NoSQL

Graph Database

Redis

Neo4j the graph database

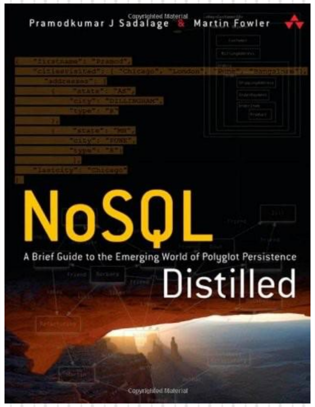
Cassandra

CouchDB relax

mongoDB

membase

riak


Pramodkumar J Sadalage & Martin Fowler

NoSQL Distilled

A Brief Guide to the Emerging World of Polyglot Persistence

Graph Database

- A graph database is a database designed to store and query data represented in the form of a graph. A graph consists of vertices (also called nodes) and edges, which represent the relationships between the vertices
- Graph is a pictorial representation of a set of objects where some pairs of objects will be connected by links. Graph is composed of two elements - nodes (vertices) and relationships (edges).
- Graph database is used for modelling data in the form of graph. Here, nodes of a graph depict the entities whereas relationships depict the association of these nodes.
- Graph databases are particularly useful for storing and querying data that has complex relationships, such as social networks, recommendation engines, and fraud detection systems.

Popular Graph Database

- Neo4j
- OrientDB, HypherGraphDB
- GraphBase
- InfiniteGraph
- JanusGraph
- Amazon Neptune
- ArangoDB
- AllegroGraph.

Graph Database Use Case

- **Social networks**

  Graph databases can be used to store and query data about relationships between people, such as friendships, family relationships, and professional connections. This can be used to build social networking platforms, recommendation engines, and other applications.

- **Fraud detection**

  Graph databases can be used to identify patterns of fraudulent activity by analyzing the relationships between entities such as individuals, businesses, and transactions.

- **Recommendation engines**

  Graph databases can be used to store and query data about users and their interactions with products or content. This can be used to build recommendation engines that suggest products or content to users based on their interests and past behavior.

- **Supply chain management**

  Graph databases can be used to store and query data about the relationships between different entities in a supply chain, such as suppliers, manufacturers, and retailers. This can be used to optimize logistics and supply chain management.

- **Bioinformatics**

  Graph databases can be used to store and query data about the relationships between different biological entities, such as genes, proteins, and diseases. This can be used to study the relationships between different biological processes and to develop new drugs and treatments.

Graph Database Properties

- **Flexible data model**

  Graph databases use a flexible data model that allows for the representation of complex relationships between entities. This makes it easy to store and query data that has many different types of relationships and connections.

- **Efficient querying**

  Graph databases are optimized for efficient querying of data, particularly when it comes to traversing relationships between entities. This makes it easy to find and retrieve data about specific entities and their relationships with other entities.

- **Scalability**

  Graph databases are designed to scale well as the size of the data increases. This makes them suitable for storing and querying large amounts of data.

- **ACID transactions**

  Many graph databases support ACID (Atomicity, Consistency, Isolation, Durability) transactions, which ensure that data is stored and accessed in a consistent and reliable manner. This is important for applications that require high levels of data integrity and reliability.

- **High performance**

  Graph databases are optimized for high performance and can handle a large number of queries and updates in real-time. This makes them suitable for use in high-traffic applications.

Why Graph Databases?

- These days, most of the data exist in the form of the relationship between different objects and more often, relationship between the data is more valuable than the data itself.
- Relational databases store highly structured data which have many records storing same type of data so that they can be used for storing 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.
- Data model for graph databases is easy when compared to other databases and they can be used with OLTP systems as they provide features like transactional integrity and operational availability.

RDBMS Vs Graph Database

S.No	RDBMS	Graph Database
1	Tables	Graphs
2	Rows	Nodes
3	Columns and Data	Properties and its values
4	Constraints	Relationships
5	Joins	Traversal

Neo4j

- Neo4j is one of the popular Graph Databases and Cypher Query Language (CQL). Neo4j is written in Java Language.
- Neo4j is the world's leading open source Graph Database which is developed using Java technology. It is highly scalable and schema free (NoSQL).
- Neo4j has its own query language that called Cypher Language. It is similar to SQL, remember one thing Neo4j does not work with tables, row or columns it deals with nodes. It is more satisfied to see the data in a graph format rather than in a table format.
- Neo4j used **ASCII-Art** to create pattern.

  (X)-[:CreatePattern]->(Y)

  In the Neo4j the nodes are represented by “( )”.

  The relationship is represented by ” -> “.

