

Fall 2020: CSCE 608 Database Systems

**Project Report on
Hospital Management System**

Submitted By:

Flavia Ratto

130004853

Project Description:

I have taken up the topic of Hospital Management system database project. This project would have patient records consisting of the patient's basic details. Each patient would be assigned to one doctor. However, each doctor could treat multiple patients. The system would also include some basic details pertaining to the individual doctor. Also, each patient would be assigned to a wardroom/ operation theater/ ICU. Additionally, the patient care staff including the nurses and ward boys would be assigned some rooms.

We are going to consider two user types – Receptionist and Doctor in this system. The system will be accessible only using valid usernames and passwords for each user type.

1. Receptionist:

The receptionist here has full access to the system.

- They can register a new doctor or another receptionist to the system
- They can add a new record for patient, doctor and other staff in the system
- They can retrieve existing record for patient, doctor and other staff
- They can edit patient and staff details and update it in the database
- They can create an appointment for a patient and assign him a doctor
- They can create/update and delete a record for a room/operation theatre/ ICU
- They can obtain the room/operation theatre/ ICU that are unassigned
- They can allocate/deallocate a room/operation theatre/ ICU to a patient and update its status in the database
- They can assign or remove assignment of the staff (nurse and ward boys) to the rooms/operation theater/ICU

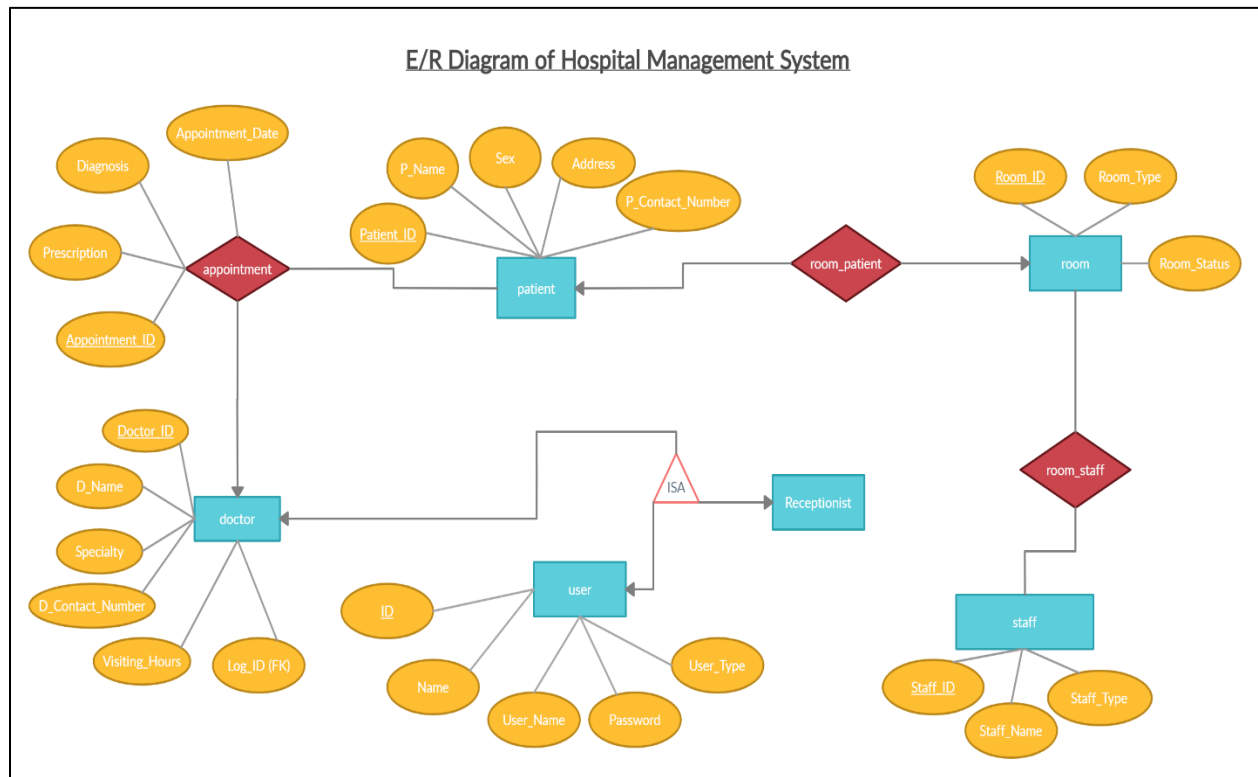
2. Doctor:

The doctor will login to the system with his username and password.

