VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Database Systems Lab (CO2014)

Semester 211

Hospital database management system

Advisor:	Assoc. Prof. Trần Minh Quang	
Students:	Nguyễn Hải Đăng	1952652
	Phạm Hoàng Tùng	1852852
	Trần Xuân Đức	1552102

HO CHI MINH CITY, OCTOBER 2021

Contents

Member list & Workload

No.	Full name	Student ID	Percentage of work
1	Nguyễn Hải Đăng	1952652	33.33%
2	Phạm Hoàng Tùng	1852852	33.33%
3	Trần Xuân Đức	1552102	33.33%

1 Introduction

1.1 Problem

In the last 2 years, Covid-19 has forced healthcare industry to evolve. And with the growth of the pandemic comes with new technology to address the issue. Hospital used to store their data in traditional file system like: Microsoft Excel, Open office, Google spreadsheets. The main drawback of traditional file system is data definition is part of application program which works only with specific application. Files are design driven, they require change in design & Coding whenever new kind of data occurs. For these systems to operate smoothly and successfully, having a efficiency and reliable database management system to manage employees is crucial. Therefore, in this assignment, we will design a Hospital database management system.

1.2 Background

A Hospital Database Management System (HDMS) is a system that facilities managing the functioning of a hospital. This system will help in making the whole functioning paperless. The disease history, reports, prescribed treatment can be accessed by doctors without much delay in order to make an accurate diagnosis and monitor the patient's health. It enables lower risks of mistakes. HDMS is for computer computerizing working environment and is capable to provide easy and effective storage of information related to patients and doctors. **Features of the HDMS:** There are 6 main features such as Managing Doctors, Patient, Report, Nurse Rooms and Bills information.

1.3 Timeline

- **Phase 1**: Database Design (17/09 -> 14/10)
  - Product's Functionality Conceptual Design.
  - Logical Design using relational databases.
  - Product's Demo for testing.
- **Phase 2**: Deliver Product (15/10 -> 11/11) (Update after Phase 1)

2 Requirement description

The purpose of the Hospital Database Management System is to make a secure and easy way of storing information of the patients, doctors, rooms, and bill payment. The database includes all the necessary patient data.
The data requirement:

- Each doctor is identified by an id, has some attributes such as name, type of doctor, age, address and phone number, and belongs to at least one department. Each department also has an id (unique) and its name and contains several doctors.
- Each patient is identified by his/her Social Security Number (SSN) and has some other information such as name, gender, date of birth, age, telephone number, e-mail and address. In addition, each of them can be insured by only one health insurance company in a particular duration before buying a new insurance. Each

insurance company has an id (unique), name, address and telephone number. It can gives insurance to many its customer.

- A patient can have an appointment with a doctor on a specific date to get a consultant form the doctor, but not all doctors are responsible for meeting patient.
- A patient also can require for a test. This test has its own id (unique), name, the date it occurs and the fee. The test is made for a patient by a doctor. An operation with id (unique), name, room number, date and time will have the same flow like the test.
- After meeting patient or do the test or the operation, the doctor then makes a report including id (unique), type of report, date its created and description. A doctor can make many report, also a patient can get many his/her report.
- Each medicine in the hospital is identified by its id, has name, price of a unit, name of the manufacturer, manufacturing date, expiration date and the quantity it is currently stored, and can cure from one to many diseases. Each disease has an id (unique), name, description and can be cured by some treatments (multi-values) or some kind of medicines.
- A report may include some kind of medicines for the patients with a particular quantity, also one kind of medicine can be provided for many patients through reports.
- A specific room in the hospital is identified by its number and has room type, corresponding cost and a status indicating available or used for other purpose. It can admit several patients and a patient can used several rooms while being in the hospital. The fee for each patient is calculated through the multiplication between the room cost and the number of date he/she used.
- Each nurse has id (unique), name, type if nurse, age and the shift for work. A nurse takes care of some patient and also a patient can be cared by some nurses while being in the hospital.
- Finally, an accountant with id (unique), name, e-mail and telephone number is responsible for creating bills. Each bill created has its number (unique), the date it is created, the charge for all service that the customer used. At this time, if the customer has an insurance, the insurance company gives a discount and the total fee is what the customer has to purchase. Customer can know the date they purchase for the bill and also can purchase by cash or credit card.

3 Tools used in this project

3.1 ERD Model Design Tool

To design our database in this assignment, **Draw.io** will be our tool of choice. **Draw.io** is a web-based tool UML, ERD, and any designed diagram. We can use either the web base or desktop application.

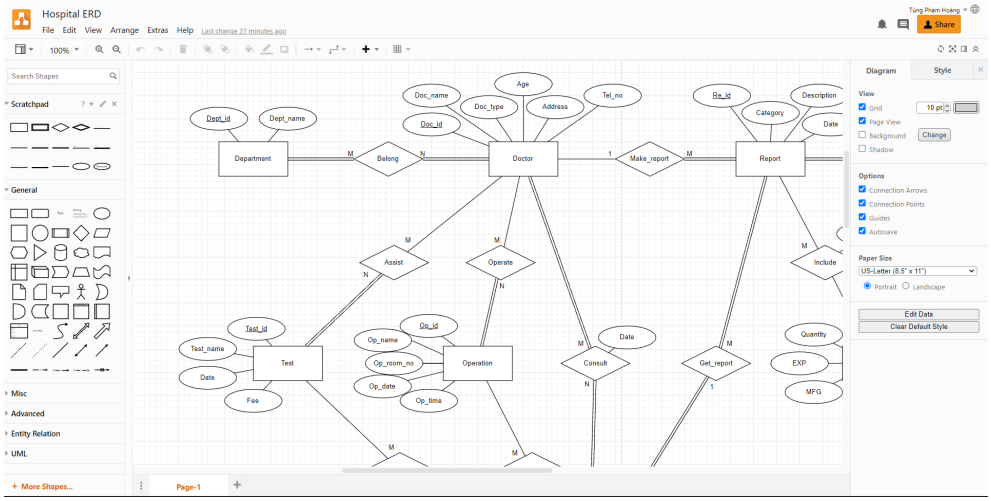

Figure 1: Draw.io Main Interface

Although **Draw.io** is a paid service, we don't need premium features, free version provides us enough to cover this assignment.

Advantages

- Easily produces good quality diagrams
- Has a rich set of predefined shapes for all sorts of different diagramming needs
- Allows grouping of shapes
- Smart connectors
- Integrates with Google Drive
- Conveniently exports to a variety of formats
- Allows for collaborative development of diagrams

3.2 Relational Database Management System (MySQL)

For implementing the database management system, we will use MySQL database management system.

MySQL is a relational database management system (RDBMS) developed by Oracle that is based on structured query language (SQL).

MySQL is one of the most recognizable technologies in the modern big data ecosystem. Often called the most popular database and currently enjoying widespread, effective use regardless of industry, it's clear that anyone involved with enterprise data or general IT should at least aim for a basic familiarity of MySQL.

With MySQL, even those new to relational systems can immediately build fast, powerful, and secure data storage systems. MySQL's programmatic syntax and interfaces are also perfect gateways into the wide world of other popular query languages and structured data stores.[1]

Why we choose MySQL:

- Is the most stable and high-speed database on the market today.

- MySQL, although it has high features, is simple to use, less complicated.
- Completely free because this is an open source. However, in case you need support from MySQL, you still have to pay.
- Support from the community because of many members.
- Works on multiple operating systems : Linux , MacOS , Window ,...

Advantage

- Accessibility and Easy to use: The setting up process is relatively basic and requires less than 30 minutes , plus source code is completely flexible and entirely free for all categories of users.
- High Efficiency: Even data is large , MySQL can perform the powerful respond against big data e-commerce sites or heavy business operation related to technology information
- Industry standard: Anyone who has ventured into the technology and data industry has used MySQL and users can also deploy projects quickly and hire data experts.
- Security: Safety is always an extremely important issue in the data industry and MySQL ensures very high security standards. Therefore, MySQL has been trusted and cooperated with dozens of popular web applications WordPress, Drupal, Joomla, Facebook and Twitter

Disadvantage

- Debugging: Compared to the other DBMS, MySQL , it does not handle in developing and debugging as good as their counterparts.
- Weak Enormous-Data Performance: Although it can manage data in large quantities, MySQL is still not capable of integrating huge and systematic data management such as: nationwide supermarket system, banks, population information management, etc. National numbers...

3.3 MySQL Workbench

MySQL Workbench is a cross-platform GUI client for MySQL database users and administrators. Workbench makes the task easier for database admins by providing important tools to manage databases and users, creating databases, running SQL queries, setting up and configuring servers, and many more.
It is a powerful tool that enables us to visualize modules for creating, executing, and optimizing several queries. So, in this article, I am going to give a tour of MySQL Workbench and show you how to use it. **Key MySQL Workbench Features**

- SQL Development: Enables you to create and manage connections to database servers. Along with enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor
- Data Modeling (Design): Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views

- Server Administration: Enables you to administer MySQL server instances by administering users, performing backup and recovery, inspecting audit data, viewing database health, and monitoring the MySQL server performance.
- Data Migration: Allows you to migrate from Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.
- MySQL Enterprise Support: Support for Enterprise products such as MySQL Enterprise Backup, MySQL Firewall, and MySQL Audit.

4 Conceptual design

4.1 Entities

Entity Name	Description
Doctor	The one belong to the specific departments who can test , operating and write a report which contains information : disease name, medicine ,...
Patient	List of patients who are treated in hospital. Each patient has each own detailed information
Report	List of reports that are formed by Doctors . Each one describes current statue of health, diagnosis and medicine offering which is sent to patients
Department	List of departments where each specific category of Doctors belongs in
Room	List of rooms and the statue of each one are admitted by patients
Operation	Containing the information of registered operation which exists in Hospital
Test	List of tests that are executed by Doctor
Medicine	List of medicines connected to a disease are include in Report form
Disease	The one diagnosed to be cured by the chosen medicine
Accountant	The one who can create the bill receipt for customer to buy medicine and service
Bill	List of bill payment for the treatment that patient has
Insurance company	They have mission to insure to their own patient and will pay for bill that the customer
Nurse	The ones have responsibility to take care of patients in the hospital
Appointment	The schedule table contains the time schedule that Doctor consult to patients

4.2 Relationships

Relationship Name	Participants and Constraints
Belong	Department (total) and Doctor (total) M:N Relationship
Do_test	Patient (partial), Doctor (partial) and Test (total) 1:1:M Relationship
Operate	Patient (partial), Doctor (partial) and Operation (total) 1:1:M Relationship
Make_Report	Doctor (partial) and Report (total) 1:N Relationship
Get_report	Patient (total) and Report(total) 1:N Relationship
Include	Report (partial) and Medicine (partial) M:N Relationship
Admit	Patient (partial) and Room(partial) M:N Relationship
Consult	Patient (partial), Doctor (partial) and Appointment (total) 1:1:M Relationship
Pay_InsCo	Insurance_company(partial) and Patient(partial) 1:N Relationship
Cure	Medicine (total) and Disease (partial) M:N Relationship
Care	Nurse (total) and Patient (partial) M:N Relationship
Insure	Insurance_company(total) and Patient(partial) 1:N Relationship
Purchase	Patient (total) and Bill(total) 1:N Relationship
Create_bill	Accountant (total) and Bill(total) 1:N Relationship

4.3 Entity-Relationship Model

Based on the description in the Requirement Description, We came with our design below. For full picture, here is the Img file

