**WeCare Hospital Management System**

****

Hospital Management System Final Report

Team #19: Nicholas Hsiao, Huynh Phan, Diana Sok

December 11, 2019

Instructor: Dr. Mike-Wu

**Table of Contents**

Table of Contents

1. Project Overview

1.1. Stakeholders

1.2. Relevance

2. System Environment

3. Functional Requirements

4. Non-Functional Requirements

5. Project Data Model

6. Database Normalization (BCNF)

## 

## **1. Project Overview**

For this database application project, we will be developing a hospital management system with both user and administrative functions. Users (patients) will be able to register as a new user and fill in a medical profile or continue as a registered user and have access to their medical profile and appointments. Users will also be able to schedule and update appointments. Administrators (doctors and nurses) will have read and write access patient medical records and be able to confirm, view and update all upcoming appointments. This project will be a great opportunity to build a real-world full stack application.

## **1.1 Stakeholders**

***Table 1.*** *Stakeholders, Interest, and Potential*

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Interest** | **Potential** |
| Patients | * Difficulty in scheduling appointments * Difficulty in retrieving updated and current medical profile | * Ease in scheduling * Improved communication * Quick access to up-to-date information * Improved customer experience |
| Doctors/Nurses | * Limited time per patient * Delay in information exchange | * Quick access to a patient’s medical history * Ease of access to schedule |
| Project Team | * Fulfill product expectations of clients within technical reason | * Creation of hospital management system |

***Table 2.*** *Stakeholders, Concerns, and Strategy*

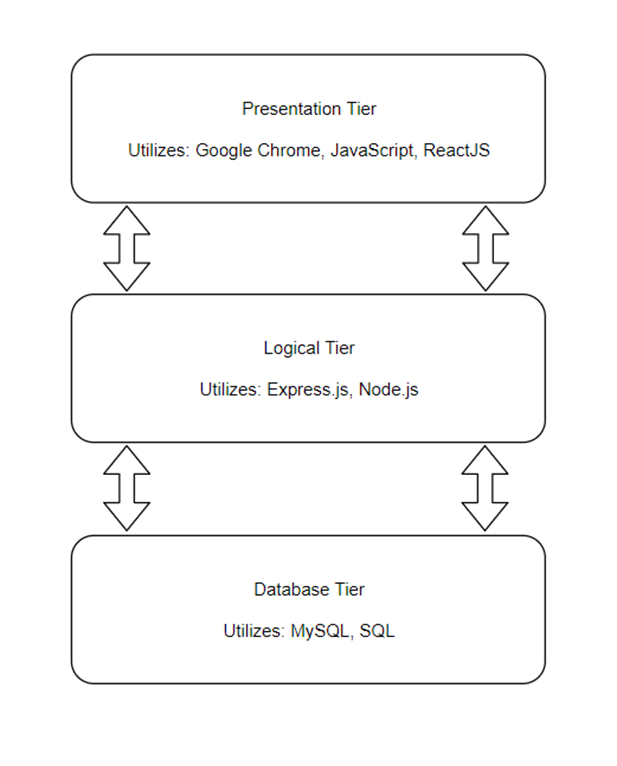
|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Concerns** | **Strategy** |
| Patients | * Scheduling conflicts * Accuracy of medical profile * Data breaches | * Ensure only authorized accounts can modify data * Prevent overlapping appointments * Defined appointment time slots |
| Doctors/Nurses | * Ensuring patient medical profile cannot be accessed or modified by unauthorized figures * Scheduling conflicts * Data breaches * Legal issues associated with data breaches | * Ensure only authorized accounts can modify data * Transaction logging * Prevent overlapping appointments * Defined appointment time slots |
| Project Team | * Ensuring timely delivery of product * Data breaches * Legal issues associated with data breaches | * Clear communication with clients of features provided and features out of scope * Constant team communication via frequent updates and reviews * Attempt to provide proper security measures |

## **1.2 Relevance**

The value of this hospital management system presents itself in improving the patient experience and lightening the workload of administrators (doctors and nurses). In providing a system in which patients and administrators have quick access to up-to-date data in a *single* destination, both parties communicate better and save time. The appointment-scheduling system to be provided will not only make it easier for patients to receive the appropriate medical attention but will also provide important patient data for use by administrators.

