# Introduction to Neo4j

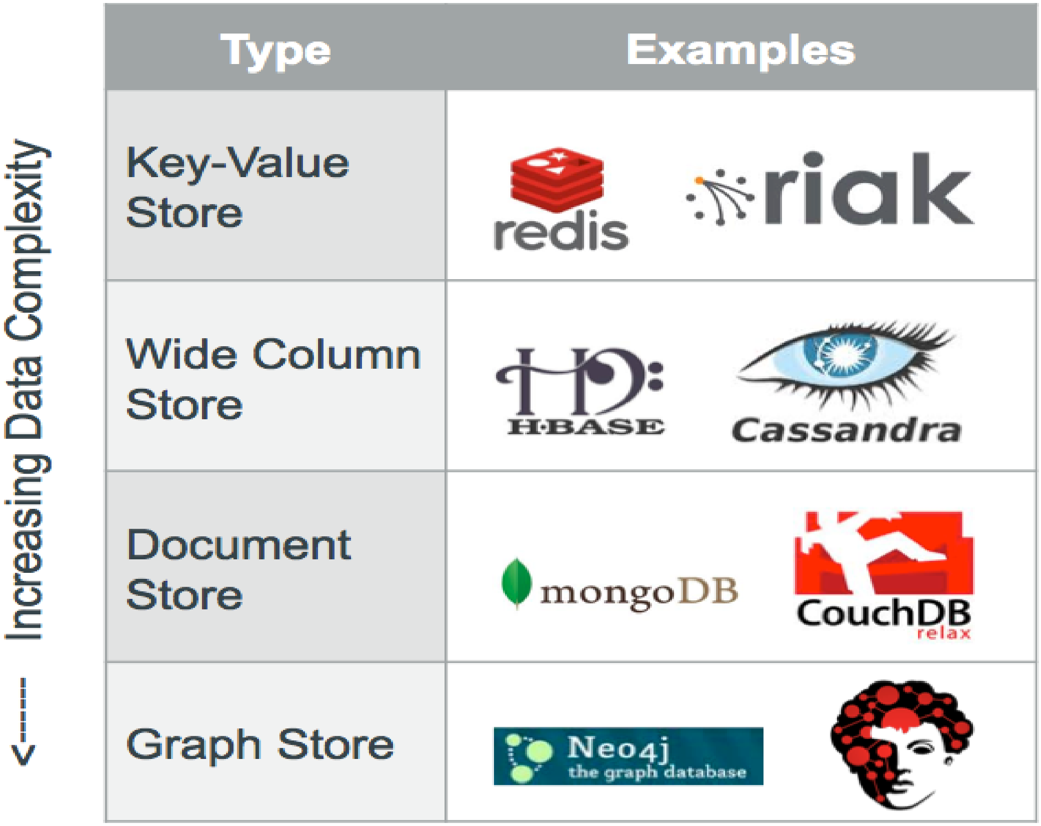
ÉCOLE
POLYTECHNIQUE
UNIVERSITÉ PARIS-SACLAY

# Outline

- Fundamental concepts
- The **Cypher** language (warm-up)
  - creating a graph
  - executing queries
- Examples:
  - Movie Database
  - The Panama Papers

# What is Neo4j

- Neo4j is a graph database management system.



