Ryan Ballenger
CSCIE63 Big Data Analytics
Assignment 10

Assignment 10 Solution

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)
```

// File types have been modified to types that Canvas accepts. one.java should be converted to
// one.sh. Also, four.java contains Cypher queries and not java code.

// The Neo4j server properties are edited to turn off authorization and allow CSV loading.
// **Options -> Edit** accesses neo4j-server.properties and the following lines are modified.

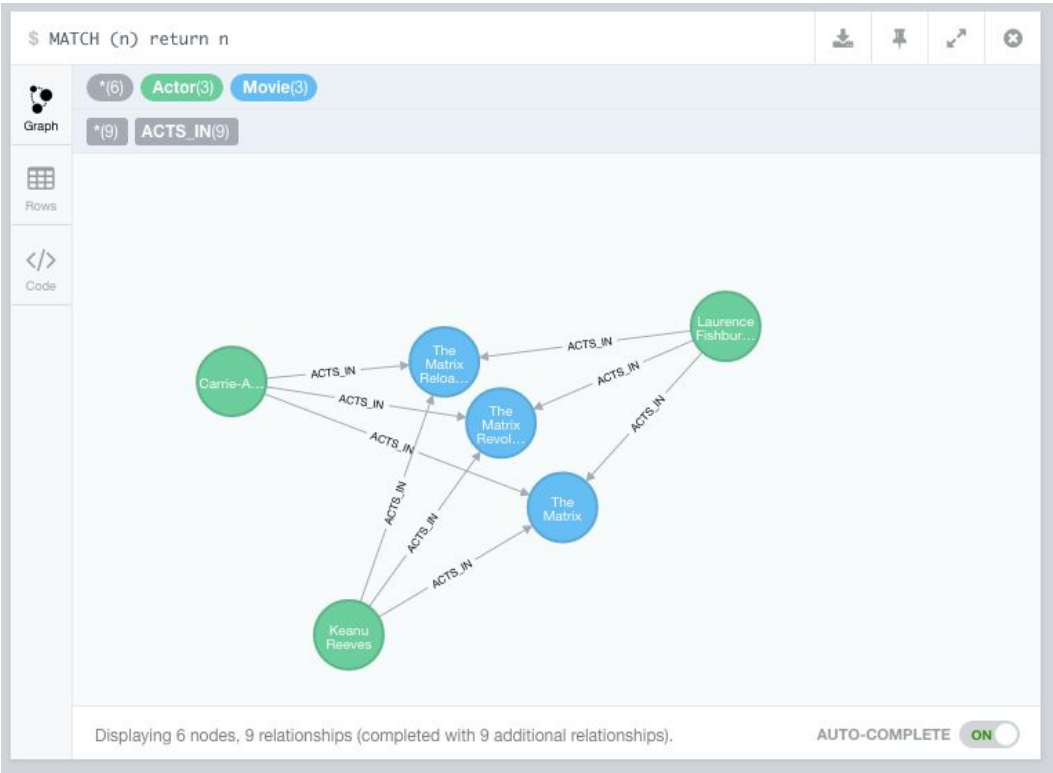
```
# Require (or disable the requirement of) auth to access Neo4j
dbms.security.auth_enabled=false

# Allow CSV file loading
allow_file_urls=true
dbms.security.load_csv_file_url_root=csv-files
```

// The curl commands are collected into one.sh where they can all be executed at once.
// The permissions on the file one.sh are set to be an executable.

