Neo4j

graph database

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Neo4j is one of the popular Graph Databases and Cypher Query Language (CQL). Neo4j is written in Java Language. This tutorial explains the basics of Neo4j, Java with Neo4j, and Spring DATA with Neo4j.

The tutorial is divided into sections such as Neo4j Introduction, Neo4j CQL, Neo4j CQL Functions, Neo4j Admin, etc. Each of these sections contain related topics with simple and useful examples.

Audience

This tutorial has been prepared for beginners to help them understand the basic to advanced concepts of Neo4j. It will give you enough understanding on Neo4j from where you can take yourself to a higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have basic knowledge of Database, Graph Theory, Java, and Spring Framework.

Copyright & Disclaimer

© Copyright 2017 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About the Tutorial i
Audience i
Prerequisites i
Copyright & Disclaimer i
Table of Contents ii

1. Neo4j – Overview 1
What is a Graph Database? 1
Advantages of Neo4j 2
Features of Neo4j 2

2. Neo4j – Data Model 4

3. Neo4j – Environment Setup 6
Neo4j Database Server Setup with Windows exe File 6
Starting the Server 9
Working with Neo4j 11

4. Neo4j – Building Blocks 12
Node 12
Properties 12
Relationships 13
Labels 14
Neo4j Data Browser 14

NEO4J – CQL 17

5. Neo4j CQL – Introduction 18
Neo4j CQL Clauses 18
Neo4j CQL Functions 20
Neo4j CQL Data Types 21
CQL Operators 21
Boolean Operators in Neo4j CQL 22
Comparison Operators in Neo4j CQL 22

6. Neo4j CQL – Creating Nodes 24
Creating a Single node 24
Creating Multiple Nodes 27
Creating a Node with a Label 30
Creating a Node with Multiple Labels 33
Create Node with Properties 36
Returning the Created Node 39

7. Neo4j CQL – Creating a Relationship 42
Creating Relationships 42
Creating a Relationship Between the Existing Nodes 44
Creating a Relationship with Label and Properties 47
Creating a Complete Path 49

tutorialspoint
SIMPLYEASYLEARNING

NEO4J CQL – WRITE CLAUSES 51

8. Neo4j – Merge Command 52
Merging a Node with a Label 52
Merging a Node with Properties 56
OnCreate and OnMatch 58
Merge a Relationship 61

9. Neo4j – Set Clause 63
Setting a Property 63
Removing a Property 65
Setting Multiple Properties 67
Setting a Label on a Node 69
Setting Multiple Labels on a Node 71

10. Neo4j – Delete Clause 74
Deleting All Nodes and Relationships 74
Deleting a Particular Node 75

11. Neo4j – Remove Clause 77
Removing a Property 77
Removing a Label From a Node 79
Removing Multiple Labels 81

12. Neo4j – Foreach Clause 84

NEO4J CQL – READ CLAUSES 87

13. Neo4j – Match Clause 88
Get All Nodes Using Match 88
Getting All Nodes Under a Specific Label 90
Match by Relationship 92
Delete All Nodes 94

14. Neo4j – Optional Match Clause 96

15. Neo4j – Where Clause 98
WHERE Clause with Multiple Conditions 101
Using Relationship with Where Clause 103

16. Neo4j – Count Function 106
Count 106
Group Count 108

NEO4J CQL – GENERAL CLAUSES 111

17. Neo4j – Return Clause 112
Returning Nodes 112
Returning Multiple Nodes 114

tutorialspoint
SIMPLYEASYLEARNING

Returning Relationships 116
Returning Properties 118
Returning All Elements 120
Returning a Variable With a Column Alias 122

18. Neo4j – Order By Clause 124
Ordering Nodes by Multiple Properties 126
Ordering Nodes by Descending Order 128

19. Neo4j – Limit Clause 131
Limit with expression 133

20. Neo4j – Skip Clause 136
Skip Using Expression 138

21. Neo4j – With Clause 140

22. Neo4j – Unwind Clause 142

NEO4J CQL – FUNCTIONS 144

23. Neo4J CQL – String Functions 145
String Functions List 145
Upper 145
Lower 147
Substring 149

24. Neo4j – Aggregation Function 152
AGGREGATION Functions List 152
COUNT 152
MAX 155
MIN 157
AVG 159
SUM 161

NEO4J CQL – ADMIN 163

25. Neo4j – Backup & Restore 164
Neo4j Database Backup 164
Neo4j Database Restore 171

26. Neo4j – Index 174
Creating an Index 174
Deleting an Index 176

27. Neo4j – Create Unique Constraint 179
Create UNIQUE Constraint 179

28. Neo4j – Drop Unique 182

tutorialspoint
SIMPLYEASYLEARNING

1. Neo4j – Overview

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).

Graph database is a database used to model the data in the form of graph. In here, the nodes of a graph depict the entities while the relationships depict the association of these nodes.

Popular Graph Databases

Neo4j is a popular Graph Database. Other Graph Databases are Oracle NoSQL Database, OrientDB, HypherGraphDB, GraphBase, InfiniteGraph, and AllegroGraph.

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.

The data model for graph databases is simpler compared to other databases and, they can be used with OLTP systems. They provide features like transactional integrity and operational availability.

RDBMS Vs Graph Database

Following is the table which compares Relational databases and Graph databases.

Sr. No.	RDBMS	Graph Database
1	Tables	Graphs
2	Rows	Nodes

3	Columns and Data	Properties and its values
4	Constraints	Relationships
5	Joins	Traversal

Advantages of Neo4j

Following are the 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.
- **Cypher query language:** Neo4j provides a declarative query language to represent the graph visually, using an **ascii-art** syntax. The commands of this language are in human readable format and very easy to learn.
- **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.

Features of Neo4j

Following are the notable features of Neo4j -

- **Data model (flexible schema):** Neo4j follows a data model named *native property graph model*. Here, the graph contains nodes (entities) and these nodes are connected with each other (depicted by relationships). Nodes and relationships store data in key-value pairs known as properties.

  In Neo4j, there is no need to follow a fixed schema. You can add or remove properties as per requirement. It also provides schema constraints.
- **ACID properties:** Neo4j supports full ACID (Atomicity, Consistency, Isolation, and Durability) rules.
- **Scalability and reliability:** You can scale the database by increasing the number of reads/writes, and the volume without effecting the query processing speed and

data integrity. Neo4j also provides support for **replication** for data safety and reliability.

- **Cypher Query Language:** Neo4j provides a powerful declarative query language known as Cypher. It uses ASCII-art for depicting graphs. Cypher is easy to learn and can be used to create and retrieve relations between data without using the complex queries like Joins.

- **Built-in web application:** Neo4j provides a built-in **Neo4j Browser** web application. Using this, you can create and query your graph data.

- **Drivers:** Neo4j can work with –
    - REST API to work with programming languages such as Java, Spring, Scala etc.
    - Java Script to work with UI MVC frameworks such as Node JS.
    - It supports two kinds of Java API: Cypher API and Native Java API to develop Java applications.

  In addition to these, you can also work with other databases such as MongoDB, Cassandra, etc.

- **Indexing:** Neo4j supports Indexes by using Apache Lucence.

tutorialspoint
SIMPLYEASYLEARNING

2. Neo4j – Data Model

Neo4j Property Graph Data Model

Neo4j Graph Database follows the Property Graph Model to store and manage its data.

Following are the key features of Property Graph 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

In Property Graph Data Model, relationships should be directional. If we try to create relationships without direction, then it will throw an error message.

In Neo4j too, relationships should be directional. If we try to create relationships without direction, then Neo4j will throw an error message saying that "Relationships should be directional".

Neo4j Graph Database stores all of its data in Nodes and Relationships. We neither need any additional RRBMS Database nor any SQL database to store Neo4j database data. It stores its data in terms of Graphs in its native format.

