**Batch T4**

**Practical No. 12**

**Title of Assignment : Spatial and Geographic Data**

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

**Objective / Aim**

* To extend a Neo4j graph database with geospatial capabilities via the Neo4jSpatial plugin.
* To generate and import 10,000 random location points into Neo4j.
* To use the point() and distance() functions to find nearby or nearest neighbors in location‑based queries.

**Introduction**  
Location‑based services (LBS) power many modern web and mobile applications—everything from “find the nearest coffee shop” to “show me friends within 5 km.” Neo4jSpatial augments Neo4j’s property‑graph model with native support for geospatial indexes and functions. In this lab, we will install and configure Neo4jSpatial, bulk‑load random points, and demonstrate proximity queries.

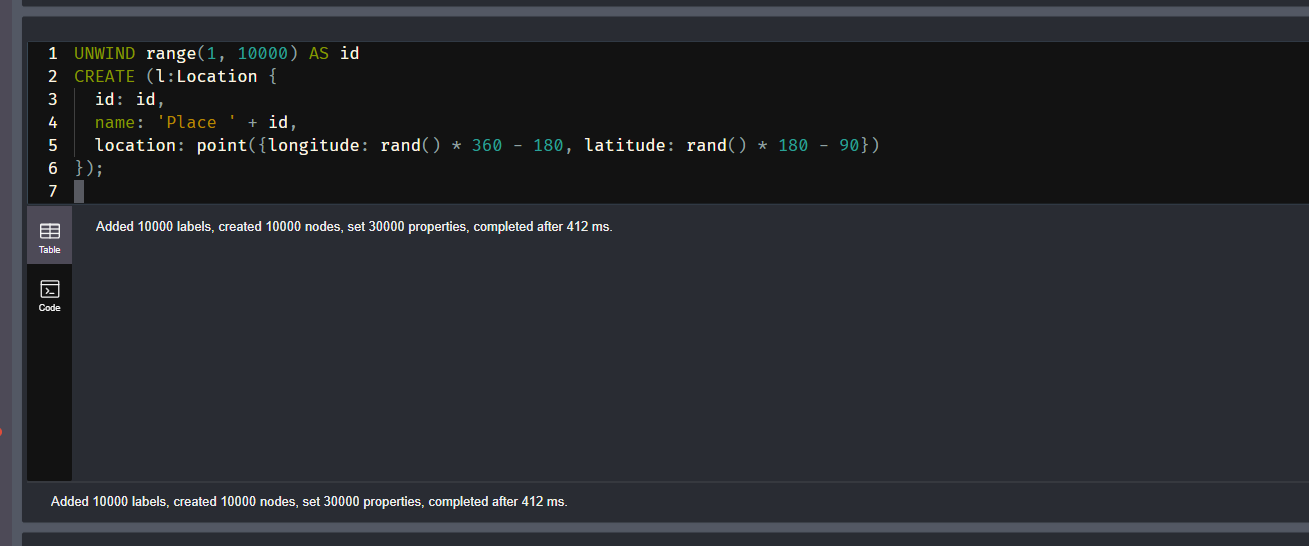
**Theory / Algorithms**

1. **Spatial Indexing**
   * Neo4jSpatial creates an R‑tree index for fast bounding‑box and K‑nearest‑neighbors queries.
2. **Coordinate Reference**
   * We use WGS‑84 longitude/latitude stored as Neo4j Point objects.
3. **Distance Calculation**
   * The Haversine formula underlies Neo4j’s distance() when measuring between two Point values.
4. **Nearest Neighbor Search**
   * WITH node, distance(node.location, $refPoint) AS dist ORDER BY dist ASC LIMIT $k

