**Hospital Management System Database**

**Report 2**

**Project by: Mohammad Hassan**

**Contents**

[**Introduction** 3](#_Toc201109955)

[**Tables** 3](#_Toc201109956)

[**Inserting Records** 8](#_Toc201109957)

[**SQL Queries and Outputs** 11](#_Toc201109958)

[**Self-Reflection** 29](#_Toc201109959)

[**Challenges Faced:** 29](#_Toc201109960)

[**Changes from Project Part 1:** 30](#_Toc201109961)

[**Lessons Learned:** 30](#_Toc201109962)

# **Introduction**

In these modern days, to handle enormous volumes of data concerning patients, employees, medical procedures, billing, and logistics, the healthcare industry need reliable systems. These tasks are automated and streamlined by a hospital management system (HMS) to increase productivity, lower errors, and improve patient care. Using a relational database technique, our effort seeks to model such a system that replicates actual hospital processes. It incorporates core functionalities such as appointment scheduling, billing, admissions, medical records, diagnosis, and staff management.

In this report, we present a comprehensive walkthrough of the SQL implementation — from table creation and data insertion to complex queries, joins, and view definitions that demonstrate the relational integrity and analytical capabilities of the system.

# **Tables**

The following section outlines the SQL queries used to create all the database tables, along with the data types, foreign key relationships, and constraints applied.

**-- Creating table for storing patient information**

CREATE TABLE Patient (

Patient\_ID INT PRIMARY KEY,

PatientName VARCHAR(100) NOT NULL,

Age INT CHECK (Age >= 0 AND Age <= 120),

Gender VARCHAR(10) CHECK (Gender IN ('Male', 'Female', 'Other')),

Address TEXT,

Contact\_Number VARCHAR(20) UNIQUE,

DOB DATE,

Blood\_Type VARCHAR(10) CHECK (Blood\_Type IN ('A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-'))

);

**-- Creating table for hospital departments**

CREATE TABLE Department (

Department\_ID VARCHAR(10) PRIMARY KEY,

Department\_Name VARCHAR(100) NOT NULL UNIQUE

);

**-- Creating table for all hospital employees**

CREATE TABLE Employee (

Employee\_ID INT PRIMARY KEY,

Role VARCHAR(50),

Department\_ID VARCHAR(10),

Joining\_Date DATE,

FOREIGN KEY (Department\_ID) REFERENCES Department(Department\_ID) ON DELETE CASCADE );

**-- Creating table for receptionists**

CREATE TABLE Receptionist (

Receptionist\_ID INT PRIMARY KEY,

R\_Name VARCHAR(100),

Number VARCHAR(20),

Email VARCHAR(100) UNIQUE,

Employee\_ID INT,

FOREIGN KEY (Employee\_ID) REFERENCES Employee(Employee\_ID) ON DELETE CASCADE

);

**-- Creating table for doctors**

CREATE TABLE Doctor (

Doctor\_ID INT PRIMARY KEY,

Doctor\_Name VARCHAR(100),

Specialization VARCHAR(100),

Contact VARCHAR(20),

Email VARCHAR(100) UNIQUE,

License\_Number VARCHAR(50) UNIQUE NOT NULL,

Employee\_ID INT,

FOREIGN KEY (Employee\_ID) REFERENCES Employee(Employee\_ID) ON DELETE CASCADE

);

**-- Creating table for appointments**

CREATE TABLE Appointment (

Appointment\_ID INT PRIMARY KEY,

Patient\_ID INT,

Doctor\_ID INT,

Receptionist\_ID INT,

Date DATE,

Time TIME,

Status VARCHAR(50) CHECK (Status IN ('Scheduled', 'Completed', 'Cancelled')),

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID) ON DELETE CASCADE,

FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID) ON DELETE CASCADE,

FOREIGN KEY (Receptionist\_ID) REFERENCES Receptionist(Receptionist\_ID) ON DELETE CASCADE

);

**-- Creating table for diagnoses**

CREATE TABLE Diagnosis (

Diagnosis\_ID INT PRIMARY KEY,

Appointment\_ID INT,

Patient\_ID INT,

Doctor\_ID INT,

Description TEXT,

Date DATE,

FOREIGN KEY (Appointment\_ID) REFERENCES Appointment(Appointment\_ID) ON DELETE CASCADE,

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID) ON DELETE CASCADE,

FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID) ON DELETE CASCADE

);

**-- Creating table for treatments**

CREATE TABLE Treatment (

Treatment\_ID VARCHAR(10) PRIMARY KEY,

Diagnosis\_ID INT,

Treatment\_Type VARCHAR(100),

Description TEXT,

Treatment\_Cost DECIMAL(10, 2) CHECK (Treatment\_Cost >= 0),

FOREIGN KEY (Diagnosis\_ID) REFERENCES Diagnosis(Diagnosis\_ID) ON DELETE CASCADE

);

**-- Creating table for hospital rooms**

CREATE TABLE Room (

Room\_ID INT PRIMARY KEY,

Room\_Number VARCHAR(10) UNIQUE,

Type VARCHAR(50) CHECK (Type IN ('ICU', 'General', 'Private')),

Status VARCHAR(20) CHECK (Status IN ('Available', 'Occupied'))

);

**-- Creating table for patient admissions**

CREATE TABLE Admission (

Admission\_ID INT PRIMARY KEY,

Patient\_ID INT,

Diagnosis\_ID INT,

Room\_ID INT,

Admission\_Date DATE,

Discharge\_Date DATE,

Room\_Cost DECIMAL(10, 2) CHECK (Room\_Cost >= 0),

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID) ON DELETE CASCADE,

FOREIGN KEY (Diagnosis\_ID) REFERENCES Diagnosis(Diagnosis\_ID) ON DELETE CASCADE,