  What kind of relationship is between the nodes are represented by ” [ ] ” like [:CreatePattern]

Neo4j

- Neo4j deals with nodes and the nodes contains labels that could be “Person”, “Employee”, “Employer” anything that can define the type of value field.
- Neo4j also have properties like “name”, “employee_id”, “phone_number”, basically that will gives us information about the nodes.
- Neo4j’s relationship also can contain the properties but this is not mandatory.
- Neo4j stores and present the data in the form of graph not in tabular format or not in a Json format. Here the whole data is represented by nodes and there you can create a relationship between nodes. That means the whole database collection will look like a graph, that’s why it is making it unique from other database management system.
- MS Access, SQL server all the relational database management system use tables to store or present the data with the help of column and row but Neo4j doesn’t use tables, row or columns like old school style to store or present the data.

Advantages of Neo4j

- **Flexible data model** − Neo4j offers a flexible, simple but yet powerful data model, which can be simply changed based on the applications and industries.
- **Real-time insights** − Neo4j produces 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, it is easy to represent connected and semi-structured data.
- **Easy retrieval** − Using Neo4j, it is not only easy to represent but it is also easily to retrieve (traverse/navigate) connected data faster when compared to other databases.
- **Cypher query language** − Neo4j offers a declarative query language for representing the graph visually, using an ascii-art syntax. 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

- **Data model (flexible schema)** − Neo4j follows a data model named native property graph model. Here, the graph consists of nodes (entities) and these nodes will be connected with each other (depicted by relationships). Nodes and relationships store data in key-value pairs called as properties.

  In Neo4j, you don't have to follow a fixed schema. You can add or remove properties as based on your requirement. It also provides schema constraints.

- **ACID properties** − Neo4j supports full ACID (Atomicity, Consistency, Isolation, and Durability) rules.

- **Scalability and reliability** – Database can be scaled 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.

Features of Neo4j

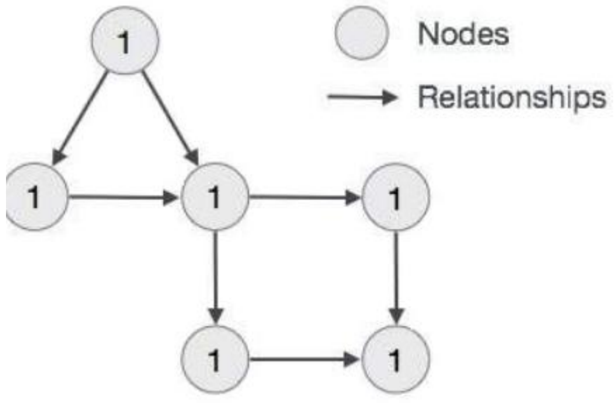
- **Cypher Query Language** – Neo4j offers a powerful declarative query language known as Cypher which uses ASCII-art for depicting graphs. Cypher is easy to learn and is used for creating and retrieving relations between data without using the complex queries like Joins.
- **Built-in web application** – Neo4j offers 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. Apart from this, you can also work with other databases like MongoDB, Cassandra, etc.
- **Indexing** – Neo4j supports Indexes by using Apache Lucence.

Neo4j Property Graph Data Model

- Neo4j Graph Database follows the Property Graph Model for storing and managing its data.
- Below are the key features of Property Graph Model
  - Model represents data in Nodes, Relationships and Properties
  - Properties will be key-value pairs
  - Nodes will be represented using circle and Relationships will be 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 consist properties
  - Relationships connects nodes
- In Property Graph Data Model, relationships should be directional. If you try to create relationships without direction, it will throw an error message.
- In Neo4j too, relationships should be directional. If we try to create relationships without direction, then Neo4j throws an error message as "Relationships should be directional".

Neo4j Property Graph Data Model

- Neo4j Graph Database will store all of its data in Nodes and Relationships. There is no need of any additional RRBMS Database nor any SQL database for storing Neo4j database data. It stores its data in terms of Graphs in its native format.
- Neo4j makes use of Native GPE (Graph Processing Engine) to work with its Native graph storage format.
- Main building blocks of Graph DB Data Model are
  - Nodes
  - Relationships
  - Properties
- Below is a simple example of a Property Graph.



Simple Graph

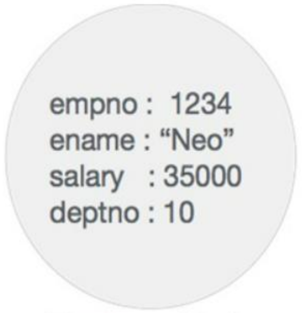
- Here, Nodes are represented using Circles and relationships are represented using Arrows. Relationships are directional. Node's data can be represented in terms of Properties (key-value pairs). In this example, each Node's Id property is represented within the Node's Circle.

Building Blocks in Neo4j

- Neo4j Graph Database has below building blocks
  - Nodes
  - Properties
  - Relationships
  - Labels
  - Data Browser

Node

Node is a fundamental unit of a Graph. It consists of properties with key-value pairs .



