Assignment – 12

**Spatial and Geographic Data Geospatial is the natural domain for Graph Database Use Neo4j and Neo4j Spatial**.

**Introduction:**

Neo4j is a native graph database platform, built from the ground up

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; they’re also flexible when expanding a data model or conforming to

changing business needs. One of the simplest and most intuitive places to start

is to ask the question: How do I find things close to other things? This is exactly

the question answered by location-based services on the web, as well as a

number of existing spatial databases. In the NoSQL area, CouchDD released

GeoCouch in 2009, and MongoDD released their geohashing index in 2010. Both

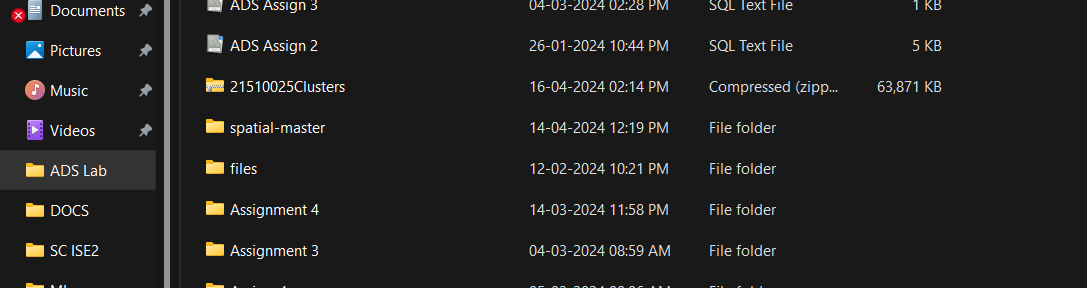
answer exactly this question. Unlike these other NoSQL databases, Neo4j

started with support for complex geometries in 2010. While simple proximity

searches have been possible, they have only recently become simple and

intuitive.

INSTALLATION :

