Project Report
Hospital Management System

Authors:

Krishna Raj Bhandari
Suraj Bhatta
Vinayak Talawar
Vojislav Andelic
Anirudh Aggarwal

Contents:

Topic

The Hospital Management System is a robust and integrated database solution aimed at transforming and streamlining the operations of healthcare facilities. This system is designed to provide efficient management of essential hospital entities, including doctors, patients, appointments, diagnoses, prescriptions, medicines, and billing.

By utilizing a normalized relational database schema, the Hospital Management System ensures high data integrity, minimizes redundancy, and supports complex data retrieval to accommodate the dynamic needs of a modern hospital environment.

Key objectives of the Hospital Management System include:

- **Centralized Data Management**: All hospital-related data is consolidated within a secure, unified database, ensuring consistency and facilitating easy access for authorized users.
- **Efficient Scheduling and Appointments**: Patients can seamlessly book, modify, or cancel appointments, while doctors are empowered to manage their schedules and view upcoming consultations efficiently.
- **Comprehensive Patient Records**: The system maintains up-to-date patient profiles, including medical histories, diagnoses, prescribed medications, and complete billing information, supporting continuity of care.
- **Automated Billing and Inventory Control**: Billing processes are automated to reduce manual errors, and medicine inventories are systematically managed with detailed tracking of batches, stock levels, expiry dates, and ingredients.
- **Enhanced Decision-Making for Administration:** Hospital administrators gain access to well-organized, actionable data, enabling improved resource allocation, compliance, and overall patient care.

System Design Principles
The Hospital Management System is built with scalability, security, and user-friendliness in mind. It adheres to healthcare data standards and best practices, making it a foundational tool for hospitals seeking to elevate patient services and streamline administrative workflows.

User Stories

Scenario Overview
The Hospital Management System is designed to meet the needs of all key stakeholders: doctors, patients, pharmacists, nurses, and system administrators. Each user story below outlines a specific requirement from the perspective of a particular user, providing a foundation for system design and implementation.

User Stories by Role:

Krishna Raj Bhandari: **Doctor**

1) As a doctor I want to see all my upcoming appointments with patient details So that I can prepare for each visit.
   Acceptance Criteria:
   - List shows all future appointments assigned to the doctor
   - Each appointment includes patient name, contact, and reason for visit
   - Sorted by appointment date and time

2) As a doctor, I want to review a patient's diagnosis history before their next appointment so that I understand their medical background.
   Acceptance Criteria:
   - Ability to access diagnosis records linked to the patient
   - Records show diagnosis date, details, and status
   - Available before or during appointment
3) As a doctor, I want to see all missed or cancelled appointments so that I can follow up with patients.
   Acceptance Criteria:
   - List includes all appointments marked missed or cancelled
   - Displays patient info and appointment date
   - Option to filter by date range

4) As a doctor, I want to see a list of all patients with a specific blood type (e.g., A+) So that I can arrange blood donation or special care.
   Acceptance Criteria:
   - Search or filter patients by blood type
   - List shows patient names and contact info
   - Blood type data is accurate and up to date

5) As a doctor, I want to see a monthly count of my appointments over the last year So that I can analyse workload trends.
   Acceptance Criteria:
   - Display monthly appointment counts for the past 12 months
   - Data shown in table or chart form
   - Includes total appointments per month

Suraj Bhatta: **Patient**

6) As a patient, I want to see all my appointments including the doctor and the status so that I stay informed.
   Acceptance Criteria:
   - List of patient's past and upcoming appointments
   - Each entry shows doctor name and appointment status (scheduled, completed, cancelled)

7) As a patient, I want to view my diagnoses and prescribed medicines for each appointment so that I understand my treatment.
   Acceptance Criteria:
   - For each appointment, display diagnosis details and prescribed medicines
   - Medicines include name, dosage, and instructions

8) As a patient, I want to see my bill details including their prices so that I know the charges.
   Acceptance Criteria:
   - Bills show itemized list with service/medicine and price
   - Total amount due and payment status included

9) As a patient, I want to know which doctors I see most often so that I understand my ongoing care pattern.
   Acceptance Criteria:
   - Display list of doctors ranked by number of visits
   - Show total visits per doctor

10) As a patient, I want to review all my cancelled or rescheduled appointments so that I can keep track.
    Acceptance Criteria:
    - List all appointments with status cancelled or rescheduled
    - Include original and updated appointment dates (if applicable)

Vinayak Talawar: **Pharmacist**

11) As pharmacy staff, I want to view all prescriptions issued by doctors so that I can prepare medicines for dispensing.
    Acceptance Criteria:
    - List of all active prescriptions with patient and doctor info
    - Shows medicine names, dosage, and quantity

12) As pharmacy staff, I want to see the current inventory of all medicines so that I can manage stock.
    Acceptance Criteria:
    - List of medicines with current stock levels
    - Stock quantities updated in real-time or daily

13) As pharmacy staff, I want to see all medicines with their ingredients so that I know composition for safety and info.
    Acceptance Criteria:
    - Each medicine entry includes a list of ingredients

- Ingredient info accessible via search or browse

14) As pharmacy staff, I want to check the expiry dates of all medicines in stock so that I avoid dispensing expired products.
    Acceptance Criteria:
    - Display expiry date for each medicine batch
    - Highlight medicines nearing or past expiry

15) As pharmacy staff, I want to see medicines not prescribed to any patient in last 6 months so that I can manage stock or promotions.
    Acceptance Criteria:
    - List medicines with zero prescriptions in past 6 months
    - Include last prescribed date (if any)

Anirudh Aggarwal: **Nurse**

16) As a nurse, I want to see the list of today's appointments and their status so that I can manage patient flow efficiently.
    Acceptance Criteria:
    - List of all today's appointments with patient name and status (arrived, waiting, completed)

17) As a nurse, I want to see which patients have arrived for their appointments today so that I can prioritize care.
    Acceptance Criteria:
    - Filter or list patients marked as arrived today
    - Includes Appointment time and doctor

18) As a nurse, I want to see all medicine prescribed to a patient so that I administer correctly.
    Acceptance Criteria:
    - Display patient's current prescriptions with medicine names, dosage, and instructions

