CSE 4410

DATABASE MANAGEMENT SYSTEMS II LAB

Notes On: Neo4J

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
ISLAMIC UNIVERSITY OF TECHNOLOGY

APRIL 4, 2024

1 Getting Started with Neo4J

Neo4j is a graph database. A graph database, instead of having rows and columns has nodes edges, and properties. It is more suitable for certain big data and analytics applications than row and column databases or document databases for many use cases.
It is used to represent relationships. The most common example of that is the Facebook Friend relationship as well as the Like relationship. You can see some of that in the graphic below from Neo4j.

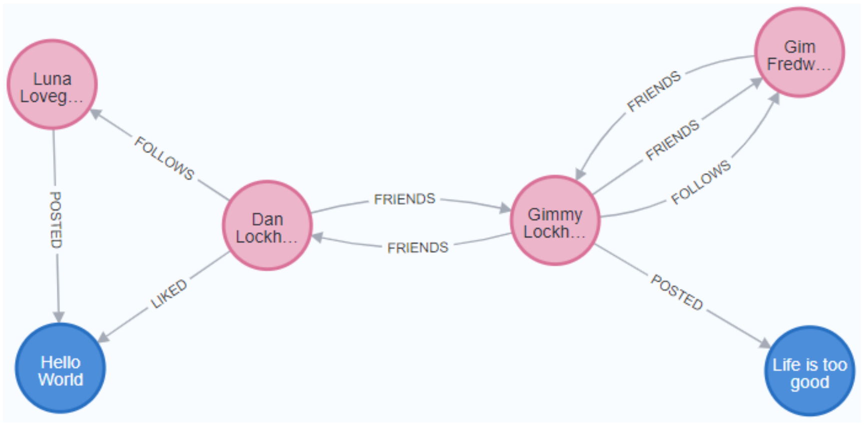


Figure 1: Graph

2 Environment Setup

Neo4j Desktop is an installable application to help you work with Neo4j, whether you are just getting started or have prior Neo4j experience. For a new user, it is designed to help you learn and experiment with Neo4j locally. A Developer edition license of the Neo4j graph database is included with Neo4j Desktop. The Developer edition offers all the capabilities and features of Neo4j Enterprise Edition, for development use by a person on a single machine except for multi-machine features.

1. To download and deploy Neo4j on your own server, go to `https://neo4j.com/deployment-center/` and download the suitable latest Neo4j Desktop for your device.

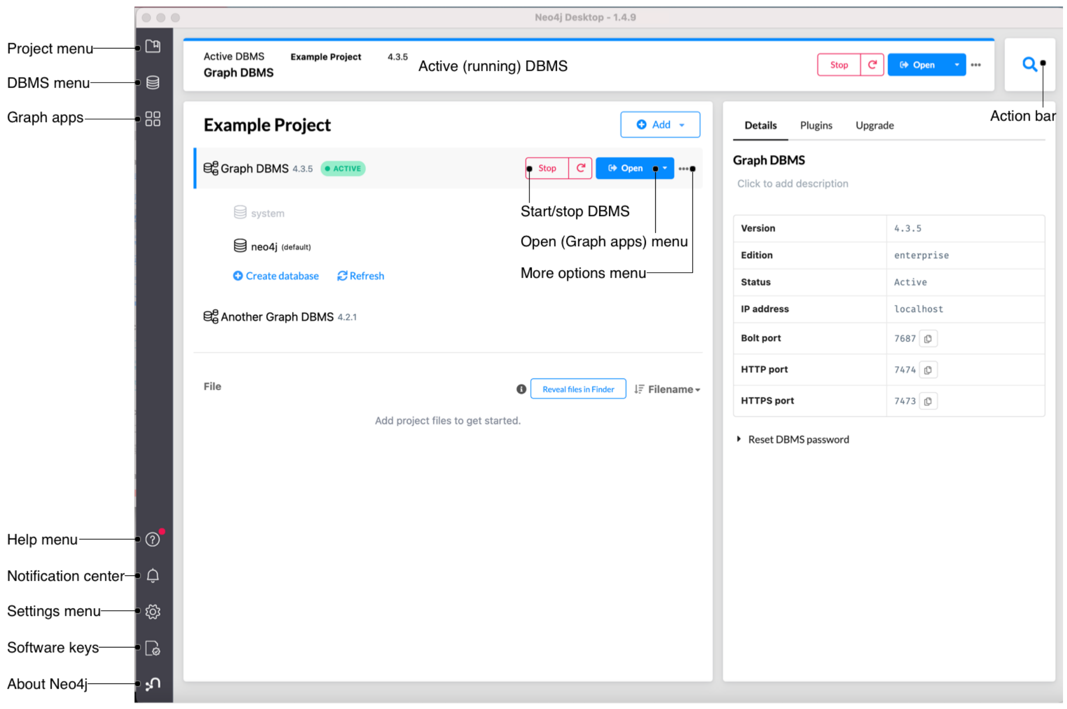
2. Register an account for Neo4J Desktop. It will lead you to another page where you will be provided with an "Activation Key".

