Health Track - CS 4347.001

Aditya Gavali, Ananiya Shenkut, Kian Hakim

Professor: Dr. Xinda Wang

04/30/2024

Table of Contents

Introduction

In our project, "Health Track," we address the significant challenge of fragmented patient health records within the healthcare system. Currently, crucial patient data is often dispersed across various institutions, making it challenging for healthcare professionals to access complete and up-to-date information. This fragmentation can lead to inefficiencies, a higher risk of errors, and ultimately, compromised patient care. Our solution is a centralized database system designed to provide secure, instant access to patient records, aiming to enhance the quality and continuity of care.

The organization of this report reflects the comprehensive approach we've taken to develop "Health Track." We begin by detailing the problem statement, which outlines the issues with current record-keeping systems and the necessity for our solution. We then introduce our target users, primarily healthcare providers and administrative staff, who require reliable access to detailed patient histories and efficient management tools. Subsequent sections will discuss our system's architecture, including the database schema and the web interfaces for both patients and healthcare professionals. This structure ensures that the report is not only informative but also provides a clear roadmap of our project's development and intended impact.

System Requirements

- **Central Database**: Stores patient records, medical records, doctor's information, and appointment details.
- **Web Interface**: Divided into two sections, the Patient Dashboard and Doctor/Staff Portal, allowing interaction with the database for different user roles.
- **Security Layer**: Ensures secure access and data protection, crucial for a healthcare database.

Interface Requirements
Web Interface:

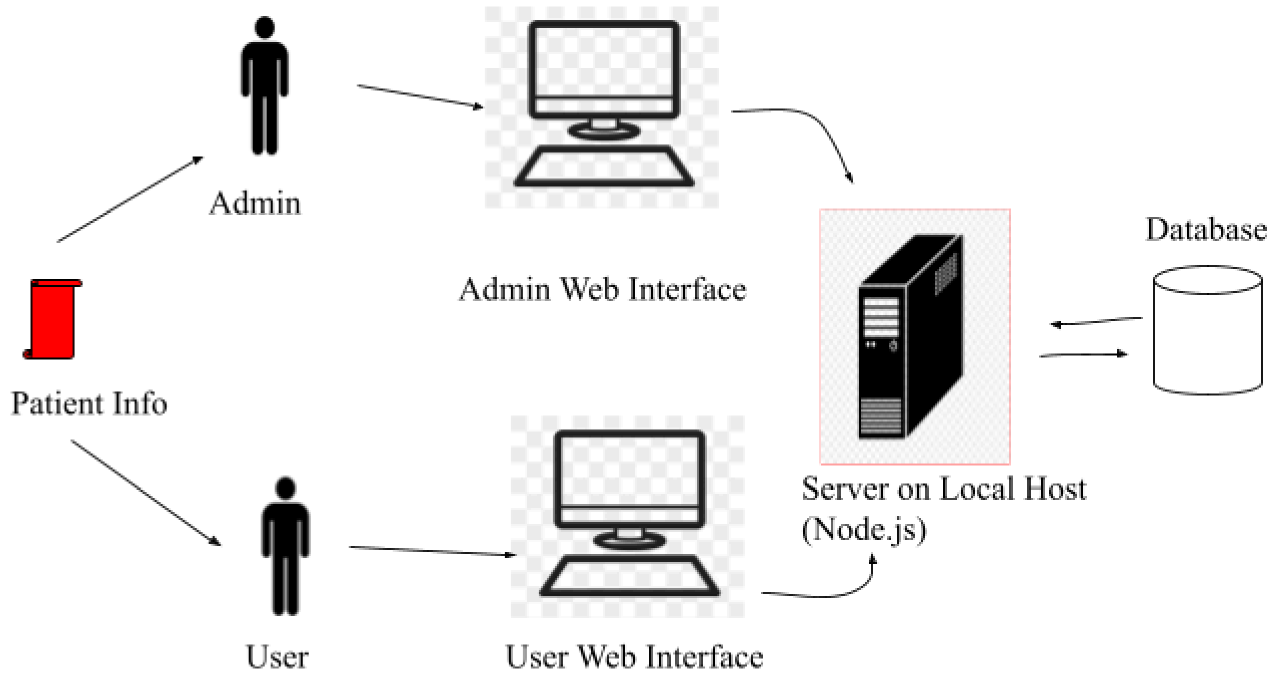
- **Patients Dashboard**: Allows patients to view and manage their medical records, appointments, and personal information.
- **Doctor/Staff Portal**: Allows healthcare providers to access and manage patient information, add notes, and handle appointments.

