# Task 1 (Part 2): Implementation

## Part 2 Postgres SQL Database Implementation

**By David J Kelley**

(David@AGILaboratory.com)(dkelle24@wgu.edu)

### Introduction

This part will include implementation of a database including the SQL DDL for Postgres SQL and associated screen shots. Use the database name “D597 Task 1” based the Part 1 design. Write an import script. Write three queries to retrieve specific data from the database providing screen shots of the queries running with results. Optimize queries and show screen shots with that being successful.

### Part 2.Task F.1 – Write DB DDL:

The “*healthtrackdb*” PostgreSQL database schema used here is designed to support a comprehensive healthcare data platform, facilitating the storage and organization of structured medical, administrative, and operational information. It includes reference, data, relational, and metadata tables that model core entities such as patients, providers, appointments, alerts, electronic health records (EHR), predictive analytics, metrics, and reported outcomes. The schema distinguishes between reference types (e.g., *metric\_type,* *document\_type*), data tables (e.g., appointments, *ehr*, *predictive\_analytics),* and relationship tables (e.g*., patient\_provider,* *provider\_contact\_values*) to ensure modularity and normalization. It supports auditing and historical tracking through timestamped fields, soft deletion flags (*is\_deleted*), and metadata columns like *created\_by* and *last\_modified\_by*. UUIDs and identity-based primary keys are used strategically, and *pgcrypto* is enabled for UUID generation. The schema also accommodates extensibility for high-volume logging (*security\_log*), XML-based predictive analytics, and fine-grained tracking of credentials, locations, and insurance relationships. Overall, the design reflects a robust and scalable architecture for a modern healthcare information system.

