

# CS 4347 - Database Systems

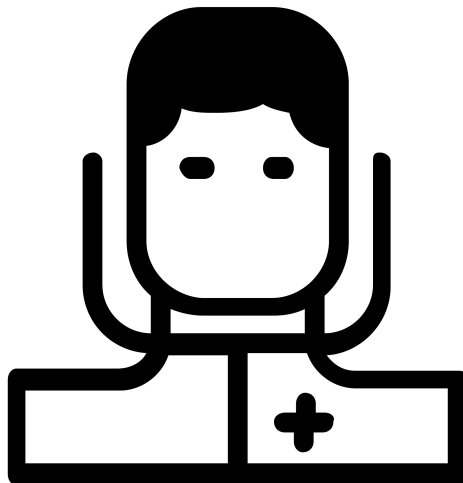
## Jalal Omer

### Patient-Link

---

Sampat Bandlamudi  
Pranit Srivastava  
Olawale Ogunbanwo  
Harry Seo  
Christian Ermias

8/10/25



# Table of Contents

1. Introduction
2. System Requirements
3. Conceptual Design of the Database
4. Logical Database Schema
5. Functional Dependencies and Database Normalization
6. The Database System
7. User Application Interface
8. Conclusions and Future Work
9. References
10. Appendix

# Introduction

The name of our project is **Patient-Link**. Our team name is called **Health Hackers** and consists of:

- Sampat Bandlamudi - sxb220228
- Pranit Srivastava - pxs 210113
- Olawale Ogunbanwo - oho200001
- Harry Seo - sxs180322
- Christian Ermias - cxe210011

Our suggested database system will solve many common issues in the healthcare sector. Some common issues are scheduling errors, lack of convenience, and inefficient productivity. In a day and age of watchful insurance and healthcare, it is imperative that medical records, usages, and deliveries are safely stored and utilized. Our solution would tackle each of these problems. Having a database system would allow for a real-time view of the schedule and let the system prevent double-bookings. The database system would also allow patients to no longer be restricted to office hours to manage their appointments. So, by being able to book, view, or modify their visits at any time would be more convenient. Finally, by being able to instantly list last-minute cancellations and allowing for those empty slots to be filled, the clinic would be able to maximize its productivity.

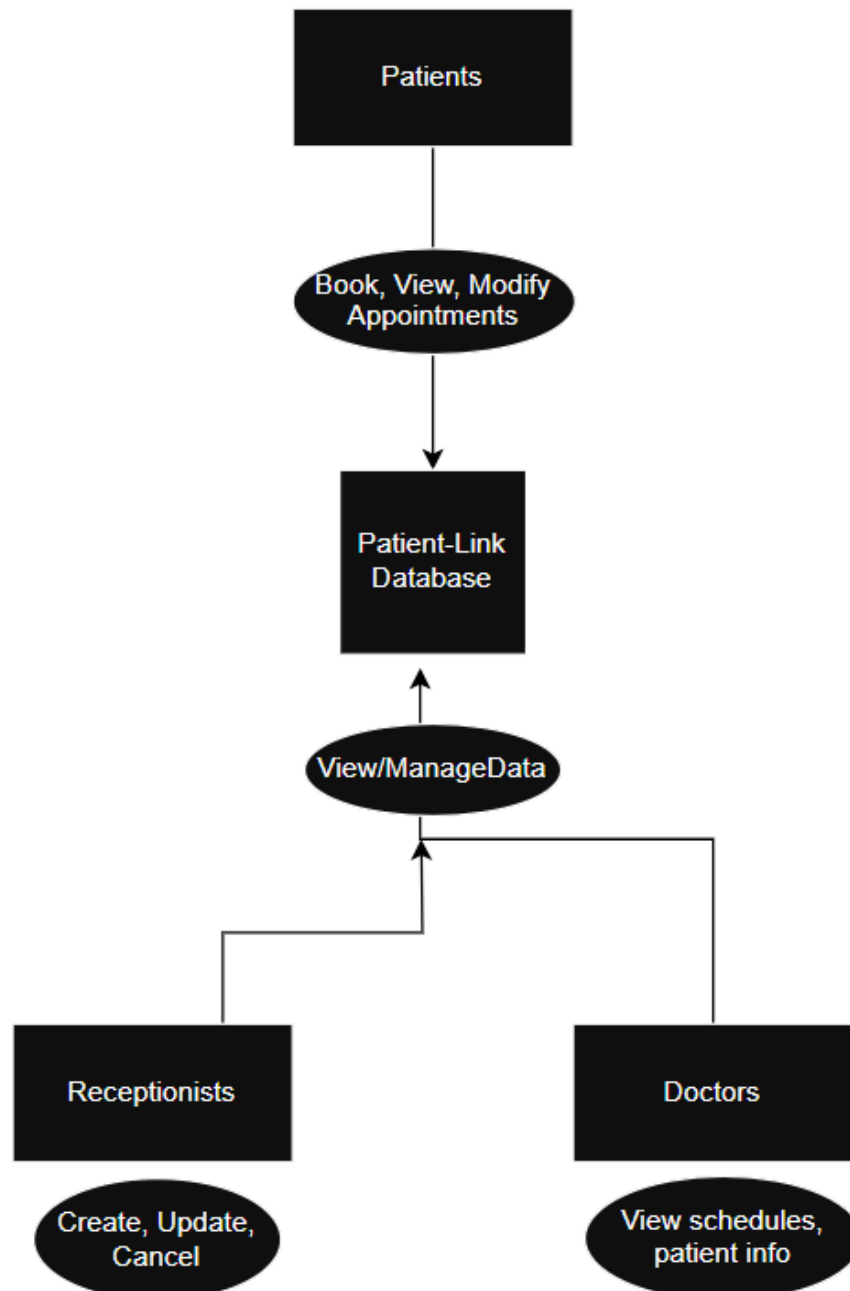
We would need a database as opposed to an Excel document for many reasons. Firstly, a medical clinic could scale up to generating thousands of appointments over time, and implementing this kind of

