Introduction

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

Learning goals

- What are graphs?
  - Definition
  - Use cases
- Starting with Neo4j and Cypher
  - Configuration and start
  - CRUD operations with Cypher
- Quering for paths and patterns
- Using graph algorithms
  - apoc library
  - algo library
- Importing data
- Refactoring graph data model

Graph

Definition

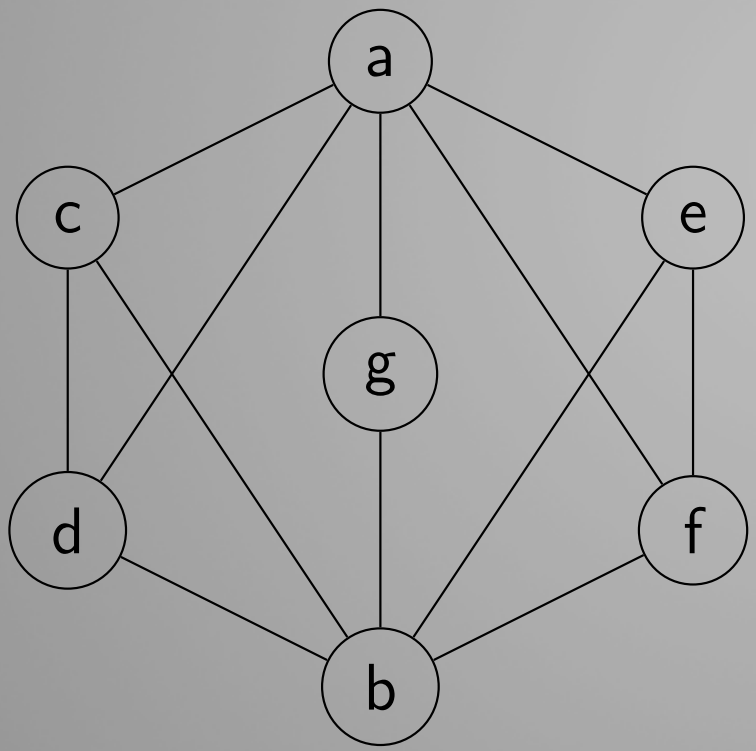
Graph is an ordered pair $G = (V, E)$ comprising a set V of *vertices*, *nodes* or *points* together with a set E of *edges*, *arcs* or *lines*, which are 2-element subsets of V.[1]

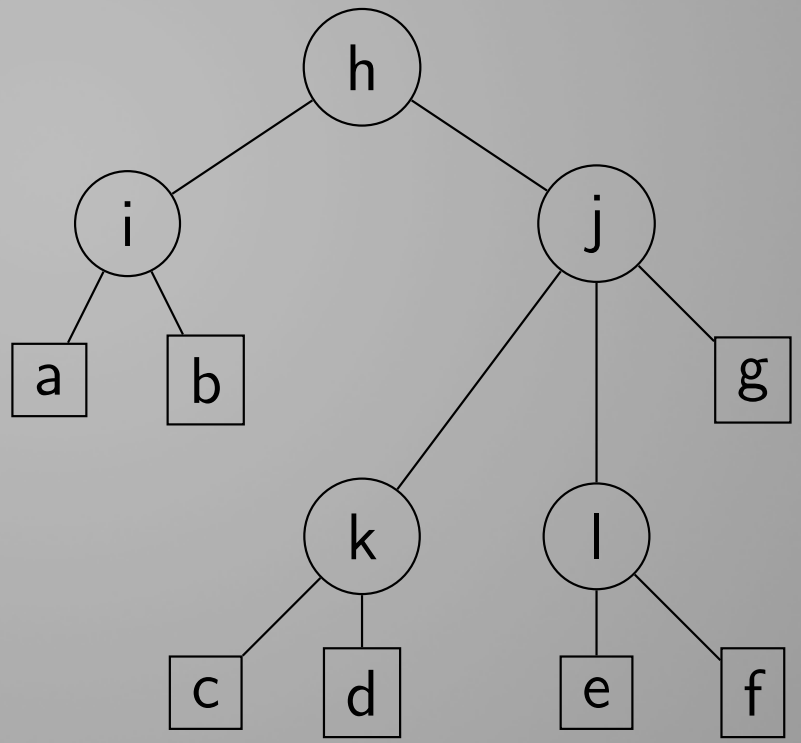
[1] en.wikipedia.org/wiki/Graph_(discrete_mathematics)

