## Owen Galvin

## 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)

**Solution:**

**I’m on windows, so download neo4j-community\_windows-x64\_2\_3\_3.exe and run the installation package. Accept the default installation directory but re-direct the database location to a path of my choosing.**

|  |  |
| --- | --- |
|  |  |

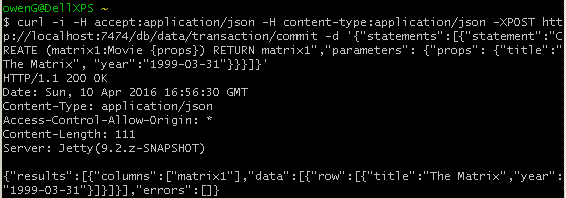
**Edit the neo4j-server.properties to both disable authentication and also add two new lines, the first to allow CSVs to be loaded from a file URL and then also to set the root directory for those CSV files.**

|  |
| --- |
|  |

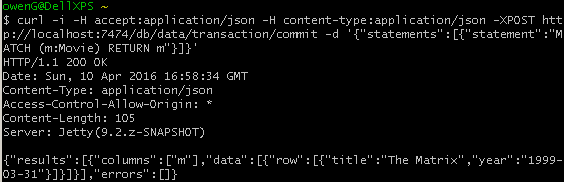
**Start the server and run simple Cypher query in the browser to confirm things are working.**

|  |
| --- |
|  |

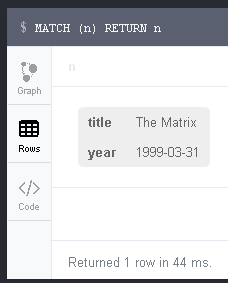
**My Cygwin has curl installed, try to create the first Movie node in the assignment using the full command line/JSON syntax discussed in the lab pdf, the CREATE command to create the Movie node and additional JSON for the properties passed in as parameter.**



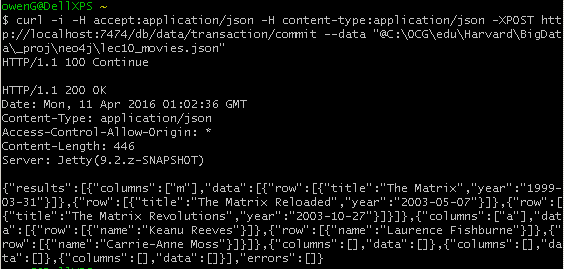
**Which seems to have worked, confirm by running MATCH on all Movie nodes, which returns (only) the one I just created.**



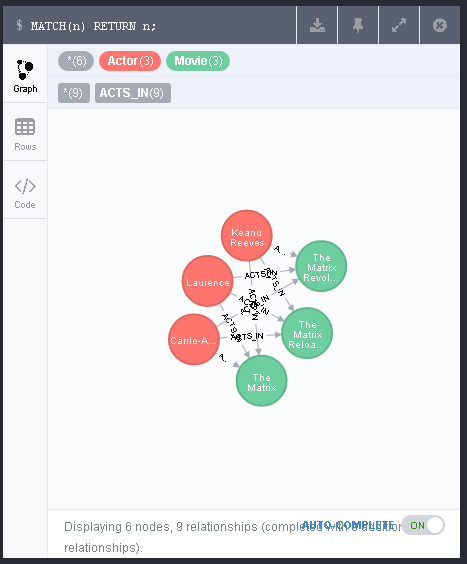
**And confirm in browser, running query to list all nodes.**



**In theory could add the rest of the data using the curl along with raw JSON but instead going to try something a little more robust. After deleting the single movie I had just added, use the @ syntax to pass in the path to a .json file with all that data that needs to be added.**



**Back to the browser, run command to return all nodes and see that we have all of the data.**



**Drag around a little to make the relationships etc. more obvious. Hover over the Laurence Fishburne -> The Matrix “ACTS\_IN” relationship to show it has a property = role, with value = “Morpheus”.**

|  |
| --- |
|  |

**The .json file breaks down to three general sections, for movies, actors, and relationships, with a “statements” array that holds the neo4j syntax for each set of actions. The movies can be simply parameterized by passing in the title/year property set for each movie, such that the CREATE statement iterates through the three Matrix movies.**

|  |
| --- |
| {  "statements": [  {  "statement": "CREATE (m:Movie {props}) RETURN m",  "parameters": {  "props": [  {  "title": "The Matrix",  "year": "1999-03-31"  },  {  "title": "The Matrix Reloaded",  "year": "2003-05-07"  },  {  "title": "The Matrix Revolutions",  "year": "2003-10-27"  }  ]  }  }, |

