Database Technology

Clinic Appointment System

BY:

Ella Raputri - 2702298154

Farrell Sevillen Arya - 2702323540

Monish Haresh Giani - 2702387834

Class L3AC

Computer Science Program

School Of Computing and Creative Arts

Bina Nusantara International University

Jakarta

2024

Table of Contents

Clinic Appointment System Report

A. Introduction

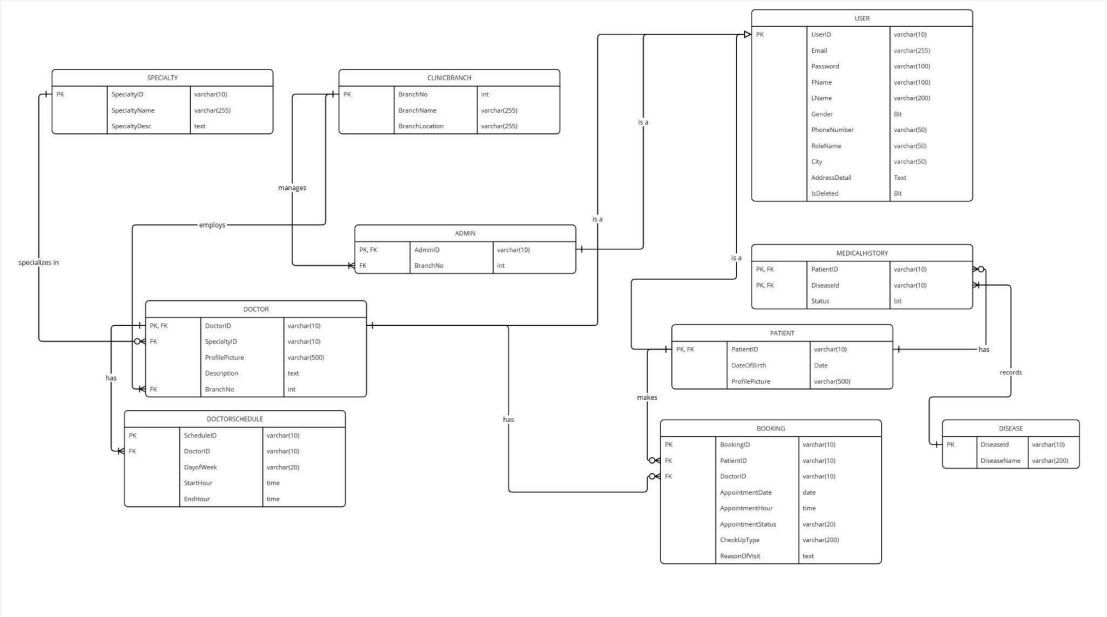
An effective management system is important in any industry. Healthcare is not an exception to it. Having a proper management system is important in maintaining a smooth and effective flow in the clinic. According to sources such as Otakoyi[1], Techimply[2], and Easy Solutions Infosystems[3], having a management system in the healthcare industry improves speed, efficiency, and data integration. In the Database Technology course, we were given the task of building a clinic appointment management system.

The system we made will have three parts (based on the roles): Patient, Administrator, and Doctor. We made the system so that the user interface is easy to use and that all the necessary data will be stored in a database. The patients will have the functionality to book and schedule their appointments. The Doctors will have access to their schedule and patient information. The administrator will be able to edit the data that is in the system based on their assigned branch.

B. Entity Relationship Diagram (ERD)

The below ERD can also be viewed in the "ERD - 3NF" file in this folder. https://drive.google.com/drive/folders/1-JWwAES_9EpL5-sQy1n1OCsHtQVmjraU?usp=sharing

[1] https://otakoyi.software/blog/key-clinic-management-system-features-and-benefits-enhancing-efficiency-and-patient-care
[2] https://www.techimply.com/blog/importance-of-clinic-management-software
[3] https://easysolution.in/hospital-management-system/blog/benefits-of-using-a-cms.php



The relationship between each entity is

- Admin, Doctor, and Patient entities are the specialization of the User entity. In other words, Admin, Doctor, and Patient each can be seen as a subclass of User, where each of them has common attributes in User, but also has their own unique attributes.
- Each specialty entity can have many doctors.
- Each branch entity can employ many admins and doctors. Patients are not tied to branches because we assume one patient can go to more than one clinic branch.
- Each doctor entity can have many schedules. A doctor can have more than one schedule in one day as long the start and end hours are different.
- A patient can have more than one medical history and one disease can also have more than one patient having that disease (medical history).
- A patient and a doctor can have zero to many bookings.

C. Normalization

Original UNF Table Columns

- BookingId
- PatientId

- DoctorId
- AppointmentDate
- AppointmentHour
- AppointmentStatus
- CheckUpType
- ReasonOfVisit
- PatientEmail
- PatientPassword
- PatientFirstName
- PatientLastName
- PatientGender
- PatientPhoneNumber
- PatientCity
- PatientAddressDetail
- PatientIsDeleted
- PatientDateOfBirth
- PatientProfilePicture
- DiseaseId
- DiseaseName
- Status
- DoctorEmail
- DoctorPassword
- DoctorFirstName
- DoctorLastName
- DoctorGender
- DoctorPhoneNumber
- DoctorCity
- DoctorAddressDetail
- DoctorIsDeleted
- DoctorSpecialtyId

- DoctorSpecialtyName
- DoctorSpecialtyDesc
- DoctorProfilePicture
- DoctorDesc
- ScheduleId
- DoctorAvailableDay
- StartHour
- EndHour
- BranchNo
- BranchName
- BranchLocation
- BranchAdminId
- AdminEmail
- AdminPassword
- AdminFirstName
- AdminLastName
- AdminGender
- AdminPhoneNumber
- AdminCity
- AdminAddressDetail
- AdminIsDeleted

1NF Rules:

1. The table should have a primary key.
2. Each column must contain atomic values (no multi-valued or composite attributes).
3. Each record must be unique.

1NF Table Columns

- BookingId

- PatientId
- DoctorId
- AppointmentDate
- AppointmentHour
- AppointmentStatus
- CheckUpType
- ReasonOfVisit
- PatientEmail
- PatientPassword
- PatientFirstName
- PatientLastName
- PatientGender
- PatientPhoneNumber
- PatientCity
- PatientAddressDetail
- PatientIsDeleted
- PatientDateOfBirth
- PatientProfilePicture
- DiseaseId
- DiseaseName
- Status
- DoctorEmail
- DoctorPassword
- DoctorFirstName
- DoctorLastName
- DoctorGender
- DoctorPhoneNumber
- DoctorCity
- DoctorAddressDetail
- DoctorIsDeleted

- DoctorSpecialtyId
- DoctorSpecialtyName
- DoctorSpecialtyDesc
- DoctorProfilePicture
- DoctorDesc
- ScheduleId
- DoctorAvailableDay
- StartHour
- EndHour
- BranchNo
- BranchName
- BranchLocation
- BranchAdminId
- AdminEmail
- AdminPassword
- AdminFirstName
- AdminLastName
- AdminGender
- AdminPhoneNumber
- AdminCity
- AdminAddressDetail
- AdminIsDeleted

1NF Functional Dependency

Primary key: BookingId, DiseaseId, ScheduleId

- BookingId → PatientId, DoctorId, AppointmentId, AppointmentHour, AppointmentStatus, CheckUpType, ReasonOfVisit
- PatientId → PatientEmail, PatientPassword, PatientFirstName, PatientLastName, PatientGender, PatientPhoneNumber, PatientCity, PatientAddressDetail, PatientIsDeleted, PatientDateOfBirth, PatientProfilePicture,

DiseaseName

- DoctorId → DoctorEmail, DoctorPassword, DoctorFirstName, DoctorLastName, DoctorGender, DoctorPhoneNumber, DoctorCity, DoctorAddressDetail, DoctorIsDeleted, DoctorDateOfBirth, DoctorProfilePicture, DoctorDesc, ScheduleId, BranchNo
- DiseaseId (partial dependency) → DiseaseName
- PatientId, DiseaseId (partial dependency) → Status
- DoctorSpecialtyId → DoctorSpecialtyName, DoctorSpecialtyDesc
- ScheduleId (partial dependency) → DoctorAvailableDay, StartHour, EndHour, DoctorId
- BranchNo → BranchName, BranchLocation
- BranchAdminId → BranchNo, AdminEmail, AdminPassword, AdminFirstName, AdminLastName, AdminGender, AdminPhoneNumber, AdminCity, AdminAddressDetail, AdminIsDeleted

---

2NF Rules:

1. Must be in 1NF.
2. No partial dependencies (attributes should not depend on part of a composite key).

2NF Tables and Their Columns

The red font color is the primary key.

Table Booking

- **BookingId**
- PatientId
- DoctorId
- AppointmentDate
- AppointmentHour
- AppointmentStatus

- CheckUpType
- ReasonOfVisit
- PatientEmail
- PatientPassword
- PatientFirstName
- PatientLastName
- PatientGender
- PatientPhoneNumber
- PatientCity
- PatientAddressDetail
- PatientIsDeleted
- PatientDateofBirth
- PatientProfilePicture
- **DiseaseId**
- DoctorEmail
- DoctorPassword
- DoctorFirstName
- DoctorLastName
- DoctorGender
- DoctroPhoneNumber
- DoctorCity
- DoctorAddressDetail
- DoctorIsDeleted
- DoctorSpecialtyId
- DoctorSpecialtyName
- DoctorSpecialtyDesc
- DoctroProfilePicture
- DoctorDesc
- **ScheduleId**
- BranchNo

- BranchName
- BranchLocation
- BranchAdminId
- AdminEmail
- AdminPassword
- AdminFirstName
- AdminLastName
- AdminGender
- AdminPhoneNumber
- AdminCity
- AdminAddressDetail
- AdminIsDeleted

Table MedicalHistory

- **PatientId**
- **DiseaseId**
- Status

Table Disease

- **DiseaseId**
- DiseaseName

Table Schedule

- **ScheduleId**
- DoctorAvailableDay
- SartHour
- EndHour
- DoctorId

2NF Functional Dependency for The Abnormal One (Table Booking)

- BookingId → PatientId, DoctorId, AppointmentDate, AppointmentHour, AppointmentStatus, CheckUpType, ReasonOfVisit

- PatientId → PatientEmail, PatientPassword, PatientFirstName, PatientLastName, PatientGender, PatientPhoneNumber, PatientCity, PatientAddressDetail, PatientIsDeleted, PatientDateOfBirth.PatientProfilePicture

- DoctorId → DoctorEmail, DoctorPassword, DoctorFirstName, DoctorLastName, DoctorGender, DoctorPhoneNumber, DoctorCity, DoctorAddressDetail, DoctorIsDeleted, DoctorSpecialtyId, DoctorProfilePicture, DoctorDesc, BranchNo
- DoctorSpecialtyId → DoctorSpecialtyName, DoctorSpecialtyDesc
- BranchNo → BranchName, BranchLocation
- BranchAdminId → BranchNo, AdminEmail, AdminPassword, AdminFirstName, AdminLastName, AdminGender, AdminPhoneNumber, AdminCity, AdminAddressDetail, AdminIsDeleted

---

3NF Rules:

1. Must be in 2NF.
2. No transitive dependencies (non-key attributes should not depend on other non-key attributes).

3NF Tables (Remove Transitive Dependencies from Booking)

Table Booking

- **BookingId**
- PatientId
- DoctorId
- AppointmentDate
- AppointmentHour

- AppointmentStatus
- CheckUpType
- ReasonOfVisit

Table MedicalHistory

- **PatientId**
- **DiseaseId**
- Status

Table Disease

- **DiseaseId**
- DiseaseName

Table Schedule

- **ScheduleId**
- DoctorAvailableDay
- StartHour
- EndHour
- DoctorId

Table Patient

- **PatientId**
- PatientEmail
- PatientPassword
- PatientFirstName
- PatientLastName
- PatientGender
- PatientPhoneNumber
- PatientCity
- PatientAddressDetail

- PatientIsDeleted
- PatientDateOfBirth
- PatientProfilePicture

Table Doctor

- **DoctorId**
- DoctorEmail
- DoctorPassword
- DoctorFirstName
- DoctorLastName
- DoctorGender
- DoctorPhoneNumber
- DoctorCity
- DoctorAddressDetail
- DoctorIsDeleted
- DoctorSpecialtyId
- DoctorSpecialtyName
- DoctorSpecialtyDesc
- DoctorProfilePicture
- DoctorDesc
- BranchNo
- BranchName
- BranchLocation

Table Admin

- **BranchAdminId**
- AdminEmail
- AdminPassword
- AdminFirstName
- AdminLastName

- AdminGender
- AdminPhoneNumber
- AdminCity
- AdminAddressDetail
- AdminIsDeleted
- BranchNo
- BranchName
- BranchLocation

Remaining transitive dependencies:

- DoctorSpecialtyId → DoctorSpecialtyName, DoctorSpecialtyDesc (in table Doctor)
- BranchNo → BranchName, BranchLocation (in table Doctor and table Admin)

3NF Tables (Remove Remaining Transitive Dependencies)

Table Booking

- **BookingId**
- PatientId
- DoctorId
- AppointmentDate
- AppointmentHour
- AppointmentStatus
- CheckUpType
- ReasoneOfVisit

