Hospital Database Project

Name	ID	Tasks
Linah Ebrahim Al-sayed	445002548	Writing the project report Drawing the UML diagram
Retaj Hussain Alhazmi	445006594	Drawing the ER diagram Assisting with code
Malath muneallah alsaeedi	443002873	Normalizing the relational schema
Mayas Hasan Abu Al farraj	445006876	Drawing the relational schema diagram
Ghala Shahir Albshri	445006326	Writing and executing SQL code Business rules

Supervision By: Dr. Emtithal Sultan Al-Afghani

1 Introduction

1.1 Problem Statement

Hospitals manage a vast amount of information related to patients, doctors, appointments, treatments, and room allocations. Without an efficient system, it becomes challenging to accurately store, retrieve, and manage this data, which may lead to errors, delays in patient care, and resource mismanagement.

The problem is to design and implement a robust database system that effectively organizes and maintains hospital data, ensuring data integrity, easy accessibility, and support for hospital operations such as patient registration, appointment scheduling, treatment tracking, and room management.

This project aims to develop a relational database model that reflects the real-world hospital environment, supports complex queries, and facilitates efficient data manipulation to improve the overall management and delivery of healthcare services.

1.2 Project Overview

Our project is a hospital database management system designed to efficiently handle critical information related to patients, doctors, appointments, treatments, and room allocations. The system uses a relational database to store and manage this data, ensuring accuracy and consistency.

Upon using the system, hospital staff can register new patients, schedule appointments with doctors, record treatments provided, and manage room availability. The system supports complex queries to help in decision making and operational workflows.

The database is implemented using SQL commands and designed with clear entity relationships to reflect real-world hospital operations. The project does not include a user interface but focuses on the backend data management and integrity, providing a solid foundation for future application development.

1.3 Motivation

We chose this hospital database project because healthcare is a fundamental aspect of society that deeply affects people's lives. Managing patient information, doctor schedules, and treatments efficiently can significantly improve the quality of care provided. We believe that building a reliable database system for a hospital will help us understand how data management supports critical real-world applications.

Additionally, this project gives us a valuable opportunity to strengthen our skills in database design, SQL programming, and conceptual modeling. These skills are essential for building robust systems that handle complex data relationships, which is a key challenge in many professional fields, especially healthcare.

2 Design and Implementation

2.1 Why we chose this approach

Choice of Relational Database:

We chose a relational database model because it provides a well-structured way to represent complex relationships between hospital entities such as patients, doctors, appointments, treatments, and rooms. Relational databases ensure data integrity through constraints like primary and foreign keys, which are essential for maintaining accurate healthcare records.

Use of SQL for Implementation:

SQL was selected as the language to define, manipulate, and query the data because it is widely supported, standardized, and powerful enough to handle complex queries needed in hospital management systems. Using SQL enables us to perform insertions, updates, deletions, and sophisticated data retrieval with ease.

ER Modeling with Chen’s and UML Notations:

We used both Chen’s notation and UML diagrams to represent the database design conceptually and visually. Chen’s notation helps in clearly identifying entities, attributes, and relationships, while UML provides a familiar and structured approach to system modeling favored in software engineering.

Normalization Process:

Normalization was applied to eliminate data redundancy and ensure data consistency, which is crucial in healthcare data to avoid discrepancies that can affect patient treatment.

Project Organization:

The project files were organized into separate folders for SQL scripts, reports, and diagrams. This separation improves maintainability and makes collaboration among team members easier.

2.2 Business Rules

We have a hospital system that provides medical services to patients and the hospital system consists of:

There are a lot of patients registered in the hospital. The identifier for the patient is PatientID. Other attributes are First Name, Last Name, Gender, and Date of Birth.

The hospital employs many doctors who specialize in different fields. The identifier for a doctor is DoctorID. Other attributes are First Name, Last Name, Specialization, and Phone Number.

