Experiment No.7

Aim: Perform CRUD Operations using a Graph-based Data Store

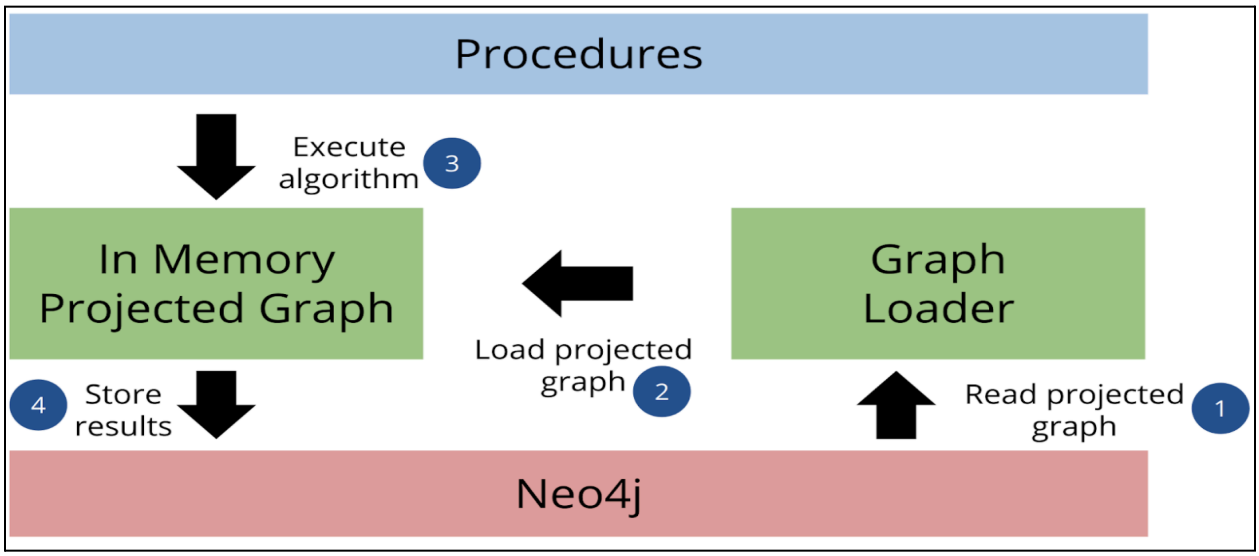
PostLab Questions:

List of Algorithms in Neo4j (Graph Data Science - GDS Library)

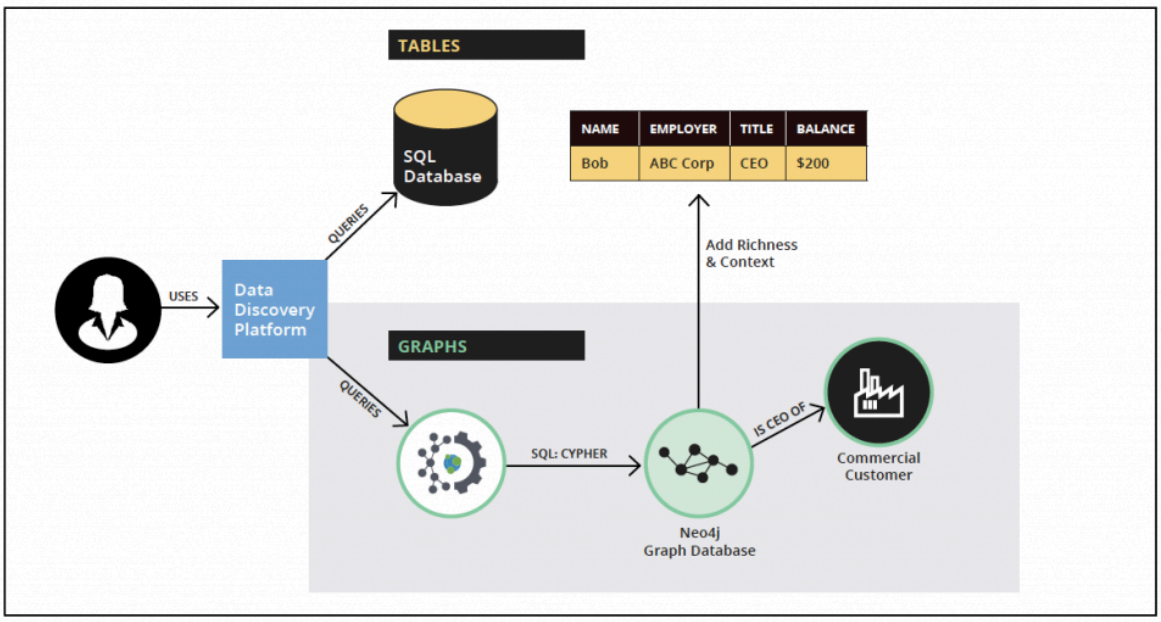
What is the importance of Neo4j?