volume of data into an Excel sheet would lead it to be extremely slow and hard to manage. A database would allow for better scalability while maintaining fast performance. Additionally, since patient information is sensitive and protected by numerous regulations, a database would ensure privacy. It could be set to allow patients to only see their own data while still allowing doctors to oversee all the appointments on their schedule. Another major benefit with the database is concurrency control. Excel is prone to conflicting files and lack of simultaneous edits. A database, on the other hand, is designed for concurrent transactions and would allow for more consistent and reliable data.

However, switching from Excel and such files are not completely lost. In fact, the database allows us to utilize these old systems for simplicity and efficiency. We can use the CSV files of data with patient or doctor information to input as attributes for our database.

# System Requirements

## System Architecture Diagram



## Interface Requirements

The *Patient-Link* system will have three primary interfaces:

### 1. Patient Portal Interface

- Login and authentication page
- Appointment Booking, view, and modification tools.
- Profile management for contact details.

### 2. Staff (Receptionist) Portal Interface

- Dashboard showing all doctors' appointments.
- Forms for registering new patients.
- Appointment booking and cancellation tools.
- Patient information update functionality.

### 3. Doctor Portal Interface

- Dashboard showing the doctor's own schedule.
- Patient profile viewing for upcoming appointments.
- Ability to update appointment statuses.

## Functional Requirements

- Patients can log in securely using unique credentials.
- Patients can view, book, and modify their own appointments.
- Receptionists can register new patients and manage their data.
- Receptionists can view all doctors' schedules and manage appointments.
- Doctors can view their personal appointments schedules and relevant patient information.
- The system must prevent double-bookings by checking schedule conflicts.

- The system must store and retrieve billing information for completed appointments.
- Administrators can manage user accounts, permissions, and roles.
- The database must maintain a complete history of appointments and billing records.
- Only authorized users should be able to modify sensitive data.

### Non-Functional Requirements

#### 1. Performance:

- The system should return appointment search results quickly under a normal load.
- The database must support concurrent transactions without performance degradation.

#### 2. Security:

- All connections to the database must be encrypted.
- Role-based access control must be enforced for different user types.
- Data must comply with HIPPA requirements for privacy.

#### 3. Availability:

- The system should have 99.5% uptime excluding planned maintenance.

#### 4. Scalability:

- The database should support the total number of patients without performance loss.

#### 5. Backup & Recovery:

- Daily automated backups must be taken.
- The system should be able to recover within a short amount of time after failure.

