

# HEALTHCARE ANALYTICS SQL PROJECT REPORT

---

## PROJECT OVERVIEW

This SQL project focuses on analyzing healthcare datasets to understand patient activity, doctor performance, appointment completion, diagnoses, and medication usage. Using SQL joins, window functions, subqueries, and aggregations, I explored how hospitals can derive insights from structured data to improve operations and patient care.

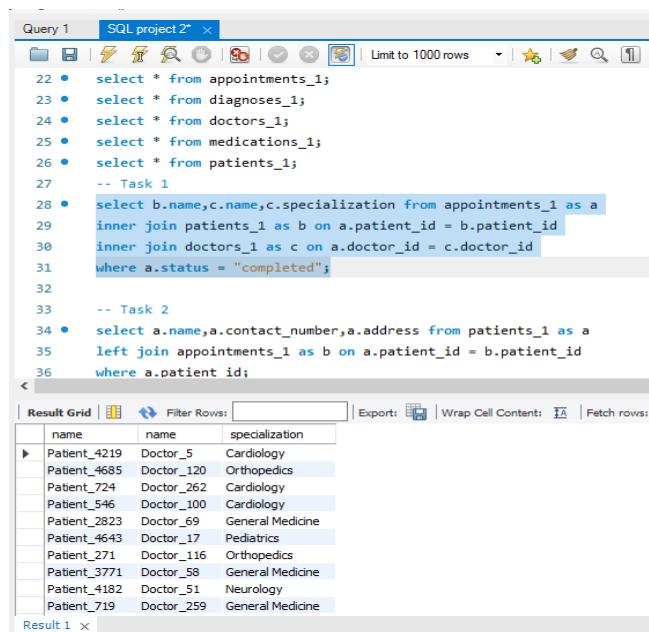
## PROBLEM STATEMENT

The objective of this project is to use SQL to analyze core healthcare operations. The tasks include tracking appointment status, identifying inactive patients, measuring doctor workload, detecting mismatched medical records, categorizing patients, evaluating medication duration, and identifying diagnosis patterns. The project aims to enhance understanding of healthcare data through SQL-based exploration

## TASK EXPLANATION

### 1. Completed Appointments:

Using INNER JOIN, I retrieved the details of all completed appointments with patient and doctor information. This helps understand successful appointment patterns and doctor involvement.



The screenshot shows a SQL Server Management Studio (SSMS) interface. The top pane is titled "Query 1" and contains the following SQL code:

```
Query 1  SQL project 2* 
22 •    select * from appointments_1;
23 •    select * from diagnoses_1;
24 •    select * from doctors_1;
25 •    select * from medications_1;
26 •    select * from patients_1;
27 -- Task 1
28 •    select b.name,c.name,c.specialization from appointments_1 as a
29     inner join patients_1 as b on a.patient_id = b.patient_id
30     inner join doctors_1 as c on a.doctor_id = c.doctor_id
31     where a.status = "completed";
32
33 -- Task 2
34 •    select a.name,a.contact_number,a.address from patients_1 as a
35     left join appointments_1 as b on a.patient_id = b.patient_id
36     where a.patient_id;
```

The bottom pane is titled "Result Grid" and displays the results of the query. The results are presented in a table with three columns: name, name, and specialization. The data is as follows:

	name	name	specialization
▶	Patient_4219	Doctor_5	Cardiology
	Patient_4685	Doctor_120	Orthopedics
	Patient_724	Doctor_262	Cardiology
	Patient_546	Doctor_100	Cardiology
	Patient_2823	Doctor_69	General Medicine
	Patient_4643	Doctor_17	Pediatrics
	Patient_271	Doctor_116	Orthopedics
	Patient_3771	Doctor_58	General Medicine
	Patient_4182	Doctor_51	Neurology
	Patient_719	Doctor_259	General Medicine

### 2. Patients Without Appointments:

Using LEFT JOIN, I extracted patients who never booked any appointment. This helps identify inactive patients and improves patient outreach strategies.

Query 1 SQL project 2\* ×

```

31     where a.status = "completed";
32
33     -- Task 2
34 •   select a.name,a.contact_number,a.address from patients_1 as a
35     left join appointments_1 as b on a.patient_id = b.patient_id
36     where a.patient_id;
37
38     -- Task 3
39 •   select a.name,a.specialization,sum(b.diagnosis_id) as total_diagnoses
40     right join diagnoses_1 as b on a.doctor_id = b.doctor_id
41     group by a.name,a.specialization;
42
43     -- Task 4
44 •   select * from appointments_1 as a
45     inner join diagnoses_1 as b on a.doctor_id = b.doctor_id

```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows

	name	contact_number	address
▶	Patient_1	98765430001	Address_1
	Patient_1	98765430001	Address_1
	Patient_10	98765430010	Address_10
	Patient_100	98765430100	Address_100
	Patient_100	98765430100	Address_100
	Patient_1000	98765431000	Address_1000
	Patient_1001	98765431001	Address_1001

Result 2 ×

### 3. Diagnoses Count per Doctor:

Using RIGHT JOIN and COUNT(), I calculated the total diagnoses for each doctor. This shows doctor efficiency and workload.