- The doctor will be able to add and update his own details in the system. They will not be able to add or update details of other doctors in the system
- They will be able to view details about other doctors in the hospital database
- They will be able to retrieve the patient records as well as the patients assigned to them
- They will also be able to view the room/operation theatre/ ICU the patient is assigned to
- They can also add/update a diagnosis and prescription for the patient

The Entity-Relationship:

Please find below the E/R diagram I have created for the Hospital Management system database application –



- **Entity Sets:**

The E/R diagram has 6 entity sets with their own set of attributes. They are –

1. **User:** This entity set would contain records of all the users (receptionist and doctors) who would be accessing the system. Each user has an ID uniquely identifying them.
2. **Doctor:** This entity set would contain details of all the doctors in the hospital who are registered as users in the system. Each doctor registered as a user in the system would create his entry in the system and put in his details. He would have a unique Doctor_ID associated to identify them. Log_ID would act as a foreign key here which references to the ID attribute in user table.
3. **Patient:** This entity set would contain details of all the patients visiting the hospital. Each patient registered in the database system would have a unique Patient_ID associated to identify them.
4. **Rooms:** This entity set would contain details of all the rooms/operation theatres/ICUs in the hospital. Each of them would have a Room_ID associated to uniquely identify them.
5. **Staff:** This entity set would contain details of all the patient care staff like nurses and ward boys working at the hospital. Each staff member registered in the database system would have a Staff_ID associated to uniquely identify them.
6. **Receptionist:** The entity set receptionist is an entity set without any attributes. It exists to signify that a user could be a receptionist.

- **Relationships:**

The E/R diagram also consists of 4 relationships. They are –

1. appointment: When a patient comes to a hospital, he would want to take an appointment with a doctor. Therefore, a patient would be assigned to a doctor. However, one doctor could treat multiple patients. Therefore, the appointment relationship between the patient and doctor is of a many-one type. This relationship would have attributes of Appointment_ID, Appointment_Date, Diagnosis and Prescription. The Appointment_ID would be the key attribute for this relationship.
2. room_patient: Each patient would be assigned to one room/operation theatre/ICU. Each room would have one patient in it. This relationship is of a one-one type. This relationship has no attributes of its own.
3. room_staff: Each patient care staff could be assigned multiple rooms to work in. Also, every room could have multiple care staff (1 nurse + 1 ward boy or 2 nurses in the ICU) assigned to it. Therefore, this is a many-many relationship.
4. ISA: This relationship defines whether a user is doctor or a receptionist. It connects the user entity with doctor and receptionists

Table Normalization:

Constructing relations based on the E/R diagram above –

- **Entity sets:**

1. user → user (ID (PK), Name, User_Name, Password, User_Type)
2. patient → patient (Patient_ID (PK), P_Name, Sex, Address, P_Contact_Number)
3. doctor → doctor (Doctor_ID (PK), D_Name, Specialty, D_Contact_Number, Visiting_Hours, Log_ID (FK))
4. rooms → rooms (Room_ID (PK), Room_Type, Room_Status)
5. staff → staff (Staff_ID (PK), Staff_Name, Staff_Type)

- **Relationships:**

1. appointment → appointment (Appointment_ID (PK), Appointment_Date, Diagnosis, Prescription, Patient_ID (FK), Doctor_ID (FK))
2. room_patient → Allocated (Patient_ID (FK), Room_ID (FK))
3. room_staff → Assigned (Room_ID (FK), Staff_ID (FK))

Combining Relations –

When there is an entity set E with a many-one relationship R from E to F, we can combine E and R into one relation with a schema consisting of:

- All attributes of E.
- The key attributes of F
- Any attributes belonging to relationship R

Therefore, we combine the below relations:

1. Relations patient and appointment is combined → pat_appointment (Patient_ID, P_Name, Sex, Address, P_Contact_Number, Doctor_ID, Appointment_ID, Appointment_Date, Diagnosis, Prescription)

We now have a total of 7 tables.

To check if all the tables are in BCNF:

1. user (ID (PK), Name, User_Name, Password, User_Type)
Functional Dependencies-
 - ID \rightarrow Name, User_Name, Password, User_TypeSuperkey – ID
Therefore, **user**(ID (PK), Name, User_Name, Password, User_Type) is already in BCNF.
2. pat_appointment (Patient_ID, P_Name, Sex, Address, P_Contact_Number, Doctor_ID, Appointment_ID, Appointment_Date, Diagnosis, Prescription)
Functional Dependencies-
 - Patient_ID \rightarrow P_Name, Sex, Address, P_Contact_Number
 - Appointment_ID \rightarrow Patient_ID, P_Name, Sex, Address, P_Contact_Number Doctor_ID, Diagnosis, Prescription, Appointment_DateWe are assuming a patient and doctor pair could have multiple appointment dates.
Superkey – Appointment_ID
Therefore, Patient_ID \rightarrow P_Name, Sex, Address, P_Contact_Number violates BCNF. Applying the BCNF algorithm on pat_appointment, we decompose the relation as **patient**(Patient_ID, P_Name, Sex, Address, P_Contact_Number) and **appointment**(Appointment_ID, Patient_ID, Doctor_ID, Appointment_Date, Diagnosis, Prescription)
3. doctor (Doctor_ID, D_Name, Specialty, D_Contact_Number, Visiting_Hours, Log_ID)
Functional Dependencies-
 - Doctor_ID \rightarrow D_Name, Specialty, D_Contact_Number, Visiting_HoursSuperkey – Doctor_ID
Therefore, **doctor** (Doctor_ID, D_Name, Specialty, D_Contact_Number, Visiting_Hours) is already in BCNF
4. room (Room_ID, Room_Type, Room_Status)
Functional Dependencies-
 - Room_ID \rightarrow Room_Type, Room_StatusSuperkey – Room_ID
Therefore, **room** (Room_ID, Room_Type, Room_Status) is already in BCNF
5. staff (Staff_ID, Staff_Name, Staff_Type)
Functional Dependencies-
 - Staff_ID \rightarrow Staff_Name, Staff_TypeSuperkey – Staff_ID
Therefore, **staff** (Staff_ID, Staff_Name, Staff_Type) is already in BCNF
6. room_patient (Patient_ID, Room_ID)
Since there are only two attributes in this table, it is already in BCNF
Therefore, **room_patient** (Patient_ID, Room_ID) is already in BCNF
7. room_staff (Room_ID, Staff_ID)
Functional Dependencies-

No nontrivial Functional Dependency holds here because it is a many-many relationship. Many employees are assigned to the same room. Also, each employee could be assigned many rooms to work in.

However, the relation **room_staff**(Room_ID, Employee_ID) is in BCNF since there are only two attributes here.

The relations have therefore been normalized in order to eliminate bad FDs which would cause presence of redundant data in the database. This would also lead to anomalies during updating and deleting data.

However, after normalization I observe that we are back to 8 relations. The 8 relations are same as the 8 we had before we performed the step of combining relations.

Also, I learnt that relation with no non-trivial FDs would be in BCNF by default.

Data Collection and Database Creation:

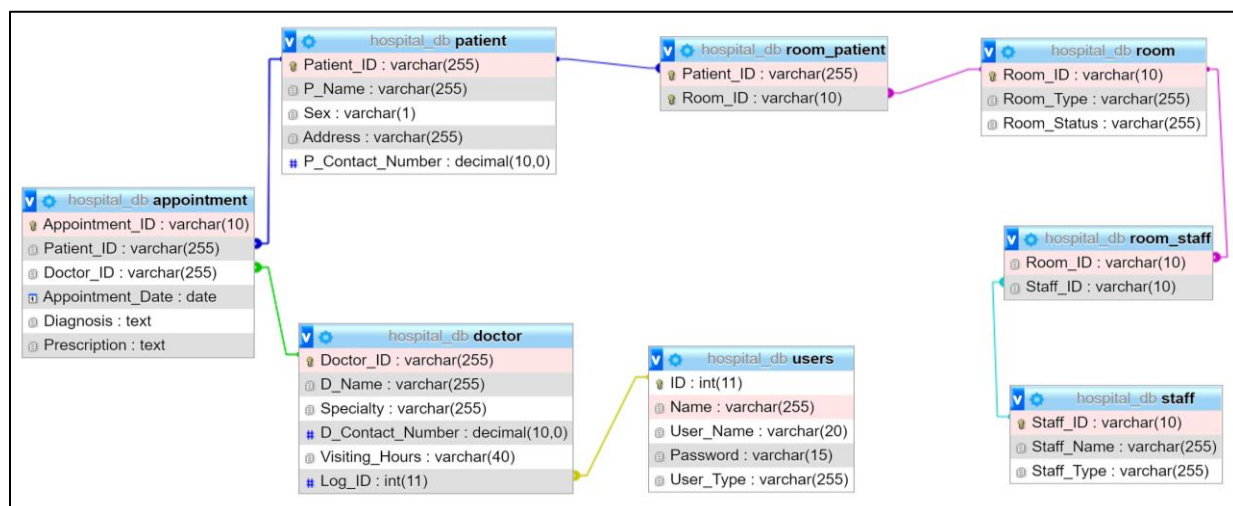
I have created by database using **MYSQL** on phpMyAdmin. The name of my database is **hospital_db** and it has 8 tables as mentioned above with all the necessary keys and constraints assigned to it.

