Database Systems Project Design Report

Patient Medical Treatment Tracking System

26.11.2018

Project URL: https://segocago.github.io/CS353_Database_Project/

Project Group No: 6

Burak Erkılıç - 21501035

Çağatay Sel - 21502938

Kaan Kıranbay - 21501103

Mert Saraç - 21401480

Course Teacher: Shervin R. Arashloo - Course TA: Arif Usta

Table of Contents

1. Revised E/R Model

After getting feedback from the teaching assistant, we revised our E/R diagram and made 8 changes to it according to the feedback:

1. We changed “emergency_contact” entity to a weak entity of “patient” entity. And changed its participation in “relative” relationship as total participation.
2. We changed “patient” and “emergency_contact” entities’ cardinality constraint of “ relative” relationship from many-to-many to one-to-many relationship where A patient is relative with several (including 0) “emergency_contact”s.
3. We added underlines to our primary keys.
4. We added a new entity called “prescriptions” that have relationships with “drug” and “ examination” entities to show what drugs are given to a patient after an examination.
5. We added a new relationship as “examination_result”. This helps us to determine what happens after an examination is done which can be tests, treatments or prescriptions.
6. We added total participation to our “works_as_doctor” relationship for both of the entities: “doctor” and “hospital”. Same is done for “work_as_pharmacist” and “stores” relationships as well. Furthermore, we changed one-to-many relationship “stores” to many-to-many relationship.
7. We added an aggregation that contains“patient”, “doctor”, and “examination”. This aggregation is added for the relationship(“rate_for”) between these entities and a new entity called “rating” to give patients the ability of evaluation of their doctors and their eexaminations.
8. We removed the foreign key “test_hospital_name” from our “test” entity. We made a many-to-many relationship called “test_executed_in” with total participation from test.

We also made the following 4 changes to our E/R diagram to make it more practical:

1. We added an entity called “user” and made it parent of “pharmacist”, “doctor”, and “patient” entities. This inheritance is an overlapping generalization.
2. We added an attribute called “vaccine_id” to our “vaccine” entity. We also removed the attribute called “vaccine_date” from our “vaccine” entity and made it a relationship attribute of “vaccinated” relationship.
3. We added an attribute called “hospital_executive_doctor_id” to our “hospital” entity.
4. We removed unnecessary attributes from our entities.

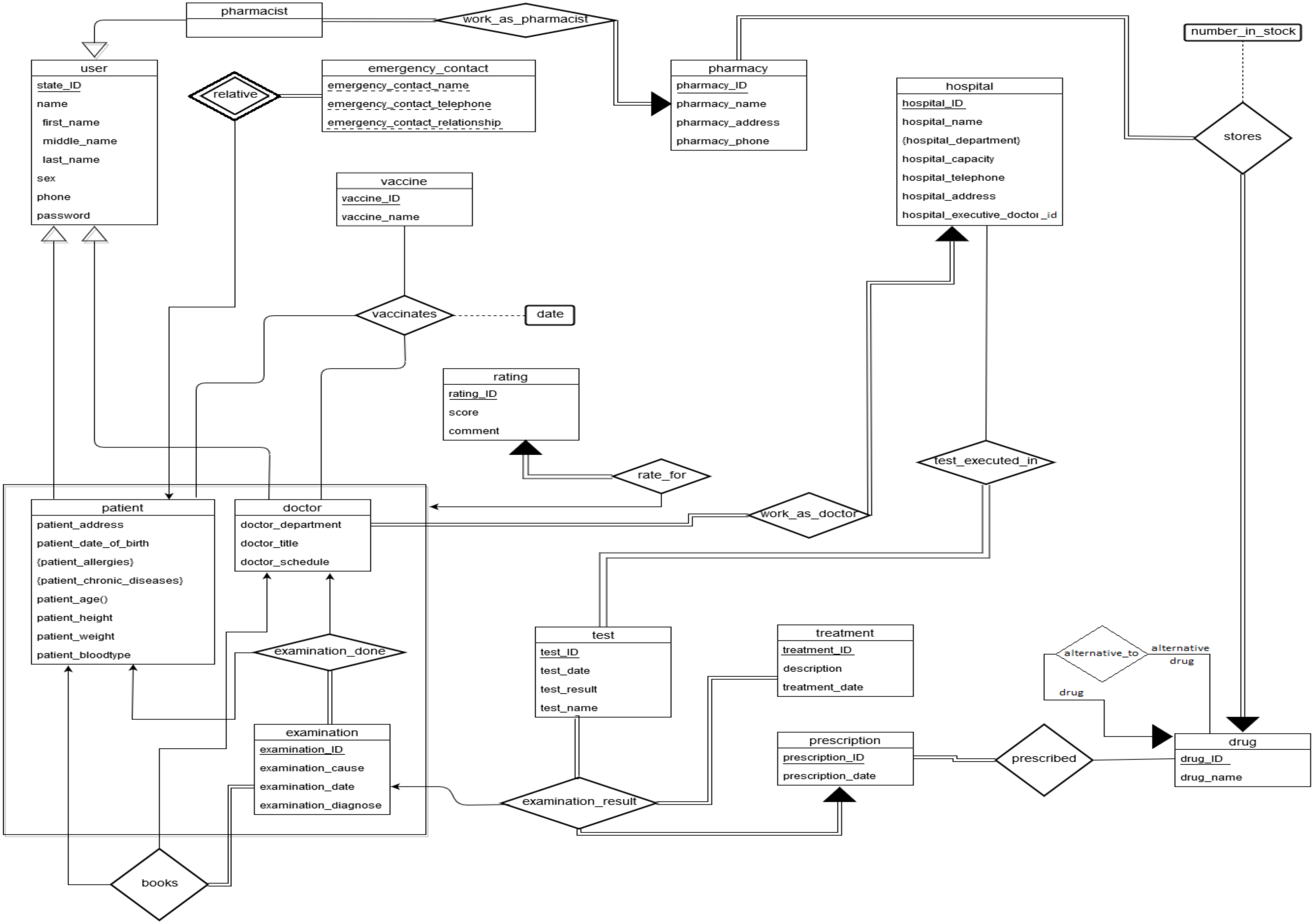


Figure 1: Revised E/R Model of Project's Database

2. Relation Schemas

2.1 User

Relational Model:

user(state_ID, first_name, middle_name, last_name, sex, phone, password)

Functional Dependencies:

state_ID → first_name, middle_name, last_name, sex, phone, password

Candidate Keys:

{ (state_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE user(
        state_ID        char(11) PRIMARY KEY,
        first_name      varchar(20),
        middle_name     varchar(20),
        last_name       varchar(20),
        sex             varchar(20),
        phone           varchar(100),
        password        varchar(40) NOT NULL);
```

2.2 Pharmacist

Relational Model:

pharmacist (state_ID)

Functional Dependencies:

none

Candidate Keys:

{ (state_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE pharmacist(
        state_ID        char(11) PRIMARY KEY,
        FOREIGN KEY (state_ID) references user);
```

2.3 Patient

Relational Model:

patient (state_ID, patient_adress, patient_date_of_birth, patient_allergies, patient_chronic_diseases, patient_height, patient_weight, patient_bloodtype)

Functional Dependencies:

state_ID → patient_adress, patient_date_of_birth, patient_allergies, patient_chronic_diseases, patient_height, patient_weight, patient_bloodtype

Candidate Keys:

{ (state_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE patient(
        state_ID                    char(11) PRIMARY KEY,
        patient_adress              varchar(100),
        patient_date_of_birth       date NOT NULL,
        patient_allergies           varchar(100),
        patient_chronic_diseases    varchar(100),
        patient_height              numeric(3,2),
        patient_weight              numeric(3,2),
        patient_bloodtype           varchar(20),
        FOREIGN KEY (state_ID) references user);
```

2.4 Doctor

Relational Model:

doctor (state_ID , doctor_department, doctor_title, doctor_schedule)

Functional Dependencies:

state_ID → doctor_department, doctor_title, doctor_schedule

Candidate Keys:

{ (state_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE doctor(
        state_ID                char(11) PRIMARY KEY,
        doctor_department       varchar(40) NOT NULL,
        doctor_title            varchar(40) NOT NULL,
        doctor_schedule         varchar(400) NOT NULL,
        FOREIGN KEY (state_ID) references user);
```

2.5 Examination

Relational Model:

examination (examination_ID, examination_cause, examination_date, examination_diagnose)

Functional Dependencies:

examination_ID→ examination_cause, examination_date, examination_diagnose

Candidate Keys:

{ (examination_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE examination(
        examination_ID              int PRIMARY KEY AUTO_INCREMENT,
        patient_state_ID            char(11),
        doctor_state_ID             char(11),
        examination_cause           varchar(400) NOT NULL,
        examination_date            timestamp NOT NULL,
        examination_diagnose        varchar(400) NOT NULL);
```

2.6 Rating

Relational Model:

rating (rating_ID, score, comment)

Functional Dependencies:

rating_ID → score, comment

Candidate Keys:

{ (rating_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE rating(
        rating_ID       int PRIMARY KEY AUTO_INCREMENT,
        score           int,
        comment         varchar(400),
        check (score between 0 and 5));
```

2.7 Test

Relational Model:

test(test_ID, test_date, test_result, test_name)

Functional Dependencies:

test_ID → test_date, test_result, test_name

Candidate Keys:

{ (test_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE test(
        test_ID         int PRIMARY KEY AUTO_INCREMENT,
        test_date       date,
        test_result     varchar(400),
        test_name       varchar(100));
```

2.8 Treatment

Relational Model:

treatment (treatment_ID, treatment_description, treatment_date)

Functional Dependencies:

treatment_ID → treatment_description, treatment_date

Candidate Keys:

{ (treatment_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE treatment(
        treatment_ID                int PRIMARY KEY AUTO_INCREMENT,
        treatment_description       varchar(400),
        treatment_date              date);
```

2.9 Prescription

Relational Model:

prescription (prescription_ID, prescription_date)

Functional Dependencies:

prescription_ID → prescription_date

Candidate Keys:

{ (prescription_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE prescription(
        prescription_ID         int PRIMARY KEY AUTO_INCREMENT,
        prescription_date       date);
```

2.10 Drug

Relational Model:

drug(drug_ID, drug_name)

Functional Dependencies:

drug_ID → drug_name

Candidate Keys:

{ (drug_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE drug(
        drug_ID         int PRIMARY KEY AUTO_INCREMENT,
        drug_name       varchar(200));
```

2.11 Pharmacy

Relational Model:

pharmacy (pharmacy_ID, pharmacy_name, pharmacy_address, pharmacy_phone)

Functional Dependencies:

pharmacy_ID → pharmacy_name, pharmacy_address, pharmacy_phone

Candidate Keys:

{ (pharmacy_ID**) }**

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE pharmacy(
        pharmacy_ID           int PRIMARY KEY AUTO_INCREMENT,
        pharmacy_name         varchar(100),
        pharmacy_address      varchar(100),
        pharmacy_phone        varchar(100));
```

2.12 Hospital

Relational Model:

hospital(hospital_ID, hospital_name, hospital_capacity, hospital_telephone, hospital_address, hospital_executive_doctor_id)

Functional Dependencies:

hospital_ID → hospital_name, hospital_capacity, hospital_telephone, hospital_address, hospital_executive_doctor_id

Candidate Keys:

{ (hospital_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE hospital(
        hospital_ID                         int PRIMARY KEY AUTO_INCREMENT,
        hospital_name                       varchar(200),
        hospital_capacity                   int,
        hospital_telephone                  varchar(100),
        hospital_address                    varchar(200),
        hospital_executive_doctor_id        char(11),
        FOREIGN KEY (hospital_executive_doctor_id) references doctor(state_ID));
```

2.13 Vaccine

Relational Model:

vaccine(vaccine_ID, vaccine_name)

Functional Dependencies:

vaccine_ID → vaccine_name

Candidate Keys:

{ (vaccine_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE vaccine(
        vaccine_ID              int PRIMARY KEY AUTO_INCREMENT,
        vaccine_name            varchar(100));
```

2.14 Emergency Contact

Relational Model:

emergency_contact (state_ID, emergency_contact_name, emergency_contact_telephone, emergency_contact_relationship)

Functional Dependencies:

state_ID, emergency_contact_name, emergency_contact_telephone, emergency_contact_relationship → state_ID, emergency_contact_name, emergency_contact_telephone, emergency_contact_ relationship

Candidate Keys:

{ (state_ID, emergency_contact_name, emergency_contact_telephone, emergency_contact_relationship) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE emergency_contact(
        state_ID                          char(11),
        emergency_contact_name            varchar(100),
        emergency_contact_telephone       varchar(100),
        emergency_contact_relationship    varchar(100),
        PRIMARY KEY (state_ID, emergency_contact_name, emergency_contact_telephone,
emergency_contact_relationship),
        FOREIGN KEY (state_ID) references patient);
```

2.15 Hospital Departments

Relational Model:

hospitalDepartment (hospital_ID, hospital_department)

Functional Dependencies:

None

Candidate Keys:

{ (hospital_ID, hospital_department)}

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE hospitalDepartment(
        hospital_ID             int AUTO_INCREMENT,
        hospital_department     varchar(40),
        PRIMARY KEY (hospital_ID, hospital_department),
        FOREIGN KEY (hospital_ID) references hospital);
```

2.16 Patient Allergies

Relational Model:

patientAllergies (state_ID, allergy_name)

Functional Dependencies:

state_ID, allergy_name→state_ID, allergy_name

Candidate Keys:

{ (state_ID,allergy_name) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE patientAllergies(
        state_ID        char(11),
        allergy_name    varchar(100),
        PRIMARY KEY (state_ID,allergy_name),
        FOREIGN KEY (state_ID) references patient);
```

2.17 Patient Chronic Diseases

Relational Model:

patientChronicDiseases (state_ID, chronic_disease)

Functional Dependencies:

state_ID, chronic_disease→ state_ID, chronic_disease

Candidate Keys:

{ (state_ID, chronic_disease) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE patientChronicDiseases(
        state_ID                char(11),
        chronic_disease         varchar(100),
        PRIMARY KEY (state_ID,chronic_disease),
        FOREIGN KEY (state_ID) references patient);
```

2.18 Examination Done

Relational Model:

examinationDone (patient_state_ID, doctor_state_ID, examination_ID)

Functional Dependencies:

No non-trivial functional dependency.

Candidate Keys:

{ (patient_state_ID, doctor_state_ID, examination_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE examinationDone (
        patient_state_ID        char(11),
        doctor_state_ID         char(11),
        examination_ID          char(11),
        PRIMARY KEY (patient_state_ID, doctor_state_ID, examination_ID),
        FOREIGN KEY (patient_state_ID) references patient(state_ID),
        FOREIGN KEY (doctor_state_ID) references doctor(state_ID));
```

2.19 Books

Relational Model:

books (state_ID, examination_ID, doctor_ID)

Functional Dependencies:

none

Candidate Keys:

{ (state_ID, examination_ID,doctor_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE books (
        state_ID                char(11),
        examination_ID          char(11)
        doctor_ID               char(11),
        PRIMARY KEY (state_ID, examination_ID, doctor_ID),
        FOREIGN KEY (state_ID) references patient,
        FOREIGN KEY (examination_ID) references examination);
```

2.20 Vaccinates

Relational Model:

vaccinate (vaccine_ID, patient_state_ID, doctor_state_ID, date)

Functional Dependencies:

vaccine_ID, patient_state_ID, doctor_state_ID → date

Candidate Keys:

{ (vaccine_ID, patient_state_ID, doctor_state_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE vaccinates(
        vaccine_ID              int,
        patient_state_ID        char(11),
        doctor_state_ID         char(11),
        date                    date,
        PRIMARY KEY (vaccine_ID, patient_state_ID, doctor_state_ID),
        FOREIGN KEY (vaccine_ID) references vaccine,
        FOREIGN KEY (patient_state_ID) references patient(state_ID),
        FOREIGN KEY (doctor_state_ID) references doctor(state_ID));
```

2.21 Works as Pharmacist

Relational Model:

worksAsPharmacist (state_ID, pharmacy_ID)

Functional Dependencies:

none

Candidate Keys:

{ (state_ID, pharmacy_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE worksAsPharmacist(
        state_ID            char(11),
        pharmacy_ID         int,
        PRIMARY KEY (state_ID, pharmacy_ID),
        FOREIGN KEY (state_ID) references pharmacist,
        FOREIGN KEY (pharmacy_ID) references pharmacy);
```

2.22 Rate for

Relational Model:

rateExamination (rating_ID, patient_state_ID, doctor_state_ID, examination_ID)

Functional Dependencies:

none

Candidate Keys:

{ (rating_ID, patient_state_ID, doctor_state_ID, examination_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE rateExamination(
        rating_ID               int,
        patient_state_ID        char(11),
        doctor_state_ID         char(11),
        examination_ID          int,
        PRIMARY KEY (rating_ID, patient_state_ID, doctor_state_ID, examination_ID),
        FOREIGN KEY (patient_state_ID) references patient(state_ID),
        FOREIGN KEY (doctor_state_ID) references doctor(state_ID),
        FOREIGN KEY (examination_ID) references examination);
```

2.23 Stores

Relational Model:

stores (pharmacy_ID, drug_ID, number_in_stock)

Functional Dependencies:

pharmacy_ID, drug_ID → number_in_stock

Candidate Keys:

{ (pharmacy_ID, drug_ID, number_in_stock) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE stores(
        pharmacy_ID         int,
        drug_ID             int,
        number_in_stock     int,
        PRIMARY KEY (pharmacy_ID, drug_ID),
        FOREIGN KEY (pharmacy_ID) references pharmacy,
        FOREIGN KEY (drug_ID) references drug);
```

2.24 Works as Doctor

Relational Model:

worksAsDoctor (state_ID, hospital_ID)

Functional Dependencies:

none

Candidate Keys:

{ (state_ID, hospital_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE worksAsDoctor(
        state_ID        char(11),
        hospital_ID     int,
        PRIMARY KEY (state_ID, hospital_ID),
        FOREIGN KEY (state_ID) references doctor,
        FOREIGN KEY (hospital_ID) references hospital(hospital_ID));
```

2.25 Test Executed in

Relational Model:

textExecutedIn (test_ID, hospital_ID)

Functional Dependencies:

none

Candidate Keys:

{ (test_ID, hospital_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE textExecutedIn (
        test_ID         int,
        hospital_ID     int,
        PRIMARY KEY (test_ID, hospital_ID),
        FOREIGN KEY (test_ID) references test,
        FOREIGN KEY (hospital_ID) references hospital);
```

2.26 Examination Result

Relational Model:

examinationResult(examination_ID, test_ID, treatment_ID, prescription_ID)

Functional Dependencies:

none

Candidate Keys:

{ (examination_ID, test_ID, treatment_ID, prescription_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE examinationResult(
        examination_ID      int,
        test_ID             int,
        treatment_ID        int,
        prescription_ID     int,
        PRIMARY KEY (examination_ID, test_ID, treatment_ID, prescription_ID),
        FOREIGN KEY (examination_ID) references examination,
        FOREIGN KEY (test_ID) references test,
        FOREIGN KEY (treatment_ID) references treatment,
        FOREIGN KEY (prescription_ID) references prescription);
```

2.27 Prescribed

Relational Model:

prescribed(prescription_ID, drug_ID)

Functional Dependencies:

none

Candidate Keys:

{ (prescription_ID, drug_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE prescribed(
        prescription_ID         int,
        drug_ID                 int,
        PRIMARY KEY (prescription_ID, drug_ID),
        FOREIGN KEY (prescription_ID) references prescription,
        FOREIGN KEY (drug_ID) references drug);
```

2.28 Alternative to

Relational Model:

alternativeTo(drug_ID, alternative_drug_ID)

Functional Dependencies:

none

Candidate Keys:

{ (drug_ID, alternative_drug_ID) }

Normal Form:

BCNF

Table Definition:

