I. Introduction

The purpose of this project is to design and implement a comprehensive Healthcare Clinic Management System that leverages the relational database model to efficiently store, manage, and retrieve data related to patients, doctors, appointments, prescriptions, and treatments By guaranteeing data accuracy and expediting administrative duties, the system seeks to enhance the quality of healthcare services. Our project's main goal is to create an intuitive application with distinct roles for patients, doctors, and administrators. Users can interact with the system in accordance with their privileges by registering and logging in securely. Doctors can access patient data and write prescriptions, while administrators can perform SQL statements, manage users, and read results. Patients are able to update their account details and make appointments.

Microsoft SQL Server 2022 was used to build the system's backend, guaranteeing reliable data management and support for more complex features like triggers and views. Because of the frontend's emphasis on accessibility and usability, users of all kinds may navigate and complete activities efficiently. Our system illustrates the useful application of the relational model in actual healthcare situations by using a user-friendly interface with a well-normalized database structure.

**II. Relational Model**

* 1. **Definition:**

The relational model is a mathematically grounded framework for organizing data in a tabular form. Each relation (table) represents an entity or concept; rows (tuples) within a table correspond to individual records, and columns (attributes) define the properties of those records. A primary key uniquely identifies each row in a table, while a foreign key establishes a link from one table to the primary key of another, enforcing referential integrity. By leveraging set theory and first-order logic, the relational model supports powerful, declarative querying (via SQL), strong consistency guarantees, and a clear separation between logical design and physical storage.

* 1. **Database Schema:**

In this part of the report, provide the database schema for the project, along with a brief explanation of each table and its purpose within the project scope.

**1) Relationships:** Explain the key relationships and normalization operations applied in the database design as in the following example:

* + - **One-to-Many Relationship (Patient to Appointment):** A patient can have multiple appointments, while each appointment belongs to exactly one patient. This relationship is established using a Foreign Key constraint in the **Appointment** table referencing the **Patient** table.
    - **One-to-Many Relationship (Doctor to Appointment):** A doctor can have multiple appointments, while each appointment is with exactly one doctor. This relationship is established using a Foreign Key constraint in the **Appointment** table referencing the **Doctor** table.

* + - **One-to-Many Relationship (Doctor to Prescription):** A doctor can write many prescriptions, while each prescription is written by exactly one doctor. This relationship is established using a Foreign Key constraint in the **Prescription** table referencing the **Doctor** table.
    - **One-to-Many Relationship (Patient to Prescription):** A patient can have many prescriptions, while each prescription is for exactly one patient. This relationship is established using a Foreign Key constraint in the **Prescription** table referencing the **Patient** table.
    - **Many-to-Many Relationship (Patient to Treatment):** A patient can receive many treatments, and each treatment can be given to many patients. This relationship is implemented via a junction table (e.g. **Patient, Treatment**) with two Foreign Key constraints—one referencing the **Patient** table and one referencing the **Treatment** table.
    - **One-to-One Relationship (Appointment to Treatment):** Each appointment results in one treatment, and each treatment is tied to exactly one appointment. This relationship is enforced by a Foreign Key constraint in the **Treatment** table referencing the **Appointment** table (together with a unique index on Treatment.AppointmentID).
    - **One-to-One Relationship (Treatment to Bill):** One bill is generated for each treatment, and each bill corresponds to exactly one treatment. This relationship is enforced by a Foreign Key constraint in the **Bill** table referencing the **Treatment** table (together with a unique index on Bill.TreatmentID).
    - **One-to-Many Relationship (Patient to Bill):** A patient can have multiple bills, while each bill belongs to exactly one patient. This relationship is established using a Foreign Key constraint in the **Bill** table referencing the **Patient** table.

**2) Normalization:** Write here the normalization level of each table**.**

* Patient Table

FD: PatientID → FirstName, LastName, DateOfBirth, Gender

Primary Key: PatientID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| PatientID | FirstName | LastName | DateOfBirth | Gender |

This table is in 3NF: all attributes are atomic, multivalued contacts have been moved to PatientContact and every non-key attribute depends only on the primary key.