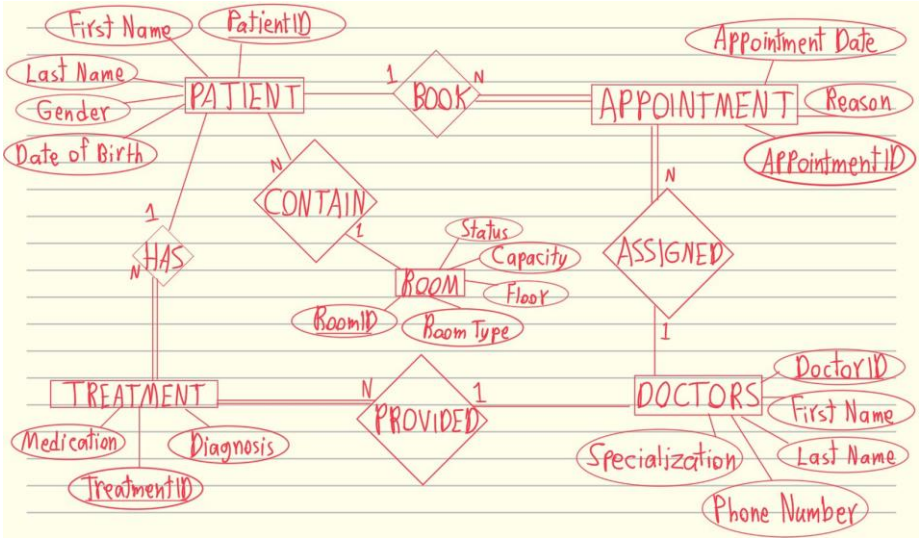
Each patient can book one or more appointments with doctors. A single appointment is uniquely identified by AppointmentID. Each appointment includes the Appointment Date, Reason, and links a patient to a specific doctor. Every appointment must be for a registered patient and assigned to a registered doctor.

The hospital has many rooms where patients can be treated or admitted. The identifier for the room is RoomID. Other attributes include Room Type, Floor, Capacity, and Status. Each room can be either available or occupied. The hospital may assign rooms to patients.

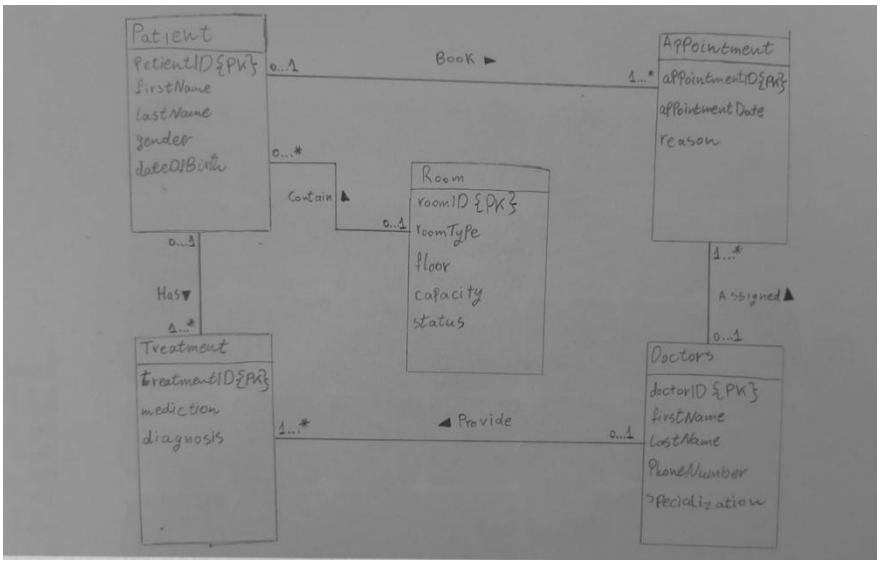
Each treatment is provided by a doctor to a patient. The identifier for treatment is TreatmentID. Other attributes are Diagnosis and Medication. Every treatment record must be linked to one existing patient and one existing doctor. A treatment cannot exist without a patient and a doctor.

2.3 **Diagrams** (ER Model)

