Joint Databases for Electronic Health Record Analysis

Teresa Lee, Tino Trangia, Micah Hunter
DSC 202

Background

As healthcare becomes more data-driven, the need for secure and efficient database systems is growing. These systems must handle vast amounts of information while remaining adaptable to industry changes.

One major challenge is integration, as healthcare data comes from various sources in different formats, requiring seamless interoperability. Future-proofing is also crucial—rigid databases struggle to accommodate new technologies and regulations, leading to costly updates. Additionally, healthcare queries range from simple patient record retrieval to complex analyses like tracking disease patterns, requiring both relational and advanced analytics capabilities.

Addressing these challenges is essential to building a scalable, efficient database that improves patient care, streamlines operations, and supports better decision-making.

Our goal is to create a healthcare database that improves data access, connects patients with diseases and insurers, and supports better diagnoses through similarity queries. By linking medical records across systems, we can enhance interoperability and reveal insights into how social factors impact health. To make this possible, the database must handle diverse queries, adapt to industry changes, and seamlessly integrate data from different sources.

Data

Medical Records

Electronic health records (EHRs) are critical for modern healthcare systems as an efficient, centralized, and interoperable way to manage patient data. There are usually records for each patient, who may have one or more existing (and previous) conditions, medications, encounters, claims, etc.

To populate a hypothetical EHR database, we used synthetic patient records from SyntheticMass. This dataset contains “realistic but fictional residents of the state of Massachusetts”. The data is generated by Synthea, a Synthetic Patient Population Simulation that uses models for disease diagnosis and progression to create health records. This allowed us to analyze data without privacy issues. We used both the 1000 sample and 100 sample pre-generated datasets.

Social Determinants of Health

Social determinants of health (SDOH) are non-medical factors within a geographic region such as economic stability, education access, demographics, and built environment characteristics. These factors can often have a significant impact on an individual’s health, hence the aphorism that “your zip code is a better predictor of health outcomes than your genetic code.” Unfortunately, these are often missing from EHRs due to a lack of interoperability, so we sought to integrate them into our database.

We used a dataset for Social Determinants of Health (SDOH) from the Agency for Healthcare Research and Quality (AHRQ). This was originally available as .xlsx but we have converted to .csv for our project. We used the 2020 zip code-based data, although data was also available at the county and census tract granularities. Zip codes were the lowest granularity we could easily use, as the patient data contains their zip code address while matching them to census tracts would require additional integration with geographic shapefiles.

Methodology (Overview)

Our approach begins with exploratory data analysis to understand the structure and patterns within the data. Then we import structured electronic health record (EHR) data into PostgreSQL for relational storage and determine the key nodes and relationships needed for a graph database. Using Python, we load this data into Neo4j, enabling more complex network analysis. To enhance the dataset, we integrate social determinants of health (SDOH) data into both the relational and graph databases, providing a more comprehensive view of patient health. Finally, we build and test queries to address common use cases, ensuring the system effectively supports real-world healthcare applications.

An example of one of our Python functions (wrapping a Cypher query) that reads in the CSV data, creates nodes and relationships, and writes them to the Neo4j instance:

```python
encounters_csv_data = []
with open("csv/encounters.csv", newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        encounters_csv_data.append(row)
```

```python
def upload_encounters(tx, records):
    cypher_query = """
    UNWIND $records as record
    MERGE (e:Encounter {Id: record.Id})
    SET e = record
    WITH e, record
    MATCH (p:Patient {Id: record.PATIENT})
    MERGE (p)-[:HAS_ENCOUNTER]->(e)
    WITH e, record
    MATCH (c:Condition {ENCOUNTER: record.Id})
    MERGE (c)-[:FROM_ENCOUNTER]->(e)
    """
    tx.run(cypher_query, records=records)

with driver.session(database="neo4j") as session:
    session.execute_write(upload_encounters, encounters_csv_data)
```

Relational Queries

For our relational-based queries, we imported the electronic health records (EHR) and social determinants of health (SDOH) data in PostgreSQL. We used the 1000 electronic health records sample with the social determinants data to solve some common questions.

Example 1

What are the top 10 most common medical condition disorders in our data?
This first example as seen below is a simple query using just one table called “Conditions”. We used a simple group by and filter to extract the top 10 most common medical condition disorders in our table. Because the “descriptions” column contains two types of description data, we have to query for those rows that are medical disorders.

