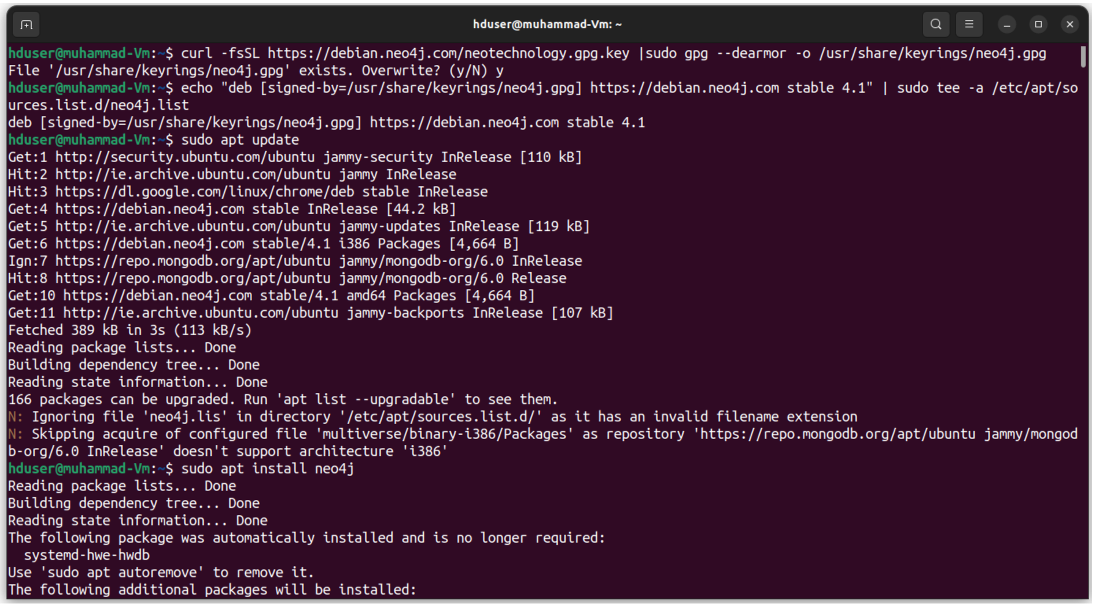
Tutorial 11
Graph Databases

1) **Neo4j** is a graph database that records relationships between data nodes, whereas traditional relational databases use rows and columns to store and structure data. Since each node stores references to all the other nodes that it is connected to, Neo4j can encode and query complex relationships with minimal overhead.

2) The official Ubuntu package repositories do not contain a copy of the Neo4j database engine. To install the upstream supported package from **Neo4j** you will need to add the **GPG key** from Neo4j to ensure package downloads are valid. Then you will add a new package source pointing to the **Neo4j** software repository, and finally install the package.

   To get started, download and pipe the output of the following curl command to the gpg --dearmor command. This step will convert the key into a format that apt can use to verify downloaded packages:

   ```
   $ curl -fsSL https://debian.neo4j.com/neotechnology.gpg.key |sudo gpg --
   dearmor -o /usr/share/keyrings/neo4j.gpg
   ```



```
hduser@muhammad-Vm:~$ curl -fsSL https://debian.neo4j.com/neotechnology.gpg.key |sudo gpg --dearmor -o /usr/share/keyrings/neo4j.gpg
File '/usr/share/keyrings/neo4j.gpg' exists. Overwrite? (y/N) y
hduser@muhammad-Vm:~$ echo "deb [signed-by=/usr/share/keyrings/neo4j.gpg] https://debian.neo4j.com stable 4.1" | sudo tee -a /etc/apt/so
urces.list.d/neo4j.list
deb [signed-by=/usr/share/keyrings/neo4j.gpg] https://debian.neo4j.com stable 4.1
hduser@muhammad-Vm:~$ sudo apt update
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Hit:2 http://ie.archive.ubuntu.com/ubuntu jammy InRelease
Hit:3 https://dl.google.com/linux/chrome/deb stable InRelease
Get:4 https://debian.neo4j.com stable InRelease [44.2 kB]
Get:5 http://ie.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:6 https://debian.neo4j.com stable/4.1 i386 Packages [4,664 B]
Ign:7 https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 InRelease
Hit:8 https://repo.mongodb.org/apt/ubuntu jammy/mongodb-org/6.0 Release
Get:10 https://debian.neo4j.com stable/4.1 amd64 Packages [4,664 B]
Get:11 http://ie.archive.ubuntu.com/ubuntu jammy-backports InRelease [107 kB]
Fetched 389 kB in 3s (113 kB/s)
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
166 packages can be upgraded. Run 'apt list --upgradable' to see them.
N: Ignoring file 'neo4j.lis' in directory '/etc/apt/sources.list.d/' as it has an invalid filename extension
N: Skipping acquire of configured file 'multiverse/binary-i386/Packages' as repository 'https://repo.mongodb.org/apt/ubuntu jammy/mongod
b-org/6.0 InRelease' doesn't support architecture 'i386'
hduser@muhammad-Vm:~$ sudo apt install neo4j
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following package was automatically installed and is no longer required:
  systemd-hwe-hwdb
Use 'sudo apt autoremove' to remove it.
The following additional packages will be installed:
```

   Next, add the Neo4j 4.1 repository to your system's APT sources:

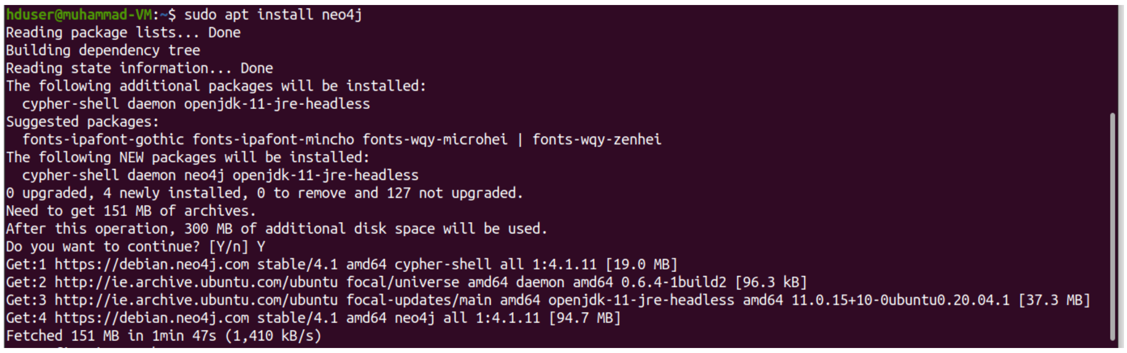
   ```
   $ echo "deb [signed-by=/usr/share/keyrings/neo4j.gpg]
   https://debian.neo4j.com stable 4.1" | sudo tee -a
   /etc/apt/sources.list.d/neo4j.list
   ```

   The [signed-by=/usr/share/keyrings/neo4j.gpg] portion of the file instructs apt to use the key that you downloaded to verify repository and file information for neo4j packages.

   The next step is to update your package lists, and then install the **Neo4j** package and all of its dependencies. This step will download and install a compatible Java package, so you can enter Y when the apt command prompts you to install all the dependencies:

   ```
   $sudo apt update
   ```

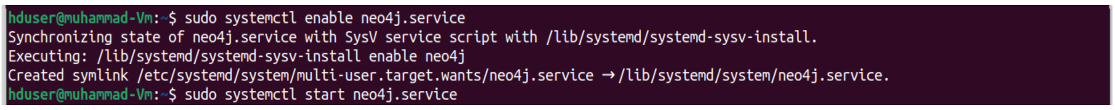
3) `$sudo apt install neo4j`



```
hduser@muhammad-VM:~$ sudo apt install neo4j
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cypher-shell daemon openjdk-11-jre-headless
Suggested packages:
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei
The following NEW packages will be installed:
  cypher-shell daemon neo4j openjdk-11-jre-headless
0 upgraded, 4 newly installed, 0 to remove and 127 not upgraded.
Need to get 151 MB of archives.
After this operation, 300 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://debian.neo4j.com stable/4.1 amd64 cypher-shell all 1:4.1.11 [19.0 MB]
Get:2 http://ie.archive.ubuntu.com/ubuntu focal/universe amd64 daemon amd64 0.6.4-1build2 [96.3 kB]
Get:3 http://ie.archive.ubuntu.com/ubuntu focal-updates/main amd64 openjdk-11-jre-headless amd64 11.0.15+10-0ubuntu0.20.04.1 [37.3 MB]
Get:4 https://debian.neo4j.com stable/4.1 amd64 neo4j all 1:4.1.11 [94.7 MB]
Fetched 151 MB in 1min 47s (1,410 kB/s)
```

Once the installation process is complete, **Neo4j** should be running. However, it is not set to start on a reboot of your system. So the last setup step is to enable it as a service and then start it:

```
$sudo systemctl enable neo4j.service
```



```
hduser@muhammad-Vm:~$ sudo systemctl enable neo4j.service
Synchronizing state of neo4j.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable neo4j
Created symlink /etc/systemd/system/multi-user.target.wants/neo4j.service → /lib/systemd/system/neo4j.service.
hduser@muhammad-Vm:~$ sudo systemctl start neo4j.service
```

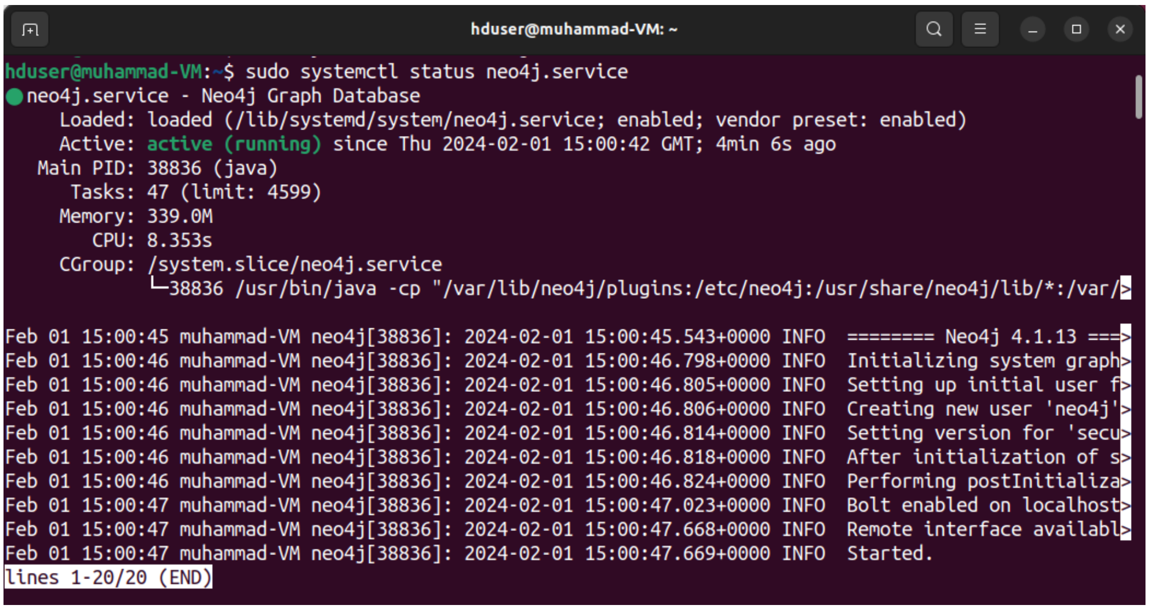
4) Now start the service if it is not already running:

```
$sudo systemctl start neo4j.service
```

