|  |  |
| --- | --- |
| **S.no** | **Tasks and Queries** |
| **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.  **Query:**  select patients.name as patients\_name,doctors.name as doctors\_name,specialization from patients  **inner join** appointments on patients.patient\_id=appointments.patient\_id  **inner join** doctors on appointments.doctor\_id=doctors.doctor\_id  **where** 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.  **Query:** select name as patients\_name,contact\_number,address from patients **left join** appointments on patients.patient\_id=appointments.patient\_id where appointments.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().  **Query:**  select distinct name as doctors\_name,specialization,**count**(diagnosis) as Total\_diagnosis from diagnoses  **right join** doctors on diagnoses.doctor\_id=doctors.doctor\_id  **group by** doctors\_name,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.  **Query:** select appointment\_id,appointments.Patient\_id,appointments.doctor\_id, null as diagnosis\_id, null as diagnosis from appointments  left join patients on appointments.Patient\_id=patients.Patient\_id  left join doctors on appointments.doctor\_id=doctors.doctor\_id  left join diagnoses on appointments.Patient\_id=diagnoses.Patient\_id and appointments.doctor\_id=diagnoses.doctor\_id  where diagnoses.diagnosis\_id is null  **union**  select null as appointment\_id,diagnoses.Patient\_id,diagnoses.doctor\_id, diagnoses.diagnosis\_id,diagnoses.diagnosis from diagnoses  left join patients on diagnoses.Patient\_id=patients.Patient\_id  left join doctors on diagnoses.doctor\_id=doctors.doctor\_id  left join appointments on diagnoses.Patient\_id=appointments.Patient\_id and diagnoses.doctor\_id=appointments.doctor\_id  where appointments.appointment\_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().  **Query:**  select doctor\_id,patient\_id,count(patient\_id) as Total\_appointments,**rank**()  6over (order by count(patient\_id) **desc**) as Patient\_ranks from appointments  group by doctor\_id,patient\_id; |
| **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.  **Query:**  select  **case**  when age >='18' **and** age <='30' then '18\_30'  when age >='31' **and** age <='50' then '31\_50'  when age >='51' **and** age <='70' then '51\_70'  when age >='71' **and** age <='90' then '71\_90'  **else** 'no\_group'  **end** as age\_group, **count**(patient\_id) as Total\_patients\_count  from patients **group** 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.  **Query:**  select **upper**(name) as Patients\_name 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.  **Query: select** distinct patients.Patient\_id from patients where patients.Patient\_id in (**select** distinct diagnoses.Patient\_id from diagnoses  join medications on diagnoses.diagnosis\_id=medications.diagnosis\_id where medication\_name = 'Insulin')  and patients.Patient\_id not in  (**select** distinct diagnoses.Patient\_id from diagnoses join medications on diagnoses.diagnosis\_id=medications.diagnosis\_id  where 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().  **Query:**  Select diagnoses.diagnosis\_id,diagnosis,medication\_name,avg(**datediff**(end\_date,start\_date)) as average\_duration\_days from medications  join diagnoses on medications.diagnosis\_id=diagnoses.diagnosis\_id  group by diagnoses.diagnosis\_id,medication\_name; |
| **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).  **Query:**  select name as doctors\_name,specialization,**count**(distinct patient\_id) as unique\_patients\_count from doctors  inner **join** appointments on doctors.doctor\_id=appointments.doctor\_id  **group** by doctors\_name,specialization; |