Server: 127.0.0.1 » Database: hospital_db

Filters

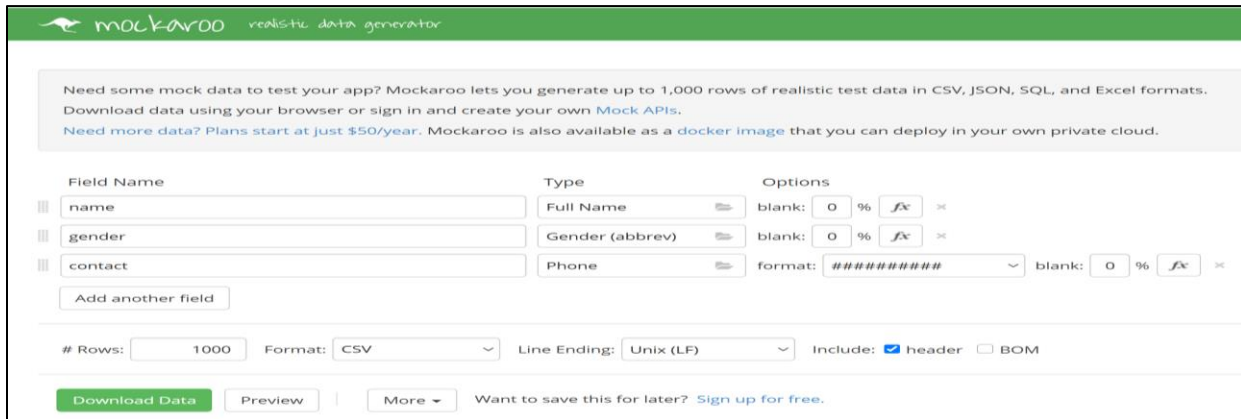
Containing the word:

Table	Action	Rows	Type	Collation	Size	Overhead
appointment	Browse Structure Search Insert Empty Drop	1,007	InnoDB	utf8mb4_general_ci	400.0 K B	-
doctor	Browse Structure Search Insert Empty Drop	98	InnoDB	utf8mb4_general_ci	32.0 K B	-
patient	Browse Structure Search Insert Empty Drop	1,502	InnoDB	utf8mb4_general_ci	128.0 K B	-
room	Browse Structure Search Insert Empty Drop	73	InnoDB	utf8mb4_general_ci	16.0 K B	-
room_patient	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	48.0 K B	-
room_staff	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_general_ci	48.0 K B	-
staff	Browse Structure Search Insert Empty Drop	425	InnoDB	utf8mb4_general_ci	48.0 K B	-
users	Browse Structure Search Insert Empty Drop	106	InnoDB	utf8mb4_general_ci	16.0 K B	-
8 tables	Sum	3,213	InnoDB	utf8_general_ci	736.0 K B	0 B



To generate data for my user, patient and doctor table, I made use of **mockaroo** website. It allows you to generate up to 1000 rows of realistic data with fields as per your requirement into a CSV file.

Link: <https://www.mockaroo.com/>



The screenshot shows the Mockaroo website interface. At the top, there's a green header with the Mockaroo logo and the text "realistic data generator". Below the header, there's a grey box with introductory text: "Need some mock data to test your app? Mockaroo lets you generate up to 1,000 rows of realistic test data in CSV, JSON, SQL, and Excel formats. Download data using your browser or sign in and create your own Mock APIs. Need more data? Plans start at just \$50/year. Mockaroo is also available as a docker image that you can deploy in your own private cloud." Below this, there's a form with three columns: "Field Name", "Type", and "Options". The "Field Name" column has three rows: "name", "gender", and "contact". The "Type" column has three rows: "Full Name", "Gender (abbrev)", and "Phone". The "Options" column has three rows: "blank: 0 % fX x", "blank: 0 % fX x", and "format: ##### blank: 0 % fX x". Below the form, there's a section for "Download Data" with fields for "# Rows: 1000", "Format: CSV", "Line Ending: Unix (LF)", and "Include: [checked] header [unchecked] BOM". There are buttons for "Download Data", "Preview", and "More". At the bottom, there's a link "Want to save this for later? Sign up for free."

I used this website for generating data for patient (1000+), user (100+), doctor (98), staff (400+) and appointment (1000+) table mainly. For the primary key entries in these tables, I added a column in the CSV file I get from above and added the primary key value for the first record in the format I need (eg. P00001 or R0001) and filled it in series for rest of the table.