See the attached document: < “D597 Task 1 DDL\_PostGRESConversion.sql” >

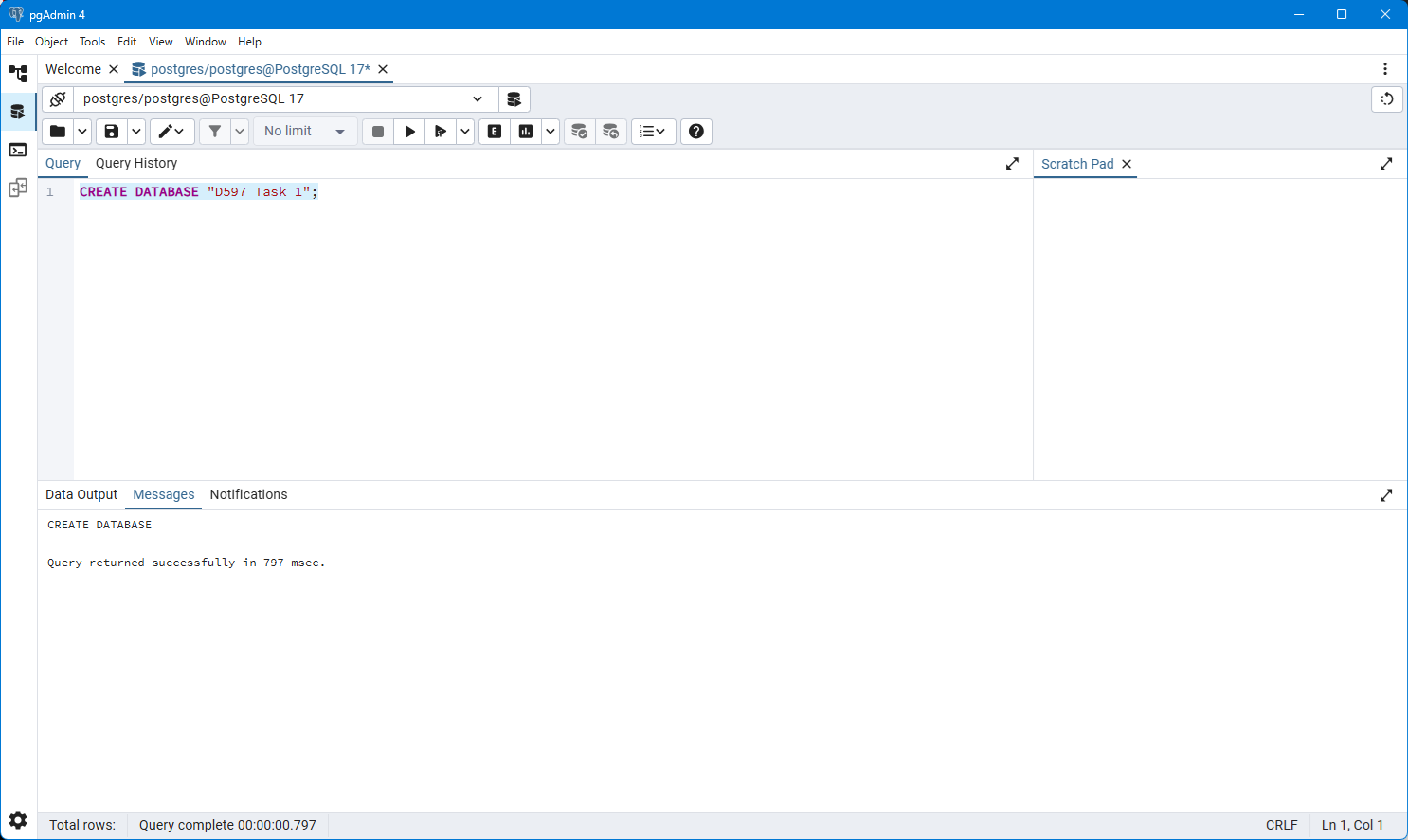


Figure 1A – Creating DB for Task 1 Part 2 F.1 Screen Shot. See attached file: < “D507 Task 1 – Part 2\_create db.png” >

