NEO4J DOCUMENTATION

- **Neo4j** is one of the popular Graph Databases and **Cypher Query Language** (CQL).
- Neo4j is the world's leading open-source Graph Database which is developed using **Java** technology. It is highly scalable and **schema free (NoSQL).**

What is a Graph Database?

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. It is composed of two elements - **nodes (vertices)** and **relationships (edges).**

Why Graph Databases?

Nowadays, most of the data exists in the form of the relationship between different objects and more often, the relationship between the data is more valuable than the data itself.

Relational databases store highly structured data which have several records storing the same type of data so they can be used to store structured data and they do not store the relationships between the data.

Unlike other databases, graph databases store **relationships and connections** as first-class entities.

Advantages of Neo4j

- **Flexible data model** – Neo4j provides a flexible simple and yet powerful data model, which can be easily changed according to the applications and industries.
- **Real-time insights** – Neo4j provides results based on real-time data.
- **High availability** – Neo4j is highly available for large enterprise real-time applications with transactional guarantees.
- **Connected and semi structures data** – Using Neo4j, you can easily represent connected and semi-structured data.
- **Easy retrieval** – Using Neo4j, you can not only represent but also easily retrieve (traverse/navigate) connected data faster when compared to other databases.
- **No joins** – Using Neo4j, it does NOT require complex joins to retrieve connected/related data as it is very easy to retrieve its adjacent node or relationship details without joins or indexes.

Neo4j Property Graph Data Model

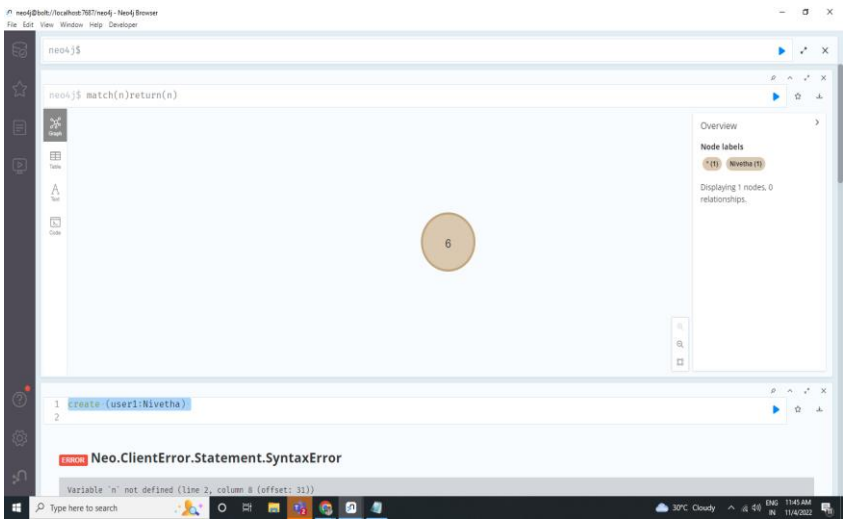
- The model represents data in **Nodes, Relationships** and **Properties**
- Properties are **key-value pairs**
- Nodes are represented using **circle** and Relationships are represented using **arrow keys**
- Relationships have directions: **Unidirectional** and **Bidirectional**
- Each Relationship contains "**Start Node**" or "**From Node**" and "**To Node**" or "**End Node**"
- Both Nodes and Relationships contain properties
- Relationships connects nodes

Simple Graph

1.1 Creating a single node

SYNTAX : create (Node: Label)

EXAMPLE: create (user1: Nivetha)



1.1 Creating a multiple node with properties

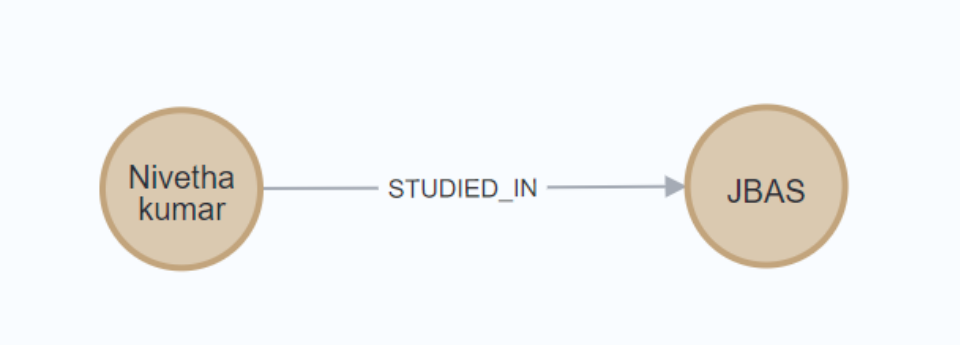
```
CREATE (user0: nive {Tag: "She Lives in Velacheri"}),
(user2: Divya {Tag: "She lives in padi"}),
(user3: kavya {Tag:"She lives in shollinganallur"}),
(user4: sharmila {Tag:"She lives in perungudi"})
RETURN user0, user2, user3, user4;
```