```
CREATE TABLE alternativeTo(
        drug_ID                 int,
        alternative_drug_ID     int,
        PRIMARY KEY (drug_ID, alternative_drug_ID),
        FOREIGN KEY (drug_ID) references drug,
        FOREIGN KEY (alternative_drug_ID) references drug);
```

3. Functional Dependencies and Normalization of Tables

Every functional dependency and every normal form are given in the relation schemas which is Section 2 of this Project Design Report. Every relation is checked in our design if the relation is in Boyce-Codd Normal Form. Since the left side of the functional dependencies in our schemas are foreign keys, they are in BCNF form and does need further decomposition.

4. Functional Components

4.1 Use Cases / Scenarios

4.1.1 Patient

- Patients can only login to the system with their state IDs and their passwords.
- Patients can only view their medical profile which are vaccine history, examinations, diagnoses treatments, prescribed drugs, allergies, test results, chronic diseases.
- Patients can view hospitals and their information with doctors who are working there.
- Patients can book an appointment from doctors.
- Patients can only view and edit their own profile which has emergency contact and profile information.

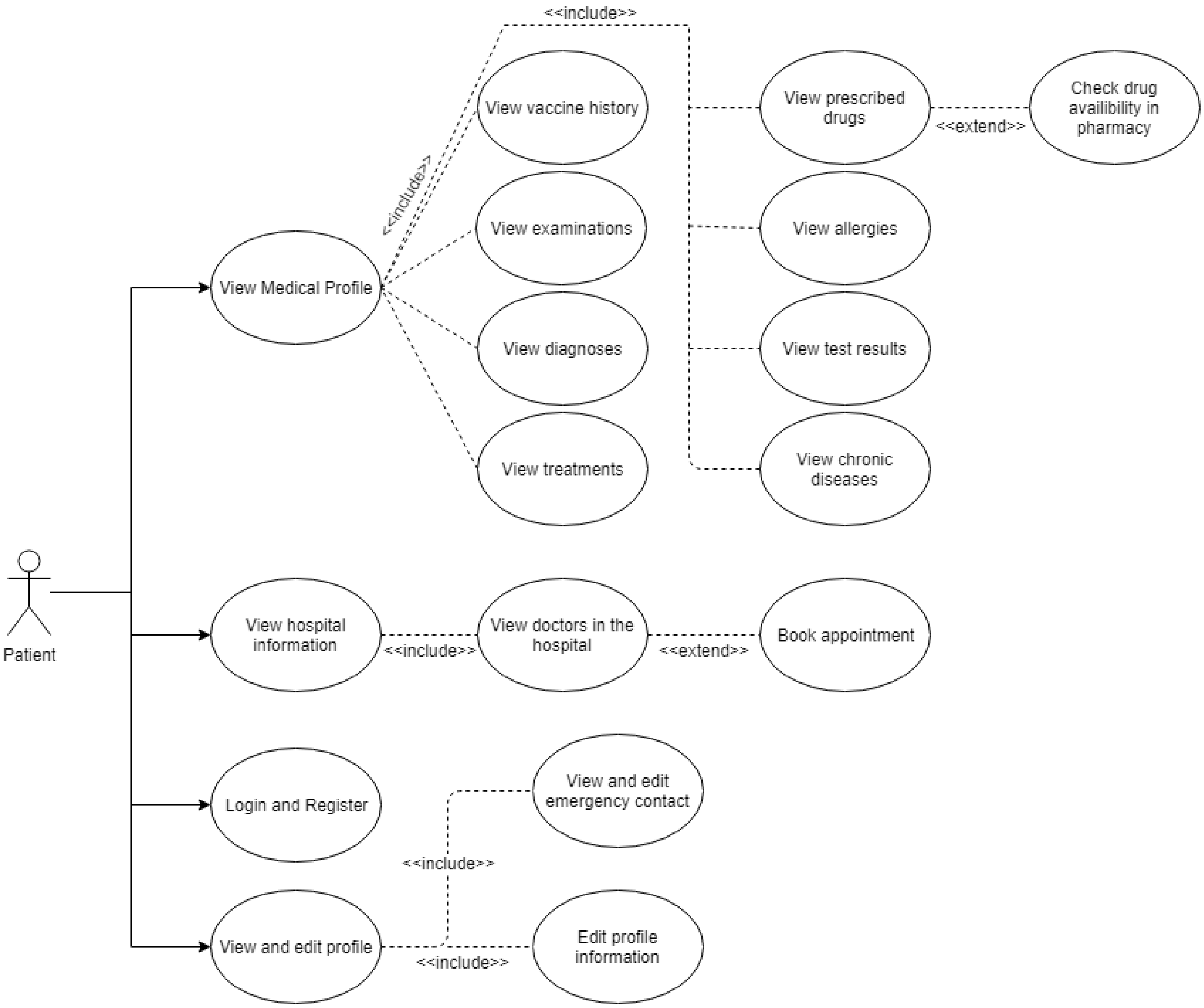


Figure 2: Patients' Use Case Diagram

4.1.2 Doctor

- Doctors can only login to the system with their state IDs and their passwords.
- Doctors will vaccinate a patient in real life then they will add this vaccination of a patient with the information of the date and the name of the vaccine with the state ID's of the patient.
- Doctors can add the examination result of a patient with the state ID's of the patient..
- Doctors can add the prescription of a patient after an examination with the state ID's of the patient.
- Doctors can add the treatment of a patient after an examination.
- Doctors can add the test results after a test is done after the examination.
- Doctors can add diagnoses such as allergies or chronic disease of a patient.
- Doctors can view hospital informations.
- Doctors can view a patient's medical information.
- Doctors can view their schedule.

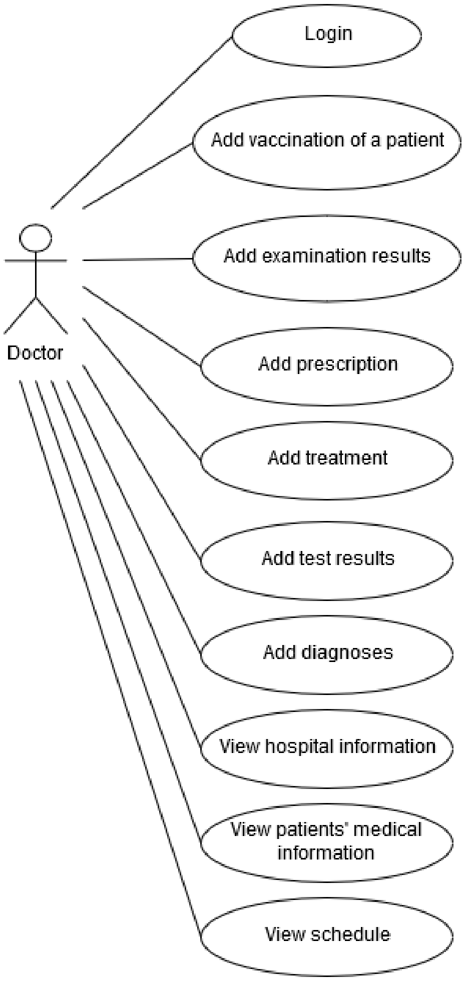


Figure 3: Doctors' Use Case Diagram

4.1.3 Pharmacist

- Pharmacists can register and login
- Pharmacists can register their new pharmacies to the system.
- Pharmacists can manage the pharmacy stock such as adding new drugs or removing drugs from the pharmacy.
- Pharmacists can view patients' prescriptions.
- Pharmacists can edit their pharmacys' information.
- Pharmacists can add or remove other pharmacists from their pharmacies.
- Pharmacists can check whether there are no drugs left in the store or not, and can check the alternative drugs for that drug.

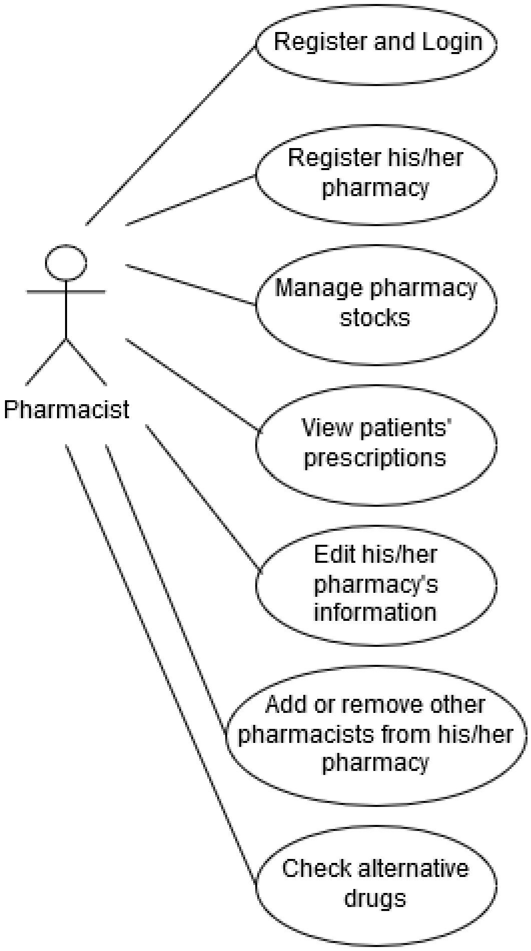

Register and Login
Register his/her pharmacy
Manage pharmacy stocks
Pharmacist
View patients' prescriptions
Edit his/her pharmacy's information
Add or remove other pharmacists from his/her pharmacy
Check alternative drugs

Figure 4: Pharmacists' Use Case Diagram

4.1.4 Executive Doctor

- Executive doctors can register and login.
- Executive doctors can edit the hospital information where they work at.
- Executive doctors can edit doctors’ examination schedules.
- Executive doctors can register new doctors to their hospitals.

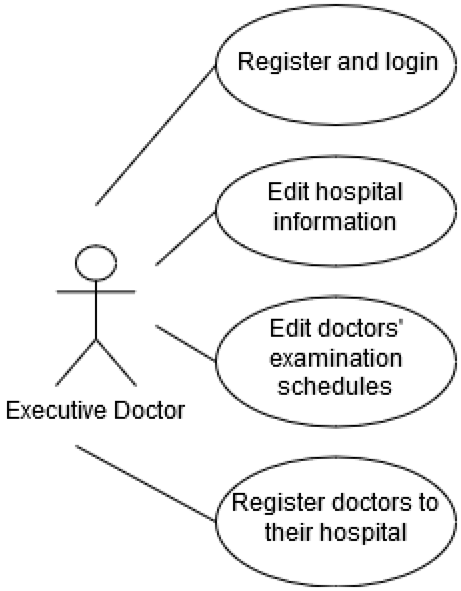

Register and login
Edit hospital information
Edit doctors' examination schedules
Executive Doctor
Register doctors to their hospital

Figure 5: Executive Doctors’ Use case Diagram

4.2 Algorithms

Since our project is mostly based on database manipulations, there are not any domain specific algorithm that will be used in the project. Application will do database queries in order to add, update or get information from the database and the information that database contains will be displayed to users. Our algorithms will be basically the queries that we write to interact with the database.

4.3 Data Structures

We have used char, varchar, date and int domains in the MySql tables. There could also be sorted array or sorted linked list structures in server side or in client side to display lists in order.

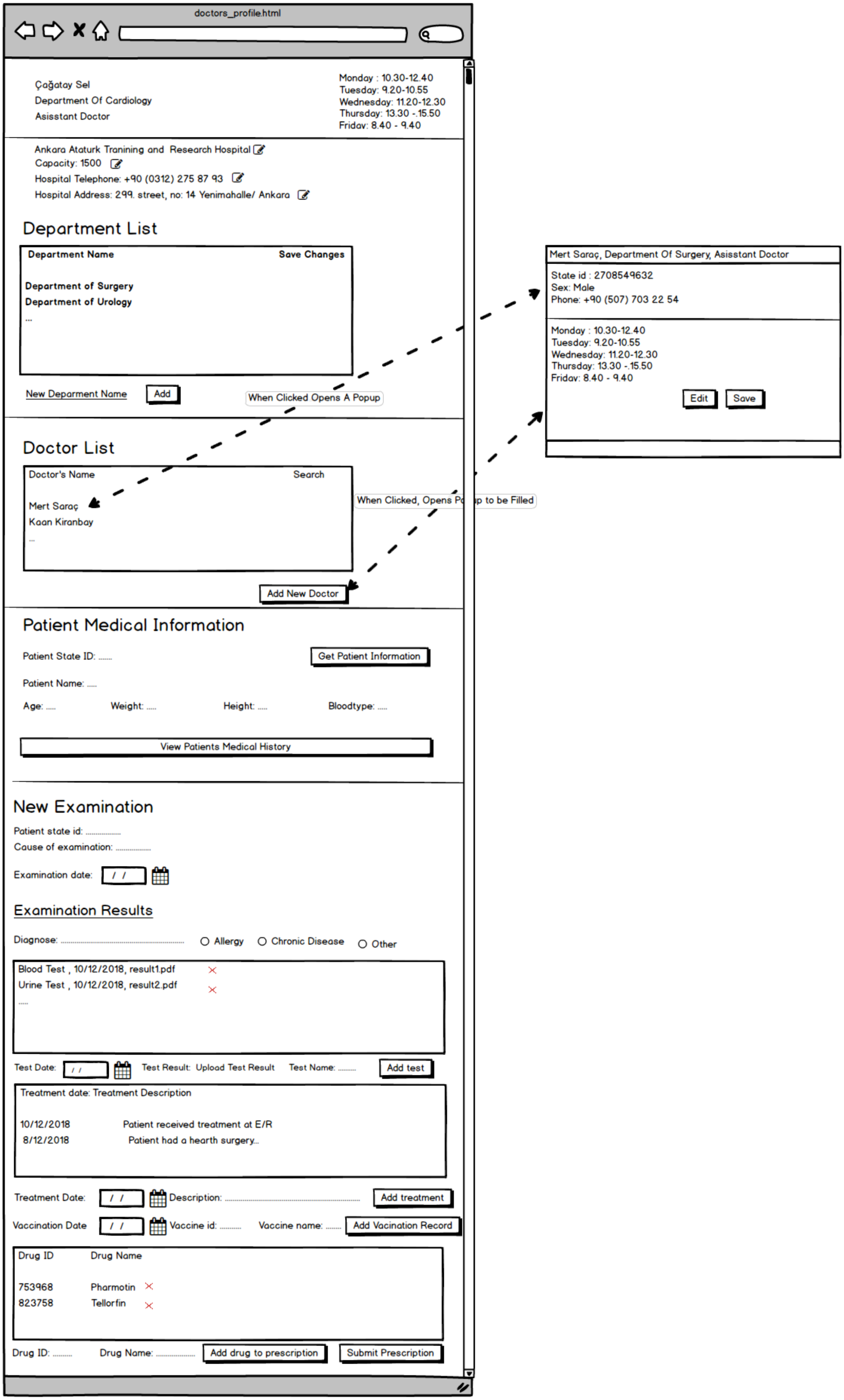
5. User Interface Design and Corresponding SQL Statements

5.1 Doctors' Page

This is page which doctors who have already registered to system will see when they login. First to sections in which hospital information and doctor list is displayed will be seen only by the executive doctor. Executive doctor will be able to click on the names of doctors to open an information card as an pop-up. In this pop-up, executive doctor will be able to change the schedule of doctors. Executive doctor will also be able to change or add departments. Other doctors will not see these sections and will not be able to edit hospital information or add new doctors to hospital.

Doctors who are not executive doctor will see their information and the top and then continue with patient medical information section so that they will not be able to change hospital related information. In the patient medical information section, they will be able to request medical information of a patient by providing the state id of the patient. View Patients Medical History button will redirect to the profile page of the patient in which medical history is displayed.

Doctors will be able to register examinations in the new examination section. They will register any diagnoses, test, treatment and prescribed drug in this section.



doctors_profile.html

Çağatay Sel
Department Of Cardiology
Asisstant Doctor

Monday : 10.30-12.40
Tuesday: 9.20-10.55
Wednesday: 11.20-12.30
Thursday: 13.30 -.15.50
Friday: 8.40 - 9.40

Ankara Ataturk Tranining and Research Hospital
Capacity: 1500
Hospital Telephone: +90 (0312) 275 87 93
Hospital Address: 299. street, no: 14 Yenimahalle/ Ankara

Department List

Department Name **Save Changes**

Department of Surgery
Department of Urology
...

New Deparment Name [Add]

When Clicked Opens A Popup

Doctor List

Doctor's Name Search

Mert Saraç
Kaan Kiranbay
...

[Add New Doctor]

When Clicked, Opens Popup to be Filled

Mert Saraç, Department Of Surgery, Asisstant Doctor

State id : 2708549632
Sex: Male
Phone: +90 (507) 703 22 54

Monday : 10.30-12.40
Tuesday: 9.20-10.55
Wednesday: 11.20-12.30
Thursday: 13.30 -.15.50
Friday: 8.40 - 9.40

[Edit] [Save]

Patient Medical Information

Patient State ID: ...... [Get Patient Information]

Patient Name: .....

Age: ..... Weight: ..... Height: ..... Bloodtype: .....

[View Patients Medical History]

New Examination

Patient state id: ....
Cause of examination: ....

Examination date: / /

Examination Results

Diagnose: .... ○ Allergy ○ Chronic Disease ○ Other

Blood Test , 10/12/2018, result1.pdf ×
Urine Test , 10/12/2018, result2.pdf ×
.....

Test Date: / / Test Result: Upload Test Result Test Name: .... [Add test]

Treatment date:	Treatment Description
10/12/2018	Patient received treatment at E/R
8/12/2018	Patient had a hearth surgery...

Treatment Date: / / Description: .... [Add treatment]

Vaccination Date / / Vaccine id: .... Vaccine name: .... [Add Vacination Record]

Drug ID	Drug Name
753968	Pharmotin ×
823758	Tellorfin ×

Drug ID: .... Drug Name: .... [Add drug to prescription] [Submit Prescription]

Figure 6: Doctor’s Page

SQL Statements

Retrieving Doctor’s Information

```
SELECT doctor_department, doctor_title, doctor_schedule
FROM doctor
WHERE doctor.state_ID = @state_ID;
```

Retrieving Hospital Information

```
SELECT hospital_ID hospital_name, hospital_capacity, hospital_telephone, hospital_address
FROM hospital
WHERE hospital_executive_doctor_ID= @state_ID;
```

Retrieving Departments

```
SELECT hospital_department
FROM hospitalDepartment
WHERE hospitalDepartment.hospital_ID =@hospital_ID;
```

Adding New Department

```
INSERT INTO hospital_department
VALUES (hospital_ID, new_department);
```

Listing Doctors in Hospital

```
SELECT first_name, middle_name, last_name,sex,phone,password
FROM user
WHERE user.state_ID in  (SELECT state_ID ,
                          FROM workAsDoctor
                          WHERE workAsDoctor.hospital_ID =@ hospital_ID) ;
```

```
SELECT doctor_department, doctor_title,doctor_schedule
FROM doctor
WHERE doctor.state_ID in (SELECT state_ID ,
                          FROM workAsDoctor
                          WHERE workAsDoctor.hospital_ID= @hospital_ID) ;
```

Getting Patient Medical Information

```
SELECT first_name , middle_name, last_name
FROM user
WHERE user.state_ID = @state_ID;
```

```
SELECT patient_weight, patient_height, patient_bloodtype
FROM patient
WHERE patient.state_ID = @state_ID;
```

Adding Vaccination Record

```
INSERT INTO vaccinates
VALUES (@vaccinate_ID , @patient_state_ID, @doctor_state_ID,@date);
```

Adding New Examination

```
INSERT INTO examination
VALUES (@examination_ID, @examination_cause, @examination_date,
@examination_diagnose);
```

