AIM: WORKING WITH Neo4j DATABASE

What is Neo4j

Neo4j is a NoSQL database. It is highly scalable and schema-free. It's the world's most popular graph database management system. Neo4j was developed by Neo technology and called an ACID-compliant transactional database with native graph storage and processing.

Neo4j is implemented in Java language and it can be accessed by other language using Cypher Query Language (CQL). Neo4j is a way faster than traditional databases.

Neo4j Working

- Neo4j stores and displays data in the form of graph. In Neo4j, data is represented by nodes and relationships between those nodes.
- Neo4j databases (as with any graph database) are a lot different to relational databases such as MS Access, SQL Server, MySQL, etc. Relational databases use tables, rows, and columns to store data. They also present data in a tabular fashion.
- Neo4j doesn't use tables, rows, or columns to store or present data.
- Neo4j is best for storing data that has many interconnecting relationships. That's why graph databases like Neo4j have an advantage and much better at dealing with relational data than relational databases are.
- The graph model doesn't usually require a predefined schema. So there is no need to create the database structure before you load the data (like you do in a relational database). In Neo4j, the data is the structure. Neo4j is a "schema-optional" DBMS.
- In Neo4j, there is no need to set up primary key/foreign key constraints to predetermine which fields can have a relationship, and to which data. You just have to define the relationships between the nodes you need.

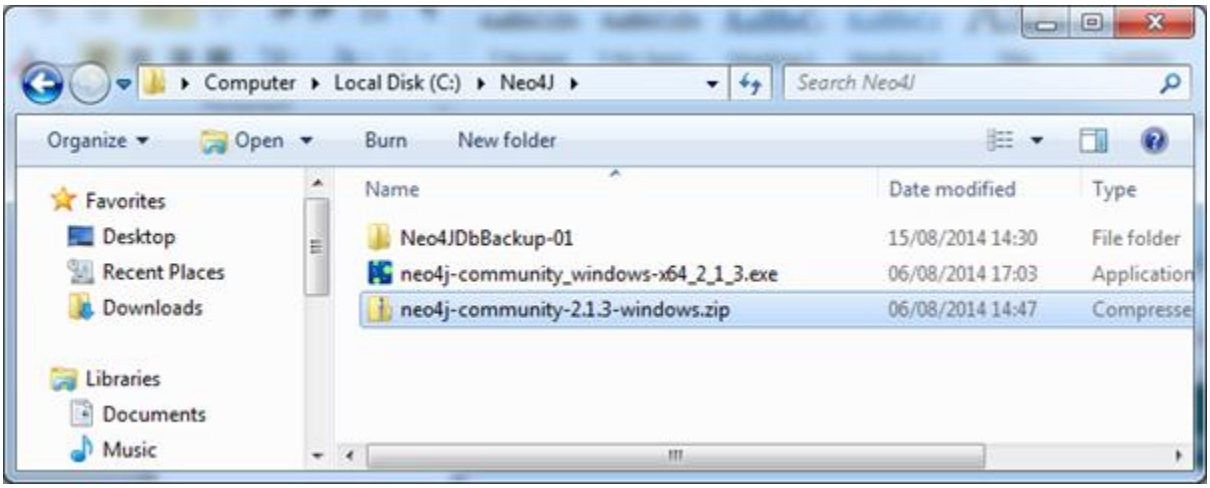
Step 1 - Download the latest Neo4j Server installation file from Neo4j website:

http://www.neo4j.org/download

Here we can find the latest version in exe format. So click on "Other Versions" link provided in this page

Step 2 - Here we can see all versions of neo4J Software in both exe or zip formats

Step 3 - Based on your OS configurations, click on 64 bit or 32 bit link provided in this page. We are going to click on "64" link to download 64-bit OS zip file



Step 4 - Extract this file to our required file system.

Steps to install NEO4J:

C:\>set NEO4J_HOME=C:\neo4j-community-3.5.15

C:\>set PATH=C:\neo4j-community-3.5.15\bin;%PATH%

C:\>echo %NEO4J_HOME%

C:\neo4j-community-3.5.15

C:\>echo %PATH%

C:\neo4j-community-3.5.15\bin;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Intel\OpenCL SDK\2.0\bin\x86;C:\Program Files\Intel\WiFi\bin\;C:\Program Files\Common Files\Intel\WirelessCommon\;C:\Program Files\Microsoft SQL Server\90\Tools\binn\;C:\Program Files\Microsoft Visual Studio 8\Common7\IDE\PrivateAssemblies\;C:\Program Files\Microsoft SQL Server\90\DTS\Binn\;C:\Program Files\Microsoft SQL Server\90\

Tools\Binn\VSShell\Common7\IDE\;C:\OpenCV2.4.9\opencv\build\x86\vc10\bin;C:\Open CV2.4.9\opencv\build\common\tbb\ia32\vc10;C:\Program Files\MATLAB\R2013a\runtime \win32;C:\Program Files\MATLAB\R2013a\bin;

C:\>neo4j.bat

Usage: neo4j {console | start | stop | restart | status | install-service | uninstall-service | update-service } < -Verbose >

