CSCE 608 Database System Project 1

Hospital Management System

Name: Siyang Yang

UID:431000499

10/17/2020

The project Hospital Management System (HMS) is aiming at helping set up, configure, and manage appointment scheduling between patients and doctors. The project can provide an intuitive and clear interface for patients to make appointments. On the other side, the doctors can query their appointments in a sufficient time from the database. The hospital management system provides plenty of automation of many vital daily processes. Medical specialists, patients, and hospital authorities can interact inside HMS and exchange information. It simplified the workload of healthcare staff, and securely manage the whole system data. Since the collections of data from patients, doctors, and hospitals are imaginable large and important, I implement this project aiming at providing a reliable platform for the healthcare industry.

The project includes registration of patients, storing personal information such as name, age, and email into a database, and create an appointment in selected specialization, doctors, and labs. Each patient will be assigned a unique ID number along with their personal information. The patient must remember their email address and password to login to the system to update their info like phone number and address etc. The HMS allows the patient to review appointment histories, which includes doctor name and date visited. This kind of automation is one of the main benefits of utilizing digital medical records. Patients would gain a better customer experience.

The hospital specialists also can register, view, and update their information in the database. Same as patients, specialists can update their specialization, consultancy fee, and contact information by logging in with email and password. In the meantime, each specialist can review their appointments history. Importantly, they can see the waitlist patient’s information in real-time. The HMS provides a great platform for them to access and research data from the database. Each specialist can save plenty of valuable time with up-to-date information.

The hospital authorities can manage all data storing in the database. They receive the automated control of all the interactions in their hospital. Whenever new record is created from patient or doctor, the admin can check simultaneously. Since the patient's and doctor’s information are extremely important for the hospital, HMS prevents data loss. For example, the admin account is the only role that can delete the tuples from the database. The admin account also maintains the room availability. Once the labs are occupied, they won't be selected by the patients while making an appointment. Of course, the system has the querying functionality. The admin can find a specific patient or doctor by searching for their cellphone number. In a real hospital facility, there are millions of data stored in the database, so it is essential to querying information in a short time.

We can make a simple conclusion that a hospital management system is an inevitable software in the healthcare area. It is powerful and convenient that patients and hospital specialists are benefited from it. So, this project helps store all kinds of records, user information, and improve day-to-day operations. It smooths the interaction between patients and specialists, also helps the hospital to do record management. Even though the system I have implemented only has basic appointment functionality, it already showed great potential. I hope I can improve my hospital management system to deliver real conceivable benefits to the hospital.

