CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF COMPUTING

CASE STUDY REPORT ON Hospital management system

Program Name: BCA

Subject Name/Code: Database Management System (23CAT-251)

Submitted by:	**Submitted to:**
Name: Ishika	**Name: Arvinder Singh**
UID: 23Bca10323	**Designation: Professor**
Section: 4(b)	

ABSTRACT

- **Introduction:**

In today's fast-paced healthcare environment, efficient and reliable management of hospital data is essential to ensure high-quality patient care and streamlined operations. Manual record-keeping systems are not only time-consuming but also prone to errors, delays, and loss of crucial information. To overcome these limitations, the use of database-driven hospital management systems has become increasingly critical.

The **Hospital Management System (HMS)** presented in this project is a comprehensive relational database model developed using **MySQL**. It is designed to manage and organize various aspects of a hospital's operations, including **patient records**, **doctor information**, **appointments**, **medical histories**, **diagnoses**, **prescriptions**, and **work schedules**.

The primary objective of the HMS is to serve as a **centralized data hub** for hospitals that allows healthcare professionals to access patient and operational information efficiently and securely. By automating administrative and clinical processes, the system minimizes the need for paperwork, reduces the chance of human error, and improves service delivery.

Key highlights of the HMS include:

Centralized Patient Management: Storing and retrieving complete patient details like demographics, medical history, and visit records.

Efficient Scheduling: Managing appointment slots, doctor schedules, and breaks in an organized and conflict-free manner.

Medical History Tracking: Maintaining accurate and detailed medical history for every patient, accessible by authorized doctors.

Doctor-Patient Interaction Records: Keeping track of diagnoses and prescriptions issued by doctors during appointments.

Secure & Scalable Structure: Built with data integrity constraints and relational principles, the system can be expanded to include billing, insurance, lab results, and more.

This database project not only helps simulate real-world hospital operations but also teaches core concepts of **relational database design**, **SQL programming**, and **data integrity enforcement**. It is ideal for students, developers, and administrators who wish to understand or implement a hospital-based data management system.

By adopting this system, hospitals can significantly improve data management efficiency, ensure better coordination among departments, and ultimately enhance the overall patient experience.

- ## Technique:

The Hospital Management System (HMS) database is built using **relational database design principles**, ensuring data consistency, ease of maintenance, and efficient data retrieval. Below are the core techniques and best practices implemented:

1. Relational Data Modeling

Each real-world entity (e.g., Patient, Doctor, Appointment) is represented as a separate table.

Primary keys are defined for each table to uniquely identify each record.

Foreign keys establish **referential integrity** between related tables.

2. Normalization

The database is normalized up to **3rd Normal Form (3NF)** to eliminate redundancy and maintain data integrity.

Ensures that each piece of data is stored only once, and all dependencies are logical and meaningful.

3. Use of Constraints

NOT NULL, PRIMARY KEY, and FOREIGN KEY constraints ensure valid data input.

ON DELETE CASCADE ensures that dependent records are automatically cleaned when a parent record is deleted.

4. Advanced SQL Techniques

Use of **JOINs** to combine multiple tables and retrieve meaningful data.

Aggregate functions (COUNT, AVG, etc.) for analytics and summaries.

Use of **GROUP BY**, **ORDER BY**, and **DISTINCT** clauses to organize data.

Application of **subqueries** and **nested SELECTs** for more complex information retrieval.

5. Scalability

The database structure is scalable, allowing future extension (e.g., billing, test reports, room assignments).

Each table is modular and can integrate seamlessly with additional features.

6. Security Best Practices

Passwords are stored as plain text in this example for simplicity, but in real-world scenarios, they should be encrypted using hashing algorithms.

- **System Configuration:**

- **Database Technology:** MySQL
- **Language:** SQL
- **Environment:** Compatible with any MySQL Client
- **Hardware:** Supports standard database server configurations.

