

A **graph database** is a database designed to treat the relationships between data as equally important to the data itself. It is intended to hold data without constricting it to a pre-defined model. **Neo4j is an open source NoSQL Graph Database.** It follows the **Property Graph Model (PGM)** to store and manage its data. Some of the key features of PGM are as follows:


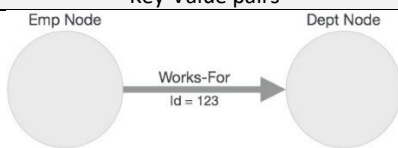
- PGM represents data in Nodes (also called as Vertices), Relationships (also called as Edges) and Properties.
- Edges connect the vertices.
- Vertices and edges contain properties. **Properties are key-value pairs.**
- Edges have directions: Unidirectional and Bidirectional
- Each edge contains "Start Node" or "From Node" and "To Node" or "End Node"

Neo4j is written in Java Language. Unlike traditional databases, which arrange data in rows, columns and tables, Neo4j has a flexible structure defined by stored relationships between data records. With Neo4j, each data record, or node, stores direct pointers to all the nodes it's connected to. Because Neo4j is designed around this simple, yet powerful optimisation, it performs queries with complex connections orders of magnitude faster, and with more depth, than other databases. Table 1 compares relational databases and graph databases. Neo4j supports ACID (Atomicity, Consistency, Isolation, and Durability) rules. Table 2 presents the building blocks of the graph database.

Table 1: Comparing RDBMS and Graph DB Terminology.

RDBMS	Graph Database
Tables	Graphs
Rows	Nodes
Columns	Properties
Relationships	Edges
Data	Values associated with Nodes and Edges
Query (Joins)	Graph Traversal

Table 2: The building blocks of the Neo4j graph database are the nodes (vertices), relationships (edges), properties, labels and data browser.

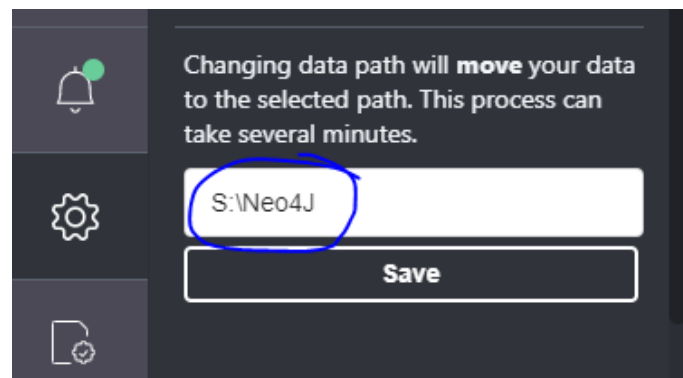
Nodes/Vertices with properties as Key-Value pairs	Relationships/Edges with properties as Key-Value pairs
 <p>Employee Node</p>	 <p>Emp and Dept are two different nodes. "WORKS_FOR" is a relationship between Emp and Dept nodes. [Emp WORKS_FOR Dept] has one property as key-value pair and which represents the id of this relationship (id=123).</p>
Labels	Neo4J Data Browser
<p>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. There are two labels in the above diagram – "Emp" and "Dept". Relationship between those two nodes also has a Label: "WORKS_FOR".</p>	<p>Neo4j Data Browser is used to execute CQL commands and view the output. It interacts with Neo4j Database Server, retrieves and displays the results as a graph or as a list (grid view). Data can be exported in CSV file format or in JSON file format.</p>

Cypher is Neo4j's graph-optimised query language that allows users to store and retrieve data from the graph database. It is a declarative pattern-matching language. With Neo4j, *connections between data are stored – not computed at query time* – and Cypher takes advantage of these stored

