Hospital Management System

Charagondla Tharun(23CSB0B25)

July 11th 2025

1.Introduction: What is Hospital Management System?

A **Hospital Management System (HMS)** is a comprehensive software solution that integrates all administrative and clinical operations within a healthcare facility. It provides a centralized platform to efficiently manage patient records, doctor schedules, appointments, medical histories, diagnoses, prescriptions, and billing information. By automating routine processes, the system enhances operational efficiency, reduces manual errors, and improves the overall quality of patient care.

HMS ensures secure and role-based access to sensitive patient data while supporting the workflows of various departments such as outpatient services, inpatient care, laboratory, and pharmacy. It also generates detailed reports that aid in decision-making, regulatory compliance, and resource planning. Through its structured relational database design, the Hospital Management System enables seamless coordination among healthcare professionals and facilitates timely, accurate information flow across the organization.

2. Problem Statement

We aim to develop a Database System for a Hospital Management Platform. The primary responsibilities of this system include:

1. **Storing and managing patient information**, including personal details, contact information, and medical histories.
2. **Maintaining doctor records**, their schedules, and specializations.
3. **Handling appointments**, including booking, tracking, and updating appointment statuses.
4. **Recording diagnoses and prescriptions** issued by doctors during consultations.
5. **Facilitating secure access** to patient data by authorized users while ensuring privacy and data protection.
6. **Generating reports and summaries** for management, compliance, and performance analysis.
7. **Supporting efficient coordination** among doctors, staff, and administrative departments to improve overall clinic operations.

3. Key Features of the Hospital Management System

Patient Information Management

Maintain comprehensive patient records including personal details, contact information, and gender.

Track each patient’s medical history (conditions, surgeries, medication).

Doctor Records and Schedules

Store doctor profiles with login credentials and personal information.

Manage doctors’ weekly schedules, working hours, and breaks.

Appointment Management

Create, update, and track patient appointments with status (e.g., Pending, Completed).

Link appointments to both patients and doctors for complete traceability.

Medical History Tracking

Record detailed medical histories for each patient.

Enable doctors to view patients’ past records to support informed treatment.

Diagnosis and Prescription Recording

Allow doctors to record diagnoses and prescribe treatments during appointments.

Associate each diagnosis with a specific appointment and doctor.

Patient-Doctor Interaction

Log patient concerns and symptoms for each appointment.

Support many-to-many relationships between patients and appointments.

Secure Data Access and Role Management

Ensure only authorized users (e.g., doctors) can view or modify sensitive data.