Query:

```
select description, count(*) as counts
from "Conditions"
group by description
having description like '%(disorder)%'
order by counts desc
limit 10;
```

Output:

	description	counts
1	Viral sinusitis (disorder)	1233
2	Acute viral pharyngitis (disorder)	678
3	Acute bronchitis (disorder)	571
4	Anemia (disorder)	324
5	Chronic sinusitis (disorder)	219
6	Streptococcal sore throat (disorder)	162
7	Acute bacterial sinusitis (disorder)	74
8	Hypertriglyceridemia (disorder)	71
9	Metabolic syndrome X (disorder)	68
10	Osteoporosis (disorder)	58

Example 2

For the top 10 most common medical condition disorder, find the break down of the condition between genders

In this second and more complex query, we build off the first example. In this query, we want to find the breakdown across genders for the top 10 most common medical disorders. Aside from the “Conditions” table, we also have to use the “Patients” table and join the two tables together. The query below shows the query code used to solve for the question. And in the output, we can see the breakdown of counts of each medical condition across gender.

Query:

```sql
-- demo example 2
create table conditionCounts as (
    select description, count(*) as counts
    from "Conditions"
    group by description
    having description like '%(disorder)%'
    order by counts desc
    limit 10
    )

create table countsGender as (
    select c.description, p.gender, count(*) as gender_count
    from "Conditions" c
    left join "Patients" p on c.patient = p.id
    where c.description in (select conditioncounts.description
                            from conditionCounts)
    group by c.description, p.gender
    )

select cc.description,
    COALESCE(SUM(CASE WHEN cg.gender = 'M' THEN cg.gender_count END), 0) AS males,
    COALESCE(SUM(CASE WHEN cg.gender = 'F' THEN cg.gender_count END), 0) AS females
from conditioncounts cc
left join countsGender cg on cc.description=cg.description
group by cc.description
ORDER BY cc.description desc
```

Output:

	description	males	females
1	Viral sinusitis (disorder)	563	670
2	Streptococcal sore throat (disorder)	65	97
3	Osteoporosis (disorder)	18	40
4	Metabolic syndrome X (disorder)	33	35
5	Hypertriglyceridemia (disorder)	35	36
6	Chronic sinusitis (disorder)	107	112
7	Anemia (disorder)	173	151
8	Acute viral pharyngitis (disorder)	317	361
9	Acute bronchitis (disorder)	283	288
10	Acute bacterial sinusitis (disorder)	36	38

Example 3

Combining the synthetic data with real demographic data on zip code, find if there is a correlation between the percentage of foreign born citizens in a zip code location to the number of medical conditions in that location.

For this example, we have to join three tables together. We use the “Conditions” and “Patients” table which are the EHR data. Then we join that with the “Citizenship” table from the SDOH data.

Our output shows the top 10 zip codes with most medical conditions compared to the bottom 10 zip codes. And we can see in our output that zip codes that have higher counts of medical condition disorders, tend to have a higher percentage of foreign born residents. We haven’t proved that there is any causal relationship between the percentage of foreign born residents and the number of medical conditions reported in a zipcode, but we noticed a correlation between the two.

Query:

```sql
with combinedTable as (
    select p.id,
           p.zip,
           co.description,
           ci.acs_pct_foreign_born_zc as percentage_foreign_born
    from "Patients" p
    join "Citizenship" ci on p.zip=ci.new_zipcode
    join "Conditions" co on p.id=co.patient
)

select zip, percentage_foreign_born, count(*) as counts
from combinedTable c
group by zip, percentage_foreign_born
order by counts desc
```

Output:

Top 10 zip codes with most medical conditions

	zip	percentage_foreign_born	counts
1	02171	37.72	603
2	02116	24.34	536
3	01020	7.57	425
4	02723	22.41	419
5	01803	23.45	385
6	01970	15.77	368
7	01940	9.23	360
8	02790	10.69	354
9	02169	31.86	339
10	02138	27.85	319

Bottom 10 zip codes with most medical conditions

212	02067	23.56	4
213	02664	12.26	4
214	01566	6.4	4
215	02191	11.62	4
216	02655	13.16	4
217	01129	8.12	3
218	01540	6.5	3
219	02554	17.31	3
220	01030	6.18	2
221	02675	6.05	2

These three examples are just a few of the many relational-data problems that can be solved using PostgreSQL.