Employee Node

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

Building Blocks in Neo4j

Properties

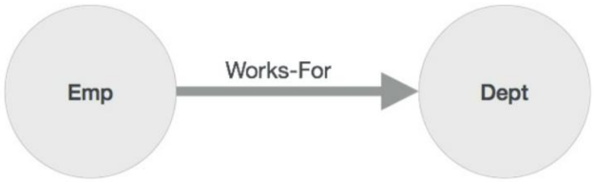
- Property is a key-value pair for describing Graph Nodes and Relationships.

  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 below figure

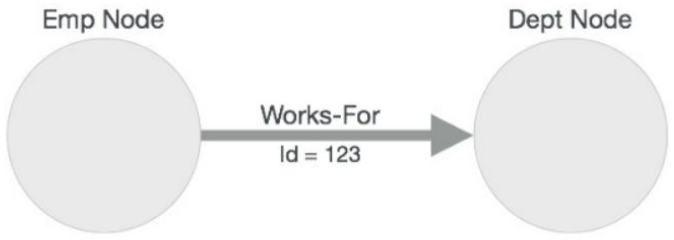


Here, Emp and Dept are two different nodes. "WORKS_FOR" is a relationship between Emp and Dept nodes.

- As it denotes, the arrow mark from Emp to Dept, this relationship describes
  - **Emp WORKS_FOR Dept**
- Each relationship consists of 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.

Building Blocks in Neo4j

Similar to nodes, relationships also consist of properties as key-value pairs.



Here, "WORKS_FOR" relationship holds 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 consists of one or more labels. New labels can be created for existing nodes or relationships and existing labels can be removed from the existing nodes or relationships.
- In the above diagram, 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".

CQL in Neo4j

What is CQL in Neo4j?

CQL stands for Cypher Query Language. Similar to Oracle Database which has query language SQL, Neo4j also has CQL as query language.

Neo4j CQL

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

Features of Neo4j

- Neo4j CQL has commands to perform Database operations.
- Neo4j CQL supports many clauses like WHERE, ORDER BY, etc., for writing very complex queries in an easy manner.
- Neo4j CQL supports few functions like String, Aggregation. Apart from these, it also supports some Relationship Functions.

Neo4j CQL Clauses

- Writing clauses
- Reading clauses
- Projecting clauses
- Reading sub-clauses
- Reading hints
- Reading/Writing clauses
- Set Operations
- Importing Data
- Schema clauses

Neo4j CQL Read Clauses

S.No	Read Clauses	Usage
1	MATCH	Used for searching data with a specified pattern.
2	OPTIONAL MATCH	This is similar to match clause, but the only difference is that it can use nulls in case of missing parts of the pattern.
3	WHERE	This clause id is used for adding contents to the CQL queries.
4	START	Used for finding the starting points through the legacy indexes.
5	LOAD CSV	Used for importing data from CSV files.

Neo4j CQL Write Clauses

S.No	Write Clauses	Usage
1	CREATE	Used for creating nodes, relationships, and properties.
2	MERGE	It verifies if the specified pattern exists in the graph. If not, it creates the pattern.
3	SET	Used for updating labels on nodes, properties on nodes and relationships.
4	DELETE	Used for deleting nodes and relationships or paths etc. from the graph.
5	REMOVE	Used for removing properties and elements from nodes and relationships.
6	FOREACH	This class is used for updating the data within a list.
7	CREATE UNIQUE	CREATE and MATCH can be used for getting a unique pattern by matching the existing pattern and creating the missing one.
8	Import CSV files with Cypher	Load CSV is used for importing data from .csv files.

Neo4j CQL General Clauses

S.No	General Clauses	Usage
1	RETURN	Used for defining what to include in the query result set.
2	ORDER BY	Used for arranging the output of a query in order. It is used along with the clauses RETURN or WITH.
3	LIMIT	Used for limiting the rows in the result to a specific value.
4	SKIP	Used for deleting nodes and relationships or paths etc. from the graph.
5	WITH	Used to chain the query parts together.
6	UNWIND	Used for expanding a list into a sequence of rows.
7	UNION	Used for combining the result of multiple queries.
8	CALL	Used to invoke a procedure deployed in the database.

Neo4j CQL Functions

S.No	CQL Functions	Usage
1	String	Strings are used to work with String literals.
2	Aggregation	Aggregation used for performing some aggregation operations on CQL Query results.
3	Relationship	A relationship is used to get details of relationships such as startnode, endnode, etc.