## **2. System Environment**

Our application will follow the three-tier architecture structure specified in the diagram on the following page. For client browser we will be using Google Chrome. The front-end and user interface will be built using JavaScript and ReactJS as the framework. Our project will use Node.js and Express.js for hosting the web server. Finally, we will be using SQL and the MySQL RDBMS for our database tier.



As we are only mimicking an actual three-tier architecture and using LocalHost, the entire three- tier architecture set-up must be replicated to run this application. Listed below are the hardware specifications of the laptops each member is using to run a server.

**Nick:**

|  |  |
| --- | --- |
| Component | Specification |
| Central Processing Unit | 2 GHz Intel Core i5 |
| Number of Processors | 1 |
| Memory | 8 GB |
| Operating System | macOS High Sierra (10.13.6) |

**Huynh:**

|  |  |
| --- | --- |
| Component | Specification |
| Central Processing Unit | Intel Core i5-8250U CPU @ 1.60 GHz |
| Number of Processors | 4 |
| Memory | 8 GB 1600MHz DDR3 |
| Operating System | Windows 10 Home 64-bit |

**Diana (MacBook):**

|  |  |
| --- | --- |
| Component | Specification |
| Central Processing Unit | 2 GHz Intel Core i5 |
| Number of Processors | 1 |
| Memory | 8 GB 1867MHz LPDDR3 |
| Operating System | macOS Mojave (10.14.5) |

## 

## **3. Functional Requirements**

**Patient Functional Requirements**

|  |  |  |
| --- | --- | --- |
| Requirement | ID | Detail Leveled Requirements |
| Create Account | 1P | Allow user to register and create an account |
|  | 2P | Prevent re-registering with an existing account |
|  | 3P | Prompt user in case of invalid registration details |
|  | 4P | Allow user to enter in personal information and fill out medical history |
|  | 5P | Assign new users a unique patient id |
| Login/ Logout | 6P | Allow users to log in after providing valid credentials |
|  | 7P | Prompt user of error in case of invalid credentials |
|  | 8P | Allow users to log out of their account |
|  | 9P | Allow users to recover or reset their password via email |
|  | 10P | A Recover or reset password attempt on an account not in the system fails |
| Personal Info and Medical History/Profile Access | 11P | Allow users to edit or update their personal information and medical history |
|  | 12P | Allow users to review their personal information and medical history |
|  | 13P | Allow users to have read access to their own medical profiles written by doctors |
| Schedule appointments | 14P | Allow user to schedule an appointment based on doctor. |
|  | 15p | Allow user to schedule appointment based on time. |
|  | 16P | Prevent scheduling an appointment conflicting with patient’s existing appointments. |
|  | 17P | Prevent scheduling an appointment conflicting with the doctor’s schedule. |
|  | 18P | Allow patient to schedule multiple appointments. |
| View/Update appointments | 19P | Allow user to view upcoming appointments. |
|  | 20P | Allow user to view past appointments. |
|  | 21P | Allow user to cancel an appointment. |
|  | 22P | Allow user to change appointment date. |
|  | 23P | Allow user to change appointment time. |
|  | 24P | Updating appointment date cannot conflict with scheduled doctor’s existing schedule. |
|  | 25P | Updating appointment date cannot conflict with patient’s existing schedule. |
|  | 26P | Updating appointment time cannot conflict with scheduled doctor’s existing schedule. |
|  | 27P | Updating appointment time cannot conflict with patient’s existing schedule. |