Graph Queries

In order to support graph-based queries, we imported both the electronic health record and social determinants of health data into Neo4j. For this project, we used the 100-patient sample to improve performance. Even with just 100 patients, the resulting graph has approximately 140,000 nodes and 160,000 edges. This is due to the complex relationships between patients, providers, and medical histories which may contain hundreds of conditions, medications, encounters, claims, and more.

Example 1

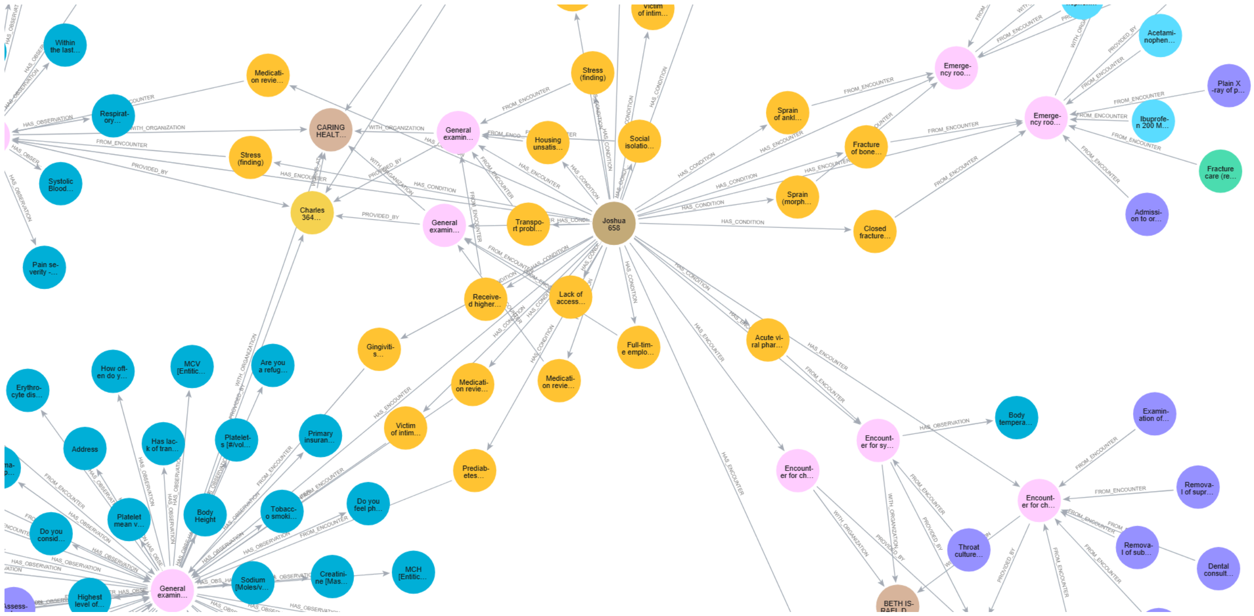
We begin by using a simple query and Neo4j browser to visualize how data is represented on our graph database. This query essentially finds a neighborhood (up to 3 edges from the patient). This gives users an idea of what a single patient's electronic health records look like when uploaded to Neo4j.

Query:

```
MATCH path = (p:Patient {Id:
"30a6452c-4297-a1ac-977a-6a23237c7b46"})-[*1..3]-(n)
RETURN path
```

Output:

The output graph reveals how one patient is connected to various conditions. These conditions may then be linked to encounters (e.g. primary care encounters, emergency room encounters) which are in turn linked to a care provider. Encounters may also be associated with various observations, including open-form questions (e.g. how are you feeling) and clinical measurements (e.g. blood pressure, body-mass index). We also see that providers and encounters are linked to organizations, usually a hospital or health clinic.



