**HU Extension Assignment 02 E63 Big Data Analytics**

**Handed out: 02/03/2017 Due by 11:59PM EST on Friday, 02/10/2017**

**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'}) return id(matrix1)

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)

**Solution:**

1. Create a json file with relevant statements to create database nodes and relationships.
2. Use Curl -d @filename option to upload and execute json file.
3. Verify database contents using Curl.

**1. statements.json**

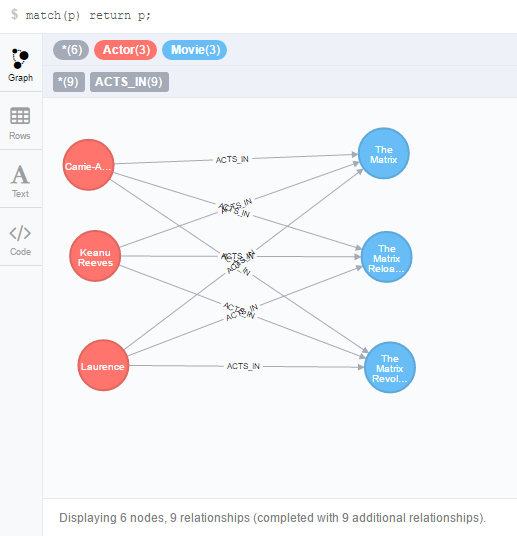
|  |
| --- |
| {"statements":[  {"statement":"CREATE (matrix1:Movie { title : 'The Matrix', year : '1999-03-31' })  CREATE (matrix2:Movie { title : 'The Matrix Reloaded', year :'2003-05-07' })  CREATE (matrix3:Movie { title : 'The Matrix Revolutions', year :'2003-10-27' })  CREATE (keanu:Actor { name:'Keanu 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)  RETURN matrix1, matrix2, matrix3, keanu, laurence, carrieanne"  }  ]  } |

**2. Upload contents using curl -d @filename option.**

|  |
| --- |
| $ *curl.exe -i -H accept:application/json -H content-type:application/json -XPOST http://localhost:7474/db/data/transaction/commit -d @statements.json*  % Total % Received % Xferd Average Speed Time Time Time Current  Dload Upload Total Spent Left Speed  100 2082 100 607 100 1475 2160 5249 --:--:-- --:--:-- --:--:-- 5249HTTP/1.1 100 Continue  HTTP/1.1 200 OK  Date: Thu, 09 Feb 2017 03:01:35 GMT  Content-Type: application/json  Access-Control-Allow-Origin: \*  Content-Length: 607  Server: Jetty(9.2.z-SNAPSHOT)  {"results":[{"columns":["matrix1","matrix2","matrix3","keanu","laurence","carrieanne"],"data":[{"row":[{"year":"1999-03-31","title":"The Matrix"},{"year":"2003-05-07","title":"The Matrix Reloaded"},{"year":"2003-10-27","title":"The Matrix Revolutions"},{"name":"Keanu Reeves"},{"name":"Laurence Fishburne"},{"name":"Carrie-Anne Moss"}],"meta":[{"id":454,"type":"node","deleted":false},{"id":455,"type":"node","deleted":false},{"id":456,"type":"node","deleted":false},{"id":457,"type":"node","deleted":false},{"id":458,"type":"node","deleted":false},{"id":459,"type":"node","deleted":false}]}]}],"errors":[]} |

