# Healthcare Analytics with SQL

## Database Schema:

```
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)
);
```

# Approach:

1. **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';
```

2. **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;
```

3. **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;
```

4. **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;
```

5. **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;
```

6. **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;
```

7. **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';
```

8. **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'
);
```

9. **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;
```

10. **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;
```