

Please find two other learning partners,

- ▶ form a standing group and
- ▶ tell them what you already know about
  - ▶ graphs,
  - ▶ graph databases and
  - ▶ Neo4j.

# Graph Data - Modelling and Querying

## with Neo4j and Cypher

Iryna Feuerstein

Open Data Science Conference

12 October 2017



# Agenda



- What are graphs?

  - Definition

  - Typical use cases

  - Not so typical use cases

- Starting with Neo4j and Cypher

  - Starting the database

  - Brief look at the configuration

  - CRUD operations with Cypher

  - Node operations

  - Relation operations

- Querying for paths and patterns

- Using graph algorithms

  - apoc library

  - algo library

- Importing data

  - Basic loading command

  - Performance issues and dealing with big data

## Definition

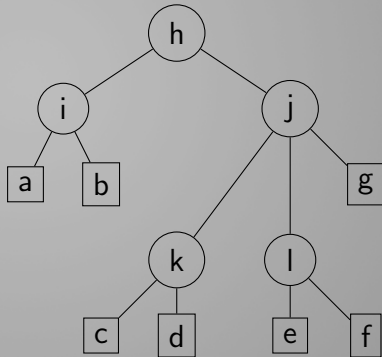
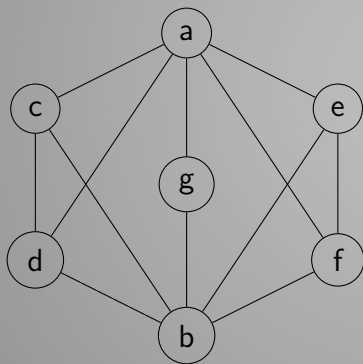
*Graph* is an ordered pair  $G = (V, E)$  comprising a set  $V$  of *vertices*, *nodes* or *points* together with a set  $E$  of *edges*, *arcs* or *lines*, which are 2-element subsets of  $V$ .<sup>1</sup>

---

<sup>1</sup>[en.wikipedia.org/wiki/Graph\\_\(discrete\\_mathematics\)](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics))

## Definition

*Graph* is an ordered pair  $G = (V, E)$  comprising a set  $V$  of *vertices*, *nodes* or *points* together with a set  $E$  of *edges*, *arcs* or *lines*, which are 2-element subsets of  $V$ .<sup>1</sup>



<sup>1</sup>[en.wikipedia.org/wiki/Graph\\_\(discrete\\_mathematics\)](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics))