## 6. Usability:

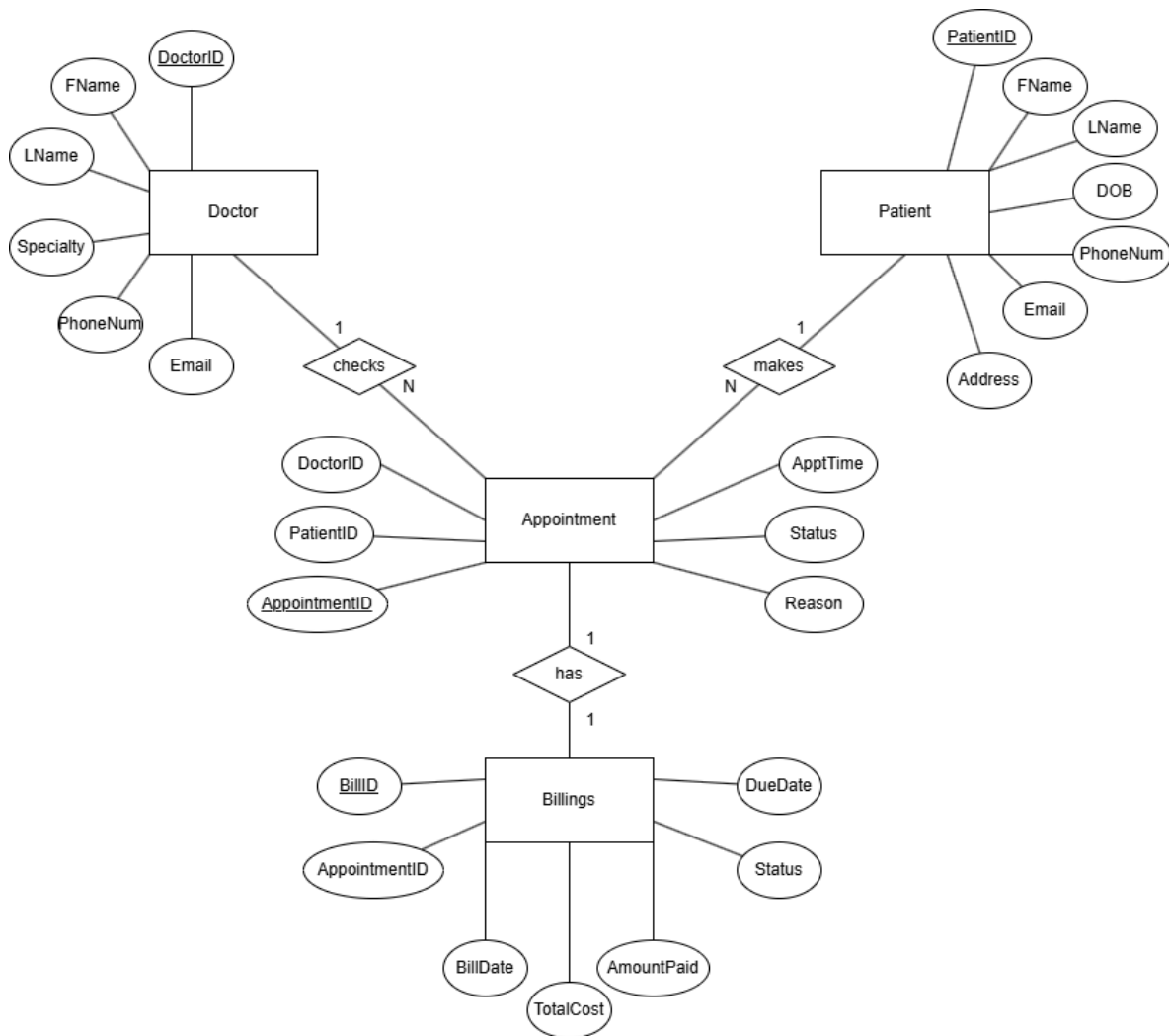
- Interfaces must be intuitive, with role-specific dashboards.

# Conceptual Design of the Database

The conceptual design of this database was created to model the key entities and relationships that are required to operate the clinic. This can be visualized by using an ER model, which is the blueprint for this system. The complete ER model visualizes the main entities of the system, which are Doctor, Patient, Appointment, and Billings. The model also shows their attributes as well as the relationships that connect them. Below is the complete ER model.



## ER Diagram



## Data Dictionary

The data dictionary helps give more context for each attribute in the ER model. Here is the relational database schema that we got using our ER model. Each relation shows the attributes, keys, datatypes, and the constraints.

### Doctors Relation - Information for each doctor

Attribute Name	Type	Description	Nullable	Key/Constraint
DoctorID	INT	ID number for each doctor	No	Primary Key

FirstName	VARCHAR(100)	First name, cannot be empty	No	Not Null
LastName	VARCHAR(100)	Last name, cannot be empty	No	Not Null
Specialty	VARCHAR(100)	Doctor's specialty	Yes	
PhoneNumber	VARCHAR(15)	Doctor's phone number	Yes	
Email	VARCHAR(100)	Doctors email address	No	Unique

### Patients Relation - Information for each patient

Attribute Name	Type	Description	Nullable	Key/Constraint
PatientID	INT	ID number for each patient	No	Primary Key
FirstName	VARCHAR(100)	First name, cannot be empty	No	Not Null
LastName	VARCHAR(100)	Last name, cannot be empty	No	Not Null
DOB	DATE	Patient's date of birth	No	Not Null
PhoneNumber	VARCHAR(15)	Patient's phone number	Yes	
Email	VARCHAR(100)	Patient's email address	No	Unique
Address	VARCHAR(255)	Patient's home address	Yes	

### Appointments Relation - Connects patients and doctors

Attribute Name	Type	Description	Nullable	Key/Constraint
AppointmentID	INT	ID number for each appointment	No	Primary Key
PatientID	INT	Link to patients table	No	Foreign Key
DoctorID	INT	Link to doctors table	No	Foreign Key
AppointmentTime	TIMESTAMP	Appointment date and time	No	Not Null

### Billings Relation - Financial information for an appointment

Attribute Name	Type	Description	Nullable	Key/Constraint
BillID	INT	ID number for each bill	No	Primary Key
AppointmentID	INT	Link to appointments table	No	Foreign Key Unique
BillDate	DATE	Bill issue date	No	Not Null
TotalCost	DECIMAL(10,2)	Service Cost	No	Not Null
AmountPaid	DECIMAL(10,2)	Amount paid so far	No	Default 0.00
Status	VARCHAR(50)	Payment status	No	Default 'Pending'
DueDate	DATE	Payment due date	No	Not Null

