

Neo4j Queries Practice Related to Searching, Sorting and Pagination Concepts

Created a new Project in Neo4j Desktop and created new DBMS, then it started to get activated. After that open it as a Neo4j Browser to practice the following queries to create a Nodes, Properties, and its Relationships with Labels and to practice the Searching, Sorting data structure concepts.

Creation of Nodes for People

```
CREATE (Divya:People{name: "Divya",mobile:9789900998,age:23,city:"Chennai"})
CREATE (Nivetha: People {name: "Nivetha",mobile:8925298928,age:23,city:"Chennai"})
CREATE (Kavya: People {name: "Kavya",mobile:9789470466,age:24,city:"Salem"})
CREATE (Sharmi: People {name: "Sharmi",mobile:8925148656,age:24,city:"Coimbatore"})
CREATE (Balaji: People {name: "Balaji",mobile:8825539047,age:23,city:"Chennai"})
CREATE (Reshma: People {name: "Reshma",mobile:8765450934,age:24,city:"Thiruvallur"})
CREATE (Nazar: People {name: "Nazar",mobile:6380377708,age:23,city:"Chennai"})
```

Here, I created a few nodes as People. Whereas Nodes are Divya,Nivetha,Kavya,Sharmi,Balaji,Reshma and Nazar. Label is People, we can even create more than one label for a single node. A node's properties are specified with a key value pair preceded by colon within a curly brace. So, it specifies the details of the node. In this example, it specifies the details of the person like Name, Mobile Number, Age, City etc... We can even create properties for relationships as well.

Creation of Relationships for People

In this example, I tried the social media concept, where people follow one another. So, the Relationship in this example is FOLLOWS. We will specify the relationship within the square braces "[]" depending on the direction of the relationship it is placed between hyphen "-" and arrow "→" as shown in the following syntax.

Kavya Follows Divya

```
MATCH (Kavya: People {name: "Kavya",mobile:9789470466,age:24,city:"Salem"}),
(Divya:People {name: "Divya",mobile:9789900998,age:23,city:"Chennai"})
CREATE (Kavya)-[kd:FOLLOWS]->(Divya)
```

Kavya Follows Sharmi

MATCH (Kavya: People {name: "Kavya", mobile: 9789470466, age: 24, city: "Salem"}),

(Sharmi: People {name: "Sharmi", mobile: 8925148656, age: 24, city: "Coimbatore"})

CREATE (Kavya)-[ks:FOLLOWS]->(Sharmi)

Divya Follows Nivetha

MATCH (Divya: People {name: "Divya", mobile: 9789900998, age: 23, city: "Chennai"}),

(Nivetha: People {name: "Nivetha", mobile: 8925298928, age: 23, city: "Chennai"})

CREATE (Divya)-[dni:FOLLOWS]->(Nivetha)

Divya Follows Nazar

MATCH (Divya: People {name: "Divya", mobile: 9789900998, age: 23, city: "Chennai"}),

(Nazar: People {name: "Nazar", mobile: 6380377708, age: 23, city: "Chennai"})

CREATE (Divya)-[dna:FOLLOWS]->(Nazar)

Sharmi Follows Balaji

MATCH (Sharmi: People {name: "Sharmi", mobile: 8925148656, age: 24, city: "Coimbatore"}),

(Balaji: People {name: "Balaji", mobile: 8825539047, age: 23, city: "Chennai"})

CREATE (Sharmi)-[sb:FOLLOWS]->(Balaji)