Part A. E/R Diagram

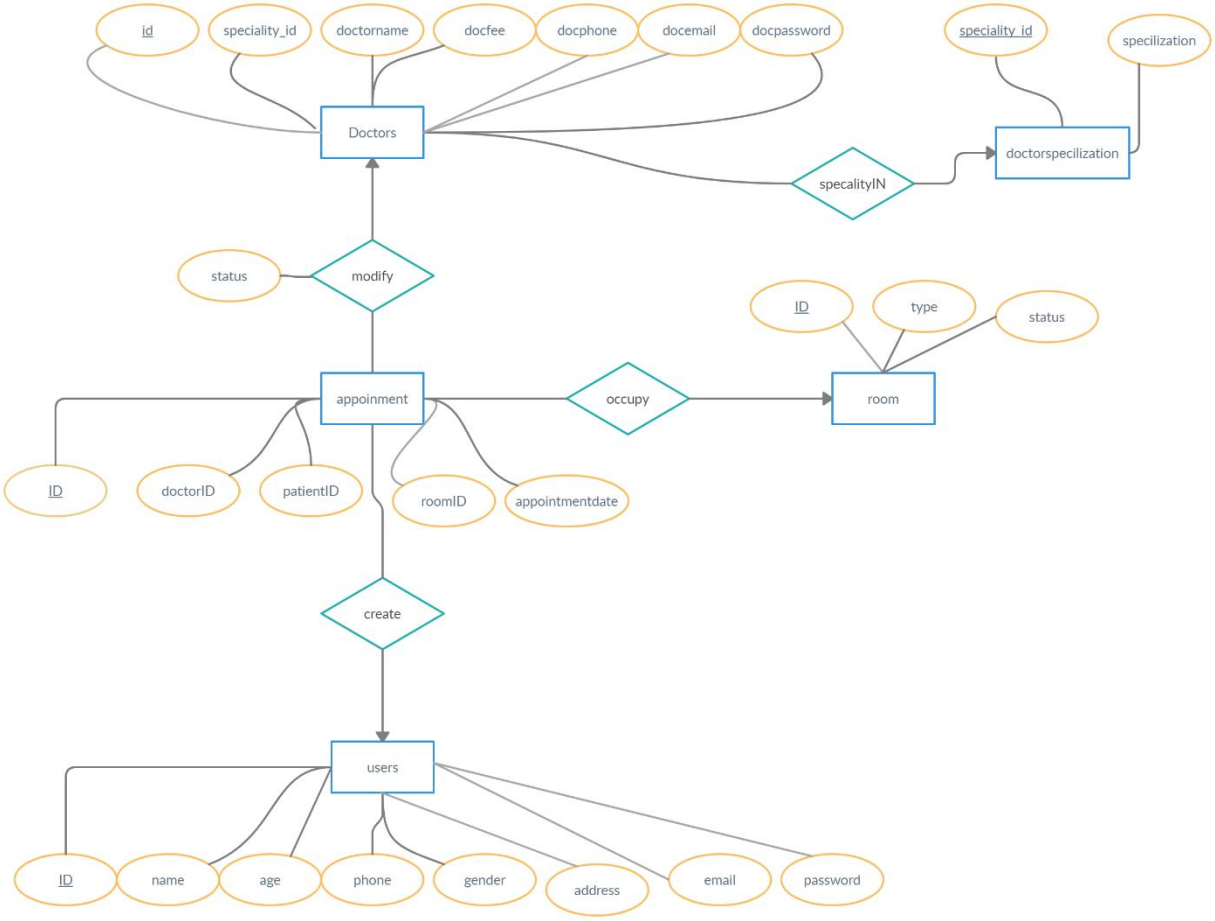


Fig 1. E/R diagram of the HMS

First, the doctors entity has ID as primary key auto-incremented, speciality_id as a foreign key along with the name, consultancy fee, phone number(unique), email address(unique), and password attributes. Once the doctor login with email and password, the ID will be stored in the session, which will be used to query all other attributes that functionally dependent on the primary key.

Every doctor should have one specialization which is showing a many-to-one relationship connected to doctorspecilization entities. If we modify a specialization name, we also have to do multiple operations on doctors entity. Thus, I set doctorspecilization as a separate entity set.

Doctorspecilization entity has ID as primary key and specialization attributes, as explained above.

Each doctor can modify multiple appointments status, which only relates to their doctorID. So, modify a relationship is a many-to-one relationship.

For the appointment entity, it also has ID as primary key, doctorID, patientID, roomID as foreign keys, and appointmentDate as attributes. Here those three foreign key IDs referencing to primary keys of other entities.

Since each appointment is only created by one patient, so an appointment has the "many" side, patient has one side. Also, each appointment occupies only one room, appointment has the "many" side and room has the "one" side.

For the room entity, ID as the primary key along with room type and Boolean value status as attributes.

For users entity, ID as primary key and patient name. age, phone, gender, address, email, and password as attributes. It is very similar to the doctors entity set. The patientID will be stored in the session and used to query all attributes that functionally dependent on it.

Noticed that there is no admin entity but have an admin table because of consistency. The admin account can be authenticated by the front end controller instead of backend since there is only one account for admin. Thus, even though no admin entity, the E/R diagram will not be affected.

