ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

Διαχείριση και Ανάλυση Γράφων

Ι. Κωτίδης

Καθηγητής ΟΠΑ

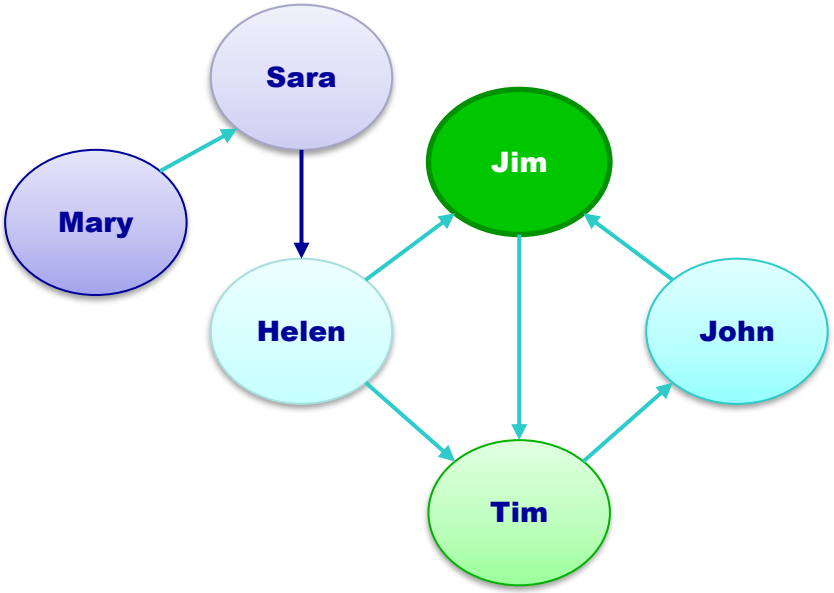
kotidis@aueb.gr

Overview

- Graph databases
- History of graphs
- Modern application examples
- Property Graph Model (neo4j)
- Creating and querying graph databases
- The ArtDB dataset as a graph

Βάση Δεδομένων Γράφων

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

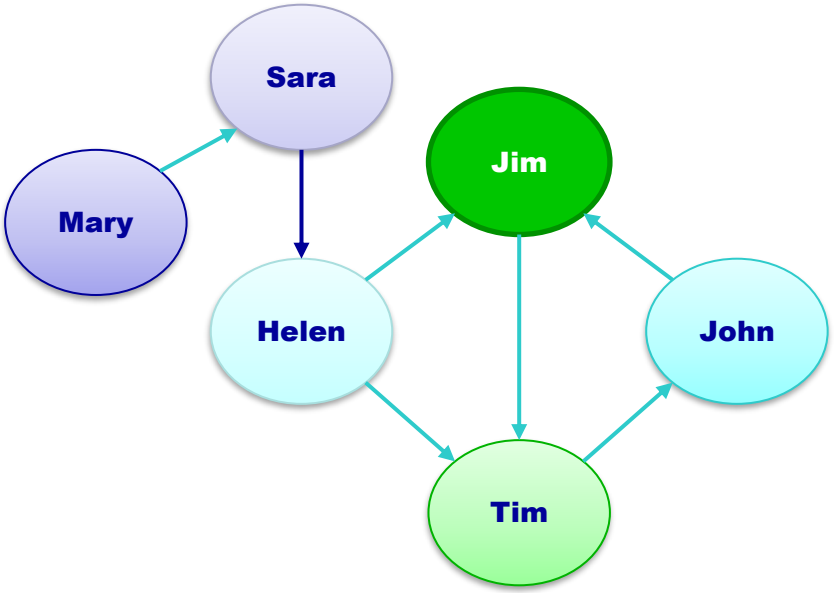


Βάση Δεδομένων Γράφων

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

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

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

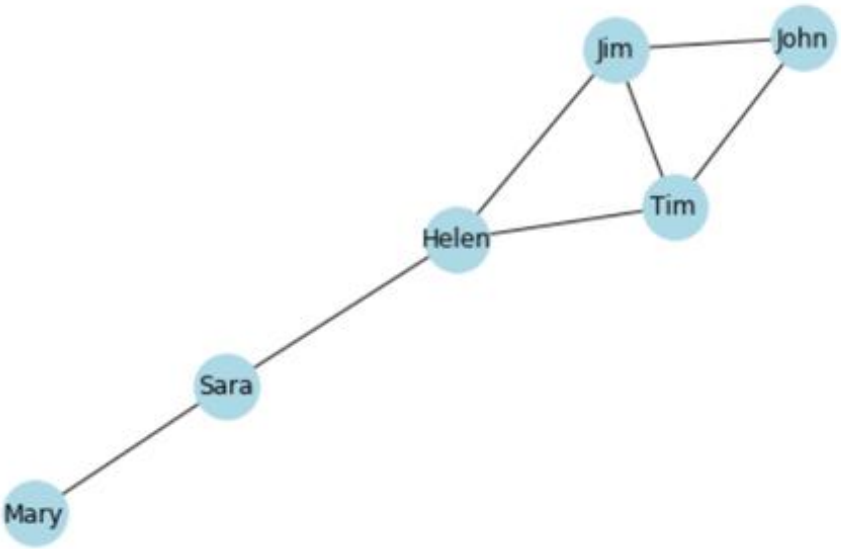


Παράδειγμα χρήσης γράφων σε μία γλώσσα προγραμματισμού (python)

