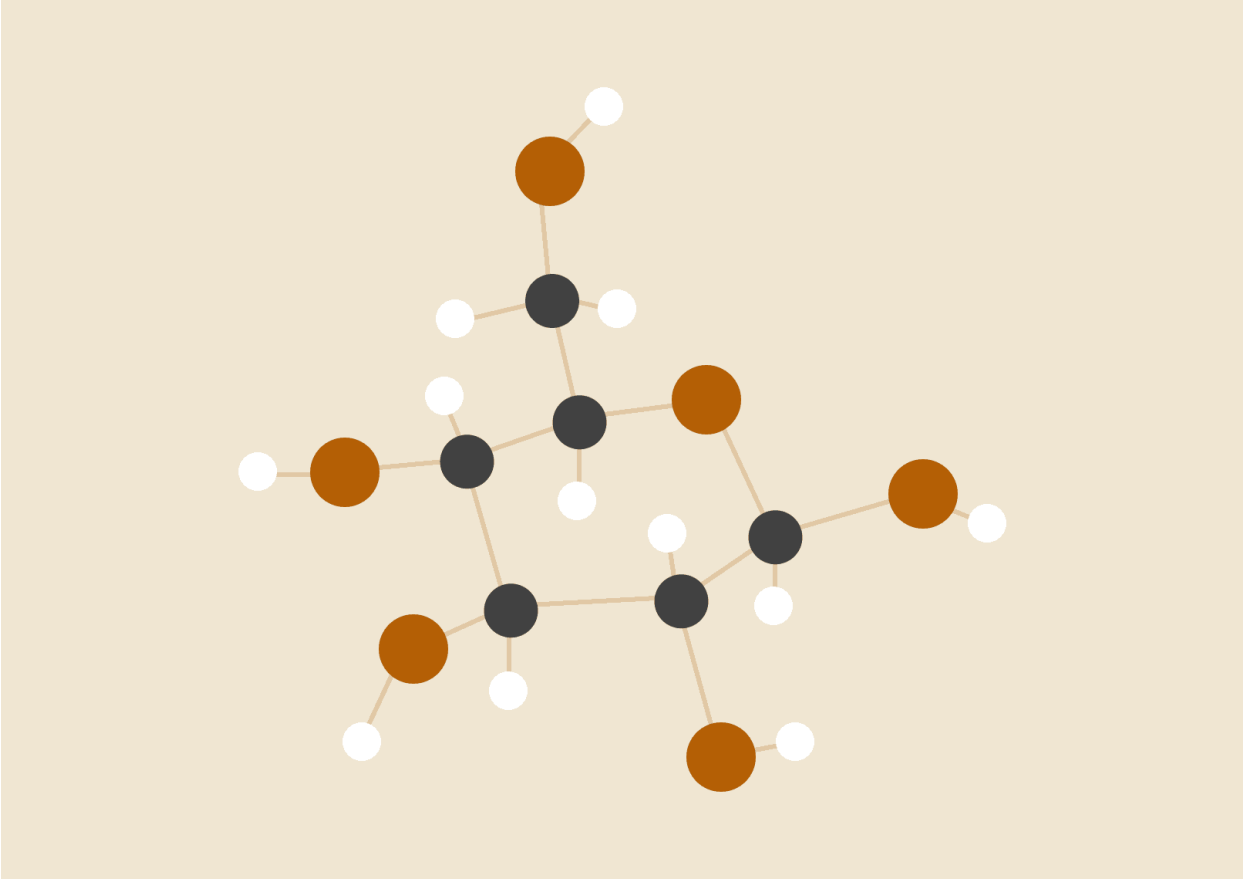
PROJECT REPORT

AJ Kimbrough ANK210005, Ariana Qozat AOQ210000, Aryan Tyagi AXT200084, Khushboo Amarnani KAA210004, Natthiya Sae Ngow NXS220110



CS 4347.501 DATABASE SYSTEMS
JALAL OMER
12/01/2024

TABLE OF CONTENTS

INTRODUCTION

(Provide a brief description of the project and the section organization of this report.)

The **Hospital Management System** (HMS) is developed to streamline and automate critical hospital operations such as patient registration, appointment scheduling, staff management, medical records handling, and billing. The system integrates all these processes into a centralized platform, ensuring data security, integrity, and ease of access for hospital staff and patients.

Purpose and Scope:
The HMS addresses common inefficiencies caused by disparate systems or manual processes within hospitals, such as duplicate data entry, scheduling conflicts, and billing errors. The project ensures compliance with health regulations while enhancing the quality of care through efficient management of hospital workflows.

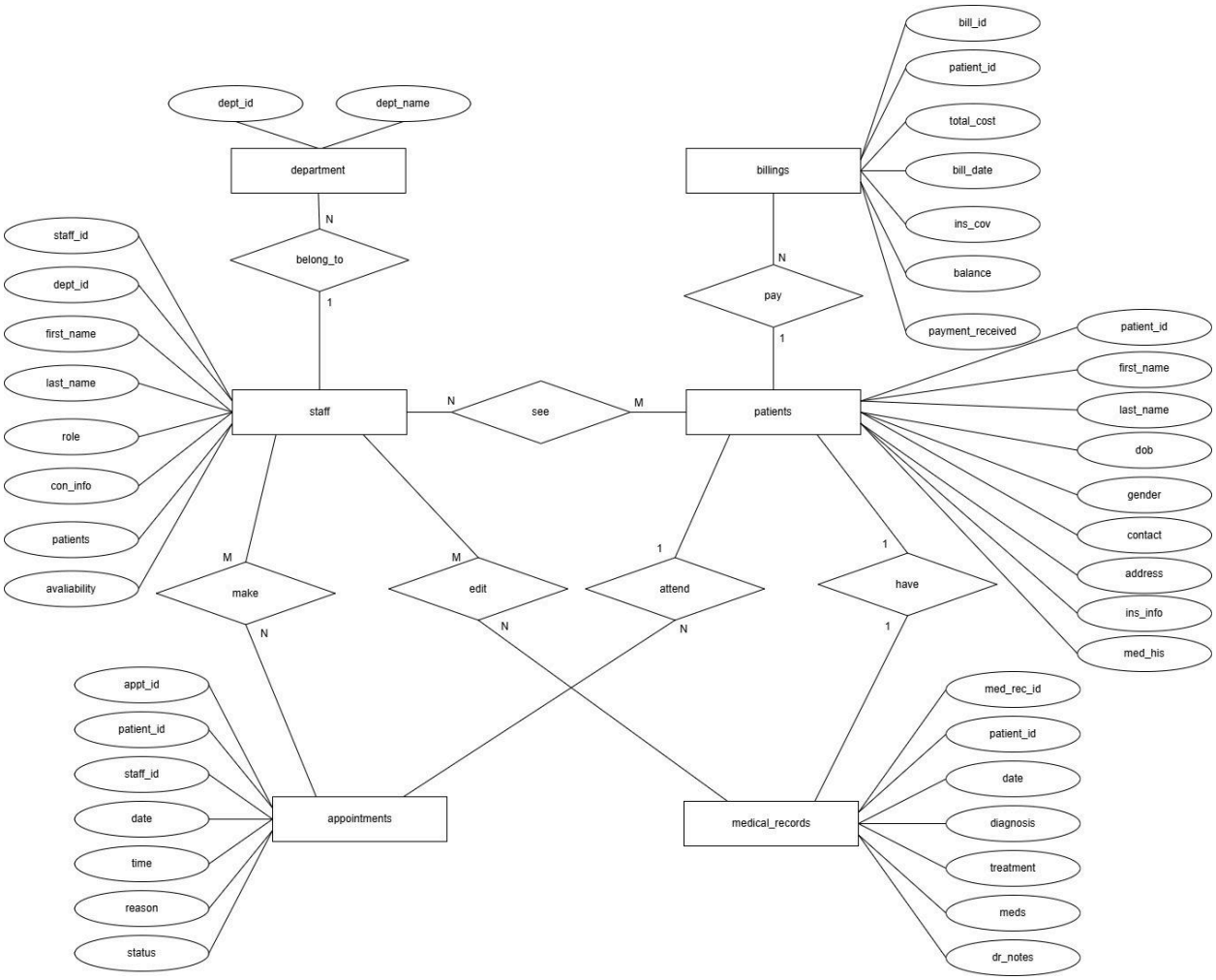
SYSTEM REQUIREMENTS

1. Context Diagram (System Architecture)
    - Users: Hospital administrators, doctor/nurses, patients, support staff, and database admin
    - Web Application (Front-End): Provides the interface for users to interact with the system
    - Database System (MySQL Backend): Stores and manages all the hospital-related data
    - External Systems: Integration with lab systems, pharmacy systems, or radiology systems
2. Interface Requirements:
    - Patient Portal: Allow patients to register, book appointments, access medical records, and manage billing information
    - Admin Dashboard: For admin to manage staff, monitor operations and generate reports
    - Electronic Medical Records(EMR): Provides secure access hospital staff to update and view patient records
    - Staff Management System: Tools for assigning shifts, managing availability, an tracking patient assignments
    - Billing and Payment Interface: Enables staff to manage invoices,