```
INSERT INTO test
VALUES (@test_ID,@test_date,@test_result,@test_name);
```

```
INSERT INTO treatment
```

```
VALUES (@treatment_ID,@treatment_description,@treatment_date);



INSERT INTO prescription
VALUES (@prescription_ID,@prescription_date);

INSERT INTO prescribed
VALUES (@prescription_ID, @drug_id);

INSERT INTO examination_result
VALUES (@examination_ID,@test_ID,@treatment_ID,@prescription_ID);

INSERT INTO examination_done
VALUES (@examination_ID, @patient_state_ID,@doctor_state_ID);
```

If patient is diagnosed with any allergy or chronic disease

```
INSERT INTO patientAllergies
VALUES (@state_ID, @allergyName);

INSERT INTO patientChronicDisease
VALUES (@state_ID, @chronicDisease);
```

5.2 Login Page

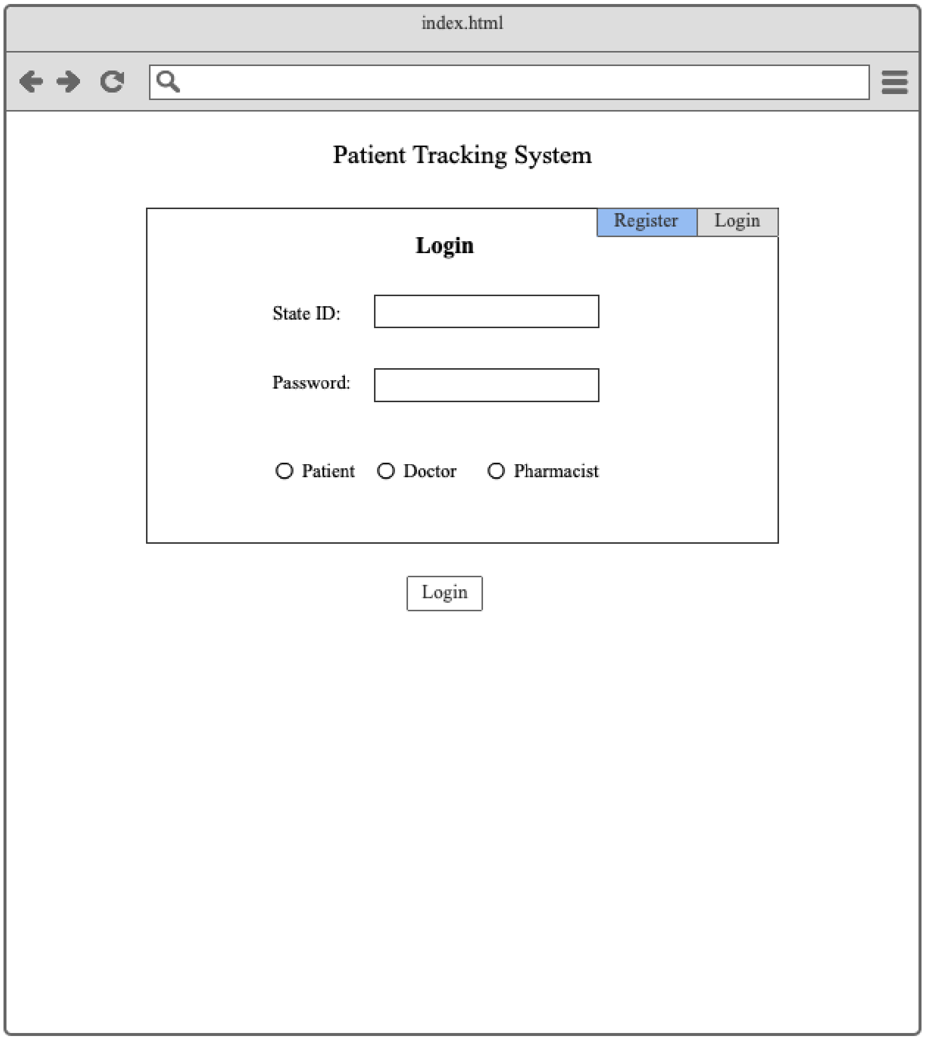


Figure 7: Login Page

In this page, user can login if he/she has already an account. Specifying type of the account (patient, account, executive doctor account or pharmacist account) is needed for login process.

SQL STATEMENTS

```
SELECT *

FROM user

WHERE user.state_ID = @state_ID, user.password = @password;
```

5.3 Register Page

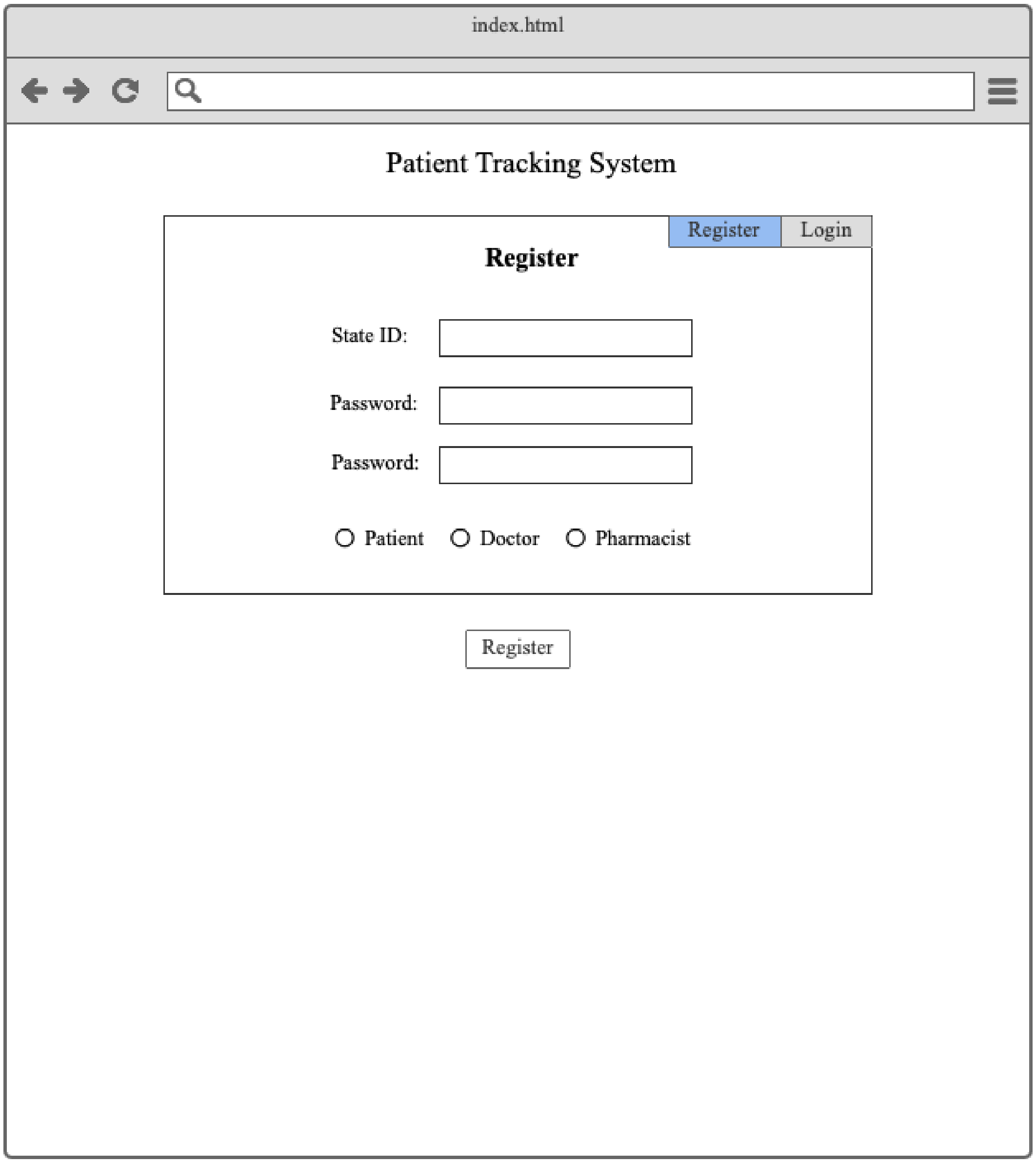


Figure 8: Register Page

If user has no account, he/she can create one easily by selecting register tab. To register, all user needs is entering state-id (TC no.) and password. Password is asked for two times in terms of reduce the likelihood of typo. Specifying type of the account is also needed here.

SQL Statements

Checking If User Exist

SELECT state_id

FROM user

```
WHERE user.state_id = @state_id;
```

Registering a User

```
INSERT INTO user
VALUES (@state_id, NULL , NULL, NULL, NULL ,NULL ,@ password);
```

5.4 Information Page

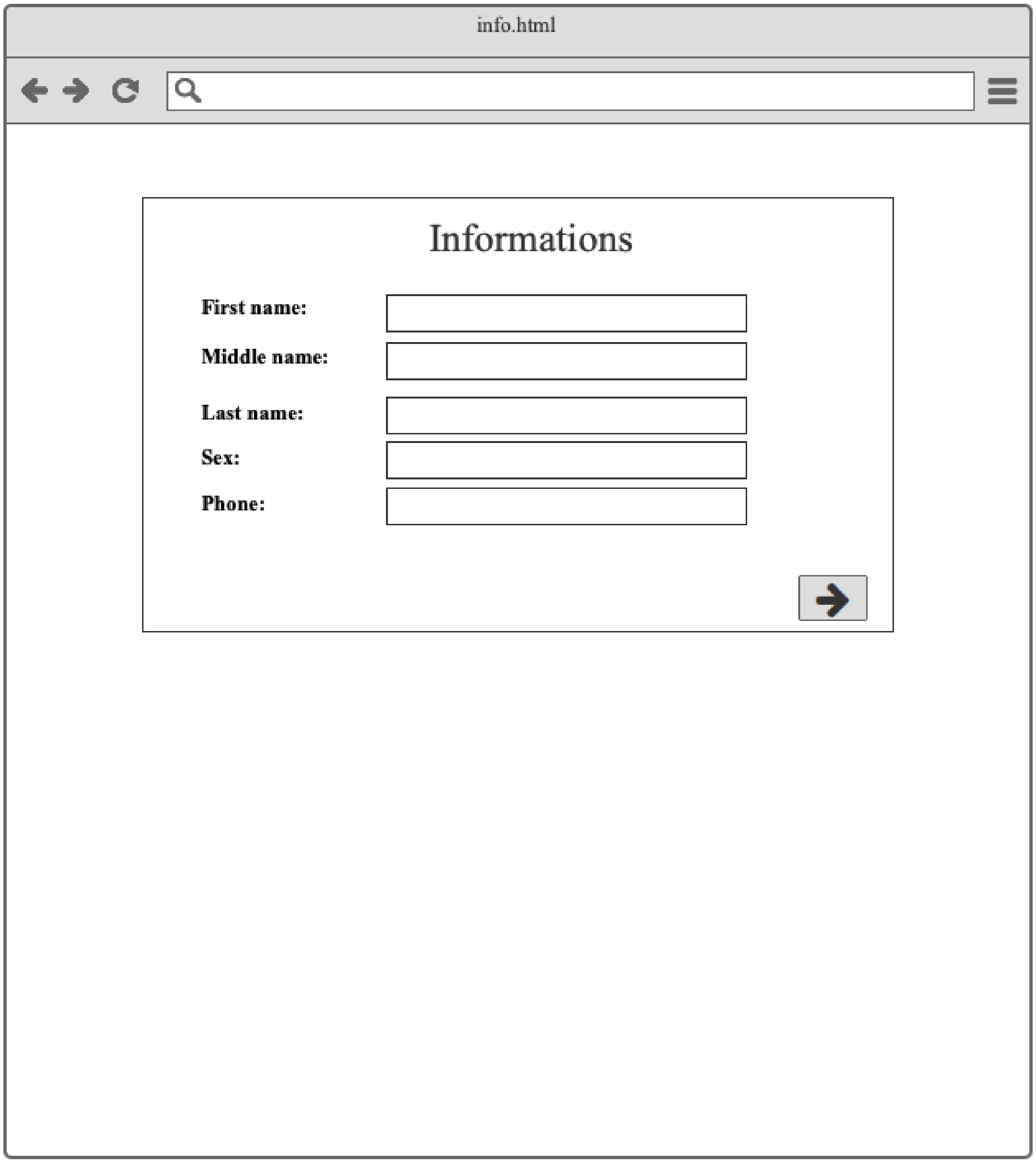


Figure 9: Information Page

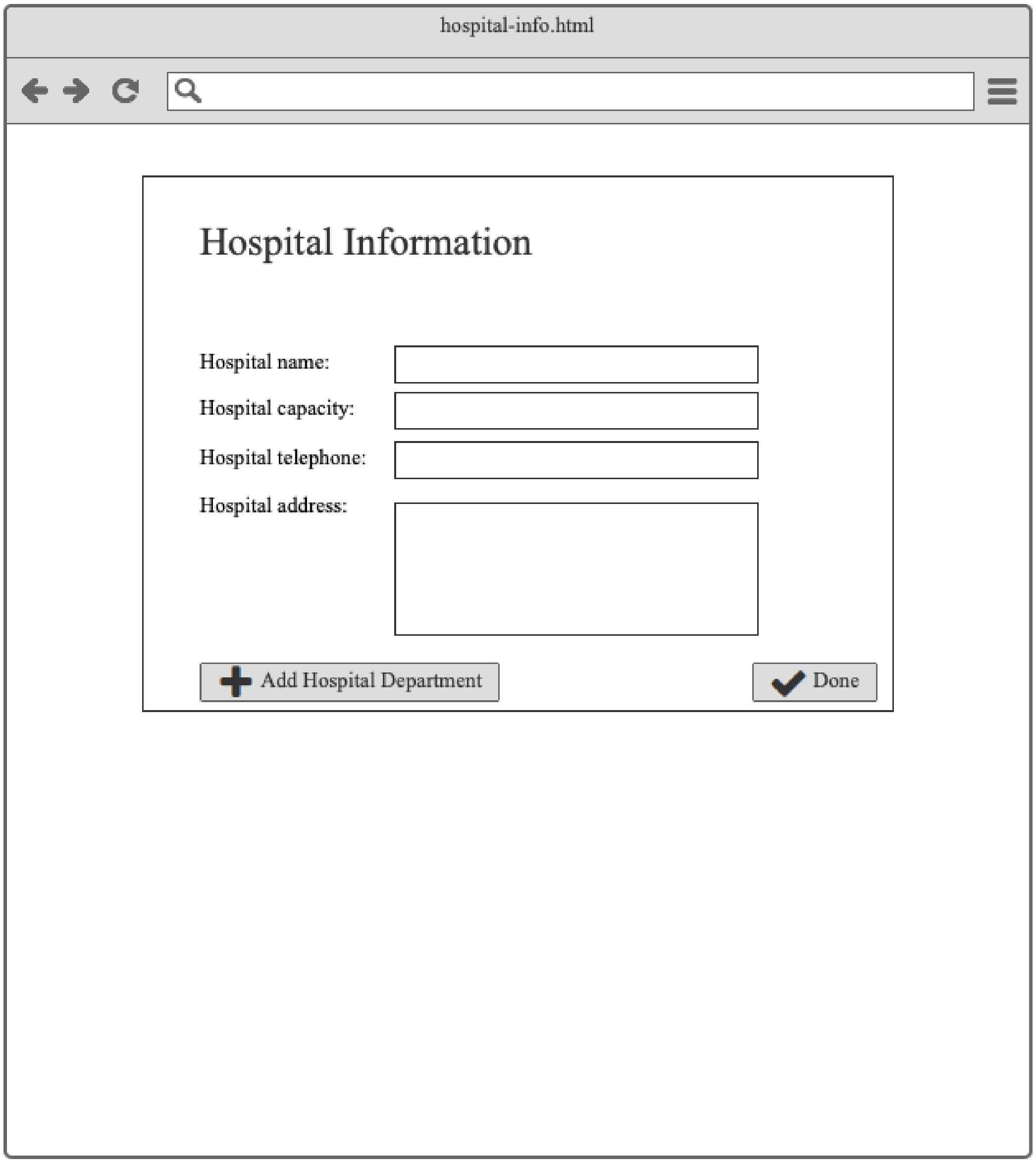
All three type of the account has common features such as first name, middle name, last name, sex and phone number of the user. For doctor account, these informations belong to an executive doctor of the hospital. Similarly, if it is a pharmacist account, these informations belong to owner of the pharmacy.

SQL Statements

Registering User Information