2.3.1 Chen's Notation



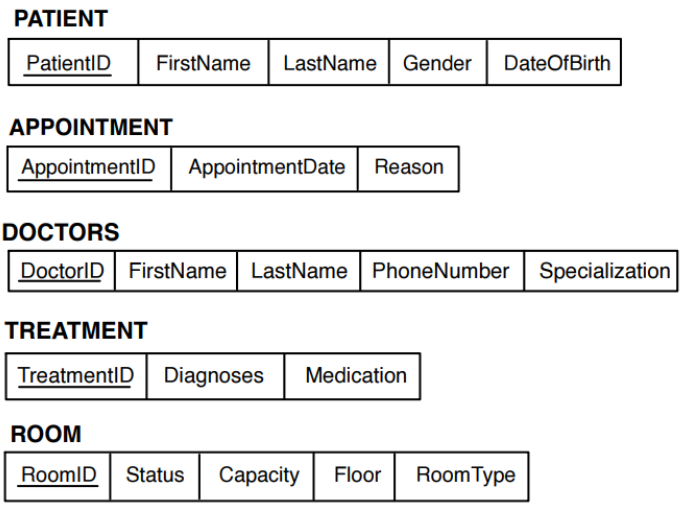
2.3.2 UML



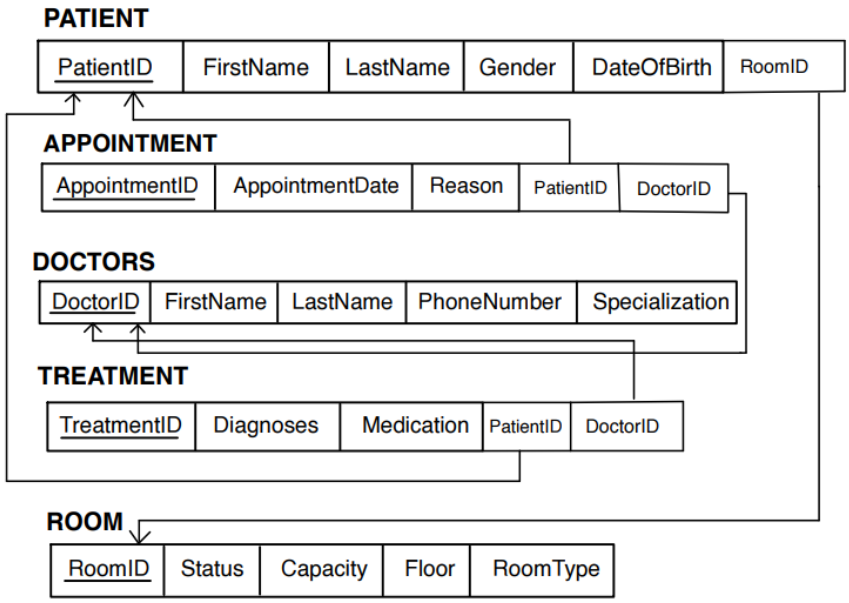
2.4 **Diagram** (Mapping of ER into Relational Schema)

2.4.1 The Relation Schema

Step 1: Mapping of **Regular** Entity Types



- Step 2: Mapping of **Weak** Entity Types **NONE**.
- Step 3: Mapping of Binary **1:1** Relationship Types **NONE**.
- Step 4: Mapping of Binary **1:N** Relationship Types



- Step 5: Mapping of Binary **M:N** Relationship Types **NONE**.

• Step6: Mapping **of Multivalued** attributes **NONE.**

• Step7: Mapping of **N-ary Relationship** Types **NONE.**

2.4.2 The normalization process of the Relation Schema

2.4.2.1 Normalize Patient table

Patient Table

PatientID	FirstName	LastName	Gender	DateOfBirth	RoomID
P001	Ali	Hassan	M	1990-05-10	R001
P002	Sara	Ahmed	F	1985-11-22	R003
P003	Mona	Khalid	F	2000-02-14	*NULL*
P004	Omar	Zaid	M	1978-08-30	R004
P005	Noura	Salem	F	1995-03-05	*NULL*

The table is already normalized

2.4.2.2 Normalize Doctor table

Doctor

DoctorID	FirstName	LastName	Specialization	Phone
D001	Faisal	Ali	Cardiologist	0501234567
D002	Layla	Nasir	Pediatrician	0502345678
D003	Sami	Yousuf	Orthopedic	0503456789
D004	Amal	Tarek	Neurologist	0504567890
D005	Nabil	Fadi	General	0505678901

The table is already normalized

2.4.2.3 Normalize Room table

Room

RoomID	RoomType	Floor	Capacity	Status
R001	ICU	1	1	Occupied
R002	General	2	2	Available
R003	General	3	4	Occupied
R004	ICU	1	1	Available
R005	General	2	3	Available