### Referential Integrity

Here are the rules that were enforced to maintain data consistency

- For the PatientID and DoctorID foreign keys in the Appointments table, the ON DELETE action is restricted. This is to prevent a doctor or a record of a patient from being deleted if they have any existing appointments. This would avoid losing historical records.
- For the AppointmentID foreign key in the Billings table, the ON DELETE action is set to cascade. So, if an appointment is deleted,

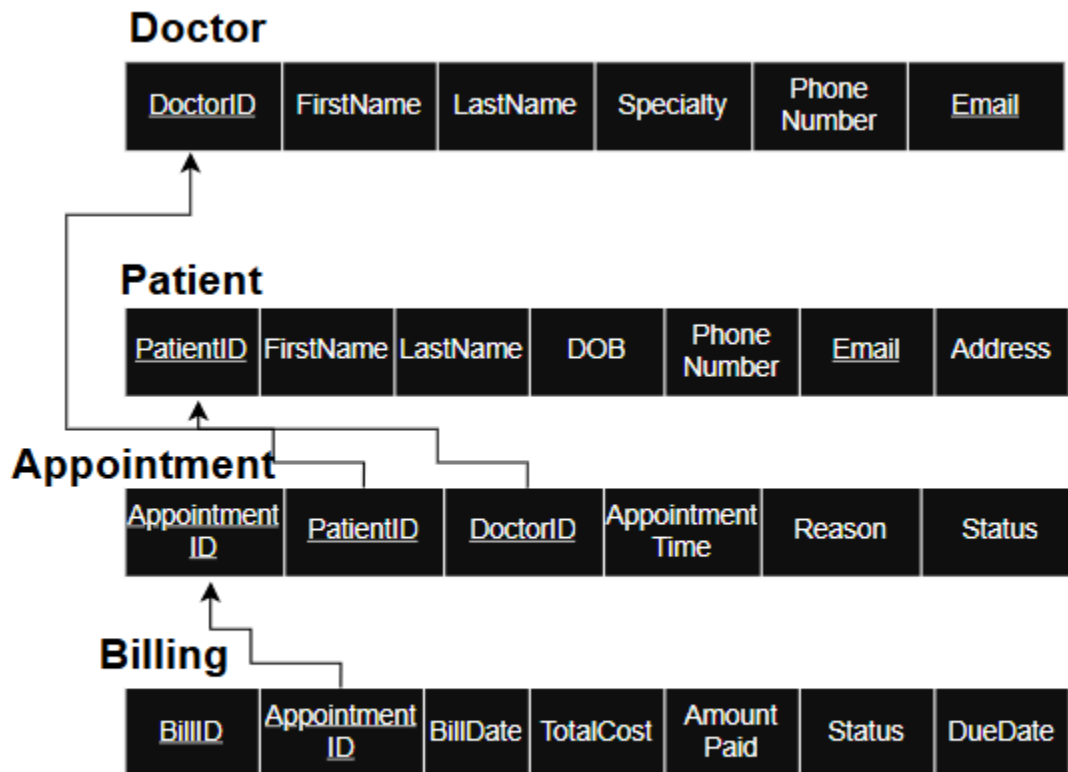
the corresponding bill for it is automatically deleted as well. This helps prevent the existence of financial records for appointments that do not exist anymore.

### Mapping from ER model to Relational Schema

When translating from the conceptual ER diagram to the logical relational schema, these were the design principles that we established:

- Entities to Tables: Every strong entity in the ER diagram was mapped to a corresponding table. For example, the Doctor entity was mapped to a Doctors table.
- Attributes to Columns: Each entity's attributes became the columns of their respective tables. The identifier for each entity was set to be the primary key of its table.
- One-to-Many Relationships: The 1-to-N relationships between the Doctor/Patient and Appointment were made by adding the primary key of the "one" side as a foreign key in the table of the "many" side.
- One-to-One Relationships: The 1-to-1 relationships between Appointment and Billings were established by adding the primary key of the Appointment table, which was AppointmentID, as a unique foreign key in the Billings table.

# Logical Database Schema



## SQL Statements

Doctors Table:

```
CREATE TABLE Doctors (  
    DoctorID INT PRIMARY KEY,  
    FirstName VARCHAR(100) NOT NULL,  
    LastName VARCHAR(100) NOT NULL,  
    Specialty VARCHAR(100),
```

```
PhoneNumber VARCHAR(15),  
  
Email VARCHAR(100) NOT NULL UNIQUE  
  
);
```

#### Patients Table:

```
CREATE TABLE Patients (  
  
    PatientID INT PRIMARY KEY,  
  
    FirstName VARCHAR(100) NOT NULL,  
  
    LastName VARCHAR(100) NOT NULL,  
  
    DOB DATE NOT NULL,  
  
    PhoneNumber VARCHAR(15),  
  
    Email VARCHAR(100) NOT NULL UNIQUE,  
  
    Address VARCHAR(255)  
  
);
```

