## HU Extension Assignment 10 E63 Big Data Analytics

### Handed out: 04/09/2016 Due by 11:30PM EST on Friday, 04/15/2016

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 ofMovies 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)

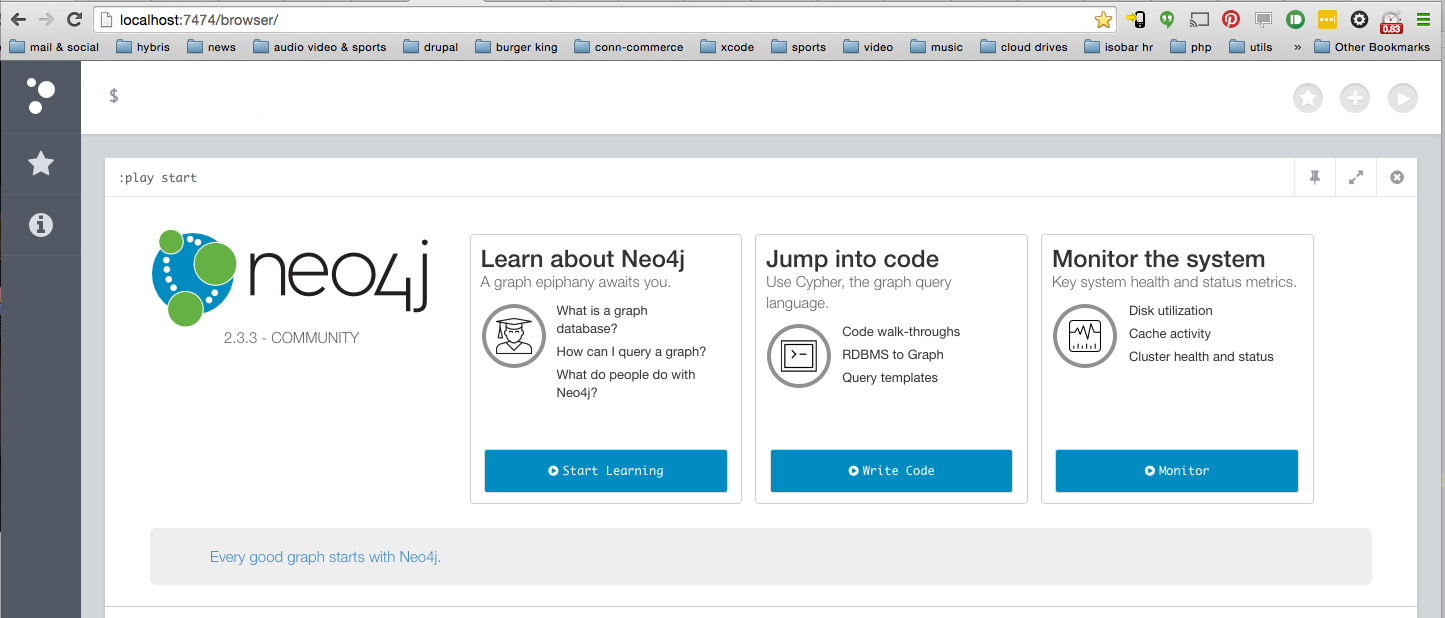
CREATE (laurence)-[:ACTS\_IN { role : 'Morpheus' }]->(matrix3)

CREATE (carrieanne)-[:ACTS\_IN { role : 'Trinity' }]->(matrix1)

CREATE (carrieanne)-[:ACTS\_IN { role : 'Trinity' }]->(matrix2)

CREATE (carrieanne)-[:ACTS\_IN { role : 'Trinity' }]->(matrix3)

I installed Neo4j and its running on <http://localhost:7474/browser/> as shown here:



These are the curl commands I used to achieve this:

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (matrix1:Movie {props})","parameters" : { "props" : {"title" : "The Matrix", "year" : "1999-03-31" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:52:17 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (matrix2:Movie {props})","parameters" : { "props" : {"title" : "The Matrix Reloaded", "year" : "2003-05-07" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:52:30 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (matrix3:Movie {props})","parameters" : { "props" : {"title" : "The Matrix Revolutions", "year" : "2003-10-27" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:52:42 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (movie:Movie) RETURN movie"}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:52:58 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 236

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":["movie"],"data":[{"row":[{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"title":"The Matrix Revolutions","year":"2003-10-27"}]}]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (keanu:Actor {props})","parameters" : { "props" : {"name" : "Keanu Reeves" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:53:13 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (laurence:Actor {props})","parameters" : { "props" : {"name" : "Laurence Fishburne" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:53:23 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (carrieanne:Actor {props})","parameters" : { "props" : {"name" : "Carrie-Anne Moss" }}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:53:33 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor) RETURN actor"}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:53:44 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 168

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":["actor"],"data":[{"row":[{"name":"Keanu Reeves"}]},{"row":[{"name":"Laurence Fishburne"}]},{"row":[{"name":"Carrie-Anne Moss"}]}]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Keanu Reeves", "movie\_title": "The Matrix", "relation\_props": {"role": "Neo"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:54:45 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Keanu Reeves", "movie\_title": "The Matrix Reloaded", "relation\_props": {"role": "Neo"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:54:57 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Keanu Reeves", "movie\_title": "The Matrix Revolutions", "relation\_props": {"role": "Neo"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:55:07 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Laurence Fishburne", "movie\_title": "The Matrix", "relation\_props": {"role": "Morpheus"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:55:17 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Laurence Fishburne", "movie\_title": "The Matrix Reloaded", "relation\_props": {"role": "Morpheus"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:55:28 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Laurence Fishburne", "movie\_title": "The Matrix Revolutions", "relation\_props": {"role": "Morpheus"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:55:42 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Carrie-Anne Moss", "movie\_title": "The Matrix", "relation\_props": {"role": "Trinity"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:55:54 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Carrie-Anne Moss", "movie\_title": "The Matrix Reloaded", "relation\_props": {"role": "Trinity"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:56:05 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor {name: {actor\_name}}),(movie:Movie {title: {movie\_title}}) CREATE (actor)-[r:ACTS\_IN {relation\_props}]-> (movie)", "parameters": {"actor\_name": "Carrie-Anne Moss", "movie\_title": "The Matrix Revolutions", "relation\_props": {"role": "Trinity"}}}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:56:16 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 50

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":[],"data":[]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (actor:Actor)-[r:ACTS\_IN]->(movie:Movie) return actor,r,movie"}]}'

HTTP/1.1 200 OK

Date: Fri, 15 Apr 2016 05:56:27 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 1016

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":["actor","r","movie"],"data":[{"row":[{"name":"Carrie-Anne Moss"},{"role":"Trinity"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Keanu Reeves"},{"role":"Neo"},{"title":"The Matrix","year":"1999-03-31"}]},{"row":[{"name":"Carrie-Anne Moss"},{"role":"Trinity"},{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"name":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix Reloaded","year":"2003-05-07"}]},{"row":[{"name":"Keanu Reeves"},{"role":"Neo"},{"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":"Laurence Fishburne"},{"role":"Morpheus"},{"title":"The Matrix Revolutions","year":"2003-10-27"}]},{"row":[{"name":"Keanu Reeves"},{"role":"Neo"},{"title":"The Matrix Revolutions","year":"2003-10-27"}]}]}],"errors":[]}rpulekar-m1:database\_exports rpulekar$

