223P

Final Project

Vidushee Amoli
Gaurav Ghati

Hospital Management System – Schema

Billing
billing_id
patient_id
appointment_id
total_amount
credit_card_details
payment_date date

Medical_Record
record_id
patient_id
appointment_id
diagnosis varchar
medication
notes

Appointment
appointment_id
patient_id
doctor_id
appointment_date
appointment_time
time

Patient
patient_id
first_name
age

Doctor
doctor_id
first_name
specialization

Partitioning Logic

Server 1 and 2

Doctors, Patients, Appointments

- Information geographically distributed between the 2 servers.

Server 3

Billings

- Separate server for enhanced security measures
- Can be outsourced to third party vendors

Server 4

Medical_Records

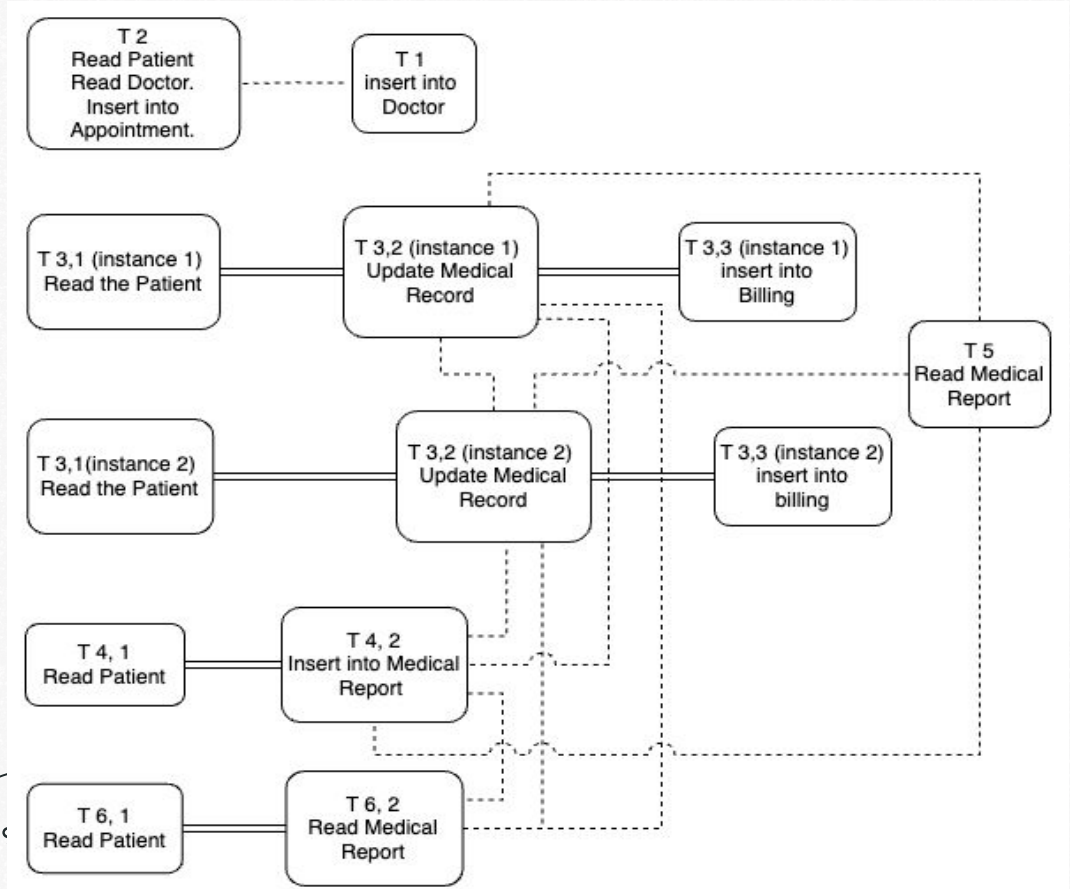
- Support future OLAP queries for medical research and studies.

