HEALTHCARE MANAGEMENT SYSTEM

JAYNAB 2023UCM2354

AYUSH 2023UCM2314

TANISH 2023UCM2379

Contents

Introduction

Overview

Definition 1. In recent years, healthcare institutions have increasingly relied on digital systems to manage their day-to-day operations and improve patient care. This project focuses on developing a comprehensive **Healthcare Management System (HMS)** that integrates various essential functions within a healthcare setting. The primary goal of this HMS is to streamline and optimize the management of healthcare services, including doctor consultations, patient appointments, lab tests, and medication inventory.

The healthcare management system includes a secure login and registration feature for doctors and patients. This allows both user types to register and access their accounts with unique usernames and encrypted passwords, ensuring data privacy and controlled access. This addition supports the system's goal of managing sensitive information securely while providing personalized access to different functionalities, such as appointment booking for patients and schedule management for doctors.

The system is designed to address several challenges faced by traditional healthcare systems, such as inefficient appointment scheduling, lack of real-time availability updates for doctors, and manual management of lab results and medication inventory. By implementing this system, healthcare providers can offer a more organized, efficient, and user-friendly experience for patients, while ensuring data consistency and integrity.

This system enables users to view the availability of blood types and organs at nearby medical facilities, improving access to vital resources during emergencies.

Objectives

Theorem 1. The main objective of this project is to develop a robust **Database Management System (DBMS)** that meets the operational requirements of a healthcare setting. Specifically, this DBMS will:

1. **Support Efficient Appointment Scheduling:** Patients can easily book appointments with doctors based on specialization and available time slots. This reduces the risk of double-booking and allows doctors to manage their schedules effectively.

2. **Manage Doctor Availability:** Doctors have the flexibility to update their availability status, which directly impacts the appointment booking system. This feature is particularly useful for handling unexpected changes in schedule, ensuring that patients always see up-to-date availability.

3. **Handle Lab Test Bookings and Results:** The HMS includes an integrated lab management system where patients can book lab tests, and both doctors and patients can view test results. This feature ensures that important diagnostic information is accessible when needed, improving the continuity of care.

4. **Oversee Medication Inventory:** The system maintains an inventory of medications, including details like quantity, price, and manufacturer. This functionality aids in tracking available stock and pricing, which is essential for both patient safety and effective medication management.

5. **Ensure Data Consistency and Integrity:** Through careful database design, including normalization up to the third normal form (3NF), this project aims to reduce data redundancy and enforce data integrity. This approach improves the reliability of the data, which is critical in a healthcare environment where accurate records are paramount.

6. **Improve Patient Experience and Satisfaction:** By providing a seamless interface for booking appointments, viewing lab test results, and accessing prescription information, this system enhances the overall patient experience, allowing individuals to actively participate in their healthcare journey.

7. To provide real-time information on the availability of blood and organs at local hospitals, enhancing response times for patients in critical need.

Scope of the Project

Example. This HMS project encompasses the design, development, and implementation of a DBMS capable of managing various healthcare functions. The system is intended to be used by three primary user groups:

- **Doctors**, who can manage their availability, view patient appointments, and review lab test results.
- **Patients**, who can book appointments, view their lab test results, and access their prescription history.
- **Administrative Staff**, who manage the inventory, oversee lab tests, and handle patient records.
- This project will cover the availability of various blood types and organs across registered medical centers, allowing users to locate nearby resources efficiently.

The database for this HMS is developed using **XAMPP** and **SQL** for creating and managing the database schema. SQL is used for data definition (DDL) and data manipulation (DML) commands, ensuring the database is functional and responsive to application needs.

The database for this HMS is developed using **XAMPP** and **SQL** for creating and managing the database schema. SQL is used for data definition (DDL) and data manipulation (DML) commands, ensuring the database is functional and responsive to application needs. The scope of this healthcare management system includes a robust login and registration module for doctors and patients, providing each with a unique username and a securely stored password hash. This functionality enhances data privacy, enabling doctors and patients to safely access their respective dashboards, view relevant information, and perform functions like appointment scheduling, test result viewing, and availability updates. This feature strengthens the system's compliance with data security standards, making it suitable for real-world healthcare applications.

ER Model

Entity-Relationship Diagram (ERD) for Healthcare Management System

Definition 2. The Entity-Relationship (ER) Diagram of the Healthcare Management System (HMS) captures the relationships between various entities, such as doctors, patients, appointments, prescriptions, lab tests, and inventory. Each entity represents a critical component of the healthcare system, and the relationships among them describe how these components interact with each other. Below is a detailed explanation of each entity, its attributes, and the relationships defined in the ER diagram.

Entity Sets

Theorem 2. 1. **Doctor**

- **Attributes**: `DoctorID` (Primary Key), `Name`, `Specialization`, `Phone_No`, `Email`, `Consultation_Fee`
- **Type**: Strong Entity
- **Reason**: The **Doctor** entity has a primary key (`DoctorID`) that uniquely identifies each record, making it self-sufficient without relying on any other entity.
- **Description**: Represents doctors in the system, capturing essential details like name, specialization, and contact information. Doctors have consultation fees, which patients can view when making appointments.

2. **Patient**

- **Attributes**: `PatientID` (Primary Key), `Name`, `Phone_No`, `Address`, `Date_Of_Birth`,

Gender

- **Type**: Strong Entity
- **Reason**: **Patient** is a standalone entity with its own primary key (PatientID) and does not rely on any other entity to uniquely identify its records.
- **Description**: Represents patients registered in the healthcare system, with information on contact details, date of birth, and gender.

3. **Appointment**

- **Attributes**: AppID (Primary Key), DoctorID, PatientID, Appointment_Date,**Start_time**
- **Type**: Strong Entity
- **Reason**: **Appointment** has a unique identifier (AppID) and exists independently, though it has foreign keys referencing Doctor and Patient.
- **Description**: This entity represents the appointments booked by patients with doctors, storing data such as the date and the associated doctor and patient IDs.

4. **Prescription**

- **Attributes**: AppID, PrID (Composite Primary Key), Date
- **Type**: Weak Entity
- **Reason**: **Prescription** does not have a unique primary key of its own. It relies on the AppID of the **Appointment** entity and its own PrID to form a composite key. It's identified by the AppID from the **Appointment** entity.
- **Description**: Stores prescriptions generated during appointments, linked to specific appointments with prescribed medicines.

5. **LabTest**

- **Attributes**: TestID (Primary Key), Test_Name, Price
- **Type**: Strong Entity
- **Reason**: **LabTest** has a unique primary key (TestID) and is independent of other entities.
- **Description**: Represents the various lab tests offered by the healthcare facility, including details such as test name and price.

6. **Inventory**

- **Attributes**: MedID (Primary Key), Manufacturer, Quantity, Price
- **Type**: Strong Entity
- **Reason**: **Inventory** has its own primary key (MedID) and does not depend on any other entity for identification.
- **Description**: Holds information about available medicines, their manufacturers, quantities, and prices.

7. **Availability**

   - **Attributes**: AvailID (Primary Key), DoctorID, Day, Start_Time, End_Time, Status
   - **Type**: Weak Entity
   - **Reason**: It relies on the DoctorID from the **Doctor** entity to identify availability records, forming a composite key with AvailID.
   - **Description**: Defines the availability slots for each doctor, specifying the day, time, and status (available or not).

8. **Prescribed_Labtest**

   - **Attributes**: UTestID (Primary Key), AppID, TestID, Date_of_Prescription, Result
   - **Type**: Weak Entity
   - **Reason**: It relies on the AppID from the **Appointment** entity and the TestID from **LabTest** to identify its records uniquely.
   - **Description**: Records the lab tests prescribed to patients, linked to specific appointments and lab tests, including the date prescribed and results.

9. **Prescribed_Medicine**

   - **Attributes**: AppID, MedID, PrID (Composite Primary Key), Dosage, Instructions
   - **Type**: Weak Entity
   - **Reason**: It is associated with AppID from **Appointment** and relies on the prescription (via PrID) for identification.
   - **Description**: Contains details on medicines prescribed in an appointment, including dosage and instructions.

10. **Type**
    - **Attributes**: TypeID, Type
    - **Type**: Strong Entity
    - **Description**: It stores the type of donation the users want, either blood or organ.
11. **Location**
    - **Attributes**: LocationID, HospitalName
    - **Type**: Strong Entity
    - **Description**: Represents entities with hospital names.

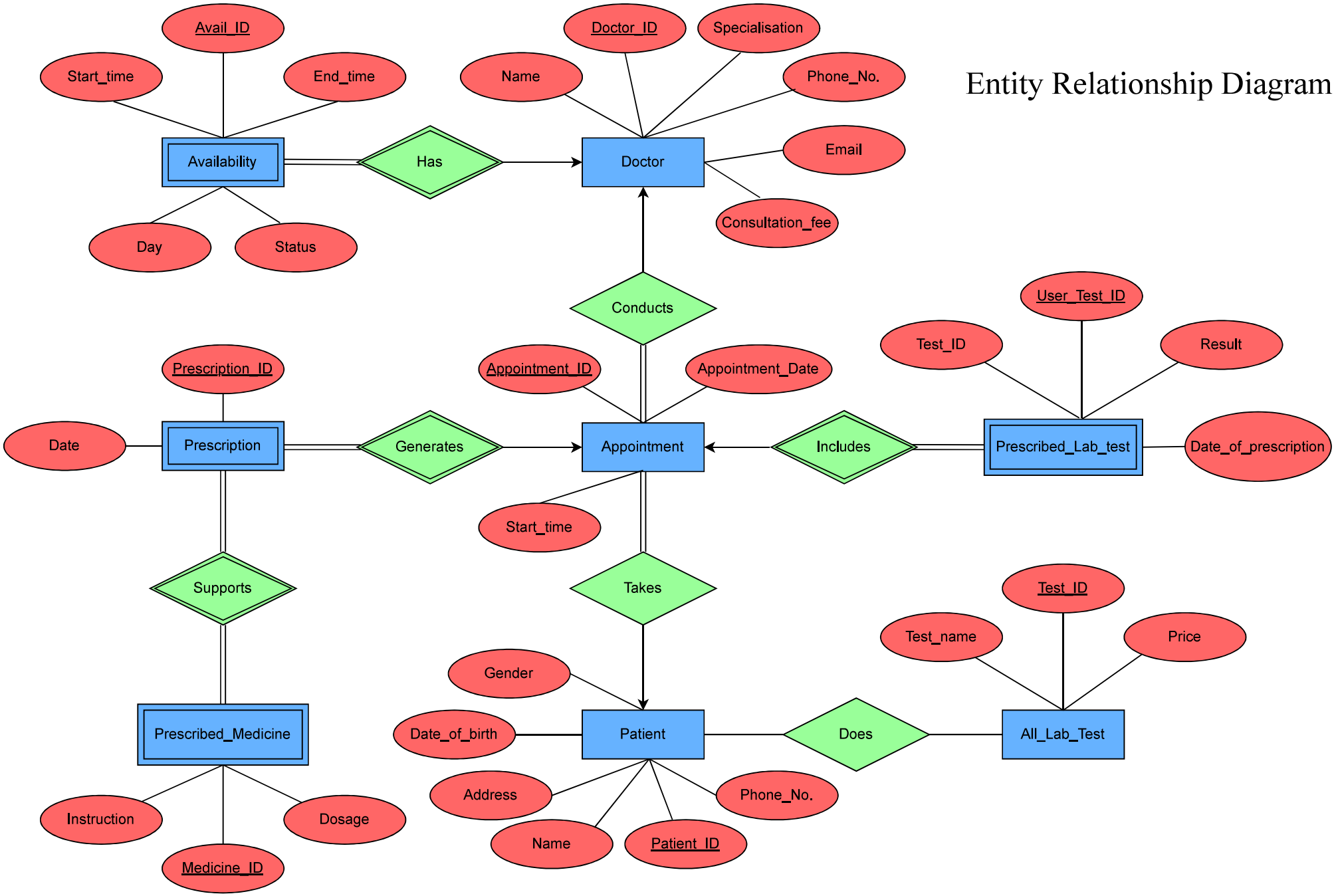
Relationship Sets

Example.
1. **has**
    - **Attributes**: None
    - **Type**: Identifying Relationship
    - **Entities Involved**: Doctor (1) to Availability (N)
    - **Description**: Represents the relationship between a doctor and their availability slots. Each doctor can have multiple availability slots, but a slot can have only one doctor associated with it.
2. **Generates**
    - **Attributes**: None
    - **Type**: Identifying Relationship
    - **Entities Involved**: Appointment (1) to Prescription (N)
    - **Description**: Represents that each appointment can generate multiple prescriptions, linked to a specific appointment.
3. **Takes**
    - **Attributes**: None
    - **Type**: Identifying Relationship

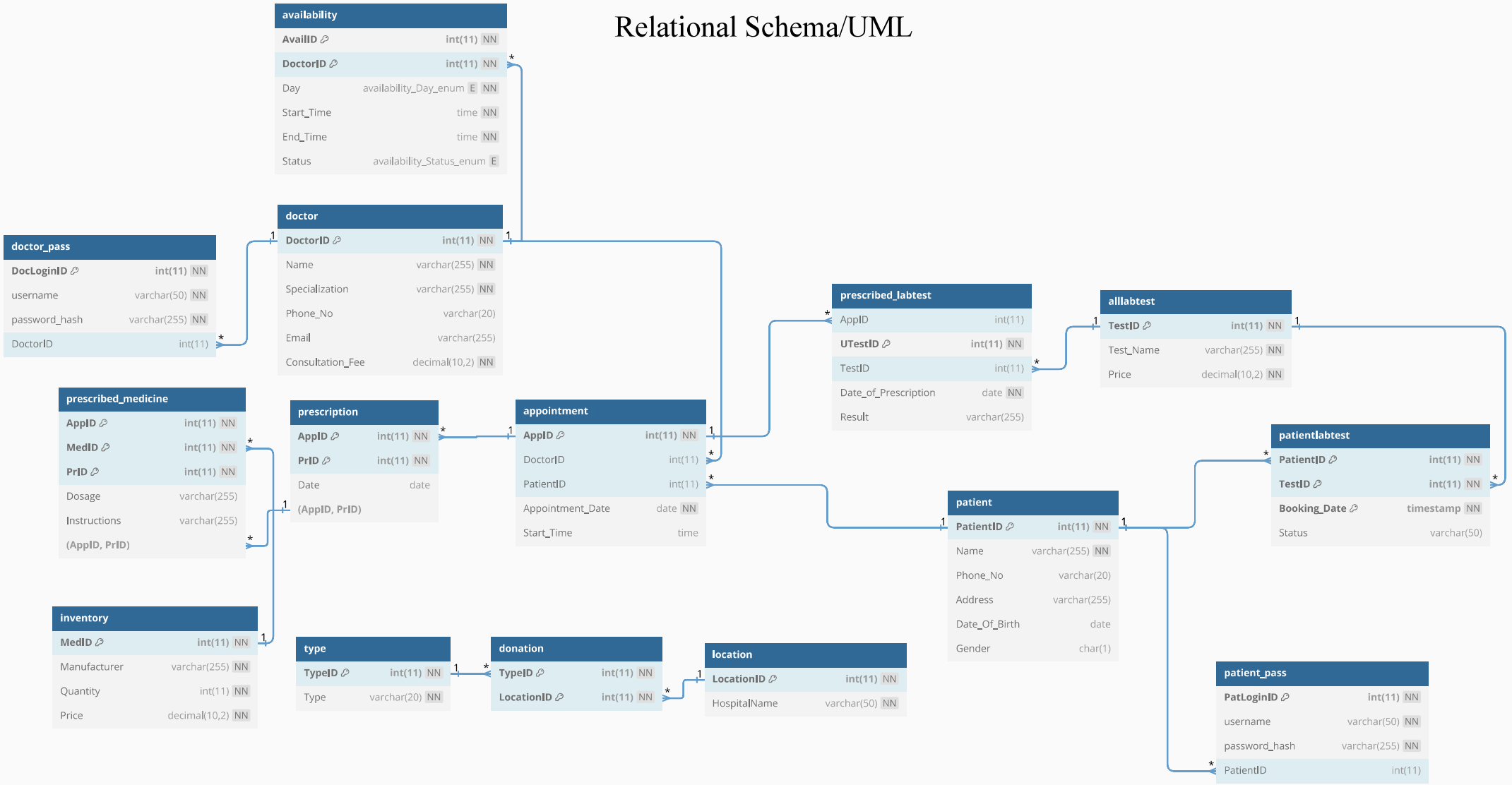
- **Entities Involved**: Patient (1) to Appointment (N)
- **Description**: Shows the relationship between patients and appointments, where a patient can have multiple appointments, but an appointment can have only one patient associated with it.