Neo4j uses Native GPE (Graph Processing Engine) to work with its Native graph storage format.

The main building blocks of Graph DB Data Model are:

- Nodes
- Relationships
- Properties

tutorialspoint
SIMPLYEASYLEARNING

Following is a simple example of a Property Graph.

Simple Graph

Here, we have represented Nodes using Circles. Relationships are represented using Arrows. Relationships are directional. We can represent Node's data in terms of Properties (key-value pairs). In this example, we have represented each Node's Id property within the Node's Circle.

tutorialspoint
SIMPLYEASYLEARNING

3. Neo4j – Environment Setup

In this chapter, we will discuss how to install Neo4j in your system using exe file.

Neo4j Database Server Setup with Windows exe File

Follow the steps given below to download Neo4j into your system.

Step 1: Visit the Neo4j official site using https://neo4j.com/. On clicking, this link will take you to the homepage of neo4j website.

Step 2: As highlighted in the above screenshot, this page has a Download button on the top right hand side. Click it.

Step 3: This will redirect you to the downloads page, where you can download the community edition and the enterprise edition of Neo4j. Download the community edition of the software by clicking the respective button.

Step 4: This will take you to the page where you can download community version of Neo4j software compatible with different operating systems. Download the file respective to the desired operating system.

This will download a file named **neo4j-community_windows-x64_3_1_1.exe** to your system as shown in the following screenshot.

Step 5: Double-click the exe file to install Neo4j Server.

Step 6: Accept the license agreement and proceed with the installation. After completion of the process, you can observe that Neo4j is installed in your system.

Starting the Server

Step 1: Click the Windows startmenu and start the Neo4j server by clicking the start menu shortcut for Neo4j.

Step 2: On clicking the shortcut, you will get a window for Neo4j Community edition. By default, it selects c:\Users\[username]\Documents\Neo4j\default.graphdb. If you want, you can change your path to a different directory.

Step 3: Click the "Start" button to start the Neo4j server.

Once the server starts, you can observe that the database directory is populated as shown in the following screenshot.

Working with Neo4j

As discussed in the previous chapters, neo4j provides an in-built browse application to work with Neo4j. You can access Neo4j using the URL http://localhost:7474/

4. Neo4j – Building Blocks

Neo4j Graph Database has the following building blocks -

- Nodes
- Properties
- Relationships
- Labels
- Data Browser

Node

Node is a fundamental unit of a Graph. It contains properties with key-value pairs as shown in the following image.

Here, Node Name = "Employee" and it contains a set of properties as key-value pairs.

Properties

Property is a key-value pair to describe Graph Nodes and Relationships.

```
Key = Value
```

Where Key is a String and Value may be represented using any Neo4j Data types.

Relationships

Relationships are another major building block of a Graph Database. It connects two nodes as depicted in the following figure.

Here, Emp and Dept are two different nodes. "WORKS_FOR" is a relationship between Emp and Dept nodes.

As it denotes, the arrow mark from Emp to Dept, this relationship describes -

```
Emp WORKS_FOR Dept
```

Each relationship contains one start node and one end node.

Here, "Emp" is a start node, and "Dept" is an end node.

As this relationship arrow mark represents a relationship from "Emp" node to "Dept" node, this relationship is known as an "Incoming Relationship" to "Dept" Node and "Outgoing Relationship" to "Emp" node.

Like nodes, relationships also can contain properties as key-value pairs.

Here, "WORKS_FOR" relationship has one property as key-value pair.

```
Id=123
```

It represents an Id of this relationship.

Labels

Label associates a common name to a set of nodes or relationships. A node or relationship can contain one or more labels. We can create new labels to existing nodes or relationships. We can remove the existing labels from the existing nodes or relationships.

From the previous diagram, we can observe that there are two nodes.

Left side node has a Label: "Emp" and the right side node has a Label: "Dept".

Relationship between those two nodes also has a Label: "WORKS_FOR".

Note: Neo4j stores data in Properties of Nodes or Relationships.

Neo4j Data Browser

Once we install Neo4j, we can access Neo4j Data Browser using the following URL

```
http://localhost:7474/browser/
```

Neo4j Data Browser is used to execute CQL commands and view the output.

Here, we need to execute all CQL commands at dollar prompt: "**$**"

Type commands after the dollar symbol and click the "Execute" button to run your commands.

It interacts with Neo4j Database Server, retrieves and displays the results just below the dollar prompt.

Use "VI View" button to view the results in diagrams format. The above diagram shows results in "UI View" format.

Use "Grid View" button to view the results in Grid View. The following diagram shows the same results in "Grid View" format.

When we use "Grid View" to view our Query results, we can export them into a file in two different formats.

CSV

Click the "Export CSV" button to export the results in csv file format.

JSON

Click the "Export JSON" button to export the results in JSON file format.

However, if we use "UI View" to see our Query results, we can export them into a file in only one format: JSON

Neo4j – CQL

tutorialspoint
SIMPLYEASYLEARNING

5. Neo4j CQL – Introduction

CQL stands for Cypher Query Language. Like Oracle Database has query language SQL, Neo4j has CQL as query language.

Neo4j CQL -

- Is a query language for Neo4j Graph Database.
- Is a declarative pattern-matching language.
- Follows SQL like syntax.
- Syntax is very simple and in human readable format.

Like Oracle SQL -

- Neo4j CQL has commands to perform Database operations.
- Neo4j CQL supports many clauses such as WHERE, ORDER BY, etc., to write very complex queries in an easy manner.
- Neo4j CQL supports some functions such as String, Aggregation. In addition to them, it also supports some Relationship Functions.

Neo4j CQL Clauses

Following are the read clauses of Neo4j **C**ypher **Q**uery **L**anguage:

Sr. No.	Read Clauses	Usage
1	MATCH	This clause is used to search the data with a specified pattern.
2	OPTIONAL MATCH	This is the same as match, the only difference being it can use nulls in case of missing parts of the pattern.
3	WHERE	This clause id is used to add contents to the CQL queries.
4	START	This clause is used to find the starting points through the legacy indexes.
5	LOAD CSV	This clause is used to import data from CSV files.

tutorialspoint
SIMPLYEASYLEARNING

Following are the write clauses of Neo4j **C**ypher **Q**uery **L**anguage:

Sr. No.	Write Clauses	Usage
1	CREATE	This clause is used to create nodes, relationships, and properties.
2	MERGE	This clause verifies whether the specified pattern exists in the graph. If not, it creates the pattern.
3	SET	This clause is used to update labels on nodes, properties on nodes and relationships.
4	DELETE	This clause is used to delete nodes and relationships or paths etc. from the graph.
5	REMOVE	This clause is used to remove properties and elements from nodes and relationships.
6	FOREACH	This class is used to update the data within a list.
7	CREATE UNIQUE	Using the clauses CREATE and MATCH, you can get a unique pattern by matching the existing pattern and creating the missing one.
8	Importing CSV files with Cypher	Using Load CSV you can import data from .csv files.

Following are the general clauses of Neo4j **C**ypher **Q**uery **L**anguage:

Sr. No.	General Clauses	Usage
1	RETURN	This clause is used to define what to include in the query result set.
2	ORDER BY	This clause is used to arrange the output of a query in order. It is used along with the clauses **RETURN** or **WITH**.
3	LIMIT	This clause is used to limit the rows in the result to a specific value.

4	SKIP	This clause is used to define from which row to start including the rows in the output.
5	WITH	This clause is used to chain the query parts together.
6	UNWIND	This clause is used to expand a list into a sequence of rows.
7	UNION	This clause is used to combine the result of multiple queries.
8	CALL	This clause is used to invoke a procedure deployed in the database.