C:\>neo4j.bat start

Neo4j service started

Type in browser:

http://localhost:7474

Summary:

`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

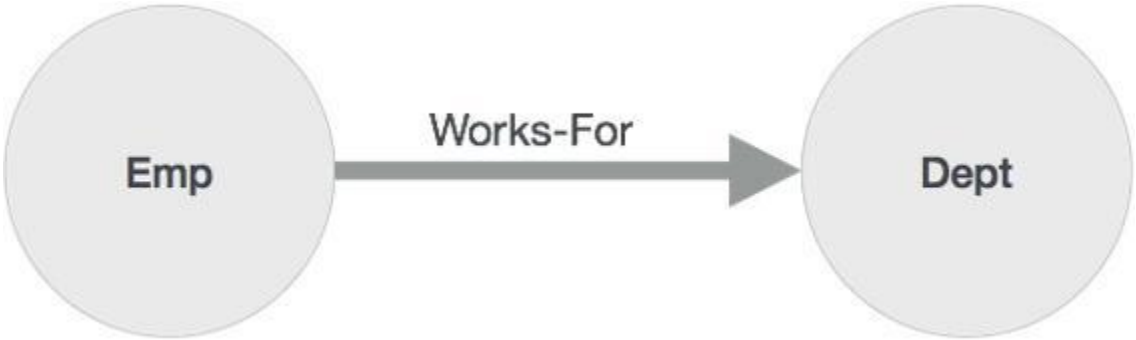
Property is a key-value pair to describe Graph Nodes and Relationships.

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 –

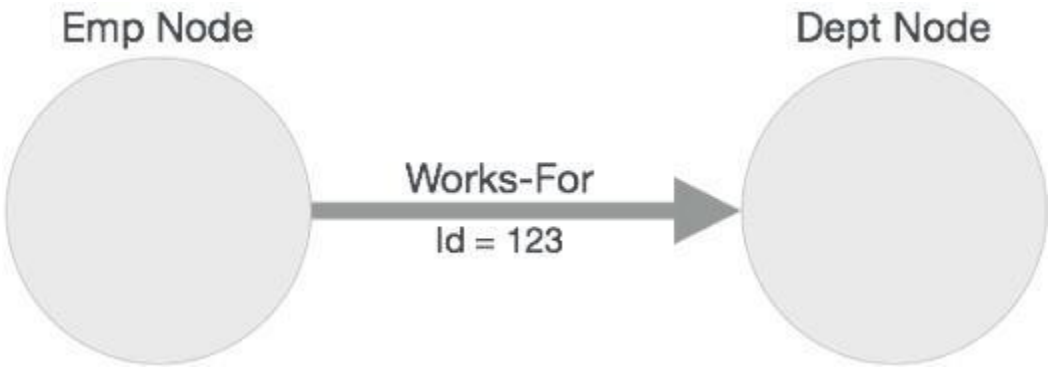
Emp WORKS_FOR Dept

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

Labels associate 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.

CQL

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.

Create a Single Node

To create a single node in Neo4j, specify the name of the node along with CREATE statement.

Syntax:

1. **CREATE** (node_name);

Verification

Execute the following code to verify the creation of the node type:

1. MATCH (n) **RETURN** n

Create Multiple Nodes

To create multiple nodes in Neo4j, use CREATE statement with the name of nodes separated by a comma.

Syntax:

1. **CREATE** (node1),(node2), (node1),???..

Create a node with a label

In Neo4j, a label is used to classify the nodes using labels. CREATE statement is used to create a label for a node in Neo4j.

Syntax:

1. **CREATE** (node:label)

Create a Node with Multiple Labels

To create multiple labels with a single node, you have to specify the labels for the node by separating them with a colon " : ".

Syntax:

1. **CREATE** (node:label1:label2:. . . . labeln)

Create Node with Properties

In Neo4j, properties are the key-value pairs which are used by nodes to store data. CREATE statement is used to create node with properties, you just have to specify these properties separated by commas within the curly braces "{ }".

Syntax:

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

Example:

Let's create a node "Prachi", having the following properties:

1. **CREATE** (Prachi:Faculty {**name**: " Prachi Shah", subject: "NoSQL", Department: "IT"})

Returning the created node

MATCH (n) RETURN n command is used to view the created nodes. This query returns all the existing nodes in the database.

But if you want to return the newly created node use the RETURN command with CREATE command:

Syntax:

1. **CREATE** (Node:Label{properties. . . . }) **RETURN** Node

Creating Relationship

While creating a relationship, a relationship should be specified within square braces "[ ]", depending on the direction of the relationship it is placed between hyphen " - " and arrow " > " as shown in the following syntax.

Syntax:

1. **CREATE** (node1)-[:RelationshipType]->(node2)

Example

Let's create two nodes "Raul" and "It" first and then specify the relationship between them.

1. **CREATE** (Raul:player{**name**: "Raul Vinci", YOB: 1973, POB: "Milan"})
2. **CREATE** (It:Country {**name**: "Italy"})
3. **RETURN** Raul, It

Now create a relationship "PLAYER_OF between these two nodes as...