The table is already normalized

2.4.2.4 Normalize Appointment table

Appointment

AppointmentID	PatientID	DoctorID	AppointmentDate	Reason
A001	P001	D005	2025-07-29	General Checkup
A002	P003	D002	2025-08-01	Fever
A003	P002	D001	2025-08-03	Chest Pain
A004	P005	D003	2025-08-05	Back Pain
A005	P004	D004	2025-08-10	Headache

The table is already normalized

2.4.2.5 Normalize Treatment table

Treatment

TreatmentID	PatientID	DoctorID	Diagnosis	Medication
T001	P001	D005	Flu	Paracetamol
T002	P002	D001	Hypertension	Amlodipine
T003	P004	D004	Migraine	Ibuprofen
T004	P003	D002	Viral Infection	Tamiflu
T005	P005	D003	Sprain	Rest & Painkiller

1NF:

Treatment

TreatmentID	PatientID(FK)	DoctorID(FK)	Diagnosis	Medication

The table is already in First Normal Form (1NF)

2NF:

Treatment

TreatmentID	PatientID(FK)	DoctorID(FK)	Diagnosis	Medication

The table is already in Second Normal Form (2NF)

3NF:

Treatment

TreatmentID	PatientID(FK)	DoctorID(FK)

Diagnosis

Diagnosis	Medication

2.5 Code snippets And justification of decisions

2.5.1 create schema

2.5.1.1 Create Database Schema

```
CREATE SCHEMA IF NOT EXISTS hospital_dbpr;
```

- Checks if a database schema named **hospital_db** already exists.
- If it does **not** exist, it creates a new schema called **hospital_db**.
- The schema acts as a container or folder to organize all related hospital data.
- Using **IF NOT EXISTS** avoids errors if the schema already exists.

2.5.1.2 Use the Database Schema

```
USE hospital_dbpr;
```

- Sets **hospital_db** as the current active database.
- All following commands will apply to this database by default.
- This is like opening the folder where all hospital data will be stored and managed.

2.5.2 create tables

2.5.2.1 Create Patient Table

```
-- Create the Patient table
CREATE TABLE Patient (
    PatientID VARCHAR(10) PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Gender CHAR(1),
    DateOfBirth DATE,
    RoomID VARCHAR(10),
    FOREIGN KEY (RoomID) REFERENCES Room(RoomID)
);
```

- Creates a table named **Patient** to store patient information.
- Defines **PatientID** as a unique identifier for each patient.
- Stores patient's **FirstName** and **LastName** for identification.
- **Gender** is stored as a single character representing the patient's gender.
- **DateOfBirth** records the patient's birthdate, important for medical history.
- **RoomID** links each patient to a specific room where they are admitted.
- The **FOREIGN KEY** constraint ensures that the RoomID assigned to a patient must exist in the **Room** table, maintaining data consistency and integrity.
- This table represents the people who inhabit the rooms, connecting human data with physical locations.

2.5.2.2 Create Doctor Table

```
-- Create the Doctor table
CREATE TABLE Doctor (
    DoctorID VARCHAR(10) PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Specialization VARCHAR(50),
    Phone VARCHAR(15)
);
```

- Creates a table named **Doctor** to store details about doctors working in the hospital.
- **DoctorID** is a unique identifier for each doctor, ensuring every doctor is distinct in the system.
- **FirstName** and **LastName** capture the doctor's personal identity.
- **Specialization** describes the medical field or expertise area of the doctor (e.g., cardiology, neurology).
- **Phone** stores contact information for communication purposes.
- This table forms the foundation for managing medical professionals within the hospital's database.

2.5.2.3 Create Room Table

```
-- Create the Room table
CREATE TABLE Room (
    RoomID VARCHAR(10) PRIMARY KEY,
    RoomType VARCHAR(20),
    Floor INT,
    Capacity INT,
    Status VARCHAR(10)
);
```

- Checks if a table named **Room** is created to store information about hospital rooms.

- Defines **RoomID** as a unique identifier (primary key) for each room, ensuring no two rooms share the same ID.
- Specifies **RoomType** to describe the type of the room (e.g., ICU, general ward).
- The **Floor** column indicates the floor number where the room is located.
- **Capacity** shows how many patients the room can hold.
- **Status** reflects whether the room is available, occupied, or under maintenance.
- This table acts as a blueprint to organize and manage physical spaces within the hospital.