**Administrative (Doctors/Nurses) Functional Requirements**

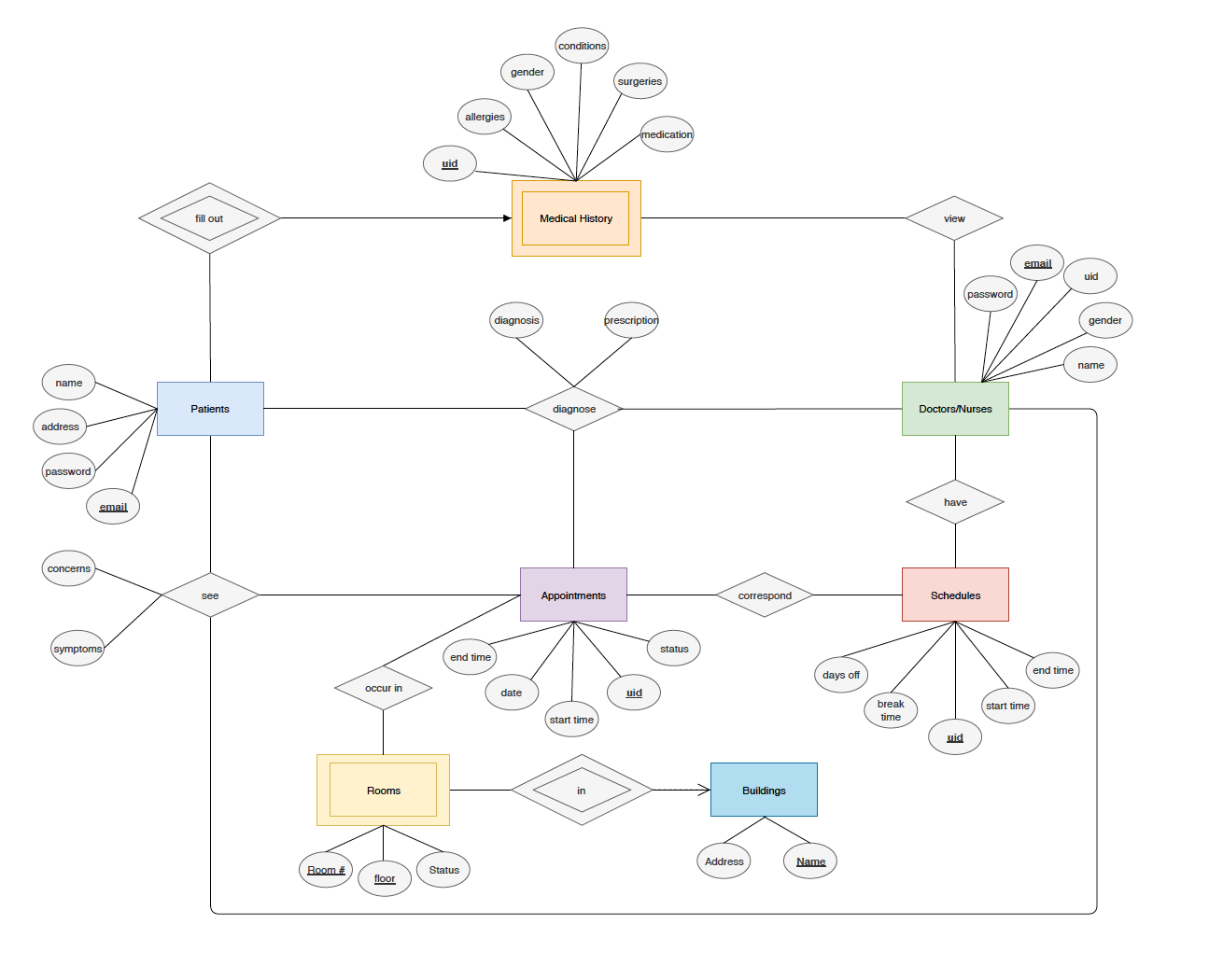
|  |  |  |
| --- | --- | --- |
| Requirement | ID | Detail Leveled Requirements |
| Create Account | 1A | Allow administrator to register and create an account upon providing their Employee ID |
|  | 2A | The Employee ID must be 9 digits long |
|  | 3A | Allow administrators to provide their first and last name, pick a department, and create a password |
|  | 4A | Upon creation, a generated work email will be provided from the information and added to the profile |
| Login/Logout | 5A | Enable administrators to log in upon providing their Employee ID and password |
|  | 6A | Allow administrators to log out of their account |
|  | 7A | Enable administrators to submit a support ticket to change their password |
| Access medical profiles | 8A | Administrators will be allowed to view all patient profiles |
|  | 9A | Administrators will be allowed to add to the medical history of a patient profile |
|  | 10A | Administrators will be allowed to view updates to a patient profile given by the patient |
|  | 11A | Administrators will be allowed to view all employee profiles |
|  | 12A | Administrators will be allowed to provide their own hours of restricted availability |
|  | 13A | Administrators will be allowed to add a prescription to a patient profile |
|  | 14A | Administrators will be allowed to view the appointments of other employee profiles |
| View/Update appointments | 15A | Allow user to view upcoming appointments. |
|  | 16A | Allow user to view past appointments. |
|  | 17A | Allow user to cancel an appointment. |
|  | 18A | Allow user to specify chunks of time that cannot be scheduled. |
|  | 19A | Update affected patient of appointment cancellation. |