ΟΠΑ AUEB

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

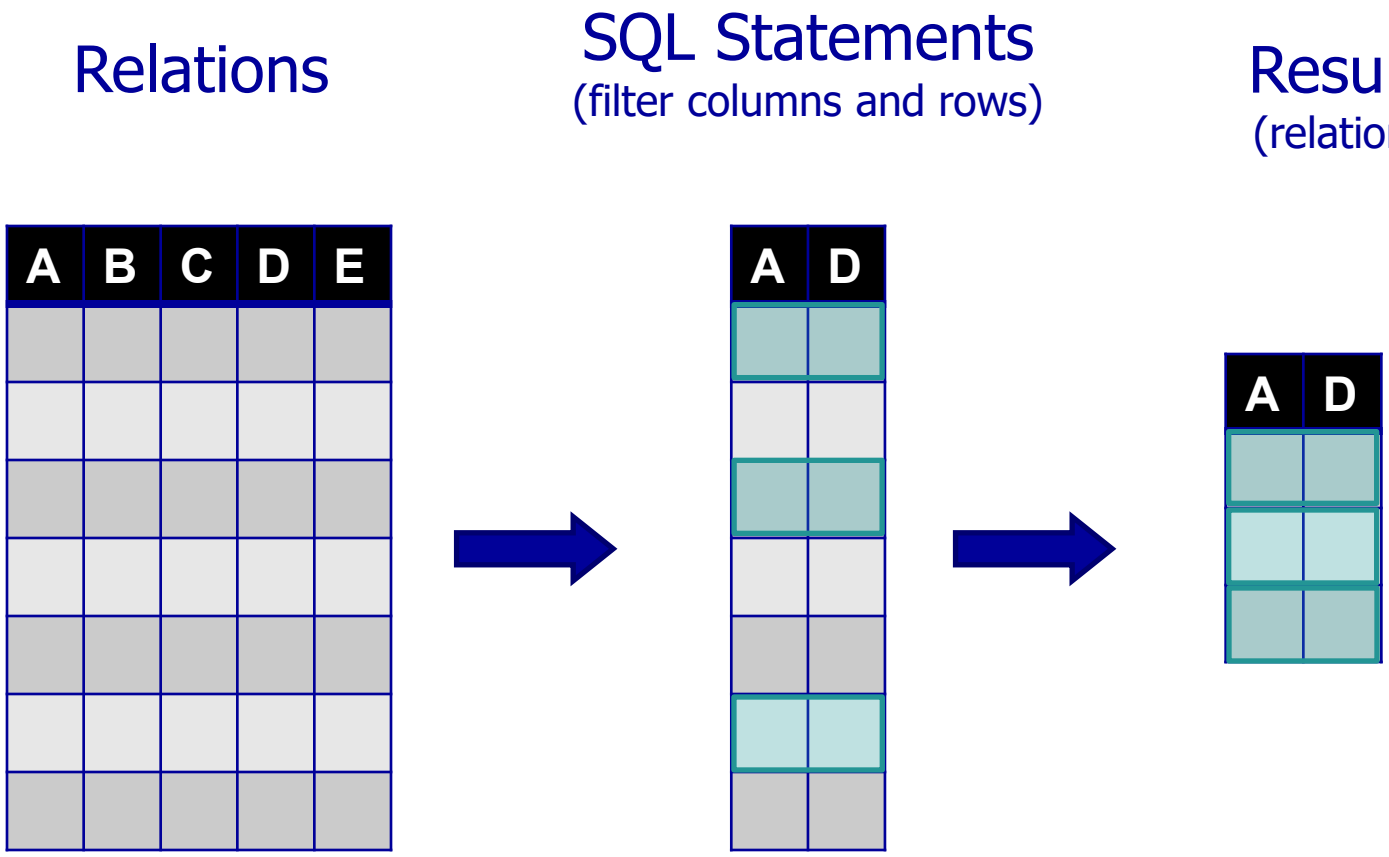
In [2]:
```
G.add_nodes_from(['John','Mary','Sara','Helen','Tim','Jim'])

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?

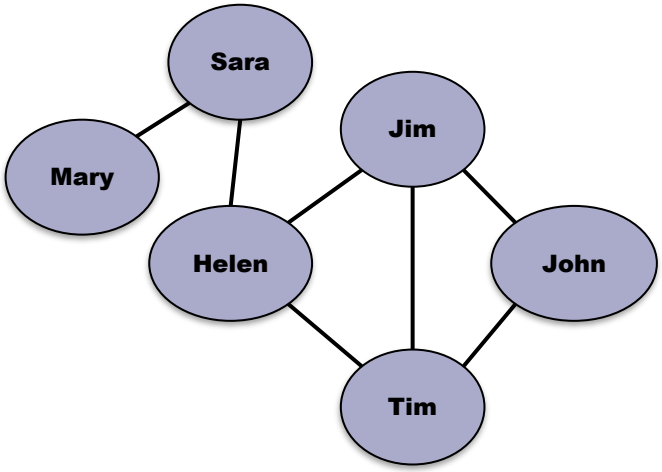
- (Graph) DBMS are optimized for storing datasets much larger than main memory into secondary storage.
- (Graph) DBMS manage issues such as concurrency, integrity, and recovery from hardware failures.
- (Graph) DBMS provide declarative languages for managing and querying large datasets.

Relational Database Usage

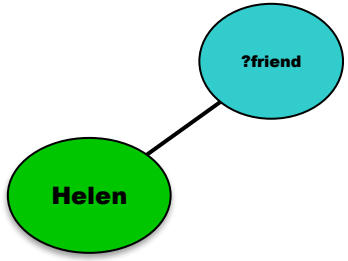


Graph Database Usage (1)

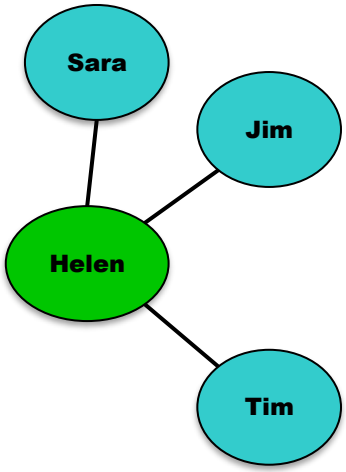
Dataset
(graph)



Statements
(graph/pattern)

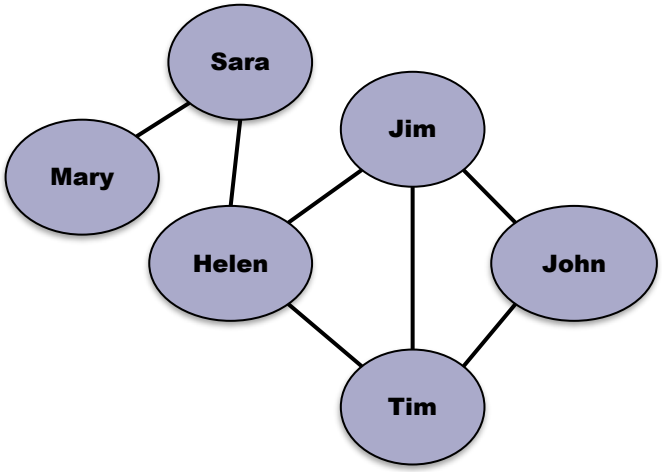


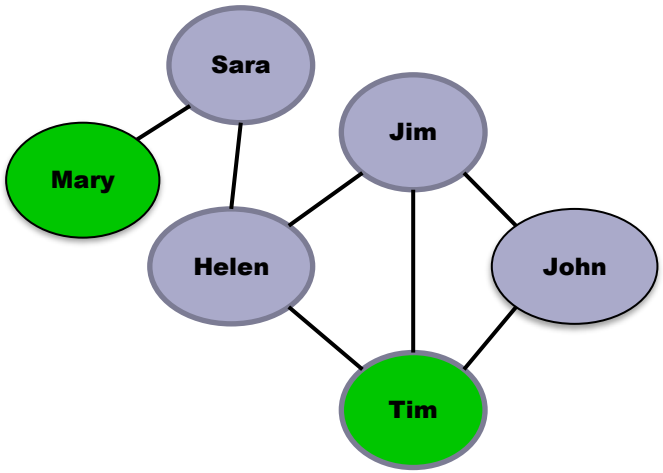
Results
(graph)

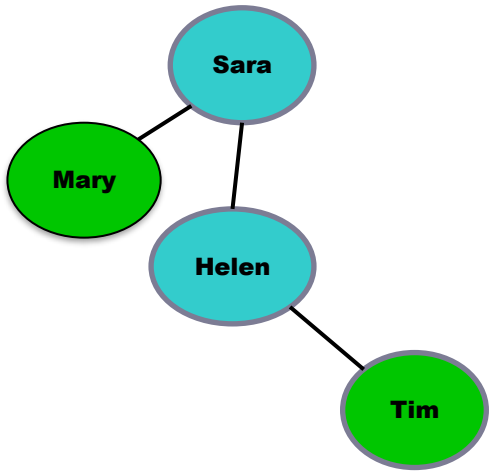


Graph Database Usage (2)

Graph → Statements (shortest paths) → Results





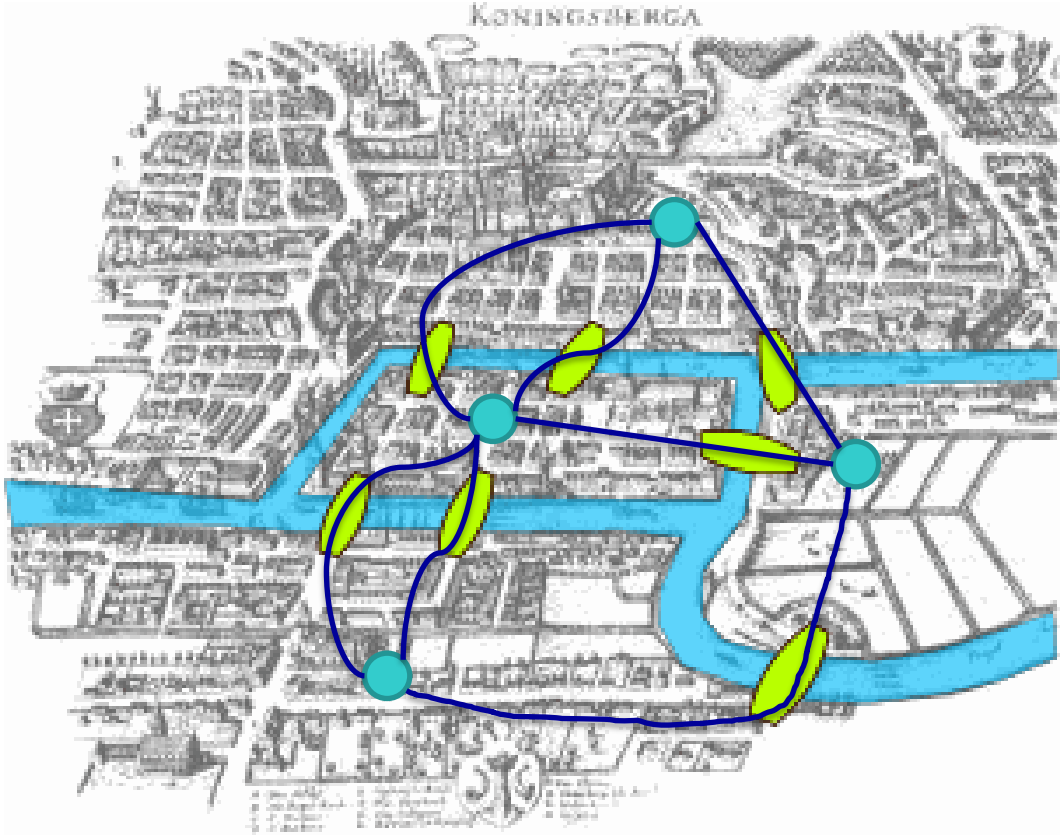


But Why Graphs?

- Graphs are elegant, versatile and easy to comprehend data structures that can be used to model many real work problems
  - The WWW, social interactions, transportation networks, supply-chains, financial networks
- Graphs exemplify interactions (relationships) that are hard to express or observe in other data models

*Λέοναρντ Όιλερ (1707-1783)

History of Graphs: The Euler* Path



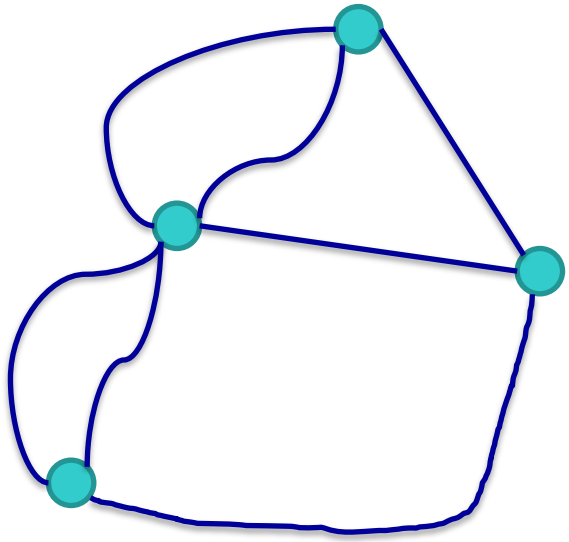
Some citizens of Königsberg
Were walking on the strand
Beside the river Pregel
With its seven bridges spanned.

"O Euler, come and walk with us,"
Those burghers did beseech.
'We'll roam the seven bridges o'er,
And pass but once by each."
"It can't be done," thus Euler cried.
Here comes the Q. E. D.
Your island are but vertices
And four have odd degree."

From Königsberg to Konig's book
So runs the graphic tale
And still it grows more colorful
In Michigan, and Yale.

"The Expanding Unicurse"
Blanche Descartes

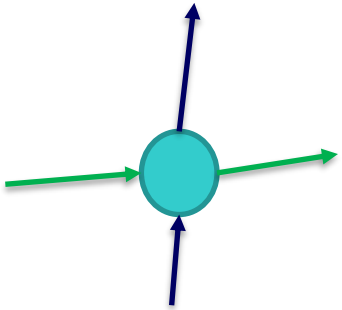
Euler's abstraction



- Map to to graph elements
  - City parts → graph nodes
  - Bridges → graph edges
- Problem abstraction
  - visit every graph node (city part), cross every edge (bridge), without having to walk a single bridge twice
- Sequence of bridges crossed is important to solving this problem

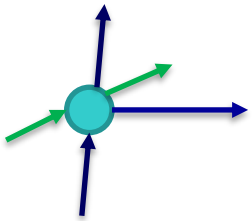
Key observation 1

- Intermediate nodes in the path need an even number of edges (bridges)
  - Because you arrive and leave from these parts of the city



Key observation 2

- Assume start and end nodes are different
- Start node must have an odd number of bridges