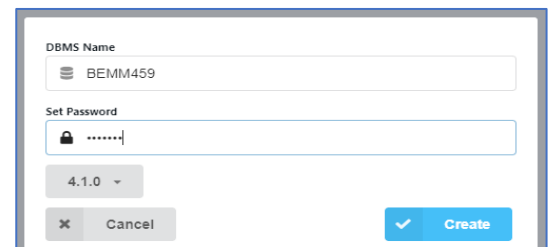
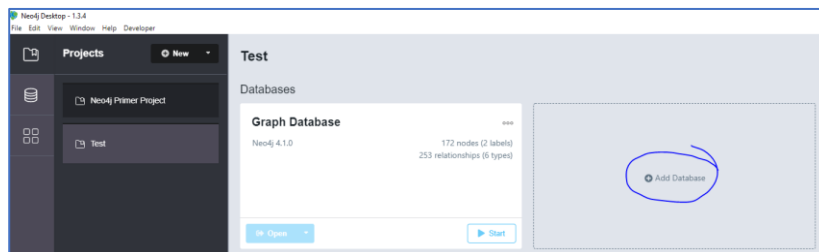
connections. When trying to find patterns or insights within data, Cypher queries are often much simpler and easier to write than large SQL JOINS. Since Neo4j doesn't have tables, there are no JOINS to worry about. Neo4j provides a built-in **Neo4j Data Browser web application**. Using this, you can create and query your graph data.

1. Connecting to Neo4J database using the browser

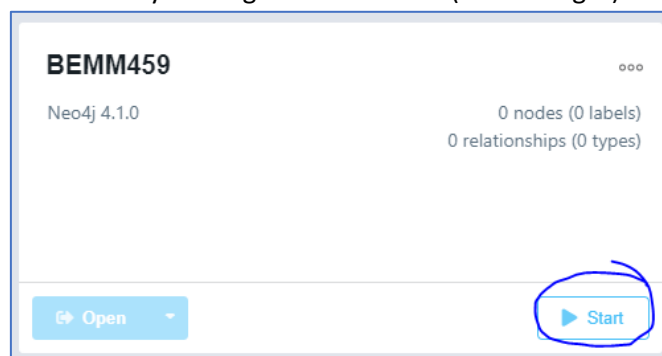
- **Start the server:** In the Exeter WVD environment, go to [Windows Start] and type "Neo4J" and open the "Neo4J Desktop" app.
- **Ensure that the database is saved in persistent storage (S:\) – first time only:** Create folder "Neo4J" directly under S:\ drive. Next, in Neo4J desktop app click [File -> Settings] and check the path variable (see screenshot below). This should be S:\Neo4J.



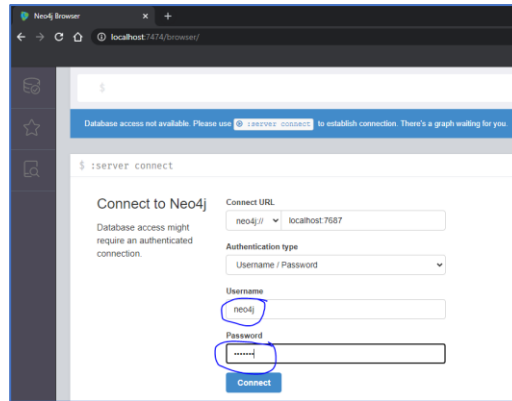
- **Create a new database – first time only:** Click on the [Add Database] and then [Create a Local Database] button. Enter a name for the database and a password (in the screenshot below - right, the database name is "BEMM459" and the password is also "BEMM459").
 - **Note:** you need to remember the password. The default username is neo4j.



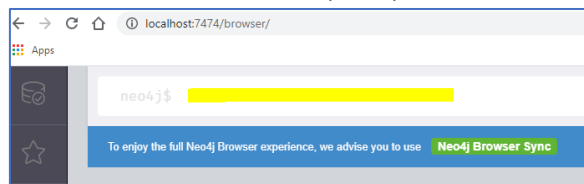
- Start the "BEMM459" database by clicking on start button (bottom right).



- Once the database has started open the **Neo4J browser interface** by entering <http://localhost:7474/> in your browser (Chrome, IE, Firefox) address bar. Next, enter the username and password for the "BEMM459" database.
 - **Note:** The default username is **neo4j**. The password is **BEMM459**.



- Neo4j Data Browser is used to execute CQL commands and view the output; the prompt is **neo4j\$**. We execute all CQL commands at dollar prompt: "\$".