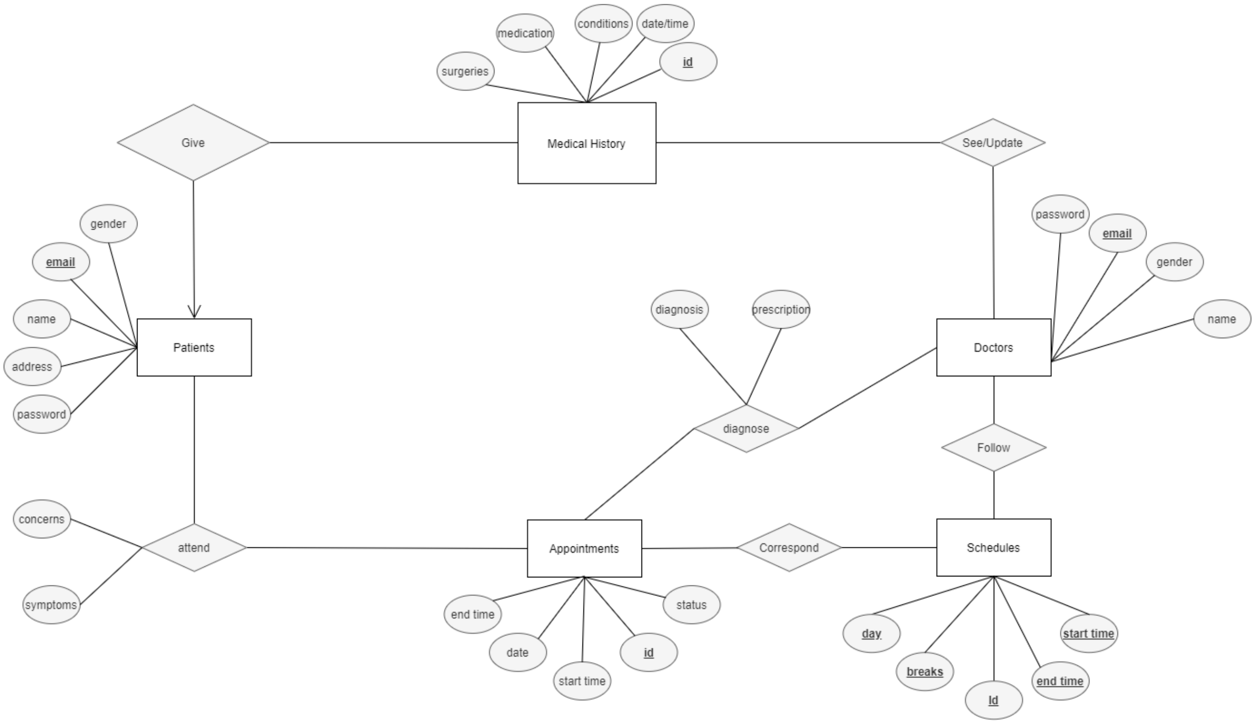
Enforce referential integrity through foreign key constraints.

Reporting and Analysis

Generate reports showing appointments, diagnoses, and medical histories.

Support administrative decision-making and compliance documentation.

4. Entity Relational Model



5. Entities

1. Patient

- **Attributes:**
  email, password, name, address, gender
- **Primary Key:**
  email

2. Doctor

- **Attributes:**
  email, password, name, gender

- **Primary Key:**
  email

3. MedicalHistory

- **Attributes:**
  id, date, conditions, surgeries, medication
- **Primary Key:**
  id

4. Appointment

- **Attributes:**
  id, date, start_time, end_time, status
- **Primary Key:**
  id

5. Schedule

- **Attributes:**
  id, day, start_time, end_time, breaks, doctor_email
- **Primary Key:**
  id

6. Relations

1.PatientsAttendAppointments

Between: Patient and Appointment
Cardinality: Many-to-Many
Why this relation is needed: To record which patients attended which appointments and to store the concerns and symptoms reported during each appointment.

2.Diagnose

Between: Appointment and Doctor
Cardinality: Many-to-Many
Why this relation is needed: To record the diagnoses and prescriptions that doctors provide for appointments.

3.PatientsFillHistory

Between: Patient and MedicalHistory
Cardinality: One-to-Many
Why this relation is needed: To associate patients with the medical history records they have created or contributed to.