Part B. Relational Schema

Transfer the E/R diagram to schema as below:

1. Doctors(ID, specialityID, doctorname, docfee, docphone, docemail, docpassword)
2. specialityIN(doctorID, specialityID)
3. doctorspecilization(specialityID, specilization)
4. Modify(appointmentID, doctorID, status)
5. Appointment(ID, doctorID, patientID, roomID, appointmentdate)
6. Occupy(appointmentID, roomID)
7. Room(ID, type, status)
8. Create(appointmentID, patientID)
9. users(ID, name, age, gender, address, phone, email, password)

The nontrivial functional dependencies as blow:

1. Doctors: ID→{ specialityID, doctorname, docfee, docphone, docemail, docpassword }
2. Doctors: { docemail, docpassword }→{ specialityID, doctorname, docfee, docphone }
3. Doctors: { docemail, docpassword }→{ID}
4. specialityIN :{DoctorID}→{ specialityID }
5. doctorspecilization :{ specialityID }→{ specilization }
6. Modify:{ appointmentID }→{ doctorID, status }
7. Appointment:{ ID } →{ doctorID, patientID, roomID, appointmentdate }
8. Occupy:{ appointmentID }→{ roomID }

9. Room:{_ID }→{ type, status }
10. Create:{ appointmentID }→{ patientID }
11. users :{_ID }→{ name, age, gender, address, phone, email, password }
12. users: { email, password }→{ name, age, gender, address, phone }
13. users: { email, password }→{ID}

Apply the BCNF to reduce redundency:

ID^+ = (ID, specialityID, doctorname, docfee, docphone, docemail, docpassword)

$(docemail, docpassword)^+$= (ID, specialityID, doctorname, docfee, docphone, docemail, docpassword)

For doctors, specialityIN: both ID and (docemail, docpassword) closure are in BCNF because the lefts sides are all superkeys.

$(DoctorID)^+$=(DoctorID , specialityID)

For specilityIN: left side is superkey, also because only two attributes, default in BCNF.

$(specialityID)^+$= (specialityID, specilization)

For doctorspecilization: speculityFD closure includes all attributes that are functionally determined by specialityID, thus, in BCNF

$(appointmentID)^+$=(doctorID, status, appointmentID)

For *Modify: left side is superkey, so in BCNF*

$(appointmentID)^+$=(appointmentID ,doctorID, status, patientID, roomID, appointmentdate)

For Appointment, Occupy, Create: the closure of appointmentID includes all attributes that are functionally determined by appointmentID, the left side is superkey. Thus, it is in BCNF

$(appointmentID)^+$=(appointmentID, roomID)

For Occupy: left side is superkey and only two attributes, default in BCNF.

$(appointmentID)^+$=(appointmentID, patientID)

For Create: left side is superkey, so in BCNF

$(roomID)^+$=(roomID, type, status)

For room: the left side is superkey, it is in BCNF

$(usersID)^+$=(usersID ,name, age, gender, address, phone, email, password)

$(useremail, userpassword)^+$=(usersID ,name, age, gender, address, phone, email, password)

For users: both ID and (useremail, userpassword) closure are in BCNF because closure includes all attributes that are functionally determined by usersID or (useremail, userpassword). Thus, it is in BCNF

Thus, we can conclude these relations are all in BCNF, however, we can simplify some of tables because of Many-To-One relationship. Some of the attributes are serving as foreign keys so that we can “merge” some schemas into the many side schema. In this way, we can show the new tables.

1. Doctors(ID, specialityID, doctorname, docfee, docphone, docemail, docpassword)
2. doctorspecilization(specialityID, specilization)
3. Appointment(ID, doctorID, patientID, roomID, appointmentdate)
4. Room(ID, type, status)
5. users(ID, name, age, gender, address, phone, email, password)

doctors		
PK	ID	bigint
FK	speciality_ID	int
	doctorname	varchar(255)
	docfees	int
	docphone	bigint(11),UNIQUE
	docemail	varchar(255),UNIQUE
	password	varchar(255)

Fig 2. Doctors table

doctorspecilization		
PK	specialityID	int
	specilization	varchar(255)

Figure 3. doctorspecilization table

room		
PK	ID	int
	type	varchar(255)
	status	tinyint

Figure 4. room table

appointment		
PK	ID	bigint
FK	doctorID	bigint
FK	patientID	bigint
FK	roomID	int
	appointmentDate	Datetime
	status	tinyint