  - Otherwise you will get stuck in that part of the city if you ever visit it again

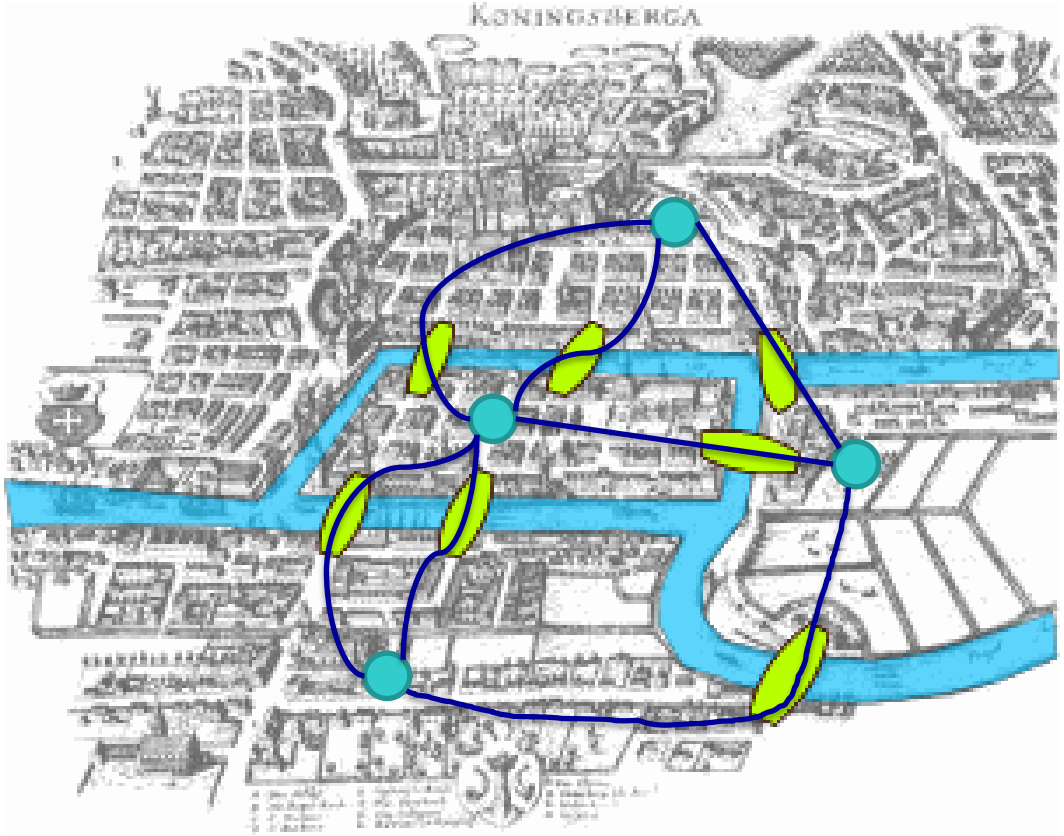


- Same argument for ending node

Euler's Conjecture

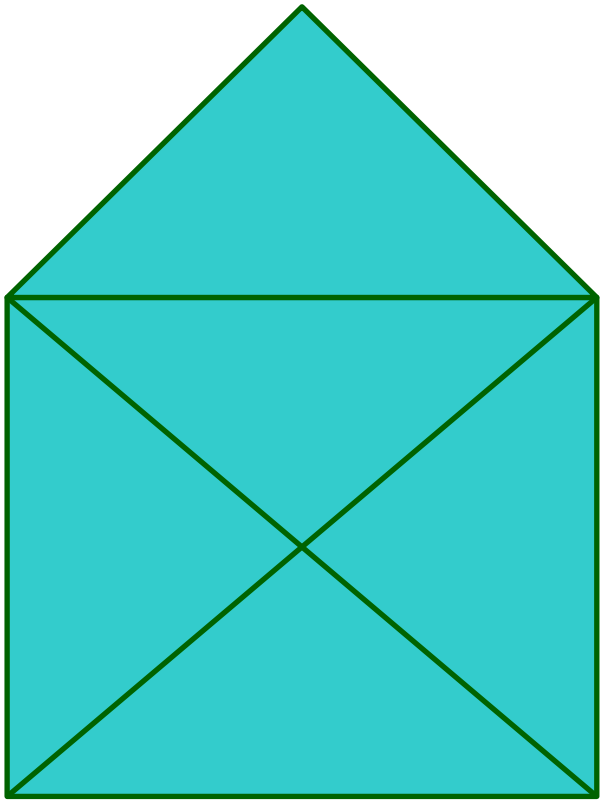
- Graph nodes must have even number of edges
- There can be zero or two nodes with odd number of edges

Back to the map

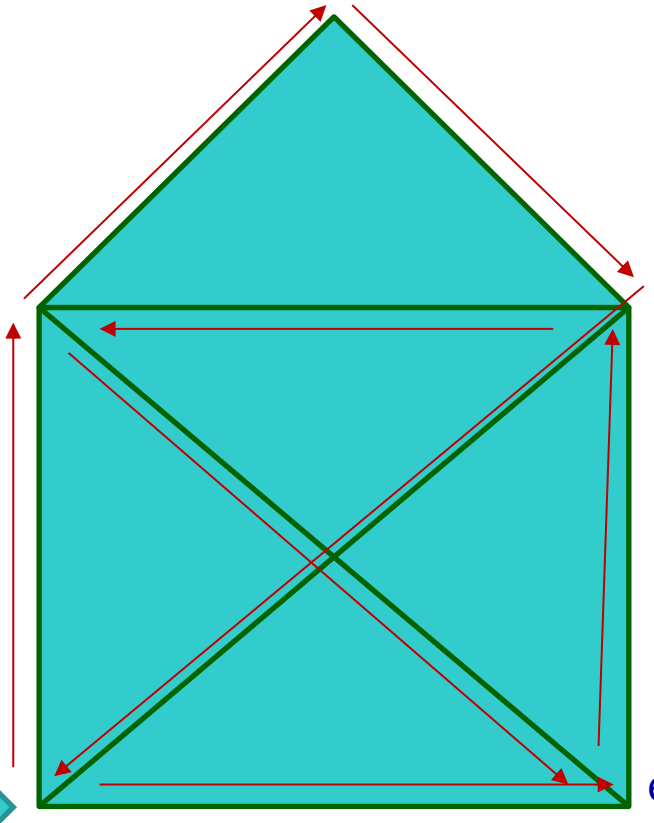


- All parts of the city have even number of bridges connecting them with the rest of the city
- Thus, no Euler Path exists

Remember this game?



Remember this game?



Euler says: start here

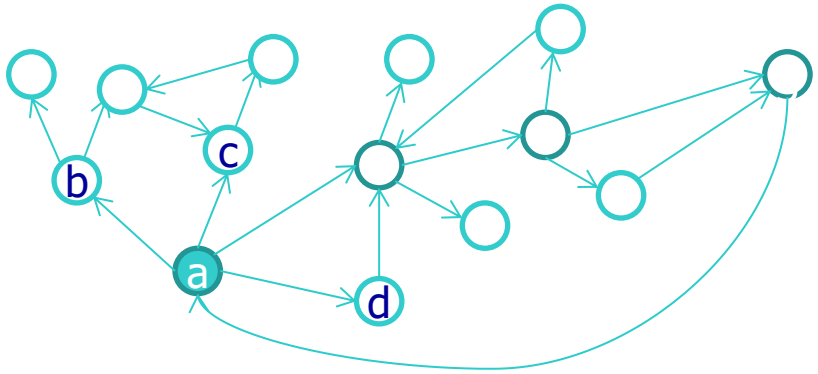
end here

Modern Application: Social Networks

- How to represent relationships in a social network dataset?
- Easy….
  - Use the Relational Data Model

Social Network Example

- FriendOf table stores relationships
  - Table stores the "edges" of the graph
  - Another table may store the "nodes" along with their properties
    - Profile of the user

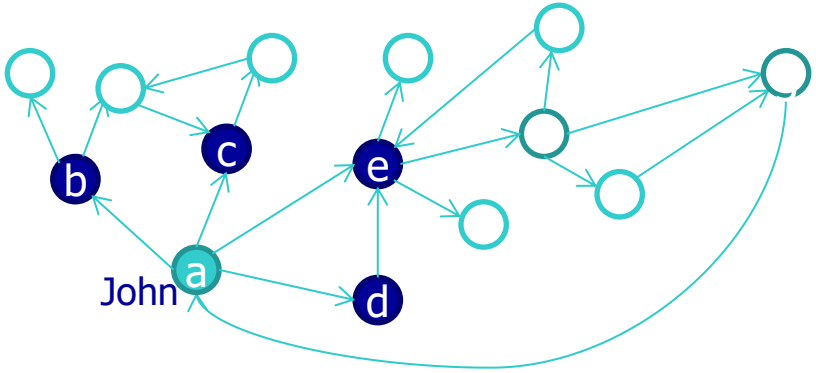


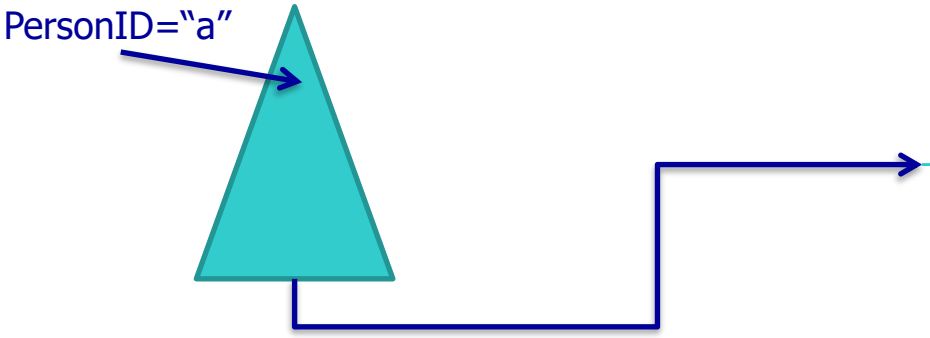
FriendOf

PersonID	FriendID
a	b
a	c
a	e
..	..
e	f

Social Network one-hop query

- Find the friends of "John" (node a)
- One-hop traversal
- Easy: index scan



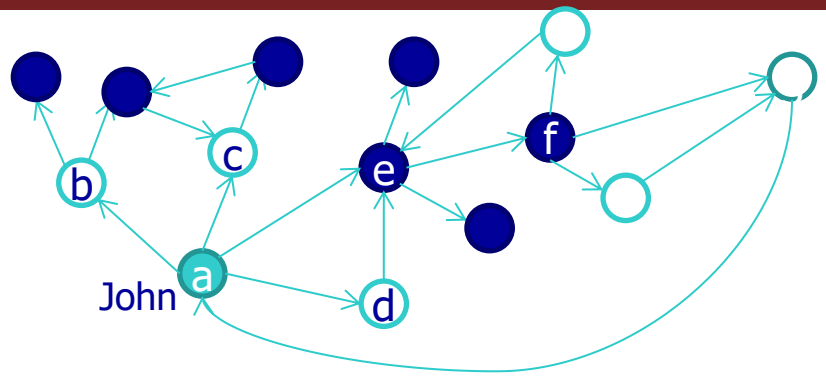


FriendOf

PersonID	FriendID
a	b
a	c
a	e
..	..
e	f

Social Network two-hop query

- Find the friends-of-friends of John
- Harder to compute: self-join
- How do we find the k-hop neighbors of John?



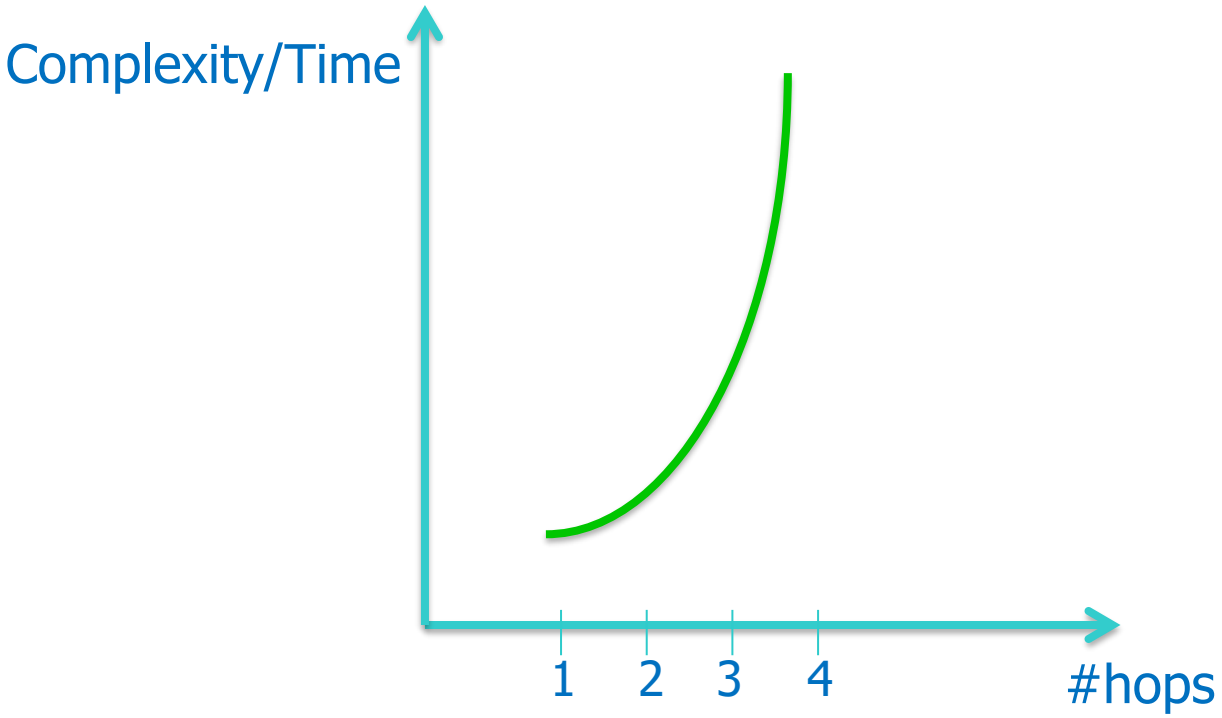
FriendOf

PersonID	FriendID
a	d

FriendOf

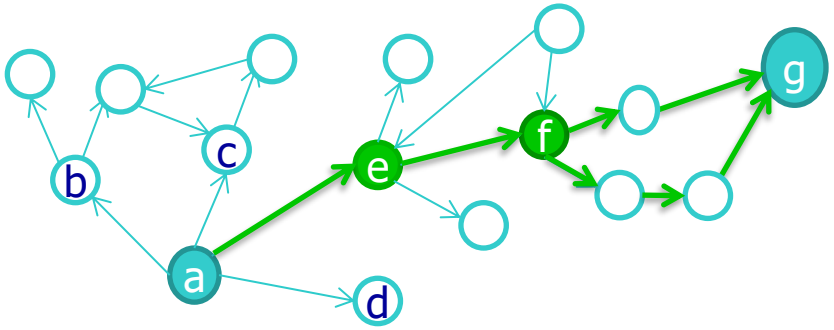
PersonID	FriendID
d	e

Performance of RDBMs on multi-hop queries



Social Network reachability query

- Is there a path connecting a and g?
- Observation: both paths include nodes e,f
  - Thus, e and f are important in communicating information from a to g
  - Removal of one of these nodes results in loss of communication between a and g

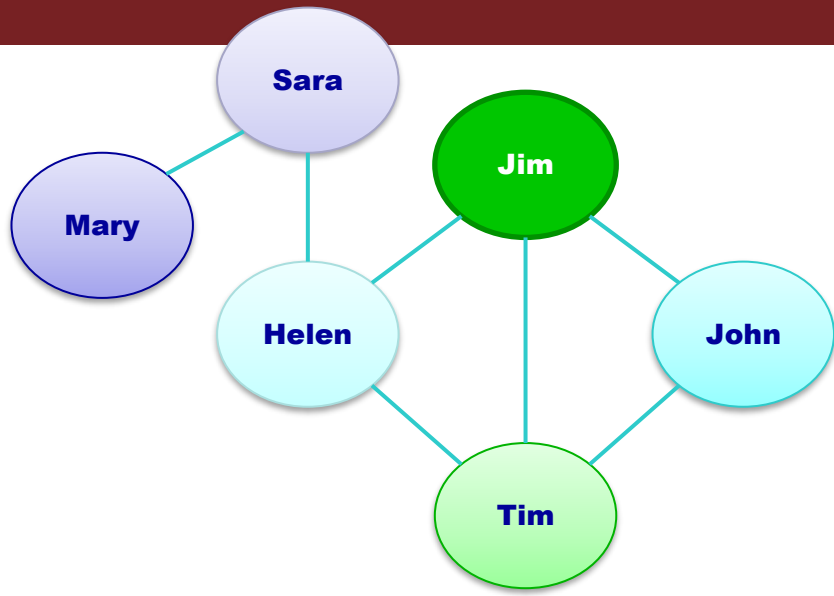


FriendOf

PersonID	FriendID
..	..
a	e
..	..
e	f
..	..
f	g

Measure connectivity/popularity of a user?

- Simplest idea: node degree
  - The degree of a vertex in an undirected graph is the number of edges incident with it

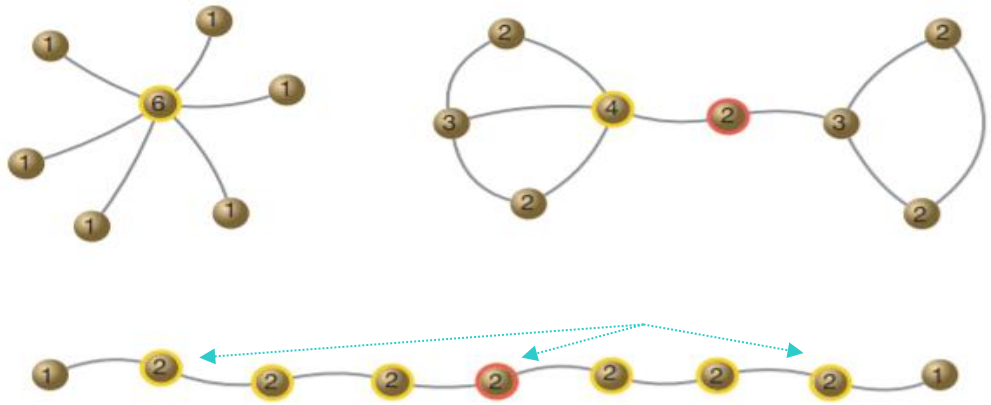


Node	Degree
Mary	1
Sara	2
Helen	3
Jim	3
Tim	3
John	2

Degree centrality

Degree Centrality

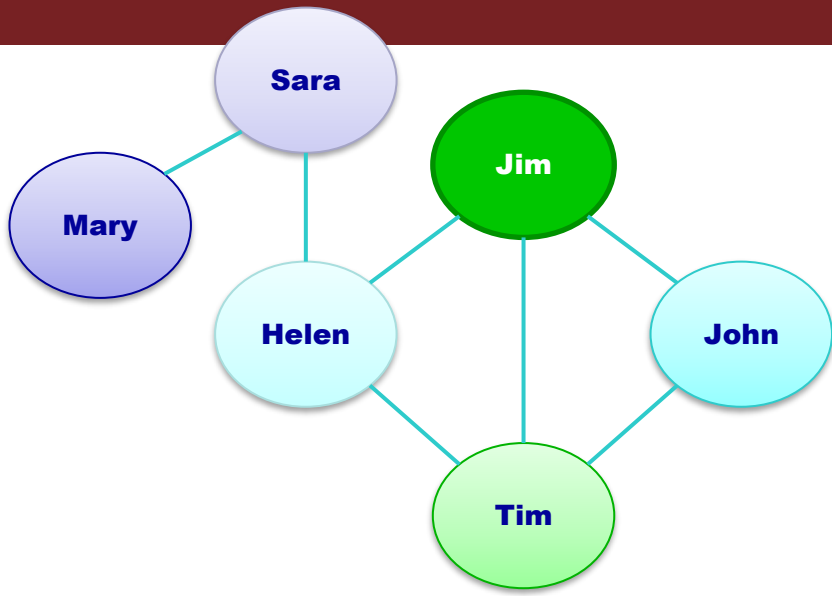
- Shortcomings of degree-based centrality:



- Often a low-degree node is crucial for enabling the communication of other nodes in the graph
- Node degree captures connectivity to adjacent nodes but ignores distances to other nodes in the graph

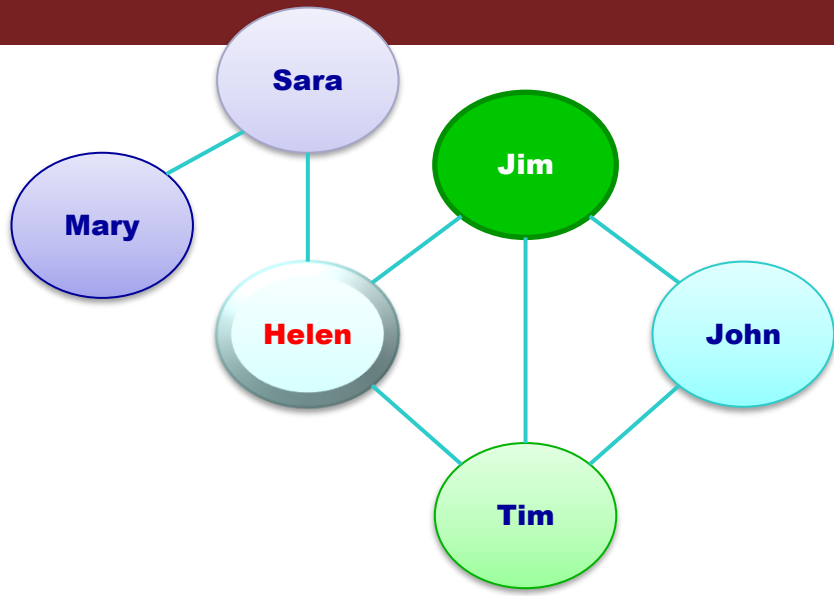
