Application Design Using Java

Lecture 23

CLASS PLAN

Main Topics

- Overview of graph databases
- Installing and using Neo4j

Neo4j API

- HTTP API
  - Transactional Cypher HTTP endpoint
  - POST to a HTTP URL to send queries, and to receive responses from Neo4j
- Drivers
  - The preferred way to access a Neo4j server from an application
  - Use the Bolt protocol and have uniform design and use
  - Available in four languages: C# .NET, Java, JavaScript, and Python
  - Additional community drivers for: Spring, Ruby, PHP, R, Go, Erlang / Elixir, C/C++, Clojure, Perl, Haskell
  - API is defined independently of any programming language
- Procedures
  - Allow Neo4j to be extended by writing custom code which can be invoked directly from Cypher
  - Written in Java and compiled into jar files
  - To call a stored procedure, use a Cypher CALL clause

Neo4j Resources

- Neo4j Web site: https://neo4j.com/
- Neo4j installation manual: https://neo4j.com/docs/operations-manual/current/deployment/single-instance/
- Cypher Refcard https://neo4j.com/docs/cypher-refcard/current/
- Coursera course “Graph Analytics for Big Data” from the University of California, San Diego (https://www.coursera.org/learn/big-data-graph-analytics) has a lesson “Graph Analytics With Neo4j”
- Webber, Jim. "A programmatic introduction to Neo4j." *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, 2012.
- Robinson, Ian, James Webber, and Emil Eifrem. *Graph databases*. Sebastopol, CA: O'Reilly, 2015
- Bruggen, Rik. *Learning Neo4j*. Birmingham, UK: Packt Pub, 2014

CLASS PLAN

Main Topics

- Overview of graph databases
- Installing and using Neo4j

Neo4j Installation

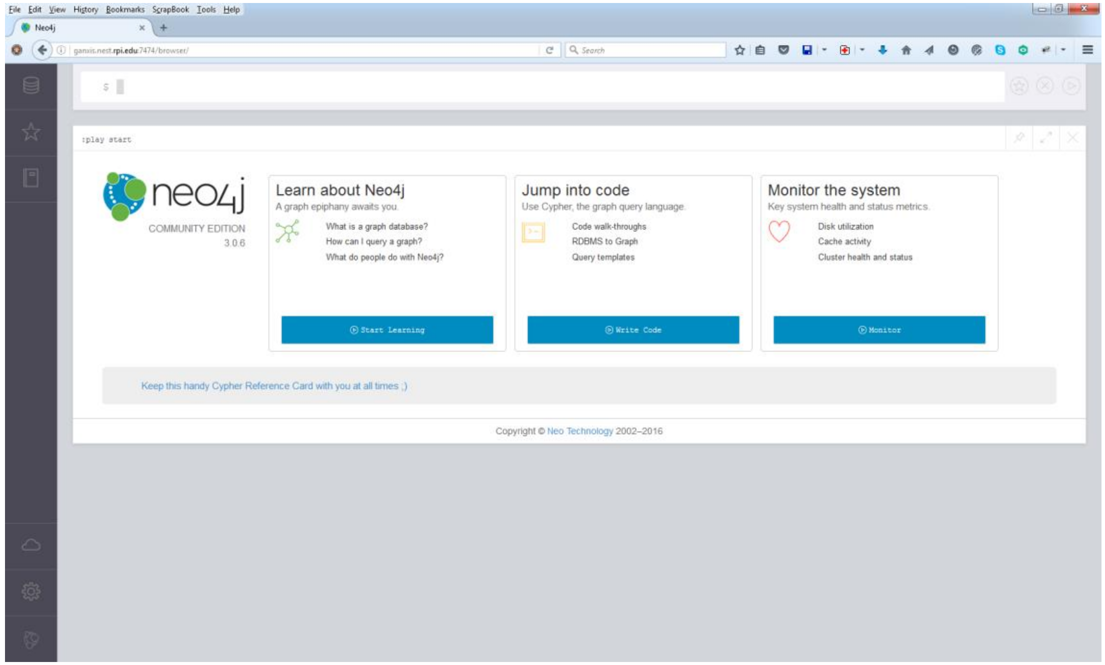
- Neo4j runs on Linux, Windows, and OS X
- A Java 8 runtime is required
- Several ways to install on Linux, depending on the Linux distro (see the “Neo4j Resources” slide)
- Check the /etc/neo4j/neo4j.conf configuration file:

```
# HTTP Connector
dbms.connector.http.type=HTTP
dbms.connector.http.enabled=true
# To accept non-local HTTP connections, uncomment this line
dbms.connector.http.address=0.0.0.0:7474
```