## 

## **4. Non-Functional Requirements**

1. Execution qualities (Qualities which are observable during operation)
   1. Security
      1. There will be no broken authentication or broken access control points through which admin privileges are given to non-admin users
      2. Patients will not be able to access restricted data
   2. Privacy
      1. Patients cannot view other patients’ data
      2. Patients cannot view any of the doctor’s private data, such as their patient list
   3. Performance
      1. The database program shall execute in timely fashion, returning queries in a reasonable amount of time
   4. Constancy
      1. The program will not need to be executed separately or restarted in a single user’s session as well as a series of concurrent users of variable permission levels. The program will run in a perpetual state throughout its use.
2. Evolution qualities
   1. Documentation
      1. The system will feature an organization of code with descriptions such that each component can be easily understood as a constituent of the system
   2. Testability
      1. The system source code will follow the 3-tier architecture. Using that to the advantage of self-testing, the divided infrastructure will enable the project team to easily identify the point of vulnerability or error as one of the three classifications once error/exception handling is implemented

**5. Project Data Model**



**Entities:**

1. Patients: Patients in the HMS are users associated with an email, password, name, and address.
2. Doctors/Nurses: Doctors in the HMS are users associated with a unique doctor id, email, password, gender and name.
3. Schedules: Doctors have schedules that specify their days off, break times, and start and end time of their days.
4. Appointments: Appointments are associated with a unique appointment id, start time, end time, date of occurrence, and the status of appointment (scheduled, unscheduled, passed).
5. Rooms: Rooms are have a number and are on some floor of some building. Each room is either booked or unbooked.
6. Buildings: Buildings are located at some address and are associated with a unique name.
7. Medical History: The medical history is associated with a specific patient id and contains information regarding the patients’ medical history (allergies, gender, medication, conditions, surgeries).