1.2 Creating Relationships

We can create a relationship using the **CREATE** clause. We will specify the relationship within the square braces "[ ]" depending on the direction of the relationship it is placed between hyphen " - " and arrow " →

```
CREATE (node1)-[:Relationship Type]->(node2)
```

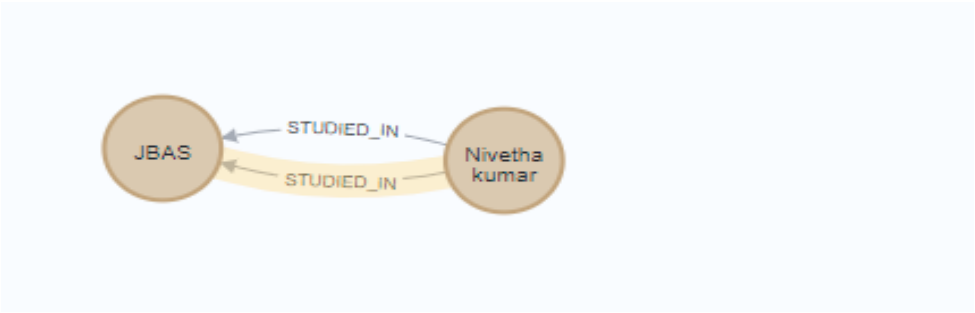
```
CREATE (Nivetha:student{name: "Nivetha kumar", YOB: 1998, AGE:23})
CREATE (Siet:College {name: "JBAS"})
CREATE (Nivetha)-[r:STUDIED_IN]->(Siet)
RETURN Nivetha, Siet
```



1.3 Creating a Relationship Between the Existing Nodes

```
MATCH (a: LabeofNode1), (b:LabeofNode2)
WHERE a.name = "nameofnode1" AND b.name = " nameofnode2"
CREATE (a)-[: Relation]->(b)
RETURN a,b
```

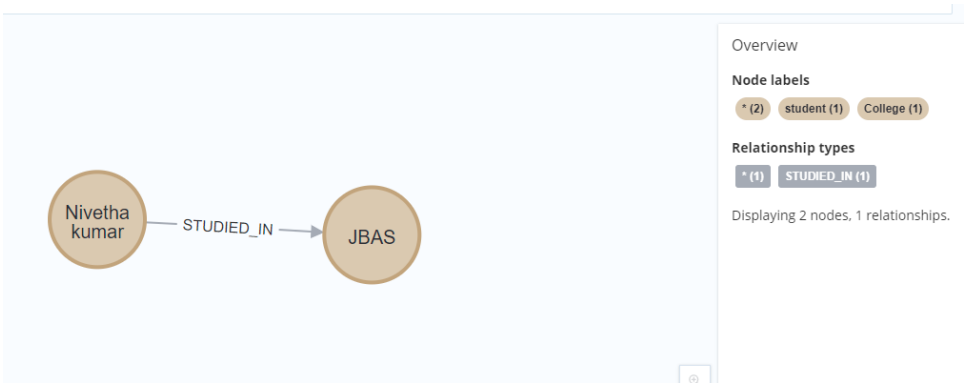
```
MATCH(a:Nivetha), (b:College) where a.name="Nivetha kumar" AND
b.name="JBAS"

CREATE (a)- [r: STUDIED_IN]->(b)

RETURN a,b
```



1.4 Creating a Relationship with Label and Properties

```
 CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]->
(node2)
```

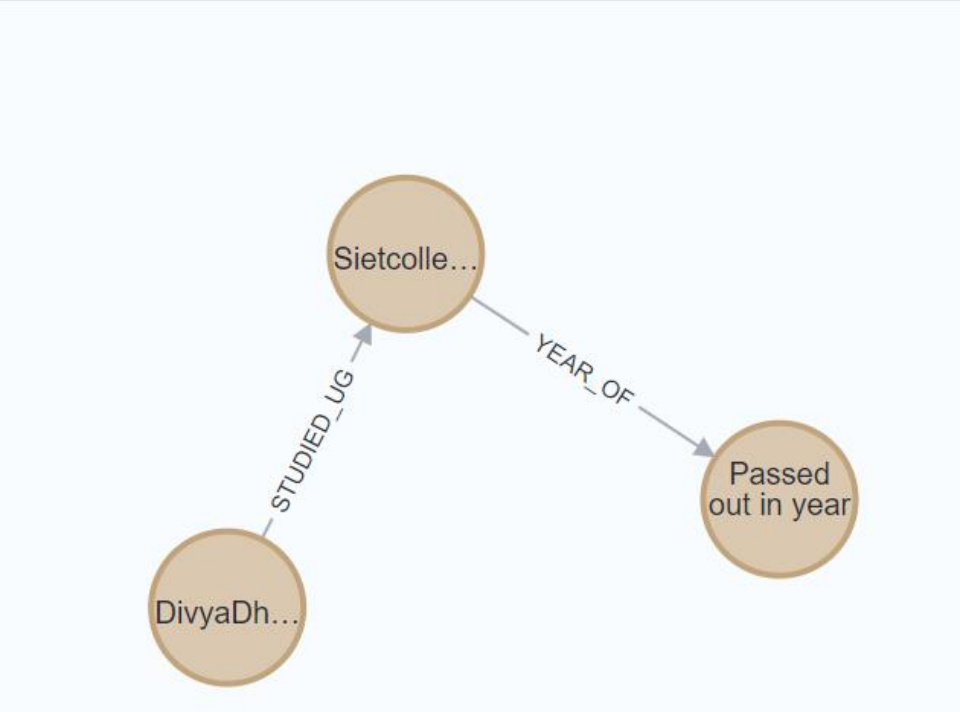
```
CREATE (Nivetha:student{name: "Nivetha kumar", YOB: 1998, AGE:23})
CREATE (Siet:College {name: "JBAS"})
CREATE (Nivetha)-[r:STUDIED_IN{year:2019, course:"Bca"}]->(Siet)
RETURN Nivetha, Siet
```



