SOMAIYA VIDYAVIHAR UNIVERSITY
K J Somaiya School of Engineering
(formerly K J Somaiya College of Engineering)

Batch: HDA2 **Roll No.: 16010123012**

Experiment No.: 05

TITLE: Create a graph database and perform basic analysis using Neo4j.

AIM: Create a property-graph database in Neo4j, populate it with nodes and relationships, and perform basic exploratory analysis using Cypher (counts, paths, and simple centrality-like measures), along with visual inspection in Neo4j Browser.

Expected OUTCOME of Experiment:
CO3: Perform the social data analytics

Books/ Journals/ Websites referred:
https://www.youtube.com/watch?v=8jNPelugC2s&ab_channel=LaitureAcademy

Pre Lab/ Prior Concepts:
Students should have a basic understanding of:

1. Graph theory basics: nodes/vertices, edges/relationships, degree, path
2. Property graph model: labels, relationship types, properties.
3. Cypher essentials: CREATE, MERGE, MATCH, WHERE, RETURN, ORDER BY, LIMIT, COUNT, WITH.
4. Constraints & indexes: why uniqueness matters; CREATE CONSTRAINT.
5. Neo4j tooling: Neo4j Desktop/Aura Free, Neo4j Browser result views (table vs graph).
6. CSV import with LOAD CSV

Instruction for Students:
Using Neo4j, design and create a small graph database of your choice (e.g., social network, movies, cities, or any domain). Insert at least 5–6 nodes and meaningful relationships, then perform basic analysis such as counting nodes/edges, finding friends-of-friends, degree of nodes, shortest path, and one additional query of your own design.

Procedure:

1) Setup

1. Install & open **Neo4j Desktop** (or use **Neo4j Aura Free**).
2. Create a new DBMS / project and start a database. Open **Neo4j Browser**.
3. In the Browser, confirm the connection (you should see a neo4j> prompt).

2) Create a clean schema

Run these in Neo4j Browser:

```
// Reset (ONLY if this is a fresh lab DB)
MATCH (n) DETACH DELETE n;
```

```
// Uniqueness constraint for Person name
CREATE CONSTRAINT person_name_unique IF NOT EXISTS
FOR (p:Person)
REQUIRE p.name IS UNIQUE;

// Optional index to speed up lookups
CREATE INDEX person_name_index IF NOT EXISTS
FOR (p:Person) ON (p.name);
```

3) Insert a mini social network

```
// Nodes
CREATE (:Person {name:'Aarav',  city:'Mumbai',  age:28});
CREATE (:Person {name:'Bhavna', city:'Pune',    age:31});
CREATE (:Person {name:'Chetan', city:'Mumbai',  age:26});
CREATE (:Person {name:'Deepti', city:'Delhi',   age:29});
CREATE (:Person {name:'Eshan',  city:'Bengaluru', age:33});

// Relationships (undirected concept, but stored as directed)
MATCH (a:Person {name:'Aarav'}), (b:Person {name:'Bhavna'}) CREATE (a)-[:KNOWS {since:2022}]->(b);
MATCH (a:Person {name:'Aarav'}), (c:Person {name:'Chetan'}) CREATE (a)-[:KNOWS {since:2023}]->(c);
MATCH (b:Person {name:'Bhavna'}), (c:Person {name:'Chetan'}) CREATE (b)-[:KNOWS {since:2021}]->(c);
MATCH (b:Person {name:'Bhavna'}), (d:Person {name:'Deepti'}) CREATE (b)-[:KNOWS {since:2020}]->(d);
MATCH (c:Person {name:'Chetan'}), (e:Person {name:'Eshan'}) CREATE (c)-[:KNOWS {since:2024}]->(e);
MATCH (d:Person {name:'Deepti'}), (e:Person {name:'Eshan'}) CREATE (d)-[:KNOWS {since:2022}]->(e);

// Optionally add reverse links to simulate undirected edges
MATCH (x)-[r:KNOWS]->(y) MERGE (y)-[:KNOWS {since:r.since}]->(x);
```