Table MedicalHistory

- **PatientId**
- **DiseaseId**
- Status

Table Disease

- **DiseaseId**
- DiseaseName

Table Schedule

- **ScheduleId**
- DoctorAvailableDay
- StartHour
- EndHour
- DoctorId

Table Patient

- **PatientId**
- PatientEmail
- PatientPassword
- PatientFirstName
- PatientLastName
- PatientGender
- PatientPhoneNumber
- PatientCity
- PatientAddressDetail
- PatientIsDeleted
- PatientDateOfBirth
- PatientProfilePicture

Table Doctor

- **DoctorId**
- DoctorEmail
- DoctorPassword
- DoctorFirstName

- DoctorLastName
- DoctorGender
- DoctorPhoneNumber
- DoctorCity
- DoctorAddressDetail
- DoctorIsDeleted
- DoctorSpecialtyId
- DoctorProfilePicture
- DoctorDesc
- BranchNo

Table Admin

- **BranchAdminId**
- AdminEmail
- AdminPassword
- AdminFirstName
- AdminLastName
- AdminGender
- AdminPhoneNumber
- AdminCity
- AdminAddressDetail
- AdminIsDeleted
- BranchNo

Table Specialty

- **DoctorSpecialtyId**
- DoctorSpecialtyName
- DoctorSpecialtyDesc

Table Branch

- **BranchNo**
- BranchName
- BranchLocation

Final 3NF (Generalization Common Attributes of Admin, Doctor, Patient into User Table)

Table Booking

- **BookingId**
- PatientId
- DoctorId
- AppointmentDate
- AppointmentHour
- AppointmentStatus
- CheckUpType
- ReasonOfvisit

Table MedicalHistory

- **PatientId**
- **DiseaseId**
- Status

Table Disease

- **DiseaseId**
- DiseaseName

Table Schedule

- **ScheduleId**
- DoctorAvailableDay
- StartHour

- EndHour
- DoctorId

Table User

- **UserId**
- UserEmail
- UserPassword
- UserFirstName
- UserLastName
- UserGender
- UserPhoneNumber
- UserCity
- UserAddressDetail
- UserIsDeleted
- RoleName

Table Patient

- **PatientId**
- PatientDateOfBirth
- PatientProfilePicture

Table Doctor

- **DoctorId**
- DoctorSpecialtyId
- DoctorProfilePicture
- DoctorDesc
- BranchNo

Table Admin

- **AdminId**

- BranchNo

Table Specialty

- **DoctorSpecialtyId**
- DoctorSpecialtyName
- DoctorSpacialtyDesc

Table Branch

- **BranchNo**
- BranchName
- BranchLocation

The ERD version of the UNF and 1NF, 2NF, and 3NF for the tables can be viewed in this folder.
https://drive.google.com/drive/folders/1-JWwAES_9EpL5-sQy1n1OCsHtQVmjraU?usp=sharing

D. Table Structure

Below is the table structure of the tables we have created for the clinic appointment system. You will be able to find the names of the attributes we created and the details of them such as the data type, constraints, and their description.

Table Booking

Column Name	Data Type	Constraints	Description
BookingId	VARCHAR(10)	PRIMARY KEY, NOT NULL	Unique identifier for a booking.

PatientId	VARCHAR(10)	FOREIGN KEY, NOT NULL	Refers to the patient making the booking.
DoctorId	VARCHAR(10)	FOREIGN KEY, NOT NULL	Refers to the doctor for the booking.
AppointmenDate	DATE	NOT NULL	Date of the appointment.
AppointmentHour	TIME	NOT NULL	Time of the appointment.
AppointmentStatus	VARCHAR(20)	NOT NULL	Status of the appointment (e.g., Pending, Completed, Cancelled).
CheckUpType	VARCHAR(200)	NOT NULL	Type of check-up for the appointment.
ReasonOfVisit	TEXT	NOT NULL	Reason for the visit.

Medical History Table

Column Name	Data Type	Constraints	Description
PatientId	VARCHAR(10)	PRIMARY KEY, FOREIGN KEY, NOT NULL	Refers to the patient.
DiseaseId	VARCHAR(10)	PRIMARY KEY, FOREIGN KEY, NOT NULL	Refers to the disease.
Status	BIT	NOT NULL	Current status of the disease

			(Ongoing or Recovered).

Disease Table

Column Name	Data Type	Constraints	Description
DiseaseId	VARCHAR(10)	PRIMARY KEY, NOT NULL	Unique identifier for the disease.
DiseaseName	VARCHAR(200)	NOT NULL	Name of the disease

DoctorSchedule Table

Column Name	Data Type	Constraints	Description
ScheduleId	VARCHAR(10)	PRIMARY KEY, NOT NULL	Unique identifier for the schedule.
DoctorId	VARCHAR(10)	FOREIGN KEY, NOT NULL	Refers to the doctor for the schedule.
DayOfWeek	VARCHAR(20)	NOT NULL	Day of availability (e.g., Monday).
StartHour	TIME	NOT NULL	Start time of availability.
EndHour	TIME	NOT NULL	End time of availability.

User Table

Column Name	Data Type	Constraints	Description
UserId	VARCHAR(10)	PRIMARY KEY, NOT NULL	Unique identifier for the user.
Email	VARCHAR(255)	NOT NULL, UNIQUE	User's email address.
Password	VARCHAR(100)	NOT NULL	Password for the user.
FirstName	VARCHAR(100)	NOT NULL	User's first name.
LastName	VARCHAR(200)	NOT NULL	User's last name.
Gender	BIT	NOT NULL	Gender of the user.
PhoneNumber	VARCHAR(50)	NOT NULL	Contact number of the user.
RoleName	VARCHAR(50)	NOT NULL	Role of the user (Doctor, Admin, or Patient).
City	VARCHAR(50)	NOT NULL	City where the user resides.
AddressDetail	TEXT	NOT NULL	Detailed address of the user
IsDeleted	BIT	DEFAULT 0	Indicates if the user account is deleted or not

Patient Table

Column Name	Data Type	Constraints	Description
PatientId	VARCHAR(10)	PRIMARY KEY, FOREIGN KEY, NOT NULL	Refers to the user who is a patient.
DateOfBirth	DATE	NOT NULL	Patient’s date of birth.
ProfilePicture	VARCHAR(500)	DEFAULT NULL	URL or path to the profile picture.

Doctor Table

Column Name	Data Type	Constraints	Description
DoctorId	VARCHAR(10)	PRIMARY KEY, FOREIGN KEY, NOT NULL	Refers to the user who is a doctor.
SpecialtyId	VARCHAR(10)	NOT NULL	Refers to the doctor's specialty.
ProfilePicture	VARCHAR(500)	DEFAULT NULL	URL or path to the profile picture.
Description	TEXT	DEFAULT NULL	Description of the doctor.
BranchNo	INT	NOT NULL	Refers to the branch the doctor is associated with.

Admin Table

Column Name	Data Type	Constraints	Description
AdminId	VARCHAR(10)	PRIMARY KEY, FOREIGN KEY, NOT NULL	Refers to the user who is an admin.
BranchNo	INT	FOREIGN KEY, NOT NULL	Refers to the branch the admin manages.

Specialty Table

Column Name	Data Type	Constraints	Description
SpecialtyId	VARCHAR(10)	PRIMARY KEY, NOT NULL	Unique identifier for the specialty
SpecialtyName	VARCHAR(255)	NOT NULL	Name of the specialty (e.g., Cardiology).
SpecialtyDescription	TEXT	DEFAULT NULL	Description of the specialty.

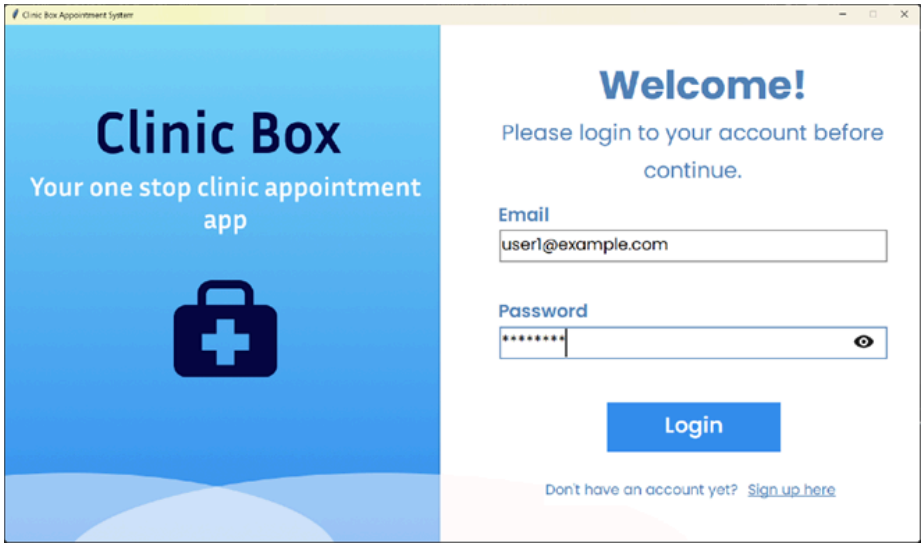
ClinicBranch Table

Column Name	Data Type	Constraints	Description
BranchNo	INT	PRIMARY KEY, NOT NULL	Unique identifier for the branch
BranchName	VARCHAR(255)	NOT NULL	Name of the branch
BranchLocation	VARCHAR(255)	NOT NULL	Location of the branch

E. UI Design

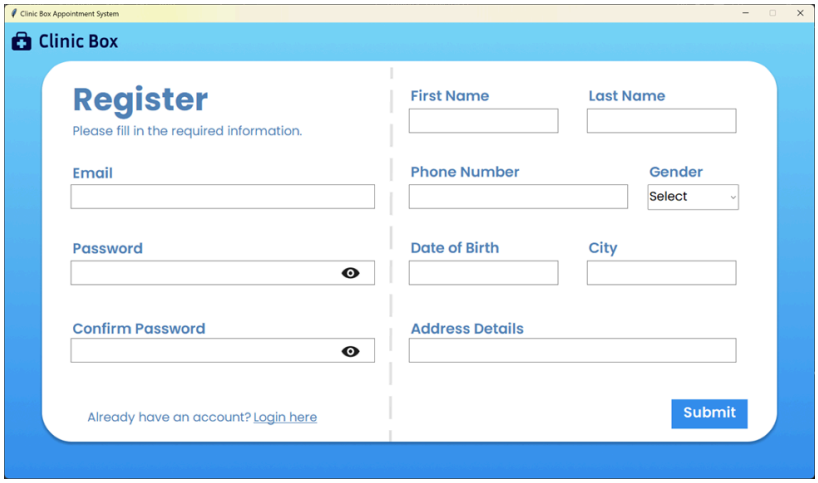
There are 3 roles for our app, which is Patient, Doctor, and Admin. Each role has its own different pages to adapt to their needs. Below are the screenshots of our UI.

1. Login Page



When a user first starts the program, they will see the Login Page. If the user already has an account, then they can log in. If not, then the user can sign up first.

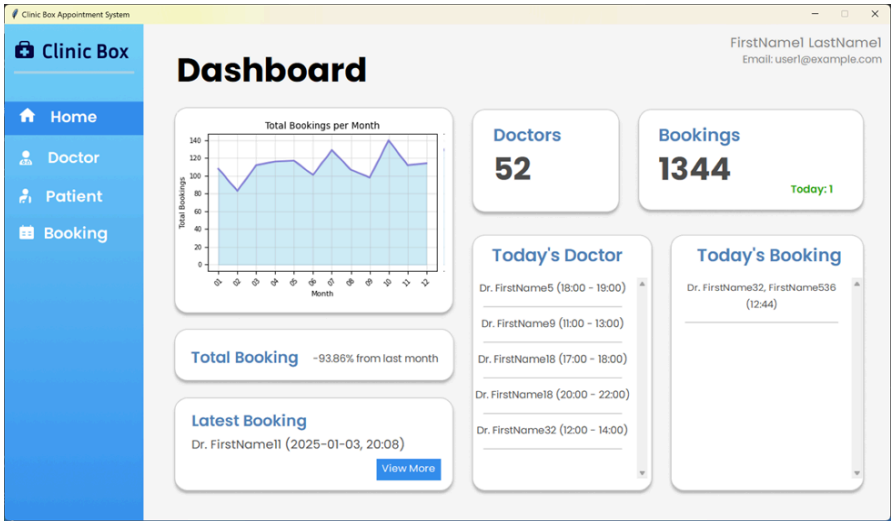
2. Register Page



The register page is only for the user who wants to register as a patient. We assume that the doctors will be added by the branch admin, while the branch

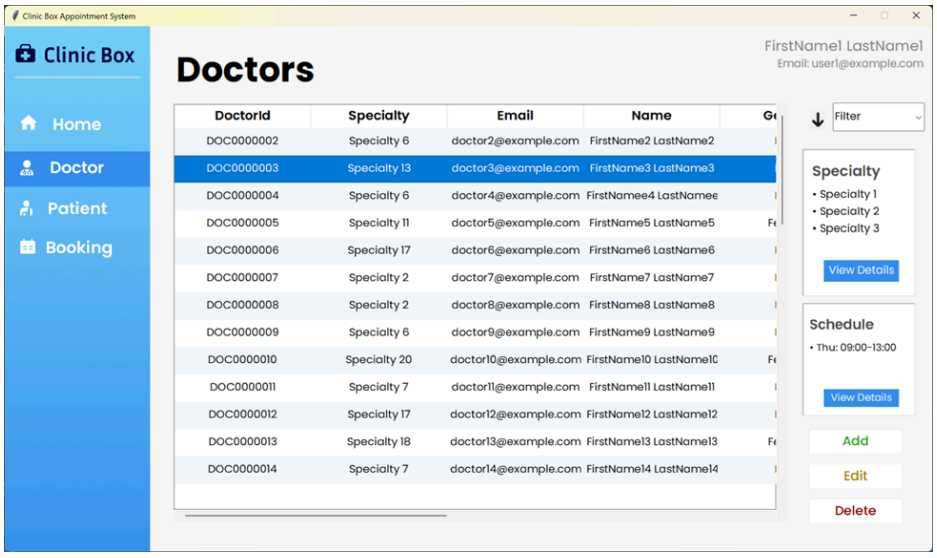
admin is added by the database administrator. User needs to fill in the required information to register.