19) As a nurse, I want to view and print a summary of a patient's prescribed medicines and instructions so that I support in-patient care.
    Acceptance Criteria:
    - Generate printable summary report of medicines and administration instructions
    - Summary includes patient info and prescribing doctor

20) As a nurse, I want to list all appointments today for patients under 18 years old so that I can prepare special care.
    Acceptance Criteria:
    - Filter today's appointments to show only patients aged under 18
    - Includes appointment time and patient details

Vojislav Andelic: **Admin**

21) As an admin, I want to see all users (doctors and patients) so that I can manage the system.
    Acceptance Criteria:
    - List all users with roles, names, and contact info
    - Ability to filter by role

22) As an admin, I want to see all medical specializations available so that I can manage specialties.
Acceptance Criteria:
   - List all medical specializations used in hospital
   - Include description or code for each specialization

23) As an admin, I want a summary of appointment counts per doctor by day so that I monitor workload distribution.
Acceptance Criteria:
   - Table or chart showing daily appointment counts per doctor
   - Data filterable by date range

24) As an admin, I want to see total revenue generated by the hospital each month so that I can analyse financial performance.
Acceptance Criteria:
   - Display monthly revenue totals for the selected year
   - Revenue sums include all bill payments recorded

25) As an admin, I want to list top 5 patients with most appointments this year so that I can identify engaged patients.
Acceptance Criteria:
   - List patients ranked by number of appointments in the current year
   - Show total visits for each patient

Entity Relationship Model

Design choices

Based on the user stories and system requirements, several candidates for entities and relationships were identified: Doctor, Patient, Appointment, Diagnosis, Medicine, Prescription, Bill, and administrative/staff roles.

- **Doctor** is implemented as an entity to store details such as name, contact information, specialization, and to manage appointments and prescriptions (see user stories 1, 2, 3, 4, and 5).
- **Patient** is a key entity, holding information such as name, gender, blood type, contact details, insurance number, and date of birth. This supports comprehensive patient records and appointment management (see user stories 6, 7, 8, 9, and 10).
- **Appointment** is introduced as an entity rather than an attribute or a simple relationship, as it carries specific properties like date, status, purpose, and is central to linking doctors and patients (see user stories across multiple roles).
- **Diagnosis** is modelled as a distinct entity since it contains structured data (symptoms, description, date/time) and is uniquely assigned to an appointment (see user story 2 and 4).
- **Medicine** is an entity to manage medicine inventory, including name, ingredients, price, stock, and expiry date, as required by pharmacy and billing functions (see user stories 11, 12, 13, 14, and 15).
- **Prescription** is realized as a relationship between Appointment and Medicine, carrying attributes such as prescription ID, dosage, duration, and instructions, reflecting real-world prescription scenarios (see user story 3).
- **Bill** is implemented as a relationship with attributes between Patient and Medicine, capturing each transaction for transparency and reporting (see user stories 8 and 13).
- **#Administrative** roles (Pharmacist/Staff, Nurse, Admin/System) are not modelled as entities for data storage, but their actions are reflected in the system through user rights and activity logs. User rights are managed outside the core ER model but are essential for access control (see user stories 21–25).
- **Inventory** is not realized as a separate entity since inventory status can be derived from the Medicine entity's stock attribute.
- **Insurance and Specializations** are modelled as attributes within Patient and Doctor entities, respectively, to support relevant user stories (see user stories 4, 9, and 23).

Relationships such as **Offers** (Doctor-Appointment), **Books** (Patient-Appointment), **Assign** (Appointment-Diagnosis), **Prescribe** (Appointment-Medicine), and **Bill** (Patient-Medicine) are explicitly modelled to capture the multi-faceted interactions in hospital operations. Relationships are given attributes where necessary (e.g., booking date/method, dosage, billing date/price).

The ER diagram aims to capture the scenario defined by the user stories without over-specification. Only those entities and attributes necessary for the system's main functionalities are included, while leaving room for future extensions such as detailed staff management or insurance claim processing.

ERM Diagram

The following diagram visually represents the core entities and relationships of the Hospital Management System, as derived from the user stories and design choices discussed above.