Graph

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.[1]



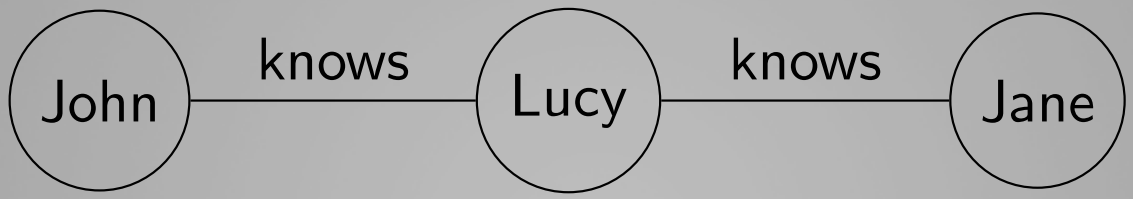


[1] en.wikipedia.org/wiki/Graph_(discrete_mathematics)

Use Cases

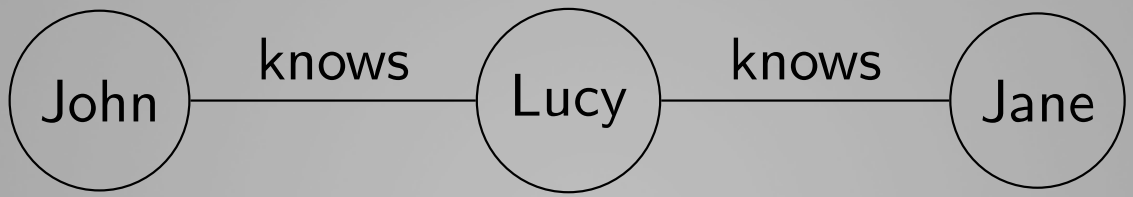
Use Cases

- Networks
  - Social networks