Figure 5.appointment table

users		
PK	ID	bigint
	name	varchar(255)
	age	tinyint
	gender	varchar(10)
	address	longtext
	phone	bigint(11),UNIQUE
	email	varchar(255),UNIQUE
	password	varchar(255)

Figure 6. users table

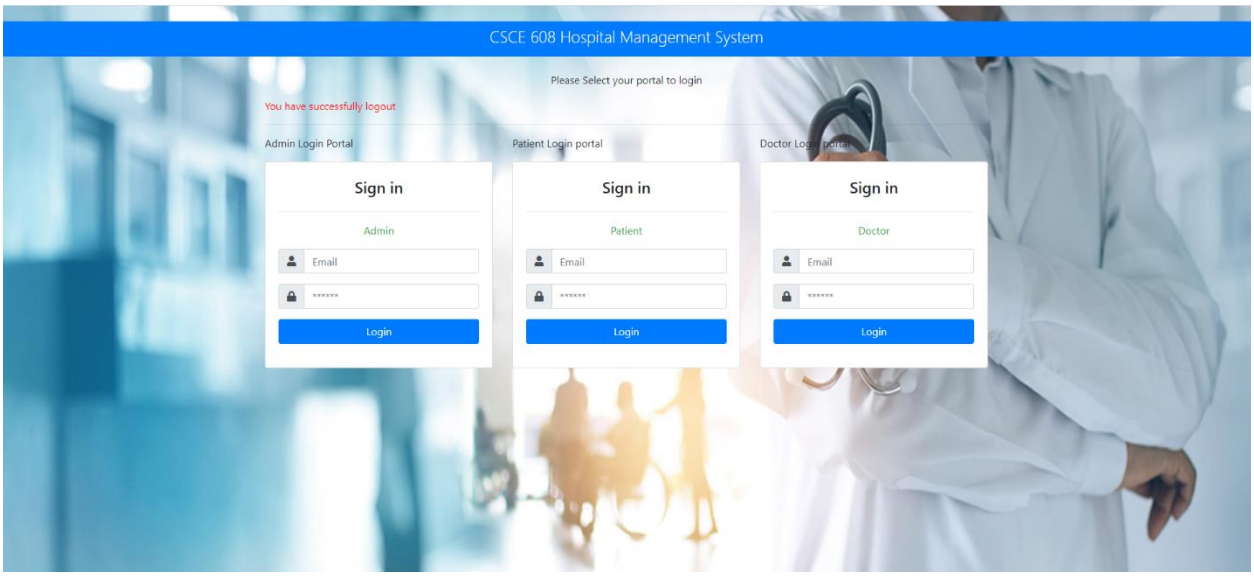
Data collection:

https://www.mockaroo.com/ is responsible for generating most SQL data for this project. It can generate a full name, random numbers in ranges, cell phone number, address, email, and password. At maximum, it can output 1000 tuples at a time, which will fulfill the requirements of the project. Both users and appointments have more than 1000 tuples and more than 300 doctor tuples. Besides, 100 rooms are generated, however, the room type is filled by clinic number. Since there are only 12 kinds of specialization, I inserted manually on the admin console. Because this website can output SQL formate after I set up the relations, I could insert all related data into the tables directly using PHPmyadmin console.

