Property Graph Model
Neo4j

Ι. Κωτίδης

Καθηγητής ΟΠΑ

kotidis@aueb.gr

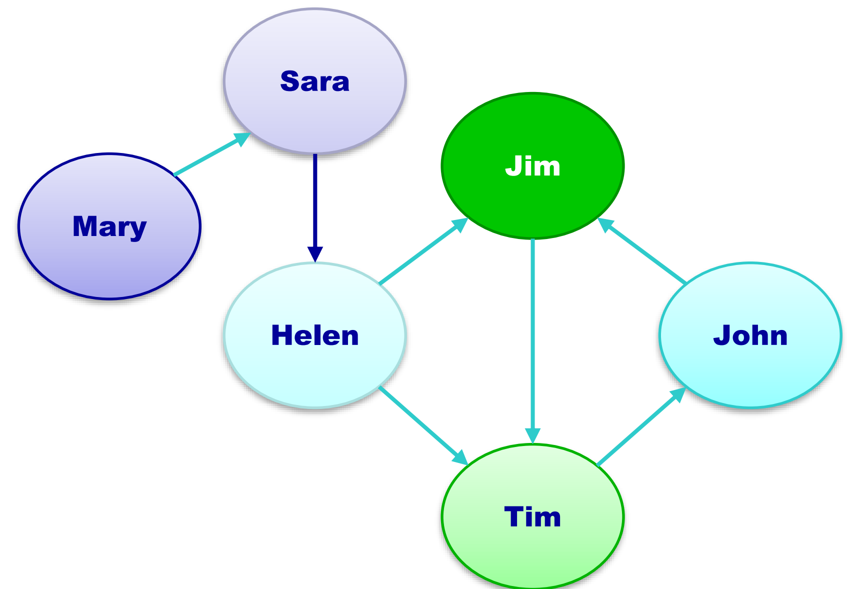
Overview

ΟΠΑ AUEB

- Property Graph Model (neo4j)
- Creating and querying graph databases (cypher)
- The ArtDB dataset as a graph

Βάση Δεδομένων Γράφων

- Οι κόμβοι (nodes) και οι ακμές (relationships/edges) του γράφου αποθηκεύουν δεδομένα

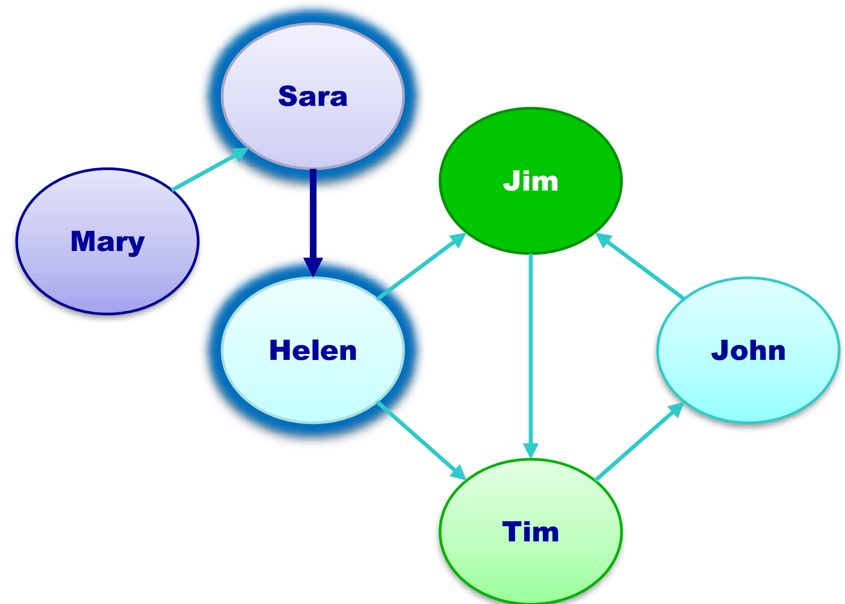


Βάση Δεδομένων Γράφων

- Μέσω των ακμών αποθηκεύουμε συνδέσεις **μέσα** στα ίδια τα δεδομένα:

  (Sara)-[:Likes]->(Helen)

- Σε μία σχεσιακή βάση οι συνδέσεις είναι αναφορικές και πρέπει να ανακτηθούν μέσω συζεύξεων (JOINs)



Παράδειγμα χρήσης γράφων σε μία γλώσσα προγραμματισμού (python)

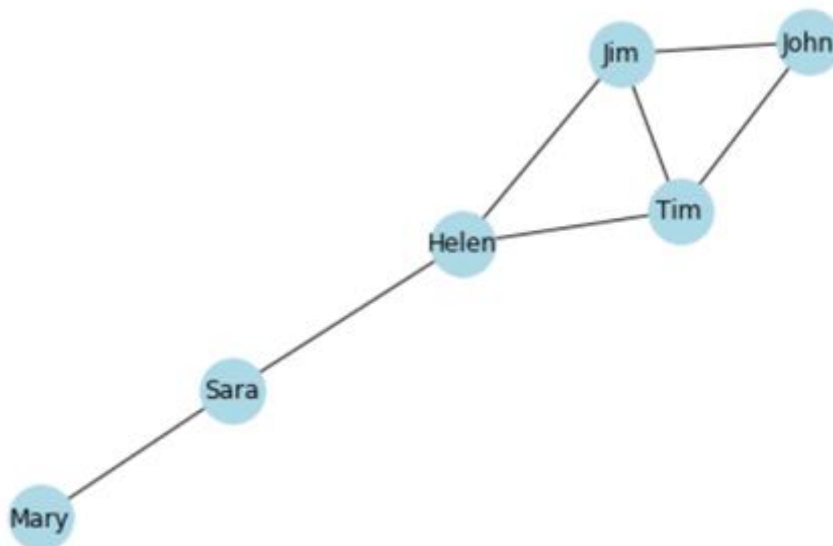
ΟΠΑ AUEB

In [1]:
```
import matplotlib.pyplot as plt
import networkx as nx
G=nx.Graph()
```

In [2]:
```
G.add_nodes_from(['John','Mary','Sara','Helen','Tim','Jim'])

G.add_edges_from([('Mary','Sara'),('Sara','Helen'),
                  ('Helen','Jim'),('Helen','Tim'),
                  ('Jim','Tim'),('Jim','John'),('Tim','John')
])

nx.draw(G,node_color='lightblue',node_size=1000,with_labels=True)
plt.show()
```



Why do I need a Graph Data Base Management System?

ΟΠΑ AUEB

- (Graph) DBMS are optimized for storing datasets much larger than main memory into secondary storage.
- (Graph) DBMS manage issues such as concurrency, integrity, and recovery from hardware failures.
- (Graph) DBMS provide declarative languages for managing and querying large datasets.

Relational Database Usage

Relations

Statements (select columns and rows)

Results

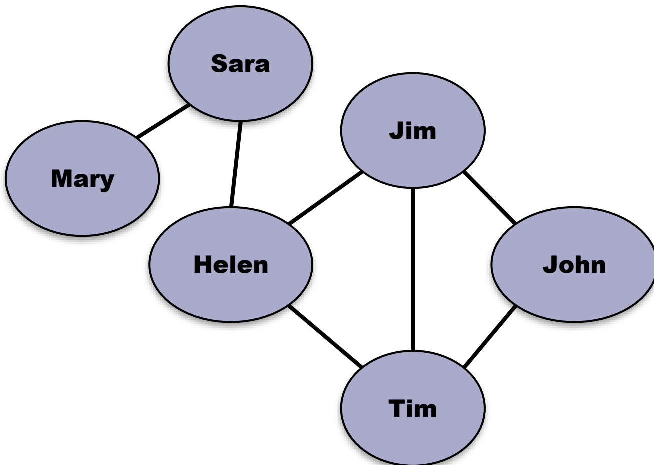
A	B	C	D	E

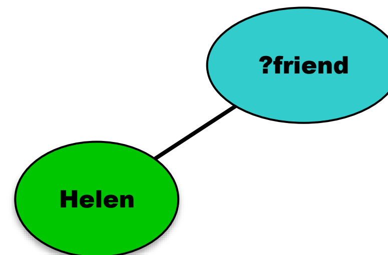
A	D

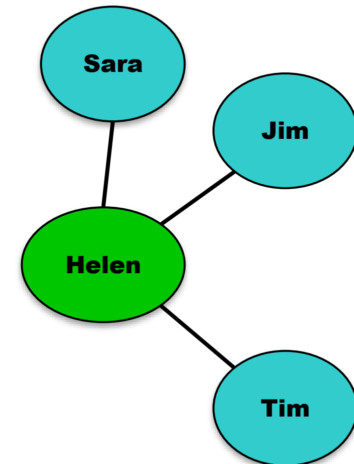
A	D

Graph Database Usage (1)

Graph Statements (patterns) Results





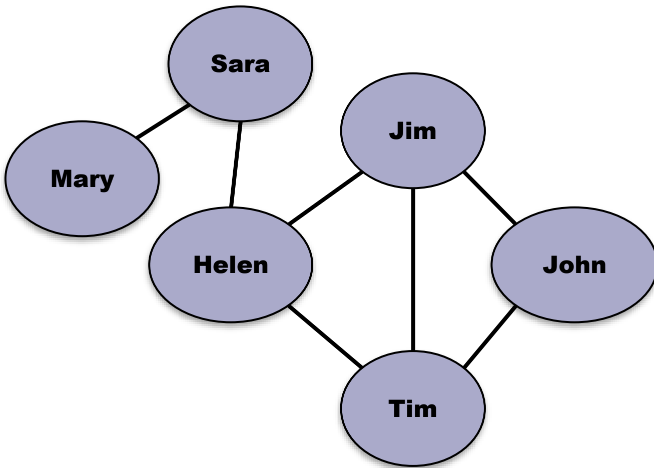


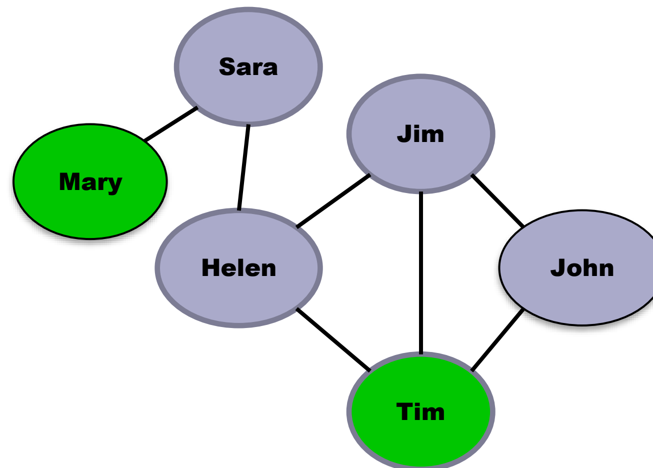
Graph Database Usage (2)

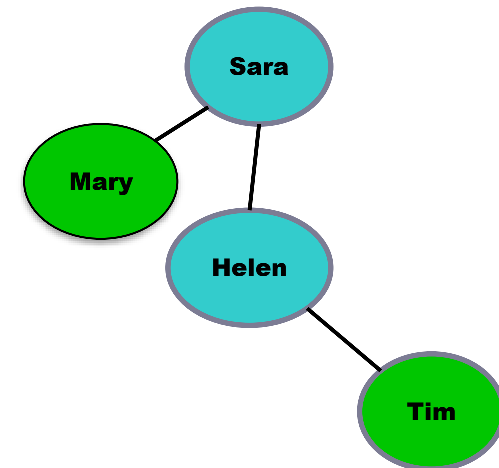
Graph

Statements
(traversals/shortest paths)