FOREIGN KEY (Room\_ID) REFERENCES Room(Room\_ID) ON DELETE CASCADE );

**-- Creating table for nurses**

CREATE TABLE Nurse (

Nurse\_ID INT PRIMARY KEY,

Nurse\_Name VARCHAR(100),

Contact\_Number VARCHAR(20) UNIQUE,

Shift\_Timings VARCHAR(100),

Room\_ID INT,

Employee\_ID INT,

FOREIGN KEY (Employee\_ID) REFERENCES Employee(Employee\_ID) ON DELETE CASCADE,

FOREIGN KEY (Room\_ID) REFERENCES Room(Room\_ID) ON DELETE CASCADE

);

**-- Creating table for prescriptions**

CREATE TABLE Prescription (

PrescriptionID VARCHAR(10) PRIMARY KEY,

DiagnosisID INT,

FOREIGN KEY (DiagnosisID) REFERENCES Diagnosis(Diagnosis\_ID) ON DELETE CASCADE

);

**-- Creating table for prescription medicines**

CREATE TABLE PrescriptionMedicine (

PrescriptionMedicineID VARCHAR(10) PRIMARY KEY,

PrescriptionID VARCHAR(10),

MedicineName VARCHAR(100),

Dosage VARCHAR(50),

Instructions TEXT,

Duration VARCHAR(50),

FOREIGN KEY (PrescriptionID) REFERENCES Prescription(PrescriptionID) ON DELETE CASCADE

);

**-- Creating table for labs**

CREATE TABLE Lab (

Lab\_ID VARCHAR(10) PRIMARY KEY,

DiagnosisID INT,

FOREIGN KEY (DiagnosisID) REFERENCES Diagnosis(Diagnosis\_ID) ON DELETE CASCADE

);

**-- Creating table for lab tests**

CREATE TABLE Test (

Test\_ID VARCHAR(10) PRIMARY KEY,

Lab\_ID VARCHAR(10),

Test\_Name VARCHAR(100),

Description TEXT,

Test\_Cost DECIMAL(10, 2) CHECK (Test\_Cost >= 0),

FOREIGN KEY (Lab\_ID) REFERENCES Lab(Lab\_ID) ON DELETE CASCADE

);

**-- Creating table for medical history of patients**

CREATE TABLE Medical\_History (

History\_ID INT PRIMARY KEY,

Patient\_ID INT,

Doctor\_ID INT,

Description TEXT,

Date\_Recorded DATE,

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID) ON DELETE CASCADE,

FOREIGN KEY (Doctor\_ID) REFERENCES Doctor(Doctor\_ID) ON DELETE CASCADE

);

**-- Creating table for billing information**

CREATE TABLE Bill (

Bill\_ID INT PRIMARY KEY,

Patient\_ID INT,

Appointment\_ID INT,

Receptionist\_ID INT,

Amount DECIMAL(10, 2) CHECK (Amount >= 0),

Bill\_Date DATE,

Bill\_Status VARCHAR(50),

FOREIGN KEY (Patient\_ID) REFERENCES Patient(Patient\_ID) ON DELETE CASCADE,

FOREIGN KEY (Appointment\_ID) REFERENCES Appointment(Appointment\_ID) ON DELETE CASCADE,

FOREIGN KEY (Receptionist\_ID) REFERENCES Receptionist(Receptionist\_ID) ON DELETE CASCADE

);

# **Inserting Records**

This section presents the SQL queries used to insert sample data into each table of our hospital management system. To keep the report concise and focused, we have included only three representative records per table. It’s important to note that the full implementation script contains a significantly larger dataset to simulate real-world usage and ensure robust query testing.

**-- Inserting sample records into Patient table**

INSERT INTO Patient (Patient\_ID, PatientName, Age, Gender, Address, Contact\_Number, DOB, Blood\_Type) VALUES

(1, 'Ali Raza', 34, 'Male', '742 Evergreen Terrace, Springfield', 03001234567, '1990-03-25', 'A+'),

(2, 'Emily Davis', 12, 'Female', '1234 Elm St, New York, USA', '001-212-4567890', '2013-06-20', 'O+'),

(3, 'Ahmed Khan', 41, 'Male', 'Street 45, G-11, Islamabad', 03111222333, '1983-07-12', 'B+');

**-- Inserting records into Department table**

INSERT INTO Department (Department\_ID, Department\_Name) VALUES

('Depart1', 'Cardiology'),

('Depart2', 'Neurology'),

('Depart3', 'Emergency');

**-- Inserting records into Employee table**

INSERT INTO Employee (Employee\_ID, Role, Department\_ID, Joining\_Date) VALUES

(13, 'Receptionist', 'Depart5', '2022-05-11'),

(28, 'Nurse', 'Depart7', '2021-02-14'),

(16, 'Doctor', 'Depart9', '2021-09-10');

**-- Inserting records into Receptionist table**

INSERT INTO Receptionist (Receptionist\_ID, R\_Name, Number, Email, Employee\_ID) VALUES

(1, 'Ayesha Khan', '0301-4567890', 'ayesha.khan@hospital.pk', 1),

(2, 'Emily Smith', '0323-1122334', 'emily.smith@hospital.com', 3),

(3, 'Fatima Noor', '0302-9988776', 'fatima.noor@hospital.pk', 7);

**-- Inserting records into Doctor table**

INSERT INTO Doctor (Doctor\_ID, Doctor\_Name, Specialization, Contact, Email, License\_Number, Employee\_ID) VALUES

(1, 'Dr. Asim Qureshi', 'Cardiology', '0301-5566778', 'asim.qureshi@hospital.pk', 'PKC-1001', 2),