Query 1 SQL-project 2\* ×

```

31     where a.status = "completed";
32
33     -- Task 2
34 •   select a.name,a.contact_number,a.address from patients_1 as a
35     left join appointments_1 as b on a.patient_id = b.patient_id
36     where a.patient_id;
37
38     -- Task 3
39 •   select a.name,a.specialization,sum(b.diagnosis_id) as total_diagnoses from doctors_1 as a
40     right join diagnoses_1 as b on a.doctor_id = b.doctor_id
41     group by a.name,a.specialization;
42
43     -- Task 4
44 •   select * from appointments_1 as a
45     inner join diagnoses_1 as b on a.doctor_id = b.doctor_id

```

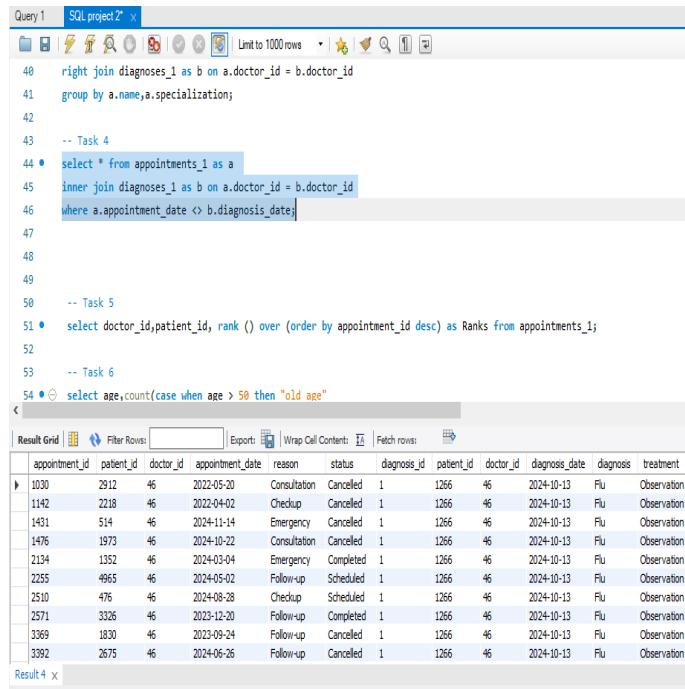
Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows

	name	specialization	total_diagnosis
▶	Doctor_1	Orthopedics	399418
	Doctor_10	General Medicine	407346
	Doctor_100	Cardiology	386061
	Doctor_101	General Medicine	493283
	Doctor_102	Pediatrics	372880
	Doctor_103	General Medicine	351219
	Doctor_104	Cardiology	411802
	Doctor_105	Orthopedics	439843
	Doctor_106	Pediatrics	372607
	Doctor_107	Neurology	388545

Result 3 ×

## 4. Appointment–Diagnosis Mismatch:

I compared appointment and diagnosis dates to identify mismatches. This helps detect data inconsistency and improves record accuracy.

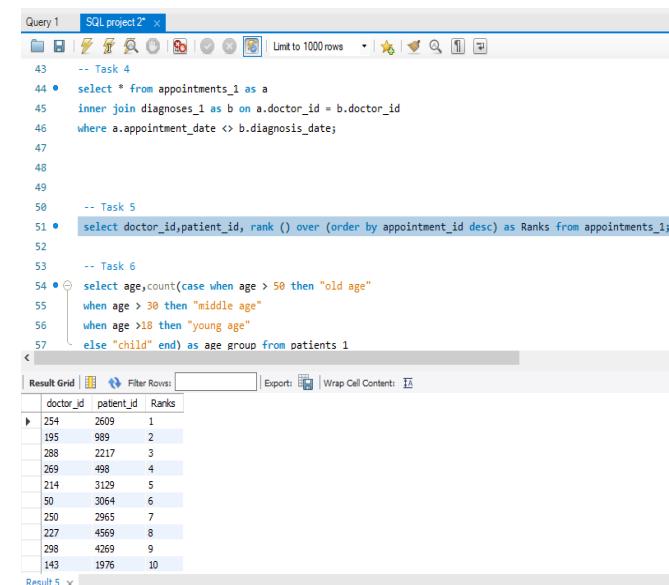


The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query editor contains several lines of SQL code, including joins and a WHERE clause for comparing appointment and diagnosis dates. The results grid displays a table with columns: appointment\_id, patient\_id, doctor\_id, appointment\_date, reason, status, diagnosis\_id, patient\_id, doctor\_id, diagnosis\_date, diagnosis, and treatment. The data shows various patient appointments and their corresponding diagnoses, with some rows having different dates.