# Use Cases



- ▶ Networks
  - ▶ Social networks

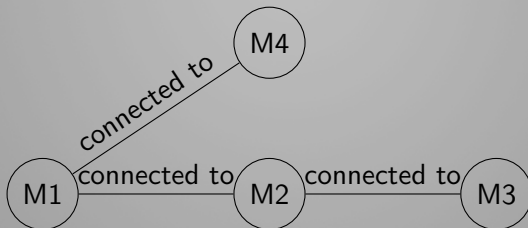


- ▶ Networks

- ▶ Social networks

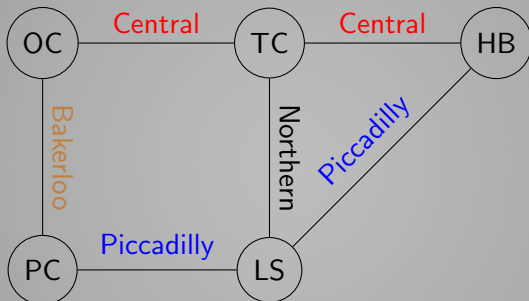


- ▶ Computer networks





- Networks
  - Transport networks



OC = Oxford Circus

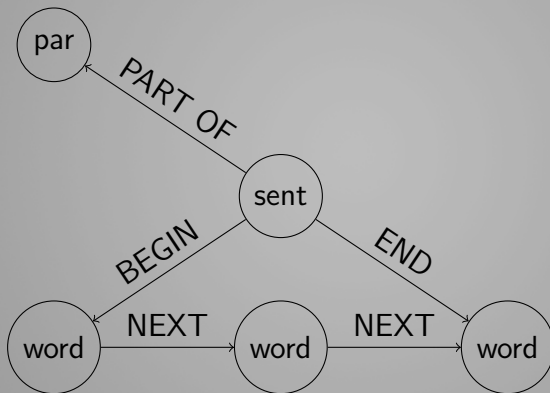
TC = Tottenham Court Road

HB = Holborn

LS = Leicester Square

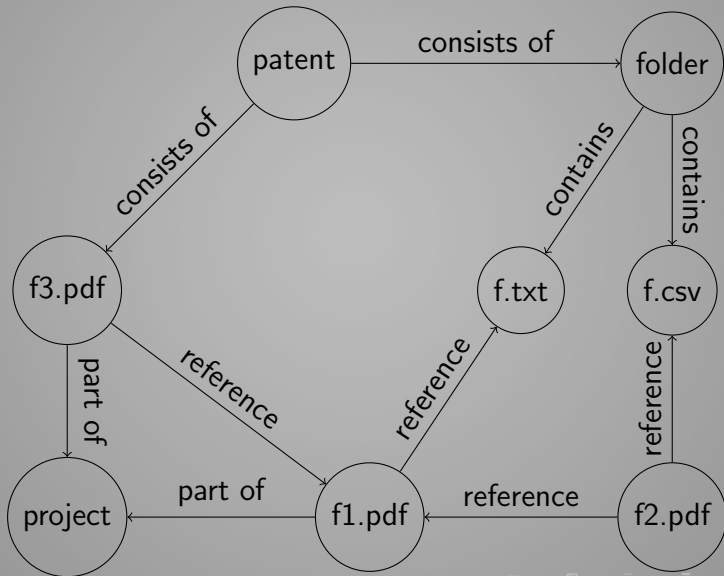
PC = Piccadilly Circus

- Natural Language Processing

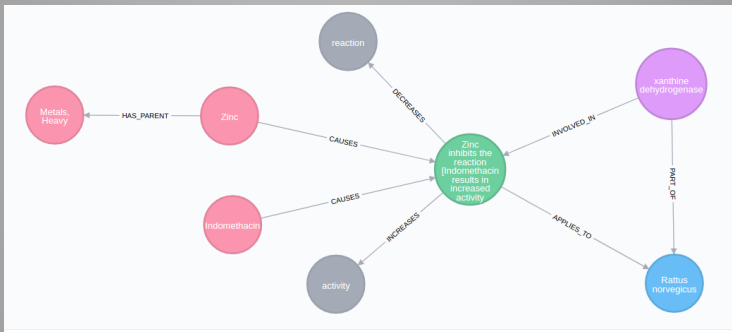


# Use Cases

- Document management



## ► Biochemistry / Genomics



<sup>1</sup><http://ctdbase.org/>

- ▶ Find the right installation file for your OS at `neo4j-training-files/neo4j` on the flash drive and install the software.

- ▶ Find the right installation file for your OS at `neo4j-training-files/neo4j` on the flash drive and install the software.
- ▶ Copy both JAR-files from the `neo4j-training-files/plugins` directory into `NEO4J_HOME/plugins` directory of your Neo4j installation.

- ▶ Find the right installation file for your OS at `neo4j-training-files/neo4j` on the flash drive and install the software.
- ▶ Copy both JAR-files from the `neo4j-training-files/plugins` directory into `NEO4J_HOME/plugins` directory of your Neo4j installation.
- ▶ Replace the `NEO4J_HOME/conf/neo4j.conf` configuration file with the one found on the flash drive at `neo4j-training-files/conf/neo4j.conf`.

- ▶ Find the right installation file for your OS at `neo4j-training-files/neo4j` on the flash drive and install the software.
- ▶ Copy both JAR-files from the `neo4j-training-files/plugins` directory into `NEO4J_HOME/plugins` directory of your Neo4j installation.
- ▶ Replace the `NEO4J_HOME/conf/neo4j.conf` configuration file with the one found on the flash drive at `neo4j-training-files/conf/neo4j.conf`.
- ▶ Copy the `neo4j-training-files/data/odsc.db` folder into your `NEO4J_HOME/data/databases/` directory



# Starting Neo4j



- ▶ Start the database with  
`NEO4J_HOME/bin/neo4j start`

# Starting Neo4j



- ▶ Start the database with  
`NEO4J_HOME/bin/neo4j start`
- ▶ Go to `http://localhost:7474` within you browser

## Important configuration entries

```
dbms.active_database=odsc.db
```

```
dbms.security.auth_enabled=false
```

```
dbms.security.procedures.unrestricted=algo.*,apoc.*
```

```
apoc.import.file.enabled=true
```

- ▶ create node

```
CREATE (c:Chemical {name: 'Helium'}) RETURN c
```

- ▶ update node

```
MERGE (c:Chemical {name: 'Helium'}) SET c.symbol =  
'He' RETURN c
```

- ▶ delete node

- ▶ without relations

```
MATCH (c:Chemical {name:'Helium'}) DELETE c  
MATCH (c:Chemical)  
    WHERE c.name = 'Helium'  
    DELETE c
```

- ▶ with existing relations

```
MATCH (c:Chemical {name:'Helium'})  
    DETACH DELETE c
```

- ▶ create relation

- ▶ between new nodes

- ```
CREATE (c:Chemical chemicalName:'Helium')-[:BELONGS_TO]->(g:ChemicalGroup groupName:'Noble gases') RETURN c,g
```