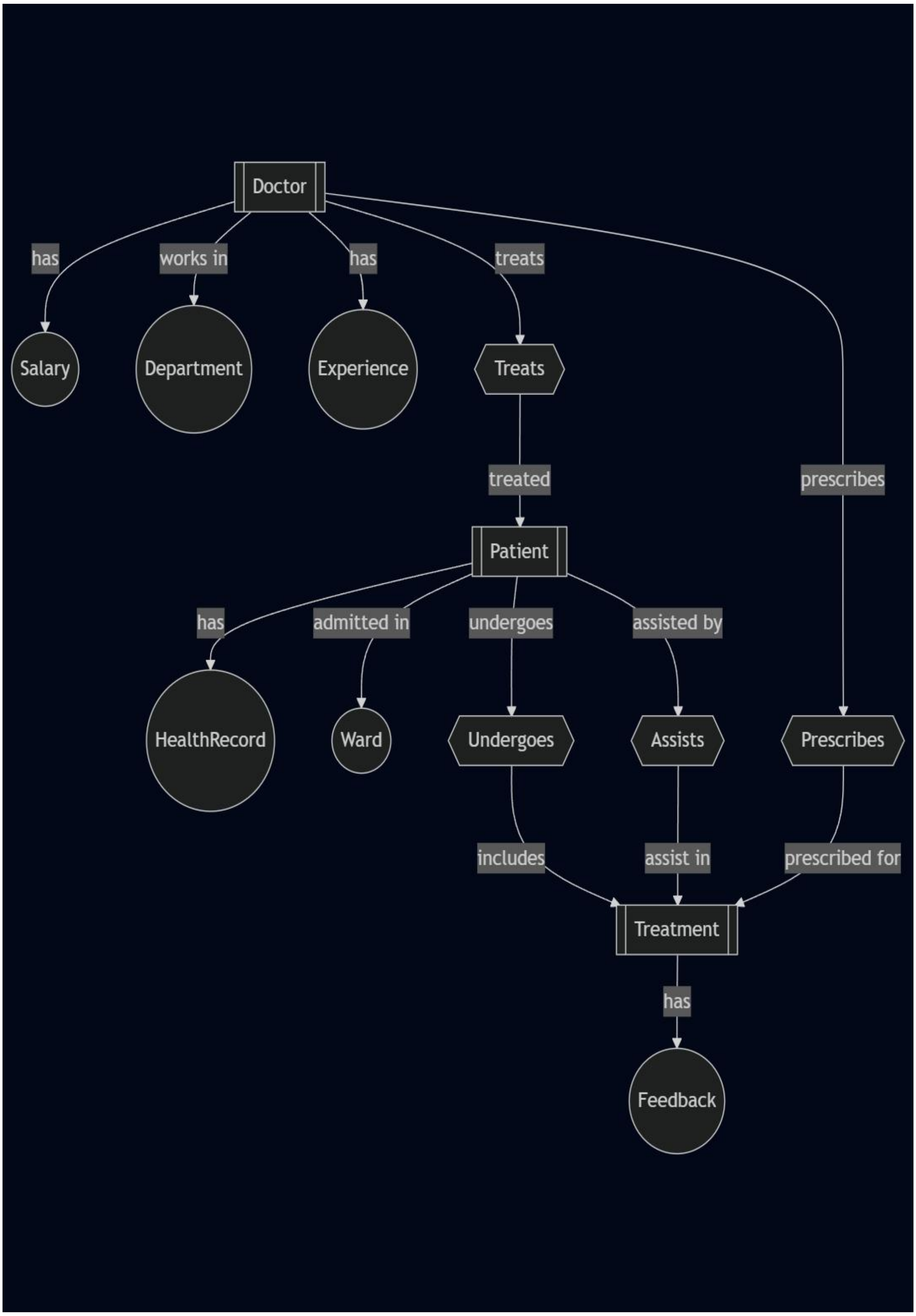
- **INPUT:**

```
CREATE DATABASE HMS;
USE HMS;

CREATE TABLE Patient(
    email VARCHAR(50) PRIMARY KEY,
    password VARCHAR(30) NOT NULL,
    name VARCHAR(50) NOT NULL,
    address VARCHAR(60) NOT NULL,
    gender VARCHAR(20) NOT NULL
);

CREATE TABLE MedicalHistory(
    id INT PRIMARY KEY,
    date DATE NOT NULL,
    conditions VARCHAR(100) NOT NULL,
    surgeries VARCHAR(100) NOT NULL,
    medication VARCHAR(100) NOT NULL
);

CREATE TABLE Doctor(
    email VARCHAR(50) PRIMARY KEY,
    gender VARCHAR(20) NOT NULL,
    password VARCHAR(30) NOT NULL,
    name VARCHAR(50) NOT NULL
);
```

```
CREATE TABLE Appointment(
  id INT PRIMARY KEY,
  date DATE NOT NULL,
  starttime TIME NOT NULL,
  endtime TIME NOT NULL,
  status VARCHAR(15) NOT NULL
);

CREATE TABLE PatientsAttendAppointments(
  patient VARCHAR(50) NOT NULL,
  appt INT NOT NULL,
  concerns VARCHAR(40) NOT NULL,
  symptoms VARCHAR(40) NOT NULL,
  FOREIGN KEY (patient) REFERENCES Patient (email) ON DELETE CASCADE,
  FOREIGN KEY (appt) REFERENCES Appointment (id) ON DELETE CASCADE,
  PRIMARY KEY (patient, appt)
);

CREATE TABLE Schedule(
  id INT NOT NULL,
  starttime TIME NOT NULL,
  endtime TIME NOT NULL,
  breaktime TIME NOT NULL,
  day VARCHAR(20) NOT NULL,
  PRIMARY KEY (id, starttime, endtime, breaktime, day)
);

CREATE TABLE PatientsFillHistory(
  patient VARCHAR(50) NOT NULL,
  history INT NOT NULL,
  FOREIGN KEY (patient) REFERENCES Patient (email) ON DELETE CASCADE,
  FOREIGN KEY (history) REFERENCES MedicalHistory (id) ON DELETE CASCADE,
  PRIMARY KEY (history)
);

CREATE TABLE Diagnose(
  appt INT NOT NULL,
  doctor VARCHAR(50) NOT NULL,
  diagnosis VARCHAR(40) NOT NULL,
  prescription VARCHAR(50) NOT NULL,
  FOREIGN KEY (appt) REFERENCES Appointment (id) ON DELETE CASCADE,
  FOREIGN KEY (doctor) REFERENCES Doctor (email) ON DELETE CASCADE,
  PRIMARY KEY (appt, doctor)
);

CREATE TABLE DocsHaveSchedules(
  sched INT NOT NULL,
  doctor VARCHAR(50) NOT NULL,
    FOREIGN KEY (sched) REFERENCES Schedule (id) ON DELETE CASCADE,
    FOREIGN KEY (doctor) REFERENCES Doctor (email) ON DELETE CASCADE,
    PRIMARY KEY (sched, doctor)
);

CREATE TABLE DoctorViewsHistory(
    history INT NOT NULL,
    doctor VARCHAR(50) NOT NULL,
    FOREIGN KEY (doctor) REFERENCES Doctor (email) ON DELETE CASCADE,
    FOREIGN KEY (history) REFERENCES MedicalHistory (id) ON DELETE CASCADE,
    PRIMARY KEY (history, doctor)
);

INSERT INTO Patient(email, password, name, address, gender)
VALUES
    ('john_doe@gmail.com', 'newpassword123', 'John Doe', 'California', 'male'),
    ('jane_smith@gmail.com', 'password321', 'Jane Smith', 'New York', 'female'),
    ('mike_jones@gmail.com', 'mikepass2025', 'Mike Jones', 'Texas', 'male'),
    ('alice_smith@gmail.com', 'securepass123', 'Alice Smith', 'California', 'female');

INSERT INTO MedicalHistory(id, date, conditions, surgeries, medication)
VALUES
    (1, '2025-01-14', 'High Fever', 'Knee Surgery', 'Paracetamol'),
    (2, '2025-02-10', 'Frequent Headaches', 'None', 'Aspirin'),
    (3, '2025-03-12', 'Back Pain', 'Spinal Surgery', 'Ibuprofen');

INSERT INTO Doctor(email, gender, password, name)
VALUES
    ('dr_athalye@gmail.com', 'male', 'doctorpass123', 'Dr. Hrishikesh Athalye'),
    ('dr_morgan@gmail.com', 'female', 'docmorgan2025', 'Dr. Emily Morgan');

INSERT INTO Appointment(id, date, starttime, endtime, status)
VALUES
    (1, '2025-04-15', '09:00', '10:00', 'Scheduled'),
    (2, '2025-04-16', '10:00', '11:00', 'Scheduled'),
    (3, '2025-04-17', '14:00', '15:00', 'Scheduled');

INSERT INTO PatientsAttendAppointments(patient, appt, concerns, symptoms)
VALUES
    ('john_doe@gmail.com', 1, 'Coughing', 'Sore Throat'),
    ('jane_smith@gmail.com', 2, 'Dizziness', 'Headache'),
    ('mike_jones@gmail.com', 3, 'Back Pain', 'Stiffness');

INSERT INTO Schedule(id, starttime, endtime, breaktime, day)
VALUES
    (1, '09:00', '17:00', '12:00', 'Monday'),
    (2, '09:00', '17:00', '12:00', 'Wednesday'),
    (3, '09:00', '17:00', '12:00', 'Friday');
```

```
INSERT INTO PatientsFillHistory(patient, history)
VALUES
    ('john_doe@gmail.com', 1),
    ('jane_smith@gmail.com', 2),
    ('mike_jones@gmail.com', 3);

INSERT INTO Diagnose(appt, doctor, diagnosis, prescription)
VALUES
    (1, 'dr_athalye@gmail.com', 'Cold', 'Rest and Drink Fluids'),
    (2, 'dr_morgan@gmail.com', 'Migraine', 'Painkillers and Rest'),
    (3, 'dr_morgan@gmail.com', 'Sciatica', 'Physical Therapy');

INSERT INTO DocsHaveSchedules(sched, doctor)
VALUES
    (1, 'dr_athalye@gmail.com'),
    (2, 'dr_morgan@gmail.com');

INSERT INTO DoctorViewsHistory(history, doctor)
VALUES
    (1, 'dr_athalye@gmail.com'),
    (2, 'dr_morgan@gmail.com'),
    (3, 'dr_morgan@gmail.com');
```