3. Admin Section Dashboard



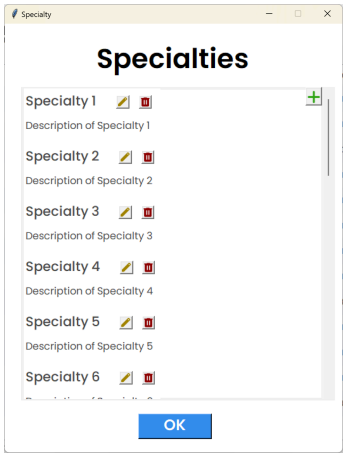
In the Admin Dashboard, the admin can see the information on booking and doctors for their corresponding branch. They can see the booking and doctor amount, the graph of total bookings per month, the latest booking, and today's doctor and booking.

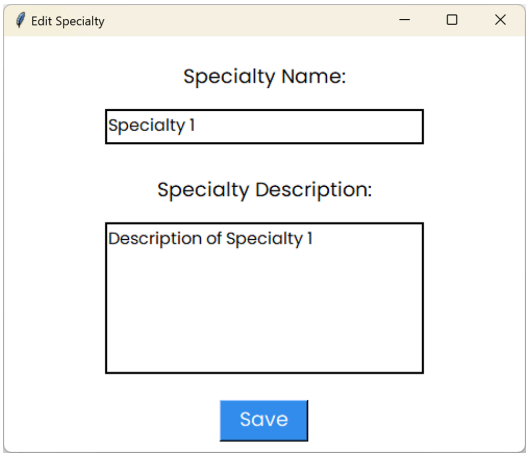
4. Admin Section Doctor Page



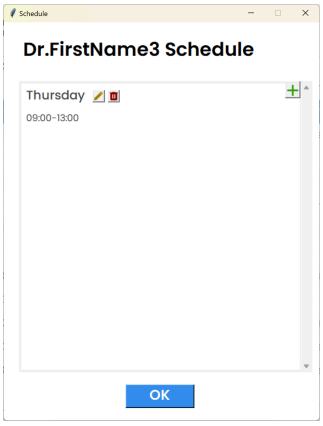
On the Admin Doctor Page, the admin can see a table with basic information of each doctor in their branch. When a record is clicked, the schedule of that doctor is also shown in the schedule panel on the right side. There is also a filter button to filter the records in the table based on the admin's needs.

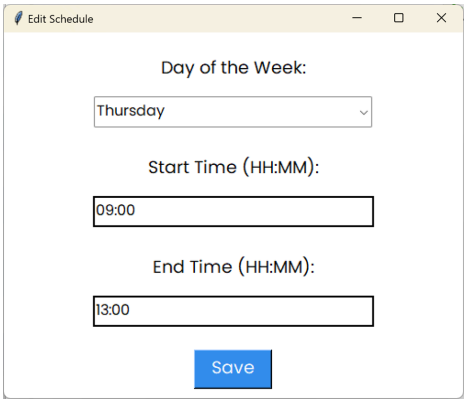
Here, the admin can also add, edit, and delete a specialty type from the database.





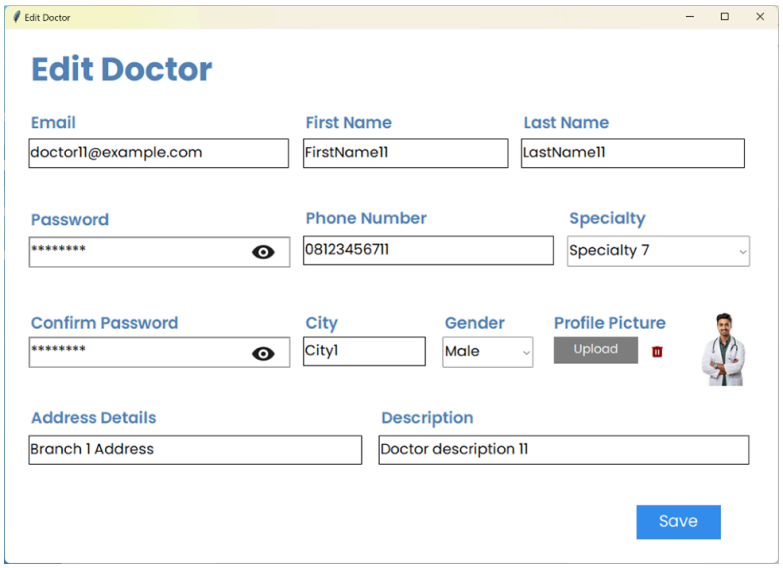
The edit specialty or add specialty window will each appear if the user clicks the edit or add button. Besides specialty, the admin can also add, edit, and delete doctor schedules. The UI for the schedule is similar to the UI for specialty.



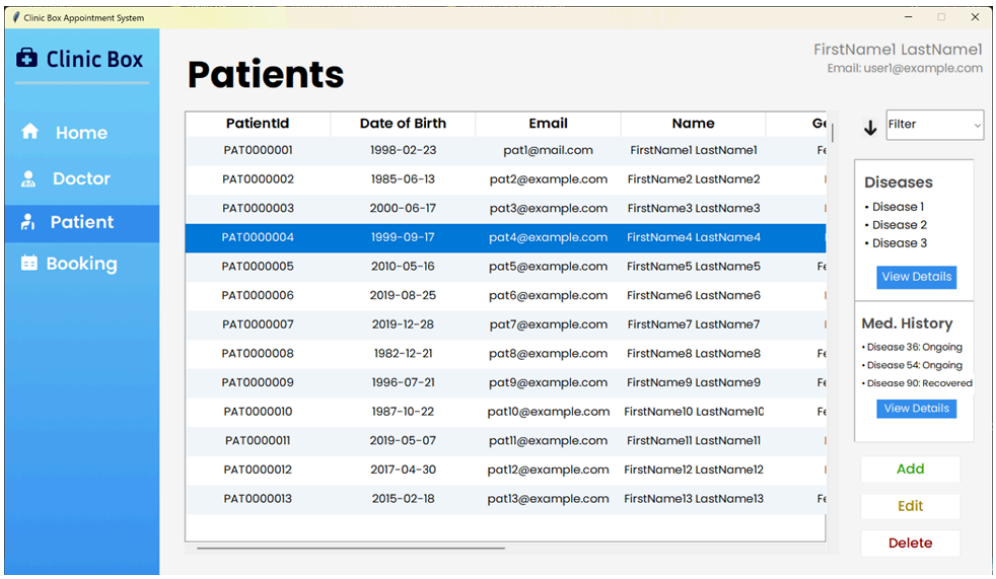


Besides specialty and schedule, the admin of a branch can also add, edit, and delete doctors in their corresponding branch by clicking the buttons in the bottom right of the admin doctor page. If the admin wants to edit or delete a doctor, they

need to select a record from the table first to perform that operation. Below is the UI for edit doctor, the UI for add doctor is similar to edit doctor, but the difference is edit doctor entries are loaded first based on the data in the database, while in the add doctor, all fields or entries are still empty when the window is opened.

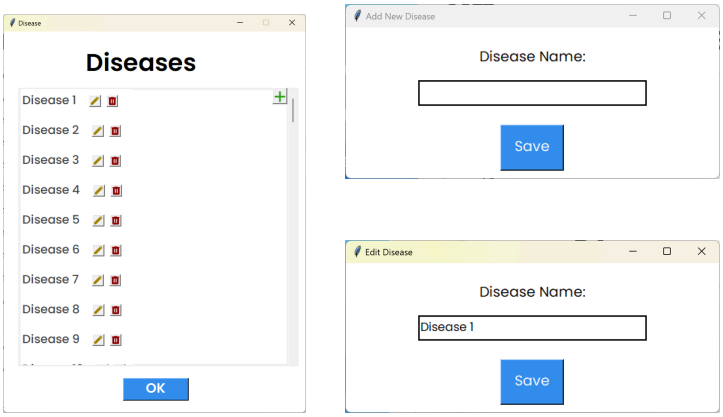


5. Admin Patient Page

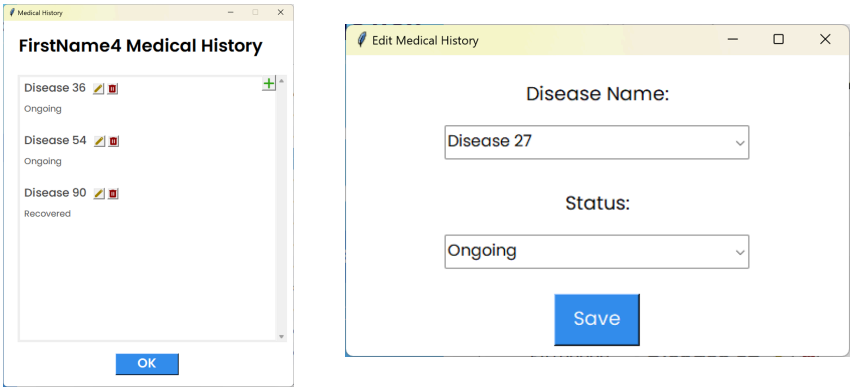


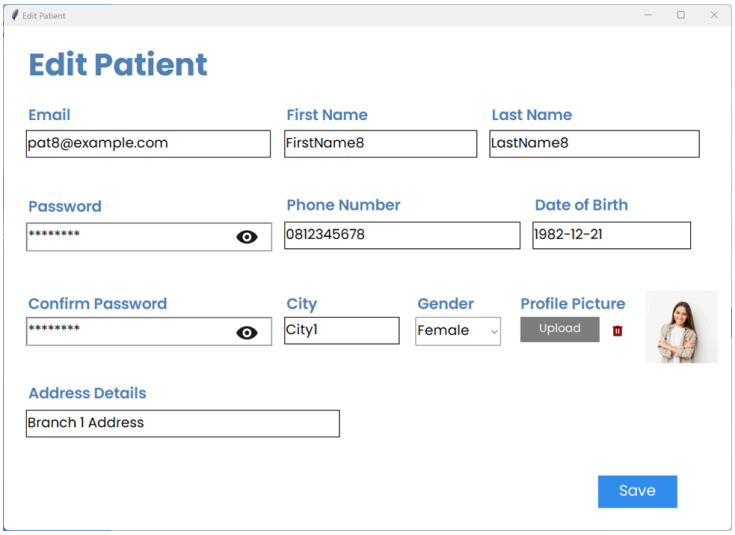
Admin Patient Page is similar to Admin Doctor Page, but instead of doctors, it displays the patient data and allows filtering. It also displays the type of disease

and the medical history of each patient. Admins are allowed to insert, update, and delete the types of diseases available in the hospital.

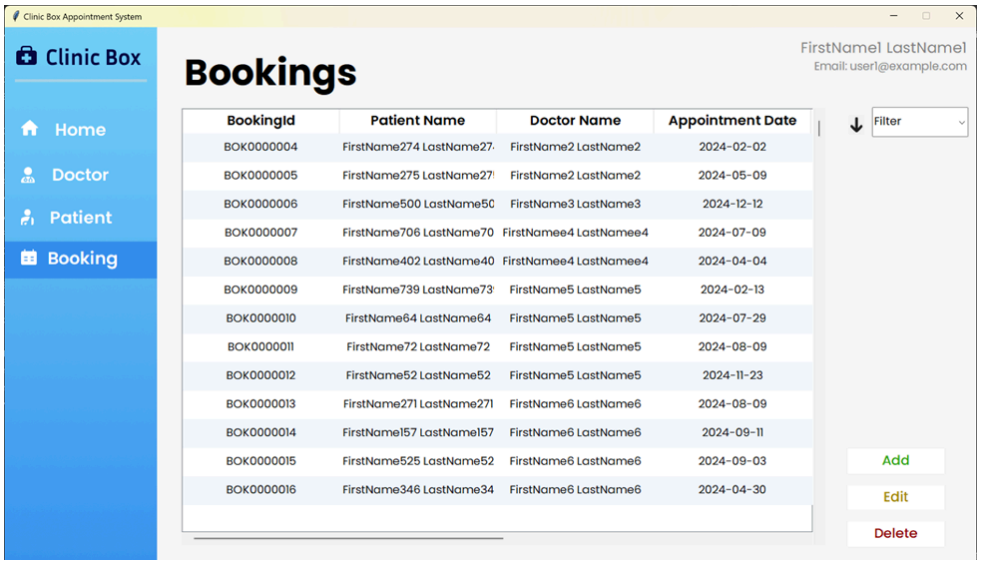


Admins also have access to add, edit, and delete the medical history of each patient and the patients from the database.