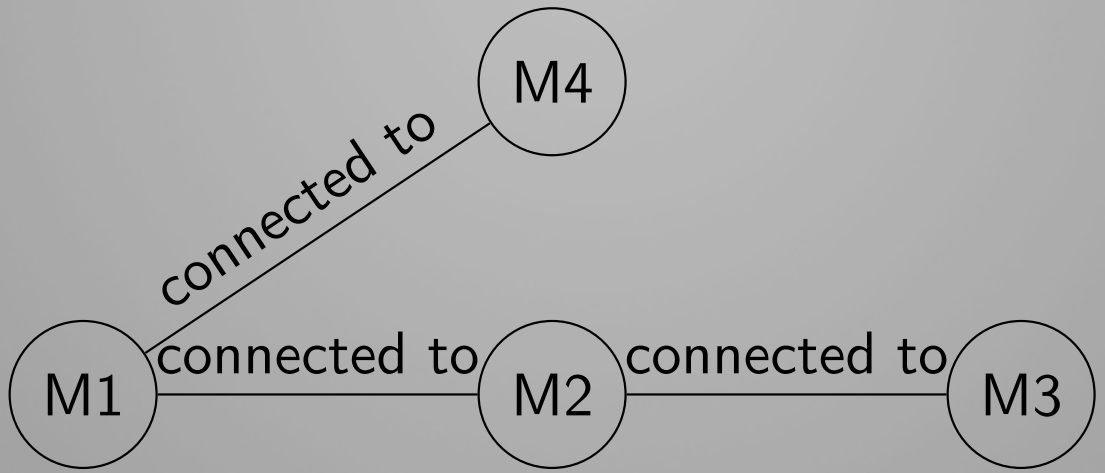


Use Cases

- Networks
  - Social networks

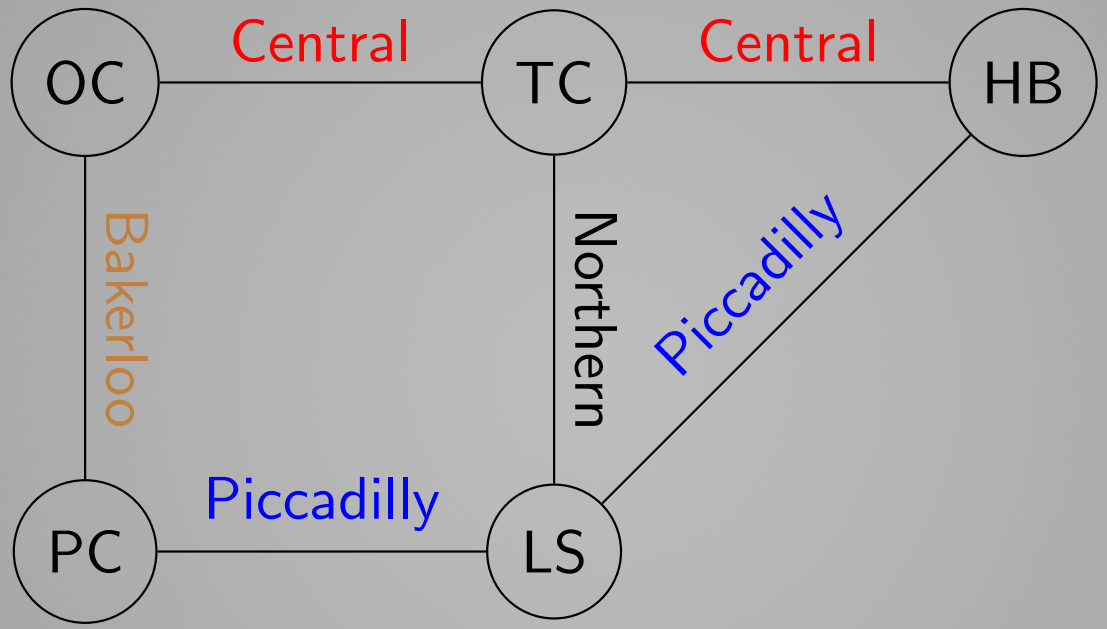


  - Computer networks



Use Cases

- Networks
  - Transport networks



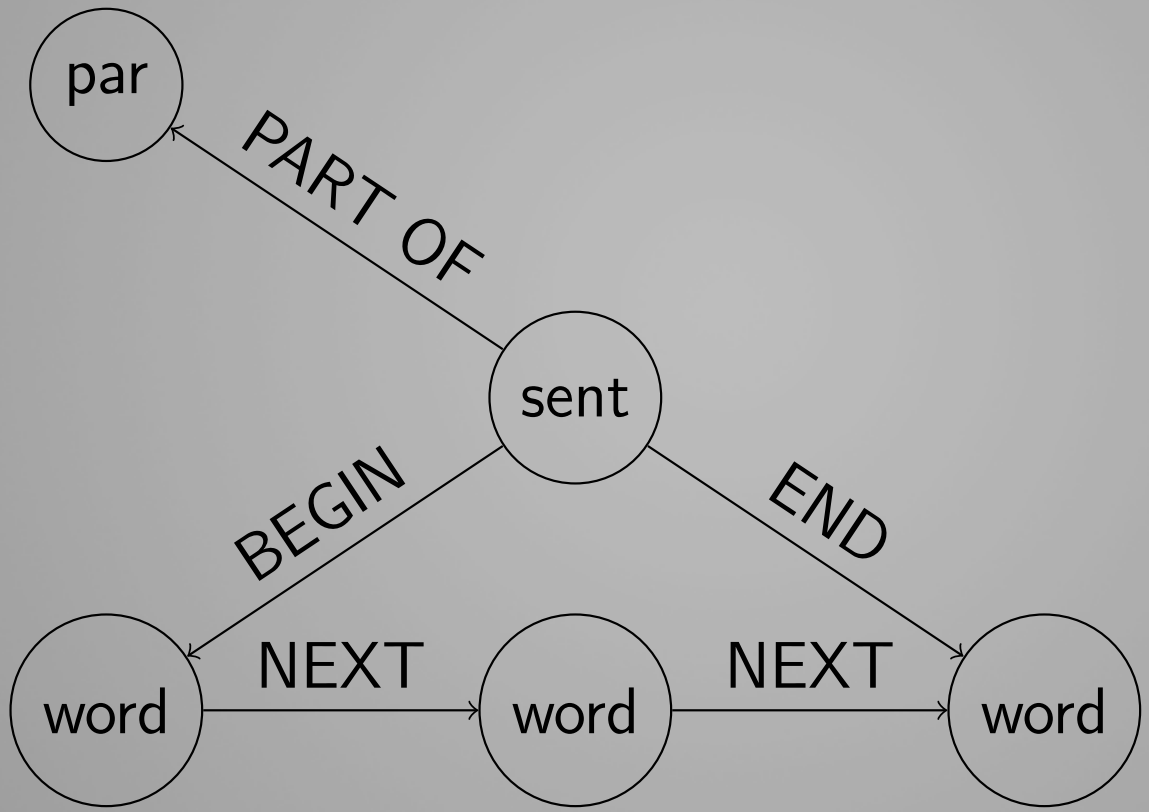
$OC =$ Oxford Circus, $LS =$ Leicester Square