**Relations:**

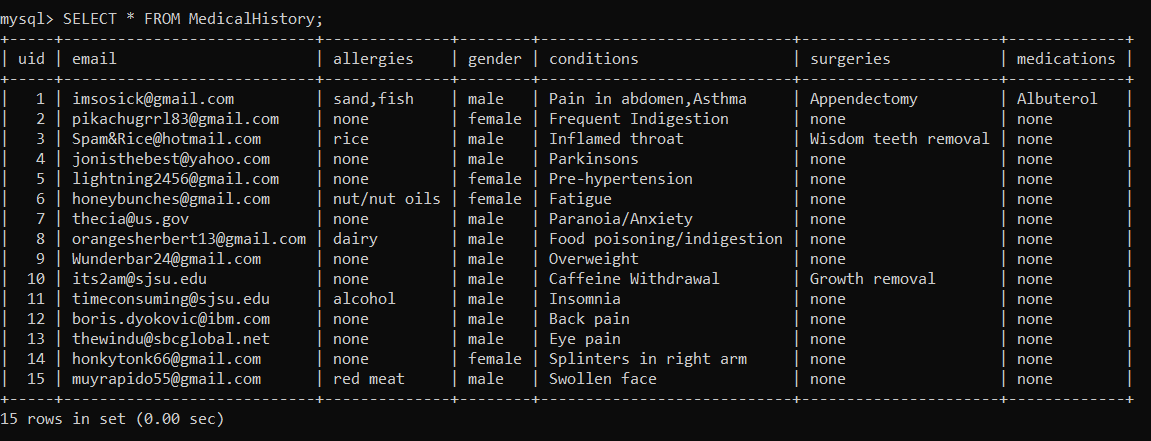
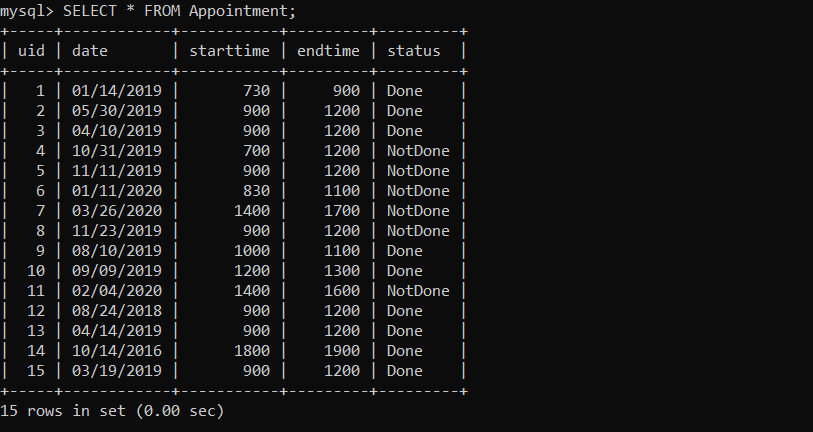
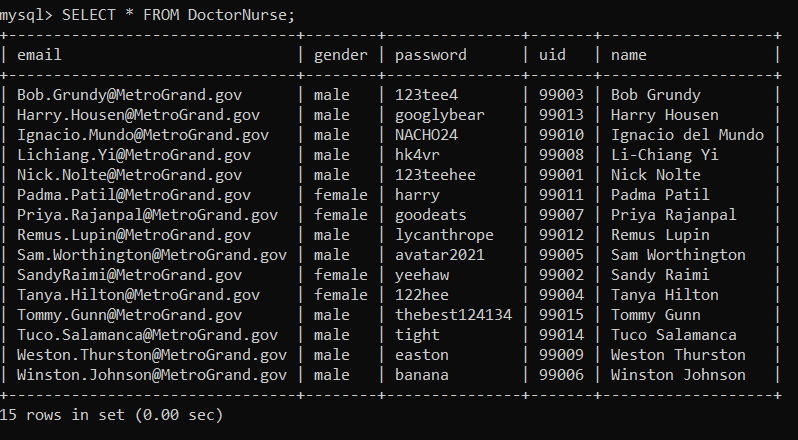
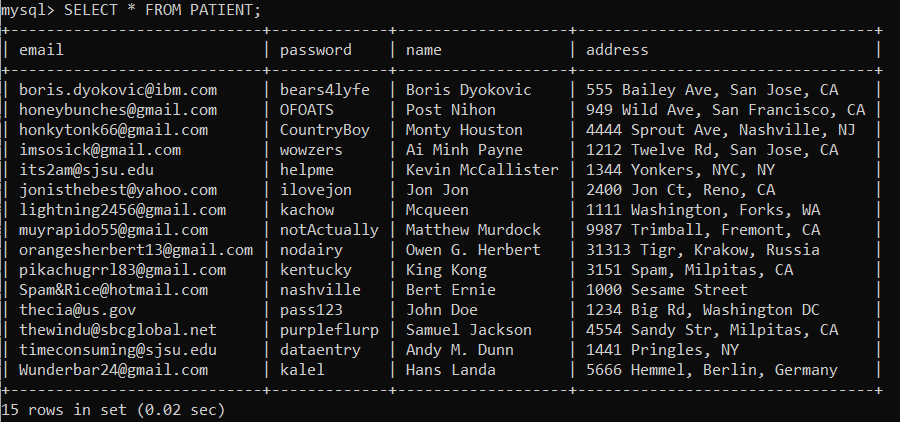
1. During an appointment, patients “see” a doctor. When patients see a doctor, they have concerns about their symptoms.
2. Each doctor “has” a schedule. These schedules define the doctors’ break time and days off and start and end time.
3. Doctors “diagnose” patients and the diagnosis may come with a prescription.
4. Patients “fill in” a single medical history.
5. Appoints are “located in” a room.
6. Rooms are located “in” a building.
7. Doctors “access” medical history.