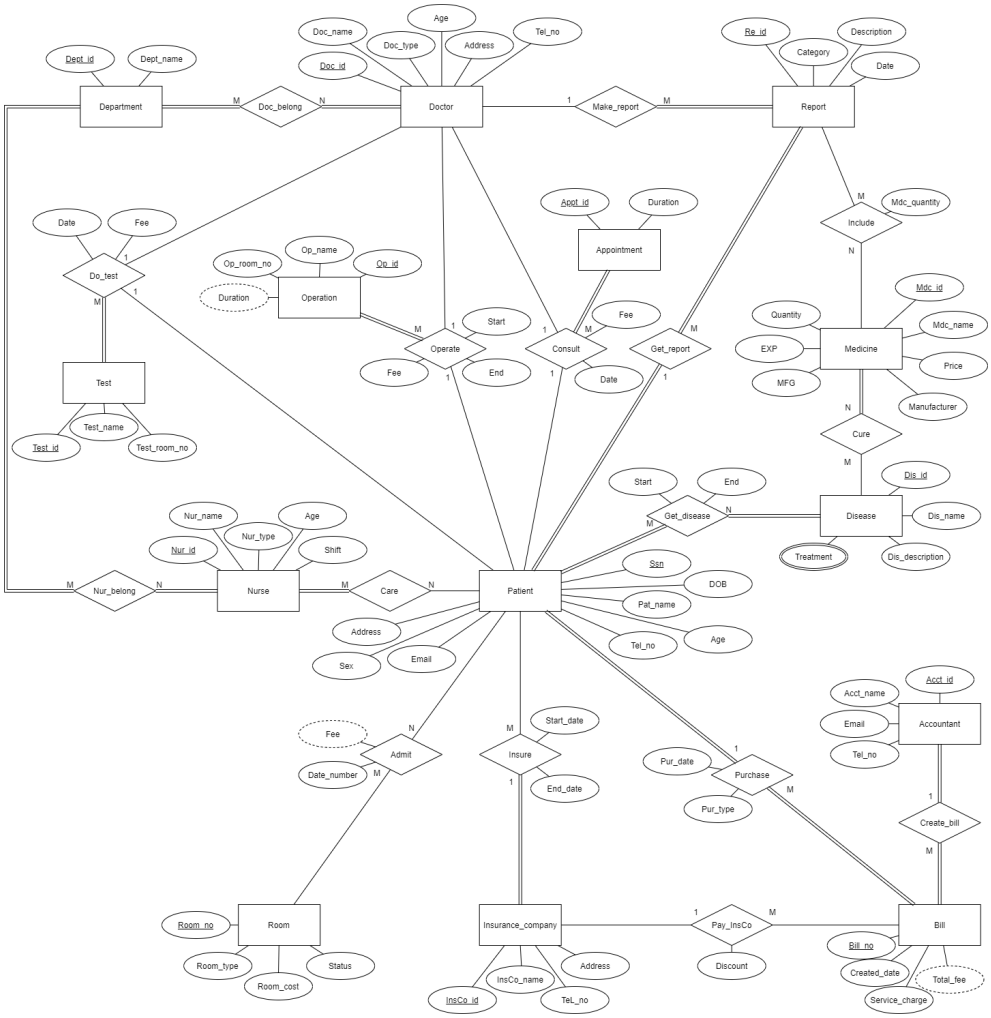

Figure 2: The ERD Model of the hospital database system

5 Logical design

From Conceptual design to Logical design using Relational data model, a several steps have been applied. However, we will list only steps that we use to mapping our design from ERD to relational data model[2].
Step 1: Mapping of Regular Entity Types. For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E.
Step 2: Mapping of Weak Entity Types. For each weak entity type W in the ER schema with owner entity type E, create a relation R and include all simple attributes (or simple components of composite attributes) of W as attributes of R.
Step 3: Mapping of Binary 1:1 Relationship Types. For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R.
Step 4: Mapping of Binary 1:N Relationship Types. There are two possible approaches: (1) the foreign key approach and (2) the cross-reference or relationship relation approach. The first approach is generally preferred as it reduces the number of tables.
Step 5: Mapping of Binary M:N Relationship Types. For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.
Step 6: Mapping of Multivalued Attributes. For each multivalued attribute A, create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as a multivalued attribute Based on our Conceptual Design, we also created our Logical Design on ERDPlus.
Step 7: Mapping of N-ary Relationship Types. We use the relationship relation option. For each n-ary relationship type R, where $n > 2$, create a new relationship relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.
For full picture, here is the Img file.

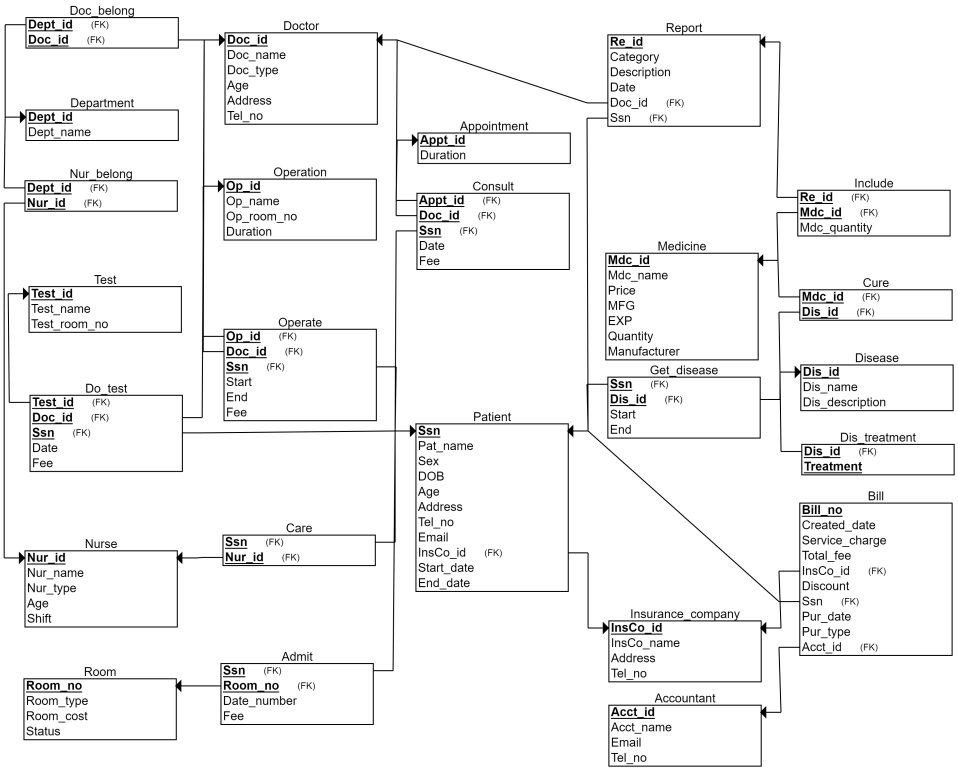

Figure 3: Logical Design of the hospital database system

6 Normalization form

6.1 Overview of normalization form

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.
The normalization procedure provides database designers with the following:

- A formal framework for analyzing relation schemas based on their keys and on the functional dependencies among their attributes
- A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be **normalized** to any desired degree

Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:

- The **nonadditive join or lossless join property**, which guarantees that the spurious tuple generation problem does not occur with respect to the relation schemas created after decomposition
- The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting after decomposition

The nonadditive join property is extremely critical and **must be achieved at any cost**, whereas the dependency preservation property, although desirable, is sometimes sacrificed.
To reach the BCNF (Boyce-Codd Normal Form), a relation must satisfied 1NF, 2NF and 3NF. Database design as practiced in industry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF. The database designers need not normalize to the highest possible normal form. Relations may be left in a lower normalization status, such as 2NF, for performance reasons. However, in this assignment, we will try to get all relations in the relation schema satisfied 3NF to get the normal form of 3NF instead of BCNF because we think 3NF is suitable and convenient for our database design.

6.2 First Normal Form

First normal form (1NF) was historically defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute. Hence, 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples. The only attribute values permitted by 1NF are single atomic (or indivisible) values.

If a relation is not in 1NF, there are three main techniques to achieve first normal form:

1. Remove the attribute that violates 1NF and place it in a separate relation along with the primary key of the original relation. The primary key of this newly formed relation is a combination. This decomposes the non-1NF relation into two 1NF relations.

2. Expand the key so that there will be a separate tuple in the original relation for each value of the attribute that violates 1NF. In this case, the primary key becomes the combination between the primary key of the original relation and each value of that attribute. This solution has the disadvantage of introducing redundancy in the relation and hence is rarely adopted.

3. If a maximum number of values is known for the attribute—for example, if it is known that at most three—replace that attribute by three atomic attributes. This solution has the disadvantage of introducing NULL values if most tuples have fewer than three values of the violated attribute. It further introduces spurious semantics about the ordering among the values; that ordering is not originally intended. Querying on this attribute becomes more difficult.

The first is generally considered best because it does not suffer from redundancy and it is completely general; it places no maximum limit on the number of values.
Since **Disease** entity in ERD contains a multivalued attribute, which is **Treatment** attribute, when mapping from ERD to Relation Schema, we have decomposed it into a new separate relation along with the primary key **Dis_id**. Hence, the hospital relation schema satisfied 1NF.

First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it. To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combine the partial key with the primary key of the original relation. This procedure can be applied recursively to a relation with multiple-level nesting to unnest the relation into a set of 1NF relations.

6.3 Second Normal Form

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore. A functional dependency X → Y is a partial dependency if some attribute A belong to X can be removed from X and the dependency still holds.

A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R. In other words, a relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on every key of R.

If a relation is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. Therefore, the number of decomposed relations is equal to the number of partial functional dependencies that the original relation contains.

Before checking whether the hospital relation schema satisfies 2NF or not, we must define functional dependencies for each relation. Although the list of functional dependencies may not have enough as expected or may have incorrect due to the ambiguity of the functional requirements, we will try to introduce the functional dependencies of some relations in the hospital relation schema as an example.

Patient

- **FD1:** Ssn → Pat_name, Sex, DOB, Age, Address, Tel_no, Email, InsCo_id, Start_date, End_date
- **FD2:** Tel_no → Ssn, Pat_name, Sex, DOB, Age, Address, Email, InsCo_id, Start_date, End_date
- **FD3:** Email → Ssn, Pat_name, Sex, DOB, Age, Address, Tel_no, InsCo_id, Start_date, End_date

Doctor

- **FD1:** Doc_id → Doc_name, Doc_type, Age, Address, Tel_no
- **FD2:** Tel_no → Doc_id, Doc_name, Doc_type, Age, Address

Operate

- **FD1:** Op_id, Doc_id, Ssn → Start, End, Fee

After carefully checking, the relation schema of the hospital database is already in 2NF because all the nonprime attributes of each relation are fully dependent on the primary key of that relation. Relation with primary key containing only one attribute is auto-satisfied. In relation with primary key, which is a combination of two or three attributes, all nonprime attributes are fully dependent on all prime attributes and cannot be inferred by only one part of the primary key.

6.4 Third Normal Form

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X $\rightarrow$ Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, and both X $\rightarrow$ Z and Z $\rightarrow$ Y hold.

A relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key. In general definition, a relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X $\rightarrow$ A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

If a relation is not in 3NF because of the transitive dependency, we can normalize by decomposing it into the two 3NF relation schemas. A NATURAL JOIN operation will recover the original relation without generating spurious tuples.

The relation schema of hospital database is in 3NF after checking. This is enough for our target in normalization; however, next we will also introduce a bit about BCNF.

6.5 Boyce-Codd Normal Form

A relation schema R is in BCNF if whenever a nontrivial functional dependency X $\rightarrow$ A holds in R, then X is a superkey of R.

BCNF differs from 3NF in the clause (b) of 3NF, which allows functional dependencies have the RHS as a prime attribute, but is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF.