Understanding the transactions

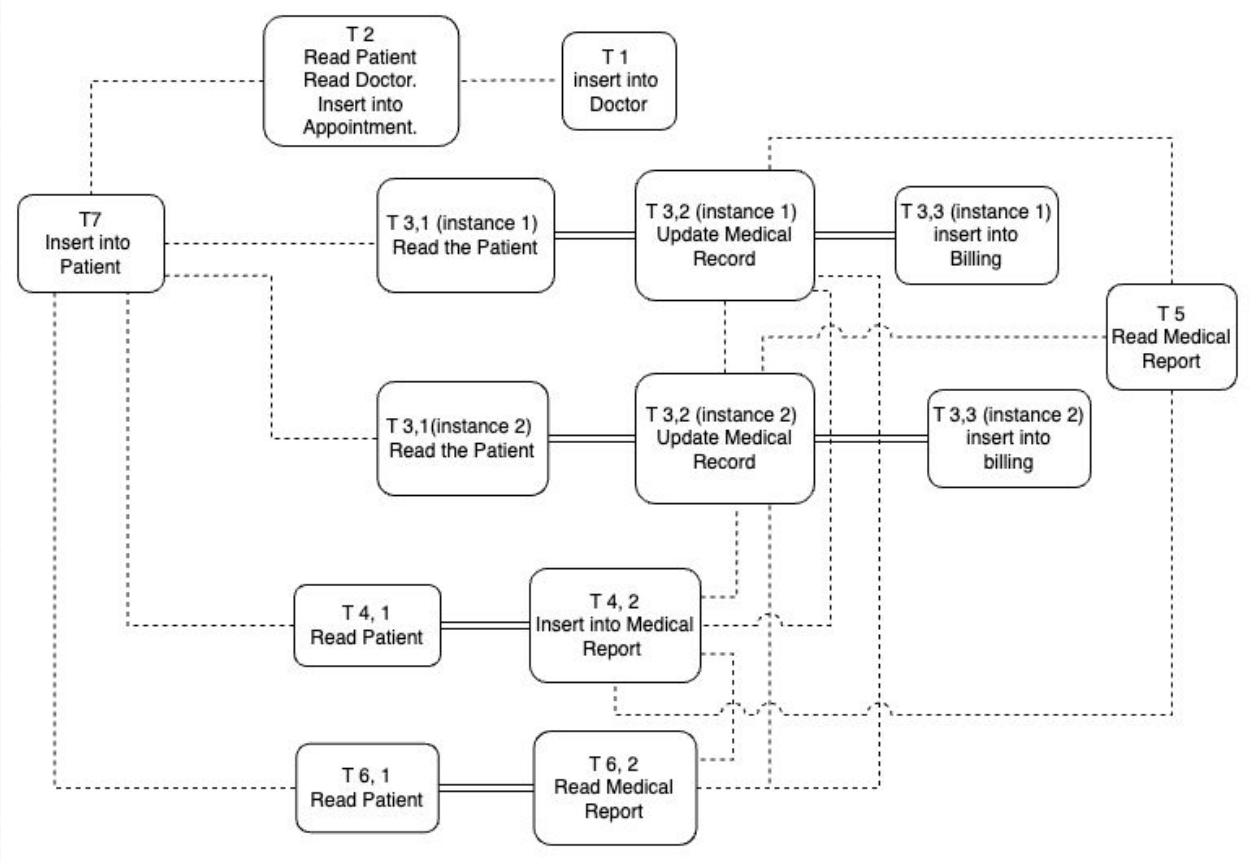
1) Adding a doctor
2) Scheduling an appointment
3) Discharging a patient
4) Adding a medical report after the appointment
5) Query all medical reports for Tuberculosis patients for a medical survey
6) Get a patient's past medical records
7) Adding a patient

SC Diagram with the first 6 txns

1) Adding a doctor
2) Scheduling an appointment
3) Discharging a patient
4) Adding a medical report after the appointment
5) Query all medical reports for a specific Disease
6) Get a patient's past medical records
7) Adding a patient



SC Diagram – including 7th txn



Implementation

```
const WebSocket = require('ws');

async function run() {
    const servers = new WebSocket('ws://localhost:8081');
    const server_report = new WebSocket('ws://localhost:8084');

    const sendMessage = (ws, server, message) => {
        ws.on('open', () => {
            ws.send(message);
            console.log(`Message sent to ${server}: ${message}`);
        });
    };

    // T1: Adding a doctor, Name: John, Specialization: Neurology
    sendMessage(servers, 'Geo-Server', 'T1;John;Neurology');

    // T7: Adding a Patient, Name: Sam
    sendMessage(servers, 'Geo-Server', 'T7;Sam');

    // T2: Scheduling an Appointment, Patient: Sam, Doctor: John
    sendMessage(servers, 'Geo-Server', 'T2;Sam;John');

    // T4: Adding a medical report after the appointment, Patient: Sam
    sendMessage(servers, 'Geo-Server', 'T4;Sam');

    // T3: Discharging a Patient: Sam
    sendMessage(servers, 'Geo-Server', 'T3;Sam');

    // T6: Get a patient's past medical records
    sendMessage(servers, 'Geo-Server', 'T6;Sam');

    // T5: Query all medical reports for a specific Disease
    sendMessage(server_report, 'Server - Medical report', 'T5;Tuberculosis');
}

run();
```

- Inner Ordering
- Origin Ordering

```
sync function execute() {
    ws_server.on('connection', (ws) => {
        ws.on('message', (message) => {

            // for each message we get from Web socket
            if(priority_queue.size() < 3) {
                priority_queue.enqueue(message);
            } else {
                while(priority_queue.size() > 0) {
                    processMessage(priority_queue.top());
                    priority_queue.pop();
                }
            }

        });
    });

    console.log('WebSocket Server Billing is running on ws://localhost:8083');
```