3. Copy the activation at the top of the page and paste it in the "Activation Key" box in the Neo4j Desktop app.

4. Alternatively, you can also generate a key from within the app by filling out the form on the right side of the app screen.

2.1 Getting known with the interface

1. **Projects**
   In Neo4J Desktop, a project represents a development folder on disk. You can create local database management systems (DBMSs), connect to remote DBMSs, and add files to your project. Neo4j Desktop allows you to manage multiple Projects. However, you can have only one active DBMS or remote connection at a time.



Version	4.3.5
Edition	enterprise
Status	Active
IP address	localhost
Bolt port	7687
HTTP port	7474
HTTPS port	7473

Figure 2: Neo4J Desktop Dashboard

2. **DBMS**
   A DBMS is a Neo4j server instance that contains a minimum of the system database and a default database. Existing DBMSs can also be accessed through the DBMS menu.

3. **Graph Apps**
   There are several ways to interact with the graph. One way is using graph applications and Desktop comes bundled with two such applications, Neo4j Browser and Bloom. Neo4j Browser and Bloom are used to visualize and query the graph, but other applications offer import tools for relational databases, monitoring tools, and query log analyzers.

2.2 Creating project and connection set up

1. To start working with Neo4J, first one has to create a project. To do so, go to `Project menu` ->click on `(+)new`->select `Create project`. Then click on the just-created project on the sidebar to rename the project and see the details.

2. Now, click on the `(+)`add drop-down and select `Local DBMS` to work locally. Here you can change the name of the `Graph DBMS` and have to provide a password of a minimum of 8 length. Upon creation, the default database is called neo4j, but you can rename it or create a new database as the default.

3. Finally, to activate a connection hover on `Graph DBMS`. As a result, a `Start` button will appear, and click on it. Make sure you don't have any other DBMS running at that time. To deactivate a connection you will find a `Stop` in place of `Start` button for an active DBMS.

3 Basics

3.1 Create

- Creating a node

```
> create (variable:Label{property:'value',...,property:'value'})
```

```
create (u:USER{name:'Gim Fredwick', age:35, country: 'UK'})
```

- Creating multiple nodes

```
> create (variable: Label{property:'value'}), (variable: Label{
property:'value'})
```

```
create (u1:USER{name:'Gimmy Lockheart', age:33, country: 'SW'}),
(u2:USER{name:'Dan Lockheart', age:29, country: 'DM'})
```

- Creating a single node with multiple Labels

```
> create (variable:Label:...:Label{property:'value'})
```

```
create (u1:USER:ADMIN{name:'Eddi Flitch', age:33, country: 'NW'})
```

- Create a relationship between nodes

```
> match (variable_1:Label_1), (variable_2:Label_2)
where variable_1.property='value' and variable_2.property='value'
create (variable_1)-
    [variable_3:Relationship_name {property:'value'}]->
    (variable_2)
```

```
match (u1:USER), (u2:USER)
where u1.name='Gimmy Lockheart' and u2.name='Gim Fredwick'
create (u1)-[f:FOLLOWS{date:date('2021-02-21')}]->(u2)
```

Alternatively,

```
match (u1:USER{name:'Gimmy Lockheart'}),
(u2:USER{name:'Gim Fredwick'})
create (u1)-[f:FOLLOWS{date:date('2021-02-21')}]->(u2)
```

- Create nodes with a relationship

```
> create (variable_1:Label_1{property:'value'}) -
    [variable_3:Relationship_name {property:'value'}] ->
    (variable_2:Label_2{property:'value'})
```

```
create (u:USER{name:'Luna Lovegood', age:31, country: 'ST'})-
[pd:POSTED{date:date('2021-02-11')}]->
(p:POST{content:'Hello World'})
```

3.2 Delete

Deletion in Neo4J, is almost similar to SQL language. Because it also follows the same rule as SQL. One has to delete the relationships that are adjacent to a node in order to delete that node.