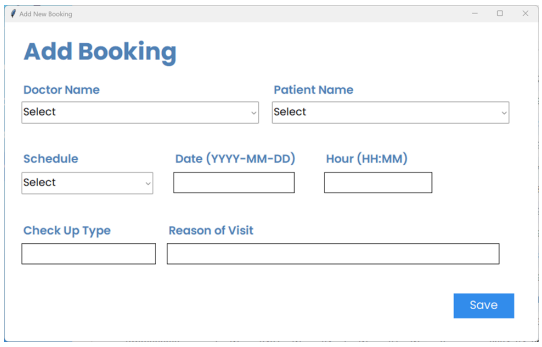


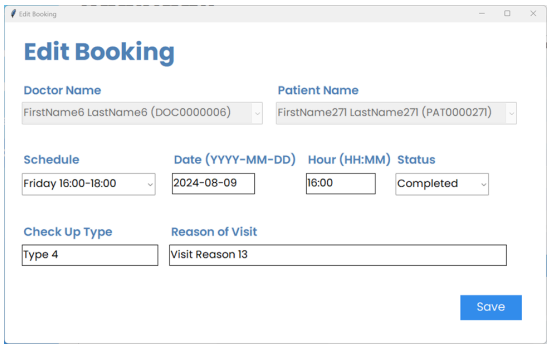


6. Admin Booking Page



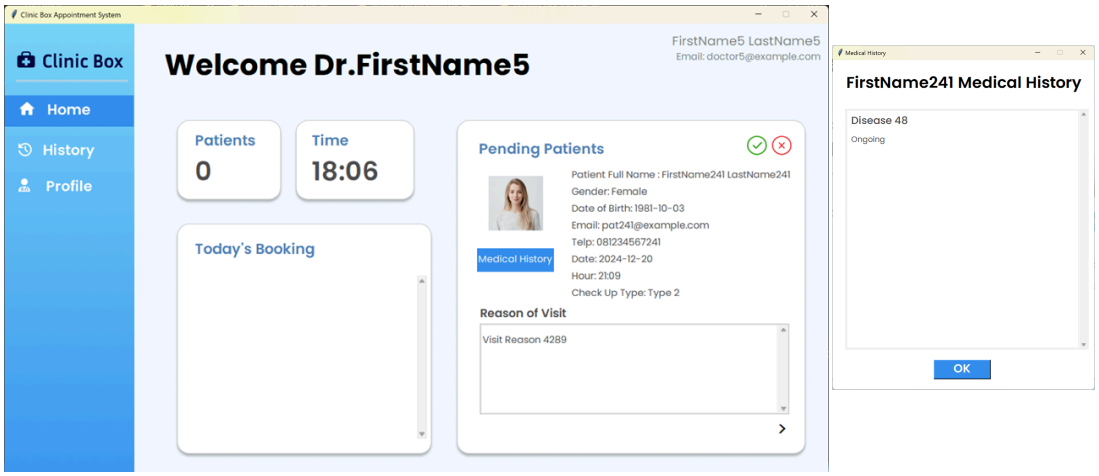
This page is also highly similar to the Admin Doctor Page and Admin Patient Page where it displays the booking data in a tabular format and allows filtering. However, there are no additional attributes like specialty, disease, or others from a booking. Therefore, in this page, the admin can only insert, update, or delete a record from the booking table.





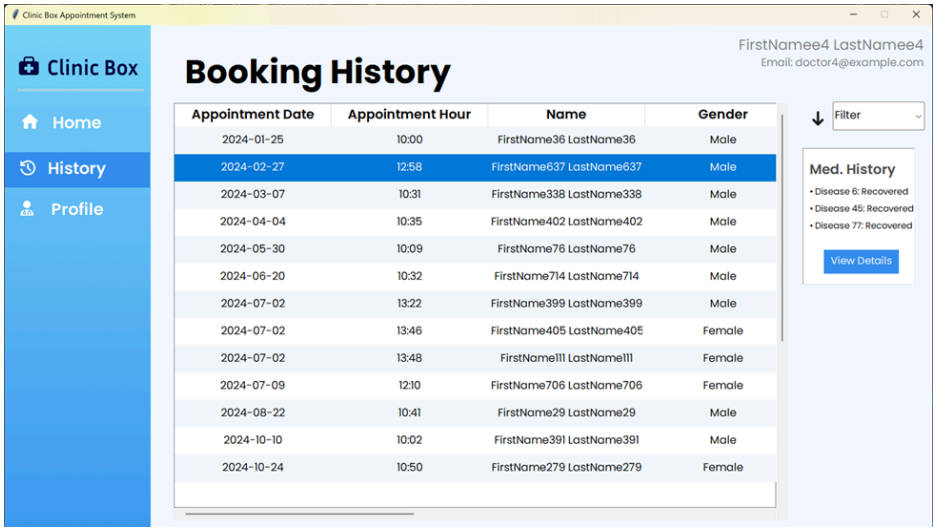
7. Doctor Dashboard (Doctor Home Page)

When a user identified as a doctor logs in to their account, they will first see this home page. Here, the doctor can see how many patients they have today, what are today's booking, the current time, and pending patients (patients that have not met the doctor yet).



The checklist and cross button in the pending patient panel are used to update the status of the booking. If the appointment is done, the doctor can click the check button, while if the doctor wants to cancel the appointment, then the doctor needs to click the cross button. The > button in the bottom right of the pending patient panel is used to navigate to the next pending patient. The doctor can also view the patient's medical history through the button below the patient profile. The window will be like the picture on the side.

8. Doctor History Page

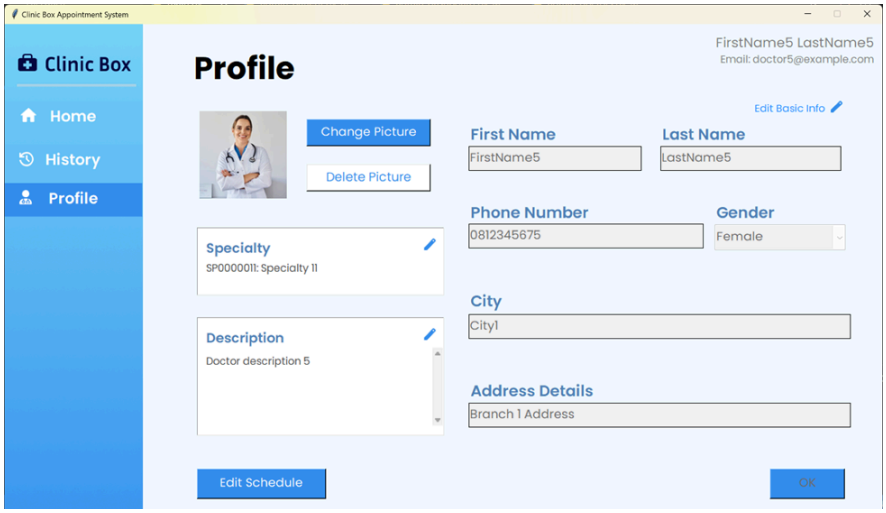


Appointment Date	Appointment Hour	Name	Gender
2024-01-25	10:00	FirstName36 LastName36	Male
2024-02-27	12:58	FirstName637 LastName637	Male
2024-03-07	10:31	FirstName338 LastName338	Male
2024-04-04	10:35	FirstName402 LastName402	Male
2024-05-30	10:09	FirstName76 LastName76	Male
2024-06-20	10:32	FirstName714 LastName714	Male
2024-07-02	13:22	FirstName399 LastName399	Male
2024-07-02	13:46	FirstName405 LastName405	Female
2024-07-02	13:48	FirstName111 LastName111	Female
2024-07-09	12:10	FirstName706 LastName706	Female
2024-08-22	10:41	FirstName29 LastName29	Male
2024-10-10	10:02	FirstName391 LastName391	Male
2024-10-24	10:50	FirstName279 LastName279	Female

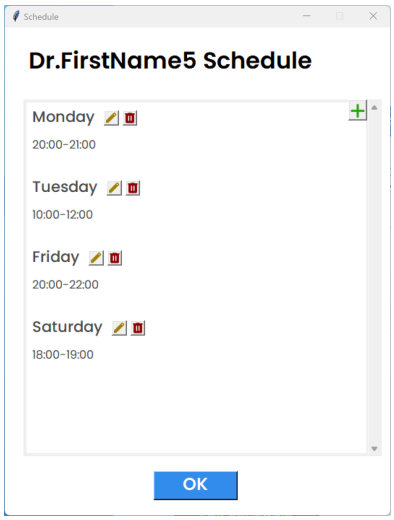
This page displays all appointments the doctor has received regardless of the status, whether they are still pending, already completed, or canceled. The doctor

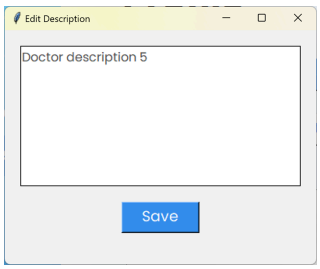
can also view the patient's medical history by clicking the record in the table. If the doctor clicks the View Details button, they will be able to see the complete patient's medical history, just like in the Doctor Home Page. This page is similar to Admin Booking Page and also allows filtering.

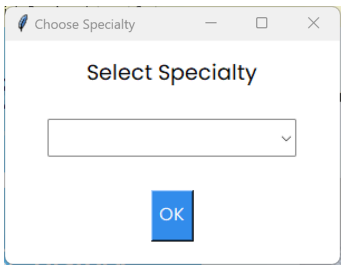
9. Doctor Profile Page



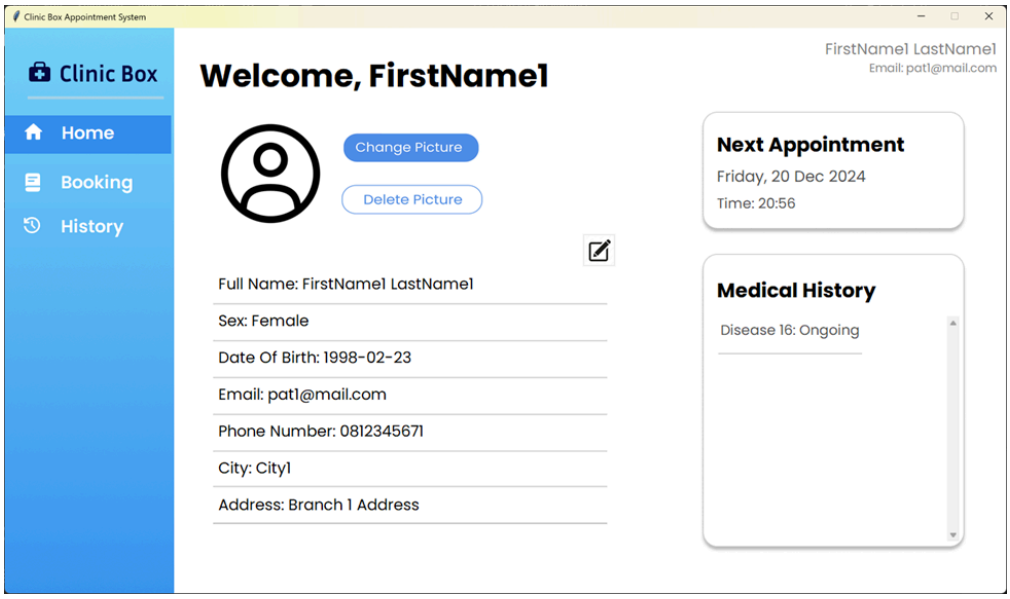
This page consists of the information of the doctor. The doctor can edit their personal information, from name to address details here. The fields are initially disabled, but if the doctor clicks the "Edit Basic Info" button on the top of the fields, then the fields can be edited and the OK button will be enabled. Besides basic information, the doctor can also edit his/her specialty, description, and schedule inside this page by clicking the corresponding buttons.