Executing Hops

```
async function processMessage(message) {
    console.log(`Server Report received message => ${message}`);
    var args = code.toString().split(';');

    if(args[0] == 'T3') {
        console.log("T3 starts executing on Report Server");
        var sql = "UPDATE MEDICAL_RECORD SET NOTES='Recovered' WHERE PATIENT_ID=" + args[1];
        await conn.promise().query(sql);
        prams = 'T3;' + args[1] + ';' + args[2];

        // Calling the next server with the given parameters and priority for Origin Ordering.
        server_billing.send(param);

        console.log("T3 Completed on Report Server");

    } else if(args[0] == 'T4') {
        console.log("T4 starts executing on Report Server");
        sql = "INSERT INTO MEDICAL_RECORD(patient_id, appointment_id, diagnosis, medication, notes) "+
                "VALUES ("+ "1234" +", 1, 'Covid-19', 'Covid-Vaccine', 'Consilt after one week of medication');";
        await conn.promise().query(sql);
        console.log("T4 Completed on Report Server");

    } else if(args[0] == 'T6') {
        console.log("T6 starts executing on Report Server");
        sql = "INSERT INTO MEDICAL_RECORD(patient_id, appointment_id, diagnosis, medication, notes) "+
                "VALUES ("+ "1234" +", 1, 'Tuberculosis', 'vaccine', 'Consult after one week of medication');";
        await conn.promise().query(sql);
        console.log("T6 Completed on Report Server");

    } else if(args[0] == 'T5') {
        console.log("T5 starts executing on Report Server");
        sql = "SELECT * from MEDICAL_RECORD WHERE diagnosis='Tuberculosis';"
        var result = await conn.promise().query(sql);
        console.log(result);
        console.log("T5 Completed on Report Server");
    }
}
```

Output Logs

Main User request Log

```
(base) gaurav@Gauravs-MacBook-Air trans project % node userInterface.js
Message sent to Server - Medical report: T5;Tuberculosis
Message sent to Geo-Server: T1;John;Neurology
Message sent to Geo-Server: T7;Sam
Message sent to Geo-Server: T2;Sam;John
Message sent to Geo-Server: T4;Sam
Message sent to Geo-Server: T3;Sam
Message sent to Geo-Server: T6;Sam
```

Server 3 - Billing

```
~/Desktop/trans project — node server3_billing.js
(base) gaurav@Gauravs-MacBook-Air trans project % node server3_billing.js
WebSocket Server Billing is running on ws://localhost:8083
Connected to the Billing Database
Server Report received message => T3
T3 started executing on Billing server
T3 Completed on Billing Server
```

server 1 and 2

```
~/Desktop/trans project — node server3_billing.js
(base) gaurav@Gauravs-MacBook-Air trans project % node servers.js
WebSocket server 1 is running on ws://localhost:8081
Connected to the Server 1 database
Server 1 received message => T1;John;Neurology
T1 started on server 1!
Server 1 received message => T7;Sam
T7 started on server 1!
Server 1 received message => T2;Sam;John
T2 started on server 1!
Server 1 received message => T4;Sam
T4 started on server 1!
Server 1 received message => T3;Sam
T3 started on server 1!
Server 1 received message => T6;Sam
T6 started on server 1!
T1 completed on server 1
T7 completed on server 1
[T2: Read Patient] Patient ID: 500128
[T4] Sending Request to Report Server
[T4: Read Patient] Patient ID: 500128
T4 completed on server 1
[T3: Read Patient] Patient ID: 500128
[T3] Sending Request to Report Server
T3 completed on server 1
[T6: Read Patient] Patient ID: undefined
[T6] Sending Request to Report Server
T6 completed on server 1
[T2: Read Doctor] Doctor ID: 4
T2 completed on server 1
```

server 4 - Medical Report

```
(base) gaurav@Gauravs-MacBook-Air trans project % node server4_reports.js
WebSocket server Report is running on ws://localhost:8084
Connected to Report database.
Server Report received message => T5;Tuberculosis
T5 starts executing on Report Server
Server Report received message => T4;500128;1
T4 starts executing on Report Server
Server Report received message => T3;500128
T3 starts executing on Report Server
Server Report received message => T6;500128;1
T6 starts executing on Report Server
```

Using direct edges between siblings

Q5.a

- No, It won't be correct to consider directed edges as SC-Cycles.

With a Transactions of at most 2 hops, if we inforce inner and origin Ordering then we can address this issue.

For Example, T1H1, T1H2, T2H1, T2H2

We have make sure:

- Inner ordering: (T1H1 before T1H2) and (T2H1 before T2H2)
- Origin ordering: (T1H1 before T2H1) and (T1H2 before T2H2)

Thank you