```
Ryans-MacBook-Pro:Desktop Ryan$ chmod +x one.sh
```

// The database is created with curl commands contained within one.sh.
Ryans-MacBook-Pro:Desktop Ryan$ ./one.sh
HTTP/1.1 200 OK
Date: Sat, 16 Apr 2016 00:08:03 GMT
Content-Type: application/json
Access-Control-Allow-Origin: *
Content-Length: 50
Server: Jetty(9.2.z-SNAPSHOT)
{"results":[{"columns":[],"data":[]}],"errors":[]}HTTP/1.1 200 OK
Date: Sat, 16 Apr 2016 00:08:03 GMT
Content-Type: application/json
Access-Control-Allow-Origin: *
Content-Length: 111
Server: Jetty(9.2.z-SNAPSHOT)
{"results":[{"columns":["matrix1"],"data":[{"row":[{"title":"The Matrix","year":"1999-03-31"}]}]}],"errors":[]}HTTP/1.1 200 OK
Date: Sat, 16 Apr 2016 00:08:04 GMT
…

// An all-inclusive query, MATCH (n) return n, through Cypher confirms the database was
// created properly.



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.

// The M2Eclipse plug in is installed which provides a robust integration with Maven.
Preferences -> Install/Update -> Available Software Sites -> Add
Paste the URL http://download.eclipse.org/technology/m2e/releases
Reload

// A new Maven project is created to run the JAVA REST API
File -> New -> Project -> Maven Project and **maven-archetype-quickstart** is selected.

// The pom.xml file is modified to include Neo4j. The following dependency is added. This
// loads the necessary Neo4j libraries to run the API.

```
<dependency>
     <groupId>org.neo4j</groupId>
     <artifactId>neo4j</artifactId>
     <version>2.0.1</version>
</dependency>
```

// The project is cleaned to load the new Neo4j libraries.
Project -> Clean and select **graphproject**

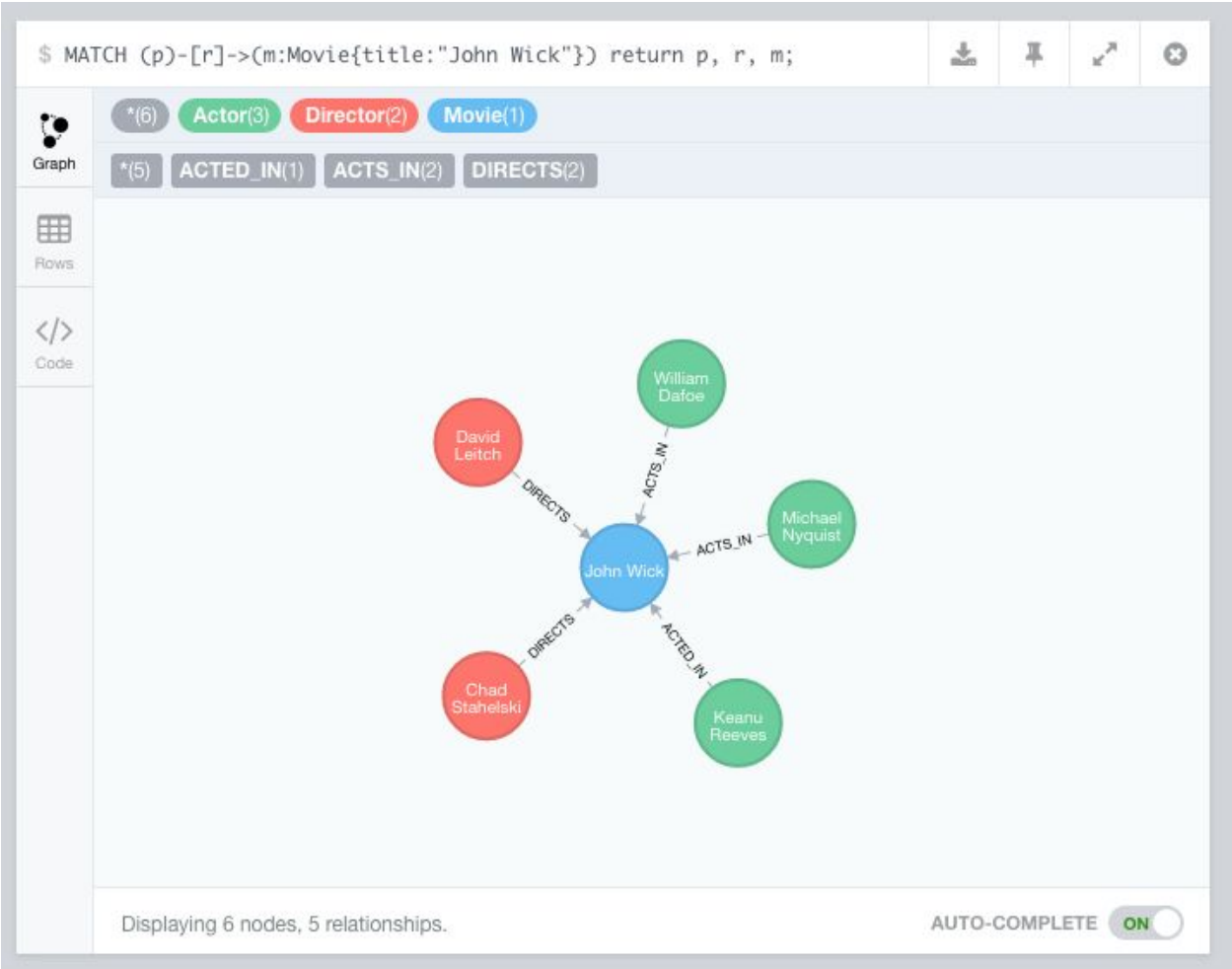
// The JRE System library is updated. A more recent version of Java must be used.
Build Path -> Configure Build Path
Add Library -> JRE System Library -> Alternate JRE and select **Java SE 8**

// The compiler is updated to 1.7. This ensures the compiled project meets the standards
// for Java 1.7.
Project -> Properties -> Java Compiler
Compiler Compliance Level is set to **1.7**

// ConnectToServer.java is run. Its main method has been modified to add the John Wick related
// items to the database. The console output is pasted below.
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/42]
PUT to [http://localhost:7474/db/data/node/42/properties/title], status code [204]
POST [{"statements" : [{"statement" : "MATCH (n { title: 'John Wick' }) SET n:Movie RETURN n"}]}] to [http://localhost:7474/db/data/transaction/commit], status code [200], returned data:
{"results":[{"columns":["n"],"data":[{"row":[{"title":"John Wick"}]}]}],"errors":[]}
PUT to [http://localhost:7474/db/data/node/42/properties/year], status code [204]
POST to [http://localhost:7474/db/data/node], status code [201], location header [http://localhost:7474/db/data/node/43]
PUT to [http://localhost:7474/db/data/node/43/properties/name], status code [204]
POST [{"statements" : [{"statement" : "MATCH (n { name: 'Chad Stahelski' }) SET n:Director RETURN n"}]}] to [http://localhost:7474/db/data/transaction/commit], status code [200], returned data:
{"results":[{"columns":["n"],"data":[{"row":[{"name":"Chad Stahelski"}]}]}],"errors":[]}
POST to [http://localhost:7474/db/data/node/43/relationships], status code [201], location
…

// The following query is run in Cypher which returns all the data pertaining to John Wick
// and confirming the updates to the database. A new relationship DIRECTS is used for the two
// directors and an ACTS_IN relationship is added for Keanu Reeves and the new movie.
// MATCH (p)-[r]->(m:Movie{title:"John Wick"}) return p, r, m;



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