track payments, and handle insurance claims
    - Reporting Dashboard: Presents hospital metrics and generates reports for management or regulatory purposes
3. Functional Requirements
    - Patient Management: Register new patients, update existing profiles, and retrieve patient details.
    - Appointment Management: Book, reschedule, and cancel appointments while checking staff availability.
    - Medical Records: View, update, and securely store medical histories and treatment plans.
    - Staff Management:Add and manage staff roles, schedules, and patient assignments.
    - Billing System: Create invoices, process payments, manage insurance details, and generate financial reports.
    - Search and Filter Functions: Allow users to search and filter records (e.g., patient names, appointment dates).
4. Non-functional Requirements
    - Performance: Ensure fast response times
    - Scalability: Handle a large number of users and data entries concurrently across departments.
    - Security: Encrypt sensitive data and implement role-based access control
    - Usability: Provide a user-friendly, intuitive interface for both technical and non-technical users.
    - Reliability:Ensure the system is available 99.9% of the time with robust error handling.
    - Compliance: Adhere to relevant health regulations and health privacy laws.

CONCEPTUAL DESIGN OF THE DATABASE

ER Diagram:



Data Dictionary:

Patients: Primary Key: Patient_id

Patient_id	First_name	Last_name	Date_of_birth	Gender	Contact_info	Address	Insurance_info	Medical_history

- Patient_id: identifier for each patient (integer, not NULL)
- First_name, Last_name: patient's name (string, not NULL)

- Date_of_birth: Patient's date of birth (date, not NULL)
- Gender: (string, NULL allowed)
- Contact_info: patient's contact details (string, not NULL)
- Address: Patient's residential address (string, not NULL)
- Inusrance_info: patient's insurance details (string, NULL allowed)
- Medical_history: general summary of patient's medical history (text, NULL allowed)

Staff: Primary Key: Staff_id

Staff_id	First_name	Last_name	Role	Department	Contact_info	Availability	Assigned_patients

- Staff_id: unique identifier for each staff member (integer, not NULL)
- First_name, Last_name: staff member's names (string, not NULL)
- Role: role of the staff (string, not NULL)
- Department: department each staff member is assigned to (string, not NULL)
- Contact_info: staff contact information (string, not NULL)
- Availability: available schedule for each staff member (string, not NULL)
- Assigned_patients: number of patients assigned to a staff member (integer, NULL allowed)

Appointments: Primary Key: Appointment_id, Foreign Keys: Patient_id, Staff_id

Appointment_id	Patient_id	Staff_id	Date	Time	Reason	Status

- Appointment_id: unique identifier for the appointment (integer, not NULL)
- Patient_id: refers to Patient.patient_id (integer, foreign key, not NULL)
- Staff_id: refers to Staff.staff_id (integer, foreign key, not NULL)
- Date: appointment date (date, not NULL)
- Time: appointment time (time, not NULL)
- Reason: reason for appointment (string, not NULL)
- Status: status of the appointment (scheduled, completed, or canceled) (string, not NULL)

Foreign Key Actions:

- Patient_id: cascade on delete (if patient record is deleted, delete the appointment)
- Staff_id: set NULL on delete (if staff record is deleted, the appointment remains but without the assigned staff)

Medical Records: Primary Key: Record_id, Foreign Keys: Patient_id

Record_id	Patient_id	Diagnosis	Treatment	Doctor_notes	Medications	Record_date

- Record_id: unique identifier for the medical record (integer, not NULL, auto-increment)
- Patient_id: refers to Patient.patient_id (integer, foreign key, not NULL)
- Diagnosis: patient's diagnosis (sting, not NULL)
- Treatment: treatment provided (string, not NULL)
- Doctor_notes: doctor's notes (text, not NULL)
- Medications: list of medications prescribed (sting, NULL allowed)
- Record_date: date of record creation (date, not NULL)

Foreign Key Actions:

- Patient_id: cascade on delete (deleting a patient also deletes related medical history)

Billing: Primary Key: Billing_id, Foreign Keys: Patient_id

Billing_id	Patient_id	Total_cost	Insurance	Coverage	Payment_recieved	Outstanding_balance	Billing_date

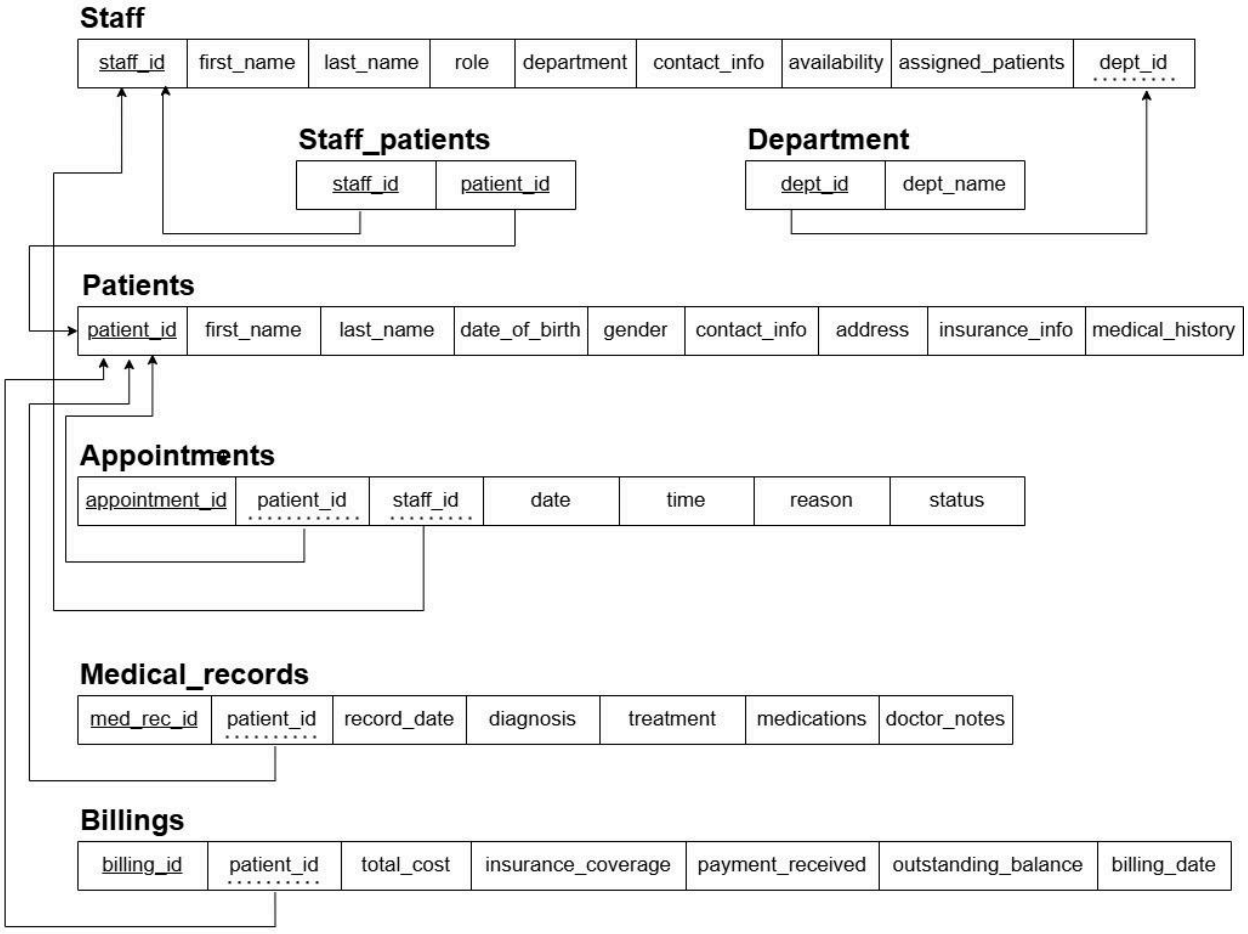
- Billing_id: unique identifier for the billing record (integer, not NULL, auto-increment)
- Patient_id: refers to Patient.patient_id (integer, foreign key, not NULL)
- Total_cost: total costs of services (decimal, not NULL)
- Insurance: amount covered by insurance (decimal, NULL allowed)
- Coverage: amount covered by insurance (decimal, NULL allowed)
- Payment_recieved: payment received (decimal, NULL allowed)
- Outstanding_balance: outstanding balance after payment is received (decimal, not NULL)
- Billing_date: date of billing issued (date, not NULL)

Foreign Key Actions:

- Patient_id: cascade on delete (deleting a patient also deletes related billings issued)

LOGICAL DATABASE SCHEMA

Convert the ER diagram into a relational database schema



Conceptual Design of the Database

- **Entities and referential constraints between tables :**
  - **Patients Table**
    - Attributes: Patient_id, First_name, Last_name, Date_of_birth, Gender, Contact_info, Address, Insurance_info, Medical_history
    - Primary Key: Patient_id
  - **Staff Table**
    - Attributes: Staff_id, First_name, Last_name, Role, Department, Contact_info, Availability, Assigned_patients
    - Primary Key: Staff_id
  - **Appointments Table**
    - Attributes: Appointment_id, Patient_id, Staff_id, Date, Time, Reason, Status
    - Primary Key: Appointment_id
    - Foreign Keys: Patient_id (references Patients.Patient_id), Staff_id (references Staff.Staff_id)

- **Medical_Records Table**
  - Attributes: Record_id, Patient_id, Diagnosis, Treatment, Doctor_notes, Medications, Record_date
  - Primary Key: Record_id
  - Foreign Key: Patient_id (references Patients.Patient_id)
- **Billing Table**
  - Attributes: Billing_id, Patient_id, Total_cost, Insurance, Coverage, Payment_received, Outstanding_balance, Billing_date
  - Primary Key: Billing_id
  - Foreign Key: Patient_id (references Patients.Patient_id)

- **Relationships and (Min, Max) Constraints:**
  - **Patient** :
    - Attend : (1, N) - A patient must attend at least one appointment (total participation), but can attend many appointments.
    - Pay : (1, N) - Each patient must have at least one billing record, but can have multiple billing records.
  - **Staff** :
    - Belong_To : (1, 1) - Every staff member must belong to one department (total participation), but a staff member can belong to only one department at a time.
    - See : (0, N) - A staff member may not see any patients, but can see many patients.
    - Make : (0, N) - A staff member may not make any appointments, but can make multiple appointments.
  - **Appointments** :
    - Attend : (1, 1) - Each appointment must have exactly one patient (total participation), but an appointment cannot have more than one patient.
    - Make : (1, 1) - Each appointment must be made by exactly one staff member, but an appointment cannot involve more than one staff member.
  - **Medical Records** :
    - Have : (1, 1) - Every medical record must belong to exactly one patient, but a medical record cannot belong to more than one patient.
    - Edit : (0, N) - A staff member may not edit any medical records, but can edit multiple medical records.
    -
  - **Billings** :

- Pay : (1, 1) - Every billing record must be associated with exactly one patient, but a billing record cannot belong to more than one patient.

The SQL Statements to Construct the Schema

- **Patients Table:**

```
CREATE TABLE Patients (
  Patient_id  int NOT NULL AUTO_INCREMENT,
  First_name  varchar(45) NOT NULL,
  Last_name  varchar(45) NOT NULL,
  Date_of_birth  date NOT NULL,
  Gender  varchar(45) DEFAULT NULL,
  Contact_info varchar(45) NOT NULL,
  Address  varchar(45) NOT NULL,
  Insurance_info  varchar(45) NOT NULL,
  Medical_history  longtext,
  PRIMARY KEY (Patient_id)
```

- **Staff Table:**

```
CREATE TABLE Staff (
  Staff_id  int NOT NULL AUTO_INCREMENT,
  First_name  varchar(45) NOT NULL,
  Last_name  varchar(45) NOT NULL,
  Role  varchar(45) NOT NULL,
  Department  varchar(45) NOT NULL,
  Contact_info  varchar(45) NOT NULL,
```

Availability varchar(45) NOT NULL,

Assigned_patients mediumtext NOT NULL,

PRIMARY KEY (Staff_id)

- **Appointments Table:**

CREATE TABLE Appointments (

Appointment_id int NOT NULL AUTO_INCREMENT,

Patient_id int NOT NULL,

Staff_id int NOT NULL,

Date date NOT NULL,

Time time NOT NULL,

Reason longtext,

Status varchar(45) NOT NULL,

PRIMARY KEY (Appointment_id),

KEY Patient_id_idx (Patient_id),

KEY Staff_id_idx (Staff_id),

CONSTRAINT Patient_id FOREIGN KEY (Patient_id) REFERENCES Patients (Patient_id),

CONSTRAINT Staff_id FOREIGN KEY (Staff_id) REFERENCES Staff (Staff_id)

- **Medical Records Table:**

CREATE TABLE Medical_Records (

Record_id int NOT NULL AUTO_INCREMENT,

Patient_id int NOT NULL,

Diagnosis longtext NOT NULL,

Treatment longtext NOT NULL,

Doctor_notes longtext NOT NULL,

Medications longtext NOT NULL,

Record_date date NOT NULL,

PRIMARY KEY (Record_id),

KEY Patient_id_idx (Patient_id)

- **Billing Table:**

  CREATE TABLE Billing (

  Billing_id int NOT NULL AUTO_INCREMENT,

  Patient_id int NOT NULL,

  Total_cost decimal(10,2) NOT NULL,

  Insurance varchar(45) NOT NULL,

  Coverage varchar(45) NOT NULL,

  Payment_received decimal(10,2) NOT NULL,

  Outstanding_balancedecimal(10,2) NOT NULL,

  Billing_date date NOT NULL,

  PRIMARY KEY (Billing_id),

  KEY Patient_id_idx (Patient_id)

The expected database operations

- **Insert Operations:**
  - Add new data or information to Patients, Staff, Appointments, Medical Records, and Billing tables.
- **Update Operations:**
  - Modify existing entries in tables:
    - Update patient contact information.
    - Update appointment status.
    - Update billing payment status.
- **Remove Operations:**

        - Remove outdated or unnecessary records:
            - Cancel appointments (delete entries).
            - Archive medical records for inactive patients.
- **Query Operations:**
    - Retrieve and analyze data.
        - Viewing a patient's medical history.
        - Generating billing summaries.
- **Reports:**
    - Generate and summarize reports.
        - Outstanding billing and payment status reports.

Estimate data volumes

- **Patients :**
    - Approximately 10,000 rows for a medium-sized hospital, with patient data growing steadily over time.
- **Staff :**
    - Approximately 500 rows, including doctors, nurses and administrative staff.
- **Appointments :**
    - Approximately 50,000 rows per year, assuming 100 appointments daily on average.
- **Medical Records :**
    - Estimated 30,000 rows, considering multiple visits per patient.
- **Billing :**
    - Approximately 20,000 rows annually, linked directly to patient appointments.

FUNCTIONAL DEPENDENCIES & DATABASE NORMALIZATION

Functional Dependencies:

- **Patients Table**: Patient_id → First_name, Last_name, Date_of_birth, Gender, Contact_info, Address, Insurance_info, Medical_history
- **Staff Table**: Staff_id → First_name, Last_name, Role, Department, Contact_info, Availability, Assigned_patients
- **Appointments Table**: Appointment_id → Patient_id, Staff_id, Date, Time, Reason, Status
- **Medical_Records Table**: Record_id → Patient_id, Diagnosis, Treatment, Doctor_notes, Medications, Record_date

- **Billing Table**: Billing_id → Patient_id, Total_cost, Insurance, Coverage, Payment_received, Outstanding_balance, Billing_date

Third Normal Form (3NF) Normalization:

- **Patients Table**: The primary key Patient_id uniquely determines all other attributes with no partial or transitive dependencies. Therefore, the table is in 3NF
- **Staff Table**: The primary Key Staff_id uniquely determines each attribute. There are no partial or transitive dependencies, so this is in 3NF
- **Appointments Table**: The primary key Appointment_id uniquely determines all other attributes, with no partial or transitive dependencies, therefore is in 3NF
- **Medical_Records Table**: The primary Key Record_id uniquely determines each attribute, and there are no partial or transitive dependencies, meaning this table is in 3NF
- **Billing Table**: The primary key Billing_id uniquely determines each attribute, and there are no partial and transitive dependencies, therefore is in 3NF

All tables in the schema meet the requirements for 3NF, with no further decomposition necessary

