DUBLIN BUSINESS SCHOOL

GRAPH AND AI

B9AI101_2223_TMD2

Contact Tracing using Graph Algorithms (Neo4j)

by

Mr. Dhavan Nagaraj (10626855)

Under the Guidance of

Dr. Terri Hoare

Master of Science in Artificial Intelligence **2023 - 2024**

Table of Contents

OVERVIEW

The most popular open-source Graph Database in the world, Neo4j, was created utilizing Java technology. It has no schema and is very scalable (NoSQL). A graph is a visual representation of a collection of things where some object pairs are linked together. There are two components to it: nodes (vertices) and relationships (edges). A database called a "graph database" is used to model data as a graph. Here, the entities are represented by the graph's nodes, while their connections between them are shown by their relationships. Most data today come in the form of relationships between various objects, and most of the time, these relationships are more useful than the individual pieces of data. Using relational databases, you may store data that is well structured and. Relational databases can be used to store structured data because they hold highly structured data, which is data that is stored in several entries of the same kind and does not store the relationships between the records. In contrast to conventional databases, graph databases maintain connections and relationships as first-class entities. Graph databases have a simpler data model than other types of databases, and they can be used with OLTP systems. Features like operational availability and transactional integrity are offered by them.

METHODOLOGY

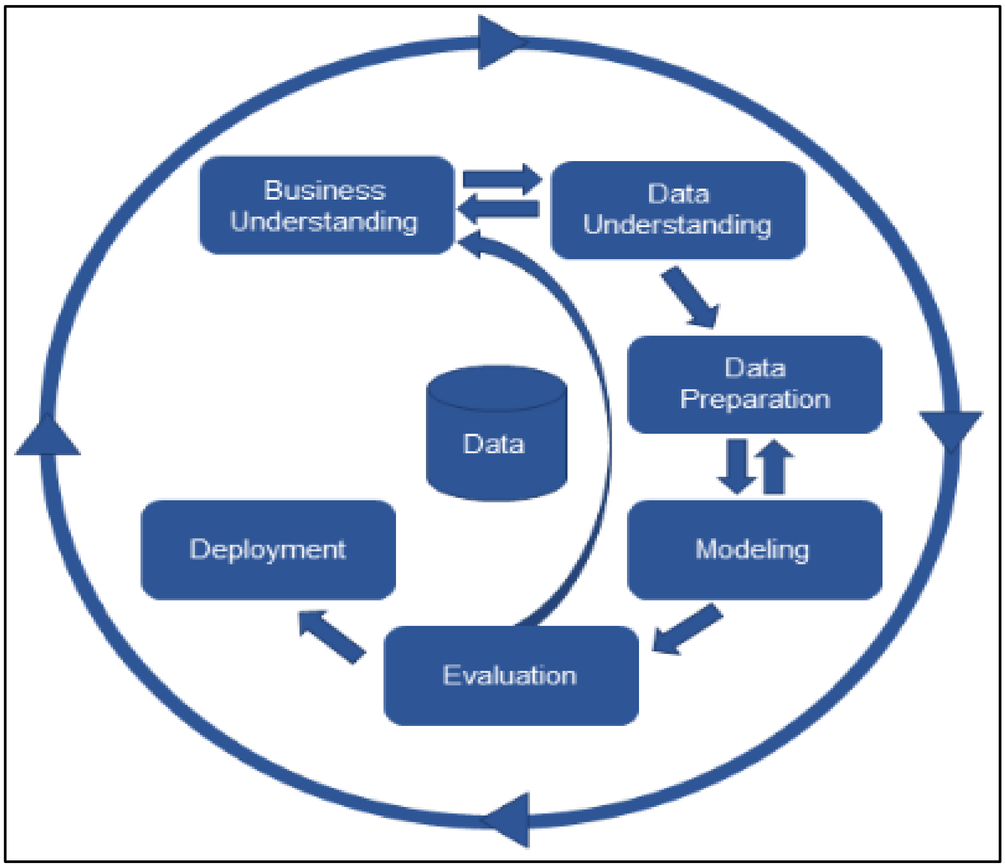


Figure 1: Crisp Dm

The six stages are depicted in the diagram above, which demonstrates how the process moves along and which stages are dependent on one another. The following are the 6 phases of crisp-dm:

1) **BUSINESS UNDERSTANDING**

This is the first phase of the crisp-dm. In this phase we focus on understanding the projects objectives and requirements from a business perspective. After this by using all the information we will convert this information into data mining problem definition and design a plan to achieve the objective.
My project aims to comprehend contact tracing utilizing neo4j graph and node embedding forecast how many healthy people would have encountered the ill person and what is the likelihood that healthy person will be affected. We will also assist in identifying the locations visited by the sick individual, the target time, who he affected and for how long using Neo4j. The Graph Data Science Library and its many Graph algorithms will be used to achieve the goal of this research. The Contact Tracing Dataset has been stored in Neo4j, a Popular NoSQL Database, for this. The work was done for the "Contact Tracing using Graph Algorithms and Exploring Graph Analytics in a Graph Database (Neo4j)" project.

The information can be viewed in an optimal way when compared to relational databases, which is one of graph databases' key benefits.
• Simple to understand since they use a natural way to describe information in graph form.
• Because of how quickly and readily data is added and removed, they are easily modified over time.
• It can accommodate petabyte-scale data storage.
• Data mining operations can also add new data kinds and be optimized.

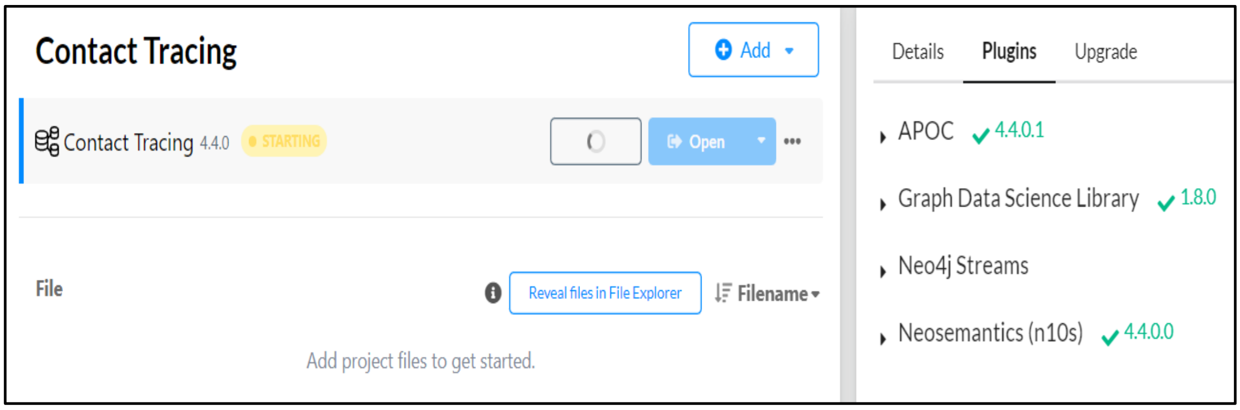
2) **DATA UNDERSTANDING**

It can be used to learn more about the qualities, relationships, and structure of data. Users may simply query and examine data relationships with Neo4J, create interactive visualizations, and find patterns and correlations rapidly. Furthermore, Neo4J offers strong tools for data analysis and processing. These technologies enable users to visualize data, identify connections between data pieces, and uncover hidden insights. Users can gain a better grasp of the given dataset by utilizing Neo4J's capabilities, which encourages data processing to come to better conclusions and produce better advancements.

It will be necessary to comprehend the people's health situation in order to trace contacts. Our information includes both sick and healthy people as well as the locations to which they go. We may also see the location and the time an ill person targets a healthy person while they are together. We were able to determine from our dataset whether a disease spreads through contact with people or through routine visits.

3) DATA PREPARATION

The following steps will help us to understand the connection of neo4j with to the clear idea on the node and their relationship.


Contact Tracing

Contact Tracing 4.4.0 STARTING

File

Add project files to get started.

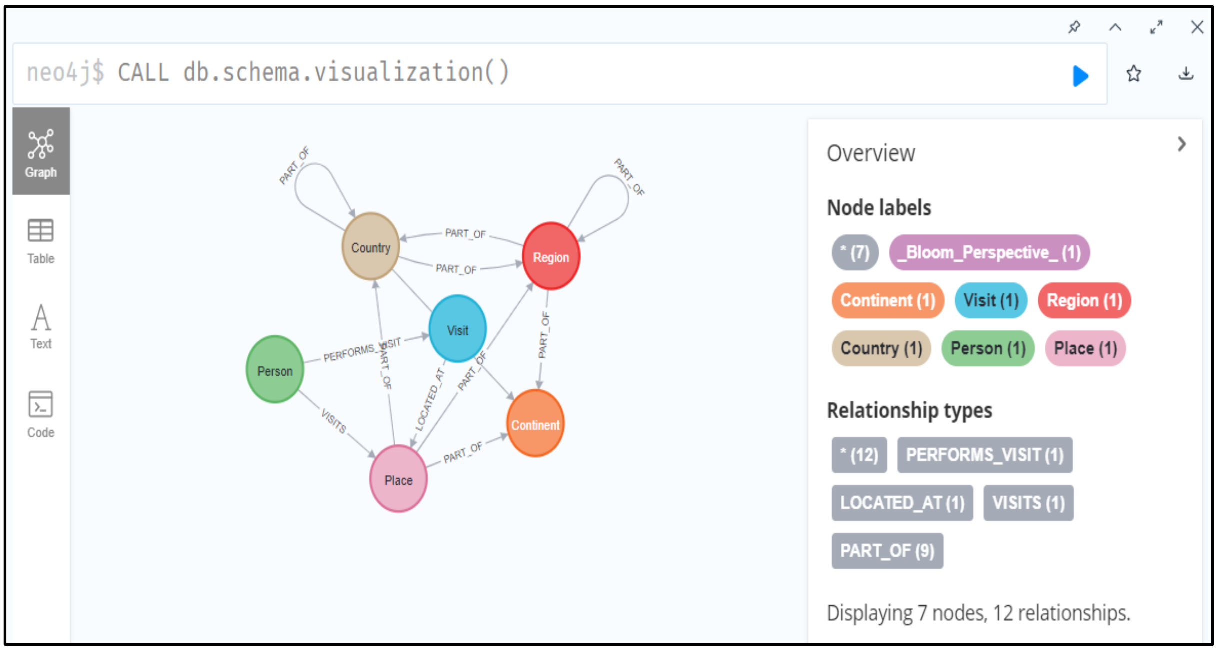
APOC ✓ 4.4.0.1

Graph Data Science Library ✓ 1.8.0

Neo4j Streams

Neosemantics (n10s) ✓ 4.4.0.0

Figure 2: Neo4j desktop interface


```
neo4j$ CALL db.schema.visualization()
```

