Hanjiao Qiu
CSCIE63 Big Data Analytics
Assignment 10

Assignment 10 Solution

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. You are not obliged to use Java or Eclipse. You are welcome to use any language and any IDE of your choice.

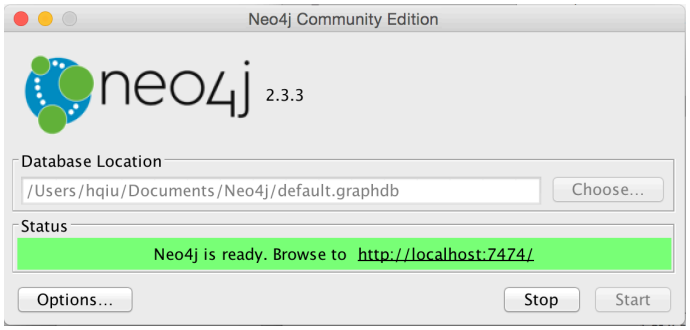
Problem 1. The following is the content of Movies database. Bring that database into Neo4J using curl.

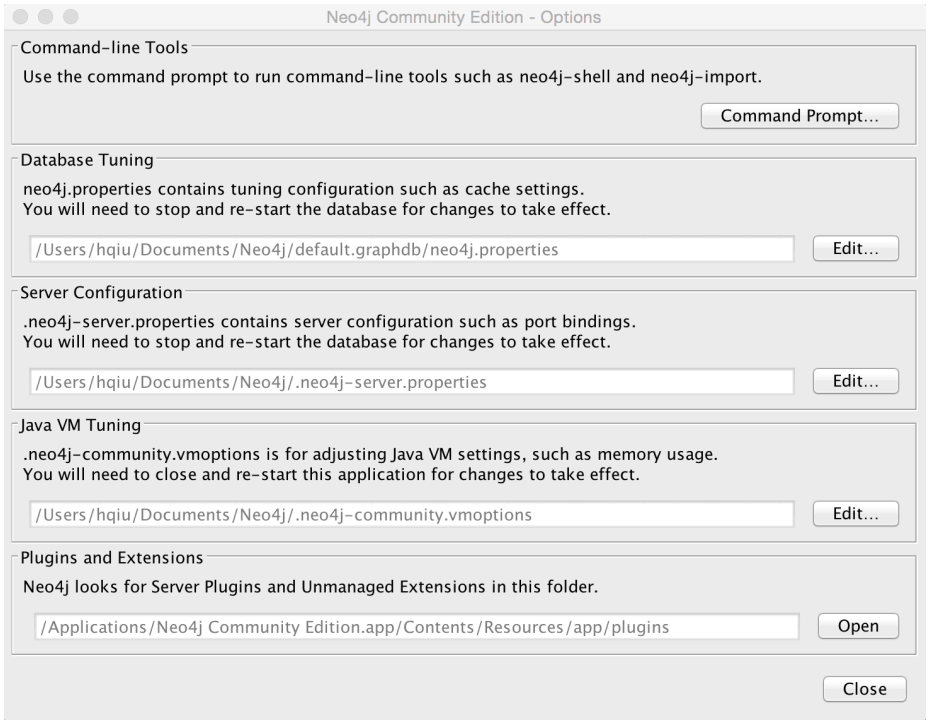
```
CREATE (matrix1:Movie { title : 'The Matrix', year : '1999-03-31' })

CREATE (matrix2:Movie { title : 'The Matrix Reloaded', year : '2003-05-07' })

CREATE (matrix3:Movie { title : 'The Matrix Revolutions', year : '2003-10-27' })

CREATE (keanu:Actor { name:'Keanu Reeves' })

CREATE (laurence:Actor { name:'Laurence Fishburne' })

CREATE (carrieanne:Actor { name:'Carrie-Anne Moss' })

CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix1)

CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix2)

CREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix3)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix1)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix2)

CREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]->(matrix3)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix1)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix2)

CREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix3)
```

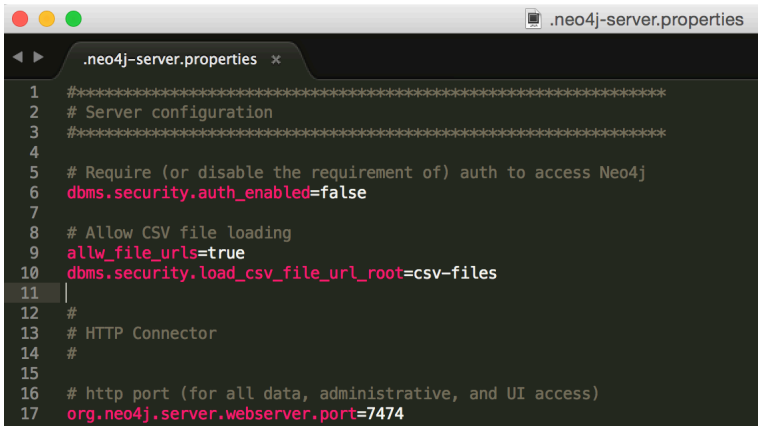
Solution:

1. Install Neo4j Community Edition from:
   http://neo4j.com/download-thanks/?edition=community

   Configure ".neo4j-server.properties" and start Neo4j server:

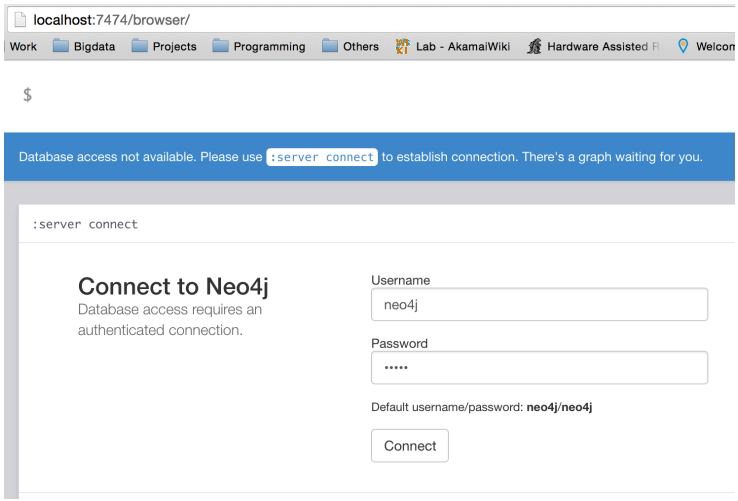






Login into Cypher browser:

http://localhost:7474/browser/



2. Use REST API to communicate with the server. Use **curl** to submit REST statements.

Here is the reference link:
http://neo4j.com/docs/stable/rest-api-transactional.html#rest-api-execute-multiple-statements

We can execute multiple statements above through 1 line by passing JSON file to curl.

As professor mentioned, a simple trick to write JSON for more complex Cypher multi-line commands is to run the statements in Cypher browser and then copy JSON payloads from Cypher browser code section (</>). Don't forget to commit after transaction.

Here is the JSON file which contains the collection of statements:

Node01.json

```
{
  "statements": [
    {
      "statement": "CREATE (matrix1:Movie { title : 'The Matrix', year : '1999-
03-31' })\nCREATE (matrix2:Movie { title : 'The Matrix Reloaded', year : '2003-
05-07' })\nCREATE (matrix3:Movie { title : 'The Matrix Revolutions', year :
'2003-10-27' })\nCREATE (keanu:Actor { name:'Keanu Reeves' })\nCREATE
(laurence:Actor { name:'Laurence Fishburne' })\nCREATE (carrieanne:Actor {
name:'Carrie-Anne Moss' })\nCREATE (keanu)-[:ACTS_IN { role : 'Neo' }]-
>(matrix1)\nCREATE (keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix2)\nCREATE
(keanu)-[:ACTS_IN { role : 'Neo' }]->(matrix3)\nCREATE (laurence)-[:ACTS_IN {
role : 'Morpheus' }]->(matrix1)\nCREATE (laurence)-[:ACTS_IN { role :
```

```
'Morpheus' }]->(matrix2)\nCREATE (laurence)-[:ACTS_IN { role : 'Morpheus' }]-
>(matrix3)\nCREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]-
>(matrix1)\nCREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]-
>(matrix2)\nCREATE (carrieanne)-[:ACTS_IN { role : 'Trinity' }]->(matrix3)",
      "resultDataContents": [
        "row",
        "graph"
      ],
      "includeStats": true
    }
  ]
}
```