Since we just want to apply up to 3NF for our design, the following figure is our relational data model after normalization:

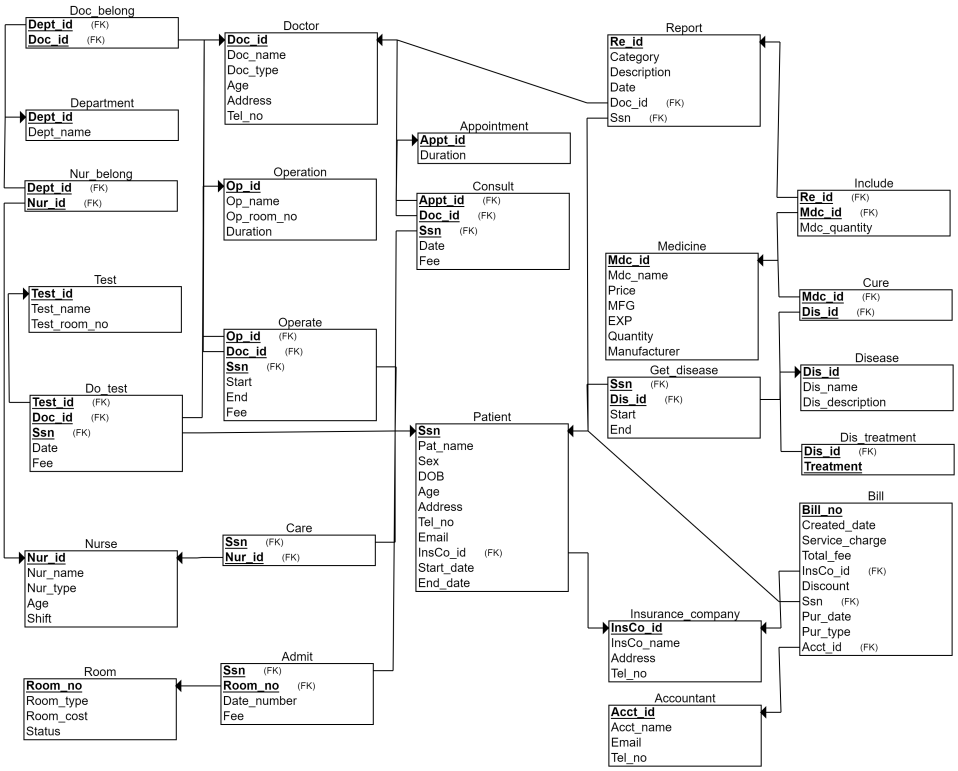

Figure 4: Relational data model after normalization

For full picture, here is the Img file.

This design seems exactly the same as the logical design in section 5 due to the reasonable design of the ERD model for the beginning and correctness when mapping it into Logical design.

7 Physical Design

In practice, we can only let attributes store data if their data types have already been declared clearly at the physical design stage. The following shows which kind of data types is suitable for each group of attributes.

- In most our tables, not including **Appointment** table, the primary keys (Ssn, id, no) should be numeric (INT) but not string based, both for space saving and for performance reasons (matching keys on strings is slower than matching on integers).
- Since the telephone numbers are sequences of 10 digits, **VARCHAR(10)** is a good choice for this attribute.
- For a name of a person or a company and an e-mail address, they are also strings, **VARCHAR()** datatype with the length of 50 characters is enough.
- In terms of an address or a note, it needs to store detailed information in the form of string, so we just double the length to 100.

- Age and money are two kinds of attribute that can be counted and do not keep a large value, so **INT** data type is suitable.
- Date will use **DATE** data type and time will use **TIME** data type, obviously.

For full picture, here is the Img file.

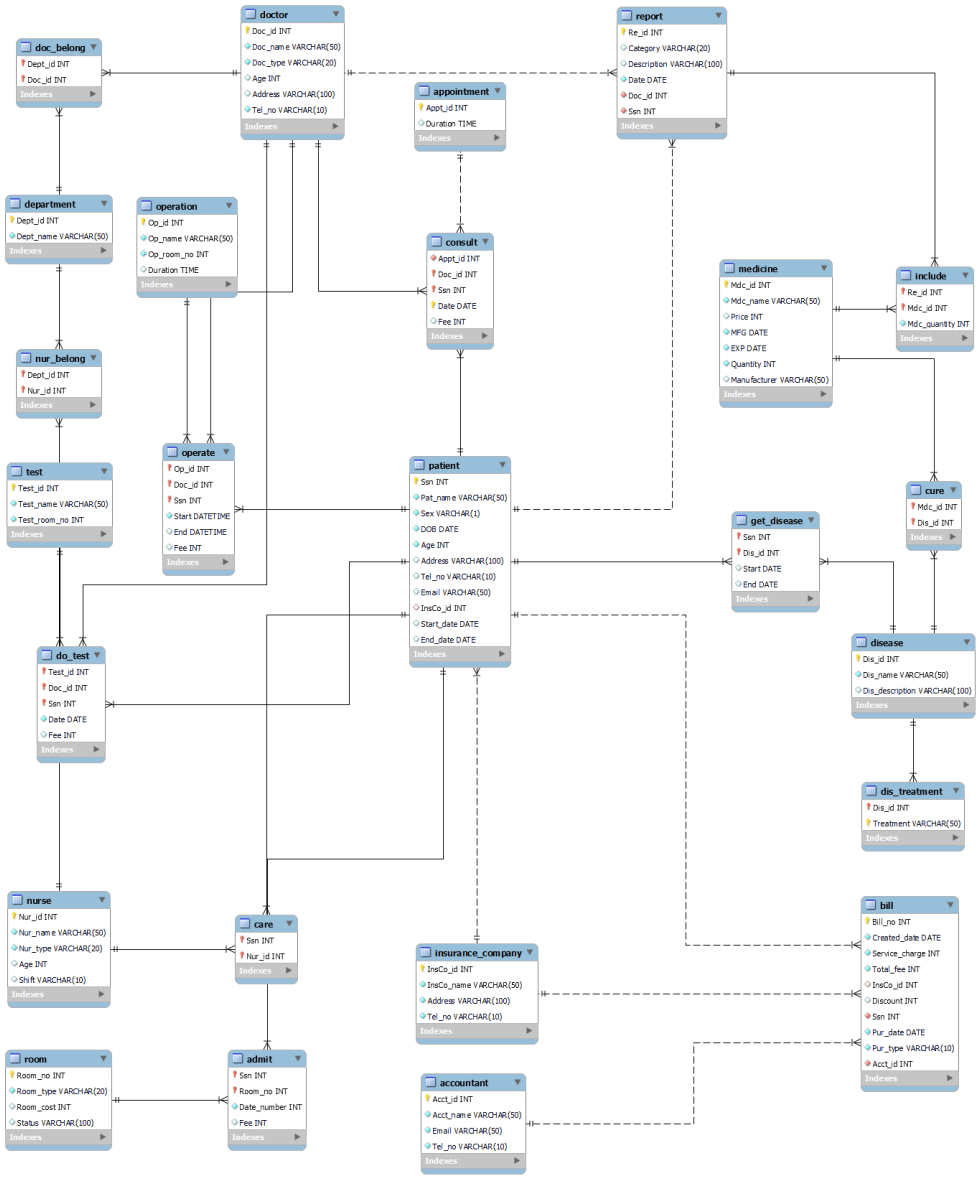

Figure 5: The Relation Schema of the hospital database system

8 Implementation

8.1 Create schema

The first thing we have to do is to create a schema for the hospital database before implementing it into MySQL and then access to it.

```
CREATE SCHEMA Hospital;
USE Hospital;
```

8.2 Create tables

Next, we create 25 tables in total. Looking through the code to create the tables, we will also find out the data type of each attribute.

8.2.1 Insurance company

```
CREATE TABLE Insurance_company (
    InsCo_id INT NOT NULL AUTO_INCREMENT,
    InsCo_name VARCHAR(50) NOT NULL,
    Address VARCHAR(100) NOT NULL,
    Tel_no VARCHAR(10) NOT NULL,
    PRIMARY KEY (InsCo_id)
);
```

8.2.2 Patient

```
CREATE TABLE Patient (
    Ssn INT NOT NULL,
    Pat_name VARCHAR(50) NOT NULL,
    Sex VARCHAR(1) NOT NULL,
    DOB DATE NOT NULL,
    Age INT NOT NULL,
    Address VARCHAR(100),
    Tel_no VARCHAR(10),
    Email VARCHAR(50),
    InsCo_id INT,
    Start_date DATE,
    End_date DATE,
    PRIMARY KEY (Ssn),
    FOREIGN KEY (InsCo_id) REFERENCES Insurance_company(InsCo_id) ON DELETE SET NULL
);
```

8.2.3 Accountant

```
CREATE TABLE Accountant (
    Acct_id INT NOT NULL AUTO_INCREMENT,
    Acct_name VARCHAR(50) NOT NULL,
    Email VARCHAR(50) NOT NULL,
```

```
    Tel_no VARCHAR(10) NOT NULL,
    PRIMARY KEY (Acct_id)
);
-- Set up starting value
ALTER TABLE Accountant AUTO_INCREMENT = 130001;
```

8.2.4 Bill

```
CREATE TABLE Bill (
    Bill_no INT NOT NULL,
    Created_date DATE NOT NULL,
    Service_charge INT NOT NULL,
    Total_fee INT NOT NULL,
    InsCo_id INT,
    Discount INT,
    Ssn INT NOT NULL,
    Pur_date DATE NOT NULL,
    Pur_type VARCHAR(10) NOT NULL,
    Acct_id INT NOT NULL,
    PRIMARY KEY (Bill_no),
    FOREIGN KEY (InsCo_id) REFERENCES Insurance_company(InsCo_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (Acct_id) REFERENCES Accountant(Acct_id)
);
```

8.2.5 Department

```
CREATE TABLE Department (
    Dept_id INT NOT NULL AUTO_INCREMENT,
    Dept_name VARCHAR(50) NOT NULL,
    PRIMARY KEY (Dept_id)
);
```

8.2.6 Doctor

```
CREATE TABLE Doctor (
    Doc_id INT NOT NULL AUTO_INCREMENT,
    Doc_name VARCHAR(50) NOT NULL,
    Doc_type VARCHAR(20) NOT NULL,
    Age INT,
    Address VARCHAR(100),
    Tel_no VARCHAR(10) NOT NULL,
    PRIMARY KEY (Doc_id)
);
-- Set up starting value
ALTER TABLE Doctor AUTO_INCREMENT = 110001;
```

8.2.7 Doc_belong

```
CREATE TABLE Doc_belong (
    Dept_id INT NOT NULL,
    Doc_id INT NOT NULL,
    PRIMARY KEY (Dept_id, Doc_id),
    FOREIGN KEY (Dept_id) REFERENCES Department(Dept_id) ON DELETE CASCADE,
    FOREIGN KEY (Doc_id) REFERENCES Doctor(Doc_id) ON DELETE CASCADE
);
```

8.2.8 Report

```
CREATE TABLE Report (
    Re_id INT NOT NULL,
    Category VARCHAR(20),
    Description VARCHAR(100),
    Date DATE NOT NULL,
    Doc_id INT NOT NULL,
    Ssn INT NOT NULL,
    PRIMARY KEY (Re_id),
    FOREIGN KEY (Doc_id) REFERENCES Doctor(Doc_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE
);
```

8.2.9 Medicine

```
CREATE TABLE Medicine (
    Mdc_id INT NOT NULL AUTO_INCREMENT,
    Mdc_name VARCHAR(50) NOT NULL,
    Price INT,
    MFG DATE NOT NULL,
    EXP DATE NOT NULL,
    Quantity INT NOT NULL,
    Manufacturer VARCHAR(50),
    PRIMARY KEY (Mdc_id)
);
-- Set up starting value
ALTER TABLE Medicine AUTO_INCREMENT = 10001;
```

8.2.10 Include

```
CREATE TABLE Include (
    Re_id INT NOT NULL,
    Mdc_id INT NOT NULL,
    Mdc_quantity INT NOT NULL,
    PRIMARY KEY (Re_id, Mdc_id),
    FOREIGN KEY (Re_id) REFERENCES Report(Re_id) ON DELETE CASCADE,
    FOREIGN KEY (Mdc_id) REFERENCES Medicine(Mdc_id)
);
```

8.2.11 Disease

```
CREATE TABLE Disease (
    Dis_id INT NOT NULL AUTO_INCREMENT,
    Dis_name VARCHAR(50) NOT NULL,
    Dis_description VARCHAR(100),
    PRIMARY KEY (Dis_id)
);
```

8.2.12 Dis_treatment

```
CREATE TABLE Dis_treatment (
    Dis_id INT NOT NULL,
    Treatment VARCHAR(50),
    PRIMARY KEY (Dis_id, Treatment),
    FOREIGN KEY (Dis_id) REFERENCES Disease(Dis_id) ON DELETE CASCADE
);
```

8.2.13 Cure

```
CREATE TABLE Cure (
    Mdc_id INT NOT NULL,
    Dis_id INT NOT NULL,
    PRIMARY KEY (Mdc_id, Dis_id),
    FOREIGN KEY (Mdc_id) REFERENCES Medicine(Mdc_id) ON DELETE CASCADE,
    FOREIGN KEY (Dis_id) REFERENCES Disease(Dis_id) ON DELETE CASCADE
);
```

