**Batch T4**

**Practical No. 12**

**Title of Assignment:** **Neo4j Spatial Database (Geographic)**

**Student Name: Abhishek Dudhpachare**

**Student PRN: 22510020**

**Objective**

The objective of this project is to:

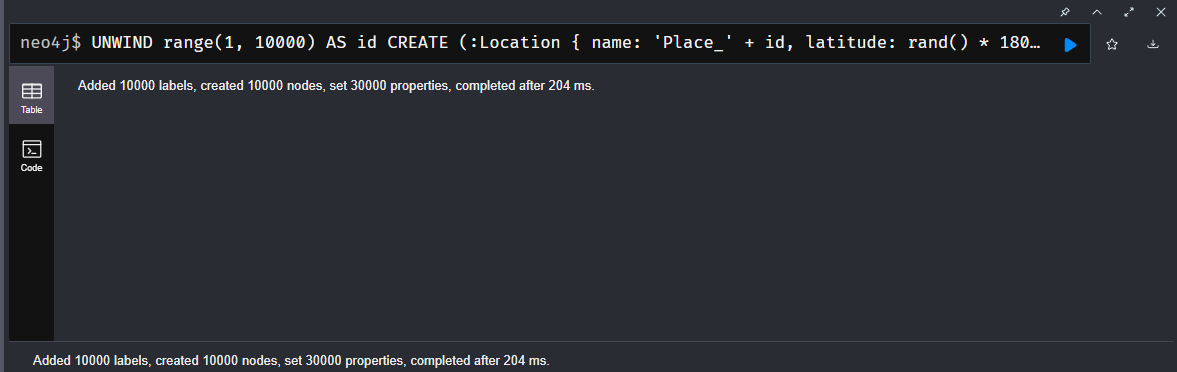
* Understand how spatial data can be modeled and queried using a graph database.
* Implement geospatial queries to answer “what is near what” using Neo4j.
* Simulate a real-world use case for location-based web services.
* Explore Neo4j’s native geospatial capabilities without relying on external plugins.

**1. Setting Up the Environment**

* Neo4j version 5.26.1 was installed and run through Neo4j Desktop.
* The database was created and accessed via the Neo4j Browser (http:localhost:7687).
* No additional plugins were installed. Neo4j 5 includes built-in support for point() and point.distance() functions.
* The query interface was used to run Cypher queries directly in the browser.

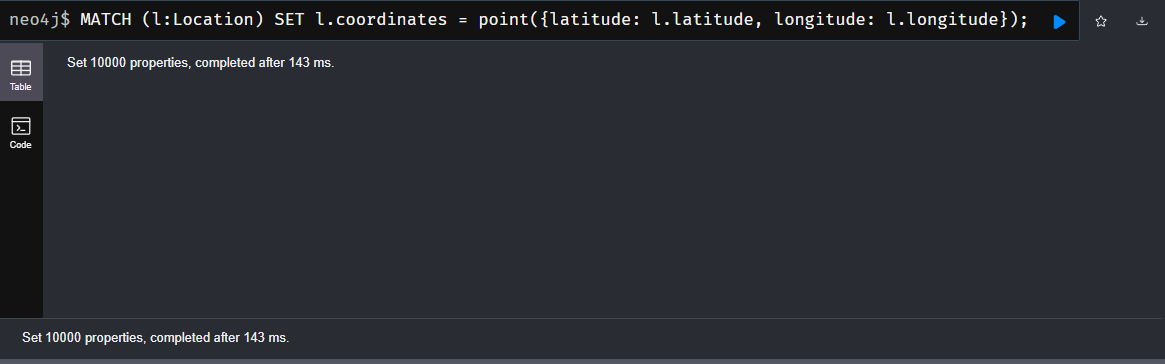
**2. Creating Sample Location Data**

* A simulation of 10,000 random locations was created near New Delhi, India.
* Each location node has:
  + A name property (e.g., Place\_1, Place\_2, ..., Place\_10000)
  + A latitude and longitude representing a real-world coordinate
  + A coordinates property, which is a point type used by Neo4j for geospatial queries
* The latitude and longitude values were slightly randomized around a center point (New Delhi) to simulate spread.