Increasing Data Complexity

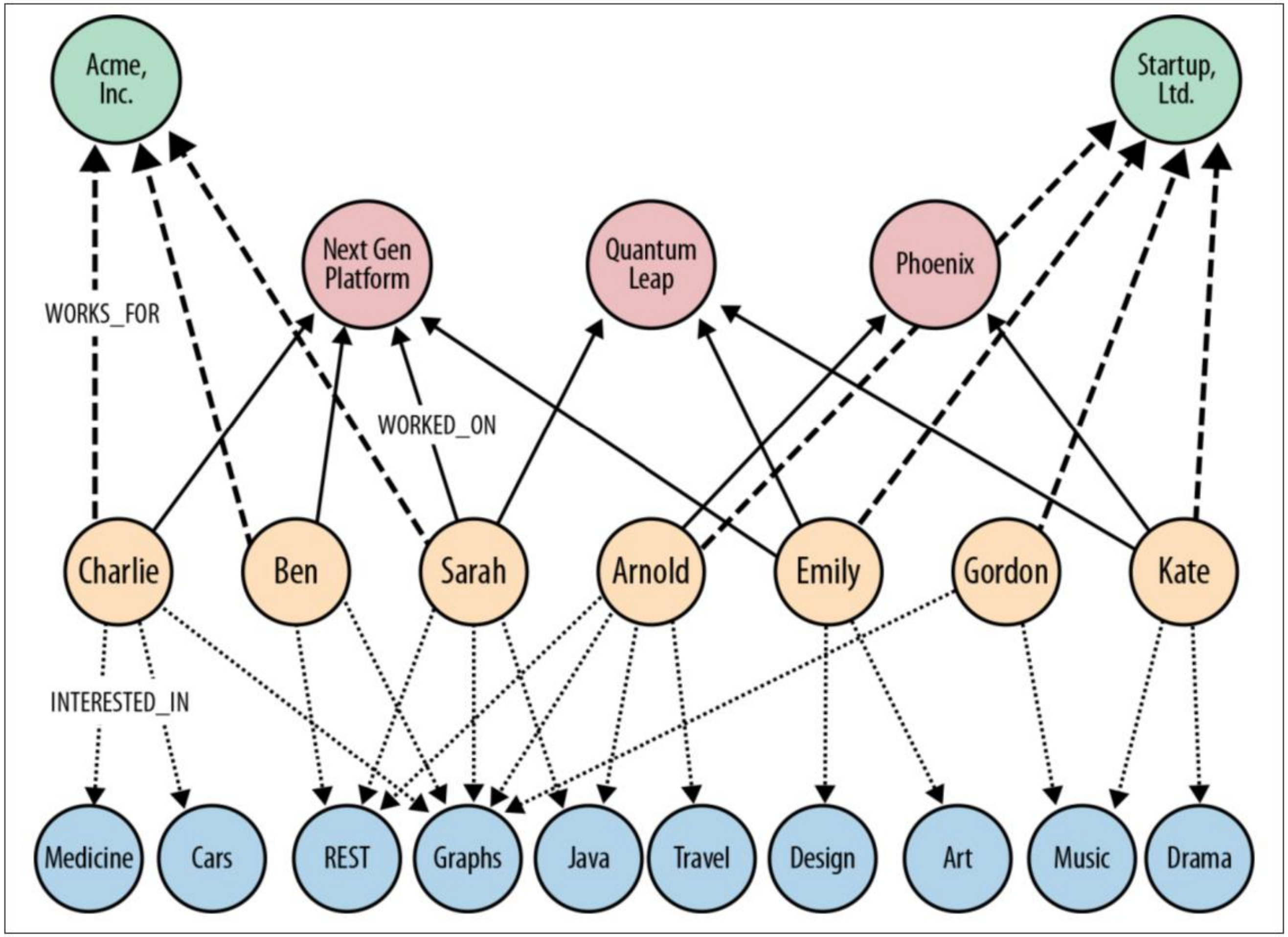
| Type | Examples |
|---|---|
| Key-Value Store | redis<br>riak |
| Wide Column Store | HBASE<br>Cassandra |
| Document Store | mongoDB<br>CouchDB relax |
| Graph Store | Neo4j the graph database |

*Source: http://sqrrl.com/product/nosql/*

# Storage Model

- Linked list of fixed size records
- In the disk every item is a record:
  - Graph Nodes. Pointers to:
    - First item of relationships
    - First item properties
  - Relationships (graph edges). Pointers to:
    - Nodes that are connected with this edge
    - First item properties
    - Next Relationships for the respective nodes
  - Properties:
    - Key value items
    - Points to the next property on the list