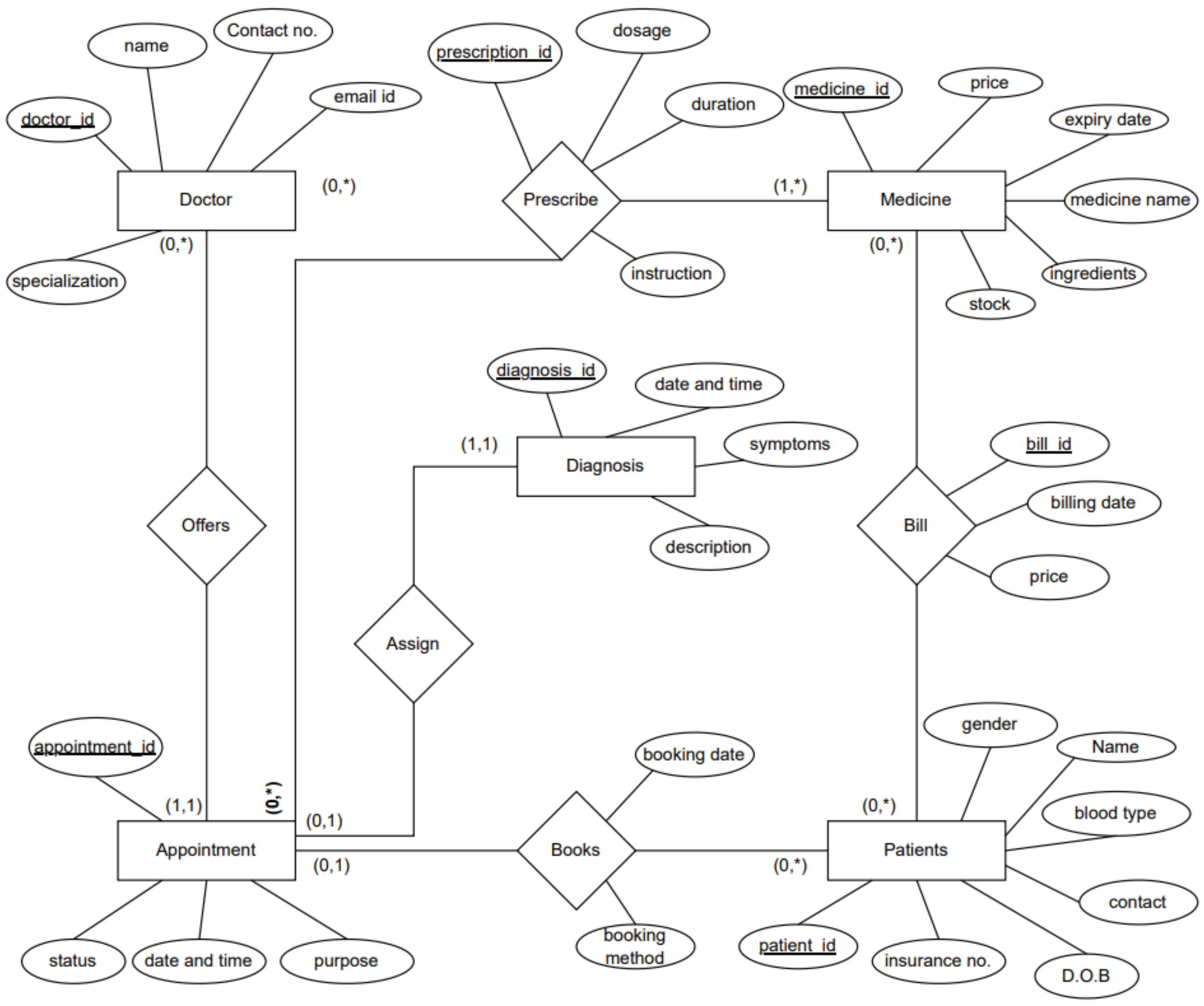

Figure: Entity-Relationship Model for the Hospital Management System

ERM Diagram Explanation

- **Doctor**: Holds details of healthcare professionals and is linked to appointments and prescriptions.
- **Patient**: Central to the system, stores all relevant patient data and is linked to appointments, billing, and prescriptions.
- **Appointment**: Connects doctors and patients, including date, status, and purpose.
- **Diagnosis**: Linked one-to-one with an appointment; records symptoms and medical findings.
- **Medicine**: Maintains the medicine inventory and is connected to prescriptions and billing.
- **Prescription**: A relationship between Appointment and Medicine, detailing dosage, duration, and instructions.
- **Bill**: Captures billing transactions between patients and medicines, tracking prices and billing dates.

Each relationship is annotated with cardinality and key attributes, ensuring that the structure supports all required queries and operations.

Relational Model and Normalization

Design Choices

To translate the ER model into a relational schema, each entity and significant relationship is mapped to a table (relation), with careful selection of primary keys (PKs) and foreign keys (FKs) to enforce integrity and support efficient queries.

- **Primary Keys:** Every table uses a unique, single-attribute or composite primary key (e.g., doctor_id, appointment_id, (diagnosis_id, symptom)).
- **Foreign Keys**: Relationships between entities are captured using foreign keys (e.g., appointment.patient_id → patient.patient_id).
- **Composite and Multi-Value Attributes:** Composite attributes such as names are split into first name, last name, etc., while multi-value attributes (e.g., specializations, symptoms, ingredients) are modelled with separate linking tables (e.g., Doctor_Specialization, Diagnosis_Symptom, Medicine_Ingredient).
- **Relationship Tables:** N:M relationships (e.g., between bills and medicines, appointments and medicines) are realized as associative entities (e.g., Bill_Medicine, Prescribe) with composite primary keys.
- **Attribute Placement:** Attributes are placed in the most appropriate relation to ensure atomicity and eliminate redundancy (e.g., batch info is separate from medicine, symptoms are separated from diagnosis).
- **Normalization Considerations:** Each relation is checked for 1NF, 2NF, and 3NF/BCNF compliance, ensuring no repeating groups, full functional dependency on the PK, and no transitive dependencies among non-key attributes.

Initial (Unnormalized) Relational Schema

The initial schema directly mirrored the ER diagram, where some attributes (such as symptoms, specializations, ingredients) could have appeared as multi-valued attributes within their parent entities (e.g., Doctor, Diagnosis, Medicine).

1. **Doctor:**(doctor_id, name, contact_no, email_id, specialization)

2. **Patient:**(patient_id, name, gender, dob, contact, insurance_no, blood_type, address)

3. **Appointment:**(appointment_id, patient_id, doctor_id, date_time, purpose, status, booking_date, booking_method)

PK (appointment_id), FK (patient_id → Patient), FK (doctor_id → Doctor)

4. **Diagnosis:**(diagnosis_id, appointment_id, date_time, description, symptom)

PK (diagnosis_id), FK (appointment_id → Appointment)

5. **Bill:**(bill_id, patient_id, billing_date, price, medicine)

PK (bill_id), FK (patient_id → Patient)

6. **Medicine:**(medicine_id, name, price, stock, ingredient)

7. **Prescribe:**(appointment_id, medicine_id, dosage, duration, instruction)

PK (appointment_id, medicine_id), FK (appointment_id → Appointment), FK (medicine_id → Medicine)

Note:

- Multi-valued attributes (e.g., specialization, symptom etc.) are present.

- Medicines and symptoms are not in separate tables, leading to redundancy and anomalies.

- The Bill table includes medicine columns directly, which is non-normalized.

Normalization Process

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. The following steps outline how the Hospital Management System schema was normalized from its initial design to the final structure, ensuring compliance with the First (1NF), Second (2NF), and Third Normal Forms (3NF), and ultimately Boyce-Codd Normal Form (BCNF).

Step 1: First Normal Form (1NF)

Definition: A table is in 1NF if all its attributes have only atomic (indivisible) values, and each record is unique.

Application in Our Schema:

- All tables were reviewed to ensure that no attributes contained repeating groups or arrays.

- Multi-valued attributes (e.g., doctor specializations, diagnosis symptoms, medicine ingredients) were placed in separate linking tables:
    - Doctor_Specialization (doctor_id, specialization)
    - Diagnosis_Symptom (diagnosis_id, symptom)
    - Medicine_Ingredient (medicine_id, ingredient)
    - Bill_Medicine(bill_id, medicine_id)
- Each table row is uniquely identified by its primary key.