Extend degree computation

- Node degree captures connectivity to adjacent nodes but ignores distances to other nodes in the graph
- Consider shortest distances of a node to all other nodes
  - Is an unweighted graph shortest distance matches the minimum number of hops between nodes



Basic distance computations

- Lengths of shortest paths from Helen to all other nodes
  - Helen→Mary : 2
  - Helen→Sara: 1
  - Helen→Jim: 1
  - Helen→Tim: 1
  - Helen→John: 2

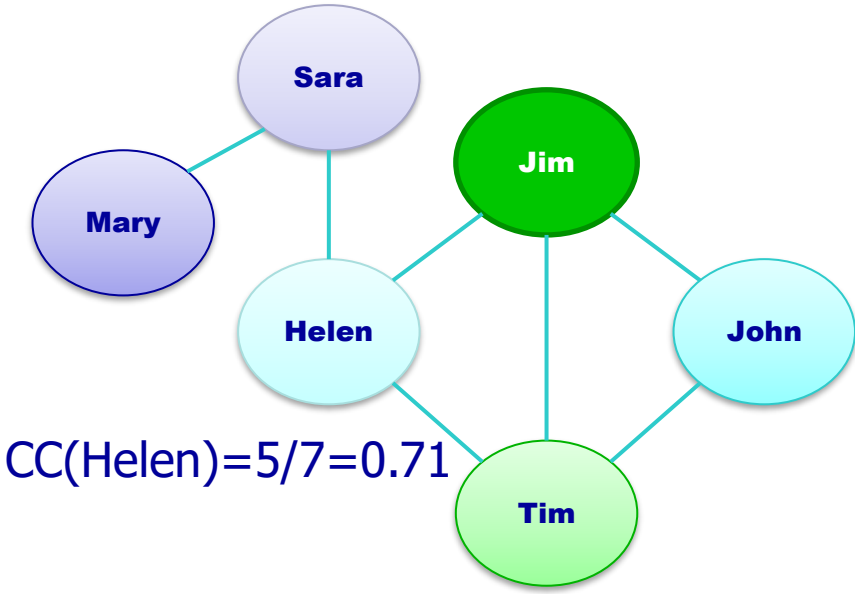


AVG DISTACE = 7/5 = 1.4

A node is deemed "central" if this number is small

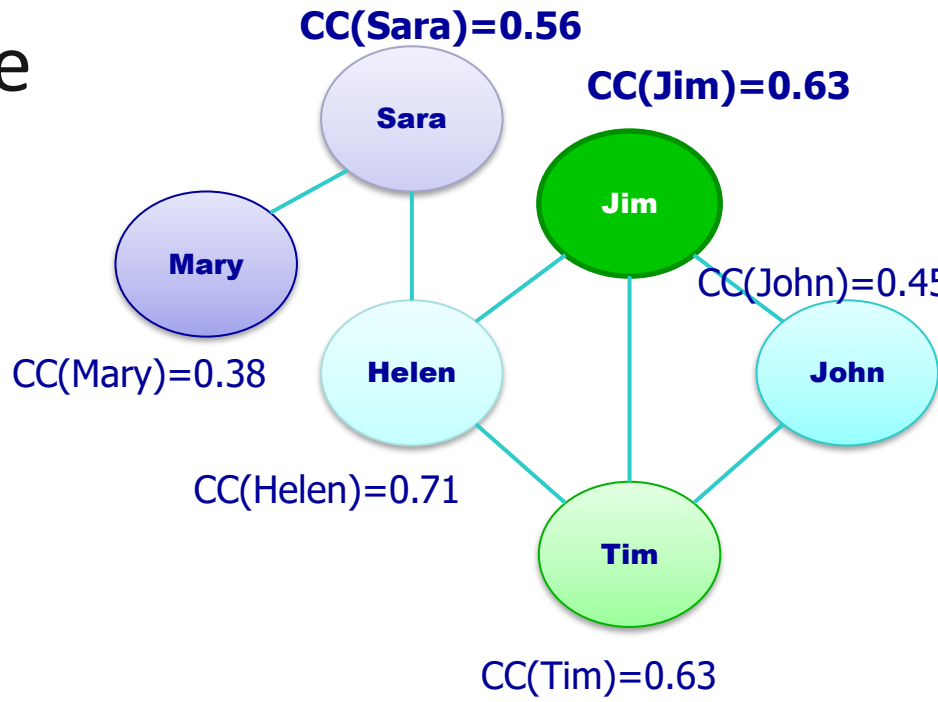
Closeness Centrality (CC)

- CC = inverse of avg distance
  - Small avg distance → high closeness centrality
  - What is the range of possible values for CC(n)?
    - When is CC(n)=1?



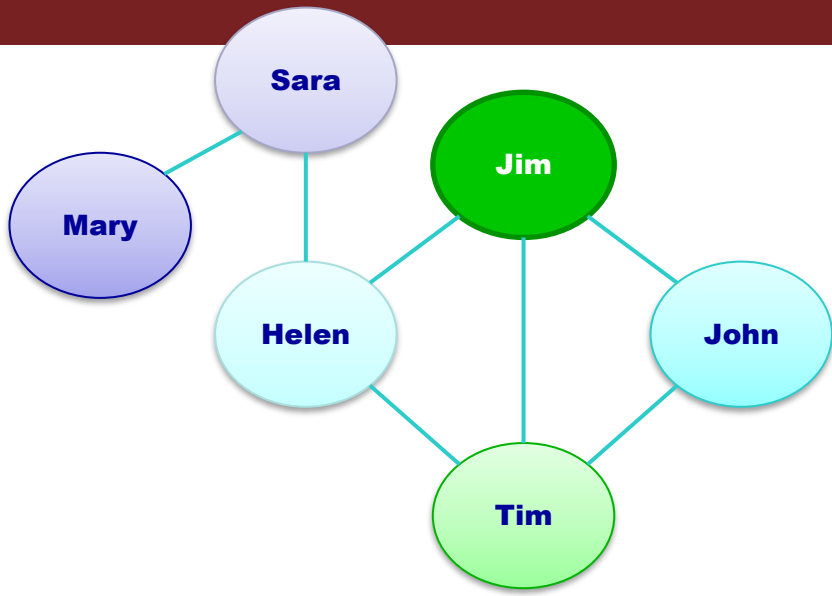
Closeness Centrality Discussion

- Note that Jim & Tim are more central than Sara
- However, removal of Sara bisects the graph



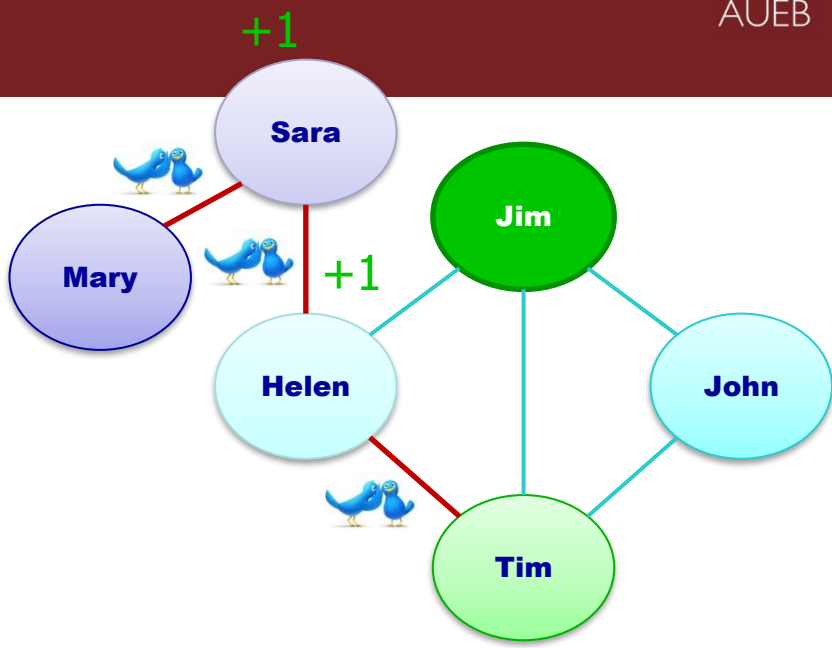
Betweenness Centrality (BC)

- Want to capture importance of nodes in information passing
- CC measures inverse of avg path length to all other nodes
  - Some of these paths are not as important if alternative routes exist



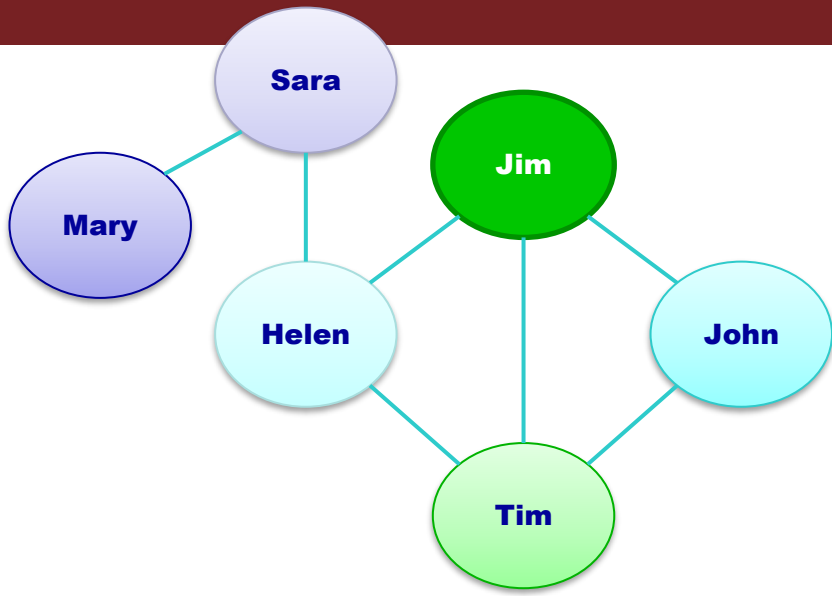
Shortest Path

- Fastest method to pass a message across
- Mary sends a message to Tim through Sara & Helen
- Sara & Helen are rewarded for their contribution



Definition of BC

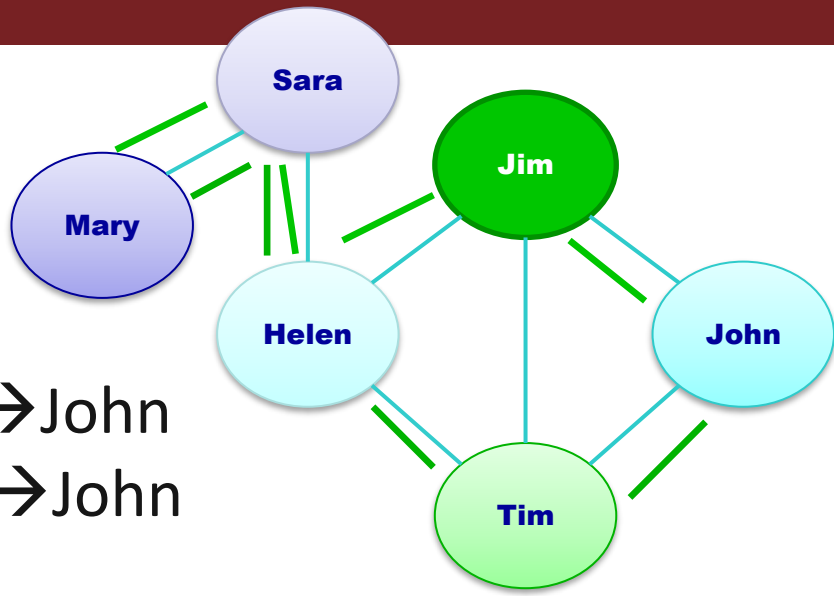
- BC = number of shortest paths from all vertices to all others that pass through that node



- Note
  - Only consider paths with more than 2 nodes
  - When multiple shortest paths exist, split rewards
    - See next slide for an example

Multiple Shortest Paths

- Mary sends message to John
  - There are two shortest paths

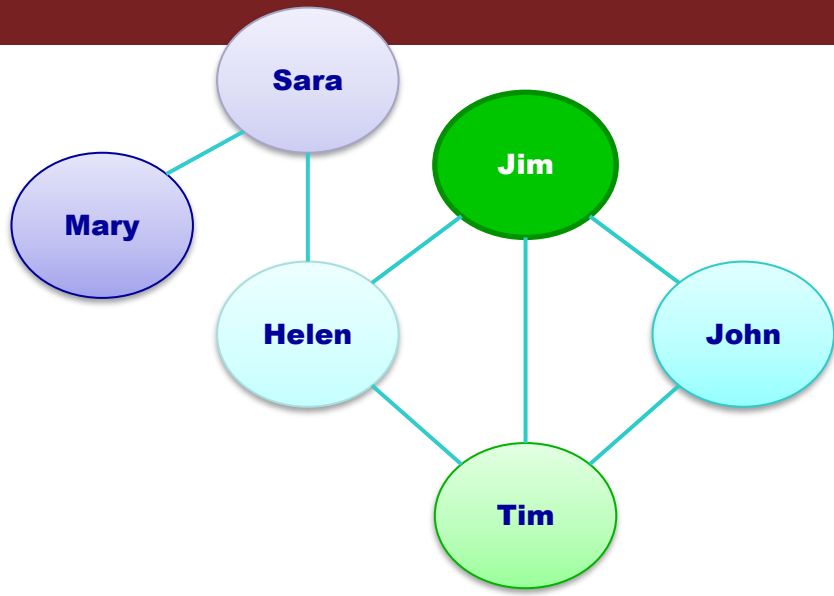


- SP1: Mary→Sara→Helen→Jim→John
- SP2: Mary→Sara→Helen→Tim→John

- Rewards:
  - Sara: +.5 + .5
  - Helen: +.5 + .5
  - Jim: +.5
  - Tim: +.5

Reward Calculation

Node	Betweenness Centrality
Mary	0
Sara	4
Helen	6
Jim	1.5
Tim	1.5
John	0



Graph Augmentation Problem

- Select k edges (**short-cuts**) to add in a graph **G** in order to improve its *connectivity*.
- To quantify the **gain** in the connectivity of the augmented graph $\mathbf{G_{aug}}$ we can compute the reduction of the average shortest path length **L:**

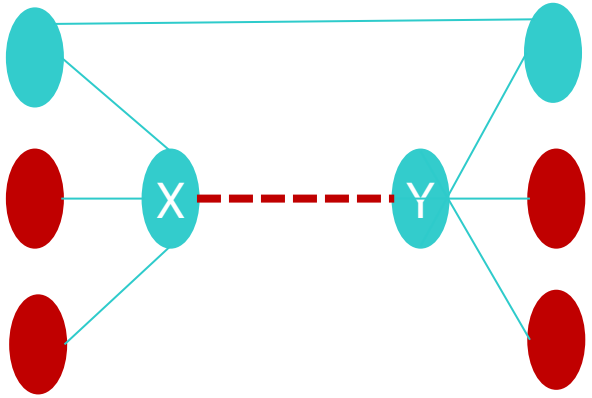
$$gain = L(G) - L(G_{aug})$$

$$\text{gain} = 3 - 2{,}28 = 0{,}72$$

Sub-graph Augmentation Problem

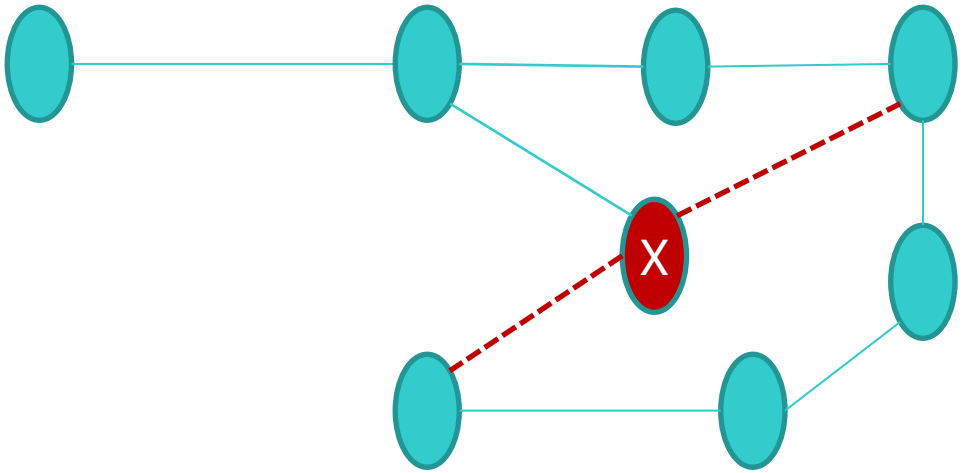