# A Heterogeneous Network



*Source: Graph Databases, O'Reilly, 2013*

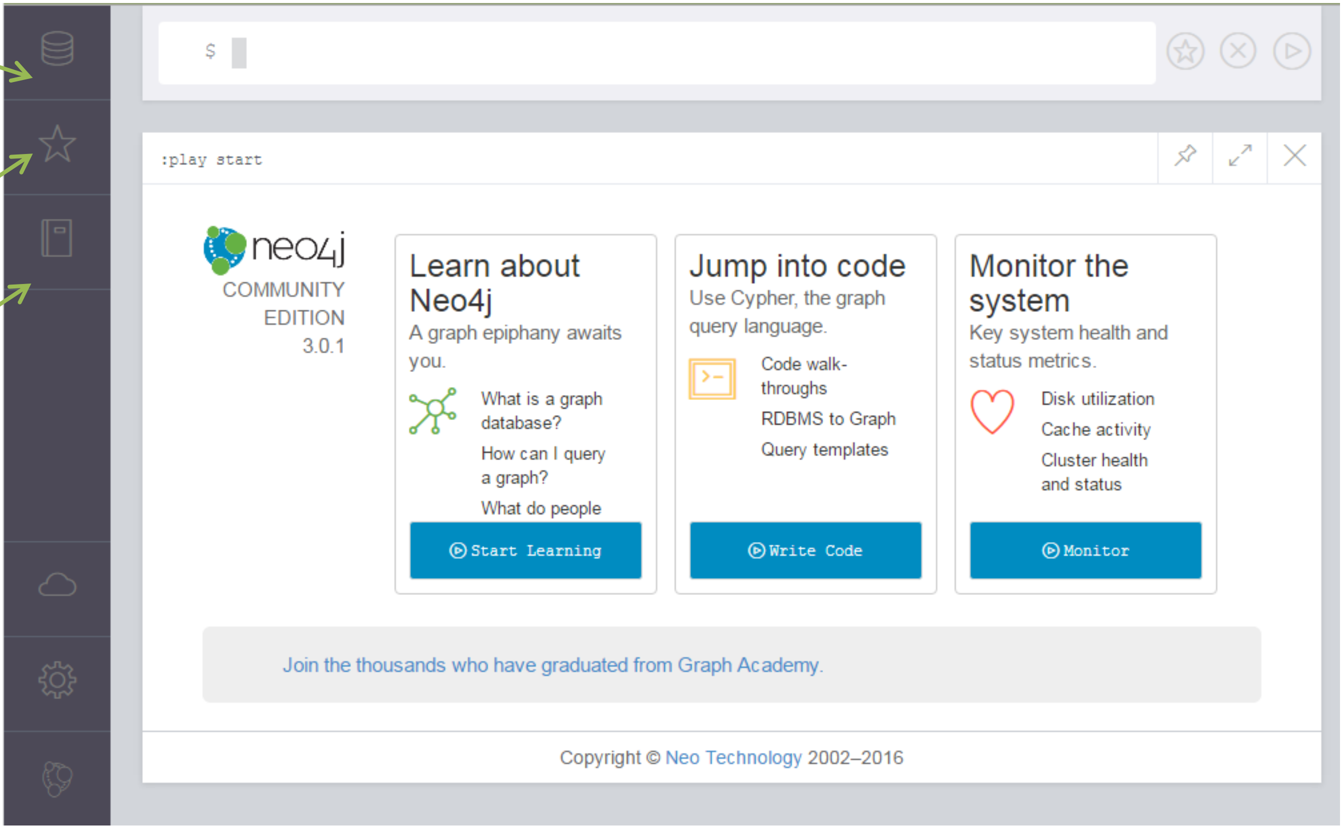
# Start Neo4j

- Use chrome of firefox to go to: [http://localhost:7474/](http://localhost:7474/)
- Username/Password: neo4j,neo4j1

Quick Reminder of all labels, keys and relationship types

Simple examples

Documentation



$

:play start

neo4j
COMMUNITY EDITION
3.0.1

Learn about Neo4j
A graph epiphany awaits you.

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

Start Learning

Jump into code
Use Cypher, the graph query language.

Code walk-throughs
RDBMS to Graph
Query templates

Write Code

Monitor the system
Key system health and status metrics.

Disk utilization
Cache activity
Cluster health and status

Monitor

Join the thousands who have graduated from Graph Academy.