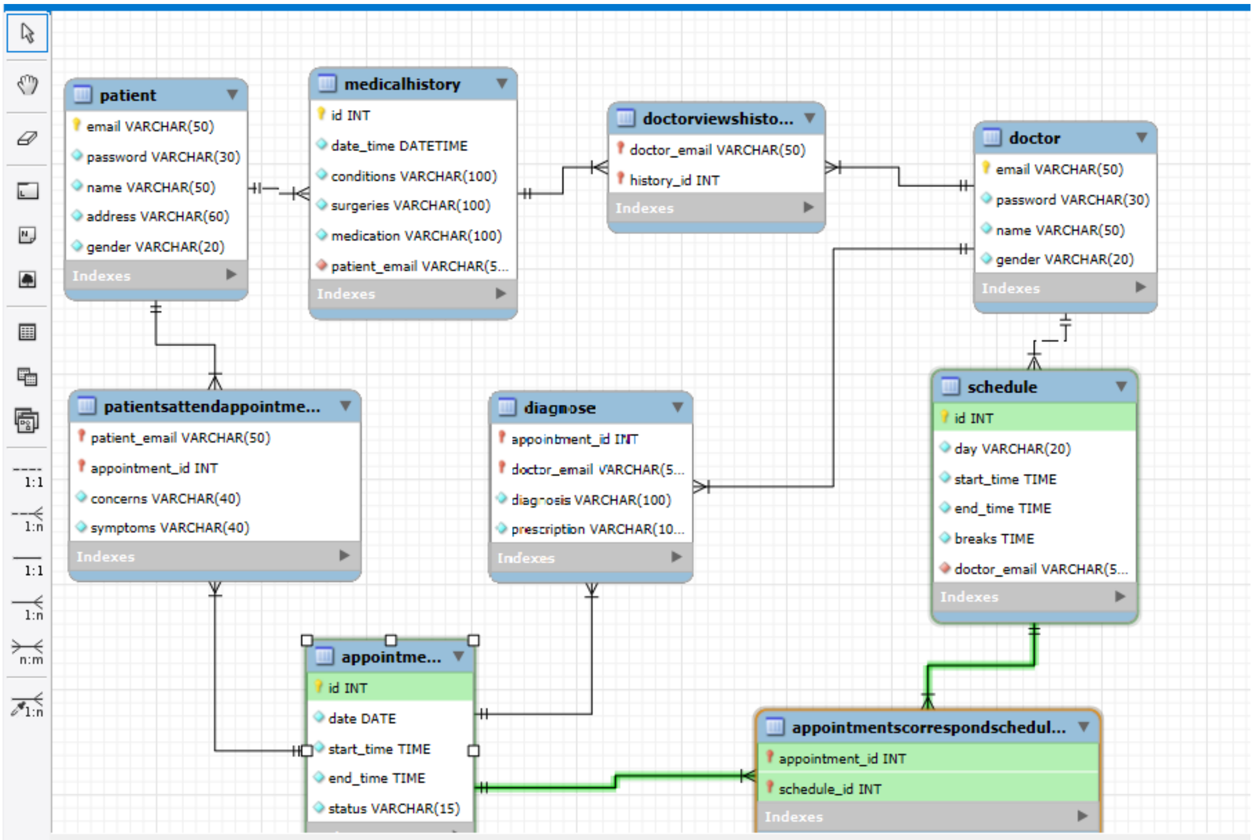
4.DoctorViewsHistory

Between: Doctor and MedicalHistory
Cardinality: Many-to-Many
Why this relation is needed: To track which doctors have viewed which medical histories for auditing and compliance purposes.

5.DocsHaveSchedules

Between: Doctor and Schedule
Cardinality: Many-to-Many
Why this relation is needed: To assign schedules to doctors, indicating when each doctor is available to work.

7. Relational Schema



8.Tables

1. Patient

Table Name: Patient

Primary Key:
email

Functional Dependencies:

- email → password, name, address, gender

Normal Form:
BCNF (email is candidate key)

2. Doctor

Table Name: Doctor

Primary Key:
email

Functional Dependencies:

- email → password, name, gender

Normal Form:
BCNF (email is candidate key)

3. MedicalHistory

Table Name: MedicalHistory

Primary Key:
id

Functional Dependencies:

- id → date_time, conditions, surgeries, medication, patient_email

Normal Form:
BCNF (id is candidate key)

4. Appointment

Table Name: Appointment

Primary Key:
id

Functional Dependencies:

- id → date, start_time, end_time, status

Normal Form:
BCNF (id is candidate key)

5. Schedule

Table Name: Schedule

Primary Key:
id

Functional Dependencies:

- id → day, start_time, end_time, breaks, doctor_email

Normal Form:
BCNF (id is candidate key)

6. DoctorViewsHistory

Table Name: DoctorViewsHistory

Primary Key:
Composite Key (doctor_email, history_id)

Functional Dependencies:

- (doctor_email, history_id) → (no additional attributes)

Normal Form:
BCNF ((doctor_email,history_id) is candidate key)