- Minimize the average shortest path length between a **subset** V_S of the nodes in V.
  - e.g. increase connectivity of nodes in a selected target group
  - optimal selection may include shortcuts between nodes not in V_S.

$$\text{Sub-graph gain} = 4 - 2{,}66 = 1{,}34$$



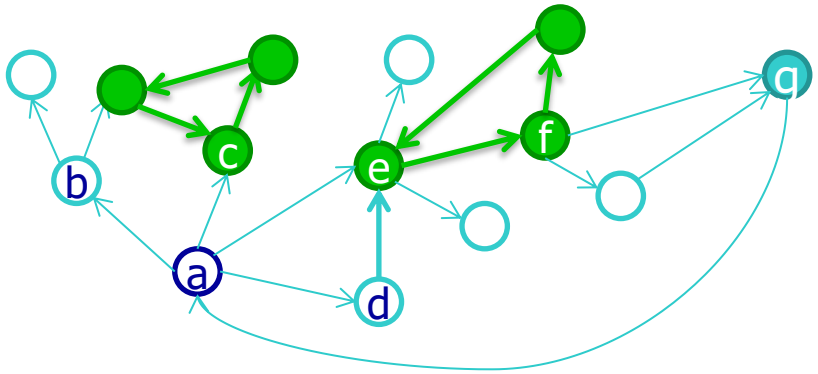
Node Augmentation Problem

- Select k edges to add in a specific node n of the graph in order to minimize its average shortest path length to all nodes in G.
  - Increases the closeness centrality (CC) of a selected node.



Using Motifs (patterns) as features

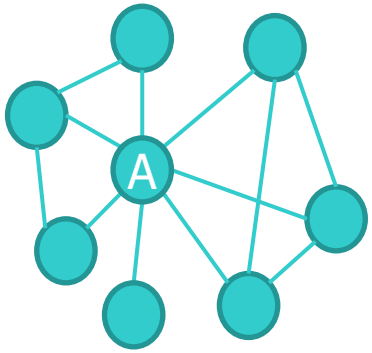
- Example: local triangle counting: count the number of triangles node v participates in

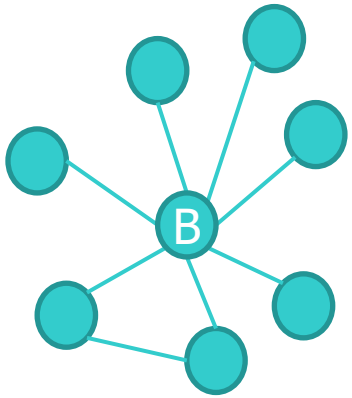


- Why is this useful?

Detect Fake Users In Social Networks

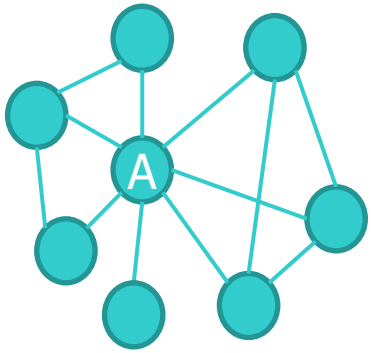
- Which of the two users A,B is (probably) a fake account?



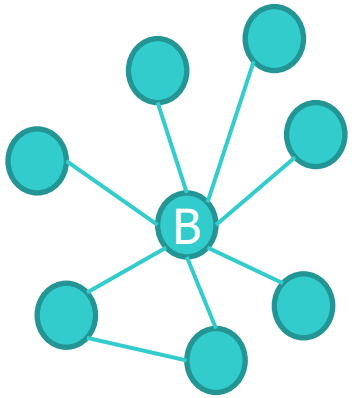


Detect Fake Users In Social Networks

- Assumption: fake accounts add friends at random
  - Thus, real users have higher local triangle counts (TC)



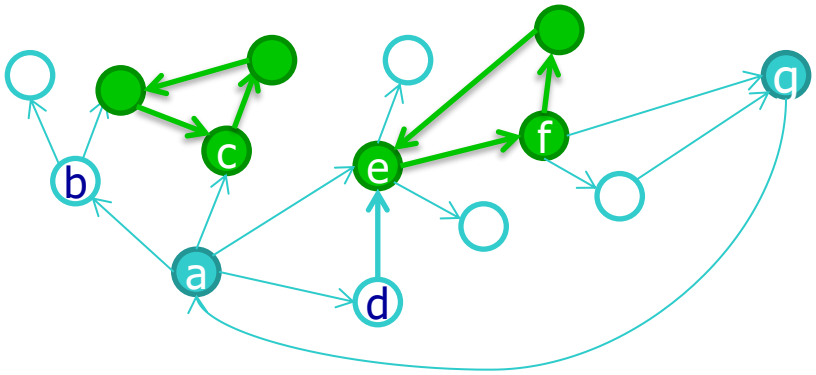
$TC_A = 5$



$TC_B = 1$

Pattern Matching

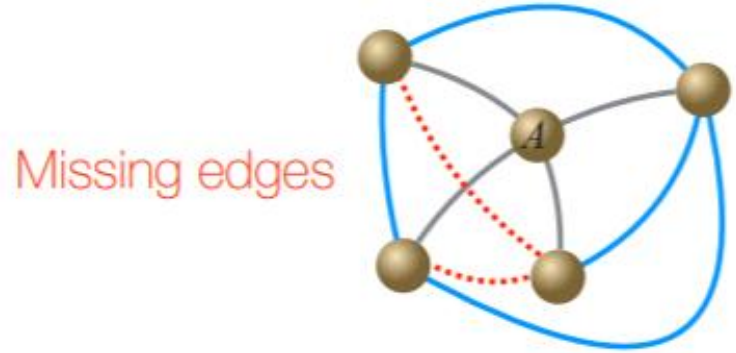
- Find all triplets of nodes that form a triangle
  - pattern : "x→y, y→z, z→x"
- Related problems
  - Connected components
  - Graph isomorphism



Local Clustering Coefficient

- The Local clustering coefficient C(A) of a node A, quantifies how close its neighbors are to being a clique (fully connected)
- Assume nodes depict users in a social network and edges their relationships
  - The clustering coefficient C(A) of node A is defined as the probability that two randomly selected friends of A are friends themselves
    - i.e. the fraction of all pairs of A's friends who are also friends
  - Defined only if A has at least two friends (otherwise 0)
  - The clustering coefficient is always between 0 and 1

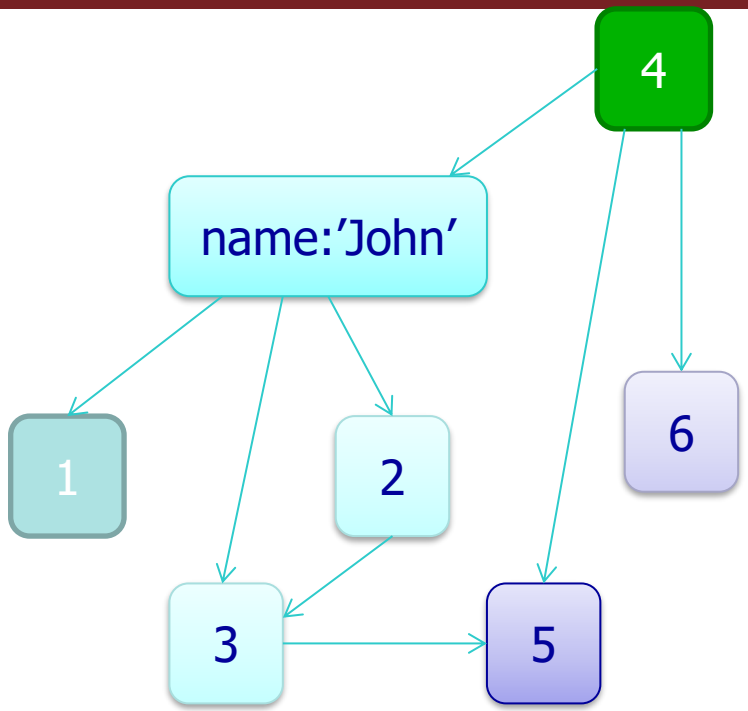
Local Clustering Coefficient
(Simple undirected graph)



- **Node A** has k=4 friends
- Among the four friends, there are k×(k-1)/2 = (4×3)/2 = 6 possible friendships
- But only four of them are actually present (blue edges)
- Two are missing (dotted red edges)
- Thus, the clustering coefficient of **node A** is C(A)=4/6=0.6666, or about 67%

Node Local Clustering Coefficient

- Probability that two neighbors of John know each other
- In this example we treat the graph as **undirected**
- John has n=4 neighbors
  - Combinations(4,2)=6
  - Nodes (2) and (3) know each other (m=1)
  - Clustering Coefficient = 1/6

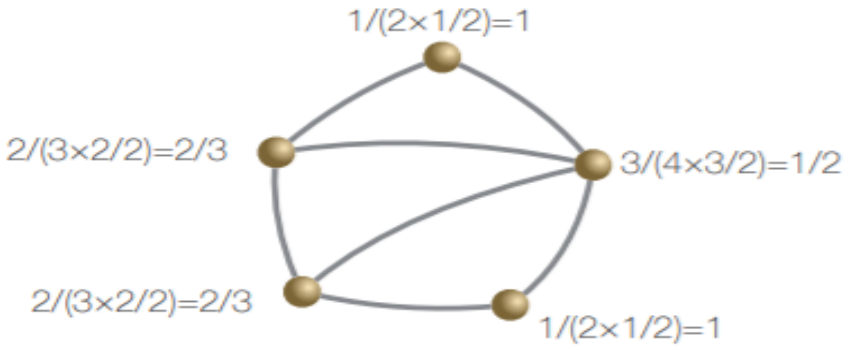