**3. Verify database contents.**

|  |
| --- |
| $ *curl.exe -i -H accept:application/json -H content-type:application/json -XPOST http://localhost:7474/db/data/transaction/commit -d "{\"statements\":[{\"statement\":\"MATCH (m) return m\"}]}"*  % Total % Received % Xferd Average Speed Time Time Time Current  Dload Upload Total Spent Left Speed  100 1829 100 1778 100 51 6684 191 --:--:-- --:--:-- --:--:-- 6684HTTP/1.1 200 OK  Date: Thu, 09 Feb 2017 02:43:18 GMT  Content-Type: application/json  Access-Control-Allow-Origin: \*  Content-Length: 1778  Server: Jetty(9.2.z-SNAPSHOT)  {"results":[{"columns":["m"],"data":[{"row":[{"year":"1999-03-31","title":"The Matrix"}],"meta":[{"id":412,"type":"node","deleted":false}]},{"row":[{"year":"2003-05-07","title":"The Matrix Reloaded"}],"meta":[{"id":413,"type":"node","deleted":false}]},{"row":[{"year":"2003-10-27","title":"The Matrix Revolutions"}],"meta":[{"id":414,"type":"node","deleted":false}]},{"row":[{"name":"Keanu Reeves"}],"meta":[{"id":415,"type":"node","deleted":false}]},{"row":[{"name":"Laurence Fishburne"}],"meta":[{"id":416,"type":"node","deleted":false}]},{"row":[{"name":"Carrie-Anne Moss"}],"meta":[{"id":417,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":418,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":419,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":420,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":421,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":422,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":423,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":424,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":425,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":426,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":427,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":428,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":429,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":430,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":431,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":432,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":433,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":434,"type":"node","deleted":false}]},{"row":[{}],"meta":[{"id":435,"type":"node","deleted":false}]}]}],"errors":[]} |



**Problem 2**. Keanu Reeves acted in the movie “John Wick” which is not in the database.

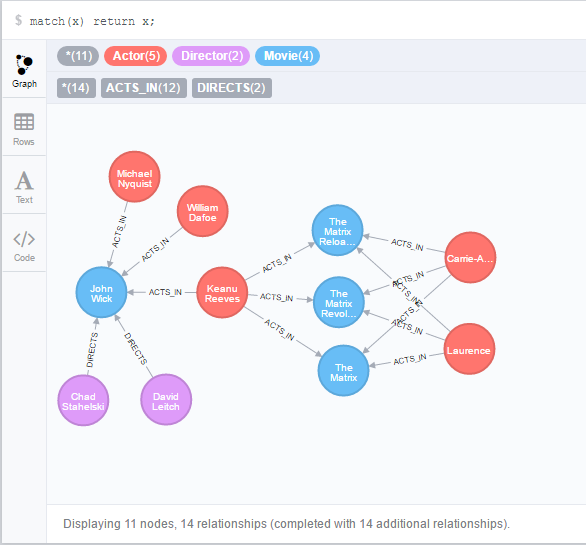
That movie was directed by Chad Stahelski and David Leitch. Cast of the movie included William Dafoe and Michael Nyquist. Add all of those people and the roles they played in this movie to the database using JAVA REST API or some other APIs for Neo4J in a language of your choice. Demonstrate that you have successfully brought data about JohnWick movie into the database. You can use Cypher Browser or any other means.

**Solution**:

1. Using python neo4j driver + bolt protocol.

|  |
| --- |
| from neo4j.v1 import GraphDatabase, basic\_auth  driver = GraphDatabase.driver("bolt://localhost:7687",auth=basic\_auth("neo4j", "neo4j"))  session = driver.session()  # create new movie and attach directors, actors and roles  session.run("MATCH (keanu:Actor { name:'Keanu Reeves'}) "  "CREATE (wick:Movie { title:'John Wick', year:'2014-05-11'}) "  "CREATE (keanu)-[:ACTS\_IN {role: 'Wick'}]->(wick) "  "CREATE (dafoe:Actor { name:'William Dafoe' }) "  "CREATE (dafoe)-[:ACTS\_IN {role: 'Marcus'}]->(wick) "  "CREATE (nyquist:Actor { name:'Michael Nyquist' }) "  "CREATE (nyquist)-[:ACTS\_IN {role: 'Viggo'}]->(wick) "  "CREATE (chad:Director { name:'Chad Stahelski' }) "  "CREATE (chad)-[:DIRECTS]->(wick) "  "CREATE (leitch:Director { name:'David Leitch' }) "  "CREATE (leitch)-[:DIRECTS]->(wick) ")  result = session.run("MATCH (m:Movie) WHERE m.title = {title} "  "RETURN m.title AS title, m.year AS year",  {"title": "John Wick"})  for record in result:  print("%s %s" %(record["title"], record["year"]))  session.close() |