$HB =$ Holborn, $PC =$ Piccadilly Circus

$TC =$ Tottenham Court Road

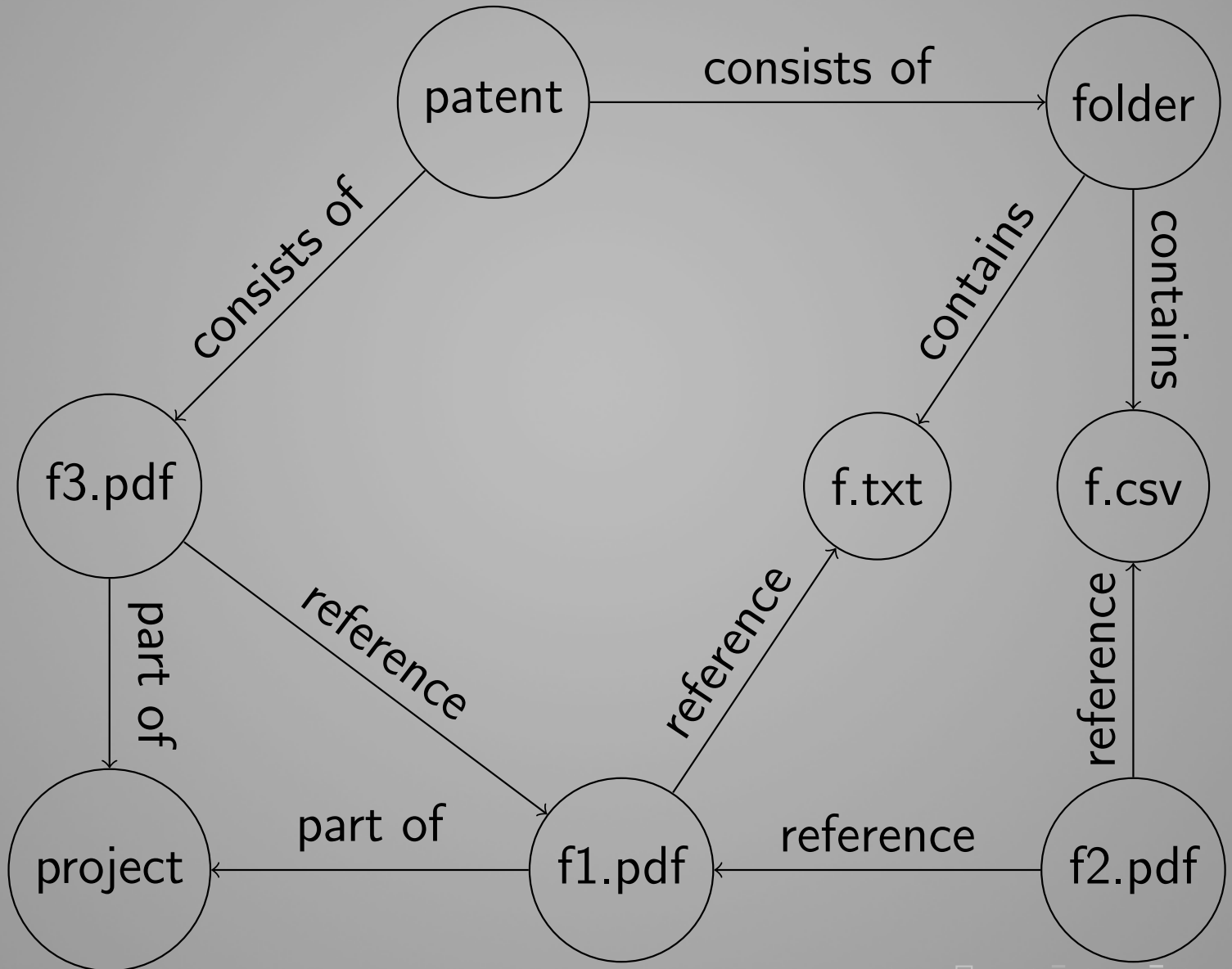
Use Cases

- Natural Language Processing



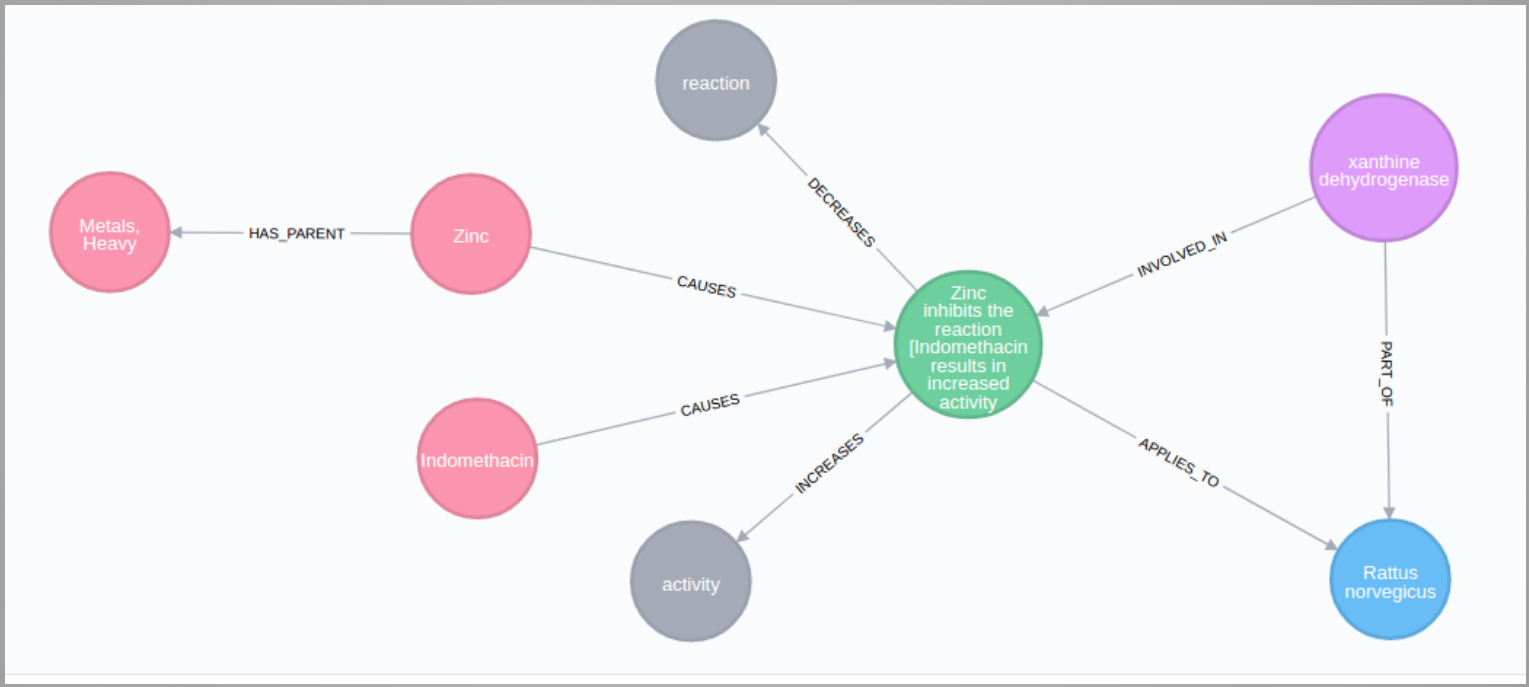
Use Cases

- Document management



Use Cases

- Biochemistry / Genomics



[1] http://ctdbase.org/

Installation

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

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.