In order to take care of the foreign key Log_ID in doctor table, I first generate records for the table users. From the CSV file, I filtered the data where the user type is a doctor took out the columns ID and Name and placed it in another CSV file. I then separately generate data for the remaining fields in doctor table and then merged both the CSVs. Thus, maintaining the uniqueness of the primary key and also references made to the foreign key. Similar approach was followed for appointment tables as well with Patient ID and Doctor ID as foreign key.

Joins -

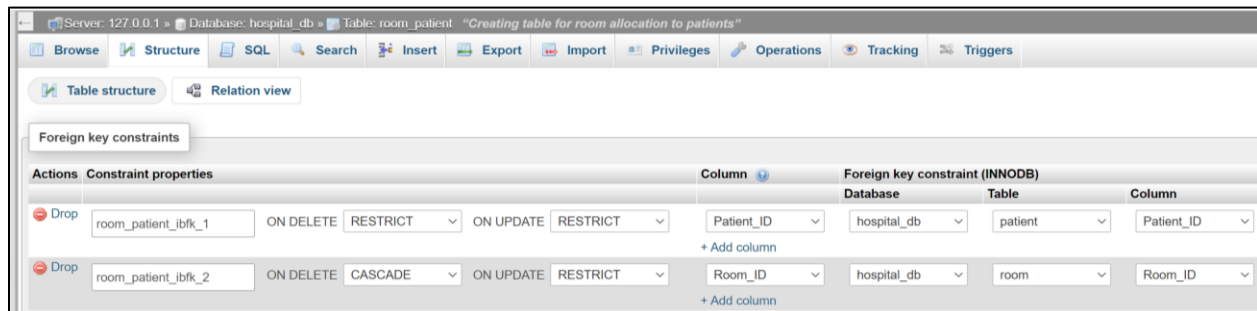
In this application, I have been required to create multiple joins between relations.

- I performed an **inner join** among the three tables – appointment, patient and doctor. This was done so that I get to access all the appointments created in the database and the patient name and doctor name associated with it.
- I also performed a **left join** on the table patient, appointment and room to get a view of the patient details, his appointment date and room assignment. I did a left join instead of an inner join here because I want to be able to see all records of the patient table even if they have no appointment or room assigned yet.

Constraints –

My database is set up in such a way that you are not allowed to delete a record from patient, doctor, user and staff table if that record is a foreign key in the tables - appointment, doctor, room_patient and room_staff. The UI however has no functionality provided to delete them. While in the database I have set up foreign key constraint **ON DELETE RESTRICT** in the tables appointment, room_patient, doctor and room_staff for keys in patient, doctor, user and staff. Also, **ON UPDATE RESTRICT** has been set for the same. This I have done because I believe that these are essential records for a hospital domain, and they should not be deleted. Also, the primary keys which act as identifiers to them should not be updated.

However, for room I have set up the foreign key constraint **ON DELETE CASCADE** in the tables room_patient and room_staff for key in room. This I have done because a record for room can be deleted. And, if a room has been deleted then its corresponding patient and staff assignment should also be deleted. Hence, CASCADE is used.



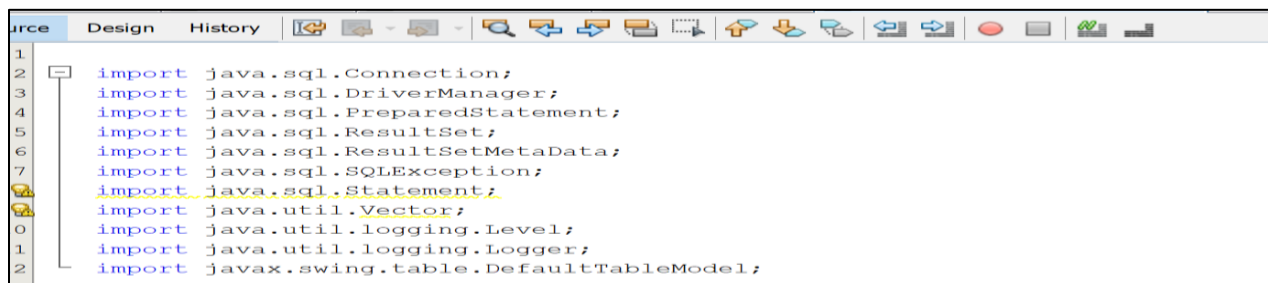
The room_patient table has only two attributes – Patient_ID and Room_ID. There is a one – one relationship between the room and patient table. Therefore, I have set both its attributes Room_ID and Patient_ID as **UNIQUE** in this table. Also, the UI takes care of not allowing multiple entries of the same Patient_ID or Room_ID in the room_patient table.

While the room_staff tables also has two attributes – Staff_ID and Room_ID. There is a **many – many relationship** between room and staff table. Therefore, no unique constraint is set for Room_ID and Staff_ID.

