**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' }) return id(matrix 1)

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

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

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

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)

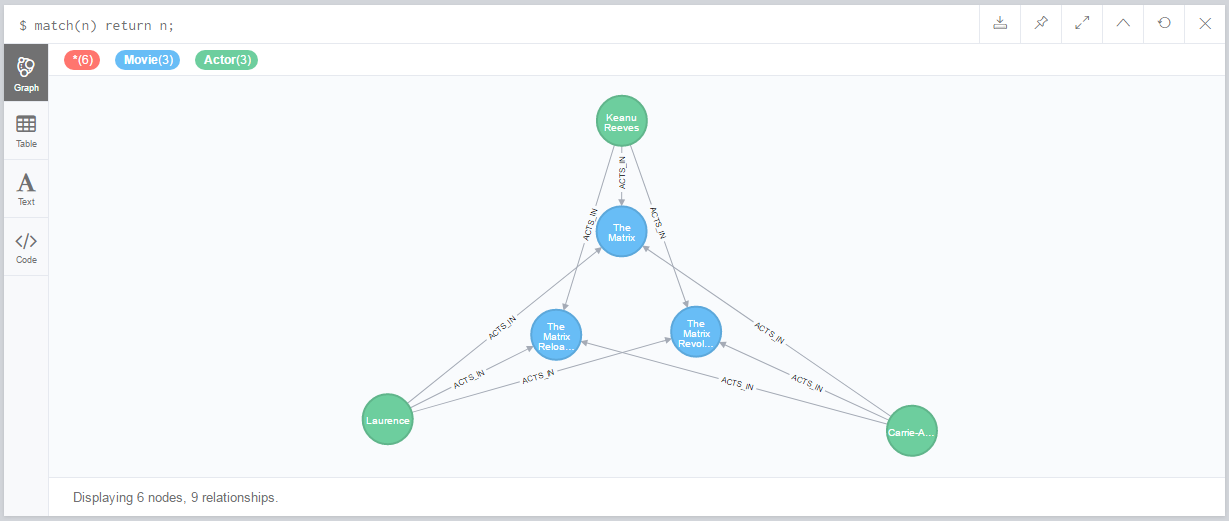
CREATE (carrieanne)-[:ACTS\_IN { role : 'Trinity' }]->(matrix2)

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

You might want to create a bash script which contains curl command with previous Cypher request and run that script from Cygwin or Linux (Unix) prompt. Following notes could be helpful:

1. you might want to use notepad++ to create and edit the bash file, and go to edit --> EOL conversion --> Unix.
2. Specify the user name and password like “-user neo4j:neo4jneo4j" before the local host url.
3. Add one space and one \ at each point you want to have a line break.
4. In the string of query, every double quote " should be changed to \".

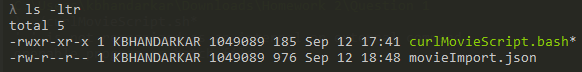
**Answer:**

****

**Solution:**

For this problem, I’ve used a bash script that contains a curl command. The curl command in turn uses a JSON file with create commands in Cypher Query Language (CQL).

* Files used:



* Bash Script (curlMovieScript.bash):

#!/bin/sh

curl -i -H accept:application/json -H content-type:application/json --user neo4j:M@nchesterUn7ted -XPOST http://localhost:7474/db/data/transaction/commit -d @movieImport.json

* JSON File (movieImport.json):

{

"statements": [

{

"statement": "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)"

}

]

}

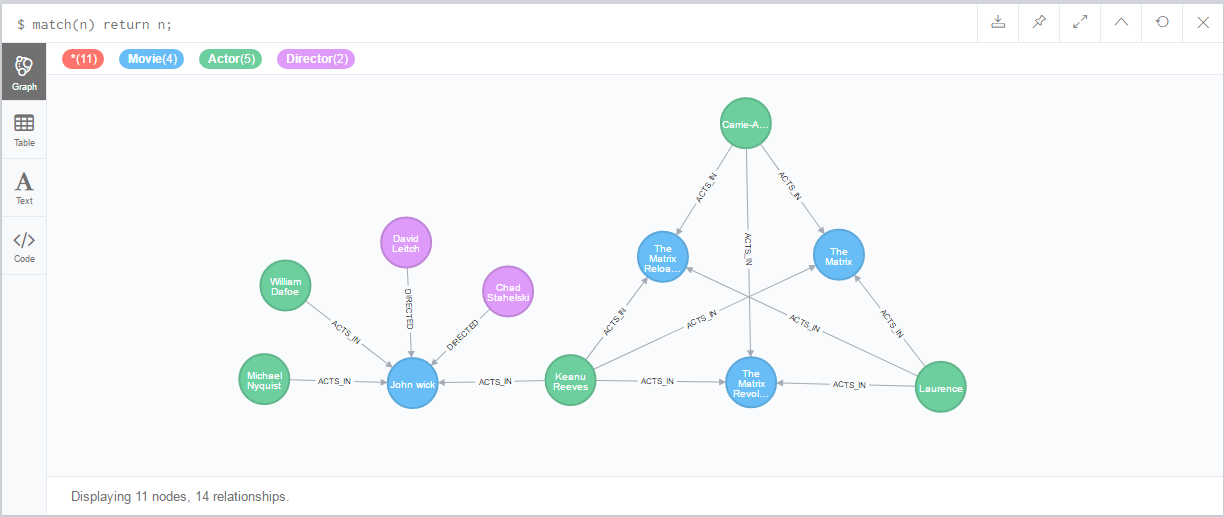
* Changes made to provided statements:

The create statements need to be run as a batch command in the same session in order to use the IDs created for relationship mapping. For this purpose, return statements need to be removed. An alternative could have been to use the IDs created in subsequent queries.