4) Quick sanity checks & exploration

```
// Counts
MATCH (n) RETURN COUNT(n) AS total_nodes;
MATCH ()-[r]->() RETURN COUNT(r) AS total_rels;

// What labels and relationship types exist?
CALL db.labels();
CALL db.relationshipTypes();
```

```
// Peek at the graph
MATCH (p:Person)-[r:KNOWS]->(q:Person)
RETURN p, r, q
LIMIT 50; // Switch to graph view in the Browser
```

5) Pattern queries (who knows whom?)

```
// People in Mumbai who know someone outside Mumbai
MATCH (p:Person {city:'Mumbai'})-[:KNOWS]->(q:Person)
WHERE p.city <> q.city
RETURN p.name AS from_mumbai, q.name AS knows_outside, q.city
ORDER BY from_mumbai;

// Friends-of-friends (length exactly 2) excluding direct friends/self
MATCH (p:Person {name:'Aarav'})-[:KNOWS]->()-[:KNOWS]->(fof:Person)
WHERE NOT (p)-[:KNOWS]->(fof) AND p <> fof
RETURN DISTINCT fof.name AS friend_of_friend;
```

6) Basic analysis

a) Degree (how connected is each person?)

```
// Total degree (since we stored both directions)
MATCH (p:Person)
RETURN p.name AS person, size( (p)--() ) AS degree
ORDER BY degree DESC;
```

b) Top “influencers” by degree (top 3)

```
MATCH (p:Person)
RETURN p.name AS person, size( (p)--() ) AS degree
ORDER BY degree DESC
LIMIT 3;
```

c) Mutual friends between two people

```
MATCH (a:Person {name:'Bhavna'})-[:KNOWS]->(m:Person)<-[:KNOWS]-(b:Person
{name:'Chetan'})
RETURN m.name AS mutual_friend;
```

d) Shortest path between two people

```
MATCH (src:Person {name:'Aarav'}), (dst:Person {name:'Eshan'})
CALL {
  WITH src, dst
  MATCH p = shortestPath( (src)-[:KNOWS*]-(dst) )
  RETURN p
}
RETURN p;
```

e) Triangles (3-cycles) through each node (simple clustering hint)

```
MATCH (a:Person)-[:KNOWS]->(b:Person),
```

(b)-[:KNOWS]->(c:Person),
(c)-[:KNOWS]->(a)
RETURN a.name AS person, COUNT(DISTINCT [a,b,c]) AS triangle_count
ORDER BY triangle_count DESC;
Optional (if Neo4j Graph Data Science (GDS) library is available in your setup): run degree/betweenness algorithms via GDS for more formal centralities.

7) Filtering, aggregation, and simple reporting

// Average age by city among people who know at least 2 others
MATCH (p:Person)
WITH p, size((p)--()) AS deg
WHERE deg >= 2
RETURN p.city AS city, ROUND(avg(p.age)) AS avg_age, COUNT(*) AS people
ORDER BY people DESC;

8) (Optional) Import from CSV (if you prepared files in Neo4j's import folder)

// Example shape, not executed unless files exist at file:///...
LOAD CSV WITH HEADERS FROM 'file:///people.csv' AS row
MERGE (p:Person {name: row.name})
SET p.city = row.city, p.age = toInteger(row.age);

LOAD CSV WITH HEADERS FROM 'file:///knows.csv' AS row
MATCH (a:Person {name: row.src}), (b:Person {name: row.dst})
MERGE (a)-[:KNOWS {since: toInteger(row.since)}]->(b)
MERGE (b)-[:KNOWS {since: toInteger(row.since)}]->(a);

9) Visualize

- In Neo4j Browser, re-run:
- MATCH (p:Person)-[r:KNOWS]->(q:Person) RETURN p,r,q;
- Switch to **Graph** view; drag nodes, examine properties, hover edges to see since.

10) Wrap-up / Clean-up

- Save your Cypher script (Browser "Saved scripts" or export).

sys stop the DB after taking screenshots for submission.