Overview

Node labels

* (7) _Bloom_Perspective_ (1) Continent (1) Visit (1) Region (1) Country (1) Person (1) Place (1)

Relationship types

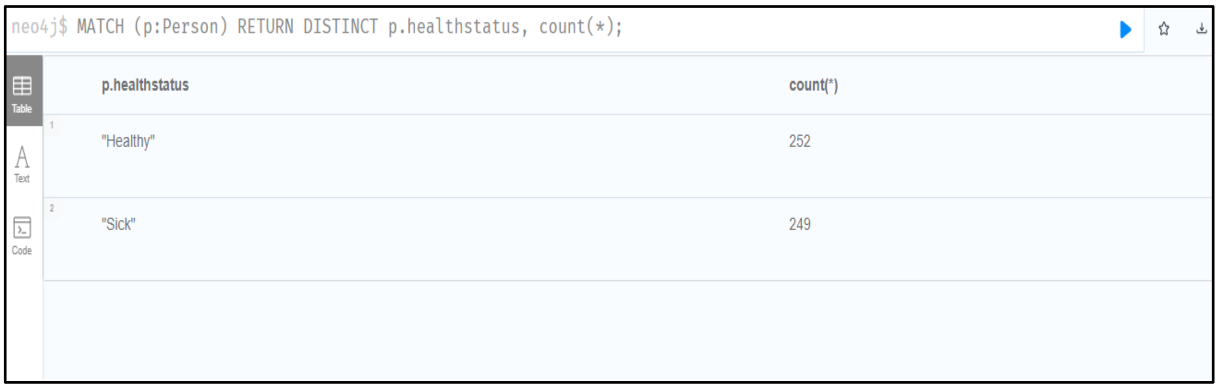
* (12) PERFORMS_VISIT (1) LOCATED_AT (1) VISITS (1) PART_OF (9)

Displaying 7 nodes, 12 relationships.

Figure 3: schema visualization

This query is used to view the database schema, which shows a graphical depiction of the nodes in the database, their relationships, and their characteristics. Upon execution, this query launches a Neo4j Browser visualization tool that shows the database's schema in a visual manner. You may graphically pan, zoom in and out, and explore the database schema with the tool. It's a handy tool for investigating the structure of your data and for locating possible places where your data model could be optimized and improved. By running this procedure, we can see the node labels and relationship types used in the dataset. There are 252 healthy people and 248 sick people in the dataset.

➢ Health status count



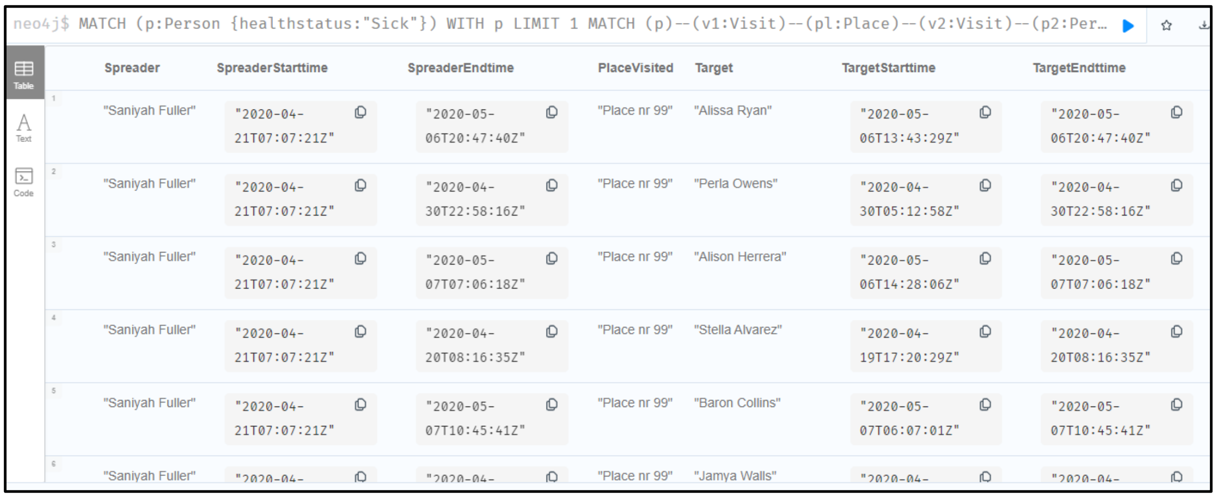
```
neo4j$ MATCH (p:Person) RETURN DISTINCT p.healthstatus, count(*);
```

	p.healthstatus	count(*)
1	"Healthy"	252
2	"Sick"	249

Figure 4: Health status count

According to the query, it is to be tallying the number of various health statuses of individuals in a graph database. The response from the query should be a table containing the columns "health status" and "count." People's individual health statuses are listed in the "health status" column, and the "count" column lists how many people belong to each health status.

➢ Potential Infection



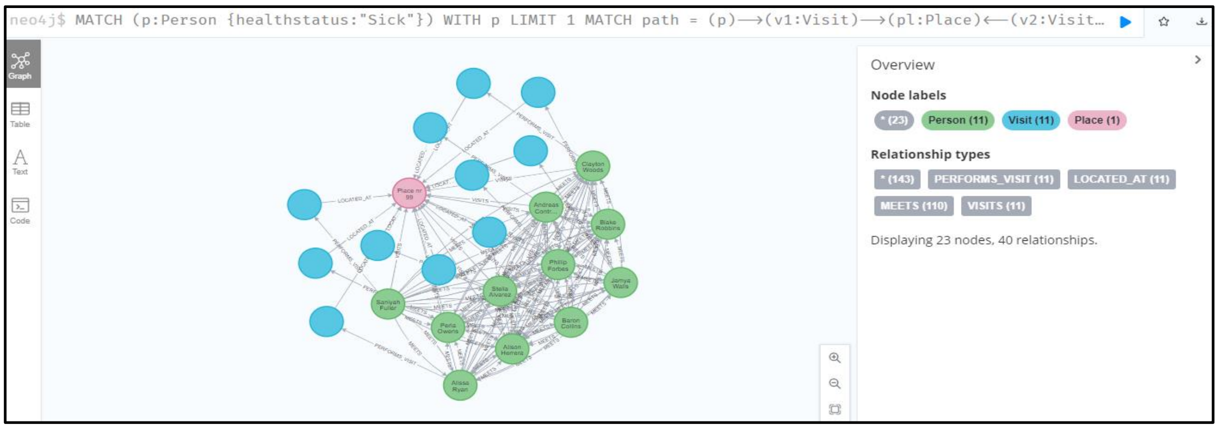
```
neo4j$ MATCH (p:Person {healthstatus:"Sick"}) WITH p LIMIT 1 MATCH (p)--(v1:Visit)--(pl:Place)--(v2:Visit)--(p2:Per…
```

	Spreader	SpreaderStarttime	SpreaderEndtime	PlaceVisited	Target	TargetStarttime	TargetEndttime
1	"Saniyah Fuller"	"2020-04-21T07:07:21Z"	"2020-05-06T20:47:40Z"	"Place nr 99"	"Alissa Ryan"	"2020-05-06T13:43:29Z"	"2020-05-06T20:47:40Z"
2	"Saniyah Fuller"	"2020-04-21T07:07:21Z"	"2020-04-30T22:58:16Z"	"Place nr 99"	"Perla Owens"	"2020-04-30T05:12:58Z"	"2020-04-30T22:58:16Z"
3	"Saniyah Fuller"	"2020-04-21T07:07:21Z"	"2020-05-07T07:06:18Z"	"Place nr 99"	"Alison Herrera"	"2020-05-06T14:28:06Z"	"2020-05-07T07:06:18Z"
4	"Saniyah Fuller"	"2020-04-21T07:07:21Z"	"2020-04-20T08:16:35Z"	"Place nr 99"	"Stella Alvarez"	"2020-04-19T17:20:29Z"	"2020-04-20T08:16:35Z"
5	"Saniyah Fuller"	"2020-04-21T07:07:21Z"	"2020-05-07T10:45:41Z"	"Place nr 99"	"Baron Collins"	"2020-05-07T06:07:01Z"	"2020-05-07T10:45:41Z"
6	"Saniyah Fuller"	"2020-04-	"2020-04-	"Place nr 99"	"Jamya Walls"	"2020-04-	"2020-04-

Figure 5: Potential infection

This Cypher search is looking for a person (p) whose "health status" property value is "Sick," then looking for a visit (v1) that person made to a location (p), then looking for a second visit (v2) to the same location made by a different person (p2) whose "health status" property value is "Healthy." Information concerning possible disease transmission from the "Sick" person to the "Healthy" person via the shared environment is returned by the query.