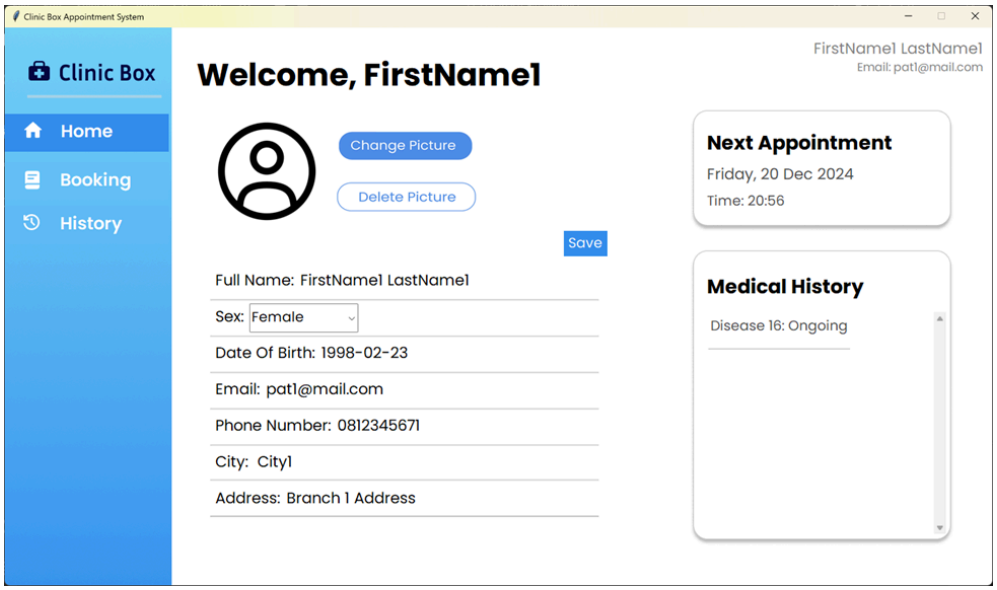




10. Patient Home Page

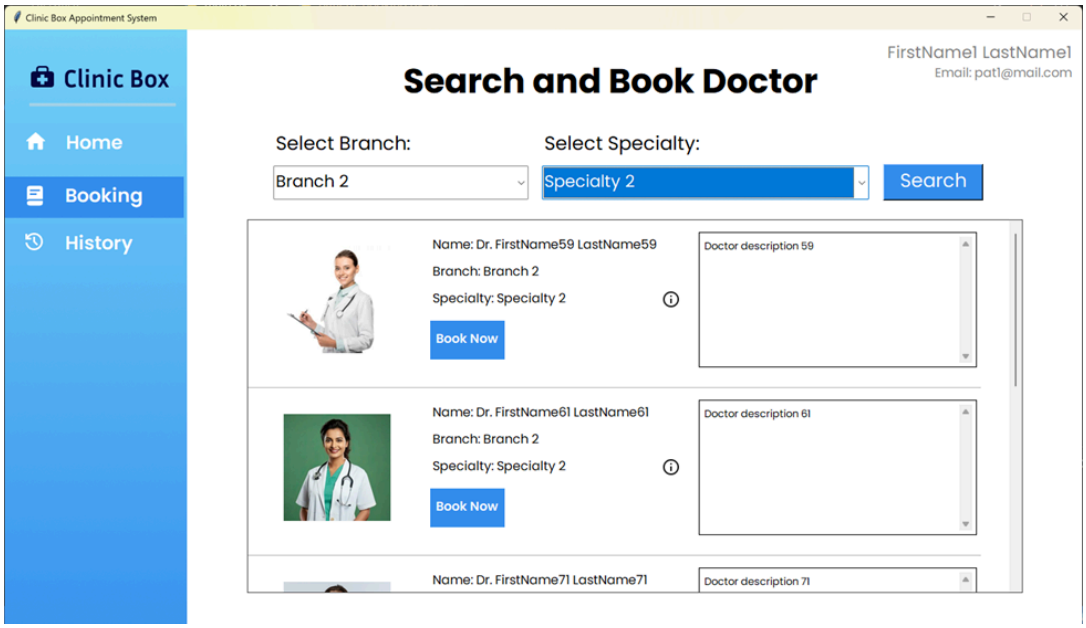


When a user with the role Patient first logs in to their account, they will be redirected to this page. On this page, users can view and edit their personal information. They can also see their medical history and next appointment. To edit their full name until the address, the user needs to click the edit button. After that, the labels become entries that can be typed and the edit button is replaced by the save button as seen below.

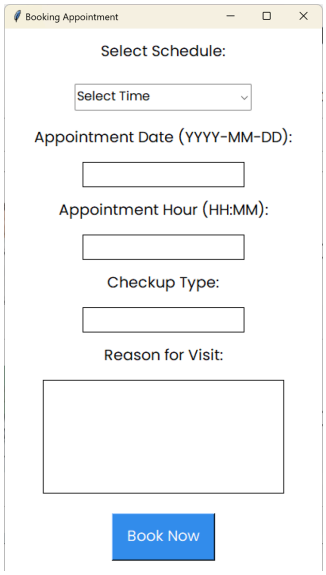


If the user clicks save, then the patient data will be updated in the database.

11. Patient Booking Page

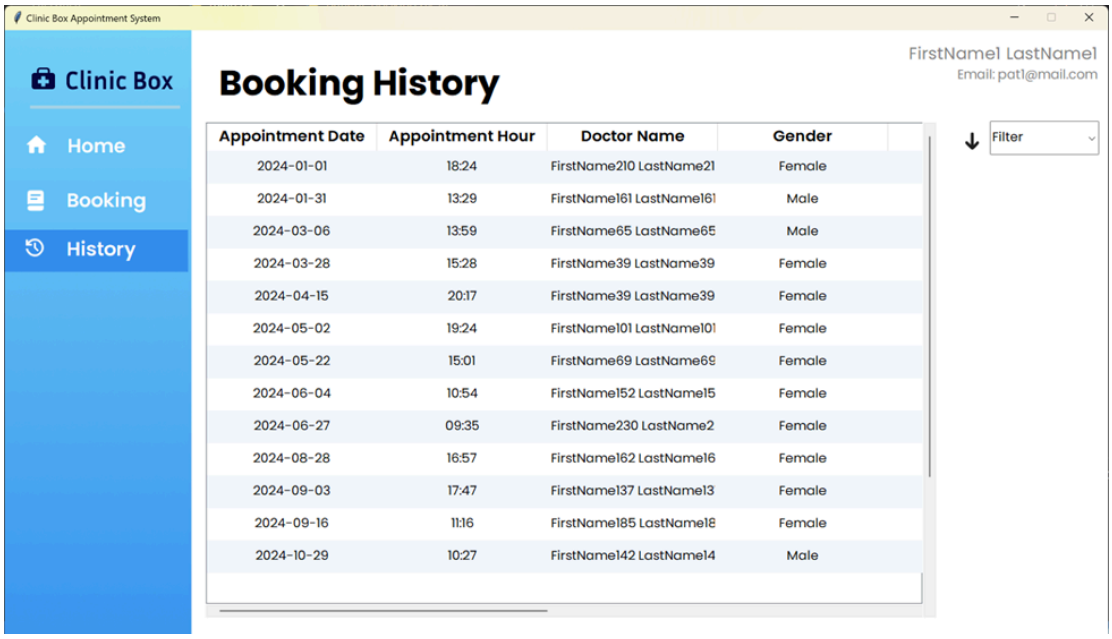


On this page, patients can find doctors based on their branch or specialty and book an appointment with them. The info button beside the doctor description scroll pane is used to display the description of the specialty of the corresponding doctor.



If the patient clicks the Book Now button for a doctor, then the patient needs to fill in the required information for a booking as shown on the picture on the side. The system will also check if the time is valid and if the doctor is available and not busy at that time. A doctor maximum has 4 patients for each practice hour.

12. Patient History Page



This page contains the booking history of this patient with various doctors. This page is highly similar to the Doctor History Page. It also allows the patient to filter the doctor to obtain better readability order of the records.

The code and documentation for the UI can also be seen in the repository at https://github.com/Ella-Raputri/Database-FinalProject.

F. SQL Code

Data Definition Language (DDL)

The Database Schema was defined using the 'create_database_and_tables.sql' script inside the 'sql_queries' folder in our repository. We also utilize a View called BranchBookings to better know the booking of patients and doctors in each branch. Below are the MySQL queries to create the database, tables, and view.

```
CREATE DATABASE ClinicSystemDB;
```

```
USE ClinicSystemDB;
CREATE TABLE ClinicBranch(
   BranchNo          INT                    NOT NULL,
   BranchName        VARCHAR(255)           NOT NULL,
   BranchLocation    VARCHAR(255)           NOT NULL,
   PRIMARY KEY (BranchNo)
);

CREATE TABLE `User`(
   UserId                  VARCHAR(10)          NOT NULL,
   Email                   VARCHAR(255)         NOT NULL,
   `Password`              VARCHAR(100)         NOT NULL,
   FirstName               VARCHAR(100)         NOT NULL,
   LastName                VARCHAR(200)         NOT NULL,
   Gender                  BIT                  NOT NULL,
   PhoneNumber             VARCHAR(50)          NOT NULL,
   RoleName                VARCHAR(50)          NOT NULL,
   City                    VARCHAR(50)          NOT NULL,
   AddressDetail           TEXT                 NOT NULL,
   IsDeleted               BIT                  DEFAULT 0,
   PRIMARY KEY (UserId)
);

CREATE TABLE Specialty(
   SpecialtyId             VARCHAR(10)          NOT NULL,
   SpecialtyName           VARCHAR(255)         NOT NULL,
   SpecialtyDescription    TEXT,
   PRIMARY KEY (SpecialtyId)
);

CREATE TABLE Doctor (
   DoctorId          VARCHAR(10)       NOT NULL,
   SpecialtyId       VARCHAR(10)       NOT NULL,
   ProfilePicture    VARCHAR(500),
   `Description`     TEXT,
   BranchNo          INT               NOT NULL,
   PRIMARY KEY (DoctorId),
   FOREIGN KEY (DoctorId) REFERENCES `User`(UserId) ON DELETE
         CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (SpecialtyId) REFERENCES Specialty(SpecialtyId),
   FOREIGN KEY (BranchNo) REFERENCES ClinicBranch(BranchNo)
);
```

```
CREATE TABLE DoctorSchedule (
   ScheduleId          VARCHAR(10)        NOT NULL,
   DoctorId            VARCHAR(10)        NOT NULL,
   DayOfWeek           VARCHAR(20)        NOT NULL,
   StartHour           TIME               NOT NULL,
   EndHour             TIME               NOT NULL,
   PRIMARY KEY (ScheduleId),
   FOREIGN KEY (DoctorId) REFERENCES Doctor(DoctorId) ON DELETE
        CASCADE ON UPDATE CASCADE
);

CREATE TABLE Patient (
   PatientId           VARCHAR(10)        NOT NULL,
   DateOfBirth         DATE               NOT NULL,
   ProfilePicture      VARCHAR(500),
   PRIMARY KEY (PatientId),
   FOREIGN KEY (PatientId) REFERENCES `User`(UserId) ON DELETE
        CASCADE ON UPDATE CASCADE
);

CREATE TABLE Disease (
   DiseaseId           VARCHAR(10)        NOT NULL,
   DiseaseName         VARCHAR(200)       NOT NULL,
   PRIMARY KEY (DiseaseId)
);

CREATE TABLE MedicalHistory (
   PatientId           VARCHAR(10)              NOT NULL,
   DiseaseId           VARCHAR(10)              NOT NULL,
   `Status`            BIT                      NOT NULL,
   PRIMARY KEY (PatientId, DiseaseId),
   FOREIGN KEY (PatientId) REFERENCES Patient(PatientId) ON DELETE
        CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (DiseaseId) REFERENCES Disease(DiseaseId)
);

CREATE TABLE `Admin` (
   AdminId             VARCHAR(10)        NOT NULL,
   BranchNo             INT               NOT NULL,
   PRIMARY KEY (AdminId),
   FOREIGN KEY (AdminId) REFERENCES `User`(UserId) ON DELETE
        CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (BranchNo) REFERENCES ClinicBranch(BranchNo)
```

```
);
CREATE TABLE Booking (
   BookingId              VARCHAR(10)      NOT NULL,
   PatientId              VARCHAR(10)      NOT NULL,
   DoctorId               VARCHAR(10)      NOT NULL,
   AppointmentDate        DATE             NOT NULL,
   AppointmentHour        TIME             NOT NULL,
   AppointmentStatus      VARCHAR(20)      NOT NULL,
   CheckUpType            VARCHAR(200)     NOT NULL,
   ReasonOfVisit          TEXT             NOT NULL,
   PRIMARY KEY (BookingId),
   FOREIGN KEY (PatientId) REFERENCES Patient(PatientId),
   FOREIGN KEY (DoctorId) REFERENCES Doctor(DoctorId)
);

CREATE VIEW BranchBookings AS
SELECT
   b.BookingId,
   b.PatientId,
   CONCAT(pu.FirstName, ' ', pu.LastName) AS PatientName,
   b.DoctorId,
   CONCAT(du.FirstName, ' ', du.LastName) AS DoctorName,
   b.AppointmentDate,
   b.AppointmentHour,
   b.AppointmentStatus,
   b.CheckUpType,
   b.ReasonOfVisit,
   d.BranchNo
FROM
   Booking b
INNER JOIN
   Doctor d ON b.DoctorId = d.DoctorId
INNER JOIN
   `User` du ON d.DoctorId = du.UserId
INNER JOIN
   `User` pu ON b.PatientId = pu.UserId
WHERE
   pu.IsDeleted = 0 AND du.IsDeleted = 0;
```

Data Manipulation Language (DML)