**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. Demonstrate that you have successfully brought data about John Wick movie into the database. You can use Cypher Browser or any other means. Delete above movie and all the cast except Keanu Reeves.

**Answer:**



* **CQL**

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

CREATE (johnwick:Movie { title : 'John wick'})

CREATE (:Director { name:'David Leitch' })-[:DIRECTED]-> (johnwick)

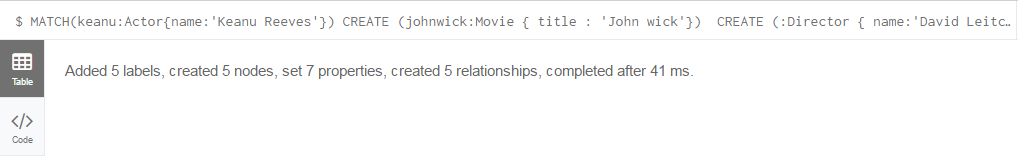
CREATE (:Director { name:'Chad Stahelski' })-[:DIRECTED]->(johnwick)

CREATE (keanu)-[:ACTS\_IN{role:'John Wick'}]->(johnwick)

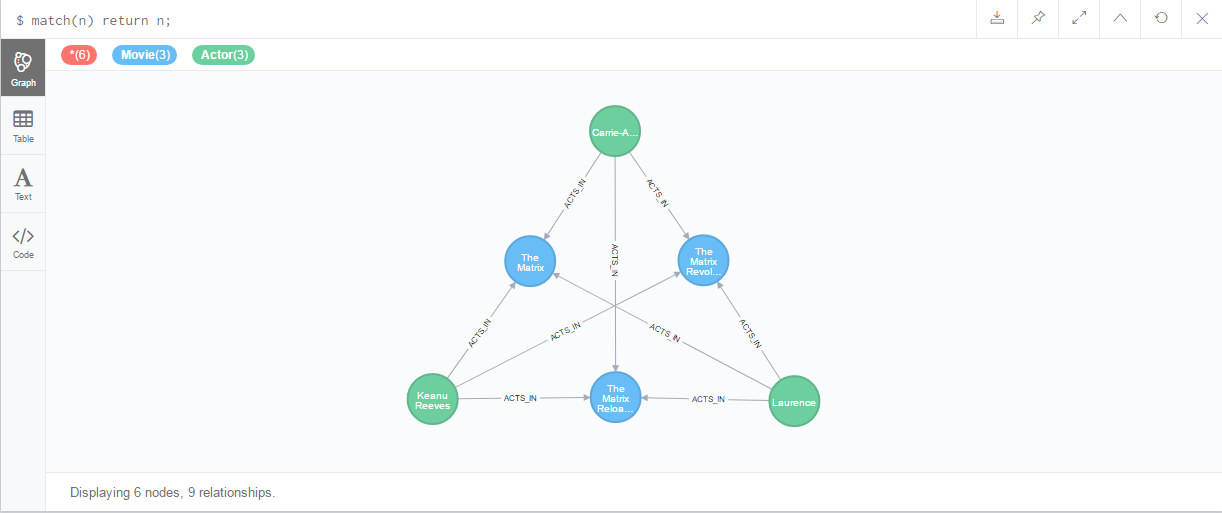
CREATE (:Actor { name:'William Dafoe' })-[:ACTS\_IN {role:'Marcus'} ]->(johnwick)

CREATE (:Actor { name:'Michael Nyquist' })-[:ACTS\_IN {role:'Viggo Tarasov'}]->(johnwick)

* **OUTPUT**

****

**Answer:**

****

* **CQL**

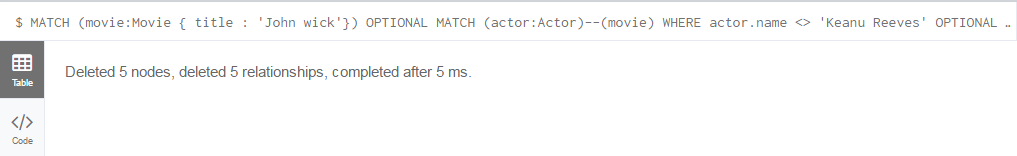
MATCH (movie:Movie { title : 'John wick'})

OPTIONAL MATCH (actor:Actor)--(movie) WHERE actor.name <> 'Keanu Reeves'