Implementation details:

```
neo4j$ CREATE (:Person {name:'Eshan', city:'Bengaluru', age:33});
```
Created 1 node, set 3 properties, added 1 label — Completed after 7 ms

```
neo4j$ CREATE (:Person {name:'Deepti', city:'Delhi', age:29});
```
Created 1 node, set 3 properties, added 1 label — Completed after 9 ms

```
neo4j$ CREATE (:Person {name:'Chetan', city:'Mumbai', age:26});
```
Created 1 node, set 3 properties, added 1 label — Completed after 7 ms

```
neo4j$ CREATE (:Person {name:'Bhavna', city:'Pune', age:31});
```
Created 1 node, set 3 properties, added 1 label — Completed after 7 ms

```
neo4j$ CREATE (:Person {name:'Aarav', city:'Mumbai', age:28});
```
Created 1 node, set 3 properties, added 1 label — Completed after 30 ms

```
neo4j$ CREATE INDEX person_name_index IF NOT EXISTS FOR (p:Person) ON (p.name);
```
No changes, no records — Completed after 30 ms

> 00NA0: Note: successful completion - index or constraint already exists

```
neo4j$ CREATE CONSTRAINT person_name_unique IF NOT EXISTS FOR (p:Person) REQUIRE p.name IS UNIQUE;
```
Added 1 constraint — Completed after 43 ms

```
neo4j$ MATCH (n) DETACH DELETE n;
```
No changes, no records — Completed after 60 ms

```
neo4j$ MATCH (a:Person {name:'Aarav'}), (b:Person {name:'Bhavna'}) CREATE (a)-[:KNOWS {since:2022}]->(b); MATCH (a:Person {
```

```
MATCH (a:Person {name:'Aarav'}),  (b:Person {name:'Bhavna'})  CREATE (a)-[:KNOWS {since:2022}]→(b);
MATCH (a:Person {name:'Aarav'}),  (c:Person {name:'Chetan'})  CREATE (a)-[:KNOWS {since:2023}]→(c);
MATCH (b:Person {name:'Bhavna'}), (c:Person {name:'Chetan'})  CREATE (b)-[:KNOWS {since:2021}]→(c);
MATCH (b:Person {name:'Bhavna'}), (d:Person {name:'Deepti'})  CREATE (b)-[:KNOWS {since:2020}]→(d);
MATCH (c:Person {name:'Chetan'}), (e:Person {name:'Eshan'})   CREATE (c)-[:KNOWS {since:2024}]→(e);
MATCH (d:Person {name:'Deepti'}), (e:Person {name:'Eshan'})   CREATE (d)-[:KNOWS {since:2022}]→(e);
MATCH (d:Person {name:'Aditey'}), (e:Person {name:'Sharma'})   CREATE (d)-[:KNOWS {since:2023}]→(e);
MATCH (b:Person {name:'Aditey'}), (d:Person {name:'Deepti'})  CREATE (b)-[:KNOWS {since:2025}]→(d);
MATCH (b:Person {name:'Sharma'}), (d:Person {name:'Chetna'})  CREATE (b)-[:KNOWS {since:2020}]→(d);
```

```
neo4j$ CREATE (:Person {name:'Sharma', city:'Kalyan', age:20});
```
Created 1 node, set 3 properties, added 1 label — Completed after 7 ms

```
neo4j$ CREATE (:Person {name:'Aditey', city:'Malad', age:54});
```
Created 1 node, set 3 properties, added 1 label — Completed after 9 ms

```
neo4j$ CALL db.labels(); CALL db.relationshipTypes();
```

```
CALL db.labels();
```

```
CALL db.relationshipTypes();
```

```
neo4j$ MATCH (n) RETURN COUNT(n) AS total_nodes; MATCH ()-[r]->() RETURN COUNT(r) AS total_rels;
```

```
MATCH (n) RETURN COUNT(n) AS total_nodes;
```

```
MATCH ()-[r]→() RETURN COUNT(r) AS total_rels;
```