```
// The movies Keanu Reeves acted in are found and then, all actors in those movies are
// returned. A list is created with collect() and distinct is used to eliminate duplicates.

match (p:Actor{name:"Keanu Reeves"})-[r]->(m:Movie)
match(z:Actor)-[w]->(m)
return collect(DISTINCT z.name);
```

// For part 2, the movies in which Keanu Reeves acted in are again found. The directors of
// those movies are matched and returned. Collect() and distinct are used to create the list of
// directors.

```
match (p:Actor{name:"Keanu Reeves"})-[r]->(m:Movie)
match(z:Director)-[w]->(m)
return collect(DISTINCT z.name);
```

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.

// The CSV files are created with Cypher to store the database.
// actors.csv is created with the actor ID and name.
match (a:Actor) return ID(a) as actorId, a.name as name;
Export to file -> Export CSV

```
$ match (p:Actor)-[r]->(m:Movie) return ID(p) as actorId...
```

actorId	movieId	role
63	68	Neo
64	68	Morpheus
65	68	Trinity
63	69	Neo
64	69	Morpheus
65	69	Trinity
65	70	Trinity
64	70	Morpheus
63	70	Neo
63	71	null
67	71	Viggo Tarasov
66	71	Marcus

Returned 12 rows in 289 ms.

// movies.csv is created to store the movies' ID, title, and year.
match (m:Movie) return ID(m) as movieId, m.title as title, m.year as year;
Export to file -> Export CSV