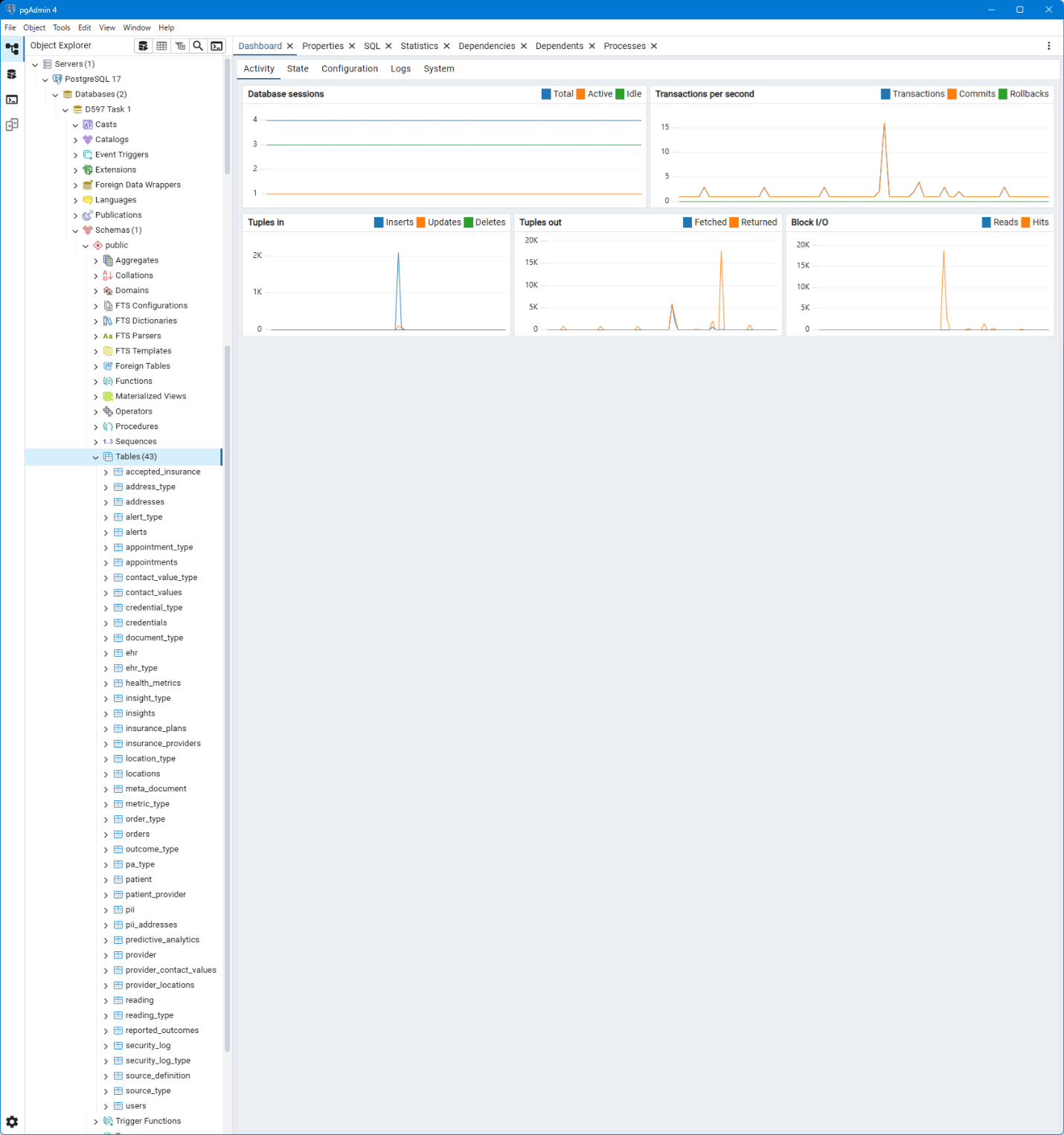


Figure 1B – Show all the tables created in the specific instance “D597 Task 1” from the created DDL See attached file: < “D507 Task 1 – Part 2\_create tables in instance.png” >

### Part 2.Task F.2 – Import Data Records

Given that I choose Scenario 1 for my project and the task doesn’t specifically say SQL script and I was not able to guess the password for the PostGres install in the VM using pgAdmin 4 I bruit forced it and installed a fresh copy and I used a C# script to import the data because in pgAdmin 4 you can’t directly import a CSV file from the SQL view so this was faster given my current skills.

