Day 3: Healthcare SQL Practice

Scenario Description

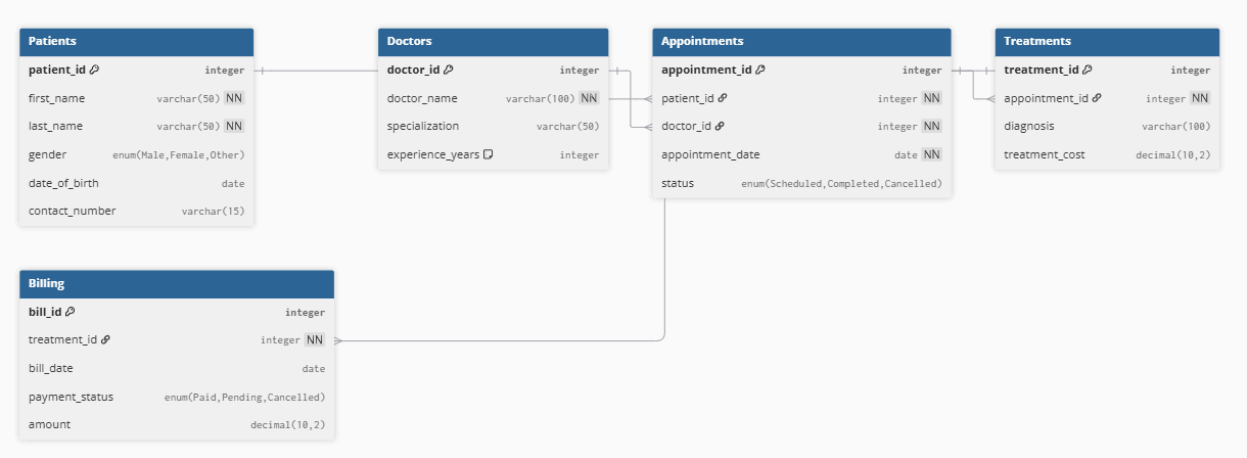
Business Context:

You are part of a data analytics team at a hospital management system. The organization manages patient records, doctor details, appointments, treatments, and billing information. Analysts use SQL to measure patient activity, track revenue, and analyze doctor performance.

Why it matters:

- Ensures accurate billing and insurance reconciliation.
- Identifies top-performing doctors and high-demand specializations.
- Detects missed appointments or delayed bill payments.
- Improves overall patient care through data-driven insights.

Database Schema



Tables

1. **Patients**

```
CREATE TABLE Patients (
    patient_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    gender ENUM('Male','Female','Other'),
    date_of_birth DATE,
    contact_number VARCHAR(15)
);
```

1. **Doctors**

```
CREATE TABLE Doctors (
    doctor_id INT PRIMARY KEY,
    doctor_name VARCHAR(100) NOT NULL,
    specialization VARCHAR(50),
    experience_years INT CHECK (experience_years >= 0)
);
```

1. **Appointments**

```
CREATE TABLE Appointments (
    appointment_id INT PRIMARY KEY,
    patient_id INT NOT NULL,
    doctor_id INT NOT NULL,
    appointment_date DATE NOT NULL,
    status ENUM('Scheduled','Completed','Cancelled'),
    FOREIGN KEY (patient_id) REFERENCES Patients(patient_id),
    FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id)
);
```

1. **Treatments**

```
CREATE TABLE Treatments (
    treatment_id INT PRIMARY KEY,
    appointment_id INT NOT NULL,
    diagnosis VARCHAR(100),
    treatment_cost DECIMAL(10,2),
    FOREIGN KEY (appointment_id) REFERENCES Appointments(appointment_i
d)
);
```

1. **Billing**

```
CREATE TABLE Billing (
    bill_id INT PRIMARY KEY,
    treatment_id INT NOT NULL,
    bill_date DATE,
    payment_status ENUM('Paid','Pending','Cancelled'),
    amount DECIMAL(10,2),
    FOREIGN KEY (treatment_id) REFERENCES Treatments(treatment_id)
);
```

Sample Data

Patients

```
INSERT INTO Patients VALUES
(1,'John','Smith','Male','1985-06-15','9876543210'),
(2,'Emily','Clark','Female','1992-09-23','8765432109'),
(3,'Raj','Patel','Male','2000-02-12','7654321098');
```

Doctors

```
INSERT INTO Doctors VALUES
(101,'Dr. Lisa Ray','Cardiology',12),
(102,'Dr. Arjun Mehta','Dermatology',7),
(103,'Dr. Sarah Khan','Neurology',15);
```