Neo4j CQL Functions

Following are the frequently used Neo4j CQL Functions:

Sr. No.	CQL Functions	Usage
1	String	They are used to work with String literals.
2	Aggregation	They are used to perform some aggregation operations on CQL Query results.
3	Relationship	They are used to get details of relationships such as startnode, endnode, etc.

We will discuss all Neo4j CQL commands, clauses and functions syntax, usage and examples in-detail in the subsequent chapters.

Neo4j CQL Data Types

These data types are similar to Java language. They are used to define properties of a node or a relationship.

Neo4j CQL supports the following data types:

Sr. No.	CQL Data Type	Usage
1	Boolean	It is used to represent Boolean literals: true, false.
2	byte	It is used to represent 8-bit integers.
3	short	It is used to represent 16-bit integers.
4	int	It is used to represent 32-bit integers.
5	long	It is used to represent 64-bit integers.
6	float	It is used to represent 32-bit floating-point numbers.
7	double	It is used to represent 64-bit floating-point numbers.
8	char	It is used to represent 16-bit characters.
9	String	It is used to represent Strings.

CQL Operators

Following are the list of operators supported by Neo4j Cypher Query language.

Sr. No.	Type	Operators
1	Mathematical	+, -, *, /, %, ^
2	Comparison	+, <>, <, >, <=, >=
3	Boolean	AND, OR, XOR, NOT
4	String	+

tutorialspoint
SIMPLYEASYLEARNING

5	List	+, IN, [X], [X.....Y]
6	Regular expression	=-
7	String matching	STARTS WITH, ENDS WITH, CONSTRAINTS

Boolean Operators in Neo4j CQL

Neo4j supports the following Boolean operators to use in Neo4j CQL WHERE clause to support multiple conditions.

Sr. No.	Boolean operators	Description
1	AND	It is a Neo4j CQL keyword to support AND operation. It is like SQL AND operator.
2	OR	It is a Neo4j CQL keyword to support OR operation. It is like SQL AND operator.
3	NOT	It is a Neo4j CQL keyword to support NOT operation. It is like SQL AND operator.
4	XOR	It is a Neo4j CQL keyword to support XOR operation. It is like SQL AND operator.

Comparison Operators in Neo4j CQL

Neo4j supports the following Comparison operators to use in Neo4j CQL WHERE clause to support conditions.

Sr. No.	Boolean operators	Description
1	=	It is a Neo4j CQL "Equal To" operator.
2	<>	It is a Neo4j CQL "Not Equal To" operator.

tutorialspoint
SIMPLYEASYLEARNING

3	<	It is a Neo4j CQL "Less Than" operator.
4	>	It is a Neo4j CQL "Greater Than" operator.
5	<=	It is a Neo4j CQL "Less Than Or Equal To" operator.
6	>=	It is a Neo4j CQL "Greater Than Or Equal To" operator.

6. Neo4j CQL – Creating Nodes

As discussed, a node is a data/record in a graph database. You can create a node in Neo4j using the **CREATE** clause. This chapter teaches you how to –

- Create a single node
- Create multiple nodes
- Create a node with a label
- Create a node with multiple labels
- Create a node with properties
- Returning the created node

Creating a Single node

You can create a node in Neo4j by simply specifying the name of the node that is to be created along with the CREATE clause.

Syntax

Following is the syntax for creating a node using Cypher Query Language.

```
CREATE (node_name);
```

Note: Semicolon (;) is optional.

Example

Following is a sample Cypher Query which creates a node in Neo4j.

```
CREATE (sample)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server as shown in the following screenshot.

tutorialspoint
SIMPLYEASYLEARNING

Step 2: Open your browser, copy paste the following URL in your address bar http://localhost:7474/. This will give you the built-in browser app of Neo4j with a dollar prompt as shown in the following screenshot.

Step 3: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node type, execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Creating Multiple Nodes

The create clause of Neo4j CQL is also used to create multiple nodes at the same time. To do so, you need to pass the names of the nodes to be created, separated by a comma.

Syntax

Following is the syntax to create multiple nodes using the CREATE clause.

```
CREATE (node1),(node2)
```

Example

Following is a sample Cypher Query which creates multiple nodes in Neo4j.

```
CREATE (sample1),(sample2)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node, type and execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Creating a Node with a Label

A label in Neo4j is used to group (classify) the nodes using labels. You can create a label for a node in Neo4j using the CREATE clause.

Syntax

Following is the syntax for creating a node with a label using Cypher Query Language.

```
CREATE (node:label)
```

Example

Following is a sample Cypher Query which creates a node with a label.

```
CREATE (Dhawan:player)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node, type and execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Creating a Node with Multiple Labels

You can also create multiple labels for a single node. You need to specify the labels for the node by separating them with a colon " **:** ".

Syntax

Following is the syntax to create a node with multiple labels.

```
CREATE (node:label1:label2:. . . . labeln)
```

Example

Following is a sample Cypher Query which creates a node with multiple labels in Neo4j.

```
CREATE (Dhawan:person:player)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node, type and execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Create Node with Properties

Properties are the key-value pairs using which a node stores data. You can create a node with properties using the CREATE clause. You need to specify these properties separated by commas within the flower braces "{ }".

Syntax

Following is the syntax to create a node with properties.

```
CREATE (node:label { key1: value, key2: value, . . . . . .  })
```

Example

Following is a sample Cypher Query which creates a node with properties.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node, type and execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Returning the Created Node

Throughout the chapter, we used the **MATCH (n) RETURN n** query to view the created nodes. This query returns all the existing nodes in the database.

Instead of this, we can use the RETURN clause with CREATE to view the newly created node.

Syntax

Following is the syntax to return a node in Neo4j.

```
CREATE (Node:Label{properties. . . . }) RETURN Node
```

Example

Following is a sample Cypher Query which creates a node with properties and returns it.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"}) RETURN
Dhawan
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

7. Neo4j CQL – Creating a Relationship

In Noe4j, a relationship is an element using which we connect two nodes of a graph. These relationships have direction, type, and the form patterns of data. This chapter teaches you how to –

- Create relationships
- Create a relationship between the existing nodes
- Create a relationship with label and properties

Creating Relationships

We can create a relationship using the CREATE clause. We will specify 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.

Syntax

Following is the syntax to create a relationship using the CREATE clause.

```
CREATE (node1)-[:RelationshipType]->(node2)
```

Example

First of all, create two nodes Ind and Dhawan in the database, as shown below.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
CREATE (Ind:Country {name: "India"})
```

Now, create a relationship named **BATSMAN_OF** between these two nodes as –

```
CREATE (Dhawan)-[r:BATSMAN_OF]->(Ind)
```

Finally, return both the nodes to see the created relationship.

```
RETURN Dhawan, Ind
```


tutorialspoint
SIMPLYEASYLEARNING

Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Creating a Relationship Between the Existing Nodes

You can also create a relationship between the existing nodes using the **MATCH** clause.

Syntax

Following is the syntax to create a relationship using the MATCH clause.

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

Example

Following is a sample Cypher Query which creates a relationship using the match clause.

```
MATCH (a:player), (b:Country)
WHERE a.name = "Shikar Dhawan" AND b.name = "India"
CREATE (a)-[r: BATSMAN_OF]->(b)
RETURN a,b
```

To execute the above query, carry out the following steps.

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Creating a Relationship with Label and Properties

You can create a relationship with label and properties using the CREATE clause.

Syntax

Following is the syntax to create a relationship with label and properties using the CREATE clause.

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

Example

Following is a sample Cypher Query which creates a relationship with label and properties.

```
MATCH (a:player), (b:Country)
WHERE a.name = "Shikar Dhawan" AND b.name = "India"
CREATE (a)-[r:BATSMAN_OF {Matches:5, Avg:90.75}]->(b)
RETURN a,b
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Creating a Complete Path

In Neo4j, a path is formed using continuous relationships. A path can be created using the create clause.