After completing all of these steps, examine Neo4j's status using the systemctl command:

```
$sudo systemctl status neo4j.service
```

You should have output that is similar to the following:



```
hduser@muhammad-VM:~$ sudo systemctl status neo4j.service
● neo4j.service - Neo4j Graph Database
     Loaded: loaded (/lib/systemd/system/neo4j.service; enabled; vendor preset: enabled)
     Active: active (running) since Thu 2024-02-01 15:00:42 GMT; 4min 6s ago
   Main PID: 38836 (java)
      Tasks: 47 (limit: 4599)
     Memory: 339.0M
        CPU: 8.353s
     CGroup: /system.slice/neo4j.service
             └─38836 /usr/bin/java -cp "/var/lib/neo4j/plugins:/etc/neo4j:/usr/share/neo4j/lib/*:/var/>

Feb 01 15:00:45 muhammad-VM neo4j[38836]: 2024-02-01 15:00:45.543+0000 INFO  ======== Neo4j 4.1.13 ===>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.798+0000 INFO  Initializing system graph>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.805+0000 INFO  Setting up initial user f>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.806+0000 INFO  Creating new user 'neo4j'>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.814+0000 INFO  Setting version for 'secu>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.818+0000 INFO  After initialization of s>
Feb 01 15:00:46 muhammad-VM neo4j[38836]: 2024-02-01 15:00:46.824+0000 INFO  Performing postInitializa>
Feb 01 15:00:47 muhammad-VM neo4j[38836]: 2024-02-01 15:00:47.023+0000 INFO  Bolt enabled on localhost>
Feb 01 15:00:47 muhammad-VM neo4j[38836]: 2024-02-01 15:00:47.668+0000 INFO  Remote interface availabl>
Feb 01 15:00:47 muhammad-VM neo4j[38836]: 2024-02-01 15:00:47.669+0000 INFO  Started.
lines 1-20/20 (END)
```

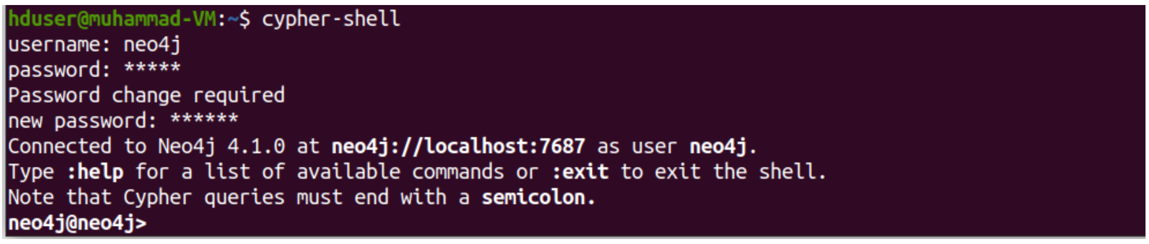
There will be other verbose lines of output, but the important things to note are the highlighted enabled and running lines. Once you have Neo4j installed and running, you can move on to the next set of steps, which will guide you through connecting to Neo4j, configuring credentials, and inserting nodes into the database. Press q to stop this.

5) **Connecting to and Configuring Neo4j**

Now that you have Neo4j installed and configured to run after any reboot, you can test connecting to the database, and configure administrator credentials. Open a new terminal and use the following commands

To interact with Neo4j on the command line, use the cypher-shell utility. Invoke the utility like this by opening a new terminal.

`$cypher-shell`



```
hduser@muhammad-VM:~$ cypher-shell
username: neo4j
password: *****
Password change required
new password: ******
Connected to Neo4j 4.1.0 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j>
```

When you first invoke the shell, you will login using the default administrative **neo4j** user and **neo4j** password combination. Once you are authenticated, Neo4j will prompt you to change the administrator password

In this example, the highlighted ******************* is the masked version of the new password. Choose your own strong and memorable password and be sure to record it somewhere safe. (Suggestion: used password for Ubuntu machine as hadoop as we did in the previous tutorials if you are OK). Once you set the password you will be connected to the interactive **neo4j@neo4j>** prompt where you can interact with Neo4j databases by inserting and querying nodes.