(2, 'Dr. Emily Grace', 'Neurology', '0312-8899002', 'emily.grace@hospital.com', 'USN-2002', 4),

(3, 'Dr. Sarah Malik', 'Emergency', '0323-4567891', 'sarah.malik@hospital.pk', 'PKE-1003', 6);

**-- Inserting records into Appointment table**

INSERT INTO Appointment (Appointment\_ID, Patient\_ID, Doctor\_ID, Receptionist\_ID, Date, Time, Status) VALUES

(1, 5, 2, 1, '2025-06-01', '09:00:00', 'Completed'),

(2, 12, 5, 3, '2025-06-02', '10:30:00', 'Completed'), (3, 8, 7, 2, '2025-06-03', '14:00:00', 'Cancelled');

**-- Inserting records into Diagnosis table**

INSERT INTO Diagnosis (Diagnosis\_ID, Appointment\_ID, Patient\_ID, Doctor\_ID, Description, Date) VALUES

(1, 1, 5, 2, 'Mild upper respiratory infection', '2025-06-01'),

(2, 2, 12, 5, 'Seasonal allergies with sneezing', '2025-06-02'),

(3, 3, 8, 7, 'Sprained ankle', '2025-06-03');

**-- Inserting records into Treatment table**

INSERT INTO Treatment (Treatment\_ID, Diagnosis\_ID, Treatment\_Type, Description, Treatment\_Cost) VALUES

(1, 1, 'Medical', 'Prescribed antibiotics and rest', 1500),

(2, 2, 'Medical', 'Antihistamines and nasal spray', 1800),

(3, 3, 'Therapy', 'Physical therapy for ankle recovery', 3000);

**-- Inserting records into Room table**

INSERT INTO Room (Room\_ID, Room\_Number, Type, Status) VALUES

(1, 'ICU-101', 'ICU', 'Occupied'),

(2, 'GEN-102', 'General', 'Available'),

(3, 'PRI-103', 'Private', 'Occupied');

**-- Inserting records into Admission table**

INSERT INTO Admission (Admission\_ID, Patient\_ID, Diagnosis\_ID, Room\_ID, Admission\_Date, Discharge\_Date, Room\_Cost) VALUES

(1, 1, 1, 1, '2024-12-01', '2024-12-05', 10000),

(2, 2, 2, 3, '2024-11-15', '2024-11-18', 15000),

(3, 3, 3, 5, '2025-01-10', '2025-01-12', 8000);

**-- Inserting records into Nurse table**

INSERT INTO Nurse (Nurse\_ID, Nurse\_Name, Contact\_Number, Shift\_Timings, Room\_ID, Employee\_ID) VALUES

(1, 'Ayesha Khan', 03001234567, 'Morning', 1, 5),

(2, 'Sarah Williams', '+447911123456', 'Evening', 2, 27),

(3, 'Nadia Akhtar', 03123456789, 'Night', 3, 28);

**-- Inserting records into Medical\_History table**

INSERT INTO Medical\_History (History\_ID, Patient\_ID, Doctor\_ID, Description, Date\_Recorded) VALUES

(1, 12, 4, 'History of seasonal asthma during childhood.', '2020-03-10'),

(2, 3, 1, 'Diagnosed with hypertension in 2021.', '2021-06-15'),

(3, 17, 6, 'Suffered minor concussion after a road accident.', '2019-12-22');

**-- Inserting records into Lab table**

INSERT INTO Lab (Lab\_ID, DiagnosisID) VALUES

('L1', 5), ('L2', 9), ('L3', 14), ('L4', 1), ('L5', 3), ('L6', 7), ('L7', 6), ('L8', 2), ('L9', 8), ('L10', 10),

('L11', 4), ('L12', 11), ('L13', 12), ('L14', 13), ('L15', 15), ('L16', 16), ('L17', 17), ('L18', 18), ('L19', 19), ('L20', 20), ('L21', 21), ('L22', 22), ('L23', 23), ('L24', 24), ('L25', 25), ('L26', 26), ('L27', 27), ('L28', 28), ('L29', 29), ('L30', 30);

**-- Inserting records into Test table**

INSERT INTO Test (Test\_ID, Lab\_ID, Test\_Name, Description, Test\_Cost) VALUES

('TEST1', 'L1', 'CBC', 'Complete Blood Count', 1500),

('TEST2', 'L2', 'Lipid Profile', 'Cholesterol and Triglycerides', 2800),

('TEST3', 'L3', 'Liver Function Test', 'Check liver health', 3200);

**-- Inserting records into Prescription table**

INSERT INTO Prescription (PrescriptionID, DiagnosisID) VALUES

('PRX1', 1),

('PRX2', 2),

('PRX3', 3);

**-- Inserting records into PrescriptionMedicine table**

INSERT INTO PrescriptionMedicine (PrescriptionMedicineID, PrescriptionID, MedicineName, Dosage, Instructions, Duration) VALUES

('MED1', 'PRX1', 'Paracetamol', '2 tablets', 'After meal', '5 days'),

('MED2', 'PRX2', 'Amoxicillin', '1 tablet', 'Before meal', '7 days'),

('MED3', 'PRX3', 'Ibuprofen', '1 tablet', 'With water', '3 days');

**-- Inserting records into Bill table**

INSERT INTO Bill (Bill\_ID, Patient\_ID, Appointment\_ID, Receptionist\_ID, Amount, Bill\_Date, Bill\_Status) VALUES