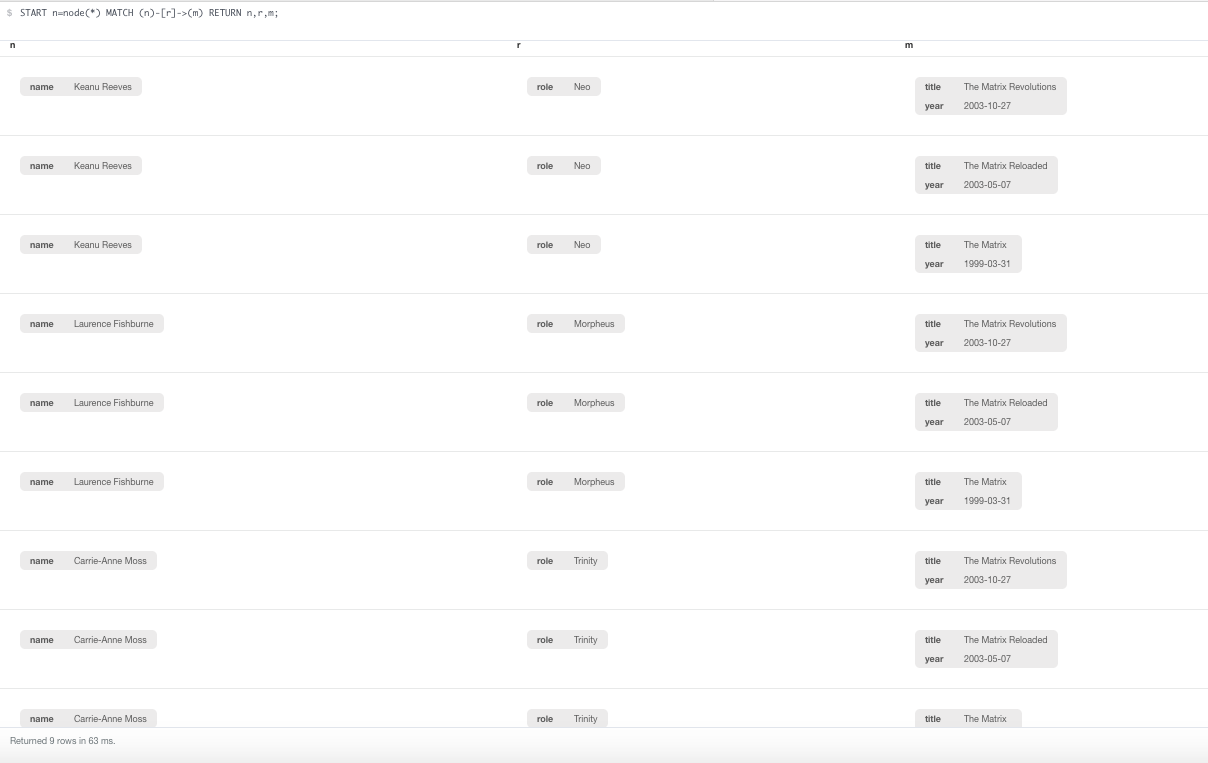
rpulekar-m1:database\_exports rpulekar$

rpulekar-m1:database\_exports rpulekar$

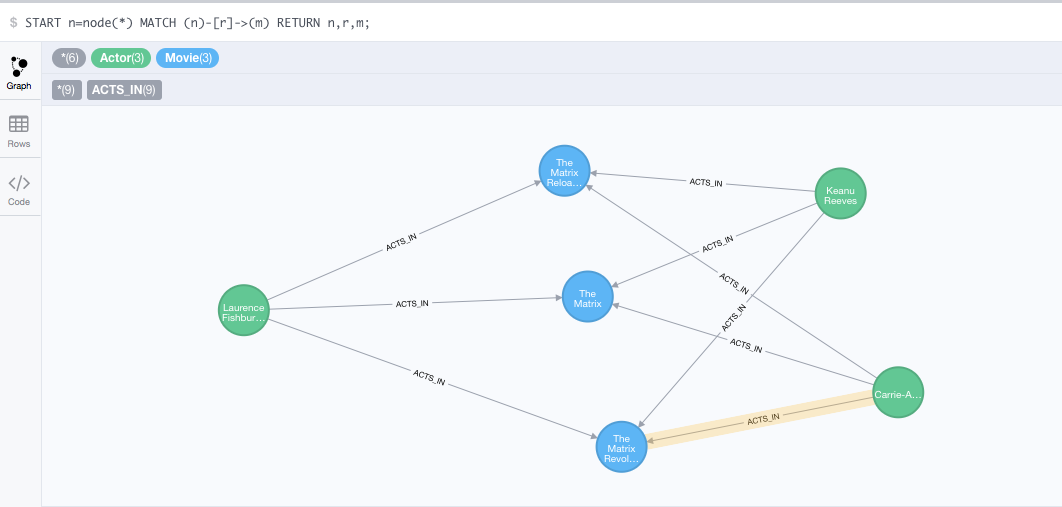
Then I checked the output in neo4j cypher browser with this command:

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

Here is the output:



And here is the graphical output:



Deliverables:

- problem1\_commands.doc (this contains all the curl commands used for Problem1)

**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.

For this I used Java REST API:

This is the code:

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

// check if database parameters are passed in as parameters

**if** (args.length >= 1) {

*DB\_SERVER\_ROOT\_URI* = args[0];

}

**if** (args.length >= 2) {

*DB\_SERVER\_USERNAME* = args[1];

}

**if** (args.length >= 3) {

*DB\_SERVER\_PASSWORD* = args[2];

}

// create web client

*webClient* = Client.*create*();

// add basic http authentication filter

*webClient*.addFilter(**new** HTTPBasicAuthFilter(*DB\_SERVER\_USERNAME*, *DB\_SERVER\_PASSWORD*));

// check if neo4j database is available

**boolean** isDatabaseAvailable = *checkIfDatabaseIsAvailable*();

**if** (!isDatabaseAvailable) {

System.***out***.println("Database not available. Will exit");

System.*exit*(1);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START OF PROBLEM 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* START OF PROBLEM 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// create node of type Movie for The Matrix movie and add properties to

// that node

URI movieTheMatrixURI = *createBlankNode*();

*setLabelOnNode*(movieTheMatrixURI, "Movie");

*addPropertyToNode*(movieTheMatrixURI, "title", "The Matrix");

*addPropertyToNode*(movieTheMatrixURI, "year", "1999-03-31");

// create node of type Movie for The Matrix Reloaded movie and add

// properties to that node

URI movieTheMatrixReloadedURI = *createBlankNode*();

*setLabelOnNode*(movieTheMatrixReloadedURI, "Movie");

*addPropertyToNode*(movieTheMatrixReloadedURI, "title", "The Matrix Reloaded");

*addPropertyToNode*(movieTheMatrixReloadedURI, "year", "2003-05-07");

// create node of type Movie for The Matrix Revolutions and add

// properties to that node

URI movieTheMatrixRevolutionsURI = *createBlankNode*();

*setLabelOnNode*(movieTheMatrixRevolutionsURI, "Movie");

*addPropertyToNode*(movieTheMatrixRevolutionsURI, "title", "The Matrix Revolutions");

*addPropertyToNode*(movieTheMatrixRevolutionsURI, "year", "2003-10-27");

// create node of type Actor for William Dafoe and add properties to

// that node

URI actorWilliamDafoeNodeURI = *createBlankNode*();

*setLabelOnNode*(actorWilliamDafoeNodeURI, "Actor");

*addPropertyToNode*(actorWilliamDafoeNodeURI, "name", "William Dafoe");

*addPropertyToNode*(actorWilliamDafoeNodeURI, "year\_born", "1955");

// create node of type Actor for Michael Nyquist and add properties to

// that node

URI actorMichaelNyquistURI = *createBlankNode*();

*setLabelOnNode*(actorMichaelNyquistURI, "Actor");

*addPropertyToNode*(actorMichaelNyquistURI, "name", "Michael Nyquist");

*addPropertyToNode*(actorMichaelNyquistURI, "year\_born", "1960");

// create node of type Actor for Keanu Reeves and add properties to that

// node

URI actorKeanuReevesURI = *createBlankNode*();

*setLabelOnNode*(actorKeanuReevesURI, "Actor");

*addPropertyToNode*(actorKeanuReevesURI, "name", "Keanu Reeves");

*addPropertyToNode*(actorKeanuReevesURI, "year\_born", "1964");

// create node of type Actor for Laurence Fishburne and add properties

// to that node

URI actorLaurenceFishburneURI = *createBlankNode*();

*setLabelOnNode*(actorLaurenceFishburneURI, "Actor");

*addPropertyToNode*(actorLaurenceFishburneURI, "name", "Laurence Fishburne");