1. **Dummy Data Insertion**

The ‘insert_dummy.sql’ inside ‘sql_queries’ folder is used to automate the

insertion of sample data into the database using the Stored Procedure. Below are the SQL queries for inserting dummy data.

```
USE ClinicSystemDB;

SET @total_branches = 5;
SET @users_per_branch = 205;
SET @total_specialties = 20;
SET @total_bookings = 1000;
SET @patients_per_branch = 150;
SET @doctors_per_branch = 50;
SET @admins_per_branch = 5;

-- Insert 5 branches
DELIMITER $$
CREATE PROCEDURE InsertDummyBranches()
BEGIN
   DECLARE i INT DEFAULT 1;
   WHILE i <= @total_branches DO
      INSERT INTO ClinicBranch (BranchNo, BranchName, BranchLocation)
      VALUES (i, CONCAT('Branch ', i), CONCAT('Location ', i));
      SET i = i + 1;
   END WHILE;
END$$
DELIMITER ;
CALL InsertDummyBranches();


-- Insert 20 specialties
DELIMITER $$
CREATE PROCEDURE InsertDummySpecialties()
BEGIN
   DECLARE i INT DEFAULT 1;
   WHILE i <= 20 DO
      INSERT INTO Specialty (SpecialtyId, SpecialtyName,
SpecialtyDescription)
      VALUES (CONCAT('SP', LPAD(i, 7, '0')), CONCAT('Specialty ', i),
CONCAT('Description of Specialty ', i));
      SET i = i + 1;
   END WHILE;
END$$
DELIMITER ;
CALL InsertDummySpecialties();
```

```
-- Insert 205 users per branch
DELIMITER $$
CREATE PROCEDURE InsertDummyUsers()
BEGIN
   DECLARE branch_id INT DEFAULT 1;
   DECLARE patient_id INT DEFAULT 1;
   DECLARE doctor_id INT DEFAULT 1;
   DECLARE admin_id INT DEFAULT 1;
   WHILE branch_id <= @total_branches DO
      -- Insert patients
      WHILE patient_id <= branch_id * @patients_per_branch DO
         INSERT INTO `User` (UserId, Email, `Password`, FirstName,
LastName, Gender, PhoneNumber, RoleName, City, AddressDetail)
         VALUES (CONCAT('PAT', LPAD(patient_id, 7, '0')), CONCAT('user',
patient_id, '@example.com'), 'password',
                 CONCAT('FirstName', patient_id), CONCAT('LastName',
patient_id), CASE WHEN RAND() < 0.5 THEN 1 ELSE 0 END,
                 CONCAT('081234567', patient_id), 'Patient', CONCAT('City',
branch_id),
                 CONCAT('Branch ', branch_id, ' Address'));

                        INSERT INTO Patient (PatientId, DateOfBirth,
ProfilePicture)
         VALUES (CONCAT('PAT', LPAD(patient_id, 7, '0')),
                 DATE_ADD('1980-01-01', INTERVAL FLOOR(RAND() * 14610)
DAY),
                 NULL);
         SET patient_id = patient_id + 1;
      END WHILE;

      -- Insert doctors
      WHILE doctor_id <= branch_id * @doctors_per_branch DO
         INSERT INTO `User` (UserId, Email, `Password`, FirstName,
LastName, Gender, PhoneNumber, RoleName, City, AddressDetail)
         VALUES (CONCAT('DOC', LPAD(doctor_id, 7, '0')),
CONCAT('doctor', doctor_id, '@example.com'), 'password',
                 CONCAT('FirstName', doctor_id), CONCAT('LastName',
doctor_id), CASE WHEN RAND() < 0.5 THEN 1 ELSE 0 END,
                 CONCAT('081234567', doctor_id), 'Doctor', CONCAT('City',
branch_id),
                 CONCAT('Branch ', branch_id, ' Address'));
         INSERT INTO Doctor (DoctorId, SpecialtyID, ProfilePicture,
`Description`, BranchNo)
         VALUES (CONCAT('DOC', LPAD(doctor_id, 7, '0')),
```

```
            CONCAT('SP', LPAD(FLOOR(1 + RAND() * 20), 7, '0')),
            NULL, CONCAT('Doctor description ', doctor_id), branch_id);
        SET doctor_id = doctor_id + 1;
    END WHILE;

    -- Insert admins
    WHILE admin_id <= branch_id * @admins_per_branch DO
        INSERT INTO `User` (UserId, Email, `Password`, FirstName,
LastName, Gender, PhoneNumber, RoleName, City, AddressDetail)
        VALUES (CONCAT('ADM', LPAD(admin_id, 7, '0')),
CONCAT('patient', admin_id, '@example.com'), 'password',
            CONCAT('FirstName', admin_id), CONCAT('LastName',
admin_id), CASE WHEN RAND() < 0.5 THEN 1 ELSE 0 END,
            CONCAT('081234567', admin_id), 'Admin', CONCAT('City',
branch_id),
            CONCAT('Branch ', branch_id, ' Address'));
        INSERT INTO `Admin` (AdminId, BranchNo)
        VALUES (CONCAT('ADM', LPAD(admin_id, 7, '0')), branch_id);
        SET admin_id = admin_id + 1;
    END WHILE;

    SET branch_id = branch_id + 1;
  END WHILE;
END$$
DELIMITER ;
CALL InsertDummyUsers();


-- Insert 100 diseases
DELIMITER $$
CREATE PROCEDURE InsertDummyDiseases()
BEGIN
  DECLARE i INT DEFAULT 1;
  WHILE i <= 100 DO
    INSERT INTO Disease (DiseaseId, DiseaseName)
    VALUES (CONCAT('DIS', LPAD(i, 7, '0')), CONCAT('Disease ', i));
    SET i = i + 1;
  END WHILE;
END$$
DELIMITER ;
CALL InsertDummyDiseases();
```

```
-- Insert Medical History
DELIMITER $$
CREATE PROCEDURE InsertDummyMedicalHistory()
BEGIN
   DECLARE patient_id INT DEFAULT 1;
   DECLARE history_count INT;
   DECLARE disease_id VARCHAR(10);

   CREATE TEMPORARY TABLE TempDiseases AS
   SELECT DiseaseId FROM Disease;
   IF (SELECT COUNT(*) FROM TempDiseases) = 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No diseases
found in the Disease table.';
   END IF;

   SELECT 'Starting insertion process' AS DebugInfo;
   WHILE patient_id <= 5 * 150 DO
      SET history_count = FLOOR(1 + RAND() * 5);
      SELECT CONCAT('Processing PatientId: PAT', LPAD(patient_id, 7, '0'))
AS DebugInfo;

      WHILE history_count > 0 DO
         -- Fetch a random DiseaseId
         SELECT DiseaseId
         INTO disease_id
         FROM TempDiseases
         ORDER BY RAND()
         LIMIT 1;

         SELECT CONCAT('Selected DiseaseId: ', disease_id) AS DebugInfo;
         IF NOT EXISTS (
            SELECT 1
            FROM MedicalHistory
            WHERE PatientId = CONCAT('PAT', LPAD(patient_id, 7, '0'))
              AND DiseaseId = disease_id
         ) THEN
            -- Insert into MedicalHistory
            INSERT INTO MedicalHistory (PatientId, DiseaseId, `Status`)
            VALUES (
               CONCAT('PAT', LPAD(patient_id, 7, '0')),
               disease_id,
               RAND() < 0.5
            );
```

```
            SELECT CONCAT('Inserted: PatientId PAT', LPAD(patient_id, 7,
'0'), ' DiseaseId: ', disease_id) AS DebugInfo;

            SET history_count = history_count - 1;
          ELSE
            SELECT CONCAT('Duplicate for PatientId PAT', LPAD(patient_id,
7, '0'), ' DiseaseId: ', disease_id) AS DebugInfo;
          END IF;
       END WHILE;

       SET patient_id = patient_id + 1;
   END WHILE;
   DROP TEMPORARY TABLE TempDiseases;
END$$

DELIMITER ;
CALL InsertDummyMedicalHistory();


-- Insert Doctor Schedules (1-5 per doctor)
DELIMITER $$
CREATE PROCEDURE InsertDummyDoctorSchedules()
BEGIN
   DECLARE i INT DEFAULT 1;
   DECLARE schedule_count INT;
   DECLARE start_hour INT;
   DECLARE end_hour INT;
   DECLARE day_of_week VARCHAR(10);
   DECLARE schedule_id INT DEFAULT 1;

   WHILE i <= @total_branches * @doctors_per_branch DO
      SET schedule_count = FLOOR(1 + RAND() * 5);

      WHILE schedule_count > 0 DO
         SET start_hour = FLOOR(9 + RAND() * 12);
         SET end_hour = start_hour + FLOOR(1 + RAND() * 4);

         SET day_of_week = CASE
                        WHEN RAND() < 0.14 THEN 'Monday'
                        WHEN RAND() < 0.28 THEN 'Tuesday'
                        WHEN RAND() < 0.42 THEN 'Wednesday'
                        WHEN RAND() < 0.57 THEN 'Thursday'
                        WHEN RAND() < 0.71 THEN 'Friday'
                        WHEN RAND() < 0.85 THEN 'Saturday'
```

```
                        ELSE 'Sunday'
                      END;

        INSERT INTO DoctorSchedule (ScheduleId, DoctorId, DayOfWeek,
StartHour, EndHour)
        VALUES (
           CONCAT('SCH', LPAD(schedule_id, 7, '0')),
           CONCAT('DOC', LPAD(i, 7, '0')),
           day_of_week,
           CONCAT(LPAD(start_hour, 2, '0'), ':00:00'),
           CONCAT(LPAD(end_hour, 2, '0'), ':00:00')
        );

        SET schedule_id = schedule_id + 1;
        SET schedule_count = schedule_count - 1;
      END WHILE;

      SET i = i + 1;
   END WHILE;
END$$
DELIMITER ;
CALL InsertDummyDoctorSchedules();


-- Insert 7000 bookings
DELIMITER $$
CREATE PROCEDURE InsertDummyBookings(
   IN total_bookings INT
)
BEGIN
   DECLARE i INT DEFAULT 1;
   DECLARE doctor_id VARCHAR(10);
   DECLARE schedule_id VARCHAR(10);
   DECLARE day_of_week VARCHAR(20);
   DECLARE start_hour TIME;
   DECLARE end_hour TIME;
   DECLARE appointment_date DATE;
   DECLARE appointment_hour TIME;
   DECLARE random_patient_id VARCHAR(10);

   DECLARE schedule_cursor CURSOR FOR
      SELECT DoctorId, ScheduleId, DayOfWeek, StartHour, EndHour
      FROM DoctorSchedule;
```

```
   DECLARE CONTINUE HANDLER FOR NOT FOUND SET schedule_id =
NULL;

   OPEN schedule_cursor;
   WHILE i <= total_bookings DO
      FETCH schedule_cursor INTO doctor_id, schedule_id, day_of_week,
start_hour, end_hour;

      IF schedule_id IS NULL THEN
         CLOSE schedule_cursor;
         OPEN schedule_cursor;
         FETCH schedule_cursor INTO doctor_id, schedule_id, day_of_week,
start_hour, end_hour;
      END IF;

      SET appointment_date = DATE_ADD('2024-01-01', INTERVAL
FLOOR(RAND() * 365) DAY);
      WHILE DAYNAME(appointment_date) != day_of_week DO
         SET appointment_date = DATE_ADD(appointment_date, INTERVAL 1
DAY);
      END WHILE;

      SET appointment_hour = ADDTIME(start_hour,
SEC_TO_TIME(FLOOR(RAND() * TIME_TO_SEC(TIMEDIFF(end_hour,
start_hour)))));

      SELECT PatientId INTO random_patient_id
      FROM Patient
      ORDER BY RAND()
      LIMIT 1;

      INSERT INTO Booking (
         BookingId, PatientId, DoctorId, AppointmentDate, AppointmentHour,
AppointmentStatus, CheckUpType, ReasonOfVisit
      )
      VALUES (
         CONCAT('BOK', LPAD(i, 7, '0')),
         random_patient_id,
         doctor_id,
         appointment_date,
         appointment_hour,
         CASE WHEN RAND() < 0.5 THEN 'Completed' ELSE 'Pending' END,
         CONCAT('Type ', FLOOR(1 + RAND() * 5)),
         CONCAT('Visit Reason ', i)
```

```
        );

        SET i = i + 1;
    END WHILE;
    CLOSE schedule_cursor;
END$$
DELIMITER ;

CALL InsertDummyBookings(7000);
```

2. In Application DML Samples

The queries for this are spread across all Python files inside the 'code' folder. We will give some samples of the DML used on each page and the snippets of the result in the MySQL Workbench. The %s is used to concatenate the parameter value to the query. Here, we will only display some of the DML (not all of them) because the project only requires some samples, and many of them are similar.

a. Login Page

```
SELECT UserId, RoleName FROM User WHERE Email = %s AND
Password = %s AND IsDeleted = 0;
```

Example: parameters = ('user1@example.com','password')

Result:

	UserId	RoleName
▸	ADM0000001	Admin

This query is used to see whether a particular user with the email and password exists in the database.

b. Register Page

```
SELECT PatientID FROM Patient ORDER BY PatientID DESC LIMIT 1;
```

Result:

	PatientID
▸	PAT0000754

This query will output the last patient ID in the table. If the result is incremented by one, then it can be our new ID.

```
SELECT COUNT(*) FROM `User` WHERE Email = %s AND UserId != %s
```

Example: parameter=('pat4@mail.com', 'PAT0000004')

Result:

COUNT(*)
0

This query will get the number of users using an email and is not the current user. This query is used to check whether the email is unique or not.

```
INSERT INTO User (UserId, Email, Password, FirstName, LastName,
Gender, PhoneNumber, RoleName, City, AddressDetail, IsDeleted)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s);
```

This query inserts the user's basic information into the User table.

```
INSERT INTO Patient (PatientID, DateOfBirth, ProfilePicture)
VALUES (%s, %s, %s);
```

This query inserts patient-specific information into the Patient table.

c. Admin Dashboard

```
SELECT Email, FirstName, LastName FROM User WHERE UserId = %s;
```

Example: parameters = ('ADM0000001')

Result:

	Email	FirstName	LastName
▶	user1@example.com	FirstName1	LastName1

This query is used to get the email and name of the user. All of the pages in the admin, doctor, and patient section use this query to get the value for the name and email label on the top right of the window.

```
SELECT COUNT(DoctorId) FROM Doctor WHERE BranchNo = %s;
```

Example: parameters = (1)

Result:

	COUNT(DoctorId)
▶	52

This query is used to count the number of doctors inside a branch with a certain branch number.

```
SELECT
        DATE_FORMAT(AppointmentDate, '%Y-%m') AS BookingMonth,
        COUNT(*) AS TotalBookings
FROM BranchBookings
WHERE BranchNo = %s
GROUP BY BookingMonth
ORDER BY BookingMonth;
```

Example: parameters = (1)

Result:

	BookingMonth	TotalBookings
▶	2024-01	108
	2024-02	83
	2024-03	112
	2024-04	116
	2024-05	117
	2024-06	101

This query is used to count the number of monthly bookings inside a branch. This data will be further used to create an area chart in the dashboard.

```
SELECT DISTINCT
        SUBSTRING_INDEX(b.DoctorName, ' ', 1) AS DoctorFirstName,
        DATE_FORMAT(ds.StartHour, '%H:%i') AS StartTime,
        DATE_FORMAT(ds.EndHour, '%H:%i') AS EndTime
FROM DoctorSchedule ds
INNER JOIN BranchBookings b ON ds.DoctorId = b.DoctorId
WHERE ds.DayOfWeek = %s AND b.BranchNo = %s
ORDER BY StartTime, EndTime ASC;
```

Example: parameters = (‘Monday’, 1)

Result:

DoctorFirstName	StartTime	EndTime
FirstName29	09:00	10:00
FirstName26	09:00	11:00
FirstName45	09:00	12:00
FirstName15	10:00	11:00
FirstName47	10:00	12:00

This query is used to get the doctor who has a practice schedule today and their practice time (start and end time). One doctor can have more than one schedule in a day.

d. Admin Doctor Page

```
SELECT SpecialtyId, SpecialtyName FROM Specialty;
```

Result:

SpecialtyId	SpecialtyName
SP0000001	Specialty 1
SP0000002	Specialty 2
SP0000003	Specialty 3
SP0000004	Specialty 4

This query is used to get all the specialties' IDs and names.

```
SELECT
        u.Email, u.Password, u.FirstName, u.LastName, u.Gender,
        u.PhoneNumber, u.City, u.AddressDetail,
        d.SpecialtyId, d.ProfilePicture, d.Description
FROM User u JOIN Doctor d ON u.UserId = d.DoctorId
WHERE u.UserId = %s;
```

Example: parameters = ('DOC0000004')

Result:

Email	Password	FirstName	LastName	Gender	PhoneNumber	City	AddressDetail	SpecialtyId	ProfilePicture	Description
doctor4@example.com	password	FirstNamee4	LastNamee4	0	08123456744	Cityy1	Branchh 1 Address	SP0000006	NULL	Doctor description

This query is used to get basic and doctor-specific information about a doctor.

```
UPDATE User
SET Email = %s, Password = %s, FirstName = %s, LastName = %s,
        Gender = %s, PhoneNumber = %s, City = %s, AddressDetail = %s
WHERE UserId = %s
```

This query is used to update basic information of a doctor.

```
UPDATE `User` SET IsDeleted = 1 WHERE UserId = %s;
```

This query is used to soft delete the user by updating the IsDeleted column to be true.

```
UPDATE Booking SET AppointmentStatus = 'Cancelled' WHERE
DoctorId = %s AND AppointmentStatus = 'Pending';
```

This query is used to update all the pending appointment status of a doctor to become canceled because the doctor is deleted.

```
SELECT
        d.DoctorId,
        s.SpecialtyName AS Specialty,
        u.Email,
        CONCAT(u.FirstName, ' ', u.LastName) AS Name,
        CASE u.Gender
                WHEN 0 THEN 'Male'
                ELSE 'Female'
        END AS Gender,
        u.PhoneNumber,
        u.City,
        u.AddressDetail AS Address,
        COUNT(b.BookingId) AS TotalBookings,
        SUM(CASE
                WHEN MONTH(b.AppointmentDate) =
MONTH(CURRENT_DATE)
                        AND YEAR(b.AppointmentDate) =
YEAR(CURRENT_DATE)
                THEN 1
                ELSE 0
                END
        ) AS ThisMonthsBookings
FROM
        Doctor d
LEFT JOIN
        Specialty s ON d.SpecialtyID = s.SpecialtyID
LEFT JOIN
        `User` u ON d.DoctorId = u.UserId
LEFT JOIN
        Booking b ON d.DoctorId = b.DoctorId
```

```
WHERE d.BranchNo = %s AND u.IsDeleted = 0
GROUP BY
        d.DoctorId, s.SpecialtyName, u.Email, u.FirstName, u.LastName,
        u.Gender, u.PhoneNumber, u.City, u.AddressDetail
ORDER BY
        {self.filter_field} {self.sort_order};
```

Example: parameters=(1), self.filter_field = d.DoctorId, self.sort_order = ASC

Result:

DoctorId	Specialty	Email	Name	Gender	PhoneNumber	City	Address	TotalBookings	ThisMonthsBo
DOC0000002	Specialty 6	doctor2@example.com	FirstName2 LastName2	Male	0812345672	City1	Branch 1 Address	21	2
DOC0000003	Specialty 13	doctor3@example.com	FirstName3 LastName3	Male	0812345673	City1	Branch 1 Address	10	1
DOC0000004	Specialty 6	doctor4@example.com	FirstNamee4 LastNamee4	Male	08123456744	Cityy1	Branchh 1 Address	23	5

This query is used to get all the information of a doctor and their bookings in total and this month in a specific branch.

```
SELECT
        CONCAT(LEFT(DayOfWeek, 3), ': ',
        TIME_FORMAT(StartHour, '%H:%i'), '-',
        TIME_FORMAT(EndHour, '%H:%i')) AS Schedule,
        ScheduleId
FROM DoctorSchedule
WHERE DoctorId = %s
ORDER BY FIELD(DayOfWeek, 'Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday', 'Sunday'),
StartHour, EndHour;
```

Example: parameters = ('DOC0000004')

Result:

Schedule	ScheduleId
Tue: 12:00-14:00	SCH0000007
Thu: 10:00-11:00	SCH0000008

This query is used to get all the schedules of a doctor sorted by day's name, start hour, and end hour.

```
INSERT INTO Specialty (SpecialtyId, SpecialtyName,
SpecialtyDescription) VALUES (%s, %s, %s);
```

This query is used to insert a new type of specialty into the table.

```
UPDATE Specialty
SET SpecialtyName = %s, SpecialtyDescription = %s
WHERE SpecialtyID = %s;
```

This query is used to update the specialty's name and description.

```
SELECT COUNT(DoctorId) FROM Doctor WHERE SpecialtyId = %s
```

Example: parameters = ('SP0000004')

Result:

COUNT(DoctorId)
5

This query is used to get the number of doctors specializing in a specialty.

```
DELETE FROM Specialty WHERE SpecialtyId = %s;
```

This query is used to delete a specialty.

```
INSERT INTO DoctorSchedule (ScheduleId, DoctorId, DayOfWeek,
StartHour, EndHour) VALUES (%s, %s, %s, %s, %s);
```

This query is used to insert a new schedule of a doctor into the table.

```
UPDATE DoctorSchedule
SET DayOfWeek = %s, StartHour = %s, EndHour = %s
WHERE ScheduleId = %s
```

This query is used to update the schedule of a doctor.

```
DELETE FROM DoctorSchedule WHERE ScheduleId = %s
```

This query is used to delete a schedule.

e. Admin Patient Page

```
SELECT DiseaseId, DiseaseName FROM Disease;
```

Result:

DiseaseId	DiseaseName
DIS0000001	Disease 1
DIS0000002	Disease 2
DIS0000003	Disease 3
DIS0000004	Disease 4

This query is used to get all the diseases' IDs and names.

```
SELECT
        u.Email, u.Password, u.FirstName, u.LastName,
        u.Gender, u.PhoneNumber, u.City, u.AddressDetail,
        p.DateOfBirth, p.ProfilePicture
FROM User u JOIN Patient p ON u.UserId = p.PatientId
WHERE u.UserId = %s
```

Example: parameters = ('PAT0000001')

Result:

Email	Password	FirstName	LastName	Gender	PhoneNumber	City	AddressDetail	DateOfBirth	ProfilePicture
pat1@mail.com	password	FirstName1	LastName1	1	0812345671	City1	Branch 1 Address	1998-02-23	NULL

This query is used to get basic and patient-specific information about a patient.

```
SELECT
        p.PatientId,
        p.DateOfBirth,
        u.Email,
        CONCAT(u.FirstName, ' ', u.LastName) AS Name,
        CASE u.Gender
                WHEN 0 THEN 'Male'
                ELSE 'Female'
        END AS Gender,
        u.PhoneNumber,
        u.City,
        u.AddressDetail AS Address,
        COUNT(b.BookingId) AS TotalBookings
FROM
        Patient p
LEFT JOIN
        `User` u ON p.PatientId = u.UserId
LEFT JOIN
        Booking b ON p.PatientId = b.PatientId
```

```
WHERE
        u.IsDeleted = 0
GROUP BY
        p.PatientId, u.Email, u.FirstName, u.LastName,
        u.Gender, u.PhoneNumber, u.City, u.AddressDetail
ORDER BY
        {self.filter_field} {self.sort_order};
```

Example: self.filter_field = p.PatientId, self.sort_order = ASC

Result:

PatientId	DateOfBirth	Email	Name	Gender	PhoneNumber	City	Address	TotalBookings
PAT0000001	1998-02-23	pat1@mail.com	FirstName1 LastName1	Female	0812345671	City1	Branch 1 Address	18
PAT0000002	1985-06-13	pat2@example.com	FirstName2 LastName2	Male	0812345672	City1	Branch 1 Address	10
PAT0000003	2000-06-17	pat3@example.com	FirstName3 LastName3	Male	0812345673	City1	Branch 1 Address	14
PAT0000004	1999-09-17	pat4@example.com	FirstName4 LastName4	Male	0812345674	City1	Branch 1 Address	14

This query is used to get all the information of a patient and their total bookings.

```
SELECT
CONCAT(d.DiseaseName, ': ',
                CASE m.Status
                        WHEN 0 THEN 'Ongoing'
                        ELSE 'Recovered'
                END
        ) AS MedHistory
FROM MedicalHistory m JOIN Disease d
ON m.DiseaseId = d.DiseaseId
WHERE m.PatientId = %s
ORDER BY m.Status ASC;
```

Example: parameters = ('PAT0000004')

Result:

MedHistory
Disease 36: Ongoing
Disease 54: Ongoing
Disease 90: Recovered

This query is used to get all the medical history of a patient.

```
INSERT INTO Disease (DiseaseId, DiseaseName) VALUES (%s, %s);
```

This query is used to insert a new type of disease into the table.

```
UPDATE Disease SET DiseaseName = %s WHERE DiseaseID = %s;
```

This query is used to update the disease's name.

```
SELECT COUNT(PatientId) FROM MedicalHistory WHERE DiseaseId =
%s;
```

Example: parameters = ('DIS0000001')

Result:

COUNT(PatientId)
28

This query is used to get the number of patients with a certain disease.

```
SELECT COUNT(*)
FROM MedicalHistory
WHERE PatientId = %s  AND DiseaseId = (
SELECT DiseaseId FROM Disease WHERE DiseaseName = %s
)
```

Example: parameters = ('PAT0000004', 'Disease 36')

Result:

COUNT(*)
1

This query is used to get the number of medical history in a patient that has that disease. Furthermore, this query is used to check whether the PatientId and DiseaseId is unique or not.

```
DELETE FROM Disease WHERE DiseaseId = %s
```

This query is used to delete a disease type.

```
INSERT INTO MedicalHistory (PatientId, DiseaseId, Status)
VALUES (%s, (SELECT DiseaseId FROM Disease WHERE DiseaseName
= %s), %s)
```

This query is used to insert a new medical history of a patient into the table.

```
UPDATE MedicalHistory
SET DiseaseId = (SELECT DiseaseId FROM Disease WHERE
DiseaseName = %s), Status = %s
WHERE PatientId = %s AND DiseaseId = (SELECT DiseaseId FROM
Disease WHERE DiseaseName = %s)
```

This query is used to update the medical history of a patient.

```
DELETE FROM MedicalHistory WHERE PatientId = %s AND DiseaseId
= (SELECT DiseaseId FROM Disease WHERE DiseaseName = %s)
```

This query is used to delete a medical history of a patient.

f. Admin Booking Page

```
SELECT
    BookingId, PatientName, DoctorName, AppointmentDate,
    AppointmentHour, AppointmentStatus, CheckUpType, ReasonOfVisit
FROM BranchBookings WHERE BranchNo = %s
ORDER BY {self.filter_field} {self.sort_order};
```

Example: parameter=(1), self.filter_field = AppointmentStatus, self.sort_order = DESC

Result:

BookingId	PatientName	DoctorName	AppointmentDate	AppointmentHour	AppointmentStatus	CheckUpType	ReasonOfVisit
BOK0005021	FirstName246 LastName246	FirstName11 LastName11	2024-12-12	22:22:48	Pending	Type 1	Visit Reason 5021
BOK0000741	FirstName249 LastName249	FirstName11 LastName11	2024-12-27	20:51:12	Pending	Type 2	Visit Reason 741
BOK0005022	FirstName330 LastName330	FirstName11 LastName11	2024-12-11	19:55:06	Pending	Type 3	Visit Reason 5022

This query is used to get all the booking information in a certain branch.

```
SELECT
        d.DoctorId AS Id,
        CONCAT(u.FirstName, ' ', u.LastName, ' (', d.DoctorId, ')') AS Name
FROM Doctor d
INNER JOIN User u ON u.UserId = d.DoctorId
WHERE d.BranchNo = %s AND u.IsDeleted = 0
ORDER BY Id ASC;
```

Example: parameter=(1)

Result:

Id	Name
DOC0000002	FirstName2 LastName2 (DOC0000002)
DOC0000003	FirstName3 LastName3 (DOC0000003)
DOC0000004	FirstNamee4 LastNamee4 (DOC0000004)

This query is used to get all the doctors' IDs and names in a certain branch.

```
SELECT COUNT(BookingId) FROM Booking
WHERE DoctorId = %s AND AppointmentDate = %s
AND AppointmentHour BETWEEN  %s AND %s;
```

Example: parameter=('DOC0000004', '2024-12-17', '12:00', '14:00')

Result:

COUNT(BookingId)
2

This query is used to get the number of bookings of a doctor between certain hours.

```
INSERT INTO Booking (BookingId, PatientId, DoctorId,
AppointmentDate, AppointmentHour, AppointmentStatus, CheckUpType,
ReasonOfVisit)  VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

This query is used to insert a new booking into the table.