Functional Requirements

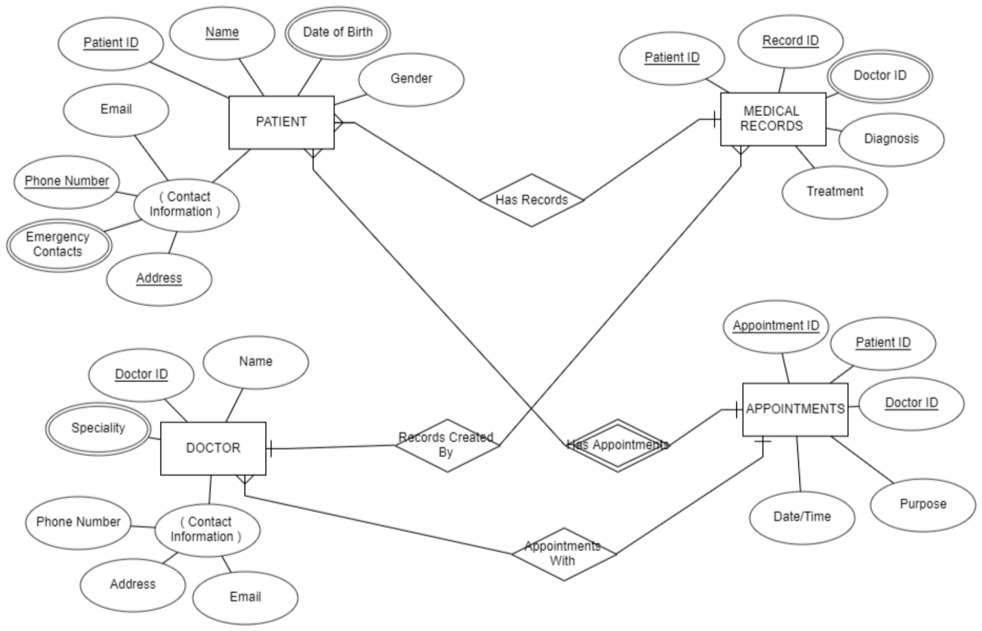
1. **Data Management**:
    - Insert, update, and delete records for patients, doctors, medical records, and appointments.
    - Auto-increment primary keys where applicable.
    - Enforce foreign key relationships with actions like "DELETE CASCADE" or "SET NULL".
2. **Search and Retrieval**:
    - Enable searching for patient histories, doctor information, and appointment details.
3. **User Authentication and Authorization**:
    - Secure login for patients and staff.
    - Role-based access control to ensure users can only access data pertinent to their roles.
4. **Reporting**:
    - Generate reports on patient histories, appointment schedules, and treatment outcomes.

Non-Functional Requirements

1. **Security**:
    - Implement standard security measures to protect sensitive medical data.
    - Compliance with healthcare regulations like HIPAA for data protection.
2. **Performance**:
    - The system should handle multiple user requests efficiently without significant delays.
3. **Scalability**:
    - The database should be capable of handling an increasing amount of data as the number of users grows.
4. **Usability**:
    - User interfaces must be intuitive and user-friendly for both patients and healthcare providers.
5. **Reliability**:
    - The system should have high availability and fault tolerance to minimize downtime.


Admin
Admin Web Interface
Database
Patient Info
Server on Local Host (Node.js)
User
User Web Interface

Conceptual Design of the Database



Relationships:

- Patients to Doctors: Many-to-many, realized through the Appointments table. A patient can have appointments with multiple doctors, and a doctor can see multiple patients.
- Patients to Medical Records: One-to-many. A patient can have multiple medical records but each medical record is associated with one patient.
- Doctors to Appointments: One-to-many. A doctor can have multiple appointments but each appointment is associated with one doctor.