**The same pattern can be used to create nodes for the three actors, though they only have the single property = name.**

|  |
| --- |
| {  "statement": "CREATE (a:Actor {props}) RETURN a",  "parameters": {  "props": [  {  "name": "Keanu Reeves"  },  {  "name": "Laurence Fishburne"  },  {  "name": "Carrie-Anne Moss"  }  ]  }  }, |

**The creation of the relations is a bit trickier, or at least I didn’t see a way of being able to directly use the movie/actor identifiers since they were never materialized with “named” identifiers. On the other hand, the fact that each actor appeared in each of the Movie nodes, with the same character name each time, allows for a MATCH to first be made against any/all Movie nodes, followed by a MATCH query on the specific actor name. As a result, each of the statements below creates three ACTS\_IN relationships between the actor, with role property appropriate to the actor/character.**

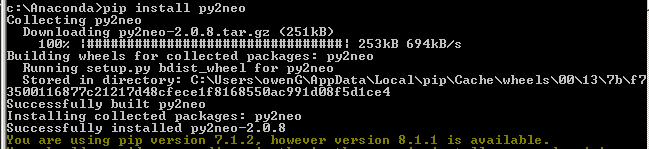
|  |
| --- |
| {  "statement": "MATCH (m:Movie) MATCH (a:Actor { name:'Keanu Reeves' }) CREATE (a)-[:ACTS\_IN { role : 'Neo' }]->(m)"  },  {  "statement": "MATCH (m:Movie) MATCH (a:Actor { name:'Laurence Fishburne' }) CREATE (a)-[:ACTS\_IN { role : 'Morpheus' }]->(m)"  },  {  "statement": "MATCH (m:Movie) MATCH (a:Actor { name:'Carrie-Anne Moss' }) CREATE (a)-[:ACTS\_IN { role : 'Trinity' }]->(m)"  }  ]  } |

|  |
| --- |
| **Cygwin/curl text :**  #use full JSON syntaxt to create node w/Title = "The Matrix"  curl -i -H accept:application/json -H content-type:application/json -XPOST http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"CREATE (matrix1:Movie {props}) RETURN matrix1","parameters": {"props": {"title":"The Matrix", "year":"1999-03-31"}}}]}'  #confirm  curl -i -H accept:application/json -H content-type:application/json -XPOST http://localhost:7474/db/data/transaction/commit -d '{"statements":[{"statement":"MATCH (m:Movie) RETURN m"}]}'  #loads in the whole movies.json file  curl -i -H accept:application/json -H content-type:application/json -XPOST http://localhost:7474/db/data/transaction/commit --data "@C:\OCG\edu\Harvard\BigData\\_proj\neo4j\lec10\_movies.json"  **#Browser text**  MATCH(n) RETURN n; |

**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:**

**To begin with, install the Python module for py2neo, since that is what I plan on working with.**



**Begin with review of my python code, pyLec10\_p2.py, which imports two necessary modules from py2neo and creates a Graph() object name graph, which is pointed to default 7474 Neo4j server url and will be used to run all commands from this file. The first command is a simple execute, passing in the Cypher syntax for creating a new node with Movie label and appropriate properties for John Wick. The returned object is a RecordList object, with length = 1. To get the created Movie node I get the first column on the first/only item in the RecordList via [0][0] index.**

|  |
| --- |
| from py2neo import Graph, Path  graph = Graph()  wick = graph.cypher.execute('CREATE (m:Movie {title: {title}, year: {year}}) RETURN m',  title='John Wick',  year = '2014-10-24')[0][0] |

**Going to create the Actor nodes using a py2neo transaction named tx2 – call .begin() to mark start of transaction and then later .commit() on same object to have the transaction committed. For each actor in the two-person list, append a Cypher CREATE command, which includes a property = name of actor in the list. Assign the resulting nodes to ‘will’ and ‘mike’ variable respectively by iterating through the transaction results. Then for each actor create a Path() object, which in py2neo results in a Neo4j relationship, in this case of type = ACTS\_IN. Pass in a dictionary of properties to that relationship such that each is set with the role played by that actor in the movie.**

|  |
| --- |
| tx2 = graph.cypher.begin()  for actor\_name in ['William Dafoe','Michael Nyquist']:  tx2.append('CREATE (a:Actor {name: {name}}) RETURN a', name=actor\_name)  will, mike = [result.one for result in tx2.commit()]  will\_path = Path(will, ('ACTS\_IN', {'role': 'Marcus'}), wick)  graph.create(will\_path)  mike\_path = Path(mike, ('ACTS\_IN', {'role': 'Viggo Tarasov'}), wick)  graph.create(mike\_path) |