7. PatientsAttendAppointments

Table Name: PatientsAttendAppointments

Primary Key:
Composite Key (patient_email, appointment_id)

Functional Dependencies:

- (patient_email, appointment_id) → concerns, symptoms

Normal Form:
BCNF ((patient_email, appointment_id) is a candidate key)

8. Diagnose

Table Name: Diagnose

Primary Key:
Composite Key (appointment_id, doctor_email)

Functional Dependencies:

- (appointment_id, doctor_email) → diagnosis, prescription

Normal Form:
BCNF ((appointment_id, doctor_email) is a candidate key)

9. AppointmentsCorrespondSchedules

Table Name: AppointmentsCorrespondSchedules

Primary Key:
Composite Key (appointment_id, schedule_id)

Functional Dependencies:

(appointment_id, schedule_id) → (no additional attributes)

Normal Form:
BCNF ((appointment_id, schedule_id) is a candidate key)

9.Verification of BCNF for All Relations

All tables in this Hospital Management System database are designed to satisfy the properties of Boyce-Codd Normal Form (BCNF). Each relation was created such that every non-trivial functional dependency has a determinant that is a superkey. In the entity tables, the primary keys are single attributes, and all other attributes are fully functionally dependent on those keys without any transitive dependencies. In the associative tables with composite primary keys, all non-key attributes depend entirely on the combination of the key attributes, and there are no partial dependencies or dependencies between non-key attributes. As a result, the schema is highly normalized, and no further decomposition was necessary. This ensures data integrity, reduces redundancy, and maintains consistency across the system.

10. SQL Queries to Create Tables

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

CREATE TABLE MedicalHistory (
   id INT PRIMARY KEY,
   date_time DATETIME NOT NULL,
   conditions VARCHAR(100) NOT NULL,
   surgeries VARCHAR(100) NOT NULL,
   medication VARCHAR(100) NOT NULL,
   patient_email VARCHAR(50) NOT NULL,
   FOREIGN KEY (patient_email) REFERENCES Patient(email) ON DELETE CASCADE
);

CREATE TABLE Appointment (
   id INT PRIMARY KEY,
   date DATE NOT NULL,
   start_time TIME NOT NULL,
   end_time TIME NOT NULL,
   status VARCHAR(15) NOT NULL
);

CREATE TABLE Schedule (
   id INT PRIMARY KEY,
   day VARCHAR(20) NOT NULL,
   start_time TIME NOT NULL,
   end_time TIME NOT NULL,
   breaks TIME NOT NULL,
   doctor_email VARCHAR(50) NOT NULL,
   FOREIGN KEY (doctor_email) REFERENCES Doctor(email) ON DELETE CASCADE
);

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

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

CREATE TABLE Diagnose (
   appointment_id INT NOT NULL,
   doctor_email VARCHAR(50) NOT NULL,
   diagnosis VARCHAR(100) NOT NULL,
   prescription VARCHAR(100) NOT NULL,
   PRIMARY KEY (appointment_id, doctor_email),
   FOREIGN KEY (appointment_id) REFERENCES Appointment(id) ON DELETE CASCADE,
   FOREIGN KEY (doctor_email) REFERENCES Doctor(email) ON DELETE CASCADE
);

CREATE TABLE AppointmentsCorrespondSchedules (
   appointment_id INT NOT NULL,
   schedule_id INT NOT NULL,
   PRIMARY KEY (appointment_id, schedule_id),
   FOREIGN KEY (appointment_id) REFERENCES Appointment(id) ON DELETE CASCADE,
   FOREIGN KEY (schedule_id) REFERENCES Schedule(id) ON DELETE CASCADE
);
```

11. SQL Queries to Insert Data

```
INSERT INTO Patient (email, password, name, address, gender) VALUES
('alice@example.com', 'alice123', 'Alice Brown', '123 Main Street', 'Female'),
('bob@example.com', 'bob123', 'Bob Smith', '456 Elm Avenue', 'Male');