- **ER DIAGRAM:**


Doctor
has
works in
has
treats
Salary
Department
Experience
Treats
treated
prescribes
Patient
has
admitted in
undergoes
assisted by
HealthRecord
Ward
Undergoes
Assists
Prescribes
includes
assist in
prescribed for
Treatment
has
Feedback

- **TABLE REALTION:**

Table 1	Table 2	Relationship Type
Patient	PatientsAttendAppointments	One-to-Many (patient can attend multiple appointments)
Appointment	PatientsAttendAppointments	One-to-Many (appointment linked to many patients)
Patient	PatientsFillHistory	One-to-One (each history belongs to one patient)
MedicalHistory	PatientsFillHistory	One-to-One (each history ID is unique)
Doctor	Diagnose	One-to-Many (a doctor can diagnose multiple appointments)
Appointment	Diagnose	One-to-One (each appointment has one diagnosis)
Schedule	DocsHaveSchedules	One-to-Many (one schedule can belong to many doctors)
Doctor	DocsHaveSchedules	One-to-Many (doctor can have multiple schedule entries)
Doctor	DoctorViewsHistory	Many-to-Many (doctor can view multiple histories)
MedicalHistory	DoctorViewsHistory	Many-to-Many (each history can be viewed by multiple doctors)

- **TABULAR FORMAT:**

Patient

email (PK)

password

name

address

gender

MedicalHistory

id (PK)

date

conditions

surgeries

medication

Doctor

email (PK)

gender

password

name

Appointment

id (PK)

date

starttime

endtime

status

PatientsAttendAppointments

patient (FK to Patient.email)

appt (FK to Appointment.id)

concerns

symptoms

Schedule

id

starttime

endtime

breaktime

day

PatientsFillHistory

patient (FK to Patient.email)

history (FK to MedicalHistory.id)

Diagnose

appt (FK to Appointment.id)

doctor (FK to Doctor.email)

diagnosis

prescription

DocsHaveSchedules

sched (FK to Schedule.id)

doctor (FK to Doctor.email)

DoctorViewsHistory

history (FK to MedicalHistory.id)

doctor (FK to Doctor.email)

- **TABLE CREATION:**

```
  -- Create the Hospital Management System Database
CREATE DATABASE HMS;
USE HMS;

-- Create the Patient Table
CREATE TABLE Patient(
   email VARCHAR(50) PRIMARY KEY,
   password VARCHAR(30) NOT NULL,
   name VARCHAR(50) NOT NULL,
   address VARCHAR(60) NOT NULL,
   gender VARCHAR(20) NOT NULL
);

-- Create the MedicalHistory Table
CREATE TABLE MedicalHistory(
   id INT PRIMARY KEY,
   date DATE NOT NULL,
   conditions VARCHAR(100) NOT NULL,
   surgeries VARCHAR(100) NOT NULL,
   medication VARCHAR(100) NOT NULL
);

-- Create the Doctor Table
CREATE TABLE Doctor(
```

```sql
    email VARCHAR(50) PRIMARY KEY,
    gender VARCHAR(20) NOT NULL,
    password VARCHAR(30) NOT NULL,
    name VARCHAR(50) NOT NULL
);

-- Create the Appointment Table
CREATE TABLE Appointment(
    id INT PRIMARY KEY,
    date DATE NOT NULL,
    starttime TIME NOT NULL,
    endtime TIME NOT NULL,
    status VARCHAR(15) NOT NULL
);

-- Create the PatientsAttendAppointments Table
CREATE TABLE PatientsAttendAppointments(
    patient VARCHAR(50) NOT NULL,
    appt INT NOT NULL,
    concerns VARCHAR(40) NOT NULL,
    symptoms VARCHAR(40) NOT NULL,
    FOREIGN KEY (patient) REFERENCES Patient(email) ON DELETE CASCADE,
    FOREIGN KEY (appt) REFERENCES Appointment(id) ON DELETE CASCADE,
    PRIMARY KEY (patient, appt)
);

-- Create the Schedule Table
CREATE TABLE Schedule(
    id INT NOT NULL,
    starttime TIME NOT NULL,
    endtime TIME NOT NULL,
    breaktime TIME NOT NULL,
    day VARCHAR(20) NOT NULL,
    PRIMARY KEY (id, starttime, endtime, breaktime, day)
);

-- Create the PatientsFillHistory Table
CREATE TABLE PatientsFillHistory(
    patient VARCHAR(50) NOT NULL,
    history INT NOT NULL,
    FOREIGN KEY (patient) REFERENCES Patient(email) ON DELETE CASCADE,
    FOREIGN KEY (history) REFERENCES MedicalHistory(id) ON DELETE CASCADE,
    PRIMARY KEY (history)
);

-- Create the Diagnose Table
CREATE TABLE Diagnose(
    appt INT NOT NULL,
    doctor VARCHAR(50) NOT NULL,
    diagnosis VARCHAR(40) NOT NULL,
    prescription VARCHAR(50) NOT NULL,
    FOREIGN KEY (appt) REFERENCES Appointment(id) ON DELETE CASCADE,
    FOREIGN KEY (doctor) REFERENCES Doctor(email) ON DELETE CASCADE,
    PRIMARY KEY (appt, doctor)
);

-- Create the DocsHaveSchedules Table
CREATE TABLE DocsHaveSchedules(
    sched INT NOT NULL,
    doctor VARCHAR(50) NOT NULL,
    FOREIGN KEY (sched) REFERENCES Schedule(id) ON DELETE
CASCADE,
    FOREIGN KEY (doctor) REFERENCES Doctor(email) ON DELETE
CASCADE,
    PRIMARY KEY (sched, doctor)
);

-- Create the DoctorViewsHistory Table
CREATE TABLE DoctorViewsHistory(
    history INT NOT NULL,
    doctor VARCHAR(50) NOT NULL,
    FOREIGN KEY (doctor) REFERENCES Doctor(email) ON DELETE
CASCADE,
    FOREIGN KEY (history) REFERENCES MedicalHistory(id) ON
DELETE CASCADE,
    PRIMARY KEY (history, doctor)
);
```