Neo4j CQL Datatypes

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

Neo4j CQL Operators

S.No	CQL Type	Usage
1	Mathematical	+, -, *, /, %, ^
2	Comparison	+, <>, <, >, <=, >=
3	Boolean	AND, OR, XOR, NOT
4	String	+
5	List	+, IN, [X], [X…..Y]
6	Regular Expression	= -
7	String matching	STARTS WITH, ENDS WITH, CONSTRAINTS

Neo4j CQL Boolean Operators

S.No	CQL Boolean	Usage
1	AND	It is a Neo4j CQL keyword which supports AND operation. It is similar to SQL AND operator.
2	OR	It is a Neo4j CQL keyword which supports OR operation. It is similar to SQL OR operator.
3	NOT	It is a Neo4j CQL keyword which supports NOT operation. It is similar to SQL NOT operator.
4	XOR	It is a Neo4j CQL keyword which supports XOR operation. It is similar to SQL XOR operator.

Neo4j CQL Comparison Operators

S.No	CQL Comparison	Usage
1	=	It is a Neo4j CQL "Equal To" operator.
2	< >	It is a Neo4j CQL "Not Equal To" operator.
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.

Neo4J CQL Creating Nodes

Create nodes using Neo4j CQL.

As discussed, a node is a data/record in a graph database which can be created in Neo4j using the CREATE clause. This chapter will explain 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
- Return the created node

Creating Single and Multiple Nodes

1. Creating a Single node

Node in Neo4j can be created by just specifying the name of the node which is created along with the CREATE clause.

Syntax

```
CREATE (node_name);
```

Note: − Semicolon (;) is optional.

Example

```
CREATE (n)
```

Verification

```
MATCH(n) Return n
```

2. Creating a Multiple node

Create clause of Neo4j CQL is used for creating multiple nodes at the same time. For this, pass the names of the nodes to be created, separated by a comma.

Syntax

```
CREATE (node1),(node2),(node3);
```

Note: − Semicolon (;) is optional.

Example

```
CREATE (n1)(n2)(n3)
```

Verification

```
MATCH(n1,n2,n3) Return n1,n2,n3
```

Create Node with Label and Multiple Labels

3. Creating a Node with a Label

A label in Neo4j is used for grouping (classify) the nodes using labels. Label can be created for a node in Neo4j using CREATE clause.

Syntax

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

Note: − Semicolon (;) is optional.

Example

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

Verification

```
MATCH(Dhawan) Return Dhawan
```

4. Creating a Node with Multiple Labels

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

Syntax

```
CREATE (node:label1:label2;label3.....labeln);
```

Note: − Semicolon (;) is optional.

Example

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

Verification

```
MATCH(Dhawan) Return Dhawan
```

Create Node with Properties

5. Creating a 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

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

Example

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

Verification

```
MATCH(Dhawan) Return Dhawan
```

6. Returning the Created Node

Throughout the chapter, we have used MATCH (n) RETURN n query to view the created nodes. This query will return 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

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

Example

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

Neo4J CQL Creating a Relationship

How to create a relationship in Neo4j?

In Noe4j, a relationship is an element using which two nodes of a graph can be connected. These relationships will have direction, type and the form patterns of data. This chapter will explain how to

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

Neo4J CQL Creating a Relationship

1. Creating Relationships

A relationship can be created using the CREATE clause. We will specify the relationship within the square braces “[ ]” based on the direction of the relationship it is placed between hyphen “ - ” and arrow “ → ” as shown in below syntax.

Syntax

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

Example

First, 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)
```

At last, return both the nodes to see the created relationship.

```
RETURN Dhawan, Ind
```

Neo4J CQL Creating a Relationship

2. Creating a Relationship Between the Existing Nodes

A relationship can be created between the existing nodes using MATCH clause.

Syntax :

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

Example

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

Neo4J CQL Creating a Relationship

3. Creating a Relationship with Label and Properties

A relationship can be created with label and properties using the CREATE clause.

Syntax :

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

Example

```
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
```

4. Creating a Complete Path

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

Syntax :

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

(Node2 {properties})[:Relationship_Type]->(Node3 {properties})

RETURN p
```

Example

```
CREATE p = (Dhawan {name:"shikhar Dhawan"})-[:TOP_SCORER_OF]->

(CT2013 {name:"Champions Trophy 2020"})

RETURN p
```

Neo4J CQL MERGE Commands

How to merge command in Neo4j?

MERGE command in Neo4j 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 will return the results. Else, it will create a new node/relationship and returns the results.In this chapter we will explain how to

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

Syntax:

```
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)
```

Neo4J CQL MERGE Commands

1. Merging a Node with a Label

A node in the database can be merged based on the label using the MERGE clause. If you are trying to merge a node based on the label, then Neo4j verifies if there exists any node with the given label. If not, the current node will be created.