INSERT INTO Doctor (email, password, name, gender) VALUES
('dr.john@example.com', 'john123', 'Dr. John Doe', 'Male'),
('dr.lisa@example.com', 'lisa123', 'Dr. Lisa Wong', 'Female');

INSERT INTO MedicalHistory (id, date_time, conditions, surgeries, medication, patient_email) VALUES
(1, '2023-01-10 09:00:00', 'Hypertension', 'Appendectomy', 'Lisinopril', 'alice@example.com'),
```

```sql
(2, '2023-02-15 11:30:00', 'Diabetes', 'None', 'Metformin',
'bob@example.com');

INSERT INTO Schedule (id, day, start_time, end_time, breaks,
doctor_email) VALUES
(1, 'Monday', '09:00:00', '17:00:00', '12:00:00',
'dr.john@example.com'),
(2, 'Tuesday', '10:00:00', '18:00:00', '13:00:00',
'dr.lisa@example.com');

INSERT INTO Appointment (id, date, start_time, end_time, status)
VALUES
(101, '2023-07-01', '10:00:00', '10:30:00', 'Completed'),
(102, '2023-07-02', '11:00:00', '11:30:00', 'Pending');

INSERT INTO PatientsAttendAppointments (patient_email,
appointment_id, concerns, symptoms) VALUES
('alice@example.com', 101, 'Headache', 'Nausea'),
('bob@example.com', 102, 'Checkup', 'Fatigue');