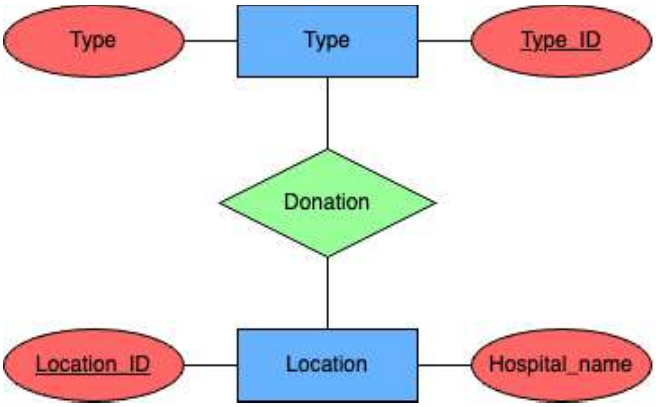
4. **Includes**
   - **Attributes**: None
   - **Type**: Identifying Relationship
   - **Entities Involved**: Appointment (1) to Prescribed_LabTest (N)
   - **Description**: Defines that an appointment may include multiple lab tests for the patient.
5. **Supports**
   - **Attributes**: None
   - **Type**: Identifying Relationship
   - **Entities Involved**: Prescription (1) to Prescribed_Medicine (N)
   - **Description**: Represents that a prescription can include multiple medicines.
6. **Conducts**
   - **Attributes**: None
   - **Type**: Non-identifying Relationship
   - **Entities Involved**: Doctor (1) to Appointment (N)
   - **Description**: This relationship signifies that a doctor conducts multiple appointments, but each appointment is associated with only one specific doctor.
7. **does**
   - **Attributes**: `Booking_Date`
   - **Type**: Many-to-Many Relationship
   - **Entities Involved**: Patient (M) to LabTest (N)
   - **Description**: Represents the association between patients and lab tests they have booked, with additional information on booking date and status.
8. **Donation**

- **Attributes**: None
- **Type**: Many-to-Many Relationship
- **Entities Involved**: Type and Location
- **Description**: Since a type of blood/organ can be available in many hospitals and a hospital can have multiple blood/organs available, it is a many-to-many relationship.

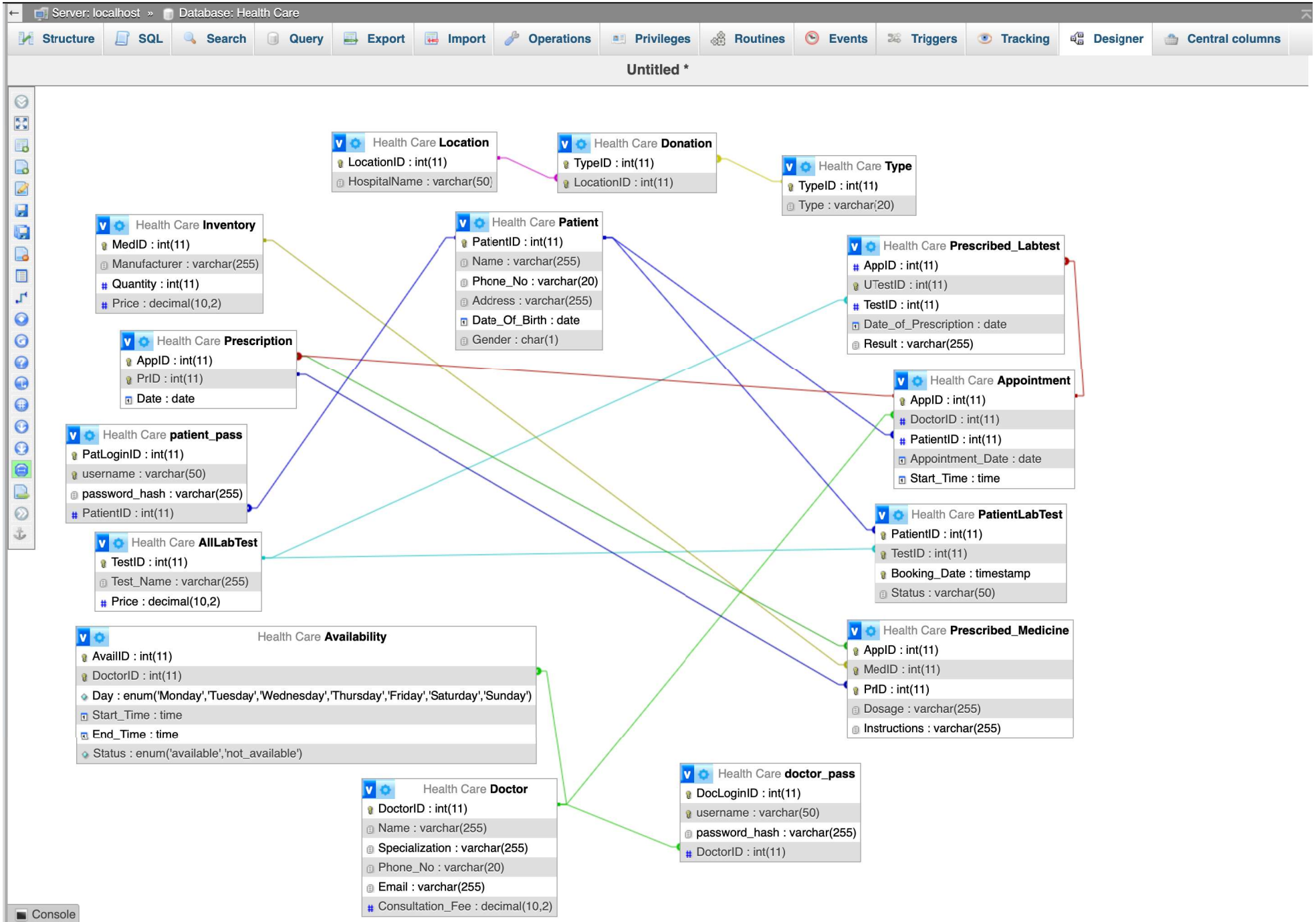

Entity Relationship Diagram
Avail_ID
Start_time
End_time
Availability
Has
Day
Status
Doctor_ID
Specialisation
Name
Phone_No.
Doctor
Email
Consultation_fee
Conducts
User_Test_ID
Test_ID
Result
Prescription_ID
Appointment_ID
Appointment_Date
Date
Prescription
Generates
Appointment
Includes
Prescribed_Lab_test
Date_of_prescription
Start_time
Supports
Takes
Test_ID
Test_name
Price
Gender
Prescribed_Medicine
Date_of_birth
Patient
Does
All_Lab_Test
Address
Phone_No.
Instruction
Dosage
Name
Patient_ID
Medicine_ID

Relational Schema/UML




Type
Type
Type_ID
Donation
Location_ID
Location
Hospital_name

Database Relationships Overview



Database Design

Introduction to Database Design

Definition 3. The **Database Design** for this healthcare management system serves as the foundation for effectively organizing, storing, and retrieving healthcare-related data. This system is essential for managing patient information, doctor schedules, appointments, lab tests, and inventory, enabling streamlined healthcare operations. A well-structured database design ensures data integrity, security, and accessibility, all of which are crucial in a healthcare setting where sensitive information is regularly managed.

The database design is based on an **ER (Entity-Relationship) model**, developed to capture the relationships between core entities such as patients, doctors, appointments, and prescriptions. After creating the ER model, it was translated into a relational schema using SQL to form the basis of the system's data structure. Tools such as **XAMPP** were utilized for SQL commands to define tables, primary and foreign keys, and constraints, while draw.io is used to visually model the relationships among entities. This structured approach allows easy conversion of real-world healthcare processes into an efficient system.

Deriving the Relational Schema from the ER Model

Theorem 3. 1. For Strong Entities

The relational schema of a strong entity set is directly represented with its attributes (in case of simple attributes) and the primary key is the primary key of the entity set itself. So the relational schema of all strong entity sets will be:

Doctor(DoctorID, Name, Specialization, Phone_No, Email, Consultation_Fee)

Patient(PatientID, Name, Phone_No, Address, Date_Of_Birth, Gender)

Appointment(AppID, DoctorID, PatientID, Appointment_Date, Start_time)

LabTest(TestID, Test_Name, Price)

Inventory(MedID, Manufacturer, Quantity, Price)

Type(typeID, type)

Location(locationID, HospitalName)

2. For Relationship Sets

The one-to-many relationships with total/partial participation are omitted, and the many-to-many relationship set is represented for the schema. The primary key of the table for this type of relationship set is the primary key from both the entities combined:

PatientLabTest(PatientID, TestID, Bookingdate, status)

(PatientID and TestID have foreign key constraints)

Similarly, for the Donation relationship:

Donation(TypeID, LocationID)

Identifying relationships are also not shown in the schema since the weak entity associated with any identifying relationship will have the same relational schema, which will cause data redundancy; hence it is omitted from the final relational schema.

3. For Weak Entity Sets

The primary key of the relational schema for a weak entity will be the primary key of the strong entity and the discriminator key of the weak entity. The rest of the attributes are taken as is:

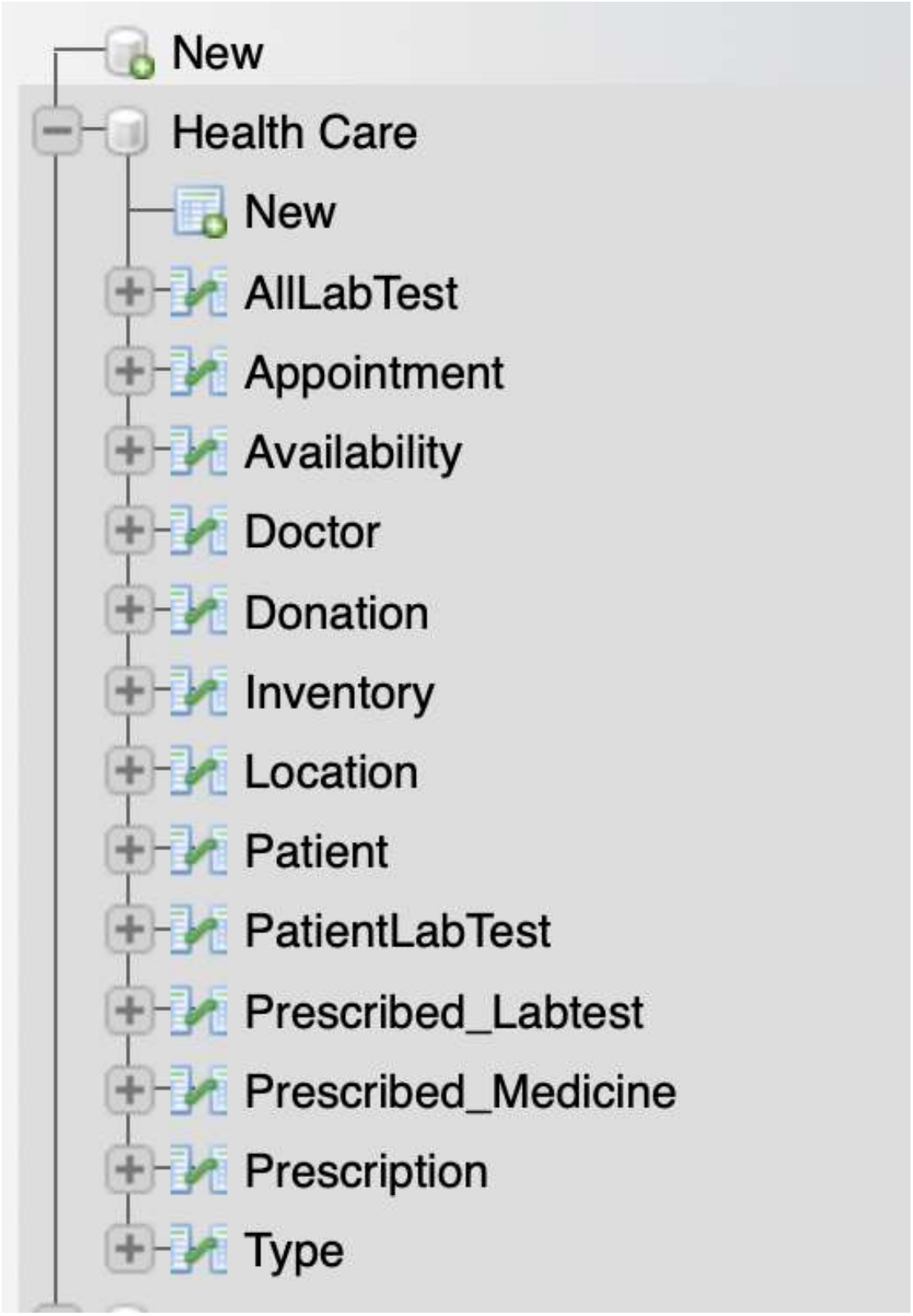
Availability(AvailID, DoctorID, Day, Start_Time, End_Time, Status)

Prescription(AppID, PrID, Date)

PrescribedLabTest(UTestID, AppID, TestID, Date_of_Prescription, Result)

PrescribedMedicine(AppID, MedID, PrID, Dosage, Instructions)

Directory


New
Health Care
New
AllLabTest
Appointment
Availability
Doctor
Donation
Inventory
Location
Patient
PatientLabTest
Prescribed_Labtest
Prescribed_Medicine
Prescription
Type

Normalization Process (up to 3NF) and Tables

1. Doctor Table

Example. Table: Doctor(DoctorID, Name, Specialization, Phone_No, Email, Consultation_Fee)
Primary Key: DoctorID

1NF Each field holds atomic values:

- **DoctorID** uniquely identifies each doctor.
- **Name** holds one name per doctor.
- **Specialization** holds a single specialization.
- **Phone_No** and **Email** hold only one phone number and one email per doctor.
- **Consultation_Fee** is a single fee.

2NF All non-key attributes (**Name**, **Specialization**, **Phone_No**, **Email**, **Consultation_Fee**) are fully dependent on the primary key **DoctorID**.

3NF There are no transitive dependencies; all non-key attributes depend only on **DoctorID**.