Syntax :

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

Example 1

Below is a sample Cypher Query which merges a node into Neo4j (based on label). When this query is executed, Neo4j will verify if there is any node with the label player. If not, it will create a node named "Jadeja" and returns it.

If any node already exists with the given label, Neo4j will return them all.

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

Example 2

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

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

Neo4J CQL MERGE Commands

2. Merging a Node with Properties

A node can be merged with a set of properties. If you do so, Neo4j will search for an equal match for the specified node, including the properties. If it doesn't find any such node it will create one.

Syntax :

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

Example

Below is a sample Cypher Query to merge a node using properties. This query will try to merge the node named "jadeja" using properties and label. As there is no such node with the exact label and properties, Neo4j will create one.

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

Neo4J CQL MERGE Commands

3. OnCreate and OnMatch

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

Syntax :

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

Example

Below 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
```

Neo4J CQL MERGE Commands

4. Merge a Relationship

Just like nodes, relationships can also be merged using the MERGE clause.

Example

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

As 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
```

Neo4J CQL - SET CLAUSE

How set clause works in Neo4j?

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

In this chapter, we will explain how to

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

Neo4J CQL - SET CLAUSE

1. Setting a Property

SET clause can be used to create a new property in a node.

Syntax:

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

Example:

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

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

Below is a sample Cypher Query for creating a property named “highestscore” with value “187”.

```
MATCH (Dhawan:player{name: "shikar Dhawan", YOB: 1985, POB:
    "Delhi"})
SET Dhawan.highestscore = 187
RETURN Dhawan
```

Neo4J CQL - SET CLAUSE

2. Removing a Property

Existing property can be removed by passing NULL as value to it.

Syntax:

Below 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"})
```

Below 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
```

Neo4J CQL - SET CLAUSE

3. Setting Multiple Properties

In a similar way, multiple properties can be created in a node using the Set clause. To do so, you have to specify these key value pairs with commas.

Syntax:

Below is the syntax for creating multiple properties in a node using the SET clause.

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

Example:

Below 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
```

Neo4J CQL - SET CLAUSE

4. Setting a Label on a Node

A label can be set to an existing node using the SET clause.

Syntax:

Below 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"})
```

Below 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
```

Neo4J CQL - DELETE CLAUSE

How delete clause works in Neo4j?

Nodes and relationships can be deleted from a database using DELETE clause.

Deleting All Nodes and Relationships

Syntax:

Below is the query for deleting all the nodes and the relationships in the database using the DELETE clause.

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

Deleting a Particular Node

To delete a particular node, mention all the details of the node in the place of "n" in the above query.

Syntax:

Below is the syntax for deleting a particular node from Neo4j using the DELETE clause.

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

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"})
```

Below 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
```

Neo4J CQL - REMOVE CLAUSE

How remove clause works in Neo4j?

REMOVE clause is used for removing the properties and labels from graph elements (Nodes or Relationships).

Main difference between Neo4j CQL DELETE and REMOVE commands is

- DELETE operation is used for deleting nodes and associated relationships.
- REMOVE operation is used for removing labels and properties.

Removing a Property

A property of a node can be removed using MATCH along with the REMOVE clause.

Syntax:

```
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"})
```

Below 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
```

Neo4J CQL - REMOVE CLAUSE

Removing a Label from a Node

Same like property, a label can be removed from an existing node using remove clause.

Syntax:

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

Example:

Below is a sample Cypher Query for removing 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
```

Neo4J CQL - REMOVE CLAUSE

Removing Multiple Labels

Multiple labels can also be removed from an existing node.

Syntax:

```
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"})
```

Below is a sample Cypher Query for removing multiple labels from a node.

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

Neo4J CQL - FOREACH CLAUSE

What is the use of Foreach clause in Neo4j?

FOREACH clause is used for updating data within a list whether components of a path, or result of aggregation.

Syntax

Below 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
```

Below 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)
```

Verification

```
Match(Dhawan) Return Dhawan
```

Neo4J CQL - MATCH CLAUSE

1. Get All Nodes Using Match

MATCH clause of Neo4j can be used to 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)
```

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

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

Neo4J CQL - MATCH CLAUSE

2.Getting All Nodes Under a Specific Label

Match clause can be used to get all the nodes under a specific label.

Syntax:

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

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

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

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

3.Match by Relationship

Nodes can be retrieved based on relationship using the MATCH clause.

Syntax:

Below is the syntax for retrieving nodes based on the relationship using the MATCH clause.