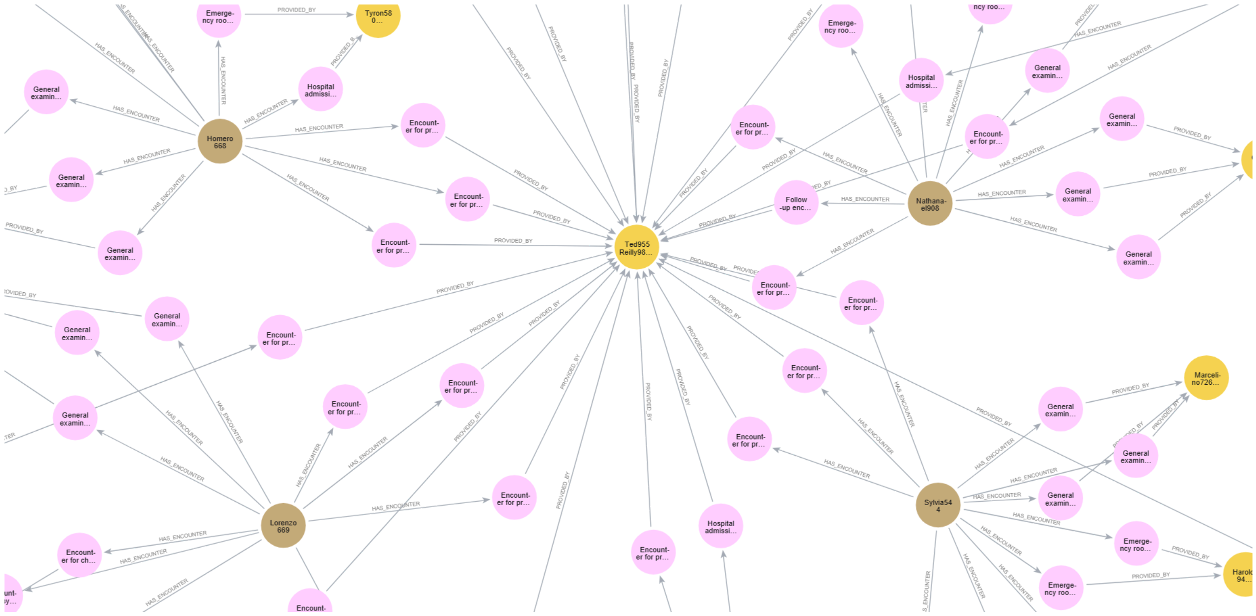
Example 2

The second example looks at a slightly more complex output by showing provider-patient networks. This query matches patient nodes with encounters and their providers (limiting to the first 3 encounters per patient, for ease of display). This provides users with a way to represent patients with similar providers and interactions with the healthcare system.

Query:

```
MATCH
(pt:Patient)-[r:HAS_ENCOUNTER]->(e:Encounter)-[:PROVIDED_BY]->(p:Provider)
WITH pt, p, e
WITH pt, p, collect(e)[0..3] AS limitedEncounters
UNWIND limitedEncounters AS e
MATCH path = (pt)-[r:HAS_ENCOUNTER]->(e)-[:PROVIDED_BY]->(p)
RETURN pt, p, collect(path) AS paths
```

Output:



Example 3

This query asks: Which patients have been diagnosed with pre-diabetes and were prescribed with insulin? Show their most recent encounters with a general practitioner. Conditions and medications are classified under codes (from the SNOMED biomedical ontology), which allows us to filter by specific conditions and medications.

Query:

```
MATCH (p:Patient)-[:HAS_CONDITION]->(c:Condition {CODE: '714628002'}),
      (p)-[:HAS_MEDICATION]->(m:Medication {CODE: '106892'}),
      (p)-[:HAS_ENCOUNTER]->(e:Encounter),
      (e)-[:PROVIDED_BY]->(pr:Provider {SPECIALITY: 'GENERAL PRACTICE'})
RETURN DISTINCT p.FIRST AS PatientFirstName,
       p.LAST AS PatientLastName,
       pr.NAME AS ProviderName,
       e.START AS EncounterStartDate
ORDER BY PatientLastName ASC, e.START DESC;
```

Output:

PatientLastName	ProviderName	EncounterStartDate
"Balistreri607"	"Erwin847 Stiedemann542"	"2024-09-08T07:48:58Z"
"Balistreri607"	"Laurena366 Anderson154"	"2023-09-17T07:48:58Z"
"Balistreri607"	"Erwin847 Stiedemann542"	"2023-09-03T07:48:58Z"
"Balistreri607"	"Laurena366 Anderson154"	"2023-07-16T07:48:58Z"
"Balistreri607"	"Laurena366 Anderson154"	"2022-09-11T07:48:58Z"
"Balistreri607"	"Erwin847 Stiedemann542"	"2022-08-28T07:48:58Z"
"Balistreri607"	"Laurena366 Anderson154"	"2021-08-29T07:48:58Z"
"Balistreri607"	"Erwin847 Stiedemann542"	"2021-08-22T07:48:58Z"
"Balistreri607"	"Laurena366 Anderson154"	"2021-07-04T07:48:58Z"

Example 4

This query asks: For pre-diabetic patients, what are their providers and what is the total claim cost of all of their encounters with the healthcare system?