Copyright © Neo Technology 2002–2016

# Creating Nodes

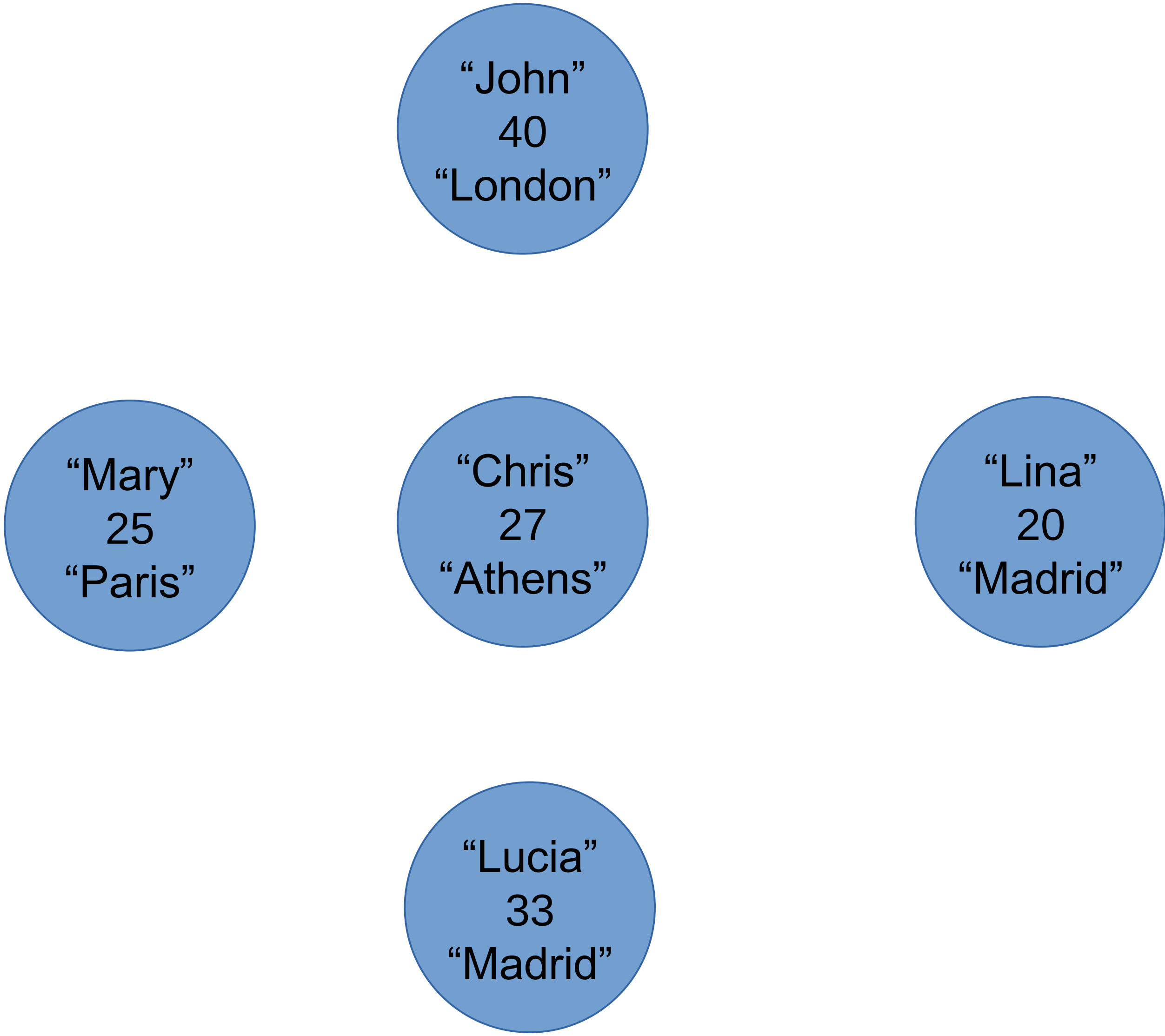
```
CREATE (:Person{name:"Mary",age:25,city:"Paris"});
CREATE (:Person{name:"John",age:40,city:"London"});
CREATE (:Person{name:"Chris",age:27,city:"Athens"});
CREATE (:Person{name:"Lucia",age:33,city:"Madrid"});
CREATE (:Person{name:"Lina",age:20,city:"Madrid"});
```

OR

```
CREATE (:Person{name:"Mary",age:25,city:"Paris"}),
       (:Person{name:"John",age:40,city:"London"}),
       (:Person{name:"Chris",age:27,city:"Athens"}),
       (:Person{name:"Lucia",age:33,city:"Madrid"}),
       (:Person{name:"Lina",age:20,city:"Madrid"});
```