8.2.14 Get_disease

```
CREATE TABLE Get_disease (
    Ssn INT NOT NULL,
    Dis_id INT NOT NULL,
    Start DATE,
    End DATE,
    PRIMARY KEY (Ssn, Dis_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (Dis_id) REFERENCES Disease(Dis_id) ON DELETE CASCADE
);
```

8.2.15 Test

```
CREATE TABLE Test (
    Test_id INT NOT NULL AUTO_INCREMENT,
    Test_name VARCHAR(50) NOT NULL,
    Test_room_no INT NOT NULL,
    PRIMARY KEY (Test_id)
);
```

8.2.16 Do_test

```
CREATE TABLE Do_test (
    Test_id INT NOT NULL,
    Doc_id INT NOT NULL,
    Ssn INT NOT NULL,
    Date DATE NOT NULL,
    Fee INT,
    PRIMARY KEY (Test_id, Doc_id, Ssn),
    FOREIGN KEY (Test_id) REFERENCES Test(Test_id) ON DELETE CASCADE,
    FOREIGN KEY (Doc_id) REFERENCES Doctor(Doc_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE
);
```

8.2.17 Operation

```
CREATE TABLE Operation (
    Op_id INT NOT NULL AUTO_INCREMENT,
    Op_name VARCHAR(50) NOT NULL,
    Op_room_no INT NOT NULL,
    Duration TIME,
    PRIMARY KEY (Op_id)
);
```

8.2.18 Operate

```
CREATE TABLE Operate (
    Op_id INT NOT NULL,
    Doc_id INT NOT NULL,
    Ssn INT NOT NULL,
    Start DATETIME NOT NULL,
    End DATETIME,
    Fee INT,
    PRIMARY KEY (Op_id, Doc_id, Ssn),
    FOREIGN KEY (Op_id) REFERENCES Operation(Op_id) ON DELETE CASCADE,
    FOREIGN KEY (Doc_id) REFERENCES Doctor(Doc_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE
);
```

8.2.19 Appointment

```
CREATE TABLE Appointment (
    Appt_id INT NOT NULL AUTO_INCREMENT,
    Duration TIME,
    PRIMARY KEY (Appt_id)
);
```

8.2.20 Consult

```
CREATE TABLE Consult (
    Appt_id INT NOT NULL,
    Doc_id INT NOT NULL,
    Ssn INT NOT NULL,
    Date DATE,
    Fee INT,
    PRIMARY KEY (Date, Doc_id, Ssn),
    FOREIGN KEY (Appt_id) REFERENCES Appointment(Appt_id) ON DELETE CASCADE,
    FOREIGN KEY (Doc_id) REFERENCES Doctor(Doc_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE
);
```

8.2.21 Room

```
CREATE TABLE Room (
    Room_no INT NOT NULL AUTO_INCREMENT,
    Room_type VARCHAR(20) NOT NULL,
    Room_cost INT,
    Status VARCHAR(100),
    PRIMARY KEY (Room_no)
);
```

8.2.22 Admit

```
CREATE TABLE Admit (
    Ssn INT NOT NULL,
    Room_no INT NOT NULL,
    Date_number INT NOT NULL,
    Fee INT,
    PRIMARY KEY (Ssn, Room_no),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (Room_no) REFERENCES Room(Room_no)
);
```

8.2.23 Nurse

```
CREATE TABLE Nurse (
    Nur_id INT NOT NULL AUTO_INCREMENT,
    Nur_name VARCHAR(50) NOT NULL,
    Nur_type VARCHAR(20) NOT NULL,
    Age INT,
    Shift VARCHAR(10),
    PRIMARY KEY (Nur_id)
);
-- Set up starting value
ALTER TABLE Nurse AUTO_INCREMENT = 120001;
```

8.2.24 Nur_belong

```
CREATE TABLE Nur_belong (
    Dept_id INT NOT NULL,
    Nur_id INT NOT NULL,
    PRIMARY KEY (Dept_id, Nur_id),
    FOREIGN KEY (Dept_id) REFERENCES Department(Dept_id) ON DELETE CASCADE,
    FOREIGN KEY (Nur_id) REFERENCES Nurse(Nur_id) ON DELETE CASCADE
);
```

8.2.25 Care

```
CREATE TABLE Care (
    Ssn INT NOT NULL,
    Nur_id INT NOT NULL,
    PRIMARY KEY (Ssn, Nur_id),
    FOREIGN KEY (Ssn) REFERENCES Patient(Ssn) ON DELETE CASCADE,
    FOREIGN KEY (Nur_id) REFERENCES Nurse(Nur_id)
);
```

After fully created our tables, we have fully working Database System.

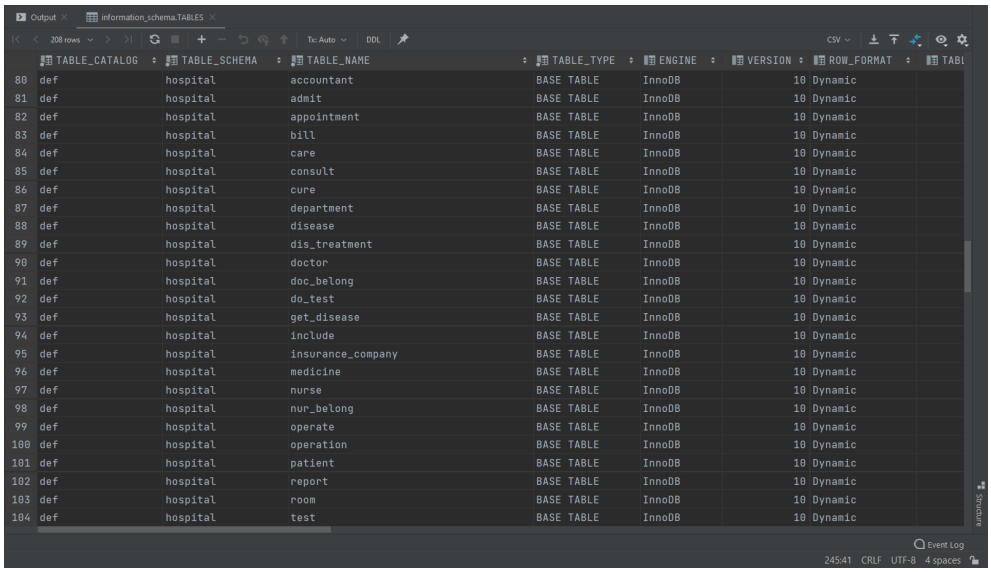

Figure 6: Created table result

8.3 Insert data

After having already created tables, we try to input some data into the database to check whether our design works well or not.

```
-- INSERT command
INSERT INTO Insurance_company
VALUES (1, 'American Health', '2157 Swain Rd, Eaton, Ohio', 9374562728),
```

```
       (2, 'Health Star', '250 Lone Pine Rd, Saint Landry, Louisiana', 3188382464);

INSERT INTO Patient
VALUES (292567083, 'Dare Lucas', 'M', '1980-02-14',        41,        '7416 Ireland Ct, El Paso,Texas',
        9452170294, 'lucas123@gmail.com', 1,        '2020-12-19', '2022-12-18'),
       (415310368, 'Raynor Kylee', 'F', '1982-01-02', 39, '10926 Highwood Way, El Paso, Texas',
        9702197670, 'kylee0201@hotmail.com', NULL, NULL, NULL),
       (575419103, 'Olson Tracy', 'F', '1956-11-25', 65, '730 Field Ave, Taft, Texas',
        NULL, NULL, 2, '2020-11-19',        '2024-11-18'),
       (151191515, 'Olson Kate', 'F', '1995-05-19', 26, '4568 Green Acres Road, Coinjock, North Carolina',
        2522024018, 'katesil@gmail.com', 2, '2020-12-19', '2022-12-18'),
       (414783198, 'John J Keener', 'M', '1961-10-08', 60, '2401 Corbin Branch Road, IOWA CITY, Iowa',
        7123923698, 'johnk@hotmail.com', 1, '2020-11-19', '2024-11-18');

INSERT INTO Accountant
VALUES (130001, 'Stokes Ashlee', 'ashleelee@gmail.com', 6012484710),
       (130002, 'Parisian Helen', 'helen321@hotmail.com', 5805694574);

INSERT INTO Bill
VALUES (61461265, '2021-06-06',        100, 50, 1,        50,        292567083, '2021-06-06', 'cash', 130001),
       (98347142, '2021-06-11',        100, 100, NULL,        0, 415310368, '2021-06-11',        'cheque', 130002
       (91479336, '2021-08-26',        5000, 100, 2, 4900,        575419103, '2021-08-26', 'cash', 130001),
       (17173171, '2021-09-02', 200, 150, 2, 50, 151191515, '2021-09-02', 'creditcard', 130001),
       (77781785, '2021-09-26', 150, 50, 1, 100, 414783198, '2021-09-26', 'creditcard', 130002),
       (72735717, '2021-10-15', 200, 200, NULL, 0, 415310368, '2021-10-15', 'cheque', 130001);

INSERT INTO Department
VALUES (1, 'cardiology'),
       (2, 'general internal medicine');

INSERT INTO Doctor
VALUES (110001, 'Ziemann Timothy', 'ENT specialist', 28, '303 Valmar, Kemah, Texas', 5753361371),
       (110002,        'White Warren',        'orthopaedic surgeon', 33, '1105 Clover Dr, Burkburnett, Texas', 58
       (110003,        'Wehner Nico', 'cardiologist', 40, '369 Arnold Dr, Gordonville, Texas',        5809202612)

INSERT INTO Doc_belong
VALUES (2, 110001),
       (2, 110002),
       (1 ,110003);

INSERT INTO Report
VALUES (41614184, 'test', 'Re-examination next 2 weeks', '2021-06-06', 110001, 292567083),
       (86116642, 'consult', 'Just need to use medicine', '2021-06-11',        110001, 415310368),
       (19782716, 'operation', 'Recoverd, re-examine next month', '2021-08-26',        110003, 575419103),
       (73717313, 'test', 'Re-examination next 2 weeks', '2021-09-02', 110002, 151191515),
       (71375531, 'consult', 'Just need to use medicine', '2021-09-26', 110003, 414783198),
       (34666337, 'consult', 'Just need to use medicine', '2021-10-15', 110001, 415310368);

INSERT INTO Medicine
VALUES (10001, 'aspirin', 4, '2020-12-15', '2022-12-15', 200, 'Pfizer'),
       (10002, 'acetaminophen',        5, '2021-01-01', '2023-01-01', 300,        'Johnson & Johnson'),
       (10003, 'tylenol', 3, '2021-03-01', '2023-03-01', 400, 'Pfizer');
```

```
INSERT INTO Include
VALUES (86116642, 10001, 10),
       (71375531, 10002, 15),
       (34666337, 10003, 20);

INSERT INTO Disease
VALUES (1, 'Cold and Flu', 'Viruses cause both colds and flu by
increasing inflammation of the membranes in the nose and throat.'),
       (2, 'Headaches',        'Affects a specific point of the head,
often the eye, and is characterized by sharp, piercing pain.');

INSERT INTO Dis_treatment
VALUES (1, 'Drink lots of clear fluids (e.g., water, tea)'),
       (2, 'Ice pack held over the eyes or forehead'),
       (2, 'Sleep, or at least resting in a dark room');

INSERT INTO Cure
VALUES (10001, 2),
       (10002, 2),
       (10003, 1);

INSERT INTO Get_disease
VALUES (415310368, 1, '2021-06-10', '2021-06-15'),
       (414783198, 2, '2021-09-25', '2021-09-30'),
       (415310368, 2, '2021-10-14', '2021-10-18');

INSERT INTO Test
VALUES (1, 'ear checking', 4),
       (2, 'COVID-19', 4);

INSERT INTO Do_test
VALUES (1, 110001, 292567083, '2021-06-06', 100),
       (2, 110002, 151191515, '2021-09-02', 200);

INSERT INTO Operation
VALUES (1, 'heart operation', 3, '11:30:00');

INSERT INTO Operate
VALUES (1, 110003, 575419103, '2021-08-17 08:00:00', '2021-08-17 19:30:00', 5000);

INSERT INTO Appointment
VALUES (1, '00:15:00'),
       (2, '00:30:00'),
       (3, '00:25:00');

INSERT INTO Consult
VALUES (1, 110001, 415310368, '2021-06-11', 10),
       (2, 110003, 414783198, '2021-09-26', 10),
       (3, 110001, 415310368, '2021-10-15', 15);

INSERT INTO Room
VALUES (1, 'recovery', 50, 'available'),
       (2, 'recovery', 100,        'available'),
```

```
       (3, 'surgery', NULL,        'used for surgery'),
       (4, 'test', NULL, 'used for test');

INSERT INTO Admit
VALUES (575419103, 2 ,10, 1000);

INSERT INTO Nurse
VALUES (120001, 'Willms Amy', 'cardiac', 40, 'day'),
       (120002,        'Schmitt Erica', 'registered', 30, 'day'),
       (120003,        'Blick Ken', 'registered', 34, 'night');

INSERT INTO Nur_belong
VALUES (1, 120001),
       (2, 120002),
       (2, 120003);

INSERT INTO Care
VALUES (575419103, 120001),
       (575419103, 120003);
```