```
MATCH (a { name: "John" })--(b)
WITH a, count(DISTINCT b) AS n
MATCH (a)--()-[r]-()--(a)
RETURN n, count(DISTINCT r) AS m
```

(ANSWER n=4, m=1)

Average Clustering Coefficient

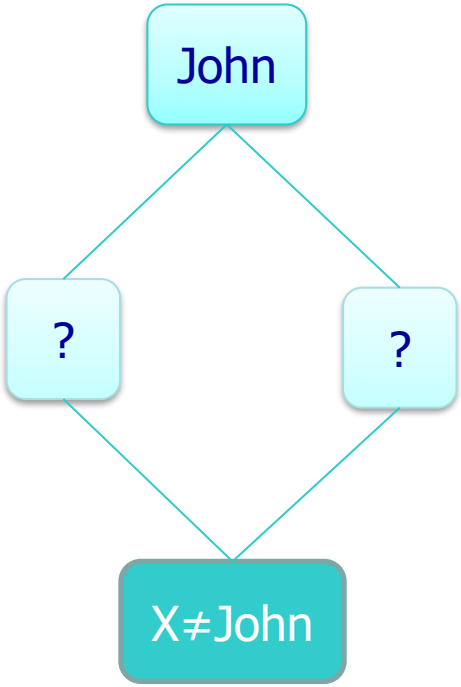
- **Average Clustering Coefficient (CC) of a graph G** is the average of the clustering coefficients of all nodes in G



$$CC = (1+2/3+2/3+1+1/2)/5 = 0.7666$$

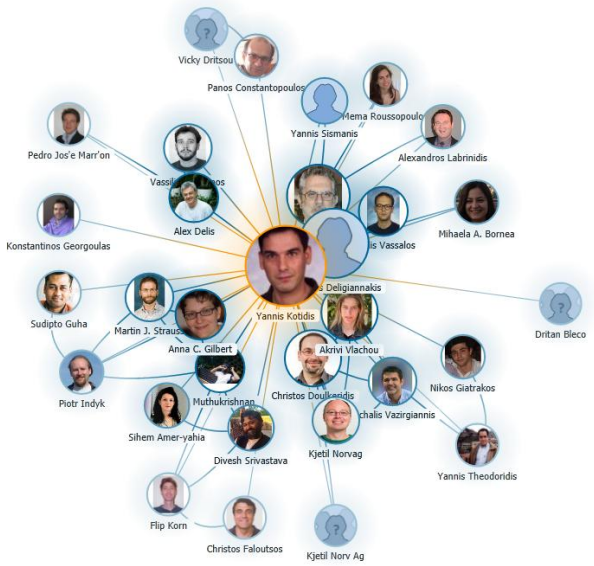
Squares Clustering Coefficient

- Probability that two neighbors of John have a common friend, other than John



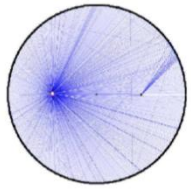
Ego-centric vs socio-centric analysis

- Socio-centric: one “network perspective”
- Ego-centric: focus on individual ego networks

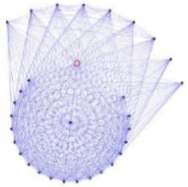


Roadmap:

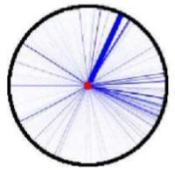
- extract several features from each egonet (#nodes,#edges,EI,weight,largest eigenvalue, etc)
- use those features to seek outliers/suspicious patterns



Near star



Near clique



Dominant link

Krackhardt E/I Ratio

- Given a subgraph (e.g. an egonet)
  - Let **E** denote the number of ***external*** connections
    - E.g. link from a node in the subgraph to another node not in the subgraph
  - Let **I** denote the number of ***internal*** connections

$$\mathbf{EI = (E-I)/(E+I)}$$

- EI varies between -1 (homophily) and +1(heterophily)

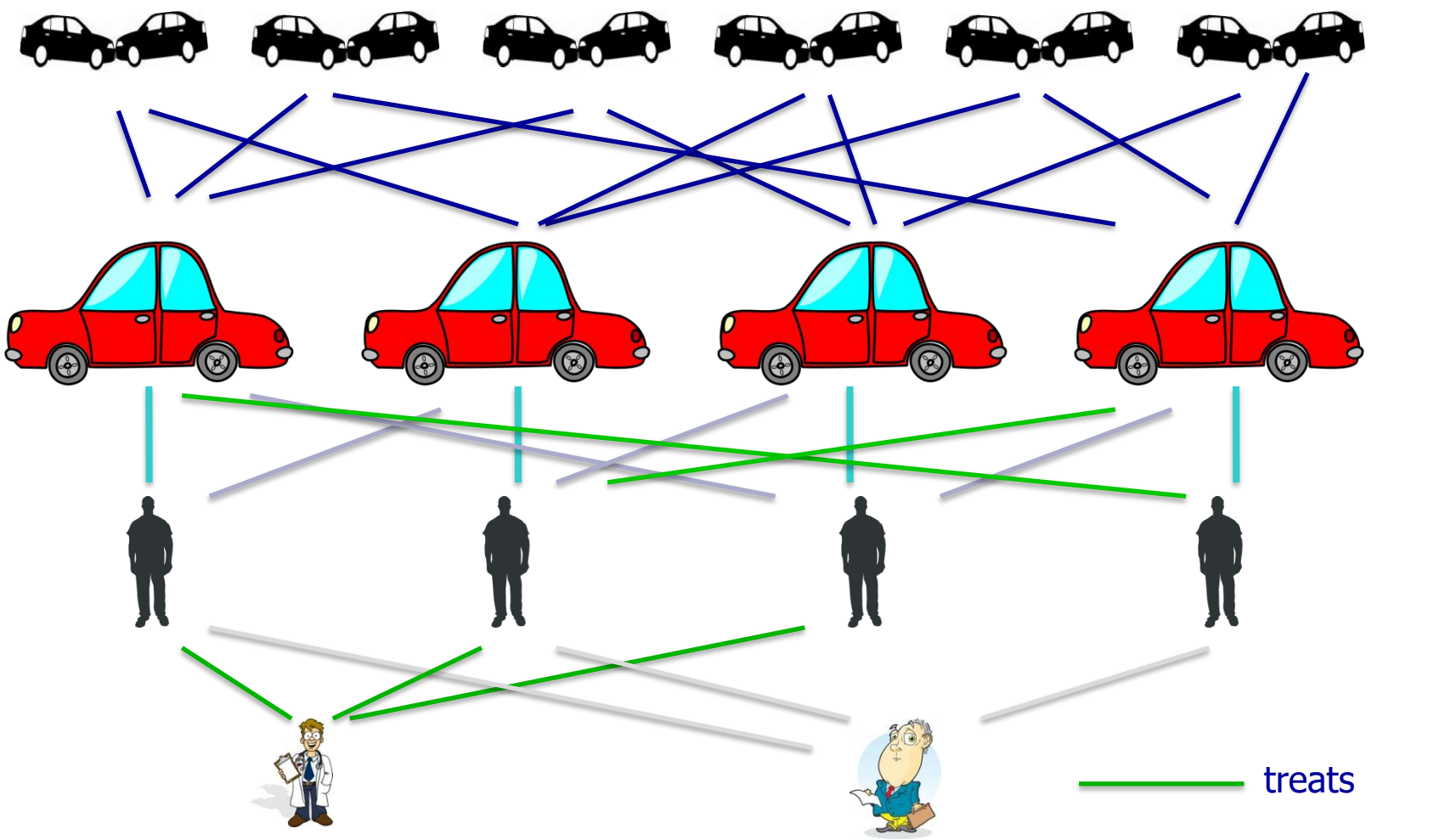
Fraud Detection Example

- Group of ≥ 2 people that create a number of fake identities by combining legitimate information (addresses, phone#, SSN, etc)
- Use these fake identities to open credit card accounts, loads, etc
- These accounts are used normally until a day the ring “busts out”, maxing out all their credit lines and disappearing

Fraud Ring


690-555-4444
Yoda
690-555-2222
Darth Vader
76 Patission Str
Athens, Greece
Han
Solo
33 Leykados Str
Athens, Greece
Luke
Skywalker
690-555-2223

Insurance Fraud

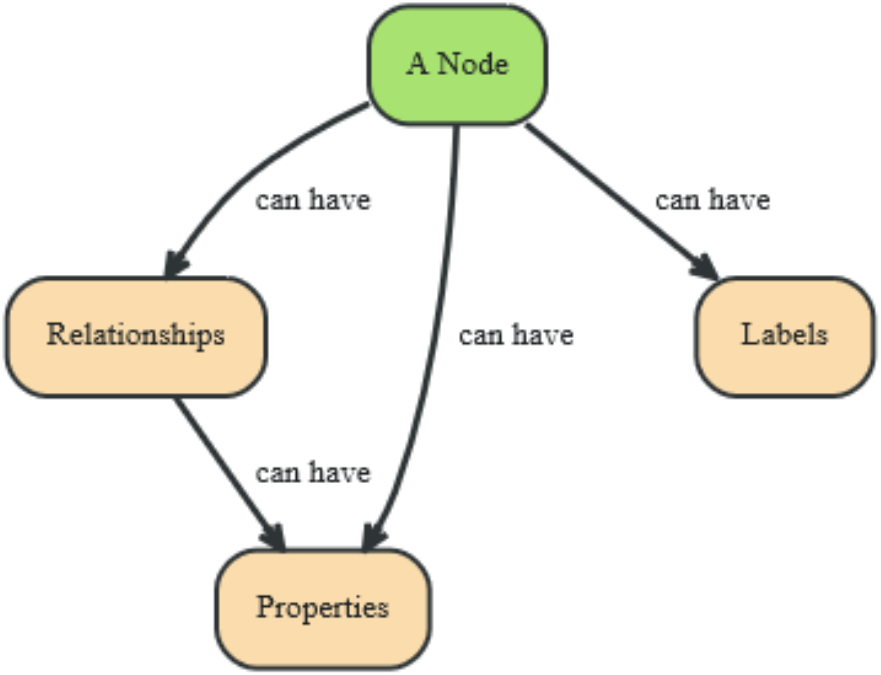


Recap: Modern application needs

- Basic Traversals
  - k-hop traversals
  - Shortest path, longest path
  - Reachability queries
- Pattern matching/motifs
- Centrality Metrics/PageRank
- Similarity?
- Clustering?

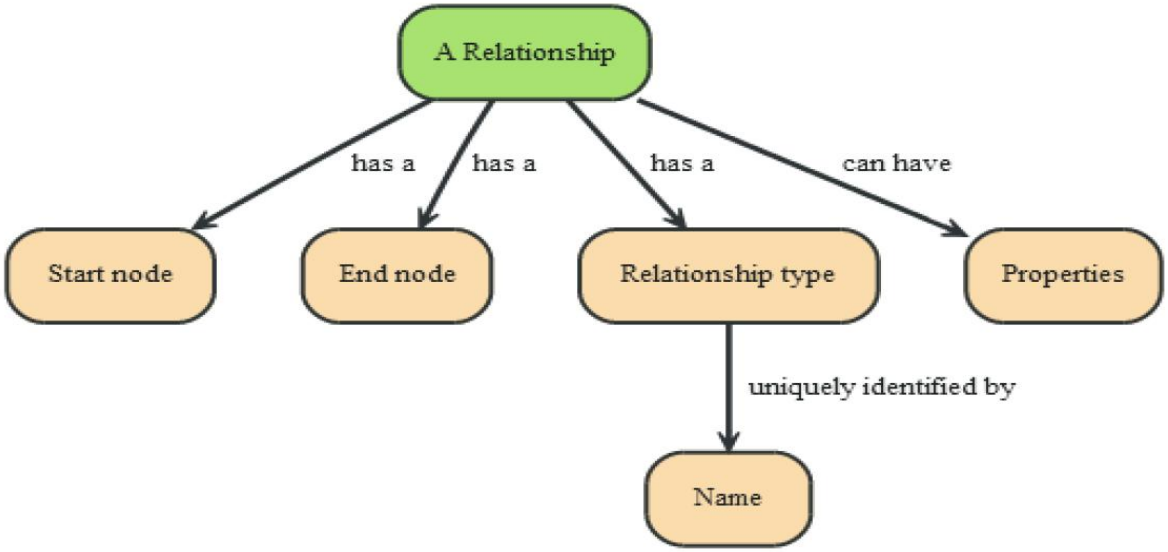
Neo4j: Property Graph Model

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



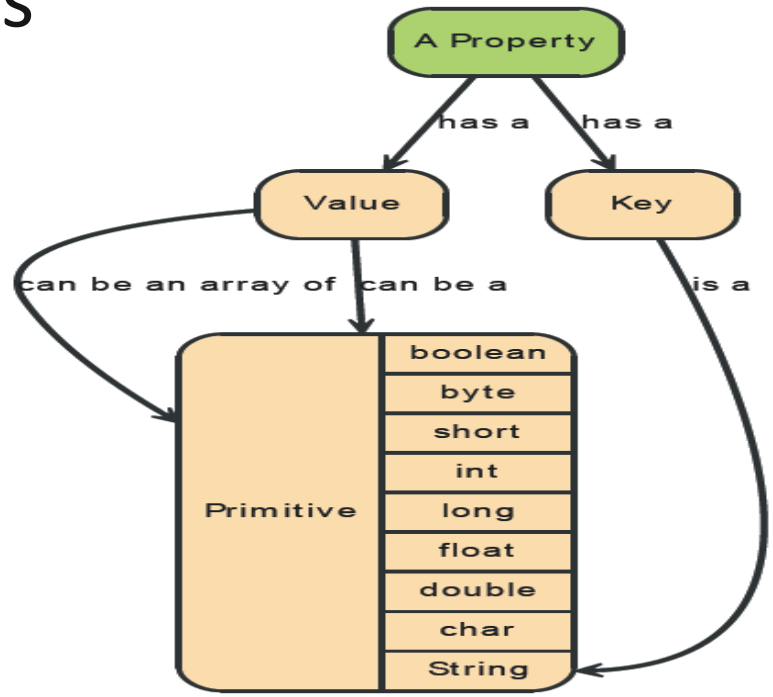
Neo4j Property Graph Model

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



Neo4j Property Graph Model

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



Node Labels (Ετικέτες)

- Named graph constructs used to group nodes into 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 ?

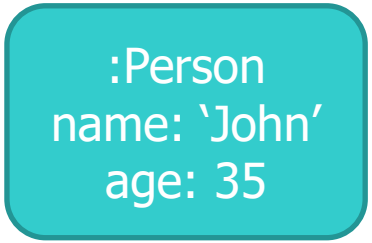
- “John and Sally know each other. They both have read the book, Graph Databases”

Represent entities as nodes with properties and labels

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

Cypher Syntax: CREATE nodes

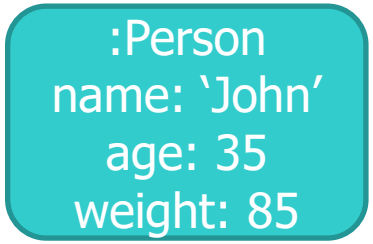
- CREATE (john:Person {name: 'John', age: 35})

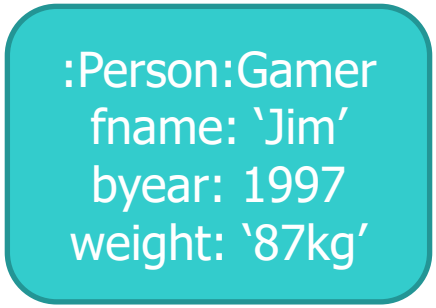


- General syntax CREATE (n:$Label_1$:…:$Label_n$ { $attr_1$:val_1, $attr_2$:val_2, …$attr_k$:val_k})
  - n is a variable that you can use to refer to that node in the same script

Freedom of choice

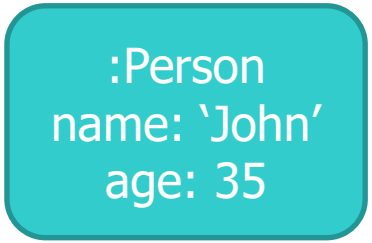
- Unlike relational databases, there is no restriction on the number and type of properties on a node
  - E.g. nodes may have different properties, or same properties of different types
  - Recall Person is just a label. It does not restrict the schema of the corresponding nodes





Existential constraints

- Assert that each Person has a name



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

Unique constraints

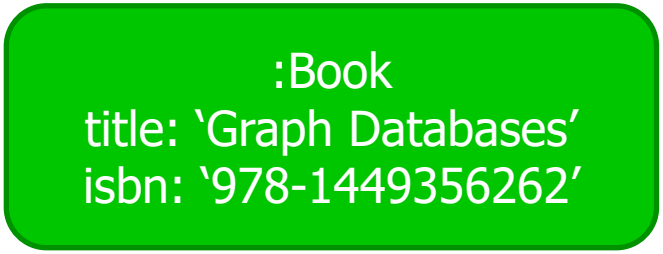
- Assert that no two books in the database can have the same isbn

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

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

Key constraint

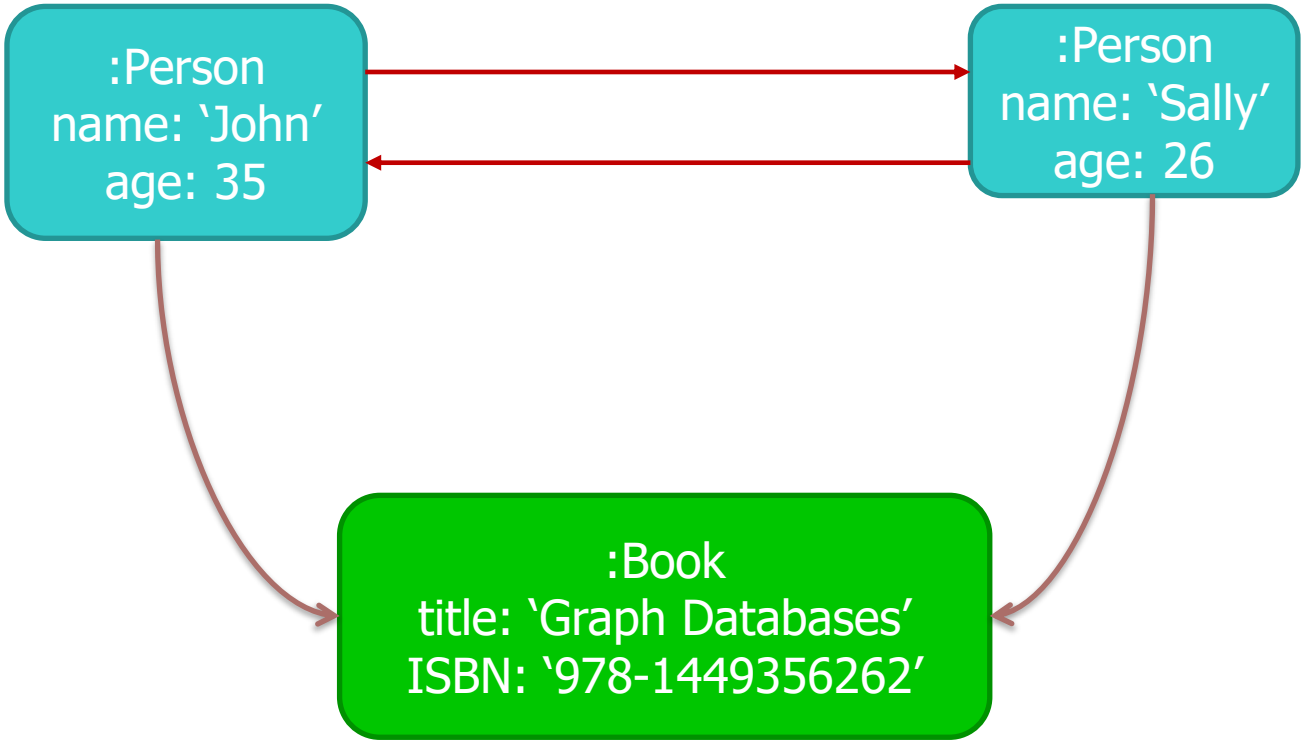
- Each book should have a unique isbn



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

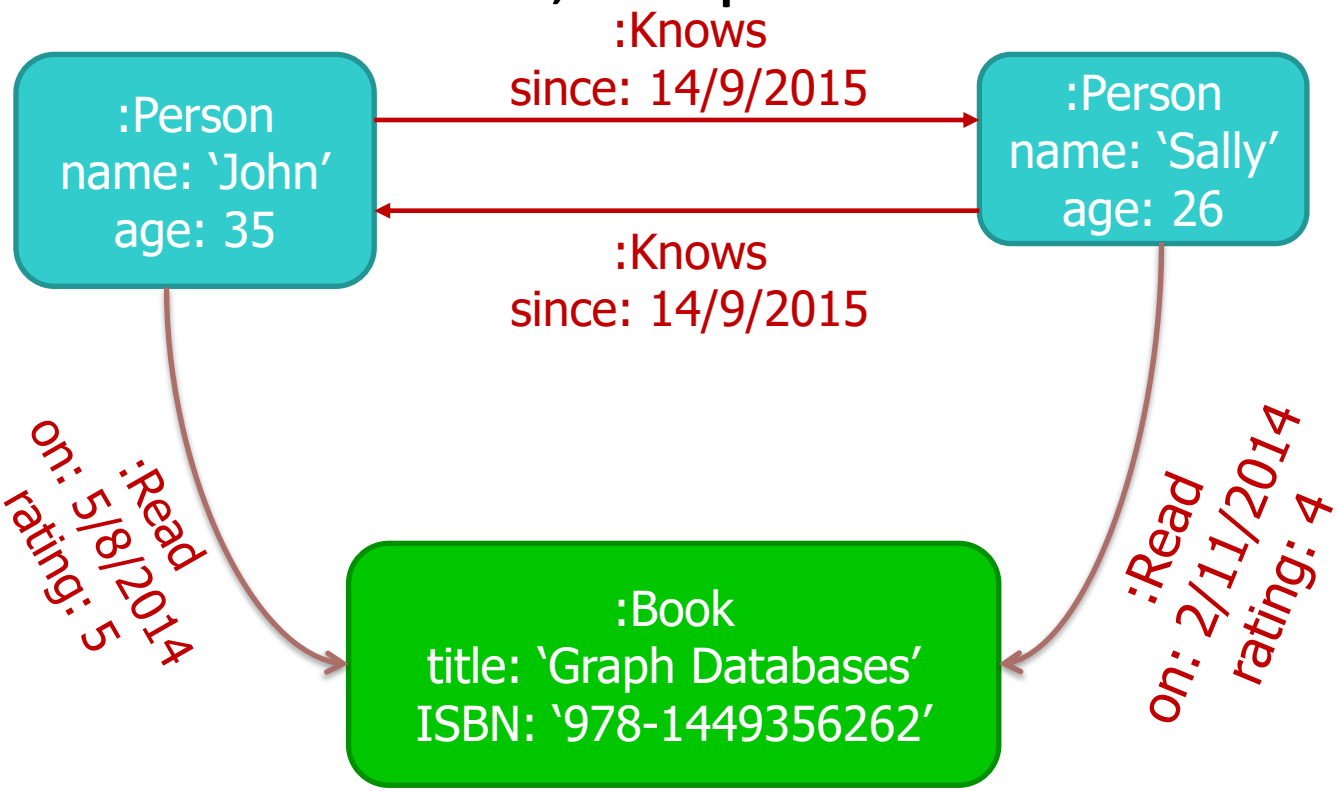
Express relationships as edges between nodes

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



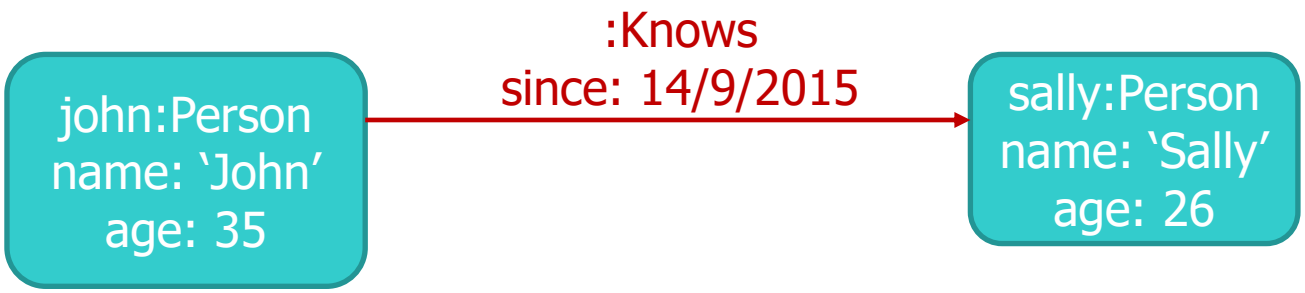
Relationships have types and (optionally) properties

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



Create relationship

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

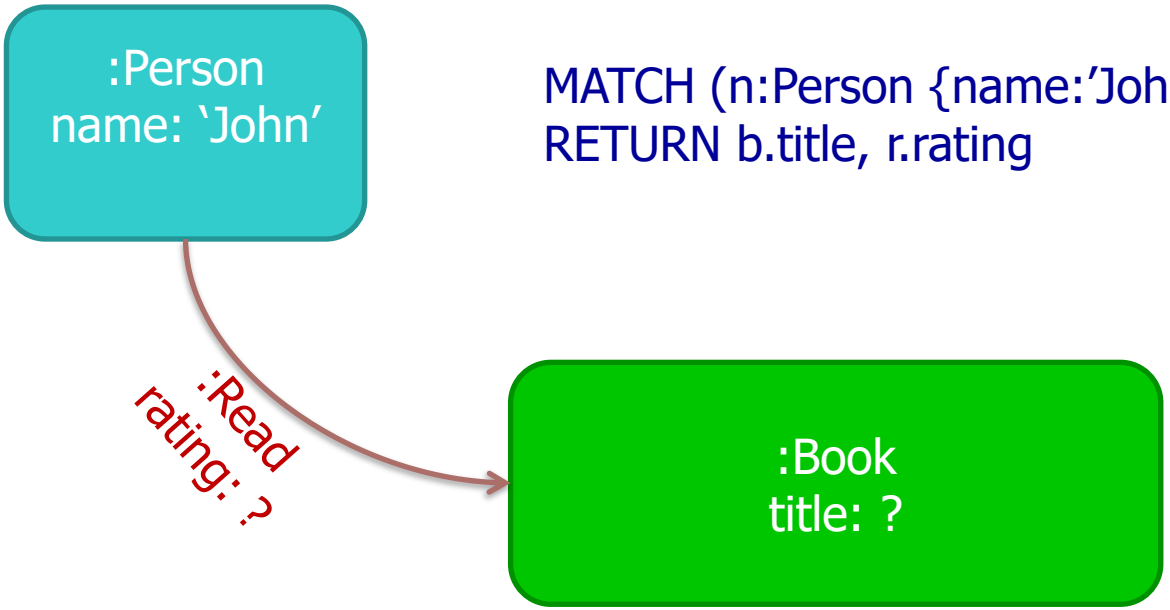


- In this example Knows is a relationship type, since is an attribute for that particular instance, john & sally are variables that refer to previously created nodes

Querying the graph database

- Queries are also graphs!

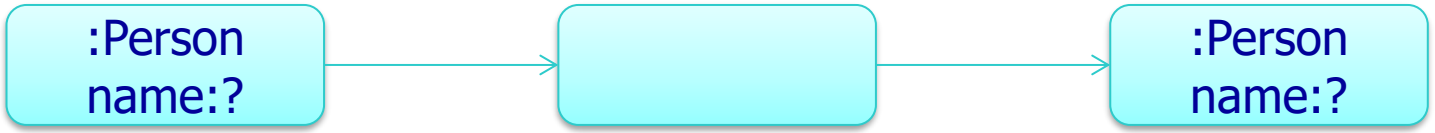
"Find the titles of all books that a person named John has read and report his ratings"




