# Hospital Management System - Complete Reference

## 1. Final Database Mapping (Entities & Relationships)

### Entities

**Patient** - Id (PK) - Name - Email - Phone - DateOfBirth - Address

**Doctor** - Id (PK) - Name - Email - Phone - Specialty

**Department** - Id (PK) - Name - Location

**Appointment** - Id (PK) - PatientId (FK → Patient.Id) - DoctorId (FK → Doctor.Id) - AppointmentDate - Status (Scheduled, Completed, Cancelled)

**MedicalRecord** - Id (PK) - PatientId (FK → Patient.Id) - DoctorId (FK → Doctor.Id) - Diagnosis - Prescription - RecordDate

**Medication** - Id (PK) - Name - Description - Quantity

**PrescriptionDetail** - Id (PK) - MedicalRecordId (FK → MedicalRecord.Id) - MedicationId (FK → Medication.Id) - Dosage - Frequency

### Relationships

| Relationship | Type | Notes |
| --- | --- | --- |
| Patient ↔ Appointment | 1-to-Many | Patient can have many appointments |
| Doctor ↔ Appointment | 1-to-Many | Doctor can have many appointments |
| Department ↔ Doctor | 1-to-Many | Each doctor belongs to one department |
| Patient ↔ MedicalRecord | 1-to-Many | Patient can have many medical records |
| Doctor ↔ MedicalRecord | 1-to-Many | Doctor can write many medical records |
| MedicalRecord ↔ Medication | Many-to-Many | Via PrescriptionDetail |

### ER Diagram (Text Version)

Patient  
 └─< Appointment >─┐  
 Doctor  
Department ─< Doctor  
  
Patient ─< MedicalRecord >─ Doctor  
MedicalRecord ─< PrescriptionDetail >─ Medication

## 2. Project Idea & Scenario

### Core Idea

Build a **Hospital Management System API** to manage patients, doctors, departments, appointments, and medical records efficiently. The API will be used by front-end clients like web or mobile applications.

### Scenario & Functionalities

1. **Patient Management**
   * Add, update, delete patients
   * View patient details and medical history
2. **Doctor Management**
   * Add, update, delete doctors
   * Assign doctors to departments
   * View doctor appointments and medical records
3. **Department Management**
   * Add, update, delete departments
   * View all doctors in each department
4. **Appointment Management**
   * Schedule appointments for patients with doctors
   * Update appointment status (Scheduled, Completed, Cancelled)
   * Prevent overlapping appointments for same doctor/patient
5. **Medical Records Management**
   * Doctors can write medical reports for patients
   * Link prescriptions to reports
   * View complete patient medical history
6. **Medication Management**
   * CRUD operations for medications
   * Track which medication was prescribed to which patient

### Technical Features

* **RESTful API** using .NET 7/8
* **Authentication & Authorization**: JWT Tokens (Roles: Admin, Doctor, Receptionist)
* **Business Rules**: Appointment conflicts, report validation
* **Advanced Features**: Search & filter, pagination, caching for common data

### API Endpoints Example (Resource-based)

GET /patients  
POST /patients  
PUT /patients/{id}  
DELETE /patients/{id}  
  
GET /doctors  
POST /doctors  
PUT /doctors/{id}  
DELETE /doctors/{id}  
  
GET /departments  
POST /departments  
PUT /departments/{id}  
DELETE /departments/{id}  
  
GET /appointments  
POST /appointments  
PUT /appointments/{id}  
DELETE /appointments/{id}  
  
GET /medicalrecords  
POST /medicalrecords  
PUT /medicalrecords/{id}  
DELETE /medicalrecords/{id}  
  
GET /medications  
POST /medications  
PUT /medications/{id}  
DELETE /medications/{id}

## 3. Project Mapping Overview (Summary)

* **Entities**: Patient, Doctor, Department, Appointment, MedicalRecord, Medication, PrescriptionDetail
* **Relationships**: 1-to-Many and Many-to-Many as defined above
* **Scenario**: Hospital workflow from patient registration → doctor assignment → appointment → medical record → prescription
* **Tech Stack**: .NET 7/8 API, EF Core, JWT Authentication, SQL Server (or any RDBMS)
* **Advanced Concepts**: DTOs, Services, Dependency Injection, Business Rules Enforcement, Pagination & Caching

*End of Document*