*addPropertyToNode*(actorLaurenceFishburneURI, "year\_born", "1961");

// create node of type Actor for Carrie-Ann Moss and add properties to

// that node

URI actorCarrieAnneMossURI = *createBlankNode*();

*setLabelOnNode*(actorCarrieAnneMossURI, "Actor");

*addPropertyToNode*(actorCarrieAnneMossURI, "name", "Carrie-Anne Moss");

*addPropertyToNode*(actorCarrieAnneMossURI, "year\_born", "1967");

// create node of type Director for Chad Stahelski and add properties to

// that node

URI directorChadStahelskiURI = *createBlankNode*();

*setLabelOnNode*(directorChadStahelskiURI, "Director");

*addPropertyToNode*(directorChadStahelskiURI, "name", "Chad Stahelski");

*addPropertyToNode*(directorChadStahelskiURI, "year\_born", "1968");

// create node of type Director for David Leitch and add properties to

// that node

URI directorDavidLeitchURI = *createBlankNode*();

*setLabelOnNode*(directorDavidLeitchURI, "Director");

*addPropertyToNode*(directorDavidLeitchURI, "name", "David Leitch");

*addPropertyToNode*(directorDavidLeitchURI, "year\_born", "1969");

// create node of type Movie for John Wick and add properties to that

// node

URI movieJohnWickURI = *createBlankNode*();

*setLabelOnNode*(movieJohnWickURI, "Movie");

*addPropertyToNode*(movieJohnWickURI, "title", "John Wick");

*addPropertyToNode*(movieJohnWickURI, "year", "2014-10-24");

// create ACTS\_IN relation between Keanu Reeves and The Matrix

URI relation = *createRelationshipBetweenNodes*(actorKeanuReevesURI, movieTheMatrixURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Neo");

// create ACTS\_IN relation between Keanu Reeves and The Matrix Reloaded

relation = *createRelationshipBetweenNodes*(actorKeanuReevesURI, movieTheMatrixReloadedURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Neo");

// create ACTS\_IN realtion between Keanu Reeves and The Matrix

// Revolutions

relation = *createRelationshipBetweenNodes*(actorKeanuReevesURI, movieTheMatrixRevolutionsURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Neo");

// create ACTS\_IN relation between Carrie Anne Moss and The Matrix

relation = *createRelationshipBetweenNodes*(actorCarrieAnneMossURI, movieTheMatrixURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Trinity");

// create ACTS\_IN relation between Carrie Anne Moss and The Matrix

// Reloaded

relation = *createRelationshipBetweenNodes*(actorCarrieAnneMossURI, movieTheMatrixReloadedURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Trinity");

// create ACTS\_IN relation between Carrie Anne Moss and The Matrix

// Revolutions

relation = *createRelationshipBetweenNodes*(actorCarrieAnneMossURI, movieTheMatrixRevolutionsURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Trinity");

// create ACTS\_IN relation between Laurence Fishburne and The Matrix

relation = *createRelationshipBetweenNodes*(actorLaurenceFishburneURI, movieTheMatrixURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Morpheus");

// create ACTS\_IN relation between Laurence Fishburne and The Matrix

// Reloaded

relation = *createRelationshipBetweenNodes*(actorLaurenceFishburneURI, movieTheMatrixReloadedURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Morpheus");

// create ACTS\_IN relation between Laurence Fishburne and The Matrix

// Revolutions

relation = *createRelationshipBetweenNodes*(actorLaurenceFishburneURI, movieTheMatrixRevolutionsURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Morpheus");

// create ACTS\_IN relation between Keanu Reeves and John Wick

relation = *createRelationshipBetweenNodes*(actorKeanuReevesURI, movieJohnWickURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "John Wick");

// create ACTS\_IN relation between William Dafoe and John Wick

relation = *createRelationshipBetweenNodes*(actorWilliamDafoeNodeURI, movieJohnWickURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Marcus");

// create ACTS\_IN relation between Michael Nyquist and John Wick

relation = *createRelationshipBetweenNodes*(actorMichaelNyquistURI, movieJohnWickURI, "ACTS\_IN");

*addPropertyToRelationship*(relation, "role", "Viggo Tarasov");

// create DIRECTED relation between Chad Stahelski and John Wick

relation = *createRelationshipBetweenNodes*(directorChadStahelskiURI, movieJohnWickURI, "DIRECTED");

*addPropertyToRelationship*(relation, "shooting\_camera\_setup", "single camera");

// create DIRECTED relation between David Leitch and John Wick

relation = *createRelationshipBetweenNodes*(directorDavidLeitchURI, movieJohnWickURI, "DIRECTED");

*addPropertyToRelationship*(relation, "shooting\_camera\_setup", "multi camera");

}

/\*\*

\* THis method creates blank node. This function is part of Problem2

\* solution

\*

\* **@return** URI of the created node

\* **@throws** CustomException

\*/

**private** **static** URI createBlankNode() {

// web URI for node

**final** String nodeWebServiceURI = *DB\_SERVER\_ROOT\_URI* + "node";

// create web resource for node URI

WebResource webResource = *webClient*.resource(nodeWebServiceURI);

// make an empty web post to create a blank node

ClientResponse clientResponse = webResource.post(ClientResponse.**class**);

// check the web client response

**if** (clientResponse.getStatus() != 201) {

System.***out***.println("Unable to create actor node");

System.*exit*(1);

}

// display message and close the client response

System.***out***.println("Blank node created successfully");

clientResponse.close();

// return URI of the just created node

**return** clientResponse.getLocation();

}

/\*\*

\* Sets label to the given node. This function is part of Problem2 solution

\*

\*

\* **@param** nodeURI

\* **@param** label

\* **@return**

\* **@throws** CustomException

\*/

**private** **static** **void** setLabelOnNode(URI nodeURI, String label) {

// create label web service end point URI for given node

**final** String labelWebServiceURI = nodeURI.toString() + "/labels";

// create web resource for label web service URI

WebResource webResource = *webClient*.resource(labelWebServiceURI);

// post label name to label web service end point

ClientResponse clientResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.post(ClientResponse.**class**, "[\"" + label + "\" ]");

// check web service response status

**if** (clientResponse.getStatus() != 204) {

System.***out***.println("Unable to set label on node");

System.*exit*(1);

}

// display status message and close the client response

System.***out***.println("Label set on node");

clientResponse.close();

}

/\*\*

\* This function adds property to the given node. This function is part of

\* Problem2 solution

\*

\* **@param** nodeUri

\* **@param** propertyName

\* **@param** propertyValue

\*/

**private** **static** **void** addPropertyToNode(URI nodeUri, String propertyName, String propertyValue) {

// create property web service URI for given node

String propertyWebServiceURIForGivenNode = nodeUri.toString() + "/properties/" + propertyName;

// create web resource for the property web service URI

WebResource webResource = *webClient*.resource(propertyWebServiceURIForGivenNode);

// use web service put method and put a string in property web service

// URI. That string will be property value

ClientResponse clientResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity("\"" + propertyValue + "\"").put(ClientResponse.**class**);

// check the web service response

**if** (clientResponse.getStatus() != 204) {

System.***out***

.println("Unable to add properties to the node. received errorcode:" + clientResponse.getStatus());

System.*exit*(1);

}

// display success message and close the client response

System.***out***.println("Successfully set the property on the node");

// close the web service response

clientResponse.close();

}

/\*\*

\* This function creates relationship between nodes. This function is part

\* of Problem2 solution

\*

\* **@param** startNodeURI

\* **@param** endNodeURI

\* **@param** relationshipType

\* **@param** jsonAttributes

\* **@return**

\* **@throws** URISyntaxException

\*/

**private** **static** URI createRelationshipBetweenNodes(URI startNodeURI, URI endNodeURI, String relationshipType)

**throws** URISyntaxException {

// create URI for relationship web service end point of start node

URI startNodeRelationshipURI = **new** URI(startNodeURI.toString() + "/relationships");

// create json for relationship and end node

String jsonForRelationship = *createJsonForRelationship*(endNodeURI, relationshipType);

// create web resource for start node relationship URI

WebResource webResource = *webClient*.resource(startNodeRelationshipURI);

// POST json for relationship to the start node relationships URI

ClientResponse clientResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(jsonForRelationship).post(ClientResponse.**class**);

// check status of web service response

**if** (clientResponse.getStatus() != 201) {

System.***out***.println("Unable to create relation. received errorcode:" + clientResponse.getStatus());

System.*exit*(1);

}

// display success message

System.***out***.println("Successfully created relation");

// close the web service response

clientResponse.close();

// return relationship URI of the relationship just created

**return** clientResponse.getLocation();

}

/\*\*

\* This method creates json for the given relationship for given to node.