# Nodes

**Person properties:**
*name*
*age*
*city*



"John"
40
"London"

"Mary"
25
"Paris"

"Chris"
27
"Athens"

"Lina"
20
"Madrid"

"Lucia"
33
"Madrid"

# Creating Relationships

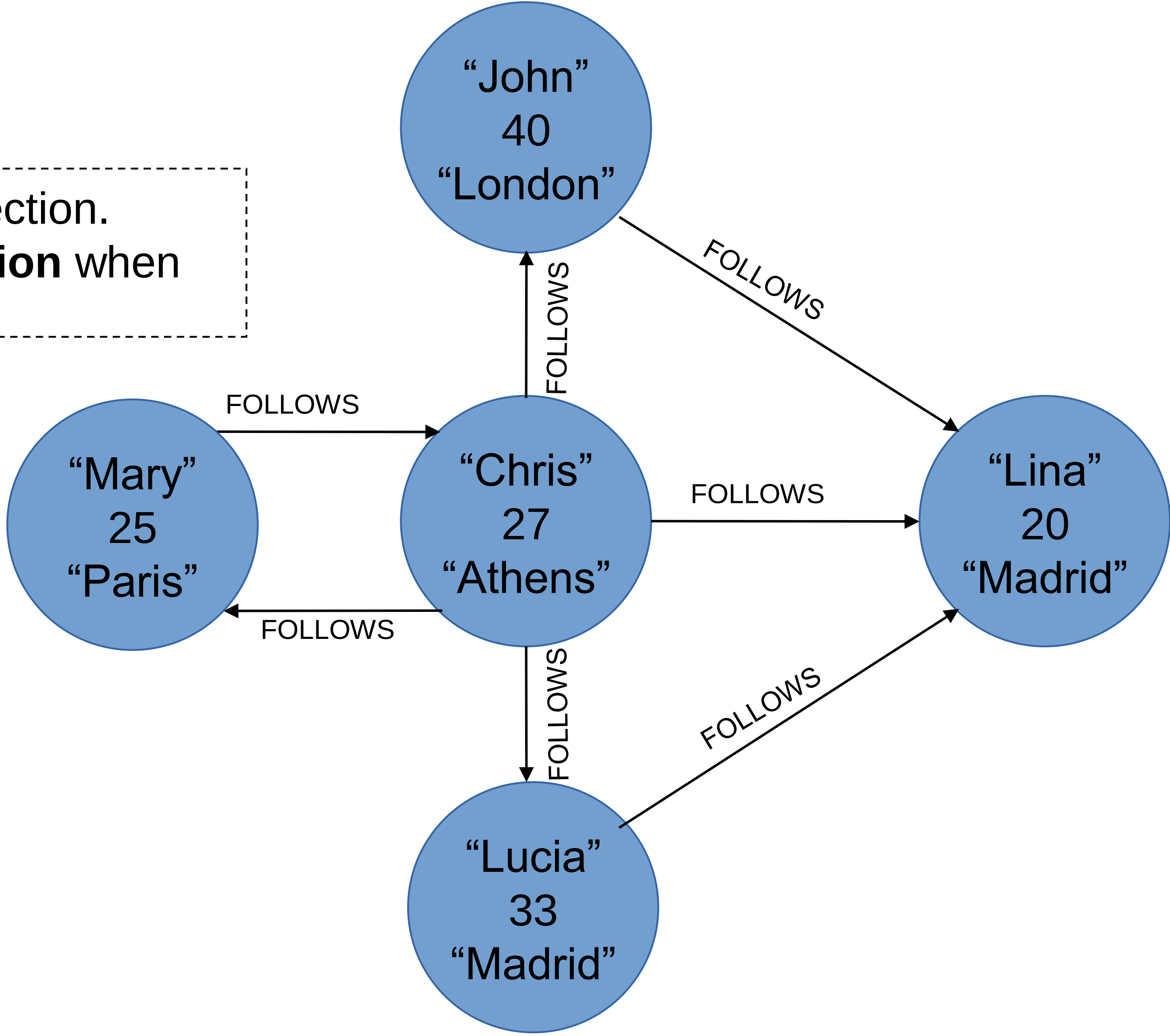
```
MATCH (a:Person),(b:Person)
WHERE a.name = 'Mary' AND b.name = 'Chris'
CREATE (a)-[r:FOLLOWS]->(b)
RETURN r;
```

Copy paste the rest from the file :

*warmup-code.txt*

# Nodes with edges

All edges must have a direction.
We **can ignore the direction** when we query the graph.