Results



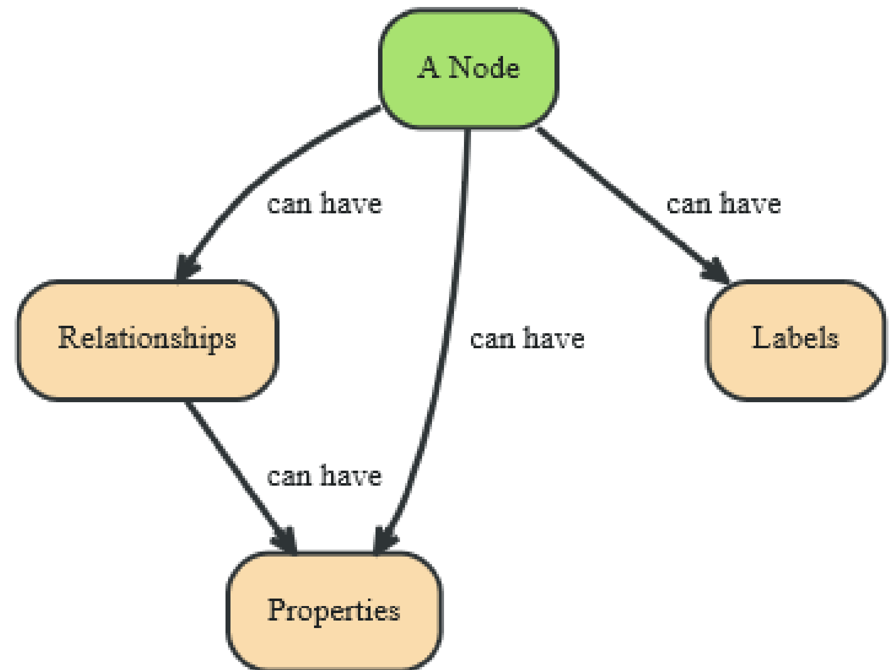




Neo4j: Property Graph Model

ΟΠΑ AUEB

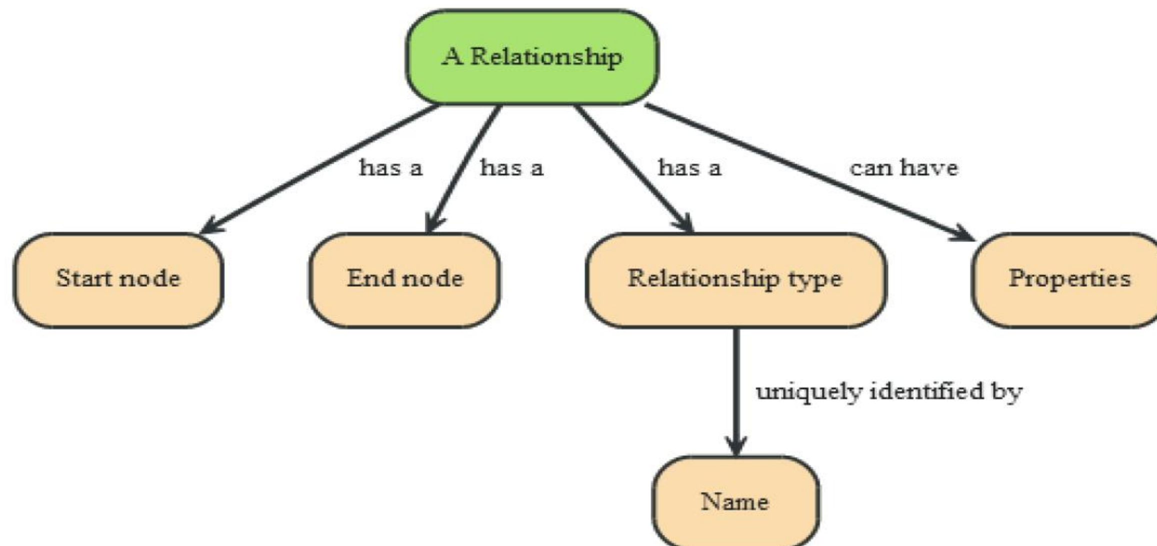
- Graph contains nodes and relationships
- Both can have properties
- Nodes can have labels



Neo4j: Property Graph Model

ΟΠΑ AUEB

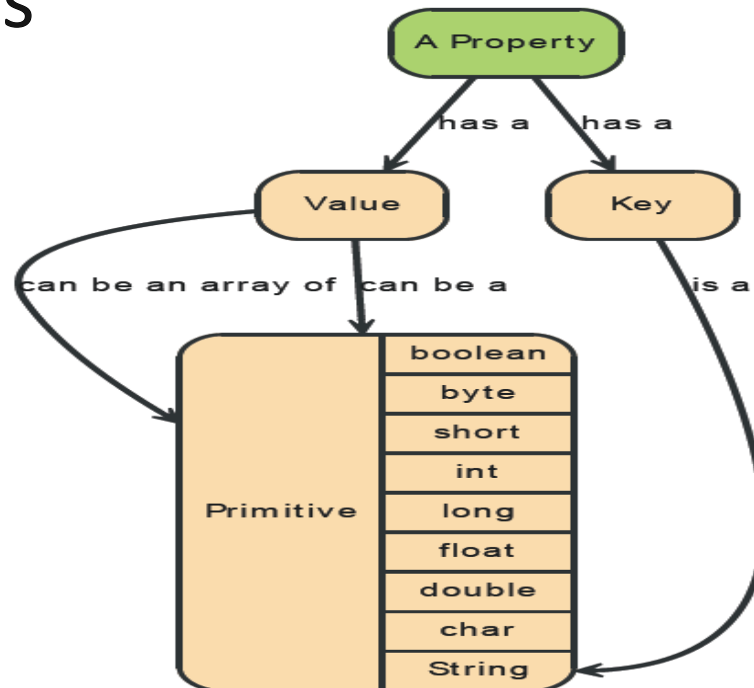
- Relationships: connect two nodes, have direction, have properties, have relationship type



Neo4j: Property Graph Model

ΟΠΑ AUEB

- Properties: key-value pairs, key is a String, values can be primitives or an array of primitives



Node Labels (Ετικέτες)

ΟΠΑ AUEB

- Named graph constructs used to group nodes into ad-hoc sets
  - "John" is a Person
  - "The Adventures of Tom Sawyer" is a Book
- A node may have multiple labels
  - "John" is both a Person and an Artist

How to model the following statement as a graph ?

- “John and Sally know each other. They both have read the book, Graph Databases”

Represent entities as nodes with properties and labels

ΟΠΑ AUEB

- "John and Sally know each other. They both have read the book, Graph Databases"

:Person
name: 'John'
age: 35

:Person
name: 'Sally'
age: 26

:Book
title: 'Graph Databases'
isbn: '978-1449356262'

Cypher Syntax: CREATE nodes

ΟΠΑ AUEB

- CREATE (john:Person {name: 'John', age: 35})

:Person
name: 'John'
age: 35

- General syntax CREATE (n:$Label_1$:…:$Label_n$ { $attr_1$:val_1, $attr_2$:val_2, …$attr_k$:val_k})
  - n is a variable that you can use to refer to that node in the same script

Freedom of choice

ΟΠΑ AUEB

- Unlike relational databases, there is no restriction on the number and type of properties on a node
  - E.g. nodes may have different properties, or same properties of different types
  - Recall Person is just a label. It does not restrict the schema of the corresponding nodes

:Person
name: 'John'
age: 35
weight: 85

:Person:Gamer
fname: 'Jim'
byear: 1997
weight: '87kg'

Existential constraints

- Assert that each Person has a name

:Person
name: 'John'
age: 35

- CREATE CONSTRAINT ON (person:Person) ASSERT exists(person.name)

Unique constraints

ΟΠΑ AUEB

- Assert that no two books in the database can have the same isbn

:Book
title: 'Graph Databases'
isbn: '978-1449356262'

- CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE

Key constraint

ΟΠΑ AUEB

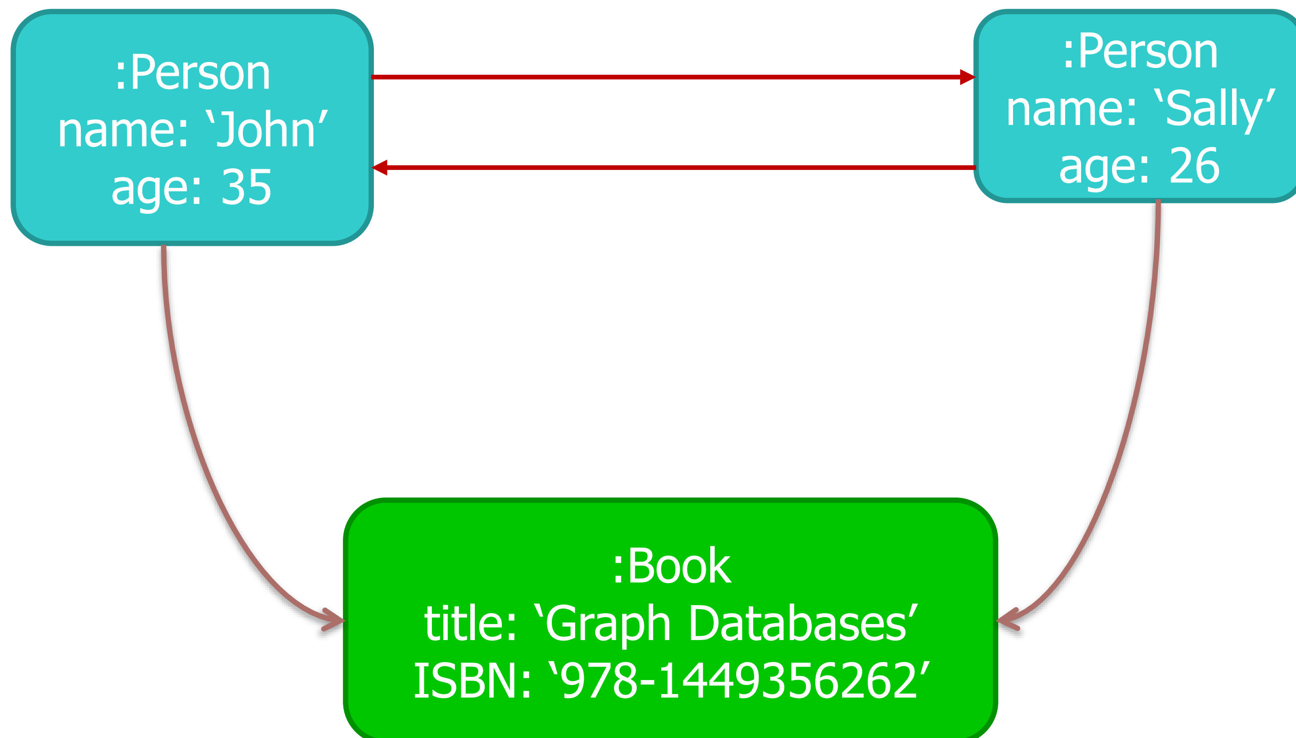
- Each book should have a unique isbn

:Book
title: ‘Graph Databases’
isbn: ‘978-1449356262’

- CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS NODE KEY

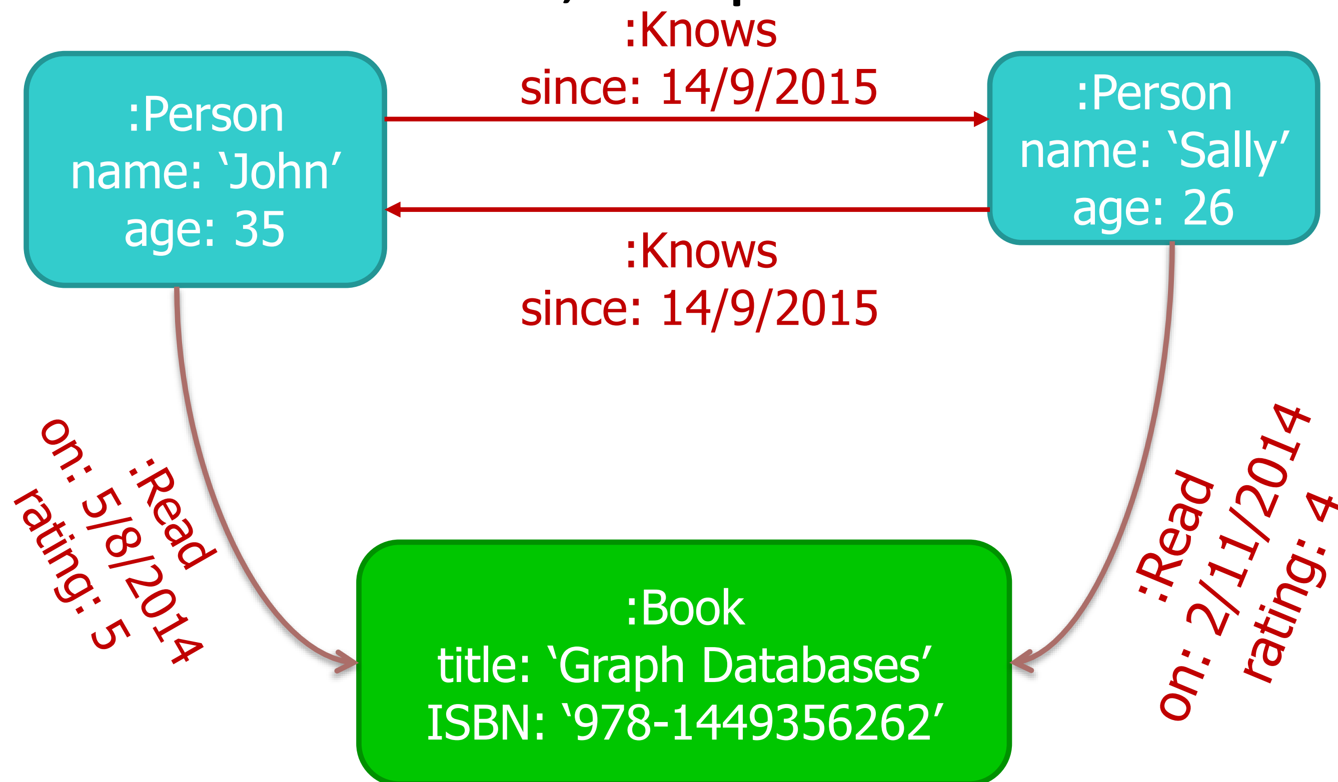
Express relationships as edges between nodes

- "John and Sally know each other. They both have read the book, Graph Databases"



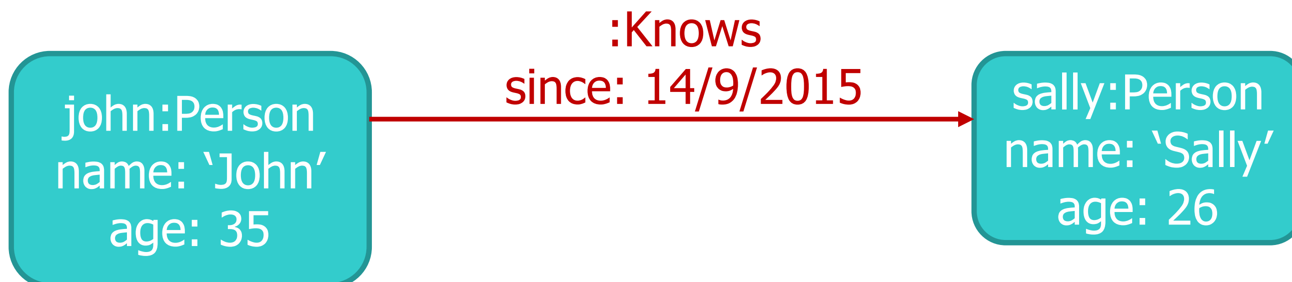
Relationships have types and (optionally) properties

- “John and Sally know each other. They both have read the book, Graph Databases”



Create relationship

- CREATE (john)-[:Knows {since: '14/9/2015'}]->(sally)

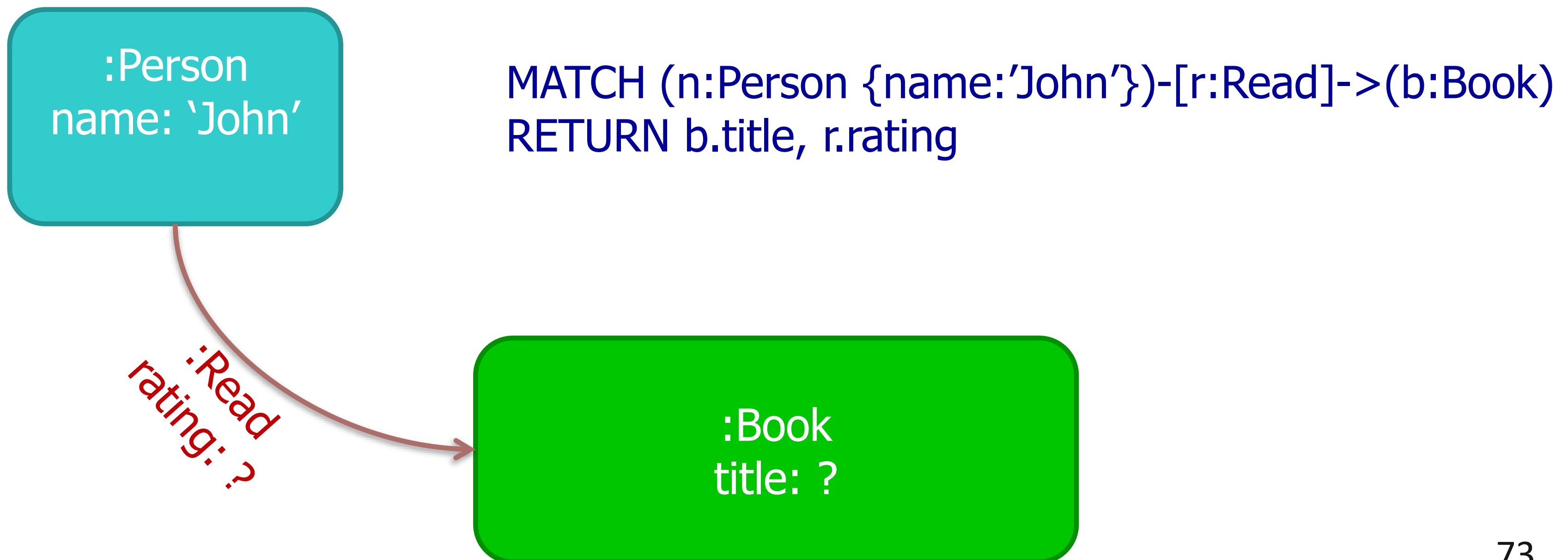


- In this example Knows is a relationship type, since is a property for that particular instance, john & sally are variables that refer to previously created nodes

Querying the graph database

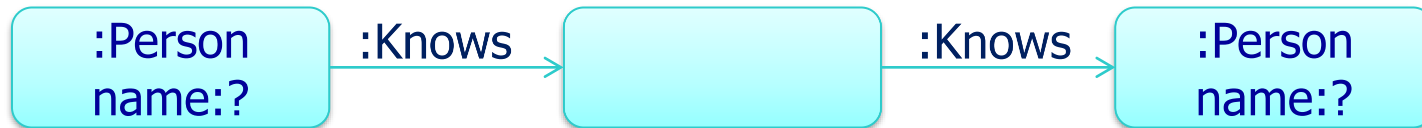
- Queries (patterns) are also graphs!

"Find the titles of all books that a person named John read and report his ratings"



More Graph Patterns

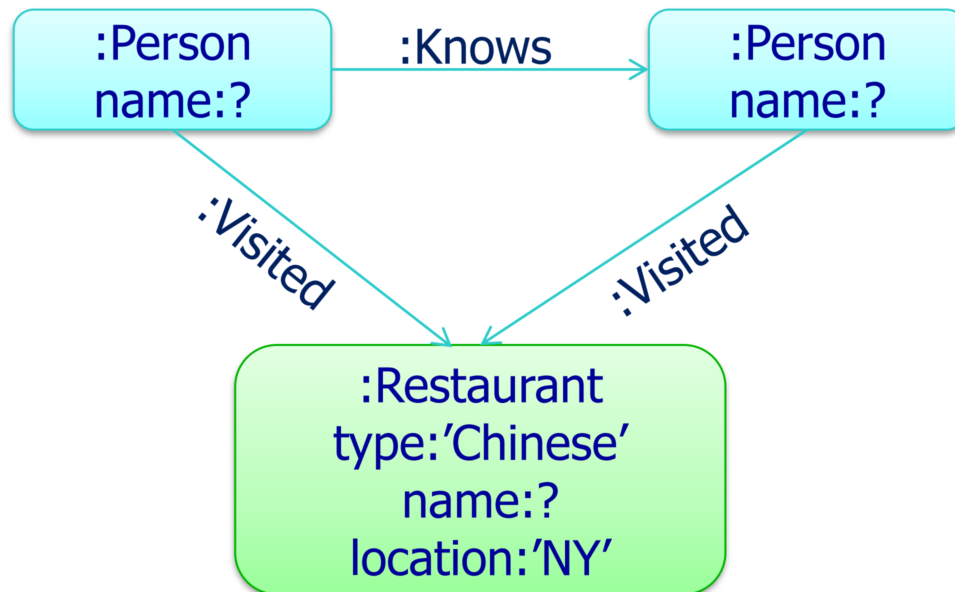
- Friend-of-friend pairs in a social network



- MATCH (x:Person)-[:Knows]->()-[:Knows]->(y:Person)
  RETURN x.name, y.name

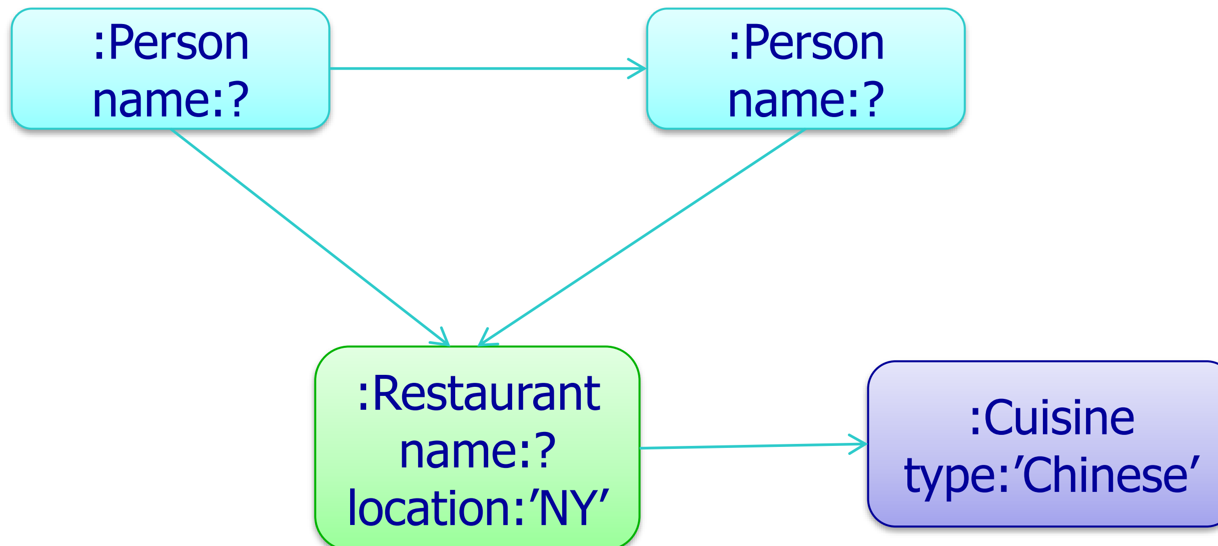
Can you write the following query?