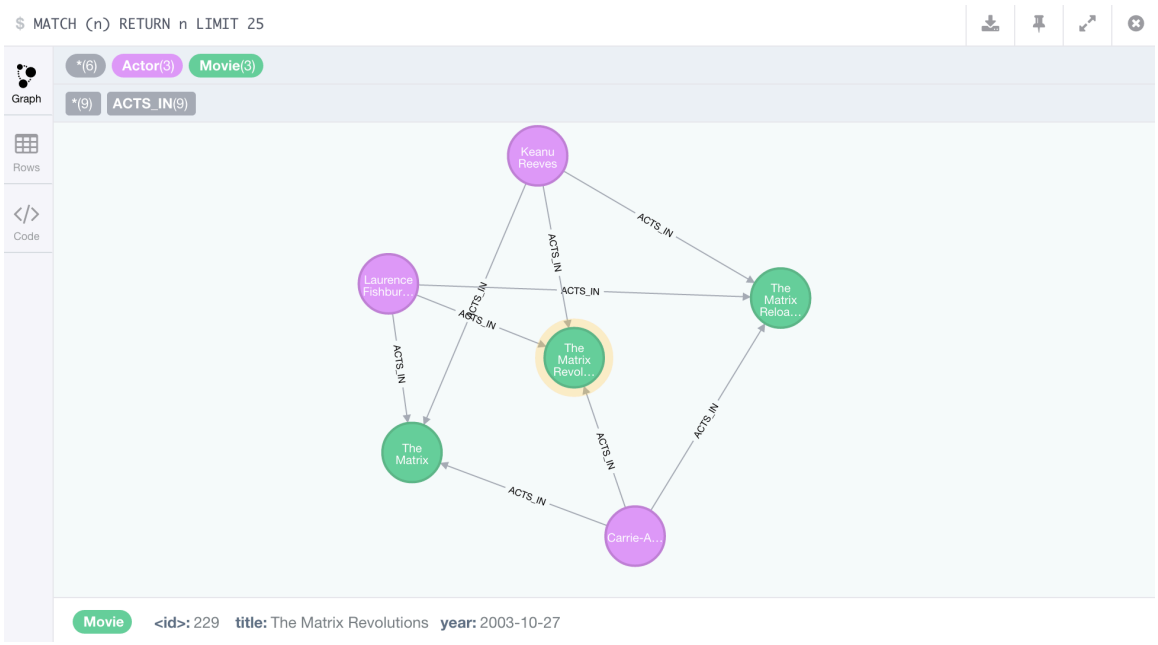
Here is the command line to run on a single line:

```
hqiu@bos-mp9cx>> curl -i -H accept:application/json -H content-
type:application/json -XPOST http://localhost:7474/db/data/transaction/commit -
-data "@node01.json"
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Date: Thu, 14 Apr 2016 02:38:25 GMT
Content-Type: application/json
Access-Control-Allow-Origin: *
Content-Length: 311
Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[],"stats":{"contains_updates":true,"nodes_cre
ated":6,"nodes_deleted":0,"properties_set":18,"relationships_created":9,"relati
onship_deleted":0,"labels_added":6,"labels_removed":0,"indexes_added":0,"indexe
s_removed":0,"constraints_added":0,"constraints_removed":0}}],"errors":[]}
```

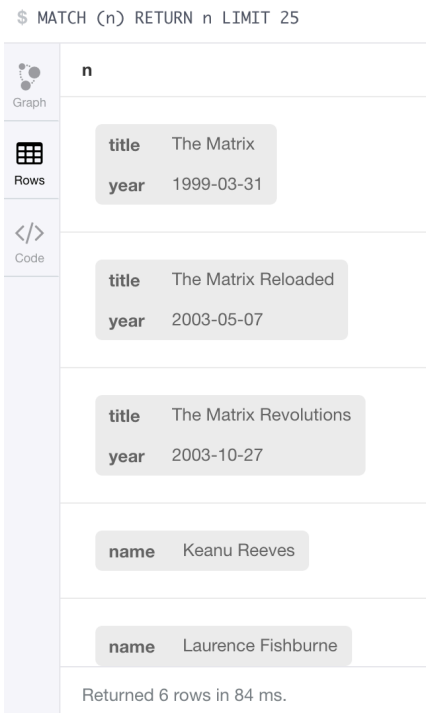
3. Check the results.

```
hqiu@bos-mp9cx>> curl -i -H accept:application/json -H content-type:application/json -XPOST http://lo
calhost:7474/db/data/transaction/commit --data "@node01.json"
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Date: Thu, 14 Apr 2016 02:38:25 GMT
Content-Type: application/json
Access-Control-Allow-Origin: *
Content-Length: 311
Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[],"stats":{"contains_updates":true,"nodes_created":6,"nodes_deleted
":0,"properties_set":18,"relationships_created":9,"relationship_deleted":0,"labels_added":6,"labels_r
emoved":0,"indexes_added":0,"indexes_removed":0,"constraints_added":0,"constraints_removed":0}}],"err
ors":[]}hqiu@bos-mp9cx>>
```

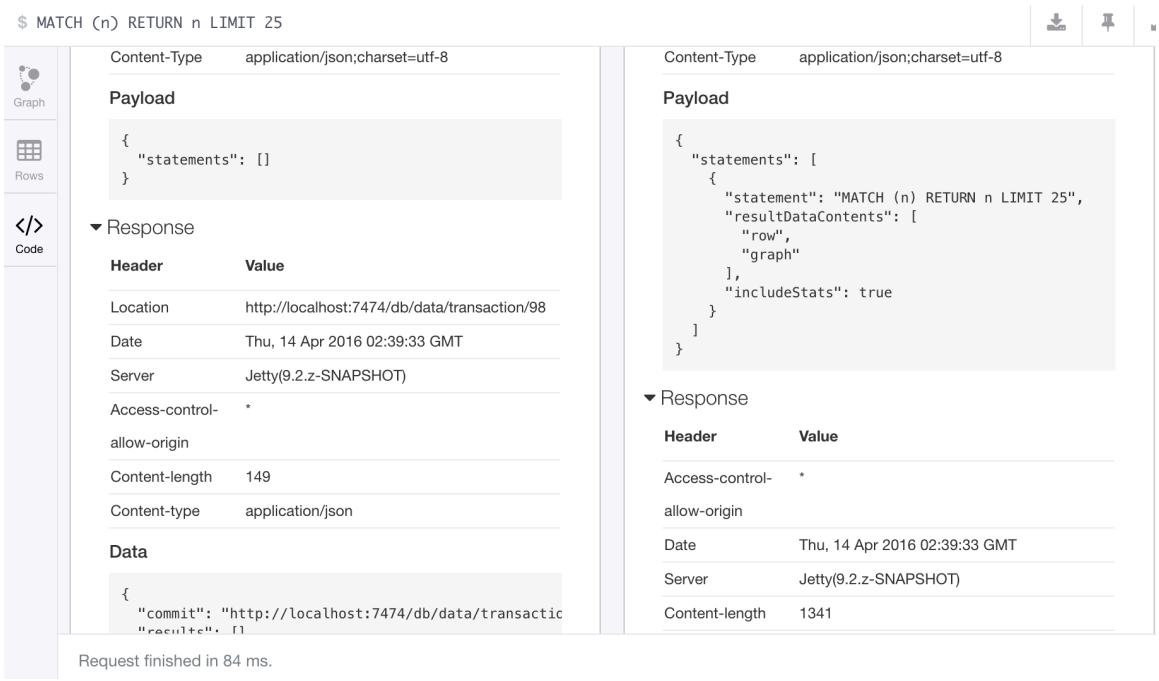
6 new nodes, 6 labels, 9 relationships and 18 properties have been added.



Check the Rows.

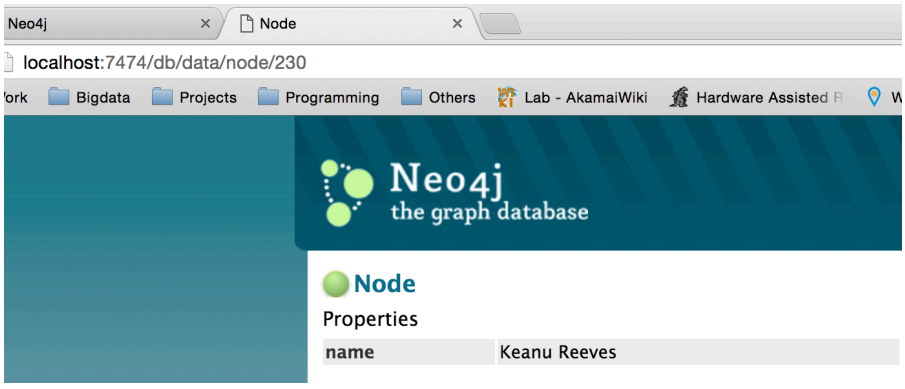


Check the Code.

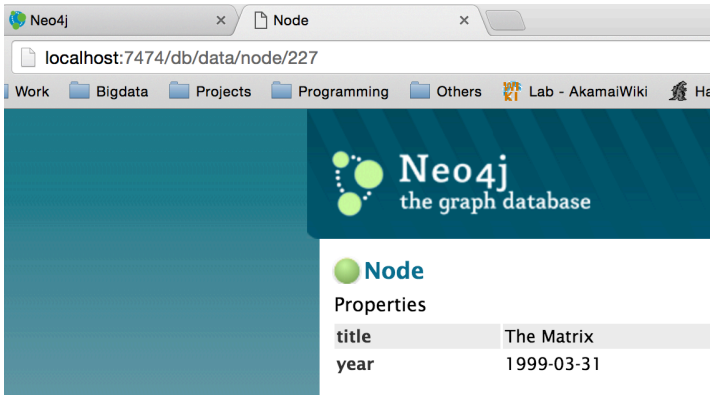


Check the payload and nodes.

```
      "nodes": [
        {
          "id": "230",
          "labels": [
            "Actor"
          ],
          "properties": {
            "name": "Keanu Reeves"
          }
        }
      ],
      "relationships": []
    }
  },
```



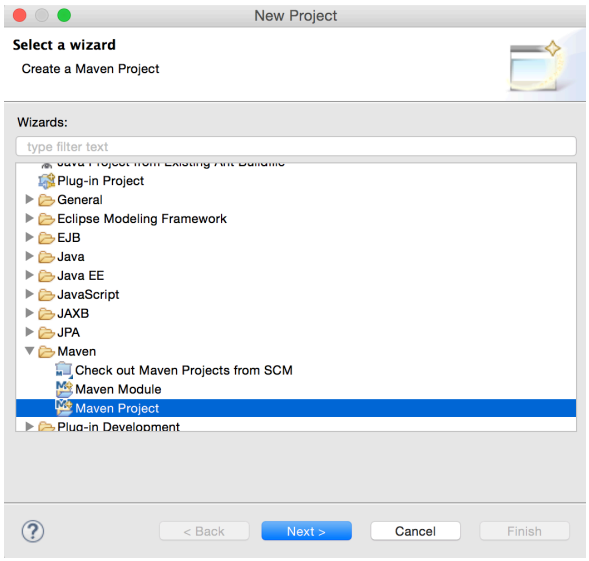
```
    "graph": {
      "nodes": [
        {
          "id": "227",
          "labels": [
            "Movie"
          ],
          "properties": {
            "title": "The Matrix",
            "year": "1999-03-31"
          }
        }
      ],
      "relationships": []
    }
  },
```

