Newtown Surgery Relational Database

1. Introduction

Newtown doctor's surgery requires a relational database to manage their appointments, patient details, and relevant medical informations. The current system which uses a spreadsheet has its limitations in terms of data organization,searchability and scalability.

Business Rule: Each patient can have one or more doctors registered with and each doctor can be registered to one or more patients.

This business rule is important because it ensures that there are no duplicate appointments for the same patient with the same doctor at the same time. By enforcing this rule 'Newton Surgery' can maintain an organized schedule and avoid confusions which could lead to a poor patient experience and inefficient use of doctor's time.

Use Case: A patient needs to book an appointment with a doctor they're not registered with.

Actors:

- Patient
- Receptionist (or the designated staff member)

Description:

A patient contacts the receptionist to schedule an appointment with a doctor and the receptionist verifies the patient's information and checks the doctor's availability and then schedule's a date and time for the appointment.

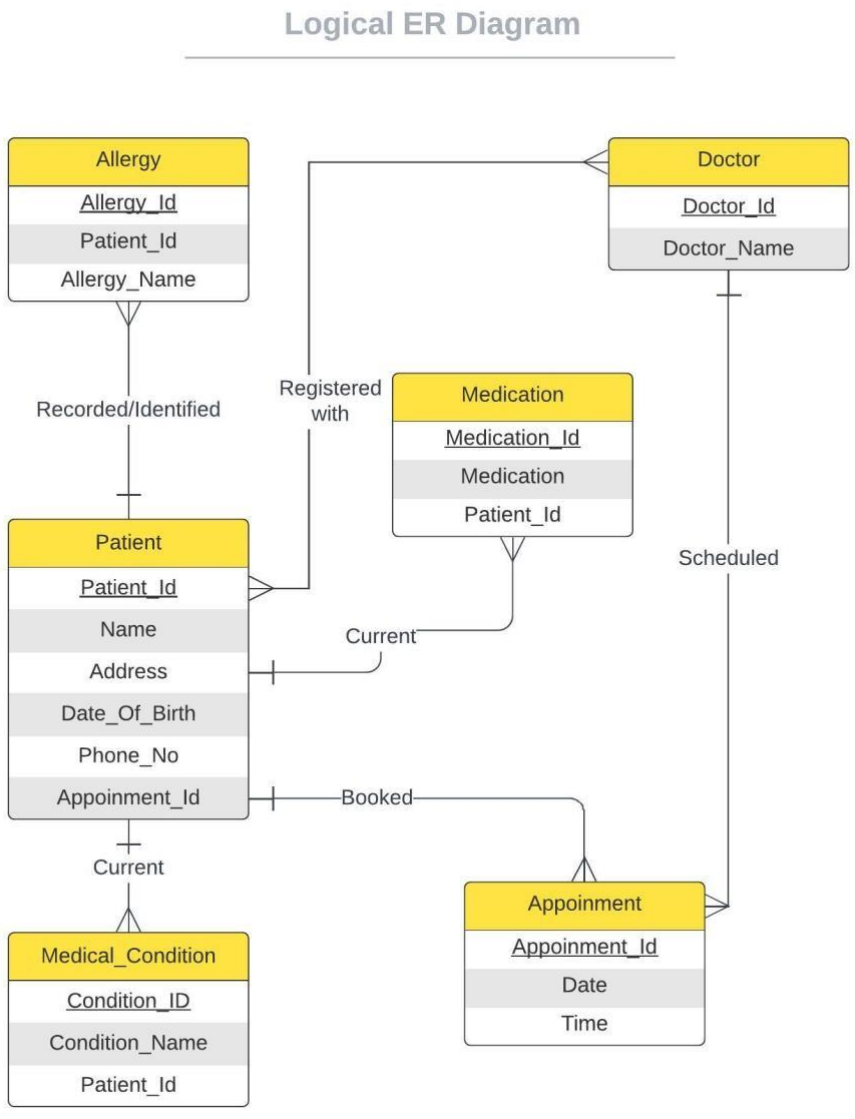
This use case is the most relevant use case because it is a common and essential process for any medical practice. It demonstrates how the database schema can be used to facilitate the scheduling of appointments ensuring that patients receive the necessary care from their doctors and additionally it highlights the need for proper management of patient, doctor, and appointment information which the database is designed to handle.