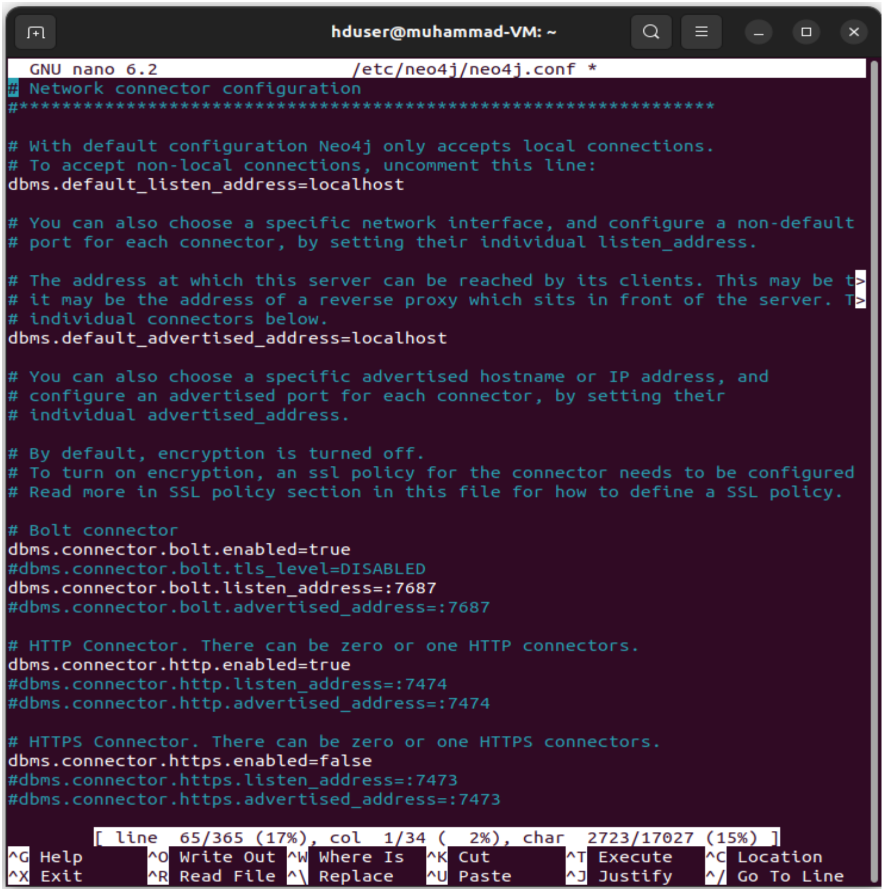
Now that you have set an administrator password and tested connecting to Neo4j, exit from the cypher-shell prompt by typing `:exit`

`Neo4j@neo4j>:exit`

Open the file using the following command on the terminal

`$sudo nano /etc/neo4j/neo4j.conf`

After opening this file, press Alt + C to open a new terminal and check the line number of the neo4j.conf file and uncomment the following lines as mentioned in the below screenshot at line no. 70 and onwards (**Alt + c** to display the line in nano editor).



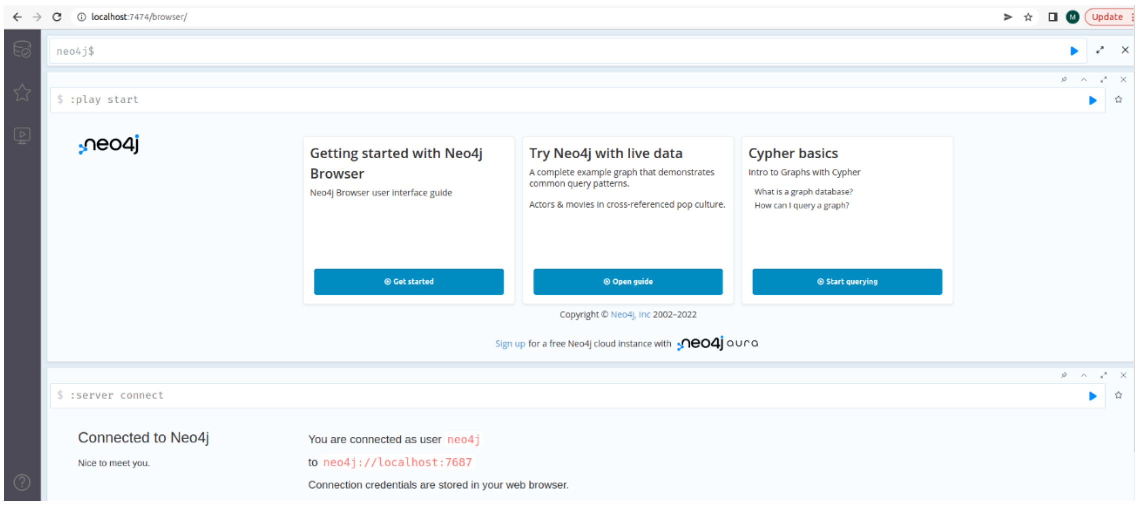
```
  GNU nano 6.2                    /etc/neo4j/neo4j.conf *
# Network connector configuration
#*******************************************************************

# With default configuration Neo4j only accepts local connections.
# To accept non-local connections, uncomment this line:
dbms.default_listen_address=localhost

# You can also choose a specific network interface, and configure a non-default
# port for each connector, by setting their individual listen_address.

# The address at which this server can be reached by its clients. This may be t>
# it may be the address of a reverse proxy which sits in front of the server. T>
# individual connectors below.
dbms.default_advertised_address=localhost

# You can also choose a specific advertised hostname or IP address, and
# configure an advertised port for each connector, by setting their
# individual advertised_address.

# By default, encryption is turned off.
# To turn on encryption, an ssl policy for the connector needs to be configured
# Read more in SSL policy section in this file for how to define a SSL policy.

# Bolt connector
dbms.connector.bolt.enabled=true
#dbms.connector.bolt.tls_level=DISABLED
dbms.connector.bolt.listen_address=:7687
#dbms.connector.bolt.advertised_address=:7687

# HTTP Connector. There can be zero or one HTTP connectors.
dbms.connector.http.enabled=true
#dbms.connector.http.listen_address=:7474
#dbms.connector.http.advertised_address=:7474

# HTTPS Connector. There can be zero or one HTTPS connectors.
dbms.connector.https.enabled=false
#dbms.connector.https.listen_address=:7473
#dbms.connector.https.advertised_address=:7473

        [ line  65/365 (17%), col  1/34 (  2%), char  2723/17027 (15%) ]
^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```

Press **ctrl + x** and then save the file by pressing **Enter** key.

If you would like to access localhost on your Ubuntu VM instance, then use the following in the browser

`http://localhost:7474/browser`

Log in by providing user name as **neo4j** and password as '**hadoop**' or the password that you have set.


localhost:7474/browser/