8.4 Rename tables

If we want to rename a table, we can use **RENAME** to do that.

```
RENAME TABLE Patient TO New_Patient;
```

Also, we can use **ALTER TABLE** as a second way to rename a table.

```
ALTER TABLE New_Patient RENAME Patient;
```

8.5 Delete command

```
-- DELETE command
-- delete a row in Bill table
DELETE FROM Bill
WHERE Bill_no = 72735717;
```

8.6 Update command

```
-- UPDATE command
-- Update Tel_no and Email of Olson Tracy
UPDATE Patient
SET Tel_no = 1561561156, Email = 'tracy@gmail.com'
WHERE Pat_name = 'Olson Tracy';
```

8.7 Select command from basic to advanced

```sql
-- SELECT command
-- select all Report table
SELECT *
FROM Report;

-- select MFG and EXP of all Medicine
SELECT Mdc_name, MFG, EXP
FROM Medicine;

-- select which patient's age >= 40
SELECT Ssn, Pat_name, Sex, DOB, Age, Tel_no
FROM Patient
Where Age >= 40;

-- Using Alias when selecting female patients
SELECT P.Ssn, P.Pat_name AS 'Full name', P.Sex AS 'Gender', P.DOB
FROM Patient AS P
WHERE P.Sex = 'F';

-- SELECT ALL is the default SELECT: use this to see the room type
SELECT ALL Room_type
FROM Room;

-- SELECT DISTINCT eliminate duplication: use this to see the room type as above
SELECT DISTINCT Room_type
FROM Room;

-- Patient: search based on Pat_name
SELECT Pat_name, Sex, DOB, Address, Tel_no, Email
FROM Patient
WHERE Pat_name LIKE CONCAT('%', 'Olson', '%');

-- Patient: whose birthdate on November
SELECT Pat_name, Sex, DOB, Address, Tel_no, Email
FROM Patient
WHERE DOB LIKE '_____11___';

-- Aggregate Functions when search based on Date1 -> Date2
SELECT COUNT(Bill_no) AS Total_bill,
       SUM(Total_fee) AS Total_revenue,
       AVG(Total_fee) AS Average_revenue,
       MAX(Total_fee) AS Max_bill_fee,
       MIN(Total_fee) AS Min_bill_fee
FROM Bill, Patient, Accountant
WHERE (Pur_date BETWEEN '2021-08-26' AND '2021-09-26')
  AND Bill.Ssn = Patient.Ssn AND Bill.Acct_id = Accountant.Acct_id;

-- arrange patient descending based on Age >= 30
SELECT *
FROM Patient
Where Age >= 30
ORDER BY Age DESC;
```

```sql
-- Select patients who do not come from Iowa
SELECT *
FROM Patient
WHERE Ssn NOT IN (SELECT Ssn
                  FROM Patient
                  WHERE Address LIKE '%Iowa');

-- Select patients who have insurance, this mean that the value of InsCo_id is not NULL
SELECT *
FROM Patient
WHERE InsCo_id IS NOT NULL;

-- Inner Join: Medicine, Cure, Disease
SELECT Medicine.Mdc_name, Medicine.Manufacturer, Disease.Dis_name
FROM ((Medicine
INNER JOIN Cure ON Medicine.Mdc_id = Cure.Mdc_id)
INNER JOIN Disease ON Cure.Dis_id = Disease.Dis_id);

-- Left Join: Bill, Accountant
SELECT Bill.Bill_no, Bill.Created_date, A.Acct_name
FROM Bill
LEFT JOIN Accountant A ON A.Acct_id = Bill.Acct_id;

-- UNION: Doctor, nurse, accountant as staff
SELECT Doc_id AS 'Staff_id', Doc_name AS 'Staff_name'
FROM Doctor
WHERE Doc_type = 'cardiologist'
UNION
SELECT Nur_id, Nur_name
FROM Nurse
WHERE Shift = 'day'
UNION
SELECT Acct_id, Acct_name
FROM Accountant
WHERE Acct_name = 'Stokes Ashlee';

-- See how many patients are male and female
SELECT Sex AS 'Gender', COUNT(Ssn)
FROM Patient
GROUP BY Sex;

-- Using EXISTS to check are there any doctor belong to cardiology department
-- and is there no nurse belong to general internal medicine department
SELECT Dept_id, Dept_name
FROM Department
WHERE EXISTS(SELECT *
             FROM Doctor, Doc_belong, Department
             WHERE Doctor.Doc_id = Doc_belong.Doc_id
               AND Doc_belong.Dept_id = Department.Dept_id
               AND Dept_name = 'cardiology')
      AND
      NOT EXISTS(SELECT *
                 FROM Nurse, Nur_belong, Department
```

```
WHERE Nurse.Nur_id = Nur_belong.Nur_id
  AND Nur_belong.Dept_id = Department.Dept_id
  AND Dept_name = 'general internal medicine');
```

8.8 Functions, Procedures and Triggers

8.8.1 Functions

A function is a database object in SQL Server. Basically, it is also a set of SQL statements that accept only input parameters and produce output in a single value form or tabular form. Function can be called from both stored procedure and trigger. However, function only accepts in parameter and returns the executed values.

8.8.2 Stored procedures

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again. So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it. You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed. The file **procedure.sql** contains the source code of some helpful functions for the hospital system.

8.8.3 Triggers

A trigger is also a set of SQL statements in the database. In contrast to previous ones, a trigger is executed automatically whenever any special event like insert, delete, update, etc. occurs in the database rather than being invoked. Trigger does not accept any type of parameter, it just uses the NEW and OLD variables to get the value that is being affected. The following table illustrates the availability of the OLD and NEW modifiers.

Trigger Event	OLD	NEW
INSERT	No	Yes
UPDATE	Yes	Yes
DELETE	Yes	No

Figure 7: Trigger modifiers

8.9 Indexing

8.9.1 Overview of indexing

Indexes are **special lookup tables** that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

An index helps to speed up **SELECT** queries and **WHERE** clauses, but it slows down data input, with the **UPDATE** and the **INSERT** statements. Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against. Indexes can be created or dropped with no effect on the data.

Creating an index involves the **CREATE INDEX** statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.

Indexes can also be unique, like the **UNIQUE** constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

8.9.2 CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

```
CREATE INDEX index_name
ON table_name;
```

For example, the following SQL syntax creates a new table called **Patient** and adds 11 columns in it.

```
CREATE TABLE Patient (
    Ssn INT NOT NULL,
    Pat_name VARCHAR(50) NOT NULL,
    Sex VARCHAR(1) NOT NULL,
    DOB DATE NOT NULL,
    Age INT NOT NULL,
    Address VARCHAR(100),
    Tel_no VARCHAR(10),
    Email VARCHAR(50),
    InsCo_id INT,
    Start_date DATE,
    End_date DATE,
    PRIMARY KEY (Ssn),
    FOREIGN KEY (InsCo_id) REFERENCES Insurance_company(InsCo_id) ON DELETE SET NULL
);
```

To create an **INDEX** on the **Age** column, to optimize the search on **Patient** for a specific age, we use the follow SQL syntax which is given below

```
CREATE INDEX idx_age
ON Patient (Age);
```

8.9.3 Listing indexes

Tables can have multiple indexes. Managing indexes will inevitably require being able to list the existing indexes on a table. The following syntax for viewing indexes in **Patient** table.

```
SHOW INDEX FROM Patient;
```

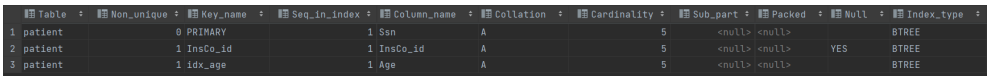

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
1	patient	0	PRIMARY	1	Ssn	A	5	<null>	<null>		BTREE
2	patient	1	InsCo_id	1	InsCo_id	A	5	<null>	<null>	YES	BTREE
3	patient	1	idx_age	1	Age	A	5	<null>	<null>		BTREE

Figure 8: Indexes of Patient

8.9.4 DROP INDEX Command

To drop an INDEX constraint from **Patient**, use the following SQL syntax.

```
ALTER TABLE Patient
DROP INDEX idx_age;
```

8.9.5 When should indexing be avoided?

Although indexes are intended to enhance a database's performance, there are times when they should be avoided. Followings are some scenarios when the use of an index should be reconsidered.

- Columns that are frequently manipulated should not be indexed.
- Indexes should not be used on columns that contain a high number of NULL values.
- Tables that have frequent, large batch updates or insert operations.
- Indexes should not be used on small tables.

8.10 Delete tables & schema

After all, if we want to delete all 23 tables, the following order helps us to avoid FOREIGN KEY constraint fails.

```
-- DROP 25 TABLES
-- Group 1: Tables referring to group 2 tables
DROP TABLE Bill;
DROP TABLE Doc_belong;
DROP TABLE Include;
DROP TABLE Report;
DROP TABLE Dis_treatment;
DROP TABLE Cure;
DROP TABLE Get_disease;
DROP TABLE Do_test;
DROP TABLE Operate;
DROP TABLE Consult;
DROP TABLE Admit;
DROP TABLE Nur_belong;
```

```
DROP TABLE Care;
-- Group 2: Tables referred by group 1 tables and referring to group 3 tables
DROP TABLE Patient;
-- Group 3: Tables referred by group 2 tables
DROP TABLE Insurance_company;
DROP TABLE Accountant;
DROP TABLE Department;
DROP TABLE Doctor;
DROP TABLE Medicine;
DROP TABLE Disease;
DROP TABLE Test;
DROP TABLE Operation;
DROP TABLE Appointment;
DROP TABLE Room;
DROP TABLE Nurse;
```

The final operation is deleting the **Hospital** schema.

```
DROP SCHEMA Hospital;
```

9 Security

9.1 Problem analysis

In a business context, the system we designed consists of a DBMS managed database and the above Web application to provide managers at two different levels to view and manage data. with appropriate limits. The app is used mainly by both area managers and store managers. However, their roles and privileges in the system are different. In a multi-user database system, the DBMS is responsible for providing techniques to allow certain users or groups of users to access selected parts of the database without access to the rest of the database. This is especially important in this retail organization. For example, some sensitive information such as the salaries of employees at different stores and their performance should be kept secret from most users of the database system in this organization. Another issue is about the privacy and integrity of data as they have to be transmitted over several communication channels. Some confidential information may be lost or fragmented if no security measures are taken.

To protect the database from the above potential threats, we need to implement some control measures which will be discussed in more detail later.

9.2 Access Control, User Accounts, and Database Testing

Whenever a person or group of people needs to access a database system, that individual or group of people must first register a user account. The DBA will then create a new account number and password for the user if there is a legitimate need to access the database. The user must log in to the DBMS by entering the account number and password whenever accessing the database is needed. The DBMS checks if the account number and password are valid; if yes, the user is allowed to use the DBMS and access the database. Application programs can also be considered users and are required to log in to the database.

It's easy to track database users and their accounts and passwords by creating an encrypted table or file with two fields: AccountNumber and Password. This table can easily be maintained by the DBMS. Whenever a new account is created, a new record is inserted in the table. When an account is activated, the corresponding record must be deleted from the table.
The database system must also keep track of all database operations applied by a given user during each login session, including the sequence of database interactions that the user performs. displayed from login to logout. When a user logs in, the DBMS can record the user's account number and associate that account number with the computer or device on which the user is logged in. All activities applied from that computer or device are attributed to the user's account until the user logs out. It is especially important to track the update operations applied to the database so that, if the database is processed, the DBA can determine which user has performed the tampering.