- File locations depend on the operating system, as described here: https://neo4j.com/docs/operations-manual/current/deployment/file-locations/
- Make sure you start the Neo4j server (e.g., “./bin/neo4j start” or “service neo4j start” on Linux)

Neo4j Browser

- Open the URL http://localhost:7474 (replace “localhost” with your server name, and 7474 with the port name as set in neo4j.conf)
- Enter the username/ password (if not set, Neo4j browser will prompt you to select the username and password)
- Start working with Neo4j by entering Cypher queries and observing their results
- Save frequently used Queries to Favorites


File Edit View History Bookmarks ScrapBook Tools Help
Neo4j
ganxis.nest.rpi.edu:7474/browser/
Search
$
neo4j
COMMUNITY EDITION
3.0.6
Start Learning
Write Code
Monitor

:play start

Learn about Neo4j

A graph epiphany awaits you.

What is a graph database?
How can I query a graph?
What do people do with Neo4j?

Jump into code

Use Cypher, the graph query language.

Code walk-throughs
RDBMS to Graph
Query templates

Monitor the system

Key system health and status metrics.

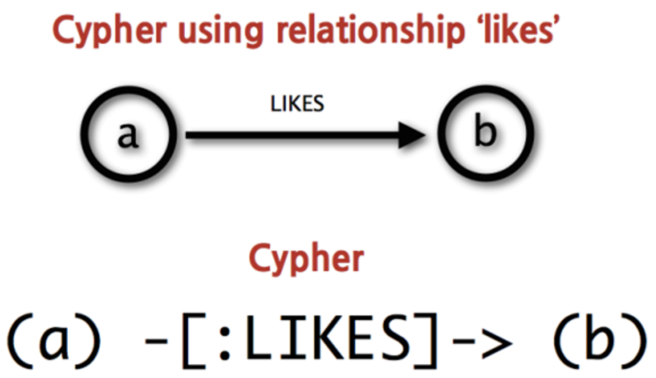
Disk utilization
Cache activity
Cluster health and status

Keep this handy Cypher Reference Card with you at all times ;)

Copyright © Neo Technology 2002–2016

The Structure of a Cypher Query

- Nodes are surrounded with parentheses which look like circles, e.g. (a)
- A relationship is basically an arrow --> between two nodes with additional information placed in square brackets inside of the arrow



Cypher using relationship 'likes'

Cypher

```
(a) -[:LIKES]-> (b)
```

- A query is comprised of several distinct clauses, like:
  - MATCH: The graph pattern to match. This is the most common way to get data from the graph.
  - WHERE: Not a clause in its own right, but rather part of MATCH, OPTIONAL MATCH and WITH. Adds constraints to a pattern, or filters the intermediate result passing through WITH.
  - RETURN: What to return.

```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof) RETURN john.name, fof.name
```

http://www.peikids.org/what-we-do/ourmission/attachment/paint-hands/

Writing Cypher Queries

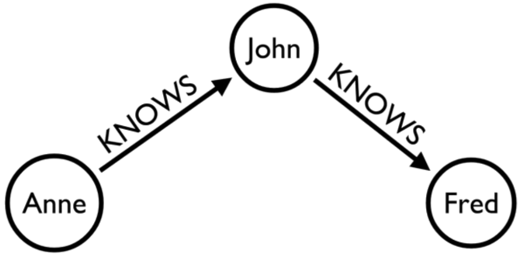
- Node labels, relationship types and property names are **case-sensitive** in Cypher
- **CREATE** creates nodes with labels and properties or more complex structures
- **MERGE** matches existing or creates new nodes and patterns. This is especially useful together with uniqueness constraints.
- **DELETE** deletes nodes, relationships, or paths. Nodes can only be deleted when they have no other relationships still existing
- **DETACH DELETE** deletes nodes and all their relationships
- **SET** sets values to properties and add labels on nodes
- **REMOVE** removes properties and labels on nodes
- **ORDER BY** is a sub-clause that specifies that the output should be sorted and how

