Date:15.04.2022

Third Year B. Tech., Sem VI 2021-22

4CS372 : Advanced Database System Lab

Assignment Submission

PRN No: 2019BTECS00064

Full name: Kunal Santosh Kadam

Batch: T2

Assignment: 12

Title of assignment: Spatial and Geographic Data using Neo4j

Problem Statement:

Finding Things Close to Other Things. Application in : location-based services on the web

Task:

1. Use Neo4j graph database installed in previous assignments.
2. Install/configure Neo4jSpatial (https://github.com/neo4jcontrib/spatial) from GitHub. It is the Neo4j plug-in that facilitates geospatial operations on data stored in Neo4j.
3. Write CQL (Cypher Query Language) script to add randomly 10,000 location points as follows. Assume any data.
4. Use the point() , distance() function of Neo4j to answer the queries “which things close/nearest to which other things”.
5. Demonstrate the result by firing different cypher queries (write CQL statement).

Objective / Aim :

1. Install/configure Neo4jSpatial
2. Use the point() , distance() function of Neo4j to answer the queries "which things close/nearest to which other things".
3. Demonstrate the result by firing different cypher queries (write CQL statement).

Introduction:

Neo4j is a native graph database platform, built from the ground up to leverage not only data but also data relationships. Neo4j connects data as it's stored, enabling queries never before imagined, at speeds never thought possible.

Geography is a natural domain for graphs and graph databases.So natural, in fact, that early map users of Neo4j simply rolled their own map support. However, it takes some effort to deal with spatial indexes, geometries and topologies, and so, since September 2010, the Neo4j Spatial project has been providing early access releases enabling a wide range of convenient and powerful geographic capabilities in the Neo4j database.

Theory:

Today's CIOs and CTOs don't just need to manage larger volumes of data they need to generate insight from their existing data. In this case, the relationships between data points matter more than the individual points themselves.

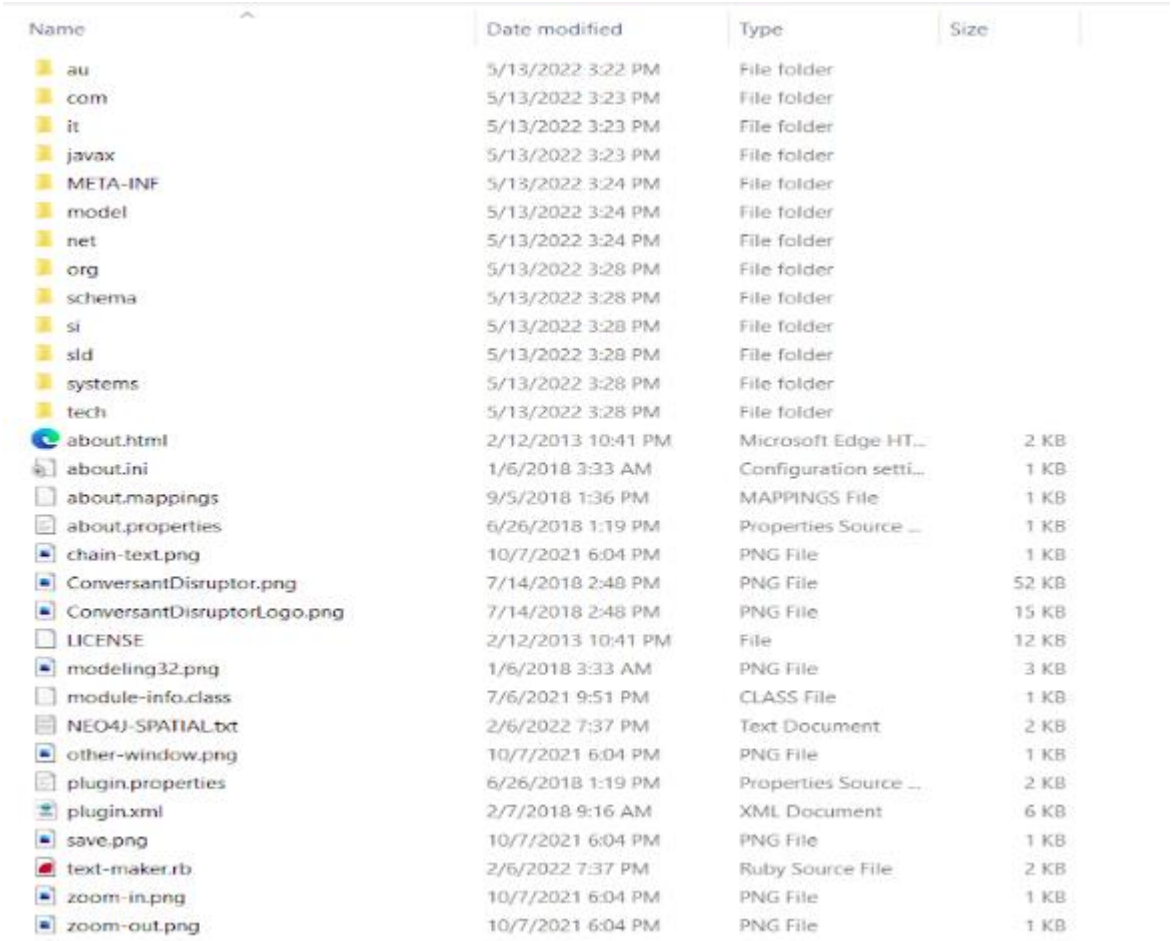
In order to leverage data relationships, organizations need a database technology that stores relationship information as a first-class entity. That technology is a graph database.