OPTIONAL MATCH (director:Director)--(movie)

DETACH DELETE actor, director, movie

* **OUTPUT**

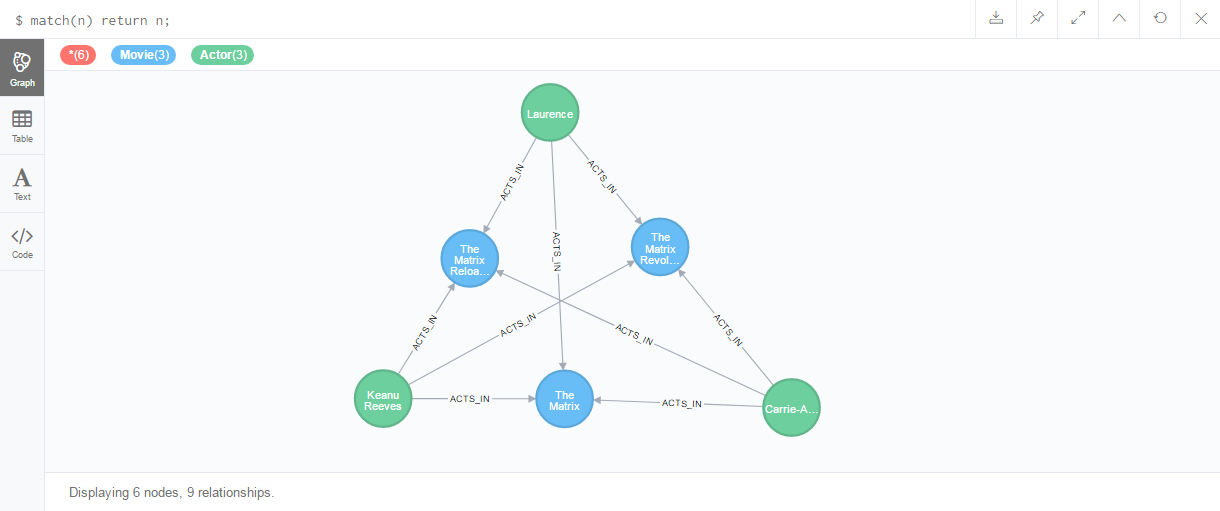


**Problem 3:**

Addall the actors and the roles they played in this movie “John Wick” to the database using JAVA REST API or some other APIs for Neo4J in a language of your choice (not Curl). Demonstrate that you have successfully brought data about John Wick movie into the database. You can use Cypher Browser or any other means.

**Answer:**

* Pre-execution Database State:



* Addall the actors and the roles they played in this movie “John Wick” to the database using JAVA REST API:

The standalone Neo4j Server can be installed on any machine and then accessed via its **binary Bolt protocol** through the official Neo4j Java driver.

Add dependency in pom as (pom.xml file submitted):

<dependency>

<groupId>org.neo4j.driver</groupId>

<artifactId>neo4j-java-driver</artifactId>

<version>1.0.3</version>

</dependency>

Setup validation response string as:

String responseString = " MATCH (actor:Actor)-[r:ACTS\_IN]->(movie:Movie { title : 'John Wick'}) " + "return movie.title as Movie, actor.name as Actor, r.role as Role " ;

Setup query string as:

String queryString = "MATCH(keanu:Actor{name:'Keanu Reeves'}) "+

"CREATE (johnwick:Movie { title : 'John Wick'}) " +

"CREATE (keanu)-[:ACTS\_IN { role : 'John Wick' }]->(johnwick) "+

"CREATE (:Actor { name:'Michael Nyqvist' })-[:ACTS\_IN { role : 'Viggo Tarasov' }]->(johnwick) " +

"CREATE (:Actor { name:'Alfie Allen' })-[:ACTS\_IN { role : 'Iosef Tarasov' }]->(johnwick) ";

Code Snippet (Class file submitted):

driver = GraphDatabase.*driver*( "bolt://localhost", AuthTokens.*basic*( "neo4j",\*" ));

session = driver.session();

session.run(queryString);

StatementResult result = session.run(responseString);

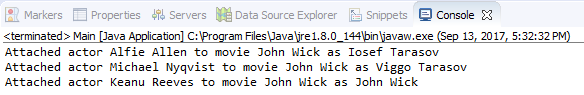
**while** (result.hasNext()) {

Record record = result.next();