http://www.peikids.org/what-we-do/ourmission/attachment/paint-hands/

Importing and Exporting Data

- Loading data from CSV is the most straightforward way of importing data into Neo4j
- For fast batch import of huge datasets, use the neo4j-import tool
- Lots of other tools for different data formats and database sizes
- More on importing data at https://neo4j.com/developer/guide-importing-data-and-etl/
- Export data using Neo4j browser or neo4j-shell-tools

Loading Data from CSV



- Understand your graph model

```
(p1:Person {userId:10,name:"Anne"})-[:KNOWS]->(p2:Person {userId:123,name:"John"})
```

- CSV files
  - people.csv
    `1,"John" 10,"Jane" 234,"Fred" 4893,"Mark" 234943,"Anne"`
  - friendships.csv
    `1,234 10,4893 234,1 4893,234943 234943,234 234943,1`
- Run the following Cypher queries:
  - `CREATE CONSTRAINT ON (p:Person) ASSERT p.userId IS UNIQUE;`

  -
    ```
    LOAD CSV FROM "file:///people.csv" AS csvLine
    MERGE (p:Person {userId: toInteger(csvLine[0]), name: csvLine[1]});
    ```

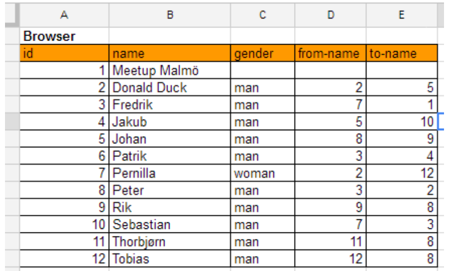
  -
    ```
    LOAD CSV FROM "file:///friendships.csv " AS csvLine MATCH (p1:Person {userId:
    toInteger(csvLine [0])}), (p2:Person {userId: toInteger(csvLine [1])})
    CREATE (p1)-[:KNOWS]->(p2);
    ```
  - `CREATE INDEX ON :Person(name);`
- Check the results:

```
MATCH (:Person {name:"Anne"})-[:KNOWS*2..2]-(p2) RETURN p2.name, count(*) as freq
ORDER BY freq DESC;
```