Ironically, legacy relational database management systems (RDBMS) are poor at handling data relationships. Their rigid schemas make it difficult to add different connections or adapt to new business requirements.

Not only do graph databases effectively store data relationships; but they're also flexible when expanding a data model or conforming to changing business needs.

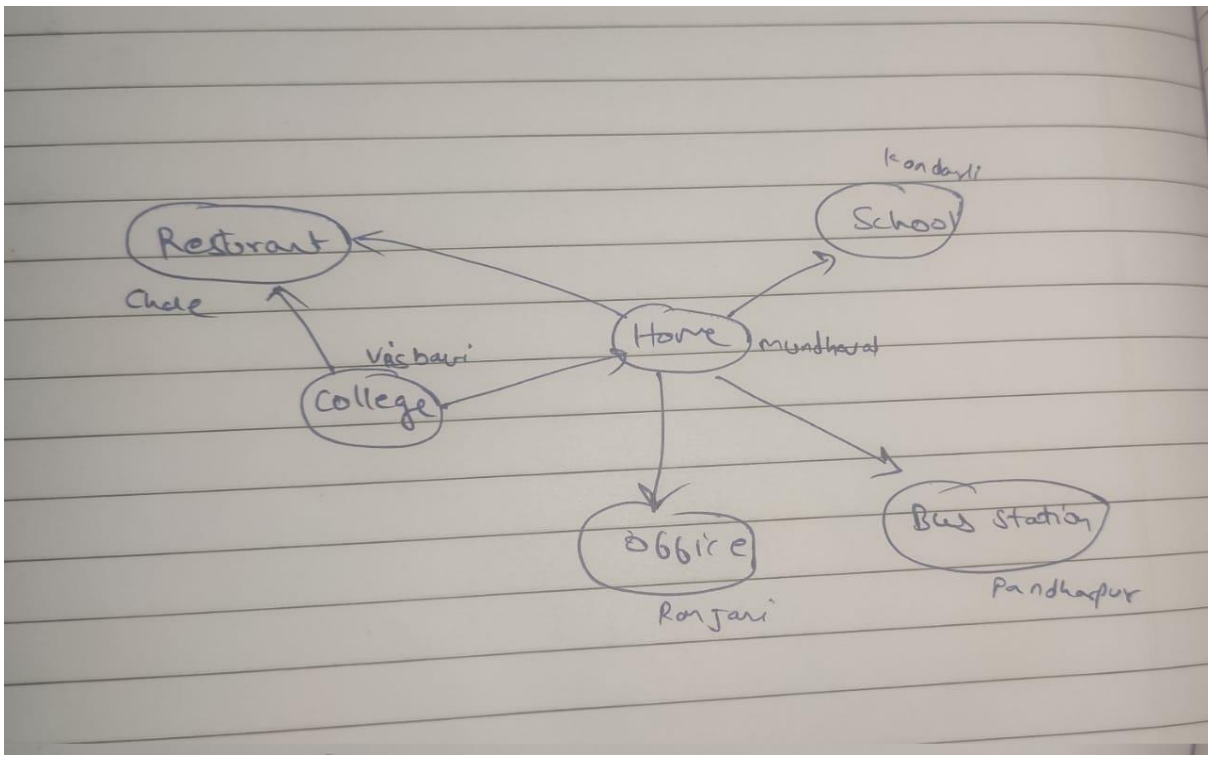
Procedure:

1. Install the Neo4jSpatial (https://github.com/neo4jcontrib/spatial) from GitHub. It is the Neo4j plug-in that facilitates geospatial operations on data stored in Neo4j. Unzip that zip/rar file and past in plugin folder in neo4j cluster and restart neo4j server.



Name	Date modified	Type	Size
au	5/13/2022 3:22 PM	File folder	
com	5/13/2022 3:23 PM	File folder	
it	5/13/2022 3:23 PM	File folder	
javax	5/13/2022 3:23 PM	File folder	
META-INF	5/13/2022 3:24 PM	File folder	
model	5/13/2022 3:24 PM	File folder	
net	5/13/2022 3:24 PM	File folder	
org	5/13/2022 3:28 PM	File folder	
schema	5/13/2022 3:28 PM	File folder	
si	5/13/2022 3:28 PM	File folder	
sld	5/13/2022 3:28 PM	File folder	
systems	5/13/2022 3:28 PM	File folder	
tech	5/13/2022 3:28 PM	File folder	
about.html	2/12/2013 10:41 PM	Microsoft Edge HT...	2 KB
about.ini	1/6/2018 3:33 AM	Configuration setti...	1 KB
about.mappings	9/5/2018 1:36 PM	MAPPINGS File	1 KB
about.properties	6/26/2018 1:19 PM	Properties Source ...	1 KB
chain-text.png	10/7/2021 6:04 PM	PNG File	1 KB
ConversantDisruptor.png	7/14/2018 2:48 PM	PNG File	52 KB
ConversantDisruptorLogo.png	7/14/2018 2:48 PM	PNG File	15 KB
LICENSE	2/12/2013 10:41 PM	File	12 KB
modeling32.png	1/6/2018 3:33 AM	PNG File	3 KB
module-info.class	7/6/2021 9:51 PM	CLASS File	1 KB
NEO4J-SPATIAL.txt	2/6/2022 7:37 PM	Text Document	2 KB
other-window.png	10/7/2021 6:04 PM	PNG File	1 KB
plugin.properties	6/26/2018 1:19 PM	Properties Source ...	2 KB
plugin.xml	2/7/2018 9:16 AM	XML Document	6 KB
save.png	10/7/2021 6:04 PM	PNG File	1 KB
text-maker.rb	2/6/2022 7:37 PM	Ruby Source File	2 KB
zoom-in.png	10/7/2021 6:04 PM	PNG File	1 KB
zoom-out.png	10/7/2021 6:04 PM	PNG File	1 KB

2. Create a graph database with properties such as City, Lattitude, and Longitude



1. Export the cipher formats of created graph:

Create:

CREATE (Home {City: "Mundhewadi", Lattitude: 17.691401, Longitude: 74.000938})-[:Home_to_restaurant]->({City: "Chale", Lattitude: 16.8541887, Longitude: 74.642293})<-[:College_to_restaurant]-({City: "Visbavi", Lattitude: 16.867634, Longitude: 74.56417})-[:College_to_Home]->(Home)-[:Home_to_office]->({City: "Ranjani", Lattitude: 17.370000, Longitude: 73.90000}),
({City: "Pandharpur", Lattitude: 17.696634, Longitude: 74.160721})<-[:Home_to_busstation]-(Home)-[:Home_to_school]->({City: "Kondharki", Lattitude: 17.286501, Longitude: 74.181427})

Match:

MATCH path0 = (Home {City: "Mundhewadi", Lattitude: 17.691401, Longitude: 74.000938})-[:Home_to_restaurant]->({City: "Chale", Lattitude: 16.8541887, Longitude: 74.642293})<-[:College_to_restaurant]-({City: "Visbavi", Lattitude: 16.867634,