2.5.2.4 Create Appointment Table

```
-- Create the Appointment table with foreign keys to Patient and Doctor
CREATE TABLE Appointment (
    AppointmentID VARCHAR(10) PRIMARY KEY,
    PatientID VARCHAR(10),
    DoctorID VARCHAR(10),
    AppointmentDate DATE,
    Reason VARCHAR(100),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);
```

- Establishes a table named **Appointment** to track meetings between patients and doctors.
- **AppointmentID** uniquely identifies each appointment.
- **PatientID** links the appointment to the patient involved.
- **DoctorID** connects the appointment to the doctor who will attend the patient.
- **AppointmentDate** records the specific date of the meeting.
- **Reason** provides a brief description or purpose for the appointment.
- The **FOREIGN KEY** constraints ensure that both the patient and doctor referenced in the appointment exist in their respective tables, preserving relational integrity.
- This table acts as the bridge between people (patients and doctors) and time (appointments), weaving together the story of care and healing.

2.5.2.5 Create Treatment Table

```
-- Create the Treatment table with foreign keys to Patient and Doctor
CREATE TABLE Treatment (
    TreatmentID VARCHAR(10) PRIMARY KEY,
    PatientID VARCHAR(10),
    DoctorID VARCHAR(10),
    Diagnosis VARCHAR(100),
    Medication VARCHAR(100),
    FOREIGN KEY (PatientID) REFERENCES Patient(PatientID),
    FOREIGN KEY (DoctorID) REFERENCES Doctor(DoctorID)
);
```

- Creates a table named Treatment to record medical treatments given to patients.
- TreatmentID is a unique identifier for each treatment record, ensuring every treatment is distinct.
- PatientID links the treatment to the specific patient receiving care.
- DoctorID connects the treatment to the doctor responsible for diagnosing and prescribing.
- Diagnosis describes the medical condition identified during the treatment.
- Medication lists the drugs or therapies prescribed for the patient.
- The FOREIGN KEY constraints guarantee that both the patient and doctor referenced in the treatment exist in their respective tables, maintaining data consistency and relational integrity.
- This table captures the essence of the healing process, linking patients, doctors, diagnoses, and prescribed care into a single narrative.

2.5.3 insert values

2.5.3.1 Insert data into Patient table

```
-- Insert sample data into Patient table
INSERT INTO Patient VALUES
('P001', 'Ali', 'Hassan', 'M', '1990-05-10', 'R001'),
('P002', 'Sara', 'Ahmed', 'F', '1985-11-22', 'R003'),
('P003', 'Mona', 'Khalid', 'F', '2000-02-14', NULL),
('P004', 'Omar', 'Zaid', 'M', '1978-08-30', 'R004'),
('P005', 'Noura', 'Salem', 'F', '1995-03-05', NULL);
```

INSERT INTO Patient → Add new patients to the Patient table.

Value Position	Meaning
'P001'	Patient ID (primary key)
'Ali'	First Name
'Hassan'	Last Name
'M' / 'F'	Gender (M = Male, F = Female)
'1990-05-10'	Date of Birth (in YYYY-MM-DD format)
'R001'	Room ID where the patient is assigned (NULL means not assigned)

NULL → SQL keyword that means “no value” or “unknown” (patient is not in any room).

2.5.3.2 Insert data into Doctor table

```
-- Insert sample data into Doctor table
INSERT INTO Doctor VALUES
('D001', 'Faisal', 'Ali', 'Cardiologist', '0501234567'),
('D002', 'Layla', 'Nasir', 'Pediatrician', '0502345678'),
('D003', 'Sami', 'Yousuf', 'Orthopedic', '0503456789'),
('D004', 'Amal', 'Tarek', 'Neurologist', '0504567890'),
('D005', 'Nabil', 'Fadi', 'General', '0505678901');
```

INSERT INTO Doctor → Add doctors to the Doctor table.

Each () is a separate doctor.

Value Position	Meaning
'D001'	Doctor ID (unique for each doctor)
'Faisal'	First Name
'Ali'	Last Name

Value Position	Meaning
'Cardiologist'	Specialization (doctor's field)
'0501234567'	Contact number (Phone)

2.5.3.3 Insert data into Room table

```
-- Insert sample data into Room table
INSERT INTO Room VALUES
('R001', 'ICU', 1, 1, 'Occupied'),
('R002', 'General', 2, 2, 'Available'),
('R003', 'General', 3, 4, 'Occupied'),
('R004', 'ICU', 1, 1, 'Available'),
('R005', 'General', 2, 3, 'Available');
```

- INSERT INTO Room → Insert new rows into the table called Room.
- VALUES → Keyword to specify the actual data being inserted.
- Each line between () represents **one row**.