Result:

All tables in our schema are in 1NF.

Step 2: Second Normal Form (2NF)

Definition: A table is in 2NF if:

- It is in 1NF.
- Every non-key attribute is fully dependent on the entire primary key (not just part of it). This is relevant for tables with composite primary keys.

Key Issue Identified:

The initial Prescription table was defined as:

- Prescription (doctor_id, medicine_id, dosage, duration, instruction)
- PK (doctor_id,medicine_id)

Problem:

- Attributes like dosage, duration, and instruction depend only on medicine_id (i.e., which medicine was prescribed), not on the combination of doctor_id and medicine_id.
- This partial dependency violates 2NF.

How We Fixed it:

- We realized that a prescription is always given in the context of an appointment (which already links doctor and patient).
- We redesigned the Prescription table to link appointments and medicines, not doctors and medicines:
    - Prescription (appointment_id, medicine_id, dosage, duration, instruction)
    - PK: (appointment_id, medicine_id)
- Now, all non-key attributes depend on the full composite key, as:
    - appointment_id provides context (who, when, by which doctor)
    - medicine_id specifies which medicine
    - dosage, duration, and instruction are specific to each (appointment, medicine) pair

Other Tables:

- Tables with single-column primary keys (Doctor, Patient, Medicine, etc.) already satisfy 2NF.
- Junction tables like Bill_Medicine (bill_id, medicine_id) and Diagnosis_Symptom (diagnosis_id, symptom) have non-key attributes dependent on both parts of their composite keys.

Summary For 2NF:

- All partial dependencies were removed.
- All non-key attributes in composite-key tables depend on the entire key.

Step 3: Third Normal Form (3NF)

Definition: A table is in 3NF if:

- It is in 2NF.
- There are no transitive dependencies (non-key attributes depending on other non-key attributes).

Application in Our Schema:

- Each non-key attribute is directly related to the primary key only.
- For instance, in Patient, attributes like contact, insurance_no, blood_type, and address each describe the patient and depend only on patient_id.
- In Medicine, price and stock depend directly on medicine_id.
- In Appointment, all details (date, status, purpose, etc.) depend on appointment_id.

Result:

No transitive dependencies remain. All tables are in 3NF.

Step 4: Boyce-Codd Normal Form (BCNF)

Definition: A table is in BCNF if, for every non-trivial functional dependency X → Y, X should be a super key.

Application:

- All relations/tables were reviewed to ensure that every determinant is a candidate key.
- Composite key tables (e.g., Prescription, Diagnosis_Symptom) only have non-key attributes dependent on the full composite key.
- No violations of BCNF found.

Summary:

- **1NF:** All tables have atomic values (no repeating groups).
- **2NF:** Partial dependencies were removed especially by redesigning Prescription from (doctor_id, medicine_id) to (appointment_id, medicine_id).
- **3NF/BCNF:** No transitive dependencies; all determinants are candidate keys.

Key Insight:

The main normalization challenge was correctly modelling prescriptions. Initially, prescriptions were linked to doctors and medicines, which led to partial dependency problems. By linking prescriptions to appointments and medicines, we achieved proper normalization, making the schema robust and flexible for complex queries and future extensions.

Final Normalized Relational Schema

The result of this normalization process is a set of relations that are:

- Free from update, insertion, and deletion anomalies
- Efficient for querying and maintenance
- Directly aligned with the requirements and business logic of the Hospital Management System

Final Schema (with PK/FK notations):

1. **Doctor** (doctor_id, name, contact_no, email_id)

2. **Doctor_Specialization** (doctor_id, specialization)

PK (doctor_id, specialization), FK (doctor_id → Doctor)

3. **Patient** (patient_id, name, gender, dob, contact, insurance_no, blood_type, address)

4. **Appointment** (appointment_id, patient_id, doctor_id, date_time, purpose, status, booking_date, booking_method)

PK (appointment_id), FK (patient_id → Patient), FK (doctor_id → Doctor)

5. **Diagnosis** (diagnosis_id, appointment_id, date_time, description)

PK (diagnosis_id), FK (appointment_id → Appointment)

6. **Diagnosis_Symptom** (diagnosis_id, symptom)

PK (diagnosis_id, symptom), FK (diagnosis_id → Diagnosis)

7. **Bill** (bill_id, patient_id, billing_date, price)

PK (bill_id), FK (patient_id → Patient)

8. **Medicine** (medicine_id, name, price, stock)

9. **Medicine_Ingredient** (medicine_id, ingredient)

PK (medicine_id, ingredient), FK (medicine_id → Medicine)

10. **Medicine_Batch** (medicine_id, batch_no, expiry_date, quantity)

PK (medicine_id, batch_no), FK (medicine_id → Medicine)

11. **Prescribe** (appointment_id, medicine_id, dosage, duration, instruction)

PK (appointment_id, medicine_id), FK (appointment_id → Appointment), FK (medicine_id → Medicine)

12. **Bill_Medicine** (bill_id, medicine_id)

PK (bill_id, medicine_id), FK (bill_id → Bill), FK (medicine_id → Medicine)

Summary:

The final schema is fully normalized (3NF/BCNF), eliminates redundancy and update anomalies, and supports all user stories and system operations efficiently.

Implementation using SQL

Design Choices on SQL Data Types

The selection of SQL data types is critical for ensuring data integrity, efficient storage, and performance. Below are the major design decisions for each table:

Doctor & Doctor_Specialization

- doctor_id INT AUTO_INCREMENT PRIMARY KEY: Numeric surrogate key, auto incremented for uniqueness.
- name VARCHAR (100): Allows for full names up to 100 characters
- contact_no VARCHAR (15): Supports international phone numbers.
- email_id VARCHAR (100) UNIQUE: Ensures unique email addresses; length accommodates most emails.
- specialization VARCHAR (50): Most medical specializations fit within 50 characters.