Query:

```
MATCH (p:Patient)-[:HAS_CONDITION]->(c:Condition {CODE: '714628002'})
WITH p, count(DISTINCT c) as conditionCount
WHERE conditionCount >= 1
MATCH (p)-[:HAS_ENCOUNTER]->(e:Encounter)-[:PROVIDED_BY]->(pr:Provider)
WITH p, collect(DISTINCT pr) as providers, sum(toFloat(e.TOTAL_CLAIM_COST))
as totalCost
RETURN p.FIRST AS PatientFirstName,
       p.LAST AS PatientLastName,
       providers,
       totalCost
ORDER BY totalCost DESC;
```

Output:

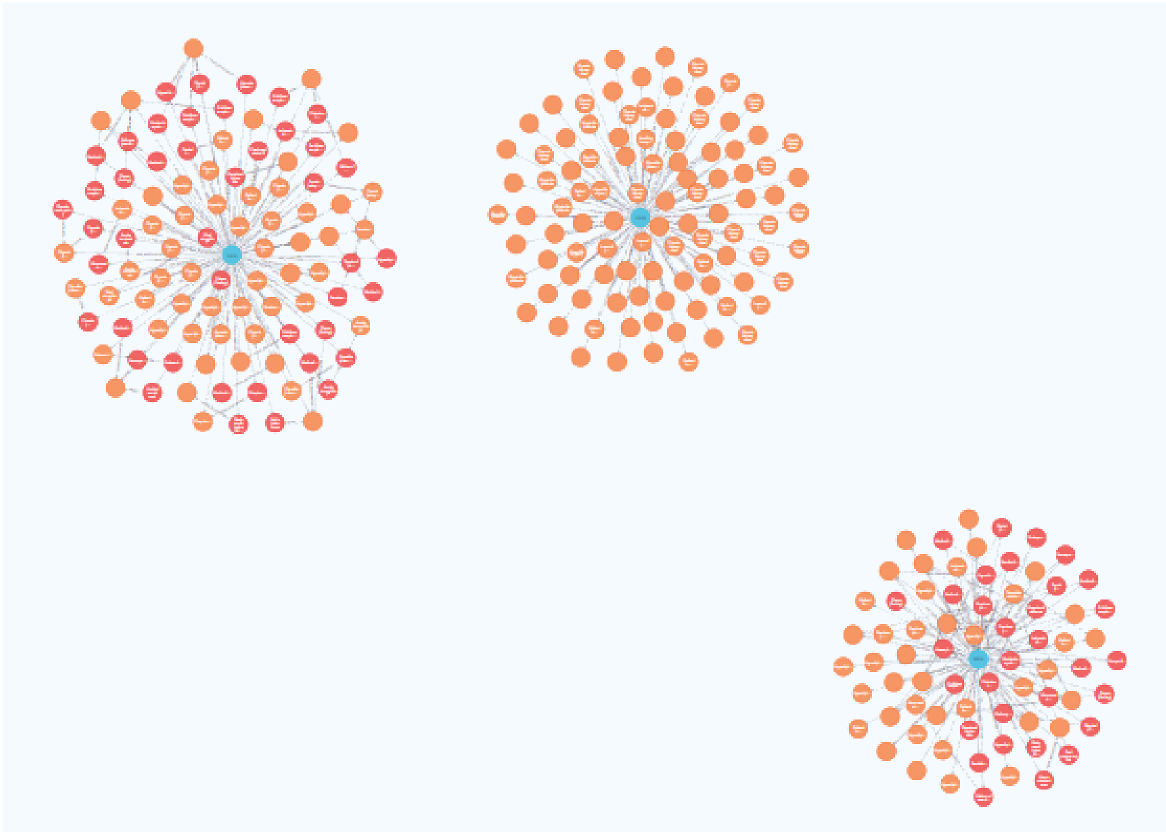
PatientFirstName	PatientLastName	providers	totalCost
"Elna874"	"Prohaska837"	[(:Provider {ZIP: "021111552", PROCEDURES: "0", STATE: "MA", LON: "-71.0631836", NAME: "Santina680 Dicki44", ORGANIZATION: "0d1570ab-371c-3898-9397-95905d8c5166", CITY: "BOSTON", ADDRESS: "750 WASHINGTON ST", GENDER: "F", Id: "8ba9dc63-e8c2-383a-9031-314602010985", SPECIALITY: "GENERAL PRACTICE", LAT: "42.3499038", ENCOUNTERS: "776"}), (:Provider {ZIP: "021253120", PROCEDURES: "0", STATE: "MA", LON: "-71.04610844302991", NAME: "Magdalena964 Torphy630", ORGANIZATION: "2b97893e-dc50-378b-a266-f089b8450329", CITY: "DORCHESTER", ADDRESS: "250 MOUNT VERNON ST", GENDER: "F", Id: "2d312216-1433-3a77-a29e-f1d766339b2d", SPECIALITY: "GENERAL PRACTICE", LAT: "42.3194571", ENCOUNTERS: "68"}), (:Provider {ZIP: "021272642", PROCEDU	645070.6300000001