	DoctorID	Name	Specialization	Phone_No	Email	Consultation_Fee
Edit Copy Delete	1	Dr. Alice Smith	Cardiologist	1234567890	alice.smith@example.com	150.00
Edit Copy Delete	2	Dr. Bob Johnson	Dermatologist	0987654321	bob.johnson@example.com	120.00
Edit Copy Delete	3	Dr. Carol Lee	Orthopedic	1122334455	carol.lee@example.com	100.00
Edit Copy Delete	4	Dr. David Brown	Neurologist	1223344556	david.brown@example.com	200.00
Edit Copy Delete	5	Dr. Emily Davis	Pediatrician	1334455667	emily.davis@example.com	110.00
Edit Copy Delete	6	Dr. Frank Wilson	Psychiatrist	1445566778	frank.wilson@example.com	140.00
Edit Copy Delete	7	Dr. Grace Miller	Cardiologist	1556677889	grace.miller@example.com	160.00
Edit Copy Delete	8	Dr. Henry Martin	Dermatologist	1667788990	henry.martin@example.com	130.00
Edit Copy Delete	9	Dr. Irene Harris	Orthopedic	1778899001	irene.harris@example.com	105.00
Edit Copy Delete	10	Dr. Jack Clark	Radiologist	1889900112	jack.clark@example.com	125.00
Edit Copy Delete	11	Dr. Karen Lewis	Ophthalmologist	1990011223	karen.lewis@example.com	145.00
Edit Copy Delete	12	Dr. Leo Walker	Oncologist	2001122334	leo.walker@example.com	250.00
Edit Copy Delete	13	Dr. Mary Hall	Endocrinologist	2112233445	mary.hall@example.com	170.00
Edit Copy Delete	14	Dr. Nancy Young	Urologist	2223344556	nancy.young@example.com	115.00
Edit Copy Delete	15	Dr. Oliver King	Gastroenterologist	2334455667	oliver.king@example.com	135.00

2. Patient Table

Example. Table: Patient(PatientID, Name, Phone_No, Address, Date_Of_Birth, Gender)
Primary Key: PatientID

1NF Each field holds atomic values:

- **PatientID** uniquely identifies each patient.
- **Name** holds one name per patient.
- **Phone_No** contains a single phone number per patient.
- **Address** holds one address per patient.
- **Date_Of_Birth** is a single date for each patient.
- **Gender** represents the patient's gender with a single value.

2NF All non-key attributes (**Name**, **Phone_No**, **Address**, **Date_Of_Birth**, **Gender**) are fully dependent on the primary key **PatientID**.

3NF There are no transitive dependencies; all non-key attributes depend only on **PatientID**.

←T→	PatientID	Name	Phone_No	Address	Date_Of_Birth	Gender
Edit Copy Delete	1	John Doe	2345678901	123 Maple St	1985-06-15	M
Edit Copy Delete	2	Jane Smith	3456789012	456 Oak Ave	1990-08-20	F
Edit Copy Delete	3	Tom Brown	4567890123	789 Pine Blvd	2000-01-01	M
Edit Copy Delete	4	Lucy White	5678901234	101 Cedar Rd	1992-04-25	F
Edit Copy Delete	5	Anna Taylor	6789012345	202 Birch Ln	1980-03-15	F
Edit Copy Delete	6	Mark Green	7890123456	303 Elm St	1975-11-30	M
Edit Copy Delete	7	Emma Harris	8901234567	404 Walnut St	1988-10-12	F
Edit Copy Delete	8	Chris Clark	9012345678	505 Spruce St	2002-07-07	M
Edit Copy Delete	9	Olivia Lee	0123456789	606 Willow Rd	1995-05-17	F
Edit Copy Delete	10	Ethan Walker	1234567899	707 Ash Ave	1998-09-09	M
Edit Copy Delete	11	Sophia Hall	2345678909	808 Poplar St	1993-02-02	F
Edit Copy Delete	12	Liam Young	3456789019	909 Redwood St	2001-12-31	M
Edit Copy Delete	13	Mia King	4567890129	1010 Fir Dr	1984-08-18	F
Edit Copy Delete	14	James Scott	5678901239	1111 Cypress Ct	1979-06-21	M
Edit Copy Delete	15	Isabella Adams	6789012349	1212 Pine Cr	2003-11-14	F

3. Appointment Table

Example. Table: Appointment(AppID, DoctorID, PatientID, Appointment_Date, Start_time)

Primary Key: AppID

Functional Dependencies: AppID → DoctorID, PatientID, Appointment_Date, Start_time

1NF Each field holds atomic values:

- **AppID** uniquely identifies each appointment.
- **DoctorID** holds one doctor ID per appointment.
- **PatientID** holds one patient ID per appointment.
- **Appointment_Date** is a single date for each appointment.
- **Start_time** is a single time for each appointment.

2NF All non-key attributes (**DoctorID**, **PatientID**, **Appointment_Date, Start_time**) are fully dependent on the primary key **AppID**.

3NF There are no transitive dependencies; all non-key attributes depend only on **AppID**.

←T→	AppID	DoctorID	PatientID	Appointment_Date	Start_Time
Edit Copy Delete	6	6	6	2024-10-20	NULL
Edit Copy Delete	7	7	7	2024-10-21	NULL
Edit Copy Delete	8	8	8	2024-10-22	NULL
Edit Copy Delete	9	9	9	2024-10-23	NULL
Edit Copy Delete	10	10	10	2024-10-24	NULL
Edit Copy Delete	11	11	11	2024-10-25	NULL
Edit Copy Delete	12	12	12	2024-10-26	NULL
Edit Copy Delete	13	13	13	2024-10-27	NULL
Edit Copy Delete	14	14	14	2024-10-28	NULL
Edit Copy Delete	15	15	15	2024-10-29	NULL

4. Prescription Table

Example. Table: Prescription(AppID, PrID, Date)
Primary Key: Composite key (AppID, PrID)
Functional Dependency: (AppID, PrID) → Date

1NF Each field holds atomic values:

- **AppID** and **PrID** together uniquely identify each prescription entry.
- **Date** holds a single date per prescription entry.

2NF All non-key attributes (**Date**) are fully dependent on the composite primary key (**AppID, PrID**).

3NF There are no transitive dependencies; **Date** depends only on the composite key (**AppID, PrID**).

←T→		AppID	PrID	Date
☐	Edit Copy Delete	6	2	2024-10-20
☐	Edit Copy Delete	7	3	2024-10-21
☐	Edit Copy Delete	8	3	2024-10-22
☐	Edit Copy Delete	9	3	2024-10-23
☐	Edit Copy Delete	10	4	2024-10-24
☐	Edit Copy Delete	11	4	2024-10-25
☐	Edit Copy Delete	12	4	2024-10-26
☐	Edit Copy Delete	13	5	2024-10-27

5. AllLabTest Table

Example. Table: AllLabTest(TestID, Test_Name, Price)

Primary Key: TestID

Functional Dependency: TestID → Test_Name, Price

1NF Each field holds atomic values:

- **TestID** uniquely identifies each lab test.
- **Test_Name** holds a single test name per entry.
- **Price** holds a single price per test.

The **AllLabTest** table is in 1NF.

2NF All non-key attributes (**Test_Name**, **Price**) are fully dependent on the primary key **TestID**.

The **AllLabTest** table is in 2NF.

3NF There are no transitive dependencies, as all non-key attributes depend directly on **TestID**. Thus, the **AllLabTest** table is in 3NF.

←T→			TestID	Test_Name	Price
Edit	Copy	Delete	4	Urine Test	40.00
Edit	Copy	Delete	5	CT Scan	500.00
Edit	Copy	Delete	6	ECG	100.00
Edit	Copy	Delete	7	Liver Function Test	60.00
Edit	Copy	Delete	8	Thyroid Profile	80.00
Edit	Copy	Delete	9	Lipid Profile	90.00
Edit	Copy	Delete	10	Ultrasound	120.00
Edit	Copy	Delete	11	Blood Sugar	30.00
Edit	Copy	Delete	12	Allergy Test	150.00
Edit	Copy	Delete	13	Vitamin D Test	70.00

6. Prescribed_Labtest Table

Example. Table: Prescribed_Labtest(AppID, UTestID, TestID, Date_of_Prescription, Result)

Primary Key: UTestID (auto-incremented unique test identifier for each prescription)

Functional Dependencies:

- UTestID → AppID, TestID, Date_of_Prescription, Result
- AppID, TestID → UTestID, Date_of_Prescription, Result

1NF Each field holds atomic values:

- **UTestID** uniquely identifies each prescribed lab test.
- **AppID** links to a single appointment per entry.
- **TestID** links to one lab test per entry.
- **Date_of_Prescription** holds a single date.
- **Result** contains a single test result per entry.

Thus, the **Prescribed_Labtest** table is in 1NF.

2NF All non-key attributes (**AppID**, **TestID**, **Date_of_Prescription**, **Result**) are fully dependent on **UTestID**.

The **Prescribed_Labtest** table is therefore in 2NF.

3NF There are no transitive dependencies, as all non-key attributes depend solely on **UTestID**. Thus, the **Prescribed_Labtest** table is in 3NF.

←T→	AppID	UTestID	TestID	Date_of_Prescription	Result
Edit Copy Delete	6	1	6	2024-10-20	Irregularities Detected
Edit Copy Delete	7	2	7	2024-10-21	Normal
Edit Copy Delete	8	3	8	2024-10-22	High Levels Detected
Edit Copy Delete	9	4	9	2024-10-23	Stable
Edit Copy Delete	10	5	10	2024-10-24	Normal
Edit Copy Delete	11	6	11	2024-10-25	Within Range
Edit Copy Delete	12	7	12	2024-10-26	Allergy Detected
Edit Copy Delete	13	8	13	2024-10-27	Deficiency Detected

7. Availability Table

Example. Table: Availability(AvailID, DoctorID, Day, Start_Time, End_Time, Status)
Primary Key: Composite key (AvailID, DoctorID)
Functional Dependency: AvailID → DoctorID, Day, Start_Time, End_Time, Status

1NF Each field holds atomic values:

- **AvailID** uniquely identifies each availability entry.
- **DoctorID** references a single doctor per entry.
- **Day** holds a single day value.
- **Start_Time** and **End_Time** hold a single start and end time per entry.
- **Status** holds a single value indicating availability.

The **Availability** table is in 1NF.

2NF All non-key attributes (**DoctorID**, **Day**, **Start_Time**, **End_Time**, **Status**) are fully dependent on the composite primary key **AvailID**.

The **Availability** table is in 2NF.

3NF There are no transitive dependencies, as all non-key attributes depend directly on **AvailID**. Thus, the **Availability** table is in 3NF.

←T→	AvailID	DoctorID	Day	Start_Time	End_Time	Status
Edit Copy Delete	1	1	Tuesday	09:00:00	13:00:00	available
Edit Copy Delete	2	2	Thursday	14:00:00	18:00:00	not_available
Edit Copy Delete	3	3	Monday	08:00:00	12:00:00	available
Edit Copy Delete	4	4	Saturday	10:00:00	14:00:00	available
Edit Copy Delete	5	5	Sunday	11:00:00	15:00:00	not_available
Edit Copy Delete	6	6	Wednesday	12:00:00	16:00:00	available
Edit Copy Delete	7	7	Friday	09:00:00	11:00:00	not_available
Edit Copy Delete	8	8	Monday	10:00:00	13:00:00	available
Edit Copy Delete	9	9	Tuesday	11:00:00	14:00:00	available
Edit Copy Delete	10	10	Thursday	08:00:00	12:00:00	available

8. Inventory Table

Example. Table: Inventory(MedID, Manufacturer, Quantity, Price)
Primary Key: MedID
Functional Dependency: MedID → Manufacturer, Quantity, Price

1NF Each field holds atomic values:

- **MedID** uniquely identifies each medicine.
- **Manufacturer** holds a single manufacturer name per medicine.
- **Quantity** represents stock as a single value.
- **Price** is a single value for the cost per unit.

The **Inventory** table is in 1NF.

2NF All non-key attributes (**Manufacturer**, **Quantity**, **Price**) are fully dependent on the primary key **MedID**.

The **Inventory** table is in 2NF.

3NF There are no transitive dependencies, as all non-key attributes depend directly on **MedID**. Thus, the **Inventory** table is in 3NF.

←T→	MedID	Manufacturer	Quantity	Price
Edit Copy Delete	4	Medicorp	250	12.00
Edit Copy Delete	5	GoodHealth Pharma	300	18.00
Edit Copy Delete	6	PureMed Ltd.	180	22.00
Edit Copy Delete	7	HealthMeds	220	25.00
Edit Copy Delete	8	Vitalife	500	30.00
Edit Copy Delete	9	MediDirect	150	35.00
Edit Copy Delete	10	GlobalCare	400	28.00
Edit Copy Delete	11	HealthSolutions	350	26.00
Edit Copy Delete	12	LifeMeds Inc.	210	32.00
Edit Copy Delete	13	WellnessPharma	175	24.00

9. Prescribed_Medicine Table

Example. Table: Prescribed_Medicine(AppID, MedID, PrID, Dosage, Instructions)
Primary Key: Composite key (AppID, MedID, PrID)
Functional Dependency: (AppID, MedID, PrID) → Dosage, Instructions

1NF Each field holds atomic values:

- **AppID**, **MedID**, and **PrID** together uniquely identify each prescribed medicine entry.
- **Dosage** holds a single dosage amount or schedule.
- **Instructions** contains usage instructions as a single value.

The **Prescribed_Medicine** table is in 1NF.

2NF All non-key attributes (**Dosage**, **Instructions**) are fully dependent on the composite primary key (**AppID, MedID, PrID**).

The **Prescribed_Medicine** table is in 2NF.

3NF There are no transitive dependencies, as all non-key attributes depend directly on the composite primary key (**AppID, MedID, PrID**). Thus, the **Prescribed_Medicine** table is in 3NF.

←T→		AppID	MedID	PrID	Dosage	Instructions
☐	Edit Copy Delete	6	6	2	1 capsule	Before sleep
☐	Edit Copy Delete	7	7	3	2 tablets	Morning and Evening
☐	Edit Copy Delete	8	8	3	5ml syrup	Once daily
☐	Edit Copy Delete	9	9	3	1 tablet	With meals
☐	Edit Copy Delete	10	10	4	1 injection	Administer every 3 days
☐	Edit Copy Delete	11	11	4	3 drops	Apply to affected area
☐	Edit Copy Delete	12	12	4	1 tablet	Daily before breakfast
☐	Edit Copy Delete	13	13	5	2 tablets	After evening meal

10. PatientLabTest Table

Example. Table: PatientLabTest(PatientID, TestID, Booking_Date, Status)
Primary Key: Composite key (PatientID, TestID, Booking_Date)
Functional Dependency: (PatientID, TestID, Booking_Date) → Status

1NF Each field holds atomic values:

- **PatientID**, **TestID**, and **Booking_Date** together uniquely identify each patient lab test entry.
- **Status** indicates the test status as a single value.

The **PatientLabTest** table is in 1NF.

2NF All non-key attributes (**Status**) are fully dependent on the composite primary key (**PatientID, TestID, Booking_Date**).

The **PatientLabTest** table is in 2NF.

3NF There are no transitive dependencies, as the non-key attribute **Status** depends directly on the composite primary key (**PatientID, TestID, Booking_Date**). Thus, the **PatientLabTest** table is in 3NF.

←T→			PatientID	TestID	Booking_Date	Status
Edit	Copy	Delete	4	4	2024-10-18 10:30:00	Completed
Edit	Copy	Delete	5	5	2024-10-19 11:00:00	Pending
Edit	Copy	Delete	6	6	2024-10-20 09:45:00	In Progress
Edit	Copy	Delete	7	7	2024-10-21 12:00:00	Completed
Edit	Copy	Delete	8	8	2024-10-22 13:30:00	Pending
Edit	Copy	Delete	9	9	2024-10-23 14:45:00	Completed
Edit	Copy	Delete	10	10	2024-10-24 15:15:00	In Progress
Edit	Copy	Delete	11	11	2024-10-25 16:00:00	Pending
Edit	Copy	Delete	12	12	2024-10-26 16:45:00	Completed
Edit	Copy	Delete	13	13	2024-10-27 17:30:00	Pending

11. Type Table

Example. Table: Type(TypeID, Type)

Primary Key: TypeID

Functional Dependency: TypeID → Type

1NF Each field holds atomic values:

- **TypeID** uniquely identifies each type entry.
- **Type** holds a single type value (e.g., blood type, organ type).

The **Type** table is in 1NF.

2NF All non-key attributes (**Type**) are fully dependent on the primary key **TypeID**.
The **Type** table is in 2NF.

3NF There are no transitive dependencies, as the non-key attribute **Type** depends directly on the primary key **TypeID**. Thus, the **Type** table is in 3NF.