Patient

- patient_id INT AUTO_INCREMENT PRIMARY KEY: Numeric surrogate key.
- name VARCHAR (100): Full name.
- gender CHAR (1): 'M', 'F', or other
- dob DATE: Date of birth, standard SQL date.
- contact VARCHAR (15): Patient phone number.
- insurance_no VARCHAR (30): Typical insurance numbers fit within 30 characters.
- blood_type VARCHAR (3): E.g., 'A+', 'O-'.
- address VARCHAR (200): Accommodates detailed address

Appointment

- appointment_id INT AUTO_INCREMENT PRIMARY KEY
- patient_id INT / doctor_id INT: Foreign keys to Patient/Doctor.
- date_time DATETIME: Appointment timestamp.
- purpose VARCHAR (100): Brief purpose/description.
- status VARCHAR (30): E.g., 'Completed', 'Pending'.
- booking_date DATE: When appointment was booked.
- booking_method VARCHAR (20): E.g., 'Online', 'Phone'.

Diagnosis & Diagnosis_Symptom

- diagnosis_id INT AUTO_INCREMENT PRIMARY KEY
- appointment_id INT: FK to Appointment.
- date_time DATETIME: Timestamp of diagnosis.
- description VARCHAR (200): Text description.
- symptom VARCHAR (50): Each symptom as a separate row (junction table).

Medicine, Medicine_Ingredient, Medicine_Batch

- medicine_id INT AUTO_INCREMENT PRIMARY KEY
- name VARCHAR (100): Medicine name.
- price DECIMAL (7,2): Up to 99,999.99.
- stock INT: Current stock.
- ingredient VARCHAR (50): Active ingredient (junction table).
- batch_no VARCHAR (20): Batch identifier.
- expiry_date DATE: Standard date.
- quantity INT: Batch size.

Prescribe

- Composite primary key: (appointment_id, medicine_id)
- dosage VARCHAR (20), duration VARCHAR (20), instruction VARCHAR (100): Text fields for prescription details.

Bill & Bill_Medicine

- bill_id INT AUTO_INCREMENT PRIMARY KEY
- patient_id INT: FK to Patient.
- billing_date DATE: When billed.
- price DECIMAL (7,2): Total bill amount.
- medicine_id INT: FK to Medicine (junction table for N:M relationship).

Table Definitions in SQL

```
CREATE TABLE Doctor (

    doctor_id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100),

    contact_no VARCHAR(15),

    email_id VARCHAR(100) UNIQUE

);


CREATE TABLE Doctor_Specialization (

    doctor_id INT,

    specialization VARCHAR(50),

    PRIMARY KEY (doctor_id, specialization),

    FOREIGN KEY (doctor_id) REFERENCES Doctor(doctor_id)

);


CREATE TABLE Patient (

    patient_id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100),

    gender CHAR(1),

    dob DATE,

    contact VARCHAR(15),

    insurance_no VARCHAR(30),

    blood_type VARCHAR(3),

    address VARCHAR(200)

);


CREATE TABLE Appointment (

    appointment_id INT AUTO_INCREMENT PRIMARY KEY,

    patient_id INT,

    doctor_id INT,

    date_time DATETIME,

    purpose VARCHAR(100),
    status VARCHAR(30),
    booking_date DATE,
    booking_method VARCHAR(20),
    FOREIGN KEY (patient_id) REFERENCES Patient(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctor(doctor_id)
);

CREATE TABLE Diagnosis (
    diagnosis_id INT AUTO_INCREMENT PRIMARY KEY,
    appointment_id INT UNIQUE,
    date_time DATETIME,
    description VARCHAR(200),
    FOREIGN KEY (appointment_id) REFERENCES
Appointment(appointment_id)
);

CREATE TABLE Diagnosis_Symptom (
    diagnosis_id INT,
    symptom VARCHAR(50),
    PRIMARY KEY (diagnosis_id, symptom),
    FOREIGN KEY (diagnosis_id) REFERENCES Diagnosis(diagnosis_id)
);

CREATE TABLE Medicine (
    medicine_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100),
    price DECIMAL(7,2),
    stock INT
);

CREATE TABLE Medicine_Ingredient (
    medicine_id INT,
    ingredient VARCHAR(50),

    PRIMARY KEY (medicine_id, ingredient),

    FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id)

);


CREATE TABLE Medicine_Batch (

    medicine_id INT,

    batch_no VARCHAR(20),

    expiry_date DATE,

    quantity INT,

    PRIMARY KEY (medicine_id, batch_no),

    FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id)

);


CREATE TABLE Prescribe (

    appointment_id INT,

    medicine_id INT,

    dosage VARCHAR(20),

    duration VARCHAR(20),

    instruction VARCHAR(100),

    PRIMARY KEY (appointment_id, medicine_id),

    FOREIGN KEY (appointment_id) REFERENCES
Appointment(appointment_id),

    FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id)

);


CREATE TABLE Bill (

    bill_id INT AUTO_INCREMENT PRIMARY KEY,

    patient_id INT,

    billing_date DATE,

    price DECIMAL(7,2),

    FOREIGN KEY (patient_id) REFERENCES Patient(patient_id)
```

```
);

CREATE TABLE Bill_Medicine (
    bill_id INT,
    medicine_id INT,
    PRIMARY KEY (bill_id, medicine_id),
    FOREIGN KEY (bill_id) REFERENCES Bill(bill_id),
    FOREIGN KEY (medicine_id) REFERENCES Medicine(medicine_id)
);
```

Data Definitions in SQL

Below is a sample of the data inserted into the system, demonstrating coverage of all major user stories and relationships:

Doctors

```
INSERT INTO Doctor (name, contact_no, email_id) VALUES
('Dr. John Smith', '1234567890', 'john.smith@hospital.com'),
('Dr. Alice Johnson', '1234567891', 'alice.johnson@hospital.com'),
('Dr. Emily Williams', '1234567892', 'emily.williams@hospital.com'),
('Dr. Michael Brown', '1234567893', 'michael.brown@hospital.com'),
('Dr. Linda Davis', '1234567894', 'linda.davis@hospital.com');
```

Doctor Specializations

```
INSERT INTO Doctor_Specialization (doctor_id, specialization) VALUES
(1, 'Cardiologist'), (1, 'General Physician'),
(2, 'Dermatologist'),
(3, 'Pediatrician'),
(4, 'Orthopedic'),
(5, 'Gynecologist');
```