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`.

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`.
- 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

Demonstration

Important configuration entries

dbms.active_database=odsc.db
dbms.security.auth_enabled=false
dbms.security.procedures.unrestricted=algo.*,apoc.*
apoc.import.file.enabled=true

Live coding session - CRUD operations

create node

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

update node

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

Live coding session - CRUD operations

delete node without relations

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

delete node without relations

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

delete node with existing relations

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

Live coding session - CRUD operations

create relation between new nodes

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

create relation between existing nodes

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

Live coding session - CRUD operations

update relation

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

delete relation

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

Live coding session

Check your indexes

```
CALL db.indexes
CREATE INDEX ON :Disease(diseaseId)
CREATE INDEX ON :Gene(geneName, geneSymbol)
```

Live coding session

Example

```
MATCH (g:Gene)
      WHERE g.geneSymbol = 'CTSD'
      RETURN g
```

Example

```
MATCH (g:Gene)<-[:ASSOCIATED_WITH]-(d:Disease)
      WHERE g.geneSymbol = 'CTSD'
      RETURN g, d
```

Example

```
MATCH (g:Gene)<-[:ASSOCIATED_WITH]-(d:Disease)
      WHERE g.geneSymbol = 'CTSD'
      RETURN g, count(d)
```

Live coding session

Example

```
MATCH (g:Gene)<-[:ASSOCIATED_WITH]-(d:Disease)
      WITH g, count(d) as diseases
      WHERE diseases >50
      RETURN g.geneName, g.geneSymbol, diseases
      ORDER BY diseases DESC
```

Example

```
MATCH (g:Gene)<-[:ASSOCIATED_WITH]-(d:Disease)
     -[:ASSOCIATED_WITH]->(otherGene:Gene)
      WHERE g.geneSymbol = 'CTSD'
      AND d.diseaseName = 'Osteoarthritis'
RETURN otherGene.geneName, otherGene.geneSymbol
```

Live coding session

Example

```
MATCH p = (c:Chemical)-[*2]-(d:Disease)
      WHERE d.diseaseName STARTS WITH 'Osteo'
      RETURN p LIMIT 20
```

Example

```
MATCH (c:Chemical)<-[:HAS_PARENT*3..4]-(d:Chemical)
      WITH c, count(d) AS descendants,
      collect(d.chemicalName) AS names
      ORDER BY descendants DESC LIMIT 10
RETURN c.chemicalName, names[1..10], descendants
```

Live coding session

Example

```
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
```

Example

```
MATCH (:InteractionType {typeName:'degradation'})
      <-[:INCREASES|:DECREASES]-
      (i:Interaction)-[:APPLIES_TO]->
      (:Organism {organismName:'Cricetulus griseus'})
RETURN i.description
```

Live coding session

Shortest path example

```
MATCH (zinc:Chemical {chemicalName:'Zinc Acetate'}),
      (metals:Chemical {chemicalName:'Metals, Heavy'}),
      p = shortestPath((zinc)-[*..15]-(metals))
RETURN p
```

Shortest path example

```
MATCH (zinc:Chemical {chemicalName:'Zinc Acetate'}),
      (metals:Chemical {chemicalName:'Metals, Heavy'}),
      p = shortestPath((zinc)-[*..15]-(metals))
WHERE NONE(r IN relationships(p)
                WHERE type(r)='CAUSES')
RETURN p
```

Live coding session

Shortest path example

```
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')

Live coding session

Definition

In a connected graph, the normalized *closeness centrality* 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.[2]

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
```

[2] en.wikipedia.org/wiki/Centrality#Closeness_centrality

Live coding session

Definition

Betweenness centrality quantifies the number of times a node acts as a bridge along the shortest path between two other nodes.[3]

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
```

[3] en.wikipedia.org/wiki/Centrality#Betweenness_centrality

Live coding session