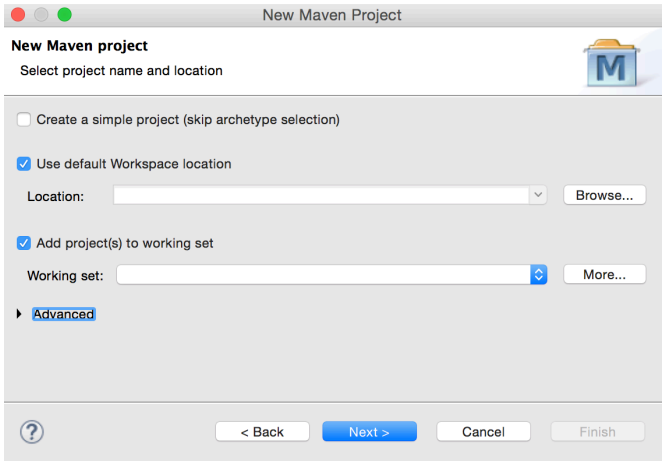



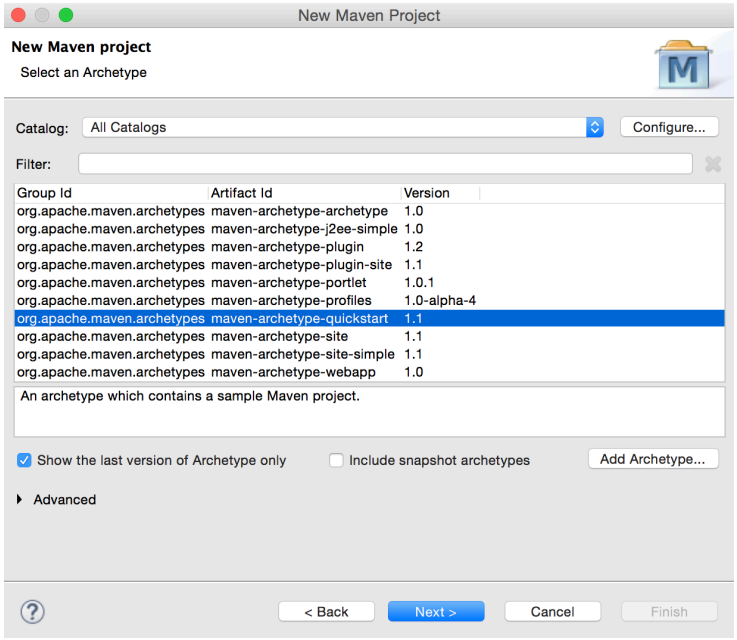
Problem 2. Keanu Reeves acted in the movie "John Wick" which is not in the database. That movie was directed by Chad Stahelski and David Leitch. Cast of the movie included William Dafoe and Michael Nyquist. Add all of those people and the roles they played in this movie to the database using JAVA REST API or one of other RESTful APIs for Neo4J in a language of your choice. Demonstrate that you have successfully brought data about John Wick movie into the database. You can use Cypher Browser or any other means.

Solution:

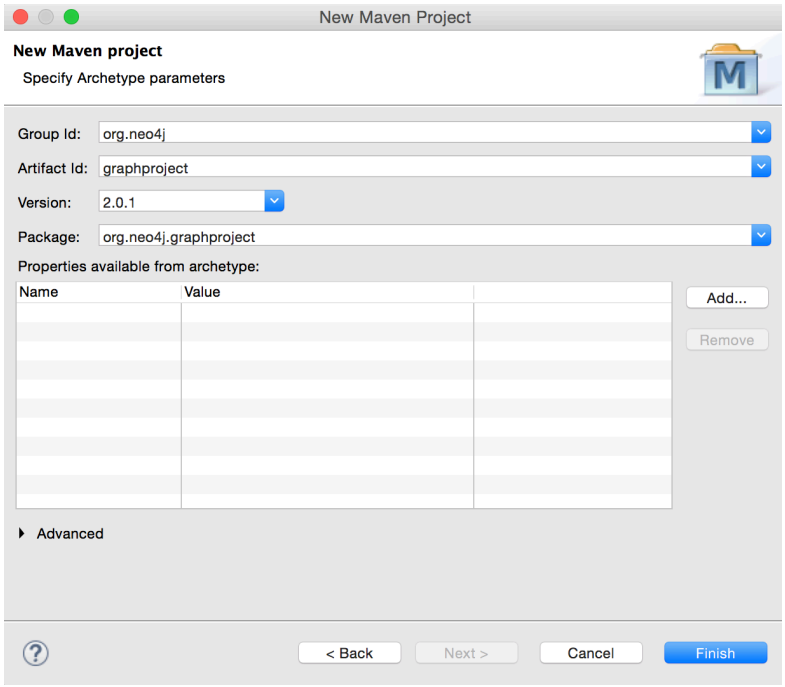
1. Configure Neo4j graph project on Eclipse. Create a Maven project.


New Project
Select a wizard
Create a Maven Project
Wizards:
type filter text
Plug-in Project
General
Eclipse Modeling Framework
EJB
Java
Java EE
JavaScript
JAXB
JPA
Maven
Check out Maven Projects from SCM
Maven Module
Maven Project
Plug-in Development
< Back
Next >
Cancel
Finish

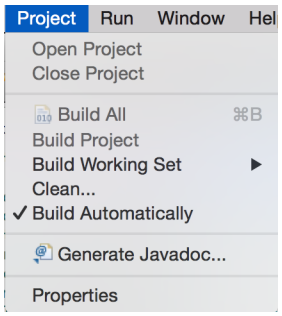

New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Cancel
Finish

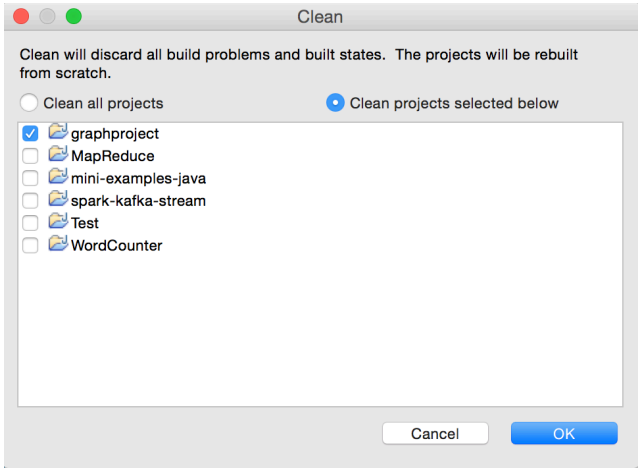

New Maven Project
New Maven project
Select an Archetype
Catalog:
All Catalogs
Configure...
Filter:
Group Id
Artifact Id
Version
org.apache.maven.archetypes maven-archetype-archetype 1.0
org.apache.maven.archetypes maven-archetype-j2ee-simple 1.0
org.apache.maven.archetypes maven-archetype-plugin 1.2
org.apache.maven.archetypes maven-archetype-plugin-site 1.1
org.apache.maven.archetypes maven-archetype-portlet 1.0.1
org.apache.maven.archetypes maven-archetype-profiles 1.0-alpha-4
org.apache.maven.archetypes maven-archetype-quickstart 1.1
org.apache.maven.archetypes maven-archetype-site 1.1
org.apache.maven.archetypes maven-archetype-site-simple 1.1
org.apache.maven.archetypes maven-archetype-webapp 1.0
An archetype which contains a sample Maven project.
Show the last version of Archetype only
Include snapshot archetypes
Add Archetype...
Advanced
< Back
Next >
Cancel
Finish

Add project name and package name.

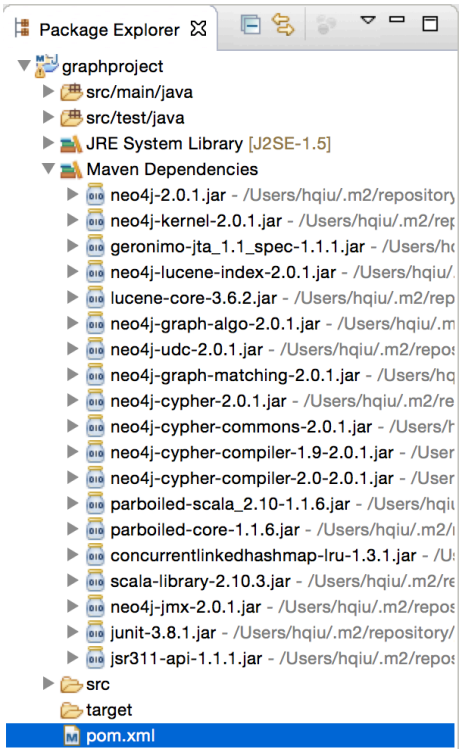


Update file “pom.xml”. Clean build path.