Patients (Sample)

```
INSERT INTO Patient (name, gender, dob, contact, insurance_no, blood_type, address) VALUES

('Anna Lee', 'F', '1990-01-05', '3331231111', 'INS12345', 'A+', '123 Main St'),

('Brian Kim', 'M', '1985-06-23', '3331231112', 'INS12346', 'O-', '234 Oak St');
```

Appointment

```
INSERT INTO Appointment (patient_id, doctor_id, date_time, purpose, status, booking_date, booking_method) VALUES

(1, 1, '2025-06-01 09:00:00', 'Routine Checkup', 'Completed', '2025-05-28', 'Online'),

(2, 2, '2025-06-02 10:30:00', 'Skin Allergy', 'Completed', '2025-05-29', 'Phone');
```

Medicine

```
INSERT INTO Medicine (name, price, stock) VALUES

('Paracetamol', 5.00, 200),

('Amoxicillin', 12.00, 150);
```

Medicine Ingredients

```
INSERT INTO Medicine_Ingredient (medicine_id, ingredient) VALUES

(1, 'Paracetamol'),

(2, 'Amoxicillin trihydrate');
```

Medicine Batch

```
INSERT INTO Medicine_Batch (medicine_id, batch_no, expiry_date, quantity) VALUES

(1, 'B202306', '2026-06-01', 100),

(2, 'B202308', '2026-08-01', 150);
```

Diagnoses

```
INSERT INTO Diagnosis (appointment_id, date_time, description) VALUES

(1, '2025-06-01 09:30:00', 'Mild fever, prescribed Paracetamol'),

(2, '2025-06-02 10:45:00', 'Skin rash, prescribed Amoxicillin');
```

Diagnosis Symptoms

```
INSERT INTO Diagnosis_Symptom (diagnosis_id, symptom) VALUES
(1, 'Fever'),
(1, 'Headache'),
(2, 'Rash'),
(2, 'Itching');
```

Prescriptions

```
INSERT INTO Prescribe (appointment_id, medicine_id, dosage, duration,
instruction) VALUES
(1, 1, '500mg', '3 days', 'Take after food'),
(2, 2, '250mg', '5 days', 'Take before food');
```

Bills

```
INSERT INTO Bill (patient_id, billing_date, price) VALUES
(1, '2025-06-01', 50.00),
(2, '2025-06-02', 60.00);
```

Bill_Medicine

```
INSERT INTO Bill_Medicine (bill_id, medicine_id) VALUES
(1, 1),
(2, 2);
```

Explanation

- **Auto-incremented IDs** are used for all main entities to ensure unique identification and simplify foreign key relationships.
- **VARCHAR** is chosen for names, addresses, and descriptive fields to allow flexibility and efficient storage.
- DECIMAL is used for price fields to support monetary values without floating-point errors.
- **DATE** and **DATETIME** store temporal information for appointments, diagnoses, batches, and billing.
- **Junction tables** (e.g., Doctor_Specialization, Diagnosis_Symptom, Medicine_Ingredient, Bill_Medicine) resolve N:M relationships and support database normalization.

This SQL implementation ensures that all user stories can be demonstrated, and that the schema supports efficient, robust hospital operations.

Implementation of User Stories in SQL

Below are SQL query implementations for each of the 25 user stories of the Hospital Management System. The queries are derived directly from the relational schema and sample data, using a range of SQL features: joins, aggregate functions, groupings, conditions, subqueries, and more. Each query is labelled and explained to clearly link it to its respective user story.

Doctor

User Story 1

As a doctor, I want to see all my upcoming appointments with patient details so that I can prepare for each visit.

```
SELECT
  a.appointment_id,
  a.date_time,
  a.purpose,
  a.status,
  p.name AS patient_name,
  p.contact AS patient_contact
FROM
  Appointment a
  JOIN Patient p ON a.patient_id = p.patient_id
WHERE
  a.doctor_id = ? -- (current doctor)
  AND a.date_time > NOW()
ORDER BY
  a.date_time;
```

User Story 2

As a doctor, I want to review a patient's diagnosis history before their next appointment to understand their medical background.

```
SELECT
  a.appointment_id,
  a.date_time,
  d.description AS diagnosis,
  GROUP_CONCAT (ds.symptom) AS symptoms
FROM
  Appointment a
  LEFT JOIN Diagnosis d ON a.appointment_id = d.appointment_id
  LEFT JOIN Diagnosis_Symptom ds ON d.diagnosis_id = ds.diagnosis_id
WHERE
  a.patient_id = 1 -- Patient's ID
  AND a.date_time < NOW()
GROUP BY
```

```
  a.appointment_id, a.date_time, d.description
ORDER BY
  a.date_time DESC;needed
```

User Story 3

As a doctor, I want to see all missed or cancelled appointments for follow-up purposes.

```
SELECT
  a.appointment_id,
  a.date_time,
  a.status,
  p.name AS patient_name
FROM
  Appointment a
  JOIN Patient p ON a.patient_id = p.patient_id
WHERE
  a.doctor_id = 1 --Doctor’s ID
  AND a.status IN ('Cancelled', 'Missed')
ORDER BY
  a.date_time DESC;
```

User Story 4

As a doctor, I want to see a list of all patients with a specific blood type (e.g., 'A+') for potential blood donation or special care.

```
SELECT
  patient_id,
  name,
  blood_type,
  contact
FROM
  Patient
WHERE
  blood_type = 'A+'; --Required Blood Type
```

User Story 5

As a doctor, I want to view a monthly count of appointments over the past year, to analyse my workload trends.

```
SELECT
  DATE_FORMAT(a.date_time, '%Y-%m') AS month,
  COUNT(a.appointment_id) AS appointment_count
FROM
  Appointment a
WHERE
  a.doctor_id = 1
  AND a.date_time >= (CURDATE() - INTERVAL 1 YEAR)
GROUP BY
  month
ORDER BY
  month;
```

Patient

User Story 6

As a patient, I want to see all my appointments, including the doctor and the status.

```
SELECT
  a.appointment_id,
  a.date_time,
  a.status,
  d.name AS doctor_name,
  a.purpose
FROM
  Appointment a
  JOIN Doctor d ON a.doctor_id = d.doctor_id
WHERE
  a.patient_id = 1 -- Anna's patient_id
ORDER BY
  a.date_time DESC;
```