Syntax

Following is the syntax to create a path in Neo4j using the CREATE clause.

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

RETURN p
```

Example

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Neo4j CQL – Write Clauses

tutorialspoint
SIMPLYEASYLEARNING

8. Neo4j – 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.

In this chapter you are going to learn how to -

- Merge a node with label
- Merge a node with properties
- OnCreate and OnMatch
- Merge a relationship

Syntax

Following is the syntax for the MERGE command.

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

Before proceeding to the examples in this section, create two nodes in the database with labels Dhawan and Ind. Create a relationship of type "BATSMAN_OF" from Dhawan to Ind as shown below.

```
CREATE (Dhawan:player{name: "Shikar Dhawan", YOB: 1985, POB: "Delhi"})
CREATE (Ind:Country {name: "India"})
CREATE (Dhawan)-[r:BATSMAN_OF]->(Ind)
```

Merging a Node with a Label

You can merge a node in the database based on the label using the MERGE clause. If you try to merge a node based on the label, then Neo4j verifies whether there exists any node with the given label. If not, the current node will be created.

Syntax

Following is the syntax to merge a node based on a label.

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

Example 1

Following is a sample Cypher Query which merges a node into Neo4j (based on label). When you execute this query, Neo4j verifies whether there is any node with the label **player**. If not, it creates a node named "Jadeja" and returns it.

If, there exists any node with the given label, Neo4j returns them all.

```
MERGE (Jadeja:player) RETURN Jadeja
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Since you have already created a node named "Dhawan" with the label "player" in the database, Neo4j returns it as shown in the following screenshot.

Example 2

Now, try to merge a node named "CT2013" with a label named Tournament. Since there are no nodes with this label, Neo4j creates a node with the given name and returns it.

```
MERGE (CT2013:Tournament{name: "ICC Champions Trophy 2013"}) RETURN CT2013,
labels(CT2013)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and Start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. As discussed, since there is no node with the given label (Tournament). Neo4j creates and returns the specified node as shown in the following screenshot.

Merging a Node with Properties

You can also merge a node with a set of properties. If you do so, Neo4j searches for an equal match for the specified node, including the properties. If it doesn't find any, it creates one.

Syntax

Following is the syntax to merge a node using properties.

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

Example

Following is a sample Cypher Query to merge a node using properties. This query tries to merge the node named "jadeja" using properties and label. Since there is no such node with the exact label and properties, Neo4j creates one.

```
MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})
RETURN Jadeja
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and Start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. As discussed, since there are no nodes with the specified label and properties, it creates one, as shown in the following screenshot.

OnCreate and OnMatch

Whenever, we execute a merge query, a node is either matched or created. Using on create and on match, you can set properties for indicating whether the node is created or matched.

Syntax

Following is the syntax of **OnCreate** and **OnMatch** clauses.

```
MERGE (node:label {properties . . . . . . . . . . .})
ON CREATE SET property.isCreated ="true"
ON MATCH SET property.isFound ="true"
```

Example

Following is a sample Cypher Query which demonstrates the usage of **OnCreate** and **OnMatch** clauses in Neo4j. If the specified node already exists in the database, then the node will be matched and the property with key-value pair isFound ="true" will be created in the node.

If the specified node doesn't exist in the database, then the node will be created, and within it a property with a key-value pair isCreated ="true" will be created.

```
MERGE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})
ON CREATE SET Jadeja.isCreated ="true"
ON MATCH SET Jadeja.isFound ="true"
RETURN Jadeja
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. As discussed, since there is no node with the specified details, Neo4j created it along with the property **isFound** as shown in the following screenshot.

Merge a Relationship

Just like nodes, you can also merge the relationships using the MERGE clause.

Example

Following is a sample Cypher Query which merges a relationship using the MATCH clause in Neo4j. This query tries to merge a relationship named **WINNERS_OF** between the nodes "ind" (label: Country & name: India) and ICC13 (label: Tournament & name: ICC Champions Trophy 2013).

Since such relation doesn't exist, Neo4j creates one.

```
MATCH (a:Country), (b:Tournament)
WHERE a.name = "India" AND b.name = "ICC Champions Trophy 2013"
MERGE (a)-[r:WINNERS_OF]->(b)
RETURN a, b
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Since the specified relation doesn't exist in the database, Neo4j creates one as shown in the following screenshot.

In the same way, you can merge multiple relationships and undirected relationships too.

9. Neo4j – Set Clause

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

In this chapter, we are going to discuss how to –

- Set a property
- Remove a property
- Set multiple properties
- Set a label on a node
- Set multiple labels on a node

Setting a Property

Using the SET clause, you can create a new property in a node.

Syntax

Following is the syntax for setting a property.

```
MATCH (node:label{properties . . .
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that a property with a key-value pair highestscore/187 is created in the node named "Dhawan".

Removing a Property

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

Syntax

Following is the syntax of removing a property from a node using the SET clause.

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

Example

Before proceeding with the example, first create a node "jadeja" as shown below.

```
Create (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})
```

Following is a sample Cypher Query which removes the property named POB from this node using the SET clause as shown below.

```
MATCH (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})
SET Jadeja.POB = NULL
RETURN Jadeja
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that the variable named **POB** was deleted.

Setting Multiple Properties

In the same way, you can create multiple properties in a node using the Set clause. To do so, you need to specify these key value pairs with commas.

Syntax

Following is the syntax to create multiple properties in a node using the SET clause.

```
MATCH (node:label {properties})
SET node.property1 = value, node.property2 = value
RETURN node
```

Example

Following is a sample Cypher Query which creates multiple properties in a node using the SET clause in Neo4j.

```
MATCH (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988})
SET Jadeja.POB: "NavagamGhed", Jadeja.HS = "90"
RETURN Jadeja
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that properties named POB and HS were created.

Setting a Label on a Node

You can set a label to an existing node using the SET clause.

Syntax

Following is the syntax to set a label to an existing node.

```
MATCH (n {properties . . . . . . . })
SET n :label
RETURN n
```

Example

Before proceeding with the example, first create a node "Anderson" as shown below.

```
CREATE (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})
```

Following is a sample Cypher Query to set a label on a node using the SET clause. This query adds the label "player" to the node Anderson and returns it.

```
MATCH (Anderson {name: "James Anderson", YOB: 1982, POB: "Burnely"})
SET Anderson: player
RETURN Anderson
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that the label named "player" is added to the node.

Setting Multiple Labels on a Node

You can set multiple labels to an existing node using the SET clause. Here you need to specify the labels by separating them with colons "**:**".

Syntax

Following is the syntax to set multiple labels to an existing node using the SET clause.

```
MATCH (n {properties . . . . . . . })
SET n :label1:label2
RETURN n
```

Example

Before proceeding with the example, first create a node named "Ishant" as shown below.

```
CREATE (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query used to create multiple labels on a node using the SET clause.

```
MATCH (Ishant {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
SET Ishant: player:person
RETURN Ishant
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that two labels - person and player – are added to the node named **Ishant.**

10. Neo4j – Delete Clause

You can delete nodes and relationships from a database using the DELETE clause.

Deleting All Nodes and Relationships

Following is the query to delete all the nodes and the relationships in the database using the DELETE clause.

Query

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

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

This will delete all the nodes and relationships from your neo4j database and make it empty.

Deleting a Particular Node

To delete a particular node, you need to specify the details of the node in the place of "**n**" in the above query.

Syntax

Following is the syntax to delete a particular node from Neo4j using the DELETE clause.

```
MATCH (node:label {properties . . . . . .  })
DETACH DELETE node
```

Example

Before proceeding with the example, create a node "Ishant" in the Neo4j database as shown below.

```
CREATE (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query which deletes the above created node using the DELETE clause.