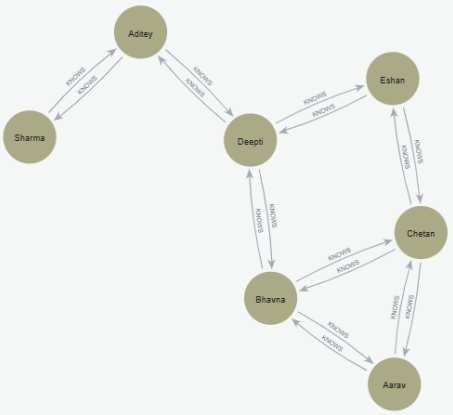
```
neo4j$ MATCH (x)-[r:KNOWS]->(y) MERGE (y)-[:KNOWS {since:r.since}]->(x);
```

Created 8 relationships, set 8 properties

Completed after 175 ms

```
neo4j$ MATCH (p:Person)-[r:KNOWS]->(q:Person) RETURN p, r, q LIMIT 50;
```

Graph | Table | RAW



Results overview

Nodes (7)

* (7) Person (7)

Relationships (16)

* (16) KNOWS (16)

Started streaming 16 records after 27 ms and completed after 31 ms.

```
neo4j$ MATCH (p:Person {name:'Aarav'})-[:KNOWS]->()-[:KNOWS]->(fof:Person) WHERE NOT (p)-[:KNOWS]->(fof) AND p <> fof RETUR
```

Table | RAW

	friend_of_friend
1	"Deepti"
2	"Eshan"

Started streaming 2 records after 97 ms and completed after 102 ms.

```
neo4j$ MATCH (p:Person {city:'Mumbai'})-[:KNOWS]->(q:Person) WHERE p.city <> q.city RETURN p.name AS from_mumbai, q.name AS
```

Table | RAW

	from_mumbai	knows_outside	q.city
1	"Aarav"	"Bhavna"	"Pune"
2	"Chetan"	"Bhavna"	"Pune"
3	"Chetan"	"Eshan"	"Bengaluru"

Started streaming 3 records after 69 ms and completed after 71 ms.

```
MATCH (p:Person)
RETURN p.name AS person, COUNT{ (p)--() } AS degree
ORDER BY degree DESC
LIMIT 3;
```

Table RAW

person	degree
"Bhavna"	6
"Chetan"	6
"Deepti"	6

Started streaming 3 records after 25 ms and completed after 29 ms.

```
neo4j$ MATCH (p:Person) RETURN p.name AS person, COUNT { (p)--() } AS degree ORDER BY degree DESC;
```

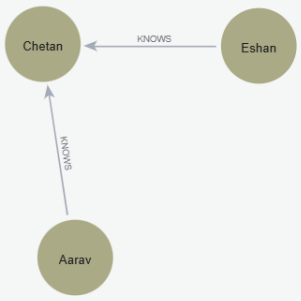
Table RAW

person	degree
"Bhavna"	6
"Chetan"	6
"Deepti"	6
"Aarav"	4
"Eshan"	4
"Aditey"	4
"Sharma"	2

Started streaming 7 records after 36 ms and completed after 38 ms.

```
neo4j$ MATCH (src:Person {name:'Aarav'}), (dst:Person {name:'Eshan'}) CALL { WITH src, dst MATCH p = shortestPath( (src)-[:
```

Graph Table RAW



Results overview

Nodes (3)

* (3) Person (3)

Relationships (2)

* (2) KNOWS (2)

Started streaming 1 record after 51 ms and completed after 56 ms.

01N00: Feature deprecated

2 info messages

```
neo4j$ MATCH (a:Person {name:'Bhavna'})-[:KNOWS]->(m:Person)<-[:KNOWS]-(b:Person {name:'Chetan'}) RETURN m.name AS mutual_f
```

Table RAW

mutual_friend
"Aarav"

Started streaming 1 record after 39 ms and completed after 40 ms.

```
MATCH (p:Person)
WITH p, COUNT{ (p)--() } AS deg
WHERE deg >= 2
RETURN p.city AS city, ROUND( avg(p.age) ) AS avg_age, COUNT(*) AS people
ORDER BY people DESC;
```

Table RAW

	city	avg_age	people
1	"Mumbai"	27.0	2
2	"Pune"	31.0	1
3	"Delhi"	29.0	1
4	"Bengaluru"	33.0	1
5	"Malad"	54.0	1
6	"Kalyan"	20.0	1

Started streaming 6 records after 10 ms and completed after 11 ms.