# Simple Queries (1)

```
// Display all nodes
MATCH (n)
RETURN n;

// Display a node with specific properties
MATCH (n {name:"Mary"})
RETURN n;

// More than one nodes may be returned
MATCH (n {city:"Madrid"})
RETURN n;

// From the nodes returned, display only specific
properties
MATCH (n {city:"Madrid"})
RETURN n.name;
```

# Simple Queries (2)

```
MATCH (n)
WHERE n.age >= 20 AND n.age <= 35
RETURN n;
```

```
MATCH (n)
RETURN ID(n),n.name;
```

```
MATCH (n)
RETURN n
ORDER BY n.age DESC;
```

# Simple Queries (3)

```
// Is Mary following Chris ?
MATCH p = (a)-[:FOLLOWS]->(b)
WHERE a.name='Mary' AND b.name='Chris'
RETURN p;
```

```
// Return all relationships between two nodes
MATCH p = (a)-[*]-(b)
WHERE a.name='Mary' AND b.name='Chris'
RETURN p
```

```
// Return the label of relation between two nodes
MATCH p = (a)-[r]-(b)
WHERE a.name='Mary' AND b.name='Chris'
RETURN DISTINCT type(r)
```

# Path Queries (1)

```
// Find all nodes in a path between two nodes.
// The path must contain 2 hops only.
MATCH p=(a)-->(b)-->(c)
WHERE a.name='Mary' AND c.name='Lina'
RETURN nodes(p);
```

- Try doing the same but specify the relationship : **FOLLOWS**

# Path Queries (2)

```
// Return all relationships in a path
MATCH p=(a)-->(b)-->(c)
WHERE a.name='Mary' AND c.name='Lina'
RETURN relationships(p)
```

- What if we do not care about link direction?

# Updates

```
// Updating properties
MATCH (n { name: "Lina" })
SET n.city = "Porto"
RETURN n;

// Back to Madrid ...
MATCH (n { name: "Lina" })
SET n.city = "Madrid"
RETURN n;
```

# Deleting Nodes

- Delete Chris and all relationships

```
MATCH (n { name:'Chris' })
DETACH DELETE n;
```

# Deleting Relationships

```
MATCH (a)-[r:FOLLOWS]->(b)
WHERE a.name='Mary' AND b.name='Chris'
DELETE r;
```

- Restore the link

```
MATCH (a:Person),(b:Person)
WHERE a.name = 'Mary' AND b.name = 'Chris'
CREATE (a)-[r:FOLLOWS]->(b)
RETURN r;
```

# Removing Properties

```
MATCH (n { name: "Mary" })
SET n.boss = "yes"
RETURN n;
```

```
MATCH (n { name: "Mary" })
REMOVE n.boss
RETURN n;
```

# Collect & Count

- Find all neighbors of node with ID = 1

```
MATCH (n)--(m)
WHERE ID(n)=1
RETURN n,collect(m);
```

- Find the 10 nodes with the highest degrees

```
MATCH (n)--(c)
RETURN n, count(*) as connections
ORDER BY connections DESC
LIMIT 10;
```

# ForEach

- Mark all nodes along a path

```
MATCH p =(s)-[*]->(d)
WHERE s.name='Mary' AND d.name='Chris'
FOREACH (n IN nodes(p) | SET n.marked = TRUE);
```

The FOREACH clause is used to update data within a collection, whether components of a path, or result of aggregation.