THE DATABASE SYSTEM

Install the database

1. Install the database management system (DBMS)
    - Download and install a DBMS such as MySQL and connect to the server. Ensure that the MySQL server is running.
2. Create the database
    - Create a Table for all entities.

```sql
CREATE TABLE `Patients` (
  `Patient_id` int NOT NULL AUTO_INCREMENT,
  `First_name` varchar(45) NOT NULL,
  `Last_name` varchar(45) NOT NULL,
  `Date_of_birth` date NOT NULL,
  `Gender` varchar(45) DEFAULT NULL,
  `Contact_info` varchar(45) NOT NULL,
  `Address` varchar(45) NOT NULL,
  `Insurance_info` varchar(45) NOT NULL,
  `Medical_history` longtext,
  PRIMARY KEY (`Patient_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```sql
CREATE TABLE `Medical_Records` (
  `Record_id` int NOT NULL AUTO_INCREMENT,
  `Patient_id` int NOT NULL,
  `Diagnosis` longtext NOT NULL,
  `Treatment` longtext NOT NULL,
  `Doctor_notes` longtext NOT NULL,
  `Medications` longtext NOT NULL,
  `Record_date` date NOT NULL,
  PRIMARY KEY (`Record_id`),
  KEY `Patient_id_idx` (`Patient_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```sql
CREATE TABLE `Staff` (
  `Staff_id` int NOT NULL AUTO_INCREMENT,
  `First_name` varchar(45) NOT NULL,
  `Last_name` varchar(45) NOT NULL,
  `Role` varchar(45) NOT NULL,
  `Department` varchar(45) NOT NULL,
  `Contact_info` varchar(45) NOT NULL,
  `Availability` varchar(45) NOT NULL,
  `Assigned_patients` mediumtext NOT NULL,
  PRIMARY KEY (`Staff_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

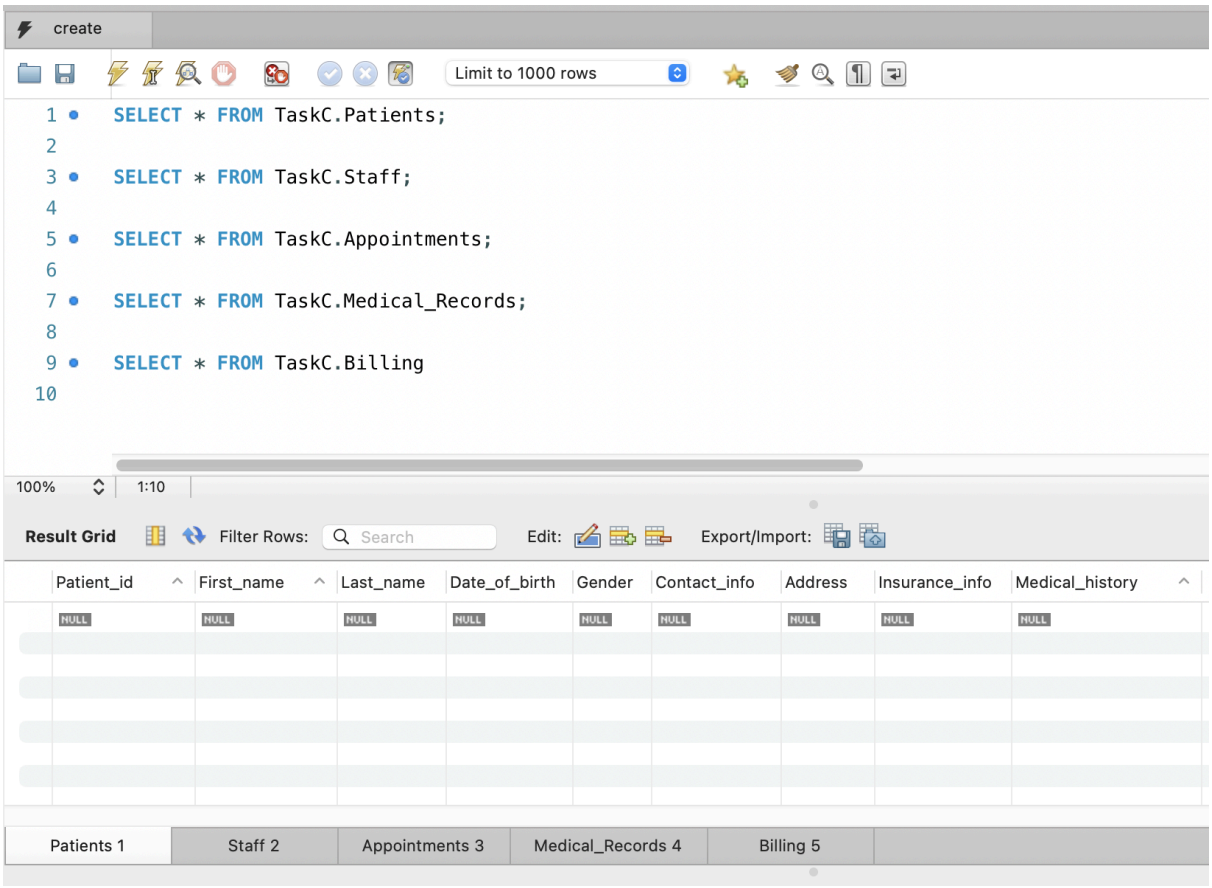
```sql
CREATE TABLE `Appointments` (
  `Appointment_id` int NOT NULL AUTO_INCREMENT,
  `Patient_id` int NOT NULL,
  `Staff_id` int NOT NULL,
  `Date` date NOT NULL,
  `Time` time NOT NULL,
  `Reason` longtext,
  `Status` varchar(45) NOT NULL,
  PRIMARY KEY (`Appointment_id`),
  KEY `Patient_id_idx` (`Patient_id`),
  KEY `Staff_id_idx` (`Staff_id`),
  CONSTRAINT `Patient_id` FOREIGN KEY (`Patient_id`) REFERENCES `Patients` (`Patient_id`),
  CONSTRAINT `Staff_id` FOREIGN KEY (`Staff_id`) REFERENCES `Staff` (`Staff_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

```sql
CREATE TABLE `Billing` (
  `Billing_id` int NOT NULL AUTO_INCREMENT,
  `Patient_id` int NOT NULL,
  `Total_cost` decimal(10,2) NOT NULL,
  `Insurance` varchar(45) NOT NULL,
  `Coverage` varchar(45) NOT NULL,
  `Payment_received` decimal(10,2) NOT NULL,
  `Outstanding_balance` decimal(10,2) NOT NULL,
  `Billing_date` date NOT NULL,
  PRIMARY KEY (`Billing_id`),
  KEY `Patient_id_idx` (`Patient_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3
```

Invoke the system

1. Execute the SQL schema Script After creating a table database, click Execute to save the SQL schema Script and save it as the create.sql file. All SQL scripts are included in the create.sql files within the SQL zip file. (Reference Appendix A)

- Click Execute to show the tables were created. The empty table will be shown in the following screenshot.



2. Adding data to the system

- Use SQL insert commands to fill the data into tables.
- Insert data to Patient_table.

```
INSERT INTO `TaskC`.`Patients` (
`Patient_id`, `First_name`, `Last_name`, `Date_of_birth`, `Gender`, `Contact_info`,
`Address`, `Insurance_info`, `Medical_history`)
VALUES ('0001', 'Henry', 'Tomson', '1991-03-01', 'M', '123-4567', '123 Oak St, City', 'Cigna', 'Allergy to peanut');
INSERT INTO `TaskC`.`Patients` (`Patient_id`, `First_name`, `Last_name`, `Date_of_birth`,
`Gender`, `Contact_info`, `Address`, `Insurance_info`, `Medical_history`)
VALUES ('0002', 'JC', 'Kind', '1986-02-01', 'M', '456-7890', '456 Stem St, City', 'Atena', 'Allergy to honey');
```

- Insert data to Staff_table.

```
INSERT INTO `TaskC`.`Staff` (
`Staff_id`, `First_name`, `Last_name`, `Role`, `Department`,
`Contact_info`, `Availability`, `Assigned_patients`)
VALUES ('1', 'Tim', 'Johnson', 'Doctor', 'General', '555-8765', 'Mon-Fri', '1,2');
INSERT INTO `TaskC`.`Staff` (
`Staff_id`, `First_name`, `Last_name`, `Role`, `Department`,
`Contact_info`, `Availability`, `Assigned_patients`)
VALUES ('2', 'Bob', 'Pepper', 'Nurse', 'Pediatrics', '555-4321', 'Tue-Thu', '1');
```

- Insert data to Appointment_table.

```
INSERT INTO `TaskC`.`Appointments` (`Appointment_id`, `Patient_id`, `Staff_id`, `Date`, `Time`,
`Reason`, `Status`) VALUES ('1', '1', '1', '2024-01-10', '9:00', 'Regular Checkup', 'Completed');
INSERT INTO `TaskC`.`Appointments` (`Appointment_id`, `Patient_id`, `Staff_id`, `Date`, `Time`,
`Reason`, `Status`) VALUES ('2', '2', '2', '2024-03-15', '14:30', 'Follow-up', 'Pending');
```

- Insert data to Medical_Records_table.

```
INSERT INTO `TaskC`.`Medical_Records` (`Record_id`, `Patient_id`, `Diagnosis`, `Treatment`,
`Doctor_notes`, `Medications`, `Record_date`) VALUES ('1', '1', 'Hypertension', 'Medication Adjusted',
'Monitor blood pressure daily', 'ACE inhibitors', '2024-01-10');
INSERT INTO `TaskC`.`Medical_Records` (`Record_id`, `Patient_id`, `Diagnosis`, `Treatment`,
`Doctor_notes`, `Medications`, `Record_date`) VALUES ('2', '2', 'Diabetes', 'Insulin Therapy',
'Regular glucose monitoring', 'Insulin', '2024-03-15');
```