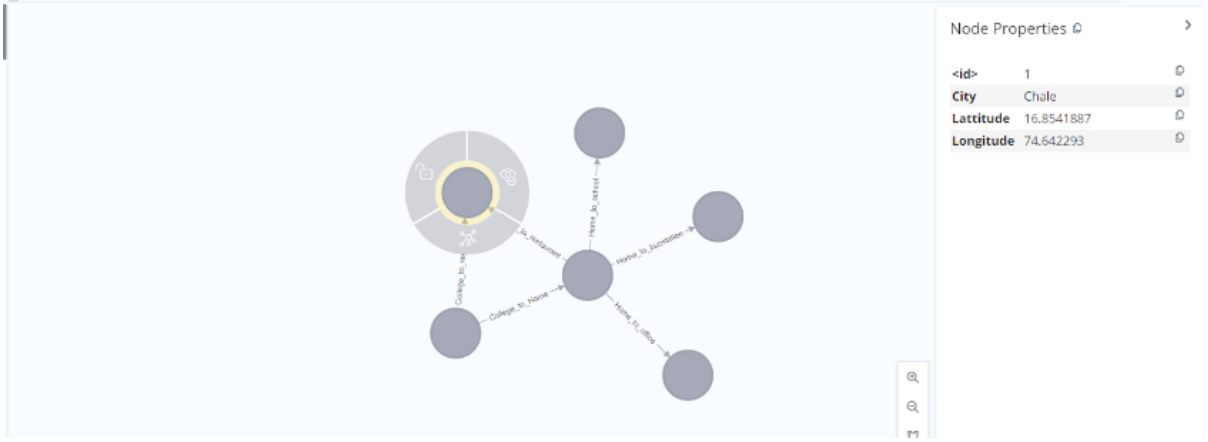
Longitude: 74.56417})-[:College_to_Home]->(Home)-[:Home_to_office]->({City: "Ranjani", Lattitude: 17.370000, Longitude: 73.90000}),
path1 = ({City: "Pandharpur", Lattitude: 17.696634, Longitude: 74.160721})<-[:Home_to_busstation]-(Home)-[:Home_to_school]->({City: "Kondharki", Lattitude: 17.286501, Longitude: 74.181427})
RETURN path0, path1

2. Run these 2 cipher queries in the Neo4j graph database:

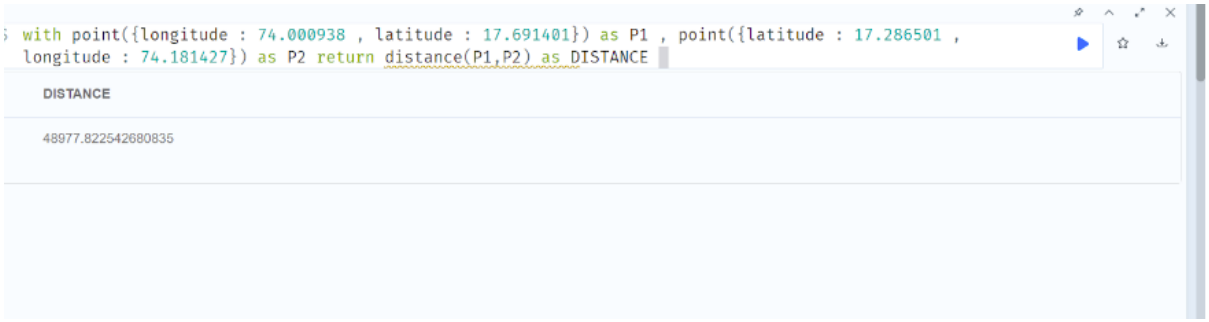
```
CREATE (Home {City: "Mundhewadi", Lattitude: 17.691401, Longitude: 74.000938})-[:Home_to_restaurant]→({City:
"Chale", Lattitude: 16.8541887, Longitude: 74.642293})←[:College_to_restaurant]-({City: "Visbavi", Lattitude:
16.867634, Longitude: 74.56417})-[:College_to_Home]→(Home)-[:Home_to_office]→({City: "Ranjani", Lattitude:
17.370000, Longitude: 73.90000}),
({City: "Pandharpur", Lattitude: 17.696634, Longitude: 74.160721})←[:Home_to_busstation]-(Home)-
[:Home_to_school]→({City: "Kondharki", Lattitude: 17.286501, Longitude: 74.181427})
```