```
MATCH (Ishant:player {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
DETACH DELETE Ishant
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that the specified node is deleted.

11. Neo4j – Remove Clause

The REMOVE clause is used to remove properties and labels from graph elements (Nodes or Relationships).

The main difference between Neo4j CQL DELETE and REMOVE commands is -

- DELETE operation is used to delete nodes and associated relationships.
- REMOVE operation is used to remove labels and properties.

Removing a Property

You can remove a property of a node using MATCH along with the REMOVE clause.

Syntax

Following is the syntax to remove a property of a node using the REMOVE clause.

```
MATCH (node:label{properties . . . . . . . })
REMOVE node.property
RETURN node
```

Example

Before proceeding with the example, create a node named **Dhoni** as shown below.

```
CREATE (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
```

Following is a sample Cypher Query to remove the above created node using the REMOVE clause.

```
MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
REMOVE Dhoni.POB
RETURN Dhoni
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here, you can observe that the node named POB was deleted.

Removing a Label From a Node

Similar to property, you can also remove a label from an existing node using the remove clause.

Syntax

Following is the syntax to remove a label from a node.

```
MATCH (node:label {properties . . . . . . . . . . . })
REMOVE node:label
RETURN node
```

Example

Following is a sample Cypher Query to remove a label from an existing node using the remove clause.

```
MATCH (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
REMOVE Dhoni:player
RETURN Dhoni
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here, you can observe that the label was deleted from the node.

Removing Multiple Labels

You can also remove multiple labels from an existing node.

Syntax

Following is the syntax to remove multiple labels from a node.

```
MATCH (node:label1:label2 {properties . . . . . }})
REMOVE node:label1:label2
RETURN node
```

Example

Before proceeding with the example, create a node *Ishant* as shown below.

```
CREATE (Ishant:player:person {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query to remove multiple labels from a node.

```
MATCH (Ishant:player:person {name: "Ishant Sharma", YOB: 1988, POB: "Delhi"})
REMOVE Ishant:player:person
RETURN Ishant
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here, you can observe that the specified labels were deleted from the node.

12. Neo4j – Foreach Clause

The **FOREACH** clause is used to update data within a list whether components of a path, or result of aggregation.

Syntax

Following is the syntax of the FOREACH clause.

```
MATCH p = (start node)-[*]->(end node)
WHERE start.node = "node_name" AND end.node = "node_name"
FOREACH (n IN nodes(p)| SET n.marked = TRUE)
```

Example

Before proceeding with the example, create a path **p** in Neo4j database as shown below.

```
CREATE p = (Dhawan {name:"Shikar Dhawan"})-[:TOPSCORRER_OF]->(Ind{name:
"India"})-[:WINNER_OF]->(CT2013{name: "Champions Trophy 2013"})
RETURN p
```

Following is a sample Cypher Query which adds a property to all the nodes along the path using the FOREACH clause.

```
MATCH p = (Dhawan)-[*]->(CT2013)
WHERE Dhawan.name = "Shikar Dhawan" AND CT2013.name = "Champions Trophy 2013"
FOREACH (n IN nodes(p)| SET n.marked = TRUE)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

To verify the creation of the node, type and execute the following query in the dollar prompt.

```
MATCH (n) RETURN n
```

This query returns all the nodes in the database (we will discuss this query in detail in the coming chapters).

On executing, this query shows the created node as shown in the following screenshot.

Neo4j CQL – Read Clauses

tutorialspoint
SIMPLYEASYLEARNING

13. Neo4j – Match Clause

In this chapter, we will learn about Match Clause and all the functions that can be performed using this clause.

Get All Nodes Using Match

Using the MATCH clause of Neo4j you can retrieve all nodes in the Neo4j database.

Example

Before proceeding with the example, create 3 nodes and 2 relationships as shown below.

```
CREATE (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
CREATE (Ind)-[r1:WINNERS_OF {NRR:0.938 ,pts:6}]->(CT2013)
CREATE(Dhoni)-[r2:CAPTAIN_OF]->(Ind)

CREATE (Dhawan:player{name: "shikar Dhawan", YOB: 1995, POB: "Delhi"})
CREATE (Jadeja:player {name: "Ravindra Jadeja", YOB: 1988, POB: "NavagamGhed"})

CREATE (Dhawan)-[:TOP_SCORER_OF {Runs:363}]->(Ind)
CREATE (Jadeja)-[:HIGHEST_WICKET_TAKER_OF {Wickets:12}]->(Ind)
```

Following is the query which returns all the nodes in Neo4j database.

```
MATCH (n) RETURN n
```


tutorialspoint
SIMPLYEASYLEARNING

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Getting All Nodes Under a Specific Label

Using match clause, you can get all the nodes under a specific label.

Syntax

Following is the syntax to get all the nodes under a specific label.

```
MATCH (node:label)
RETURN node
```

Example

Following is a sample Cypher Query, which returns all the nodes in the database under the label **player**.

```
MATCH (n:player)
RETURN n
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Match by Relationship

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

Syntax

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

```
MATCH (node:label)<-[: Relationship]-(n)
RETURN n
```

Example

Following is a sample Cypher Query to retrieve nodes based on relationship using the MATCH clause.

```
MATCH (Ind:Country {name: "India", result: "Winners"})<-[: TOP_SCORER_OF]-(n)
RETURN n.name
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Delete All Nodes

You can delete all the nodes using the MATCH clause.

Query

Following is the query to delete all the nodes in Neo4j.

```
MATCH (n) detach delete n
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

14. Neo4j – Optional Match Clause

The **OPTIONAL MATCH** clause is used to search for the pattern described in it, while using nulls for missing parts of the pattern.

OPTIONAL MATCH is similar to the match clause, the only difference being it returns null as a result of the missing parts of the pattern.

Syntax

Following is the syntax of the OPTIONAL MATCH with relationship.

```
MATCH (node:label {properties. . . . . .})
OPTIONAL MATCH (node)-->(x)
RETURN x
```

Example

Following is a sample Cypher Query which tries to retrieve the relations from the node ICCT2013. Since there are no such nodes, it returns null.

```
MATCH (a:Tornament {name: "ICC Champions Trophy 2013"})
OPTIONAL MATCH (a)-->(x)
RETURN x
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that since there are no matches for the required pattern, Neo4j returned null.

15. Neo4j – Where Clause

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

Syntax

Following is the syntax of the WHERE clause.

```
MATCH (label)
WHERE label.country = "property"
RETURN label
```

Example

Before proceeding with the example, create five nodes in the database as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363, country:
"India"}
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"}
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177, country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, country:"India"})
CREATE (Ind:Country {name: "India", result: "Winners"})
```

Following is a sample Cypher Query which returns all the players (nodes) that belongs to the country India using WHERE clause.

```
MATCH (player)
WHERE player.country = "India"
RETURN player
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

WHERE Clause with Multiple Conditions

You can also use the WHERE clause to verify multiple conditions.

Syntax

Following is the syntax to use WHERE clause in Neo4j with multiple conditions.

```
MATCH (emp:Employee)
WHERE emp.name = 'Abc' AND emp.name = 'Xyz'
RETURN emp
```

Example

Following is a sample Cypher Query which filters the nodes in the Neo4j database using two conditions.

```
MATCH (player)
WHERE player.country = "India" AND player.runs >=175
RETURN player
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Using Relationship with Where Clause

You can also use Where clause to filter the nodes using the relationships.

Example

Assume we have the following graph in the database.