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

Below 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
```

3.Delete All Nodes: **All the nodes can be deleted using MATCH clause.**

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

Neo4J CQL - OPTIONAL MATCH CLAUSE

What is optional match clause in Neo4j?

OPTIONAL MATCH clause is used for searching the pattern described in it and nulls are used for missing parts of the pattern.

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

Syntax:

Below is the syntax of the OPTIONAL MATCH with relationship.

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

Below is a sample Cypher Query which tries to retrieve the relations from the node ICCT2013.

As there are no such nodes, it will return null.

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

Neo4J CQL - WHERE CLAUSE

What is the use of where clause in Neo4j?

Like SQL, Neo4j CQL offers WHERE clause in CQL MATCH command for filtering the results of a MATCH Query.

Syntax:

Below is the syntax of the WHERE clause.

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

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"})
```

Below is a sample Cypher Query which will return all the players (nodes) which belong to the country India using WHERE clause.

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

Neo4J CQL - WHERE CLAUSE

WHERE Clause with Multiple Conditions

WHERE clause can also be used to verify multiple conditions.

Syntax:

Below 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
```

Below 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
```

Neo4J CQL - WHERE CLAUSE

Using Relationship with Where Clause

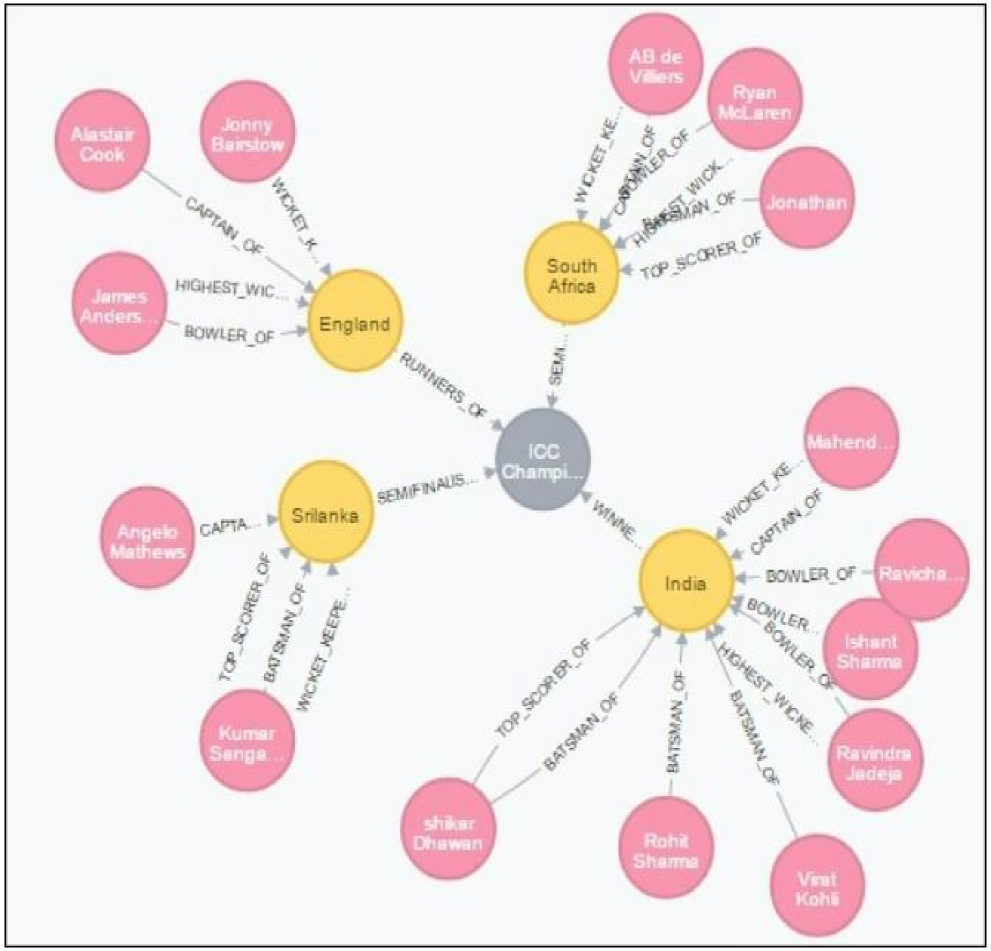
Where clause can also be used to filter the nodes using the relationships.

Example

Below is a sample Cypher Query for retrieving the top scorer of India using WHERE clause as shown below.