```
MATCH (startNode:Category
                    {name:'Endocrine system disease'})
      CALL apoc.algo.cliquesWithNode(startNode, 4)
      YIELD clique
RETURN clique
```

Live coding session

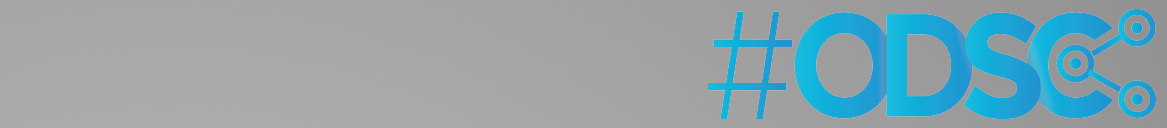
PageRank example

```
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;
```

Partitioning into connected components

```
CALL algo.unionFind('InteractionType',
                'HAS_PARENT', write:true,
                partitionProperty:"partition")
YIELD nodes, setCount, loadMillis,
                computeMillis, writeMillis
```

Live coding session



Closeness

```
CALL algo.closeness('Chemical', 'HAS_PARENT',
        {write:true, writeProperty:'centrality'})
YIELD nodes, loadMillis, computeMillis, writeMillis
```

Closeness

```
MATCH (c:Chemical)
        WHERE c.centrality >200
RETURN c.chemicalName, c.centrality
        ORDER BY c.centrality DESC LIMIT 10
```

LOAD CSV

View the data

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM
"file:///.../_Disease-GO_biological_process_associations.csv"
AS line
RETURN line LIMIT 10
```

[3] http://ctdbase.org/

LOAD CSV

Import the data

```
USING PERIODIC COMMIT 500
LOAD CSV WITH HEADERS FROM
"file:///.../Disease-GO_biological_process_associations.csv"
      AS line LIMIT 20
MATCH (d:Disease)
      WHERE last(split(d.diseaseID,':')) = line.DiseaseID
MERGE (b:BiologicalProcess goid:line.GOID)
      SET b.goName = line.GOName
MERGE (b)<-[:AFFECTED_BY
            {inferenceGeneQty:line.InferenceGeneQty,

inferenceGeneSymbols:line.InferenceGeneSymbols}]-(d)
```

[3] http://ctdbase.org/

apoc.load.csv

View the data

```
CALL apoc.load.csv(
        'file:///.../CTD_chem_go_enriched.csv',
)
YIELD lineNo, map AS line RETURN lineNo, line limit 5
```

[3] http://ctdbase.org/

Loading big files

```
CALL apoc.periodic.iterate(
        "CALL apoc.load.csv(
                'file:///.../CTD_chem_go_enriched.csv',
        )
        YIELD lineNo, map AS line RETURN lineNo, line",
        "MATCH (c:Chemical {chemicalID : line.ChemicalID})
        MERGE (o:Ontology {name : line.Ontology})
        MERGE (t:Term {termID : line.GOTermID})
                SET t.termName = line.GOTermName
                SET t.level = line.HighestGOLevel
        MERGE (c)<-[r:AFFECTED_BY]-(t)-[:TERM_OF]->(o)
                SET r.pValue = line.PValue
                SET r.correctedPValue = line.CorrectedPValue",
        {batchSize:10000, iterateList:true}
)
```

[3] http://ctdbase.org/

Live coding session

Add labels

```
MATCH (p:Pathway)
        WHERE toLower(p.pathwayName)
        CONTAINS 'reaction'
CALL apoc.create.addLabels(id(p), ['Reaction'])
        YIELD node
RETURN count(node)
```

Rename relation

```
MATCH ()-[r:PART_OF]-()
CALL apoc.refactor.setType(r, 'BELONGS_TO')
YIELD input, output
RETURN count(input), count(output)
```

References

- Curated chemical–gene, chemical–disease and gene–disease interactions data were retrieved from the Comparative Toxicogenomics Database (CTD), MDI Biological Laboratory, Salisbury Cove, Maine, and NC State University, Raleigh, North Carolina. URL: http://ctdbase.org/. [October, 2017].
- Cypher Reference Card https://neo4j.com/docs/cypher-refcard/current/
- APOC User Guide https://neo4j-contrib.github.io/neo4j-apoc-procedures/
- Efficient Graph Algorithms in Neo4j https://neo4j.com/blog/efficient-graph-algorithms-neo4j/