- Find friends that visited same Chinese restaurant in NY. Return their name and the name of the restaurant



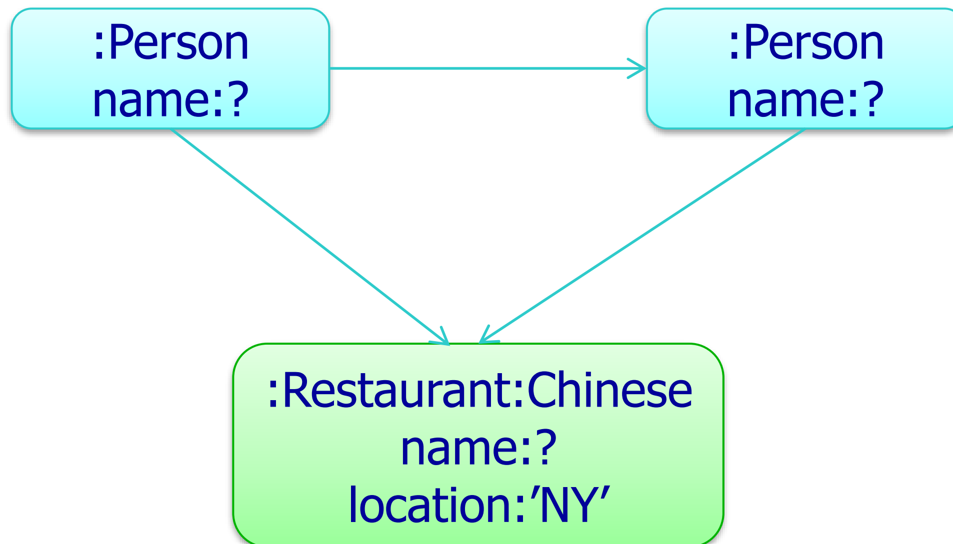
Alternative model

- Find friends that visited same Chinese restaurant in NY. Return their name and the name of the restaurant



Alternative model

- Find friends that visited same Chinese restaurant in NY. Return their name and the name of the restaurant



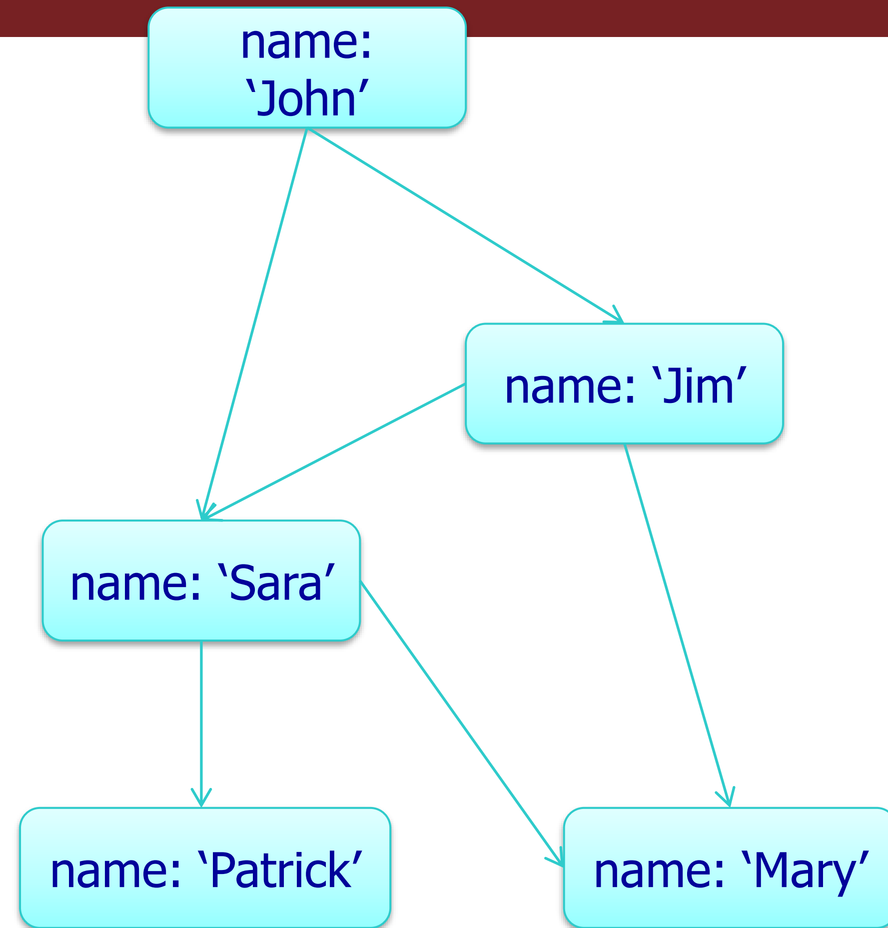
More on Cypher

ΟΠΑ AUEB

- Constructs for ordering, aggregation, joins
  - E.g. friends who have all visited same restaurant
    - visit by at least 3 friends (COUNT)
    - each gave a rating ≥ 4 (filter)
    - order by most recent visit (ORDER)
- These are mainly filters or involve post-processing of matches found (sorting, aggregation)

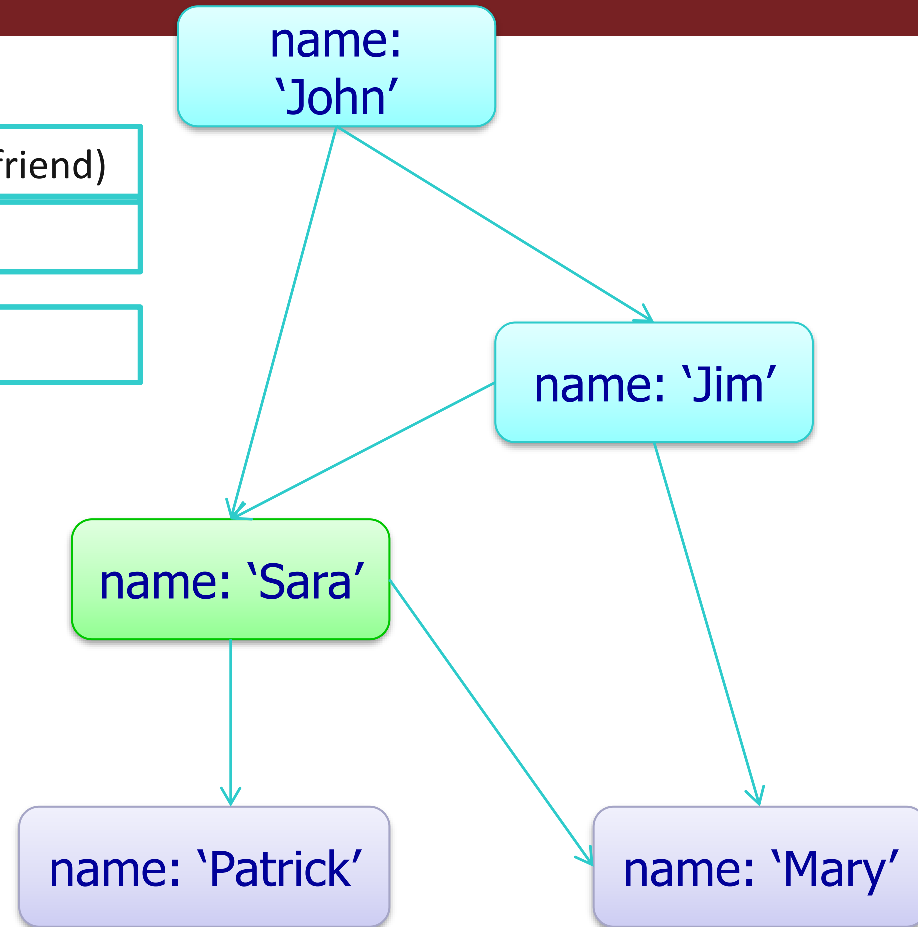
Example: Friend suggestions

- Find out the friends of John's friends that are not already his friends
  - Order them by the number of connections to them, and secondly by their name



Pattern Matching using Cypher

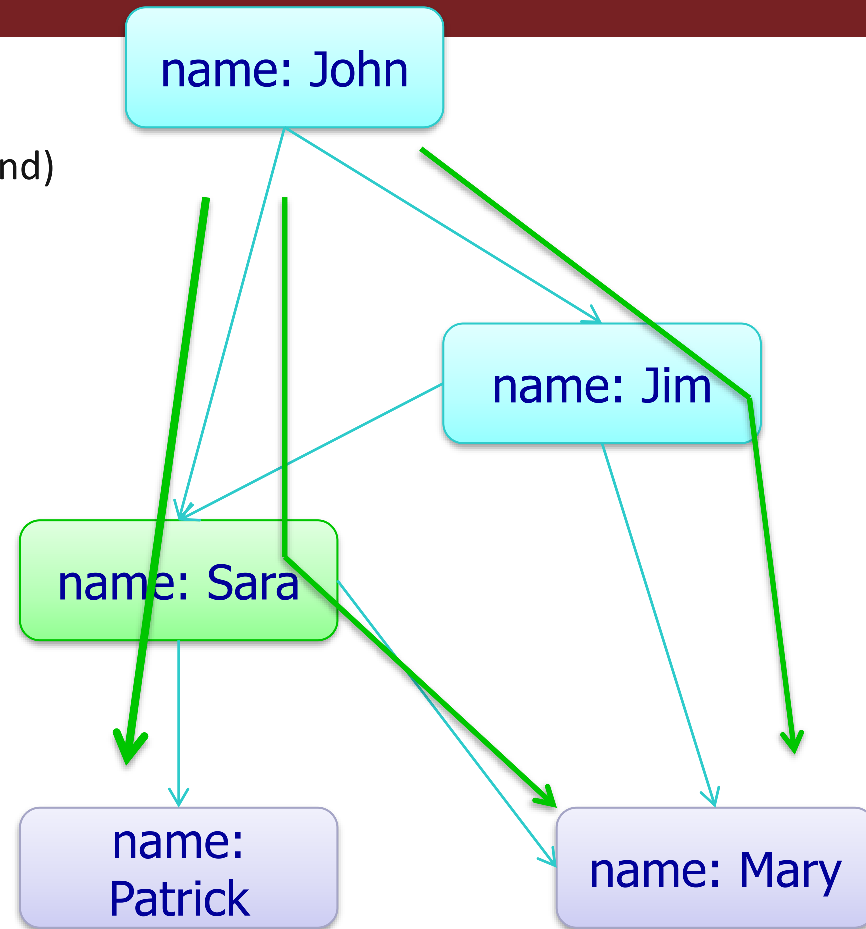
```
MATCH (john {name:'John'})-[:Knows*2..2]-(friend_of_friend)
WHERE NOT (john)-[:Knows]-(friend-of-friend)
RETURN friend_of_friend.name, COUNT(*)
ORDER BY COUNT(*) DESC, friend_of_friend.name
```