# Nodes of Different Type

```
CREATE (:Car{brand:"Citroen",model:"C5",sn:"112"});
CREATE (:Car{brand:"Peugeot",model:"106",sn:"423"});
CREATE (:Car{brand:"Citroen",model:"C3",sn:"110"});
```

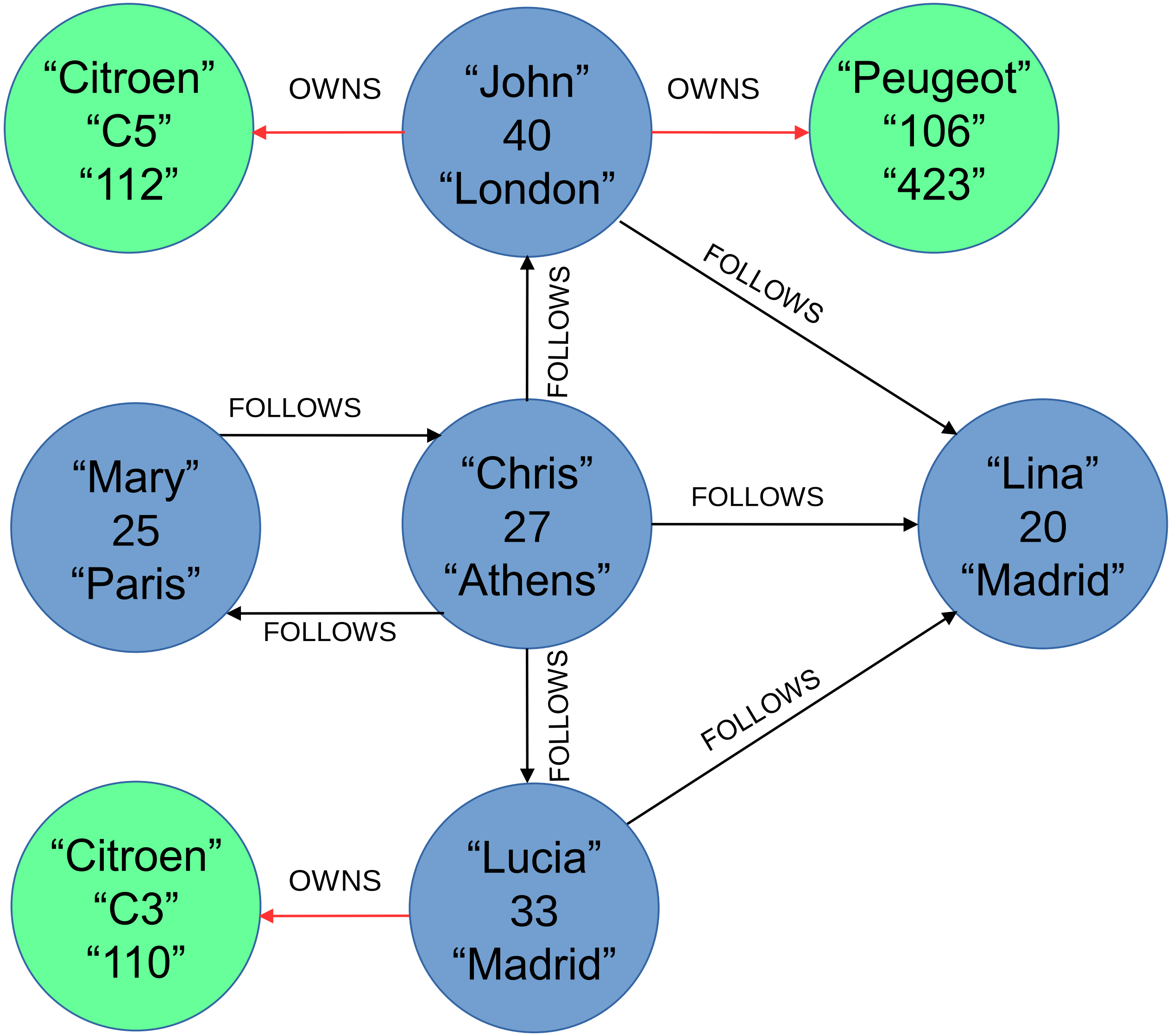
# Relationships of Different Type

```
MATCH (a:Person), (b:Car)
WHERE a.name = "John" AND b.sn = 112
CREATE (a)-[r:OWNS]->(b)
RETURN r;

MATCH (a:Person), (b:Car)
WHERE a.name = "John" AND b.sn = 423
CREATE (a)-[r:OWNS]->(b)
RETURN r;

MATCH (a:Person), (b:Car)
WHERE a.name = "Lucia" AND b.sn = 110
CREATE (a)-[r:OWNS]->(b)
RETURN r;
```