```
neo4j$
```

```
$ :play start
```

Getting started with Neo4j Browser
Neo4j Browser user interface guide

Try Neo4j with live data
A complete example graph that demonstrates common query patterns.
Actors & movies in cross-referenced pop culture.

Cypher basics
Intro to Graphs with Cypher
What is a graph database?
How can I query a graph?

Copyright © Neo4j, Inc 2002-2022

Sign up for a free Neo4j cloud instance with neo4j aura

```
$ :server connect
```

Connected to Neo4j
Nice to meet you.

You are connected as user neo4j
to neo4j://localhost:7687
Connection credentials are stored in your web browser.

Note: The Community Edition of Neo4j supports running a single database at a time. Additionally, the Community version does not include the capability to assign roles and permissions to the users, so those steps are not included in this tutorial. For more information about various features that are supported by the Community Edition of Neo4j, consult the Neo4j Documentation mentioned in the reference. You can use graphical user interface after creation of nodes in the database.

Once you have logged into Neo4j with your username and password, you can query and add nodes and relationships to the database.

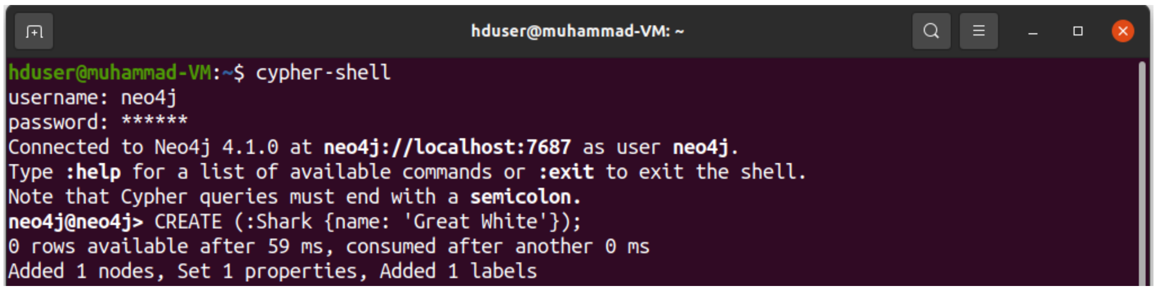
Again start cyber-shell in a new terminal by using following command as mentioned in the screenshot below

`$cyber-shell`

Provide username and password. To get started, add a Great White shark node to Neo4j. The following command will create a node of type Shark, with a name Great White.

`$CREATE (:Shark {name: 'Great White'});`

After each command you will receive output that is similar to the following:

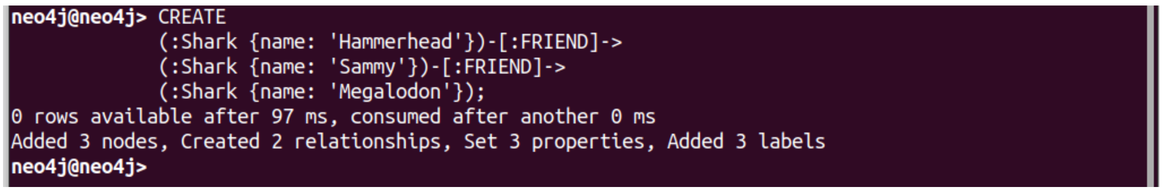

hduser@muhammad-VM: ~

```
hduser@muhammad-VM:~$ cypher-shell
username: neo4j
password: ******
Connected to Neo4j 4.1.0 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j> CREATE (:Shark {name: 'Great White'});
0 rows available after 59 ms, consumed after another 0 ms
Added 1 nodes, Set 1 properties, Added 1 labels
```

Note: A full explanation of each of the following cypher queries is beyond the scope of this tutorial. For details about the syntax of the cypher query language, refer to the link **https://neo4j.com/docs/cypher-manual/current/**.

Next, add some more sharks, and relate them using a relationship called FRIEND. Neo4j allows you to relate nodes with arbitrarily named relationships, so FRIEND can be whatever label for a relationship that you would like to use.

In the following example we'll add three sharks, and link them together using a relationship called FRIEND:



```
neo4j@neo4j> CREATE
             (:Shark {name: 'Hammerhead'})-[:FRIEND]->
             (:Shark {name: 'Sammy'})-[:FRIEND]->
             (:Shark {name: 'Megalodon'});
0 rows available after 97 ms, consumed after another 0 ms
Added 3 nodes, Created 2 relationships, Set 3 properties, Added 3 labels
neo4j@neo4j>
```

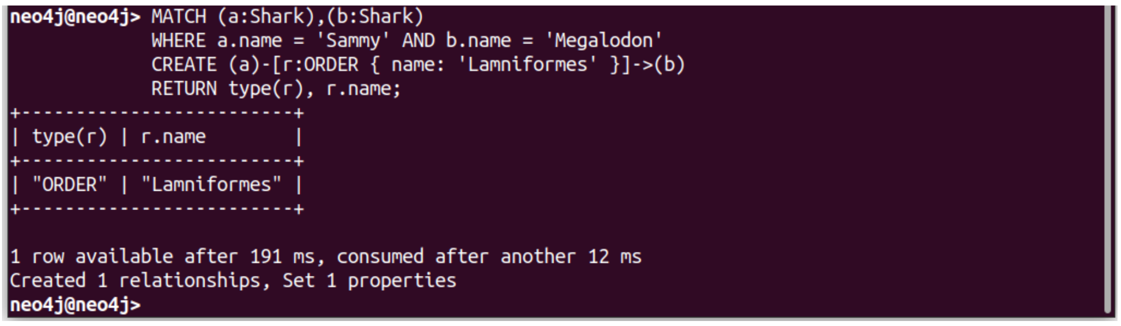
6) You should receive output that indicates the three new sharks were added to the

   Neo4j allows you to relate nodes using arbitrary names for relationships, so in addition to their existing FRIEND relation, Sammy and Megalodon can also be related using a taxonomic rank.

   Sammy and Megalodon share a common order of Lamniformes. Since relationships can have properties just like nodes, we'll create an ORDER relationship with a name property that is set to Lamniformes to help describe one of Sammy and Megalodon's relationships:

```
MATCH (a:Shark),(b:Shark)
WHERE a.name = 'Sammy' AND b.name = 'Megalodon'
CREATE (a)-[r:ORDER { name: 'Lamniformes' }]->(b)
RETURN type(r), r.name;
```

   After adding that relationship, you should have output like the following:

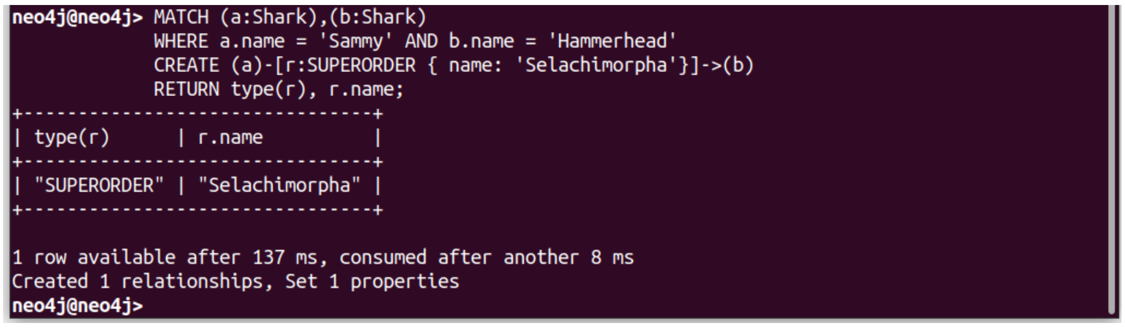


```
neo4j@neo4j> MATCH (a:Shark),(b:Shark)
             WHERE a.name = 'Sammy' AND b.name = 'Megalodon'
             CREATE (a)-[r:ORDER { name: 'Lamniformes' }]->(b)
             RETURN type(r), r.name;
+------------------------+
| type(r) | r.name       |
+------------------------+
| "ORDER" | "Lamniformes" |
+------------------------+

1 row available after 191 ms, consumed after another 12 ms
Created 1 relationships, Set 1 properties
neo4j@neo4j>
```

7) Next, add a SUPERORDER relationship between Sammy and Hammerhead based on their taxonomic superorder, which is Selachimorpha. Again, the relationship is given a name property, which is set to Selachimorpha:

```
MATCH (a:Shark),(b:Shark)
WHERE a.name = 'Sammy' AND b.name = 'Hammerhead'
CREATE (a)-[r:SUPERORDER { name: 'Selachimorpha'}]->(b)
RETURN type(r), r.name;
```

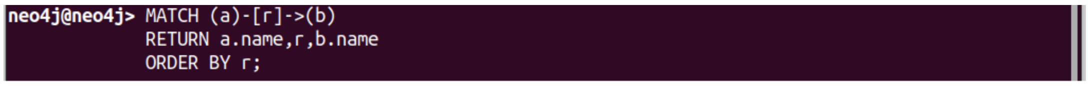
   Again you will receive output that indicates the type of the relationship, along with the name that was added to describe the relation.