- Insert data to Billing_table.

```
SELECT * FROM TaskC.Billing;

SELECT Billing_id FROM TaskC.Billing;
ALTER TABLE TaskC.Billing MODIFY Billing_id INT AUTO_INCREMENT;

INSERT INTO TaskC.Billing (Billing_id, Patient_id, Total_cost, Insurance, Coverage, Payment_received,
Outstanding_balance, Billing_date)
VALUES (1, 1, 300, 'Cigna', 0.90, 0.9 * 300, 300 - (0.9 * 300), '2024-01-10');

INSERT INTO TaskC.Billing (Billing_id, Patient_id, Total_cost, Insurance, Coverage, Payment_received,
Outstanding_balance, Billing_date)
VALUES (2, 2, 500, 'Atena', 0.80, 0.8 * 500 , 500 - (0.9 * 500), '2024-03-15');
```

3. Click Execute to apply the data updates to the tables. The table will display the output as shown below.
    - The output of inserting data into the Patients_Table.

```sql
SELECT * FROM TaskC.Billing;

SELECT Billing_id FROM TaskC.Billing;
ALTER TABLE TaskC.Billing MODIFY Billing_id INT AUTO_INCREMENT;

INSERT INTO TaskC.Billing (Billing_id, Patient_id, Total_cost, Insurance, Coverage, Payment_received,
Outstanding_balance, Billing_date)
VALUES (1, 1, 300, 'Cigna', 0.90, 0.9 * 300, 300 - (0.9 * 300), '2024-01-10');

INSERT INTO TaskC.Billing (Billing_id, Patient_id, Total_cost, Insurance, Coverage, Payment_received,
Outstanding_balance, Billing_date)
VALUES (2, 2, 500, 'Atena', 0.80, 0.8 * 500 , 500 - (0.9 * 500), '2024-03-15');
```

Patient_id	First_name	Last_name	Date_of_birth	Gender	Contact_info	Address	Insurance_info	Medical_history
1	Henry	Tomson	1991-03-01	M	123-4567	123 Oa…	Cigna	Allergy to peanut
2	JC	Kind	1986-02-01	M	456-7890	456 Ste…	Atena	Allergy to honey
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Patients 1 | Staff 2 | Appointments 3 | Medical_Records 4 | Billing 5

- The output of inserting data into the Staff_Table.

Staff_id	First_name	Last_name	Role	Department	Contact_info	Availability	Assigned_patients
1	Tim	Johnson	Doctor	General	555-8765	Mon-Fri	1,2
2	Bob	Pepper	Nurse	Pediatrics	555-4321	Tue-Thu	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Patients 1 | Staff 2 | Appointments 3 | Medical_Records 4 | Billing 5

- The output of inserting data into the Appointments_Table.

Appointment_id	Patient_id	Staff_id	Date	Time	Reason	Status
1	1	1	2024-01-10	09:00:...	Regular Checkup	Completed
2	2	2	2024-03-15	14:30:...	Follow-up	Pending
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Patients 1 | Staff 2 | Appointments 3 | Medical_Records 4 | Billing 5

- The output of inserting data into the Medical_Records_Table.

Record_id	Patient_id	Diagnosis	Treatment	Doctor_notes	Medications	Record_date
1	1	Hypertension	Medication Adjusted	Monitor blood pressure daily	ACE inhibitors	2024-01-10
2	2	Diabetes	Insulin Therapy	Regular glucose monitoring	Insulin	2024-03-15
NULL	NULL	NULL	NULL	NULL	NULL	NULL

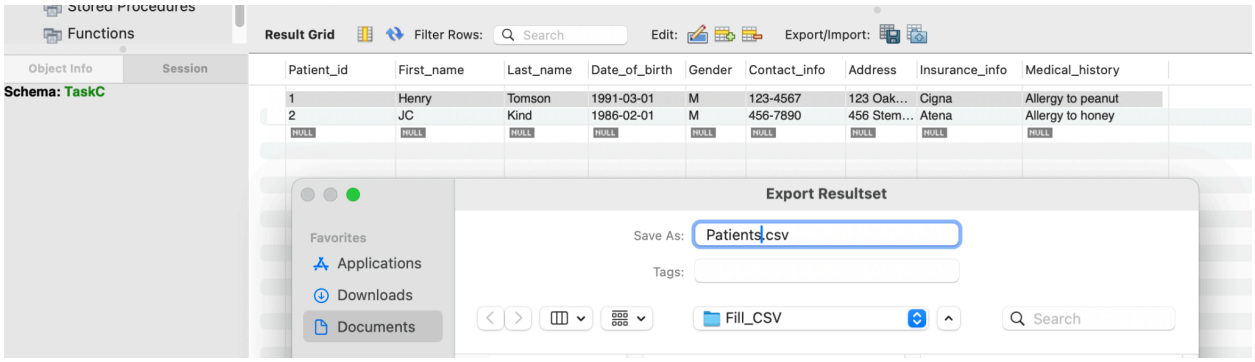
Patients 1 | Staff 2 | Appointments 3 | Medical_Records 4 | Billing 5

- The output of inserting data into the Billing_Table.

Billing_id	Patient_id	Total_cost	Insurance	Coverage	Payment_received	Outstanding_balance	Billing_date
1	1	300.00	Cigna	0.90	270.00	30.00	2024-01-10
2	2	500.00	Atena	0.80	400.00	50.00	2024-03-15
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Patients 1 | Staff 2 | Appointments 3 | Medical_Records 4 | Billing 5

4. Click Export to generate the CSV files. All CSV files are included in the SQL zip file. (Reference Appendix A)



- The output of the Patients_Table, with data generated into the CSV files.

Patients

Patient_id	First_name	Last_name	Date_of_birth	Gender	Contact_info	Address	Insurance_info	Medical_history
1	Henry	Tomson	1991-03-01	M	123-4567	123 Oak St, City	Cigna	Allergy to peanut
2	JC	Kind	1986-02-01	M	456-7890	456 Stem St, City	Atena	Allergy to honey

- The output of the Staff_Table, with data generated into the CSV files.

Staff

Staff_id	First_name	Last_name	Role	Department	Contact_info	Availability	Assigned_patients
1	Tim	Johnson	Doctor	General	555-8765	Mon-Fri	1,2
2	Bob	Pepper	Nurse	Pediatrics	555-4321	Tue-Thu	1

- The output of the Appointment_Table, with data generated into the CSV files.

Appointments

Appointment_id	Patient_id	Staff_id	Date	Time	Reason	Status
1	1	1	2024-01-10	09:00:00	Regular Checkup	Completed
2	2	2	2024-03-15	14:30:00	Follow-up	Pending

- The output of the Medical_Records_Table, with data generated into the CSV

files.

Medical_Records

Record_id	Patient_id	Diagnosis	Treatment	Doctor_notes	Medications	Record_date
1	1	Hypertension	Medication Adjusted	Monitor blood pressure daily	ACE inhibitors	2024-01-10
2	2	Diabetes	Insulin Therapy	Regular glucose monitoring	Insulin	2024-03-15

- The output of the Billing_Table, with data generated into the CSV files.

Billing

Billing_id	Patient_id	Total_cost	Insurance	Coverage	Payment_received	Outstanding_balance	Billing_date
1	1	300.00	Cigna	0.90	270.00	30.00	2024-01-10
2	2	500.00	Atena	0.80	400.00	50.00	2024-03-15

5. To load data in bulk by using a csv file to upload data into MySQL. All SQL scripts are included in the load.sql files within the SQL zip file. (Reference Appendix A)

   5.1 Create the table in the database to load the data.

   - Create Loaddata schema.
   - Create the Patients_data table to prepare for loading the CSV file.

```sql
CREATE DATABASE IF NOT EXISTS Loaddata;

USE Loaddata;

DROP TABLE IF EXISTS Patients_data;

CREATE TABLE Patients_data
(   Patient_id INT PRIMARY KEY,
    First_name VARCHAR(255),
    Last_name VARCHAR(255),
    Date_of_birth DATE,
    Gender VARCHAR(255),
    Contact_info VARCHAR(255),
    Address VARCHAR(255),
    Insurance_info VARCHAR(255),
    Medical_history TEXT
);

```

5.2 Save the CSV file (Patient_data.csv) to the SQL library on your computer to prepare for loading the data.

5.3 Select All Records from the Patients_data Table.

- Using command SELECT * FROM Patients_data. This SQL query retrieves and displays all rows and columns from the Patients_data table.

5.4 Load Data into Patients_data Table.

- Using command LOAD DATA INFILE to load data from the CSV file Patient_data.csv into the the Patients_data table.