2. Creating Nodes

2.1 Create a single node (-- examples in green)

Neo4j\$ CREATE (node_name);

CREATE (station)

Notes:

- Semicolon is optional.
- <id> is added by Neo4j and starts with 0.
- To verify the creation of the node, execute the following query: **MATCH (n) RETURN n** (this query returns all the existing nodes in the database)

2.2 Create a multiple nodes

Neo4j\$ CREATE (node_name1), (node_name1)

CREATE (train), (passenger)

2.3 Create a node with a label

Neo4j\$ CREATE (node_name:label)

CREATE (exetercentral:station)

2.4 Create a node with multiple labels

Neo4j\$ CREATE (node_name:label1:label2:...:labelN)

CREATE (st_thomas:station:cyclestore)

2.5 Create a node with properties

Neo4j\$ CREATE (node:label { key1: value, key2: value, })

CREATE (exe_central:station:cyclestore {name:"Exeter Central Station", address:" Exeter EX4 3SB ", platforms:3})

2.6 Create and return a node

Neo4j\$ CREATE (Node:Label{properties. . . }) RETURN Node

```
CREATE (stu1:student {name:"Student One", address:"1 Penn", age:31}) RETURN stu1
```

3. Creating Relationships

3.1 Creating relationships between newly created nodes using CREATE

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

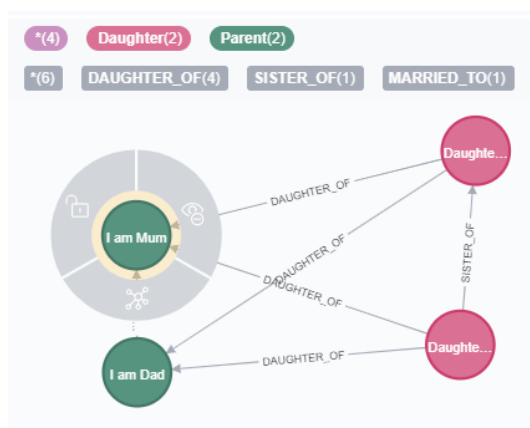
Neo4j\$ CREATE (node1)-[:RelationshipType]->(node2)

Here we create nodes with two labels “module” and “students” and create a link between them. Finally we display all the nodes.

```
CREATE (M__4591:module {module_id: "Module_BEMM459", name: "Neo4J2", teacher: "Nav"})
CREATE (stu2:student {name:"Student TWO", address:"34 Penn", age:28})
CREATE (stu3:student {name:"Student THREE", address:"162 East Close", age:32})
CREATE (stu4:student {name:"Student FOUR", address:"94 Bonhey Road", age:25})
CREATE (stu2) - [r:ENROLLED_IN] -> (M__4591)
CREATE (stu3) - [r1:ENROLLED_IN] -> (M__4591)
CREATE (stu4) - [r2:ENROLLED_IN] -> (M__4591)
RETURN stu2, stu3, stu4, M__4591, r, r1, r2
```

Example of multiple relationships between nodes with labels.

```
CREATE (D1:Daughter{name: "Daughter_1", YOB: 2009, POB: "Cardiff", CTRY: "Wales" })
CREATE (D2:Daughter{name: "Daughter_2", YOB: 2012, POB: "Nottingham", CTRY: "England" })
CREATE (Dad:Parent{name: "I am Dad", YOB: 1970, POB: "Edinburgh", CTRY: "Scotland" })
CREATE (Mum:Parent{name: "I am Mum", YOB: 1968, POB: "Belfast", CTRY: "Northern Ireland" })
CREATE (D1)-[r:SISTER_OF]->(D2)
CREATE (Dad)-[r1:MARRIED_TO]->(Mum)
CREATE (D1)-[r2:DAUGHTER_OF]->(Mum)
CREATE (D1)-[r3:DAUGHTER_OF]->(Dad)
CREATE (D2)-[r4:DAUGHTER_OF]->(Mum)
CREATE (D2)-[r5:DAUGHTER_OF]->(Dad)
RETURN D1, D2, Mum, Dad, r1, r2, r3, r4, r5
```



*Displaying a meta-graph (select “star” – left side of the Neo4j data browser – and then select “common procedures” → **Show meta-graph.***

3.1.2 Display **meta-graph** (Returns a virtual graph that represents the labels and relationship-types available in your database and how they are connected)

Neo4j\$ CALL db.schema.visualization()



3.2 Creating relationships with a **label** between existing nodes using **MATCH** and **CREATE**

```
Neo4j$ MATCH (a:LabeofNode1), (b:LabeofNode2)
WHERE a.name = "nameofnode1" AND b.name = " nameofnode2"
CREATE (a)-[: Relation]->(b)
RETURN a,b
```