2.5.3.4 Insert data into Appointment table

```
-- Insert sample data into Appointment table
INSERT INTO Appointment VALUES
('A001', 'P001', 'D005', '2025-07-29', 'General Checkup'),
('A002', 'P003', 'D002', '2025-08-01', 'Fever'),
('A003', 'P002', 'D001', '2025-08-03', 'Chest Pain'),
('A004', 'P005', 'D003', '2025-08-05', 'Back Pain'),
('A005', 'P004', 'D004', '2025-08-10', 'Headache');
```

INSERT INTO Appointment → Add visit records to the Appointment table.

Value Position	*Meaning*
'A001'	*Appointment ID (unique per appointment)*
'P001'	*Patient ID (linked to Patient table)*
'D005'	*Doctor ID (linked to Doctor table)*
'2025-07-29'	*Date of the appointment (YYYY-MM-DD)*
'General Checkup'	*Reason for visit*

2.5.3.5 insert data into Treatment table

```
-- Insert sample data into Treatment table
INSERT INTO Treatment VALUES
('T001', 'P001', 'D005', 'Flu', 'Paracetamol'),
('T002', 'P002', 'D001', 'Hypertension', 'Amlodipine'),
('T003', 'P004', 'D004', 'Migraine', 'Ibuprofen'),
('T004', 'P003', 'D002', 'Viral Infection', 'Tamiflu'),
('T005', 'P005', 'D003', 'Sprain', 'Painkiller');
```

Value Position	**Meaning**
'T001'	Treatment ID (unique identifier for this treatment case)
'P001'	Patient ID → Who is receiving the treatment
'D005'	Doctor ID → Who prescribed the treatment
'Flu'	Diagnosis (what the patient has)
'Paracetamol'	Medication given to the patient

2.5.3.6 Insert New Appointment

```
INSERT INTO Appointment (AppointmentID, PatientID, DoctorID, AppointmentDate, Reason)
VALUES ('A006', 'P001', 'D001', '2025-08-15', 'Follow-up');
```

- INSERT INTO Appointment (...) → Add a new appointment record
- AppointmentID = 'A006' → Unique ID
- PatientID = 'P001' → Refers to patient Ali
- DoctorID = 'D001' → Refers to doctor Faisal

- AppointmentDate = '2025-08-15'
- Reason = 'Follow-up'

This creates a connection between an existing patient and doctor for a future visit

2.5.4 Select (Where, Group by, Having, Order by, subqueries, join)

*2.5.4.1 Select With Condition (**Where**)*

```
-- Select all female patients
SELECT * FROM Patient
WHERE Gender = 'F';
```

- SELECT * → Fetch **all columns** (first name, last name, etc.).
- FROM Patient → From the **Patient** table.
- WHERE Gender = 'F' → Only show rows where the gender is **female** (F = Female).

The Output:

PatientID	FirstName	LastName	Gender	DateOfBirth
P002	Sara	Ahmed	F	1985-11-22
P003	Mona	Khalid	F	2000-02-14
P005	Noura	Salem	F	1995-03-05
NULL	NULL	NULL	NULL	NULL

*2.5.4.2 Aggregate Query AND Grouping (**Group by**)*

```
-- Count of appointments per doctor
SELECT DoctorID, COUNT(*) AS AppointmentCount
FROM Appointment
GROUP BY DoctorID;
```

- SELECT DoctorID → Show the ID of the doctor.
- COUNT(*) AS AppointmentCount → Count how many appointments each doctor has.
  - ❖ COUNT(*) → Counts the number of rows per doctor
  - ❖ AS AppointmentCount → Gives the column an alias name (for readability).
- FROM Appointment → Source table is **Appointment**

- GROUP BY DoctorID → Group rows by doctor; count will be done per group.

This is how we generate **summary statistics** (analytics) in SQL.