**There is essentially no difference in workflow when it comes to adding the two directors and creating DIRECTED relationships, though the latter is simpler since there are no properties to be set. I also have a commented out line showing more concise Python syntax for creating these nodes – it works fine but is noticeably less readable than the ‘live’ code version.**

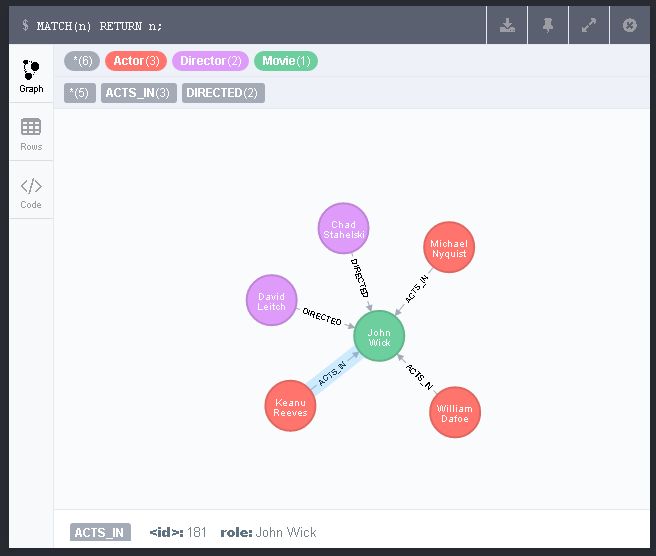
|  |
| --- |
| tx3 = graph.cypher.begin()  for director\_name in ['Chad Stahelski', 'David Leitch']:  tx3.append('CREATE (d:Director {name: {name}}) RETURN d', name=director\_name)  chad, david = [result.one for result in tx3.commit()]  chad\_path = Path(chad, "DIRECTED", wick)  graph.create(chad\_path)  david\_path = Path(david, "DIRECTED", wick)  graph.create(david\_path)  #concise but less readable, don't need to worry about properties on the DIRECTED relationship  #[graph.create(Path(result.one, 'DIRECTED', wick)) for result in tx3.commit()] |

**The final part involves a goal not explicitly stated in the assignment but which I think makes sense – to add an ACTS\_IN relationship from Keanu to the new John Wick movie. Begin a transaction, with a first statement that uses MERGE – that way if this is run on the database as it exists post-Problem 1, another Keanu Actor node will not be created if it already exists. Second command is to return a Node object matching that of Actor = Keanu, whether it pre-existed or was just then created. Since there were two statements in the transaction, can skip the result on the MERGE and use [1] subscript to get the MATCH’d Keanu node from the second statement. From there an ACTS\_IN relationship is created from Keanu to John Wick movie, with a role property = character he played, i.e. “John Wick”.**

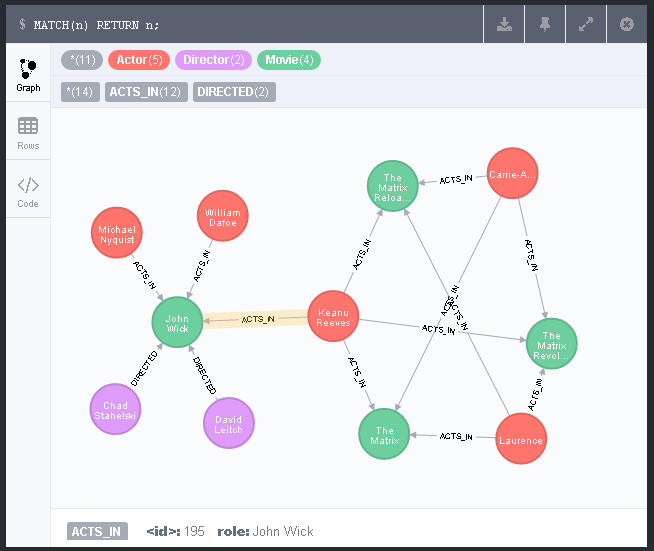
**(I’ll also point out the opportunity to create another Movie for the forthcoming “John Wick: Chapter Two” sequel, which would be interesting in part because Keanu will be appearing alongside Laurence Fishburne once again.)**

|  |
| --- |
| tx4 = graph.cypher.begin()  tx4.append('MERGE (a:Actor { name:"Keanu Reeves" })')  tx4.append('MATCH (a:Actor { name:"Keanu Reeves" }) RETURN a')  keanu = tx4.commit()[1].one  keanu\_path = Path(keanu, ("ACTS\_IN", {'role': 'John Wick'}), wick)  graph.create(keanu\_path)  #John Wick: Chapter Two |