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

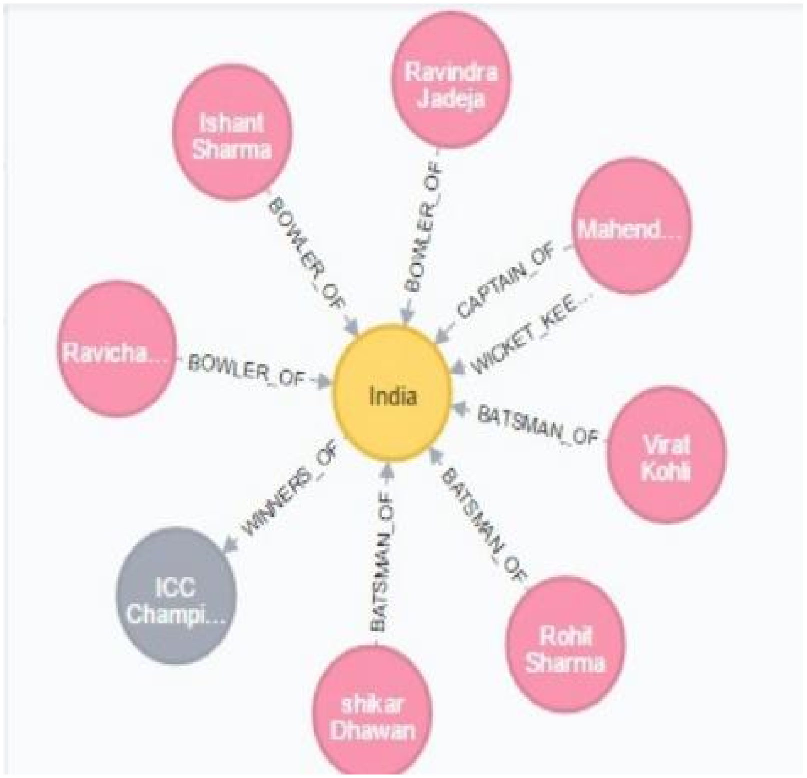
Assume that we have below graph in the database.



Neo4J CQL - COUNT Function

How to use count function in Neo4j?

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



Count

count() function is used for counting the number of rows.

Syntax:

Below is the syntax of the count function.

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

Example

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

Match(n{name: "India", result: "Winners"})--(x)

RETURN n, count(*)

Group Count :

COUNT clause is also used for counting the groups of relationship types.

Below 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(*)

Neo4J CQL - RETURN Clause

What is the use of Return clause in Neo4j?

RETURN clause is used to return nodes, relationships, and properties in Neo4j. In this chapter, we will explain how to

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

1. Returning Nodes

You can return a node using the RETURN clause.

Syntax:

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

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

RETURN node
```

Example

Below 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
```

Neo4J CQL - RETURN Clause

2. Returning Multiple Nodes

Multiple nodes can be returned using the return clause.

Syntax:

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

```
Create (node:label {properties}),(node1:label {properties})
RETURN node, node1
```

Example

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

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

Neo4J CQL - RETURN Clause

3. Returning Relationships

You can also return relationships using the Return clause.

Syntax:

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

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

Example

Below 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
```

Neo4J CQL - RETURN Clause

4. Returning Properties

You can also return properties using the Return clause.

Syntax:

Below is the syntax to return properties using the RETURN clause.

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

Example

Below 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
```

5. Returning All Elements

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

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

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

Neo4J CQL - RETURN Clause

5. Returning a Variable with a Column Alias

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

Example

Below 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
```

Neo4J CQL - ORDER BY Clause

What is the use of order by clause in Neo4j?

Result data can be arranged in order using the ORDER BY clause.

Syntax

```
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"})
```

Below 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
```

Neo4J CQL - ORDER BY Clause

Ordering Nodes by Multiple Properties

Nodes can be arranged based on multiple properties using ORDEYBY clause.

Syntax

```
MATCH (n)
RETURN n
ORDER BY n.properties1, n.properties
```

Example

Below 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
```

Neo4J CQL - ORDER BY Clause

Ordering Nodes by Descending Order

Nodes can be arranged in a database in a descending order using the ORDERBY clause.

Syntax

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

Example

Below 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
```

Neo4J CQL - LIMIT Clause

How to use limit clause neo4j?

limit clause is used for limiting the number of rows in the output.

Syntax

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

Example

Below 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
```

Limit with expression

LIMIT clause can also be used with expression.

Example

Below 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
```

Neo4J CQL - SKIP Clause

What is the use of skip clause in Neo4j?

SKIP clause is used for defining from which row to start including the rows in the output.

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

Below 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
```

Skip using expression

Records of a result can be skipped using an expression.

Example

Below 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
```

Neo4J CQL - WITH Clause

What is the use of with clause in Neo4j?

Query arts can be chained together using WITH clause.

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

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

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

Neo4J CQL - UNWIND Clause

What is unwind clause in Neo4j?

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

Example

Below is a sample Cypher Query which unwinds a list.

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

Neo4J CQL - STRING Functions

What are string functions in Neo4j?

Similar to SQL, Neo4J CQL also provides a set of String functions which are used in CQL Queries for getting the required results.

String Fucntions List

- UPPER - > This is used for changing all letters into upper case letters.
- LOWER - > This is used for changing all letters into lower case letters.
- SUBSTRING - > This is used for getting substring of a given String.
- REPLACE -> This is used for replacing a substring with a given substring of a String.