9.3 Role-Based Access Control (RBAC)

Role-based access control is a method of restricting network access based on an individual user's role in an enterprise. RBAC allows employees to only have access to the information they need to do their jobs and prevents them from accessing information that is not relevant to them.
Advantages of RBAC:

- **Improve operational efficiency.** With RBAC, companies can reduce the need for paperwork and password changes when they hire new employees or switch roles of existing employees. It also cuts down on the possibility of errors when user permissions are being specified.
- **Enhance compliance.** All participants must comply with local, state, and federal regulations.
- **Increased administrator visibility.** RBAC gives network administrators and managers more visibility and monitoring over the business, and ensures that authorized users and guests on the system are only given access to what they need to do their job.
- **Discount.** By disallowing users from accessing certain processes and applications, companies can save or make more efficient use of resources, such as network bandwidth, memory, and storage.
- **Reduce the risk of data breaches and leaks.** Implementing RBAC means restricting access to sensitive information, thereby reducing the possibility of a data breach or data leak.

Disadvantages of RBAC:

- **Difficult to manage and maintain.** Usually, admins will keep adding roles for users but never remove them. You end up with users that dozens if not hundreds of roles and permissions.
- **It is static.** RBAC cannot use contextual information e.g. time, user location, device type...
- **Evaluating access rights is difficult, error-prone, and lengthy.** Also, permissions can only be assigned to user roles, not objects and activities.

10 Application

10.1 Flask

Flask is a web framework, it's a Python module that lets you develop web applications easily. It's has a small and easy-to-extend core: it's a micro-framework. Micro" does not mean that your whole web application has to fit into a single Python file (although it certainly can), nor does it mean that Flask is lacking in functionality. The "micro" in micro-framework means Flask aims to keep the core simple but extensible. Flask won't make many decisions for you, such as what database to use. Those decisions that it does make, such as what templating engine to use, are easy to change. Everything else is up to you, so that Flask can be everything you need and nothing you don't. Because it's focus on simplicity, it's perfect for small assignment

10.2 Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS- and (optionally) JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.
Bootstrap is an HTML, CSS and JS Library that focuses on simplifying the development of informative web pages (as opposed to web apps). The primary purpose of adding it to a web project is to apply Bootstrap's choices of color, size, font and layout to that project. As such, the primary factor is whether the developers in charge find those choices to their liking. Once added to a project, Bootstrap provides basic style definitions for all HTML elements. The result is a uniform appearance for prose, tables and form elements across web browsers. In addition, developers can take advantage of CSS classes defined in Bootstrap to further customize the appearance of their contents. For example, Bootstrap has provisioned for light- and dark-colored tables, page headings, more prominent pull quotes, and text with a highlight.