# Heterogeneous Network

**Person properties:**
*name*
*age*
*city*

**Car properties:**
*brand*
*model*
*sn*



# Unwind

- Find all distinct labels in the graph

```
MATCH (n)
WITH DISTINCT labels(n) as labels
UNWIND labels as label
RETURN distinct label
ORDER BY label
```

- *Get the intermediate result of the first two lines*

# Shortest Path

```
MATCH (a:Person { name:"Mary" }),
 (b:Person { name:"Lina" }),
 p = shortestPath((a)-[:FOLLOWS*]-(b))
RETURN p;
```

# More Examples

- Find out the friends of Mary's friends that are not already her friends

```
MATCH (mary { name: 'Mary' })-[:FOLLOWS*2..2]->(friend_of_friend)
WHERE NOT (mary)-[:FOLLOWS]-(friend_of_friend)
RETURN friend_of_friend.name, COUNT(*)
ORDER BY COUNT(*) DESC , friend_of_friend.name;
```

- Find the sn of the cars owned by the friends of the friends of Mary

```
MATCH (a)-[:FOLLOWS]->(b)-[:FOLLOWS]->(c)-[:OWNS]-
>(acar)
WHERE a.name="Mary"
RETURN acar.sn;
```

# Movie Database

- Create a folder **import** in .. Documents\Neo4j\default.graphdb
- Copy the tsv files in import
- Then, use the file load-movies.txt in order to create and populate the movie database
  - Don’t forget the indexes

# IMDB Queries (1)

- For 'Reiner, Rob', fetch all of his associated movies. Order the results by his role in the movie (Actor/Director)
- Find all people acted  in the same movies with 'Reeves, Keane‘
- Find all the movies in which  'Reeves, Keane' and 'Moss, Carolyn' have co-stared. Order them by title and year
- For 'Bale, Christian' find his average movie rating
  - *Helpful functions avg, round*

# IMDB Queries (2)

- Find the titles of the movies that the co-actors of ' Reeves, Keane' have played in, excluding the titles of the movies where ' Reeves, Keane' has acted
- Find the names of the actors that acted with 'Baldwin, Alec' and the number of movies as well.
  - Limit to 15 results
  - Include the titles

# Reduce

- What if we wanted to know the average rating per movie that they co-stared?

```
//ASSUME name:nm, movie collection: MC
//pass it to the reduce function -> WITH

RETURN nm,
REDUCE(srt=0, tmpM  in mc | srt+tmpM.rating)/cnt AS avgR
```

# Real World Example: **Panama Papers**

- The International Consortium of Investigative Journalists (ICIJ) used Neo4j
  - exposed how offshore tax havens are used at scale by elites
- Let’s see part of what they did:
  - family of the **Azerbaijan’s President Ilham Aliyev**
  - http://neo4j.com/graphgist/ec65c2fa-9d83-4894-bc1e-98c475c7b57a

# New Database

- Stop Neo4j
- Create new folder in :
  - Documents\Neo4j e.g. panama.graphdb
- Change the path in the Database Location
- Start Neo4j
  - New password will be needed
- Run the code from *panama_create.txt*

# Interesting Queries (1)

```
//Family Involvements
MATCH (o:Officer)-[r]-(c:Company)
WHERE toLower(o.name) CONTAINS "aliyev"
RETURN o,r,c
```

```
//Who Are the Officers of a Company and Their Roles
MATCH (c:Company)-[r]-(o:Officer)
WHERE c.name = "Exaltation Limited"
RETURN *
```

```
//Shortest Path between Two People
MATCH p=shortestPath((a:Officer)-[*]-(b:Officer))
WHERE a.name="Mehriban Aliyeva" and b.name="Arzu Aliyeva"
RETURN p
```

# Interesting Queries (2)

```
//Joint Company Involvements of Family Members
MATCH (o1:Officer)-[r1]->(c:Company)<-[r2]-(o2:Officer)
WITH o1.name AS first, o2.name AS second,
      count(*) AS cnt,
      collect({ name: c.name, kind1: type(r1),
         kind2:type(r2)}) AS involvements
WHERE cnt > 1 AND first < second
RETURN first, second, involvements, cnt
```

http://neo4j.com/blog/analyzing-panama-papers-neo4j/

# THANK YOU