```
UPDATE user
SET
        first_name = @first_name,
        middle_name = @middle_name,
        last_name = @last_name,
        sex = @sex,
        phone = @phone
WHERE user.state_ID = @state_ID;
```

5.5 Hospital Information Page



Hospital Information

Hospital name:

Hospital capacity:

Hospital telephone:

Hospital address:

Figure 10: Hospital Information Page

In this page, user should enter informations about the hospital as it can be seen. By using "Add Hospital Department" button, he/she can create a department for the hospital and name it.

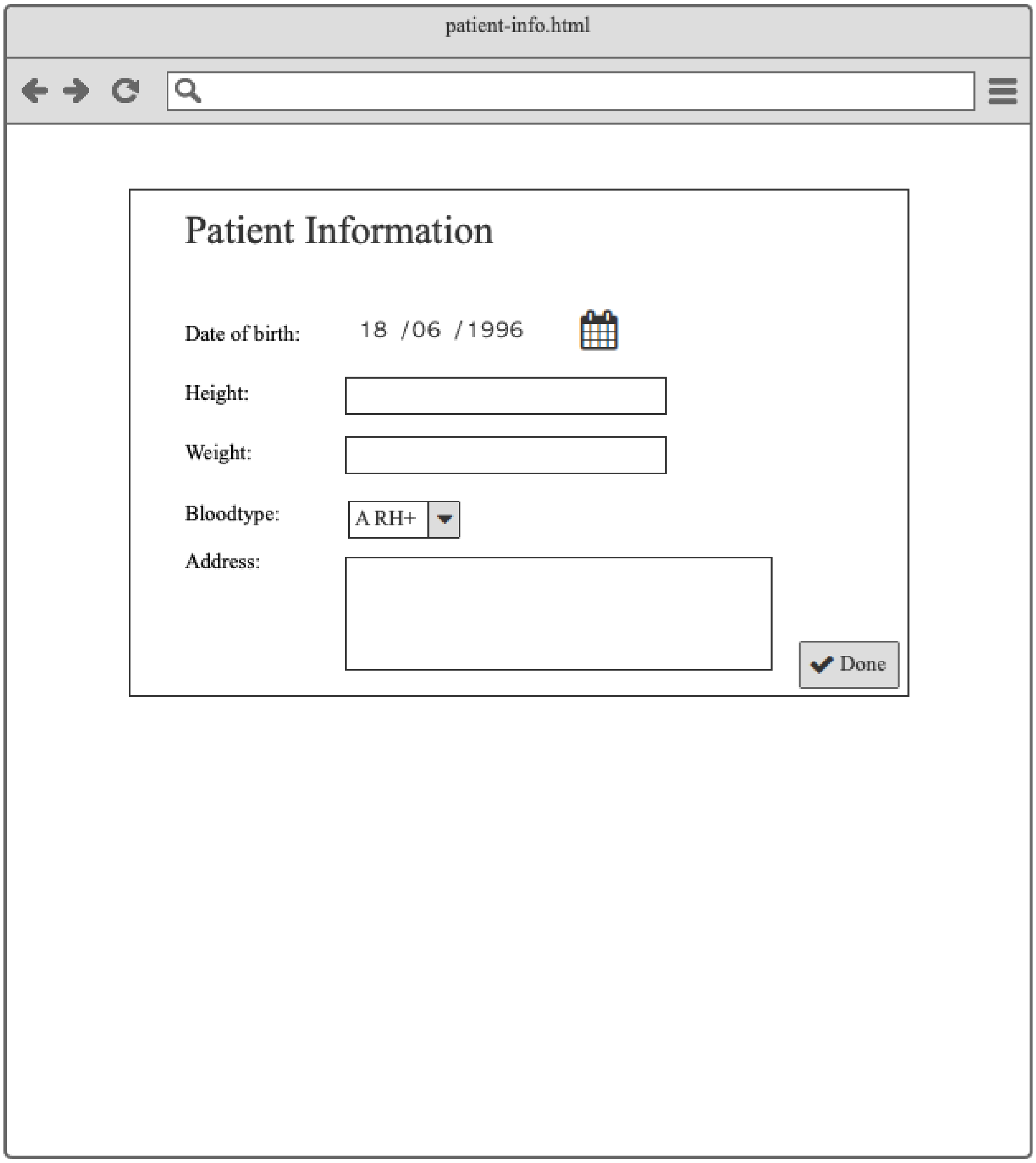
SQL Statements

Executive Doctor Registering His/Her Hospital

```
INSERT INTO hospital

VALUES (NULL, @hospital_name, @hospital_capacity, @hospital_telephone,
@hospital_address, @state_ID);
```

5.6 Patient Information Page



Patient Information

Date of birth: 18 /06 /1996

Height:

Weight:

Bloodtype: A RH+

Address:

Figure 11: Patient Information Page

SQL Statements

Patient Registering to System

INSERT INTO patient

VALUES (@state_ID, @patient_address, @patient_date_of_birth, @patient_weight, @patient_height, @patient_bloodtype);

5.7 Pharmacy Information Page

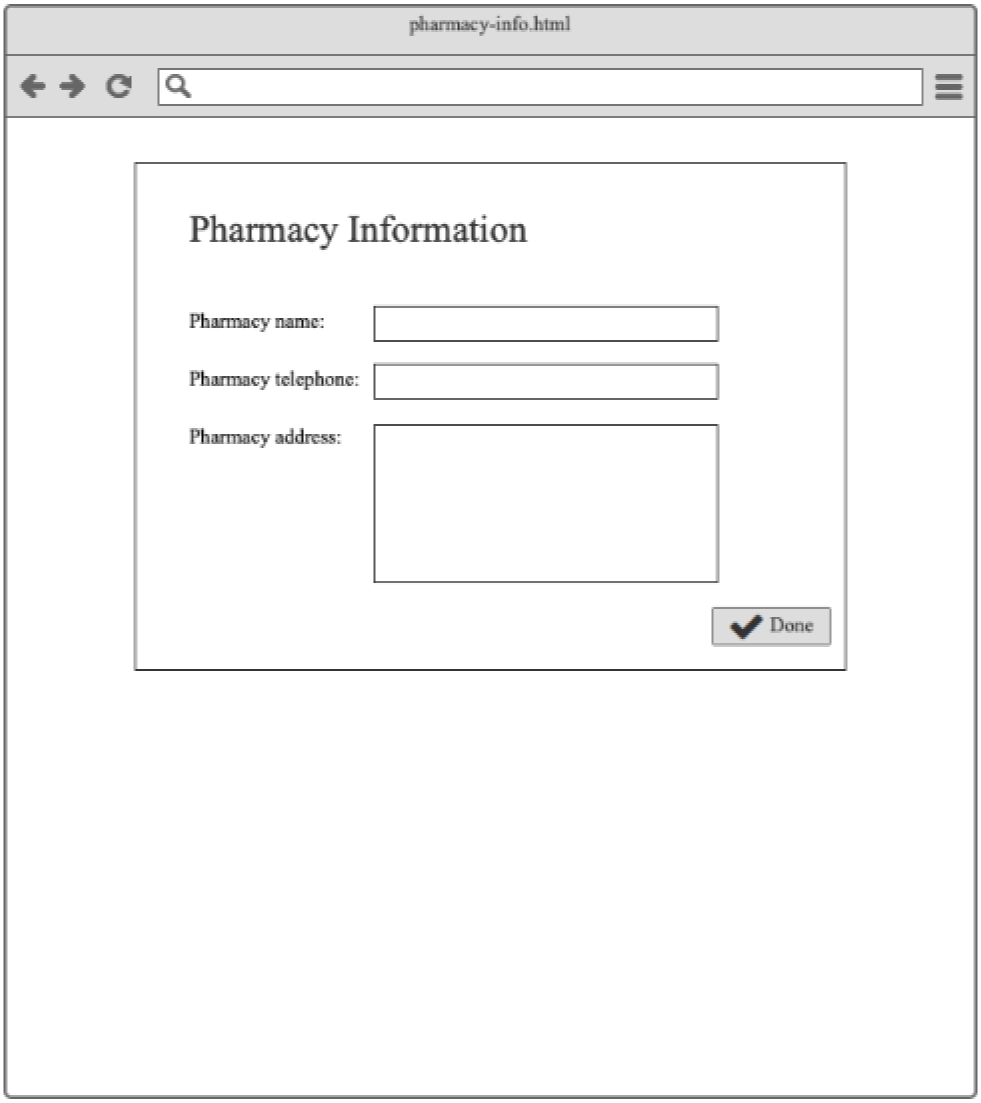


Figure 12: Pharmacy Information Page

SQL Statements

Pharmacist Registering His/Her Pharmacy to System

INSERT INTO pharmacy

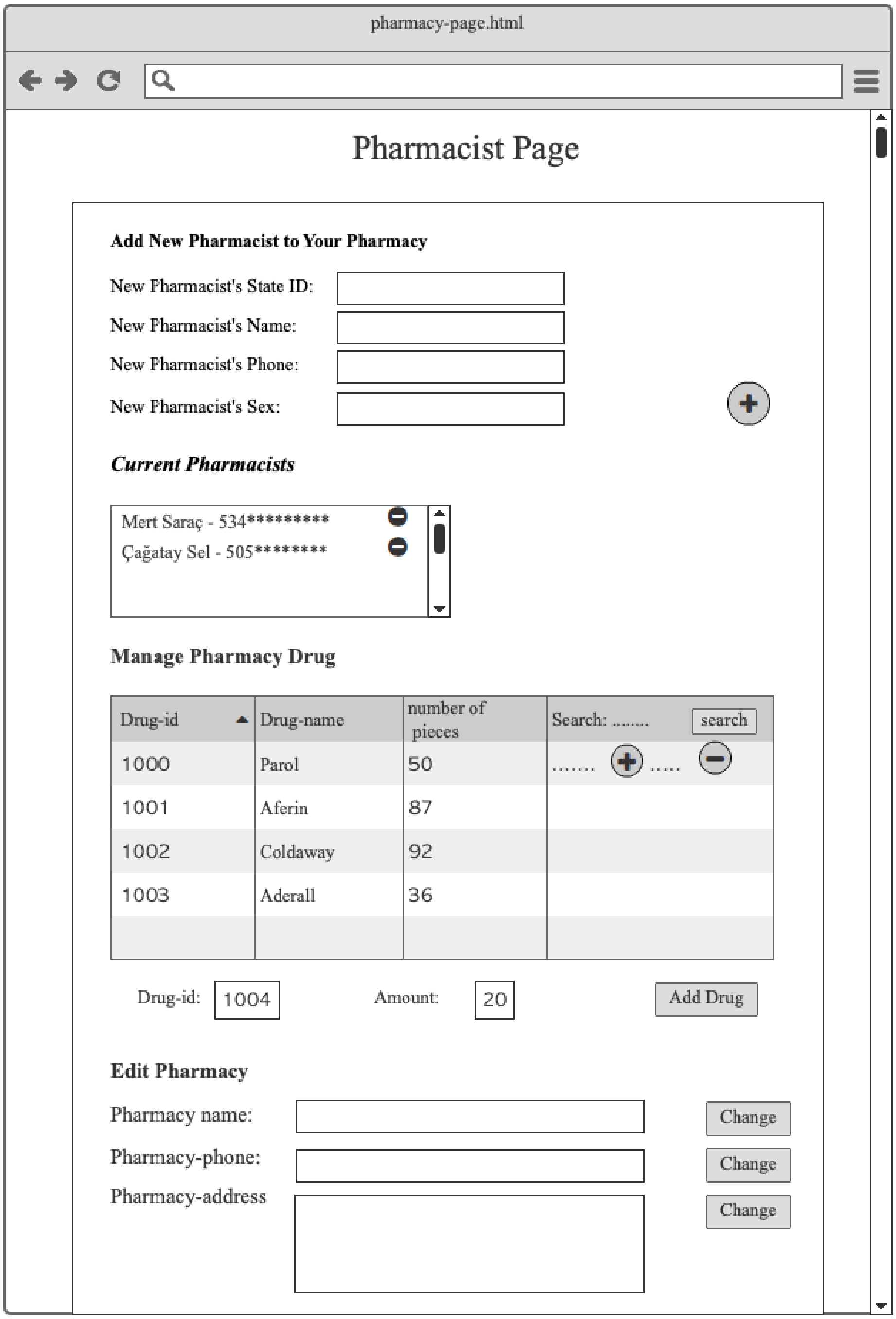
VALUES (NULL , @pharmacy_name, @pharmacy_address, @pharmacy_phone);

Adding Pharmacist as a Worker to His/Her Pharmacy

INSERT INTO worksAsPharmacist

VALUES (@state_ID , @pharmacy_ID);

5.8 Pharmacist Page



Pharmacist Page

Add New Pharmacist to Your Pharmacy

New Pharmacist's State ID:

New Pharmacist's Name:

New Pharmacist's Phone:

New Pharmacist's Sex:

Current Pharmacists

Mert Saraç - 534*********

Çağatay Sel - 505********

Manage Pharmacy Drug

Drug-id	Drug-name	number of pieces	Search: .... search
1000	Parol	50	.... + ... −
1001	Aferin	87	
1002	Coldaway	92	
1003	Aderall	36	

Drug-id: 1004 Amount: 20

Edit Pharmacy

Pharmacy name:

Pharmacy-phone:

Pharmacy-address

Figure 13: Pharmacist Page

SQL Statements

Adding New Pharmacist