Following is a sample Cypher Query to retrieve the top scorer of India using WHERE clause as shown below.

```
MATCH (n)
WHERE (n)-[: TOP_SCORER_OF]->( {name: "India", result: "Winners"})
RETURN n
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that Neo4j returned the node, which has the relation TOP_SCORER_OF to the country with the node having the name India.

16. Neo4j – Count Function

Assume we have created a graph in the database with the following details.

Count

The **count()** function is used to count the number of rows.

Syntax

Following is the syntax of the count function.

```
MATCH (n { name: 'A' })-->(x)
RETURN n, count(*)
```

Example

Following is a sample Cypher Query which demonstrates the usage of the **count()** function.

```
Match(n{name: "India", result: "Winners"})--(x)
RETURN n, count(*)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Group Count

The **COUNT** clause is also used to count the groups of relationship types.

Example

Following is a sample Cypher Query which counts and returns the number of nodes participating in each relation.

```
Match(n{name: "India", result: "Winners"})-[r]-(x)
RETURN type (r), count(*)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Neo4j CQL – General Clauses

tutorialspoint
SIMPLYEASYLEARNING

17. Neo4j – Return Clause

The RETURN clause is used return nodes, relationships, and properties in Neo4j. In this chapter, we are going to learn how to —

- Return nodes
- Return multiple nodes
- Return relationships
- Return properties
- Return all elements
- Return a variable with column alias

Returning Nodes

You can return a node using the RETURN clause.

Syntax

Following is a syntax to return nodes using the RETURN clause.

```
Create (node:label {properties})
RETURN node
```

Example

Before proceeding with the example, create 3 nodes and 2 relationships as shown below.

```
Create (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
CREATE (Ind)-[r1:WINNERS_OF {NRR:0.938 ,pts:6}]->(CT2013)
CREATE(Dhoni)-[r2:CAPTAIN_OF]->(Ind)
```

Following is a sample Cypher Query which creates a node named Dhoni and returns it.

```
Create (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
RETURN Dhoni
```


tutorialspoint
SIMPLYEASYLEARNING

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Returning Multiple Nodes

You can also return multiple nodes using the return clause.

Syntax

Following is the syntax to return multiple nodes using the return clause.

```
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
RETURN Ind, CT2013
```

Example

Following is a sample Cypher Query to return multiple nodes using the return clause.

```
CREATE (Ind:Country {name: "India", result: "Winners"})
CREATE (CT2013:Tornament {name: "ICC Champions Trophy 2013"})
RETURN Ind, CT2013
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result. Here you can observe that Neo4j returned 2 nodes.

Returning Relationships

You can also return relationships using the Return clause.

Syntax

Following is the syntax to return relationships using the RETURN clause.

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

Example

Following is a sample Cypher Query which creates two relationships and returns them.

```
CREATE (Ind)-[r1:WINNERS_OF {NRR:0.938 ,pts:6}]->(CT2013)
CREATE(Dhoni)-[r2:CAPTAIN_OF]->(Ind)
RETURN r1, r2
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Returning Properties

You can also return properties using the RETURN clause.

Syntax

Following is a syntax to return properties using the RETURN clause.

```
Match (node:label {properties . . . . . }})
Return node.property
```

Example

Following is a sample Cypher Query to return the properties of a node.

```
Match (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
Return Dhoni.name, Dhoni.POB
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Returning All Elements

You can return all the elements in the Neo4j database using the RETURN clause.

Example

Following is an example Cypher Query to return all the elements in the database.

```
Match p = (n {name: "India", result: "Winners"})-[r]-(x)
RETURN *
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Returning a Variable With a Column Alias

You can return a particular column with alias using RETURN clause in Neo4j.

Example

Following is a sample Cypher Query which returns the column POB as Place Of Birth.

```
Match (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
Return Dhoni.POB as Place Of Birth
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

18. Neo4j – Order By Clause

You can arrange the result data in order using the ORDER BY clause.

Syntax

Following is the syntax of the ORDER BY clause.

```
MATCH (n)
RETURN n.property1, n.property2 . . . . . .
ORDER BY n.property
```

Example

Before proceeding with the example, create 5 nodes in Neo4j database as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363, country:
"India"})
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177, country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, country:"India"})
```

Following is a sample Cypher Query which returns the above created nodes in the order of the runs scored by the player using the ORDERBY clause.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Ordering Nodes by Multiple Properties

You can arrange the nodes based on multiple properties using **ORDEYBY** clause.

Syntax

Following is the syntax to arrange nodes by multiple properties using the ORDERBY clause.

```
MATCH (n)
RETURN n
ORDER BY n.age, n.name
```

Example

Following is a sample Cypher Query which arranges the nodes created earlier in this chapter based on the properties - runs and country.

```
MATCH (n)
RETURN n.name, n.runs, n.country
ORDER BY n.runs, n.country
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Ordering Nodes by Descending Order

You can arrange the nodes in a database in a descending order using the **ORDERBY** clause.

Syntax

Following is the syntax to arrange the nodes in a database.

```
MATCH (n)
RETURN n
ORDER BY n.name DESC
```

Example

Following is a sample Cypher Query which arranges the nodes in a database in a descending order using the ORDERBY clause.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

19. Neo4j – Limit Clause

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

Syntax

Following is the syntax of the LIMIT clause.

```
MATCH (n)
RETURN n
ORDER BY n.name
LIMIT 3
```

Example

Before proceeding with the example, create 5 nodes in the Neo4j database as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363, country:
"India"})
CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})
CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})
CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177, country:"India"})
CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, country:"India"})
```

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, n.runs
ORDER BY n.runs DESC
LIMIT 3
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Limit with expression

You can also use the LIMIT clause with expression.

Example

Following is a sample Cypher Query which limits the records using an expression.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
LIMIT toInt(3 * rand())+ 1
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

20. Neo4j – Skip Clause

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

Example

Before proceeding with the example, create 5 nodes as shown below.

```
CREATE(Dhawan:player{name:"shikar Dhawan", YOB: 1985, runs:363, country:
"India"})

CREATE(Jonathan:player{name:"Jonathan Trott", YOB:1981, runs:229,
country:"South Africa"})

CREATE(Sangakkara:player{name:"Kumar Sangakkara", YOB:1977, runs:222,
country:"Srilanka"})

CREATE(Rohit:player{name:"Rohit Sharma", YOB: 1987, runs:177, country:"India"})

CREATE(Virat:player{name:"Virat Kohli", YOB: 1988, runs:176, country:"India"})
```

Following is a sample Cypher Query which returns all the nodes in the database skipping the first 3 nodes.

```
MATCH (n)

RETURN n.name, n.runs

ORDER BY n.runs DESC

SKIP 3
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Skip Using Expression

You can skip the records of a result using an expression.

Example

Following is a sample Cypher Query which uses the SKIP clause with an expression.

```
MATCH (n)
RETURN n.name, n.runs
ORDER BY n.runs DESC
SKIP toInt (2*rand())+ 1
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

21. Neo4j – With Clause

You can chain the query arts together using the WITH clause.

Syntax

Following is the syntax of the WITH clause.

```
MATCH (n)
WITH n
ORDER BY n.property
RETURN collect(n.property)
```

Example

Following is a sample Cypher Query which demonstrates the usage of the WITH clause.

```
MATCH (n)
WITH n
ORDER BY n.name DESC LIMIT 3
RETURN collect(n.name)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

22. Neo4j – Unwind Clause

The unwind clause is used to unwind a list into a sequence of rows.

Example

Following is a sample Cypher Query which unwinds a list.

```
UNWIND [a, b, c, d] AS x
RETURN x
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Neo4j CQL – Functions

tutorialspoint
SIMPLYEASYLEARNING

23. Neo4J CQL – String Functions

Like SQL, Neo4J CQL has provided a set of String functions to use them in CQL Queries to get the required results.

Here, we are going to discuss some of the important and frequently used functions.

String Functions List

Following is the list of prominent String functions in Neo4j.

Sr. No.	Function	Description
1	UPPER	It is used to change all letters into upper case letters.
2	LOWER	It is used to change all letters into lower case letters.
3	SUBSTRING	It is used to get substring of a given String.
4	REPLACE	It is used to replace a substring with a given substring of a String.

Upper

It takes a string as an input and converts it into upper case letters. All CQL functions should use "( )" brackets.

Syntax

Following is the syntax of the function **upper()** in Neo4j.