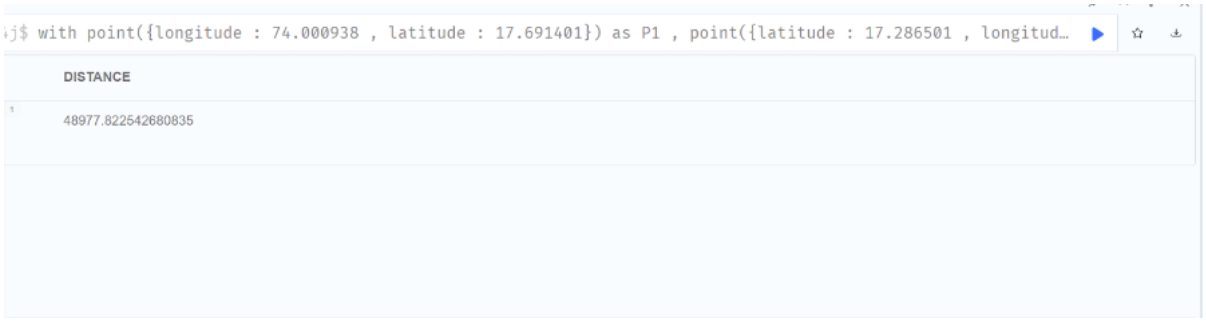
Created 6 nodes, set 18 properties, created 6 relationships, completed after 3 ms.

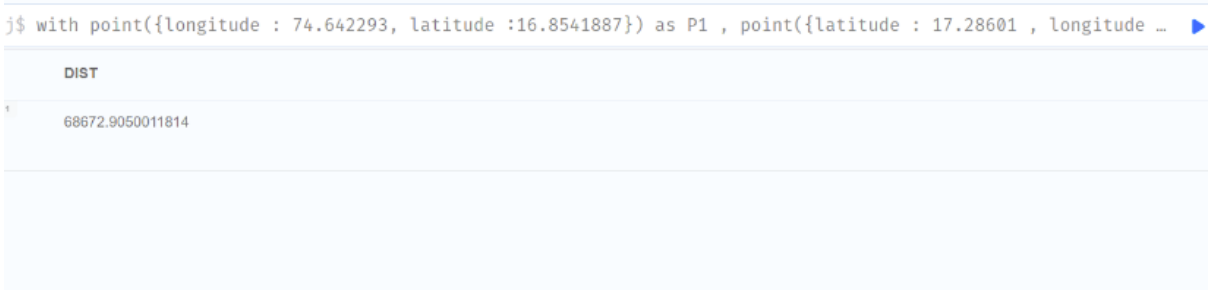
```
Lattitude: 16.867634, Longitude: 74.56417})-[:College_to_Home]→(Home)-[:Home_to_office]→({City: "Ranjani",
Lattitude: 17.370000, Longitude: 73.90000}),
path1 = ({City: "Pandharpur", Lattitude: 17.696634, Longitude: 74.160721})←[:Home_to_busstation]-(Home)-
[:Home_to_school]→({City: "Kondharki", Lattitude: 17.286501, Longitude: 74.181427})
RETURN path0, path1
```



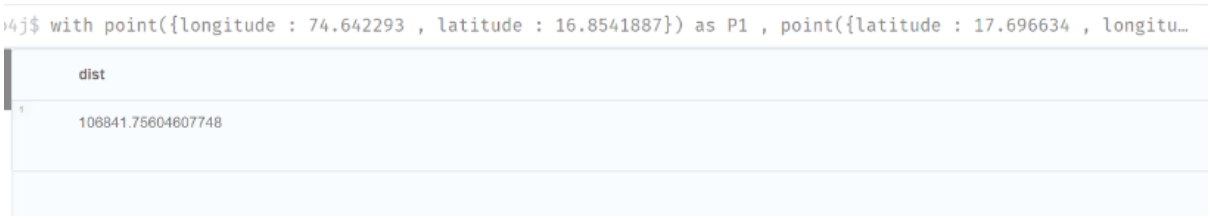
3. With point() and distance() functions finding the distance between locations by taking home as a reference point:







4. Distance between a restaurant in chale and the bus station in Pandharpur



Results:

With distance() and point() functions we found out the distance between the restaurant in chale and the bus station in Pandharpur. And found many other distances from home to other places.

Conclusion:

With Neo4J graph database and geospatial data, we find the close/ nearest place with respect to reference points inefficiently way.