INSERT INTO Diagnose (appointment_id, doctor_email, diagnosis,
prescription) VALUES
(101, 'dr.john@example.com', 'Migraine', 'Ibuprofen'),
```

(102, 'dr.lisa@example.com', 'Routine Check', 'Vitamin D');

INSERT INTO DoctorViewsHistory (doctor_email, history_id) VALUES

('dr.john@example.com', 1),

('dr.lisa@example.com', 2);

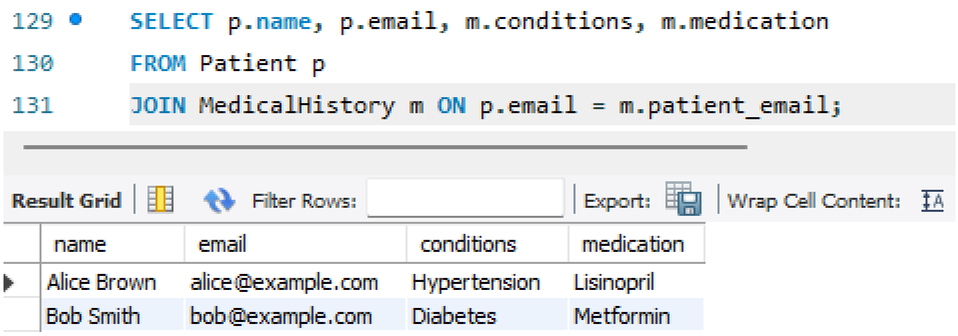
INSERT INTO AppointmentsCorrespondSchedules (appointment_id, schedule_id) VALUES

(101, 1),

(102, 2);

12. Database in Action

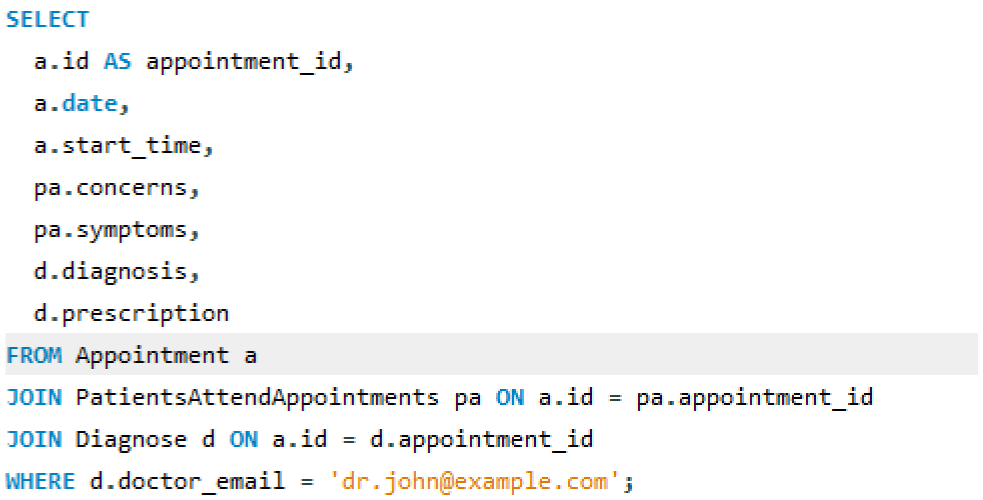
Q1. List all patients along with their medical conditions



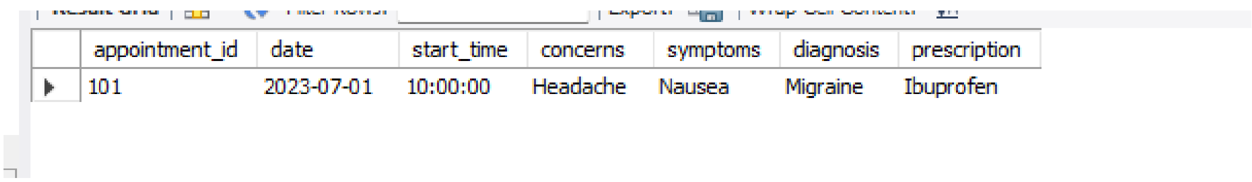
```
SELECT p.name, p.email, m.conditions, m.medication
FROM Patient p
JOIN MedicalHistory m ON p.email = m.patient_email;
```

name	email	conditions	medication
Alice Brown	alice@example.com	Hypertension	Lisinopril
Bob Smith	bob@example.com	Diabetes	Metformin

Q2. Find all appointments for Dr. John Doe including patient concerns

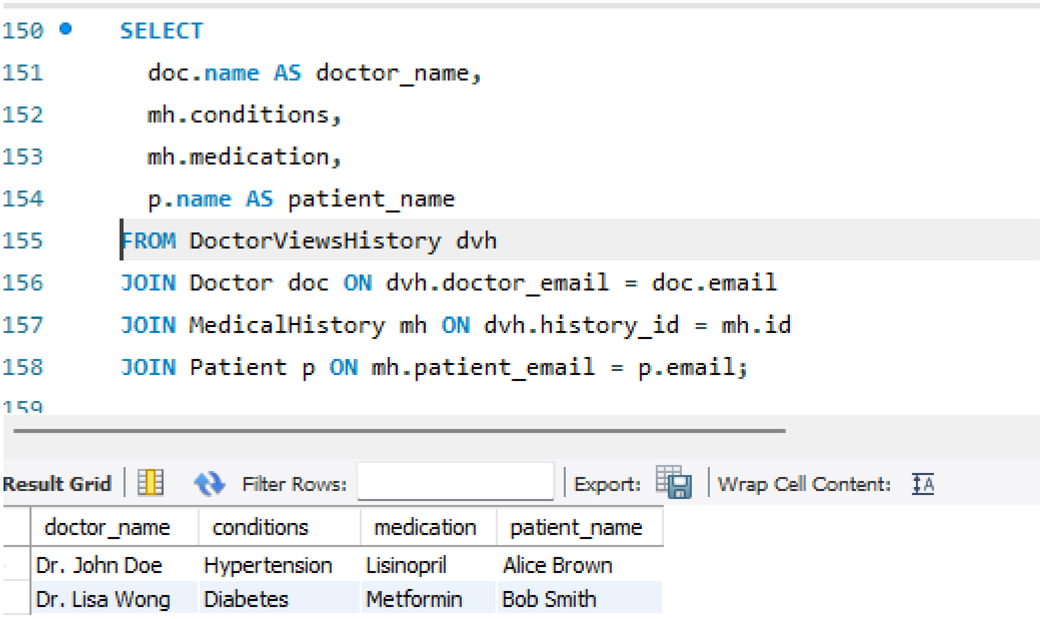