5.5 Verify Data in the Patients_data Table.

- Using command SELECT * FROM Patients_data. This SQL query is run again to display all rows and columns from the Patients_data table.
- The data from the CSV file has been successfully inserted into the table, as shown below.

```
19  SELECT * FROM Patients_data;
20
21
22  LOAD DATA INFILE 'Patient_data.csv' INTO TABLE Patients_data
23  FIELDS TERMINATED BY ','
24  ENCLOSED BY '"'
25  LINES TERMINATED BY '\n'
26  IGNORE 1 LINES;
27
28  SELECT * FROM Patients_data
29
```

1:29

Result Grid Filter Rows: Search Edit: Export/Import:

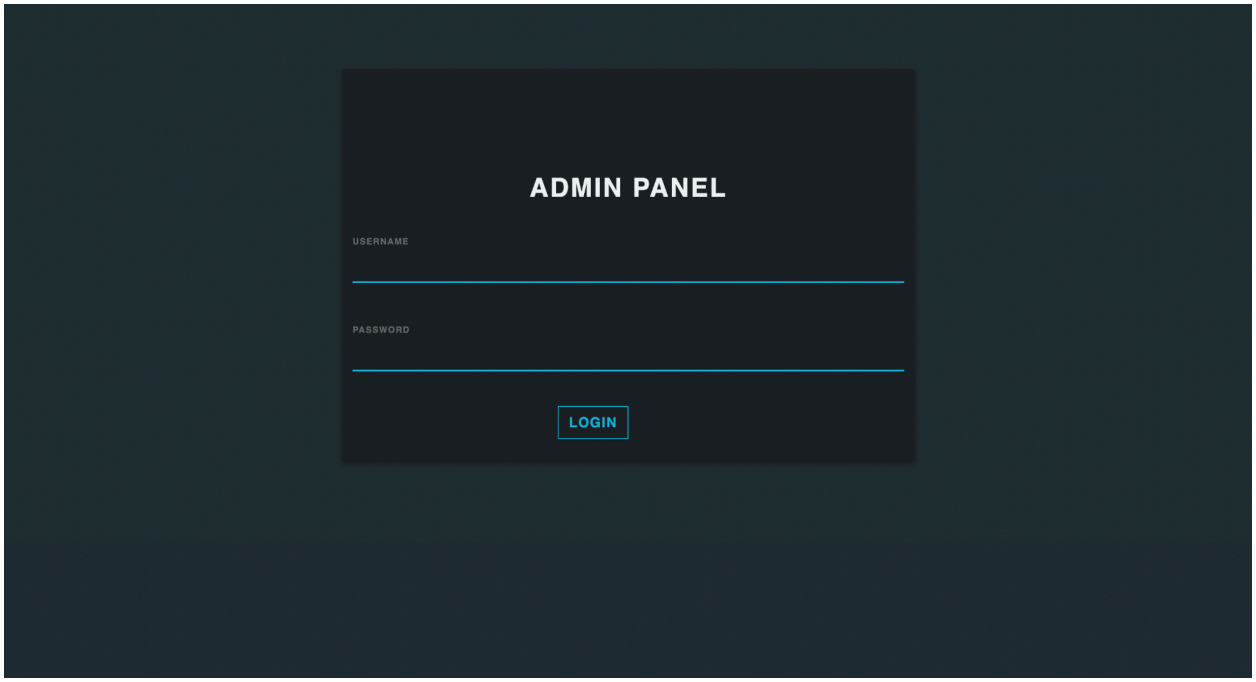
Patient_id	First_name	Last_name	Date_of_birth	Gender	Contact_info	Address	Insurance_info	Medical_history
1	A	Arron	1990-03-01	M	555-1234	123 Main St City	Aetna	Persistent pain
2	B	Barron	1985-02-14	F	555-5678	456 Debby St City	Blue Cross	Allergy to penicillin
3	C	Cane	1980-01-31	M	555-10122	124 Main St City	Cigna	Depression
4	D	Diddy	1975-01-16	F	555-14566	457 Debby St City	Aetna	Diabetes
5	E	Eel	1970-01-01	M	555-19010	125 Main St City	Blue Cross	Sleep disturbances
6	F	Fantacy	1964-12-17	F	555-23454	458 Debby St City	Cigna	migraines
7	G	Gilbert	1959-12-03	M	555-27898	126 Main St City	Aetna	Joint pain
8	H	Hamington	1954-11-18	F	555-32342	459 Debby St City	Blue Cross	seizures
9	I	Inner	1949-11-03	M	555-36786	127 Main St City	Cigna	None

Patients_data 9

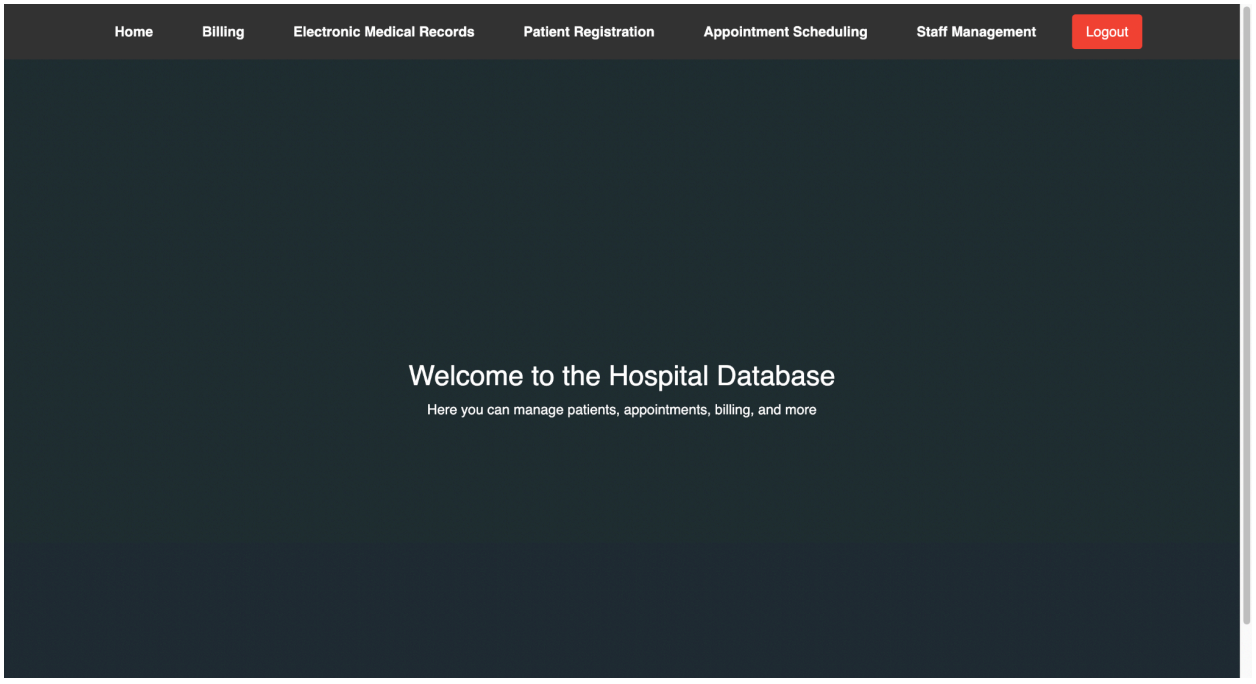
USER APPLICATION INTERFACE

To build the user interface for this system I used JavaScript React. The different tables from the database were broken down into different components to keep everything relatively organized and easier to debug. The App.js file is the main component and has all the other components linked to it via import statements. I then systematically went through each component and built the functionality needed. The components use UseEffect and UseState to reflect user input. They use CRUD to create, edit and delete entries for the website. For the actual UX I used bootstrap and CSS to create a visually appealing design. In order to create an easy to follow and easy to navigate website, all of the buttons and text fields use helpful descriptive text and there are error prompts as well as success prompts to give the user feedback.

To invoke and use the webpage you'll need to follow the directions listed in The Database System section. Once everything is connected and running correctly you'll be redirected to a page that populates a login screen for admin.