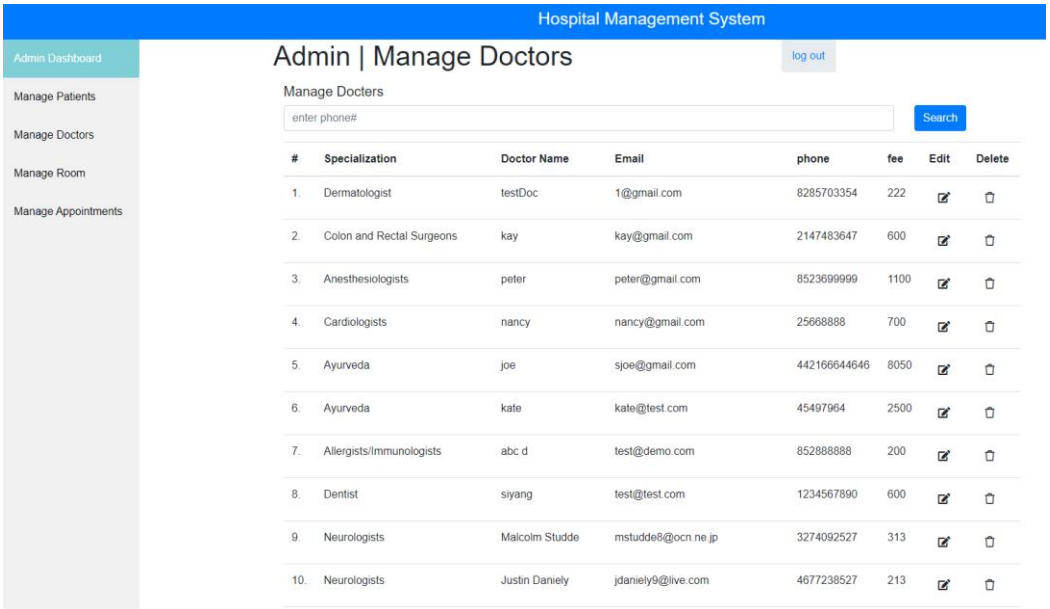
Since I have created the table and set restriction like key uniqueness, if there exist duplicate data, I would be noticed by the PHPMyadmin console. Thus, in this way, I can ensure there is no duplicate data for an important table.

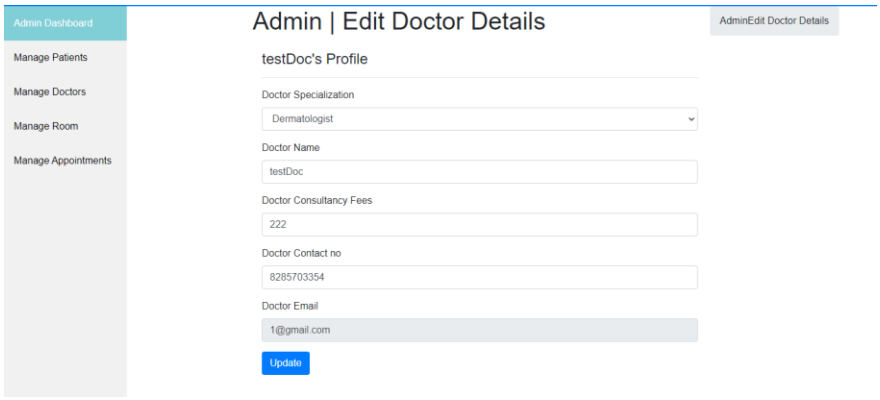
User interface:

I implemented this project using php, boostrap CSS as front. Ajax,Php as back end, and mysqli as database management system.



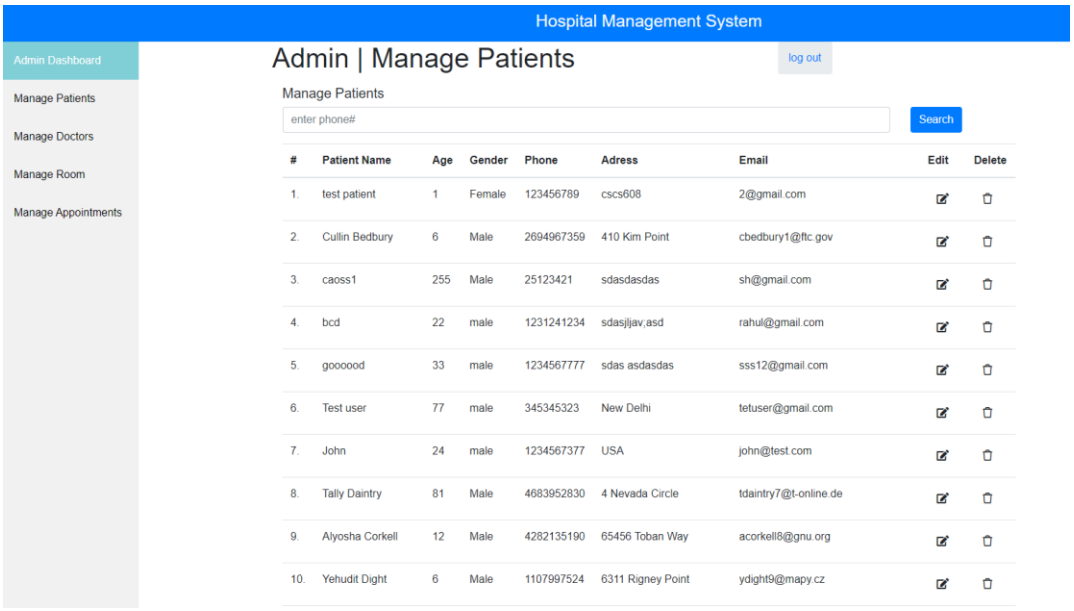
On the main page, admin, patient and doctor can login their portal using email and password that stored in the database. The most important functionalities will be showed as below and the recorded video will cover more functionalities.