1.5 Creating a Relationship with more than one node

```
CREATE p = (Node1 {properties})-[:Relationship_Type]->
   (Node2 {properties})[:Relationship_Type]->(Node3 {properties})
RETURN p
```

```
CREATE p =(Divya:student{name:"DivyaDharshini"})-[: STUDIED_UG]-
>(Siet:college{name:"Sietcollege"})-[: YEAR_OF]->(Batch:batch{name:"Passed
out in year of 2019"})

RETURN p
```



2. Merge command

MERGE command is a combination of **CREATE** command and **MATCH** command.

Neo4j CQL MERGE command searches for a given pattern in the graph. **If it exists, then it returns the results.**

If it does NOT exist in the graph, then it creates a **new node/relationship and returns the results.**

```
MERGE (node: label {properties . . . . . })
```

2.1 Merging a Node with a Label

```
MERGE (node:label) RETURN node
```

```
CREATE (Nivetha:student{name: "Nivetha kumar", YOB: 1998, AGE:23})
CREATE (Siet:College {name: "JBAS"})
CREATE (Nivetha)-[r:STUDIED_IN{year:2019 , course:"Bca"}]->(Siet)
RETURN Nivetha, Siet
merge(reshma:student)RETURN reshma
```

"reshma"
{"YOJ":2019,"name":"balaji mohan","YOC":2022}
{"name":"Nivetha kumar","YOB":1998,"AGE":23}
{"name":"Nivetha kumar","YOB":1998,"AGE":23}
{"name":"Nivetha kumar","YOB":1998,"AGE":23}
{"name":"DivyaDharshini"}
{"name":"DivyaDharshini"}

2.2 Merging a Node with Properties

MERGE (node:label {key1:value, key2:value, key3:value . . . . . })

```
MERGE (reshma:student {name: "Reshma.r", YOB: 2000, AGE:22})
RETURN reshma
```

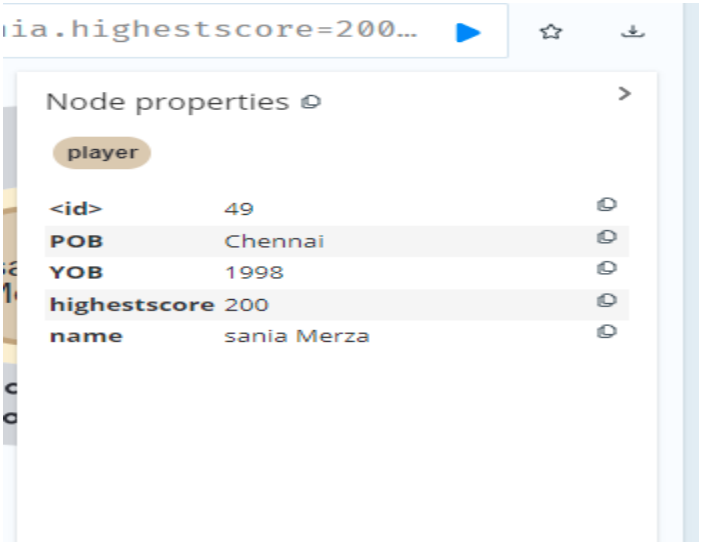
3. Set Command

Using the **Set clause,** you can add new properties to an existing Node or Relationship, and also add or update existing Properties values.

3.1 Setting a property

```
MATCH (node:label{properties . . . . . . . . . . . . . . . . })
SET node.property = value
RETURN node
```

```
CREATE (Sania:player{name: "sania Merza", YOB: 1998, POB: "Chennai"})
SET Sania.highestscore=200
return Sania
```



3.2 Removing a Property

You can remove an existing property by passing **NULL** as value to it.

```
MATCH (node:label {properties})
SET node.property = NULL
RETURN node
```