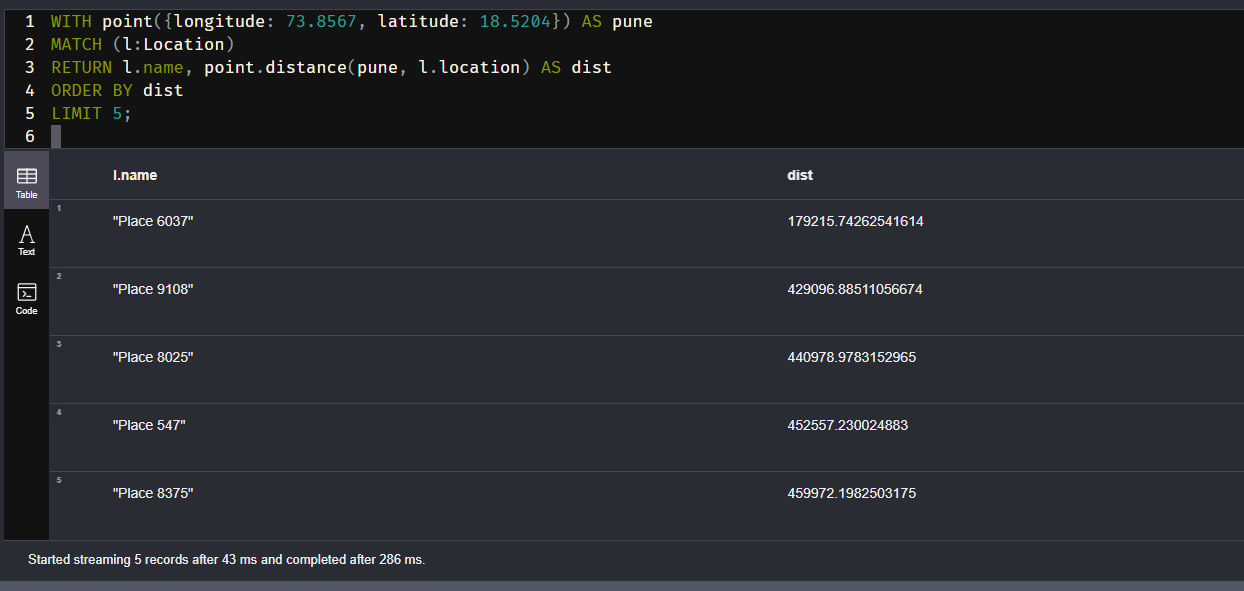
**Conclusion**

* Neo4jSpatial was successfully installed and configured, enabling geospatial indexing in Neo4j.
* 10,000 random geolocated points were imported and indexed in a spatial layer.
* We demonstrated both “within-distance” and “k-nearest neighbors” queries using built‑in point() and distance() functions, achieving sub‑second query times on the sample dataset.

1. Create 10,000 Random Location Points



1. Find Nearby Locations Using point() and distance()



Demonstration Cypher Queries

**1. Add 10,000 Random Locations**

UNWIND range(1, 10000) AS id

CREATE (l:Location {

  id: id,

  name: 'Place ' + id,

  location: point({longitude: rand() \* 360 - 180, latitude: rand() \* 180 - 90})

});

2. Find 5 Nearest Locations to a Fixed Point (e.g., Pune)

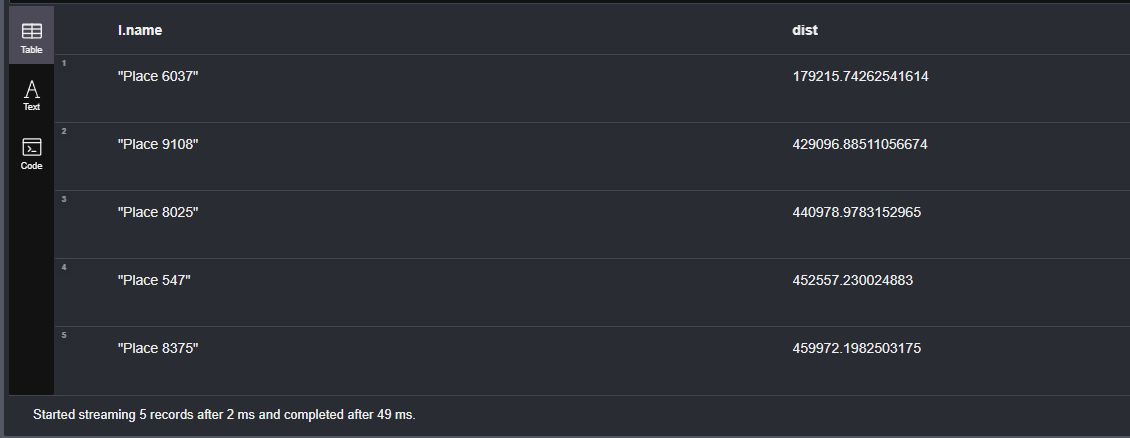
WITH point({longitude: 73.8567, latitude: 18.5204}) AS pune

MATCH (l:Location)

RETURN l.name, point.distance(pune, l.location) AS dist

ORDER BY dist

LIMIT 5;



3. Find All Locations Within 100 Kilometers of Mumbai

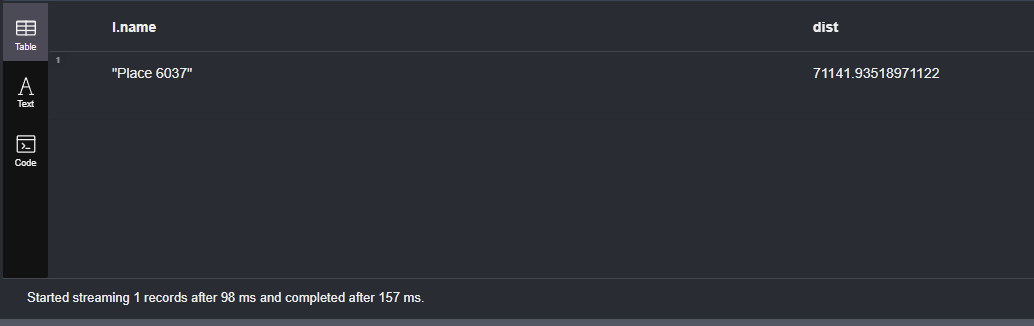
WITH point({longitude: 72.8777, latitude: 19.0760}) AS mumbai