➢ Potential Infection Graph



```
neo4j$ MATCH (p:Person {healthstatus:"Sick"}) WITH p LIMIT 1 MATCH path = (p)⟶(v1:Visit)⟶(pl:Place)⟵(v2:Visit…
```

Overview

Node labels

* (23) Person (11) Visit (11) Place (1)

Relationship types

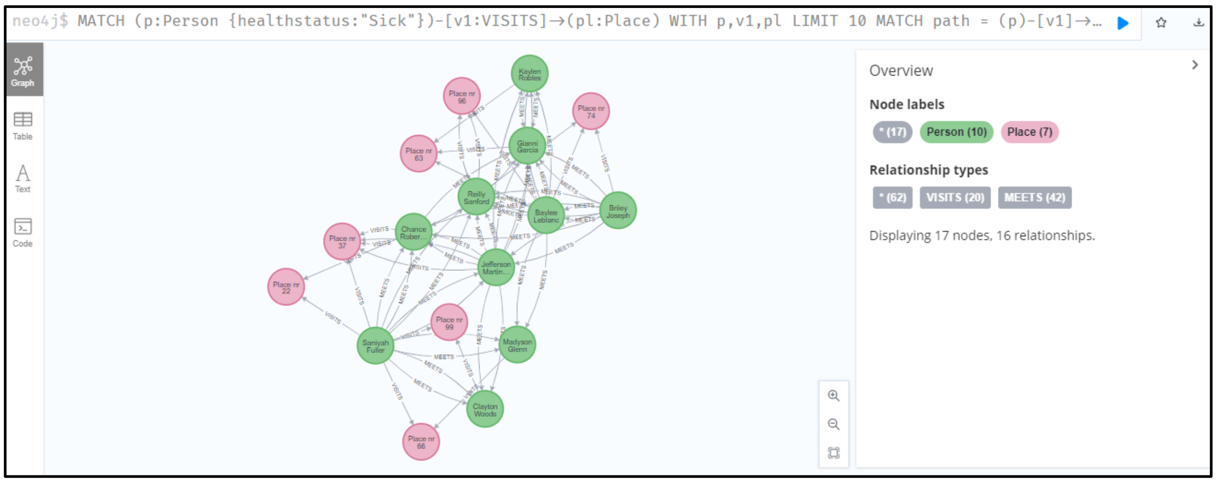
* (143) PERFORMS_VISIT (11) LOCATED_AT (11) MEETS (110) VISITS (11)

Displaying 23 nodes, 40 relationships.

Figure 6: Potential Infection Graph

This Cypher query is looking for a person (p) whose "health status" property value is "Sick," then looking for a visit (v1) that person made to a location (p), and then looking for a second visit (v2) to the same location made by a different person (p2) whose "health status" property value is "Healthy." This search gives the route that passes via the shared site and joins the "Healthy" person with the "Sick" person. The query returns a path that begins at the "Sick" person, travels through that person's visit (v1) to the shared location, travels through the shared location itself (pl), travels through the visit (v2) that the "Healthy" person made to the same shared location, and finally ends at the "Healthy" person.

➢ Adding Overlap time



```
neo4j$ MATCH (p:Person {healthstatus:"Sick"})-[v1:VISITS]→(pl:Place) WITH p,v1,pl LIMIT 10 MATCH path = (p)-[v1]→…
```

Overview

Node labels

* (17) Person (10) Place (7)

Relationship types

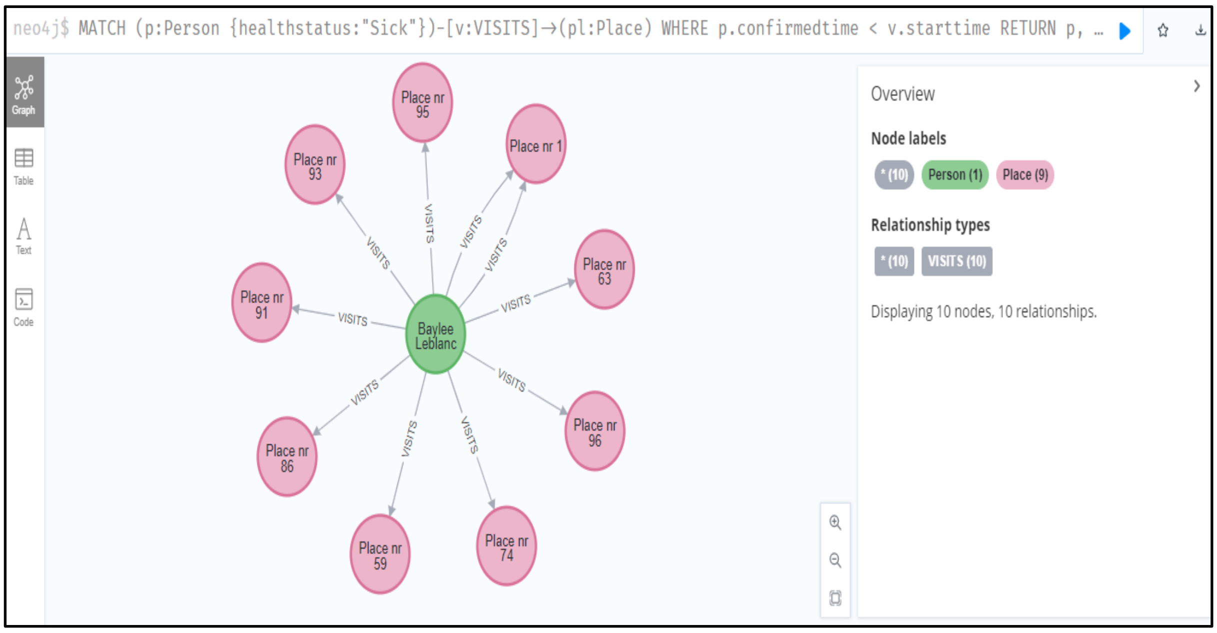
* (62) VISITS (20) MEETS (42)

Displaying 17 nodes, 16 relationships.

Figure 7: Adding overlap time.

The matching of the pathways is extended in this Cypher query by the addition of a temporal constraint. This query examines that the time periods of the visits to the common site overlap in addition to establishing a route between a "Sick" person and a "Healthy" person through it in order to find probable transmission events.

➢ Places visited by sick person after being infected.



```
neo4j$ MATCH (p:Person {healthstatus:"Sick"})-[v:VISITS]→(pl:Place) WHERE p.confirmedtime < v.starttime RETURN p, …
```

Overview

Node labels

* (10) Person (1) Place (9)

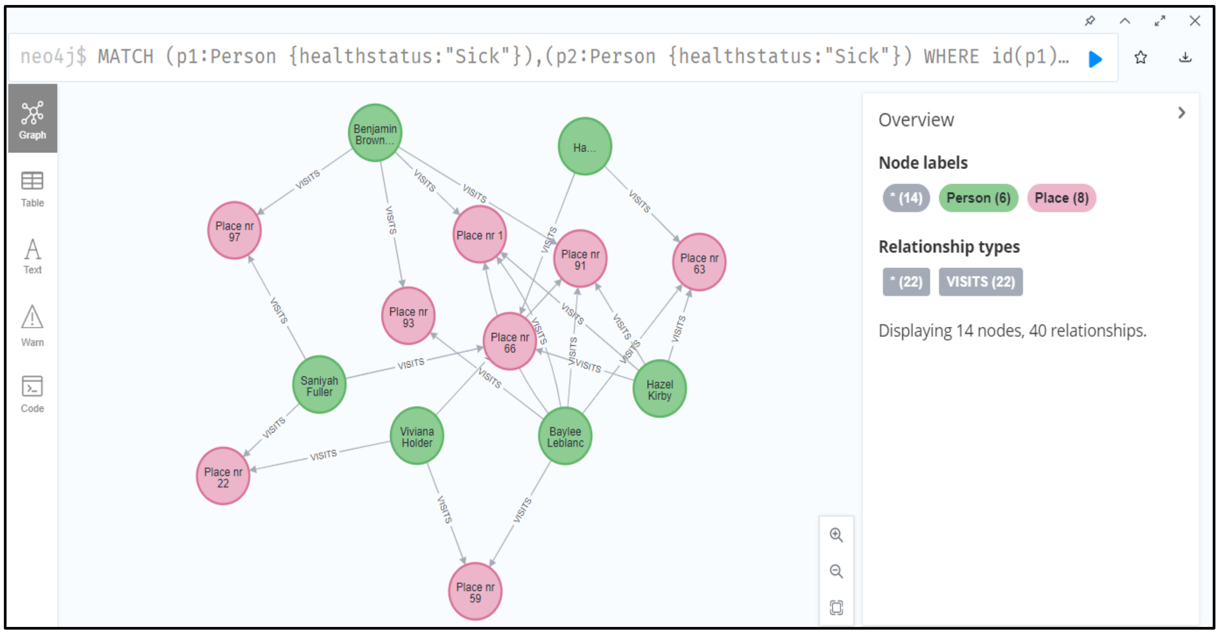
Relationship types

* (10) VISITS (10)

Displaying 10 nodes, 10 relationships.

Figure 8: Place visited by sick person after being infected.