```
neo4j@neo4j> MATCH (a:Shark),(b:Shark)
             WHERE a.name = 'Sammy' AND b.name = 'Hammerhead'
             CREATE (a)-[r:SUPERORDER { name: 'Selachimorpha'}]->(b)
             RETURN type(r), r.name;
+---------------------------------+
| type(r)      | r.name           |
+---------------------------------+
| "SUPERORDER" | "Selachimorpha" |
+---------------------------------+

1 row available after 137 ms, consumed after another 8 ms
Created 1 relationships, Set 1 properties
neo4j@neo4j>
```

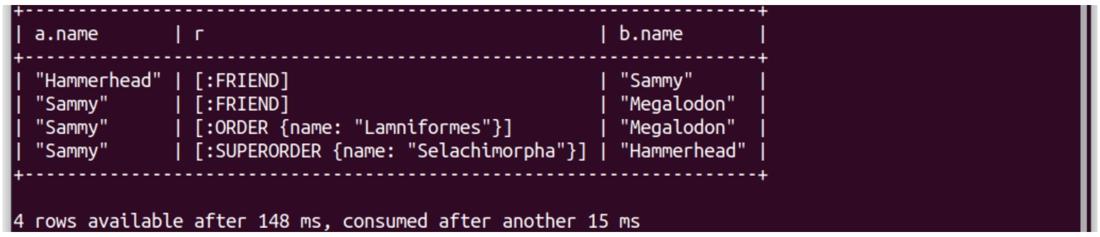
8) Finally, with all these nodes and relationships defined and stored in Neo4j, examine the data using the following query

```
MATCH (a)-[r]->(b)
RETURN a.name,r,b.name
ORDER BY r;
```



```
neo4j@neo4j> MATCH (a)-[r]->(b)
             RETURN a.name,r,b.name
             ORDER BY r;
```

You should receive output like the following:



```
+-------------------------------------------------------------------+
| a.name       | r                                  | b.name       |
+-------------------------------------------------------------------+
| "Hammerhead" | [:FRIEND]                          | "Sammy"      |
| "Sammy"      | [:FRIEND]                          | "Megalodon"  |
| "Sammy"      | [:ORDER {name: "Lamniformes"}]     | "Megalodon"  |
| "Sammy"      | [:SUPERORDER {name: "Selachimorpha"}] | "Hammerhead" |
+-------------------------------------------------------------------+

4 rows available after 148 ms, consumed after another 15 ms
```

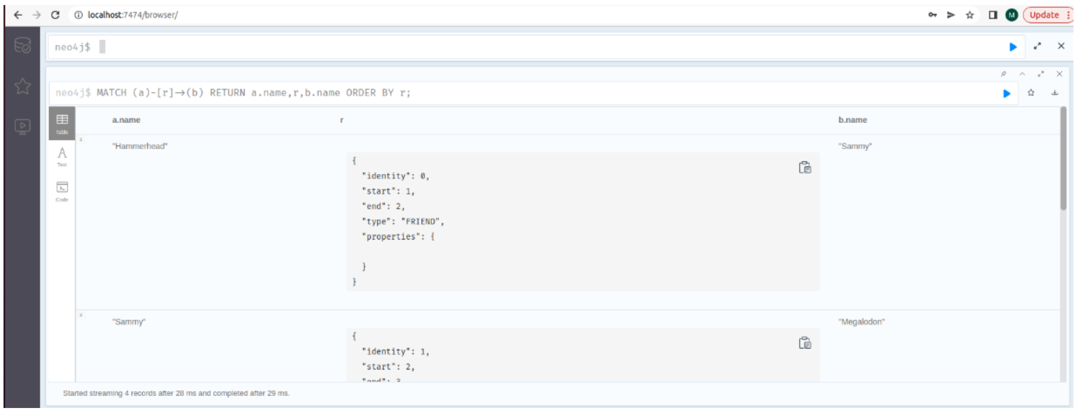
The output includes the FRIEND relationships that were defined between Hammerhead, Sammy, and Megalodon, as well as the ORDER and SUPERORDER taxonomic relationships.

When you are finished adding and exploring nodes and relationships to your Neo4j database, type the :exit command to leave the cypher-shell.

9) All commands can be executed on the localhost browser terminal also as you can see the screenshots

User Name: neo4j

Password: “hadoop” or your own choice



```
localhost:7474/browser/
neo4j$
neo4j$ MATCH (a)-[r]→(b) RETURN a.name,r,b.name ORDER BY r;
a.name          r                          b.name
"Hammerhead"    {                          "Sammy"
                  "identity": 0,
                  "start": 1,
                  "end": 2,
                  "type": "FRIEND",
                  "properties": {

                  }
                }
"Sammy"         {                          "Megalodon"
                  "identity": 1,
                  "start": 2,
Started streaming 4 records after 28 ms and completed after 29 ms.
```

Create a node and a relationship

```
neo4j@CREATE (n:Person {name: 'John'})-[r:KNOWS]->(m:Person {name: 'Alice'})
RETURN n, r, m;
```

To retrieve all nodes in the graph, you can use the following query:

```
neo4j@MATCH (n)
RETURN n;
```

To retrieve a specific node by its property (e.g., "name" equals "John"):

```
neo4j@MATCH (n:Person {name: 'John'})
RETURN n;
```

To retrieve relationships between nodes, you can use:

```
neo4j@MATCH ()-[r]->()
RETURN r;
```

To find people that John knows:

```
neo4j@MATCH (n:Person {name: 'John'})-[:KNOWS]->(friend)
RETURN friend;
```

To find friends of friends of John:

```
neo4j@MATCH (n:Person {name: 'John'})-[:KNOWS*2]-(fof)
WHERE n <> fof
RETURN DISTINCT fof;
```

To count the number of friends John knows:

```
neo4j@MATCH (n:Person {name: 'John'})-[:KNOWS]->(friend)
RETURN count(friend) AS numberOfFriends;
```

To update a node's property (e.g., change John's name):

```
neo4j@MATCH (n:Person {name: 'John'})
SET n.name = 'Johnny'
RETURN n;
```

To delete the "KNOWS" relationship between John and Alice:

```
neo4j@MATCH (n:Person {name: 'John'})-[r:KNOWS]->(m:Person {name: 'Alice'})
DELETE r;
```

To delete a node and all its relationships (e.g., delete the "Alice" node and her relationships):

```
neo4j@MATCH (n:Person {name: 'Alice'})
DETACH DELETE n;
```

To create a new node and a relationship with John:

```
neo4j@CREATE (newPerson:Person {name: 'Eva'})
CREATE (newPerson)-[:KNOWS]->(john)
RETURN newPerson;
```

If you would like to delete all nodes and relationships in the database, clearing the entire graph.

```
MATCH(n)
DETACH n;
```

10)

```
neo4j@CREATE (shakespeare:Author {firstname:'William', lastname:'Shakespeare'}),
       (juliusCaesar:Play {title:'Julius Caesar'}),
       (shakespeare)-[:WROTE_PLAY {year:1599}]->(juliusCaesar),
       (theTempest:Play {title:'The Tempest'}),
       (shakespeare)-[:WROTE_PLAY {year:1610}]->(theTempest),
       (rsc:Company {name:'RSC'}),
       (production1:Production {name:'Julius Caesar'}),
       (rsc)-[:PRODUCED]->(production1),
       (production1)-[:PRODUCTION_OF]->(juliusCaesar),
       (performance1:Performance {date:20120729}),
       (performance1)-[:PERFORMANCE_OF]->(production1),
       (production2:Production {name:'The Tempest'}),
       (rsc)-[:PRODUCED]->(production2),
       (production2)-[:PRODUCTION_OF]->(theTempest),
       (performance2:Performance {date:20061121}),
       (performance2)-[:PERFORMANCE_OF]->(production2),
       (performance3:Performance {date:20120730}),
       (performance3)-[:PERFORMANCE_OF]->(production1),
       (billy:User {name:'Billy'}),
       (review:Review {rating:5, review:'This was awesome!'}),
       (billy)-[:WROTE_REVIEW]->(review),
       (review)-[:RATED]->(performance1),
       (theatreRoyal:Venue {name:'Theatre Royal'}),
       (performance1)-[:VENUE]->(theatreRoyal),
       (performance2)-[:VENUE]->(theatreRoyal),
       (performance3)-[:VENUE]->(theatreRoyal),
       (greyStreet:Street {name:'Grey Street'}),
       (theatreRoyal)-[:STREET]->(greyStreet),
       (newcastle:City {name:'Newcastle'}),
       (greyStreet)-[:CITY]->(newcastle),
       (tyneAndWear:County {name:'Tyne and Wear'}),
       (newcastle)-[:COUNTY]->(tyneAndWear),
       (england:Country {name:'England'}),
       (tyneAndWear)-[:COUNTRY]->(england),
       (stratford:City {name:'Stratford upon Avon'}),
       (stratford)-[:COUNTRY]->(england),
       (rsc)-[:BASED_IN]->(stratford),
       (shakespeare)-[:BORN_IN]->(stratford);
```

11) The queries are based on the above Graph are mentioned below

```
MATCH
(theater:Venue {name:'Theatre Royal'}),
(newcastle:City {name:'Newcastle'}),
(bard:Author {lastname:'Shakespeare'}),
(newcastle)<-[:STREET|CITY*1..2]-(theater)
<-[:VENUE]-()-[:PERFORMANCE_OF]->()
-[:PRODUCTION_OF]->(play)<-[:WROTE_PLAY]-(bard)
RETURN DISTINCT play.title AS play;
```

12)

```
MATCH
(theater:Venue {name:'Theatre Royal'}),
(newcastle:City {name:'Newcastle'}),
(bard:Author {lastname:'Shakespeare'}),
(newcastle)<-[:STREET|CITY*1..2]-(theater)
<-[:VENUE]-()-[:PERFORMANCE_OF]->()
-[:PRODUCTION_OF]->(play)<-[w:WROTE_PLAY]-(bard)
WHERE w.year > 1608
```

```
RETURN DISTINCT play.title AS play;
```

12)

```
MATCH
(theater:Venue {name:'Theatre Royal'}),
(newcastle:City {name:'Newcastle'}),
(bard:Author {lastname:'Shakespeare'}),
(newcastle)<-[:STREET|CITY*1..2]-(theater)
<-[:VENUE]-()-[p:PERFORMANCE_OF]->()
-[:PRODUCTION_OF]->(play)<-[:WROTE_PLAY]-(bard)
RETURN   play.title AS play, count(p) AS performance_count
ORDER BY performance_count DESC;
```

Conclusion:

You have now installed, configured, and added data to Neo4j on your server. You also optionally configured Neo4j to accept connections from remote systems and secured it using UFW.

If you want to learn more about using Neo4j and the cypher query language, consult the official Neo4j Documentation.

Reference:

- https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-neo4j-on-ubuntu-20-04
- https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-neo4j-on-ubuntu-22-04
- https://neo4j.com/docs/cypher-manual/current/
- https://neo4j.com/blog/neo4j-video-tutorials/