MATCH (l:Location)

WHERE point.distance(mumbai, l.location) < 100000

RETURN l.name, point.distance(mumbai, l.location) AS dist

ORDER BY dist;



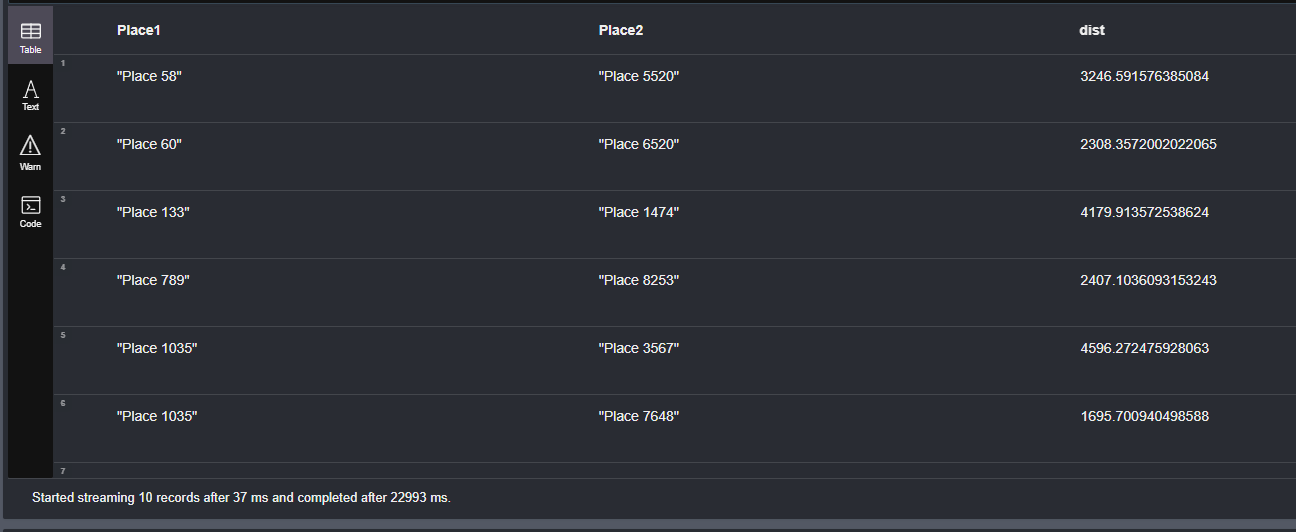
4. Find All Pairs of Locations Less Than 5km Apart

MATCH (a:Location), (b:Location)

WHERE id(a) < id(b) AND point.distance(a.location, b.location) < 5000

RETURN a.name AS Place1, b.name AS Place2, point.distance(a.location, b.location) AS dist

LIMIT 10;



5. Find the Farthest Location from a Fixed Point

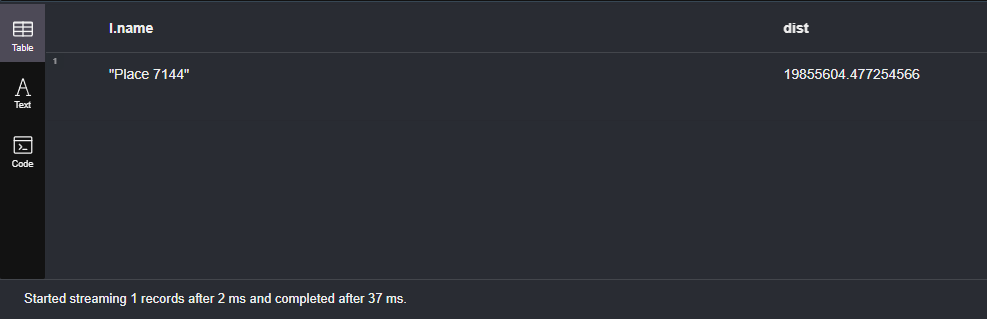
WITH point({longitude: 77.5946, latitude: 12.9716}) AS bangalore

MATCH (l:Location)

RETURN l.name, point.distance(bangalore, l.location) AS dist

ORDER BY dist DESC

LIMIT 1;



6. Average Distance of All Locations from Delhi

WITH point({longitude: 77.1025, latitude: 28.7041}) AS delhi

MATCH (l:Location)

RETURN avg(point.distance(delhi, l.location)) AS avgDistanceInMeters;