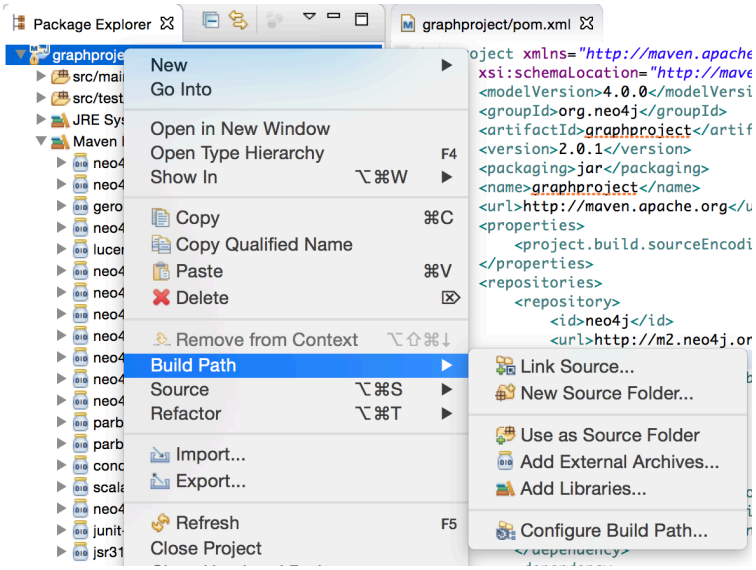


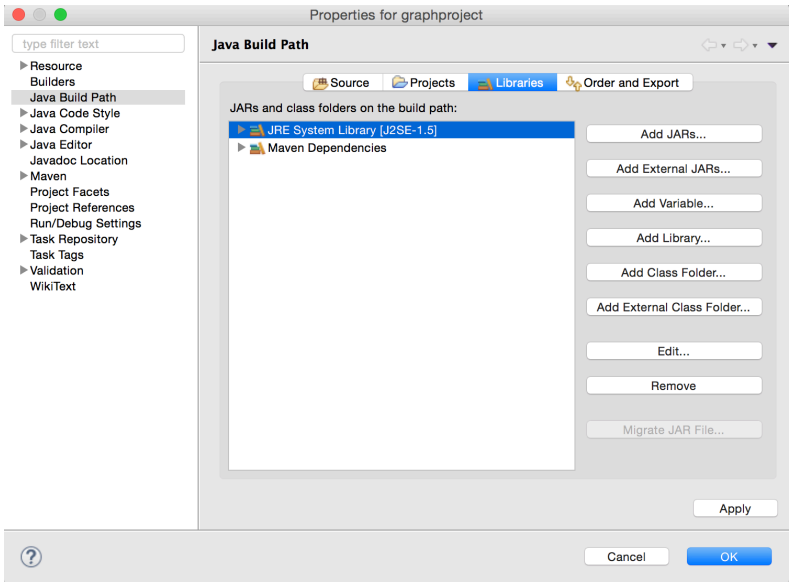


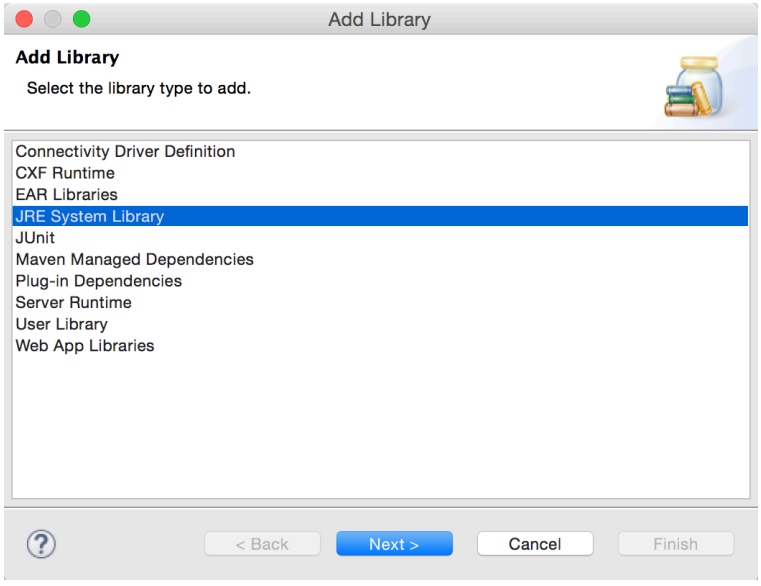
All dependencies in the “pom.xml” will be automatically loaded into the project.

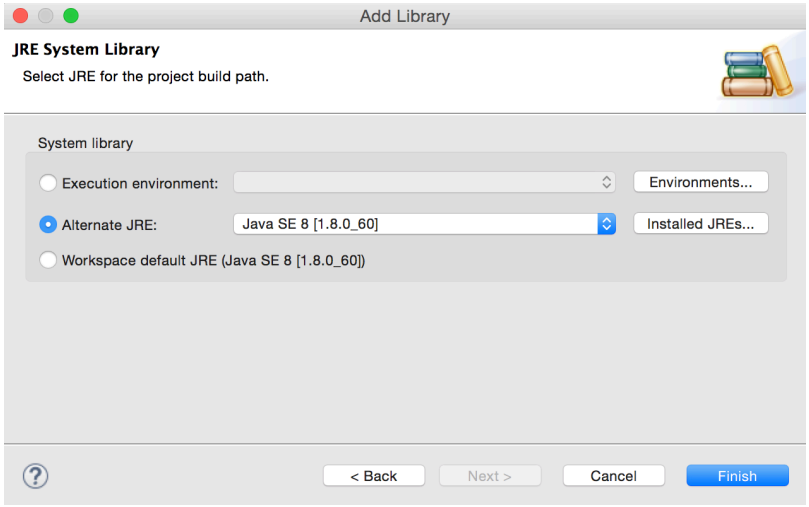


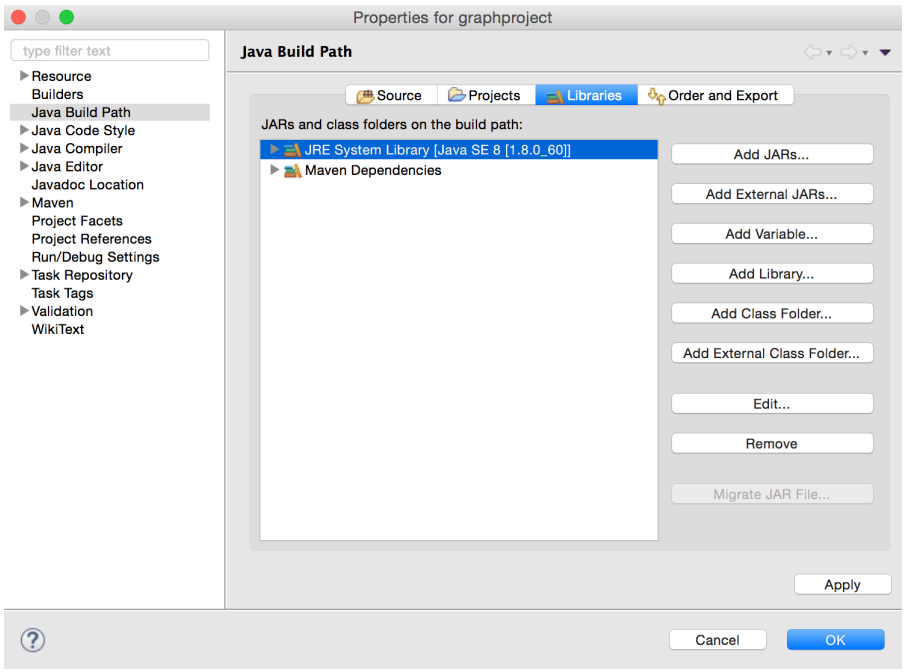
Right click on project, select “BuildPath -> Configure BuildPath ->Libraries”. Remove JRE System Library [J2SE-1.5] and then “Add Library -> JRE System Library”.



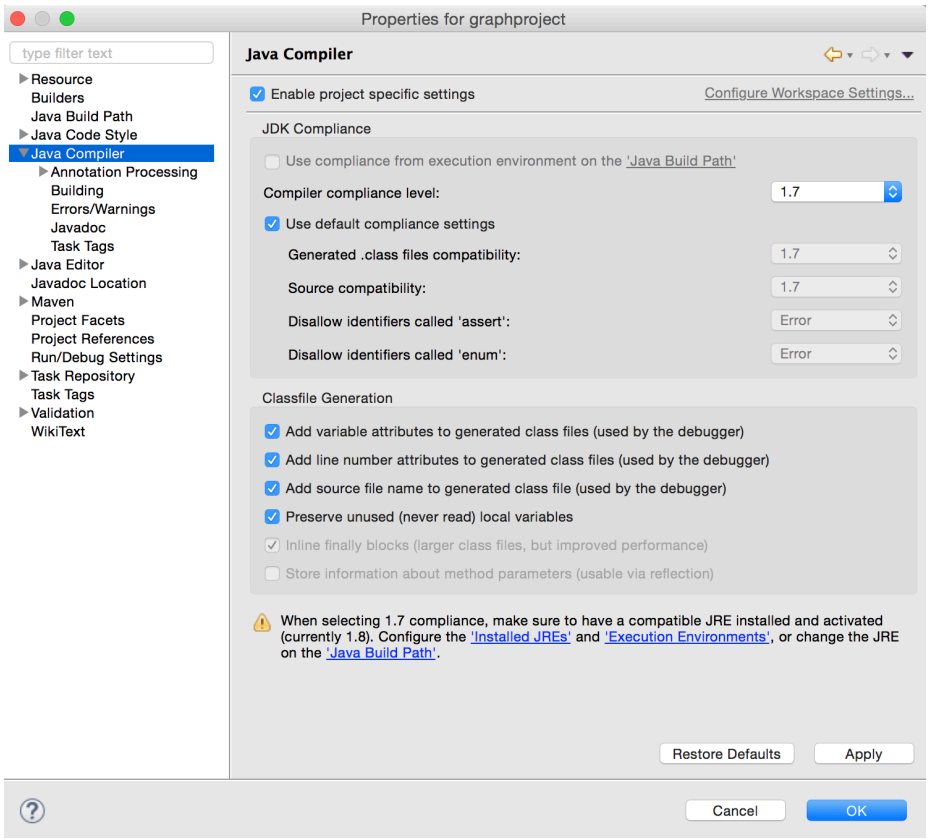

Properties for graphproject
type filter text
Resource
Builders
Java Build Path
Java Code Style
Java Compiler
Java Editor
Javadoc Location
Maven
Project Facets
Project References
Run/Debug Settings
Task Repository
Task Tags
Validation
WikiText
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [J2SE-1.5]
Maven Dependencies
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Add External Class Folder...
Edit...
Remove
Migrate JAR File...
Apply
Cancel
OK