```
UPDATE Booking
SET AppointmentDate = %s, AppointmentHour = %s, AppointmentStatus
= %s, CheckUpType = %s, ReasonOfVisit = %s
WHERE BookingId = %s
```

This query is used to update the booking information.

```
DELETE FROM Booking WHERE BookingId = %s;
```

This query is used to delete a booking.

g. Doctor Dashboard

```
SELECT
        COUNT(BookingId) FROM BranchBookings
```

```
WHERE DoctorId = %s AND AppointmentDate = CURDATE();
```

Example: parameter=('DOC0000005')

Result:

COUNT(BookingId)
0

This query is used to get today's total bookings of a doctor.

```
SELECT
        bb.BookingId,
        bb.PatientId,
        u.Email,
        bb.PatientName,
        u.Gender,
        u.PhoneNumber,
        p.ProfilePicture,
        DATE_FORMAT(bb.AppointmentDate, '%Y-%m-%d') AS
AppointmentDate,
        DATE_FORMAT(bb.AppointmentHour, '%H:%i') AS
AppointmentHour,
        bb.AppointmentStatus,
        bb.CheckUpType,
        bb.ReasonOfVisit,
        p.DateOfBirth
FROM BranchBookings bb
JOIN `User` u ON bb.PatientId = u.UserId
LEFT JOIN Patient p ON bb.PatientId = p.PatientId
WHERE bb.DoctorId = %s AND bb.AppointmentStatus = 'Pending'
ORDER BY bb.AppointmentDate ASC,
STR_TO_DATE(bb.AppointmentHour, '%H:%i') ASC;
```

Example: parameter=('DOC0000004')

Result:

BookingId	PatientId	Email	PatientName	Gender	PhoneNumbe	ProfilePicture	AppointmentDat	AppointmentHou	AppointmentStatus	CheckUpType	ReasonOfVisit	DateOfBirth
BOK000...	PAT000...	pat1...	FirstName1...	1	0812345671	NULL	2024-12-10	13:00	Pending	Type 22	Consultation	1998-02-23
BOK000...	PAT000...	pat28...	FirstName2...	1	08123456...	profile_pic...	2024-12-17	13:03	Pending	Type 1	Visit Reaso...	1985-10-22
BOK000...	PAT000...	pat74...	FirstName7...	0	08123456...	profile_pic...	2024-12-17	13:11	Pending	Type 4	Visit Reaso...	2007-04-24

This query is used to get all patients of that doctor with the appointment status of ‘Pending’ regardless of the appointment date.

```
UPDATE Booking SET AppointmentStatus = %s
WHERE BookingId = %s
```

This query will update the pending booking status to be canceled or completed based on the user input.

h. Doctor History Page

```
SELECT
        bb.BookingId,
        bb.PatientId,
        DATE_FORMAT(bb.AppointmentDate, '%Y-%m-%d') AS
AppointmentDate,
        DATE_FORMAT(bb.AppointmentHour, '%H:%i') AS
AppointmentHour,
        bb.PatientName,
        CASE u.Gender
                WHEN 0 THEN 'Male'
                ELSE 'Female'
        END AS Gender,
        p.DateOfBirth,
        u.Email,
        u.PhoneNumber,
        bb.AppointmentStatus,
        bb.CheckUpType,
        bb.ReasonOfVisit
FROM BranchBookings bb
JOIN `User` u ON bb.PatientId = u.UserId
LEFT JOIN Patient p ON bb.PatientId = p.PatientId
WHERE bb.DoctorId = %s
ORDER BY
{self.filter_field} {self.sort_order};
```

Example: parameter = ('DOC0000004'), self.filter_field = CONCAT(AppointmentDate, ' ', STR_TO_DATE(bb.AppointmentHour, '%H:%i')), self.sort_order = ASC

Result:

BookingId	PatientId	AppointmentDate	AppointmentHour	PatientName	Gender	DateOfBirth	Email	PhoneNumber	AppointmentStatus	CheckUpType	ReasonOfVisit
BOK0004286	PAT000...	2024-01-25	10:00	FirstName3...	Male	1992-06-24	pat36...	08123456...	Completed	Type 4	Visit Reason 42
BOK0006424	PAT000...	2024-02-27	12:58	FirstName6...	Male	2010-02-11	pat63...	08123456...	Completed	Type 2	Visit Reason 64
BOK0001434	PAT000...	2024-03-07	10:31	FirstName3...	Male	2016-11-01	pat33...	08123456...	Completed	Type 1	Visit Reason 14
BOK0000008	PAT000...	2024-04-04	10:35	FirstName4...	Male	2008-03-04	pat40...	08123456...	Completed	Type 5	Visit Reason 8

This query is used to get all the booking history of a doctor.

i. Doctor Profile Page

```
SELECT
        d.ProfilePicture, d.Description, d.SpecialtyId, s.SpecialtyName
FROM Doctor d JOIN Specialty s ON d.SpecialtyId = s.SpecialtyId
WHERE d.DoctorId = %s;
```

Example: parameter=('DOC0000004')

Result:

ProfilePicture	Description	SpecialtyId	SpecialtyName
NULL	Doctor description 4 lorem ipsum dolor sit amet	SP0000006	Specialty 6

This query is used to get the doctor's profile picture, description, and specialty information.

```
UPDATE User SET FirstName = %s, LastName = %s, Gender = %s,
PhoneNumber = %s, City = %s, AddressDetail = %s
WHERE UserId = %s
```

This query is used to update the doctor's information in the database based on the doctor's inputted data.

```
SELECT CONCAT(SpecialtyId, ': ', SpecialtyName) FROM Specialty
```

Result:

CONCAT(SpecialtyId, ': ', SpecialtyName)
SP0000001: Specialty 1
SP0000002: Specialty 2
SP0000003: Specialty 3
SP0000004: Specialty 4

This query will get all the specialties' names and IDs concatenated in a string.

```
UPDATE Doctor SET SpecialtyId = %s WHERE DoctorId = %s;
```

This query is used to update the doctor's specialty.

```
UPDATE Doctor SET Description = %s WHERE DoctorId = %s
```

This query is used to update the doctor's description.

```
UPDATE Doctor SET ProfilePicture = NULL WHERE DoctorId = %s
```

This query is used to update the doctor's profile picture to null. It will reset the doctor's picture to a default placeholder picture in the application.

j. Patient Home Page

```
SELECT ProfilePicture, DATE_FORMAT(DateOfBirth, '%Y-%m-%d') AS formatted_dob FROM Patient WHERE PatientId = %s;
```

Example: parameter=('PAT0000004')

Result:

ProfilePicture	formatted_dob
profile_pictures...	1999-09-17

This query is used to get patient-specific information of a patient.

```
SELECT
        DATE_FORMAT(AppointmentDate, '%Y-%m-%d') AS Date,
        DATE_FORMAT(AppointmentHour, '%H:%i') AS Hour
from BranchBookings
WHERE AppointmentDate >= CURDATE() and PatientId = 'PAT0000004'
ORDER BY AppointmentDate, AppointmentHour ASC LIMIT 1;
```

Example: parameter=('PAT0000004')

Result:

Date	Hour
2024-12-25	14:31

This query is used to get the next appointment of this patient.

```
UPDATE Patient SET DateOfBirth = %s WHERE PatientId = %s;
```

This query is used to update the patient's date of birth.

k. Patient Booking Page

```
SELECT BranchNo, BranchName FROM ClinicBranch;
```

Result:

BranchNo	BranchName
1	Branch 1
2	Branch 2
3	Branch 3
4	Branch 4
5	Branch 5

This query is used to get the number (unique identifier) and name of a branch.

```
SELECT d.DoctorId, d.ProfilePicture, d.Description, s.SpecialtyName,
s.SpecialtyDescription,
u.FirstName, u.LastName, cb.BranchName
FROM Doctor d
JOIN Specialty s ON d.SpecialtyId = s.SpecialtyId
JOIN `User` u ON d.DoctorId = u.UserId
JOIN ClinicBranch cb ON d.BranchNo = cb.BranchNo
WHERE u.IsDeleted = 0;
```

Result:

DoctorId	ProfilePicture	Description	SpecialtyName	SpecialtyDescription	FirstName	LastName	BranchName
DOC0000002	profile_pictures/doctors\test.jpg	Doctor description 2	Specialty 6	Description of Specialty 6	FirstName2	LastName2	Branch 1
DOC0000003	profile_pictures\doctors\doctor2.jpg	Doctor description 3	Specialty 13	Description of Specialty 13	FirstName3	LastName3	Branch 1
DOC0000004	NULL	Doctor description 4 ...	Specialty 6	Description of Specialty 6	FirstNamee4	LastNamee4	Branch 1
DOC0000005	profile_pictures\doctors\doctorf4.jpg	Doctor description 5	Specialty 11	Description of Specialty 11	FirstName5	LastName5	Branch 1

This query is used to get all the doctor's information from name, profile picture, description, specialty, and branch. If the user search the doctor based on the branch and specialty, then there is an additional to the WHERE condition. The query will instead be like this

```
SELECT d.DoctorId, d.ProfilePicture, d.Description, s.SpecialtyName,
s.SpecialtyDescription,
u.FirstName, u.LastName, cb.BranchName
FROM Doctor d
JOIN Specialty s ON d.SpecialtyId = s.SpecialtyId
JOIN `User` u ON d.DoctorId = u.UserId
JOIN ClinicBranch cb ON d.BranchNo = cb.BranchNo
WHERE u.IsDeleted = 0 AND cb.BranchName = %s AND s.SpecialtyName
= %s;
```

Example: parameter=('Branch 5', 'Specialty 1')

Result:

DoctorId	ProfilePicture	Description	SpecialtyName	SpecialtyDescription	FirstName	LastName	BranchName
DOC0000207	profile_pictures\doctors\doctor5.jpg	Doctor description 207	Specialty 1	Description of Specialty 1	FirstName207	LastName...	Branch 5
DOC0000212	profile_pictures\doctors\doctor2.jpg	Doctor description 212	Specialty 1	Description of Specialty 1	FirstName212	LastName...	Branch 5

```
INSERT INTO Booking (BookingId, PatientId, DoctorId, AppointmentDate,
AppointmentHour, AppointmentStatus, CheckUpType, ReasonOfVisit)
VALUES (%s, %s, %s, %s, %s, %s, %s, %s);
```

This query is used to insert the new booking into the table.

l. Patient History Page

```
SELECT
        bb.DoctorId,
        DATE_FORMAT(bb.AppointmentDate, '%Y-%m-%d') AS
AppointmentDate,
        DATE_FORMAT(bb.AppointmentHour, '%H:%i') AS
AppointmentHour,
        bb.DoctorName,
        CASE u.Gender
                WHEN 0 THEN 'Male'
                ELSE 'Female'
        END AS Gender,
        s.SpecialtyName,
        br.BranchName,
        bb.AppointmentStatus,
        bb.CheckUpType,
        bb.ReasonOfVisit
FROM BranchBookings bb
JOIN `User` u ON bb.DoctorId = u.UserId
LEFT JOIN Doctor d ON bb.DoctorId = d.DoctorId
LEFT JOIN Specialty s ON d.SpecialtyId = s.SpecialtyId
LEFT JOIN ClinicBranch br ON d.BranchNo = br.BranchNo
WHERE bb.PatientId = %s
ORDER BY {self.filter_field} {self.sort_order};
```

Example: parameters = ('PAT0000004'), self.filter_field = BranchName, self.sort_order = ASC

Result:

DoctorId	AppointmentDate	AppointmentHour	DoctorName	Gender	SpecialtyName	BranchName	AppointmentStatus	CheckUpType	ReasonOf
DOC0000028	2024-03-07	17:24	FirstName28 LastName28	Male	Specialty 9	Branch 1	Completed	Type 1	Visit Reason
DOC0000037	2024-06-14	12:22	FirstName37 LastName37	Female	Specialty 20	Branch 1	Completed	Type 4	Visit Reason
DOC0000028	2024-08-08	18:16	FirstName28 LastName28	Male	Specialty 9	Branch 1	Completed	Type 5	Visit Reason

This query is used to get the booking history of the patient.

G. Conclusion

The making of this Clinical Appointment System app has provided us with valuable insights on how to apply database tech on to address real world challenges in healthcare management. This app focuses on scheduling and managing appointments, catered to 3 different types of users: Patients, Doctors, and Hospital Admins.

Designing and implementing this project has demonstrated the effectiveness of database solutions in enhancing user experience. Some of our key achievements include creating a GUI and a well-optimized database architecture. Despite challenges we encountered, like refining the user interface, balancing complexity with usability, matching the tables with real life user needs, or normalizing the tables, this project has deepened our understanding of the database design, SQL programming and user centric development. Overall, this project demonstrates and helped us understand the effectiveness of databases in improving healthcare operations.