```
UPPER (<input-string>)
```

Example

Before proceeding with the example, create 5 nodes in Neo4j as shown below.

```
CREATE (Dhawan:player{name: "shikar Dhawan", YOB: 1995, POB: "Delhi"})
CREATE (Jonathan:player {name: "Jonathan Trott", YOB: 1981, POB: "CapeTown"})
CREATE (Sangakkara:player {name: "Kumar Sangakkara", YOB: 1977, POB: "Matale"})
CREATE (Rohit:player {name: "Rohit Sharma", YOB: 1987, POB: "Nagpur"})
CREATE (Virat:player {name: "Virat Kohli", YOB: 1988, POB: "Delhi"})
```

Following is a sample cypher query which demonstrates the usage of the function **UPPER()** in Neo4j. Here we are trying to convert the names of all the players into upper case.

```
MATCH (n:player)
RETURN UPPER(n.name), n.YOB, n.POB
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Lower

It takes a string as an input and converts it into lower case letters. All CQL functions should use "( )" brackets.

Syntax

Following is the syntax of the function **lower()** in Neo4j.

```
LOWER (<input-string>)
```

Example

Following is a sample cypher query which demonstrates the usage of the function **LOWER()** in Neo4j. Here we are trying to convert the names of all the players into lower case.

```
MATCH (n:player)
RETURN LOWER(n.name), n.YOB, n.POB
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Substring

It takes a string as an input and two indexes: one is the start of the index and another is the end of the index and returns a substring from Start Index to End Index-1. All CQL Functions should use "**( )**" brackets.

Syntax

Following is the syntax of the function **SUBSTRING()** in Neo4j.

```
LOWER (<input-string>)
```

Example

Following is a sample Cypher query which demonstrates the usage of the function **SUBSTRING()** in Neo4j. Here, we are trying get the substring of the names of all the players.

```
MATCH (n:player)
RETURN SUBSTRING(n.name,0,5), n.YOB, n.POB
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

24. Neo4j – Aggregation Function

Like SQL, Neo4j CQL has provided some aggregation functions to use in RETURN clause. It is similar to GROUP BY clause in SQL.

We can use this RETURN + Aggregation Functions in MATCH command to work on a group of nodes and return some aggregated value.

AGGREGATION Functions List

Following is the list of aggregation functions in Neo4j.

Sr. No.	AGGREGATION Function	Description
1	COUNT	It returns the number of rows returned by MATCH command.
2	MAX	It returns the maximum value from a set of rows returned by MATCH command.
3	MIN	It returns the minimum value from a set of rows returned by MATCH command.
4	SUM	It returns the summation value of all rows returned by MATCH command.
5	AVG	It returns the average value of all rows returned by MATCH command.

COUNT

It takes the results from MATCH clause and counts the number of rows present in that result and returns the count value. All CQL functions should use "( )" brackets.

Syntax

Following is the syntax for **COUNT()** function.

```
COUNT(<value>)
```


tutorialspoint
SIMPLYEASYLEARNING

Example

Before proceeding with the example, create 4 nodes in Neo4j database as shown below.

```
CREATE (Ram:employee{name: "Ram", sal: 20000, City: "Delhi"})
CREATE (Rahim:employee{name: "Rahim", sal: 25000, City: "Hyderabad"})
CREATE (Robert:employee{name: "Robert", sal: 30000, City: "Chennai"})
CREATE (Raju:employee{name: "Raju", sal: 35000, City: "Nagpur"})
```

Following is a sample Cypher query which demonstrates the usage of the function **COUNT()** in Neo4j. Here we are trying to count the employees whose salary is greater than 27000.

```
MATCH (n:employee)
WHERE n.sal>27000
RETURN COUNT(n)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

MAX

It takes a set of rows and a <property-name> of a node or relationship as an input and finds the maximum value from the given <property-name> column of the given rows.

Syntax

Following is the syntax of the **MAX()** function.

```
MAX(<property-name>)
```

Example

Following is a sample Cypher query, which demonstrates the usage of the function **MAX()** in Neo4j. Here we are trying to calculate the maximum salaries of the employees.

```
MATCH (n:employee) RETURN MAX(n.sal)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

MIN

It takes a set of rows and a <property-name> of a node or relationship as an input and finds the minimum value from the given <property-name> column of given rows.

Syntax

Following is the syntax of the **MIN()** function.

```
MIN(<property-name>)
```

Example

Following is a sample Cypher query which demonstrates the usage of the function **MIN()** in Neo4j. Here, we are trying to calculate the minimum salaries of the employees.

```
MATCH (n:employee) RETURN MIN(n.sal)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

AVG

It takes a set of rows and a <property-name> of a node or relationship as an input and finds the average value from the given <property-name> column of given rows.

Syntax

Following is the syntax of the **AVG()** function.

```
AVG(<property-name>)
```

Example

Following is a sample Cypher query which demonstrates the usage of the function **AVG()** in Neo4j. Here, we are trying to calculate the average salaries of the employees.

```
MATCH (n:employee) RETURN AVG(n.sal)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

SUM

It takes a set of rows and a <property-name> of a node or relationship as an input and finds the summation value from the given <property-name> column of given rows.

Syntax

Following is the syntax of the **SUM()** function.

```
SUM(<property-name>)
```

Example

Following is a sample Cypher query which demonstrates the usage of the function **SUM()** in Neo4j. Here we are trying to calculate the SUM of the salaries of the employees.

```
MATCH (n:employee) RETURN SUM(n.sal)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Neo4j CQL – Admin

tutorialspoint
SIMPLYEASYLEARNING

25. Neo4j – Backup & Restore

In real-time applications, we should take backup of our application database regularly, so that we can restore to some working condition at any failure point.

This rule is applicable for both RDBMS and NoSQL databases.

In this section, we are going to discuss about two important DBA tasks.

- How to back up a Neo4j Database.
- How to restore Neo4j Database to a specific backup.

Note: These steps are applicable to Windows Operating System only. We should use similar kind of commands to do the same steps in other operating systems.

Neo4j Database Backup

Step 1: Click "Neo4j Community" using the following path:

Windows "Start" button -> "All Programs" -> "Neo4j Community" -> "Neo4j Community"

By default, it selects **c:\Users\[username]\Documents\Neo4j\default.graphdb**. However if we wish to, we can change the path to a different directory.

Step 2: Here we have changed to our Neo4j Database folder.

C:\Ne04j2.0db

Step 3: Click the "Start" button.

Once the server starts, we can observe our Neo4j Database files are generated at a specified directory.

Before taking a database backup, the first and foremost thing we should do is shutdown the Neo4j Database server.

Step 4: Click the "Stop" button to shut down the server.

Neo4j Database files are available at C:\Ne04j2.0db

Step 5: Open the command prompt.

Step 6: Create a folder "Neo4jDbBackup-01" at C:\Neo4j (This may be any location in your file system).

```
mkdir C:\Neo4j\Neo4jDbBackup-01
```

It creates a new folder "Neo4jDbBackup-01" at the specified file system location "C:\Neo4j\"

Step 7: Type the following command and press Enter key.