2. Verify results using Cypher browser.



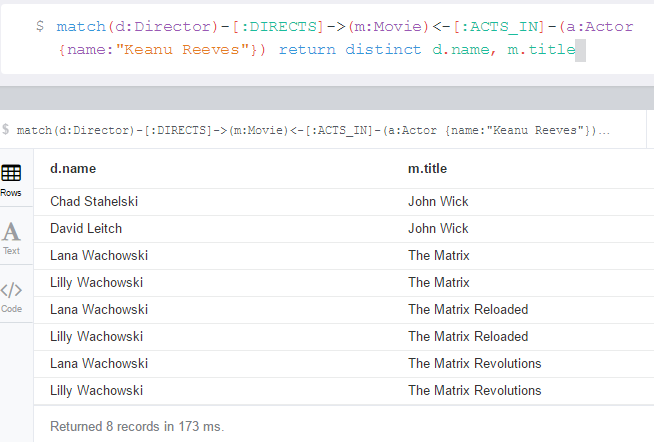
**Problem 3.** Find a list of actors playing in movies in which Keanu Reeves played. Find directors of movies in which K. Reeves played. Please use any programming language of your convenience. Verify your results using Cypher queries in Cypher Browser

1. Find a list of actors playing in movies in which Keanu Reeves played

|  |
| --- |
| from neo4j.v1 import GraphDatabase, basic\_auth  driver = GraphDatabase.driver("bolt://localhost:7687",auth=basic\_auth("neo4j", "neo4j"))  session = driver.session()  # Find a list of actors playing in movies in which Keanu Reeves played  keanu = "Keanu Reeves"  actors = session.run("match(a1:Actor)-[:ACTS\_IN]->(m:Movie)<-[:ACTS\_IN]-(a:Actor {name:\""+keanu+"\"}) return distinct a1.name as name ")  print("==== List of actors playing in movies in which %s played ====" %keanu)  for actor in actors:  print("%s " %(actor["name"]))  print("\n")  #  # Find directors of movies in which Keanu Reeves Played  # **Note**: We have added directors for one movie only (Problem 2)  #  results = session.run("match(d:Director)-[:DIRECTS]->(m:Movie)<-[:ACTS\_IN]-(a:Actor {name:\""+keanu+"\"}) return d.name as name, m.title as title ")  print("==== Directors of movies in which %s played ====" %keanu)  for record in results:  print("%s (Movie: %s)" %(record["name"], record["title"]))  session.close() |

Verify using cypher query:

*match(a1:Actor)-[:ACTS\_IN]->(m:Movie)<-[:ACTS\_IN]-(a:Actor {name:"Keanu Reeves"}) return distinct a1.name*

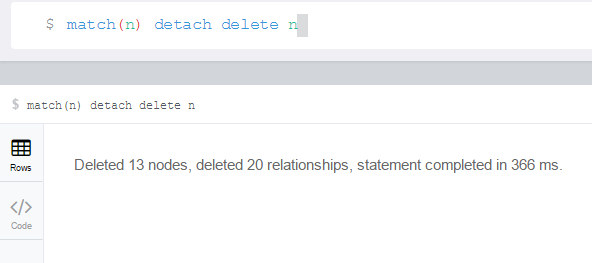


**Problem 4.** Find a way to export data from Neo4j into a set of CSV files. Delete your database and demonstrate that you can recreate the database by loading those CSV files.Please use any programming language of your convenience: Java, Python, R, or Scala.

1. Export data from database as a set of CSV files

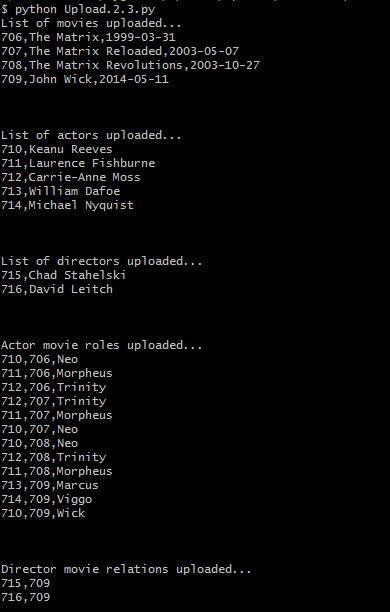
|  |
| --- |
| Please see attached python script: **Download.2.4.py**  **Output:**   1. actors.csv (id, name) 2. movies.csv (id, title, year) 3. directors.csv (id, name) 4. movie\_actor\_roles.csv (actorId, movieId, role) 5. dir\_relations.csv (directorId, movieId) |

2. Delete database contents



3. Upload CSV contents

|  |
| --- |
| Please see attached python script: Upload.2.4.py |



4. Verify db contents