**If the above Python code is run against a clean Neo4j db it will result in a graph like below, one Movie plus two Director nodes + three Actor nodes.**



**If it is instead run against the db as it existed after the Cypher/JSON in Problem 1, get a more complex and interesting graph. Note that Keanu still only exists the once, as he was first created in Problem 1.**



**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:**

**To find a list of actors playing movies in which Keanu acted, a query like below will return the Actor nodes. Once a movie is found that Keanu acted in, find any ACTS\_IN relationships that poin to that same move. Use DISTINCT to skip double counting. A discussion board comment by a TA mentioned exporting to CSV, so I’ll submit the output as lec10\_p3a.csv.**

|  |
| --- |
| MATCH (a:Actor {name: 'Keanu Reeves'}) - [r:ACTS\_IN] -> (m:Movie) <- [r2:ACTS\_IN] - (a2:Actor) RETURN DISTINCT a2 AS Colleague; |

**Alternatively, RETURN a2.name instead of the full node gives just textual actor names.**

|  |
| --- |
|  |

**Next, a query to find directors of movies in which Keanu Reeves played is similar to the earlier one, only with a DIRECTED filter on the other side of the MATCHd Movie node.**

|  |
| --- |
| MATCH (a:Actor {name: 'Keanu Reeves'}) - [r:ACTS\_IN] -> (m:Movie) <- [:DIRECTED] - (d:Director) RETURN d; |

**As graph.**

|  |
| --- |
|  |

**And finally as text names – the output is so minimal that instead of submitting a csv I’ll just put the contents of the export in cell below the screenshot.**

|  |
| --- |
|  |
| d.name  David Leitch  Chad Stahelski |

**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:**

**The first step will involve running a series of Cypher queries in the browser and performing an Export to file from the browser after each (also, rename the exported .csv to something meaningful). Include text of query followed by screenshot for each of below.**

1. **Create movies.csv by grabbing movie title and “year” released.**

|  |
| --- |
| MATCH (m:Movie) RETURN m.title AS title, m.year AS year; |
|  |

1. **Create actors.csv, only need a simple list of the value from name property of Actor nodes.**

|  |
| --- |
| MATCH (a:Actor) RETURN a.name AS name; |
|  |

1. **Export the roles.csv by finding all relationships with a non-NULL “role” property. Could also have filtered on ACTED\_IN relationship but this is what I came up with first and since this is such a tiny db with limited complexity, the query does what it needs to.**

|  |
| --- |
| MATCH(a) MATCH (a)-[r]->(m) WHERE r.role IS NOT NULL  RETURN a.name AS actor,r.role AS role,m.title AS title; |
|  |

1. **Final query creates the directors.csv and returns the name of any node that held a DIRECTED relationship to any other node.**

|  |
| --- |
| MATCH (d)-[r:DIRECTED]->(m) RETURN d.name AS director,m.title AS title; |
|  |

**Delete contents of database by deleting all nodes.**

|  |
| --- |
| MATCH(n) DETACH DELETE n; |
|  |

**Now to re-create the database by reloading the four .csv files back in using appropriate commands.**

1. **Create Movie nodes, with properties for title and year read from the .csv. Show immediate output and then run command to display all nodes and take screenshot.**

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

1. **Same for actors.csv**

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

1. **Processing the roles.csv is more complicated but not too bad. MATCH the existing Movie nodes against movie title from csv, also MATCH for Actor nodes/actor column in csv. From those matches, create an ACTS\_IN relationship, adding the value from matched role value in the csv as a property named role.**

|  |
| --- |
| LOAD CSV WITH HEADERS FROM "file:///roles.csv" AS line  MATCH (m:Movie {title:line.title})  MATCH (a:Actor {name:line.actor} )  CREATE (a) - [:ACTS\_IN {role:line.role}] -> (m); |
|  |
|  |

1. **Final piece is to load the directors.csv in, which requires, at least in terms of the file I had exported, that the WITH keyword be used in order to do everything in one statement. This allows the aliased Director nodes created at the beginning to be accessed in the final line in order to create the DIRECTED relationships.**

**After that, the final screenshot looks like the database that had been populated via Curl/JSON & Python/JSON.**

|  |
| --- |
| LOAD CSV WITH HEADERS FROM "file:///directors.csv" AS line  MATCH (m:Movie {title:line.title})  CREATE (d:Director {name:line.director})  WITH d, m  CREATE (d) - [:DIRECTED] -> (m); |
|  |
|  |