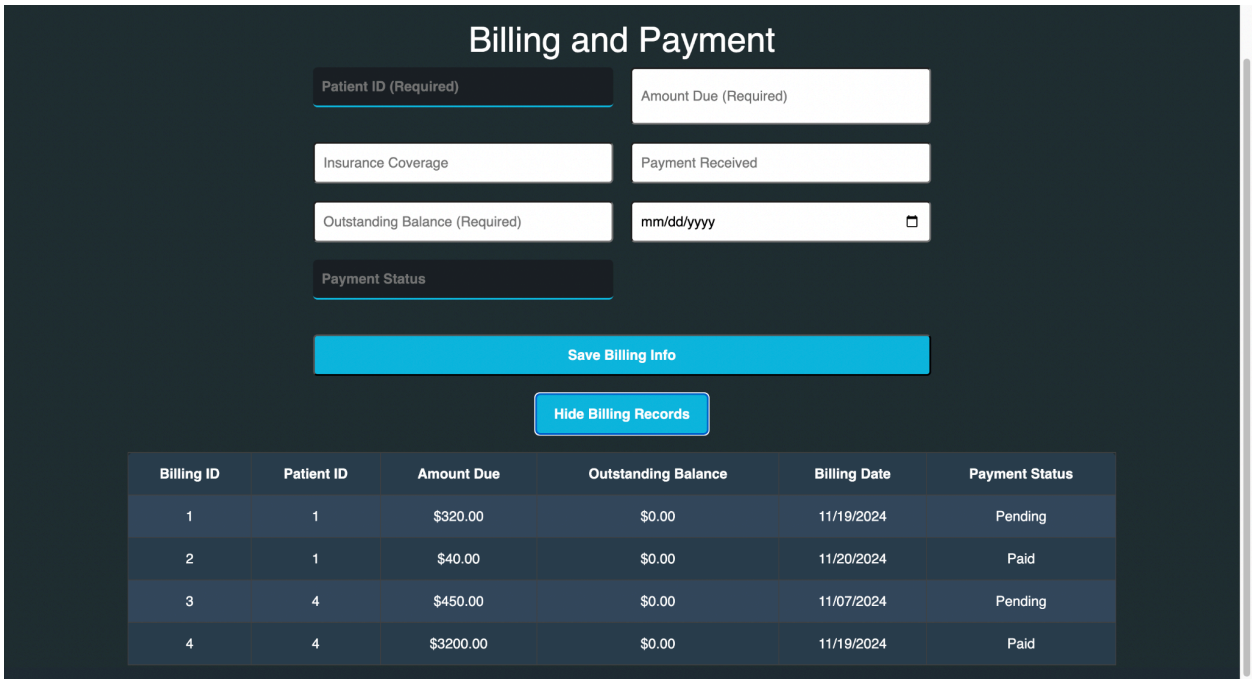


After successful login in you'll see the Home page. Across the top of the screen is the navbar listing the different pages on the site as well as a logout button for when the user has finished using the website.



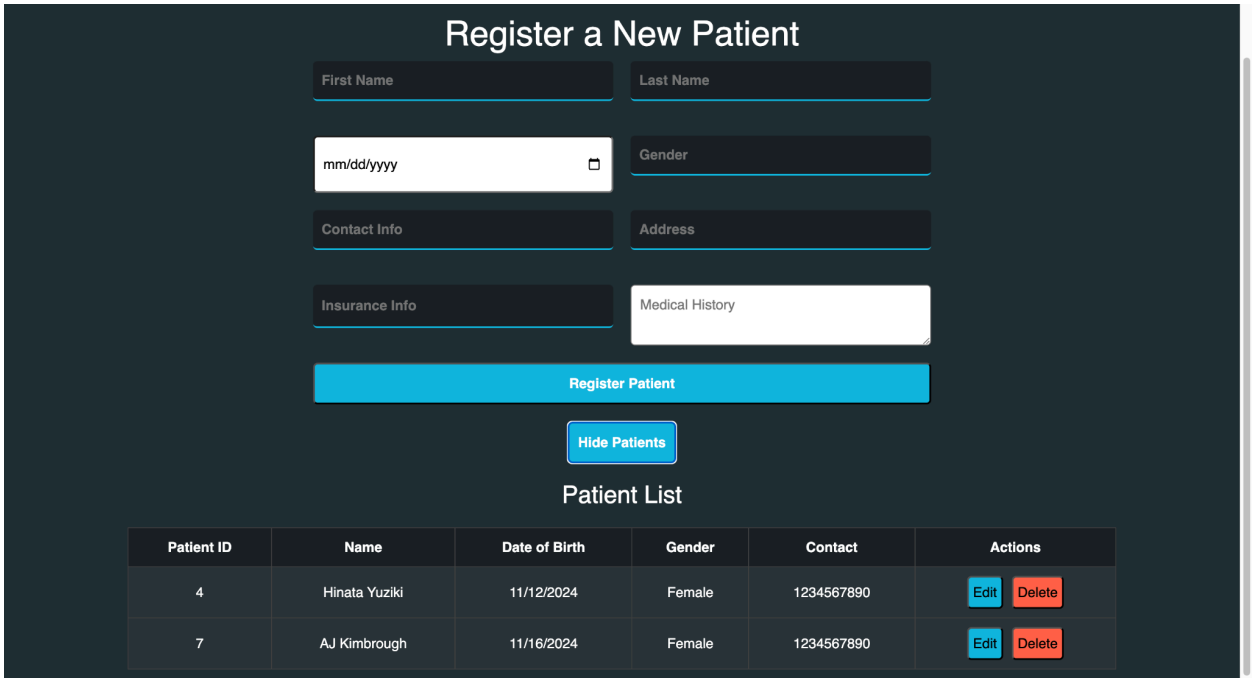
Billing

The Billing page allows the user to create a new bill and view, edit or delete existing bills. To do this simply read the text next to the input fields to fill in the required information. The patient ID can be found by viewing the patient list via the Registration page. After successful completion of the bill, the current bills can be seen by clicking the view button below the creation form. Here the user can delete or edit the bill as needed. Note that even if the user chooses not to change anything after clicking the edit button the date will still need to be updated. This reflects the need to document everything in an actual hospital setting.



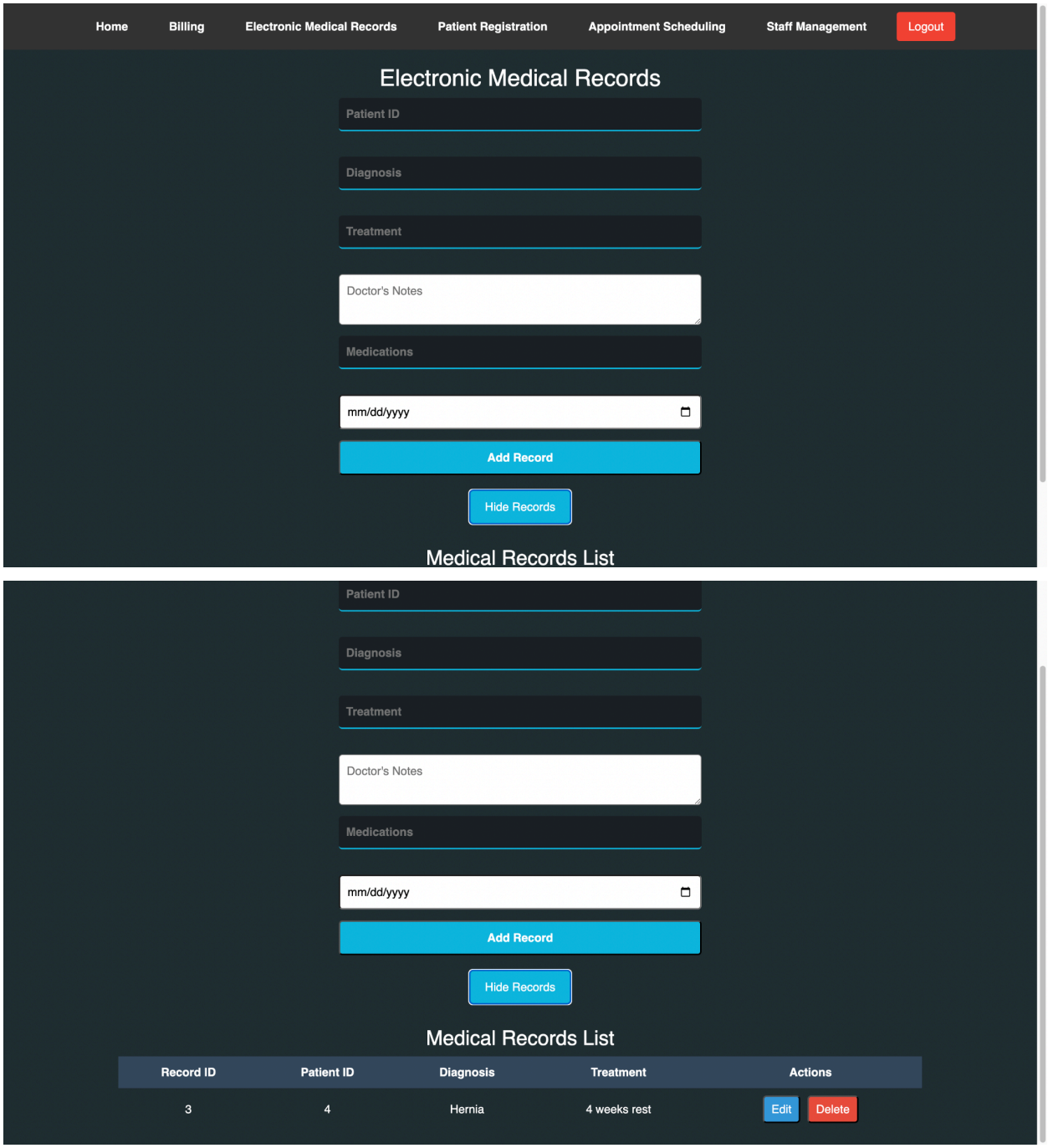
Registration

Registration allows new patients to be registered in the system. Here the user simply reads the text next to the input fields and enters the required information. Note for phone numbers the format does not include any special characters, i.e (,),-. Simply enter the number. For example, 1234567890. The patient list can be seen by clicking the view button under the registration form. Here the patients can be edited or deleted from the system.