10.3 Demo of the application

Github repo: https://github.com/tombroskipc/Hospital-DBMS

10.3.1 Find records

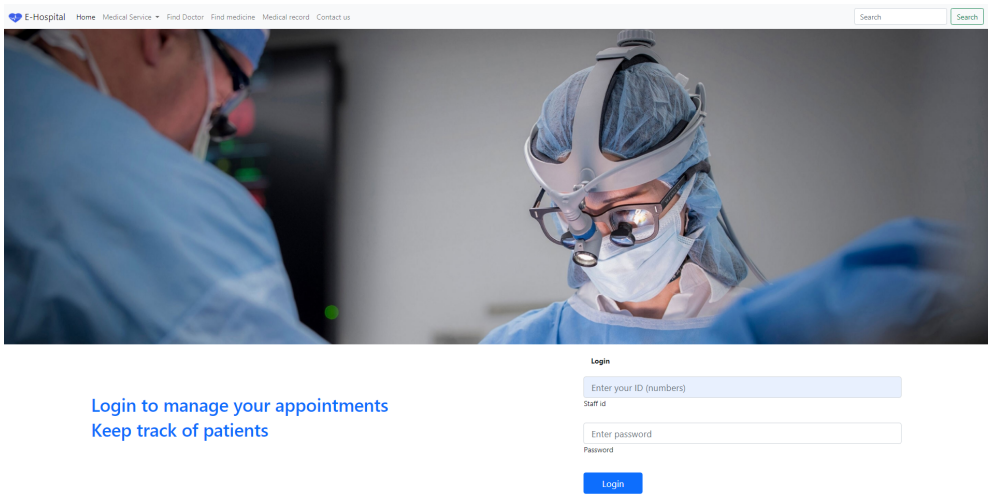

Figure 9: Login page

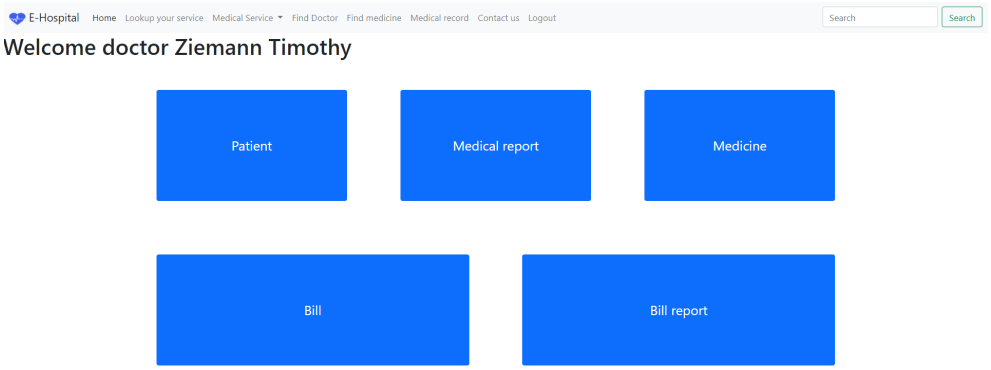

Figure 10: Dashboard

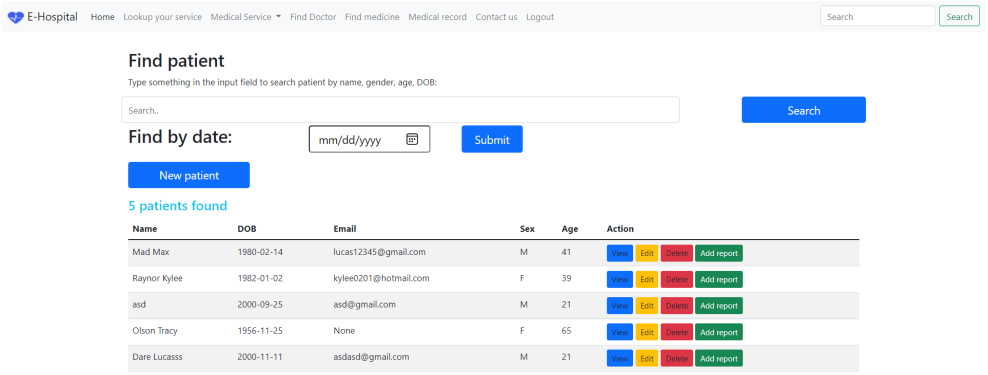

Figure 11: View all patients

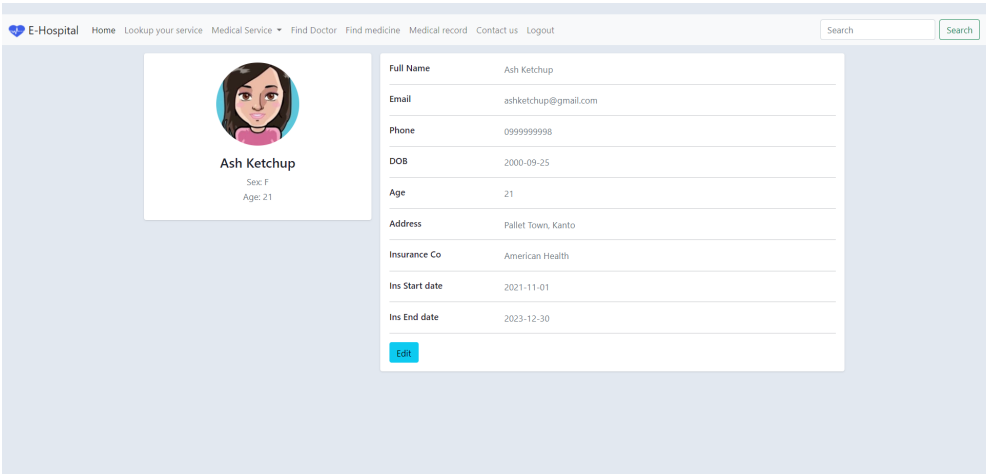

Figure 12: view detail about patient

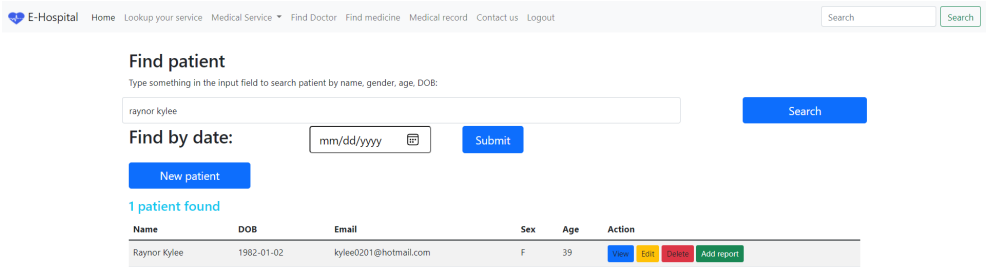

Figure 13: Find patients by name

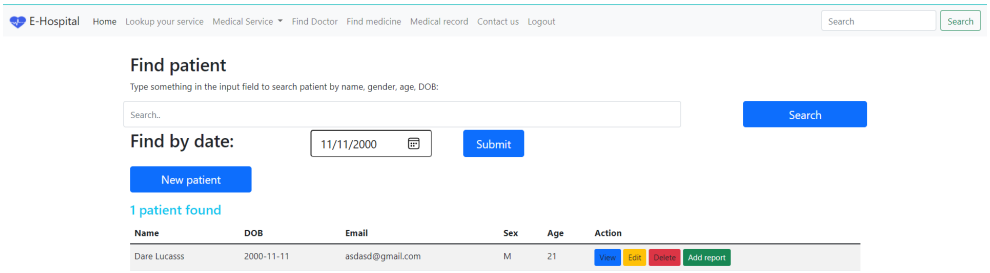

Figure 14: Find patients by DOB

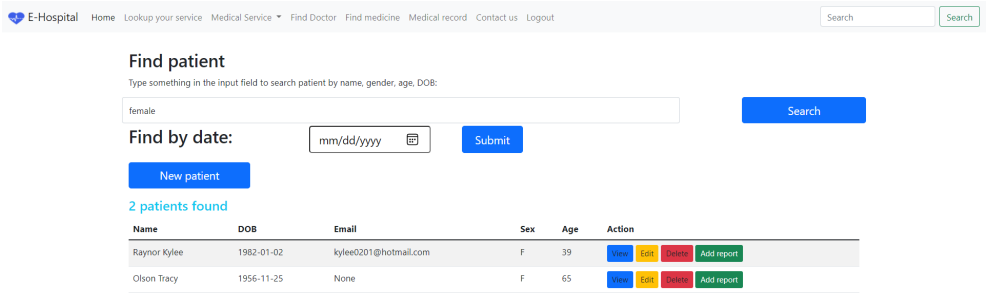

Figure 15: Find patients by gender

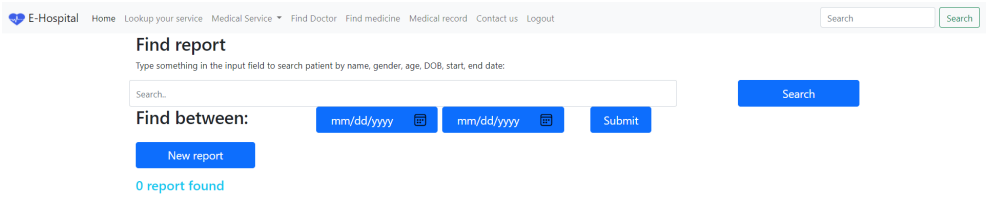

Figure 16: Find medical record page

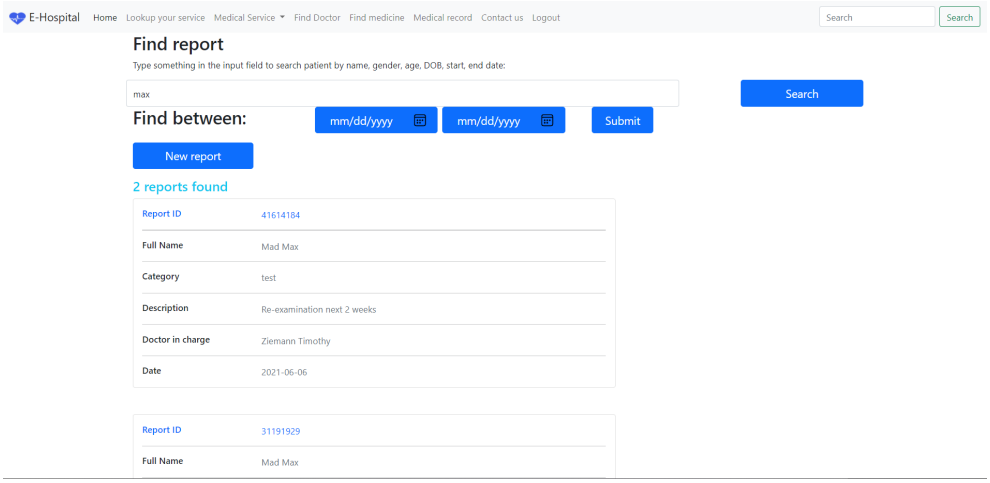

Figure 17: Find medical records by name

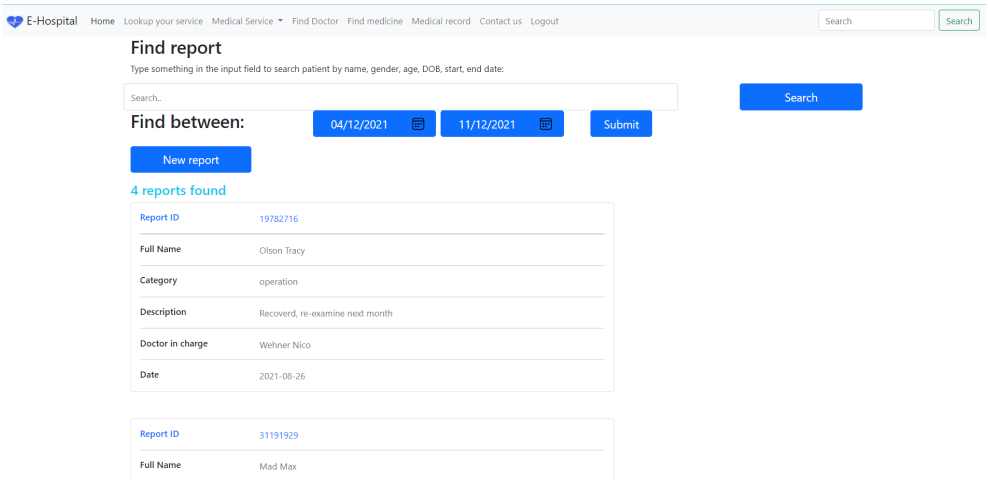

Figure 18: Find medical records between 2 days

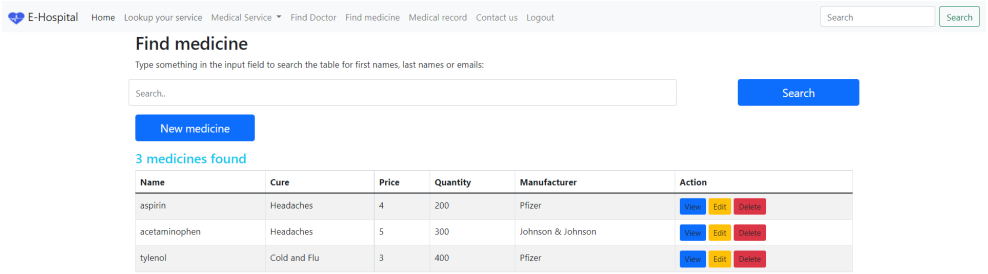

Figure 19: View all medicines

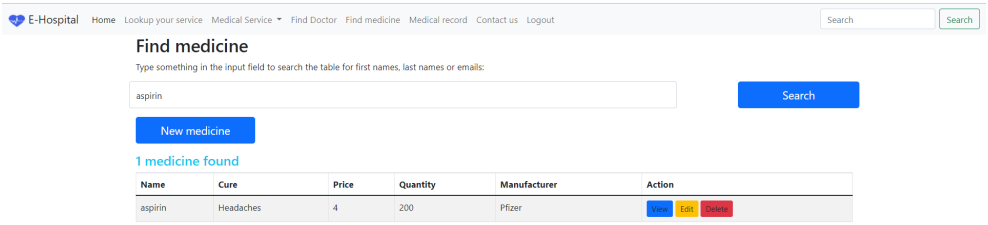

Figure 20: Find medicine by name

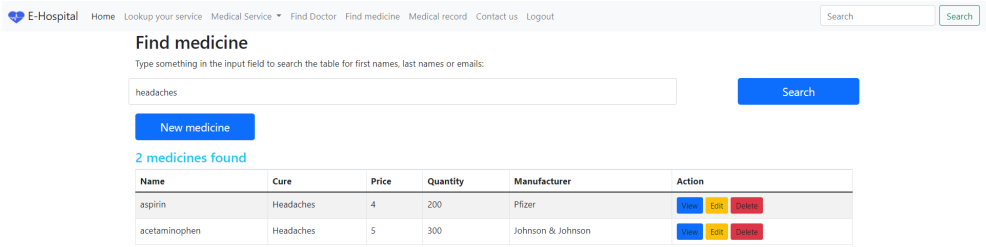

Figure 21: Find medicine by disease

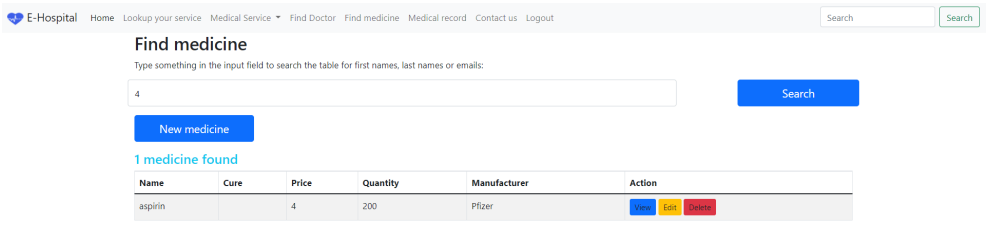

Figure 22: Find medicine by price

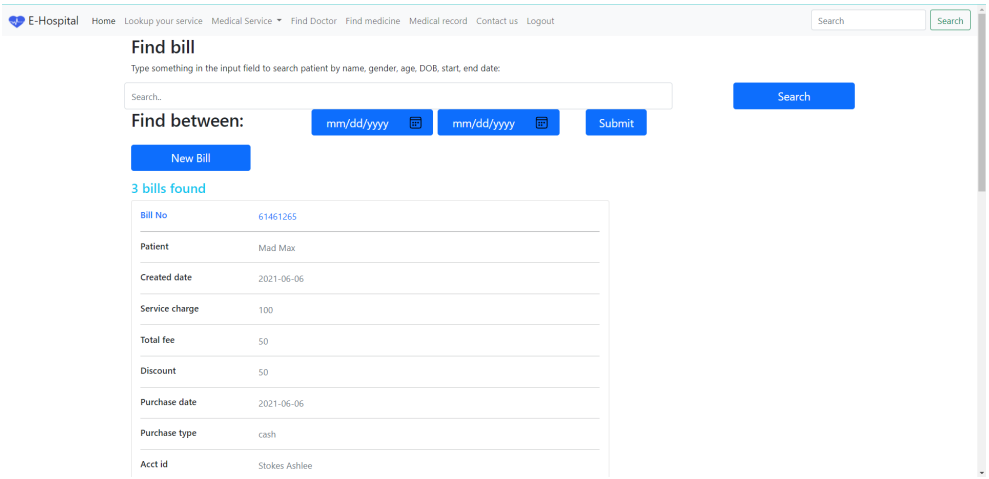

Figure 23: View all bills

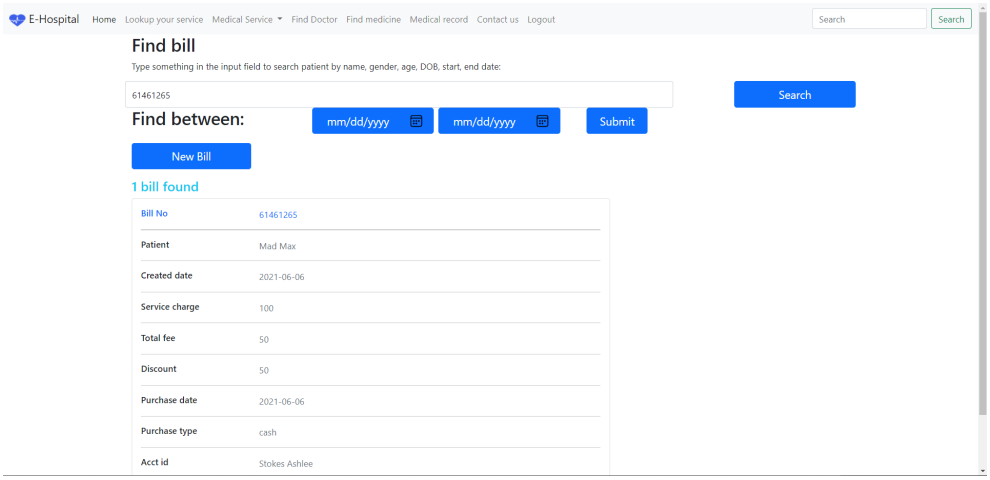

Figure 24: Find bill by bill no

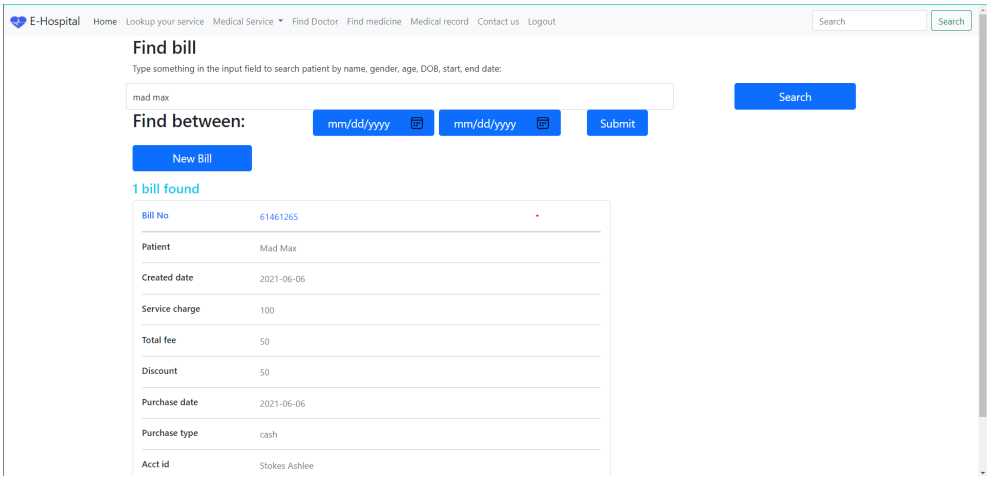

Figure 25: Find bill by patient name

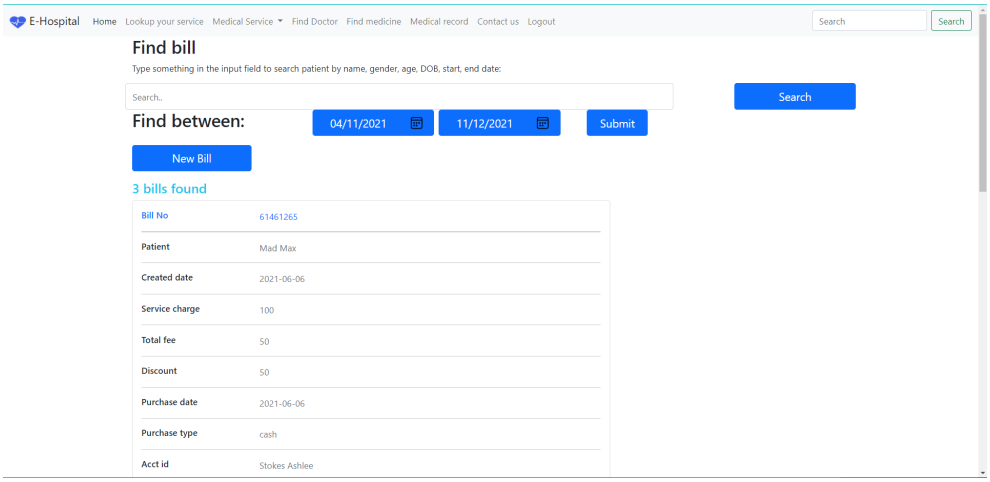

Figure 26: Find bill between 2 days

10.3.2 Insert data

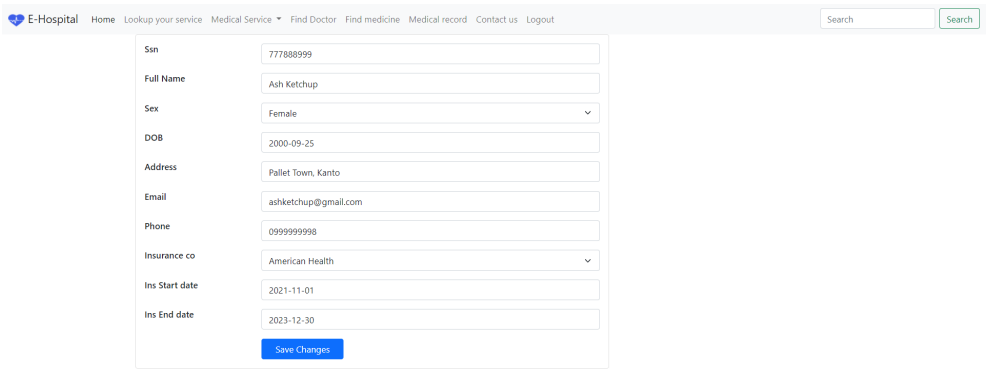

Figure 27: Add new patient