```sql
SELECT
  a.id AS appointment_id,
  a.date,
  a.start_time,
  pa.concerns,
  pa.symptoms,
  d.diagnosis,
  d.prescription
FROM Appointment a
JOIN PatientsAttendAppointments pa ON a.id = pa.appointment_id
JOIN Diagnose d ON a.id = d.appointment_id
WHERE d.doctor_email = 'dr.john@example.com';
```



appointment_id	date	start_time	concerns	symptoms	diagnosis	prescription
101	2023-07-01	10:00:00	Headache	Nausea	Migraine	Ibuprofen

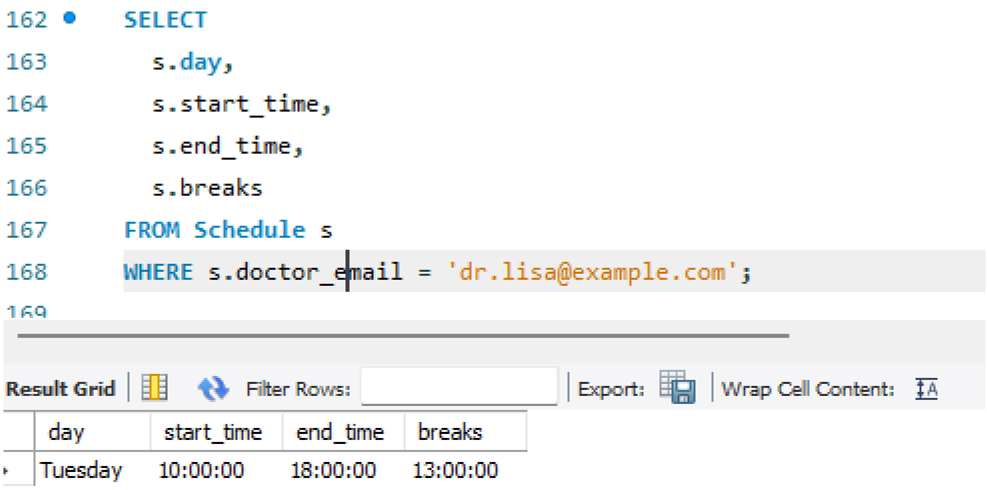
Q3. Show doctors and the medical histories they have viewed



```sql
SELECT
  doc.name AS doctor_name,
  mh.conditions,
  mh.medication,
  p.name AS patient_name
FROM DoctorViewsHistory dvh
JOIN Doctor doc ON dvh.doctor_email = doc.email
JOIN MedicalHistory mh ON dvh.history_id = mh.id
JOIN Patient p ON mh.patient_email = p.email;
```

doctor_name	conditions	medication	patient_name
Dr. John Doe	Hypertension	Lisinopril	Alice Brown
Dr. Lisa Wong	Diabetes	Metformin	Bob Smith

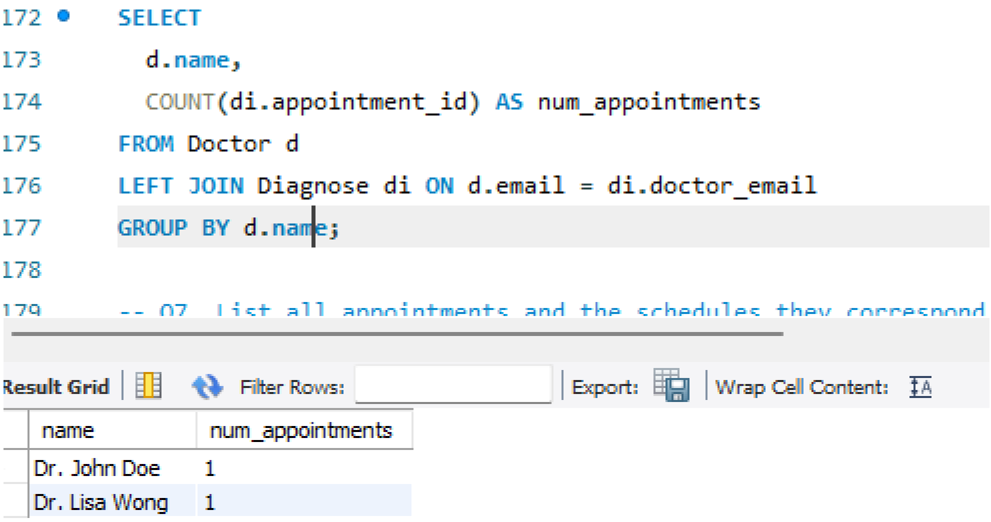
Q4. List all schedules for Dr. Lisa Wong