First **CREATE** one new node..

```
CREATE (stu5:student {name:"Student FIVE", address:"34 Penn", age:25})
```

.. then **CREATE** a relationship with existing node (label:module) using the **MATCH** clause and **RETURN** results

```
MATCH (a:student), (b:module) WHERE a.name = "Student FIVE" AND b.module_id = "Module_BEMM459"
CREATE (a)-[r:ENROLLED_IN]->(b)
RETURN a, b, r
```

3.3 Creating relationship with **label** and **properties** between existing nodes using **MATCH** and **CREATE**

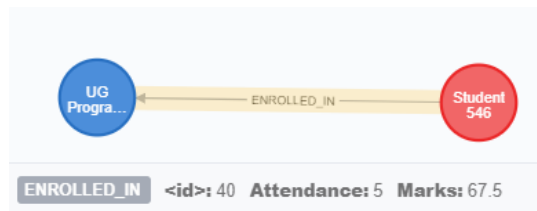
Neo4j\$ CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]-> (node2)

First **CREATE** one new node..

```
CREATE (stu7:student {name:"Student SEVEN", address:"UOE East Park Lane", age:29})
```

.. then **CREATE** a relationship (**with label and properties**) with existing node (label:module) using the **MATCH** clause and **RETURN** the results (note: the figure below is only a representative graph).

```
MATCH (a:student), (b:module) WHERE a.name = "Student SEVEN" AND b.module_id = "Module_BEMM459"
CREATE (a)-[r:ENROLLED_IN {Attendance:5, Marks:67.5}]->(b)
RETURN a, b, r
```

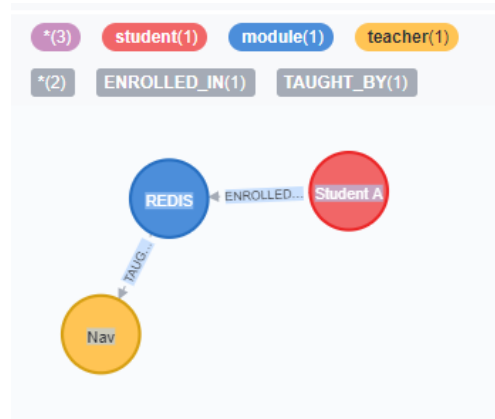


3.4 Creating a complete **path** by connecting multiple nodes through relationships

```
Neo4j$ CREATE p = (Node1 {properties})-[:Relationship_Type]->
(Node2 {properties})[:Relationship_Type]->(Node3 {properties})
RETURN p
```

Create a path between three new nodes (labels: student, module, teacher) with two relations (note: the figure below is only a representative graph).

```
CREATE path = (stu8:student {name:"Student EIGHT", address:"Holland Hall", age:32})-[r:ENROLLED_IN]->
(M__5000:module {module_id: "BEMM_5000", name: " Business Analytics Dissertations ", term: "Term 3"})-
[r1:TAUGHT_BY]->(tch1:teacher {module_id: "Teacher AA", teacher_name: "Nav", teacher_phone: "Not Known"})
RETURN path
```



4. Update Nodes and Relationships – new properties, new labels, for each

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

4.1 Set new properties in an existing node using SET

SET to be used with MATCH.

Single property

```
Neo4j$ MATCH (node:label{properties . . . . . })
SET node.property = value
RETURN node
```

Multiple properties

```
Neo4j$ MATCH (node:label {properties})
SET node.property1 = value, node.property2 = value
RETURN node
```

Creating a new property called phone_number and assigning a value to an existing node.

```
MATCH (stu7:student{name: "Student SEVEN"})
SET stu7.phone_number = "763456789"
RETURN stu7
```

Creating multiple properties and assigning a value to an existing node.

```
MATCH (stu7:student{name: "Student SEVEN"})
SET stu7.highest_qualification = "BSc in Analytics", stu7.email = "Student7.SEVEN@UOE.com"
RETURN stu7
```

4.2 Removing a property in an existing node using SET

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

```
Neo4j$ MATCH (node:label {properties})
SET node.property = NULL
RETURN node
```

```

MATCH (stu7:student{name: "Student SEVEN"})
SET stu7. phone_number = NULL
RETURN stu7

```

4.3 Set a label on a node

SET to be used with MATCH.