Add Library
Add Library
Select the library type to add.
Connectivity Driver Definition
CXF Runtime
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries
< Back
Next >
Cancel
Finish

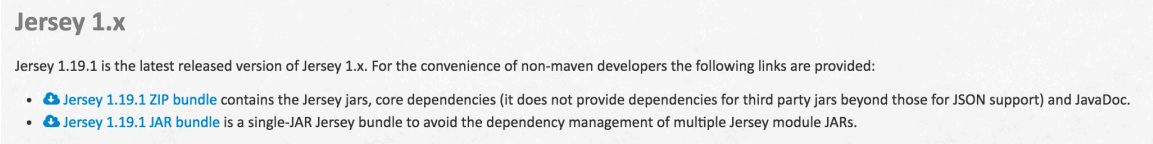

Add Library
JRE System Library
Select JRE for the project build path.
System library
Execution environment:
Environments...
Alternate JRE:
Java SE 8 [1.8.0_60]
Installed JREs...
Workspace default JRE (Java SE 8 [1.8.0_60])
< Back
Next >
Cancel
Finish

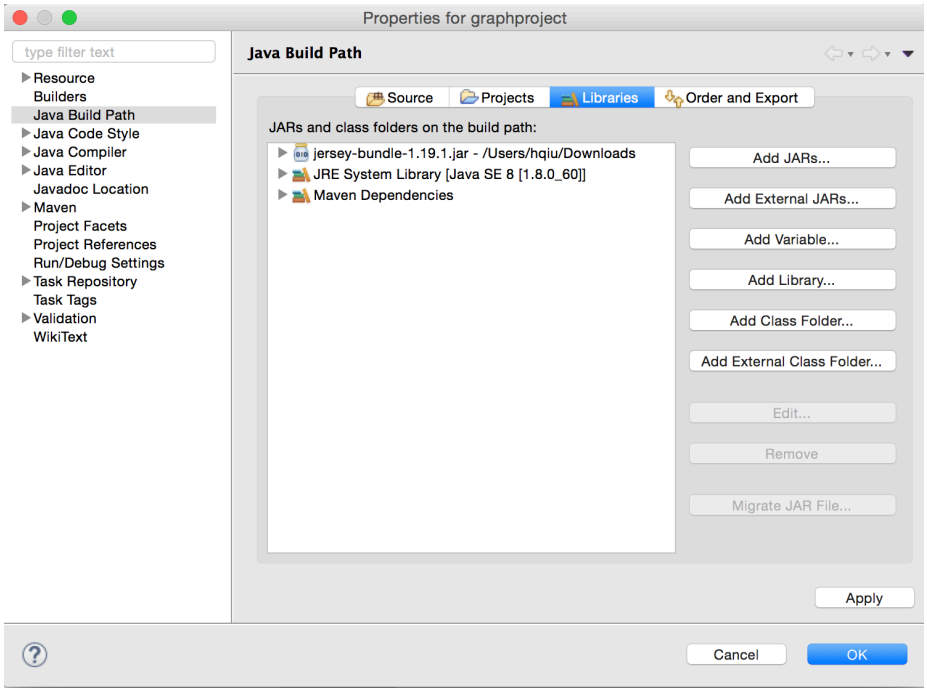


Goto “Project > Properties > Java Compiler”. Set Compiler compliance level to 1.7.



Download file “jersey-bundle-1.19.1.jar” from https://jersey.java.net/ and added to our project as an external library.





Run the source file “ConnectToServer.java” given by Professor. Check if we can connect to the server successfully.

2. Brought data about John Wick movie into the database. I’ve used two ways to solve the problem. They both use the REST API from Java, but implemented in different ways. The source code for the first method is “ConnectToServer.java”. In this file, I used fine-grained REST API calls.

Reference link for the client example file:
http://neo4j.com/docs/stable/server-java-rest-client-example.html

The source file already implements the method “createNode()”. Here I add a new method called “addLabel()”. Reference link to create a label:
http://neo4j.com/docs/stable/rest-api-node-labels.html
It will do POST calls to URI http://localhost:7474/db/data/node/{node_id}/labels and give a label to the node. Response code would be 204 if works correctly.

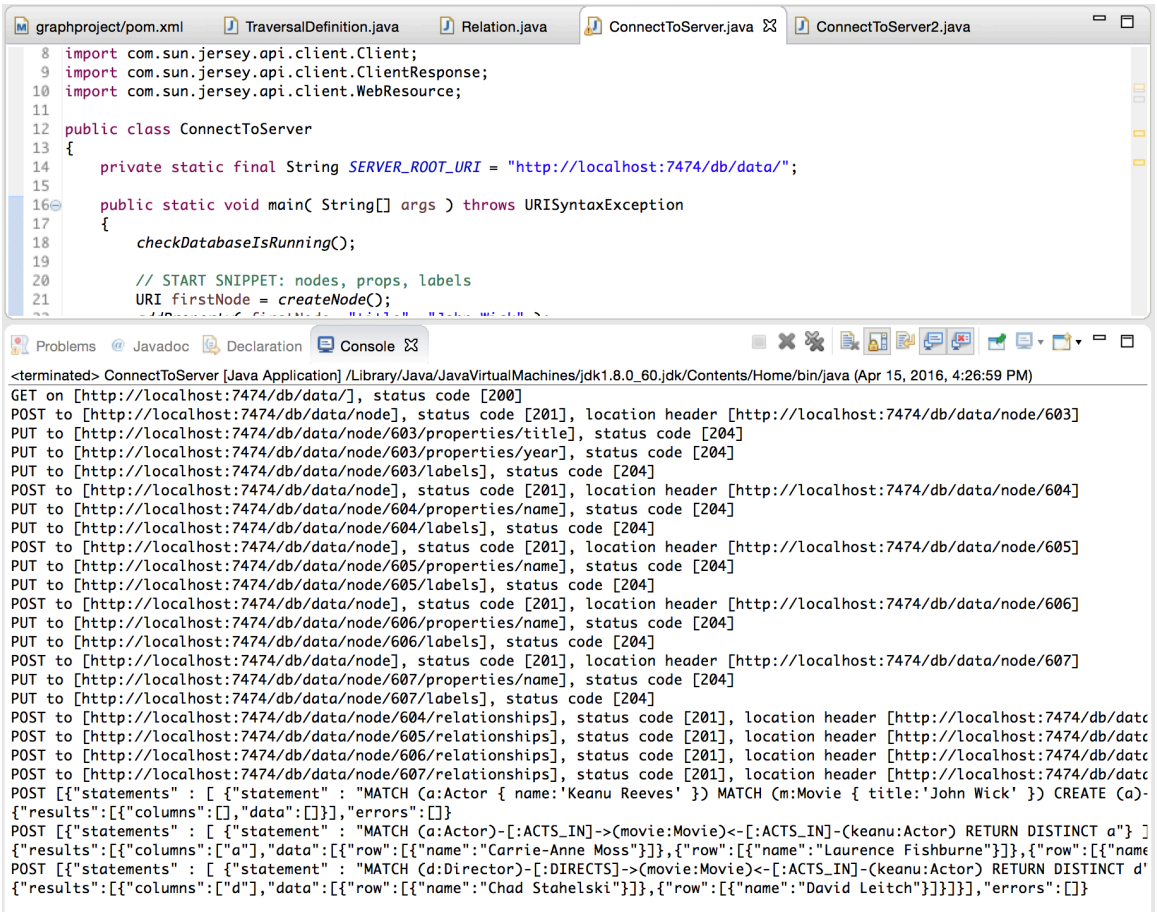
Method “addRelationship()” will add relationship to two nodes. More information can be added in JSON format.

```
public class ConnectToServer
{
   private static final String SERVER_ROOT_URI = "http://localhost:7474/db/data/";

   public static void main( String[] args ) throws URISyntaxException
   {
          checkDatabaseIsRunning();

          // START SNIPPET: nodes, props, labels
          URI firstNode = createNode();
          addProperty( firstNode, "title", "John Wick" );
          addProperty( firstNode, "year", "2014-10-24" );
          addLabel( firstNode, "Movie");

          URI secondNode = createNode();
          addProperty( secondNode, "name", "Chad Stahelski" );
          addLabel( secondNode, "Director");

          URI thirdNode = createNode();
          addProperty( thirdNode, "name", "David Leitch" );
          addLabel( thirdNode, "Director");

          URI fourthNode = createNode();
          addProperty( fourthNode, "name", "William Dafoe" );
          addLabel( fourthNode, "Actor");

          URI fifthNode = createNode();
          addProperty( fifthNode, "name", "Michael Nyquist" );
          addLabel( fifthNode, "Actor");
          // END SNIPPET: nodes, props, labels

         // START SNIPPET: addRel
          addRelationship( secondNode, firstNode, "DIRECTS", "{ \"role\" : \"Director\"}" );

          addRelationship( thirdNode, firstNode, "DIRECTS", "{ \"role\" : \"Director\"}" );

          addRelationship( fourthNode, firstNode, "ACTS_IN", "{ \"role\" : \"Marcus\"}" );

          addRelationship( fifthNode, firstNode, "ACTS_IN", "{ \"role\" : \"Viggo\"}" );
          // END SNIPPET: addRel

          sendTransactionalCypherQuery( "MATCH (a:Actor { name:\'Keanu Reeves\' }) "
                       + "MATCH (m:Movie { title:\'John Wick\' }) "
                       + "CREATE (a)-[:ACTS_IN { role : 'John' }]->(m)" );

   }

   private static void addLabel( URI nodeUri, String labelValue )
   {
          // START SNIPPET: addLabel
          String propertyUri = nodeUri.toString() + "/labels";
          // http://localhost:7474/db/data/node/{node_id}/labels
```

```
        WebResource resource = Client.create()
                    .resource( propertyUri );
        ClientResponse response = resource.accept( MediaType.APPLICATION_JSON )
                    .type( MediaType.APPLICATION_JSON )
                    .entity( "\"" + labelValue + "\"" )
                    .post( ClientResponse.class );

        System.out.println( String.format( "PUT to [%s], status code [%d]",
                    propertyUri, response.getStatus() ) );
        response.close();
        // END SNIPPET: addLabel
}
```

The source file contains a method to traverse a node. But it needs the URI of the start node. Since node "Keanu Reeves" is created in problem 1, here I will use method "sendTransactionalCypherQuery()" and pass in the statement to add in the relationship between actor "Keanu Reeves" and movie "John Wick".

