Neo4j is a world's leading open source Graph Database. It is completely developed by using Java Language by Neo Technology. Neo4j is an open source, Schema-free, No SQL, Graph Database. Graph Database (GDMS) is a database which stores data in the form of graph structures. It stores our application's data in terms of nodes, relationships and properties. Just like RDBMS stores data in the form of "rows, columns" of Tables, GDBMS stores data in the form of "graphs". A Graph is a set of nodes and the relationships that connect those nodes. Graph stores data in Nodes and Relationships in the form of Properties. Properties are key-value pairs to represent data.

Neo4j Advantages

- It is very easy to represent connected data.
- It is very easy and faster to retrieve/traversal/navigation of more connected data.
- It represents semi-structured data very easily.
- Neo4j CQL query language commands are in humane readable format and very easy to learn.
- It uses simple and powerful data model.
- It does NOT require complex Joins to retrieve connected/related data as it is very easy to retrieve it's adjacent node or relationship details without Joins or Indexes.

Basic Commands in Neo4j:

1) Create node

   CREATE (<node-name>:<label-name>)

   eg:

   ->CREATE (emp:Employee) //create node without properties

->CREATE (emp:Employee { emp_id:10,Name:"Shyam",location:"Kathmandu" })

//create node with properties

2) To get data from graph

->MATCH(emp:Employee) return emp;

emp_id	10
Name	Shyam
location	Kathmandu
emp_id	11
Name	Ram
location	Kathmandu

2) Relationship between nodes

CREATE (<node1-name>:<label1-name>)-[(<relationship-name>:<relationship-label-name>)]->(<node2-name>:<label2-name>)

eg:

->CREATE (p1:Profile1)-[r1:LIKES]->(p2:Profile2)

3) Where clause

WHERE <condition>

eg:

->MATCH (emp:Employee) WHERE emp.name = 'Shyam' RETURN

emp_id	10
Name	Shyam
location	Kathmandu

4) Delete node

```
DELETE <node-name-list>
```

eg:

```
->MATCH (e: Employee) DELETE e
```

Movie Graph example:

1) create node, set properties, set relationship

```
CREATE (matrix1:Movie { title : 'The Matrix', year : '1999-03-31' })

CREATE (matrix2:Movie { title : 'The Matrix Reloaded', year : '2003-05-07' })

CREATE (matrix3:Movie { title : 'The Matrix Revolutions', year : '2003-10-27' })

CREATE (keanu:Actor { name:'Keanu Reeves' })

CREATE (laurence:Actor { name:'Laurence Fishburne' })

CREATE (carrieanne:Actor { name:'Carrie-Anne Moss' })

CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix1)

CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix2)
CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix3)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix1)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix2)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix3)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix1)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix2)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix3)
```

2) Return a single node, by name:

```
->MATCH (movie:Movie { title: 'The Matrix' })  RETURN movie;

title    The Matrix

year     1999-03-31
```

3) Show all actors with order

```
->MATCH (actor:Actor) RETURN actor.name ORDER BY actor.name;

Carrie-Anne Moss

Keanu Reeves

Laurence Fishburne
```

4) Count the actors

```
->MATCH (actor:Actor) RETURN count(*);

count(*)

3
```

5) Get only the actors whose name end with "s"

```
->MATCH (actor:Actor) WHERE actor.name =~ ".*s$" RETURN actor.name;

actor.name

Keanu Reeves

Carrie-Anne Moss
```