**Healthcare Analytics with SQL**

CREATE TABLE Patients (

patient\_id INT PRIMARY KEY,

name VARCHAR(15) NOT NULL,

age INT NOT NULL,

gender VARCHAR(8) NOT NULL,

address VARCHAR(15) NOT NULL,

contact\_number VARCHAR(15) NOT NULL

);

CREATE TABLE DOCTORS (

doctor\_id INT PRIMARY KEY,

name VARCHAR(15) NOT NULL,

specialization VARCHAR(20) NOT NULL,

experience\_years INT CHECK (experience\_years >= 0),

contact\_number VARCHAR(15) UNIQUE NOT NULL

);

CREATE TABLE Appointments (

appointment\_id INT PRIMARY KEY,

patient\_id INT NOT NULL,

doctor\_id INT NOT NULL,

appointment\_date DATE NOT NULL,

reason VARCHAR(50),

status VARCHAR(20) NOT NULL,

FOREIGN KEY (patient\_id) REFERENCES Patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES Doctors(doctor\_id)

);

CREATE TABLE Diagnoses (

diagnosis\_id INT PRIMARY KEY,

patient\_id INT NOT NULL,

doctor\_id INT NOT NULL,

diagnosis\_date DATE NOT NULL,

diagnosis VARCHAR(15),

treatment VARCHAR(20),

FOREIGN KEY (patient\_id) REFERENCES Patients(patient\_id),

FOREIGN KEY (doctor\_id) REFERENCES Doctors(doctor\_id)

);

CREATE TABLE Medications (

medication\_id INT PRIMARY KEY,

diagnosis\_id INT NOT NULL,

medication\_name VARCHAR(50) NOT NULL,

dosage VARCHAR(50) NOT NULL,

start\_date DATE NOT NULL,

end\_date DATE,

FOREIGN KEY (diagnosis\_id) REFERENCES Diagnosis(diagnosis\_id),

CHECK (start\_date <= end\_date)

);

1. **Approach: Inner and Equi Joins**

**Task: Write a query to fetch details of all completed appointments, including the patient’s name, doctor’s name, and specialization. Expected Learning: Demonstrates understanding of Inner Joins and filtering conditions.**

SELECT

p.name AS patient\_name,

d.name AS doctor\_name,

d.specialization,

a.appointment\_date

FROM Appointments a

INNER JOIN Patients p ON a.patient\_id = p.patient\_id

INNER JOIN Doctors d ON a.doctor\_id = d.doctor\_id

WHERE a.status = 'Completed';

1. **Left Join with Null Handling**

**Task: Retrieve all patients who have never had an appointment. Include their name, contact details, and address in the output. Expected Learning: Use of Left Joins and handling NULL values.**

SELECT

p.name AS patient\_name,

p.contact\_number,

p.address

FROM Patients p

LEFT JOIN Appointments a ON p.patient\_id = a.patient\_id

WHERE a.patient\_id IS NULL;

1. **Right Join and Aggregate Functions**

**Task: Find the total number of diagnoses for each doctor, including doctors who haven’t diagnosed any patients. Display the doctor’s name, specialization, and total diagnoses. Expected Learning: Utilization of Right Joins with aggregate functions like COUNT().**

SELECT

d.name AS doctor\_name,

d.specialization,

COUNT(di.diagnosis\_id) AS total\_diagnoses

FROM Doctors d

RIGHT JOIN Diagnoses di ON d.doctor\_id = di.doctor\_id

GROUP BY d.doctor\_id, d.name, d.specialization;

1. **Full Join for Overlapping Data**

**Task: Write a query to identify mismatches between the appointments and diagnoses tables. Include all appointments and diagnoses with their corresponding patient and doctor details.**

**Expected Learning: Handling Full Joins for comparing data across multiple tables.**

**In PostgreSQL, we can directly use FULL OUTER JOIN to retrieve all appointments and diagnoses, including mismatches where there is no corresponding record in either table.**