```
MATCH (n:Person {name:'John'})-[r:Read]->(b:Book)
RETURN b.title, r.rating
```

More Graph Patterns

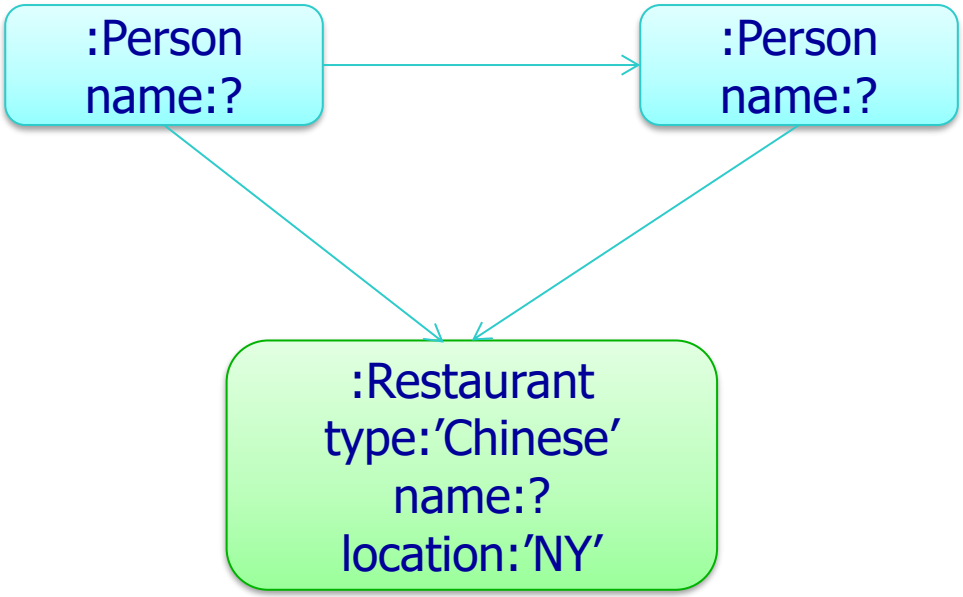
- Friend-of-friend pairs in a social network



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

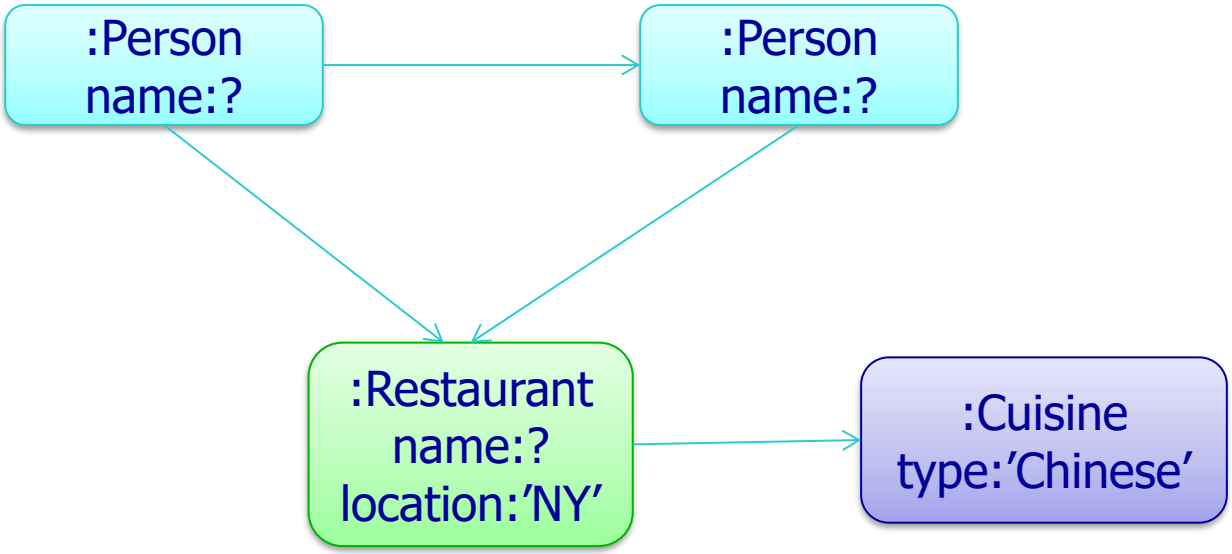
Can you write the following query?

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



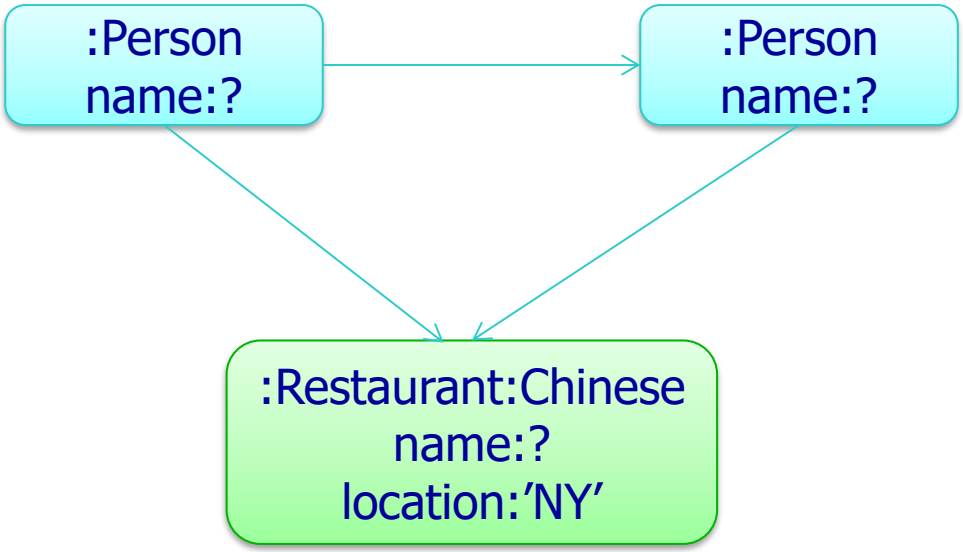
Alternative model

- 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

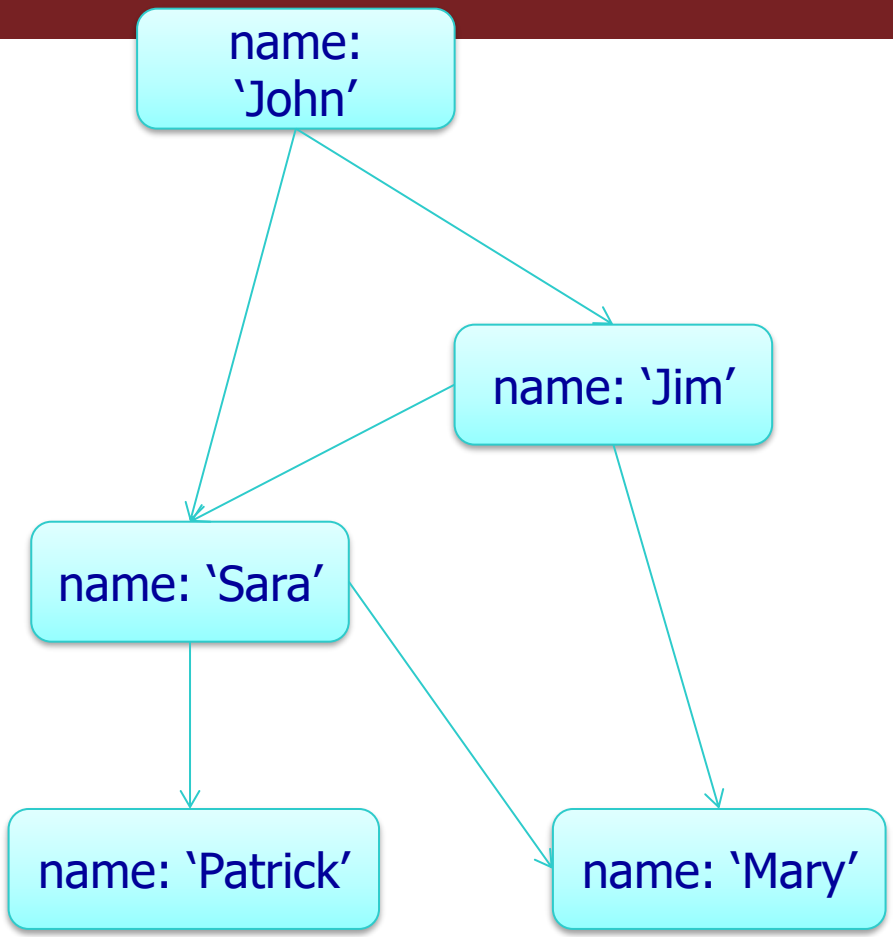


More on Cypher

- 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 patterns found (sorting, aggregation)

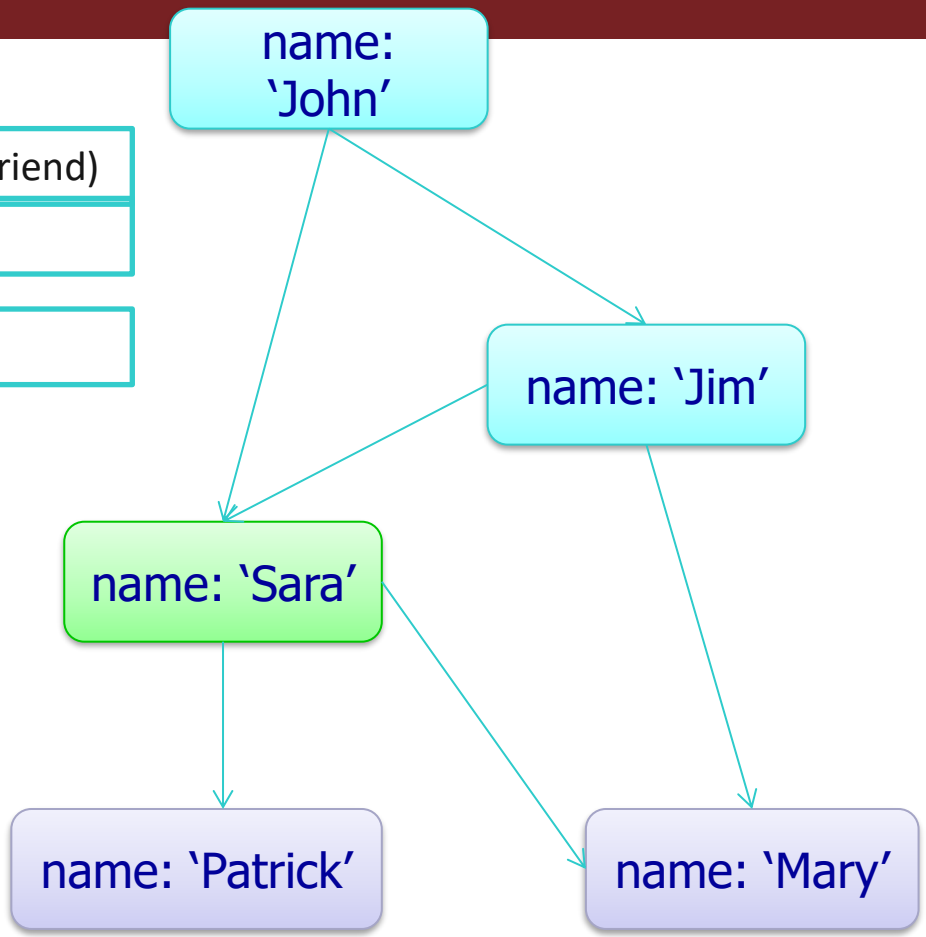
Example: Friend suggestions

- 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

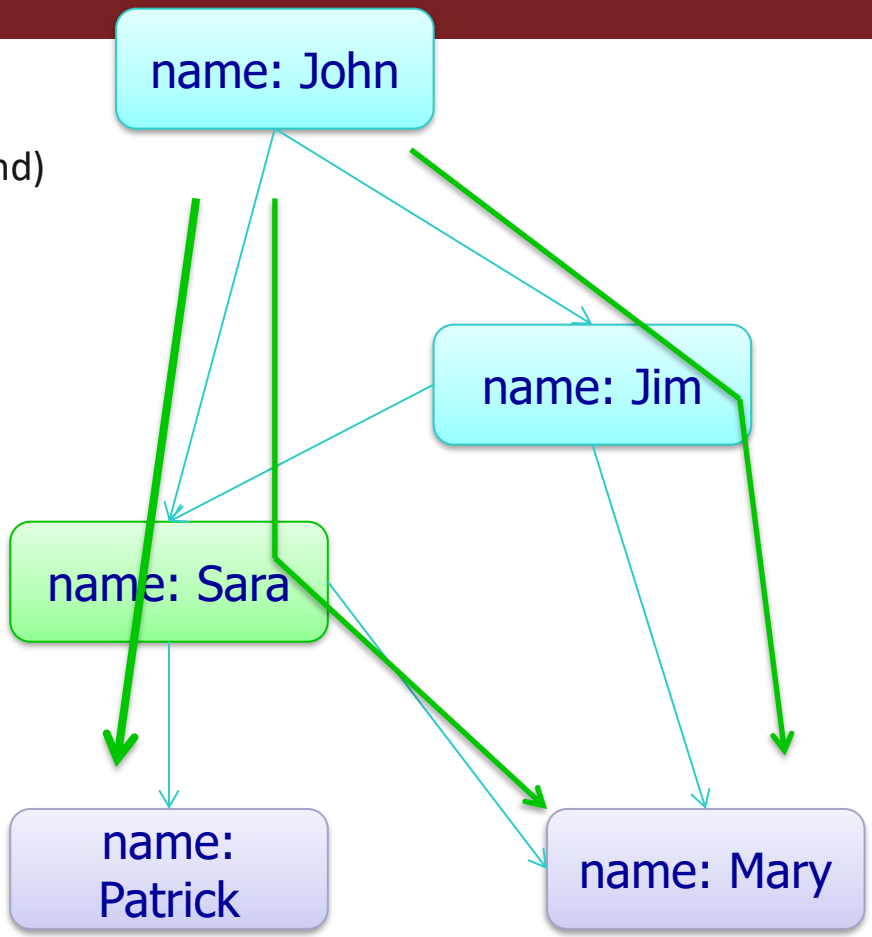
```
MATCH (john {name:'John'})-[:Knows*2..2]-(friend_of_friend)
WHERE NOT (john)-[:Knows]-(friend-of-friend)
RETURN friend_of_friend.name, COUNT(*)
ORDER BY COUNT(*) DESC, friend_of_friend.name
```



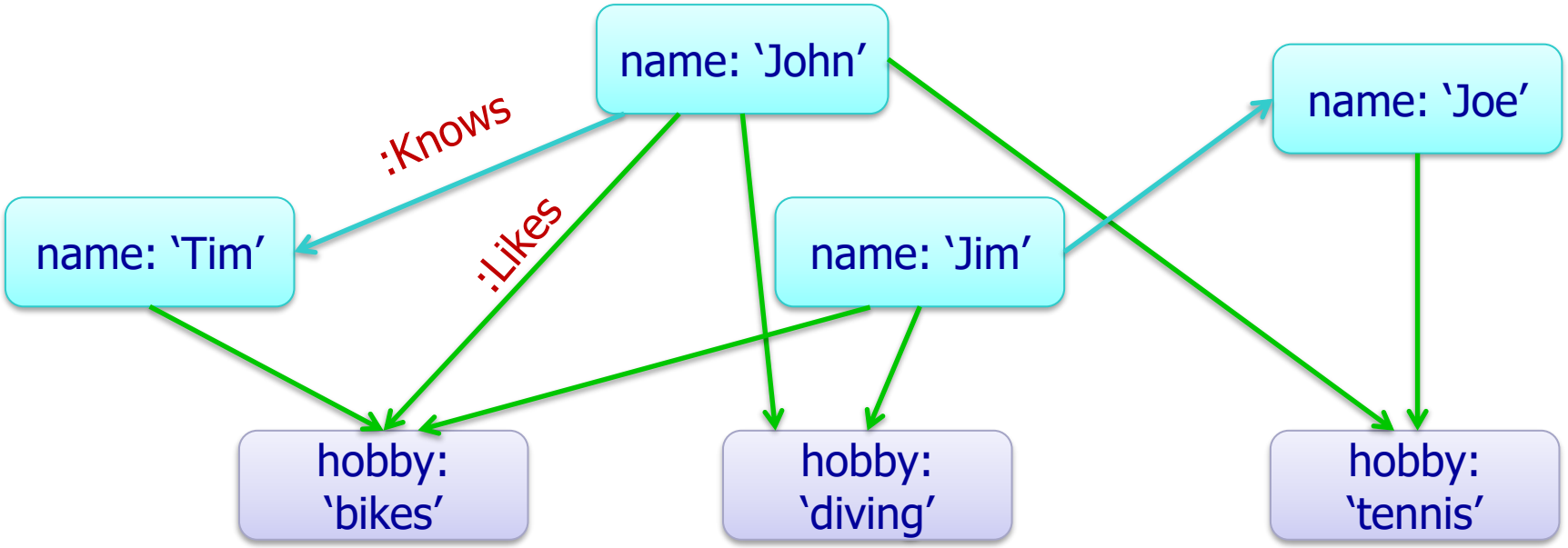
Pattern Matching using Cypher

