**Clinic Management System**

Project3

|  |  |
| --- | --- |
| Name | ID |
| Eman Atef | **231002233** |
| Shahenda Ali | **231002314** |
| Aya Esam | **231000779** |
| Tasabih Farid | **231001581** |

**1. Introduction**

The **Clinic Management System** project aims to create an integrated and well-structured database for managing the core operations within a medical clinic.  
This includes:

* Organizing patient appointments
* Storing and maintaining medical history
* Managing clinic data and resources
* Tracking doctor availability

The system addresses common issues such as **appointment conflicts**, **inaccurate records**, and **dependency on paper-based files**.  
By implementing a database, patient information can be accessed **faster and more accurately**, leading to improved quality and efficiency of healthcare services.

**Requirements – Extended Version**

The system must provide a set of essential features to efficiently serve the clinic’s needs. The key requirements are:

**Patient Data Management**

* Store personal details (name, gender, date of birth, address, occupation).
* Maintain medical history, previous diagnoses, and treatment records.

**Doctor Data Management**

* Store doctor details (name, specialty, phone number, address).
* Link each doctor to their respective department.
* Allow association of a doctor with multiple clinics if necessary.

**Department & Clinic Management**

* Each department consists of multiple clinics.
* Each clinic has a name and location within the hospital or medical center.

**Appointment Management**

* Record new appointments with assigned patient, doctor, and clinic.
* Edit or cancel appointments while keeping a modification log.
* Ensure no time conflicts in scheduling.

**Medical Record Management**

* Store diagnosis, treatment, and record date for each patient.
* Allow quick retrieval of past medical records.

**Reports & Queries**

* Generate lists of appointments by date, doctor, or patient.
* Create reports on the number of patients per department or total revenue for a given period.

**2.1 Entities**

* **Department**: Department ID, Department Name
* **Clinic**: Clinic ID, Clinic Name, Address, Department ID (FK)
* **Doctor**: Doctor ID, Doctor Name, Phone Number(s), Address, Department ID (FK)
* **Patient**: Patient ID, Patient Name, Contact Details, Date of Birth, Occupation
* **Appointment**: Appointment ID, Date, Start Time, End Time, Cost, Status, Diagnosis, Patient ID (FK), Doctor ID (FK)

**2.2 Key Features**

* Store complete patient medical history and personal information
* Organize and manage appointments
* Link doctors to their departments
* Useful queries such as:
  + Diagnoses by date
  + Appointments by doctor or department
  + Total payments for a specific patient

**3. Conceptual Design**

**3.1 Entities & Attributes**

**Patient**

* Patient\_ID (PK)
* Name (Composite)
* Phone\_Number (Multivalued)
* Birth\_Date
* Job
* Address (Composite)
* Age (Derived)

**Doctor**

* Doctor\_ID (PK)
* Doctor\_Name
* Phone\_Number (Multivalued)
* Address (Composite)
* Department\_ID (FK)

**Appointment**

* Appointment\_ID (PK)
* Appointment\_Date
* Start\_Time
* End\_Time
* Status
* Diagnosis
* Cost
* Patient\_ID (FK)
* Doctor\_ID (FK)

**Clinic**

* Clinic\_ID (PK)
* Clinic\_Name
* Address (Composite)
* Department\_ID (FK)

**Department**

* Department\_ID (PK)
* Department\_Name

**Entity Relationships – Extended Version**

**Patient**

* Can have multiple appointments with different doctors → **One-to-Many** relationship with Appointment.
* Can have multiple medical records over time → **One-to-Many** with Medical Record.

**Doctor**

* Works in one department only → **Many-to-One** relationship with Department.
* Can work in multiple clinics → **Many-to-Many** with Clinic via a linking table.
* Can have multiple appointments with different patients → **One-to-Many** with Appointment.
* Can create multiple medical records → **One-to-Many** with Medical Record.

**Department**

* Contains multiple doctors → **One-to-Many**.
* Contains multiple clinics → **One-to-Many**.

**Clinic**

* Belongs to one department only → **Many-to-One** with Department.
* Can have multiple appointments → **One-to-Many** with Appointment.
* Can have multiple doctors working in it → **Many-to-Many** with Doctor via a linking table.

**Appointment**

* Linked to one patient, one doctor, and one clinic.
* Contains diagnosis, cost, and status details.

**Medical Record**

* Linked to one patient and one doctor.
* Contains diagnosis, treatment, and date of record.

**Assumptions – Extended Version**

* Age is not stored directly since it changes over time; it is calculated from the date of birth when needed.
* An appointment cannot involve more than one patient or more than one doctor to ensure clarity and simplify management.
* Each clinic belongs to only one department and cannot be shared across departments.
* A doctor can work in multiple clinics, especially if they have multiple specialties or provide various services.
* Phone numbers are stored in a separate table to allow multiple numbers for the same person (doctor or patient).
* Appointment statuses are limited to: **Scheduled**, **In Progress**, **Postponed**, and **Completed**.
* Address is stored in structured form (City – District – Street – Building Number) to allow more accurate search queries.
* A patient may visit the clinic multiple times, and each visit is recorded as a separate appointment.