(1, 5, 1, 1, 5000, '2025-06-01', 'Paid'),

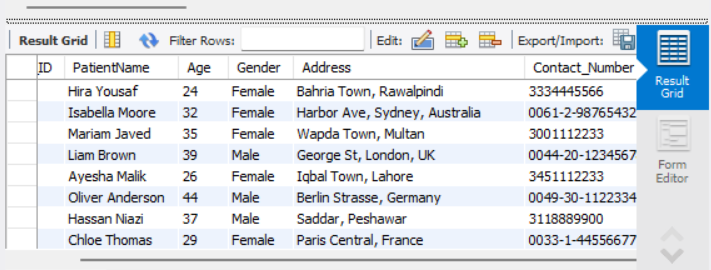
(2, 12, 2, 2, 3500, '2025-06-02', 'Paid'),

(3, 8, 3, 3, 2500, '2025-06-03', 'Refunded');

# **SQL Queries and Outputs**

**-- Selecting all records from Patient table**

SELECT \* FROM Patient;



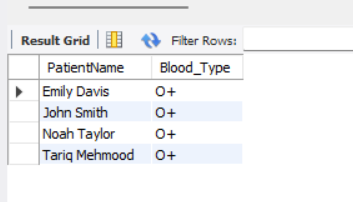
**-- Selecting patient names with Blood Type 'O+'**

SELECT PatientName, Blood\_Type

FROM Patient

WHERE Blood\_Type = 'O+'

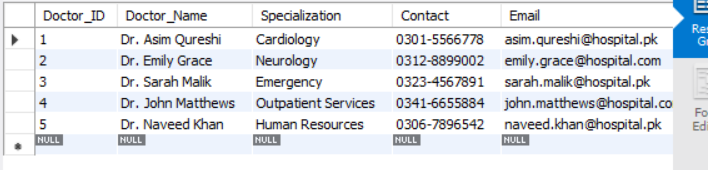
ORDER BY PatientName ASC;



**-- Selecting limited records from Doctor Table**

SELECT \* FROM Doctor

LIMIT 5;



**-- Joining Appointment, Patient, and Doctor Tables**

SELECT a.Appointment\_ID, p.PatientName AS Patient, d.Doctor\_Name AS Doctor, a.Date

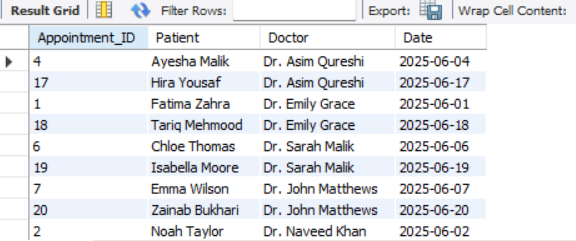
FROM Appointment a

INNER JOIN Patient p

ON a.Patient\_ID = p.Patient\_ID

INNER JOIN Doctor d

ON a.Doctor\_ID = d.Doctor\_ID;



**-- Finding patients who have not been admitted**

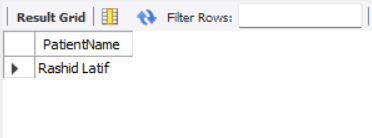
SELECT p.PatientName

FROM Patient p

LEFT JOIN Admission adm

ON p.Patient\_ID = adm.Patient\_ID

WHERE adm.Admission\_ID IS NULL;



**-- List room numbers and patient occupants**

SELECT r.Room\_Number, p.PatientName AS Occupant

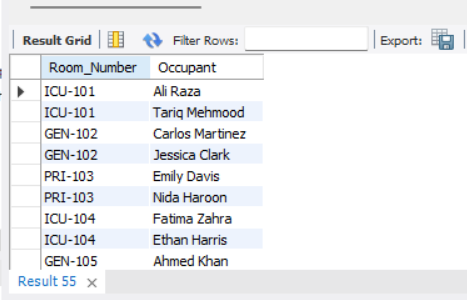
FROM Room r

LEFT JOIN Admission adm

ON r.Room\_ID = adm.Room\_ID

LEFT JOIN Patient p

ON adm.Patient\_ID = p.Patient\_ID;



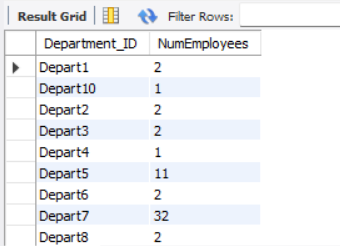
**-- Count employees per department**

SELECT Department\_ID, COUNT(\*) AS NumEmployees

FROM Employee

GROUP BY Department\_ID

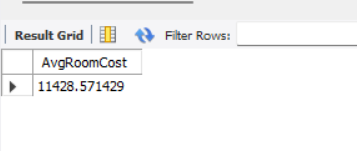
ORDER BY Department\_ID ASC;



**-- Calculating average room cost**

SELECT AVG(Room\_Cost) AS AvgRoomCost

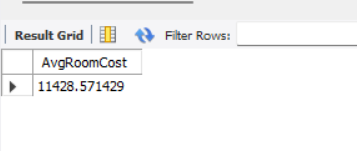
FROM Admission;



**-- Finding highest treatment cost**

SELECT MAX(Treatment\_Cost) AS HighestTreatmentCost

FROM Treatment;



**-- Patients with scheduled appointments**

SELECT PatientName

FROM Patient

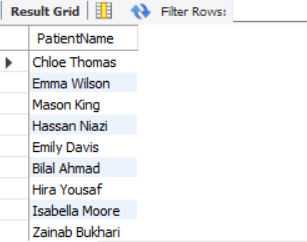
WHERE Patient\_ID IN (

SELECT Patient\_ID

FROM Appointment

WHERE Status = 'Scheduled'

);



**-- Patients who have paid all their bills**

SELECT p.PatientName

FROM Patient p

WHERE NOT EXISTS (

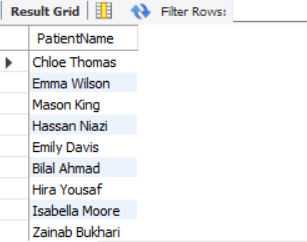
SELECT 1

FROM Bill b

WHERE b.Patient\_ID = p.Patient\_ID

AND b.Bill\_Status = 'Unpaid'