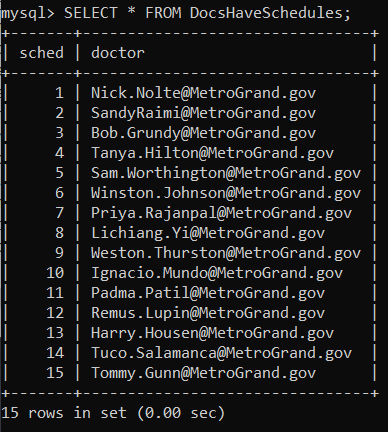
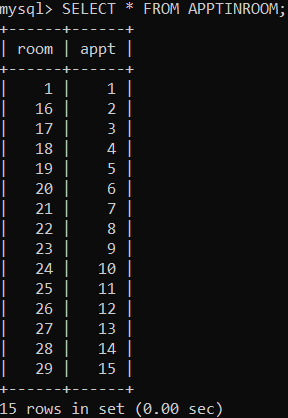
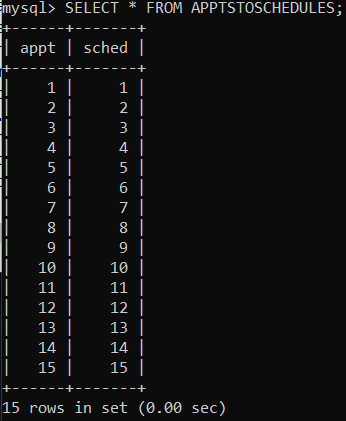
**Tables:**

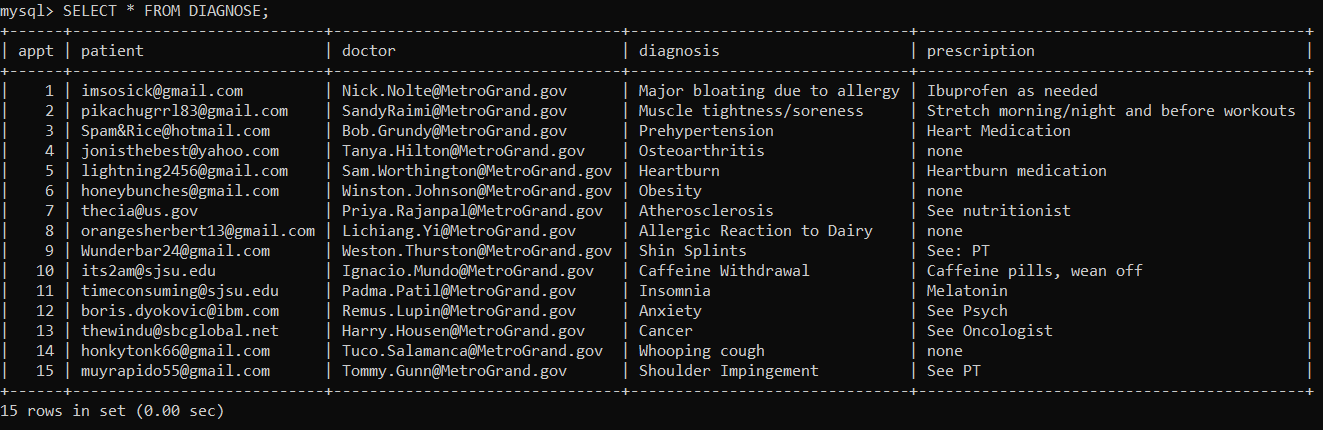
* Patients
* Doctors/Nurses
* Appointments
* Buildings
* Rooms
* Medical History
* Schedules
* During an appointment, patients “see” a doctor
* During an appointment, doctors “diagnose” patients
* Appointments “occur in” rooms

