2025

Healthcare Analytics with SQL

REPORT DOCUMENT

VIJAYAPERUMAL SOUNDRAPANDIAN

Report Document

Problem Statement:

The project aims to analyze healthcare data, focusing on extracting meaningful insights about patients, doctors, appointments, diagnoses, and treatments using advanced SQL techniques. Learners will apply SQL operations like joins, subqueries, window functions, and more to analyze healthcare metrics.

Business Use Cases:

- **Patient Management**: Analyzing patient demographics, their medical histories, and appointment trends.
- **Doctor Performance Evaluation**: Measuring doctor efficiency by tracking diagnoses and treatment frequency.
- **Appointment Scheduling & Completion**: Understanding appointment trends, cancellations, and completions.
- **Medication Analysis**: Identifying the most common medications prescribed to patients based on diagnoses.
- **Revenue and Billing Analysis**: Linking appointment data with billing information to understand revenue generation.

Inner and Equi Joins

Task – 1: 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 a.appointment_id, p.name, d.name, d.specialization 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 status = 'completed';
```

- INNER JOIN returns records only when there is a match in both tables. By joining the appointments, patients, and doctor's tables, I was able to show only those appointments that were marked as 'Completed' and link them to relevant doctor and patient information.

Left Join with Null Handling

Task – 2: 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,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;
```

- I learned that a LEFT JOIN helps identify unmatched data. In this case, I used it to find patients who exist in the patients table but don't have any records in the appointments table. This is useful for identifying inactive patients or outreach opportunities.

Right Join and Aggregate Functions

Task - 3: 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 di.doctor_id, d.name, d.specialization, count(di.diagnosis) as total_diagnosis from doctors d
right join diagnoses di on d.doctor_id = di.doctor_id
group by di.doctor_id, d.name, d.specialization
order by di.doctor_id;
```

- I used a RIGHT JOIN to list all doctors, including those who haven't diagnosed any patients. Then, I applied COUNT() to find the total number of diagnoses per doctor. This taught me how to perform aggregation with joins, and how RIGHT JOIN ensures we include all doctors, even those without data in the joined table.

Full Join for Overlapping Data

Task - 4: 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 d.doctor_id, d.name, p.patient_id, p.name, 'appointment only' as source from appointments a
 left join diagnoses di on a.doctor_id = di.doctor_id and a.patient_id = di.patient_id
 join doctors d on d.doctor_id = a.doctor_id
 join patients p on p.patient_id = a.patient_id
 where di.diagnosis_id is null
 union
select d.doctor_id, d.name, p.patient_id, p.name, 'diagnoses only' as source from diagnoses di
 left join appointments a on di.doctor_id = a.doctor_id and di.patient_id = a.patient_id
 join doctors d on di.doctor_id = d.doctor_id
 join patients p on di.patient_id = p.patient_id
 where a.appointment_id is null;
```

- I learned how FULL OUTER JOIN can be used to compare records between two tables and identify mismatches. I applied this to find differences between appointments and diagnoses, ensuring no data is missed from either table. This is useful for **data integrity checks and audits**.

Window Functions (Ranking and Aggregation)

Task - 5: 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, total_appointments, rank() over( partition by doctor_id order by total_appointments desc)
 as rank_per_doctor
 from (select doctor_id, patient_id, count(*) as total_appointments
from appointments
group by doctor_id, patient_id)
 as appoint_summary
 order by doctor_id, rank_per_doctor;
```

- I understood how window functions work over a "window" or group. By using PARTITION BY doctor_id, I calculated the rank of each patient based on appointment frequency per doctor, which helps in identifying most frequently visiting patients for each doctor.

Conditional Expressions

Task - 6: 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 >= 18 and age <= 30 then '18-30'
when age >= 31 and age <= 50 then '31-50'
else '51+'
end as age_group
from patients;
```

- I used a CASE statement to group patients into age categories like 18–30, 31–50, and 51+. Then I counted the number of patients in each group. This task taught me how to use conditional logic in SQL for data classification and summarization.

Numeric and String Functions

Task - 7: 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), contact_number from patients
where contact_number like '%1234';
```

- I practiced using string functions like UPPER() and LIKE to filter patients whose contact numbers end in "1234" and convert their names to uppercase. This helped me manipulate text fields and apply **pattern matching**.

Subqueries for Filtering

Task - 8: 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 from patients p
where p.patient_id in (
select di.patient_id from diagnoses di
join medications m on di.diagnosis_id = m.diagnosis_id
group by di.patient_id
having count(distinct case when m.medication_name <> 'insulin' then m.medication_name end) = 0
and count(*) >0);
```

- This task taught me how to use a subquery to filter patients based on complex logic. By counting non-'Insulin' medications using a CASE inside a HAVING clause, I was able to find patients whose only medication was Insulin.

Date and Time Functions

Task - 9: Calculate the average duration (in days) for which medications are prescribed for each diagnosis.

Expected Learning: Working with date functions like DATEDIFF().

```
select m.diagnosis_id, avg(datediff(m.end_date, m.start_date)) as avg_duration_days from medications m
where start_date is not null and end_date is not null
group by m.diagnosis_id;
```

- I used the DATEDIFF() function to compute how long each medication was prescribed. Then I grouped the data by diagnosis to get the average treatment duration. This helps understand treatment timelines.

Complex Joins and Aggregation

Task - 10: 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 doctors d
left 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;
```

- I combined multiple tables using JOIN, grouped by doctor, and applied COUNT(DISTINCT patient_id) to find the doctor with the **most unique patients treated**. This helped me understand how to perform **advanced joins and aggregations** for business reporting.