- **SQL QUERIES WITH OUTPUT (at least 10 to 15):**

_1. Count number of appointments per doctor

```
SELECT doctor, COUNT(appt) AS appointment_countFROM
DiagnoseGROUP BY doctor;
```

2. Number of patients treated by each doctor

```
SELECT doctor, COUNT(DISTINCT appt) AS patient_countFROM
DiagnoseGROUP BY doctor;
```

3. Average appointment duration (in minutes)

```
SELECT AVG(TIMESTAMPDIFF(MINUTE, starttime, endtime)) AS avg_durationFROM Appointment;
```

4. All appointments for a specific patient

```
SELECT p.name, a.date, a.starttime, a.endtimeFROM Patient pJOIN PatientsAttendAppointments paa ON p.email = paa.patientJOIN Appointment a ON paa.appt = a.idWHERE p.email = 'alice_smith@gmail.com';
```

5. All schedules for a specific doctor

```
SELECT d.name, s.day, s.starttime, s.endtimeFROM Doctor dJOIN DocsHaveSchedules dhs ON d.email = dhs.doctorJOIN Schedule s ON dhs.sched = s.idWHERE d.email = 'dr_morgan@gmail.com';
```

6. Patients and their diagnosed conditions

```
SELECT p.name, diag.diagnosisFROM Patient pJOIN PatientsAttendAppointments paa ON p.email = paa.patientJOIN Diagnose diag ON paa.appt = diag.appt;
```

7. Insert a new doctor

```
INSERT INTO Doctor(email, gender, password, name)VALUES ('dr_ram@gmail.com', 'male', 'ramdoc2025', 'Dr. Ram Sharma');
```

8. Update an appointment's status

```
UPDATE AppointmentSET status = 'Completed'WHERE id = 2;
```

9. List appointments with diagnosis and prescription

```
SELECT a.date, d.name AS doctor_name, diag.diagnosis, diag.prescriptionFROM Appointment aJOIN Diagnose diag ON a.id = diag.apptJOIN Doctor d ON diag.doctor = d.email;
```

10. List doctors who viewed a specific patient history

```
SELECT d.nameFROM Doctor dJOIN DoctorViewsHistory dvh ON d.email = dvh.doctorWHERE dvh.history = 1;
```

11. Find all patients diagnosed with 'Cold'

```
SELECT p.nameFROM Patient pJOIN PatientsAttendAppointments paa ON p.email = paa.patientJOIN Diagnose diag ON paa.appt = diag.apptWHERE diag.diagnosis = 'Cold';
```

12. Show the schedule of all doctors on Friday

```
SELECT d.name, s.starttime, s.endtimeFROM Doctor dJOIN DocsHaveSchedules dhs ON d.email = dhs.doctorJOIN Schedule s ON dhs.sched = s.idWHERE s.day = 'Friday';
```

13. Count of male vs female patients

```
SELECT gender, COUNT(*) AS totalFROM PatientGROUP BY gender;
```

14. Doctors who have never diagnosed any patient

```
SELECT nameFROM DoctorWHERE email NOT IN (SELECT DISTINCT doctor FROM Diagnose);
```

15. Patients with more than 1 appointment

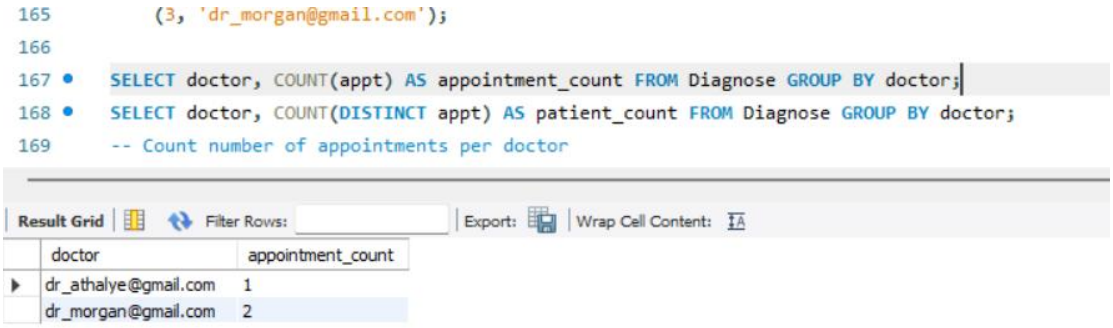
sql

CopyEdit

```
SELECT patient, COUNT(*) AS total_appointmentsFROM PatientsAttendAppointmentsGROUP BY patientHAVING COUNT(*) > 1;
```

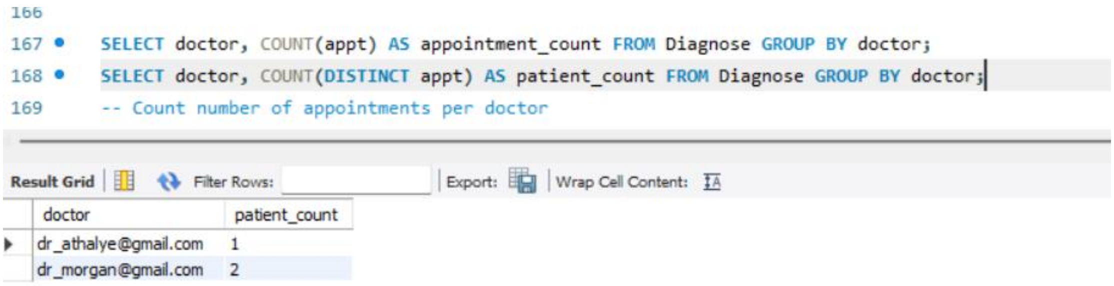
16. List all diagnoses made on '2025-04-15'

```
SELECT d.name AS doctor, diag.diagnosis, a.dateFROM Diagnose diagJOIN Doctor d ON diag.doctor = d.emailJOIN Appointment a ON diag.appt = a.idWHERE a.date = '2025-04-15';
```