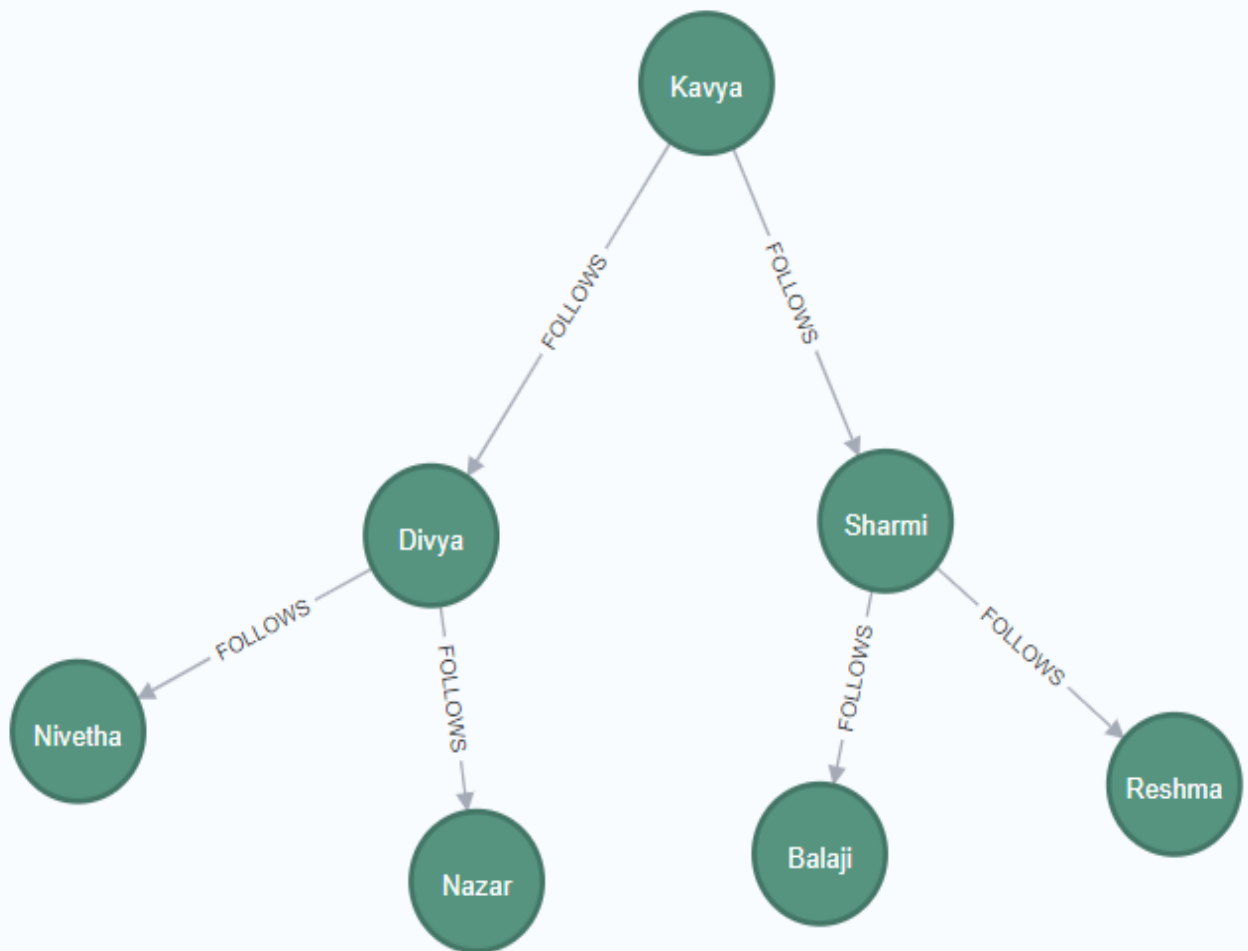
Sharmi Follows Reshma

MATCH (Sharmi: People {name: "Sharmi", mobile: 8925148656, age: 24, city: "Coimbatore"}),

(Reshma: People {name: "Reshma", mobile: 8765450934, age: 24, city: "Thiruvallur"})

CREATE (Sharmi)-[sr:FOLLOWS]->(Reshma)

GRAPH STRUCTURE



SEARCHING

Search is **an algorithm scheme that visits vertices or vertices and edges in a graph, in an order based on the connectivity of the graph**. The most general searches visit both edges and vertices.

In this example, I tried to fetch out the people who live in Chennai city by using where clause.

Neo4j CQL has provided WHERE clause in CQL MATCH command to filter the results of a MATCH Query.

WHERE is not a clause in its own right — rather, it is part of MATCH, OPTIONAL MATCH and WITH.