);



**-- Average bill amount above overall average**

SELECT AVG(Amount) AS AvgAboveOverall

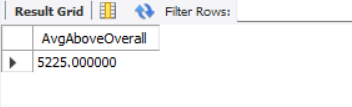
FROM Bill

WHERE Amount > (

SELECT AVG(Amount)

FROM Bill

);



**-- Updating patient address**

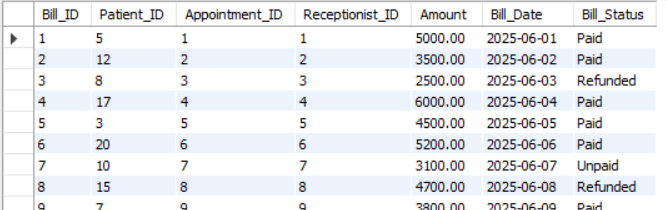
UPDATE Patient

SET Address = 'House #99, New Address, Lahore'

WHERE Patient\_ID = 1;



SELECT \* FROM Bill;



**-- Updating bill status**

UPDATE Bill

SET Bill\_Status = 'Paid'

WHERE Bill\_ID = 6;



**-- Updating doctor specialization**

UPDATE Doctor

SET Specialization = 'General Medicine'

WHERE Doctor\_ID = 8;



**-- Deleting cancelled appointments**

DELETE FROM Appointment

WHERE Status = 'Cancelled';



**-- Deleting prescription medicines with '3 days' duration**

DELETE FROM PrescriptionMedicine

WHERE Duration = '3 days';



**-- Deleting patient with ID 30**

DELETE FROM Patient

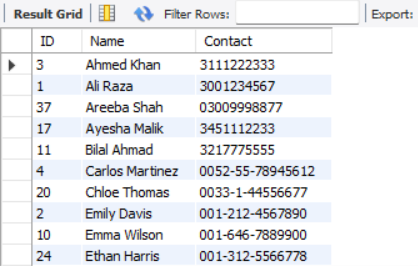
WHERE Patient\_ID = 30;

**-- Displaying patient contact info**

SELECT Patient\_ID AS ID, PatientName AS Name, Contact\_Number AS Contact

FROM Patient

ORDER BY PatientName ASC;



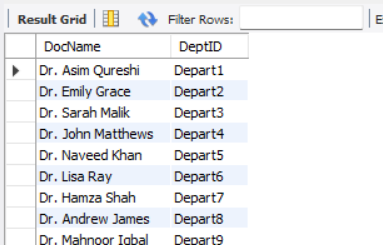
**-- Joining Doctor and Employee to display the Doctor Name with Department ID**

SELECT d.Doctor\_Name AS DocName, e.Department\_ID AS DeptID

FROM Doctor d

JOIN Employee e

ON d.Employee\_ID = e.Employee\_ID;



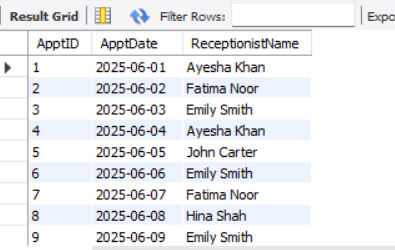
**-- Joining Appointment and Receptionist**

SELECT a.Appointment\_ID AS ApptID, a.Date AS ApptDate, r.R\_Name AS ReceptionistName

FROM Appointment a

JOIN Receptionist r

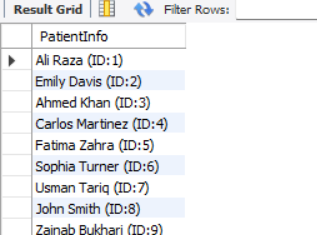
ON a.Receptionist\_ID = r.Receptionist\_ID;



**-- Displaying patient info using CONCAT**

SELECT CONCAT(PatientName, ' (ID:', Patient\_ID, ')') AS PatientInfo

FROM Patient;



**-- Generates a readable doctor profile summary combining name, specialization, department, and contact.**

SELECT CONCAT

(

d.Doctor\_Name,

' specializes in ', d.Specialization,

' (Dept: ', e.Department\_ID,

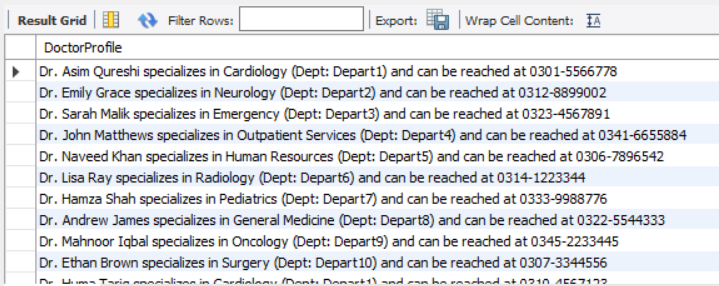
') and can be reached at ', d.Contact

) AS DoctorProfile

FROM Doctor d

JOIN Employee e

ON d.Employee\_ID = e.Employee\_ID;



**-- Produces a descriptive sentence for each patient detailing their diagnosis and treatment plan.**

SELECT

CONCAT(

'Patient ', p.PatientName,

' was diagnosed with "', d.Description,

'" and received treatment: "', t.Description, '".'