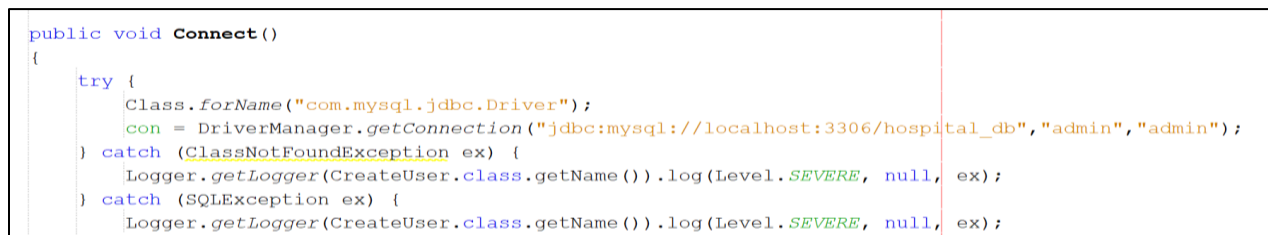
User Interface:

For my project, I have created a UI in **JAVA** in Netbeans IDE and my database is in **MYSQL** phpMyAdmin. I have used **JDBC** connector for connecting my database to the application I was creating. The jar file for this is included in the libraries of my project.

I imported multiple classes as below from the **java.sql.*** package in order to connect the java program to the MySQL database and to implement the SQL queries.



The DriverManager.getConnection method here is used to create a connection to my database – “hospital_db” with username – “admin” and password – “admin” using JDBC is done as shown below –



I have written SQL queries in JAVA using the below syntax –

```

pst = con.prepareStatement("select * from room_patient where Patient_ID = ?");
pst.setString(1, pat_id);

rs = pst.executeQuery();
if (rs.next()) {
    String oldroom = rs.getString("Room_ID");
    pst = con.prepareStatement("update room set Room_Status=? where Room_ID = ?");
    pst.setString(1, "Unassigned");
    pst.setString(2, oldroom);

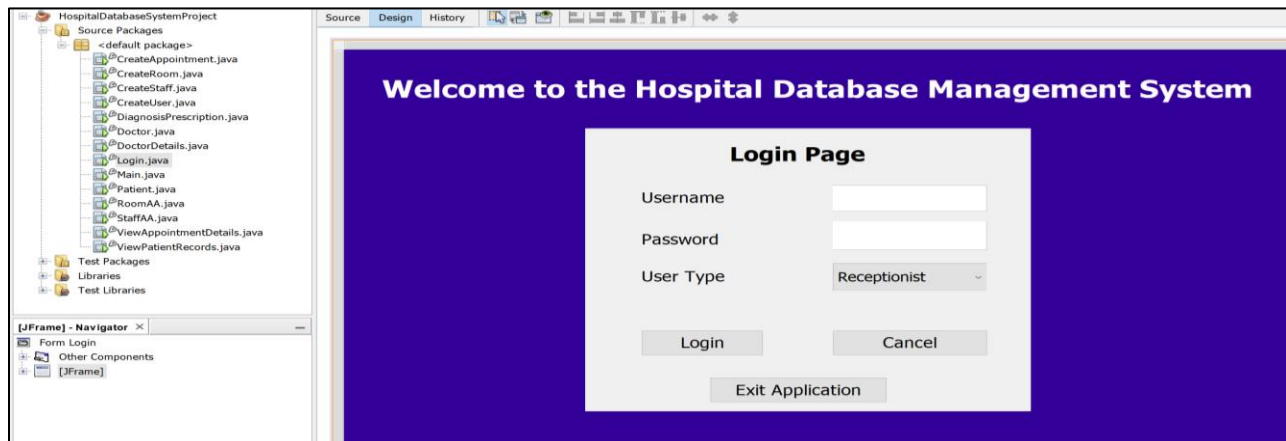
    pst.executeUpdate();

    pst = con.prepareStatement("update room_patient set Room_ID=? where Patient_ID = ?");
    pst.setString(1, rid);
    pst.setString(2, pat_id);

    pst.executeUpdate();
    JOptionPane.showMessageDialog(this, "Room assignment updated sucessfully");
}
else {
    pst = con.prepareStatement("insert into room_patient (Patient_ID,Room_ID) values (?, ?)");
    pst.setString(1, pat_id);
    pst.setString(2, rid);
}

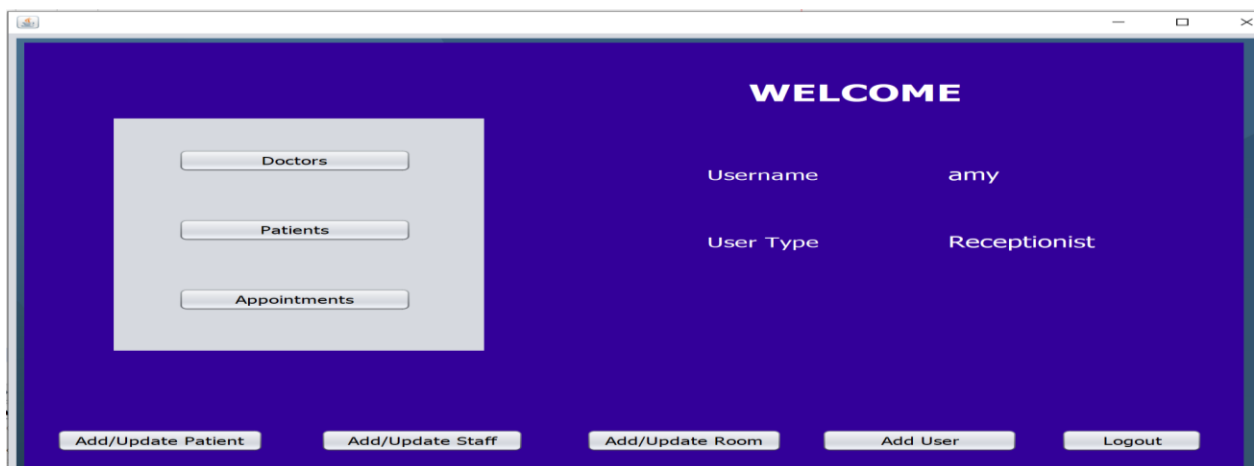
```