https://dzone.com/articles/how-neo4j-data-import-minimal

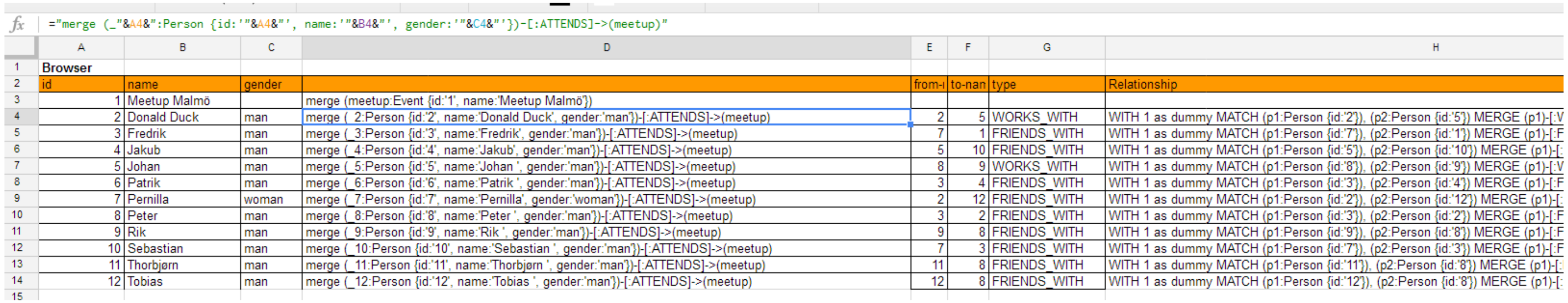
Loading Data from a Spreadsheet

- Lay out your data in a spreadsheet



A	B	C	D	E
Browser				
id	name	gender	from-name	to-name
1	Meetup Malmö			
2	Donald Duck	man	2	5
3	Fredrik	man	7	1
4	Jakub	man	5	10
5	Johan	man	8	9
6	Patrik	man	3	4
7	Pernilla	woman	2	12
8	Peter	man	3	2
9	Rik	man	9	8
10	Sebastian	man	7	3
11	Thorbjørn	man	11	8
12	Tobias	man	12	8

- Use formulas to generate the required Cypher statements



```
="merge (_"&A4&":Person {id:'"&A4&"', name:'"&B4&"', gender:'"&C4&"'})-[:ATTENDS]->(meetup)"
```

	A	B	C	D	E	F	G	H
1	Browser							
2	id	name	gender		from-i	to-nan	type	Relationship
3	1	Meetup Malmö		merge (meetup:Event {id:'1', name:'Meetup Malmö'})				
4	2	Donald Duck	man	merge (2:Person {id:'2', name:'Donald Duck', gender:'man'})-[:ATTENDS]->(meetup)	2	5	WORKS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'2'}), (p2:Person {id:'5'}) MERGE (p1)-[:V
5	3	Fredrik	man	merge (_3:Person {id:'3', name:'Fredrik', gender:'man'})-[:ATTENDS]->(meetup)	7	1	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'7'}), (p2:Person {id:'1'}) MERGE (p1)-[:F
6	4	Jakub	man	merge (_4:Person {id:'4', name:'Jakub', gender:'man'})-[:ATTENDS]->(meetup)	5	10	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'5'}), (p2:Person {id:'10'}) MERGE (p1)-[:
7	5	Johan	man	merge (_5:Person {id:'5', name:'Johan ', gender:'man'})-[:ATTENDS]->(meetup)	8	9	WORKS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'8'}), (p2:Person {id:'9'}) MERGE (p1)-[:V
8	6	Patrik	man	merge (_6:Person {id:'6', name:'Patrik ', gender:'man'})-[:ATTENDS]->(meetup)	3	4	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'3'}), (p2:Person {id:'4'}) MERGE (p1)-[:F
9	7	Pernilla	woman	merge (_7:Person {id:'7', name:'Pernilla', gender:'woman'})-[:ATTENDS]->(meetup)	2	12	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'2'}), (p2:Person {id:'12'}) MERGE (p1)-[:
10	8	Peter	man	merge (_8:Person {id:'8', name:'Peter ', gender:'man'})-[:ATTENDS]->(meetup)	3	2	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'3'}), (p2:Person {id:'2'}) MERGE (p1)-[:F
11	9	Rik	man	merge (_9:Person {id:'9', name:'Rik ', gender:'man'})-[:ATTENDS]->(meetup)	9	8	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'9'}), (p2:Person {id:'8'}) MERGE (p1)-[:F
12	10	Sebastian	man	merge (_10:Person {id:'10', name:'Sebastian ', gender:'man'})-[:ATTENDS]->(meetup)	7	3	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'7'}), (p2:Person {id:'3'}) MERGE (p1)-[:F
13	11	Thorbjørn	man	merge (_11:Person {id:'11', name:'Thorbjørn ', gender:'man'})-[:ATTENDS]->(meetup)	11	8	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'11'}), (p2:Person {id:'8'}) MERGE (p1)-[:
14	12	Tobias	man	merge (_12:Person {id:'12', name:'Tobias ', gender:'man'})-[:ATTENDS]->(meetup)	12	8	FRIENDS_WITH	WITH 1 as dummy MATCH (p1:Person {id:'12'}), (p2:Person {id:'8'}) MERGE (p1)-[:
15								