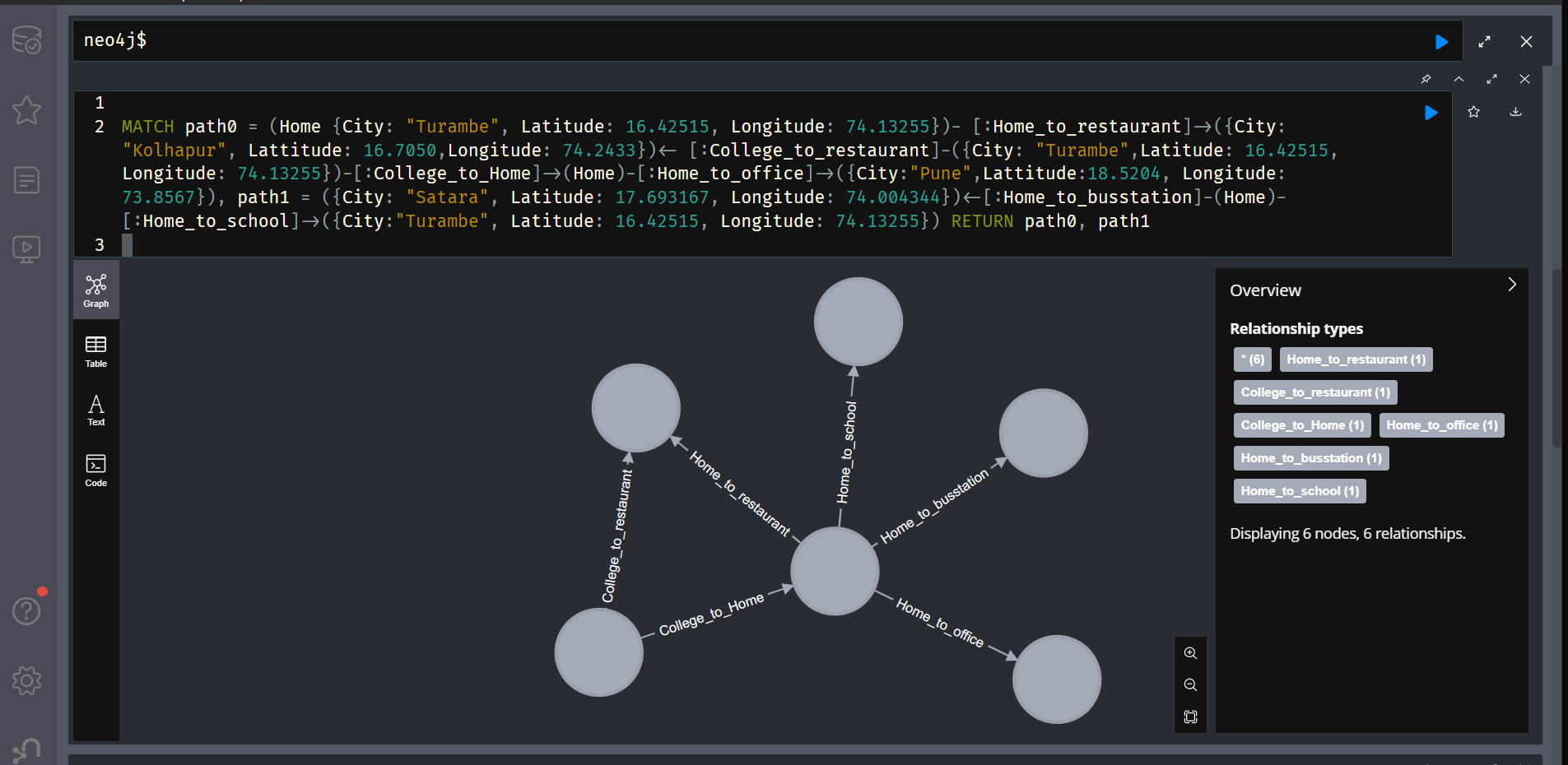


* RUNNING THE CYPHER QUERY in NEO4J DATABASE

CREATE (Home {City: "Turambe", Latitude: 16.42515, Longitude: 74.13255})-[:Home\_to\_restaurant]->({City: "Kolhapur", Lattitude: 16.7050,Longitude: 74.2433})<-[:College\_to\_restaurant]-({City: "Turambe",Latitude: 16.42515, Longitude: 74.13255})-[:College\_to\_Home]->(Home)-[:Home\_to\_office]->({City: "Pune", Lattitude:18.5204, Longitude:73.8567}), ({City: "Satara", Latitude: 17.693167, Longitude: 74.004344})<-[:Home\_to\_busstation]-(Home)-[:Home\_to\_school]- >({City:"Turambe", Latitude: 16.42515, Longitude: 74.13255})