I have created multiple .java files as below according to the required functionality and made calls to them accordingly. **Login.java** is the main file which is launched when we run the application –



There are two possible users for my application – **Receptionist and Doctor.**

Receptionist:



- As a receptionist, I can add a new patient, staff, room and user to the database. This is implemented using **INSERT** query.
- I can create an appointment for a patient with a doctor using **INSERT** query in the appointment table
- I can view all existing records of patient, doctor and appointments stored in the database in the view I desire using **SELECT** query along with the appropriate **JOINS** where necessary.
- I can also search and filter out records from the patient, doctor and appointment tables in the database using **SELECT** query with the appropriate **WHERE CONDITION** using **LIKE** and **EXPRESSION MATCHING**
- I can edit the records for patient, staff and room type using **UPDATE** query
- I can assign room to a patient and staff using **INSERT** query and remove the assignments using **DELETE** query. When a patient is assigned to a room, the status of the room is automatically updated to "Assigned" status using **UPDATE** query. I can also update the assignment of patient to a room using **UPDATE** query and automatically update the room status of his old room to "Unassigned".
- I can also filter out unassigned rooms according to their type using **SELECT** query and **WHERE CONDITION** which checks the Room_Status attribute of room table.

Patient Database

Search

Patient ID	Patient Name	Sex	Contact	Address	Appointment Date	Room Number
P00001	Anna Todd	F	9852014788	College Station, ...	2020-10-01	R0001
P00002	Rick Morley	M	2452583360	Dallas, Texas	2020-10-09	
P00003	Had Errichiello	M	2028444802	Washington, Dis...		
P00004	Halsey Hakey	M	7205886445	Denver, Colorado		
P00005	Marlowe Morley	M	4048003411	Atlanta, Georgia	2020-10-17	R0011
P00006	Abdel Garces	M	9049794693	Saint Augustine,...	2020-10-16	
P00007	Allina Maris	F	4144764158	Milwaukee, Wisc...	2020-10-20	
P00008	Lib O'Giany	F	8169078452	Kansas City, Mis...	2020-07-19	
P00009	Benedetto Dagwelli	M	7652190360	Muncie, Indiana	2020-07-31	
P00010	Dewitt Adrianello	M	6166498489	Grand Rapids, M...	2020-10-20	R0010
P00010	Dewitt Adrianello	M	6166498489	Grand Rapids, M...	2020-03-17	R0010
P00011	Sarah Olivetta	F	4194071061	Toledo, Ohio	2020-02-14	
P00012	Bern Gelling	M	2547324116	Gatesville, Texas	2020-05-21	
P00013	Cass Nolton	M	3048691036	Huntington, Wes...	2020-09-14	

Doctor:

WELCOME

Username
john

User Type
Doctor

- As a doctor, I will be able to add my details to the system using **INSERT** query and update my details using **UPDATE** query.
- I can view other doctors in the database and search doctors as per their name or specialty using **SELECT** query with the appropriate **WHERE CONDITION** using **LIKE** and **EXPRESSION MATCHING**. However, I will not be able to edit the details of other doctors in the database.
- In a similar way, I can also view the appointments in the system and search for my appointment on any date.
- I can also add a diagnosis and prescription for a patient during an appointment to the database using **UPDATE** query

Appointment ID	Patient ID	Patient Name	Doctor Name	Appointment Date	Diagnosis	Prescription
A00008	P00008	Lib O'Giany	Skyler Monroe	2020-07-19	Activity, oth involv...	Monitor Skin/Brea...
A00290	P00290	Matteo Paske	Skyler Monroe	2020-07-01	Anal sphincter tea...	Supplement R Gl...
A00764	P00764	Base Roelvink	Skyler Monroe	2020-07-09	Breakdown (mech...	Excision of Bladd...

Additionally, the application uses SQL queries with **inner joins** and **left joins** to combine two to three tables to give us a better overall view of the records in the database. For example, in the above image we have selected the required columns to display by performing an inner join between appointment, patient and doctor table.

Also, I have made use of **aggregate function – MAX** in order to auto increment and generate the primary key for patient, doctor, staff and room table.

```
Statement s = con.createStatement();
rs = s.executeQuery("select MAX(Patient_ID) from patient");
rs.next();
rs.getString("Max(Patient_ID)");
```

Discussion:

- This project gave me an opportunity to create a database application all by myself. I got a very practical experience of implementing all my theoretical concepts related to database like generating an Entity Relationship diagram according to our application requirement, converting it to a relational schema, finding functional dependencies, normalization in BCNF, creating a database schema with all the necessary constraints and keys in a real world application.
- I observed an interesting thing while converting ER diagram to tables and then normalization. Initially when I converted the entity sets and relationships into their corresponding table, I got a total of 8 tables. Then, as mentioned in the textbook I combined two tables that were a part of a

many one relationship and then got a total of 7 tables. However, later when checked for Functional dependency violation, I realized the table I combine above had a FD that violated BCNF. Normalizing this table according to BCNF then gave me the same two original tables that I had combined before. Hence giving me 8 tables again.

- Also, while checking for FDs and normalization process I learnt that relations that have no non-trivial FDs are in BCNF by default. Since, BCNF violators only consider non-trivial FDs for violation.
- I learnt more about the significance of assigning foreign key constraints and the difference between setting the constraint to RESTRICT, CASCADE and SET NULL. I understood better about where I should use which constraint according to my application requirement.
- Similarly, I understood better about joins and their types and practical usage of when I should use left join instead of inner join.
- I learnt a lot about how I could establish a connection between my database and JAVA application using JDBC connection and how to write and implement SQL queries in JAVA. It was a challenge for me because I did not know much about it and had never used it before. So, I referred to a lot of resources which gave me a better understanding of the same and made it helped me a lot while I was creating the application.
- Initially, I did face issues in connecting to the database. It was either because I did not place the JDBC connectors .jar file in the libraries or when I did not put the correct URL, username and password in the DriverManager.getConnection method. However, after a few tries I was able to troubleshoot and fix the issues I was facing with establishing the connection.
- Also, it was a challenge for me to set the foreign key constraints in the database correctly. Even after setting constraints, I still had instances where the constraints were not being enforced. After multiple attempts, troubleshooting and referring to various resources, I figured out that in order to set foreign key constraints, the two keys (Foreign Key and the Primary Key it references to) should have the same data type. The Foreign keys should also be indexed, and the table should be created using INNO DB engine which supports foreign keys. After ensuring all of this, I was able to get the tables set up with the correct foreign key constraints.
- Finally, to make sure that I could package and distribute the Java application I created correctly, I read about how I could create a .jar file of my source code, zip and share it so that another user can run it on their system in the easiest way possible.

Overall, there were quite some challenges I faced while developing a complete database application from start to end. But I got to learn a lot of new things in the entire process and implemented a lot of concepts which I might not have been exposed to until I created an application like this by myself.

Thank you!

References:

1. Random Data Generation - <https://www.mockaroo.com/>
2. Connect to MySQL using JDBC Driver - <https://www.mysqltutorial.org/connecting-to-mysql-using-jdbc-driver/>
3. JDBC Tutorial - <https://www.tutorialspoint.com/jdbc/index.htm>
4. Setting foreign keys –
<https://dev.mysql.com/doc/refman/5.6/en/create-table-foreign-keys.html>
<https://bobcares.com/blog/create-foreign-key-mysql-phpmyadmin/>
5. Packaging and Distributing Java Applications - <https://netbeans.org/kb/docs/java/javase-deploy.html>