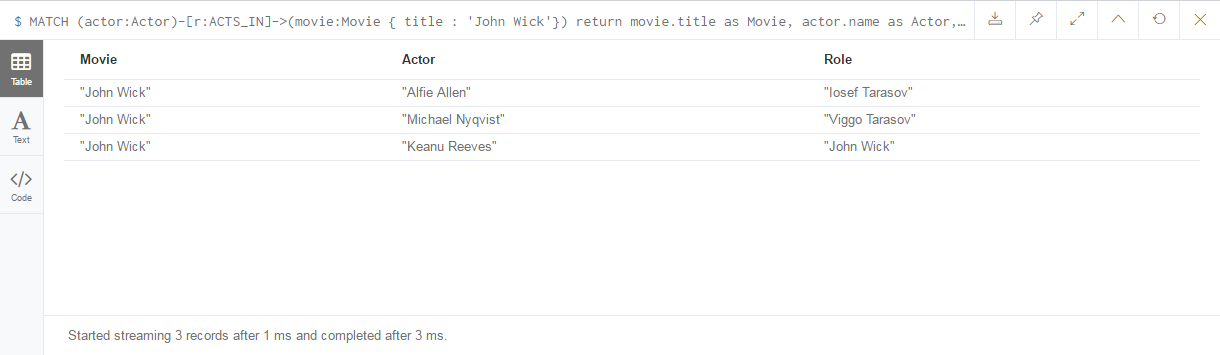
System.***out***.println( "Attached actor "+record.get( "Actor" ).asString() + " to movie " + record.get("Movie").asString() + " as "+ record.get("Role").asString());

}

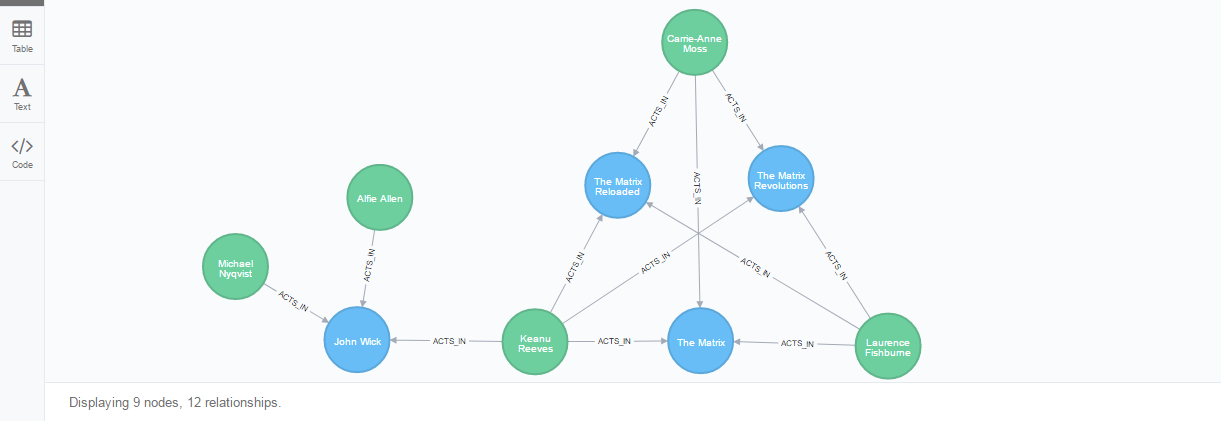
Console Output:



Cypher Confirmation (executing validation response string):



* Post-execution Database State:



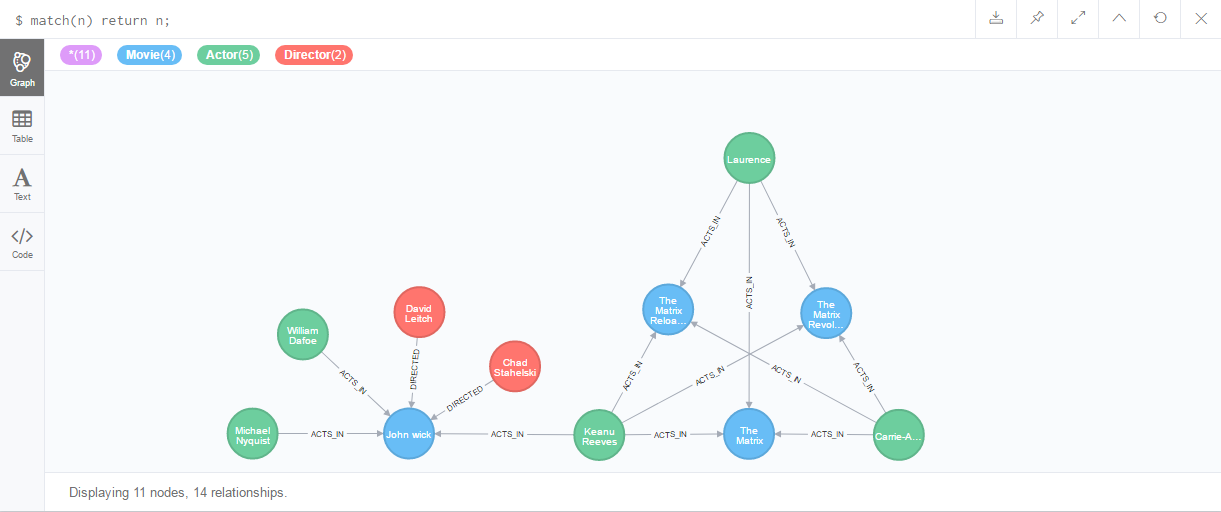
**Problem 4**:

Find a list of actors playing in movies in which Keanu Reeves played. Find directors of movies in which K. Reeves played. Please use any language of your convenience (Java, Python, C#, R, curl). Verify your results using Cypher queries in Cypher Browser

**Answer:**

I have used Problem 2 database state as a starting point for this problem.