←T→			TypeID	Type
Edit	Copy	Delete	4	Blood Type B+
Edit	Copy	Delete	5	Blood Type AB+
Edit	Copy	Delete	6	Organ Liver
Edit	Copy	Delete	7	Blood Type A-
Edit	Copy	Delete	8	Blood Type B-
Edit	Copy	Delete	9	Organ Heart
Edit	Copy	Delete	10	Organ Lung
Edit	Copy	Delete	11	Blood Type O-
Edit	Copy	Delete	12	Blood Type AB-
Edit	Copy	Delete	13	Organ Eye

12. Location Table

Example. Table: Location(LocationID, HospitalName)

Primary Key: LocationID

Functional Dependency: LocationID → HospitalName

1NF Each field holds atomic values:

- **LocationID** uniquely identifies each location entry.
- **HospitalName** holds a single hospital name.

The **Location** table is in 1NF.

2NF All non-key attributes (**HospitalName**) are fully dependent on the primary key **LocationID**.

The **Location** table is in 2NF.

3NF There are no transitive dependencies, as the non-key attribute **HospitalName** depends directly on the primary key **LocationID**. Thus, the **Location** table is in 3NF.

←T→			LocationID	HospitalName
Edit	Copy	Delete	4	Green Valley Clinic
Edit	Copy	Delete	5	Westside Medical Center
Edit	Copy	Delete	6	Easttown Hospital
Edit	Copy	Delete	7	Downtown Health Center
Edit	Copy	Delete	8	Northern Lights Clinic
Edit	Copy	Delete	9	Southside Medical Plaza
Edit	Copy	Delete	10	Lakeside Hospital
Edit	Copy	Delete	11	Hilltop Health Facility
Edit	Copy	Delete	12	Park Avenue Clinic
Edit	Copy	Delete	13	Sunrise Health Clinic

13. Donation Table

Example. Table: Donation(TypeID, LocationID)

Primary Key: Composite key (TypeID, LocationID)

Functional Dependency: (TypeID, LocationID) → (no other attributes, this is a junction table)

1NF Each field holds atomic values:

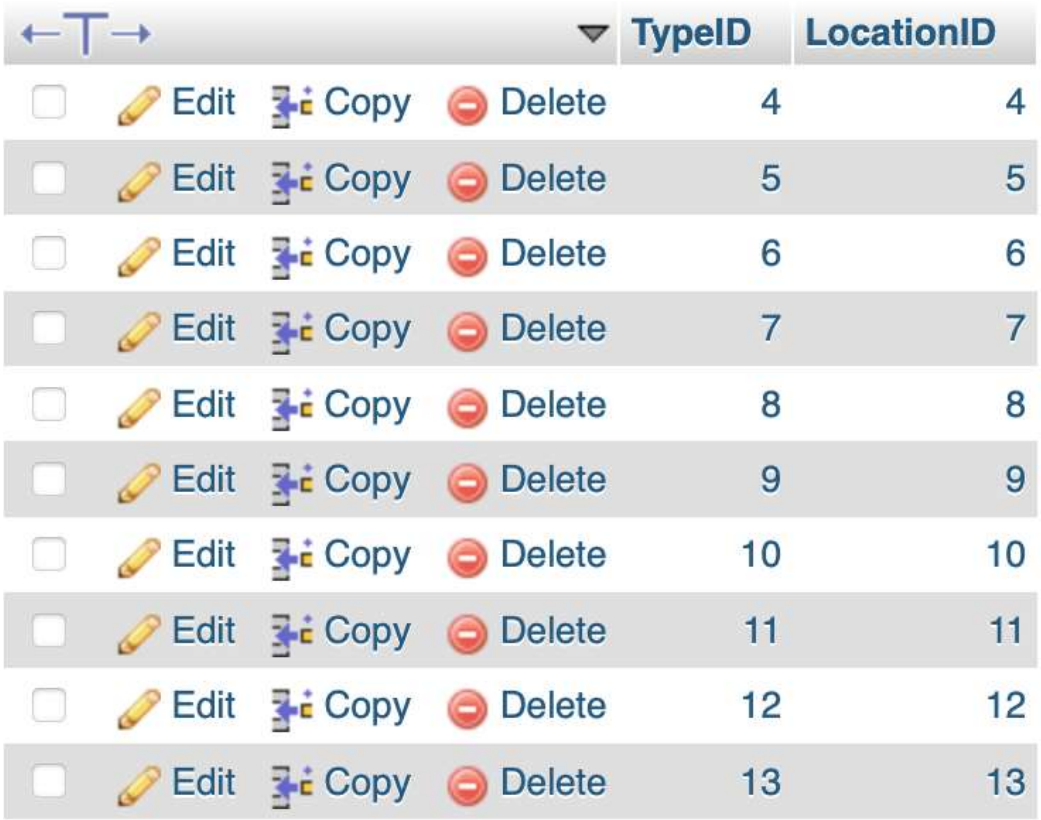
- **TypeID** and **LocationID** together uniquely identify each donation entry, representing a many-to-many relationship between types and locations.

The **Donation** table is in 1NF.

2NF There are no non-key attributes to depend on; thus, all attributes are dependent on the composite primary key (**TypeID, LocationID**).

The **Donation** table is in 2NF.

3NF Since there are no transitive dependencies, the **Donation** table is in 3NF. All key attributes are directly related to the composite primary key (**TypeID, LocationID**).



←T→			TypeID	LocationID
Edit	Copy	Delete	4	4
Edit	Copy	Delete	5	5
Edit	Copy	Delete	6	6
Edit	Copy	Delete	7	7
Edit	Copy	Delete	8	8
Edit	Copy	Delete	9	9
Edit	Copy	Delete	10	10
Edit	Copy	Delete	11	11
Edit	Copy	Delete	12	12
Edit	Copy	Delete	13	13

Implementation of Integrity Constraints

Primary Key Constraints

Theorem 4.

- **Doctor:** `DoctorID` is set as the primary key, ensuring each doctor has a unique identifier.
- **Patient:** `PatientID` is set as the primary key, ensuring each patient has a unique identifier.
- **Appointment:** `AppID` is the primary key, uniquely identifying each appointment.
- **Prescription:** Composite primary key of `AppID` and `PrID` to uniquely identify each prescription within an appointment.
- **AllLabTest:** `TestID` is the primary key, ensuring each lab test has a unique identifier.
- **Prescribed_Labtest:** `UTestID` is an auto-increment primary key to uniquely identify each prescribed test.
- **Availability:** Composite primary key of `AvailID` and `DoctorID`, ensuring a unique availability entry for each doctor and day.
- **Inventory:** `MedID` serves as the primary key for each medicine entry.
- **Prescribed_Medicine:** Composite primary key of `AppID`, `MedID`, and `PrID` uniquely identifies each prescribed medicine for an appointment.
- **PatientLabTest:** Composite primary key of `PatientID`, `TestID`, and `Booking_Date` ensures a unique record for each patient lab test booking.
- **Type:** `TypeID` is the primary key, providing unique identification for each blood or organ type.

- **Location:** `LocationID` serves as the primary key, ensuring unique identifiers for hospital locations.
- **Donation:** Composite primary key of `TypeID` and `LocationID` enforces uniqueness for each blood/organ type and location pair.

Foreign Key Constraints

Theorem 5.

- **Appointment:** `DoctorID` references `Doctor(DoctorID)`, and `PatientID` references `Patient(PatientID)`. These enforce relationships between appointments, doctors, and patients, ensuring valid doctor-patient references.
- **Prescription:** `AppID` references `Appointment(AppID)` to ensure that prescriptions are linked to valid appointments.
- **Prescribed_Labtest:** `AppID` references `Appointment(AppID)` and `TestID` references `AllLabTest(TestID)`, enforcing that each prescribed lab test is linked to a valid appointment and lab test.
- **Availability:** `DoctorID` references `Doctor(DoctorID)`, ensuring each availability entry is linked to a valid doctor.
- **Prescribed_Medicine:** `AppID` and `PrID` reference `Prescription(AppID, PrID)`, while `MedID` references `Inventory(MedID)`, ensuring medicines are prescribed within valid prescriptions and reference existing inventory.
- **PatientLabTest:** `PatientID` references `Patient(PatientID)` and `TestID` references `AllLabTest(TestID)`, ensuring each lab test booking has valid patient and test references.
- **Donation:** `TypeID` references `Type(TypeID)` and `LocationID` references `Location`, linking each donation to valid types and locations.

Unique Constraints

Definition 4. No additional unique constraints are explicitly defined beyond primary keys and foreign keys, which already enforce uniqueness for specific attributes.

Check Constraints

Definition 5.

- **Patient Table:** `Gender CHAR(1)` is restricted with a `CHECK` constraint to allow only 'M', 'F', or 'O' values, ensuring valid entries for gender.

Default Constraints

Definition 6.

- **Prescribed_Labtest:** `Date_of_Prescription` is set to the current date if not specified, to automatically timestamp when a test is prescribed.
- **PatientLabTest:** `Booking_Date` defaults to the current timestamp, recording the exact time of booking.
- **Availability:** `Status` defaults to 'available', assuming doctors are available unless otherwise specified.

Auto-Increment Constraints

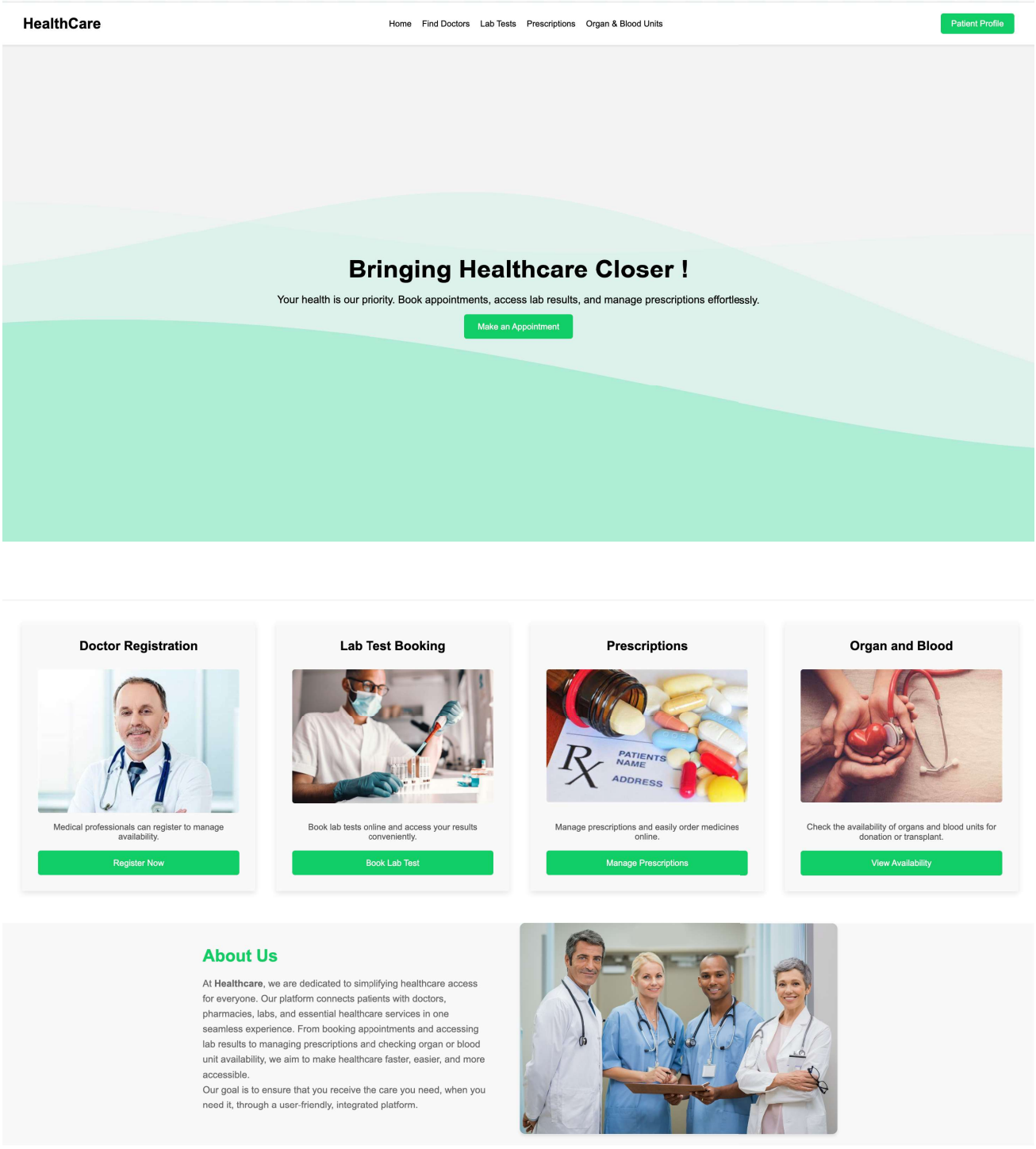
Definition 7. `Prescribed_Labtest (UTestID)`, `Availability (AvailID)`, `DoctorID`, `PatientID`, and `AppID` are auto-incremented, providing unique, automatically generated values for each new entry.

On Delete Cascade Constraints

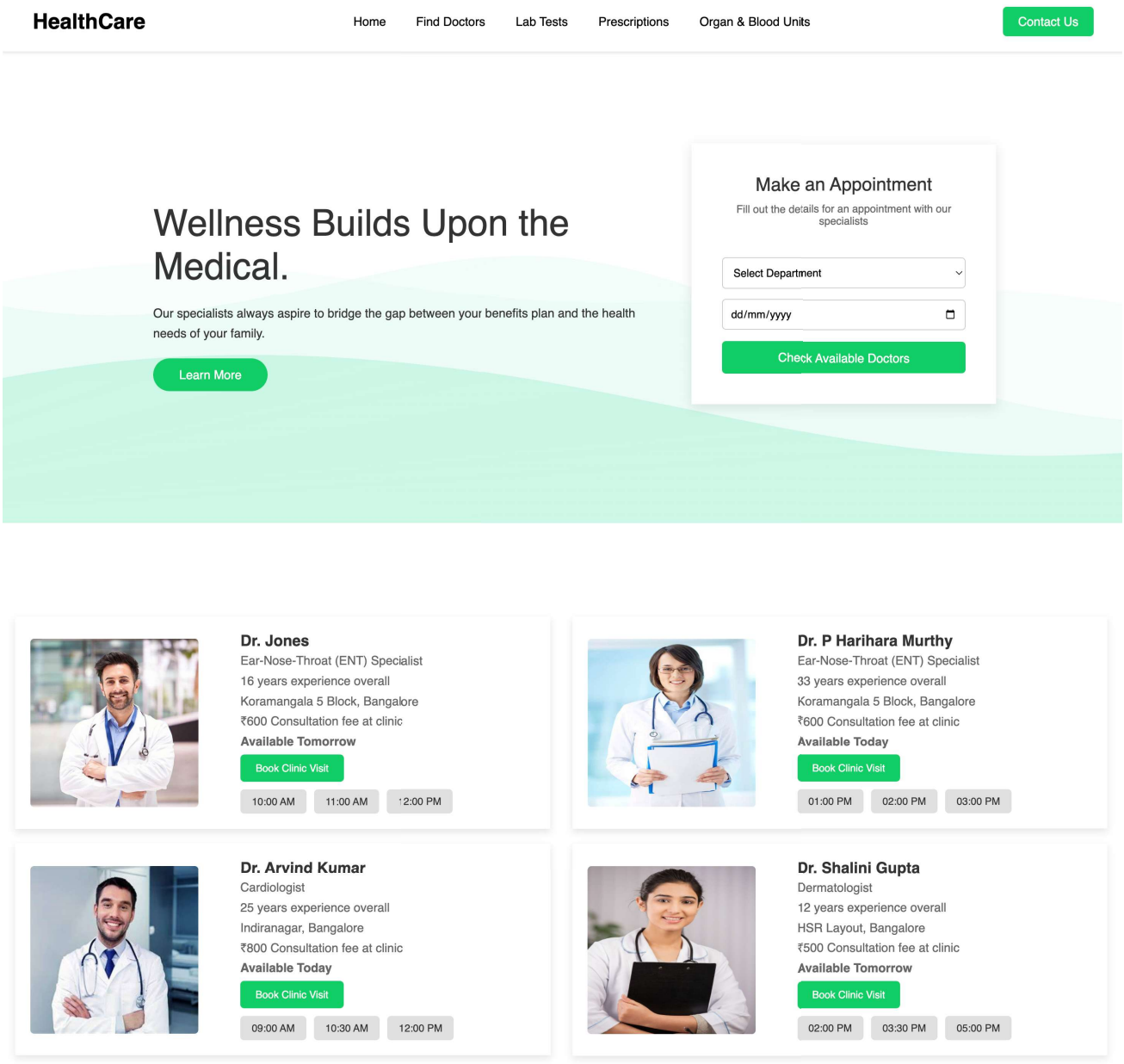
Definition 8. The `Appointment`, `Prescription`, `Prescribed_Labtest`, `Availability`, and `Prescribed_Medicine` tables have `ON DELETE CASCADE` applied to foreign keys referencing `DoctorID`, `PatientID`, `AppID`, `PrID`, and `MedID`. This ensures that deleting a doctor, patient, or appointment will automatically remove related entries from dependent tables, maintaining referential integrity across the database.