Appointments

```
INSERT INTO Appointments VALUES
(1001,1,101,'2025-04-01','Completed'),
(1002,2,102,'2025-04-03','Cancelled'),
(1003,3,103,'2025-04-05','Completed'),
(1004,1,103,'2025-04-08','Scheduled');
```

Treatments

```
INSERT INTO Treatments VALUES
(201,1001,'Heart Checkup',250.00),
(202,1003,'Migraine Therapy',400.00);
```

Billing

```
INSERT INTO Billing VALUES
(301,201,'2025-04-02','Paid',250.00),
(302,202,'2025-04-06','Pending',400.00);
```

ERD (Textual)

```
Patients (1) ──< Appointments (M) >── Doctors (1)
Appointments (1) ──< Treatments (M)
```

```
Treatments (1) ──< Billing (M)
```

Relationships:

- Patients ↔ Appointments: 1:M
- Doctors ↔ Appointments: 1:M
- Appointments ↔ Treatments: 1:M
- Treatments ↔ Billing: 1:M

SQL Questions

Easy

1. List all patients and their assigned doctor names for scheduled appointments.

Medium

1. Show total treatment cost for each doctor from completed appointments.

Hard

1. Find patients who have **no completed appointments**.

Difficult

1. Retrieve all doctors who have **treated more than one patient**.

Expert

1. Identify the **top-performing specialization** by **total revenue (Paid bills only)**, including total revenue and number of treatments.

Solutions with Explanations

Easy

```
SELECT p.first_name AS patient, d.doctor_name AS doctor, a.appointment_dat
e
FROM Patients p
JOIN Appointments a ON p.patient_id = a.patient_id
JOIN Doctors d ON a.doctor_id = d.doctor_id
WHERE a.status = 'Scheduled';
```

Explanation:

- Basic joins across Patients → Appointments → Doctors.
- Filters for `Scheduled` appointments.
- Useful for hospital scheduling dashboards.

Medium

```
SELECT d.doctor_name, SUM(t.treatment_cost) AS total_treatment_cost
FROM Doctors d
JOIN Appointments a ON d.doctor_id = a.doctor_id
JOIN Treatments t ON a.appointment_id = t.appointment_id
WHERE a.status = 'Completed'
GROUP BY d.doctor_name;
```

Explanation:

- Combines doctor, appointment, and treatment data.
- Filters only `Completed` appointments.
- Aggregates treatment costs per doctor.

Tip:

Add index on `Appointments.status` for faster filtering.

Hard

```
SELECT p.patient_id, p.first_name, p.last_name
FROM Patients p
LEFT JOIN Appointments a ON p.patient_id = a.patient_id
AND a.status = 'Completed'
WHERE a.appointment_id IS NULL;
```

Explanation:

- `LEFT JOIN` ensures all patients are included.
- `WHERE a.appointment_id IS NULL` finds patients without completed appointments.
- Excellent for follow-up targeting or engagement analytics.

Difficult

```
SELECT d.doctor_id, d.doctor_name, COUNT(DISTINCT a.patient_id) AS total_
patients
FROM Doctors d
JOIN Appointments a ON d.doctor_id = a.doctor_id
WHERE a.status = 'Completed'
GROUP BY d.doctor_id, d.doctor_name
HAVING COUNT(DISTINCT a.patient_id) > 1;
```

Explanation:

- Counts unique patients per doctor with completed appointments.
- Filters for doctors who treated more than one patient.
- Demonstrates grouping, distinct counting, and HAVING clause.

Tip:

Use `COUNT(DISTINCT ...)` cautiously on large data — can be expensive without indexing.

Expert

```
SELECT d.specialization,
       SUM(b.amount) AS total_revenue,
       COUNT(b.bill_id) AS total_treatments
FROM Doctors d
JOIN Appointments a ON d.doctor_id = a.doctor_id
JOIN Treatments t ON a.appointment_id = t.appointment_id
JOIN Billing b ON t.treatment_id = b.treatment_id
WHERE b.payment_status = 'Paid'
GROUP BY d.specialization
ORDER BY total_revenue DESC
LIMIT 1;
```

Explanation:

- Multi-table join across all hospital entities.
- Considers only paid bills.
- Aggregates total revenue per specialization and identifies top performer.
- Useful for strategic business reporting.

Optimization Tip:

Add indexes on `Billing.payment_status` and `Treatments.treatment_id` to improve query performance on large datasets.