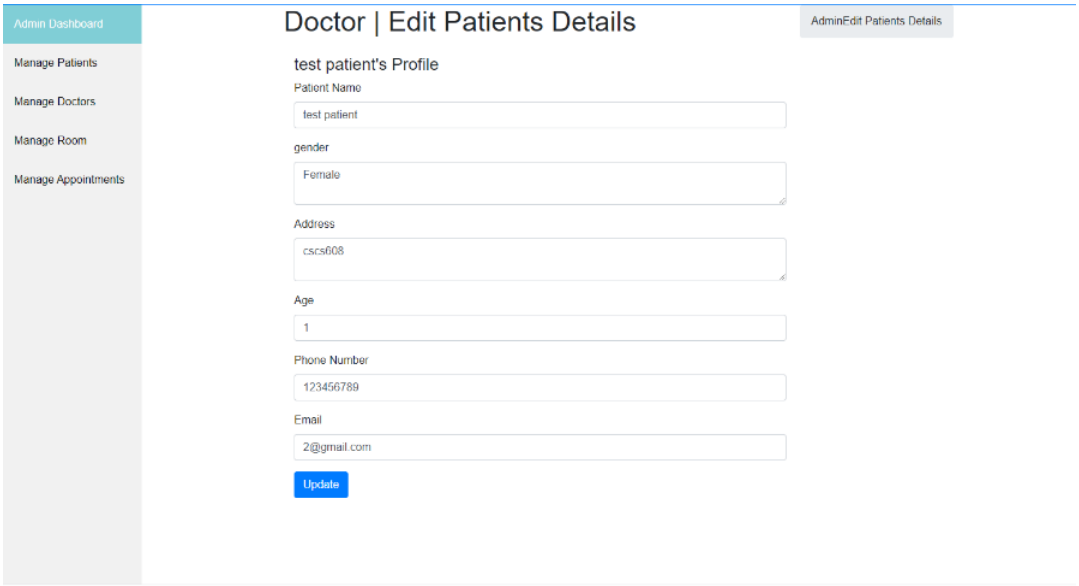




Admin can view, edit and delete all data about every doctor's information and specilization, and can search specific doctor info by entering the phone number. Those operation can be achived by using SQL *SELECT*, *UPDATE*, *AND DELETE* semantics. In addition, I develop the user-friendly layout that each page is showing limited number of information.