Results from Eclipse:



Plain text for the results:

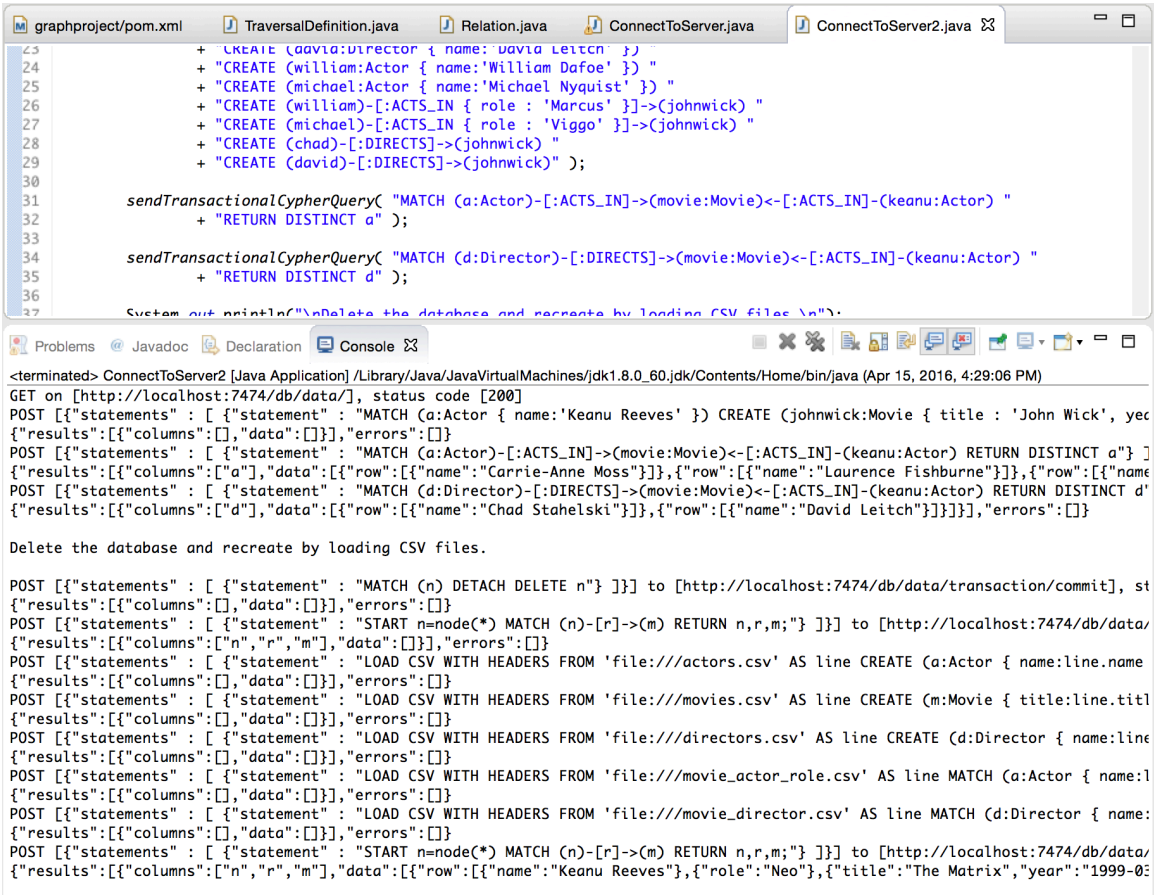
```
GET on [http://localhost:7474/db/data/], status code [200]
POST to [http://localhost:7474/db/data/node], status code [201], location
header [http://localhost:7474/db/data/node/603]
PUT to [http://localhost:7474/db/data/node/603/properties/title], status code
[204]
PUT to [http://localhost:7474/db/data/node/603/properties/year], status code
[204]
PUT to [http://localhost:7474/db/data/node/603/labels], status code [204]
POST to [http://localhost:7474/db/data/node], status code [201], location
header [http://localhost:7474/db/data/node/604]
PUT to [http://localhost:7474/db/data/node/604/properties/name], status code
[204]
PUT to [http://localhost:7474/db/data/node/604/labels], status code [204]
POST to [http://localhost:7474/db/data/node], status code [201], location
header [http://localhost:7474/db/data/node/605]
PUT to [http://localhost:7474/db/data/node/605/properties/name], status code
[204]
PUT to [http://localhost:7474/db/data/node/605/labels], status code [204]
POST to [http://localhost:7474/db/data/node], status code [201], location
header [http://localhost:7474/db/data/node/606]
PUT to [http://localhost:7474/db/data/node/606/properties/name], status code
[204]
PUT to [http://localhost:7474/db/data/node/606/labels], status code [204]
POST to [http://localhost:7474/db/data/node], status code [201], location
header [http://localhost:7474/db/data/node/607]
PUT to [http://localhost:7474/db/data/node/607/properties/name], status code
[204]
PUT to [http://localhost:7474/db/data/node/607/labels], status code [204]
POST to [http://localhost:7474/db/data/node/604/relationships], status code
[201], location header [http://localhost:7474/db/data/relationship/720]
POST to [http://localhost:7474/db/data/node/605/relationships], status code
[201], location header [http://localhost:7474/db/data/relationship/721]
POST to [http://localhost:7474/db/data/node/606/relationships], status code
[201], location header [http://localhost:7474/db/data/relationship/722]
POST to [http://localhost:7474/db/data/node/607/relationships], status code
[201], location header [http://localhost:7474/db/data/relationship/723]
POST [{"statements" : [ {"statement" : "MATCH (a:Actor { name:'Keanu Reeves' })
MATCH (m:Movie { title:'John Wick' }) CREATE (a)-[:ACTS_IN { role : 'John' }]-
>(m)"} ]}] to [http://localhost:7474/db/data/transaction/commit], status code
[200], returned data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
```

3. Method 2, directly use the method “sendTransactionalCypherQuery()” and add in all cypher query statements.

```
public class ConnectToServer2
{
   private static final String SERVER_ROOT_URI = "http://localhost:7474/db/data/";

   public static void main( String[] args ) throws URISyntaxException
   {
          checkDatabaseIsRunning();
```

```
        sendTransactionalCypherQuery( "MATCH (a:Actor { name:'Keanu Reeves' }) "
                    + "CREATE (johnwick:Movie { title : 'John Wick', year : '2014-10-24' }) "
                    + "CREATE (a)-[:ACTS_IN { role : 'John' }]->(johnwick)"
                    + "CREATE (chad:Director { name:'Chad Stahelski' }) "
                    + "CREATE (david:Director { name:'David Leitch' }) "
                    + "CREATE (william:Actor { name:'William Dafoe' }) "
                    + "CREATE (michael:Actor { name:'Michael Nyquist' }) "
                    + "CREATE (william)-[:ACTS_IN { role : 'Marcus' }]->(johnwick) "
                    + "CREATE (michael)-[:ACTS_IN { role : 'Viggo' }]->(johnwick) "
                    + "CREATE (chad)-[:DIRECTS]->(johnwick) "
                    + "CREATE (david)-[:DIRECTS]->(johnwick)" );
    }
}
```

Results from Eclipse:



Plain text results:

```
GET on [http://localhost:7474/db/data/], status code [200]
POST [{"statements" : [ {"statement" : "MATCH (a:Actor { name:'Keanu Reeves' })
CREATE (johnwick:Movie { title : 'John Wick', year : '2014-10-24' }) CREATE
(a)-[:ACTS_IN { role : 'John' }]->(johnwick)CREATE (chad:Director { name:'Chad
Stahelski' }) CREATE (david:Director { name:'David Leitch' }) CREATE
(william:Actor { name:'William Dafoe' }) CREATE (michael:Actor { name:'Michael
```

```
Nyquist' }) CREATE (william)-[:ACTS_IN { role : 'Marcus' }]->(johnwick) CREATE
(michael)-[:ACTS_IN { role : 'Viggo' }]->(johnwick) CREATE (chad)-[:DIRECTS]-
>(johnwick) CREATE (david)-[:DIRECTS]->(johnwick)"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
```

Problem 3. Find a list of actors playing in movies in which Keanu Reeves played. Find directors of movies in which K. Reeves played.