\* The json created is something like: { "to" :

\* "http://localhost:7474/db/data/node/244", "type" : "ACTS\_IN" }

\*

\* This function is part of Problem2 solution

\*

\* **@param** endNodeURI

\* **@param** relationshipType

\* **@return** String (json representation)

\*/

**private** **static** String createJsonForRelationship(URI endNodeURI, String relationshipType) {

StringBuilder jsonStringBuilder = **new** StringBuilder();

jsonStringBuilder.append("{ \"to\" : \"");

jsonStringBuilder.append(endNodeURI.toString());

jsonStringBuilder.append("\", ");

jsonStringBuilder.append("\"type\" : \"");

jsonStringBuilder.append(relationshipType);

jsonStringBuilder.append("\"");

jsonStringBuilder.append(" }");

**return** jsonStringBuilder.toString();

}

/\*\*

\* Create property for relationship. This function is part of Problem2

\* solution

\*

\* **@param** relationshipURI

\* **@param** propertyName

\* **@param** propertyValue

\* **@throws** URISyntaxException

\*/

**private** **static** **void** addPropertyToRelationship(URI relationshipURI, String propertyName, String propertyValue)

**throws** URISyntaxException {

// create web service URI for properties of the relationship

URI relationshipPropertyURI = **new** URI(relationshipURI.toString() + "/properties");

// create json string representation of name and value

String entity = *createJSONForNameAndValue*(propertyName, propertyValue);

// create web resource from relationship property web service URI

WebResource webResource = *webClient*.resource(relationshipPropertyURI);

// put json string for name and value to web resource

ClientResponse clientResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(entity).put(ClientResponse.**class**);

// check client response status

**if** (clientResponse.getStatus() != 204) {

System.***out***.println("Unable to add property to the relationship");

System.*exit*(1);

}

System.***out***.println("Successfully added property to the relationship");

// close the web service response

clientResponse.close();

}

/\*\*

\* Create json string for name and value.

\*

\* e.g. { "role" : "Neo" }

\*

\* This function is part of Problem2 solution

\*

\* **@param** name

\* **@param** value

\* **@return** String (json representation of name and value)

\*/

**private** **static** String createJSONForNameAndValue(String name, String value) {

**return** String.*format*("{ \"%s\" : \"%s\" }", name, value);

}

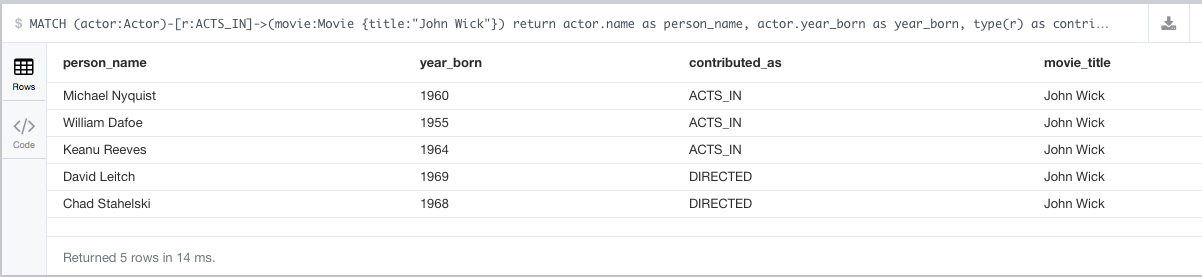
To demonstrate that I have successfully brought data about Jon Wick movie into the database, I gave this command in Cypher browser:

MATCH (actor:Actor)-[r:ACTS\_IN]->(movie:Movie {title:"John Wick"}) return actor.name as person\_name, actor.year\_born as year\_born, type(r) as contributed\_as, movie.title as movie\_title

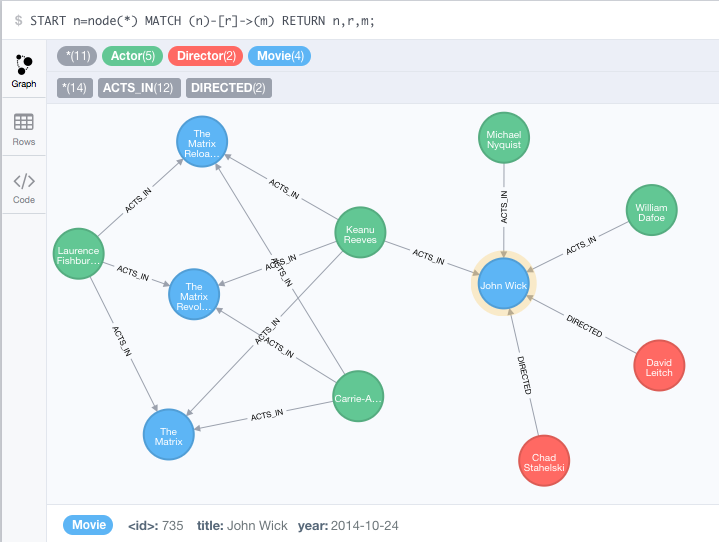
union

MATCH (director:Director)-[r:DIRECTED]->(movie:Movie {title:"John Wick"}) return director.name as person\_name, director.year\_born as year\_born, type(r) as contributed\_as, movie.title as movie\_title

This is the output I see:



And, this is the whole graph:



Deliverables:

- Problem2And3.java (this has the java code for problem2 and problem3). I have clearly marked in the comments as well as program output, which part is of which problem.

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

I have used 3 approaches to solve this problem:

Approach using JAVA API:

For this, the code is in file Problem2And3.java (which has java code for both problem 2 and 3)

In that java class, I have implemented two functions:

findOtherActorsInMoviesPlayedByKeanu(actorKeanuReevesURI)

findDirectorsOfMoviesPlayedByKeanu(actorKeanuReevesURI)

These are the functions that find actors and directors that have acted/directed in movies played by Keanu Reeves.

You can also find these functions in the assignment submission Problem2And3.java file:

**private** **static** **void** findOtherActorsInMoviesPlayedByKeanu(URI keanuReevesNodeURI) **throws** URISyntaxException {

// create traversal definition such that we can navigate movies in which

// given actor has acted

TraversalDefinition traversalDefinition = **new** TraversalDefinition();

traversalDefinition.setOrder(TraversalDefinition.***DEPTH\_FIRST***);

traversalDefinition.setUniqueness(TraversalDefinition.***NODE***);

traversalDefinition.setMaxDepth(10);

traversalDefinition.setReturnFilter(TraversalDefinition.***ALL***);

traversalDefinition.setRelationships(**new** Relation("ACTS\_IN", Relation.***OUT***));

// create web service URI for traversal from Keanu Reeves actor node

URI actorNodeTraverserURI = **new** URI(keanuReevesNodeURI.toString() + "/traverse/node");

// create resource for web service for traversal from actor node

WebResource webResource = *webClient*.resource(actorNodeTraverserURI);

// convert traverse definition to json string

String jsonTraverserPayload = traversalDefinition.toJson();

// make a post to get json response containing movies having ACTS\_IN

// relationship with this actor

ClientResponse webResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(jsonTraverserPayload).post(ClientResponse.**class**);

// validate the web service response

**if** (webResponse == **null** || webResponse.getStatus() != 200) {

System.***out***.println("Something went wrong while searching for movies acted In By Keanu");

System.*exit*(1);