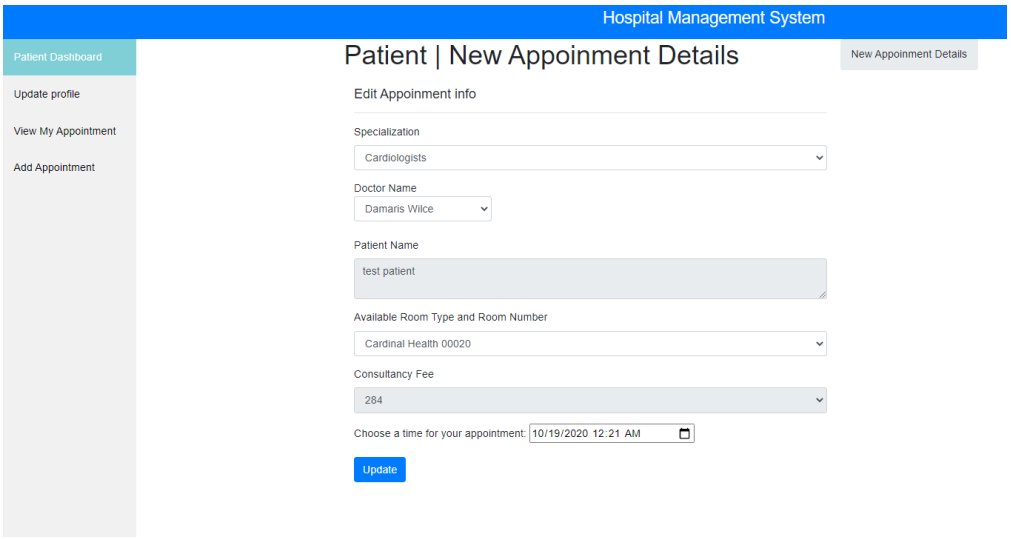
```
if(isset($_GET['del']))
{
        mysqli_query($con,"delete from doctors where id = '".$_GET['id']."'");
        $_SESSION['msg']="Doctor data deleted !!";
}

if (isset($_POST['search'])){
        $search = $_POST['search'];
        $search_id = mysqli_query($con, "select * from doctors where docphone = '$search' ");
        $doc_info=mysqli_fetch_array($search_id);

        if($doc_info){
                header("Location: edit-doctor.php?id= ".intval($doc_info['id'])." ");
        };
}

$sql = mysqli_query($con,"select * from doctors ORDER BY id ASC LIMIT $start_from,
$result_per_page ")


if(isset($_POST['submit']))
{
        $docspecialization=$_POST['Doctorspecialization'];
        $docname=$_POST['docname'];
        $docfees=$_POST['docfees'];
        $docphone=$_POST['doccontact'];
        $docemail=$_POST['docemail'];
        $sql=mysqli_query($con,"Update doctors set
specialityID='$docspecialization',doctorName='$docname',docFees='$docfees',docphone='$doc
phone',docEmail='$docemail' where id='$did'");
        if($sql)
        {
                $msg="Doctor Details updated Successfully";
        }
        }
```





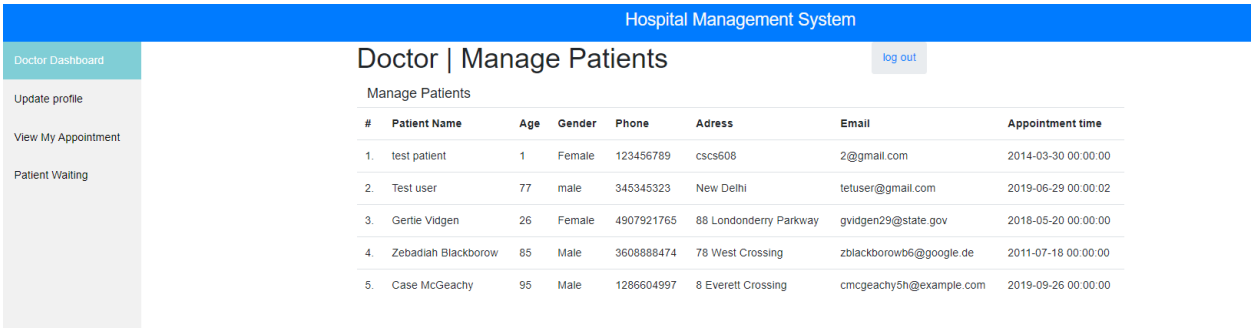
Similarly for the managing patient page, admin can view, edit and delete data about patietns. Also able to find specific patient by their phone number. In SQL, *WHERE* can check whether the phone number is equal to the search number. Whenver there is successfully update or delete operation, the system will notify the user the result.

```php
if(isset($_GET['del']))
{
	mysqli_query($con,"delete from users where id = '".$_GET['id']."'");
	$_SESSION['msg']="patient data deleted !!";
}

if (isset($_POST['search'])){
	$search = $_POST['search'];
	$search_id = mysqli_query($con, "select * from users where phone = '$search' ");
	$patient_info=mysqli_fetch_array($search_id);

	if($patient_info){
		header("Location: edit-patient.php?id= ".intval($patient_info['id'])." ");

	};

$sql = mysqli_query($con,"select * from users ORDER BY id ASC LIMIT $start_from, $result_per_page ");

if(isset($_POST['submit']))
{
	$patientname=$_POST['patientname'];
	$patientage=$_POST['patientage'];
	$patientphone=$_POST['patientphone'];
	$patientaddr=$_POST['patientaddr'];
	$patientgender=$_POST['patientgender'];
	$patientemail=$_POST['patientemail'];

	$sql=mysqli_query($con,"Update users set fullName='$patientname',age='$patientage',phone='$patientphone',email='$patientemail',address='$patientaddr' ,gender='$patientgender' where id='$pid'");
	if($sql)
	{
		$msg="Patient Details updated Successfully";
	}else{
		echo("Error");
	}
```