) AS MedicalSummary

FROM Patient p

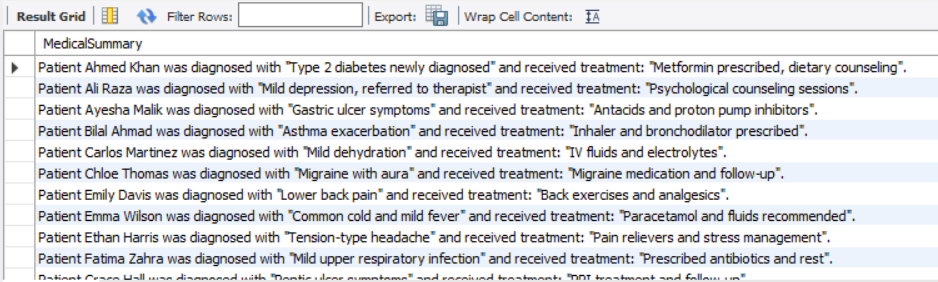
JOIN Diagnosis d

ON p.Patient\_ID = d.Patient\_ID

JOIN Treatment t

ON d.Diagnosis\_ID = t.Diagnosis\_ID

ORDER BY p.PatientName;

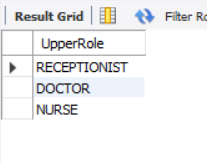


**-- Displaying employee roles in uppercase**

SELECT UPPER(Role) AS UpperRole

FROM Employee

GROUP BY Role;

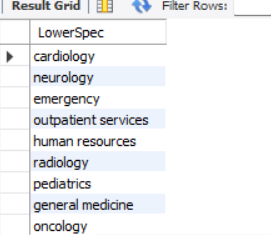


**-- Displaying doctor specializations in lowercase**

SELECT LOWER(Specialization) AS LowerSpec

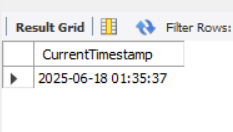
FROM Doctor

GROUP BY Specialization;



**-- Showing current timestamp**

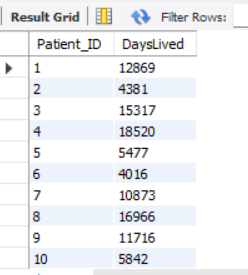
SELECT NOW() AS CurrentTimestamp;



**-- Calculating patient age in days and considering it as DaysLived**

SELECT Patient\_ID, DATEDIFF(NOW(), DOB) AS DaysLived

FROM Patient;



**-- Extracting year from appointment date**

SELECT Appointment\_ID, YEAR(Date) AS YearOfAppointment

FROM Appointment;



**-- Categorizing patients as adult or minor**

SELECT Patient\_ID,

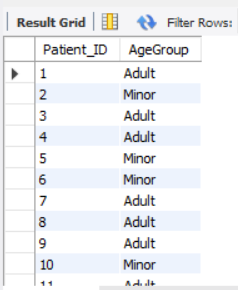
CASE

WHEN DATEDIFF(NOW(), DOB)/365 >= 18 THEN 'Adult'

ELSE 'Minor'

END AS AgeGroup

FROM Patient;



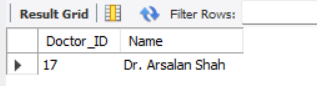
**-- Finding doctors with no diagnoses**

SELECT d.Doctor\_ID, d.Doctor\_Name AS Name

FROM Doctor d

LEFT JOIN Diagnosis diag ON d.Doctor\_ID = diag.Doctor\_ID

WHERE diag.Diagnosis\_ID IS NULL;



**-- Right join to show room and admitted patient info**

SELECT

r.Room\_Number,

a.Admission\_ID,

p.PatientName,

a.Admission\_Date

FROM Room r

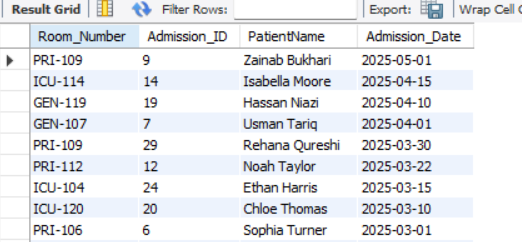
RIGHT JOIN Admission a

ON r.Room\_ID = a.Room\_ID

LEFT JOIN Patient p

ON a.Patient\_ID = p.Patient\_ID

ORDER BY a.Admission\_Date DESC;



**-- Full outer join for doctors and appointments**

SELECT

d.Doctor\_ID,

d.Doctor\_Name,

a.Appointment\_ID,

a.Date

FROM Doctor d

LEFT JOIN Appointment a

ON d.Doctor\_ID = a.Doctor\_ID

UNION

SELECT

d.Doctor\_ID,

d.Doctor\_Name,

a.Appointment\_ID,

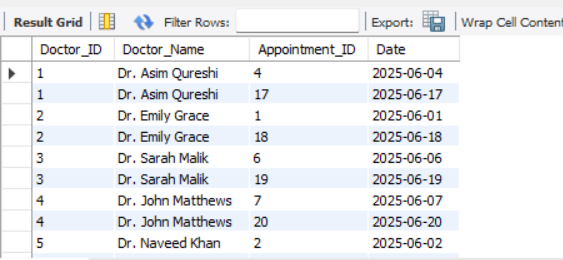
a.Date

FROM Doctor d

RIGHT JOIN Appointment a

ON d.Doctor\_ID = a.Doctor\_ID

ORDER BY Doctor\_ID;



**-- Full outer join to check if employees are doctors. NULL indicating that those are not doctors**

SELECT

e.Employee\_ID,

e.Role,

d.Doctor\_ID,

d.Doctor\_Name

FROM Employee e

LEFT JOIN Doctor d ON e.Employee\_ID = d.Employee\_ID

UNION

SELECT

e.Employee\_ID,

e.Role,

d.Doctor\_ID,

d.Doctor\_Name

FROM Employee e

RIGHT JOIN Doctor d ON e.Employee\_ID = d.Employee\_ID

ORDER BY Employee\_ID;