appointment_id	patient_id	doctor_id	appointment_date	reason	status	diagnosis_id	patient_id	doctor_id	diagnosis_date	diagnosis	treatment
1030	2912	46	2022-05-20	Consultation	Cancelled	1	1266	46	2024-10-13	Flu	Observation
1142	2218	46	2022-04-02	Checkup	Cancelled	1	1266	46	2024-10-13	Flu	Observation
1451	514	46	2024-11-14	Emergency	Canceled	1	1266	46	2024-10-13	Flu	Observation
1476	1973	46	2024-10-22	Consultation	Cancelled	1	1266	46	2024-10-13	Flu	Observation
2134	1352	46	2024-03-04	Emergency	Completed	1	1266	46	2024-10-13	Flu	Observation
2255	4965	46	2024-05-02	Follow-up	Scheduled	1	1266	46	2024-10-13	Flu	Observation
2510	476	46	2024-08-28	Checkup	Scheduled	1	1266	46	2024-10-13	Flu	Observation
2571	3326	46	2023-12-20	Follow-up	Completed	1	1266	46	2024-10-13	Flu	Observation
3369	1830	46	2023-09-24	Follow-up	Cancelled	1	1266	46	2024-10-13	Flu	Observation
3392	2675	46	2024-06-26	Follow-up	Cancelled	1	1266	46	2024-10-13	Flu	Observation

## 5. Patient Ranking by Doctor:

Using RANK() window function, I ranked patients based on appointment frequency. This helps identify highly active patients.



The screenshot shows a SQL Server Management Studio window with a query editor and a results grid. The query editor contains SQL code for ranking patients by appointment frequency and classifying them into age groups. The results grid displays a table with columns: doctor\_id, patient\_id, and Ranks. The data shows patient counts for each doctor, with ranks assigned based on the count.

doctor_id	patient_id	Ranks
254	2609	1
195	989	2
288	2217	3
269	498	4
214	3129	5
50	3064	6
250	2965	7
227	4569	8
298	4269	9
143	1976	10

## 6. Patient Age Group Categorization:

Using CASE, I categorized patients into age groups such as child, young, middle-age, and old-age. This supports demographic analysis.

The screenshot shows a SQL Server Management Studio (SSMS) interface. The top window is titled "Query 1" and contains a block of SQL code. The code includes several comments (--) and numbered tasks (52 through 66). Task 6 contains a CASE statement for age group categorization. Task 7 and Task 8 are also visible. The bottom window is titled "Result Grid" and displays the output of the query. It shows two columns: "age" and "age\_group". The data consists of pairs of age values (e.g., 69, 59; 19, 67; etc.) and their corresponding age groups (e.g., "old age", "middle age", "young age", "child").

age	age_group
69	59
19	67
61	62
67	62
24	55
25	82
82	60
74	70
84	67
76	84

## 7. Contact Number Matching:

Using LIKE and UPPER(), I retrieved patients whose contact numbers end with '1234'. This shows usage of string filters.

The screenshot shows a SQL Server Management Studio (SSMS) interface. The top window is titled "Query 1" and contains a block of SQL code. The code includes several comments (--) and numbered tasks (55 through 69). Task 7 contains a query using the WHERE clause with a LIKE filter to find contact numbers ending in '1234'. Task 8 and Task 9 are also visible. The bottom window is titled "Result Grid" and displays the output of the query. It shows two columns: "Name" and "contact\_number". The data shows a single row where the name is "PATIENT\_1234" and the contact number is "98765431234".

Name	contact_number
PATIENT_1234	98765431234

## 8. Patients Prescribed Insulin:

Using a subquery, I identified patients diagnosed with Insulin-only prescriptions. This helps identify diabetic patients.

```
Query 1 SQL project 2* x
55     when age > 30 then "middle age"
56     when age >18 then "young age"
57     else "child" end) as age_group from patients_1
58     group by age;
59
60     -- Task 7
61 •     select upper(name) as Name,contact_number from patients_1
62     where contact_number like '%1234' ;
63
64
65     -- Task 8
66 •     select name from patients_1
67     where patient_id in (select patient_id from diagnoses_1    where diagnosis = "insulin");
68
69     -- Task 9
```

The screenshot shows a SQL query editor window titled "Query 1". The code is a script with several tasks numbered 55 through 69. Task 8 is highlighted with a blue selection bar around its lines. The result grid below shows a single column named "name" with one row containing the value "name". The status bar at the bottom indicates "patients\_1 8 x".