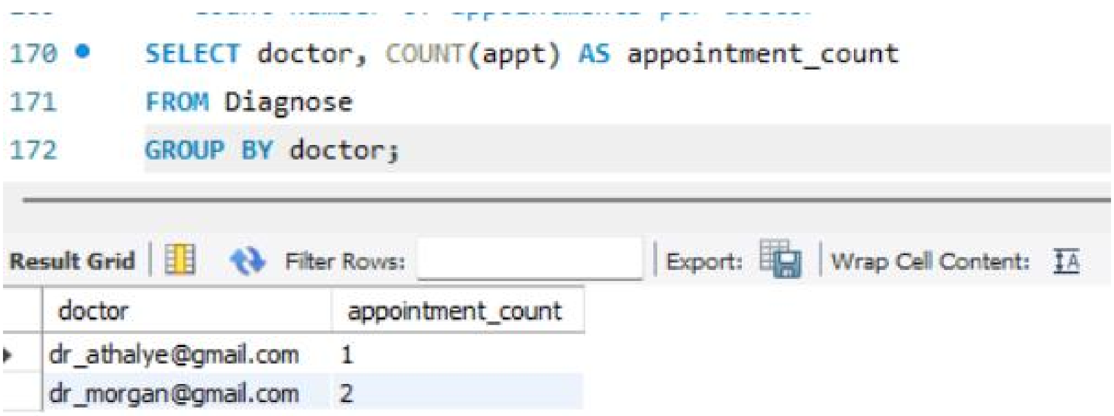
```
        (3, 'dr_morgan@gmail.com');

SELECT doctor, COUNT(appt) AS appointment_count FROM Diagnose GROUP BY doctor;
SELECT doctor, COUNT(DISTINCT appt) AS patient_count FROM Diagnose GROUP BY doctor;
-- Count number of appointments per doctor
```

doctor	appointment_count
dr_athalye@gmail.com	1
dr_morgan@gmail.com	2



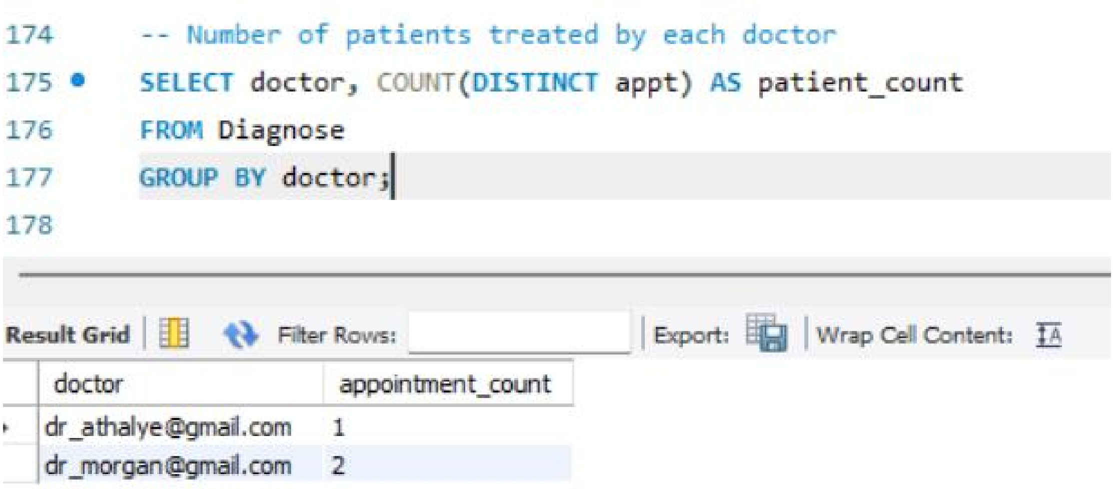
```

SELECT doctor, COUNT(appt) AS appointment_count FROM Diagnose GROUP BY doctor;
SELECT doctor, COUNT(DISTINCT appt) AS patient_count FROM Diagnose GROUP BY doctor;
-- Count number of appointments per doctor
```

doctor	patient_count
dr_athalye@gmail.com	1
dr_morgan@gmail.com	2



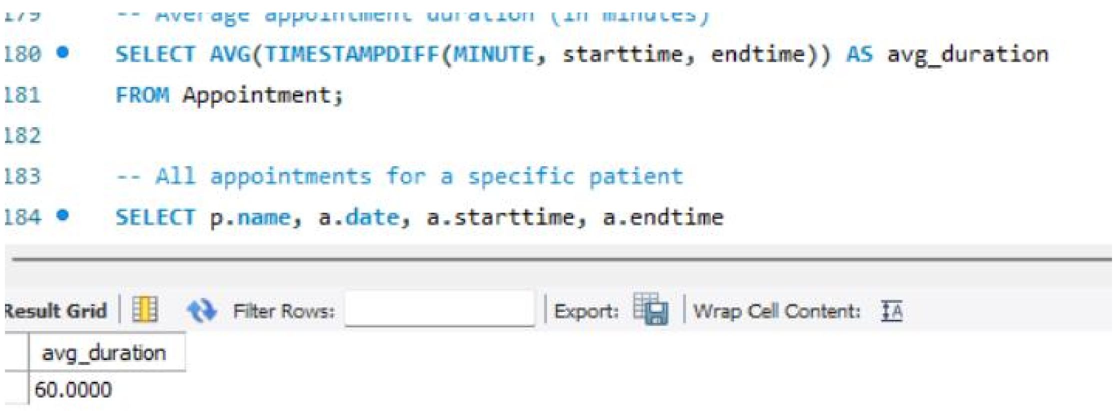
```
-- Count number of appointments per doctor
SELECT doctor, COUNT(appt) AS appointment_count
FROM Diagnose
GROUP BY doctor;
```

doctor	appointment_count
dr_athalye@gmail.com	1
dr_morgan@gmail.com	2



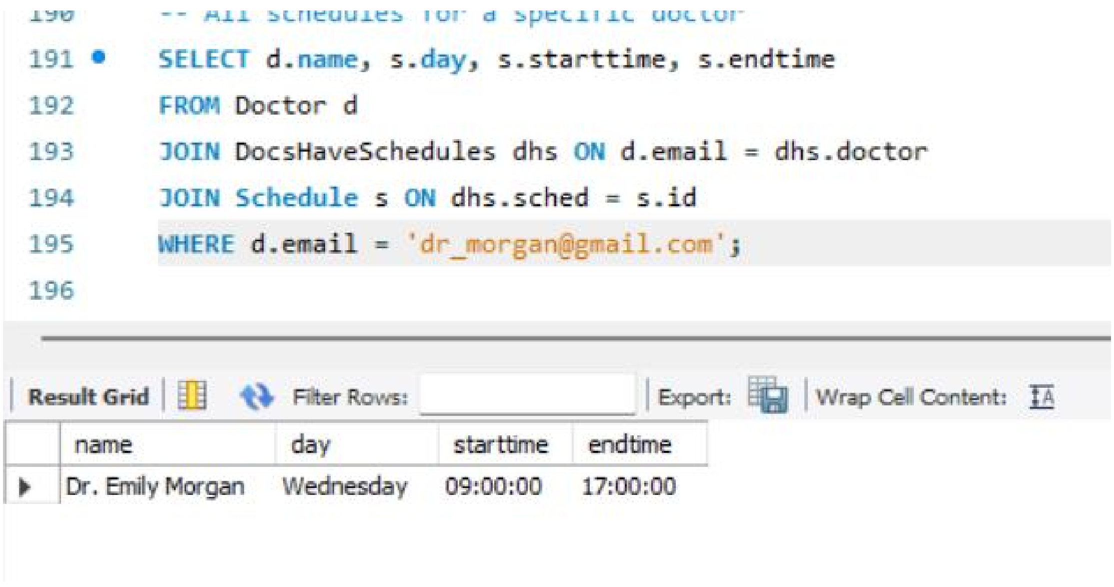
```sql
-- Number of patients treated by each doctor
SELECT doctor, COUNT(DISTINCT appt) AS patient_count
FROM Diagnose
GROUP BY doctor;
```