This Cypher search is looking for a person (p) with the property value "Sick" for their "health status" and a visit (v) to a location (pl) made by that person, where the confirmed time of the visit is starting time is earlier than the confirmed time of the person's sickness. When trying to determine potential origins of infection for a particular person, this is a common pattern.



```
neo4j$ MATCH (p1:Person {healthstatus:"Sick"}),(p2:Person {healthstatus:"Sick"}) WHERE id(p1)…
```

Overview

Node labels

* (14) Person (6) Place (8)

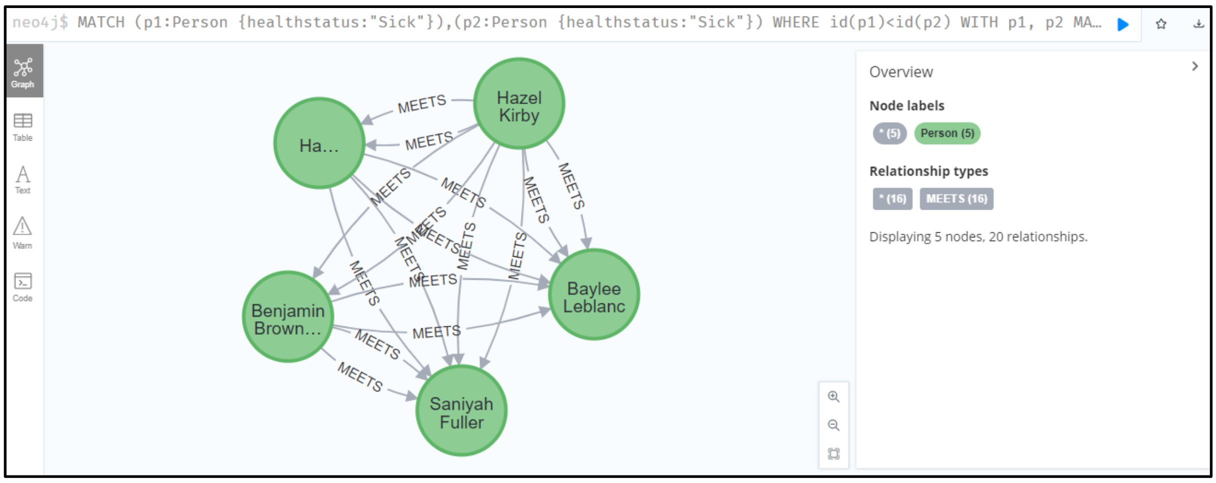
Relationship types

* (22) VISITS (22)

Displaying 14 nodes, 40 relationships.

Figure 9:

- Find connection between sick and healthy people.

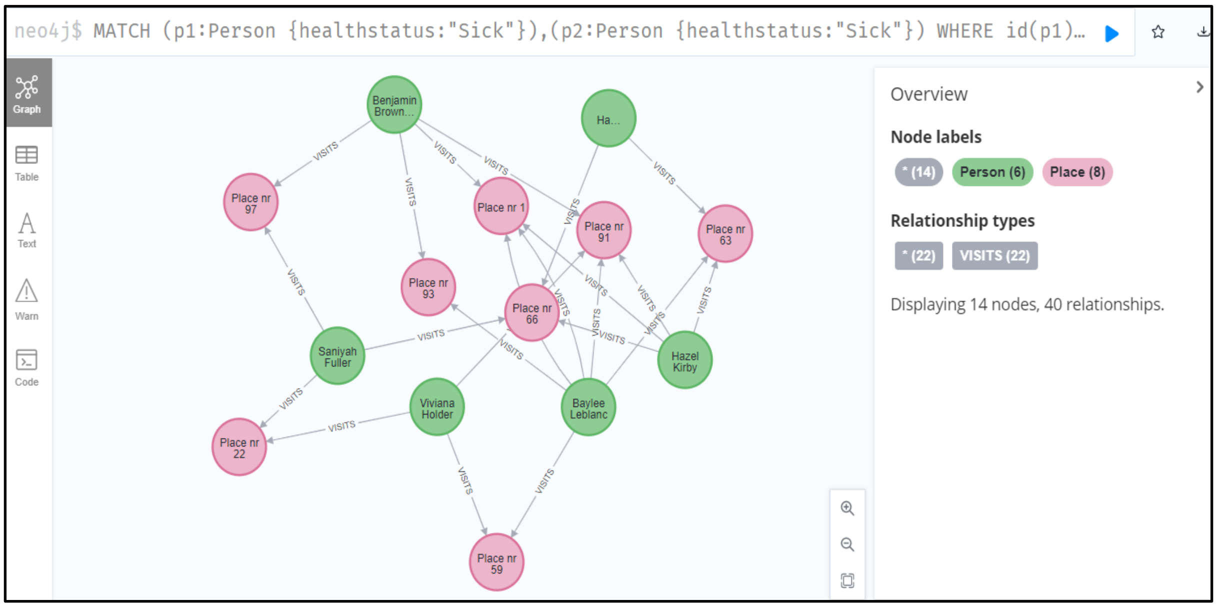


```
neo4j$ MATCH (p1:Person {healthstatus:"Sick"}),(p2:Person {healthstatus:"Sick"}) WHERE id(p1)<id(p2) WITH p1, p2 MA…
```

Displaying 5 nodes, 20 relationships.

Figure 10: Connections

The shortest pathways between each pair of "Sick" individuals (p1 and p2) in the graph are then determined using the Cypher algorithm.

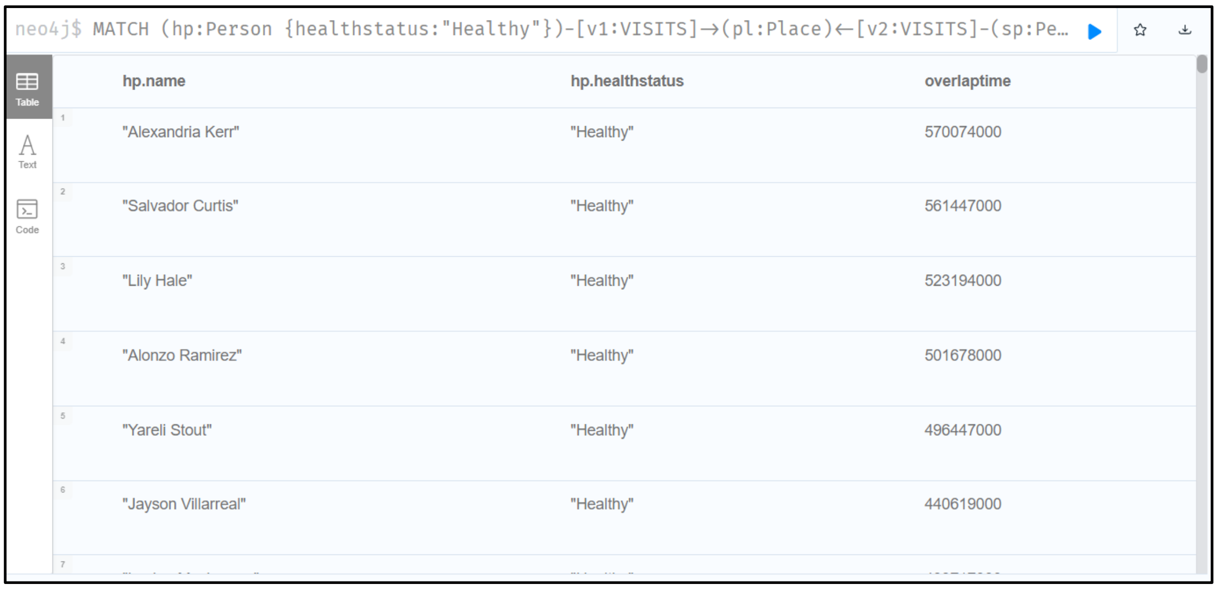


```
neo4j$ MATCH (p1:Person {healthstatus:"Sick"}),(p2:Person {healthstatus:"Sick"}) WHERE id(p1)…
```

Displaying 14 nodes, 40 relationships.

Figure 11:

- This Cypher query determines how much time the healthy person and sick person spent at the location together by searching for healthy people (hp) who have visited a location (pl) that a sick person (sp) has also visited.

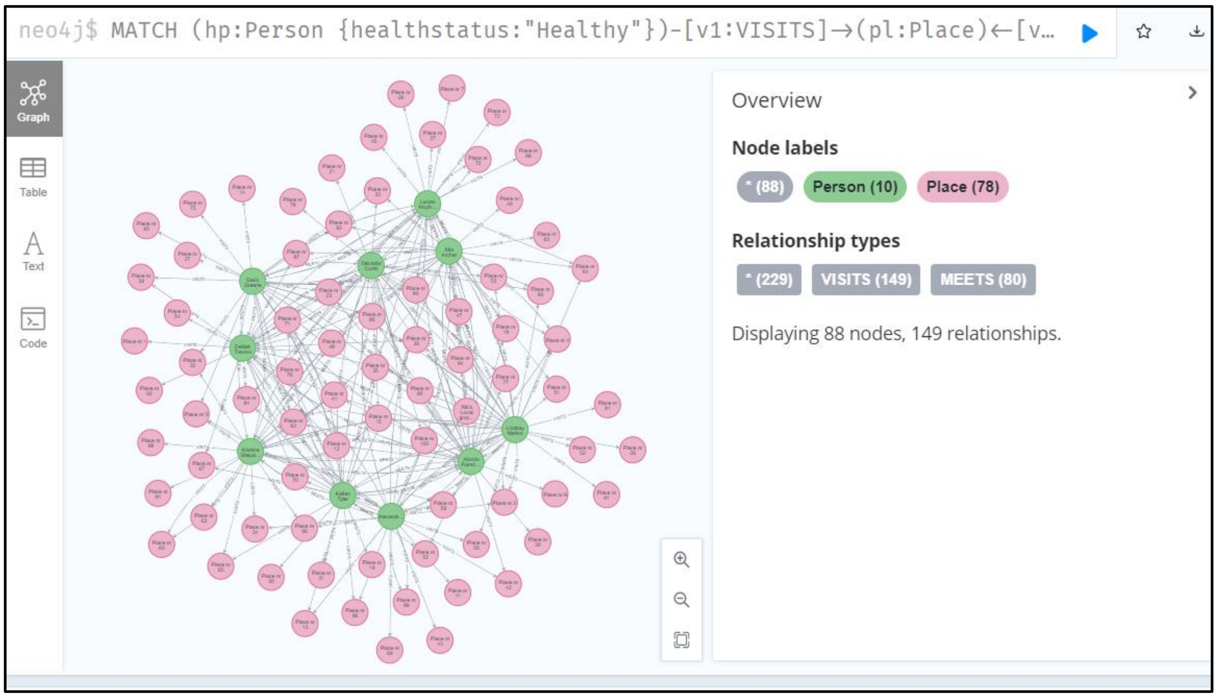


```
neo4j$ MATCH (hp:Person {healthstatus:"Healthy"})-[v1:VISITS]→(pl:Place)←[v2:VISITS]-(sp:Pe…
```

	hp.name	hp.healthstatus	overlaptime
1	"Alexandria Kerr"	"Healthy"	570074000
2	"Salvador Curtis"	"Healthy"	561447000
3	"Lily Hale"	"Healthy"	523194000
4	"Alonzo Ramirez"	"Healthy"	501678000
5	"Yareli Stout"	"Healthy"	496447000
6	"Jayson Villarreal"	"Healthy"	440619000
7			