User Interface Design

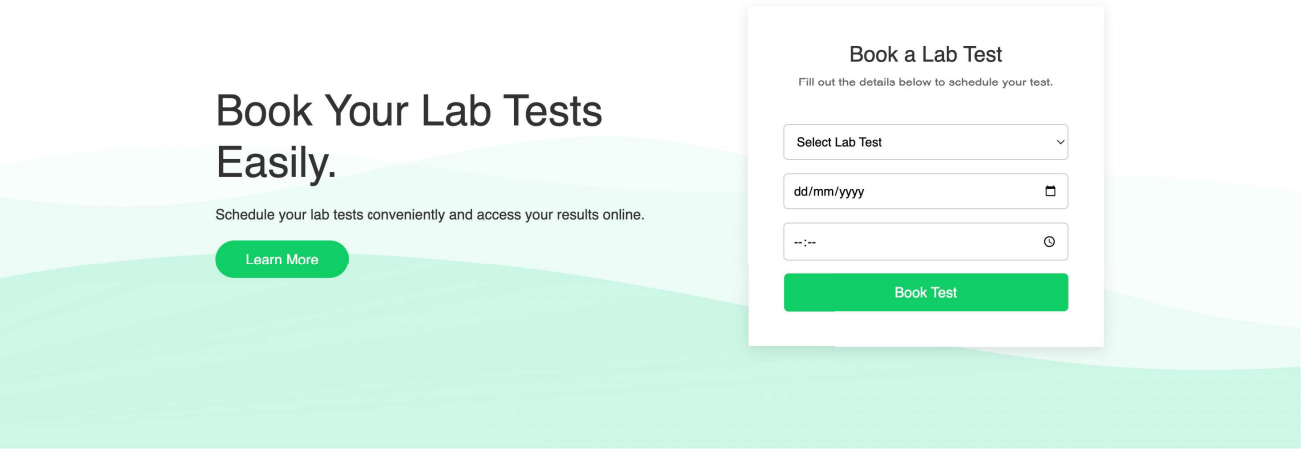
Homepage of the Website



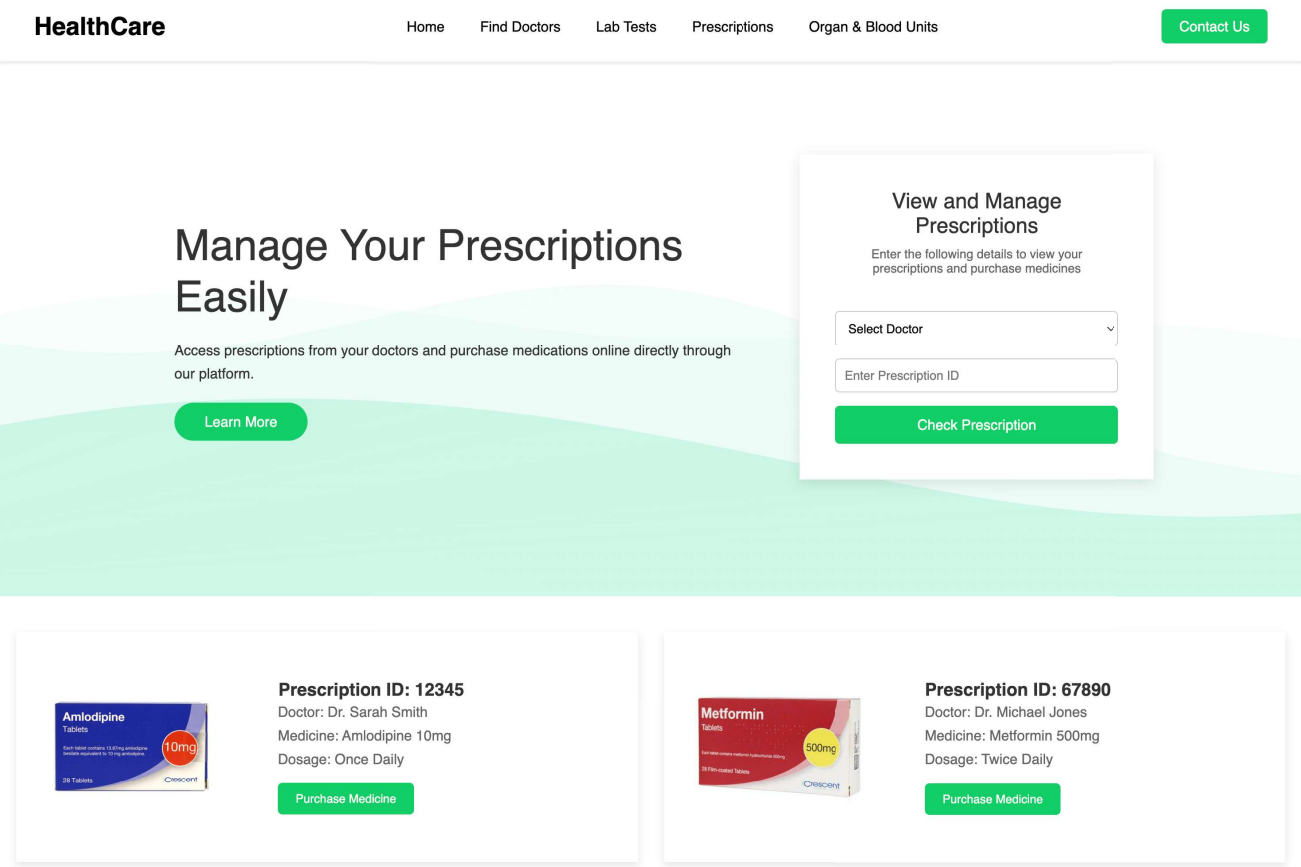
Appointment Scheduling Interface



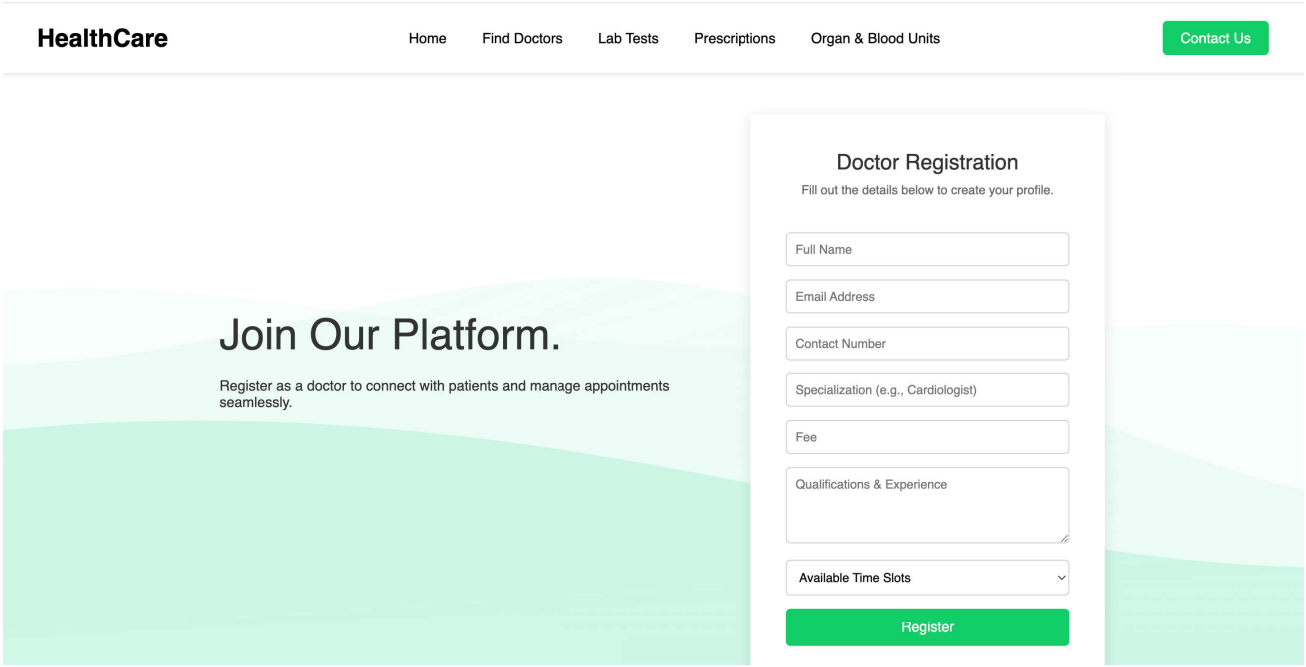
Lab Test Booking



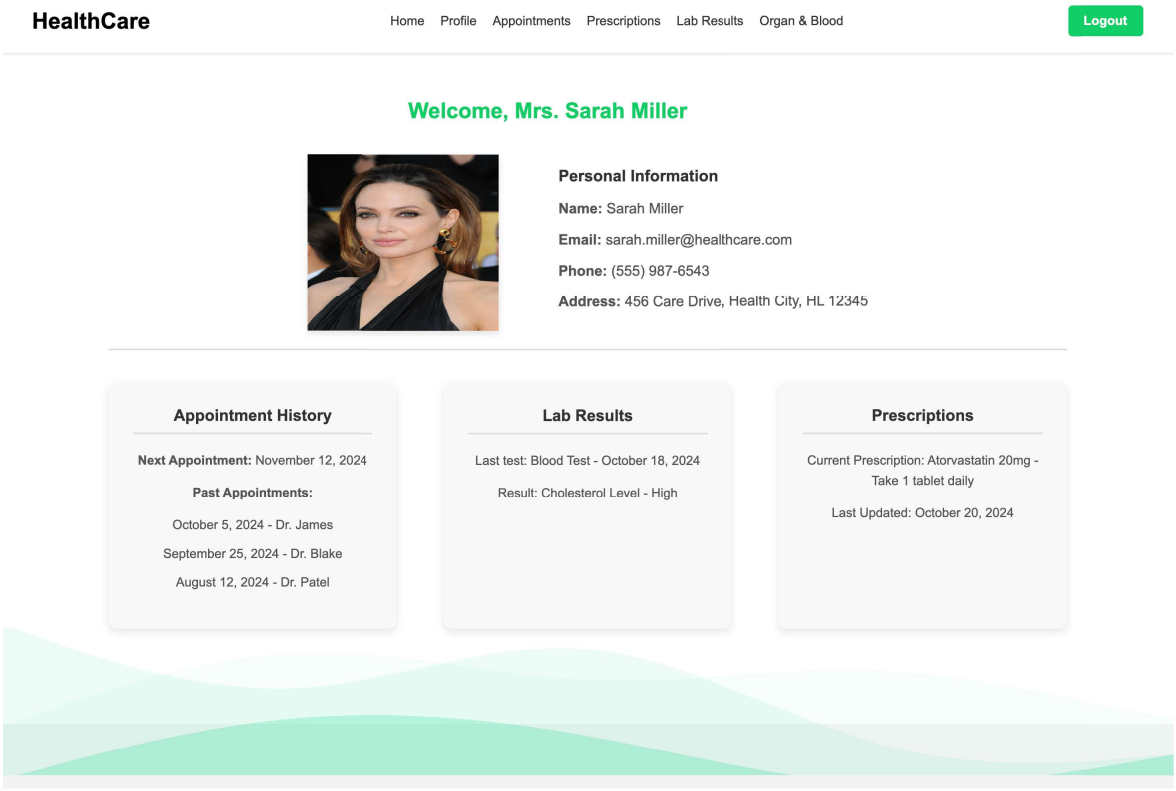
Prescriptions



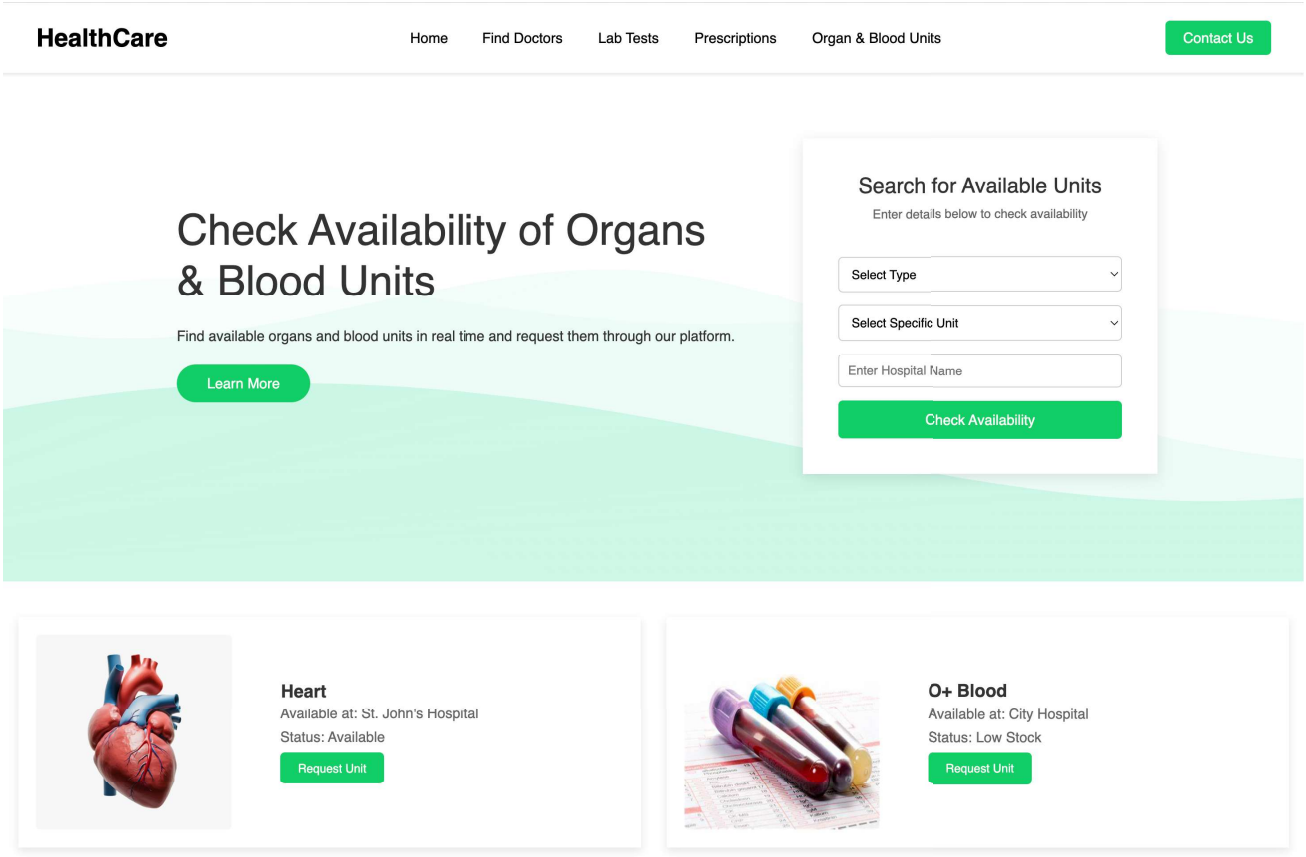
Doctor Registration



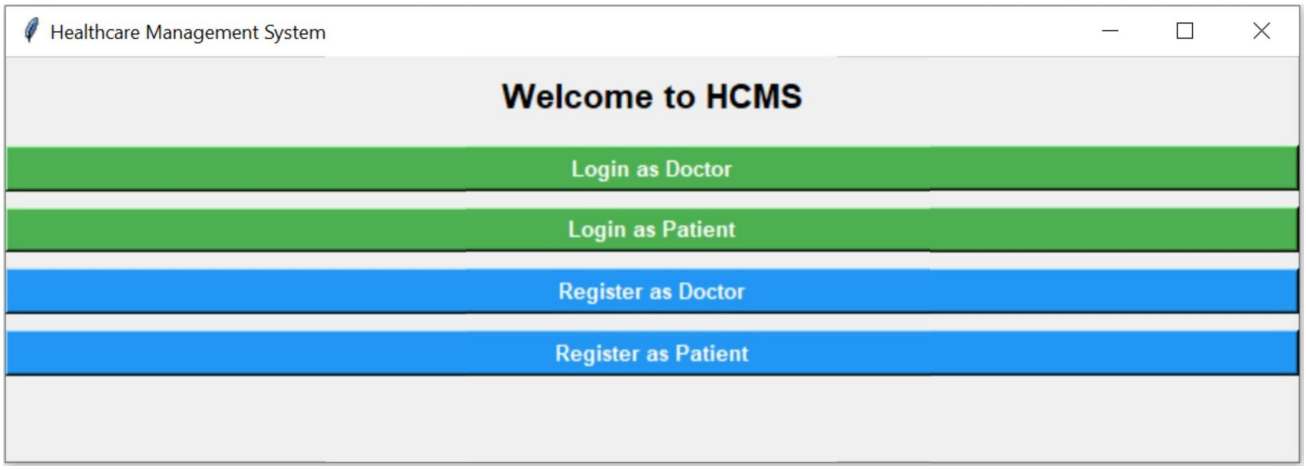
Patient Profile Page

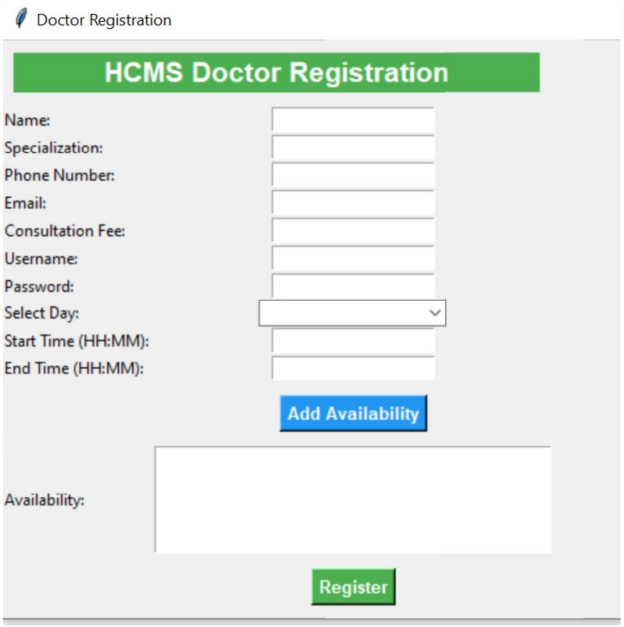


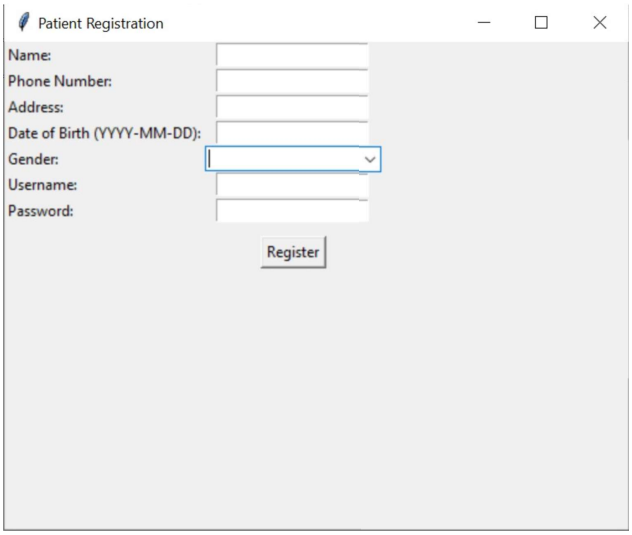
Organ and Blood Unit Availability

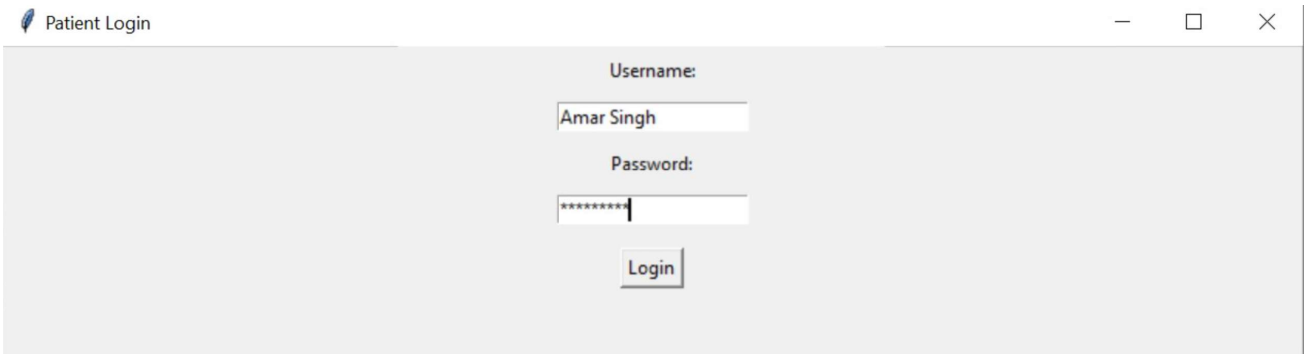


Additional Graphical User Interface










```
import tkinter as tk
from tkinter import messagebox, ttk
import mysql.connector
import bcrypt

# Function to connect to the MySQL database
def connect_to_db():
return mysql.connector.connect(
host="localhost",
user="root",
password="Ayush",
database="HCMS2"
)

# Patient Registration Function
def register_patient():
name = entry_name.get()
phone_no = entry_phone.get()
address = entry_address.get()
dob = entry_dob.get()
gender = gender_combobox.get()
username = entry_username.get()
password = entry_password.get()

# Hash the password
hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

try:
conn = connect_to_db()
cursor = conn.cursor()

# Insert into Patient table
cursor.execute(
"INSERT INTO Patient (Name, Phone_No, Address, Date_Of_Birth, Gender) "
"VALUES (%s, %s, %s, %s, %s)",
(name, phone_no, address, dob, gender)
)
patient_id = cursor.lastrowid  # Get the newly generated PatientID

# Insert into patient_pass table
cursor.execute(
"INSERT INTO patient_pass (username, password_hash, PatientID) "
"VALUES (%s, %s, %s)",
(username, hashed_password, patient_id)
)

# Commit changes
conn.commit()
```

```
    messagebox.showinfo("Success", "Patient registered successfully!")

    except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Database error: {err}")
    finally:
    cursor.close()
    conn.close()

    # Doctor Registration Function
    def register_doctor():
    name = entry_doctor_name.get()
    specialty = entry_specialty.get()
    phone_no = entry_doctor_phone.get()
    hospital = entry_hospital.get()
    username = entry_doctor_username.get()
    password = entry_doctor_password.get()

    # Hash the password
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    try:
    conn = connect_to_db()
    cursor = conn.cursor()

    # Insert into Doctor table
    cursor.execute(
    "INSERT INTO Doctor (Name, Specialty, Phone_No, Hospital) "
    "VALUES (%s, %s, %s, %s)",
    (name, specialty, phone_no, hospital)
    )
    doctor_id = cursor.lastrowid  # Get the newly generated DoctorID

    # Insert into doctor_pass table
    cursor.execute(
    "INSERT INTO doctor_pass (username, password_hash, DoctorID) "
    "VALUES (%s, %s, %s)",
    (username, hashed_password, doctor_id)
    )

    # Commit changes
    conn.commit()
    messagebox.showinfo("Success", "Doctor registered successfully!")

    except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Database error: {err}")
    finally:
    cursor.close()
    conn.close()

    # Patient Login Function
    def patient_login():
    username = entry_patient_login_username.get()
    password = entry_patient_login_password.get()

    try:
    conn = connect_to_db()
    cursor = conn.cursor()

    # Retrieve password hash from patient_pass table
    cursor.execute("SELECT password_hash FROM patient_pass WHERE username = %s", (username,))
    result = cursor.fetchone()

    if result and bcrypt.checkpw(password.encode('utf-8'), result[0].encode('utf-8')):
    messagebox.showinfo("Success", "Login successful!")
    else:
    messagebox.showerror("Error", "Invalid username or password.")

    except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Database error: {err}")
    finally:
    cursor.close()
    conn.close()

    # Doctor Login Function
    def doctor_login():
```

```
    username = entry_doctor_login_username.get()
    password = entry_doctor_login_password.get()

    try:
    conn = connect_to_db()
    cursor = conn.cursor()

    # Retrieve password hash from doctor_pass table
    cursor.execute("SELECT password_hash FROM doctor_pass WHERE username = %s", (username,))
    result = cursor.fetchone()

    if result and bcrypt.checkpw(password.encode('utf-8'), result[0].encode('utf-8')):
    messagebox.showinfo("Success", "Login successful!")
    else:
    messagebox.showerror("Error", "Invalid username or password.")

    except mysql.connector.Error as err:
    messagebox.showerror("Error", f"Database error: {err}")
    finally:
    cursor.close()
    conn.close()

    # Patient Registration GUI
    def patient_register_gui():
    global entry_name, entry_phone, entry_address, entry_dob, gender_combobox, entry_username,
    entry_password

    register_window = tk.Toplevel()
    register_window.title("Patient Registration")

    tk.Label(register_window, text="Name:").grid(row=0, column=0, sticky="w")
    entry_name = tk.Entry(register_window)
    entry_name.grid(row=0, column=1)

    tk.Label(register_window, text="Phone Number:").grid(row=1, column=0, sticky="w")
    entry_phone = tk.Entry(register_window)
    entry_phone.grid(row=1, column=1)