Theory:

1. Reading the graph data from Neo4j Database
2. Loading (*projecting*) the data into an in-memory graph
3. Running an algorithm on a projected graph
4. Writing the results back to Neo4j Database (if the algorithm runs in write mode)



Experiment No.7



What is the importance of Neo4j?

Use **Neo4j** if your data is **highly connected**, requires **fast relationship queries**, and benefits from a **graph-based model**—ideal for social networks, recommendations, and fraud detection.

Neo4j vs. Traditional RDBMS (SQL Databases)

Feature	Neo4j (Graph DB)	Traditional RDBMS (SQL)
Data Model	Nodes & relationships (Graph)	Tables & rows (Relational)
Query Language	Cypher (Pattern-based)	SQL (Table-based)
Joins	No joins (relationships are native)	Expensive joins (foreign keys)
Performance	Faster for connected data	Slows down with complex joins
Scalability	Horizontally scalable	Mostly vertical scaling
Best For	Highly connected data (e.g., social networks, fraud detection, recommendation systems)	Structured, transactional data (e.g., banking, inventory, ERP systems)

Experiment No.7

What is Cypher Query in Neo4j?

Cypher is Neo4j's **graph query language**, designed to work with nodes, relationships, and properties in a **declarative** way, similar to SQL but optimized for graph data.

Key Features of Cypher:

✅ **Pattern-based** (works with nodes and relationships)
✅ **Readable & expressive** (less complex than SQL joins)
✅ **Optimized for traversal queries** (fast relationship lookups)

Step 1: Install neo4j https://neo4j.com/download/

Setup and Connect to Neo4j

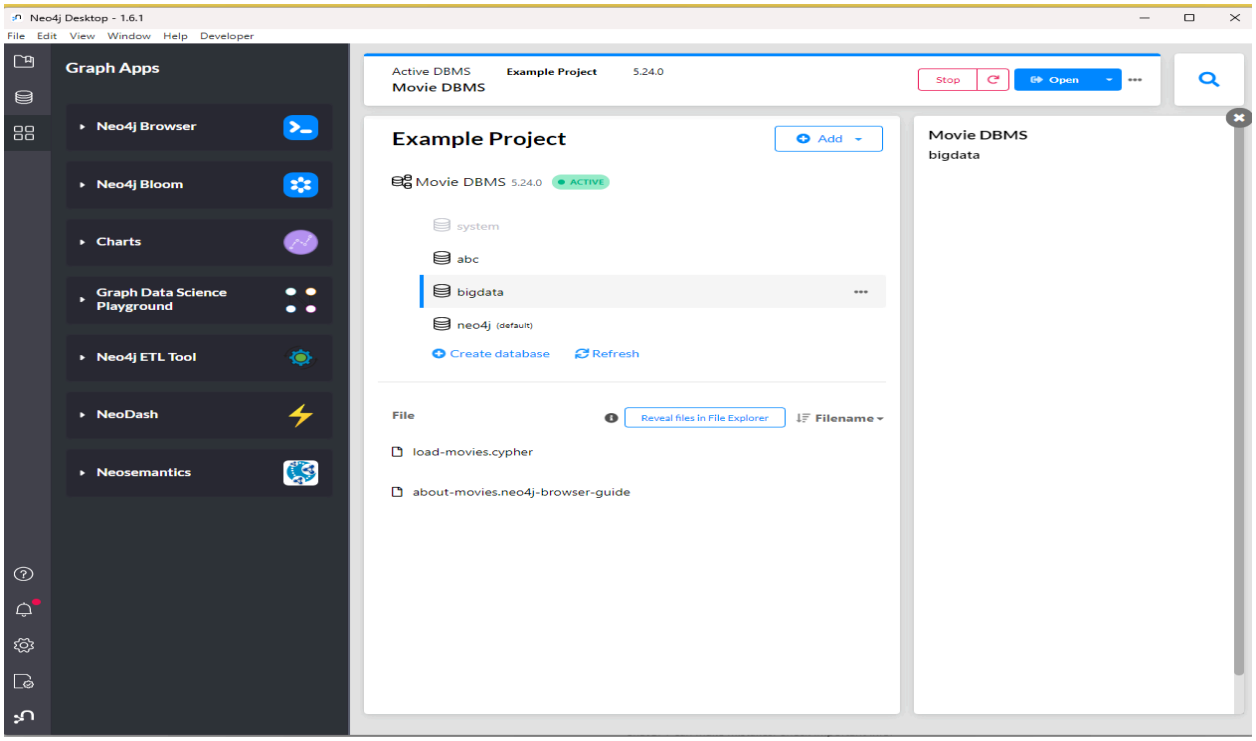
Before performing CRUD operations, ensure Neo4j is installed and running. You can access it via:

- **Neo4j Desktop**
- **Neo4j Browser** (http://localhost:7474)
- **Neo4j Bolt Driver (Python, Java, etc.)**
- **Cypher-shell** (neo4j console or cypher-shell)

Experiment No.7

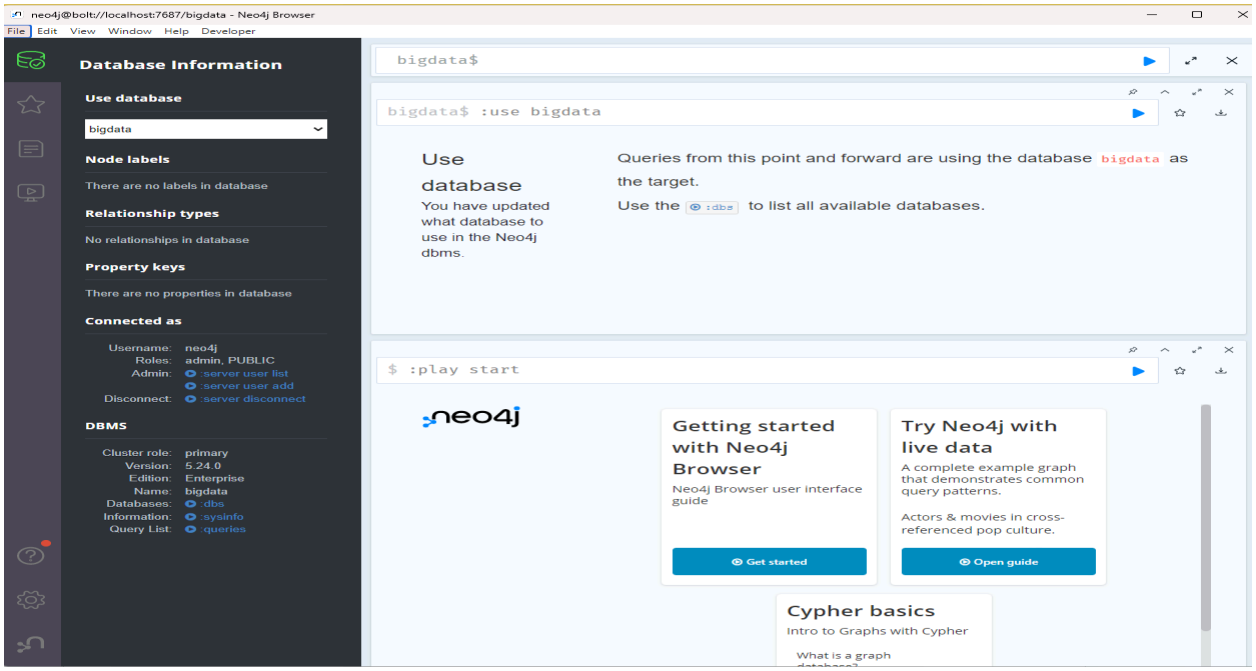
Step 2: Create account and copy key

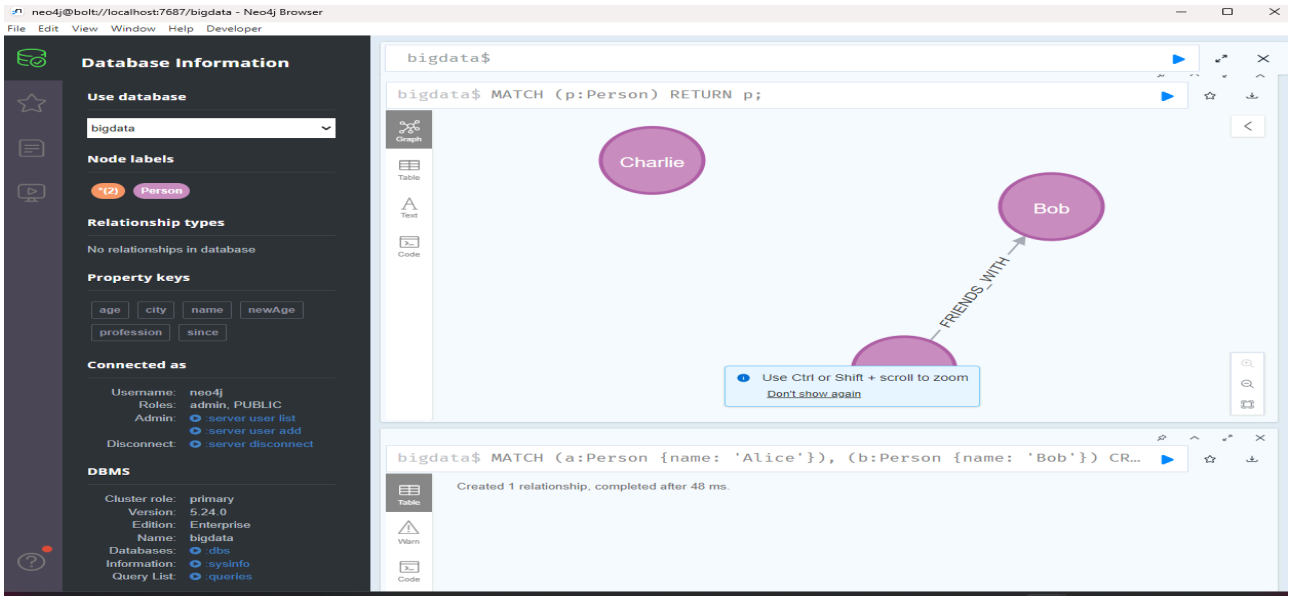
Step 3 : Create Database name bigdata



Step 4: Start Database and open ne04j browser

Experiment No.7





Practical

1. Create Operation

To create nodes and relationships, use the CREATE clause.

Experiment No.7

a) Creating a Node

#INSERT INTO Person (name, age) VALUES ('Alice', 30);

```
CREATE (p:Person {name: 'Alice', age: 30, city: 'New York'})
RETURN p;
```

- Creates a Person node with properties (name, age, city).

b) Creating Multiple Nodes

```
CREATE (p: stud {name: 'Alice', age: 30}),
      (p1:stud {name: 'Bob', age: 25}),
      (p2:stud {name: 'Charlie', age: 35})
return p,p1,p2;
```

c) Creating Relationships

```
MATCH (a:stud {name: 'Alice'}), (b:stud {name: 'Bob'})
CREATE (a)-[:FRIENDS_WITH]->(b);
```

- Creates a relationship FRIENDS_WITH between Alice and Bob.

2. Read Operation

To retrieve data, use the MATCH clause.

a) Retrieve All Nodes

```
MATCH (p:stud) RETURN p;
```

b) Retrieve Specific Node

```
MATCH (p:stud {name: 'Alice'})
RETURN p;
```

c) Retrieve Relationships

```
MATCH (a:stud)-[r:FRIENDS_WITH]->(b:stud)
RETURN a, r, b;
```

d) Retrieve Nodes with Conditions

```