## 9. Average Medication Duration:

I used DATEDIFF() to calculate the average days each medication was prescribed. This helps analyze treatment patterns.

```
Query 1 SQL project 2* x
57     else "child" end) as age_group from patients_1
58     group by age;
59
60     -- Task 7
61 •     select upper(name) as Name,contact_number from patients_1
62     where contact_number like '%1234' ;
63
64
65     -- Task 8
66 •     select name from patients_1
67     where patient_id in (select patient_id from diagnoses_1    where diagnosis = "insulin");
68
69     -- Task 9
70 •     select medication_name,avg(datediff(end_date,start_date)) as Avg_date from medications_1
71     group by medication name;
```

medication_name	Avg_date
Painkillers	4.0917
Insulin	8.9640
Paracetamol	5.2965
Antidepressants	-5.7164
Antibiotics	5.9341

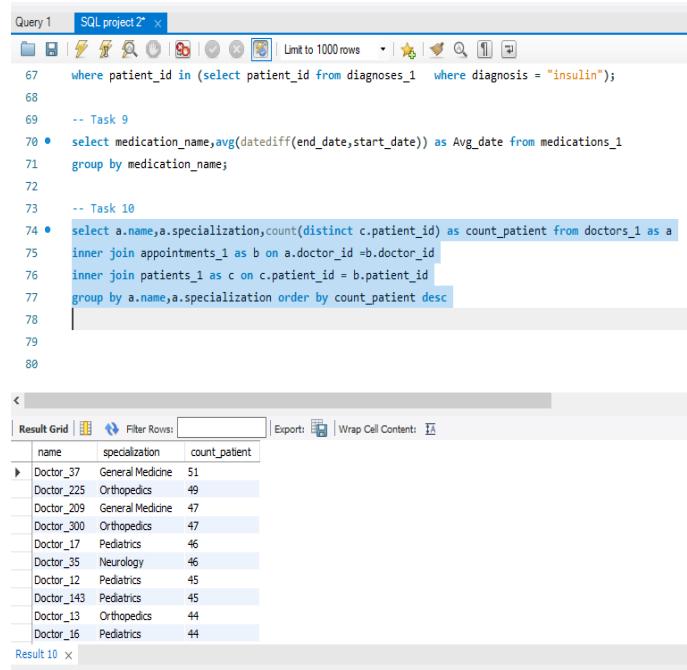
The screenshot shows a SQL query editor window titled "Query 1". The code is identical to the previous screenshot but includes Task 9. Task 9 is highlighted with a blue selection bar. The result grid below shows a table with two columns: "medication\_name" and "Avg\_date". The data is as follows:

medication_name	Avg_date
Painkillers	4.0917
Insulin	8.9640
Paracetamol	5.2965
Antidepressants	-5.7164
Antibiotics	5.9341

The status bar at the bottom indicates "Result 9 x".

## 10. Doctor With Most Unique Patients:

Using COUNT(DISTINCT) with joins, I found the doctor who attended the highest number of unique patients. This identifies top-performing doctors.



The screenshot shows a SQL Server Management Studio interface. The top window is titled "Query 1" and contains a SQL script. The bottom window is titled "Result Grid" and displays the query results as a table.

```
Query 1  SQL project 2* x
67     where patient_id in (select patient_id from diagnoses_1  where diagnosis = "insulin");
68
69     -- Task 9
70 •   select medication_name,avg(datediff(end_date,start_date)) as Avg_date from medications_1
71     group by medication_name;
72
73     -- Task 10
74 •   select a.name,a.specialization,count(distinct c.patient_id) as count_patient from doctors_1 as a
75     inner join appointments_1 as b on a.doctor_id = b.doctor_id
76     inner join patients_1 as c on c.patient_id = b.patient_id
77     group by a.name,a.specialization order by count_patient desc
78
79
80
```

name	specialization	count_patient
Doctor_37	General Medicine	51
Doctor_225	Orthopedics	49
Doctor_209	General Medicine	47
Doctor_300	Orthopedics	47
Doctor_17	Pediatrics	46
Doctor_35	Neurology	46
Doctor_12	Pediatrics	45
Doctor_143	Pediatrics	45
Doctor_13	Orthopedics	44
Doctor_16	Pediatrics	44

## CONCLUSION

This healthcare SQL project provided a strong understanding of data relationships in hospital systems. The tasks helped identify doctor performance, appointment patterns, patient demographics, medication usage, and data inaccuracies. Through this project, I strengthened my SQL skills and gained the ability to perform real-world operational analysis using structured healthcare data.