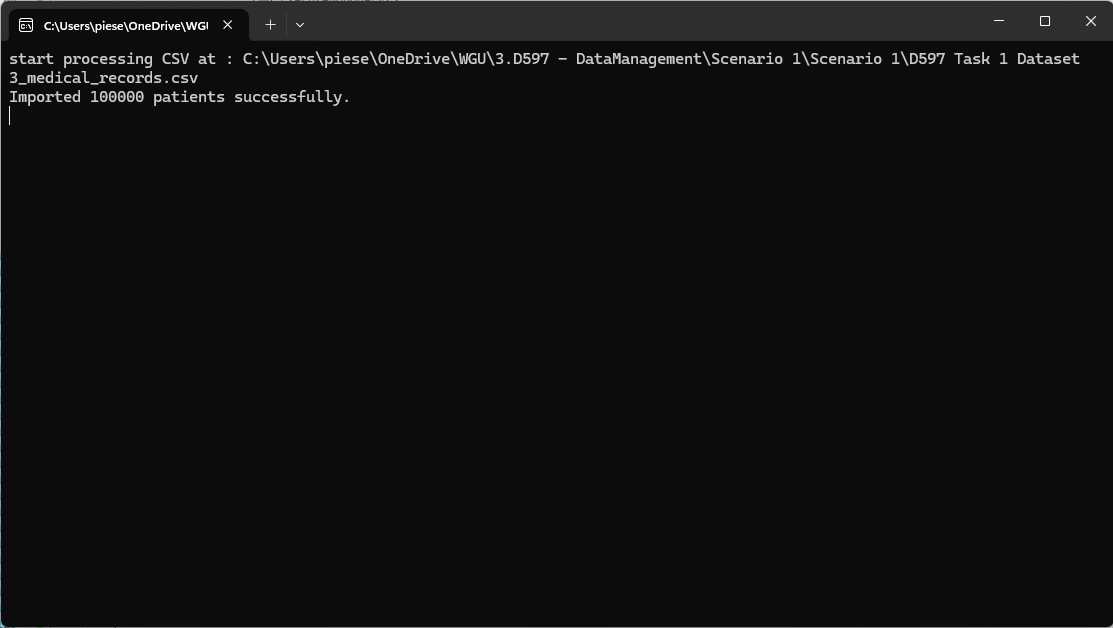
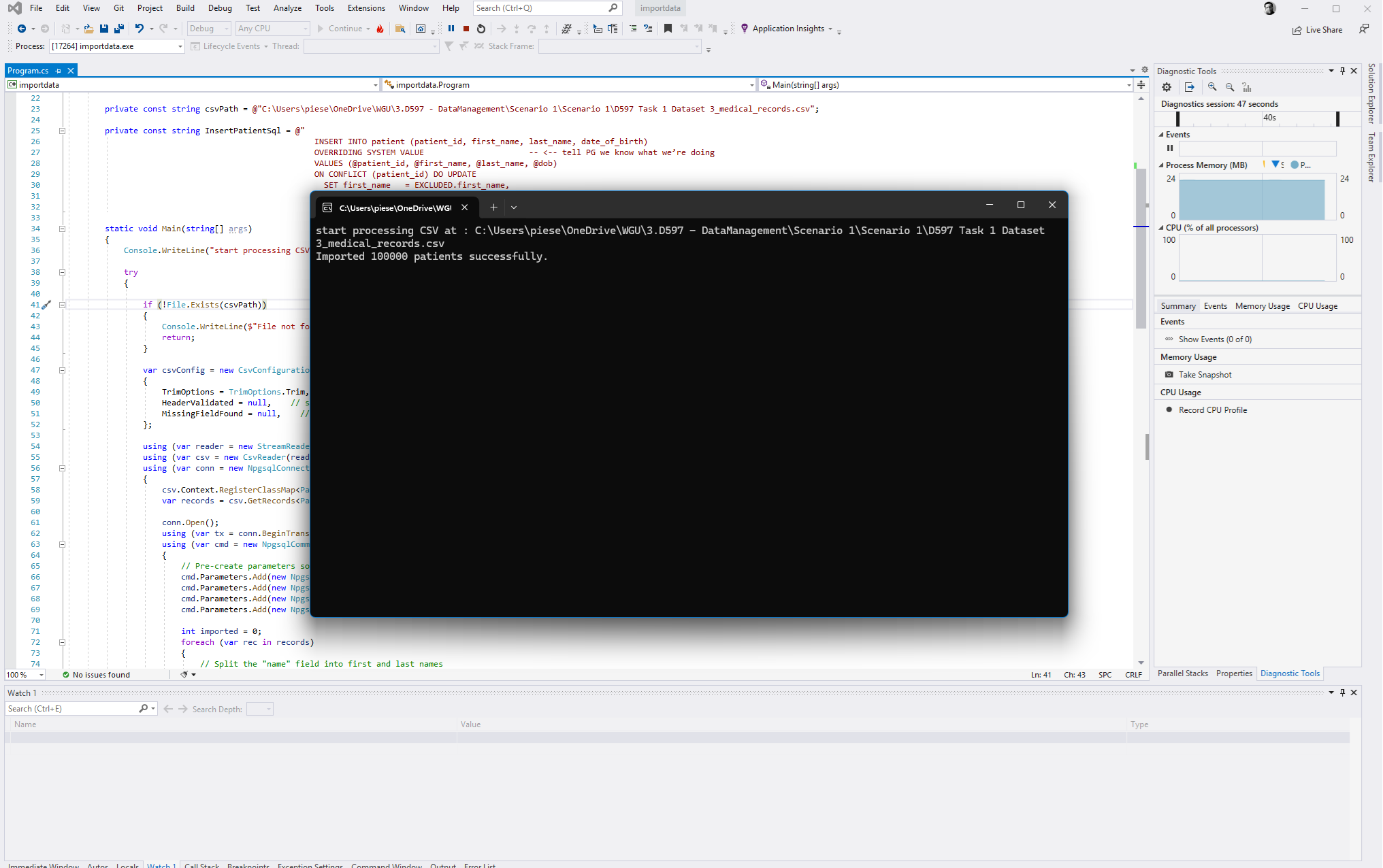