Example 5

This query identifies patients with multiple risk factors for heart attacks such as hypertension, high cholesterol, obesity, coronary artery disease, or diabetes. It returns patients sorted by the number of risk factors. We kept the risk factor threshold at 2 for meaningful insights.

```
MATCH (p:Patient)-[:HAS_CONDITION]->(c:Condition)
WHERE c.DESCRIPTION CONTAINS "Hypertension"
   OR c.DESCRIPTION CONTAINS "Hyperlipidemia"
   OR c.DESCRIPTION CONTAINS "Obesity"
   OR c.DESCRIPTION CONTAINS "Diabetes"
   OR c.DESCRIPTION CONTAINS "Coronary Artery Disease"
WITH p, COUNT(c) AS riskFactors
WHERE riskFactors >= 2
RETURN p, riskFactors
ORDER BY riskFactors DESC
```

Output:



Relational and Graphical Combined

Example 1

This is the SQL query to find the ZIP codes with the highest healthcare expenses, along with the associated races from the synthetic patient dataset:

```
SELECT zip, race, AVG(healthcare_expenses) AS avg_expenses
FROM patients
GROUP BY zip, race
ORDER BY avg_expenses DESC
LIMIT 100;
```

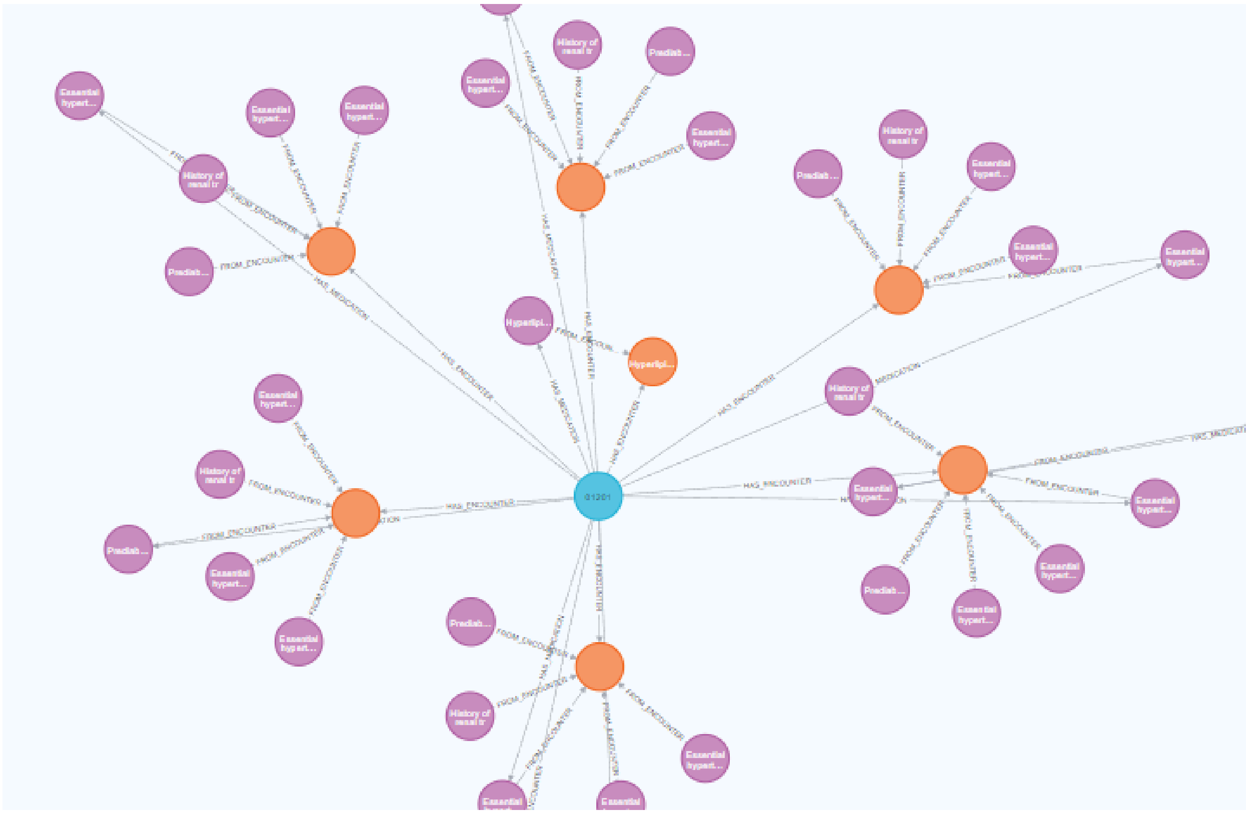
Output:

	zip	race	avg_expenses
1	02116	black	11839006.005750002
2	02721	white	3683373.9988750005
3	02476	white	3285140.7290000003
4	01063	white	3171906.983500001
5	02768	black	2984580.6865000012
6	01550	white	2983892.6684999997
7	02136	asian	2829171.319
8	01852	asian	2756377.8550000004
9	01952	white	2700908.87
10	01373	white	2677158.173
11	02670	white	2623445.944999999
12	01020	black	2473354.888

This Cypher query retrieves information about patients and the medications they are prescribed. It first matches all patients who are linked to medications through the `HAS_MEDICATION` relationship. The query then collects the unique medications for each patient based on their race (SDOH data imported to Neo4j as a csv) which was imported from the output of a relational query in Postgres. The result is a list of races, with each race showing the distinct medications prescribed to patients of that race.

```
MATCH (p:Patient)-[:HAS_MEDICATION]->(m:Medication)
RETURN p.race AS race, COLLECT(DISTINCT m) AS
unique_medications
```

Output:



Lessons Learned

Throughout this project, we learned a lot about the complexities of integrating healthcare data. One of the biggest challenges was ensuring interoperability between different databases, as structured SQL data and relationship-based Neo4j data required careful mapping to work together seamlessly. We also saw firsthand how important data quality is—while we worked with synthetic data, real-world healthcare datasets often have missing values and inconsistencies that must be addressed. Using Neo4j for graph-based queries opened up new ways to analyze complex relationships between patients, conditions, and providers, providing insights that would be difficult to extract from a traditional relational database. However, we also had to consider computational performance, as large datasets required optimizations like indexing and query tuning to keep things running smoothly.

Incorporating social determinants of health (SDOH) data was another key takeaway, as it showed how non-medical factors like economic stability and education can influence health outcomes. This reinforced the need for healthcare databases to move beyond just clinical data to provide a more complete picture of patient well-being. Scalability and future-proofing were also critical considerations, as evolving regulations and standards like HL7 FHIR will require flexible database architectures. Lastly, even though we used synthetic data, working on this project highlighted the importance of privacy and ethical data handling, which are essential in real-world applications to ensure compliance with laws like HIPAA. Overall, this experience gave us a deeper understanding of healthcare data challenges and the potential of integrated systems to improve patient care and decision-making.

Conclusion (and future work)

In conclusion, our project seeks to bridge the gaps in healthcare data management by combining structured data through SQL and relationship-based data using Neo4j. By doing so, we aim to enhance healthcare insights, address the fragmentation of patient data, and ultimately improve diagnostics and patient care access. The integration of diverse data sources, including social determinants of health, will pave the way for more personalized and informed healthcare decisions.

Looking ahead, our future work will focus on developing pipelines for handling JSON data, specifically tailored to support the HL7 FHIR (Fast Healthcare Interoperability Resource) standard, which is critical for ensuring interoperability across the healthcare ecosystem. Additionally, we plan to further extend the integration between PostgreSQL, Neo4j, and potentially document-based databases such as MongoDB, to create even more flexible and scalable solutions that can keep pace with the evolving demands of the healthcare industry.

References

- Electronic Health Records | CMS
- Social Determinants of Health - Healthy People 2030 | odphp.health.gov
- Overview - FHIR v5.0.0