    tk.Label(register_window, text="Address:").grid(row=2, column=0, sticky="w")
    entry_address = tk.Entry(register_window)
    entry_address.grid(row=2, column=1)

    tk.Label(register_window, text="Date of Birth (YYYY-MM-DD):").grid(row=3, column=0, sticky="w
    ")
    entry_dob = tk.Entry(register_window)
    entry_dob.grid(row=3, column=1)

    tk.Label(register_window, text="Gender:").grid(row=4, column=0, sticky="w")
    genders = ['M', 'F', 'O']
    gender_combobox = ttk.Combobox(register_window, values=genders)
    gender_combobox.grid(row=4, column=1)

    tk.Label(register_window, text="Username:").grid(row=5, column=0, sticky="w")
    entry_username = tk.Entry(register_window)
    entry_username.grid(row=5, column=1)

    tk.Label(register_window, text="Password:").grid(row=6, column=0, sticky="w")
    entry_password = tk.Entry(register_window, show="*")
    entry_password.grid(row=6, column=1)

    register_button = tk.Button(register_window, text="Register", command=register_patient)
    register_button.grid(row=7, column=1, pady=10)

    # Doctor Registration GUI
    def doctor_register_gui():
    global entry_doctor_name, entry_specialty, entry_doctor_phone, entry_hospital,
    entry_doctor_username, entry_doctor_password

    register_window = tk.Toplevel()
    register_window.title("Doctor Registration")

    tk.Label(register_window, text="Name:").grid(row=0, column=0, sticky="w")
    entry_doctor_name = tk.Entry(register_window)
    entry_doctor_name.grid(row=0, column=1)
```

```
    tk.Label(register_window, text="Specialty:").grid(row=1, column=0, sticky="w")
    entry_specialty = tk.Entry(register_window)
    entry_specialty.grid(row=1, column=1)

    tk.Label(register_window, text="Phone Number:").grid(row=2, column=0, sticky="w")
    entry_doctor_phone = tk.Entry(register_window)
    entry_doctor_phone.grid(row=2, column=1)

    tk.Label(register_window, text="Hospital:").grid(row=3, column=0, sticky="w")
    entry_hospital = tk.Entry(register_window)
    entry_hospital.grid(row=3, column=1)

    tk.Label(register_window, text="Username:").grid(row=4, column=0, sticky="w")
    entry_doctor_username = tk.Entry(register_window)
    entry_doctor_username.grid(row=4, column=1)

    tk.Label(register_window, text="Password:").grid(row=5, column=0, sticky="w")
    entry_doctor_password = tk.Entry(register_window, show="*")
    entry_doctor_password.grid(row=5, column=1)

    register_button = tk.Button(register_window, text="Register", command=register_doctor)
    register_button.grid(row=6, column=1, pady=10)

    # Patient Login GUI
    def patient_login_gui():
    global entry_patient_login_username, entry_patient_login_password

    login_window = tk.Toplevel()
    login_window.title("Patient Login")

    tk.Label(login_window, text="Username:").grid(row=0, column=0, sticky="w")
    entry_patient_login_username = tk.Entry(login_window)
    entry_patient_login_username.grid(row=0, column=1)

    tk.Label(login_window, text="Password:").grid(row=1, column=0, sticky="w")
    entry_patient_login_password = tk.Entry(login_window, show="*")
    entry_patient_login_password.grid(row=1, column=1)

    login_button = tk.Button(login_window, text="Login", command=patient_login)
    login_button.grid(row=2, column=1, pady=10)

    # Doctor Login GUI
    def doctor_login_gui():
    global entry_doctor_login_username, entry_doctor_login_password

    login_window = tk.Toplevel()
    login_window.title("Doctor Login")

    tk.Label(login_window, text="Username:").grid(row=0, column=0, sticky="w")
    entry_doctor_login_username = tk.Entry(login_window)
    entry_doctor_login_username.grid(row=0, column=1)

    tk.Label(login_window, text="Password:").grid(row=1, column=0, sticky="w")
    entry_doctor_login_password = tk.Entry(login_window, show="*")
    entry_doctor_login_password.grid(row=1, column=1)

    login_button = tk.Button(login_window, text="Login", command=doctor_login)
    login_button.grid(row=2, column=1, pady=10)

    # Main GUI with options
    def main_gui():
    root = tk.Tk()
    root.title("Healthcare Management System")
    root.geometry("300x250")

    title_label = tk.Label(root, text="Welcome to HCMS", font=("Helvetica", 16, "bold"))
    title_label.pack(pady=10)

    doctor_login_button = tk.Button(root, text="Login as Doctor", command=doctor_login_gui, bg="
    #4CAF50", fg="white", font=("Arial", 10, "bold"))
    doctor_login_button.pack(fill="x", pady=5)

    patient_login_button = tk.Button(root, text="Login as Patient", command=patient_login_gui, bg
    ="#4CAF50", fg="white", font=("Arial", 10, "bold"))
    patient_login_button.pack(fill="x", pady=5)
```

```
270
271     doctor_register_button = tk.Button(root, text="Register as Doctor", command=
        doctor_register_gui, bg="#2196F3", fg="white", font=("Arial", 10, "bold"))
272     doctor_register_button.pack(fill="x", pady=5)
273
274     patient_register_button = tk.Button(root, text="Register as Patient", command=
        patient_register_gui, bg="#2196F3", fg="white", font=("Arial", 10, "bold"))
275     patient_register_button.pack(fill="x", pady=5)
276
277     root.mainloop()
278
279     # Run the main GUI
280     main_gui()
281
282
```