In the case of WITH, WHERE simply filters the results.

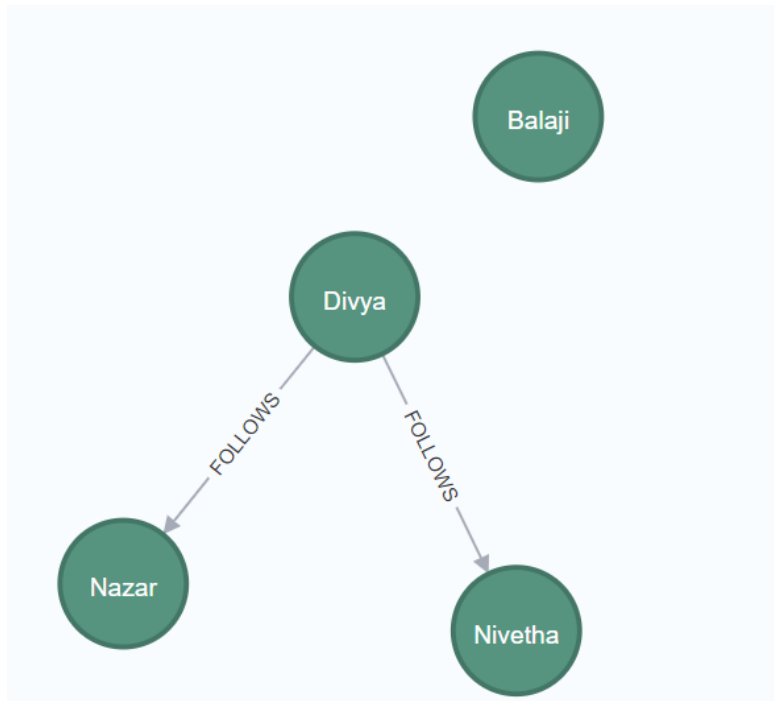
Syntax

Following is the syntax of the WHERE clause.

MATCH(People)

WHERE People.city = "Chennai"

RETURN People



In this example, I tried to fetch out the people who live in another cities which is out of Chennai by using the same where clause.

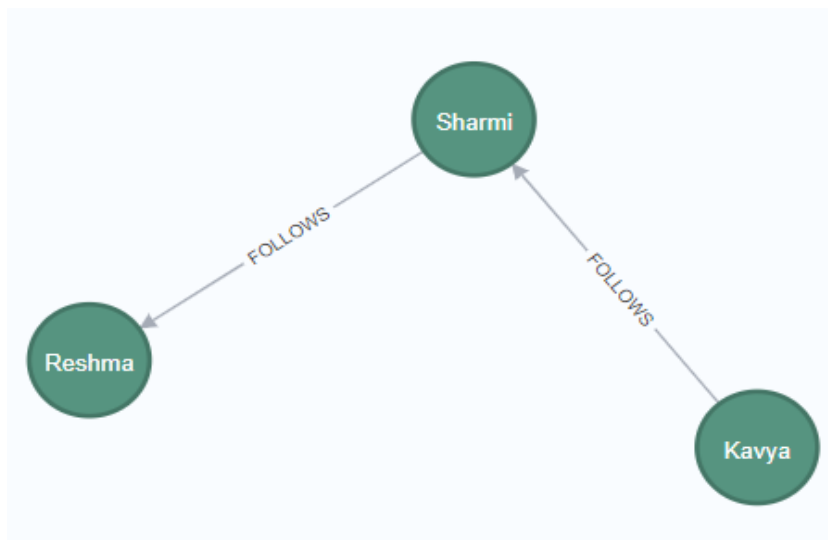
Syntax

Following is the syntax of the WHERE clause.

MATCH(People)

WHERE People.city <> "Chennai"

RETURN People



SORTING

A sorting input type is generated for every object type defined in your type definitions, allowing for Query results to be sorted by each individual field.

Neo4j CQL has provided "ORDER BY" Clause in MATCH Command to sort the results returned by a MATCH query.

We can sort rows either in ascending order or descending order.