SQL Statement for Patient Table:

|  |  |
| --- | --- |
| CREATE TABLE Patient ( |  |
| PatientID INT | PRIMARY KEY, |
| FirstName VARCHAR (50) | NOT NULL, |
| LastName VARCHAR (50) | NOT NULL, |
| DateOfBirth DATE | NOT NULL, |
| Gender CHAR (1)  ); | NOT NULL, |

* PatientContact Table

FD: {ContactID} → {PatientID, Phone, Email}

Primary Key: {ContactID}

|  |  |  |  |
| --- | --- | --- | --- |
| ContactID | PatientID | Phone | Email |

This table is in 3NF: it isolates the multivalued phone/email attributes from Patient, and every nonkey attribute depends solely on the surrogate key.

SQL Statement for Patient Table

CREATE TABLE PatientContact (

ContactID INT PRIMARY KEY,

PatientID INT NOT NULL,

Phone VARCHAR(20),

Email VARCHAR(100),

CHECK (Phone IS NOT NULL OR Email IS NOT NULL),

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)

);

* Doctor Table

FD: {DoctorID} → {FirstName, Lastname, Specilization}

Primary Key: {DoctorID}

|  |  |  |  |
| --- | --- | --- | --- |
| DoctorID | FirstName | LastName | Specilization |

This table is in 3NF: multivalued contacts have been moved to DoctorContact, and all non-key attributes depend only on the primary key.

SQL Statement for Doctor Table

CREATE TABLE Doctor (

|  |  |  |
| --- | --- | --- |
| DoctorID | INT | PRIMARY KEY, |
| FirstName | VARCHAR(50) | NOT NULL, |
| LastName | VARCHAR(50) | NOT NULL, |
| Specialization  ); | VARCHAR(100) | NOT NULL |

* DoctorContact Table

FD: {ContactID} → {DoctorID, Phone, Email}

Primary Key: {ContactID}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ContactID | | DoctorID | | Phone | | Email | |
| This table is in 3NF: it captures all phone/email entries for doctors and every nondepends on the surrogate key.    SQL Statement for DoctorContact Table    CREATE TABLE DoctorContact (  ContactID INT PRIMARY KEY,  DoctorID INT NOT NULL,  Phone VARCHAR(20),  Email VARCHAR(100),  CHECK (Phone IS NOT NULL OR Email IS NOT NULL),  FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)  );      • Appointment Table    FD: {AppointmentID} → {PatientID,DoctorID,ApptDate,ApptTime,Status}    Primary Key : {AppointmentID} | | | | | | | key attribute |
| AppointmentID | PatientID | | DoctorID | ApptDate | ApptTime | | Status |

This table is in 3NF: it has atomic attributes, a single-column key, and no transitive dependencies.

SQL Statement for Appointment Table

|  |  |  |
| --- | --- | --- |
| CREATE TABLE Appointment ( |  |  |
| AppointmentID INT |  | PRIMARY KEY, |
| PatientID INT |  | NOT NULL, |
| DoctorID INT |  | NOT NULL, |

ApptDate DATE NOT NULL,

ApptTime TIME NOT NULL,

Status VARCHAR(20) NOT NULL

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)

);

* Prescription Table

FD: {PrescriptionID} → {DoctorID, PatientID, DateIssued, Medication, Dosage, Notes}

Primary Key : {PrescriptionID}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PrescriptionID | DoctorID | PatientID | DateIssued | Medication | Dosage | Notes |

This table is in 3NF: each attribute is fully dependent on the primary key, with no redundancies.

SQL Statement for Prescription Table

CREATE TABLE Prescription (

|  |  |  |  |
| --- | --- | --- | --- |
| PrescriptionID |  | INT | PRIMARY KEY, |
| DoctorID |  | INT | NOT NULL, |
| PatientID |  | INT | NOT NULL, |
| DateIssued |  | DATE | NOT NULL, |
| Medication |  | VARCHAR(100) | NOT NULL, |
| Dosage |  | VARCHAR(50) | NOT NULL, |
| Notes |  | TEXT |  |

FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)

FOREIGN KEY (PatientID) REFERENCES Patient(PatientID)

);

* Treatment Table

FD1: {TreatmentID} → {AppointmentID, TreatmentType, Cost, TreatmentDate}

FD2: {AppointmentID} → {TreatmentID, TreatmentType, Cost, TreatmentDate} (by unique constraint)

Primary Key : {TreatmentID}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| TreatmentID | AppointmentID | TreatmentType | Cost | TreatmentDate |

This table is in 3NF: it enforces a 1–1 link to Appointment via a UNIQUE on AppointmentID and all non-key attributes depend only on TreatmentID.