* Pre-execution Database State



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

The standalone Neo4j Server can be installed on any machine and then accessed via its **binary Bolt protocol** through the official Neo4j Java driver.

Add dependency in pom as (pom.xml file submitted):

<dependency>

<groupId>org.neo4j.driver</groupId>

<artifactId>neo4j-java-driver</artifactId>

<version>1.0.3</version>

</dependency>

Set up celebrity query string as:

String celebrityQueryString = "MATCH (keanu:Actor{name:\"Keanu Reeves\"}) " +

"MATCH (keanu)-[:ACTS\_IN]->(keanuMovies) " +

"MATCH (keanuMovieActor)-[r:ACTS\_IN]->(keanuMovies) WHERE keanuMovieActor.name <> keanu.name " +

"RETURN distinct keanuMovieActor.name AS Celebrity,'Actor' as Contribution, r.role as Role " +

"UNION " +

"MATCH (keanuMovieDirector)-[r:DIRECTED]->(keanuMovies) " +

"RETURN distinct keanuMovieDirector.name AS Celebrity, 'Director' as Contribution, null as Role";

Set up complete relation query string as:

String completeRelationQueryString = "MATCH (keanu:Actor{name:\"Keanu Reeves\"}) " +

"MATCH (keanu)-[:ACTS\_IN]->(keanuMovies) " +

"MATCH (keanuMovieCelebs)-[r]->(keanuMovies) WHERE keanuMovieCelebs.name <> keanu.name " +

"RETURN distinct keanuMovieCelebs.name AS CelebName, r.role AS RoleName, keanuMovies.title AS MovieName";

Code Snippet (Class file submitted):

Driver driver = **null**;

Session session = **null**;

**try** {

driver = GraphDatabase.*driver*( "bolt://localhost", AuthTokens.*basic*( "neo4j", "\*" ));

session = driver.session();

System.***out***.println("List of actors that played in movies and directors that directed movies staring Keanu Reeves:");

StatementResult celebrity = session.run(celebrityQueryString);

**while** (celebrity.hasNext())

{

Record celebRecord = celebrity.next();

StringBuilder sb1 = **new** StringBuilder();

**if**((celebRecord.get("Contribution").asString()).equalsIgnoreCase("Actor")){

sb1.append(celebRecord.get("Celebrity") + " acted in a movie with Keanu Reeves");

} **else** {

sb1.append(celebRecord.get("Celebrity") + " directed a movie starring Keanu Reeves");

}

System.***out***.println(sb1.toString());

}

System.***out***.println("Detailed Relationships: ");

StatementResult completeRelation = session.run(completeRelationQueryString);

**while** (completeRelation.hasNext())

{

Record record = completeRelation.next();

StringBuilder sb = **new** StringBuilder();

sb.append("Retrieved "+record.get( "CelebName" ).asString()+" who ");

**if**(!(record.get( "RoleName" ).asString()).equalsIgnoreCase("null")) {

sb.append("played the role of " + record.get( "RoleName" ).asString() + " in ");

} **else** {

sb.append("directed ");

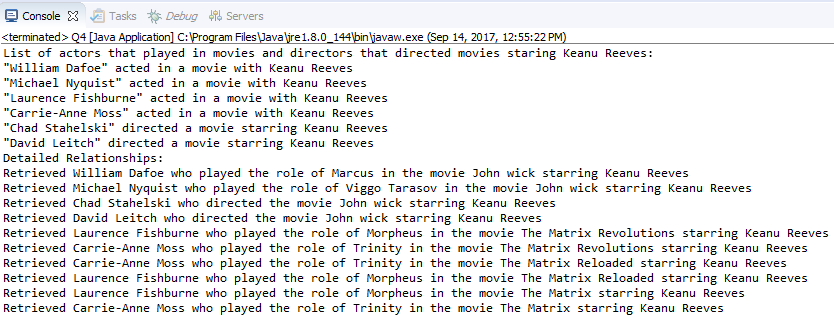
}

sb.append("the movie " + record.get( "MovieName" ).asString() + " starring Keanu Reeves");

System.***out***.println(sb.toString());

}

Console Output:



* Cypher Confirmation from Cypher Browser

CQL:

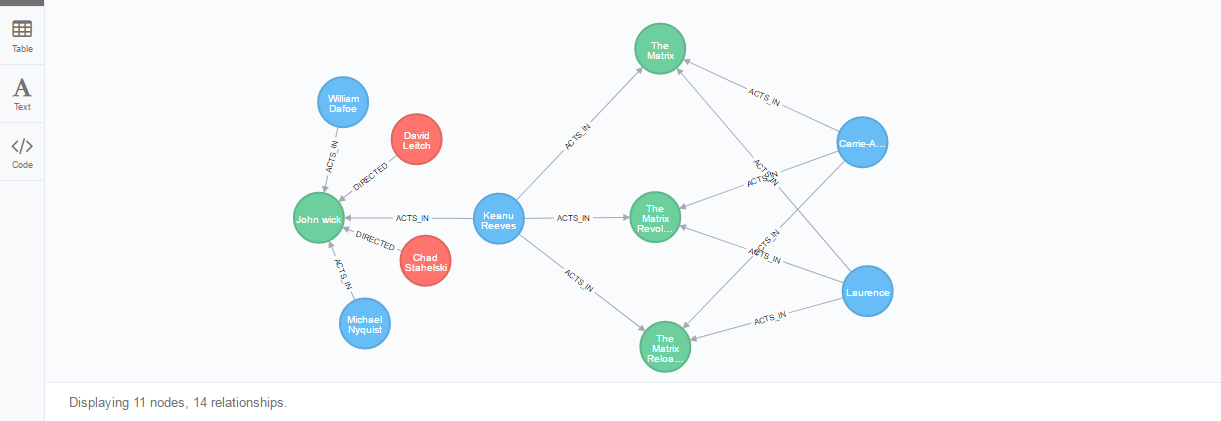
MATCH (keanu:Actor{name:"Keanu Reeves"})

MATCH (keanu)-[:ACTS\_IN]->(keanuMovies)

MATCH (keanuMovieCelebs)-[]->(keanuMovies)

return keanuMovieCelebs, keanuMovies

Cypher Browser Output:



**Problem 5:**

Find a way to export data from Neo4j into a set of CSV files. Delete your database and demonstrate that you can recreate the database by loading those CSV files. Please use any programming language of your convenience: Java, Python, R, C# or Scala.

**Answer:**

Setup CQL Queries For Data Export(Actor.csv and Director.csv files submitted):

**private** **static** **final** String ***actorExportQueryString*** = "CALL apoc.export.csv.query(\"MATCH (p:Actor)-[r]->(m:Movie) return p.name as name,"

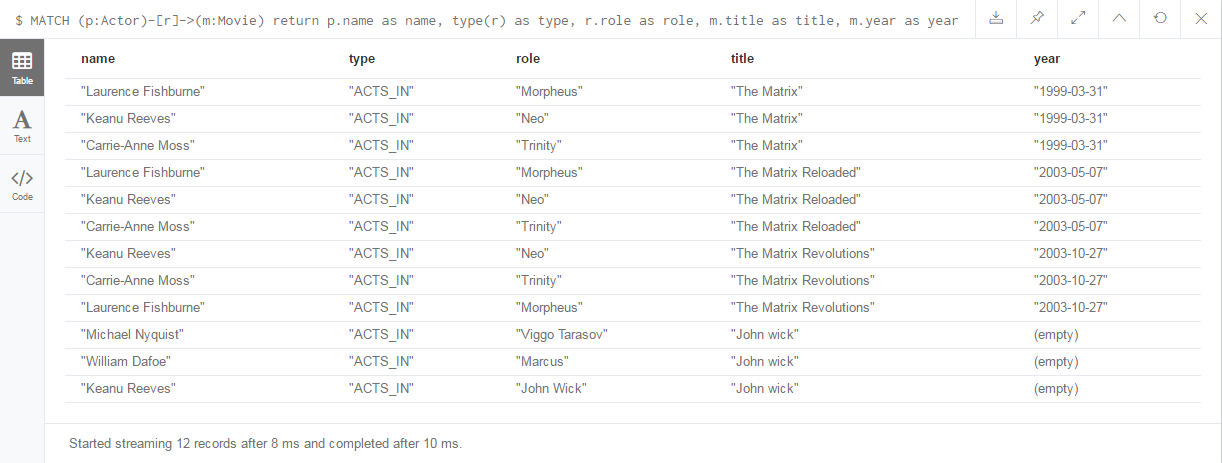
+ " type(r) as type, r.role as role, m.title as title,"

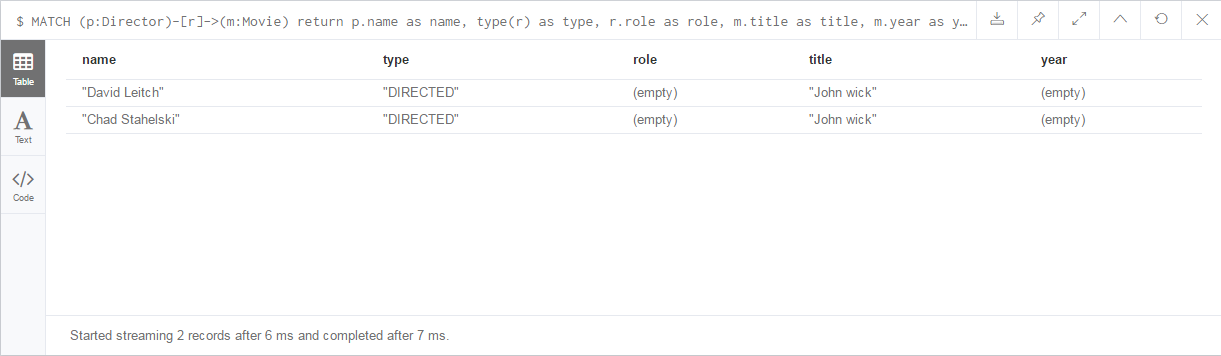
+ " m.year as year\", \"C:/Users/kbhandarkar/Actor.csv\", {})";