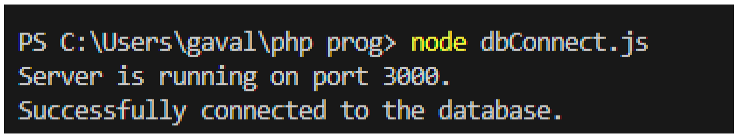
Data Dictionary and Business Rules

1. Primary and Foreign Key Constraints: Enforce uniqueness and referential integrity.
2. Auto-increment on Primary Keys where applicable to ensure unique identifiers are generated automatically.
3. Not Null Constraints on crucial fields to ensure data completeness.
4. Foreign Key Actions like "DELETE CASCADE" ensure that when a patient is deleted, all associated medical records and appointments are also removed, maintaining data consistency.
5. Nullable fields such as gender or email allow flexibility in data entry, acknowledging that not all information may be available at the time of record creation.

The Database System

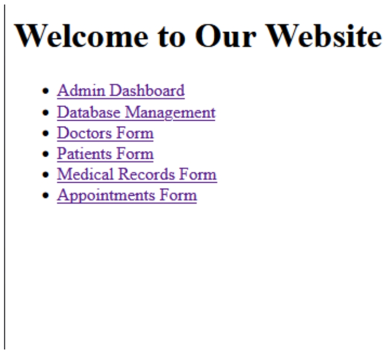
In order to run our database, you need to have MySQL and Visual Studio Code with HTML and Node installed in your VS Code. We used MySQL to allow users to be able to store the data they would like to enter. For the backend, we used Node.js so that we connect the database from MySQL to HTML. This allows us to then fetch the data which allows the admin to be able to create, read, update, or even delete data. For the frontend, we used HTML. We've created 8 different HTML files, patients, doctors, appointments, medicalrecords, admin, adminlogin, and index in order to allow the user to actually utilize the features.

This is how we connect the database:



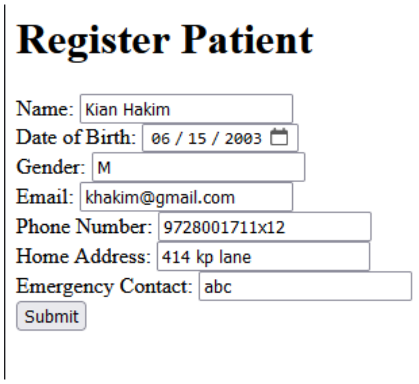
```
PS C:\Users\gaval\php prog> node dbConnect.js
Server is running on port 3000.
Successfully connected to the database.
```

This what is shown in the localhost:



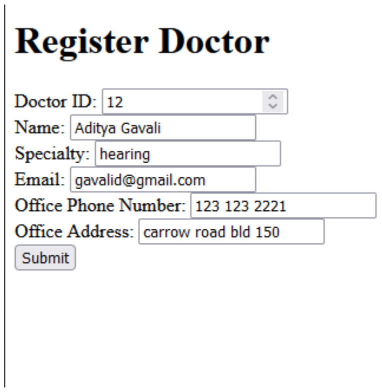
Welcome to Our Website

- Admin Dashboard
- Database Management
- Doctors Form
- Patients Form
- Medical Records Form
- Appointments Form



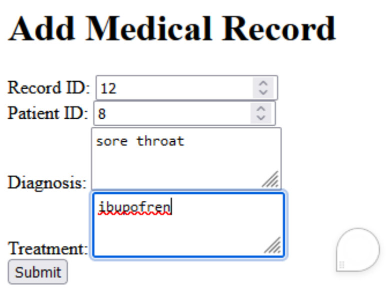
Register Patient

Name: Kian Hakim
Date of Birth: 06 / 15 / 2003
Gender: M
Email: khakim@gmail.com
Phone Number: 9728001711x12
Home Address: 414 kp lane
Emergency Contact: abc



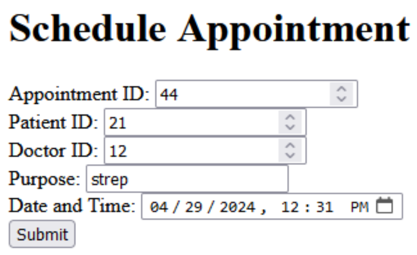
Register Doctor

Doctor ID: 12
Name: Aditya Gavali
Specialty: hearing
Email: gavalid@gmail.com
Office Phone Number: 123 123 2221
Office Address: carrow road bld 150
Submit