Pattern Matching using Cypher

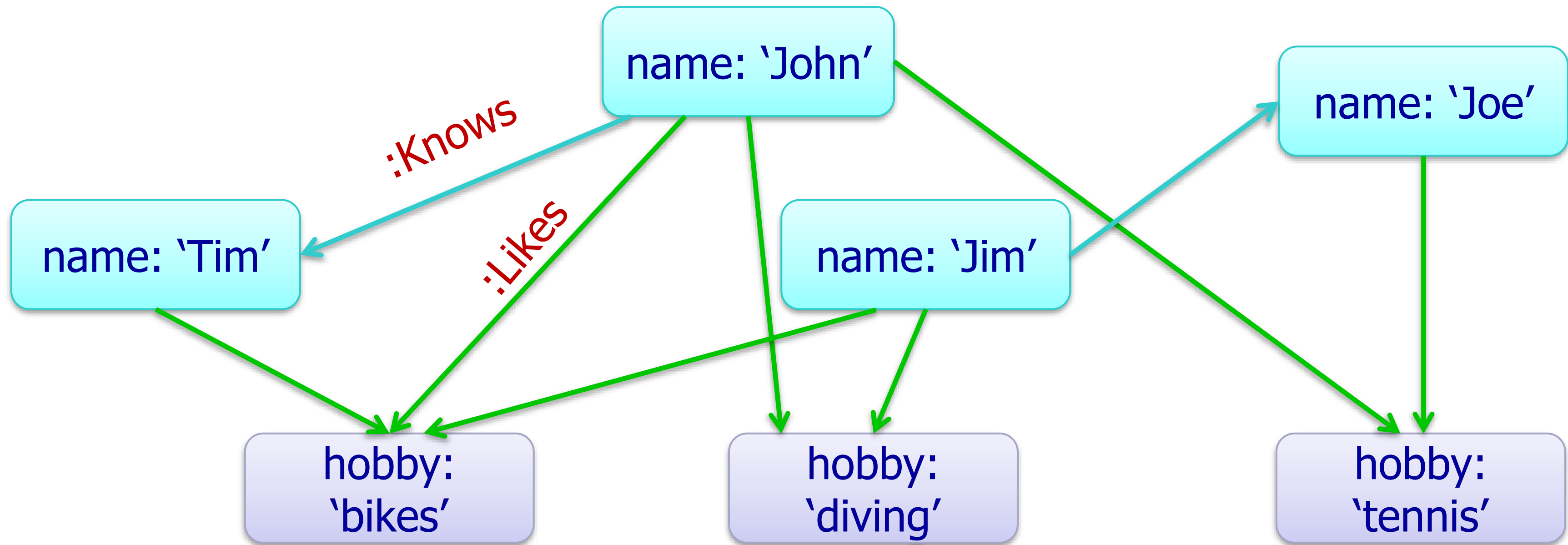
```
MATCH (john {name:'John'})-[:Knows*2..2]-(friend_of_friend)
WHERE NOT (john)-[:Knows]-(friend-of-friend)
RETURN friend_of_friend.name, COUNT(*)
ORDER BY COUNT(*) DESC, friend_of_friend.name
```

COUNT(*)	friend_of_friend.name
2	Mary
1	Patrick



Note: we are ignoring direction of edges in this exaple

Suggest user with most similar likes




```
MATCH (me {name:'John'})-[:Likes]->(something)<-[:Likes]-(someone)
WHERE NOT (me)-[:Knows]-(someone)
RETURN someone.name, COUNT(something)
ORDER BY COUNT(something) DESC LIMIT 1
```

someone.name	COUNT()
Jim	2

ΟΠΑ AUEB

Property Graph– Παράδειγμα (ArtDB)

Θέλουμε να σχεδιάσουμε μία βάση δεδομένων για ένα ινστιτούτο το αντικείμενο του οποίου είναι η καταγραφή της πορείας της σύγχρονης τέχνης στην Ελλάδα. Στη βάση δεδομένων θα πρέπει να αποθηκεύονται πληροφορίες σχετικά με την συλλογή του ινστιτούτου η οποία περιλαμβάνει στοιχεία για **έργα τέχνης** (περίπου 26000), για **καλλιτέχνες** (περίπου 5000) και για **εκθέσεις** (περίπου 31000) που έχουν λάβει χώρα από το 1945 μέχρι σήμερα. Για κάθε **έργο** το οποίο ταυτοποιείται με ένα μοναδικό κωδικό, θέλουμε να αποθηκεύσουμε τον τίτλο του, τον **δημιουργό** και το έτος δημιουργίας του, τις διαστάσεις του (ύψος, πλάτος, βάθος), την τιμή πώλησής του και την κατηγορία του (πίνακας, γλυπτό, χαρακτικό, video art, κόσμημα κ.λπ). Επιπλέον για κάθε έργο πρέπει να γνωρίζουμε τα υλικά με τα οποία έχει φτιαχτεί (λαδομπογιά, μελάνι, χαλκός, μάρμαρο κ.λπ.) καθώς επίσης και τις θεματικές ενότητες στις οποίες εντάσσεται.

Για κάθε **έκθεση** το ινστιτούτο καταγράφει τον τίτλο της, την τοποθεσία και το χρονικό διάστημα κατά το οποίο έλαβε χώρα, τον τύπο της (ατομική ή ομαδική) και τον **φορέα** ή τους φορείς που οργάνωσαν την έκθεση. Επιπλέον θέλουμε να γνωρίζουμε τους δημιουργούς και τα έργα των οποίων εκτέθηκαν σε κάθε έκθεση.

Στη βάση δεδομένων θα πρέπει να καταγράφονται για κάθε **καλλιτέχνη** το ονοματεπώνυμό του, η κύρια ιδιότητά του (ζωγράφος, αγιογράφος, χαράκτης, γλύπτης κ.λπ.), το φύλλο του, καθώς επίσης το έτος και ο τόπος γέννησης.

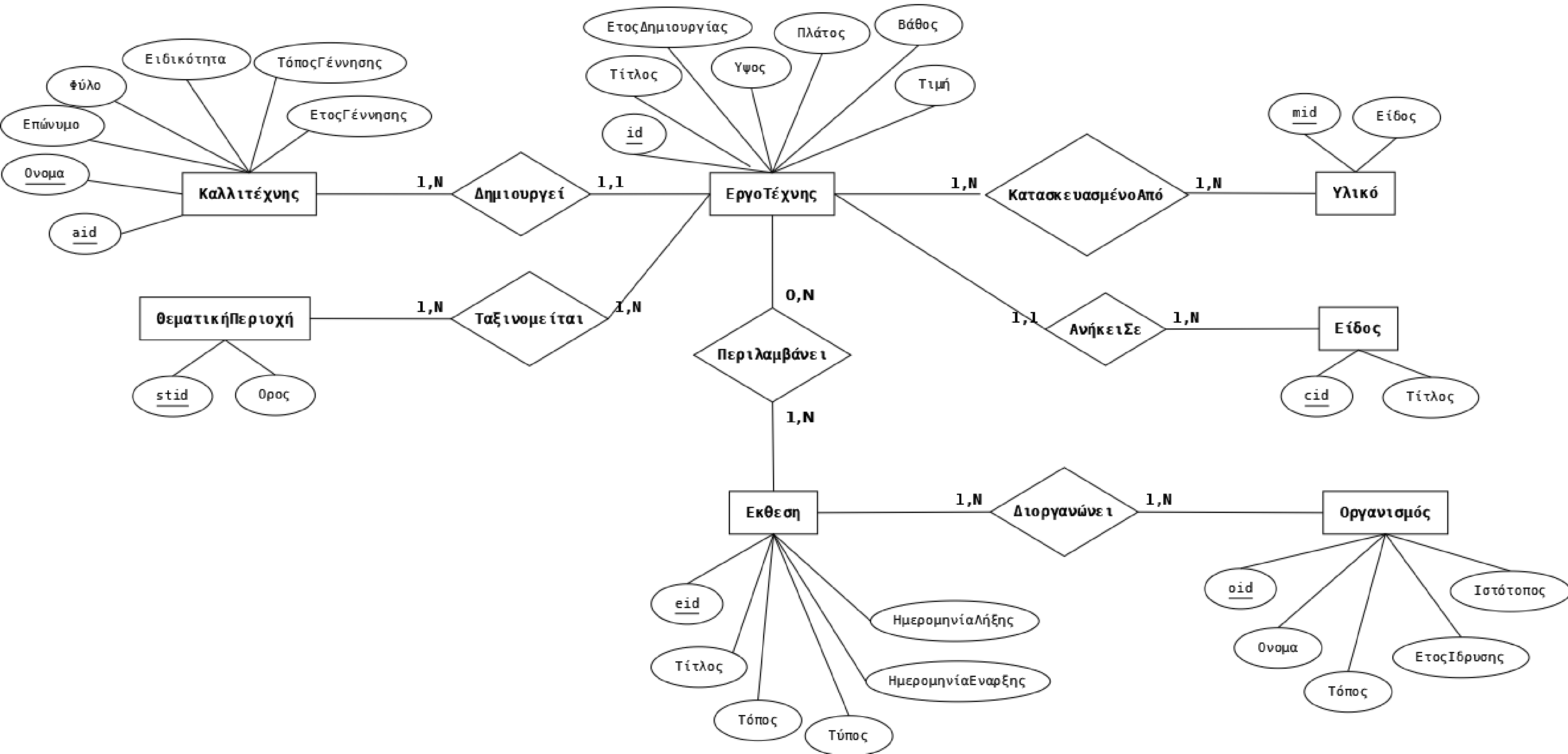
Τέλος στη βάση θα καταγράφονται και τα στοιχεία των **φορέων/οργανισμών** διοργανωτών των εκθέσεων. Για κάθε φορέα θα καταγράφονται η ονομασία του, η τοποθεσία το έτος ίδρυσης και η διεύθυνση της ιστοσελίδας του.