**private** **static** **final** String ***directorExportQueryString*** = "CALL apoc.export.csv.query(\"MATCH (p:Director)-[r]->(m:Movie) return p.name as name,"

+ " type(r) as type, m.title as title, m.year as year\","

+ " \"C:/Users/kbhandarkar/Director.csv\", {})";





Setup CQL Query for Database Deletion:

**private** **static** **final** String ***deleteQueryString*** = "MATCH (n) OPTIONAL MATCH (n)-[r]-() DETACH DELETE n,r";

Setup CQL Query for Database Import:

To ensure performant loading and avoiding duplicates it is recommended to create a unique constraint upfront.

CREATE CONSTRAINT ON (a:Person) ASSERT p.name IS UNIQUE;

CREATE CONSTRAINT ON (m:Movie) ASSERT m.title IS UNIQUE;

**private** **static** **final** String ***actorImportQueryString*** = "LOAD CSV WITH HEADERS FROM "file:///Actor.csv\" AS row " +

"MERGE (m:Movie {title:row.title}) ON CREATE SET m.year = row.year " +

"MERGE (a:Actor {name:row.name}) " +

"FOREACH (\_ in CASE row.type WHEN \"ACTS\_IN\" then [1] else [] end | " +

" MERGE (a)-[r:ACTS\_IN]->(m) ON CREATE SET r.role = row.role " + ") ";

**private** **static** **final** String ***directorImportQueryString*** = "LOAD CSV WITH HEADERS FROM \"file:///Director.csv\" AS row " +

"MERGE (m:Movie {title:row.title}) ON CREATE SET m.year = row.year " +

"MERGE (a:Director{name:row.name}) " +

"FOREACH (\_ in CASE row.type WHEN \"DIRECTED\" then [1] else [] end | " + " MERGE (a)-[r:DIRECTED]->(m) " + ") ";

Setup Steps:

1. Download the apoc plugin and place it in the plugins folder
2. Add the following lines at the end of neo4j.conf:
   1. dbms.directories.plugins=c:/Program\ Files/Neo4j\ CE\ 3.2.3/plugins
   2. dbms.security.procedures.unrestricted=apoc.\*
3. Add the following lines in the ‘Server Configuration’ section of neo4j.conf:
   1. apoc.import.file.enabled=true
   2. apoc.export.file.enabled=true
4. Restart Neo4j. Just stopping and starting server is not enough.

Note: Community edition requires the process ’neo4j-ce.exe’ to be killed manually from Task Manager.

Code Snippet(Class file submitted):

driver = GraphDatabase.*driver*( "bolt://localhost", AuthTokens.*basic*( "neo4j", "\*" ));

session = driver.session();

System.***out***.println("Export the movies database into Actor.csv and Director.csv files");

session.run(***actorExportQueryString***);

session.run(***directorExportQueryString***);

System.***out***.println("Delete the contents of the database");

session.run(***deleteQueryString***);

System.***out***.println("Import the contents of the .csv");

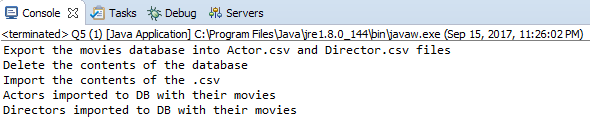
session.run(***actorImportQueryString***);

System.***out***.println("Actors imported to DB with their movies");

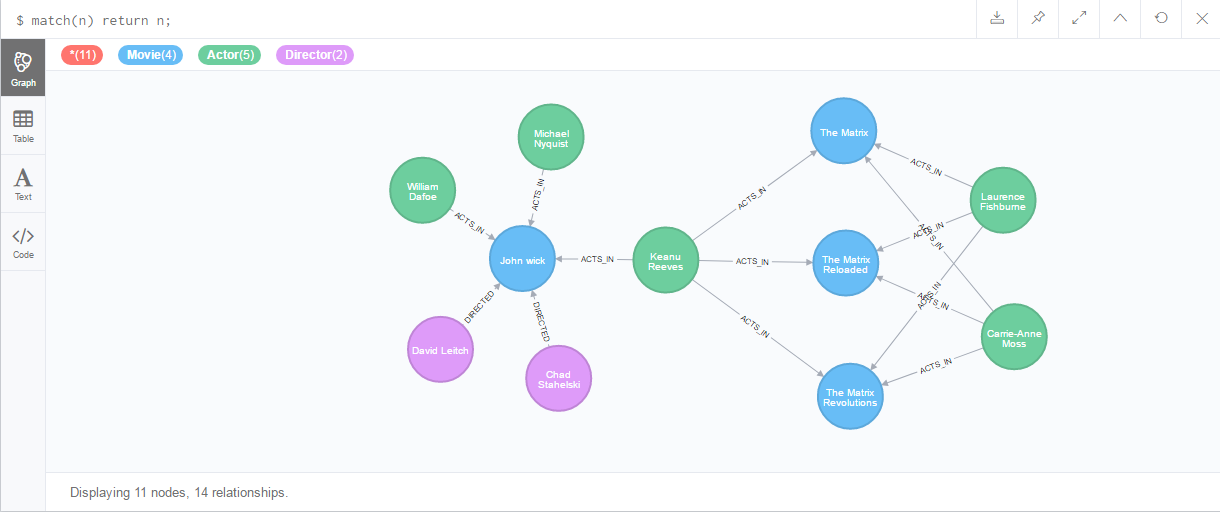
session.run(***directorImportQueryString***);