}

// get the json response

String jsonResponseAsString = webResponse.getEntity(String.**class**);

// convert string json response to json obect

JSONArray jsonArrayOfMovies = **new** JSONArray(jsonResponseAsString);

**if** (jsonArrayOfMovies == **null** || jsonArrayOfMovies.length() == 0) {

System.***out***.println("No movies found for the given actor");

}

// create variable for final collection of actors that have acted in

// movies played by Keanu Reeves

Set<String> finalListOfActors = **new** HashSet<String>();

// loop through the movies played by Keanu Reeves

**for** (Object object : jsonArrayOfMovies) {

**if** (object == **null** || !(object **instanceof** JSONObject)) {

**continue**;

}

JSONObject jsonObjectOfMovie = (JSONObject) object;

// get the movie name and display it

String movieName = String.*valueOf*(((JSONObject) jsonObjectOfMovie.get("data")).get("title").toString());

System.***out***.println(String.*format*("Keanu Reeves acted in movie: %s", movieName));

// get the movie node URI string and create URI object out of it

String movieNodeURIString = String.*valueOf*(jsonObjectOfMovie.get("self"));

URI movieNodeURI = **new** URI(movieNodeURIString);

// call the below function to get actors who have acted in the movie

Set<String> actorsInTheMovie = *findActorsInAMovie*(movieNodeURI);

// remove Keanu Reeves onject from that list since we are lookign

// for actors other than Keanu Reeves

actorsInTheMovie.remove("Keanu Reeves");

// display the actor names

System.***out***.println(String.*format*("Other actors in that movie were: %s", actorsInTheMovie));

System.***out***.print(System.*lineSeparator*());

// add to the final list of actors, the actors list from current

// movie

finalListOfActors.addAll(actorsInTheMovie);

}

// display the full list of actors

System.***out***.print(String.*format*("So full list of actors who acted in movies in which Keanu Reeves played: %s",

finalListOfActors));

// close the response

webResponse.close();

}

/\*\*

\* This function returns a collection of actors who acted in the given

\* movie.

\*

\* This function is part of Problem3 solution

\*

\* **@param** movieNodeURI

\* **@return** Set<String> (collection of actors)

\* **@throws** URISyntaxException

\*/

**private** **static** Set<String> findActorsInAMovie(URI movieNodeURI) **throws** URISyntaxException {

// create variable for final collection of actors that have acted in the

// given movie

Set<String> listOfActors = **new** HashSet<String>();

// create traversal definition such that we can navigate actors who have

// acted in the given movie

TraversalDefinition traversalDefinition = **new** TraversalDefinition();

traversalDefinition.setOrder(TraversalDefinition.***DEPTH\_FIRST***);

traversalDefinition.setUniqueness(TraversalDefinition.***NODE***);

traversalDefinition.setMaxDepth(10);

traversalDefinition.setReturnFilter(TraversalDefinition.***ALL***);

traversalDefinition.setRelationships(**new** Relation("ACTS\_IN", Relation.***IN***));

// create web service URI for traversal from given movie node

URI movieNodeTraverserURI = **new** URI(movieNodeURI.toString() + "/traverse/node");

// create resource for web service for traversal from movie node

WebResource webResource = *webClient*.resource(movieNodeTraverserURI);

// convert traverse definition to json string

String jsonTraverserPayload = traversalDefinition.toJson();

// make a post to get json response containing actors having ACTS\_IN

// relationship with this movie

ClientResponse webResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(jsonTraverserPayload).post(ClientResponse.**class**);

// validate the web service response

**if** (webResponse == **null** || webResponse.getStatus() != 200) {

System.***out***.println("Something went wrong while searching for movies acted In By actor");

System.*exit*(1);

}

// get the json response

String jsonResponseAsString = webResponse.getEntity(String.**class**);

// convert string json response to json obect

JSONArray jsonArrayOfActors = **new** JSONArray(jsonResponseAsString);

**if** (jsonArrayOfActors == **null** || jsonArrayOfActors.length() == 0) {

**return** listOfActors;

}

// loop through the movies played by Keanu Reeves

**for** (Object object : jsonArrayOfActors) {

**if** (object == **null** || !(object **instanceof** JSONObject)) {

**continue**;

}

JSONObject jsonObjectOfActor = (JSONObject) object;

// get the actor name

String actorName = String.*valueOf*(((JSONObject) jsonObjectOfActor.get("data")).get("name"));

// add the actor name to list of actors

listOfActors.add(actorName);

}

// close the web service response

webResponse.close();

// return list of actors

**return** listOfActors;

}

/\*\*

\* This function finds movies acted in by Keanu and then finds directors who

\* directed those movies.

\*

\* This function is part of Problem3 solution

\*

\* **@param** keanuReevesNodeURI

\* **@throws** URISyntaxException

\*/

**private** **static** **void** findDirectorsOfMoviesPlayedByKeanu(URI keanuReevesNodeURI) **throws** URISyntaxException {

// create traversal definition such that we can navigate movies in which

// given actor has acted

TraversalDefinition traversalDefinition = **new** TraversalDefinition();

traversalDefinition.setOrder(TraversalDefinition.***DEPTH\_FIRST***);

traversalDefinition.setUniqueness(TraversalDefinition.***NODE***);

traversalDefinition.setMaxDepth(10);

traversalDefinition.setReturnFilter(TraversalDefinition.***ALL***);

traversalDefinition.setRelationships(**new** Relation("ACTS\_IN", Relation.***OUT***));

// create web service URI for traversal from Keanu Reeves actor node

URI actorNodeTraverserURI = **new** URI(keanuReevesNodeURI.toString() + "/traverse/node");

// create resource for web service for traversal from actor node

WebResource webResource = *webClient*.resource(actorNodeTraverserURI);

// convert traverse definition to json string

String jsonTraverserPayload = traversalDefinition.toJson();

// make a post to get json response containing movies having ACTS\_IN

// relationship with this actor

ClientResponse webResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(jsonTraverserPayload).post(ClientResponse.**class**);

// validate the web service response

**if** (webResponse == **null** || webResponse.getStatus() != 200) {

System.***out***.println("Something went wrong while searching for movies acted In By Keanu");

System.*exit*(1);

}

// get the json response

String jsonResponseAsString = webResponse.getEntity(String.**class**);

// convert string json response to json obect

JSONArray jsonArrayOfMovies = **new** JSONArray(jsonResponseAsString);

**if** (jsonArrayOfMovies == **null** || jsonArrayOfMovies.length() == 0) {

System.***out***.println("No movies found for the given actor");

}

// create variable for final collection of directors that have directed

// in

// movies played by Keanu Reeves

Set<String> finalListOfDirectors = **new** HashSet<String>();

// loop through the movies played by Keanu Reeves

**for** (Object object : jsonArrayOfMovies) {

**if** (object == **null** || !(object **instanceof** JSONObject)) {

**continue**;

}

JSONObject jsonObjectOfMovie = (JSONObject) object;

// get the movie name and display it

String movieName = String.*valueOf*(((JSONObject) jsonObjectOfMovie.get("data")).get("title").toString());

System.***out***.println(String.*format*("Keanu Reeves acted in movie: %s", movieName));

// get the movie node URI string and create URI object out of it

String movieNodeURIString = String.*valueOf*(jsonObjectOfMovie.get("self"));

URI movieNodeURI = **new** URI(movieNodeURIString);

// call the below function to get directors who have directed the

// movie

Set<String> directorsOfTheMovie = *findDirectorsOfTheMovie*(movieNodeURI);

// display the director names

**if** (!directorsOfTheMovie.isEmpty()) {

System.***out***.println(String.*format*("Directors in that movie were: %s", directorsOfTheMovie));

} **else** {

System.***out***.println("No directors found for this movie");

}

System.***out***.print(System.*lineSeparator*());

// add to the final list of directors, the directors list from

// current

// movie

finalListOfDirectors.addAll(directorsOfTheMovie);

}

// display the full list of actors

System.***out***.print(String.*format*("So full list of directors who directed movies in which Keanu Reeves played: %s",

finalListOfDirectors));

// close the response

webResponse.close();

}

/\*\*

\* This function returns a collection of directors who directed the given

\* movie.