Solution:

1. Do it using REST API from Java.

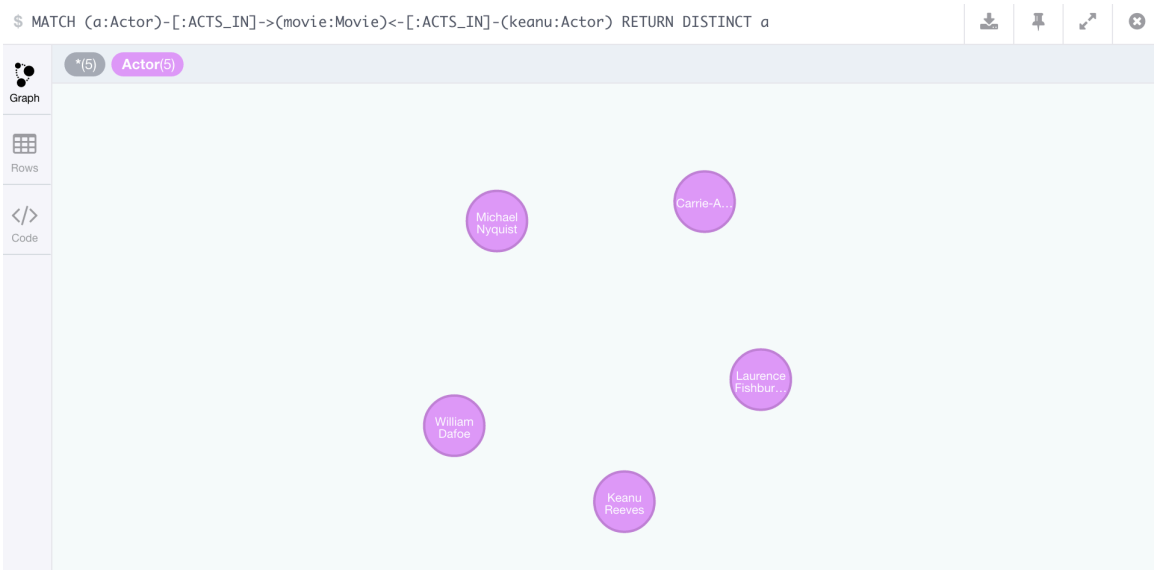
```
public class ConnectToServer
{
   private static final String SERVER_ROOT_URI = "http://localhost:7474/db/data/";

   public static void main( String[] args ) throws URISyntaxException
   {

          ...........

          sendTransactionalCypherQuery( "MATCH (a:Actor)-[:ACTS_IN]->(movie:Movie)<-[:ACTS_IN]-
(keanu:Actor) "
                        + "RETURN DISTINCT a" );

          sendTransactionalCypherQuery( "MATCH (d:Director)-[:DIRECTS]->(movie:Movie)<-[:ACTS_IN]-
(keanu:Actor) "
                        + "RETURN DISTINCT d" );

   }
}
```

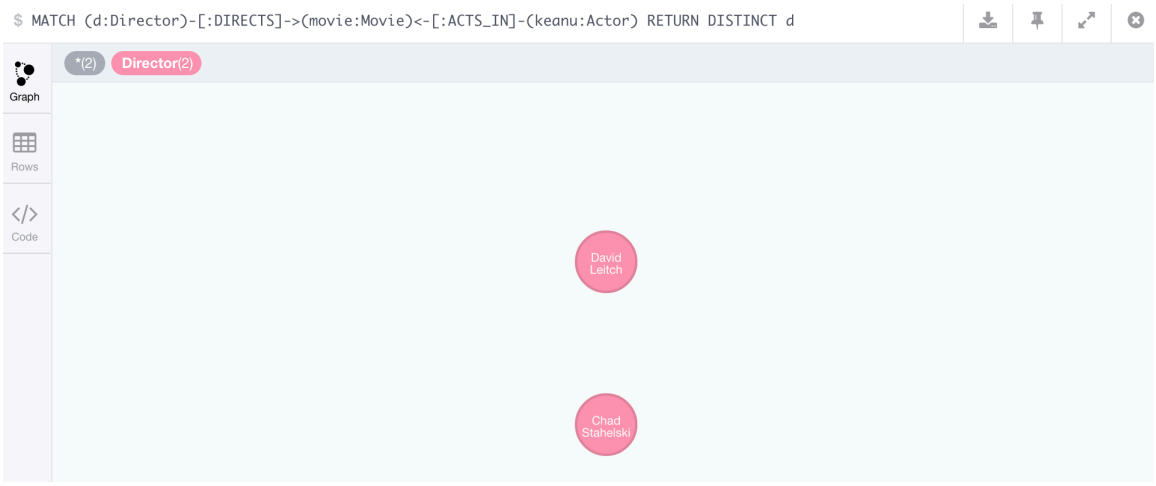
Results from Eclipse:

```
POST [{"statements" : [ {"statement" : "MATCH (a:Actor)-[:ACTS_IN]-
>(movie:Movie)<-[:ACTS_IN]-(keanu:Actor) RETURN DISTINCT a"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":["a"],"data":[{"row":[{"name":"Carrie-Anne
Moss"}]},{"row":[{"name":"Laurence Fishburne"}]},{"row":[{"name":"Keanu
Reeves"}]},{"row":[{"name":"William Dafoe"}]},{"row":[{"name":"Michael
Nyquist"}]}]}],"errors":[]}
POST [{"statements" : [ {"statement" : "MATCH (d:Director)-[:DIRECTS]-
>(movie:Movie)<-[:ACTS_IN]-(keanu:Actor) RETURN DISTINCT d"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":["d"],"data":[{"row":[{"name":"Chad
Stahelski"}]},{"row":[{"name":"David Leitch"}]}]}],"errors":[]}
```

2. Do it from Cyber Browser.

```
MATCH (a:Actor)-[:ACTS_IN]->(movie:Movie)<-[:ACTS_IN]-(keanu:Actor)
RETURN DISTINCT a

MATCH (d:Director)-[:DIRECTS]->(movie:Movie)<-[:ACTS_IN]-(keanu:Actor)
RETURN DISTINCT d
```





Problem 4. Find a way to export data from Neo4j into a set of CSV files. Delete your database and demonstrate that you can recreate it by loading those CSV files.

Solution:

1. Export data from Neo4j into a set of CSV files through cypher browser.

```
MATCH (a:Actor) return a.name as name
```

```
$ MATCH (a:Actor) return a.name as name
```

Rows / Code

name
Keanu Reeves
Laurence Fishburne
Carrie-Anne Moss
William Dafoe
Michael Nyquist

Save as “actors.csv”.

```
MATCH (m:Movie) return m.title as title, m.year as year
```

```
$ MATCH (m:Movie) return m.title as title, m.year as year
```

Rows / Code

title	year
The Matrix	1999-03-31
The Matrix Reloaded	2003-05-07
The Matrix Revolutions	2003-10-27
John Wick	2014-10-24

Save as “movies.csv”.

```
MATCH (d:Director) return d.name as name
```

```
$ MATCH (d:Director) return d.name as name
```

Rows / Code

name
Chad Stahelski
David Leitch

Save as “directors.csv”.

```
MATCH (a:Actor)-[r]->(m:Movie)
return a.name as name, type(r) as type, r.role as role, m.title as title
```

```
$ MATCH (a:Actor)-[r]->(m:Movie) return a.name as name, type(r) as type, r.role as role, m.title as tit...
```

Rows / Code

name	type	role	title
Carrie-Anne Moss	ACTS_IN	Trinity	The Matrix
Keanu Reeves	ACTS_IN	Neo	The Matrix
Laurence Fishburne	ACTS_IN	Morpheus	The Matrix
Laurence Fishburne	ACTS_IN	Morpheus	The Matrix Reloaded
Carrie-Anne Moss	ACTS_IN	Trinity	The Matrix Reloaded
Keanu Reeves	ACTS_IN	Neo	The Matrix Reloaded
Keanu Reeves	ACTS_IN	Neo	The Matrix Revolutions
Laurence Fishburne	ACTS_IN	Morpheus	The Matrix Revolutions
Carrie-Anne Moss	ACTS_IN	Trinity	The Matrix Revolutions
Keanu Reeves	ACTS_IN	John	John Wick
Michael Nyquist	ACTS_IN	Viggo	John Wick
William Dafoe	ACTS_IN	Marcus	John Wick

Returned 12 rows in 27 ms.

Save as “movie_actor_role.csv”.

```
MATCH (d:Director)-[r]->(m:Movie)
return d.name as name, type(r) as type, m.title as title
```

```
$ MATCH (d:Director)-[r]->(m:Movie) return d.name as name, type(r) as type, m.title as title
```

Rows / Code

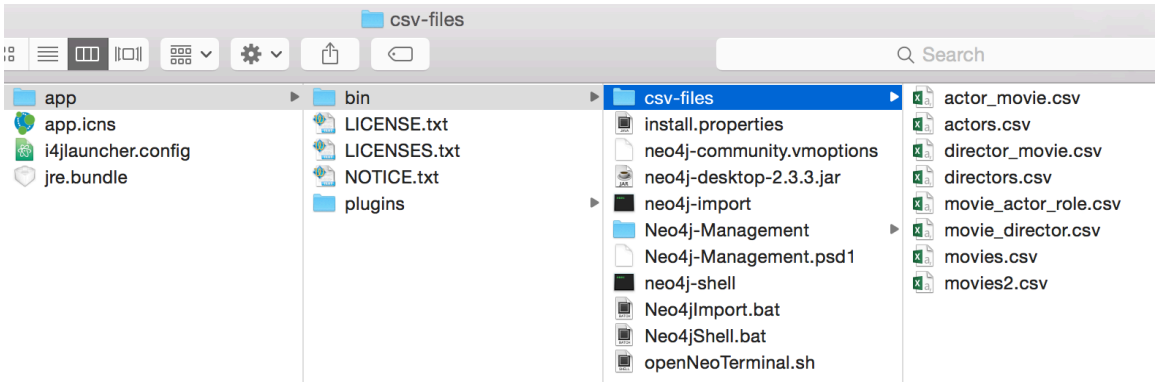
name	type	title
Chad Stahelski	DIRECTS	John Wick
David Leitch	DIRECTS	John Wick

Save as “movie_director.csv”.