#### Appointment's Table:

```
CREATE TABLE Appointments (  
  
    AppointmentID INT PRIMARY KEY,  
  
    PatientID INT NOT NULL,  
  
    DoctorID INT NOT NULL,  
  
    AppointmentTime TIMESTAMP NOT NULL,  
  
    Reason VARCHAR(300),
```

```

Status VARCHAR(100) DEFAULT 'Scheduled',

CONSTRAINT fk_patient FOREIGN KEY (PatientID)

REFERENCES Patients(PatientID)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT fk_doctor FOREIGN KEY (DoctorID)

REFERENCES Doctors(DoctorID)

ON DELETE SET NULL

ON UPDATE CASCADE

);

```

#### Billings Table:

```

CREATE TABLE Billings (

    BillID INT PRIMARY KEY,

    AppointmentID INT NOT NULL UNIQUE,

    BillDate DATE NOT NULL,

    TotalCost DECIMAL(10,2) NOT NULL,

    AmountPaid DECIMAL(10,2) DEFAULT 0.00 NOT NULL,

    Status VARCHAR(50) DEFAULT 'Pending',

    DueDate DATE NOT NULL,

    CONSTRAINT fk_appointment FOREIGN KEY (AppointmentID)

```

REFERENCES Appointments(AppointmentID)

ON DELETE CASCADE

ON UPDATE CASCADE

);

### Expected Operations and Data Volumes

Doctors Table -

- **Insert:** Add new doctors
- **Update:** Change phone number, specialty, or email
- **Select:** Search by name, specialty
- **Delete:** Remove doctor record
- **Estimated Records/Year:** ~50-100 :: Low Growth Rate

Patients Table -

- **Insert:** Add new patients
- **Update:** Change address, phone, or email
- **Select:** Search by name, DOB, email
- **Delete:** Remove patient
- **Estimated Records/Year:** ~2000-5000 :: Medium Growth Rate

Appointments Table -

- **Insert:** Schedule new appointment
- **Update:** Reschedule or change status
- **Select:** View doctor/patient schedules
- **Delete:** Cancel appointment
- **Estimated Records/Year:** ~10,000-20,000 :: High Growth Rate



## Billings Table -

- **Insert:** Create bill when appointment is completed
- **Update:** Change payment status, update amount paid
- **Select:** View outstanding payments, overdue bills
- **Delete:** Remove bill if appointment is deleted
- **Estimated Records/Year:** ~10,000-20,000 :: High Growth Rate

# Functional Dependencies and Database Normalization

We conducted an analysis of the functional dependencies for each relation in order to ensure that the database schema is efficiently structured and is clear of any data anomalies. These relations were all evaluated against normal forms. Below is a 3NF analysis for each relation.

### Doctors Relation

Attributes: DoctorID (Primary Key), FirstName, LastName, Specialty, PhoneNumber, Email

Function Dependencies:

DoctorID -> FirstName, LastName, Specialty, PhoneNumber, Email

- This is because the primary key DoctorID determines all other attributes in the relation

Email -> DoctorID, Firstname, LastName, Specialty, PhoneNumber

- This is because the Email attribute is defined with a UNIQUE constraint, so it becomes a candidate key

3NF Analysis:

- The relation is in 1NF since all attribute values are basically atomic

- The relation is in 2NF because the primary key is a single attribute, so there are no partial dependencies anywhere
- The relation is in 3NF because there is no case where a non-key attribute is dependent on another non-key attribute.

The Doctors relation is already in 3NF.

#### Patients Relation:

Attributes: PatientID (Primary Key), FirstName, LastName, DOB, PhoneNumber, Email, Address

Function Dependencies:

PatientID -> FirstName, LastName, DOB, PhoneNumber, Email, Address

- This is because the primary key PatientID determines all other attributes in the relation

Email -> PatientID, FirstName, LastName, DOB, PhoneNumber, Email, Address

- This is because the Email attribute is defined with a UNIQUE constraint, so it becomes a candidate key

3NF Analysis:

- The relation is in 1NF since all attribute values are basically atomic
- The relation is in 2NF because the primary key is a single attribute, so there are no partial dependencies anywhere
- The relation is in 3NF because there is no case where a non-key attribute determines what another non-key attribute is.

The Patients relation is already in 3NF.

#### Appointments Relation:

Attributes: AppointmentID (Primary Key), PatientID (Foreign Key), DoctorID (Foreign Key), AppointmentTime, Reason, Status

Function Dependencies:

AppointmentID -> PatientID, DoctorID, AppointmentTime, Reason, Status

- This is because the primary key AppointmentID is a unique identifier and basically determines what the details of that specific appointment are

3NF Analysis:

- The relation is in 1NF since all attribute values are basically atomic
- The relation is in 2NF because the primary key is a single attribute, so there are no partial dependencies anywhere
- The relation is in 3NF because the attributes are all mutually independent and only depend on the AppointmentID

The Appointments relation is already in 3NF.