- Collect Cypher queries and run them
- Check the results:

```
MATCH (p1:Person)-[:ATTENDS]-(e:Event{name:"Meetup Malmö"})-[:ATTENDS]-
(p2:Person) WHERE (p1)-[:FRIENDS_WITH]-(p2) RETURN p1, p2, e;
```

Based on a blog post by Rik Van Bruggen (https://neo4j.com/blog/importing-data-into-neo4j-the-spreadsheet-way/)

Loading Data from a GraphML file

- Use **neo4j-shell-tools** from https://github.com/jexp/neo4j-shell-tools
- Populate the database from a GraphML file

```
import-graphml -i /usr/share/neo4j/import/airlines.graphml -r
HAS_DIRECT_FLIGHTS_TO -b 20000 -c -t
```

- Check the results:

```
MATCH (a)--()
WITH a.tooltip as airport, count(*) as flights
RETURN airport, flights ORDER BY flights DESC LIMIT 10
```

Loading Data from an Arbitrary Format

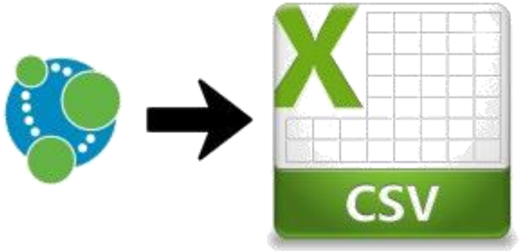
- Write a simple program to convert your file into a set of two CSV files
- Load data from the CSV file into a Neo4j database
  - `CREATE CONSTRAINT CharacterNameUnique ON (c:Character) ASSERT c.name IS UNIQUE;`
  - `LOAD CSV WITH HEADERS FROM "file:///Marvel-nodes.csv" AS csvLine MERGE (c:Character {name: csvLine.NodeID});`
  - `LOAD CSV WITH HEADERS FROM "file:///Marvel-edges.csv" AS csvLine MATCH (c1:Character {name: csvLine.EdgeFrom}), (c2:Character {name: csvLine.EdgeTo}) CREATE (c1)-[:APPEARED_WITH]->(c2);`
  - `CREATE INDEX FOR (c:Character) on (c.name);`
- Check the results:

```
MATCH (c:Character)-[r]-()
WITH c as characters, count(distinct r) as degree
RETURN degree, count(characters) ORDER BY degree ASC
```

```
from sys import argv


def read_edge_list(filename):
    nodeset= set([])
    edgelist = []
    with open(filename, 'r') as file_handle:
        for line in file_handle:
            if line[0] != '#':
                data = line.split('","')
                node_from = data[0] + '"'
                node_to = '"' + data[1].strip()
                nodeset.add(node_from)
                nodeset.add(node_to)
                edgelist.append([node_from, node_to])
    return nodeset, edgelist


def write_csv_nodes(nodes, file_nodes):
    with open(file_nodes, 'w') as file_handle:
        file_handle.write("NodeID\n")
        for node in nodes:
            file_handle.write('{0}\n'.format(node))



def write_csv_edges(edges, file_nodes):
    with open(file_nodes, 'w') as file_handle:
        file_handle.write("EdgeFrom,EdgeTo\n")
        for edge in edges:

file_handle.write('{0},{1}\n'.format(edge[0], edge[1]))

script, input_file, output_file_nodes,
output_file_edges = argv
nodes, edges = read_edge_list(input_file)
write_csv_nodes(nodes, output_file_nodes)
write_csv_edges(edges, output_file_edges)
```

Wikipedia vote network from http://snap.stanford.edu/data/wiki-Vote.html

Exporting Data From Neo4j

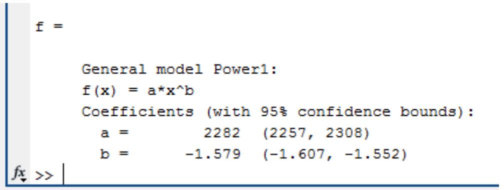


- Click the download icon on the table view of the Cypher query results
- Use neo4j-shell-tools to export results of a Cypher query to a CSV or GraphML file
- Access the graph data with Neo4j API and save it in the desired format