User Story 7

As a patient, I want to view my diagnoses and prescribed medicines for each appointment.

```
SELECT
  a.appointment_id,
  a.date_time,
  d.description AS diagnosis,
  GROUP_CONCAT(ds.symptom) AS symptoms,
  GROUP_CONCAT(m.name, ' (', pr.dosage, ')') AS prescriptions
FROM
  Appointment a
  LEFT JOIN Diagnosis d ON a.appointment_id = d.appointment_id
  LEFT JOIN Diagnosis_Symptom ds ON d.diagnosis_id = ds.diagnosis_id
  LEFT JOIN Prescribe pr ON a.appointment_id = pr.appointment_id
  LEFT JOIN Medicine m ON pr.medicine_id = m.medicine_id
WHERE
  a.patient_id = 1 --Patient’s ID
GROUP BY
  a.appointment_id
ORDER BY
  a.date_time DESC;
```

User Story 8

As a patient, I want to see my bill details, including their prices for each bill.

```
SELECT
  b.bill_id,
  b.billing_date,
  b.price AS total_price,
  GROUP_CONCAT(m.name, ' (', m.price, ')') AS medicines
FROM
```

```
  Bill b
  JOIN Bill_Medicine bm ON b.bill_id = bm.bill_id
  JOIN Medicine m ON bm.medicine_id = m.medicine_id
WHERE
  b.patient_id = ?
GROUP BY
  b.bill_id, b.billing_date, b.price
ORDER BY
  b.billing_date DESC;
```

User Story 9

As a patient, I want to know which doctors I see most often for my ongoing care.

```
SELECT
  d.name AS doctor_name,
  COUNT(a.appointment_id) AS num_visits
FROM
  Appointment a
  JOIN Doctor d ON a.doctor_id = d.doctor_id
WHERE
  a.patient_id = 1
GROUP BY
  d.doctor_id, d.name
ORDER BY
  num_visits DESC;
```

User Story 10

As a patient, I want to review all my cancelled or rescheduled appointments to keep track.

Cancel:

```
SELECT
  appointment_id,
  date_time,
  status,
  purpose
FROM
  Appointment
WHERE
  patient_id = 1
  AND status IN ('Cancelled', 'Rescheduled')
ORDER BY
  date_time DESC;
```

Pharmacist/Staff

User Story 11

As pharmacy staff, I want to view all prescriptions issued by doctors to prepare medicines for dispensing.

```
SELECT
  pr.appointment_id,
  m.name AS medicine,
  pr.dosage,
  pr.duration,
  pr.instruction,
  a.date_time,
  d.name AS doctor
FROM
  Prescribe pr
  JOIN Medicine m ON pr.medicine_id = m.medicine_id
  JOIN Appointment a ON pr.appointment_id = a.appointment_id
  JOIN Doctor d ON a.doctor_id = d.doctor_id
ORDER BY
  a.date_time DESC;
```

User Story 12

As a pharmacist, I want to see the current inventory of all medicines.

Dispense:

```
SELECT

  medicine_id,

  name,

  stock

FROM

  Medicine

ORDER BY

  stock;
```

User Story 13

As a pharmacist, I want to see a list of all medicines along with their ingredients.

Generate bill for printing:

```
SELECT
  m.medicine_id,
  m.name AS medicine_name,
  GROUP_CONCAT(mi.ingredient ORDER BY mi.ingredient SEPARATOR ', ')
AS ingredients
FROM Medicine m
LEFT JOIN Medicine_Ingredient mi
```

```
  ON m.medicine_id = mi.medicine_id
GROUP BY m.medicine_id, m.name
ORDER BY m.name;
```

User Story 14

As pharmacy staff, I want to check the expiry dates of all medicines in stock to avoid dispensing expired products.

```
SELECT
  m.name,
  mb.batch_no,
  mb.expiry_date,
  mb.quantity
FROM
  Medicine_Batch mb
  JOIN Medicine m ON mb.medicine_id = m.medicine_id
WHERE
  mb.expiry_date <= (CURDATE() + INTERVAL 3 MONTH)
ORDER BY
  mb.expiry_date ASC;
```

User Story 15

As a pharmacist, I want to see a list of all medicines that have not been prescribed to any patient in the last 6 months.

```
SELECT
  m.medicine_id,
  m.name AS medicine_name,
  m.stock
FROM Medicine m
LEFT JOIN (
  SELECT DISTINCT medicine_id
  FROM Prescribe pr
  JOIN Appointment a USING (appointment_id)
  WHERE a.date_time >= CURDATE() - INTERVAL 6 MONTH
) recent_pr ON m.medicine_id = recent_pr.medicine_id
WHERE recent_pr.medicine_id IS NULL
ORDER BY m.name;
```

Nurse

User Story 16

As a nurse, I want to see the list of today's appointments and their status to manage patient flow efficiently.

```
SELECT
  a.appointment_id,
  a.date_time,
  a.status,
  p.name AS patient_name,
  d.name AS doctor_name
FROM
  Appointment a
  JOIN Patient p ON a.patient_id = p.patient_id
  JOIN Doctor d ON a.doctor_id = d.doctor_id
WHERE
  DATE(a.date_time) = CURDATE()
ORDER BY
  a.date_time;
```

User Story 17

As a nurse, I want to see which patients have arrived for their appointments today.

```
SELECT
  a.appointment_id,
  a.date_time,
  p.name AS patient_name,
  d.name AS doctor_name
FROM
  Appointment a
  JOIN Patient p ON a.patient_id = p.patient_id
  JOIN Doctor d ON a.doctor_id = d.doctor_id
WHERE
  DATE(a.date_time) = CURDATE()
  AND a.status = 'Completed'
ORDER BY
  a.date_time;
```

User Story 18

As a nurse, I want to see all medicines prescribed to a patient for correct administration.

```
SELECT

  a.appointment_id,

  a.date_time,

  m.name AS medicine,

  pr.dosage,

  pr.duration,

  pr.instruction

FROM

  Appointment a

  JOIN Prescribe pr ON a.appointment_id = pr.appointment_id

  JOIN Medicine m ON pr.medicine_id = m.medicine_id

WHERE
```

```
  a.patient_id = 1

ORDER BY

  a.date_time DESC;
```