doctor	appointment_count
dr_athalye@gmail.com	1
dr_morgan@gmail.com	2



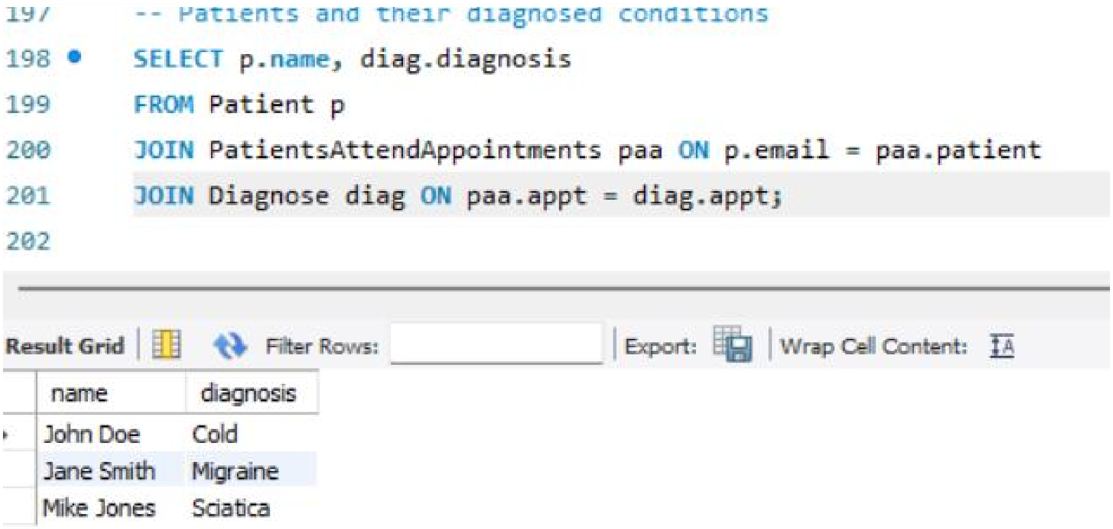
```sql
-- Average appointment duration (in minutes)
SELECT AVG(TIMESTAMPDIFF(MINUTE, starttime, endtime)) AS avg_duration
FROM Appointment;

-- All appointments for a specific patient
SELECT p.name, a.date, a.starttime, a.endtime
```

avg_duration
60.0000



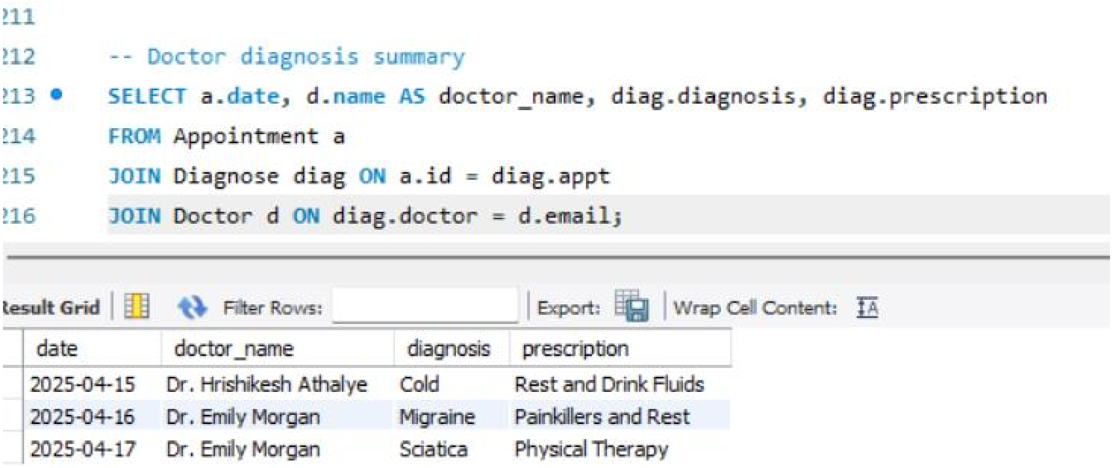
```sql
-- All schedules for a specific doctor
SELECT d.name, s.day, s.starttime, s.endtime
FROM Doctor d
JOIN DocsHaveSchedules dhs ON d.email = dhs.doctor
JOIN Schedule s ON dhs.sched = s.id
WHERE d.email = 'dr_morgan@gmail.com';
```

name	day	starttime	endtime
Dr. Emily Morgan	Wednesday	09:00:00	17:00:00



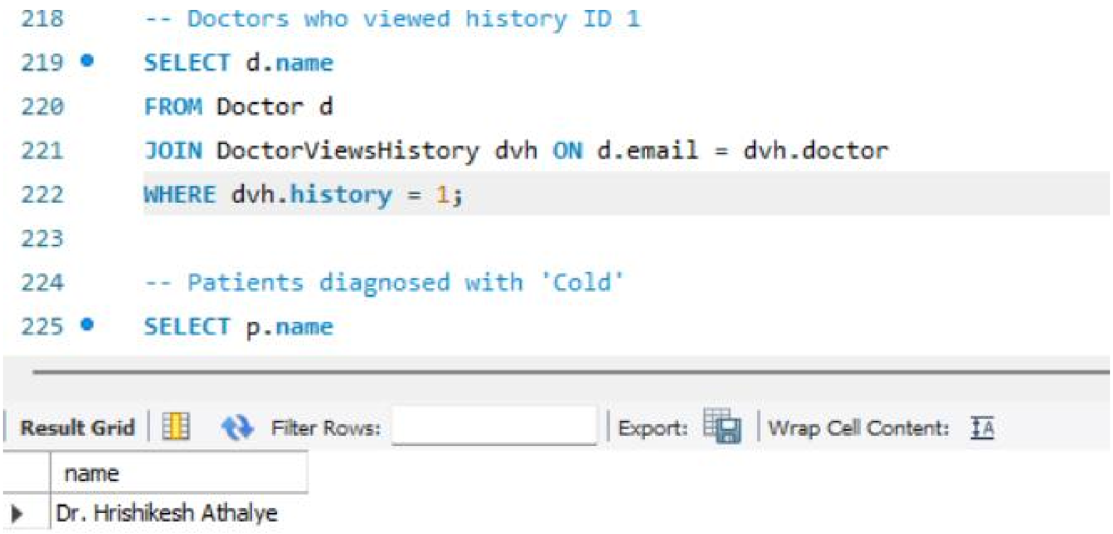
```sql
-- Patients and their diagnosed conditions
SELECT p.name, diag.diagnosis
FROM Patient p
JOIN PatientsAttendAppointments paa ON p.email = paa.patient
JOIN Diagnose diag ON paa.appt = diag.appt;
```