```
INSERT INTO user
VALUES (state_id, first_name, middle_name, last_name, sex, phone);
```

Remove Pharmacist

```
DELETE FROM pharmacist
WHERE state_ID = @pharmacist_id
DELETE FROM worksAsPharmacist
WHERE state_ID = @pharmacist_id
```

Showing Current Pharmacist

```
SELECT first_name, middle_name, last_name,sex,phone,password
FROM user
WHERE user.state_id in  (SELECT state_id ,
                          FROM workAsPharmacist
                          WHERE workAsPharmacist.pharmacy_id = @pharmacy_id) ;
```

Showing Pharmacy Drugs

```
SELECT drug_ID, drug_name, number_in_stock
FROM  pharmacy NATURAL JOIN store NATURAL JOIN drug
WHERE  pharmacy.pharmacy_id = @pharmacy_id;
```

Adding Drugs to the Stock

```
UPDATE store
SET number_in_stock = number_in_stock + 1
WHERE  store.pharmacy_id = @pharmacy_id;
```

Removing Drugs from the Stock

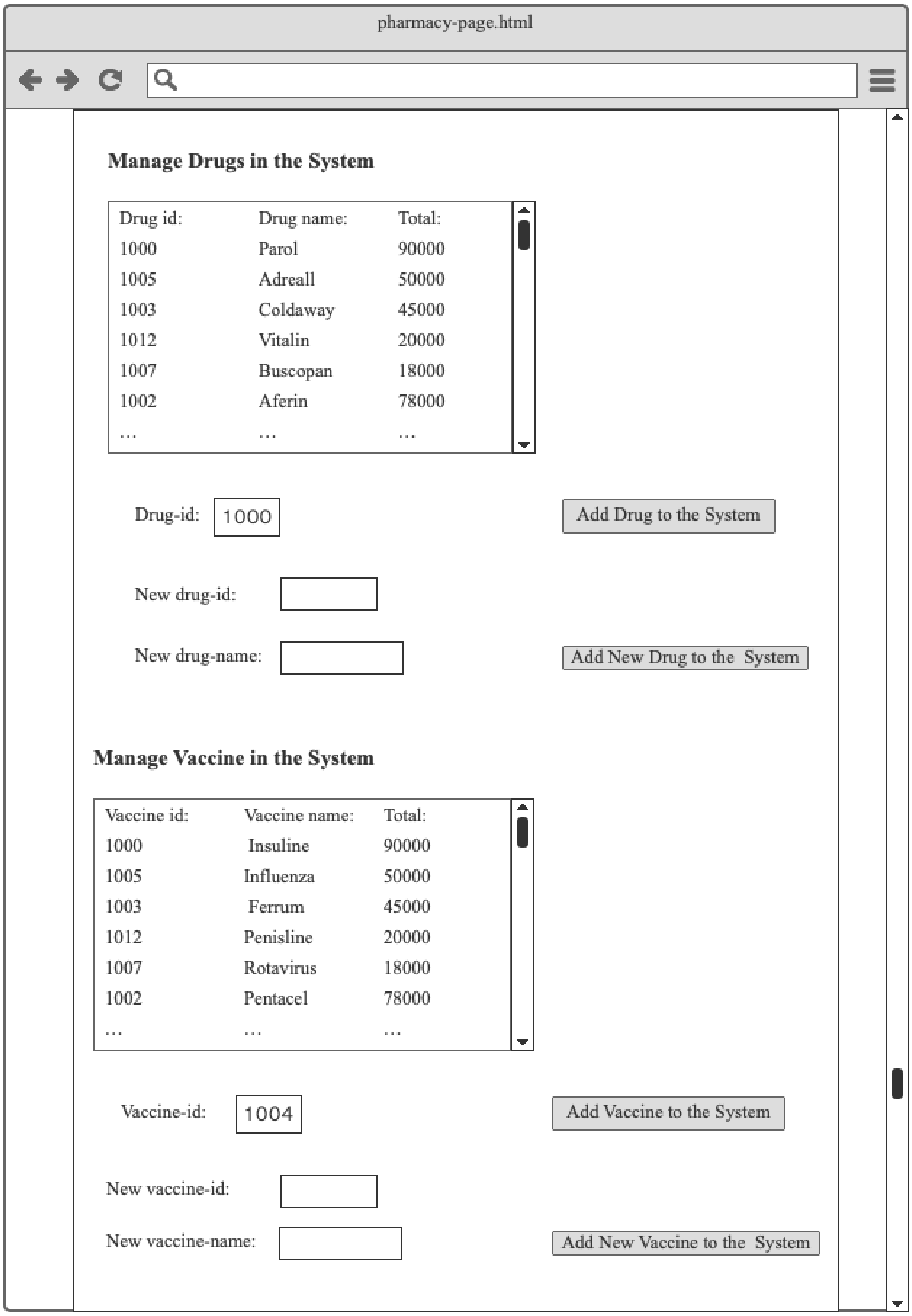
```
UPDATE store
SET number_in_stock = number_in_stock - 1
WHERE  store.pharmacy_id = @pharmacy_id;
```

Adding New Drugs from the Stock

```
INSERT INTO stores
VALUES (@pharmacy_id, @drug_id, @number_in_stock);
```

Edit Pharmacy

```
UPDATE pharmacy
SET pharmacy_name = @pharmacy_name, pharmacy_address = @pharmacy_address,
pharmacy_phone = @pharmacy_phone
WHERE  pharmacy.pharmacy_id = @pharmacy_id;
```


pharmacy-page.html
Add Drug to the System
Add New Drug to the System
Add Vaccine to the System
Add New Vaccine to the System

Manage Drugs in the System

Drug id:	Drug name:	Total:
1000	Parol	90000
1005	Adreall	50000
1003	Coldaway	45000
1012	Vitalin	20000
1007	Buscopan	18000
1002	Aferin	78000
…	…	…

Drug-id: 1000

New drug-id:

New drug-name:

Manage Vaccine in the System

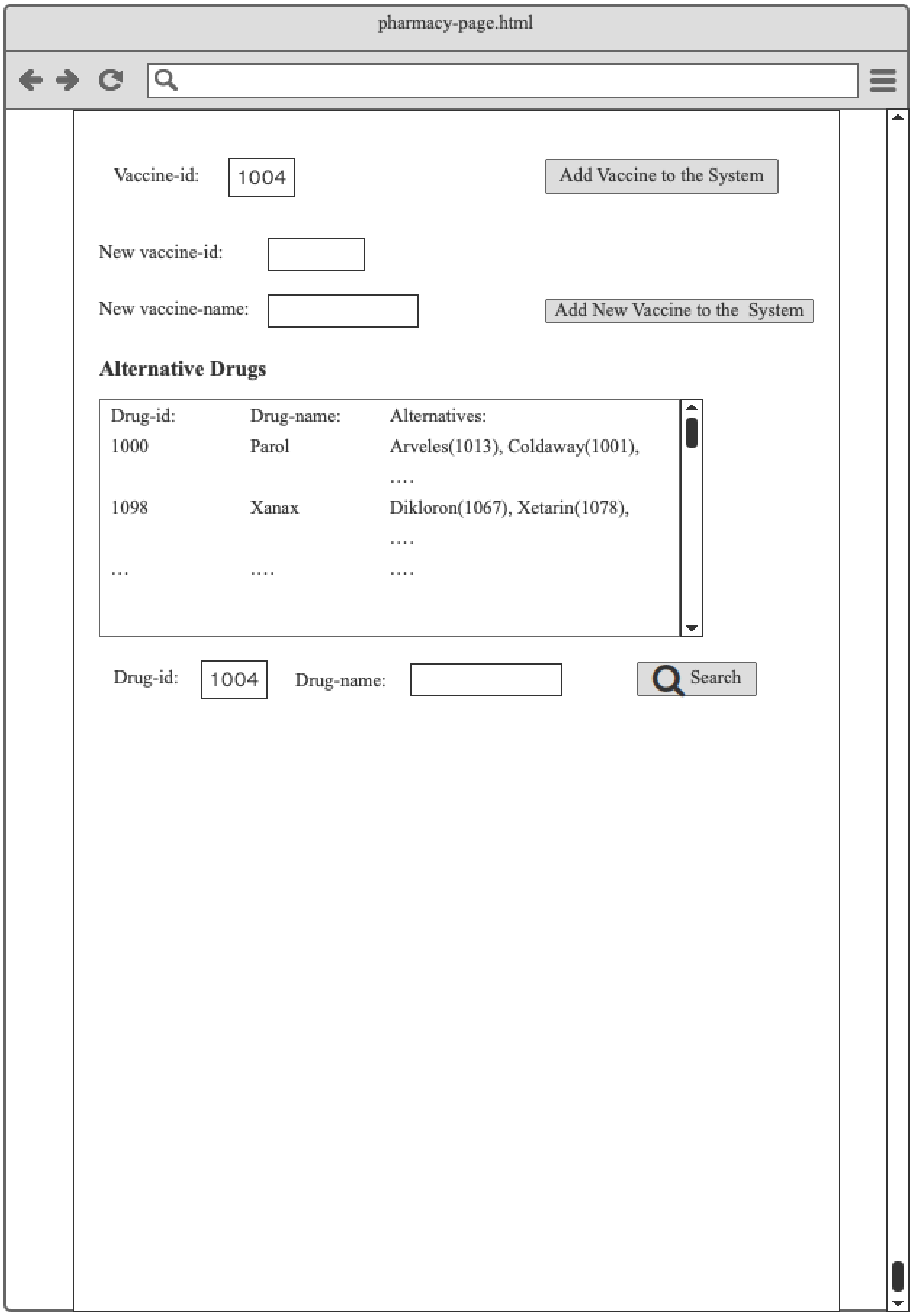
Vaccine id:	Vaccine name:	Total:
1000	Insuline	90000
1005	Influenza	50000
1003	Ferrum	45000
1012	Penisline	20000
1007	Rotavirus	18000
1002	Pentacel	78000
…	…	…

Vaccine-id: 1004

New vaccine-id:

New vaccine-name:

Figure 14: Pharmacist Page continued



Vaccine-id: 1004

New vaccine-id:

New vaccine-name:

Alternative Drugs

Drug-id:	Drug-name:	Alternatives:
1000	Parol	Arveles(1013), Coldaway(1001), ….
1098	Xanax	Dikloron(1067), Xetarin(1078), ….
…	….	….

Drug-id: 1004 Drug-name:

Figure 15: Pharmacist Page continued

SQL STATEMENTS

Show Drugs on The System