**3. Adding Spatial Indexes**

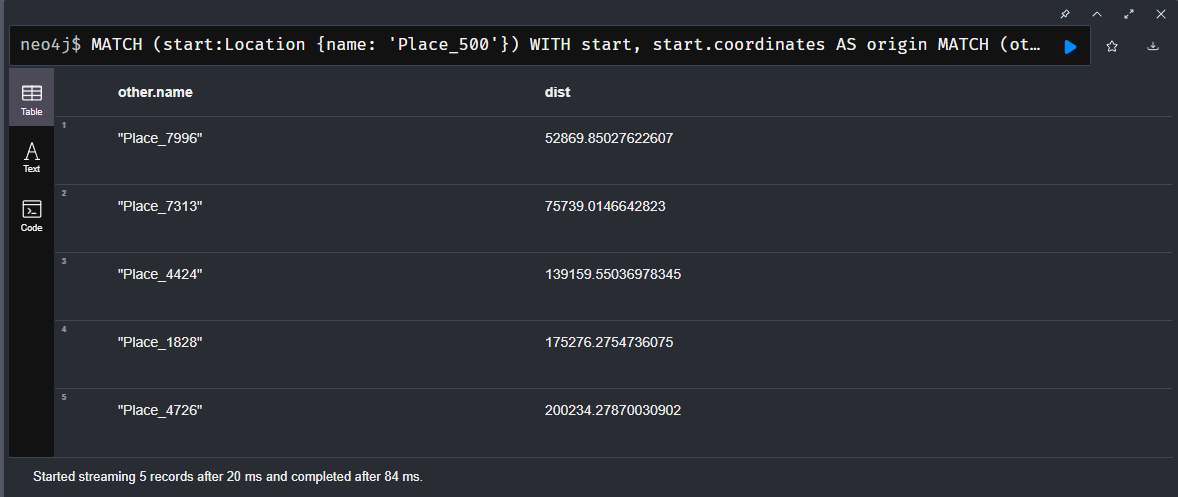
* To improve performance during geospatial queries, a spatial index was created on the coordinates property of all Location nodes.
* Indexing is essential when querying large datasets, as it reduces lookup time significantly.
* Without indexing, distance-based filtering on tens of thousands of nodes would be inefficient.



**4. Performing Geospatial Queries**

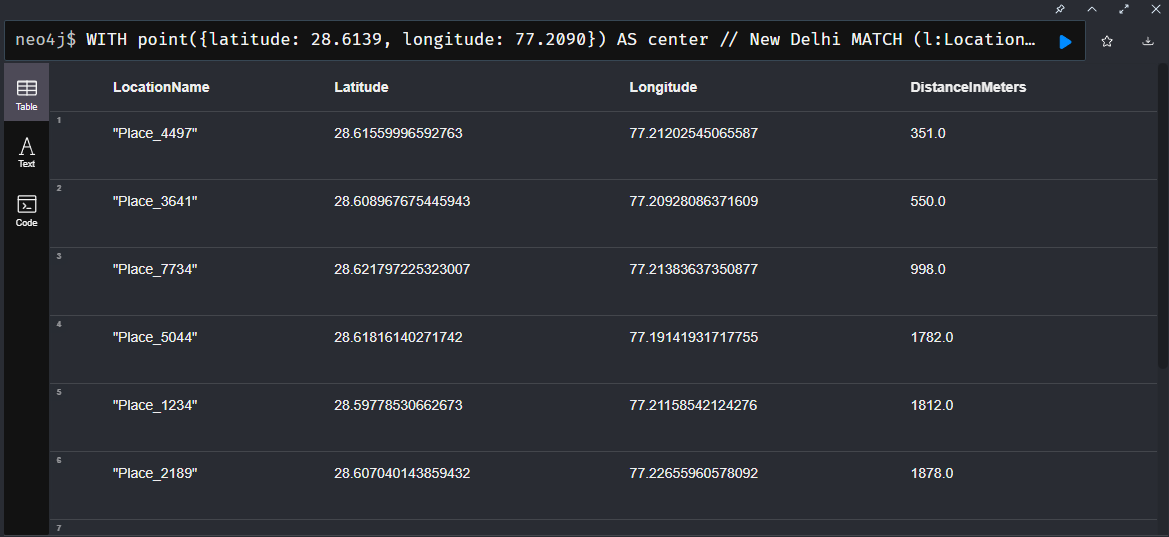
**Query: Finding Places Near a Specific Location**

* A fixed location (New Delhi) was used as the central point.
* A query was executed to find all locations within 100 km (100,000 meters).
* The results were sorted by the shortest distance to the reference point.
* The nearest 10 results were displayed, each including:
  + Location name
  + Latitude and longitude
  + Distance in meters



**5. Finding Nearby Locations to a Selected Node**

* In this use case, a specific node was selected (e.g., Place\_500).
* The spatial coordinates of the selected node were used as the origin.
* A query was executed to find all other locations sorted by their spatial distance from the selected node.
* The top 5 nearest neighbors were returned.
* This kind of query is commonly used in recommendation engines, navigation apps, and local search features.



**6. Advantages of Using Native Neo4j Geospatial Functions**

* No need for external libraries or plugins such as Neo4j Spatial.
* Built-in functions like point() and point.distance() are simple and efficient.
* Works seamlessly with Cypher and supports indexing for performance.
* Suitable for real-world use cases including:
  + Local business listings
  + Emergency response routing
  + Nearby event or venue search
  + Geo-based social media tagging

**7. Summary**

This project demonstrates how Neo4j 5 can be effectively used to:

* Model and store geospatial data using native point types.
* Query locations based on geographic distance without third-party tools.
* Simulate real-world problems like “finding things near other things.”
* Handle large spatial datasets efficiently with the help of point indexing.