name	diagnosis
John Doe	Cold
Jane Smith	Migraine
Mike Jones	Sciatica



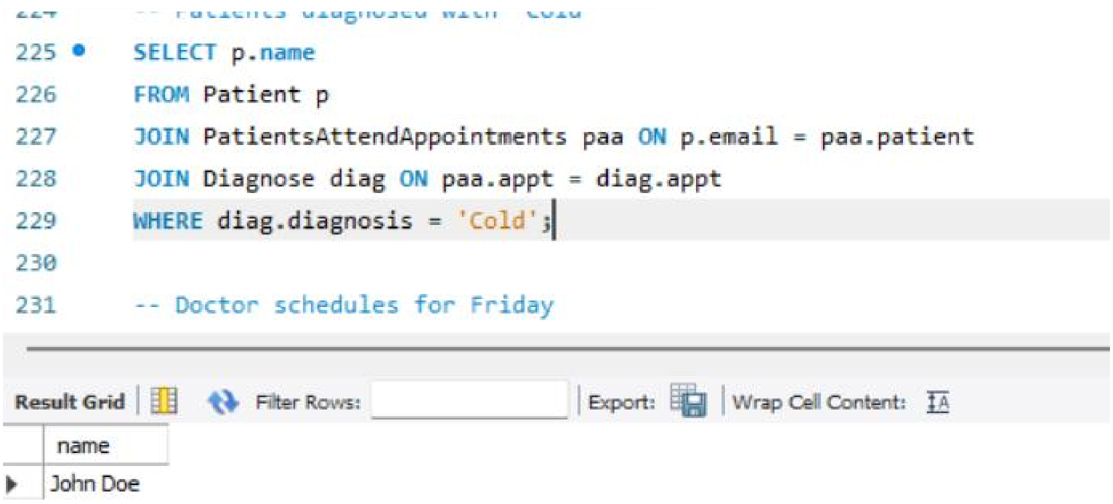
```sql
-- Doctor diagnosis summary
SELECT a.date, d.name AS doctor_name, diag.diagnosis, diag.prescription
FROM Appointment a
JOIN Diagnose diag ON a.id = diag.appt
JOIN Doctor d ON diag.doctor = d.email;
```

date	doctor_name	diagnosis	prescription
2025-04-15	Dr. Hrishikesh Athalye	Cold	Rest and Drink Fluids
2025-04-16	Dr. Emily Morgan	Migraine	Painkillers and Rest
2025-04-17	Dr. Emily Morgan	Sciatica	Physical Therapy



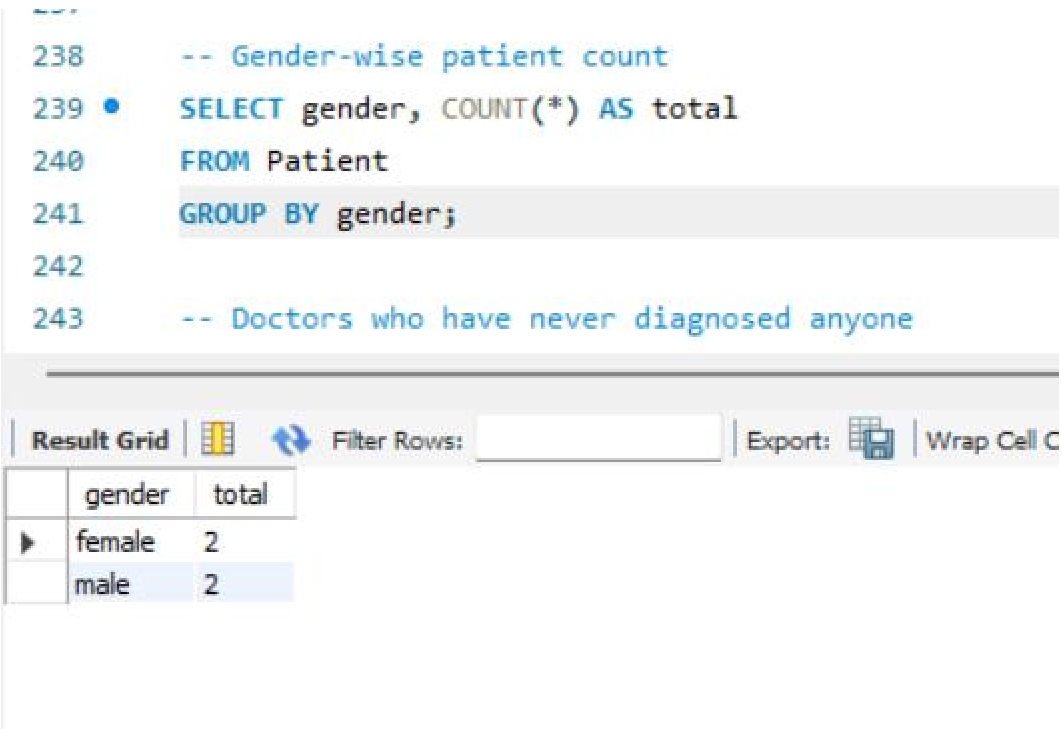
```sql
-- Doctors who viewed history ID 1
SELECT d.name
FROM Doctor d
JOIN DoctorViewsHistory dvh ON d.email = dvh.doctor
WHERE dvh.history = 1;

-- Patients diagnosed with 'Cold'
SELECT p.name
```