SQL Statement for Treatment Table

CREATE TABLE Treatment (

TreatmentID INT PRIMARY KEY,

AppointmentID INT NOT NULL UNIQUE REFERENCES Appointment(AppointmentID),

TreatmentType VARCHAR(100) NOT NULL,

Cost DECIMAL(10,2) NOT NULL,

TreatmentDate DATE NOT NULL

);

* Patient\_Treatment Table

FD: {PatientID, TreatmentID} → -

Primary Key: {PatientID, TreatmentID}

|  |  |
| --- | --- |
| PatientID | TreatmentID |

This pure junction table is in 3NF: it contains only key attributes and implements the many-to-many between patients and treatments.

SQL Statement for Patient\_Treatment Table

CREATE TABLE Patient\_Treatment (

PatientID INT NOT NULL REFERENCES Patient(PatientID),

TreatmentID INT NOT NULL REFERENCES Treatment(TreatmentID), PRIMARY KEY (PatientID, TreatmentID)

);

* Bill Table

FD1: {BillID} → {TreatmentID, Amount, DateIssued}

FD2: {TreatmentID} → {BillID, Amount, DateIssued} (by unique constraint)

Primary Key : {BillID}

|  |  |  |  |
| --- | --- | --- | --- |
| BillID | TreatmentID | Amount | DateIssued |

This table is in 3NF: it enforces the 1–1 link to Treatment and all non-key attributes depend solely on BillID.

SQL Statement for Bill Table

CREATE TABLE Bill (

BillID INT PRIMARY KEY,

TreatmentID INT NOT NULL UNIQUE REFERENCES Treatment(TreatmentID),

Amount DECIMAL(10,2) NOT NULL,

DateIssued DATE NOT NULL

);

### III. Database Structures

This section provides a comprehensive overview of the database structures used in the Healthcare Clinic Management System. This project includes core and supporting tables with defined attributes, constraints, and relationships. The system is implemented using MS SQL Server 2022 Developer Edition.

**Table Structures**:

## Patient

* **Columns**: PatientID (PK), FirstName, LastName, DateOfBirth, Gender
* **Constraints**: NOT NULL for all, PK on PatientID

## • Normalization: 3NF

### 2. PatientContact

* **Columns**: ContactID (PK), PatientID (FK), Phone, Email
* **Constraints**: CHECK that at least one of Phone or Email is not NULL.

## • Normalization: 3NF

### 3. Doctor

* **Columns**: DoctorID (PK), FirstName, LastName, Specialization
* **Constraints**: NOT NULL for all fields, PK on DoctorID

## • Normalization: 3NF

### 4. DoctorContact

* **Columns**: ContactID (PK), DoctorID (FK), Phone, Email
* **Constraints**: Same as PatientContact

## • Normalization: 3NF

### 5. Appointment

* **Columns**: AppointmentID (PK), PatientID (FK), DoctorID (FK), ApptDate, ApptTime, Status
* **Constraints**: NOT NULL for all, FK references Patient and Doctor, constraint to prevent overlapping appointments.

## • Normalization: 3NF

### 6. Prescription

* **Columns**: PrescriptionID (PK), DoctorID (FK), PatientID (FK), DateIssued, Medication, Dosage, Notes
* **Constraints**: Only doctors can modify prescriptions

## • Normalization: 3NF

• **Weak Entity Consideration**: Prescription can be considered weak as it depends on Doctor and Patient.