Figure 12:

➢ Healthy person who has the highest risk graph



```
neo4j$ MATCH (hp:Person {healthstatus:"Healthy"})-[v1:VISITS]→(pl:Place)←[v…
```

Overview

Node labels

* (88) Person (10) Place (78)

Relationship types

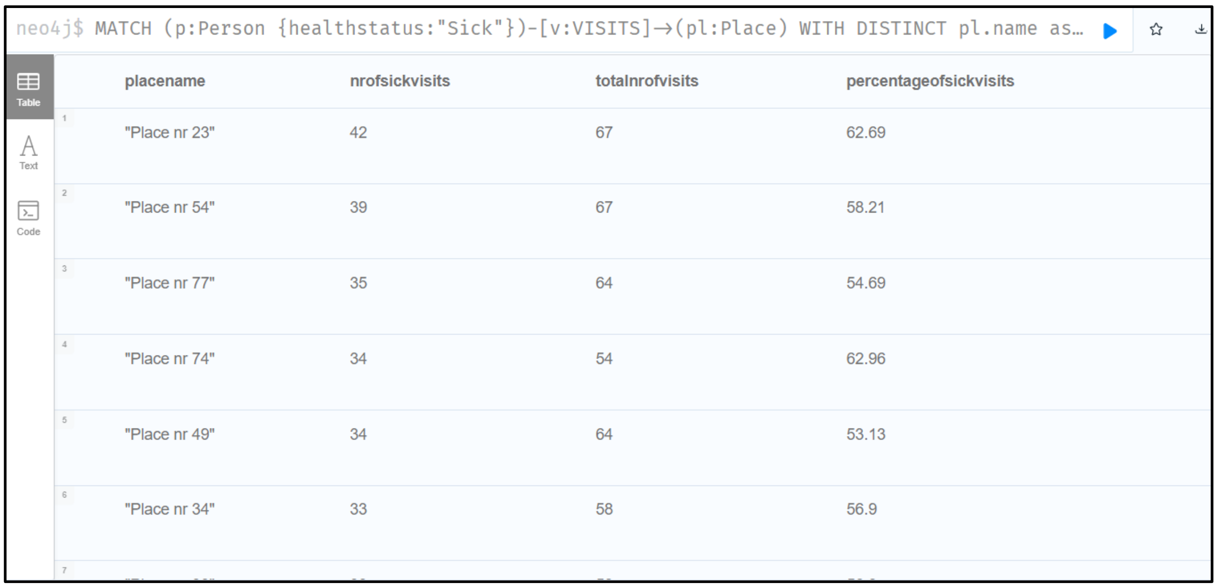
* (229) VISITS (149) MEETS (80)

Displaying 88 nodes, 149 relationships.

Figure 13: Person highest risk graph

This Cypher search is looking for the healthy person (hp) who has the greatest chance of contracting the disease based on the amount of time they spent around a sick person (sp) at a location (pl).

➢ Places with most sick visits -Table

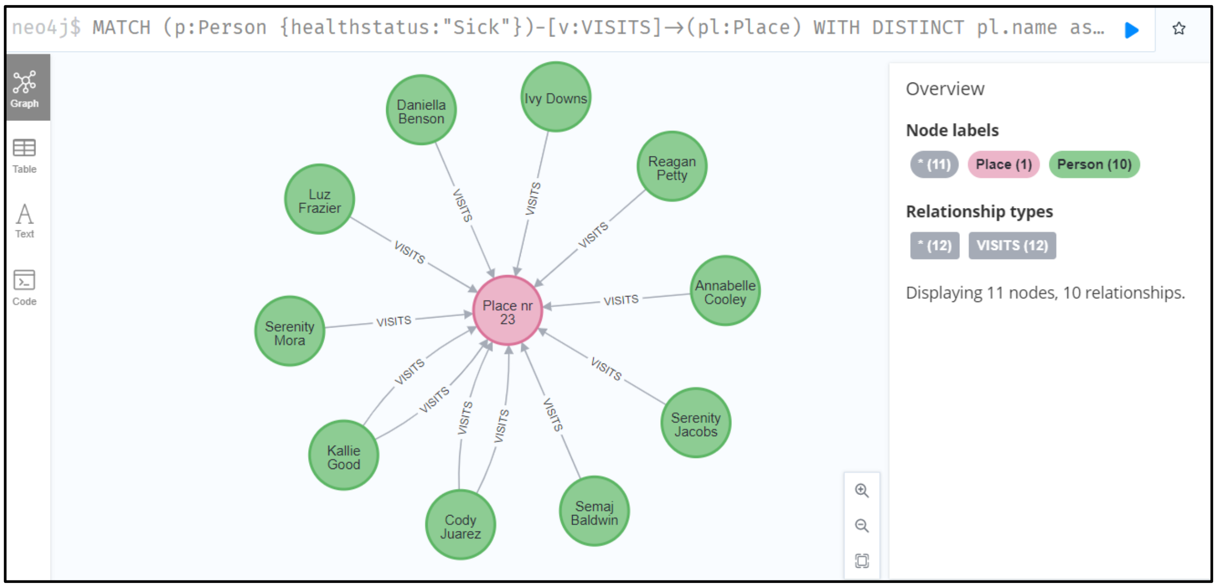


```
neo4j$ MATCH (p:Person {healthstatus:"Sick"})-[v:VISITS]→(pl:Place) WITH DISTINCT pl.name as…
```

	placename	nrofsickvisits	totalnrofvisits	percentageofsickvisits
1	"Place nr 23"	42	67	62.69
2	"Place nr 54"	39	67	58.21
3	"Place nr 77"	35	64	54.69
4	"Place nr 74"	34	54	62.96
5	"Place nr 49"	34	64	53.13
6	"Place nr 34"	33	58	56.9
7				

Figure 14: Places with most sick visits

➢ Place with most sick visits - Graph



```
neo4j$ MATCH (p:Person {healthstatus:"Sick"})-[v:VISITS]→(pl:Place) WITH DISTINCT pl.name as…
```

Overview

Node labels

* (11) Place (1) Person (10)

Relationship types

* (12) VISITS (12)

Displaying 11 nodes, 10 relationships.

Figure 15: Place with most sick visit graph

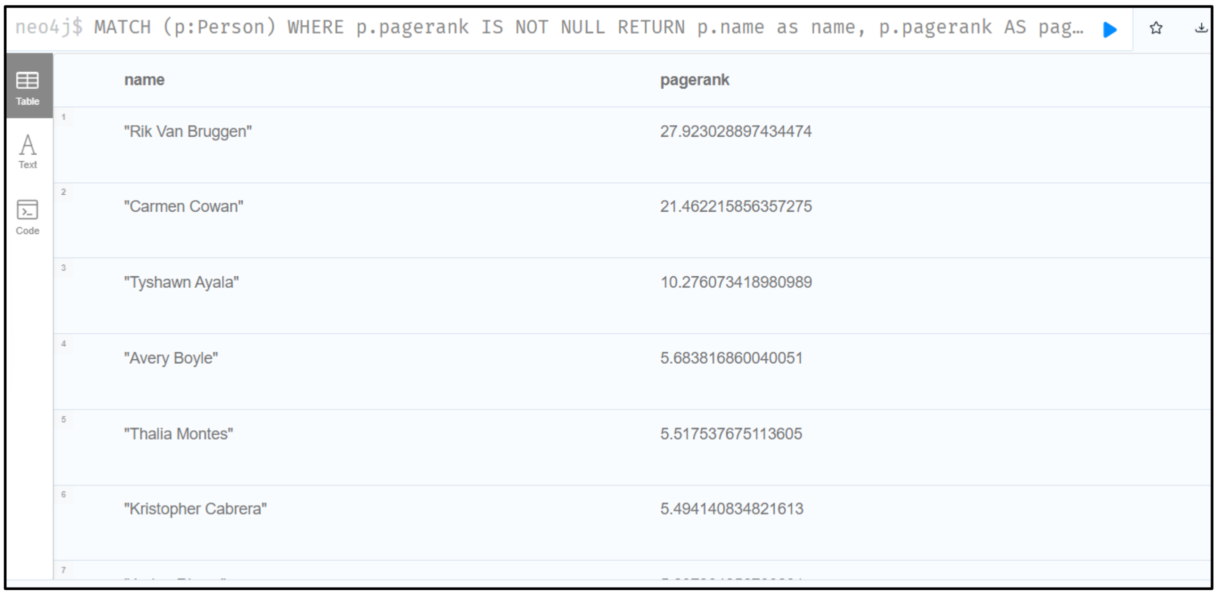
4) MODELING

Neo4j's data modelling platform offers a practical method for extracting knowledge from data. It is feasible to finish links between commodities and find hidden connections between the data representation by utilizing Neo4j's graph structure.
Users of Neo4j can benefit greatly from using data modelling to better understand the modelling of data sets. Neo4j is a great tool for data modelling because it offers a robust and customizable graph database that enables users to quickly and effectively view and analyse data. By requiring links between distinct data elements, Neo4j's data modelling may be leveraged to provide insights and enable users to easily identify patterns and correlations.