By default, it sorts rows in ascending order. If we want to sort them in descending order, we need to use DESC clause

In this example, I tried to fetch out the people's name in Ascending Order by using the ORDER BY clause.

Syntax

MATCH(n)

RETURN n.name

ORDER BY n.name

```
1 MATCH (n)
2 RETURN n.name
3 ORDER BY n.name
```

| | n.name |
|---|-----------|
| 1 | "Balaji" |
| 2 | "Divya" |
| 3 | "Kavya" |
| 4 | "Nazar" |
| 5 | "Nivetha" |
| 6 | "Reshma" |
| 7 | "Sharmi" |

Started streaming 7 records after 1 ms and completed after 2 ms.

In this example, I tried to fetch out the people's name in Descending Order by using the ORDER BY clause.

Syntax

MATCH(n)

RETURN n.name

ORDER BY n.name **DESC**

```
1 MATCH (n)
2 RETURN n.name
3 ORDER BY n.name DESC
```

| | n.name |
|---|-----------|
| 1 | "Sharmi" |
| 2 | "Reshma" |
| 3 | "Nivetha" |
| 4 | "Nazar" |
| 5 | "Kavya" |
| 6 | "Divya" |
| 7 | "Balaji" |

RETRIEVING THE DETAILS OF A PARTICULAR NODE FROM A GRAPH STRUCTURE

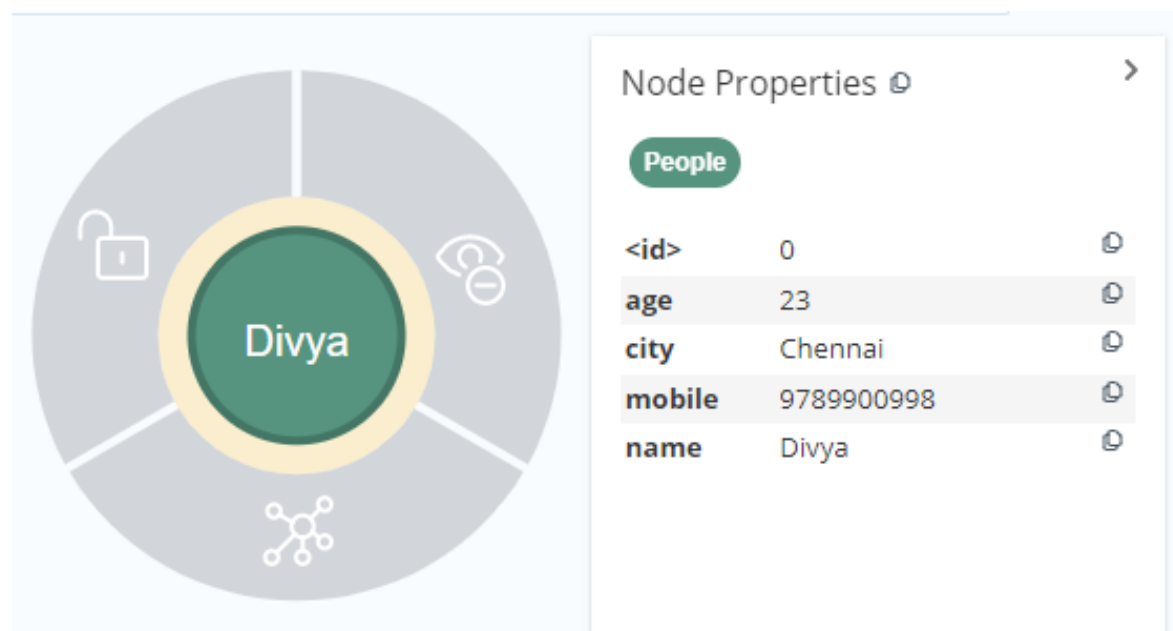
You can retrieve nodes based on relationship using the MATCH clause.

Following is the syntax of retrieving nodes based on the relationship using the MATCH clause.