- To delete all the nodes (when there is no relationship)

```
> match (n) delete (n)
```

```
match (n) delete n
```

- To delete the full graph (with relation)

```
> match (n) detach delete (n)
```

```
match (n) detach delete n
```

- To delete a certain node

```
> match (variable: {property: "value"}) delete variable
```

```
match (u:USER{country:'NW'}) delete u
```

- To delete a certain node with an adjacent relationship

```
> match (variable: {property: "value"}) detach delete variable
```

```
match (u:USER{name:'Gimmy Lockheart'}) detach delete u
```

- To delete a relationship

```
> match (variable_1: {property: "value"}) -
    [variable_3:Relationship_name] ->
    (variable_2:{property: "value"})
    delete variable_3
```

```
match (u1)-[f:FOLLOWS]->(u2) delete f
```

3.3 Update

- To add/update a property to a certain node

```
> match (variable {property: "value"})
set variable.property='value'
```

```
match (u{name:'Luna Lovegood'}) set u.age=28
```

- To add/update multiple property to a certain node

```
> match (variable {property: "value"})
set variable.property='value', ...,variable.property='value'
```

```
match (u{name:'Luna Lovegood'})
set u.age=29, u.hobby='Studying mythical creatures'
```

- To add a label to a certain node

```
> match (variable {property: "value"})
set variable:Label_name
```

```
match (u{name:'Luna Lovegood'}) set u:Content_Creator
```

- To add/update a property of a certain relationship

```
> match (variable1 {property: "value"}) -
[variable_3:Relationship_name] ->
(variable2: {property: "value"})
set variable_3.property='value'
```

```
match (u)-[pd:POSTED]->(p:POST) where id(p)=3 set pd.from='JP'
```

- To remove a property from a certain node

```
> match (variable {property: "value"})
remove variable.property
```

```
match (p) where id(p)=5 remove p.language'
```

- To remove a label from a certain node

```
> match (variable Lable {property: "value"})
remove variable:Lable
```

```
match (u{name:'Luna Lovegood'}) remove u:Content_Creator
```

- To remove a property from a certain relationship

```
> match (variable1 {property: "value"}) -
[variable_3:Relationship_name] ->
(variable2: {property: "value"})
remove variable_3.property='value'
```

```
match (u)-[pd:POSTED]->(p:POST) where id(p)=3 remove pd.from
```

3.4 Simple Query

- To see the whole graph

```
> match (variable) return variable
```

```
match (n) return n
```

- To see the nodes of a specific label

```
> match (variable: Label) return variable
```

```
match (n:POST) return n
```

- To see the properties of a specific label

```
> match (variable: Label) return variable.property
```

```
match (n:USER) return n.age
```

- To show node based on property

```
> match (variable: Label)
    where variable.property='value'
    return variable
```

```
match (n:USER) where n.age>30 return n
```

Exception,

```
match (n:USER) where id(n)=2 return n
```

```
> match (variable: Label {property='value'})
    return variable
```

```
match (n:USER{age:29}) return n
```

- And/ or operator

```
> match (variable: Label)
    where variable.property='value' and variable.property='value'
    return variable
```

```
match (n:USER) where n.country='UK' or n.country='ST' return n
```

- Sorting the nodes

```
> match (variable: Label)
    [where variable.property='value']
    return variable
    order by variable.property [Desc}
```

```
match (n:USER) return n.name, n.age order by n.age
```

- Limiting the nodes

```
> match (variable: Label)
    [where variable.property='value']
    return variable
    limit value
```

```
match (n:USER) return n order by n.age limit 3
```

- Skipping node

```
> match (variable: Label)
    [where variable.property='value']
    return variable
    skip value
```

```
match (n:USER) return n order by n.age skip 3
```

- Showing nodes of multiple labels

```
> match (variable_1: Label_1), (variable_2: Label_2)
    return variable_1, variable_2
```

```
match (p:POST), (u:USER) return p, u
```

- Showing all nodes that have a certain relationship

```
> match (variable_1: Label_1) -
    [variable_3: Relationship_name] ->
    (variable_2: Label_2)
    return variable_1, variable_2
```

```
match (u1:USER)-[f:FRIENDS]->(u2:USER) return u1,u2
```

Alternatively,

```
match r=()-[f:FRIENDS]->() return r
```

- Nested search

```
> match (variable_1: Label_1{property:'value'}) -
    [variable_3: Relationship_name] ->
    (variable_2: Label_2)
        match (variable_2)-
            [variable_5: Relationship_name] ->
            (variable_4: Label_4)
            where variable_5.property='value'
            return variable_2
```

```
match (u1:USER{name:'Luna Lovegood'})-[pd:POSTED]->(p:POST)
match (u2:USER)-[l:LIKED]->(p)
return u2
```

- Aggregate functions

```
    > match (variable_1: Label_1) -
        [variable_3: Relationship_name] ->
        (variable_2: Label_2)
        return variable_1.property, count(variable_3)
```

```
match (u1:USER{name:'Gimmy Lockheart'})-[f:FRIENDS]->(u2:USER)
return u1.name, count(f)
```

```
match (u1:USER{name:'Gimmy Lockheart'})-[f:FRIENDS]->(u2:USER)
return u1.name, avg(u2.age)
```

```
match (u1:USER{name:'Gimmy Lockheart'})-[f:FRIENDS]->(u2:USER)
return u1.name, min(u2.age)
```