**Entity-Relationship (ER) Diagram:**

**A diagram of a company

AI-generated content may be incorrect.**

**ER-to-Relational schema:**

A diagram of a patient

AI-generated content may be incorrect.

1. **SQL code:**

These are the main SQL codes that we used, or in other words the most important codes in our project.

A screenshot of a computer program

AI-generated content may be incorrect.

A trigger that prevents double booking by blocking appointments if a doctor already has one at the same date and time.

A computer screen shot of a computer code

AI-generated content may be incorrect.This SQL query retrieves appointment details along with patient names, doctor names, and clinic information by joining the APPOINTMENT, PATIENT, DOCTOR, and CLINIC tables.

A computer screen shot of a data

AI-generated content may be incorrect.

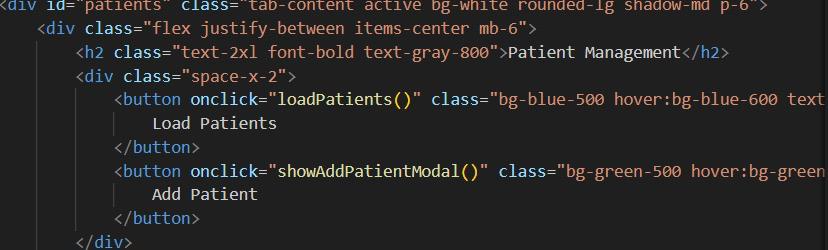
This SQL statement inserts 10 new patient records with their IDs, names, addresses, birth dates, and jobs into the PATIENT table.

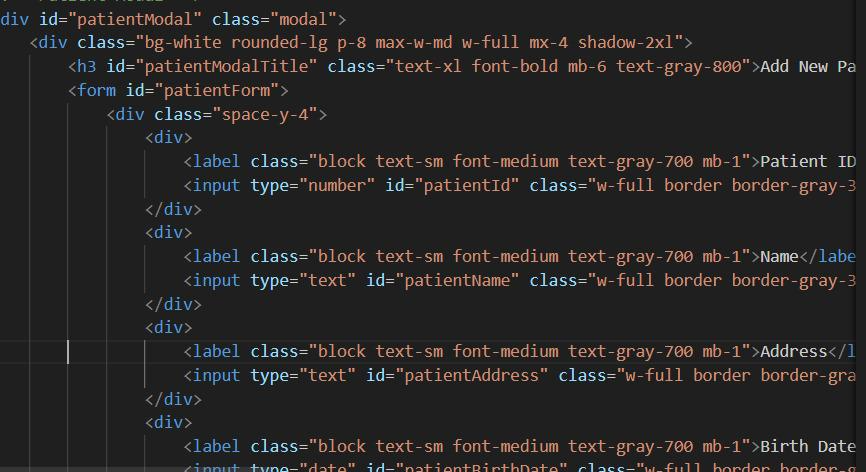
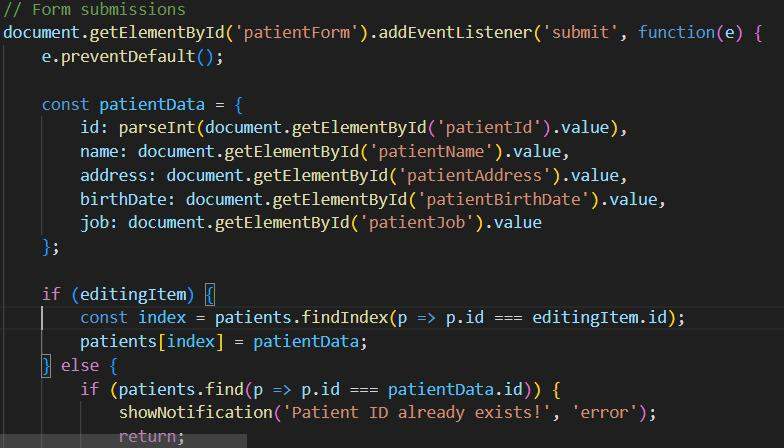
A computer code with blue text

AI-generated content may be incorrect.

Creates DEPARTMENT and CLINIC tables, linking each clinic to its department via a foreign key.

**Bonus:**

  
This part of the code creates a “Patient Management” section with two buttons.

The first button runs loadPatients() when clicked to show patient data, and the second button runs showAddPatientModal() to open a form for adding a new patient.  
  
  
  
  
  
This code creates a pop-up modal form for adding a new patient, with fields for Patient ID, Name, Address, and Birth Date. It uses HTML for structure and Tailwind CSS classes for styling (white background, rounded corners, shadows). The form is designed to be clean and user-friendly.  
  
  
  
  
  
  
  
This code handles the patient form submission.

It collects the input values (ID, name, address, birth date, job) into patientData. If we are editing a patient, it updates their data; otherwise, it adds a new patient, but first checks if the ID already exists to avoid duplicates.  
  
  
This function displays all patients in a table.

It clears the table body, then loops through each patient and creates a row with their details (ID, name, address, birth date, job) plus buttons to edit or delete that patient.

**The output :**

**A screenshot of a medical form

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.