- **PageRank**:

Web search engines employ the well-known PageRank algorithm to gauge the significance or popularity of web pages. The fundamental tenet of PageRank is that a page is significant if significant pages link to it. In other words, a website's PageRank score will increase the more incoming connections it gets from other websites. The ranking of the page being linked to will also be significantly influenced by the ranking of the page linking to it. Each page receives an initial score from the PageRank algorithm, which is then iteratively modified based on the scores of pages that connect to it. This procedure is repeated until a steady value is reached for all page scores.

Page rank to distinguish the healthiest individual who will be impacted by a sick person. In page rank, we are utilizing the write feature.



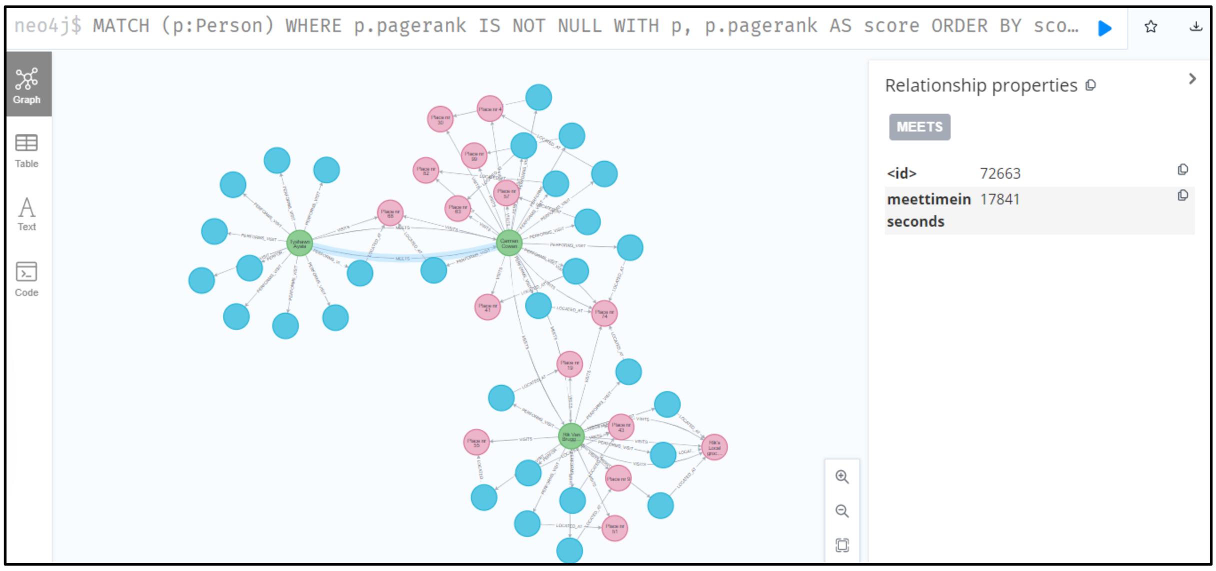
```
neo4j$ MATCH (p:Person) WHERE p.pagerank IS NOT NULL RETURN p.name as name, p.pagerank AS pag…
```

	name	pagerank
1	"Rik Van Bruggen"	27.923028897434474
2	"Carmen Cowan"	21.462215856357275
3	"Tyshawn Ayala"	10.276073418980989
4	"Avery Boyle"	5.683816860040051
5	"Thalia Montes"	5.517537675113605
6	"Kristopher Cabrera"	5.494140834821613
7	[illegible]	[illegible]

Figure 16: Top page rank person

From the above query we get to know that Rik Van Bruggen is have the highest page rank followed by Carmen Cowan

The below graph shows Top Page Rank Persons graphs.



```
neo4j$ MATCH (p:Person) WHERE p.pagerank IS NOT NULL WITH p, p.pagerank AS score ORDER BY sco…
```

Relationship properties

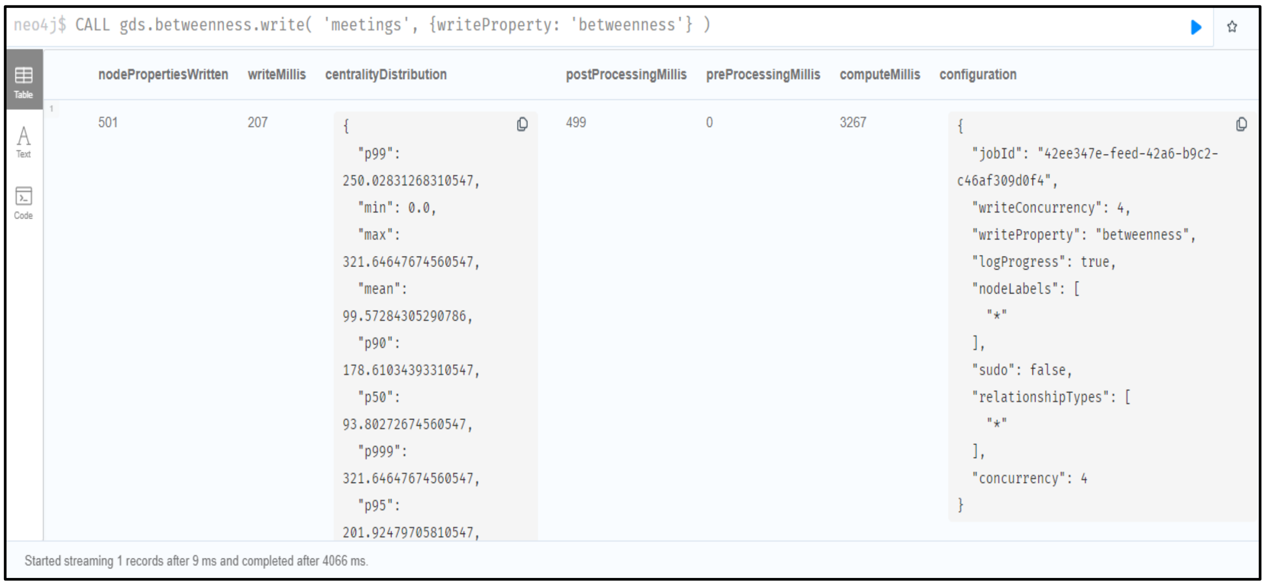
MEETS

<id>	72663
meettimein seconds	17841

Figure 17:Top page rank persons graph

- **Betweenness Centrality**

The ability of a node or edge to serve as a link or middleman between other nodes or edges in the network determines its relevance in a network, which is known as betweenness centrality. In more detail, it assesses the percentage of shortest paths in the network that traverse a specific node or edge. As they allow the movement of resources or information across various regions of the network, nodes or edges with high betweenness centrality are seen as being essential to the connection and operation of the network. Low betweenness centrality nodes or edges, on the other hand, could not be as crucial to the networks overall structure and functionality.



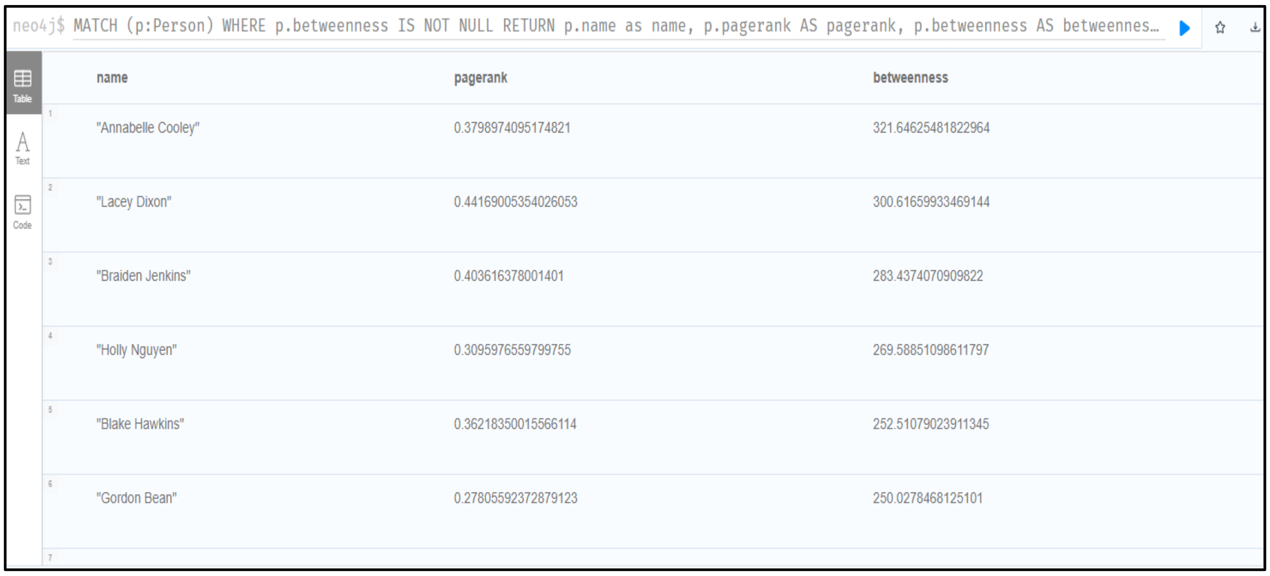
```
neo4j$ CALL gds.betweenness.write( 'meetings', {writeProperty: 'betweenness'} )
```

nodePropertiesWritten	writeMillis	centralityDistribution	postProcessingMillis	preProcessingMillis	computeMillis	configuration
501	207	{ "p99": 250.02831268310547, "min": 0.0, "max": 321.64647674560547, "mean": 99.57284305290786, "p90": 178.61034393310547, "p50": 93.80272674560547, "p999": 321.64647674560547, "p95": 201.92479705810547,	499	0	3267	{ "jobId": "42ee347e-feed-42a6-b9c2-c46af309d0f4", "writeConcurrency": 4, "writeProperty": "betweenness", "logProgress": true, "nodeLabels": ["*"], "sudo": false, "relationshipTypes": ["*"], "concurrency": 4 }

Started streaming 1 records after 9 ms and completed after 4066 ms.