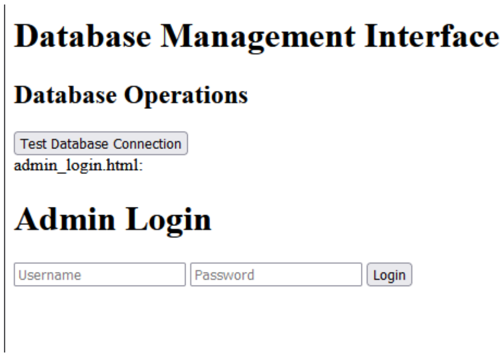
Add Medical Record

Record ID: 12
Patient ID: 8
Diagnosis: sore throat
Treatment: ibupofren
Submit



Schedule Appointment

Appointment ID: 44
Patient ID: 21
Doctor ID: 12
Purpose: strep
Date and Time: 04 / 29 / 2024 , 12 : 31 PM
Submit



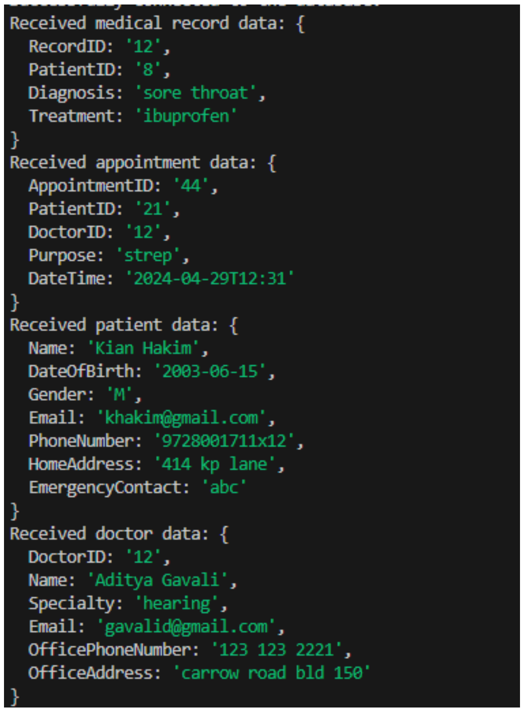
Database Management Interface

Database Operations

admin_login.html:

Admin Login

What this first shows is the actual home page, known as the index. It allows the user to input data wherever they need. In the 4 following images, this is how the user can input the data that needs to be inputted. Then if needed, the admin can then login and update any data that they need to update.



```
Received medical record data: {
  RecordID: '12',
  PatientID: '8',
  Diagnosis: 'sore throat',
  Treatment: 'ibuprofen'
}
Received appointment data: {
  AppointmentID: '44',
  PatientID: '21',
  DoctorID: '12',
  Purpose: 'strep',
  DateTime: '2024-04-29T12:31'
}
Received patient data: {
  Name: 'Kian Hakim',
  DateOfBirth: '2003-06-15',
  Gender: 'M',
  Email: 'khakim@gmail.com',
  PhoneNumber: '9728001711x12',
  HomeAddress: '414 kp lane',
  EmergencyContact: 'abc'
}
Received doctor data: {
  DoctorID: '12',
  Name: 'Aditya Gavali',
  Specialty: 'hearing',
  Email: 'gavalid@gmail.com',
  OfficePhoneNumber: '123 123 2221',
  OfficeAddress: 'carrow road bld 150'
}
```