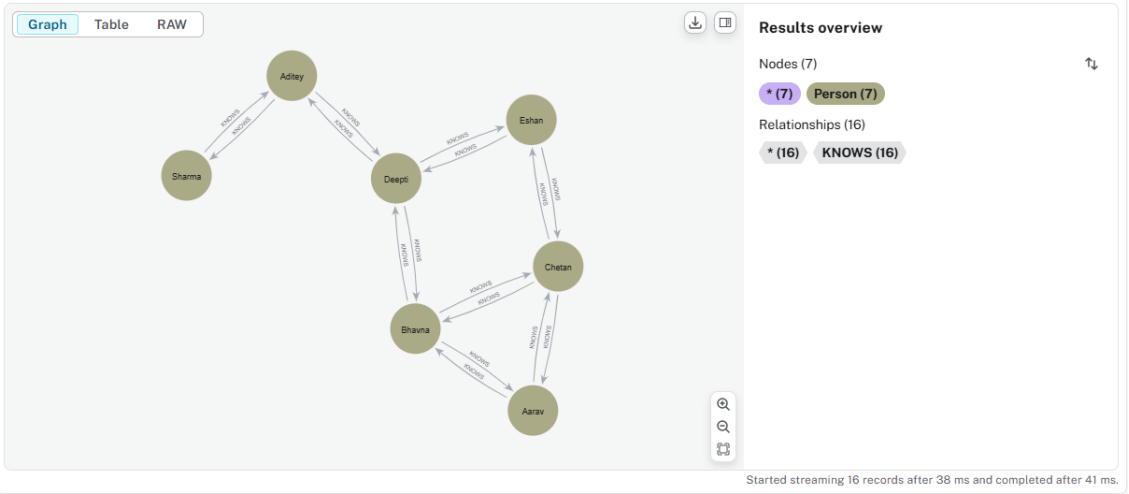
```
neo4j$ MATCH (a:Person)-[:KNOWS]->(b:Person), (b)-[:KNOWS]->(c:Person), (c)-[:KNOWS]->(a) RETURN a.name AS person, COUNT(DI
```

Table RAW

	person	triangle_count
1	"Chetan"	2
2	"Bhavna"	2
3	"Aarav"	2

Started streaming 3 records after 80 ms and completed after 95 ms.

```
neo4j$ MATCH (p:Person)-[r:KNOWS]->(q:Person) RETURN p,r,q;
```



Started streaming 16 records after 38 ms and completed after 41 ms.

Date: 22/09/25 **Signature of faculty in-charge**

Post Lab Descriptive Questions:

1. **How is data representation in a graph database different from a relational database? In what scenarios would a graph database be more efficient?**

- **Graph Database:** Uses nodes (entities), edges (relationships), and properties (data). Ideal for complex, connected data, e.g., social networks or recommendation engines.

- **Relational Database:** Uses tables with rows (records) and columns (attributes). Best for structured data with simple relationships, but struggles with complex, many-to-many connections.
- **Efficiency:** Graph databases excel when relationships are central to the queries, like finding connections between people or products. Relational databases may become inefficient due to complex joins.

2. **What does node degree signify? How did you compute it in your dataset?**

   Node Degree is the number of edges connected to a node. It measures how connected a node is in a graph.

   Computation: In Neo4j, you can compute node degree with degree(n) in a Cypher query:

   MATCH (n)

   RETURN n, degree(n)

3. **What insights can shortest path queries provide in real-world scenarios?**

   Shortest path queries help find the most efficient route between two nodes, useful in:

- Navigation systems (shortest route)
- Social networks (finding connections between people)
- Supply chains (optimizing product delivery)
- Recommendation engines (connecting users to products)

4. **What did the graph visualization in Neo4j Browser help you understand that a tabular result did not? Give at least one observation or insight you discovered in your dataset that would not have been obvious in a traditional database table.**

   Graph Visualization reveals relationships and connections more intuitively than tables:

- **Centrality:** Easily spot highly connected nodes (hubs).
- **Clusters:** Identify groups of closely connected nodes.
- **Pathways:** Visualize direct paths between nodes.