MATCH(Divya:People{name: "Divya",mobile:9789900998,age:23,city:"Chennai"})

<-[kd:FOLLOWS]- (Kavya)

RETURN Divya



```
"Divya"
```

```
{"city":"Chennai","name":"Divya","mobile":9789900998,"age":23}
```

PAGINATION

SKIP and **LIMIT** are indeed how pagination is done for results in Neo4j. **SKIP** effectively controls where to start (which page number you're one) and **LIMIT** controls how many are on a page.

SKIP and **LIMIT** themselves are very fast,

The **SKIP** clause is used to define from which row to start including the rows in the output.

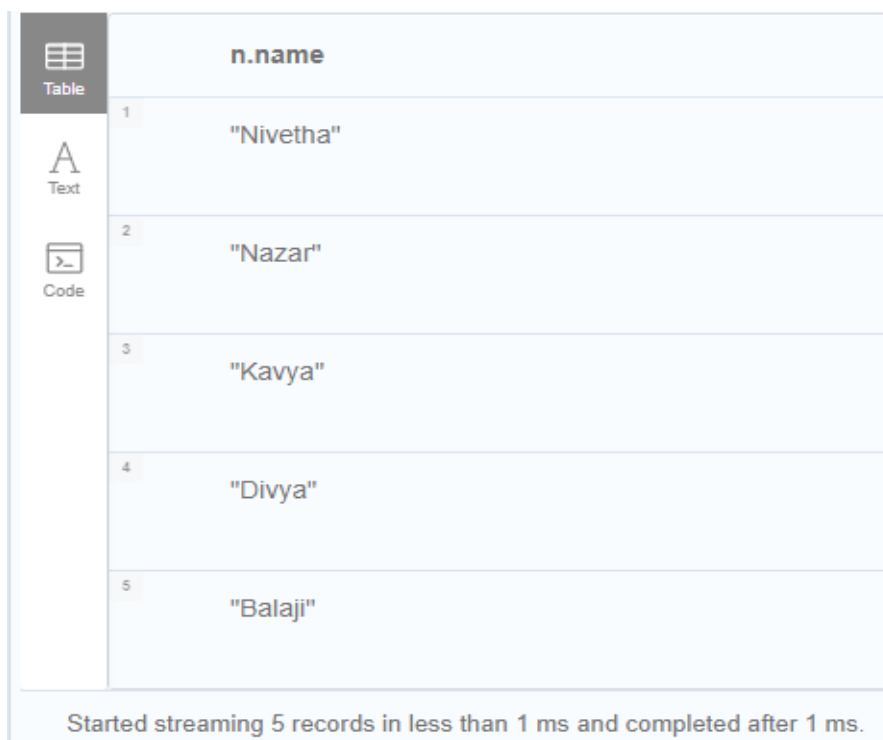
Following is a sample Cypher Query which returns all the nodes in the database skipping the 2 nodes which comes at last by the descending order.

MATCH (n)

RETURN n.name

ORDER BY n.name **DESC**

SKIP 2



The screenshot shows a Neo4j query result interface. On the left, there is a sidebar with three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'. The 'Table' icon is selected. The main area displays a table with the header 'n.name'. The table contains five rows of data, numbered 1 to 5 in the first column. The names are 'Nivetha', 'Nazar', 'Kavya', 'Divya', and 'Balaji' respectively. At the bottom of the table, a status bar reads: 'Started streaming 5 records in less than 1 ms and completed after 1 ms.'

| | n.name |
|---|-----------|
| 1 | "Nivetha" |
| 2 | "Nazar" |
| 3 | "Kavya" |
| 4 | "Divya" |
| 5 | "Balaji" |

Started streaming 5 records in less than 1 ms and completed after 1 ms.

The **Limit** clause is used to limit the number of rows in the output.

Syntax

Following is a sample Cypher Query which returns the nodes created above in a descending order and limits the records in the result to 3.

MATCH (n)

RETURN n.name

ORDER BY n.name **DESC**

LIMIT 3

| | n.name |
|---|-----------|
| 1 | "Sharmi" |
| 2 | "Reshma" |
| 3 | "Nivetha" |
| | |

Started streaming 3 records after 4 ms and completed after 12 ms.