\*

\* This function is part of Problem3 solution

\*

\* **@param** movieNodeURI

\* **@return** Set<String> (collection of directors)

\* **@throws** URISyntaxException

\*/

**private** **static** Set<String> findDirectorsOfTheMovie(URI movieNodeURI) **throws** URISyntaxException {

// create variable for final collection of directors that have directed

// the

// given movie

Set<String> listOfDirectors = **new** HashSet<String>();

// create traversal definition such that we can navigate directors who

// have

// directed in the given movie

TraversalDefinition traversalDefinition = **new** TraversalDefinition();

traversalDefinition.setOrder(TraversalDefinition.***DEPTH\_FIRST***);

traversalDefinition.setUniqueness(TraversalDefinition.***NODE***);

traversalDefinition.setMaxDepth(10);

traversalDefinition.setReturnFilter(TraversalDefinition.***ALL***);

traversalDefinition.setRelationships(**new** Relation("DIRECTED", Relation.***IN***));

// create web service URI for traversal from given movie node

URI movieNodeTraverserURI = **new** URI(movieNodeURI.toString() + "/traverse/node");

// create resource for web service for traversal from movie node

WebResource webResource = *webClient*.resource(movieNodeTraverserURI);

// convert traverse definition to json string

String jsonTraverserPayload = traversalDefinition.toJson();

// make a post to get json response containing directors having ACTS\_IN

// relationship with this movie

ClientResponse webResponse = webResource.accept(MediaType.***APPLICATION\_JSON***).type(MediaType.***APPLICATION\_JSON***)

.entity(jsonTraverserPayload).post(ClientResponse.**class**);

// validate the web service response

**if** (webResponse == **null** || webResponse.getStatus() != 200) {

System.***out***.println("Something went wrong while searching for movies acted In By director");

System.*exit*(1);

}

// get the json response

String jsonResponseAsString = webResponse.getEntity(String.**class**);

// convert string json response to json obect

JSONArray jsonArrayOfDirectors = **new** JSONArray(jsonResponseAsString);

**if** (jsonArrayOfDirectors == **null** || jsonArrayOfDirectors.length() == 0) {

**return** listOfDirectors;

}

// loop through the movies played by Keanu Reeves

**for** (Object object : jsonArrayOfDirectors) {

**if** (object == **null** || !(object **instanceof** JSONObject)) {

**continue**;

}

JSONObject jsonObjectOfDirector = (JSONObject) object;

// get the director name

String directorName = String.*valueOf*(((JSONObject) jsonObjectOfDirector.get("data")).get("name"));

// add the director name to list of directors

listOfDirectors.add(directorName);

}

// close the web service response

webResponse.close();

// return list of directors

**return** listOfDirectors;

}

Output of this program is:

Keanu Reeves acted in movie: John Wick

Other actors in that movie were: [Michael Nyquist, William Dafoe]

Keanu Reeves acted in movie: The Matrix Revolutions

Other actors in that movie were: [Carrie-Anne Moss, Laurence Fishburne]

Keanu Reeves acted in movie: The Matrix Reloaded

Other actors in that movie were: [Carrie-Anne Moss, Laurence Fishburne]

Keanu Reeves acted in movie: The Matrix

Other actors in that movie were: [Carrie-Anne Moss, Laurence Fishburne]

So full list of actors who acted in movies in which Keanu Reeves played: [Michael Nyquist, Carrie-Anne Moss, Laurence Fishburne, William Dafoe]

Keanu Reeves acted in movie: John Wick

Directors in that movie were: [Chad Stahelski, David Leitch]

Keanu Reeves acted in movie: The Matrix Revolutions

No directors found for this movie

Keanu Reeves acted in movie: The Matrix Reloaded

No directors found for this movie

Keanu Reeves acted in movie: The Matrix

No directors found for this movie

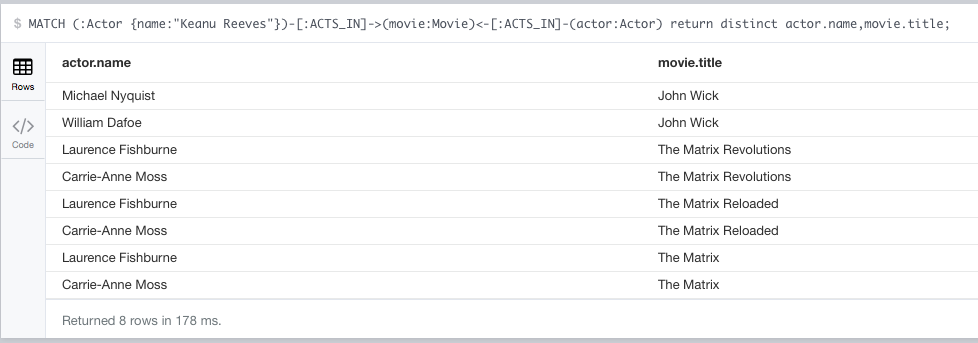
So full list of directors who directed movies in which Keanu Reeves played: [Chad Stahelski, David Leitch]

Approach using Cypher browser commands:

To get actors who have acted in movies played by Keanu Reeves:

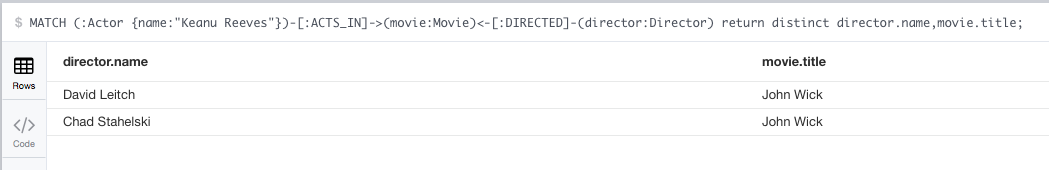
MATCH (:Actor {name:"Keanu Reeves"})-[:ACTS\_IN]->(movie:Movie)<-[:ACTS\_IN]-(actor:Actor) return distinct actor.name,movie.title;

Output:



To get list of directors who directed movies played by Keanu Reeves:

MATCH (:Actor {name:"Keanu Reeves"})-[:ACTS\_IN]->(movie:Movie)<-[:DIRECTED]-(director:Director) return distinct director.name,movie.title;



Approach using curl:

To get actors who have acted in movies played by Keanu Reeves:

rpulekar-m1:eclipse\_workspace\_for\_course rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (:Actor {name:{actor\_name}})-[:ACTS\_IN]->(movie:Movie)<-[:ACTS\_IN]-(actor:Actor) return distinct actor.name,movie.title;", "parameters": {"actor\_name": "Keanu Reeves"}}]}'

HTTP/1.1 200 OK

Date: Sat, 16 Apr 2016 01:29:24 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 453

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":["actor.name","movie.title"],"data":[{"row":["Michael Nyquist","John Wick"]},{"row":["William Dafoe","John Wick"]},{"row":["Laurence Fishburne","The Matrix Revolutions"]},{"row":["Carrie-Anne Moss","The Matrix Revolutions"]},{"row":["Laurence Fishburne","The Matrix Reloaded"]},{"row":["Carrie-Anne Moss","The Matrix Reloaded"]},{"row":["Laurence Fishburne","The Matrix"]},{"row":["Carrie-Anne Moss","The Matrix"]}]}],"errors":[]}rpulekar-m1:eclipse\_workspace\_for\_course rpulekar$

Find directors of movies in which K. Reeves played using curl command:

rpulekar-m1:eclipse\_workspace\_for\_course rpulekar$ curl -i -H accept:application/json -H content-type:application/json -XPOST --user neo4j:Elcapitan1011 http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (:Actor {name:{actor\_name}})-[:ACTS\_IN]->(movie:Movie)<-[:DIRECTED]-(director:Director) return distinct director.name,movie.title;", "parameters": {"actor\_name": "Keanu Reeves"}}]}'