#### Billings Relation:

Attributes: BillID (Primary Key), AppointmentID (Foreign Key, Unique), BillDate, TotalCost, AmountPaid, Status, DueDate

Function Dependencies:

BillID → AppointmentID, BillDate, TotalCost, AmountPaid, Status, DueDate

- This is because the primary key BillID is a unique identifier and basically determines what the details of that specific bill are
- AppointmentID → BillID, BillDate, TotalCost, AmountPaid, Status, DueDate
- The AppointmentID is a foreign key that has a UNIQUE constraint. So, this means that there is a 1:1 relationship between an appointment and a bill. So AppointmentID is a candidate key for the Billings relation

3NF Analysis:

- The relation is in 1NF since all attribute values are basically atomic
- The relation is in 2NF because the primary key is a single attribute, so there are no partial dependencies anywhere
- The relation is in 3NF because the attributes are all mutually independent and only depend on the AppointmentID

The Billings relation is already in 3NF

Since our analysis confirmed that all of the relations in the original schema are already in 3NF, we didn't need to perform any decomposition or restructuring.

## The Database System

The appropriate schema utilizing SQL

**create.sql**

```
CREATE TABLE Doctors (  
  DoctorID INT NOT NULL PRIMARY KEY,  
  FirstName VARCHAR(100) NOT NULL,  
  LastName VARCHAR(100) NOT NULL,  
  Specialty VARCHAR(100),  
  PhoneNumber VARCHAR(15),  
  Email VARCHAR(100) NOT NULL UNIQUE  
);  
CREATE TABLE Patients (  
  PatientID INT NOT NULL PRIMARY KEY,  
  FirstName VARCHAR(100) NOT NULL,  
  LastName VARCHAR(100) NOT NULL,  
  DOB DATE NOT NULL,  
  PhoneNumber VARCHAR(15),  
  Email VARCHAR(100) NOT NULL UNIQUE,  
  Address VARCHAR(255)  
);  
CREATE TABLE Appointments (  
  AppointmentID INT NOT NULL PRIMARY KEY,  
  PatientID INT NOT NULL,  
  DoctorID INT NOT NULL,
```

```
AppointmentTime TIMESTAMP NOT NULL,  
Reason VARCHAR(300),  
Status VARCHAR(100) NOT NULL DEFAULT 'Scheduled',  
FOREIGN KEY (PatientID) REFERENCES Patients(PatientID) ON DELETE RESTRICT,  
FOREIGN KEY (DoctorID) REFERENCES Doctors(DoctorID) ON DELETE RESTRICT  
);
```

### **Load.sql**

```
LOAD DATA INFILE 'doctors.csv'  
INTO TABLE Doctors  
FIELDS TERMINATED BY ','  
ENCLOSED BY ''''  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;  
LOAD DATA INFILE 'patients.csv'  
INTO TABLE Patients  
FIELDS TERMINATED BY ','  
ENCLOSED BY ''''  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;  
LOAD DATA INFILE 'appointments.csv'  
INTO TABLE Appointments  
FIELDS TERMINATED BY ','  
ENCLOSED BY ''''  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;  
LOAD DATA INFILE 'billings.csv'  
INTO TABLE Billings  
FIELDS TERMINATED BY ','  
ENCLOSED BY ''''  
LINES TERMINATED BY '\n'  
IGNORE 1 ROWS;
```

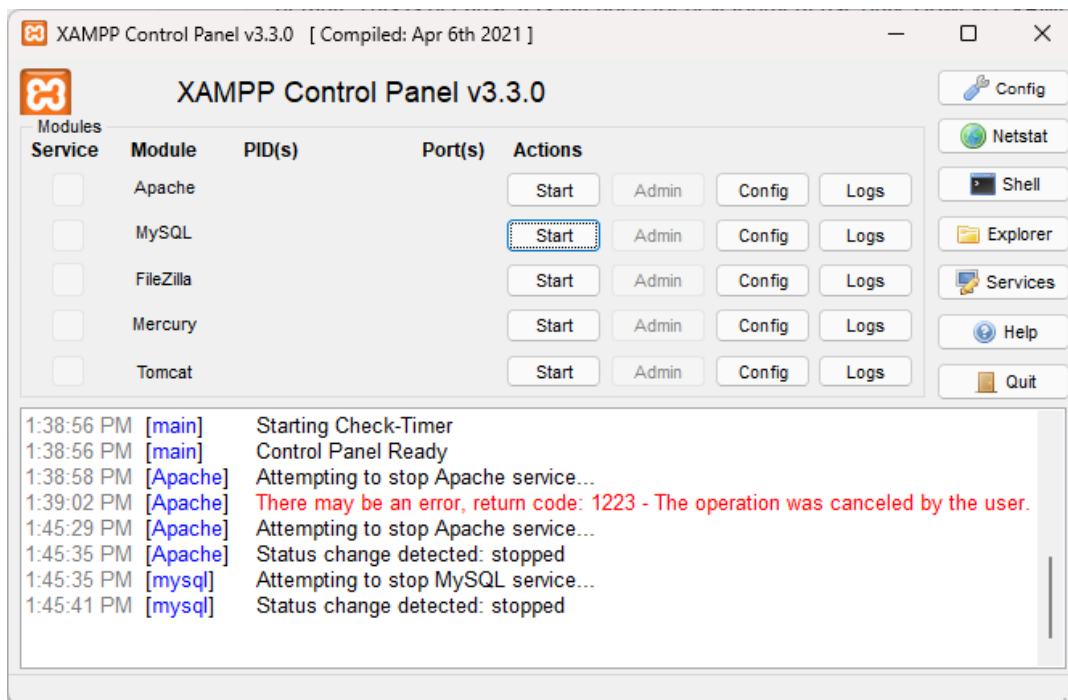
Data included would be patient and Doctor data reflected from CSV files.