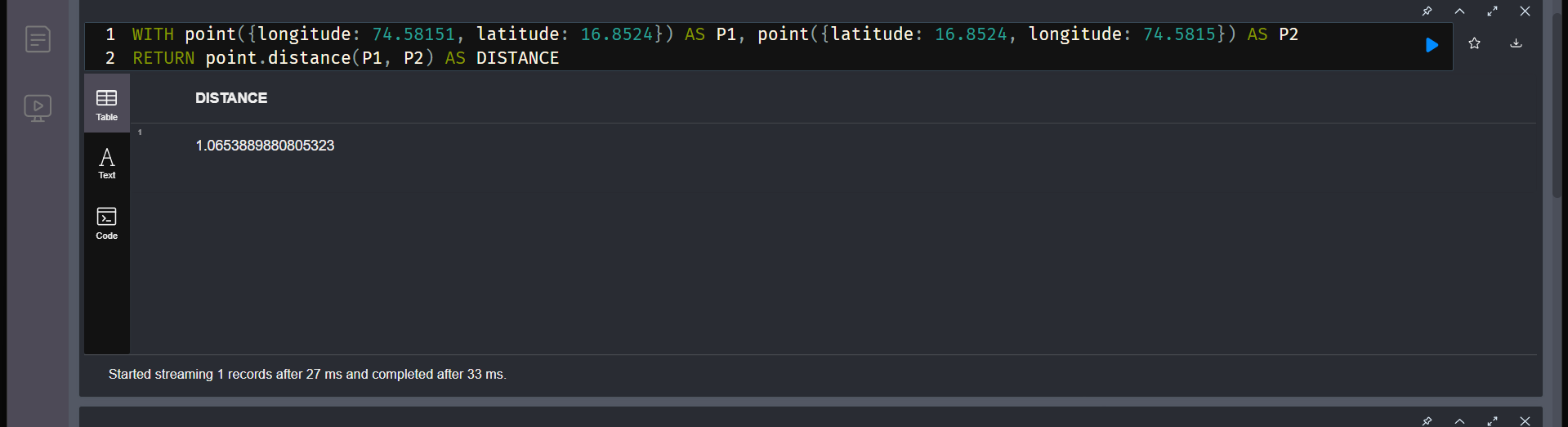


MATCH path0 = (Home {City: "Turambe", Latitude: 16.42515, Longitude: 74.13255})- [:Home\_to\_restaurant]->({City: "Kolhapur", Lattitude: 16.7050,Longitude: 74.2433})<- [:College\_to\_restaurant]-({City: "Turambe",Latitude: 16.42515, Longitude: 74.13255})-[:College\_to\_Home]->(Home)-[:Home\_to\_office]->({City:"Pune",Lattitude:18.5204, Longitude: 73.8567}), path1 = ({City: "Satara", Latitude: 17.693167, Longitude: 74.004344})<-[:Home\_to\_busstation]-(Home)-[:Home\_to\_school]->({City:"Turambe", Latitude: 16.42515, Longitude: 74.13255}) RETURN path0, path1



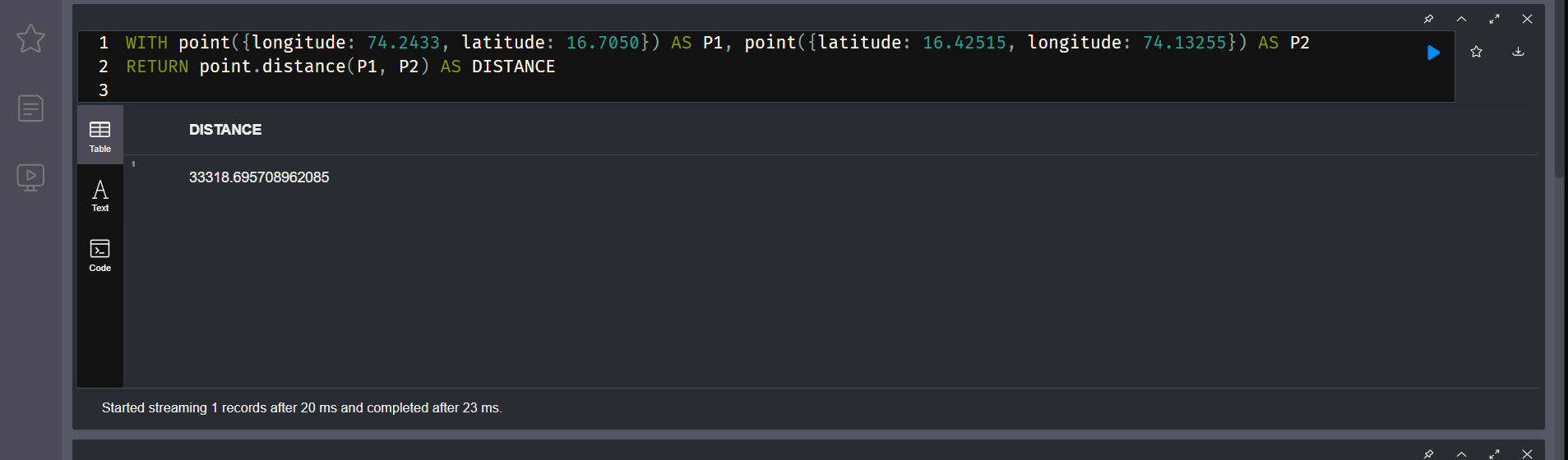
WITH point({longitude: 74.58151, latitude: 16.8524}) AS P1, point({latitude: 16.8524, longitude: 74.5815}) AS P2

RETURN point.distance(P1, P2) AS DISTANCE



WITH point({longitude: 74.2433, latitude: 16.7050}) AS P1, point({latitude: 16.42515, longitude: 74.13255}) AS P2

RETURN point.distance(P1, P2) AS DISTANCE



**Results:**

With distance() and point() functions we found out the distance between

restaurant in chale and bus station in Pandharpur. And found many other

distances from home to other places.

**Conclusion:**

With the Neo4J graph database and geospatial data, we find the

close/ nearest place with respect to reference points in an efficient way.

The queries are present in the attached text file