```
$ match (m:Movie) return ID(m) as movieId, m.title as title, m...
```

movieId	title	year
68	The Matrix	1999-03-31
69	The Matrix Reloaded	2003-05-07
70	The Matrix Revolutions	2003-10-27
71	John Wick	2014-10-24

Returned 4 rows in 220 ms.

// directors.csv is created to store the directors' ID and name.
match (a:Director) return ID(a) as directorId, a.name as name;
Export to file -> Export CSV

```
$ match (a:Director) return ID(a) as directorId, a.name as nam...
```

directorId	name
72	Chad Stahelski
73	David Leitch

Returned 2 rows in 220 ms.

// acting.csv is created to store the ACTS_IN relationship data.
match (p:Actor)-[r]->(m:Movie) return ID(p) as actorId, ID(m) as movieId, r.role as role;
Export to file -> Export CSV

$ match (p:Actor)-[r]->(m:Movie) return ID(p) as actorId, ID(m...

Rows

Code

actorId	movieId	role
63	68	Neo
64	68	Morpheus
65	68	Trinity
63	69	Neo
64	69	Morpheus
65	69	Trinity
65	70	Trinity
64	70	Morpheus
63	70	Neo
63	71	null
67	71	Viggo Tarasov
66	71	Marcus

Returned 12 rows in 211 ms.

// directing.csv is created to store the DIRECTS relationship data.
match (p:Director)-[r]->(m:Movie) return ID(p) as directorId, ID(m) as movieId, r.role as role;
Export to file -> Export CSV

$ match (p:Director)-[r]->(m:Movie) return ID(p) as directorId...

Rows

Code

directorId	movieId	role