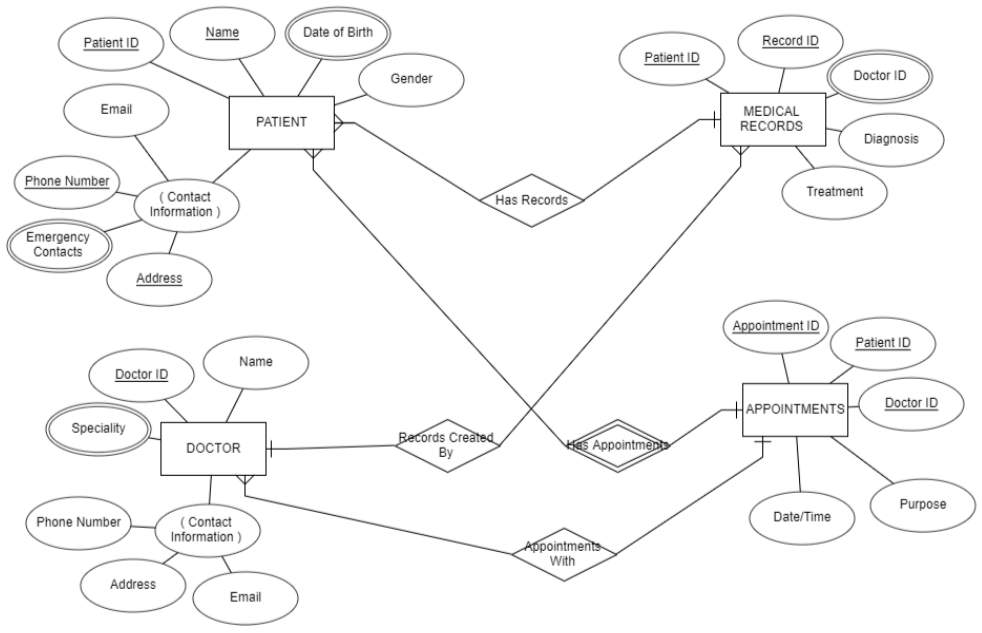
And now, it is able to receive the data in the backend and store it into the database in MySQL, which then allows us to query the data to answer patient related questions in an efficient manner.

User Application Interface

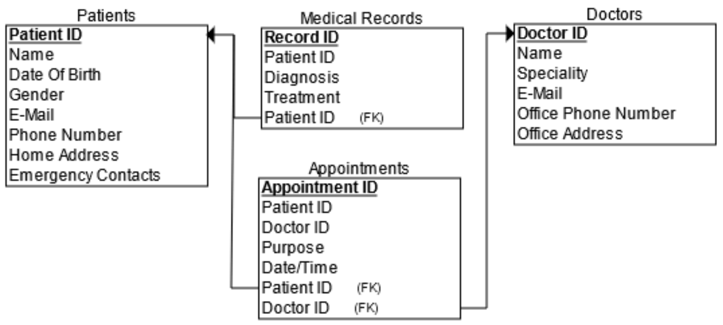
Its main use case is for doctors that want to be able to store patient data. With the UI we made, we've been able to make it seamless for not only patients, but for doctors to be able create, read, upload and delete data. We've also created admin capabilities which allows for physicians to view and update information as needed. Using node.js as the backend has allowed us to increase security and in turn will lead to less injection attacks.

Logical Database Schema:

ER Diagram:



Relational Database Schema:



SQL statements for Patient, Doctors, Medical Records, and Appointments:

```
CREATE TABLE `patients` (
  `PatientID` int NOT NULL,
  `Name` varchar(255) NOT NULL,
  `DateofBirth` datetime NOT NULL,
  `Gender` varchar(1) NOT NULL,
  `Email` varchar(255) DEFAULT NULL,
  `PhoneNumber` varchar(100) DEFAULT NULL,
  `HomeAddress` varchar(255) DEFAULT NULL,
  `EmergencyContact` varchar(100) DEFAULT NULL,
  PRIMARY KEY (`PatientID`),
  UNIQUE KEY `PatientID_UNIQUE` (`PatientID`)
ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `doctors` (
  `DoctorID` int NOT NULL,
  `Name` varchar(255) NOT NULL,
  `Specialty` varchar(255) NOT NULL,
  `Email` varchar(255) DEFAULT NULL,
  `OfficePhoneNumber` varchar(40) DEFAULT NULL,
  `OfficeAddress` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`DoctorID`),
  UNIQUE KEY `DoctorID_UNIQUE` (`DoctorID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `appointments` (
  `AppointmentID` int NOT NULL,
  `PatientID` int NOT NULL,
  `DoctorID` int NOT NULL,
  `Purpose` varchar(255) DEFAULT NULL,
  `DateTime` datetime NOT NULL,
  PRIMARY KEY (`AppointmentID`),
  UNIQUE KEY `AppointmentID_UNIQUE` (`AppointmentID`),
  KEY `PatientID_idx` (`PatientID`),
  KEY `DoctorID_idx` (`DoctorID`),
  CONSTRAINT `DoctorID` FOREIGN KEY (`DoctorID`) REFERENCES `doctors`
(`DoctorID`),
  CONSTRAINT `pID` FOREIGN KEY (`PatientID`) REFERENCES `patients`
(`PatientID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