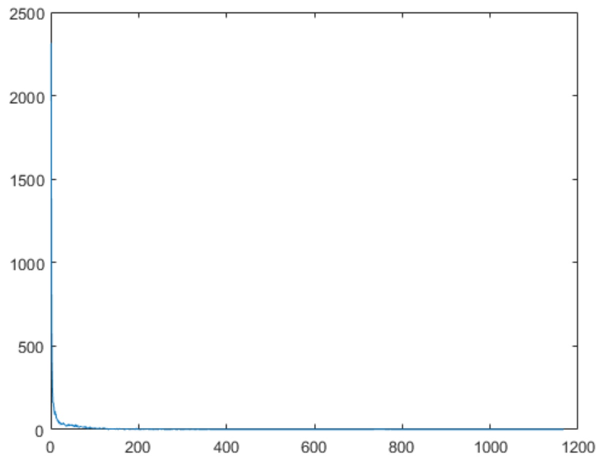
Analyzing Graph Data with MATLAB

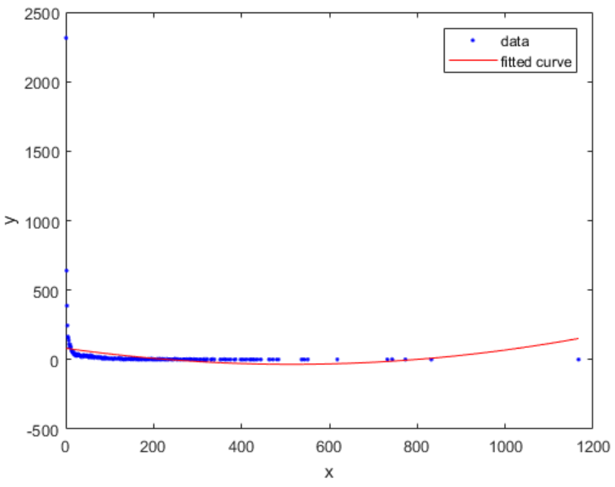
- Load CSV data exported from Neo4j into MATLAB
- Use MATLAB to perform additional analysis and to draw plots
- Export analysis results and plots for publication

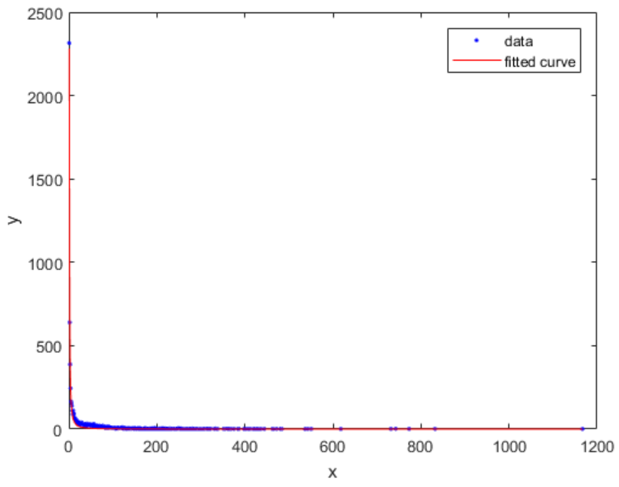
```
filename = 'filename.csv';
M = csvread(filename,1,0);
x = M(:,1);
y = M(:,2);
plot(x,y)
f=fit(x,y,'poly2')
plot(f,x,y)
f=fit(x,y,'power1')
plot(f,x,y)
```



```
f =

     General model Power1:
     f(x) = a*x^b
     Coefficients (with 95% confidence bounds):
       a =        2282  (2257, 2308)
       b =      -1.579  (-1.607, -1.552)
```