Λεξικό εννοιών - Ορολογία

ΟΠΑ
AUEB

Όρος	Περιγραφή	Συνώνυμο	Συνδέσεις
ΈργοΤέχνης	Έργο τέχνης.	Έργο	Καλλιτέχνης Έκθεση
Καλλιτέχνης	Δημιουργός έργων τέχνης	Δημιουργός	Έργο Έκθεση
Έκθεση	Περιέχει έργα από διάφορους καλλιτέχνες.		Έργο Καλλιτέχνης Φορέας
Φορέας	Οργανώνει εκθέσεις	Οργανισμός	Έκθεση

Διάγραμμα ER



Πλάνο

ΟΠΑ AUEB

- Θα σχεδιάσουμε έναν γράφο ο οποίος θα ενσωματώνει όλα τα στιγμιότυπα των οντοτήτων και συσχετίσεων θέλουμε να καταγράψουμε
- Ο γράφος θα ακολουθεί το μοντέλο του«γράφου με ιδιότητες» (property graph model)

Ας αρχίσουμε με την οντότητα «Καλλιτέχνης»

- Ένας Καλλιτέχνης (Artist) έχει τα παρακάτω γνωρίσματα:
  - aid: μοναδικό αναγνωριστικό («πρωτεύων κλειδί»)
  - fname: επώνυμο
  - lname: όνομα
  - sex: male/female
  - bplace, byear: τόπος/χρονολογία γεννήσεως
  - specialty: ειδικότητα

Ετικέτες Κόμβων

ΟΠΑ AUEB

- Κάθε στιγμιότυπο μίας οντότητας μπορεί να περιγραφεί ως ένας κόμβος στη βάση του neo4j

- Προτείνεται η χρήση ετικετών ώστε να διακρίνουμε στιγμιότυπα διαφορετικών οντοτήτων (πχ καλλιτέχνες από έργα τέχνης) στους κόμβους του γράφου

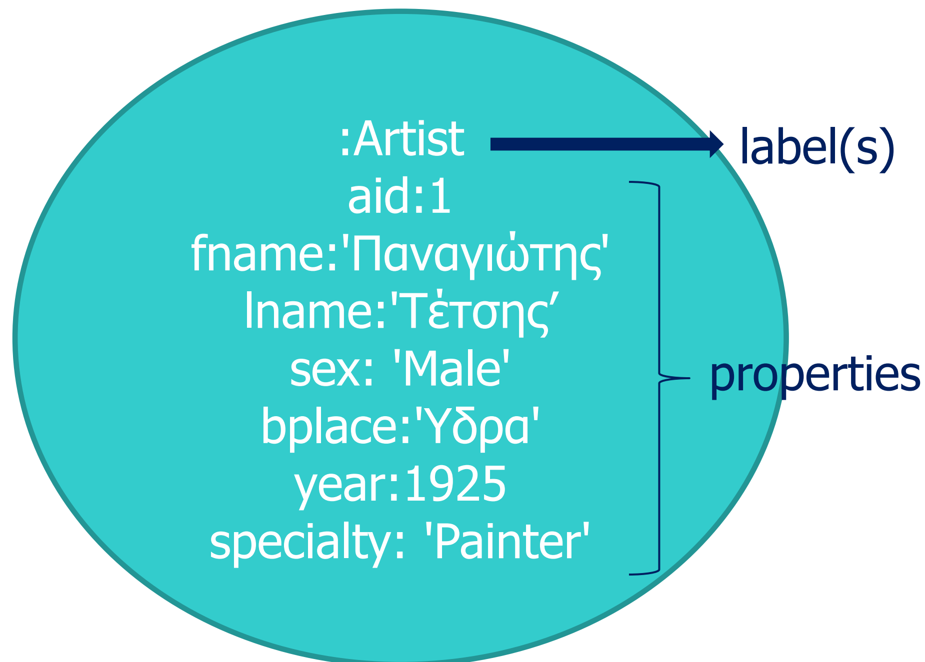
Ετικέτες

ΟΠΑ
AUEB

- Χρησιμοποιούνται για το διαχωρισμό των κόμβων της βάσης σε σύνολα

- Ένας κόμβος μπορεί να έχει πολλαπλές ετικέτες (πχ «καλλιτέχνης», «άνδρας»)

Ο καλλιτέχνης ως κόμβος ενός γράφου



Καλλιτέχνες ως «κόμβοι/nodes» - cypher

- create (:Artist {aid:1, fname:'Παναγιώτης', lname:'Τέτσης', sex: 'Male',bplace:'Ύδρα', year:1925,specialty: 'Painter'})

Περιορισμός κλειδιού

ΟΠΑ AUEB

- Κάθε καλλιτέχνης έχει (υποχρεωτικά) ένα μοναδικό αναγνωριστικό aid (artist-id)

- In cypher:

```
CREATE CONSTRAINT ON (n:Artist) ASSERT (n.aid) IS
NODE KEY
```

Περιορισμοί τιμών?

ΟΠΑ AUEB

- Όπως αναφέραμε, δεν υπάρχει περιορισμός στον τύπο και αριθμό γνωρισμάτων που μπορεί να έχει ένας κόμβος
- Η ετικέτα «Artist» δεν περιορίζει τον τύπο του κόμβου. Οι παρακάτω δύο ορισμοί είναι επιτρεπτοί
  - create (:Artist { aid:1, fname:'Παναγιώτης', lname:'Τέτσης', sex: 'Male',bplace:'Υδρα', year:1925,specialty: ‘Painter'})
  - create (:Artist { aid:2, **fistName**:'Γιάννης', **lastName**:'Αδαμάκος', sex: **'A'**, bplace: 'Πύργος Ηλείας', year: '**1952**',**url: 'http://dp.iset.gr/artist/view.html?id=1462**'})
- Επομένως θα πρέπει να είμαστε προσεκτικοί κατά τη δημιουργία της βάσης!

Properties vs Labels

ΟΠΑ AUEB

- Κατηγορικά γνωρίσματα με μικρό πεδίο τιμών (male/female)
- Γνώρισμα ως property:
  - create (:Artist { aid:1, fname:'Παναγιώτης', lname:'Τέτσης', sex: 'Male',bplace:'Υδρα', year:1925,specialty: ‘Painter’})
- Γνώρισμα ως label:
  - create (:Artist:Male { aid:1, fname:'Παναγιώτης', lname:'Τέτσης',bplace:'Υδρα', year:1925, specialty: 'Painter’})

Properties vs Labels

- Παρατήρηση: ερωτήματα που αναφέρονται σε ετικέτες είναι ποιο γρήγορα στην επεξεργασία τους από ερωτήματα σε γνωρίσματα χωρίς ευρετήριο
  - match (who:Artist:Male) return who.fname

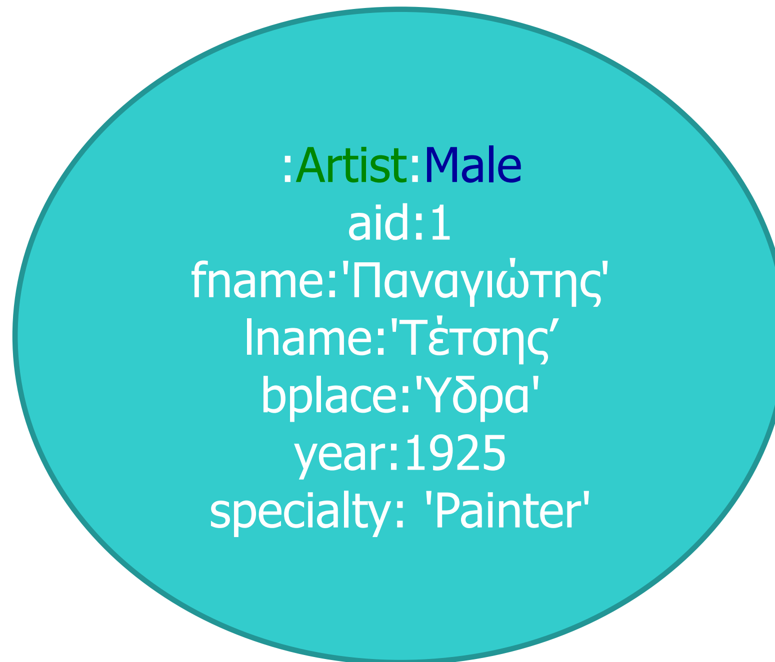
    vs

  - match (who:Artist {sex: 'Male'}) return who.fname

- Φυσικά θα πρέπει να αξιολογήσουμε αν μας ενδιαφέρουν (ή αν είναι συχνά) τέτοια ερωτήματα

Ποιο άλλο γνώρισμα θα μπορούσε να αναβαθμιστεί ως ετικέτα?

ΟΠΑ AUEB



- Οι τιμές του specialty έρχονται από ένα περιορισμένο/ελεγχόμενο λεξιλόγιο

Καλλιτέχνες ως «κόμβοι/nodes» - λήψη 2

:Artist:Male:Painter
aid:1
fname:'Παναγιώτης'
lname:'Τέτσης'
bplace:'Υδρα'
year:1925

- create (tetsis:Artist:Male:Painter { aid:1, fname:'Παναγιώτης', lname:'Τέτσης',bplace:'Υδρα', year:1925})

Οντότητα «Artwork»

ΟΠΑ AUEB

- Σε έναν σχεσιακό κόσμο:

```
CREATE TABLE artwork
   (id INT PRIMARY KEY,  title VARCHAR(200) NOT NULL,  cyear  INT,
    height  INT, width  INT, depth  INT, price DECIMAL (8,2) CHECK(price >=0),
    cid  INT, aid INT,
    CONSTRAINT fk_cid FOREIGN KEY (cid) REFERENCES categories(cid),
    CONSTRAINT fk_aid  FOREIGN KEY (aid) REFERENCES artists(aid)
   );
```

- Το ξένο κλειδί cid χρησιμεύει για να συνδέσουμε ένα έργο τέχνης με τη κατηγορία στην οποία ανήκει και η οποία καταγράφεται στον πίνακα (σχέση) categories

Πόσες κατηγορίες έργων τέχνης έχουμε?

- ArtDB sample dataset
  - Πίνακας
  - Γλυπτό
  - Χαρακτικό
  - Ψηφιδωτό
  - Video Art
  - Αγιογραφία
  - Κόσμημα

Executive Decision

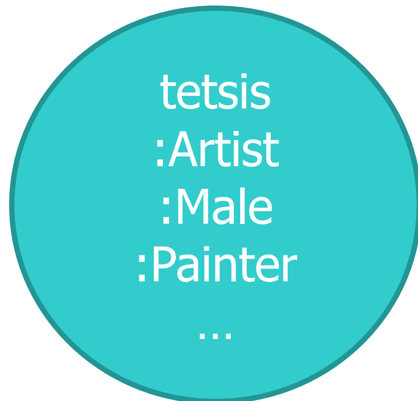
ΟΠΑ AUEB

- Ας χρησιμοποιήσουμε ετικέτες για να αναφερόμαστε στις διαφορετικές κατηγορίες έργων τέχνης: Painting, Sculpture, Engraving,…
- Η κατηγορία όπου ανήκει ένα έργο αποτυπώνεται στην ετικέτα του
  - Επομένως δε χρειάζεται να καταγράψουμε κάπου αλλού στο σχήμα τις κατηγορίες (όπως κάνουμε με τον πίνακα categories σε ένα RDBMS)
  - Με αυτό τον τρόπο δε χρειαζόμαστε πλέον το γνώρισμα cid

Ας φτιάξουμε ένα έργο τέχνης του Τέτση

ΟΠΑ AUEB

- CREATE (nauphgeia:Artwork:Engraving {id:1, title: 'Ναυπηγεία', cyear:1977, height:72, width:90, price:5000})



Παρατήρηση 2

ΟΠΑ AUEB

```
CREATE TABLE artwork
   ( id INT PRIMARY KEY,  title VARCHAR(200) NOT NULL,  cyear  INT,
    height  INT, width  INT, depth  INT, price DECIMAL (8,2) CHECK(price >=0),
    cid  INT, aid INT,
    CONSTRAINT fk_cid FOREIGN KEY (cid) REFERENCES categories(cid),
    CONSTRAINT fk_aid  FOREIGN KEY (aid) REFERENCES artists(aid)
   );
```

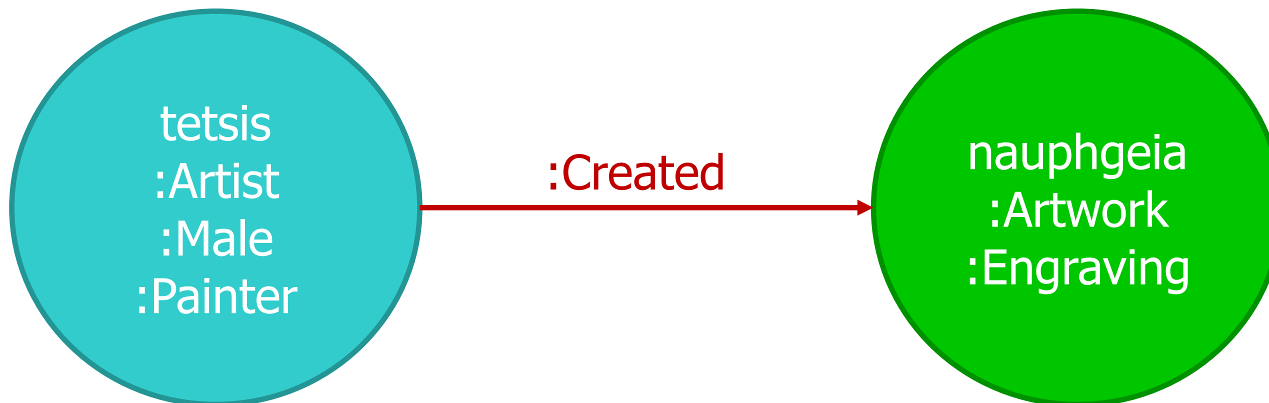
- Τα ξένα κλειδιά χρησιμοποιούνται για τη δημιουργία συνδέσεων μεταξύ οντοτήτων σε μια σχεσιακή βάση.
- Σε μία graph database, οι συνδέσεις μπορούν να οριστούν άμεσα μεταξύ κόμβων!