2. Logical ER Diagram

Patient (Patient_Id, Name, Address, Date_Of_Birth, Phone_No)
Doctor (Doctor_Id, Doctor_Name)
Appointment (Appointment_Id, Date_Time, Doctor_Id, Patient_Id)
Allergy (Allergy_Id, Allergy_Name, Patient_Id)
Medication (Medication_Id, Medication_Name, Patient_Id, Condition_Id)
Medical_Condition (Condition_Id, Condition_Name, Patient_Id)

Relationships:

Patient - Doctor (many-to-many)
Patient - Allergy (one-to-many)
Patient - Medication (one-to-many)
Patient – Medical_Condition (one-to-many)
Patient - Appointment (one-to-many)
Doctor - Appointment (one-to-many)



3. Database Schema

Database Schema diagram for the decomposed design. The design is in 3NF.

First Normal Form (1NF):

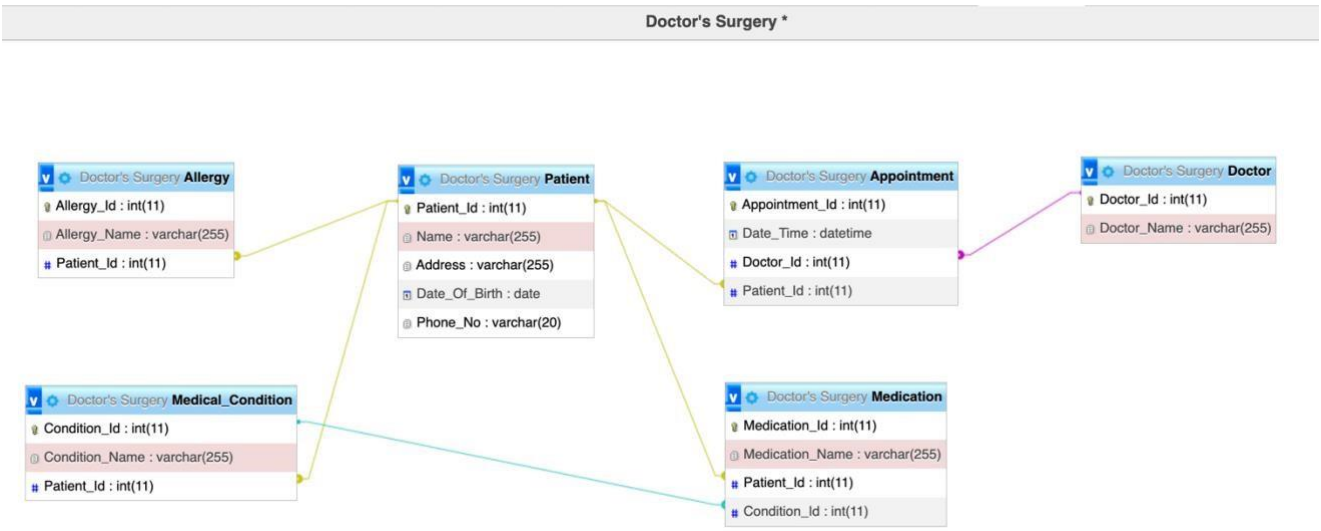
Each table has a primary key (Patient_Id, Doctor_Id, Allergy_Id, Condition_Id, Medication_Id, Appointment_Id), and all the attributes contain single values.

Second Normal Form (2NF):

The schema also satisfies 2NF as there are no partial dependencies on any primary key. All non-prime attributes in each table are fully functionally dependent on the primary key.

Third Normal Form (3NF):

The schema is in 3NF because there are no transitive dependencies.