Single label

```

Neo4j$ MATCH (n {properties . . . . . })
SET n :label
RETURN n

```

Multiple labels

```

Neo4j$ MATCH (n {properties . . . . . })
SET n :label1:label2
RETURN n

```

Creating a new label for an existing node.

```

MATCH (stu7:student{name: "Student SEVEN"})
SET stu7: JudoClub
RETURN stu7

```

Creating new labels for an existing node.

```

MATCH (stu7:student{name: "Student SEVEN"})
SET stu7: StudentGuild:SwimmingTeam
RETURN stu7

```

4.4 Set a label on a node

The FOREACH clause is used to update data within a list, e.g., the different nodes in a path.

```

Neo4j$ 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)

```

First create a path (no bearing on the code below)

```

CREATE path = (stu9:student {name:"Student NINE", address:"Holland Hall", age:32})-[r:ENROLLED_IN]->
  (M__9999:module {module_id: "BEMM_9999", name: "Neo4J", term: "Term 3"})-[r1:TAUGHT_BY]-
  >(tch2:teacher {module_id: "Teacher BB", teacher_name: "Nav Again", teacher_phone: "Not Known"})
RETURN path

```

*Add a new property (phone_number_to_call) to all nodes along the **path** (stu9 -> tch2) using the **FOREACH** clause.*

```

MATCH path = (x)-[*]->(y)
WHERE x.name = "Student NINE" AND y. teacher_name = "Nav Again"
FOREACH (n IN nodes(path) | SET n.Phone_number_to_call = "76453213")
RETURN path

```

5. Removing properties and labels and deleting nodes and relationships

The main difference between DELETE and REMOVE commands are:

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

REMOVE and DELETE to be used with MATCH.

5.1 Removing a property

```
Neo4j$ MATCH (node:label{properties . . . . . })  
REMOVE node.property  
RETURN node
```

First return the node to check properties (no bearing on the code below)

```
MATCH (x:student{name: "Student NINE"})  
RETURN x
```

Next, remove property (age) using the REMOVE clause

```
MATCH (x:student{name: "Student NINE"})  
REMOVE x.age  
RETURN x
```

5.2 Removing a label from a node

```
Neo4j$ MATCH (node:label {properties . . . . . })  
REMOVE node:label  
RETURN node
```

First return the node to check labels (no bearing on the code below)

```
MATCH (x:student{name: "Student NINE"})  
RETURN x
```

Next, remove label (student) using the REMOVE clause

```
MATCH (x:student{name: "Student NINE"})  
REMOVE x:student  
RETURN x
```

5.3 Deleting a particular node

```
Neo4j$ MATCH (node:label {properties . . . . . })  
DETACH DELETE node
```

Deleting a node

```
MATCH (x:student{name: "Student NINE"})  
DETACH DELETE x
```

5.4 Deleting all nodes and relationships

Deleting all nodes and relationships

```
Neo4j$ MATCH (n) DETACH DELETE n
```


Neo4J (Extended tutorial)

6. Read (MATCH and WHERE Clause)

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

6.1 Return all nodes and relationships

```
Neo4j$ MATCH (n) RETURN n
```

6.2 Return all nodes under a specific label

```
Neo4j$ MATCH (node:label)
RETURN node
```

```
MATCH (n:student)
RETURN n
```

6.3 Retrieve nodes based on a relationship using the MATCH clause

```
Neo4j$ MATCH (node:label)<-[: Relationship]-(n)
RETURN n
```

```
MATCH (r:module {module_id: "Module_BEMM459"}) <-[: ENROLLED_IN]-(n)
RETURN n.name
```

Note: results appear in table view.

6.4 Use of WHERE clause in MATCH command to filter results of a MATCH query

```
Neo4j$ MATCH (label)
WHERE label.x = "property"
RETURN label
```

```
MATCH (student)
WHERE student.age = 25
RETURN student
```

6.5 Use of WHERE with multiple conditions (AND) to filter results of a MATCH query

```
Neo4j$ MATCH (label)
WHERE label.x = "property" AND label.y = "property"
RETURN label
```

```
MATCH (student)
WHERE student.age = 25 AND student.address = "34 Penn"
RETURN student
```

Acknowledgement:

- <https://neo4j.com/blog/>
- <https://neo4j.com/docs/>
- <https://www.tutorialspoint.com/>
- Sullivan, D.. (2015). *NoSQL for mere mortals*. Addison-Wesley Professional.