The patient can add an new appointment record into the database by using SQL *INSERT*. The thing is complicated is how to pre-select doctors from different speciailization category. I was using ajax to help find the doctors. For the specilization, using the speciality_ID that matches the ID in doctorspecilization table, which in SQL *WHERE* can check the condition using subquery.

```
if(isset($_POST['submit']))
{

$doctorid=$_POST['doctor'];
$userid=$_SESSION['id'];
$roomId=$_POST['room'];
$apptime=$_POST['appointment-time'];

        $sql=mysqli_query($con, "insert into appointment(doctorId, userId,
roomId,appointmentDate,status)
                values('$doctorid' ,'$userid', '$roomId','$apptime' ,'1')");
        if($sql)
        {
                $msg="Appointment Details Added Successfully";
        }
}


if(!empty($_POST["specilizationid"]))
{

 $sql=mysqli_query($con,"select doctorName,id from doctors where
specialityID='".$_POST['specilizationid']."'");?>
 <option selected="selected">Select Doctor </option>
 <?php
 while($row=mysqli_fetch_array($sql))
        {?>
  <option value="<?php echo htmlentities($row['id']); ?>"><?php echo
htmlentities($row['doctorName']); ?></option>
  <?php
}
}

if(!empty($_POST["doctor"]))
{

 $sql=mysqli_query($con,"select docFees from doctors where
id='".$_POST['doctor']."'");
 while($row=mysqli_fetch_array($sql))
        {?>
 <option value="<?php echo htmlentities($row['docFees']); ?>"><?php echo
htmlentities($row['docFees']); ?></option>
  <?php
}
}
```



Doctor can check the current patietns on the waitlist, whose appointment status is active. The doctor can make changes on the appointment status and date so that they can remove or add patients on the waitlist. This kind of operation will both interact with appointment table and patient table. By applying the *JOIN* in SQL, I can find where the patientID and doctorID are matched in the appointment table.

```
$sql = mysqli_query($con,"select users.fullName,
        users.age,
        users.gender,
        users.phone,
        users.address,
        users.email,
        appointment.*  from appointment

        join users on users.id = appointment.userId
        join doctors on doctors.id=appointment.doctorId

        where appointment.doctorId = '".$_SESSION['id']."' AND
                                appointment.status = '1' ");
```

Discussion:

This is positively a great project to learn the database system. To utilize, modify, and create data using a database management system let me understand database systems more comprehensively. As a developer, I have built my database development skills from scratch. Starting from an idea to creating the database schema, then building both the front and backend, querying data successfully. The whole developing experience is tremendously helpful. The biggest obstacle I have faced was to select the topic and come up with an E/R diagram. I have given up many interesting topics because I could not create a perfect diagram to fit the requirement. However, I was able to find this hospital management system project to work on. Secondly, It was hard for me to establish the whole scope such as how to connect to the database in my framework. Through hours and hours of learning from other great projects on the website, I can start to build my HMS from scratch.

In the end, I wish to improve my project by implementing it on a popular framework such as VUE.js or Angular. In that way, I could learn much more about how to develop full-stack software. There are many more functionalities could be added into the HMS, in that way, both patient and specialists can interact more with each other.