4. Create Tables

```
CREATE TABLE Patient (
  Patient_Id INT AUTO_INCREMENT PRIMARY KEY,
  Name VARCHAR(255) NOT NULL,
  Address VARCHAR(255) NOT NULL,
  Date_Of_Birth DATE NOT NULL,
```

```
  Phone_No VARCHAR(20) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE Doctor (
  Doctor_Id INT AUTO_INCREMENT PRIMARY KEY,
  Doctor_Name VARCHAR(255) NOT NULL
) ENGINE=InnoDB;

CREATE TABLE Allergy (
  Allergy_Id INT AUTO_INCREMENT PRIMARY KEY,
  Allergy_Name VARCHAR(255) NOT NULL,
  Patient_Id INT,
  FOREIGN KEY (Patient_Id) REFERENCES Patient(Patient_Id) ON
DELETE CASCADE
) ENGINE=InnoDB;

CREATE TABLE Medical_Condition (
  Condition_Id INT AUTO_INCREMENT PRIMARY KEY,
  Condition_Name VARCHAR(255) NOT NULL,
  Patient_Id INT,
  FOREIGN KEY (Patient_Id) REFERENCES Patient(Patient_Id) ON
DELETE CASCADE
) ENGINE=InnoDB;

CREATE TABLE Medication (
  Medication_Id INT AUTO_INCREMENT PRIMARY KEY,
  Medication_Name VARCHAR(255) NOT NULL,
  Patient_Id INT,
  Condition_Id INT,
  FOREIGN KEY (Patient_Id) REFERENCES Patient(Patient_Id) ON
DELETE CASCADE,
  FOREIGN KEY (Condition_Id) REFERENCES
Medical_Condition(Condition_Id) ON DELETE CASCADE
) ENGINE=InnoDB;

CREATE TABLE Appointment (
  Appointment_Id INT AUTO_INCREMENT PRIMARY KEY,
  Date_Time DATETIME NOT NULL,
  Doctor_Id INT,
  Patient_Id INT,
  FOREIGN KEY (Doctor_Id) REFERENCES Doctor(Doctor_Id) ON
DELETE CASCADE,
  FOREIGN KEY (Patient_Id) REFERENCES Patient(Patient_Id) ON
DELETE CASCADE
) ENGINE=InnoDB;
```

5. Insert Data

```
INSERT INTO Patient (Patient_Id, name, Address, Date_Of_Birth, Phone_No)
VALUES (1, 'Jenna Smith', '23 Smart Lane, Newtown FK40 1DD', '1970-04-27',
'01234567890');
```

6. SQL Queries

i. Return a list of all patient names

```
SELECT Name FROM Patient;
```

ii. Return a list of all patient addresses , showing each address only once

```
SELECT DISTINCT Address FROM Patient;
```

iii. Write a query to count how many patients have Dr Jenkins as one of their registered doctors.

```
SELECT COUNT(*) FROM Appointment
JOIN Doctor ON Appointment.Doctor_Id = Doctor.Doctor_Id WHERE
Doctor.Doctor_Name = 'Dr. Jenkins';
```

iv. Calculate the average age of all patients

```
SELECT AVG(TIMESTAMPDIFF(YEAR, Date_Of_Birth, CURDATE()))
AS average_age FROM Patient;
```

v. Return all the patients whose last name is ‘Jones’

```
SELECT * FROM Patient
WHERE Name LIKE '%Jones';
```

vi. Find the names of the patients that were born before 1st January 1980

```
SELECT Name FROM Patient
WHERE Date_Of_Birth < '1980-01-01';
```

vii. List all the patients names along with their registered doctors names

```
SELECT Patient.Name as Patient_Name, Doctor.Doctor_Name as
Doctor_Name
```

```
FROM Patient
JOIN Appointment ON Patient.Patient_Id = Appointment.Patient_Id JOIN
Doctor ON Appointment.Doctor_Id = Doctor.Doctor_Id;
```

viii. List all the patients who are currently taking medication. Give the name of the patient, their current medication and the recent condition they are taking the medication for

```
SELECT Patient.Name as Patient_Name, Medication.Medication_Name,
Medical_Condition.Condition_Name
FROM Patient
JOIN Medication ON Patient.Patient_Id = Medication.Patient_Id
JOIN Medical_Condition ON Medication.Condition_Id =
Medical_Condition.Condition_Id;
```

ix. List all patients, giving their name and date of birth and, if the patient has had a recent condition, provide the medication they are taking. Otherwise, if the patient has had no recent condition, return null in the current medication field

```
SELECT Patient.Name as Patient_Name, Patient.Date_Of_Birth,
Medication.Medication_Name
FROM Patient
LEFT JOIN Medication ON Patient.Patient_Id = Medication.Patient_Id
LEFT JOIN Medical_Condition ON Medication.Condition_Id =
Medical_Condition.Condition_Id;
```