**-- Lists doctors and number of diagnoses they've made**

SELECT d.Doctor\_Name, COUNT(diag.Diagnosis\_ID) AS DiagnosesHandled

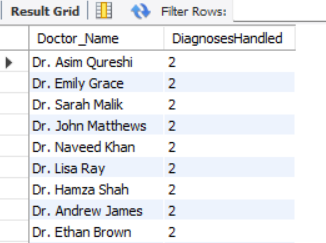
FROM Doctor d

LEFT JOIN Diagnosis diag

ON d.Doctor\_ID = diag.Doctor\_ID

GROUP BY d.Doctor\_ID, d.Doctor\_Name

ORDER BY DiagnosesHandled DESC;



**-- Finds the top 5 patients with the longest hospital stay**

SELECT p.PatientName, DATEDIFF(a.Discharge\_Date, a.Admission\_Date) AS StayLength, r.Type AS RoomType

FROM Admission a

JOIN Patient p

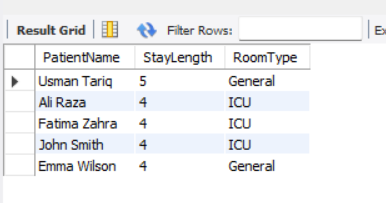
ON a.Patient\_ID = p.Patient\_ID

JOIN Room r

ON a.Room\_ID = r.Room\_ID

ORDER BY StayLength DESC

LIMIT 5;



**-- Creating a view to show patient diagnosis info**

CREATE VIEW PatientDiagnosisView AS

SELECT

p.Patient\_ID,

p.PatientName,

p.Blood\_Type,

d.Diagnosis\_ID,

d.Description AS Diagnosis\_Description,

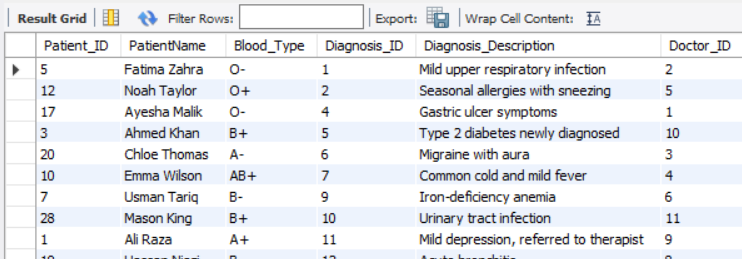
d.Doctor\_ID

FROM Patient p

JOIN Diagnosis d ON p.Patient\_ID = d.Patient\_ID;



SELECT \* FROM PatientDiagnosisView;



**-- Creating a view for full patient medical details**

CREATE VIEW PatientFullMedicalView AS

SELECT

p.Patient\_ID,

p.PatientName,

d.Description AS Diagnosis\_Description,

pm.MedicineName,

t.Description AS Treatment\_Description,

tst.Test\_Name AS LabTest,

h.Description AS History\_Description

FROM Patient p

LEFT JOIN Diagnosis d ON p.Patient\_ID = d.Patient\_ID

LEFT JOIN Prescription pr ON d.Diagnosis\_ID = pr.DiagnosisID

LEFT JOIN PrescriptionMedicine pm ON pr.PrescriptionID = pm.PrescriptionID

LEFT JOIN Treatment t ON d.Diagnosis\_ID = t.Diagnosis\_ID

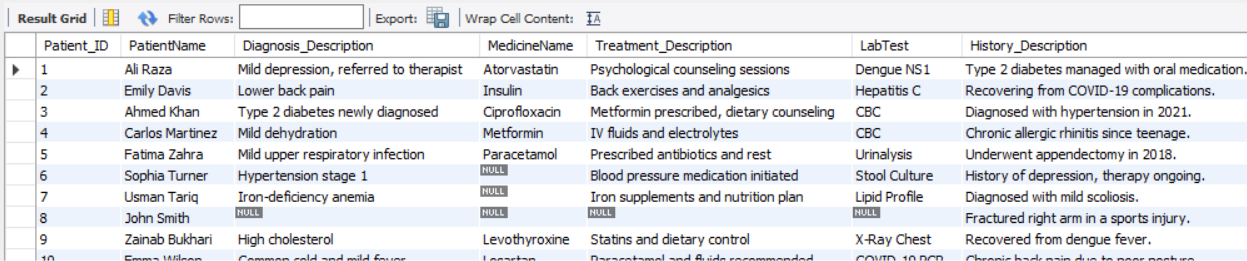
LEFT JOIN Lab l ON d.Diagnosis\_ID = l.DiagnosisID

LEFT JOIN Test tst ON l.Lab\_ID = tst.Lab\_ID

LEFT JOIN Medical\_History h ON p.Patient\_ID = h.Patient\_ID;



SELECT \* FROM PatientFullMedicalView;



**-- Creates a cleaner view showing only actual hospital charges per patient, without billed summary fields**

CREATE VIEW PatientBillBreakdownView AS

SELECT

p.Patient\_ID,

p.PatientName,

IFNULL(t.Treatment\_Cost, 0) AS Treatment\_Cost,

IFNULL(a.Room\_Cost, 0) AS Room\_Cost,

IFNULL(tst.Test\_Cost, 0) AS Test\_Cost,

(IFNULL(t.Treatment\_Cost, 0) + IFNULL(a.Room\_Cost, 0) + IFNULL(tst.Test\_Cost, 0)) AS Computed\_Total\_Charges,