User Story 19

As a nurse, I want to view and print a summary of a patient's prescribed medicines and instructions for in-patient care.

```
SELECT
  a.appointment_id,
  a.date_time,
  m.name AS medicine,
  pr.dosage,
  pr.duration,
  pr.instruction
FROM
  Appointment a
  JOIN Prescribe pr ON a.appointment_id = pr.appointment_id
  JOIN Medicine m ON pr.medicine_id = m.medicine_id
WHERE
  a.patient_id = ?
ORDER BY
  a.date_time DESC;
```

User Story 20

As a nurse, I want to list all appointments today for patients under 18 years old, to prepare special care for children.

```
SELECT
  a.appointment_id,
  p.name AS patient_name,
  p.gender,
  a.date_time,
  d.name AS doctor_name
FROM
  Appointment a
  JOIN Patient p ON a.patient_id = p.patient_id
  JOIN Doctor d ON a.doctor_id = d.doctor_id
WHERE
  DATE(a.date_time) = CURDATE()
  AND (YEAR(CURDATE()) - YEAR(p.dob)) < 18
ORDER BY
  a.date_time;
```

Admin/System

User Story 21

As an admin, I want to see all users (doctors and patients) in the system.

```
SELECT 'Doctor' AS user_type,
       d.doctor_id AS user_id,
       d.name,
       d.contact_no AS contact,
       d.email_id AS email
FROM Doctor d

UNION ALL

SELECT 'Patient' AS user_type,
       p.patient_id AS user_id,
       p.name,
       p.contact AS contact,
       NULL AS email
FROM Patient p

ORDER BY user_type, name;
```

User Story 22

As an admin, I want to see all medical specializations available in the hospital.

```
SELECT DISTINCT specialization FROM Doctor_Specialization;
```

User Story 23

As an admin, I want a summary of appointment counts per doctor by day to monitor workload distribution.

```
SELECT
  a.doctor_id,
  d.name,
  DATE(a.date_time) AS day,
  COUNT(*) AS appt_count
FROM Appointment a
JOIN Doctor d USING(doctor_id)
GROUP BY a.doctor_id, d.name, day
ORDER BY a.doctor_id, day DESC;
```

User Story 24

As an admin, I want to see the total revenue generated by the hospital in each month of the year for financial analysis.

```
SELECT
  YEAR(b.billing_date) AS year,
  MONTH(b.billing_date) AS month,
  SUM(b.price) AS total_revenue
FROM Bill b
GROUP BY YEAR(b.billing_date), MONTH(b.billing_date)
ORDER BY year, month;
```

User Story 25

As an admin, I want to list the top 5 patients with the highest number of visits (appointments) this year to identify our most engaged patients.

```
SELECT
  p.patient_id,
  p.name AS patient_name,
  COUNT(a.appointment_id) AS appointment_count
FROM Patient p
JOIN Appointment a ON p.patient_id = a.patient_id
WHERE a.date_time >= CURDATE() - INTERVAL 1 MONTH
GROUP BY p.patient_id, p.name
ORDER BY appointment_count DESC
LIMIT 5;
```

Reflection

The database design for the Hospital Management System focused on several key aspects: patient management, appointment scheduling, diagnoses, prescriptions, pharmacy inventory, billing, and user roles. The primary goals were to avoid redundancy, ensure data integrity, and provide flexible support for complex healthcare workflows.

Evaluation of the Schema and Queries

The normalized schema—achieving BCNF—proved highly effective for implementing a wide range of queries. Joins between entities such as Patient, Doctor, Appointment, Diagnosis, and Medicine were straightforward, supporting queries that required combining multiple types of information (e.g., patient histories, prescription overviews, billing summaries). The use of junction tables for many-to-many relationships (like Doctor_Specialization, Prescribe, Bill_Medicine) ensured that data remained atomic and easily queryable, with no update or deletion anomalies.

Complex queries, such as filtering appointments by doctor or patient, retrieving a patient's full medical history, or generating pharmacy inventory reports, could be executed efficiently. The schema's separation of multi-valued attributes into their own tables (e.g., symptoms, ingredients, specializations) made aggregate operations, flexible searching, and future attribute expansion simple.

Areas for Improvement and Alternative Designs

While the schema supports the user stories and most operational queries well, some areas could be improved or expanded:

- **User Authentication & Authorization**: User roles (doctor, nurse, admin, pharmacist, patient) are implied but not directly managed in the schema. Integrating a dedicated user management and roles table would enhance security and auditability.

- **Audit Logging**: While the schema supports core operations, explicit audit logging for critical actions (e.g., who updated a record and when) would provide better traceability and compliance, especially valuable in healthcare.
- **Handling Allergies & Medical History**: Patient allergies, chronic conditions, and family medical history could be modelled more explicitly for more comprehensive care.
- **Appointment Notes & Attachments**: Structured support for visit notes, attached files (e.g., reports, images), or standardized observations would make the system more robust for real-world use.
- **Internationalization**: If deploying in multiple regions, additional fields for localization (e.g., contact formats, insurance requirements) should be considered.
- **Deletions & Soft Deletes**: For legal compliance, implementing soft deletes (logical removal) and versioning for sensitive records might be preferable to physical deletion.

Expandability & Future Extensions

The schema is highly expandable:

- New entities (such as departments, wards, insurance providers, or medical devices) can be added without major rework.
- Additional relationships (e.g., patient-doctor associations, care teams) can be modeled through new linking tables.
- Extension for electronic health records (EHR), appointment reminders, telemedicine integration, or advanced reporting can be supported with only incremental changes.

Insights Gained

The normalization process greatly simplified query writing and maintenance, especially for joins and aggregate queries. Careful attention to junction tables and key constraints ensured data consistency and performance. The most challenging queries involved filtering overlapping appointments and handling many-to-many relationships, but these were manageable with the chosen structure.

Clear naming conventions and schema documentation proved invaluable for both implementation and query writing. The modular design means the database will remain maintainable and adaptable as requirements evolve.