- ▶ between existing nodes

- ```
MATCH (g:ChemicalGroup groupName:'Noble gases'), (p:ChemicalGroup groupName:'Gases') CREATE (g)-[:HAS_PARENT]->(p) RETURN g,p
```

- ▶ update relation

- ```
MATCH ()-[:BELONGS_TO]-() SET r.updateTime = timestamp() RETURN r
```

- ▶ delete relation

- ```
MATCH ()-[:BELONGS_TO]-() DELETE r
```

## Examples:

- ▶ `MATCH (g:Gene) WHERE g.geneSymbol = 'CTSD'`  
`RETURN g`
- ▶ `MATCH (g:Gene)i-[:ASSOCIATED_WITH]-(d:Disease)`  
`WHERE g.geneSymbol = 'CTSD' RETURN g, d`
- ▶ `MATCH (g:Gene)i-[:ASSOCIATED_WITH]-(d:Disease)`  
`WHERE g.geneSymbol = 'CTSD' RETURN g, count(d)`
- ▶ `MATCH (g:Gene)i-[:ASSOCIATED_WITH]-(d:Disease)`  
`WITH g, count(d) as diseases WHERE diseases < 50`  
`RETURN g.geneName, g.geneSymbol, diseases ORDER`  
`BY diseases DESC`
- ▶ `MATCH (g:Gene)i-[:ASSOCIATED_WITH]-(d:Disease)-`  
`[:ASSOCIATED_WITH]-(otherGene:Gene) WHERE`  
`g.geneSymbol = 'CTSD' AND d.diseaseName =`  
`'Osteoarthritis' RETURN otherGene.geneName,`  
`otherGene.geneSymbol`

- ▶ MATCH p = (c:Chemical)-[\*2]-(d:Disease) where d.diseaseName STARTS WITH 'Osteo' RETURN p LIMIT 20
- ▶ MATCH (c:Chemical)i-[:HAS\_PARENT\*3..4]-(descendant:Chemical) WITH c, count(descendant) AS descendants , collect(descendant.chemicalName) as names ORDER BY descendants DESC LIMIT 10 RETURN c.chemicalName, names[1..10], descendants
- ▶ MATCH (c:Chemical) WHERE c.chemicalName = 'Zinc Acetate' MATCH (d:Disease) WHERE d.diseaseName = 'Alzheimer Disease' MATCH p = (c)-[\*1..3]-(d) RETURN p LIMIT 20
- ▶ MATCH (c:Chemical chemicalName:'Zinc Acetate'), (d:Disease diseaseName:'Alzheimer Disease') MATCH path = allShortestPaths( (c)-[\*..3]-(d) ) RETURN path

# Calling procedures



- ▶ CALL db.schema
- ▶ CALL dbms.procedures
- ▶ call dbms.functions
- ▶ CALL apoc.help('dijkstra')



## Definition

In a connected graph, the normalized closeness centrality (or closeness) of a node is the average length of the shortest path between the node and all other nodes in the graph. Thus the more central a node is, the closer it is to all other nodes.<sup>2</sup>

## Closeness Centrality Example

```
MATCH (node:Chemical) WHERE node.chemicalName
CONTAINS 'Vitamin' WITH collect(node) AS nodes CALL
apoc.algo.closeness(['HAS_PARENT'],nodes,'BOTH') YIELD
node, score RETURN node, score ORDER BY score DESC
```

---

<sup>2</sup>[en.wikipedia.org/wiki/Centrality#Closeness\\_centrality](https://en.wikipedia.org/wiki/Centrality#Closeness_centrality)

## Definition

Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.<sup>3</sup>

## Betweenness Centrality Example

```
MATCH (node:Disease) WHERE node.diseaseName
CONTAINS 'deficiency' WITH collect(node) AS nodes CALL
apoc.algo.betweenness(['HAS_PARENT'],nodes,'BOTH')
YIELD node, score RETURN node.diseaseName, score
ORDER BY score DESC LIMIT 10
```

---

<sup>3</sup>[en.wikipedia.org/wiki/Centrality#Betweenness\\_centrality](https://en.wikipedia.org/wiki/Centrality#Betweenness_centrality) 🔍

## PageRank example

```
call algo.pageRank.stream('InteractionType',  
'HAS_PARENT',iterations:20) YIELD node, score WITH *  
ORDER BY score DESC LIMIT 5 RETURN node.typeName,  
node.code, score;
```

```
MATCH (n:Movie) CALL apoc.create.addLabels( id(n), [
n.genre ] ) YIELD node REMOVE node.genre RETURN node
CALL apoc.periodic.iterate( "MATCH (p:Person) WHERE
(p)-[:ACTED_IN]->() RETURN p", "SET p:Actor",
batchSize:10000, parallel:true)
call apoc.refactor.setType(rel, 'NEW-TYPE')
```