```sql
SELECT
  s.day,
  s.start_time,
  s.end_time,
  s.breaks
FROM Schedule s
WHERE s.doctor_email = 'dr.lisa@example.com';
```

day	start_time	end_time	breaks
Tuesday	10:00:00	18:00:00	13:00:00

Q5. Count the number of appointments each doctor has diagnosed

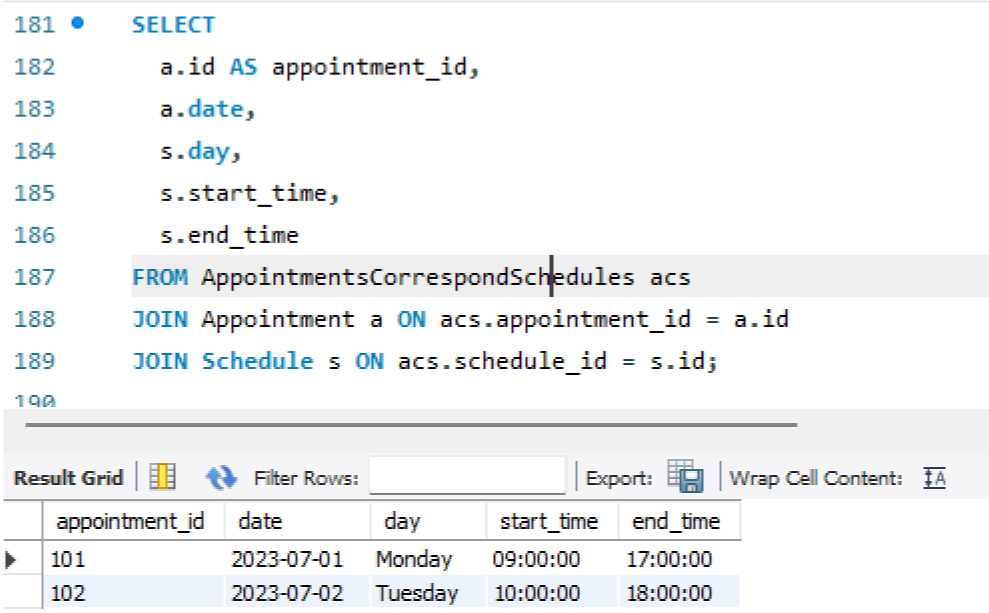


```sql
SELECT
  d.name,
  COUNT(di.appointment_id) AS num_appointments
FROM Doctor d
LEFT JOIN Diagnose di ON d.email = di.doctor_email
GROUP BY d.name;

-- Q7. List all appointments and the schedules they correspond
```

name	num_appointments
Dr. John Doe	1
Dr. Lisa Wong	1

Q6. List all appointments and the schedules they correspond to



```sql
SELECT
  a.id AS appointment_id,
  a.date,
  s.day,
  s.start_time,
  s.end_time
FROM AppointmentsCorrespondSchedules acs
JOIN Appointment a ON acs.appointment_id = a.id
JOIN Schedule s ON acs.schedule_id = s.id;
```

appointment_id	date	day	start_time	end_time
101	2023-07-01	Monday	09:00:00	17:00:00
102	2023-07-02	Tuesday	10:00:00	18:00:00