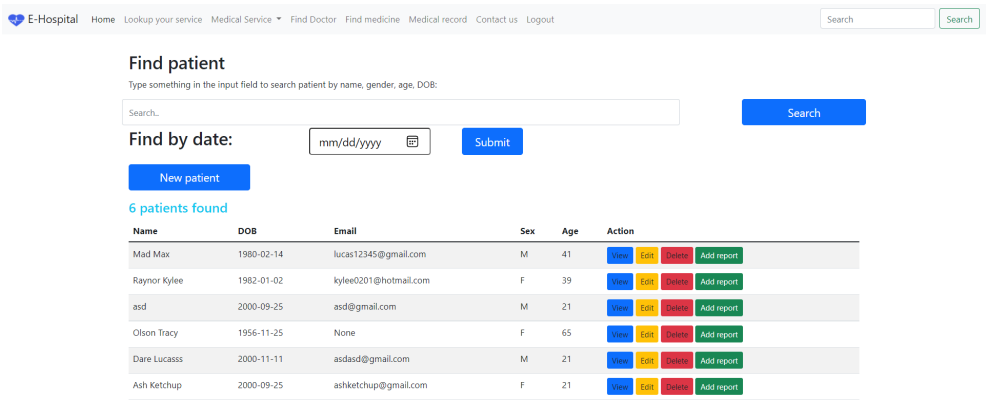

Figure 28: After adding patient

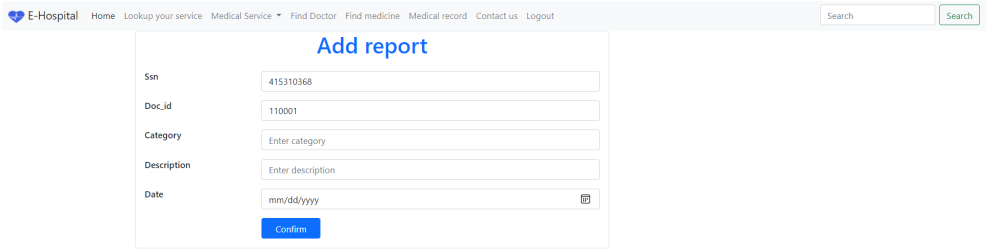

Figure 29: Add report with doctor and patient cache

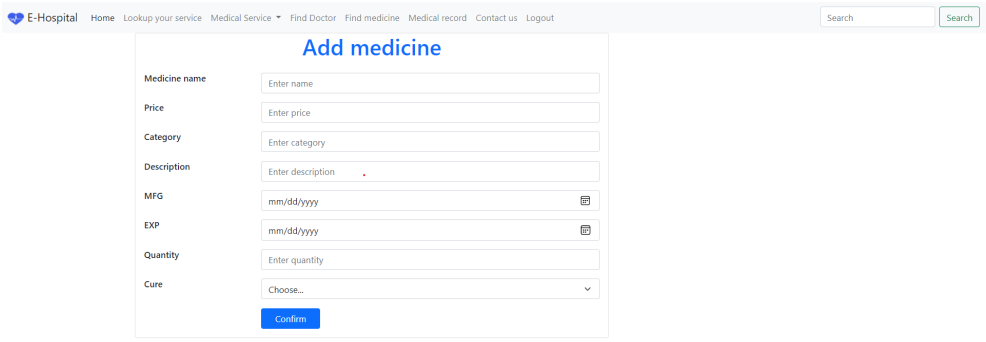

Figure 30: Add medicine

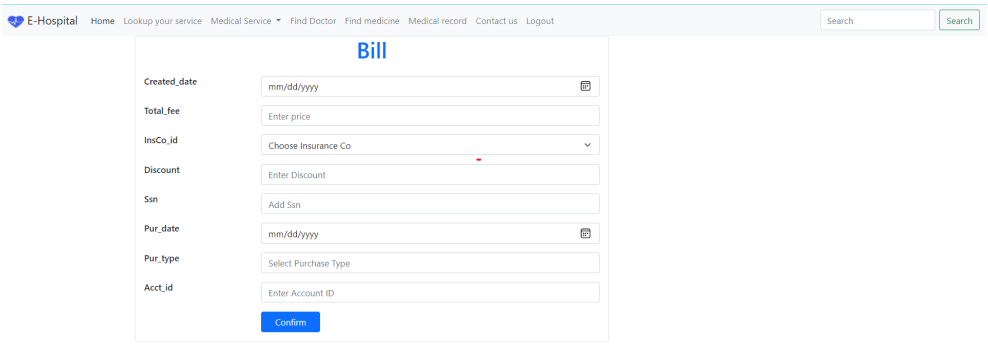

Figure 31: Add bill

10.3.3 Update or Delete

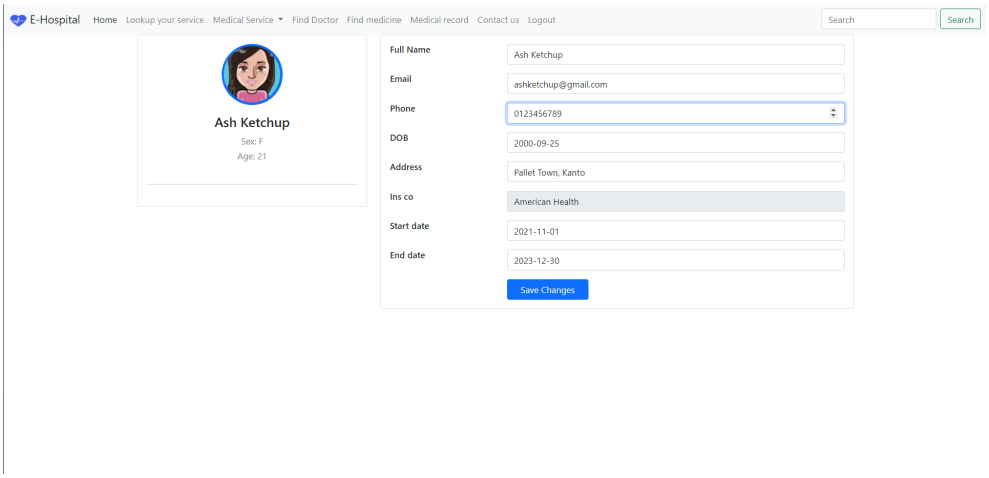

Figure 32: Edit patient information

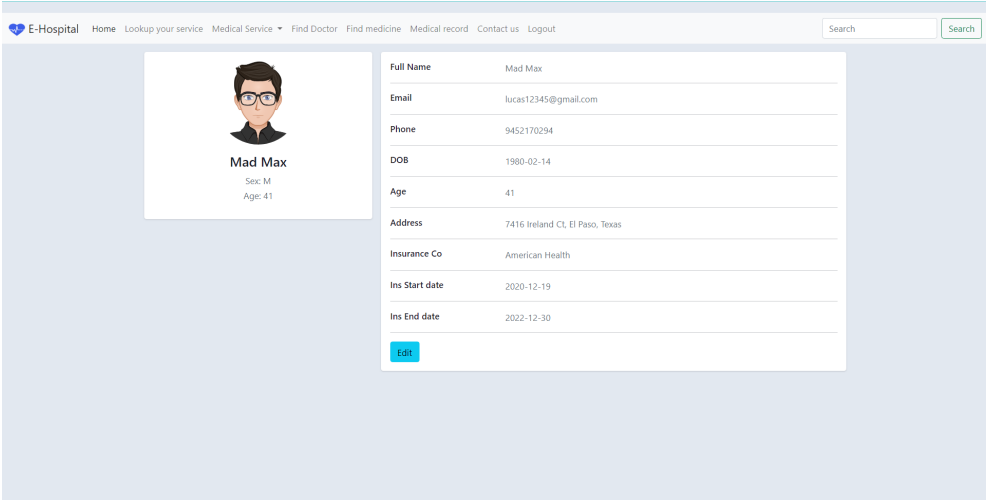

Figure 33

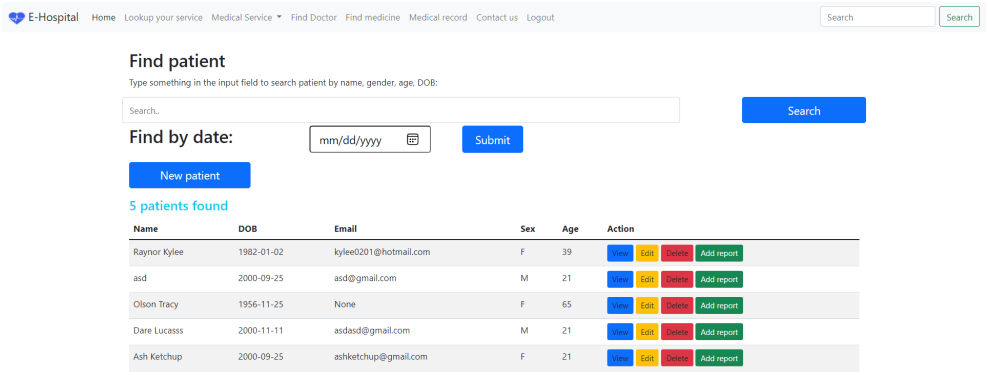

Figure 34: Patient after delete record name Mad Max

11 Conclusion

This Hospital Database Management System was made to make our system more strongly reality-implemented and maintain the reader appropriate information to handle errors or manipulate easily.

By the time Covid-19 in most area increased thus the database got bigger and bigger. Hence, Hospital Database Management is probably necessary for storing variety and large amount of database. Moreover, this requirement has been researched and developed more since the population of countries witnessed continuous bigger growth.

However, we still struggle to design our pattern in some aspect when we cannot use all the function that DBMS tool can support us for free at all . Not entirely our table may get some trouble, but we ensure the best replaceable method that we can learn and achieve from academic environment.

References

[1] What is mysql? everything you need to know, Oct 2020.

[2] Ramez Elmasri and Sham Navathe. *Fundamentals of Database Systems*, chapter 9, pages 291–296. Pearson, 2020.