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.status

FROM

Appointments a

JOIN

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

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.appointment\_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

diagnoses di

RIGHT JOIN

doctors d ON di.doctor\_id = d.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.

SELECT

a.appointment\_id,

a.appointment\_date,

a.status,

a.reason,

di.diagnosis\_id,

di.diagnosis\_date,

di.diagnosis,

di.treatment,

p.name AS patient\_name,

d.name AS doctor\_name,

d.specialization

FROM

diagnoses di

LEFT JOIN

appointments a ON di.patient\_id = a.patient\_id AND di.doctor\_id = a.doctor\_id

LEFT JOIN

patients p ON di.patient\_id = p.patient\_id

LEFT JOIN

doctors d ON di.doctor\_id = d.doctor\_id;

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

doctor\_id,

patient\_id,

COUNT(\*) AS total\_appointments,

RANK() OVER (PARTITION BY doctor\_id ORDER BY COUNT(\*) DESC) AS appointment\_rank,

DENSE\_RANK() OVER (PARTITION BY doctor\_id ORDER BY COUNT(\*) DESC) AS appointment\_dense\_rank

FROM

appointments

GROUP BY

doctor\_id,

patient\_id;

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 total\_patients

FROM

patients

GROUP 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\_uppercase,

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

p.patient\_id,

p.name AS patient\_name

FROM

patients p

WHERE

-- Patient has at least one 'Insulin' medication

EXISTS (

SELECT 1

FROM diagnoses d

JOIN medications m ON d.diagnosis\_id = m.diagnosis\_id

WHERE d.patient\_id = p.patient\_id

AND m.medication\_name = 'Insulin'

)

-- Patient does not have any other medication

AND NOT EXISTS (

SELECT 1

FROM diagnoses d

JOIN medications m ON d.diagnosis\_id = m.diagnosis\_id

WHERE d.patient\_id = p.patient\_id

AND m.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

diagnosis\_id,

AVG(DATEDIFF(end\_date, start\_date)) AS avg\_prescription\_duration

FROM

medications

GROUP BY

diagnosis\_id;

SELECT

diagnosis\_id,

AVG(DATEDIFF(end\_date, start\_date)) AS avg\_prescription\_duration

FROM

medications

WHERE

DATEDIFF(end\_date, start\_date) >= 0

GROUP BY

diagnosis\_id;

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.doctor\_id,

d.name AS doctor\_name,

d.specialization,

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

FROM

appointments a

JOIN

doctors d ON a.doctor\_id = d.doctor\_id

GROUP BY

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

ORDER BY

unique\_patient\_count DESC

LIMIT 1;

Create a relationship between each tables using their Primary Key & Foreign Key

Then created an ER Diagram