72	71	Director
73	71	Director

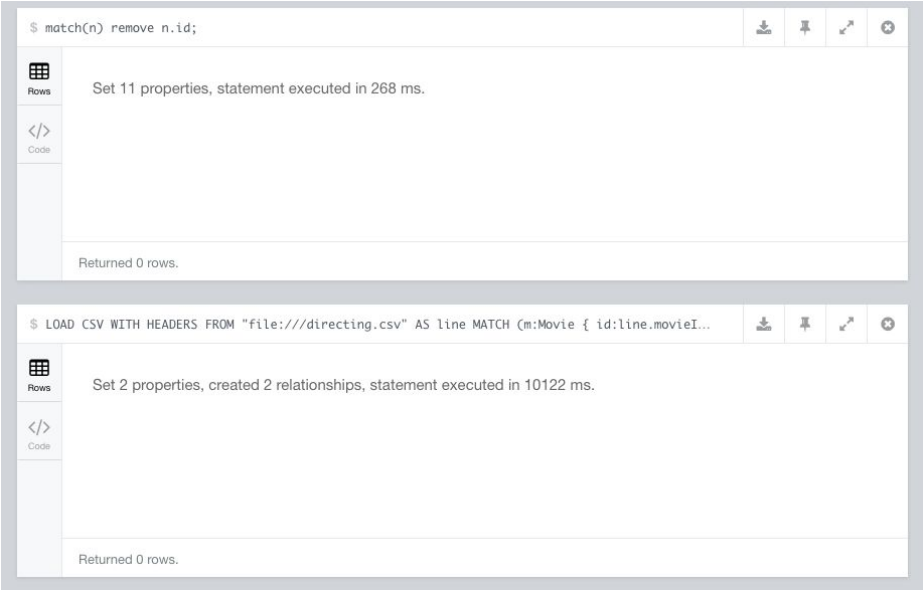
Returned 2 rows in 278 ms.

// The CSV files are copied to Neo4JCommunityEdition/Contents/Resources/app/bin/csv-files
// The database is cleared with Cypher statement MATCH (n) DETACH DELETE n

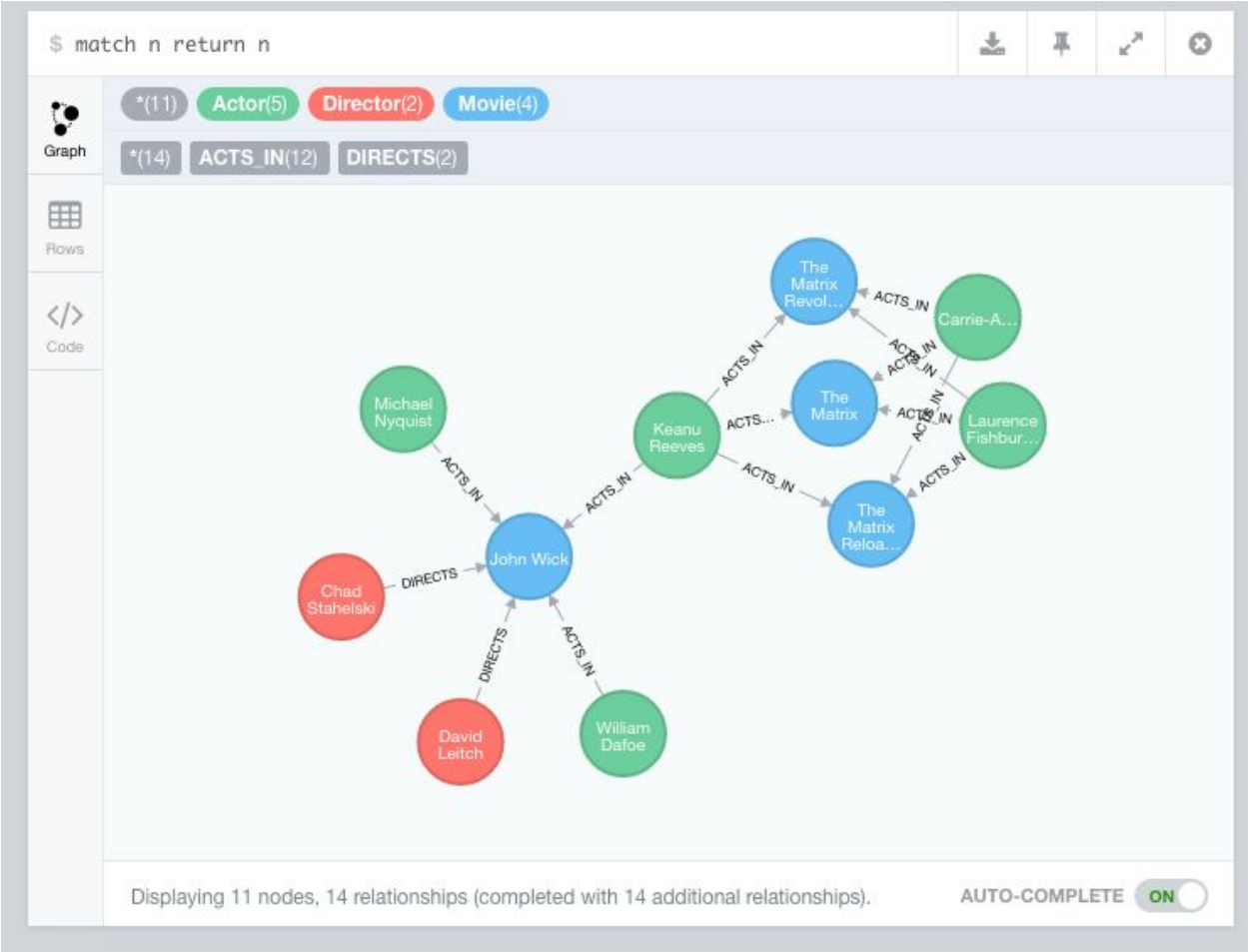
// The following 5 CREATE statements are run in Cypher to reload the database.
// The queries are also submitted in the file four. In order, they create the actors, the movies, the
// directors, the acting relationships, and finally the directing relationships. Lastly the id value is
// dropped from everything because it was only used to reconstruct the database. The beginning
// of the output is pasted below.

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

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

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

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

LOAD CSV WITH HEADERS FROM "file:///directing.csv" AS line
MATCH (m:Movie { id:line.movieId })
MATCH (a:Director { id:line.directorId })
CREATE (a)-[:DIRECTS { role:line.role}]->(m);
```

match(n) remove n.id;



...

// The inclusive query, match n return n, shows the database was reconstructed. Further

// analysis confirms all the roles, years, and other details are present.