SELECT

COALESCE(a.appointment\_id, d.diagnosis\_id) AS record\_id,

p.name AS patient\_name,

doc.name AS doctor\_name,

doc.specialization,

a.appointment\_date,

d.diagnosis,

d.treatment

FROM Appointments a

FULL OUTER JOIN Diagnoses d

ON a.patient\_id = d.patient\_id

AND a.doctor\_id = d.doctor\_id

LEFT JOIN Patients p

ON COALESCE(a.patient\_id, d.patient\_id) = p.patient\_id

LEFT JOIN Doctors doc

ON COALESCE(a.doctor\_id, d.doctor\_id) = doc.doctor\_id

WHERE a.appointment\_id IS NULL OR d.diagnosis\_id IS NULL;

1. **Window Functions (Ranking and Aggregation)**

**Task: For each doctor, rank their patients based on the number of appointments in descending order. Expected Learning: Application of Ranking Functions such as RANK() or DENSE\_RANK().**

SELECT

d.Doctor\_ID,

d.name,

COUNT(a.Appointment\_ID) AS Appointment\_Count,

RANK() OVER (ORDER BY COUNT(a.Appointment\_ID) DESC) AS Patient\_Rank

FROM Appointments a

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

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

GROUP BY d.Doctor\_ID, d.name;

1. **Conditional Expressions**

**Task: Write a query to categorize patients by age group (e.g., 18-30, 31-50, 51+). Count the number of patients in each age group. Expected Learning: Using CASE statements for conditional logic.**

SELECT

CASE

WHEN age BETWEEN 18 AND 30 THEN '18-30'

WHEN age BETWEEN 31 AND 50 THEN '31-50'

WHEN age >= 51 THEN '51+'

ELSE 'Unknown'

END AS age\_group,

COUNT(\*) AS patient\_count

FROM patients

GROUP BY age\_group

ORDER BY age\_group;

1. **Numeric and String Functions**

**Task: Retrieve a list of patients whose contact numbers end with "1234" and display their names in uppercase. Expected Learning: Use of string functions like UPPER () and LIKE.**

SELECT UPPER (name) AS patient\_name, contact\_number

FROM patients

WHERE contact\_number LIKE '%1234';

1. **Subqueries for Filtering**

**Task: Find patients who have only been prescribed "Insulin" in any of their diagnoses. Expected Learning: Writing Subqueries for advanced filtering.**

SELECT DISTINCT a.Patient\_ID

FROM appointments a

JOIN diagnoses d ON a.patient\_ID = d.patient\_ID

JOIN medications m ON d.Diagnosis\_ID = m.Diagnosis\_ID

WHERE a.Patient\_ID NOT IN (

SELECT DISTINCT a2.Patient\_ID

FROM appointments a2

JOIN diagnoses d2 ON a2.patient\_ID = d2.patient\_ID

JOIN medications m2 ON d2.Diagnosis\_ID = m2.Diagnosis\_ID

WHERE m2.Medication\_Name <> 'Insulin'

);

1. **Date and Time Functions**

**Task: Calculate the average duration (in days) for which medications are prescribed for each diagnosis. Expected Learning: Working with date functions like DATEDIFF().**

SELECT

d.Diagnosis\_ID,

d.Diagnosis,

Start\_Date,

End\_Date,

AVG(ABS(m.End\_Date - m.Start\_Date)) AS Avg\_Duration\_Days

FROM diagnoses d

JOIN medications m ON d.Diagnosis\_ID = m.Diagnosis\_ID

GROUP BY d.Diagnosis\_ID, d.Diagnosis, Start\_Date, End\_Date;

1. **Complex Joins and Aggregation**

**Task: Write a query to identify the doctor who has attended the most unique patients. Include the doctor’s name, specialization, and the count of unique patients. Expected Learning: Combining Joins, Grouping, and COUNT(DISTINCT).**

SELECT d.name AS doctor\_name,

d.specialization,

COUNT(DISTINCT a.patient\_id) AS unique\_patient\_count

FROM doctors d

JOIN appointments a ON d.doctor\_id = a.doctor\_id

GROUP BY d.doctor\_id, d.name, d.specialization

ORDER BY unique\_patient\_count DESC

LIMIT 1;