```
copy C:\Ne04j2.0db C:\Neo4j\Neo4jDbBackup-01
```

This means our files are copied to the required destination folder. Access that folder and observe that the folder has our database files.

« Local Disk (C:) ▸ Neo4J ▸ Neo4JDbBackup-01

Search Neo4JDbBackup-01

Organize ▾ Include in library ▾ Share with ▾ Burn New folder

Name	Date modified	Type	Size
active_tx_log	20/05/2014 15:00	File	1 KB
messages.log	06/08/2014 14:24	Text Document	77 KB
neo4j.properties	20/05/2014 15:00	PROPERTIES File	2 KB
neostore	15/08/2014 14:06	File	1 KB
neostore.id	15/08/2014 14:06	ID File	1 KB
neostore.labeltokenstore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.labeltokenstore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.labeltokenstore.db.names	15/08/2014 14:06	NAMES File	1 KB
neostore.labeltokenstore.db.names.id	15/08/2014 14:06	ID File	1 KB
neostore.nodestore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.nodestore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.nodestore.db.labels	15/08/2014 14:06	LABELS File	1 KB
neostore.nodestore.db.labels.id	15/08/2014 14:06	ID File	1 KB
neostore.propertystore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.propertystore.db.arrays	15/08/2014 14:06	ARRAYS File	1 KB
neostore.propertystore.db.arrays.id	15/08/2014 14:06	ID File	1 KB
neostore.propertystore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.propertystore.db.index	15/08/2014 14:06	INDEX File	1 KB
neostore.propertystore.db.index.id	15/08/2014 14:06	ID File	1 KB
neostore.propertystore.db.index.keys	15/08/2014 14:06	KEYS File	1 KB
neostore.propertystore.db.index.keys.id	15/08/2014 14:06	ID File	1 KB
neostore.propertystore.db.strings	15/08/2014 14:06	STRINGS File	1 KB
neostore.propertystore.db.strings.id	15/08/2014 14:06	ID File	1 KB
neostore.relationshipgroupstore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.relationshipgroupstore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.relationshipstore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.relationshipstore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.relationshiptypestore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.relationshiptypestore.db.id	15/08/2014 14:06	ID File	1 KB
neostore.relationshiptypestore.db.names	15/08/2014 14:06	NAMES File	1 KB
neostore.relationshiptypestore.db.names...	15/08/2014 14:06	ID File	1 KB
neostore.schemastore.db	15/08/2014 14:06	Data Base File	1 KB
neostore.schemastore.db.id	15/08/2014 14:06	ID File	1 KB
nioneo_logical.log.active	15/08/2014 14:06	ACTIVE File	1 KB

42 items

tutorialspoint
SIMPLYEASYLEARNING

Step 8: Use any Windows compression/decompression tool like WinZip, 7 Zip, or WinRAR to zip our Database folder.

Step 9: Now our Neo4jDbBackup-01.zip file is created. If you have any memory constraints in your file system, then remove "Neo4jDbBackup-01" folder at "C:\Neo4j\"

Neo4j Database Restore

Step 1: Shutdown the database server. Please refer to the previous steps to shut down the server.

Step 2: Empty the current database folder.

Step 3: Use any Windows compression/decompression tool like WinZip, 7 Zip, or WinRar to unzip our backup folder.

Step 4: Open the command prompt and execute the following command.

```
Copy C:\Neo4j\Neo4jDbBackup-01 C:\Ne04j2.0db
```

Now we can observe that our database folder contains working backup files

Step 5: Start the server by clicking the "Start" button.

Step 6: Execute some MATCH + RETURN commands to verify whether we have restored our database properly.

26. Neo4j – Index

Neo4j SQL supports Indexes on node or relationship properties to improve the performance of the application. We can create indexes on properties for all nodes, which have the same label name.

We can use these indexed columns on MATCH or WHERE or IN operator to improve the execution of CQL command.

In this chapter, we will discuss how to –

- Create an Index
- Delete an Index

Creating an Index

Neo4j CQL provides "CREATE INDEX" command to create indexes on Node or Relationship properties.

Syntax

Following is the syntax to create an index in Neo4j.

```
CREATE INDEX ON:label (node)
```

Example

Before proceeding with the example, create a node Dhawan as shown below.

```
CREATE (Dhawan:player{name: "shikar Dhawan", YOB: 1995, POB: "Delhi"})
```

Following is a sample Cypher Query to create an index on the node Dhawan in Neo4j.

```
CREATE INDEX ON:player(Dhawan)
```


tutorialspoint
SIMPLYEASYLEARNING

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown below.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Deleting an Index

Neo4j CQL provides a "DROP INDEX" command to drop an existing index of a Node or Relationshis property.

Syntax

Following is the syntax to create an index in Neo4j.

```
DROP INDEX ON:label(node)
```

Example

Following is a sample Cypher Query to create an index on the node named “Dhawan” in Neo4j.

```
DROP INDEX ON:player(Dhawan)
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

27. Neo4j – Create Unique Constraint

In Neo4j database, CQL CREATE command always creates a new node or relationship which means even though you use the same values, it inserts a new row. As per our application requirements for some nodes or relationships, we have to avoid this duplication. For this, we should use some database constraints to create a rule on one or more properties of a node or relationship.

Like SQL, Neo4j database also supports UNIQUE constraint on node or relationship properties. UNIQUE constraint is used to avoid duplicate records and to enforce data integrity rule.

Create UNIQUE Constraint

Neo4j CQL provides "CREATE CONSTRAINT" command to create unique constraints on node or relationship properties.

Syntax

Following is the syntax to create a UNIQUE constraint in Neo4j.

```
MATCH (root {name: "Dhawan"})
CREATE UNIQUE (root)-[:LOVES]-(someone)
RETURN someone
```

Example

Before proceeding with the example, create 4 nodes as shown below.

```
CREATE(Dhawan:player{id:001, name: "shikar Dhawan", YOB: 1995, POB: "Delhi"})
CREATE(Jonathan:player {id:002, name: "Jonathan Trott", YOB: 1981, POB:
"CapeTown"})
CREATE(Sangakkara:player {id:003, name: "Kumar Sangakkara", YOB: 1977, POB:
"Matale"})
CREATE(Rohit:player {id:004, name: "Rohit Sharma", YOB: 1987, POB: "Nagpur"})
CREATE(Virat:player {id:005, name: "Virat Kohli", YOB: 1988, POB: "Delhi"})
```

Following is a sample Cypher Query to create a UNIQUE constraint on the property id using Neo4j.

```
CREATE (Jadeja:player {id:002, name: "Ravindra Jadeja", YOB: 1988, POB:
"NavagamGhed"})
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.

Verification

Now, try to add another node with a redundant id value. Here, we are trying to create a node with id **002**.

```
CREATE (Jadeja:player {id:002, name: "Ravindra Jadeja", YOB: 1988, POB:
"NavagamGhed"})
```

If you execute this query, you will get an error message as shown in the following screenshot.

28. Neo4j – Drop Unique

We have already discussed creating UNIQUE constraint operations with examples in the previous chapter. In this chapter, we will discuss dropping UNIQUE constraint operation with examples.

Neo4j CQL provides "DROP CONSTRAINT" command to delete existing Unique constraint from a node or relationship property.

Syntax

Following is the syntax for dropping a UNIQUE constraint in Neo4j.

```
DROP CONSTRAINT ON (node:label)
ASSERT node.id IS UNIQUE
```

Example

Following is a sample Cypher Query to remove the UNIQUE constraint on the property **id**.

```
DROP CONSTRAINT ON (n:player)
ASSERT n.id IS UNIQUE
```

To execute the above query, carry out the following steps -

Step 1: Open the Neo4j desktop App and start the Neo4j Server. Open the built-in browser app of Neo4j using the URL http://localhost:7474/ as shown in the following screenshot.

Step 2: Copy and paste the desired query in the dollar prompt and press the play button (to execute the query) highlighted in the following screenshot.

Result

On executing, you will get the following result.