SQL Code for Database Creation and Table Definitions

```sql

    -- 1. Doctor Table
    CREATE TABLE Doctor (
    DoctorID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Specialization VARCHAR(255) NOT NULL,
    Phone_No VARCHAR(20),
    Email VARCHAR(255),
    Consultation_Fee DECIMAL(10, 2) NOT NULL
    );

    -- 2. Patient Table
    CREATE TABLE Patient (
    PatientID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(255) NOT NULL,
    Phone_No VARCHAR(20),
    Address VARCHAR(255),
    Date_Of_Birth DATE,
    Gender CHAR(1) CHECK (Gender IN ('M', 'F', 'O'))
    );

    -- 3. Appointment Table
    CREATE TABLE Appointment (
    AppID INT PRIMARY KEY AUTO_INCREMENT,
    DoctorID INT,
    PatientID INT,
    Appointment_Date DATE NOT NULL,
    start_time TIME,
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID) ON DELETE CASCADE,
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID) ON DELETE CASCADE
    );

    -- 4. Prescription Table
    CREATE TABLE Prescription (
    AppID INT,
    PrID INT,
    Date DATE,
    PRIMARY KEY (AppID, PrID),
    FOREIGN KEY (AppID) REFERENCES Appointment(AppID) ON DELETE CASCADE
    );

    -- 5. AllLabTest Table
    CREATE TABLE AllLabTest (
    TestID INT PRIMARY KEY,
    Test_Name VARCHAR(255) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL
    );

    -- 6. Prescribed_Labtest Table
    CREATE TABLE Prescribed_Labtest (
    AppID INT,
    UTestID INT AUTO_INCREMENT,
    TestID INT,
    Date_of_Prescription DATE NOT NULL,
    Result VARCHAR(255),
    PRIMARY KEY (UTestID),
    FOREIGN KEY (AppID) REFERENCES Appointment(AppID) ON DELETE CASCADE,
    FOREIGN KEY (TestID) REFERENCES AllLabTest(TestID) ON DELETE CASCADE
    );

    -- 7. Availability Table
    CREATE TABLE Availability (
    AvailID INT AUTO_INCREMENT,
    DoctorID INT,
    Day ENUM('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday') NOT
    NULL,
    Start_Time TIME NOT NULL,
    End_Time TIME NOT NULL,
    Status ENUM('available', 'not_available') DEFAULT 'available',
    PRIMARY KEY (AvailID, DoctorID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID) ON DELETE CASCADE
    );

```

```
-- 8. Inventory Table
CREATE TABLE Inventory (
MedID INT PRIMARY KEY,
Manufacturer VARCHAR(255) NOT NULL,
Quantity INT NOT NULL,
Price DECIMAL(10, 2) NOT NULL
);

-- 9. Prescribed_Medicine Table
CREATE TABLE Prescribed_Medicine (
AppID INT,
MedID INT,
PrID INT,
Dosage VARCHAR(255),
Instructions VARCHAR(255),
PRIMARY KEY (AppID, MedID, PrID),
FOREIGN KEY (AppID, PrID) REFERENCES Prescription(AppID, PrID) ON DELETE CASCADE,
FOREIGN KEY (MedID) REFERENCES Inventory(MedID) ON DELETE CASCADE
);

-- 10. PatientLabTest Table (junction table for Patient and AllLabTest)
CREATE TABLE PatientLabTest (
PatientID INT,
TestID INT,
Booking_Date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
Status VARCHAR(50),
PRIMARY KEY (PatientID, TestID, Booking_Date),
FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
FOREIGN KEY (TestID) REFERENCES AllLabTest(TestID)
);

-- 11. Type Table (for listing unique blood and organ types)
CREATE TABLE Type (
TypeID INT PRIMARY KEY,
Type VARCHAR(20) NOT NULL
);

-- 12. Location Table (for listing unique hospital names or locations)
CREATE TABLE Location (
LocationID INT PRIMARY KEY,
HospitalName VARCHAR(50) NOT NULL
);

-- 13. Donation Table (junction table for Type and Location)
CREATE TABLE Donation (
TypeID INT,
LocationID INT,
PRIMARY KEY (TypeID, LocationID),
FOREIGN KEY (TypeID) REFERENCES Type(TypeID),
FOREIGN KEY (LocationID) REFERENCES Location(LocationID)
);

-- Doctor Login Table
CREATE TABLE doctor_pass (
DocLoginID INT PRIMARY KEY AUTO_INCREMENT,
username VARCHAR(50) NOT NULL UNIQUE,
password_hash VARCHAR(255) NOT NULL,
DoctorID INT,
FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID) ON DELETE CASCADE
);

-- Patient Login Table
CREATE TABLE patient_pass (
PatLoginID INT PRIMARY KEY AUTO_INCREMENT,
username VARCHAR(50) NOT NULL UNIQUE,
password_hash VARCHAR(255) NOT NULL,
PatientID INT,
FOREIGN KEY (PatientID) REFERENCES Patient(PatientID) ON DELETE CASCADE
);
```

SQL Code for Data Insertion

```sql

    INSERT INTO Doctor (DoctorID, Name, Specialization, Phone_No, Email, Consultation_Fee) VALUES
    (1, 'Dr. Alice Smith', 'Cardiologist', '1234567890', 'alice.smith@example.com', 150.00),
    (2, 'Dr. Bob Johnson', 'Dermatologist', '0987654321', 'bob.johnson@example.com', 120.00),
    (3, 'Dr. Carol Lee', 'Orthopedic', '1122334455', 'carol.lee@example.com', 100.00),
    (4, 'Dr. David Brown', 'Neurologist', '1223344556', 'david.brown@example.com', 200.00),
    (5, 'Dr. Emily Davis', 'Pediatrician', '1334455667', 'emily.davis@example.com', 110.00),
    (6, 'Dr. Frank Wilson', 'Psychiatrist', '1445566778', 'frank.wilson@example.com', 140.00),
    (7, 'Dr. Grace Miller', 'Cardiologist', '1556677889', 'grace.miller@example.com', 160.00),
    (8, 'Dr. Henry Martin', 'Dermatologist', '1667788990', 'henry.martin@example.com', 130.00),
    (9, 'Dr. Irene Harris', 'Orthopedic', '1778899001', 'irene.harris@example.com', 105.00),
    (10, 'Dr. Jack Clark', 'Radiologist', '1889900112', 'jack.clark@example.com', 125.00),
    (11, 'Dr. Karen Lewis', 'Ophthalmologist', '1990011223', 'karen.lewis@example.com', 145.00),
    (12, 'Dr. Leo Walker', 'Oncologist', '2001122334', 'leo.walker@example.com', 250.00),
    (13, 'Dr. Mary Hall', 'Endocrinologist', '2112233445', 'mary.hall@example.com', 170.00),
    (14, 'Dr. Nancy Young', 'Urologist', '2223344556', 'nancy.young@example.com', 115.00),
    (15, 'Dr. Oliver King', 'Gastroenterologist', '2334455667', 'oliver.king@example.com',
    135.00);

    -- Insert data into Patient table
    INSERT INTO Patient (PatientID, Name, Phone_No, Address, Date_Of_Birth, Gender) VALUES
    (1, 'John Doe', '2345678901', '123 Maple St', '1985-06-15', 'M'),
    (2, 'Jane Smith', '3456789012', '456 Oak Ave', '1990-08-20', 'F'),
    (3, 'Tom Brown', '4567890123', '789 Pine Blvd', '2000-01-01', 'M'),
    (4, 'Lucy White', '5678901234', '101 Cedar Rd', '1992-04-25', 'F'),
    (5, 'Anna Taylor', '6789012345', '202 Birch Ln', '1980-03-15', 'F'),
    (6, 'Mark Green', '7890123456', '303 Elm St', '1975-11-30', 'M'),
    (7, 'Emma Harris', '8901234567', '404 Walnut St', '1988-10-12', 'F'),
    (8, 'Chris Clark', '9012345678', '505 Spruce St', '2002-07-07', 'M'),
    (9, 'Olivia Lee', '0123456789', '606 Willow Rd', '1995-05-17', 'F'),
    (10, 'Ethan Walker', '1234567899', '707 Ash Ave', '1998-09-09', 'M'),
    (11, 'Sophia Hall', '2345678909', '808 Poplar St', '1993-02-02', 'F'),
    (12, 'Liam Young', '3456789019', '909 Redwood St', '2001-12-31', 'M'),
    (13, 'Mia King', '4567890129', '1010 Fir Dr', '1984-08-18', 'F'),
    (14, 'James Scott', '5678901239', '1111 Cypress Ct', '1979-06-21', 'M'),
    (15, 'Isabella Adams', '6789012349', '1212 Pine Cr', '2003-11-14', 'F');

    -- Insert data into Appointment table
    INSERT INTO Appointment (AppID, DoctorID, PatientID, Appointment_Date) VALUES
    (6, 6, 6, '2024-10-20'),
    (7, 7, 7, '2024-10-21'),
    (8, 8, 8, '2024-10-22'),
    (9, 9, 9, '2024-10-23'),
    (10, 10, 10, '2024-10-24'),
    (11, 11, 11, '2024-10-25'),
    (12, 12, 12, '2024-10-26'),
    (13, 13, 13, '2024-10-27'),
    (14, 14, 14, '2024-10-28'),
    (15, 15, 15, '2024-10-29');

    -- Insert data into Prescription table
    INSERT INTO Prescription (AppID, PrID, Date) VALUES
    (6, 2, '2024-10-20'),
    (7, 3, '2024-10-21'),
    (8, 3, '2024-10-22'),
    (9, 3, '2024-10-23'),
    (10, 4, '2024-10-24'),
    (11, 4, '2024-10-25'),
    (12, 4, '2024-10-26'),
    (13, 5, '2024-10-27');

    -- Insert data into AllLabTest table
    INSERT INTO AllLabTest (TestID, Test_Name, Price) VALUES
    (4, 'Urine Test', 40.00),
    (5, 'CT Scan', 500.00),
    (6, 'ECG', 100.00),
    (7, 'Liver Function Test', 60.00),
    (8, 'Thyroid Profile', 80.00),
    (9, 'Lipid Profile', 90.00),
    (10, 'Ultrasound', 120.00),
    (11, 'Blood Sugar', 30.00),
    (12, 'Allergy Test', 150.00),
    (13, 'Vitamin D Test', 70.00);
```

```
73
74  -- Insert data into Prescribed_Labtest table
75  INSERT INTO Prescribed_Labtest (AppID, TestID, Date_of_Prescription, Result) VALUES
76  (6, 6, '2024-10-20', 'Irregularities Detected'),
77  (7, 7, '2024-10-21', 'Normal'),
78  (8, 8, '2024-10-22', 'High Levels Detected'),
79  (9, 9, '2024-10-23', 'Stable'),
80  (10, 10, '2024-10-24', 'Normal'),
81  (11, 11, '2024-10-25', 'Within Range'),
82  (12, 12, '2024-10-26', 'Allergy Detected'),
83  (13, 13, '2024-10-27', 'Deficiency Detected');
84
85  -- Insert data into Availability table
86  INSERT INTO Availability (DoctorID, Day, Start_Time, End_Time, Status) VALUES
87  (1, 'Tuesday', '09:00:00', '13:00:00', 'available'),
88  (2, 'Thursday', '14:00:00', '18:00:00', 'not_available'),
89  (3, 'Monday', '08:00:00', '12:00:00', 'available'),
90  (4, 'Saturday', '10:00:00', '14:00:00', 'available'),
91  (5, 'Sunday', '11:00:00', '15:00:00', 'not_available'),
92  (6, 'Wednesday', '12:00:00', '16:00:00', 'available'),
93  (7, 'Friday', '09:00:00', '11:00:00', 'not_available'),
94  (8, 'Monday', '10:00:00', '13:00:00', 'available'),
95  (9, 'Tuesday', '11:00:00', '14:00:00', 'available'),
96  (10, 'Thursday', '08:00:00', '12:00:00', 'available');
97
98  -- Insert data into Inventory table
99  INSERT INTO Inventory (MedID, Manufacturer, Quantity, Price) VALUES
100 (4, 'Medicorp', 250, 12.00),
101 (5, 'GoodHealth Pharma', 300, 18.00),
102 (6, 'PureMed Ltd.', 180, 22.00),
103 (7, 'HealthMeds', 220, 25.00),
104 (8, 'Vitalife', 500, 30.00),
105 (9, 'MediDirect', 150, 35.00),
106 (10, 'GlobalCare', 400, 28.00),
107 (11, 'HealthSolutions', 350, 26.00),
108 (12, 'LifeMeds Inc.', 210, 32.00),
109 (13, 'WellnessPharma', 175, 24.00);
110
111 -- Insert data into Prescribed_Medicine table
112 INSERT INTO Prescribed_Medicine (AppID, MedID, PrID, Dosage, Instructions) VALUES
113 (6, 6, 2, '1 capsule', 'Before sleep'),
114 (7, 7, 3, '2 tablets', 'Morning and Evening'),
115 (8, 8, 3, '5ml syrup', 'Once daily'),
116 (9, 9, 3, '1 tablet', 'With meals'),
117 (10, 10, 4, '1 injection', 'Administer every 3 days'),
118 (11, 11, 4, '3 drops', 'Apply to affected area'),
119 (12, 12, 4, '1 tablet', 'Daily before breakfast'),
120 (13, 13, 5, '2 tablets', 'After evening meal');
121
122 -- Insert data into PatientLabTest table
123 INSERT INTO PatientLabTest (PatientID, TestID, Booking_Date, Status) VALUES
124 (4, 4, '2024-10-18 10:30:00', 'Completed'),
125 (5, 5, '2024-10-19 11:00:00', 'Pending'),
126 (6, 6, '2024-10-20 09:45:00', 'In Progress'),
127 (7, 7, '2024-10-21 12:00:00', 'Completed'),
128 (8, 8, '2024-10-22 13:30:00', 'Pending'),
129 (9, 9, '2024-10-23 14:45:00', 'Completed'),
130 (10, 10, '2024-10-24 15:15:00', 'In Progress'),
131 (11, 11, '2024-10-25 16:00:00', 'Pending'),
132 (12, 12, '2024-10-26 16:45:00', 'Completed'),
133 (13, 13, '2024-10-27 17:30:00', 'Pending');
134
135 -- Insert data into Type table
136 INSERT INTO Type (TypeID, Type) VALUES
137 (4, 'Blood Type B+'),
138 (5, 'Blood Type AB+'),
139 (6, 'Organ Liver'),
140 (7, 'Blood Type A-'),
141 (8, 'Blood Type B-'),
142 (9, 'Organ Heart'),
143 (10, 'Organ Lung'),
144 (11, 'Blood Type O-'),
145 (12, 'Blood Type AB-'),
146 (13, 'Organ Eye');
147
```

```
    -- Insert data into Location table
    INSERT INTO Location (LocationID, HospitalName) VALUES
    (4, 'Green Valley Clinic'),
    (5, 'Westside Medical Center'),
    (6, 'Easttown Hospital'),
    (7, 'Downtown Health Center'),
    (8, 'Northern Lights Clinic'),
    (9, 'Southside Medical Plaza'),
    (10, 'Lakeside Hospital'),
    (11, 'Hilltop Health Facility'),
    (12, 'Park Avenue Clinic'),
    (13, 'Sunrise Health Clinic');

    -- Insert data into Donation table
    INSERT INTO Donation (TypeID, LocationID) VALUES
    (4, 4),
    (5, 5),
    (6, 6),
    (7, 7),
    (8, 8),
    (9, 9),
    (10, 10),
    (11, 11),
    (12, 12),
    (13, 13);