The Output:

	DoctorID	AppointmentCount
▶	D001	1
	D002	1
	D003	1
	D004	1
	D005	1

*2.5.4.3 Ordering Results (**Order by**)*

```
-- List doctors ordered by specialization
SELECT * FROM Doctor
ORDER BY Specialization ASC;
```

- SELECT * FROM Doctor → Select all columns for all doctors.
- ORDER BY Specialization → Sort the output **by the Specialization column**.
- ASC → Sort in **Ascending** order (A → Z).

You could use DESC for **Descending** order (Z → A).

The Output:

	DoctorID	FirstName	LastName	Specialization	Phone
▶	D001	Faisal	Ali	Cardiologist	0501234567
	D005	Nabil	Fadi	General	0505678901
	D004	Amal	Tarek	Neurologist	0504567890
	D003	Sami	Yousuf	Orthopedic	0503456789
	D002	Layla	Nasir	Pediatrician	0502345678
*	NULL	NULL	NULL	NULL	NULL

*2.5.4.4 Subquery -Nested Query- (**subqueries**)*

```
-- Doctors who treated Hypertension
SELECT * FROM Doctor
WHERE DoctorID IN (
    SELECT DoctorID FROM Treatment
    WHERE Diagnosis = 'Hypertension'
);
```

Outer Query:

```
SELECT * FROM Doctor
WHERE DoctorID IN ( ... );
```

Find doctors **whose ID is in the result of the subquery below**.

Inner Subquery:

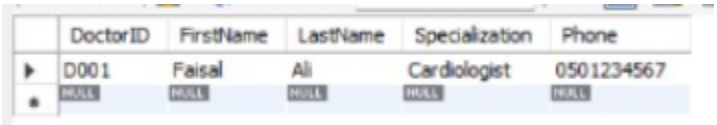
```
SELECT DoctorID FROM Treatment
WHERE Diagnosis = 'Hypertension';
```

This returns the IDs of doctors who treated a patient diagnosed with **Hypertension**.

Logic:

- First, find all DoctorIDs from Treatment table **where Diagnosis = 'Hypertension'`.
- Then, retrieve all details from Doctor table for those doctors only.

The Output:



DoctorID	FirstName	LastName	Specialization	Phone
D001	Faisal	Ali	Cardiologist	0501234567
NULL	NULL	NULL	NULL	NULL

*2.5.4.5 Join Appointments with Patient and Doctor (**Joint**)*

```
-- Join appointments with patient and doctor names
SELECT
    a.AppointmentID,
    p.FirstName AS PatientName,
    d.FirstName AS DoctorName,
    a.AppointmentDate,
    a.Reason
FROM Appointment a
JOIN Patient p ON a.PatientID = p.PatientID
JOIN Doctor d ON a.DoctorID = d.DoctorID;
```

- SELECT → Choose which columns to display
- a.AppointmentID → From alias a (table: Appointment)
- p.FirstName AS PatientName → From alias p (table: Patient), rename column to PatientName
- d.FirstName AS DoctorName → From alias d (table: Doctor), rename column to DoctorName
- a.AppointmentDate, a.Reason → More fields from Appointment

JOINS:

Clause	Meaning
`JOIN Patient p ON a.PatientID = p.PatientID`	→ Connects each appointment with its patient info
`JOIN Doctor d ON a.DoctorID = d.DoctorID`	→ Connects each appointment with its doctor info

Table Aliases:

Alias	Table
a	Appointment
p	Patient
d	Doctor

The Output:

AppointmentID	PatientName	DoctorName	AppointmentDate	Reason
A001	Ali	Nabil	2025-07-29	General Checkup
A002	Mona	Layla	2025-08-01	Fever
A003	Sara	Faisal	2025-08-03	Chest Pain
A004	Noura	Sami	2025-08-05	Back Pain
A005	Omar	Amal	2025-08-10	Headache

*2.5.4.6 Find Doctors With Multiple Appointments (**Having**)*

```
-- Doctors with more than one appointment
SELECT DoctorID, COUNT(*) AS AppointmentCount
FROM Appointment
GROUP BY DoctorID
HAVING COUNT(*) > 1;
```

- SELECT DoctorID, COUNT(*) → Count number of appointments per doctor
- GROUP BY DoctorID → Group rows by DoctorID
- HAVING COUNT(*) > 1 → Only include doctors with **more than 1 appointment**

Difference Between:

Clause	Used For
`WHERE`	Before grouping (row-level filters)
`HAVING`	After grouping (group-level filters)

The Output:

DoctorID	AppointmentCount
D001	2

2.5.5 Update

```
-- Update patient's last name
UPDATE Patient
SET LastName = 'AlHassan'
WHERE PatientID = 'P001';
```

- UPDATE Patient → Modify a row in the Patient table.
- SET LastName = 'AlHassan' → Change the value of the LastName field to 'AlHassan'.
- WHERE PatientID = 'P001' → Only apply this change to the patient with ID P001.
- NOTE: Without the WHERE clause, all patients' last names would be changed (dangerous!).

Before Update:

Result Grid | Filter Rows: | Edit:

PatientID	FirstName	LastName	Gender	DateOfBirth	RoomID
P001	Ali	Hassan	M	1990-05-10	R001
P002	Sara	Ahmed	F	1985-11-22	R003
P003	Mona	Khalid	F	2000-02-14	NULL
P004	Omar	Zaid	M	1978-08-30	R004
P005	Noura	Salem	F	1995-03-05	NULL
NULL	NULL	NULL	NULL	NULL	NULL

The Output (after Update):

PatientID	FirstName	LastName	Gender	DateOfBirth
P001	Ali	AlHassan	M	1990-05-10
P002	Sara	Ahmed	F	1985-11-22
P003	Mona	Khalid	F	2000-02-14
P004	Omar	Zaid	M	1978-08-30
P005	Noura	Salem	F	1995-03-05
NULL	NULL	NULL	NULL	NULL

2.5.6 delete

```
-- Delete a room record
DELETE FROM Room
WHERE RoomID = 'R005';
```

- DELETE FROM Room → Remove rows from the **Room** table.

- WHERE RoomID = 'R005' → Only delete the room whose ID is 'R005'.
- Again, WHERE is very important. Without it, **the entire Room table** would be deleted.

The Output:

RoomID	RoomType	Floor	Capacity	Status
R001	ICU	1	1	Occupied
R002	General	2	2	Available
R003	General	3	4	Occupied
R004	ICU	1	1	Available
NULL	NULL	NULL	NULL	NULL

3 Conclusion

3.1 Summary of Achievements:

A hospital management database system was designed and implemented using SQL, aiming to efficiently organize and manage hospital-related data. The system includes five main entities: **Patient, Doctor, Appointment, Room, and Treatment**, each with its relevant attributes and relationships.

The database was created and populated using **MySQL Workbench**, with complete **data definition** and **data manipulation** commands applied. Queries included a variety of operations such as **selection with conditions**, **grouping and aggregation**, **ordering**, **joining multiple tables**, and **subqueries**.

An Entity-Relationship model was designed using both **Chen's** and **UML** notations. The model was then mapped into a **relational schema**, and normalization steps were applied to ensure data consistency and eliminate redundancy. The final database structure supports efficient data retrieval and reflects real-world hospital operations clearly and logically.

3.2 Limitations and Challenges:

While the hospital database system successfully models core hospital operations, it does have some limitations. The current system focuses primarily on **basic patient-doctor interactions**, without covering more advanced modules such as **billing, staff scheduling, or medical history tracking**.

Additionally, the system **relies on static sample data** with limited rows, which may not reflect the complexity or volume of real-world hospital data. No advanced **security measures or user authentication layers** were implemented, as the focus was solely on the structural and query aspects of the database.

Some challenges encountered during development included ensuring **referential integrity between multiple entities**, designing meaningful **queries with JOIN and subqueries**, and applying **normalization without losing essential relationships**. Despite these challenges, the system remains functional and aligned with the core objectives of the project.

3.3 Potential Future Work:

Future improvements to the hospital database system could include expanding the data model to cover **additional real-world functionalities** such as **billing, prescription tracking, staff assignments, and patient medical histories**. These additions would provide a more comprehensive view of hospital operations.

To enhance usability, the system could be integrated with a **graphical user interface (GUI)** or migrated into a **web-based application**, allowing healthcare staff to interact with the database more intuitively.

Furthermore, implementing **data validation rules**, **user authentication**, and **role-based access control** would improve data security and integrity. The system could also be optimized to handle **larger datasets** efficiently, supporting the needs of hospitals with high patient volumes and complex medical records.