Αν το κάναμε όπως σε μία σχεσιακή βάση...

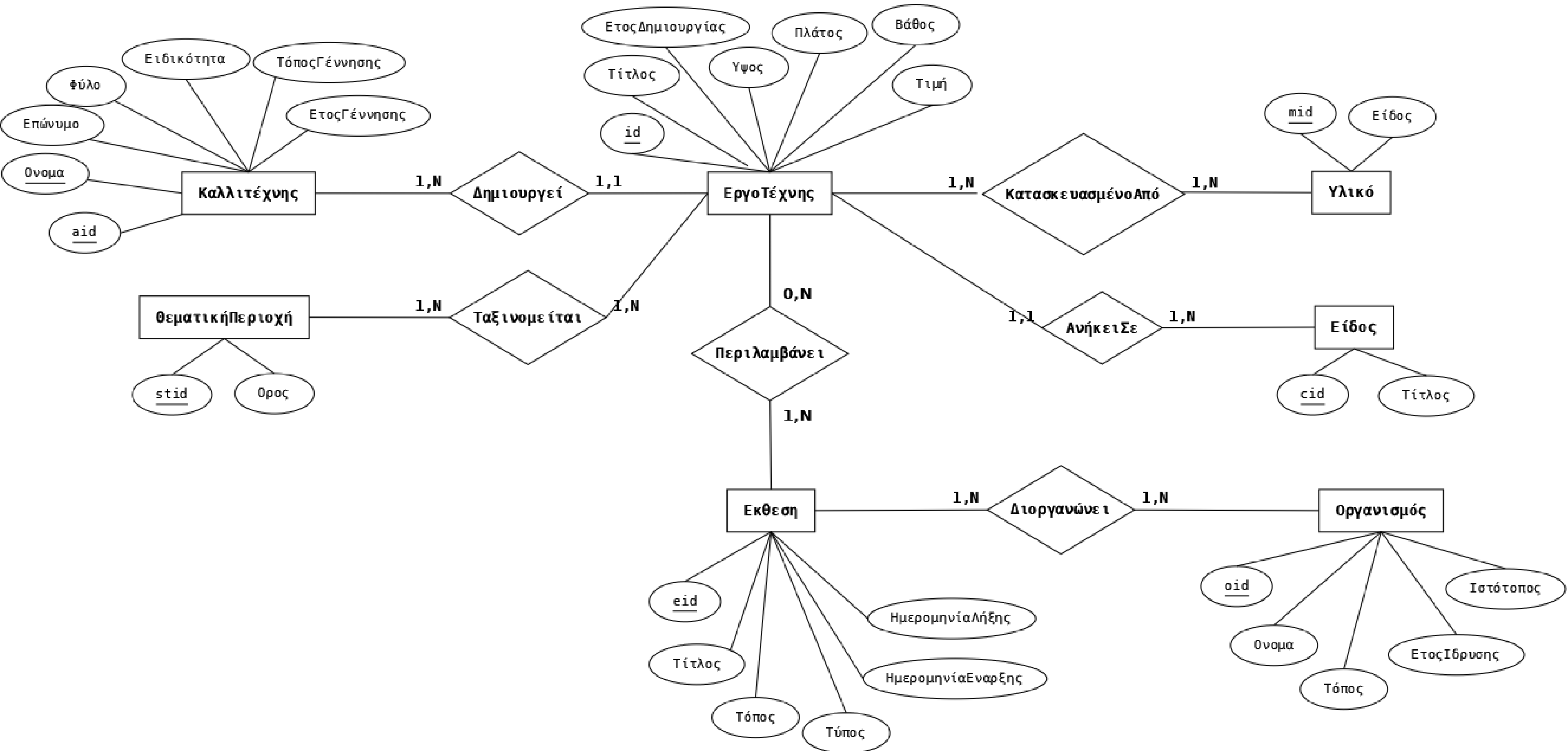
Όμως σε ένα γράφο η αναφορά μπορεί να ορισθεί άμεσα!

ΟΠΑ
AUEB

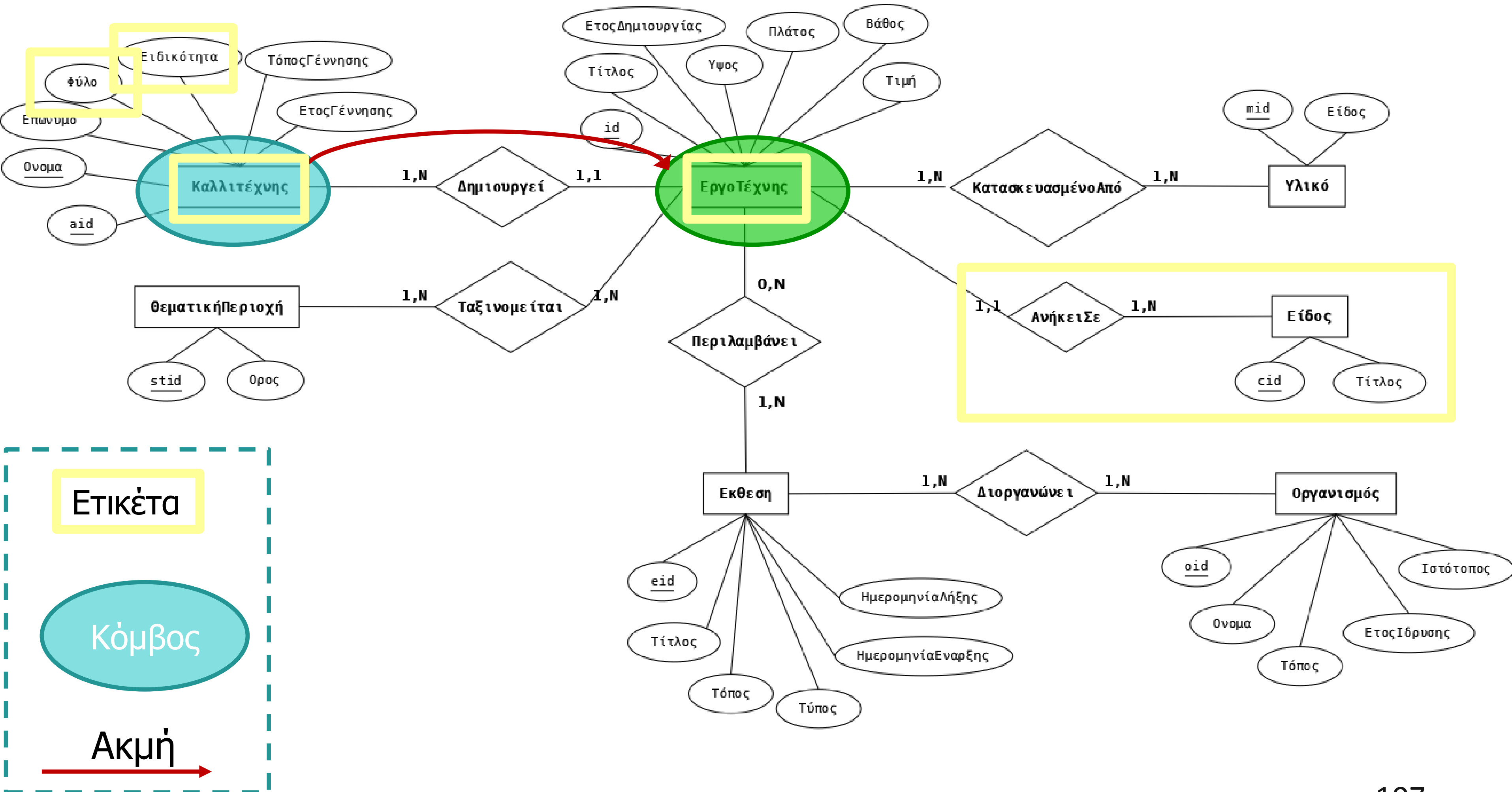
- create (tetsis)-[:Created]->(nauphgeia)



Διάγραμμα ER



Property Graph Model



ΟΠΑ
AUEB

Neo4j Examples

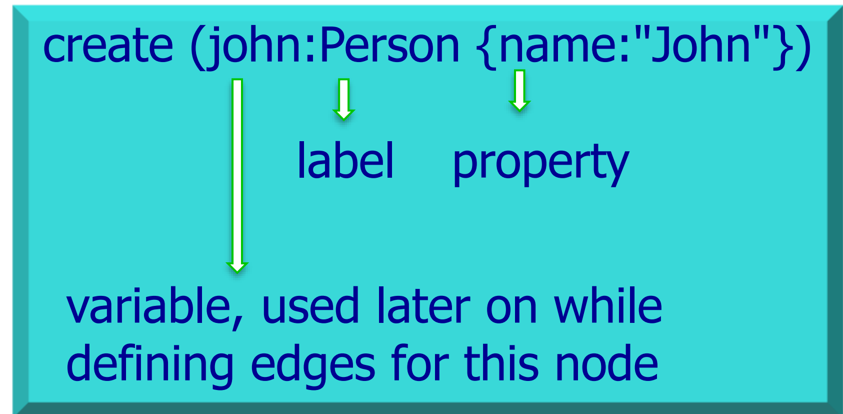
Yannis Kotidis

http://pages.cs.aueb.gr/~kotidis/

Create Social Graph script

```
//cleanup
//DETACH removes all relationships of a node, DELETE removes the node
MATCH (n) DETACH DELETE n;

//create social graph
//label each node as “Person”
create (john:Person {name:"John"})
create (sara:Person {name:"Sara"})
create (jim:Person {name:"Jim"})
create (patrick:Person {name:"Patrick"})
create (mary:Person {name:"Mary"})
create (john)-[:Knows]->(jim)
create (john)-[:Knows]->(sara)
create (jim)-[:Knows]->(sara)
create (sara)-[:Knows]->(patrick)
create (sara)-[:Knows]->(mary)
create (jim)-[:Knows]->(mary);
```



View Graph in Browser

Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

File Action Media View Help

Activities Neo4j Desktop Δευ 09:34

neo4j@bolt://localhost:7687 - Neo4j Browser

File Edit View Window Help Developer

```
$ match (n) return (n);
```

*(5) Person(5)

*(6) Knows(6)

Graph Table Text Code

John, Sara, Jim, Patrick, Mary — Knows

Displaying 5 nodes, 6 relationships.