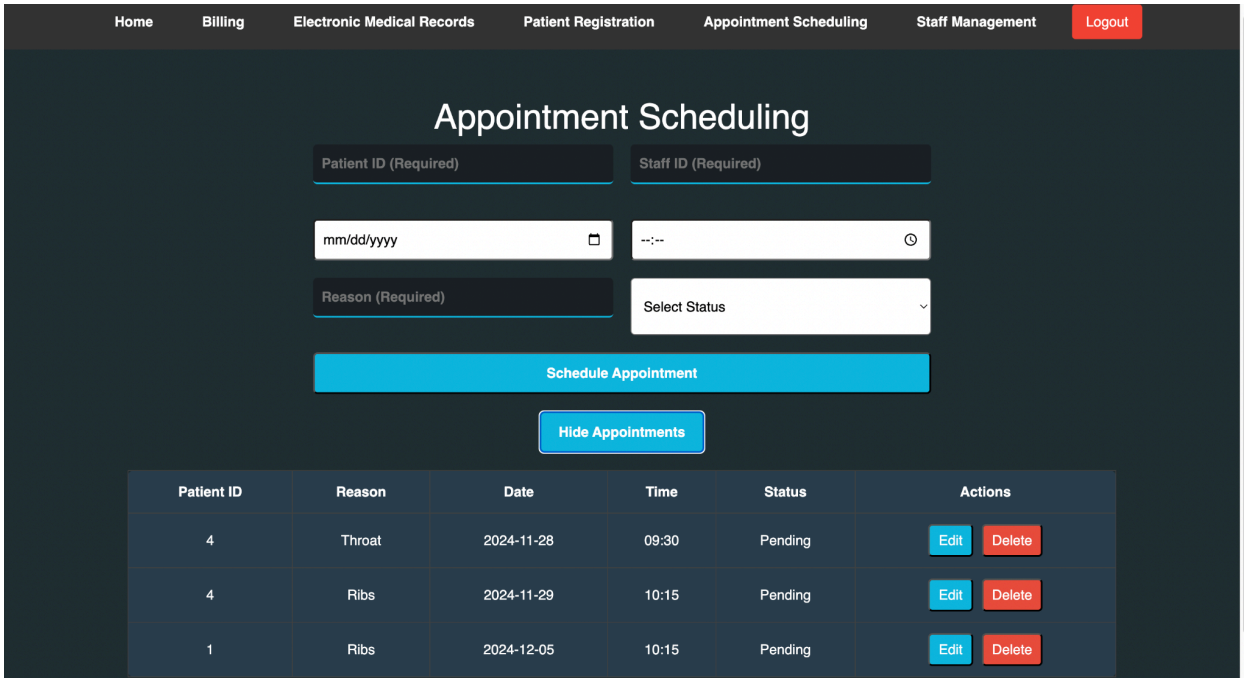
EMR

Electronic Medical Records or EMR allows for the patient record to reflect diagnosis and treatment. Like the other pages the user needs to read the text and fill in the required information. If there are no doctors notes simply input N/A. The records can be viewed by clicking the view button under the form. Here the patient record can be edited or deleted. Like Billing if the edit option is selected the date must be updated, reflecting the need to document in hospitals.



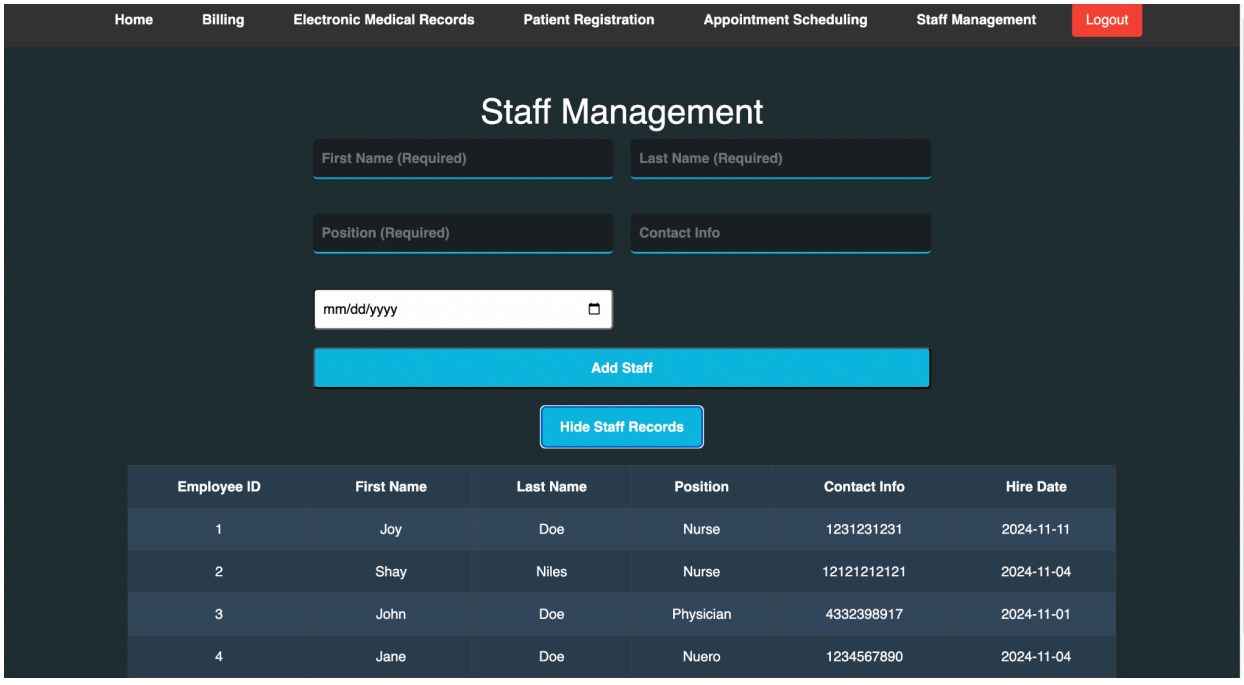
Appointment Scheduling

Appointment Scheduling allows for patient appointments to be scheduled. Here you need the patient ID (found under the patient list on the Registration page) and the ID of the staff member the patient will be seeing (found under view staff on the Staff page). Fill in the required information and the appointment will be made. Current appointments can be viewed, deleted or edited via the view button under the form. Like Billing and EMR if the appointment is edited the date must be updated.



Staff

The staff page allows admin to manage the hospital staff. Here new staff can be added and current staff can be viewed, edited or deleted from the system. To add a new staff member simply read the text displayed next to the input boxes and fill in the required information. You can view the current staff via the view button under the form.



After the user has completed their tasks and is ready to logout of the website simply click on the Logout button in the top right of the screen. This will log the user out and return to the initial Login screen.

Thank you for using the Hospital Database.

CONCLUSION & FUTURE WORK

The development of the **Hospital Management System (HMS)** has successfully addressed the challenges of managing complex hospital workflows. By centralizing processes such as patient registration, appointment scheduling, staff management, medical records handling, and billing, the system enhances operational efficiency, reduces manual errors, and improves the quality of patient care. The implementation of a secure and scalable MySQL database ensures data integrity and compliance with privacy regulations.

Throughout this project, we gained valuable insights into database design, normalization, and the integration of front-end and back-end technologies. The experience also highlighted the importance of user-friendly interfaces in healthcare applications, where usability can significantly impact staff productivity and patient satisfaction

Some future endeavours could be enhancing the security system, or making the system bigger to scale multiple networks. Enhancing the security system could include adding a two-factor authorization to further increase sensitivity and privacy for patients. This would help when expanding the program multiple networks to make sure information is only accessed to those authorized.

REFERENCES

Fundamentals of Database Systems, 7th Edition by Romez Elmasri, Shamkant B, Navathe.

APPENDIX

Appendix A - SQL_zip The SQL zip file contains all SQL files, the source CSV file for loading data, the CSV files for the output, and the ReadMe file.

Appendix B - Hospital_databse_application_zip_. Complete Application, with frontend

and backend files.