### 7. Treatment

* **Columns**: TreatmentID (PK), AppointmentID (FK, UNIQUE), TreatmentType, Cost, TreatmentDate
* **Constraints**: One-to-one relationship with Appointment via UNIQUE constraint

## • Normalization: 3NF

### 8. Patient\_Treatment (junction table)

* **Columns**: PatientID (FK), TreatmentID (FK)
* **Constraints**: Composite PK (PatientID, TreatmentID)
* **Purpose**: Enforces many-to-many tracking, although only one treatment is allowed per appointment

### 9. Bill

* **Columns**: BillID (PK), TreatmentID (FK, UNIQUE), Amount, DateIssued
* **Constraints**: Enforces one-to-one relationship with Treatment

## • Normalization: 3NF

**Indices:**

* Primary key indices are automatically created on all primary keys.
* Unique constraints on fields like AppointmentID in Treatment and TreatmentID in Bill act as indices to enforce 1:1 relationships and optimize joins.

# Views

The following views could be implemented:

A view for patient billing summaries.

A view combining appointments and treatments for doctor dashboards.

CREATE VIEW UpcomingAppointments AS

SELECT

a.AppointmentID,

p.FirstName AS PatientName,

d.FirstName AS DoctorName,

a.ApptDate,

a.ApptTime,

a.Status

FROM Appointment a

JOIN Patient p ON a.PatientID = p.PatientID

JOIN Doctor d ON a.DoctorID = d.DoctorID

WHERE a.ApptDate >= GETDATE();

Note: This view can be used to generate a real-time appointment dashboard for doctors and administrative staff.

Triggers:

The following Triggers could be implemented:

Auto-generating a bill when a treatment is added.

Preventing insertion of overlapping appointments for the same doctor or patient.

CREATE TRIGGER trg\_AutoGenerateBill

ON Treatment

AFTER INSERT

AS

BEGIN

INSERT INTO Bill (TreatmentID, Amount, DateIssued)

SELECT

TreatmentID,

Cost,

GETDATE()

FROM inserted;

END;

**Note: This automation enforces business logic and eliminates the need for manual bill entry.**

• By detailing each of these database structures with relevant SQL statements, this section provides a clear understanding of the architecture and functionality of the Healthcare Clinic Management System. Features such as automatic bill generation via triggers, and simplified data access through views like UpcomingAppointments can illustrate how the relational model supports healthcare operations. Additionally, the use of user interface tools such as **Windows Forms or WPF** will allow doctors and patients to interact with the system whether it’s booking appointments, reviewing prescriptions, or managing billing, which enhances functionality and usability.

## IV. UI Design

The User Interface (UI) design of our Healthcare Clinic Management System aims to ensure intuitive, efficient, and secure interaction between the user and the application. The design is structured around different user roles (patients, doctors, and administrators) and their needs. Based on the system’s architecture, the following UI components are implemented:

## 1. Login / Registration Page

This page allows users to log in or register to the system securely. It consists of two separate forms: one for login with email and password, and one for registration with name, email, and password. Input validation and error prompts are integrated to guide the user.

## 2. Start Page (Dashboard)

After login, the user is directed to a personalized dashboard. This page displays an overview including welcome messages, quick access to department filters, hospital location, and daily appointment statistics via charts. It serves as the central hub for navigation.

## 3. Admin Page

Accessible only by users with admin privileges. This page provides navigation to key administrative functionalities: managing users (patients/doctors), appointments, prescriptions, reports, and running SQL commands. Each section includes “Home” and “Logout” buttons for better navigation.

## 4. Manage Account

This page allows users to update their personal information (name, address, password). It includes form fields with “Confirm” and “Home” buttons. The layout ensures clarity and avoids confusion with consistent labeling.

## 5. Manage Users (Manage Patients / Doctors)

Admins can view and remove user accounts. Results of operations (e.g., search, removal) are displayed in a dynamically updated result section. Buttons like “View All Accounts” and “Remove Account” are present.

## 6. Manage Appointments

Users can view, add, or cancel appointments. Admins and doctors can oversee these appointments and take necessary actions. The results section updates based on interaction, and the design keeps appointment flow clear.

## 7. Manage Prescriptions

Doctors can view, add, remove, or edit prescriptions associated with patients. This page includes input validation and a result display area to confirm actions.

## 8. Manage Reports

Admins can view, add, and delete reports related to appointments or patient feedback. This ensures system oversight and auditing capabilities.

## 9. Run SQL Commands

An advanced interface for administrators to execute SQL commands directly. The results of the command are shown below the input area. This feature is intended for experienced users and includes safety validations.

**V. UI Design Flow**