Figure 18: Betweenness centrality

- Top person based on betweenness centrality - table.



```
neo4j$ MATCH (p:Person) WHERE p.betweenness IS NOT NULL RETURN p.name as name, p.pagerank AS pagerank, p.betweenness AS betweennes…
```

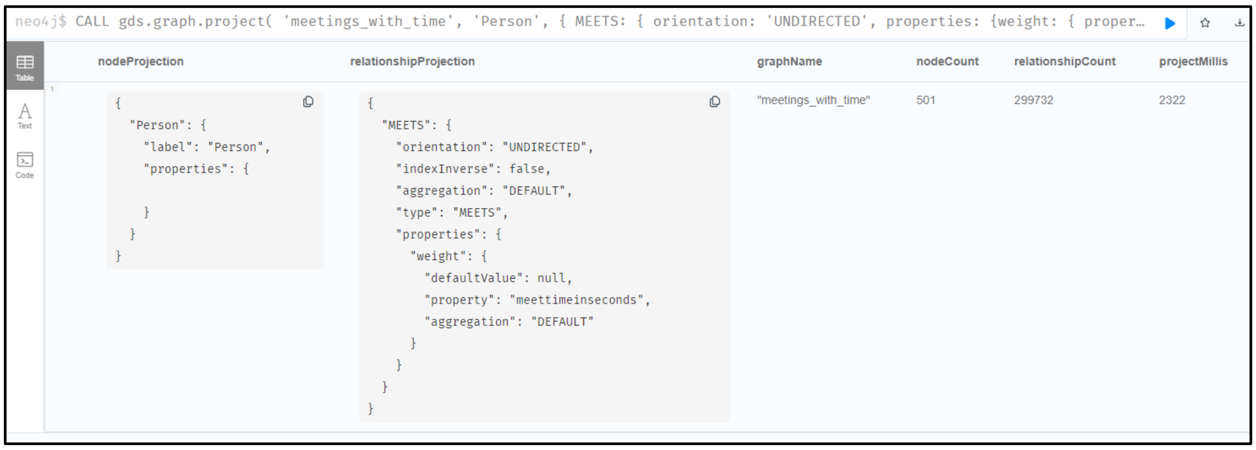
	name	pagerank	betweenness
1	"Annabelle Cooley"	0.3798974095174821	321.64625481822964
2	"Lacey Dixon"	0.44169005354026053	300.61659933469144
3	"Braiden Jenkins"	0.403616378001401	283.4374070909822
4	"Holly Nguyen"	0.3095976559799755	269.58851098611797
5	"Blake Hawkins"	0.36218350015566114	252.51079023911345
6	"Gordon Bean"	0.27805592372879123	250.0278468125101
7			

Figure 19: Centrality table

- **Louvain community detection**

  A modularity function is optimized by the Louvain method, a hierarchical clustering algorithm, to find communities within a network. The modularity function gauges how closely connected nodes inside a community are to one another as opposed to nodes outside of the community. In order to maximize the modularity score, communities of nodes are iteratively merged in the Louvain algorithm. The program uses the same procedure on the communities it has found at each level of the hierarchy until it reaches a point where it can no longer increase the modularity score.

  The Louvain algorithm is a powerful tool for analysing complex networks and has been applied in a wide range of fields, including social networks, biology, and computer science.



```
neo4j$ CALL gds.graph.project( 'meetings_with_time', 'Person', { MEETS: { orientation: 'UNDIRECTED', properties: {weight: { proper…
```

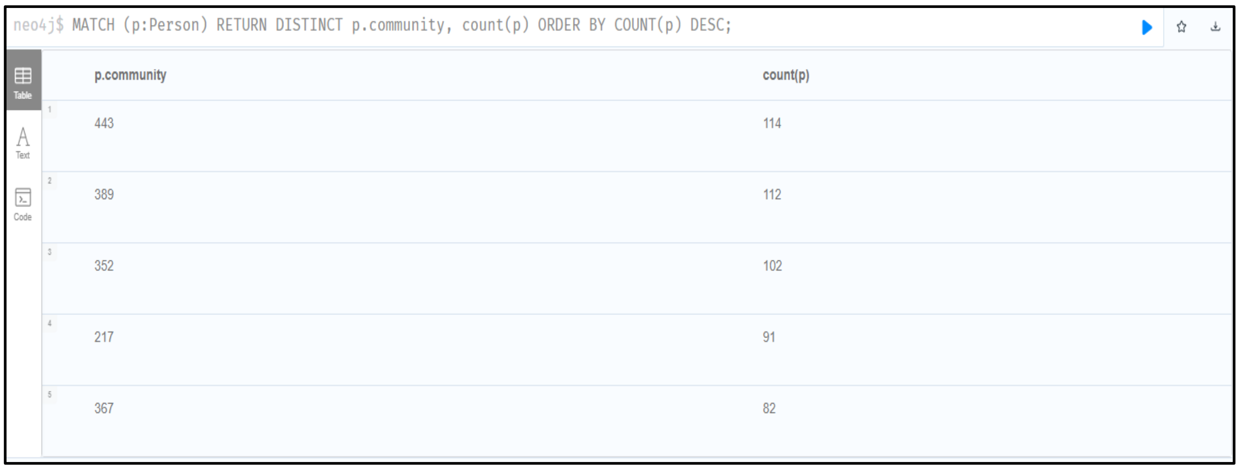
	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	`{ "Person": { "label": "Person", "properties": { } } }`	`{ "MEETS": { "orientation": "UNDIRECTED", "indexInverse": false, "aggregation": "DEFAULT", "type": "MEETS", "properties": { "weight": { "defaultValue": null, "property": "meettimeinseconds", "aggregation": "DEFAULT" } } } }`	"meetings_with_time"	501	299732	2322

Figure 20: Community detection

The Louvain algorithm is a community discovery algorithm that divides a graph into communities depending on the strength of connections between nodes. Integer values are used to distinguish the resulting communities.

- **Different Communities**

The Louvain algorithm has already been run, and the communities are kept in the nodes' "community" property. This query can be used to examine the graph's community structure and determine which communities have the most nodes. This data can be utilized to obtain understanding of the graph's organization, structure, and node-to-node interactions.



```
neo4j$ MATCH (p:Person) RETURN DISTINCT p.community, count(p) ORDER BY COUNT(p) DESC;
```

	p.community	count(p)
1	443	114
2	389	112
3	352	102
4	217	91
5	367	82

Figure 21: Different Communities

5) EVALUATION

These algorithms follow unsupervised machine learning and are non-deterministic in nature. To determine the quantity and quality of relationships coming into a Person node and the significance of the Person node, we use the page rank method. Using betweenness centrality, we may determine how a sick person's influence affects a healthy person. The following business use cases have been assessed by our team:

6) DEPLOYMENT

Most of the graphs used to describe the process of information storage use linked tests-like structures to store data. This might be done by saving the actual links to the data, as opposed to merely holding similar kinds of goods. Neo4j graphs are perfect for constructing complicated communication that can be transmitted over huge data. Result would be deployed in automated and implemented in an app or web app.

7) RESULTS

Exploratory data analysis

On the Neo4j Knowledge Graph, the following visualizations are executed.

Step1: Connecting neo4j and python.

```
graph = Graph("bolt://44.211.162.203:7687", auth=("neo4j", "monitors-origins-morphine"))
```

Step2: Running following query to check nodes and database.

```
graph.run("CALL db.schema.visualization()").data()
```