```

Queries

Appointment Insertion Query for Healthcare Database

```
1
2     -- Assume the same input values for demonstration
3     SET @PatientID = 1;                                    -- Replace with the patient's ID
4     SET @DoctorSpecialization = 'Cardiologist' COLLATE utf8mb4_unicode_ci;  -- Replace with
      doctor specialization
5     SET @StartTime = '09:00:00';                           -- Replace with desired
      appointment start time
6     SET @AppointmentDate = '2024-11-01';                   -- Replace with desired
      appointment date
7
8     -- Step 1: Find an available doctor with the given specialization
9     -- Reset @DoctorID to NULL before attempting to set it
10    SET @DoctorID = NULL;
11
12    -- Select a doctor who matches the specialization and is available at the specified time
13    SELECT d.DoctorID INTO @DoctorID
14    FROM Doctor d
15    JOIN Availability a ON d.DoctorID = a.DoctorID
16    WHERE d.Specialization COLLATE utf8mb4_unicode_ci = @DoctorSpecialization
17    AND a.Start_Time <= @StartTime
18    AND a.End_Time > @StartTime
19    AND a.Status = 'available'
20    LIMIT 1;
21
22    INSERT INTO Appointment (AppID, DoctorID, PatientID, Appointment_Date, start_time)
23    VALUES (16, @DoctorID, @PatientID, @AppointmentDate, @StartTime);
24
```

Before booking appointment

	AppID	DoctorID	PatientID	Appointment_Date	Start_Time
Edit Copy Delete	6	6	6	2024-10-20	NULL
Edit Copy Delete	7	7	7	2024-10-21	NULL
Edit Copy Delete	8	8	8	2024-10-22	NULL
Edit Copy Delete	9	9	9	2024-10-23	NULL
Edit Copy Delete	10	10	10	2024-10-24	NULL
Edit Copy Delete	11	11	11	2024-10-25	NULL
Edit Copy Delete	12	12	12	2024-10-26	NULL
Edit Copy Delete	13	13	13	2024-10-27	NULL
Edit Copy Delete	14	14	14	2024-10-28	NULL
Edit Copy Delete	15	15	15	2024-10-29	NULL

After booking appointment

	AppID	DoctorID	PatientID	Appointment_Date	Start_Time
Edit Copy Delete	6	6	6	2024-10-20	NULL
Edit Copy Delete	7	7	7	2024-10-21	NULL
Edit Copy Delete	8	8	8	2024-10-22	NULL
Edit Copy Delete	9	9	9	2024-10-23	NULL
Edit Copy Delete	10	10	10	2024-10-24	NULL
Edit Copy Delete	11	11	11	2024-10-25	NULL
Edit Copy Delete	12	12	12	2024-10-26	NULL
Edit Copy Delete	13	13	13	2024-10-27	NULL
Edit Copy Delete	14	14	14	2024-10-28	NULL
Edit Copy Delete	15	15	15	2024-10-29	NULL
Edit Copy Delete	16	1	1	2024-11-01	09:00:00

Query to generate bill for an appointment

```
1
2     SELECT
3     p.PatientID,
4     app.AppID,
5     d.Name AS Doctor_Name,
6     p.Name AS Patient_Name,
7     COALESCE(SUM(i.Price), 0) AS Total_Medicine_Cost,
8     COALESCE(SUM(alt.Price), 0) AS Total_LabTest_Cost,
9     COALESCE(SUM(i.Price), 0) + COALESCE(SUM(alt.Price), 0) AS Total_Price
10    FROM
```

```
    Appointment app
    JOIN
    Doctor d ON app.DoctorID = d.DoctorID
    JOIN
    Patient p ON app.PatientID = p.PatientID
    LEFT JOIN
    Prescription pr ON app.AppID = pr.AppID
    LEFT JOIN
    Prescribed_Medicine pm ON pr.AppID = pm.AppID AND pr.PrID = pm.PrID
    LEFT JOIN
    Inventory i ON pm.MedID = i.MedID
    LEFT JOIN
    Prescribed_Labtest plt ON app.AppID = plt.AppID
    LEFT JOIN
    AllLabTest alt ON plt.TestID = alt.TestID
    GROUP BY
    p.PatientID, app.AppID, d.Name, p.Name;

```

PatientID	AppID	Doctor_Name	Patient_Name	Total_Medicine_Cost	Total_LabTest_Cost	Total_Price
6	6	Dr. Frank Wilson	Mark Green	22.00	100.00	122.00
7	7	Dr. Grace Miller	Emma Harris	25.00	60.00	85.00
8	8	Dr. Henry Martin	Chris Clark	30.00	80.00	110.00
9	9	Dr. Irene Harris	Olivia Lee	35.00	90.00	125.00
10	10	Dr. Jack Clark	Ethan Walker	28.00	120.00	148.00
11	11	Dr. Karen Lewis	Sophia Hall	26.00	30.00	56.00
12	12	Dr. Leo Walker	Liam Young	32.00	150.00	182.00
13	13	Dr. Mary Hall	Mia King	24.00	70.00	94.00
14	14	Dr. Nancy Young	James Scott	0.00	0.00	0.00
15	15	Dr. Oliver King	Isabella Adams	0.00	0.00	0.00

Doctors Who Prescribe the Most Expensive Medicines on Average

```

    SELECT
    d.DoctorID,
    d.Name AS Doctor_Name,
    AVG(i.Price) AS Avg_Medicine_Price
    FROM
    Doctor d
    JOIN
    Appointment app ON d.DoctorID = app.DoctorID
    JOIN
    Prescription pr ON app.AppID = pr.AppID
    JOIN
    Prescribed_Medicine pm ON pr.AppID = pm.AppID AND pr.PrID = pm.PrID
    JOIN
    Inventory i ON pm.MedID = i.MedID
    GROUP BY
    d.DoctorID
    ORDER BY
    Avg_Medicine_Price DESC
    LIMIT 10;

```

DoctorID	Doctor_Name	Avg_Medicine_Price 1
9	Dr. Irene Harris	35.000000
12	Dr. Leo Walker	32.000000
8	Dr. Henry Martin	30.000000
10	Dr. Jack Clark	28.000000
11	Dr. Karen Lewis	26.000000
7	Dr. Grace Miller	25.000000
13	Dr. Mary Hall	24.000000
6	Dr. Frank Wilson	22.000000

Generate a Report on Donation Availability per Blood or Organ Type

```
1
2     SELECT
3     t.Type AS Blood_Or_Organ_Type,
4     l.HospitalName AS Location,
5     IFNULL(COUNT(d.TypeID), 0) AS Donation_Count
6     FROM
7     Type t
8     LEFT JOIN
9     Donation d ON t.TypeID = d.TypeID
10    LEFT JOIN
11    Location l ON d.LocationID = l.LocationID
12    GROUP BY
13    t.Type, l.HospitalName
14    ORDER BY
15    Donation_Count DESC;
16
```

Blood_Or_Organ_Type	Location	Donation_Count 1
Blood Type B+	Green Valley Clinic	1
Blood Type O-	Hilltop Health Facility	1
Blood Type AB+	Westside Medical Center	1
Blood Type AB-	Park Avenue Clinic	1
Organ Liver	Easttown Hospital	1
Organ Eye	Sunrise Health Clinic	1
Blood Type A-	Downtown Health Center	1
Blood Type B-	Northern Lights Clinic	1
Organ Heart	Southside Medical Plaza	1
Organ Lung	Lakeside Hospital	1

Calculate the Total Revenue from Consultation Fees by Doctor and Specialization

```
1
2    SELECT
3    d.Specialization,
4    d.Name AS Doctor_Name,
5    SUM(d.Consultation_Fee) AS Total_Revenue
6    FROM
7    Doctor d
8    JOIN
9    Appointment app ON d.DoctorID = app.DoctorID
10   GROUP BY
11   d.DoctorID, d.Specialization
12   ORDER BY
13   Total_Revenue DESC;
14
```

Specialization	Doctor_Name	Total_Revenue ▾ 1
Oncologist	Dr. Leo Walker	250.00
Endocrinologist	Dr. Mary Hall	170.00
Cardiologist	Dr. Grace Miller	160.00
Ophthalmologist	Dr. Karen Lewis	145.00
Psychiatrist	Dr. Frank Wilson	140.00
Gastroenterologist	Dr. Oliver King	135.00
Dermatologist	Dr. Henry Martin	130.00
Radiologist	Dr. Jack Clark	125.00
Urologist	Dr. Nancy Young	115.00
Orthopedic	Dr. Irene Harris	105.00

Some other basic queries

```sql

-- 1. List all doctors along with their specialization and consultation fee.
SELECT DoctorID, Name AS Doctor_Name, Specialization, Consultation_Fee
FROM Doctor;

-- 2. Retrieve all patients who are male.
SELECT PatientID, Name AS Patient_Name, Gender
FROM Patient
WHERE Gender = 'M';

-- 3. Find all appointments scheduled for a specific date, e.g., '2024-10-29'.
SELECT AppID, DoctorID, PatientID, Appointment_Date
FROM Appointment
WHERE Appointment_Date = '2024-10-29';

-- 4. Get the total number of patients in the system.
SELECT COUNT(*) AS Total_Patients
FROM Patient;

-- 5. Show the names of doctors and their available days and times.
SELECT d.Name AS Doctor_Name, a.Day, a.Start_Time, a.End_Time
FROM Doctor d
JOIN Availability a ON d.DoctorID = a.DoctorID;

-- 6. List all lab tests along with their prices.
SELECT TestID, Test_Name, Price
FROM AllLabTest;

-- 7. Find the number of appointments each doctor has.
SELECT d.DoctorID, d.Name AS Doctor_Name, COUNT(app.AppID) AS Appointment_Count
FROM Doctor d
LEFT JOIN Appointment app ON d.DoctorID = app.DoctorID
GROUP BY d.DoctorID, d.Name;

-- 8. Retrieve the names and quantities of all medicines in inventory.
SELECT MedID, Manufacturer, Quantity
FROM Inventory;

-- 9. List all prescriptions associated with a specific appointment, e.g., AppID = 1.
SELECT AppID, PrID, Date
FROM Prescription
WHERE AppID = 1;

-- 10. Show the total price of all lab tests prescribed for a specific appointment, e.g.,
AppID = 2.
SELECT SUM(alt.Price) AS Total_LabTest_Cost
FROM Prescribed_Labtest plt
JOIN AllLabTest alt ON plt.TestID = alt.TestID
WHERE plt.AppID = 2;

```

Output Query 1

	DoctorID	Doctor_Name	Specialization	Consultation_Fee
Edit Copy Delete	1	Dr. Alice Smith	Cardiologist	150.00
Edit Copy Delete	2	Dr. Bob Johnson	Dermatologist	120.00
Edit Copy Delete	3	Dr. Carol Lee	Orthopedic	100.00
Edit Copy Delete	4	Dr. David Brown	Neurologist	200.00
Edit Copy Delete	5	Dr. Emily Davis	Pediatrician	110.00
Edit Copy Delete	6	Dr. Frank Wilson	Psychiatrist	140.00
Edit Copy Delete	7	Dr. Grace Miller	Cardiologist	160.00
Edit Copy Delete	8	Dr. Henry Martin	Dermatologist	130.00
Edit Copy Delete	9	Dr. Irene Harris	Orthopedic	105.00
Edit Copy Delete	10	Dr. Jack Clark	Radiologist	125.00
Edit Copy Delete	11	Dr. Karen Lewis	Ophthalmologist	145.00
Edit Copy Delete	12	Dr. Leo Walker	Oncologist	250.00
Edit Copy Delete	13	Dr. Mary Hall	Endocrinologist	170.00
Edit Copy Delete	14	Dr. Nancy Young	Urologist	115.00
Edit Copy Delete	15	Dr. Oliver King	Gastroenterologist	135.00

Output Query 2

	PatientID	Patient_Name	Gender
Edit Copy Delete	1	John Doe	M
Edit Copy Delete	3	Tom Brown	M
Edit Copy Delete	6	Mark Green	M
Edit Copy Delete	8	Chris Clark	M
Edit Copy Delete	10	Ethan Walker	M
Edit Copy Delete	12	Liam Young	M
Edit Copy Delete	14	James Scott	M

Output Query 3

	AppID	DoctorID	PatientID	Appointment_Date
Edit Copy Delete	15	15	15	2024-10-29

Output Query 4

Total_Patients
15

Output Query 5

Doctor_Name	Day	Start_Time	End_Time
Dr. Alice Smith	Tuesday	09:00:00	13:00:00
Dr. Bob Johnson	Thursday	14:00:00	18:00:00
Dr. Carol Lee	Monday	08:00:00	12:00:00
Dr. David Brown	Saturday	10:00:00	14:00:00
Dr. Emily Davis	Sunday	11:00:00	15:00:00
Dr. Frank Wilson	Wednesday	12:00:00	16:00:00
Dr. Grace Miller	Friday	09:00:00	11:00:00
Dr. Henry Martin	Monday	10:00:00	13:00:00
Dr. Irene Harris	Tuesday	11:00:00	14:00:00
Dr. Jack Clark	Thursday	08:00:00	12:00:00

Output Query 6

←T→	TestID	Test_Name	Price
Edit Copy Delete	4	Urine Test	40.00
Edit Copy Delete	5	CT Scan	500.00
Edit Copy Delete	6	ECG	100.00
Edit Copy Delete	7	Liver Function Test	60.00
Edit Copy Delete	8	Thyroid Profile	80.00
Edit Copy Delete	9	Lipid Profile	90.00
Edit Copy Delete	10	Ultrasound	120.00
Edit Copy Delete	11	Blood Sugar	30.00
Edit Copy Delete	12	Allergy Test	150.00
Edit Copy Delete	13	Vitamin D Test	70.00

Output Query 7

DoctorID	Doctor_Name	Appointment_Count
1	Dr. Alice Smith	0
2	Dr. Bob Johnson	0
3	Dr. Carol Lee	0
4	Dr. David Brown	0
5	Dr. Emily Davis	0
6	Dr. Frank Wilson	1
7	Dr. Grace Miller	1
8	Dr. Henry Martin	1
9	Dr. Irene Harris	1
10	Dr. Jack Clark	1
11	Dr. Karen Lewis	1
12	Dr. Leo Walker	1
13	Dr. Mary Hall	1
14	Dr. Nancy Young	1
15	Dr. Oliver King	1

Output Query 8

	MedID	Manufacturer	Quantity
Edit Copy Delete	4	Medicorp	250
Edit Copy Delete	5	GoodHealth Pharma	300
Edit Copy Delete	6	PureMed Ltd.	180
Edit Copy Delete	7	HealthMeds	220
Edit Copy Delete	8	Vitalife	500
Edit Copy Delete	9	MediDirect	150
Edit Copy Delete	10	GlobalCare	400
Edit Copy Delete	11	HealthSolutions	350
Edit Copy Delete	12	LifeMeds Inc.	210
Edit Copy Delete	13	WellnessPharma	175

Output Query 9

AppID	PrID	Date

Output Query 10

Total_LabTest_Cost
NULL

Conclusion

Example. The healthcare management system developed in this project is a comprehensive solution designed to streamline operations within a healthcare setting. By integrating essential functionalities like appointment scheduling, lab test management, inventory tracking, and secure login and registration features, the system addresses critical challenges in patient care and administrative efficiency. The structured relational database, built on a carefully designed ER model, supports the seamless handling of sensitive data and complex relationships between doctors, patients, appointments, and prescriptions.

The login and registration functionality strengthens data privacy and provides a personalized experience for both doctors and patients, ensuring that only authorized users can access or manage information. Doctors can efficiently manage their schedules, update their availability, and access patient histories, while patients have the convenience of booking appointments, reviewing test results, and managing their healthcare records.

Overall, this project not only simplifies administrative tasks but also enhances the patient-doctor interaction by providing timely access to relevant information. The system's scalable and secure design makes it adaptable for future enhancements, such as integrating telemedicine or expanding with new modules. This healthcare management solution lays a solid foundation for modern, digitalized patient care, offering an efficient, secure, and user-friendly platform suitable for real-world healthcare environments.