```
SELECT drug_ID, drug_name
FROM  drug;
```

Add Drugs to The System

```
INSERT INTO drug
VALUES (@drug_id, @drug_name);
```

Show Vaccine on The System

```
SELECT vaccine_ID, vaccine_name
FROM  vaccine;
```

Add Drugs to The System

```
INSERT INTO vaccine
VALUES (@vaccine_id, @vaccine_name);
```

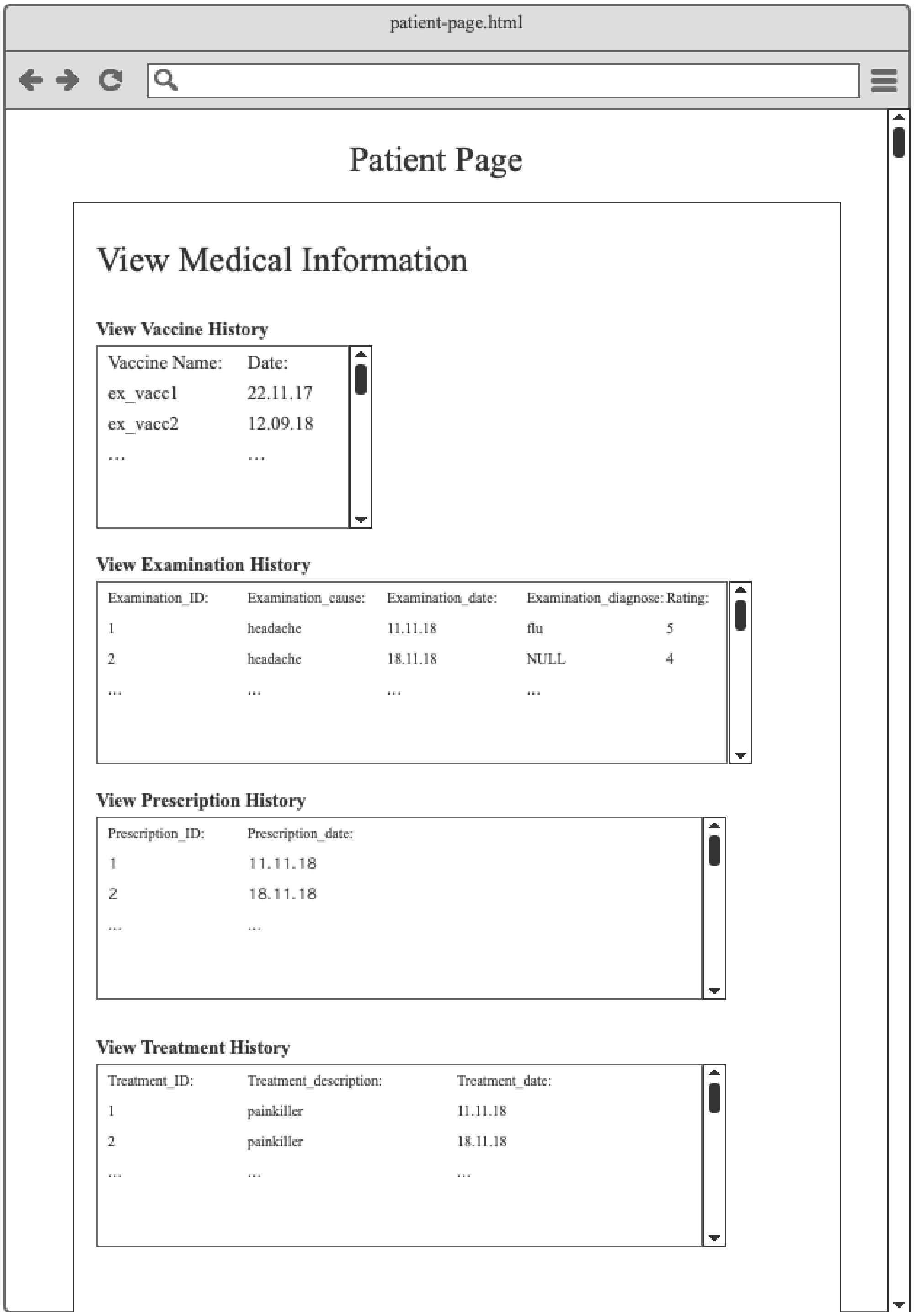
Show Alternative Drugs by ID

```
SELECT d.drug_ID, d.drug_name, a.drug_ID, a.drug_name
FROM  drug AS d, alternativeTo AS a
WHERE a.drug_ID = @drug_id;
```

Show Alternative Drugs by Name

```
SELECT d.drug_ID, d.drug_name, a.drug_ID, a.drug_name
FROM  drug AS d, alternativeTo AS a
WHERE a.drug_name = @drug_name;
```

5.9 Patient Page



Patient Page

View Medical Information

View Vaccine History

Vaccine Name:	Date:
ex_vacc1	22.11.17
ex_vacc2	12.09.18
…	…

View Examination History

Examination_ID:	Examination_cause:	Examination_date:	Examination_diagnose:	Rating:
1	headache	11.11.18	flu	5
2	headache	18.11.18	NULL	4
…	…	…	…	

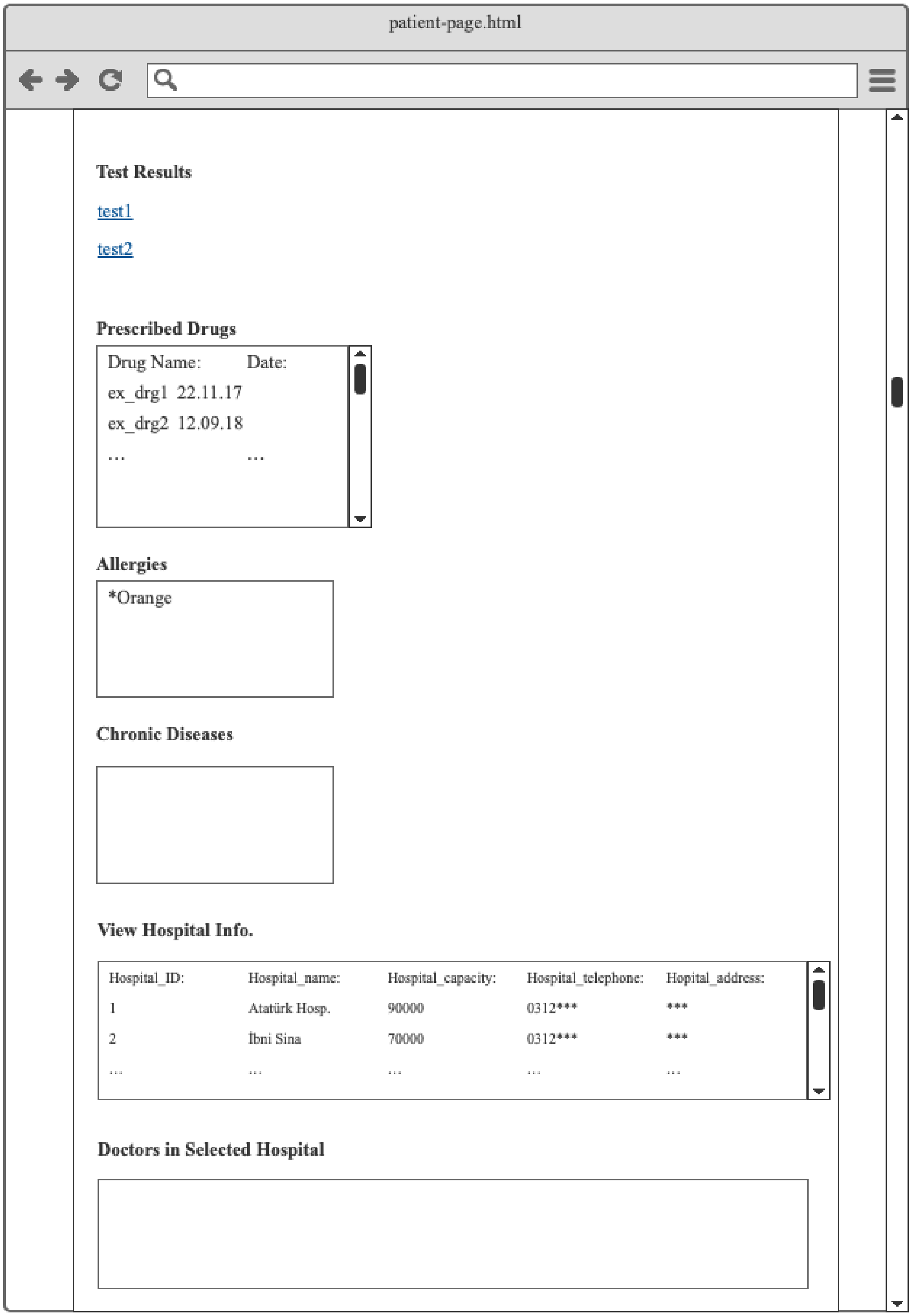
View Prescription History

Prescription_ID:	Prescription_date:
1	11.11.18
2	18.11.18
…	…

View Treatment History

Treatment_ID:	Treatment_description:	Treatment_date:
1	painkiller	11.11.18
2	painkiller	18.11.18
…	…	…

Figure 16: Patient Page



Test Results

test1

test2

Prescribed Drugs

Drug Name:	Date:
ex_drg1	22.11.17
ex_drg2	12.09.18
…	…

Allergies

*Orange

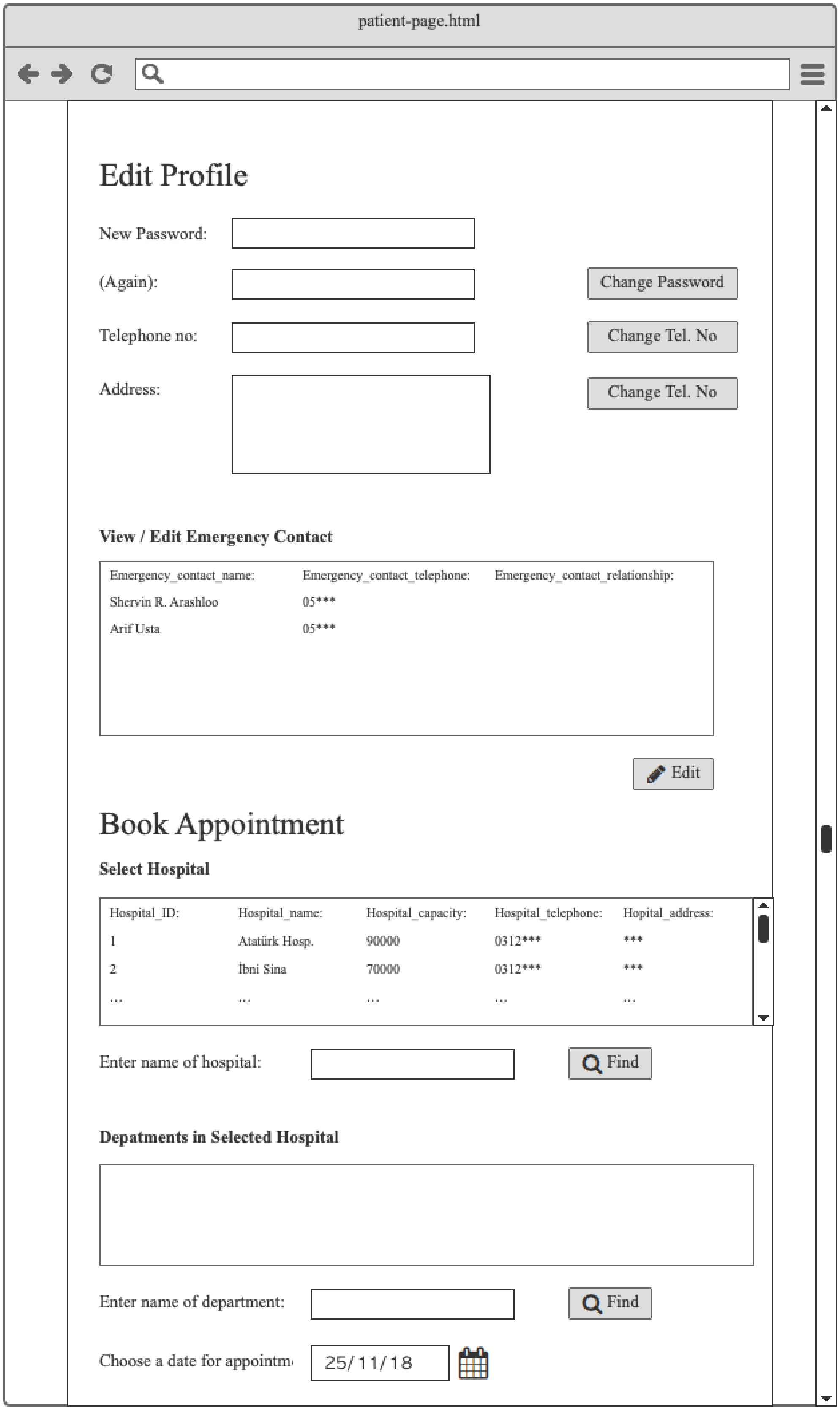
Chronic Diseases

View Hospital Info.

Hospital_ID:	Hospital_name:	Hospital_capacity:	Hospital_telephone:	Hopital_address:
1	Atatürk Hosp.	90000	0312***	***
2	İbni Sina	70000	0312***	***
…	…	…	…	…

Doctors in Selected Hospital

Figure 17: Patient Page continued



Edit Profile

New Password:

(Again):

Telephone no:

Address:

View / Edit Emergency Contact

Emergency_contact_name:	Emergency_contact_telephone:	Emergency_contact_relationship:
Shervin R. Arashloo	05***	
Arif Usta	05***	

Book Appointment

Select Hospital

Hospital_ID:	Hospital_name:	Hospital_capacity:	Hospital_telephone:	Hopital_address:
1	Atatürk Hosp.	90000	0312***	***
2	İbni Sina	70000	0312***	***
…	…	…	…	…

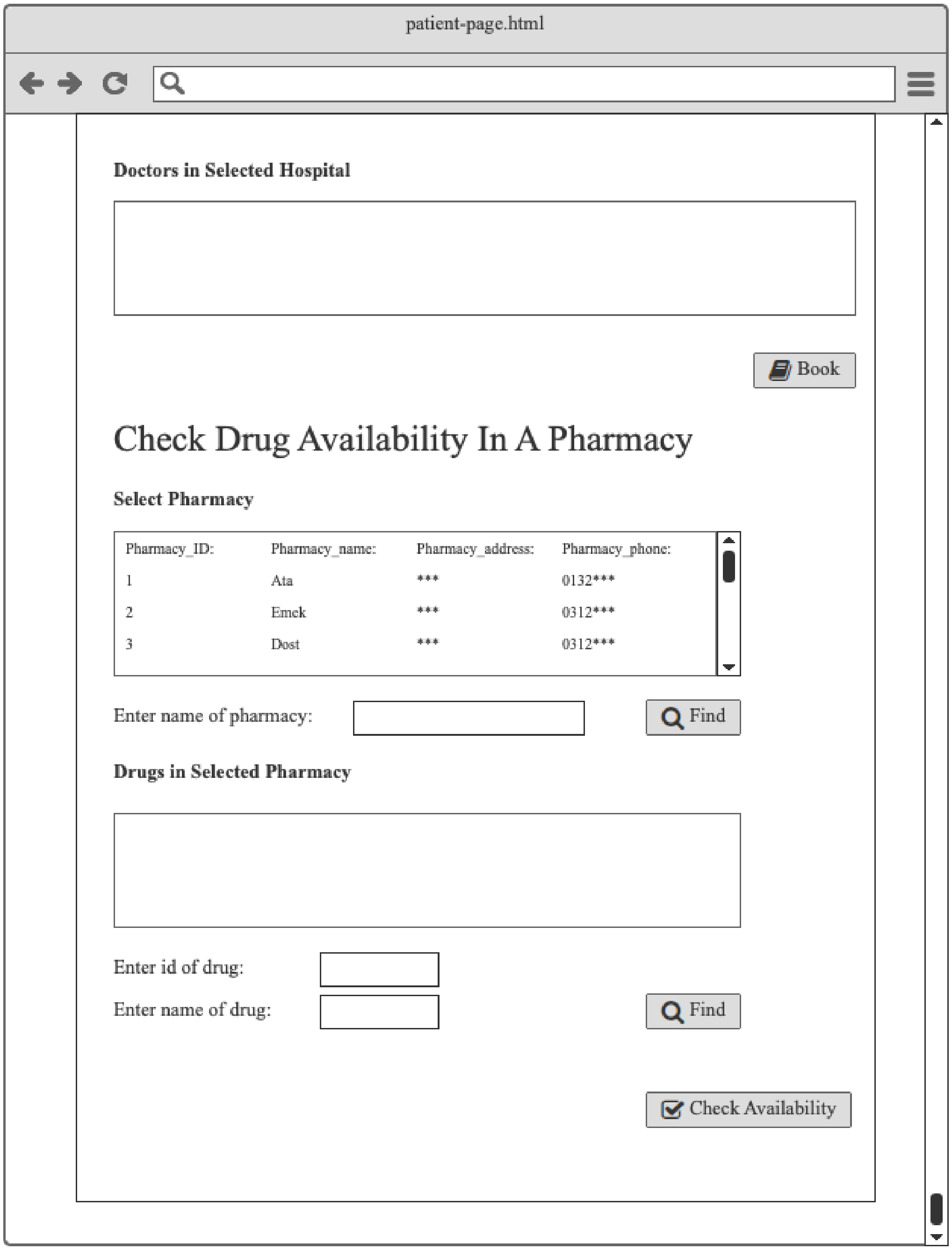
Enter name of hospital:

Depatments in Selected Hospital

Enter name of department:

Choose a date for appointm 25/11/18

Figure 18: Patient Page continued



Doctors in Selected Hospital

Check Drug Availability In A Pharmacy

Select Pharmacy

Pharmacy_ID:	Pharmacy_name:	Pharmacy_address:	Pharmacy_phone:
1	Ata	***	0132***
2	Emek	***	0312***
3	Dost	***	0312***

Enter name of pharmacy:

Drugs in Selected Pharmacy

Enter id of drug:

Enter name of drug:

Figure 19: Patient Page continued

Show Vaccine History