Figure 2A – this is the C# script running (strictly speaking a console app) See attached file: < D597 Task 1 – Part 2\_F.2 importing.png >   
  
  
Figure 2B – this shows the script running in VS with the output also visible. See attached file: <D597 Task 1 – Part 2\_Running in VS.png > also See the actual code here in attached file: < D597 Task 1 – Part 2\_F.2 import script.cs>

A screenshot of a computer

AI-generated content may be incorrect.

Figure 2C - This screen shot shows a query against the patients table which shows all of the records from the CSV file imported in correctly. See attached file: < D597 Task 1 - Part 2\_F2\_checking table after import.png >

### Part 2.Task F.3 – Example Queries

Given that the assignment solution I have is relatively complex I did create some test data to run some more complex queries. That additional test data is included [ D597 Task 1 - Part 2\_F3 Extended Patient Test Data.sql ] in the attachments and is important to show various relationship between elements of the database and show casing functionality as it relates to the business case.

**Patients by device**

SELECT

p.patient\_id,

p.first\_name,

p.last\_name,

p.date\_of\_birth,

sd.source\_name,

st.source\_type\_name,

ps.associated\_date

FROM

patient\_source ps

JOIN

patient p ON ps.patient\_id = p.patient\_id

JOIN

source\_definition sd ON ps.source\_id = sd.source\_id

JOIN

source\_type st ON sd.source\_type\_id = st.source\_type\_id

WHERE

sd.source\_name ILIKE 'Band 4';

This query is designed to generate all the patients associated with a given device. Given the business case around ‘HealthTrack’ is about collecting source data from wearable devices and other sources and bringing all of this medical data together one report I would expect is the ability to see patients using a specific device and this is especially true if ‘HealthTrack’ includes smart devices they are selling or giving a way and being able to see how much their devices vs say patient Apple Watches contribute is important to understand the demographic as well as their campaign effectiveness.

**Patient Record:**

WITH patient\_info AS (

SELECT

p.patient\_id,

p.first\_name,

p.last\_name,

p.date\_of\_birth

FROM patient p

WHERE p.patient\_id = 2

)