```
CREATE TABLE `medicalrecords` (
  `RecordID` int NOT NULL,
  `PatientID` int NOT NULL,
  `Diagnosis` text,
  `Treatment` text,
  PRIMARY KEY (`RecordID`),
  UNIQUE KEY `RecordID_UNIQUE` (`RecordID`),
  KEY `PatientID_idx` (`PatientID`),
  CONSTRAINT `PatientID` FOREIGN KEY (`PatientID`) REFERENCES `patients`
(`PatientID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Expected Database Operations:
CRUD Operations: Insert, update, delete, and retrieve records in each table.
Search Operations: Query records based on various attributes, such as searching for appointments by doctor or patient, or looking up medical records by patient.
Reporting: Generate reports such as patient histories, appointment schedules for doctors, or statistical health reports.
Notification System: Trigger notifications for upcoming appointments or medical follow-ups.

Estimated Data Volumes:
Patients: A medium-sized clinic may manage information for around 500 to 5000 patients.
Doctors: A typical hospital might employ between 100 and 1,000 doctors.
Medical Records: Can range into hundreds of thousands; where each patient might have multiple records.
Appointments: A busy clinic might schedule anywhere from 50 to 170 appointments daily, leading to tens of thousands annually.

Functional Dependencies and Database Normalization

The health track consists of 4 main tables: Patients, Doctors, Medical Records, and Appointments. The functional dependencies have already been identified. For each table, we first identify the functional dependencies. Then we check if each table is 1NF by having a primary key. If the table includes a primary key, and meets all the requirements of 1NF, we go to the second normal form and continue on until we normalize each table to 3NF if possible.
Functional Dependencies:
Patient ID → Name, Date Of Birth, Gender, E-Mail, Phone Number, Home Address, Emergency Contacts
Record ID → Patient ID, Diagnosis, Treatment
Doctor ID → Name, Specialty, E-Mail, Office Phone Number, Office Address
Appointment ID → Patient ID, Doctor ID, Purpose, DateTime

1NF: All tables Patients, Doctors, MedicalRecords, and Appointments are already in 1NF since each has a primary key and all column values are atomic.

2NF: All four tables either have a single attribute as a primary key or do not show partial dependency, hence they are in 2NF:
Patients (PatientID → all other attributes)
Doctors (DoctorID → all other attributes)
MedicalRecords (RecordID → all other attributes)

Appointments (AppointmentID → all other attributes)

3NF: Since there are no transitive dependencies in any of the tables and all attributes depend only on the primary key, all tables are already in 3NF:
Patients have all information dependent only on PatientID.
Doctors have all information dependent only on DoctorID.
MedicalRecords has all information dependent only on RecordID.
Appointments has all information dependent only on AppointmentID.

Patients Table:

```sql
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    Name VARCHAR(255),
    DateOfBirth DATE,
    Gender VARCHAR(10),
    Email VARCHAR(255),
    PhoneNumber VARCHAR(15),
    HomeAddress VARCHAR(255),
    EmergencyContact VARCHAR(255)
);
```

Doctor Table:

```sql
CREATE TABLE Doctors (
    DoctorID INT PRIMARY KEY,
    Name VARCHAR(255),
    Specialty VARCHAR(255),
    Email VARCHAR(255),
    OfficePhoneNumber VARCHAR(15),
    OfficeAddress VARCHAR(255)
);
```

Medical Records Table:

```sql
CREATE TABLE MedicalRecords (
    RecordID INT PRIMARY KEY,
    PatientID INT,
    Diagnosis TEXT,
    Treatment TEXT,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
);
```

Appointments Table:

```
CREATE TABLE Appointments (
    AppointmentID INT PRIMARY KEY,
    PatientID INT,
    DoctorID INT,
    Purpose VARCHAR(255),
    DateTime DATETIME,
    FOREIGN KEY (PatientID) REFERENCES Patients(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID)
);
```

Conclusions and Future Work

In the future, we’d like to be able to enhance the frontend. By implementing a nice color scheme and making the site look nicer, that’d be a great place to start. For the backend, being able to allow patients to view their data on not just the admin would be something that we could look at. And for the database, being able to add extension numbers for not only doctors, but patients as well would be great.

References

Video we used to connect the MySQL Database to HTML using Node.js

Video we used to help us create the database and tables in MySQL