System.***out***.println("Directors imported to DB with their movies");

Console Output:



Database Final State:

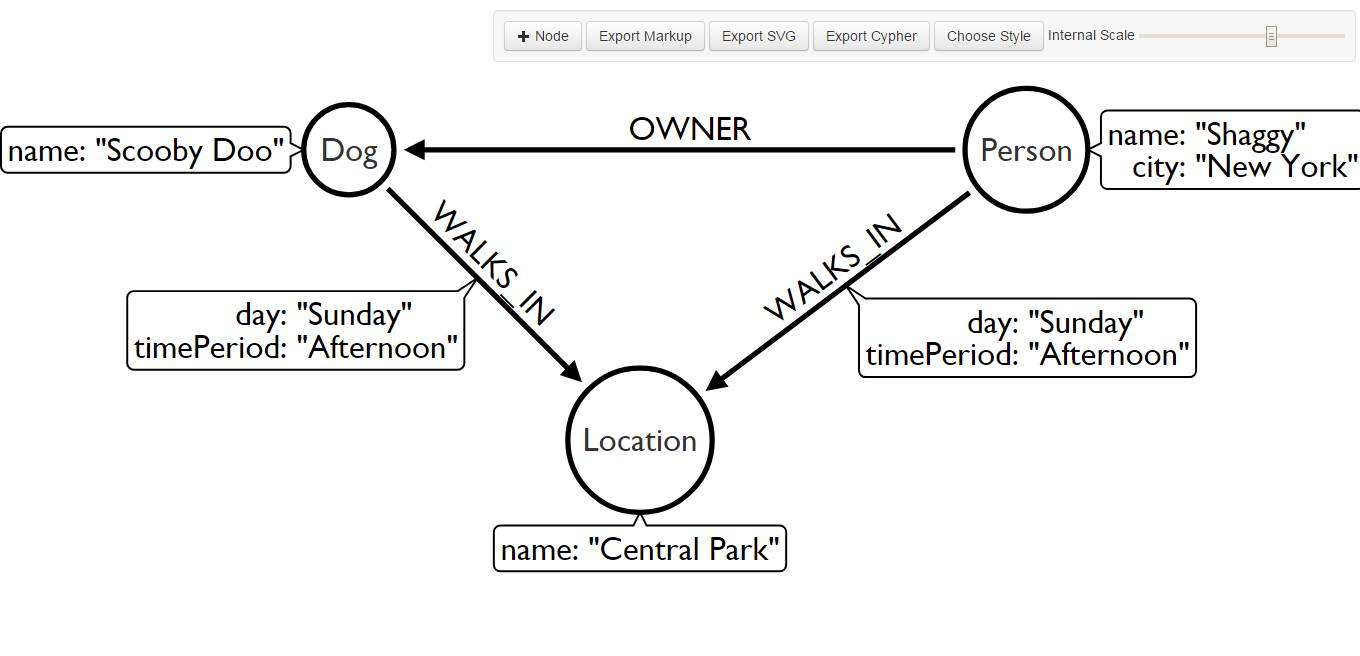


**Problem 6.**

Find a way to use Arrow Tool (<http://www.apcjones.com>) to paint a relationship between a dog and his owner who live in New York and walk through the Central Park on Sunday afternoon. Add Labels and necessary properties to all nodes and relationships. Export your graph in Cypher format and then adjust (if necessary) generated Cypher so that you can create that graph in Neo4J database. Verify that your graph is indeed created using Cypher Browser.

**Answer:**

* Paint a relationship between a dog and his owner who live in New York and walk through the Central Park on Sunday afternoon



* Export your graph in Cypher format



* Adjust (if necessary) generated Cypher



* Graph Setup Query

CREATE (`1` :Person {name:"Shaggy",city:"New York"}) , (`2` :Dog {name:"Scooby Doo"}) , (`3` :Location {name:"Central Park"}) , (`1`)-[:`OWNER`]->(`2`), (`2`)-[:`WALKS\_IN`]->(`3`), (`1`)-[:`WALKS\_IN`]->(`3`)

* Graph Validation Query

start n=node(\*) return n

* Cypher Browser Validation