```
SELECT vaccinates.vaccine_name, vaccinates.date
FROM vaccinates NATURAL JOIN user
WHERE user.state_ID = @user_id;
```

Show Examination History

```
SELECT examination_ID, examination_cause, examination_date, examination_diagnose
FROM examination NATURAL JOIN user
WHERE  user.state_ID = @user_id
```

Show Prescription History

```
SELECT prescription_ID, prescription_date
FROM prescription NATURAL JOIN user
WHERE user.state_ID = @user_id;
```

Show Treatment History

```
SELECT treatment_ID, treatment_description, treatment_date
FROM treatment NATURAL JOIN user
WHERE user.state_ID = @user_id;
```

Show Patient’s Allergies

```
SELECT patient.allergies
FROM patient NATURAL JOIN user
WHERE patient.state_ID = @state_id;
```

Show Patient’s Chronic Disease

```
SELECT patient.chronic_disease
FROM patient NATURAL JOIN user
WHERE patient.state_ID = @state_id;
```

View Hospitals

```
SELECT hospital_id, hospital_name, hospital_capacity, hospital_telephone, hospital_address
FROM hospital;
```

View Doctors in Selected Hospital

```
SELECT first_name, middle_name, last_name,sex,phone,password
FROM user
WHERE user.state_ID in  (SELECT state_ID ,
                          FROM workAsDoctor
                          WHERE workAsDoctor.hospital_ID = hospital_ID) ;
```

Edit Password

```
UPDATE user
SET user.password  = @password
WHERE  user.state_id = @state_id;
```

Edit Telephone

```
UPDATE user
SET user.telephone = @telephone
WHERE  user.state_id = @state_id;
```

Edit Address

```
UPDATE user
```

```
SET user.address = @address
WHERE  user.state_id = @state_id;
```

Change Emergency Contact

```
UPDATE emergency_contact
SET emergency_contact_name = @emergency_contact_name, emergency_contact_telephone =
@emergency_contact_telephone, emergency_contact_relationship =
@emergency_contact_relationship
WHERE state_id = @state_id;
```

Book Appointment

```
INSERT INTO book
VALUES (@patient_id, @examination_ID, @doctor_id);
WHERE state_id = @state_id;
```

Check Availability of Drug

```
SELECT drug_id, drug_name
FROM drug NATURAL JOIN pharmacy
WHERE pharmacy.id in  (SELECT pharmacy_id
                        FROM store
                        WHERE number_in_stock > 0) ;
```

Give Rating To Examination

```
INSERT INTO rating
VALUES (NULL, @score, @comment)
```

6. Advanced Database Components

6.1 Views

6.1.1 Patient Age

This view will be used to get age of the users from their date of birth. Age was an deprived attribute in E/R diagram so it should be represented as a view.

```
CREATE VIEW patient_age as
SELECT state_ID, TIMESTAMPDIFF (YEAR, patient_date_of_birth,CURDATE()) AS age
FROM user;
```

6.1.2 Doctors Examination Rating

This view will be used to calculate average rating of patients so that patients can access this information while booking examination.

```
CREATE VIEW doctor_rating as
SELECT doctor_state_ID , avg(score) as avg_doctore_score
FROM rateExamination NATURAL JOIN rating
GROUP BY doctor_state_ID
```

6.1.3 Hospital Rating

This view will be used to calculate average rating of the doctors in a hospital so that patients can see hospital ratings.

```
CREATE VIEW hospital_rating as
SELECT hospital_ID , hospital_name,  avg(avg_doctore_score) as avg_hospital_score
FROM doctor_rating, works_as_doctor
WHERE doctor_rating.doctor_state_ID = works_as_doctor.stateID
GROUP BY hospital_ID,hospital_name;
```

6.2 Stored Procedures

Some of our queries such as queries for listing doctors or patient information can be written as an stored procedure since these queries will be executed many times by many users. Also stored procedures could be used to hide the internal information.

Stored procedure will also be used to add multiple rows of drugs to prescribed relation. Since we enable doctors to add multiple drugs to prescription and submit the prescription as whole, a stored procedure can add multiple tuples in batches.

6.3 Reports

6.3.1 Total Number of Examinations Annually

This report will be used to calculate the number of examinations that are done in the last 7 days of a hospital.

```
SELECT count(examination_ID) as examination_numbers
FROM (worksAsDoctor inner join examinationDone on worksAsDoctor.state_ID =
examinationDone.doctor_state_ID) inner join examination on
examination_Done.examination_ID = examination.examination_ID
WHERE examination.examination_date >= DATE(NOW()) - INTERVAL 365 DAY
```

6.3.2 Total Number of Examinations Annually for Each Hospital

This report will be used to calculate the number of examinations that are done in the last year of a hospital.

```
SELECT worksAsDoctor.hospital_ID, count(examination_ID) as examination_numbers
FROM (worksAsDoctor inner join examinationDone on worksAsDoctor.state_ID =
examinationDone.doctor_state_ID) inner join examination on
examination_Done.examination_ID = examination.examination_ID
WHERE examination.examination_date >= DATE(NOW()) - INTERVAL 365 DAY
GROUP BY worksAsDoctor.hospital_ID;
```

6.3.3 Total Number of Prescriptions Annually for Each Hospital

This report will be used to calculate the number of prescriptions that are written in the last year of a hospital.

```
SELECT worksAsDoctor.hospital_ID, count(examination_result.prescription_ID) as
prescription_numbers
FROM (worksAsDoctor inner join examinationDone on worksAsDoctor.state_ID =
examination_Done.doctor_state_ID) inner join examination_result on
examination_Done.examination_ID = examination_result .examination_ID
WHERE examination_result.prescription_date >= DATE(NOW()) - INTERVAL 365 DAY
GROUP BY worksAsDoctor.hospital_ID;
```

6.4 Triggers

- A trigger will be used to prevent executive doctors from registering a doctor to different hospitals. This trigger will be triggered after each insertion to workAsPharmacist table to ensure a doctor’s state id is only in one tuples.
- A trigger will be used to prevent a pharmacist from being registered into 2 different pharmacy. A pharmacist should not be working on 2 pharmacy at the same time. This trigger will be called on insertions to workAsPharmacist table.
- A trigger will be used to prevent a patient from booking 2 examinations that have overlapping times. This trigger will be called on insertion to books table and check if a patient have booked 2 overlapping examination.

- A trigger will be used to prevent a patient from rating an examination more than once. This trigger will be called after insertions on rateFor table and ensure that all the rows are unique and there are not any duplicates.
- A trigger will be used to prevent a drug that is not registered in the patient tracking system to be added to pharmacy inventory. A pharmacist must register the drug to system first. This trigger will be called after insertion to stores table and check if the drug is listed in drug table.

6.5 Constraints

- There are foreign key constraints for tables that ensure the referential integrity among the database.
- Users register and login to system with their state id. Our system does not ensure that every user will use their state id and not state id of others.
- Our triggers prevent a doctor from working at 2 hospitals at the same time or a pharmacist from working at 2 pharmacy at the same time.
- A patient can not book 2 examinations that have overlapping times.
- A patient can not rate an examination more than once.
- A drug that is not registered to list of drugs in the system can’t be added to inventory of a pharmacies. Drugs have to be added to system first.
- Doctors can not register to system by themselves. They have to be registered by their executive doctor.
- Doctors can not change their examination schedule. Executive doctor in every hospital manages the schedules of the doctors.

7. Implementation Plan

In our project implementation, MySQL is going to be used for database system. PHP is going to be used for web application development in the server side. HTML5, CSS3x and Javascript is going to used for user interface development and designing.