name
Dr. Hrishikesh Athalye



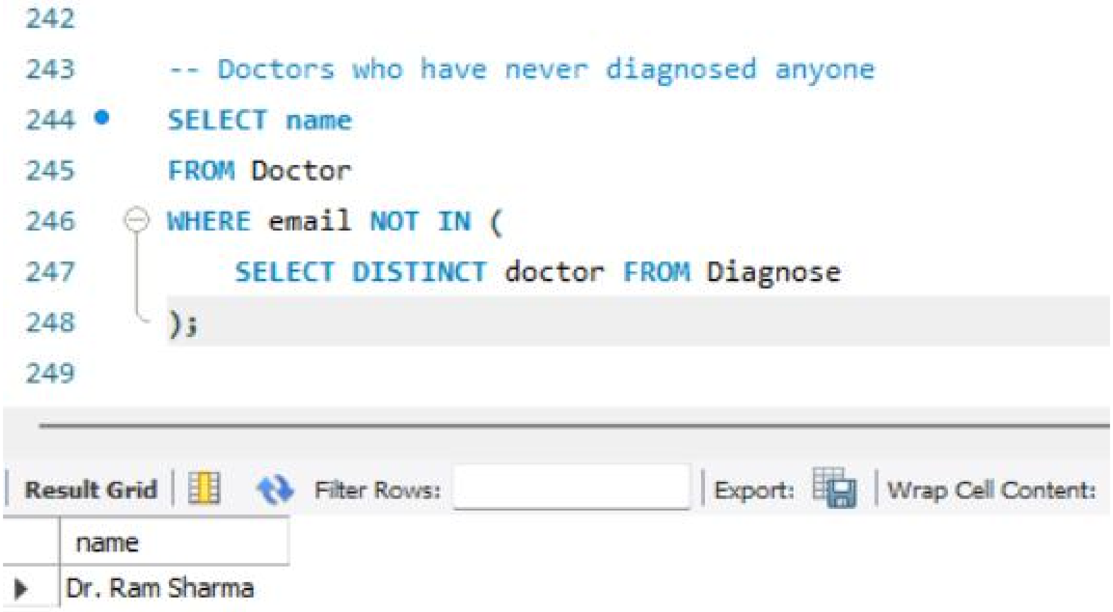
```sql
-- Patients diagnosed with 'Cold'
SELECT p.name
FROM Patient p
JOIN PatientsAttendAppointments paa ON p.email = paa.patient
JOIN Diagnose diag ON paa.appt = diag.appt
WHERE diag.diagnosis = 'Cold';

-- Doctor schedules for Friday
```

name
John Doe



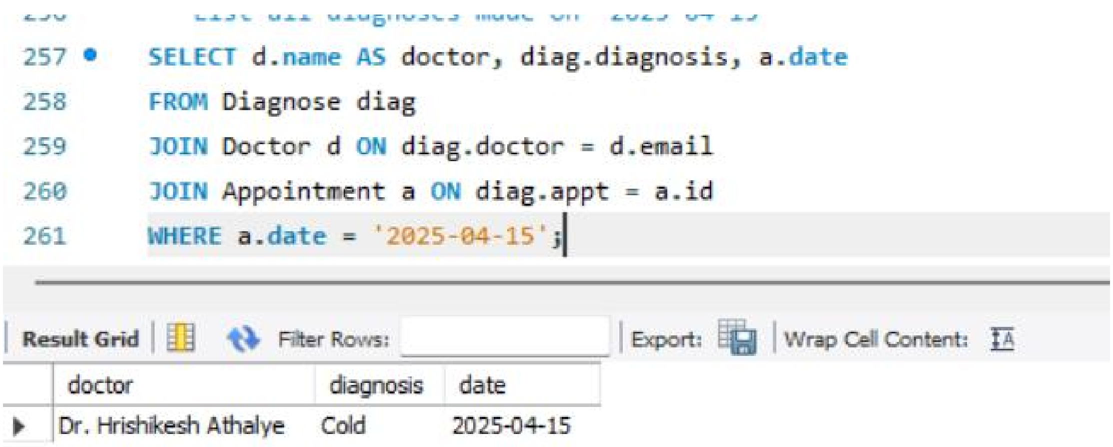
```sql
-- Gender-wise patient count
SELECT gender, COUNT(*) AS total
FROM Patient
GROUP BY gender;

-- Doctors who have never diagnosed anyone
```

gender	total
female	2
male	2



```sql
-- Doctors who have never diagnosed anyone
SELECT name
FROM Doctor
WHERE email NOT IN (
    SELECT DISTINCT doctor FROM Diagnose
);
```

```
name
Dr. Ram Sharma
```



```sql
-- List all diagnoses made on '2025-04-15'
SELECT d.name AS doctor, diag.diagnosis, a.date
FROM Diagnose diag
JOIN Doctor d ON diag.doctor = d.email
JOIN Appointment a ON diag.appt = a.id
WHERE a.date = '2025-04-15';
```

```
doctor                  diagnosis   date
Dr. Hrishikesh Athalye  Cold        2025-04-15
```

- **SUMMARY:**

The Hospital Management System (HMS) project is a comprehensive database system that models the operations and workflows of a hospital. It handles vital components such as patient management, doctor assignments, appointments, medical histories, schedules, and diagnoses.

The database design encapsulates various real-world relationships, such as a doctor treating many patients, or a patient attending multiple appointments. It follows the relational model and enforces strict data integrity via constraints and normalized design.

A key highlight of the HMS is its query capability. From listing a doctor's upcoming appointments to analyzing treatment frequency or average appointment duration, the system provides insightful reporting through SQL. This enhances administrative efficiency and ensures timely and accurate medical services.

Moreover, the HMS database is designed to be scalable and extensible. Modules like billing systems, lab tests, hospital rooms, and emergency alerts can be easily integrated. The use of MySQL as a backend ensures compatibility with a wide range of application front-ends, making it suitable for web and desktop-based hospital management solutions.

Overall, the HMS ensures that patient care is supported with clean, accessible, and actionable data for both clinical and operational staff.

- **CONCLUSION:**

The Hospital Management System (HMS) database offers a robust solution to the complexities of modern healthcare data management. It simplifies core hospital workflows such as patient registration, appointment tracking, doctor scheduling, diagnosis documentation, and historical medical record-keeping.

By applying relational database techniques, the system ensures data accuracy and consistency across all modules. The extensive use of foreign keys and normalization prevents duplication and supports relational integrity, which is crucial in a hospital where data sensitivity is high.

Furthermore, the system empowers medical staff with tools to quickly query and analyze data, leading to improved diagnosis and better treatment outcomes. It also reduces administrative burden, allowing hospitals to focus more on patient care.

In future expansions, the HMS can be upgraded with features such as:

Online patient portals

Billing and insurance processing

Real-time emergency alerts

Machine learning modules for predictive diagnosis

This project lays the groundwork for a full-fledged hospital management system, serving as both an academic model and a potential foundation for real-world applications.