MATCH (p:stud) WHERE p.age > 30 RETURN p;
```

3. Update Operation

To modify existing nodes and relationships, use SET.

a) Update a Node Property

```
MATCH (p:stud {name: 'Alice'})
SET p.age = 31
RETURN p;
```

b) Add a New Property

```
MATCH (p:stud {name: 'Alice'})
SET p.profession = 'Doctor'
RETURN p;
```

c) Rename a Property

```
MATCH (p:Stud {name: 'Alice'})
SET p.newAge = p.age
REMOVE p.age
RETURN p;
```

d) Update a Relationship

```
MATCH (a:stud {name: 'Alice'})-[r:FRIENDS_WITH]->(b:stud {name: 'Bob'})
```

```
SET r.since = 2023
RETURN r;
```

4. Delete Operation

To delete nodes and relationships, use DELETE or DETACH DELETE.

a) Delete a Relationship

```
MATCH (a:Stud {name: 'Alice'})-[r:FRIENDS_WITH]->(b:Stud {name: 'Bob'})
DELETE r;
```

b) Delete a Node (Fails if Relationships Exist)

```
MATCH (p:stud {name: 'Alice'})
DELETE p;
```

- This will fail if the node has relationships.

c) Delete a Node with All Relationships

```
MATCH (p:stud {name: 'Alice'})
DETACH DELETE p;
```

- This removes Alice and all associated relationships.

d) Delete All Nodes and Relationships (Use with Caution)

```
MATCH (n) DETACH DELETE n;
```

Conclusion: Neo4j provides an intuitive way to manage connected data using graph structures. The experiment demonstrated how to efficiently create, query, and manipulate graph data.

References:

Experiment No.7

https://github.com/harblaith7/Neo4j-Crash-Course/blob/main/01-initial-data.cypher