Step3: Visualization

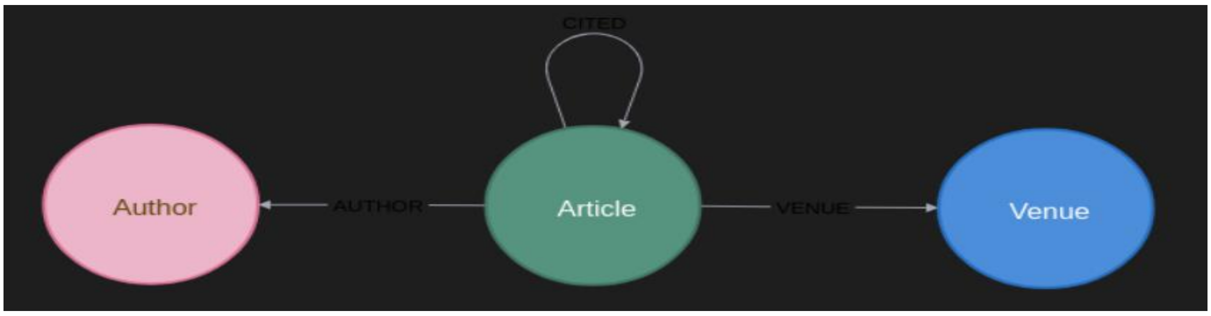


Figure 22: Visualization

Step4: Visualizing the counts using matplotlib.

```
nodes_df.plot(kind='bar', x='label', y='count', legend=None, title="Node Cardinalities")
plt.yscale("log")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

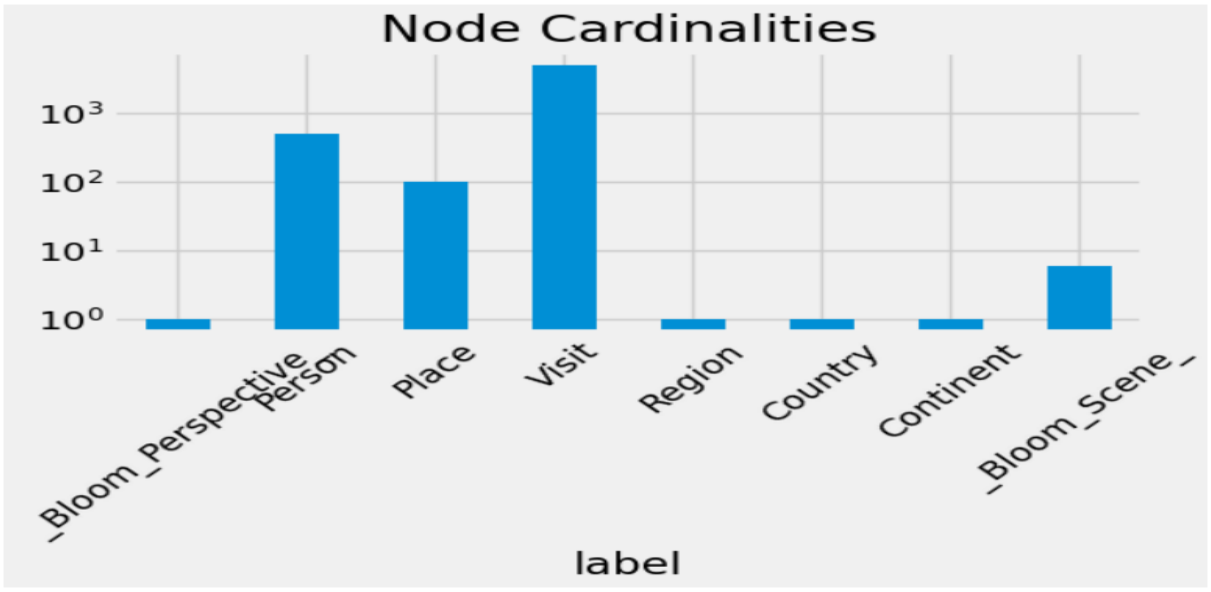
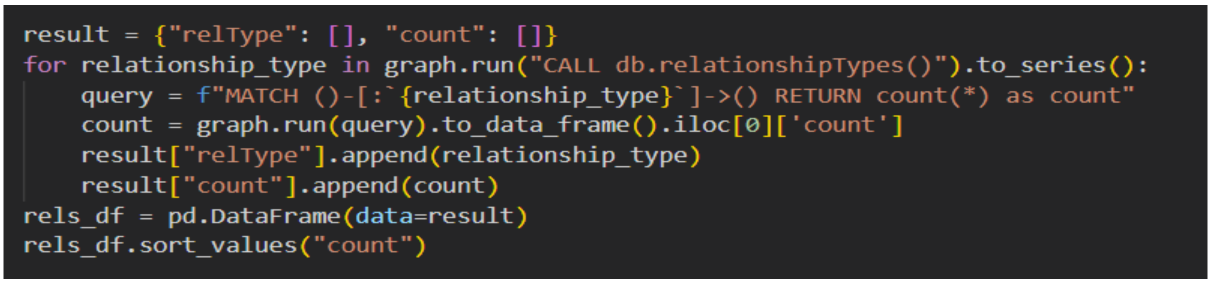


Figure 23: Node Cardinalities

Step5: Relationship Cardinalities visualizing using matplotlib code.



```
result = {"relType": [], "count": []}
for relationship_type in graph.run("CALL db.relationshipTypes()").to_series():
    query = f"MATCH ()-[:`{relationship_type}`]->() RETURN count(*) as count"
    count = graph.run(query).to_data_frame().iloc[0]['count']
    result["relType"].append(relationship_type)
    result["count"].append(count)
rels_df = pd.DataFrame(data=result)
rels_df.sort_values("count")
```

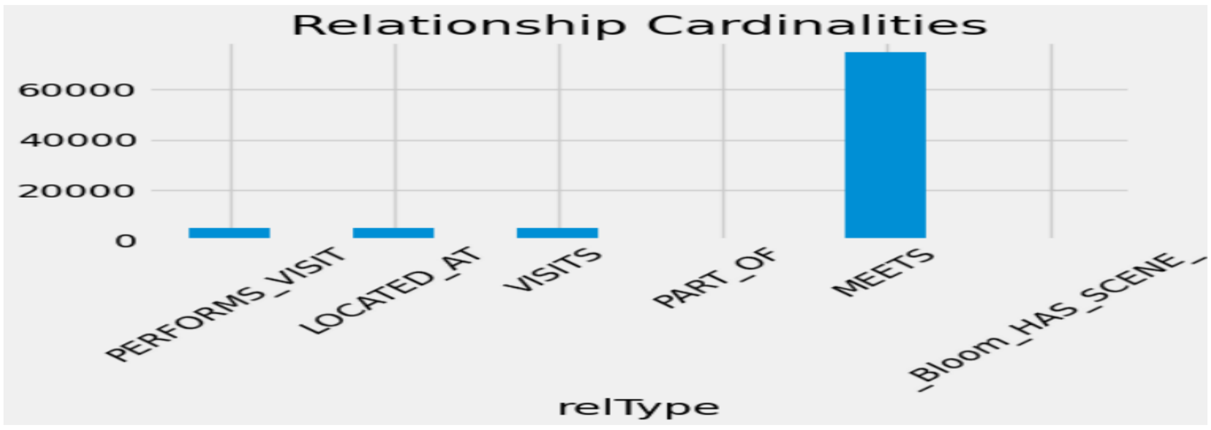
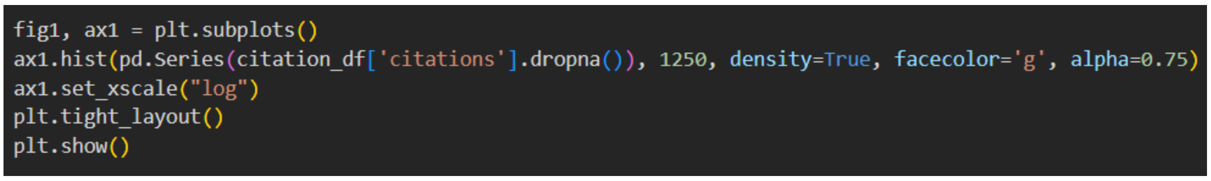


Figure 24: Relationship Cardinalities

Step6: Citation data and using a log scale histogram to visualize the data.



```
fig1, ax1 = plt.subplots()
ax1.hist(pd.Series(citation_df['citations'].dropna()), 1250, density=True, facecolor='g', alpha=0.75)
ax1.set_xscale("log")
plt.tight_layout()
plt.show()
```

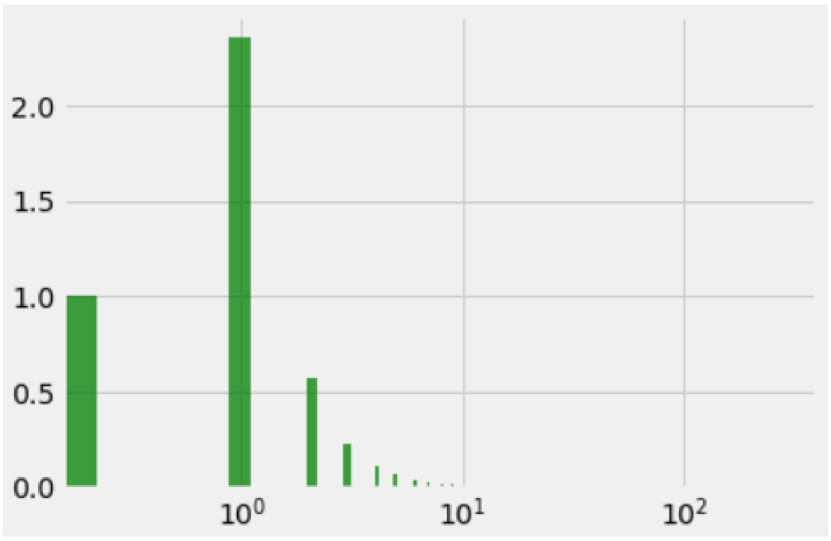


Figure 25: Citation data

8) CONCLUSION

The algorithm utilized for this research has been thoroughly presented, and I would like to draw a conclusion. The algorithm from Neo4j has been applied in these circumstances and according to the use case. A real-time application in Neo4j was created using the technical knowledge of the various techniques using the open-source dataset Contact Tracing. We were able to comprehend the relationship between the COVID 19 sick people and the healthy people by using the knowledge graph and data science techniques including PageRank, overlap similarity, degree of centrality, betweenness centrality, and Louvain community detection. It was extremely difficult to load the large amount of data and identify relationships within the data in order to design the graph data structure without impairing the real relationships. Therefore, I would like to draw the conclusion that employing knowledge graphs is very efficient and allows queries to execute quickly. It also allows us to expand the database, if necessary, without impairing present functionality.

9) REFERENCES

Klinkenberg, D., Fraser, C. and Heesterbeek, H., (2006). The effectiveness of contact tracing in emerging epidemics. PloS one, 1(1), p.e12.

Neo4j sandbox: https://neo4j.com/blog/extensive-representation-of-contact-tracing-with-neo4j-in-the-canton-of-geneva/

Needham, M. and Hodler A.E. (2019) *Graph Algorithms*. O’Reilly Media.

Needham, M. and Hodler, A (2019). Graph Algorithms - Practical Examples in Apache Spark and Neo4j. O'Reilly.

Robinson, I., Webber, J. and Eifrem, E. (2015). Graph Databases 2nd edn. O’Reilly Media.