HTTP/1.1 200 OK

Date: Sat, 16 Apr 2016 01:30:03 GMT

Content-Type: application/json

Access-Control-Allow-Origin: \*

Content-Length: 154

Server: Jetty(9.2.z-SNAPSHOT)

{"results":[{"columns":["director.name","movie.title"],"data":[{"row":["David Leitch","John Wick"]},{"row":["Chad Stahelski","John Wick"]}]}],"errors":[]}rpulekar-m1:eclipse\_workspace\_for\_course rpulekar$

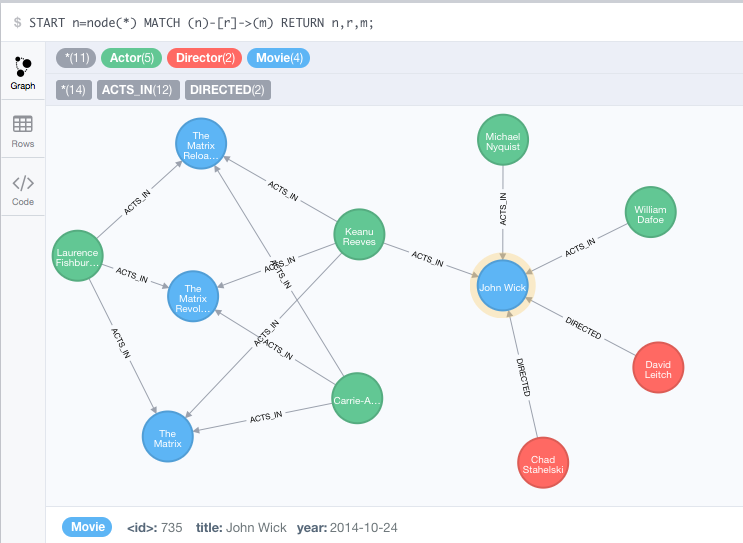
Deliverables:

* problem3\_commands.doc (this contains the cypher browser commands and the curl commands for problem3)
* Problem2And3.java (this Java class file contains solution for Problem2 and Problem3. I have clearly marked in the comments as well as program output, which part is of which problem).

**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.

Current state of my database:

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



To export all actors:

I gave this command in Cypher browser:

MATCH (actor:Actor) return actor.name as name,actor.year\_born as year\_born;

And I used Export CSV option of Cypher and exported this to

export\_actor.csv

name,year\_born

William Dafoe,1955

Michael Nyquist,1960

Keanu Reeves,1964

Laurence Fishburne,1961

Carrie-Anne Moss,1967

Then I gave this command in Cypher browser:

MATCH (director:Director) return director.name as name,director.year\_born as year\_born;

And I used Export CSV option of Cypher and exported this to

export\_director.csv:

name,year\_born

Chad Stahelski,1968

David Leitch,1969

Then I gave this command in Cypher browser:

MATCH (movie:Movie) return movie.title as title,movie.year as year;

And I used Export CSV option of Cypher and exported this to

export\_movie.csv

title,year

The Matrix,1999-03-31

The Matrix Reloaded,2003-05-07

The Matrix Revolutions,2003-10-27

John Wick,2014-10-24

Then I gave this command in Cypher browser:

MATCH (actor:Actor)-[r:ACTS\_IN]->(movie:Movie) return actor.name as actor\_name, r.role as role, movie.title as movie\_title;

export them to export\_acts\_in\_relationship.csv

And I used Export CSV option of Cypher and exported this to

export\_acts\_in\_relationship.csv:

actor\_name,role,movie\_title

Laurence Fishburne,Morpheus,The Matrix

Carrie-Anne Moss,Trinity,The Matrix

Keanu Reeves,Neo,The Matrix

Laurence Fishburne,Morpheus,The Matrix Reloaded

Carrie-Anne Moss,Trinity,The Matrix Reloaded

Keanu Reeves,Neo,The Matrix Reloaded

Laurence Fishburne,Morpheus,The Matrix Revolutions

Carrie-Anne Moss,Trinity,The Matrix Revolutions

Keanu Reeves,Neo,The Matrix Revolutions

Michael Nyquist,Viggo Tarasov,John Wick

William Dafoe,Marcus,John Wick

Keanu Reeves,John Wick,John Wick

Then I gave this command in Cypher browser:

MATCH (director:Director)-[r:DIRECTED]->(movie:Movie) return director.name as director\_name, r.shooting\_camera\_setup as shooting\_camera\_setup,movie.title as movie\_title;

And I used Export CSV option of Cypher and exported this to

export\_directed\_relationship.csv:

director\_name,shooting\_camera\_setup,movie\_title

Chad Stahelski,single camera,John Wick

David Leitch,multi camera,John Wick

Then I deleted the entire database with:

MATCH (n)

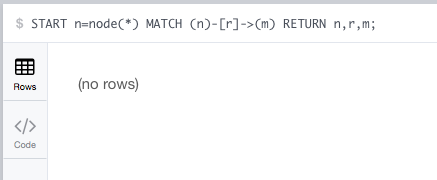
OPTIONAL MATCH (n)-[r]-()

DELETE n,r

In Cypher browser.

To make sure the database is deleted:

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



Then I gave these commands in Cypher browser to import the csv files:

LOAD CSV WITH HEADERS FROM "file:///Users/rpulekar/work/big-data-analytics-harvard/lectures\_and\_assignments/week-10-apr-8-2016/assignment/solution/database\_exports/export\_movie.csv" AS line MERGE (movie:Movie { title:line.title}) ON CREATE SET movie.year=line.year;

Added 4 labels, created 4 nodes, set 8 properties, statement executed in 148 ms.

LOAD CSV WITH HEADERS FROM "file:///Users/rpulekar/work/big-data-analytics-harvard/lectures\_and\_assignments/week-10-apr-8-2016/assignment/solution/database\_exports/export\_actor.csv" AS line MERGE (a:Actor { name:line.name, year\_born:line.year\_born}) ON CREATE SET a.year\_born=line.year\_born;

Added 5 labels, created 5 nodes, set 15 properties, statement executed in 392 ms.

LOAD CSV WITH HEADERS FROM "file:///Users/rpulekar/work/big-data-analytics-harvard/lectures\_and\_assignments/week-10-apr-8-2016/assignment/solution/database\_exports/export\_director.csv" AS line MERGE (d:Director { name:line.name, year\_born:line.year\_born}) ON CREATE SET d.year\_born=line.year\_born;

Added 2 labels, created 2 nodes, set 6 properties, statement executed in 7691 ms.

LOAD CSV WITH HEADERS FROM "file:///Users/rpulekar/work/big-data-analytics-harvard/lectures\_and\_assignments/week-10-apr-8-2016/assignment/solution/database\_exports/export\_acts\_in\_relationship.csv" AS line FIELDTERMINATOR "," MERGE (m:Movie { title:line.movie\_title }) MERGE (a:Actor { name:line.actor\_name }) MERGE (a)-[:ACTS\_IN { role:line.role}]->(m);