SQL functions utilized include COUNT() SUM() AVG() MIN() while the data volumes that were utilized through inputted CSV's would be Date, Time, Email, AppointmentID, and so on.

## Database Installation

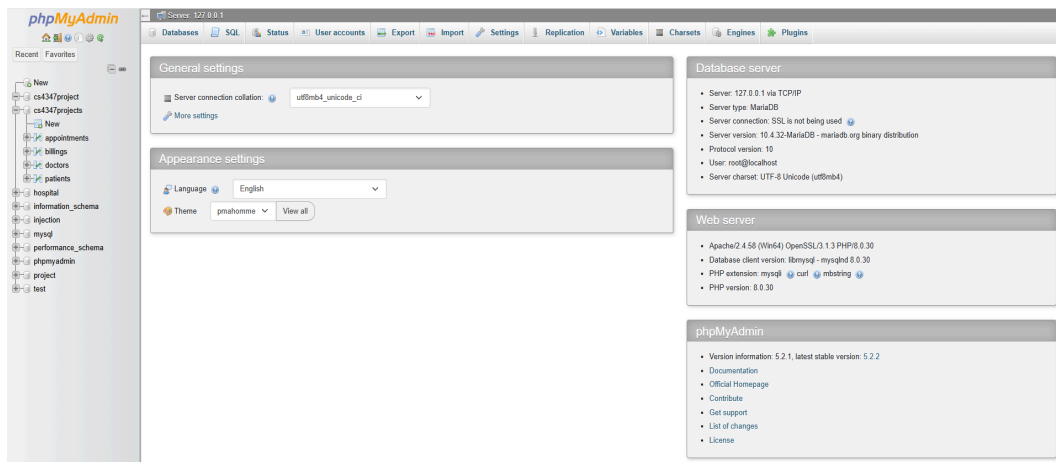
For our database, we used XAMPP to run Apache and MariaDB / MySQL to create a local server environment for our project. To give clear instructions on how to install and run XAMPP and set up the database for our system, we will go through step by step from installation to creating entries for the database.

- a) If not done so already, install the latest version of XAMPP. Keep the default settings and proceed to install. After it has finished installing, open up the control panel.
- b) Make sure you have your php and html files where your localhost will be able to access it. The path should be (by default) C:\xampp\htdocs\ (your project folder). If you installed XAMPP somewhere else, make sure you locate the htdocs folder and make a new folder where you will move your php and html files.
- c)

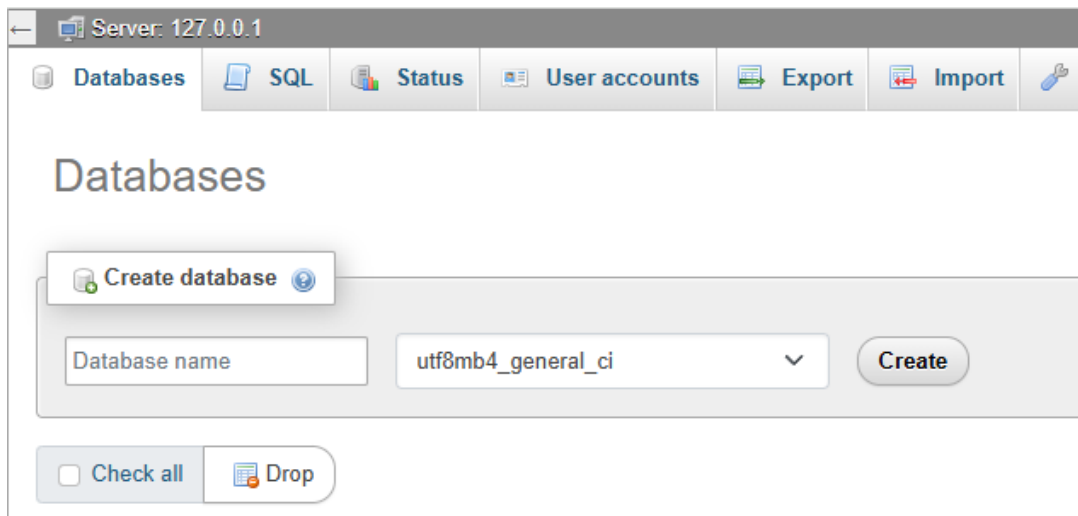


Once you have finished moving the files, start up Apache and MySQL. Now you should be able to access localhost on your browser. Try typing in localhost/(folder name) to see if you can access the files in the browser.

d) Now to set up our database, we need to go to localhost/phpMyAdmin.