Neo4J CQL - STRING Functions

What is upper function in Neo4j?

Upper function takes a string as an input and converts it into upper case letters. All CQL functions should use "( )" brackets. UPPER (<input-string>)

Example: Create 5 nodes in Neo4j

Below is a sample cypher query which demonstrates the usage of the function UPPER() in Neo4j. Here we are converting names of all the players into upper case.

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

What is lower function in Neo4j?

Lower function takes a string as an input and converts it into lower case letters. All CQL functions should use "( )" brackets. LOWER (<input-string>)

Example: Create 5 nodes in Neo4j

Below is a sample cypher query which demonstrates the usage of the function LOWER() in Neo4j. Here we are converting names of all the players into lower case.

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

Neo4J CQL - STRING Functions

What is a substring in Neo4j?

Substring Function takes a string as an input and two indexes: one is the start of the index and the other one is the end of the index and will retunr a substring from Start Index to End Index-1. All CQL Functions should use "( )" brackets. SUBSTRING (<input-string>)

Example: Create 5 nodes in Neo4j

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

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

Neo4J CQL - STRING Functions

What is a substring in Neo4j?

Substring Function takes a string as an input and two indexes: one is the start of the index and the other one is the end of the index and will retunr a substring from Start Index to End Index-1. All CQL Functions should use "( )" brackets. SUBSTRING (<input-string>)

Example: Create 5 nodes in Neo4j

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

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

Neo4J CQL - AGGREGATE Functions

What is the use of aggregation functions in Neo4j?

Similar to SQL, Neo4j CQL also has some aggregation functions which are used in RETURN clause. It is similar to GROUP BY clause in SQL.

This RETURN + Aggregation Functions can be used in MATCH command to work on a group of nodes and return some aggregated value.

Aggregate Function List

- COUNT - It returns the number of rows returned by MATCH command.
- MAX - It returns the maximum value from a set of rows returned by MATCH command.
- MIN - It returns the minimum value from a set of rows returned by MATCH command.
- SUM - It returns the summation value of all rows returned by MATCH command.
- AVG - It returns the average value of all rows returned by MATCH command.

Neo4J CQL - AGGREGATE Functions

1. 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: COUNT(<value>)

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"})
```

Example:

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)
```

Neo4J CQL - AGGREGATE Functions

2. 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: 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)

3. 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: 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)

Neo4J CQL - AGGREGATE Functions

4. 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: 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)
```

5. 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: 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)
```

Neo4J CQL - CREATE INDEX

What is the use of Index in Neo4j?

Neo4j SQL provides support using Indexes on node or relationship properties to improve the performance of the application. Indexes can be created on properties for all nodes which have the same label name.

These indexed columns can be used on MATCH or WHERE or IN operator for improving the execution of CQL command.

In this chapter, we will discuss how to

- Create an Index
- Delete an Index

Neo4J CQL - CREATE INDEX

1. Creating an Index

Neo4j CQL offers "CREATE INDEX" command for creating indexes on Node or Relationship properties.

Syntax: CREATE INDEX INDEX_NAME FOR(Node:label) ON(node.property)

Example:

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

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

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

CREATE INDEX IND1 FOR(Dhawan:player) ON(Dhawan.name)

2. Deleting an Index

Neo4j CQL provides a "DROP INDEX" command which is used to drop an existing index of a Node or Relationships property.

Syntax: DROP INDEX INDEX_NAME FOR(Node:label) ON(node.property)

Example:

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

DROP INDEX IND1 FOR(Dhawan:player) ON(Dhawan.name)

Neo4J CQL - CREATE UNIQUE CONSTRAINT

How to create a unique constraint in Neo4j?

In Neo4j database, CQL CREATE command will always create a new node or relationship which means even though you use the same values, it will insert a new row. Based on the application requirements for some nodes or relationships, this duplication should be avoided. For this, some database constraints should be used to create a rule on one or more properties of a node or relationship.

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

Neo4J CQL - CREATE UNIQUE CONSTRAINT

Create UNIQUE Constraint

Neo4j CQL offers "CREATE CONSTRAINT" command for creating unique constraints on node or relationship properties.

Syntax:

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

Example:

Before proceeding with the example, create 5 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"})
```

Below is a sample Cypher Query for creating a UNIQUE constraint on the property id using Neo4j.

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

Verification: Now, try adding another node with a redundant id value. Here, we are trying to create a node with id 002. Same above query

Neo4J CQL - CREATE UNIQUE CONSTRAINT

What is the use of drop unique constraint in Neo4j?

Neo4j CQL offers "DROP CONSTRAINT" command for deleting existing Unique constraint from a node or relationship property.

Syntax:

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

Example:

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

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