SELECT

pi.patient\_id,

pi.first\_name,

pi.last\_name,

pi.date\_of\_birth,

-- Health Metrics

hm.metric\_id,

mt.metric\_type\_name,

hm.metric\_value,

hm.recorded\_date,

-- Reported Outcomes

ro.outcome\_id,

ot.outcome\_type\_name,

ro.outcome\_text,

ro.outcome\_date,

-- Predictive Analytics

pa.pa\_id,

pt.pa\_type\_name,

pa.xml\_payload AS predictive\_xml,

pa.created\_date AS predictive\_created,

-- EHR Records

ehr.ehr\_id,

eht.ehr\_type\_name,

ehr.ehr\_content,

-- Clinical Insights

ins.insight\_id,

it.insight\_type\_name,

ins.insight\_text,

ins.created\_date AS insight\_created,

-- Sensor Readings

r.reading\_id,

rt.reading\_type\_name,

r.reading\_value,

r.reading\_unit,

r.reading\_timestamp

FROM patient\_info pi

-- Health Metrics

LEFT JOIN health\_metrics hm ON pi.patient\_id = hm.patient\_id

LEFT JOIN metric\_type mt ON hm.metric\_type\_id = mt.metric\_type\_id

-- Reported Outcomes

LEFT JOIN reported\_outcomes ro ON pi.patient\_id = ro.patient\_id

LEFT JOIN outcome\_type ot ON ro.outcome\_type\_id = ot.outcome\_type\_id

-- Predictive Analytics

LEFT JOIN predictive\_analytics pa ON pi.patient\_id = pa.patient\_id

LEFT JOIN pa\_type pt ON pa.pa\_type\_id = pt.pa\_type\_id

-- EHR

LEFT JOIN ehr ehr ON pi.patient\_id = ehr.patient\_id

LEFT JOIN ehr\_type eht ON ehr.ehr\_type\_id = eht.ehr\_type\_id

-- Insights

LEFT JOIN insights ins ON pi.patient\_id = ins.patient\_id

LEFT JOIN insight\_type it ON ins.insight\_type\_id = it.insight\_type\_id

-- Readings

LEFT JOIN reading r ON pi.patient\_id = r.patient\_id

LEFT JOIN reading\_type rt ON r.reading\_type\_id = rt.reading\_type\_id

ORDER BY pi.patient\_id, hm.recorded\_date NULLS LAST, ro.outcome\_date NULLS LAST;  
  
  
This query is a bit more complex but any time you are doing to do a query that returns all of the medical related data as part of medical records there is a certain level of complexity at least implied and in a real EHR system like Epic this is going to be very complex. That said, given the complexity unless the company can’t afford a more robust RDMS system like Oracle or SQL Server using Postgres as in this case is more cumbersome and this query show cases that. In SQL Server for example I could do the same query as a set of queries and they would return as a composite record set but Postgres doesn’t like to do that so to get a single record set I end up doing this multi-level LEFT JOIN that returns more ‘bits’ then is necessary strictly speaking. In any case, we get our readings, insights, HER data, predictive analytics data, reported outcomes, metrics, etc. That said the query does use an interesting technique to use one query to populate this series of left joins.

**Patient Readings Report:**

SELECT

r.reading\_id,

r.patient\_id,

p.first\_name,

p.last\_name,

rt.reading\_type\_name,

r.reading\_value,

r.reading\_unit,

r.reading\_timestamp

FROM

reading r

JOIN

reading\_type rt ON r.reading\_type\_id = rt.reading\_type\_id

JOIN

patient p ON r.patient\_id = p.patient\_id

WHERE

r.patient\_id = 2

ORDER BY

r.reading\_timestamp DESC;

This a series of queries like this is more inline what I would use in a more robust example but also more inline with the kind of queries I would typically expect to be run to get a specific set of data and in this case source readings for a *patient*. In this example we are looking for readings for a given *patient* so we join the “*reading\_type*” reference table and the patient table to get our set of readings with the additional meta data.

Optimization  
  
Optimization of these queries would include a number of things. the last query for example we could remove the join on the patient table as we probably know that info and the type data could be cached so we really are only calling the query to the ‘reading’ table. For example:

SELECT

r.reading\_id,

r.patient\_id,

r.reading\_value,

r.reading\_unit,

r.reading\_timestamp

FROM

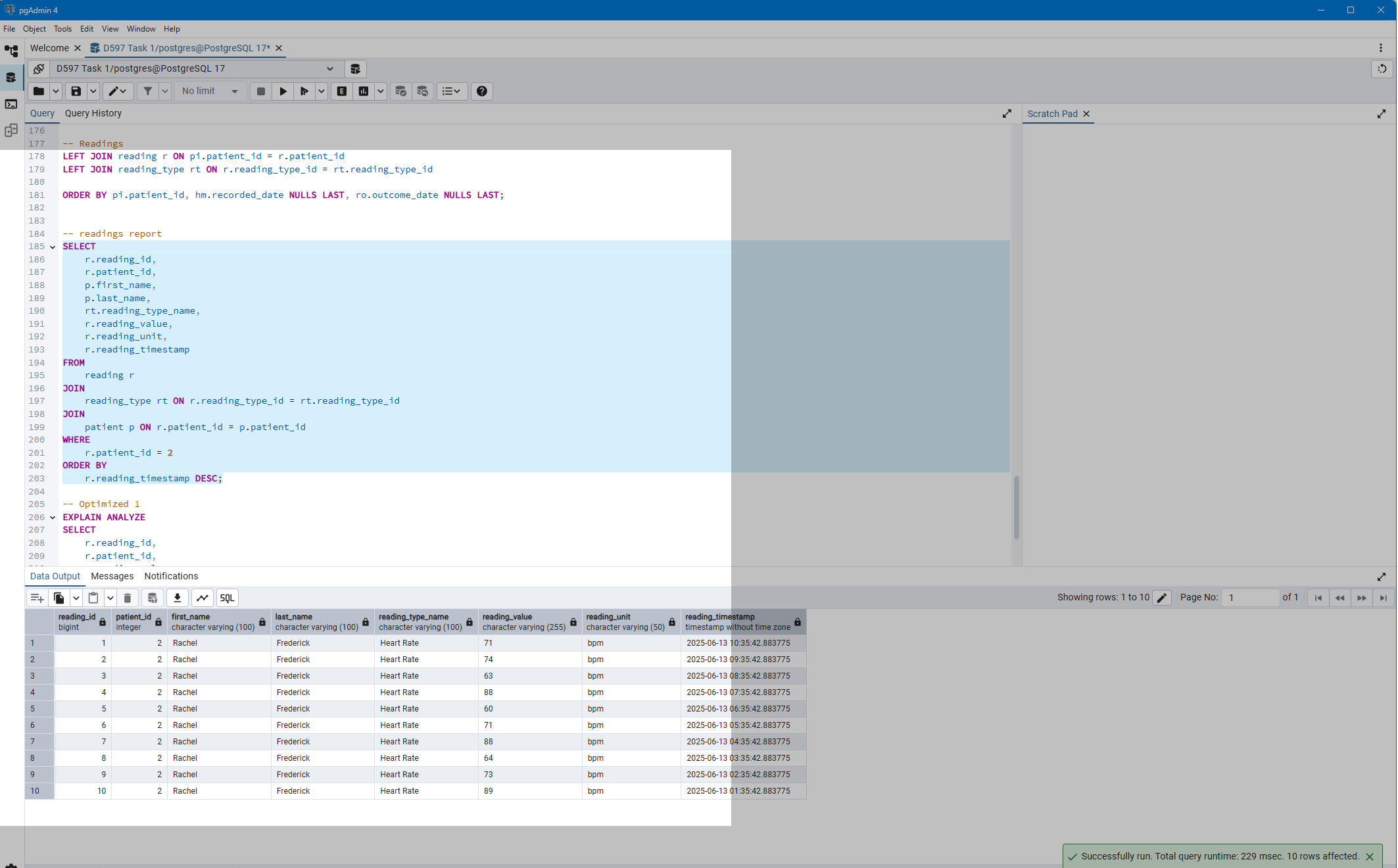
reading r

WHERE

r.patient\_id = 2

ORDER BY

r.reading\_timestamp DESC LIMIT 100;

On top of that there are a lot of other things we can do, like using the query analyzer or in PostGres in pgAdmin 4 we could use the ‘explain analyze’ feature. Generally creating index’s for things that are done a lot of would help too. In this one creating index’s on the foreign key’s and making composite indexes. In this one case you might use this example:   
  
CREATE INDEX idx\_reading\_patient\_timestamp\_desc ON reading (patient\_id, reading\_timestamp DESC);  
  
So in running the original query this is the result given the current test data:  
  


After all of the optimizations in this example we got this result:   
A screenshot of a computer

AI-generated content may be incorrect.

Note the difference is 229 msec vs 144 msec, which is about a 63% increase in speed, while not a huge difference as such when the query only takes milliseconds performance is increasingly an issue when scaling up any given system.

### Part 2.References

Local Post Gress Database used from this location: <https://www.postgresql.org/> installed locally: “postgresql-17.5-1-windows-x64.exe” 061102025