On the top left, you should see a list of your databases. Click on New to create a new database for your system.



Name your database and click on create. You should be able to see your database on the left toolbar and you should be taken to a page where it allows you to create your tables.

- e) Instead of creating tables manually, you can click on SQL on the upper toolbar and you should be able to paste your SQL code to create the tables you need for your database. Paste and click on Go on the bottom right of the page. You should be able to see your tables from the dropdown menu of your database. \*Make sure each table allows entries to be auto-incremented. You can run this SQL code for each table:

```
ALTER TABLE (table_name) MODIFY (ID) INT NOT NULL AUTO_INCREMENT;
```

- f) Now with the tables set up, we can import our CSV files into each table. Use the left dropdown menu to navigate to the table you wish to insert data to. On the top toolbar, click on Import and upload the CSV file you wish to use. Make sure that the format is set to CSV. Click on import.

0.0.1 » Database: cs4347projects » Table: appointments

Structure SQL Search Insert Export Import Privileges Operations Trac

**File to import:**

File may be compressed (gzip, bzip2, zip) or uncompressed.  
A compressed file's name must end in `.[format].[compression]`. Example: `.sql.zip`

Browse your computer: (Max: 40MiB)

Choose File No file chosen

You may also drag and drop a file on any page.

Character set of the file:

utf-8

**Partial import:**

☒ Allow the interruption of an import in case the script detects it is close to the PHP timeout limit.  
This might be a good way to import large files, however it can break transactions.

Skip this number of queries (for SQL) starting from the first one:

0

**Other options**

☒ Enable foreign key checks

**Format**

CSV

Note: If the file contains multiple tables, they will be combined into one.

Now your database should be all set up to use.



# User Application Interface

We imagine our website to have a simple and easy design to increase accessibility and user-friendly interactions for people of all ages and needs. We will require our patients to create an account and log in for booking, viewing, and cancelling appointments, and checking their billings. For our doctors, we will have an account made for each doctor and they will be able to do the same things as their patients- but in addition, doctors will be able to create patient entries in case of walk-in appointments.

## Register New Patient

First Name:

Last Name:

Date of Birth:

Phone Number:

Email:

Address:

Register Patient

- 1) A simple registration page for patients.

## Login

Email or Phone Number:

Password:


Login

2) Patients can then log in to their account.


## Book an Appointment

Doctor ID:

Appointment Date:

Appointment Time:


 


Reason for Visit:

Book Appointment

3) Patients can then book appointments with the doctor they choose.

## View Appointments

Select Doctor: -- Choose a Doctor -- 

Select Date:  

View Appointments

- 4) Doctors and other staff can see which doctors have appointments on a given date.

## Cancel Appointment

- 5) Patients, doctors, and staff members can cancel appointments.

## Conclusions and Future Work

The Patient-Link project replaced an error-prone spreadsheet workflow with a simple, normalized (3NF) MySQL schema and a PHP web front-end. Across phases, the team designed the ER model, translated it to relations with keys/constraints, implemented core patient/appointment flows, and demonstrated secure data access using prepared statements (contrasted with deliberately vulnerable pages in the SQL-injection demo).

What worked well

- Clean schema: Doctors, Patients, Appointments, Billings with clear keys and foreign keys.
- Normalization reduced anomalies and kept CRUD predictable.
- XAMPP + PHP + MySQL stack is simple to develop and demo.
- Prepared statements eliminated injection risks in secure flows.

Current limitations

- Role-based access checks are not enforced everywhere (session/role gating inconsistent).
- Passwords lack proper hashing/salting and reset/verification flows.
- Limited server-side validation (e.g., timeslot conflicts, email/phone format).
- No audit logging, backup strategy, or provider-availability logic yet.

#### Proposed future work

- Full RBAC with guarded routes for Patients/Receptionists/Doctors/DBAs.
- Secure auth: hashed passwords (bcrypt/Argon2), email verification and password reset.
- Scheduling intelligence: provider availability, conflict detection, waitlists, reminders.
- Security & compliance: least-privilege DB user, audit trails, backups/PITR, HIPAA-aware practices.
- Performance & UX: proper indexes, pagination, input validation, accessible/responsive UI.
- DevOps: Dockerized setup, seed/migration scripts, CI for tests/lint, environment configs.
- Integrations: optional FHIR/HL7 bridge, insurance/claims APIs, analytics dashboards.

## References

MySQL 8.0 Reference Manual (DDL/DML, constraints, indexing).

PHP Manual: mysqli / PDO prepared statements and parameter binding.

OWASP Cheat Sheet Series: SQL Injection Prevention.

Chen, P. “The Entity-Relationship Model—Toward a Unified View of Data.”

W3C HTML Forms; MDN Web Docs (form semantics/validation).

XAMPP Documentation (Apache, PHP, MariaDB/MySQL).

phpMyAdmin User Guide (schema import/export, admin).

## Appendix

[https://utdallas.box.com/s/q6d2cen511r99oni16klrbe  
k74j08xr3](https://utdallas.box.com/s/q6d2cen511r99oni16klrbe<br/>k74j08xr3)