Using Transactional Cypher HTTP Endpoint

- Allows you to execute a series of Cypher statements within the scope of a transaction
- The transaction may be kept open across multiple HTTP requests, until the client chooses to commit or roll back
- Each HTTP request can include a list of statements
- Requests should include an Authorization header, with a value of Basic <payload>, where "payload" is a base64 encoded string of "username:password"

```
import requests
from requests.exceptions import ConnectionError
import json

NEO4J_SERVER = 'http://localhost:7474'
NEO4J_COMMIT_ENDPOINT = '/db/data/transaction/commit'
NEO4J_CREDENTIALS = ‘************'

def execute_neo4j_cypher(url, credentials, query, parameters):
    result = None
    query_text = json.dumps(dict(statements = [dict(statement =
query, parameters = parameters)]))

    headers = {'Accept' : 'application/json', 'Content-type' :
'application/json', 'Authorization:' : 'Basic ' + credentials}
    try:
        resp = requests.post(url, headers = headers, data =
query_text)
        result = resp.json()
    except ConnectionError as exception:
        print exception # Log error
    if len(result['errors']) > 0:
        print '@@@ ERROR! Error executing Cypher query' # Log
error
        print '@@@ ', query, '<-', parameters
        print '@@@ ' + str(result)
    return result

query = 'MERGE (p: Person {id:{userid}, name:{name}}) ON CREATE
SET p.created = timestamp() ON MATCH SET p.matched =
timestamp() RETURN p'
parameters = dict()
parameters['userid'] = 17
parameters['name'] = 'J J'
execute_neo4j_cypher(NEO4J_SERVER + NEO4J_COMMIT_ENDPOINT,
NEO4J_CREDENTIALS, query, parameters)
```

Using Drivers to Access Neo4j

- Binary Bolt protocol (starting with Neo4j 3.0)
- Binary protocol is enabled in Neo4j by default and can be used in any language driver that supports it
- Native Java driver officially supported by Neo4j
- Drivers implement all low level connection and communication tasks

```
import org.neo4j.driver.v1.*;

public class Neo4j
{
  public static void javaDriverDemo() {
     Driver driver = GraphDatabase.driver("bolt://localhost", "neo4j", "neo4j"));
     Session session = driver.session();

     StatementResult result = session.run("MATCH (a)-[]-(b)-[]-(c)-[]-(a) WHERE a.id < b.id AND b.id < c.id RETURN DISTINCT a,b,c");
     int counter = 0;
     while (result.hasNext())
     {
               counter++;
               Record record = result.next();
               System.out.println(record.get("a").get("id") + " \t" + record.get("b").get("id") + " \t" + record.get("c").get("id"));
     }
     System.out.println("Count: " + counter);
     session.close();
     driver.close();
  }
  public static void main(String [] args)
  {
     javaDriverDemo();
  }
}
```

Using Core Java API

- Native Java API performs database operations directly with Neo4j core

```
import java.io.*;
import java.util.*;
import org.neo4j.graphdb.*

public class Neo4j
{
   public enum NodeLabels implements Label { NODE; }
   public enum EdgeLabels implements RelationshipType{ CONNECTED; }
   public static void javaNativeDemo(int nodes, double p) {
      Node node1, node2; Random randomgen = new Random();
      GraphDatabaseFactory dbFactory = new GraphDatabaseFactory();
      GraphDatabaseService db = dbFactory.newEmbeddedDatabase(new File("TestNeo4jDB"));
      try (Transaction tx = db.beginTx()) {
         for (int i = 1; i <= nodes; i++) {
            Node node = db.createNode(NodeLabels.NODE);
            node.setProperty("id", i);
         }
         for (int i = 1; i <= nodes; i++)
            for (int j = i + 1; j <= nodes; j++) {
               if (randomgen.nextDouble() < p) {
                  node1 = db.findNode(NodeLabels.NODE, "id", i);
                  node2 = db.findNode(NodeLabels.NODE, "id", j);
                  Relationship relationship = node1.createRelationshipTo(node2,EdgeLabels.CONNECTED);
                  relationship = node2.createRelationshipTo(node1,EdgeLabels.CONNECTED);
               }
            }
         tx.success();
      }
      db.shutdown();
   }
   public static void main(String [] args) {
      javaNativeDemo(100, 0.2); }
}
```

//TODO before next lecture:

- Homework 5 due on 4/30 at 11:59 pm EDT. Must be submitted on Submitty.