1. **CREATE** (Raul)-[r:PLAYER_OF]->(It)
2. **RETURN** Raul, It

Create a Relationship between existing Nodes

MATCH statement is used to create relationships between the existing Nodes.

Syntax:

1. MATCH (a:LabeofNode1), (b:LabeofNode2)

   WHERE a.**name** = "nameofnode1" AND b.**name** = " nameofnode2"

   CREATE (a)-[: Relation]->(b)

   RETURN a,b

Neo4j Delete a Node

In Neo4j, DELETE statement is always used with MATCH statement to delete whatever data is matched. The DELETE command is used in the same place we used the RETURN clause in our previous examples.

Example

1. MATCH (Kohli:person {**Name**: "Virat Kohli"}) **DELETE** Kohli

Delete All Nodes

To delete all nodes from the database, don?t use any filter criteria.

1. MATCH (n) **DELETE** n

Delete Multiple Nodes

You can delete multiple nodes by using MATCH and DELETE commands in a single statement. You just have to put the different nodes separated by a column.

1. MATCH (a:Student {**Name**: "Chris Grey"}), (b:Employee {**Name**: "Mark Twin"})

DELETE a,b

Neo4j Delete a Relationship

Deleting relationships is as simple as deleting nodes. Use the MATCH statement to match the relationships you want to delete. You can delete one or many relationships or all relationships by using one statement.

Example:

Delete the relationship named "PLAYER_OF" from the database:

1. MATCH (Raul)-[r:PLAYER_OF]->(It)

   DELETE r

EXAMPLE:

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

CREATE (Ind:Country {name: "India"})

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

RETURN Dhawan, Ind
```

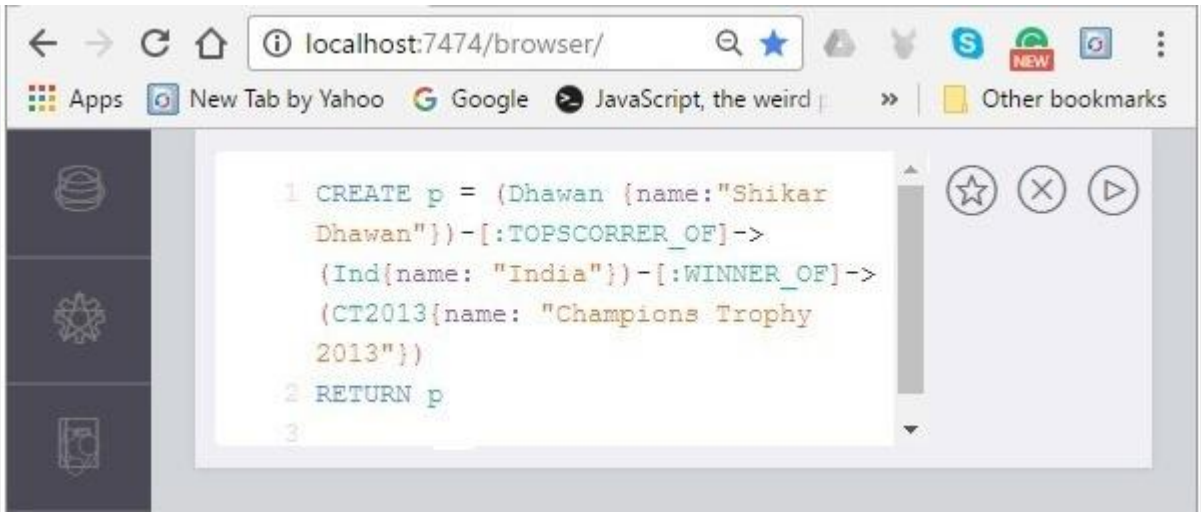
```
CREATE (Dhoni:player {name: "MahendraSingh Dhoni", YOB: 1981, POB: "Ranchi"})
```

```
Creating relation on existing nodes:

MATCH (a:player), (b:Country) WHERE a.name = "Shikar Dhawan" AND b.name = "India"

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

RETURN a,b
```




```
MERGE (Jadeja:player) RETURN Jadeja

MERGE (CT2013:Tournament{name: "ICC Champions Trophy 2013"})

RETURN CT2013, labels(CT2013)

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

RETURN Jadeja
```

Add a property

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

SET Dhawan.highestscore = 187

RETURN Dhawan
```

Remove a property

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

SET Jadeja.POB = NULL

RETURN Jadeja

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

```
REMOVE Dhoni.POB

RETURN Dhoni
```

Remove a label

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

REMOVE Dhoni:player

RETURN Dhoni
```

Match example:

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

RETURN n.name
```

THE COMPLETE DATASET:

```
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)
```

```
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"})
```

Where example:

```
MATCH (player)

WHERE player.country = "India" AND player.runs >=175

RETURN player
```

```
MATCH (n)

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

RETURN n
```

COUNT example:

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

RETURN n, count(*)
```

Order by:

```
MATCH (n)

RETURN n.name, n.runs

ORDER BY n.runs DESC
```

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

LIMIT:

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

SKIP:

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