Save file to directory “csv-files”.

Full directory:

/Applications/Neo4j\ Community\ Edition.app/Contents/Resources/app/bin/csv-files/



2. Recreate the database by loading the CSV files.

```
LOAD CSV WITH HEADERS FROM "file:///actors.csv" AS line
CREATE (a:Actor { name:line.name });

LOAD CSV WITH HEADERS FROM "file:///movies.csv" AS line
CREATE (m:Movie { title:line.title, year:line.year });

LOAD CSV WITH HEADERS FROM "file:///directors.csv" AS line
CREATE (d:Director { name:line.name });

LOAD CSV WITH HEADERS FROM "file:///movie_actor_role.csv" AS line
MATCH (a:Actor { name:line.name })
MATCH (m:Movie { title:line.title })
CREATE (a)-[:ACTS_IN { role: line.role }]->(m);

LOAD CSV WITH HEADERS FROM "file:///movie_director.csv" AS line
MATCH (d:Director { name:line.name })
MATCH (m:Movie { title:line.title })
CREATE (d)-[:DIRECTS]->(m);
```

```
$ LOAD CSV WITH HEADERS FROM "file:///movies.csv" AS line CREATE (m:Movie { title:line.title, year:line.y...
```

Rows

Added 4 labels, created 4 nodes, set 8 properties, statement executed in 123 ms.

Code

```
$ LOAD CSV WITH HEADERS FROM "file:///directors.csv" AS line CREATE (d:Director { name:line.name });
```

Rows

Added 2 labels, created 2 nodes, set 2 properties, statement executed in 125 ms.

Code

```
$ LOAD CSV WITH HEADERS FROM "file:///movie_actor_role.csv" AS line MATCH (a:Actor { name:line.name }) MA...
```

Rows

Set 12 properties, created 12 relationships, statement executed in 126 ms.

Code

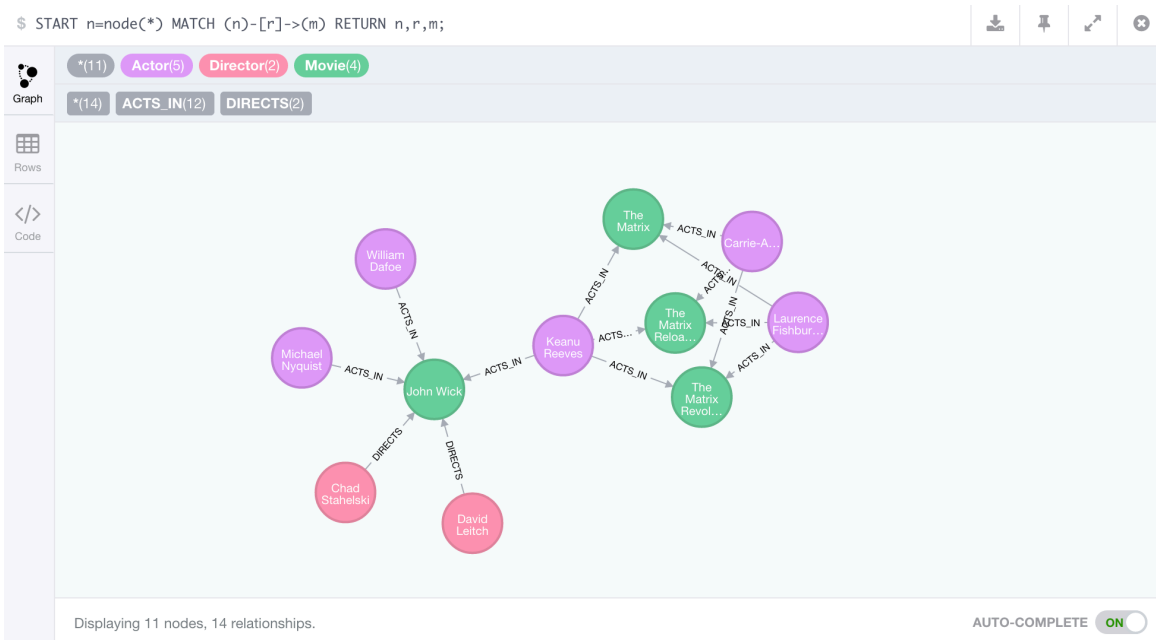
```
$ LOAD CSV WITH HEADERS FROM "file:///movie_director.csv" AS line MATCH (d:Director { name:line.name }) M...
```

Rows

Created 2 relationships, statement executed in 116 ms.

Code

Show all the nodes after recreation.



3. Build the database using REST Java API.

```
public class ConnectToServer2
{
   private static final String SERVER_ROOT_URI = "http://localhost:7474/db/data/";

   public static void main( String[] args ) throws URISyntaxException
   {
          ...........

          System.out.println("\nDelete the database and recreate by loading CSV files.\n");

          sendTransactionalCypherQuery ("MATCH (n) DETACH DELETE n");

          sendTransactionalCypherQuery ("START n=node(*) MATCH (n)-[r]->(m) RETURN n,r,m;");

          sendTransactionalCypherQuery ("LOAD CSV WITH HEADERS FROM 'file:///actors.csv' AS line "
                       + "CREATE (a:Actor { name:line.name });");

          sendTransactionalCypherQuery ("LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS line "
                       + "CREATE (m:Movie { title:line.title, year:line.year });");

          sendTransactionalCypherQuery ("LOAD CSV WITH HEADERS FROM 'file:///directors.csv' AS line
"
                       + "CREATE (d:Director { name:line.name });");

          sendTransactionalCypherQuery ("LOAD CSV WITH HEADERS FROM 'file:///movie_actor_role.csv'
AS line "
                       + "MATCH (a:Actor { name:line.name }) "
                       + "MATCH (m:Movie { title:line.title }) "
                       + "CREATE (a)-[:ACTS_IN { role: line.role }]->(m);");
```

```
        sendTransactionalCypherQuery ("LOAD CSV WITH HEADERS FROM 'file:///movie_director.csv' AS
line "
                        + "MATCH (d:Director { name:line.name }) "
                        + "MATCH (m:Movie { title:line.title }) "
                        + "CREATE (d)-[:DIRECTS]->(m);");

        sendTransactionalCypherQuery ("START n=node(*) MATCH (n)-[r]->(m) RETURN n,r,m;");
    }
```

Results from Eclipse:

```
Delete the database and recreate by loading CSV files.

POST [{"statements" : [ {"statement" : "MATCH (n) DETACH DELETE n"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "START n=node(*) MATCH (n)-[r]->(m)
RETURN n,r,m;"} ]}] to [http://localhost:7474/db/data/transaction/commit],
status code [200], returned data:
{"results":[{"columns":["n","r","m"],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "LOAD CSV WITH HEADERS FROM
'file:///actors.csv' AS line CREATE (a:Actor { name:line.name });"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "LOAD CSV WITH HEADERS FROM
'file:///movies.csv' AS line CREATE (m:Movie { title:line.title, year:line.year
});"} ]}] to [http://localhost:7474/db/data/transaction/commit], status code
[200], returned data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "LOAD CSV WITH HEADERS FROM
'file:///directors.csv' AS line CREATE (d:Director { name:line.name });"} ]}]
to [http://localhost:7474/db/data/transaction/commit], status code [200],
returned data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "LOAD CSV WITH HEADERS FROM
'file:///movie_actor_role.csv' AS line MATCH (a:Actor { name:line.name }) MATCH
(m:Movie { title:line.title }) CREATE (a)-[:ACTS_IN { role: line.role }]-
>(m);"} ]}] to [http://localhost:7474/db/data/transaction/commit], status code
[200], returned data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "LOAD CSV WITH HEADERS FROM
'file:///movie_director.csv' AS line MATCH (d:Director { name:line.name })
MATCH (m:Movie { title:line.title }) CREATE (d)-[:DIRECTS]->(m);"} ]}] to
[http://localhost:7474/db/data/transaction/commit], status code [200], returned
data:
{"results":[{"columns":[],"data":[]}],"errors":[]}
POST [{"statements" : [ {"statement" : "START n=node(*) MATCH (n)-[r]->(m)
RETURN n,r,m;"} ]}] to [http://localhost:7474/db/data/transaction/commit],
status code [200], returned data:
{"results":[{"columns":["n","r","m"],"data":[{"row":[{"name":"Keanu
```

Reeves"},{"role":"Neo"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Keanu Reeves"},{"role":"John"},{"title":"John Wick","year":"2014-10-24"}]},{"row":[{"name":"Keanu Reeves"},{"role":"Neo"},{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"name":"Keanu Reeves"},{"role":"Neo"},{"title":"The Matrix Revolutions","year":"2003-10-27"}]},{"row":[{"name":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"name":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix Revolutions","year":"2003-10-27"}]},{"row":[{"name":"Carrie-Anne Moss"},{"role":"Trinity"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Carrie-Anne Moss"},{"role":"Trinity"},{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"name":"Carrie-Anne Moss"},{"role":"Trinity"},{"title":"The Matrix Revolutions","year":"2003-10-27"}]},{"row":[{"name":"William Dafoe"},{"role":"Marcus"},{"title":"John Wick","year":"2014-10-24"}]},{"row":[{"name":"Michael Nyquist"},{"role":"Viggo"},{"title":"John Wick","year":"2014-10-24"}]},{"row":[{"name":"Chad Stahelski"},{},{"title":"John Wick","year":"2014-10-24"}]},{"row":[{"name":"David Leitch"},{},{"title":"John Wick","year":"2014-10-24"}]}]}],"errors":[]}