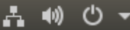
```
$ create (john:Person {name:"John"}) create (sara:Person {name:"Sara"}) create (jim:Person {name:"Jim"}) create (patri…
```

Friend Suggestion

Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

File Action Media View Help

Activities Neo4j Desktop ▾ Δευ 09:51



neo4j@bolt://localhost:7687 - Neo4j Browser

File Edit View Window Help Developer

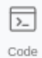
```
1 //suggest friends for john
2 match (john:Person {name:"John"})-[:Knows]-(friend)-[:Knows]->(friend_of_friend)
3 where NOT (john)-[:Knows]->(friend_of_friend)
4 return friend_of_friend, COUNT(*) as weight order by weight desc;
```

$ match (john:Person {name:"John"})-[:Knows]-(friend)-[:Knows]->(friend_of_friend) where NOT (john)-[:Knows]->(f...

Graph

Table

Text



Code

"friend_of_friend"	"weight"
{"name":"Mary"}	2
{"name":"Patrick"}	1

MAX COLUMN WIDTH:

$ match (john:Person {name:"John"})-[:Knows]-(friend)-[:Knows]->(friend_of_friend) where NOT (john)-[:Knows]->(f...

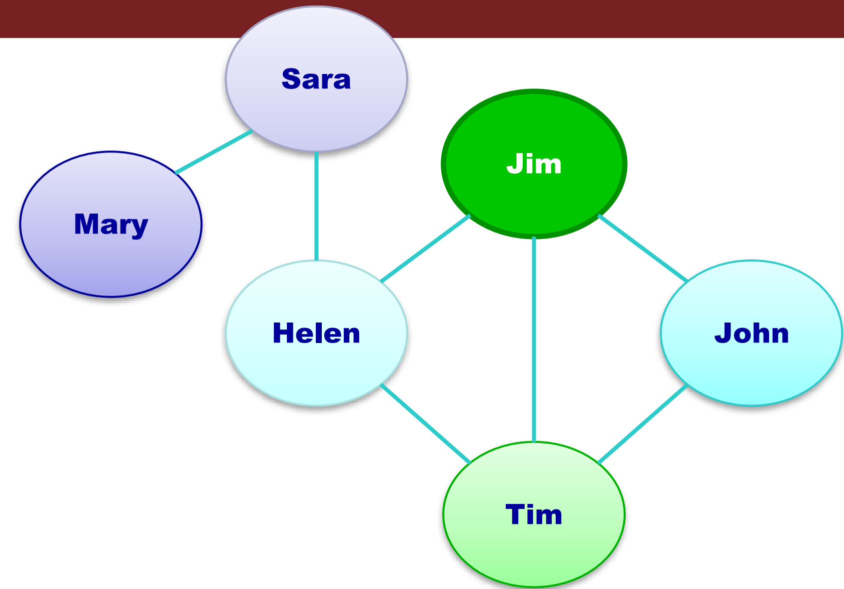
Graph

"friend_of_friend"	"weight"
{"name":"Patrick"}	1

Social Graph 2

ΟΠΑ AUEB

```
//create social graph
merge (mary:Person {name:"Mary"})
merge (sara:Person {name:"Sara"})
merge (jim:Person {name:"Jim"})
merge (helen:Person {name:"Helen"})
merge (tim:Person {name:"Tim"})
merge (john:Person {name:"John"})
merge (john)-[:Knows]->(jim)
merge (john)-[:Knows]->(tim)
merge (jim)-[:Knows]->(tim)
merge (jim)-[:Knows]->(helen)
merge (tim)-[:Knows]->(helen)
merge (sara)-[:Knows]->(helen)
merge (sara)-[:Knows]->(mary)
```

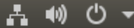


Resulting Graph

Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

File Action Media View Help

Activities Neo4j Desktop Τρι 18:36



neo4j@bolt://localhost:7687 - Neo4j Browser

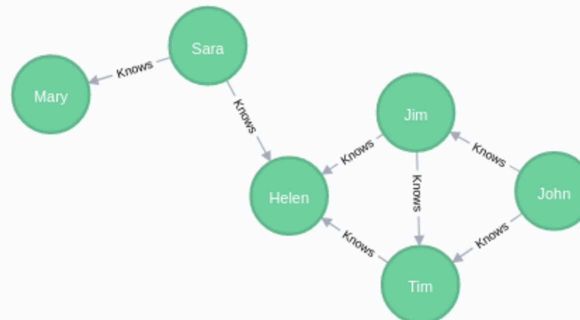
File Edit View Window Help Developer

```
$ match(n) return n
```

Graph Table Text Code

*(6) Person(6)

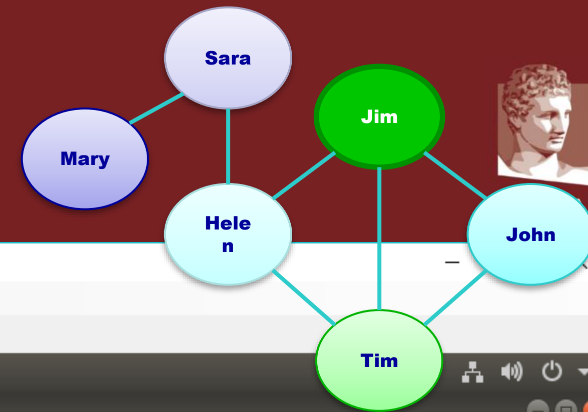
*(7) Knows(7)



Displaying 6 nodes, 7 relationships.

```
$ :play start
```

Single Source Shortest Paths



Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

File Action Media View Help

Activities Neo4j Desktop Τρι 18:56

neo4j@bolt://localhost:7687 - Neo4j Browser

File Edit View Window Help Developer

```
$ match (x:Person{name:"John"}), p=allShortestPaths((x)-[*]-(y)) where x<>y return x as SOURCE,y as TARGET, p as ShortestPath;
```

```
$ match (x:Person{name:"John"}), p=allShortestPaths((x)-[*]-(y)) where x<>y return x as SOURCE,y as TARGET, p as...
```

Graph · Table · Text · Code

"SOURCE"	"TARGET"	"ShortestPath"
{"name":"John"}	{"name":"Mary"}	[{"name":"John"},{},{"name":"Tim"},{"name":"Tim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"},{"name":"Sara"},{},{"name":"Mary"}]
{"name":"John"}	{"name":"Mary"}	[{"name":"John"},{},{"name":"Jim"},{"name":"Jim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"},{"name":"Sara"},{},{"name":"Mary"}]
{"name":"John"}	{"name":"Sara"}	[{"name":"John"},{},{"name":"Tim"},{"name":"Tim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"}]
{"name":"John"}	{"name":"Sara"}	[{"name":"John"},{},{"name":"Jim"},{"name":"Jim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"}]
{"name":"John"}	{"name":"Jim"}	[{"name":"John"},{},{"name":"Jim"}]
{"name":"John"}	{"name":"Helen"}	[{"name":"John"},{},{"name":"Jim"},{"name":"Jim"},{},{"name":"Helen"}]
{"name":"John"}	{"name":"Helen"}	[{"name":"John"},{},{"name":"Tim"},{"name":"Tim"},{},{"name":"Helen"}]
{"name":"John"}	{"name":"Tim"}	[{"name":"John"},{},{"name":"Tim"}]

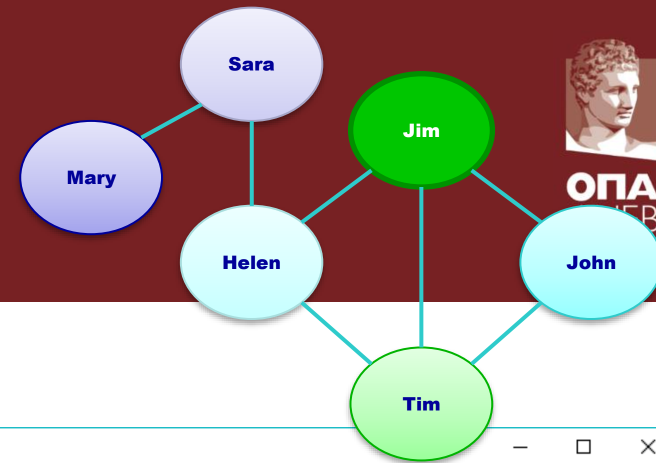
MAX COLUMN WIDTH:

Closeness Centrality Algorithm

ΟΠΑ AUEB

```
CALL algo.closeness.stream('Person', 'Knows')
YIELD nodeId, centrality
RETURN algo.getNodeById(nodeId).name AS
node, centrality
ORDER BY centrality DESC;
```

Closeness Centrality Calculations on Sample Graph



Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

neo4j@bolt://localhost:7687 - Neo4j Browser

```
CALL algo.closeness.stream('Person', 'Knows')
YIELD nodeId, centrality

RETURN algo.getNodeById(nodeId).name AS node, centrality
ORDER BY centrality DESC
LIMIT 20;
```

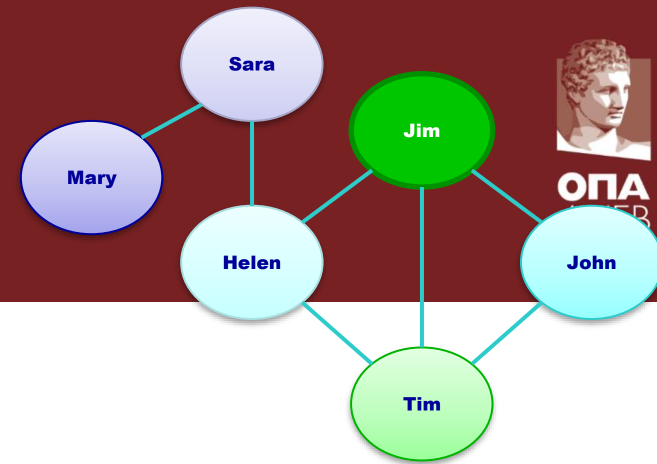
$ CALL algo.closeness.stream('Person', 'Knows') YIELD nodeId, centrality RETURN algo.getNodeById(nodeId).name A...

node	centrality
"Helen"	0.7142857142857143
"Jim"	0.625
"Tim"	0.625
"Sara"	0.5555555555555556
"John"	0.45454545454545453
"Mary"	0.38461538461538464

Betweenness Centrality Algorithm

```
CALL algo.betweenness.stream('Person','Knows',{direction:'Both’})
YIELD nodeId, centrality
RETURN algo.getNodeById(nodeId).name  AS name, centrality
ORDER BY centrality DESC;
```

Betweenness Centrality Calculations on Sample Graph



Ubuntu 18.04.1 LTS Neo4j on DESKTOP-7NP8VAL - Virtual Machine Connection

File Action Media View Help

Activities Neo4j Desktop Τρι 09:58

neo4j@bolt://localhost:7687 - Neo4j Browser

File Edit View Window Help Developer

```
CALL algo.betweenness.stream('Person','Knows',{direction:'Both'})
YIELD nodeId, centrality

RETURN algo.getNodeById(nodeId).name  AS name, centrality
ORDER BY centrality DESC;
```

$ CALL algo.betweenness.stream('Person','Knows',{direction:'Both'}) YIELD nodeId, centrality RETURN algo.getNod...

name	centrality
"Helen"	6.0
"Sara"	4.0
"Jim"	1.5
"Tim"	1.5
"Mary"	0.0
"John"	0.0