received at time 1638873621103
Client request received at time 1638873691704
```



```
loadBalancer (2) [Java Application] C:\Program Files\Java\jdk-14.0.1\bin\javaw.exe  (07-Dec-2021, 4:08:08 pm)
Load balancing server is running now.
Redirecting request to server 0
Redirecting request to server 1
```

Conclusion

1. Data replication is the process in which the data is copied at multiple locations (Different computers or servers) to improve the availability of data.
2. A distributed system maintains copies of its data on multiple machines in order to provide high availability and scalability.
3. When an application makes a change to a data item on one machine, that change has to be propagated to the other replicas. This is called data Consistency.
4. We implemented data consistency and data replication in our banking application. We created 2 data stores, if one of the datastore is not accessible then the data will be retrieved from the second datastore.