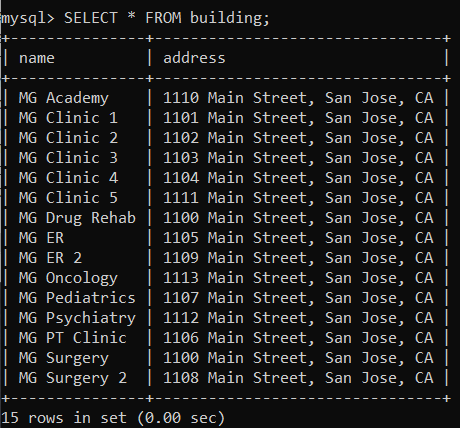
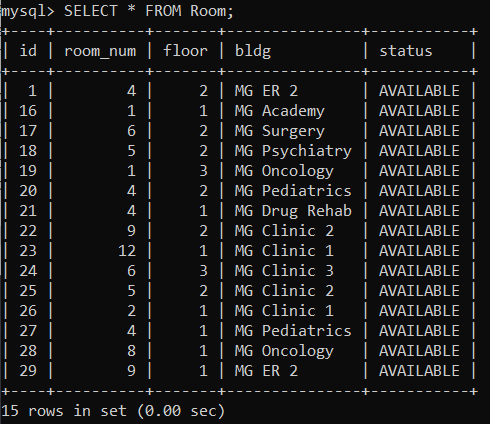
**Schemas:**

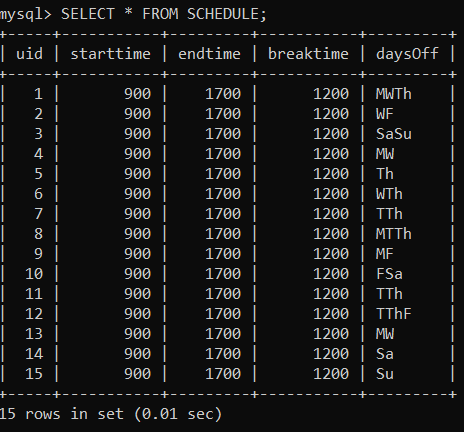
* **Patients**(name, address, password, email)
* **Doctors**(name, password, uid, gender, email)
* **Appointments**(endTime, startTime, date, status, uid)
* **Buildings**(address, name)
* **Rooms**(status, floor, roomNumber)
* **MedicalHistory**(allergies, gender, conditions, surgeries, medication, uid)
* **Schedules**(daysOff, breakTime, startTime, endTime, uid)
* **PatientsSeeDoc**(concerns, symptoms, patientEmail, appointmentUid, doctorEmail)
* **Diagnoses**(diagnosis, prescriptions, patientEmail, appointmentUid, doctorEmai)
* **AppointmentToRooms**(appointmentUid, roomNumber, floor, buildingName)

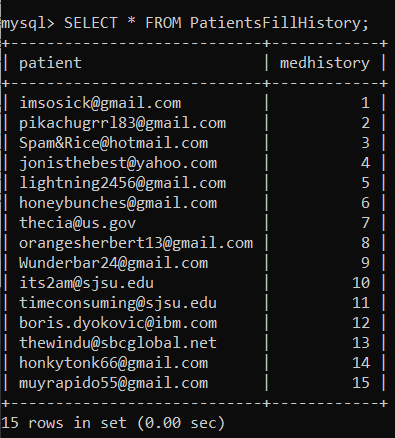












6. Database Normalization (BCNF):

* **Patients** (email, name, address, password)
* **Doctors** (email, name, password, gender,)
* **Appointments** (uid, startTime, endTime, date, status)
* **Buildings** (name, address)
* **Rooms** (building\_name, roomNumber, status, floor)
* **MedicalHistory** (uid, allergies, gender, conditions, surgeries, medication)
* **Schedules** (uid, daysOff, breakTime, startTime, endTime)
* **PatientsSeeDoc** (appointment\_uid, doctor\_email, patient\_email, concerns, symptoms)
* **Diagnoses** (patient\_email, appointment\_uid, doctor\_email, diagnosis, prescriptions)
* **AppointmentToRooms** (building\_name, appointment\_uid, roomNumber, floor)

**7. Implementation**

Overview: Our database management system application is built on MySQL database, Node.Js server and ReactJs user interface. All of the data for our application is stored in a MySQL database and accessed by the backend server which is built on Express, a Node.Js framework. The server queries the database and stores the results as JSON. The client side, which is built on ReactJs, a JavaScript framework, allows the users to use the application. The front-end application pulls the necessary information from backend server.

8. Conclusion