CONCAT('Breakdown - Treatment: Rs.', IFNULL(t.Treatment\_Cost, 0), ', Room: Rs.', IFNULL(a.Room\_Cost, 0), ', Tests: Rs.', IFNULL(tst.Test\_Cost, 0)) AS ChargeDetails,

b.Bill\_Status

FROM Patient p

LEFT JOIN Diagnosis d

ON p.Patient\_ID = d.Patient\_ID

LEFT JOIN Treatment t

ON d.Diagnosis\_ID = t.Diagnosis\_ID

LEFT JOIN Admission a

ON p.Patient\_ID = a.Patient\_ID

LEFT JOIN Lab l

ON d.Diagnosis\_ID = l.DiagnosisID

LEFT JOIN Test tst

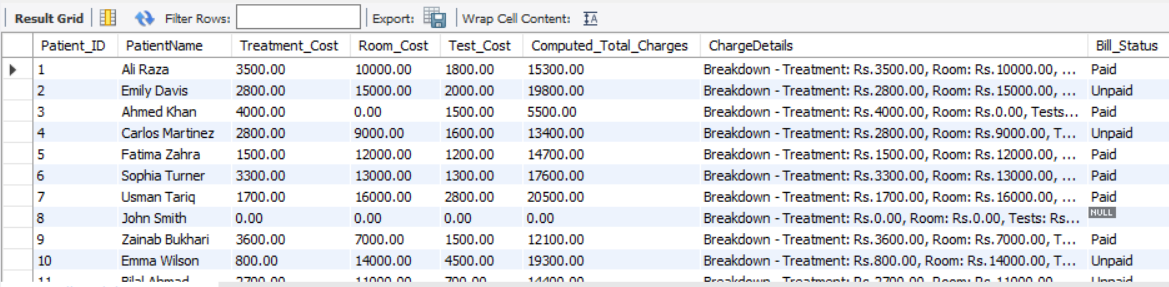
ON l.Lab\_ID = tst.Lab\_ID

LEFT JOIN Bill b

ON p.Patient\_ID = b.Patient\_ID;



SELECT \* FROM PatientBillBreakdownView;



**-- Creating index on Patient\_ID**

CREATE INDEX Patient\_ID\_index

ON Patient(Patient\_ID);

**-- Creating index on Appointment\_ID**

CREATE INDEX Appointment\_ID\_index

ON Appointment(Appointment\_ID);

**-- Creating index on Diagnosis\_ID**

CREATE INDEX Diagnosis\_ID\_index

ON Diagnosis(Diagnosis\_ID);

**-- Creating index on Doctor\_ID**

CREATE INDEX Doctor\_ID\_index

ON Doctor(Doctor\_ID);

# **Self-Reflection**

## **Challenges Faced:**

Throughout the course of this project, we encountered several challenges—some straightforward to resolve, while others required deeper problem-solving and research. One significant issue emerged when attempting to delete a patient record by ID. The deletion failed because Patient\_ID was referenced in multiple related tables as a foreign key. After investigating, we understood the issue stemmed from referential integrity constraints. To resolve it, we added ON DELETE CASCADE to the appropriate foreign key constraints, ensuring that related records could be automatically removed when a patient was deleted.

Another obstacle involved our initial assumption around using DEFAULT 0 for billing-related numeric fields such as Amount, Treatment\_Cost, and Room\_Cost. We believed this would ensure a 0 value would display where no cost was applied. However, we later realized that in the case of missing joined records (e.g., a patient without a treatment entry), the result was still NULL. After consulting with peers, we adopted the IFNULL() function in our queries to explicitly convert NULL values to 0, particularly for calculated totals and billing summaries.

We also ran into minor complications due to inconsistencies in our dummy data. Some test records included incomplete or mismatched foreign keys, which caused joins or inserts to fail. We resolved these by carefully reviewing and aligning the inserted data with the schema design, ensuring referential integrity and functional query behavior across all modules.

We also encountered challenges in optimizing the structure and performance of complex queries, especially those involving multiple joins and aggregations across several tables such as Patient, Diagnosis, Treatment, Lab, and Bill. Initially, the queries produced either incomplete results or incorrect totals due to join inconsistencies and missing filters. Through trial, error, and iterative debugging, we learned the importance of using functions like IFNULL(), enforcing clean foreign key relationships, and carefully designing views for reusability and clarity. These issues not only enhanced our SQL skills but also deepened our understanding of relational database behavior under real-world complexity.

## **Changes from Project Part 1:**

In Part 2 of the project, we made several key improvements to our database schema based on the lessons learned from implementation and testing. One significant change was modifying certain data types that were initially defined in our ERD. For example, we changed the data type of the Contact\_Number attribute from INT to VARCHAR(20) to allow for special characters such as '+' and '-' in formats like '0307-4298770', which are common in real-world contact details.

We also introduced a variety of **constraints**—including CHECK, DEFAULT, and NOT NULL—which were not emphasized in Part 1 but became essential for ensuring data validation, relational integrity, and meaningful query results during SQL development.

These changes highlighted the importance of testing and refining a database through each development phase. While the ERD served as a strong foundation, working through the SQL scripting process brought new insights that shaped a more functional and realistic implementation.

## **Lessons Learned:**

This project offered a rich, hands-on experience in designing and implementing a relational database for a real-world scenario. We deepened our understanding of data modeling principles, normalization, and the significance of enforcing constraints to ensure data integrity across interconnected tables. From crafting meaningful relationships with foreign keys to designing robust views, each step reinforced the importance of structured thinking and planning ahead.

We also learned how to debug and optimize SQL queries, especially in scenarios involving multiple joins and aggregate functions. The challenge of handling NULL values in billing calculations pushed us to explore functions like IFNULL() and exposed us to the realities of data sparsely in relational design. Collaborative problem-solving and peer feedback proved essential in refining our approach, and experimenting with dummy data highlighted the need for precision when preparing a test dataset.

Overall, this project not only enhanced our technical skills with SQL and database design but also strengthened our ability to think critically, communicate clearly, and translate real-world processes into scalable digital solutions.