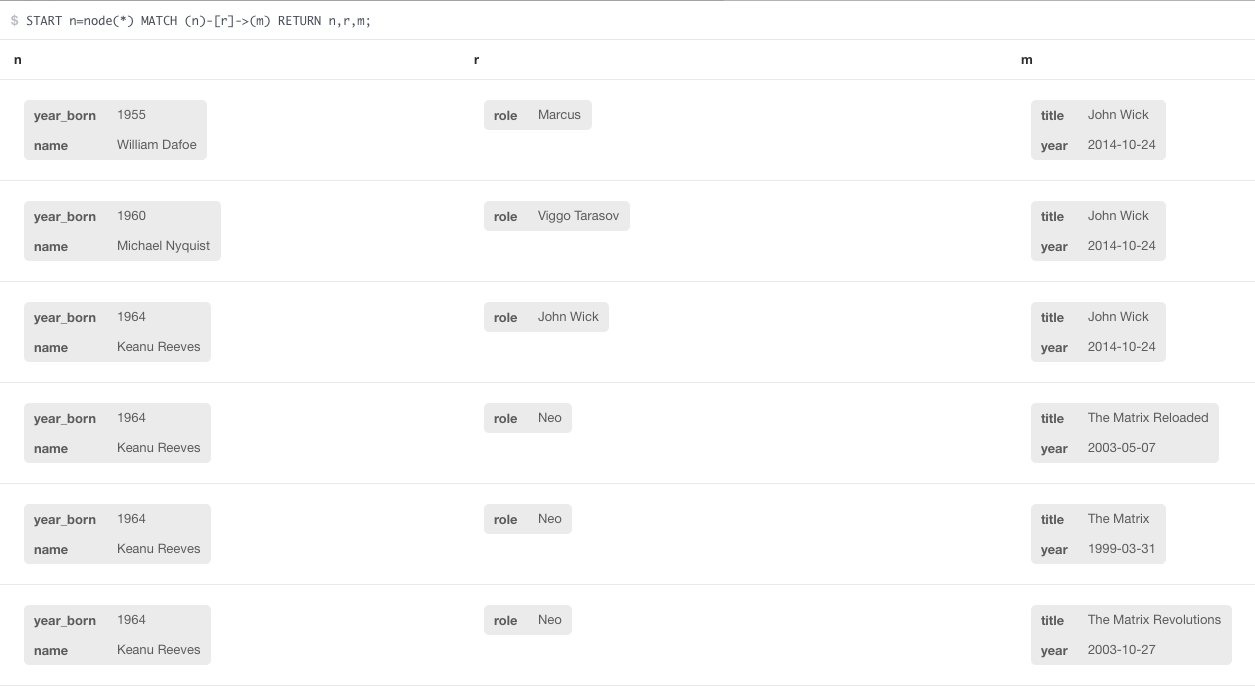
Set 12 properties, created 12 relationships, statement executed in 7513 ms.

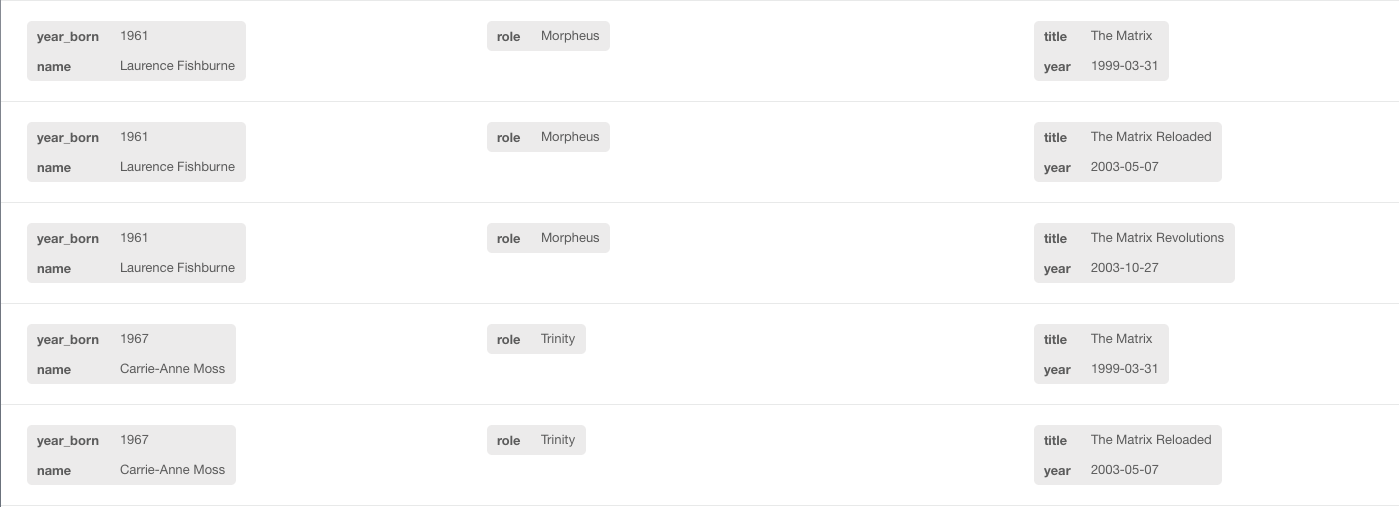
LOAD CSV WITH HEADERS FROM "file:///Users/rpulekar/work/big-data-analytics-harvard/lectures\_and\_assignments/week-10-apr-8-2016/assignment/solution/database\_exports/export\_directed\_relationship.csv" AS line FIELDTERMINATOR "," MERGE (m:Movie { title:line.movie\_title }) MERGE (d:Director { name:line.director\_name }) MERGE (d)-[:DIRECTED { shooting\_camera\_setup:line.shooting\_camera\_setup}]->(m);

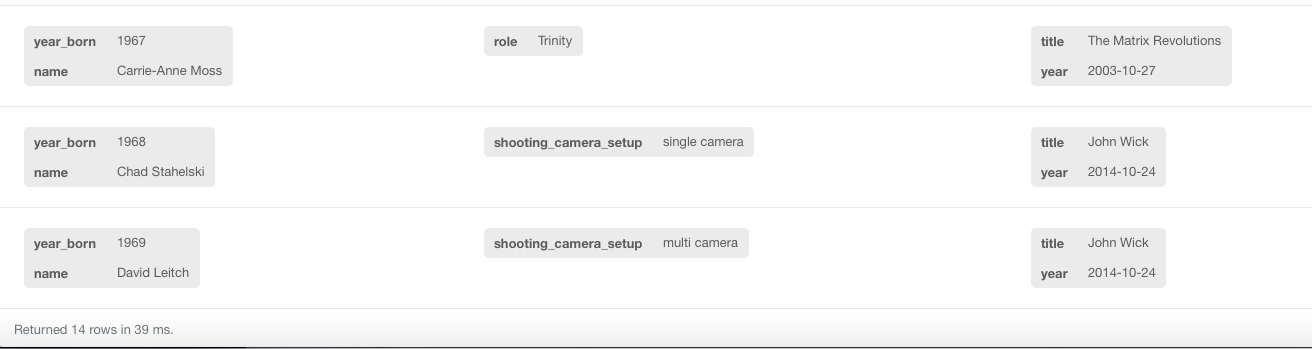
Set 2 properties, created 2 relationships, statement executed in 7368 ms.

To make sure that all data is imported, I gave this command in Cypher browser:

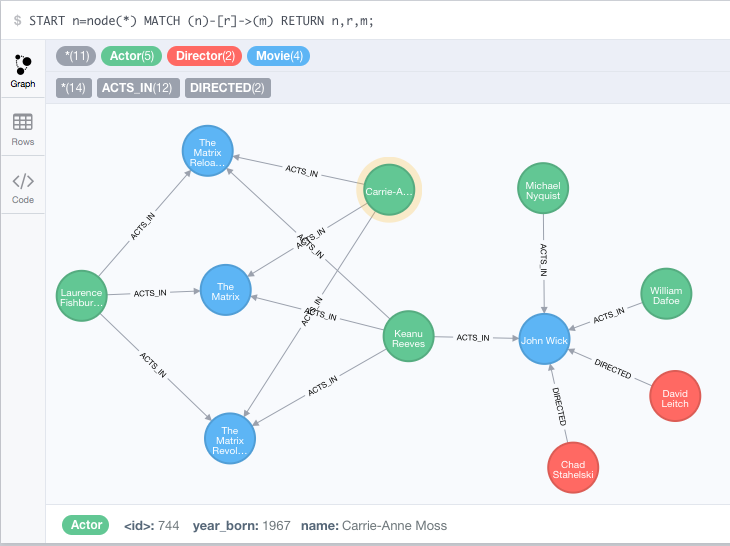
START n=node(\*) MATCH (n)-[r]->(m) RETURN n,r,m;







And this is the graph output:



Deliverables:

* problem4\_commands.doc (this file contains cypher commands for problem4)
* export\_actor.csv (export of actors)
* export\_director.csv (export of directors)
* export\_movie.csv (export of movies)
* export\_acts\_in\_relationship.csv (export of ACTS\_IN relationship)
* export\_directed\_relationship.csv (export of DIRECTED relationship)