```
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



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

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.

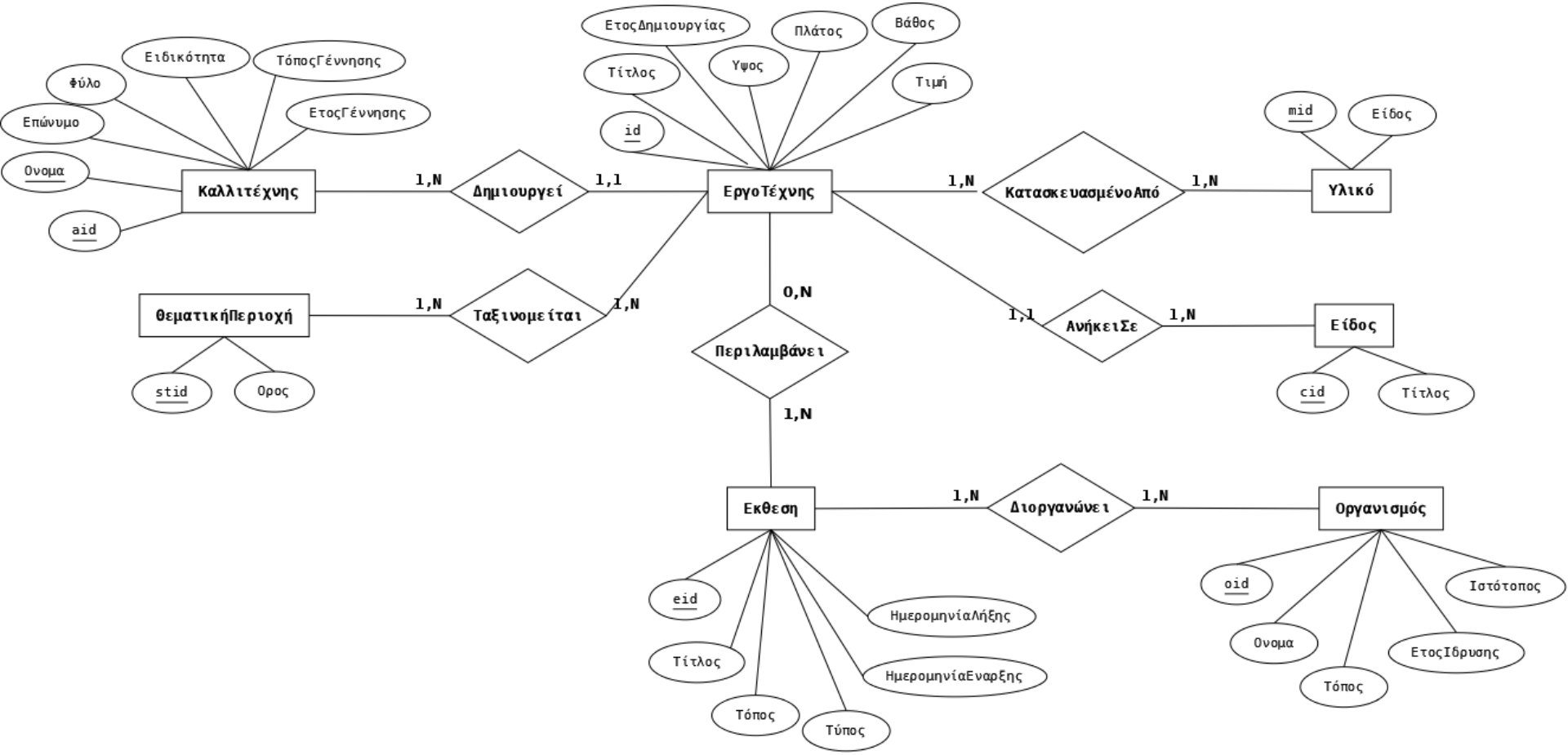
Στη βάση δεδομένων θα πρέπει να καταγράφονται για κάθε **καλλιτέχνη** το ονοματεπώνυμό του, η κύρια ιδιότητά του (ζωγράφος, αγιογράφος, χαράκτης, γλύπτης κ.λπ.), το φύλλο του, καθώς επίσης το έτος και ο τόπος γέννησης.

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

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

- 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

- Κάθε στιγμιότυπο μίας οντότητας μπορεί να αναπαρασταθεί ως ένας κόμβος στη neo4j

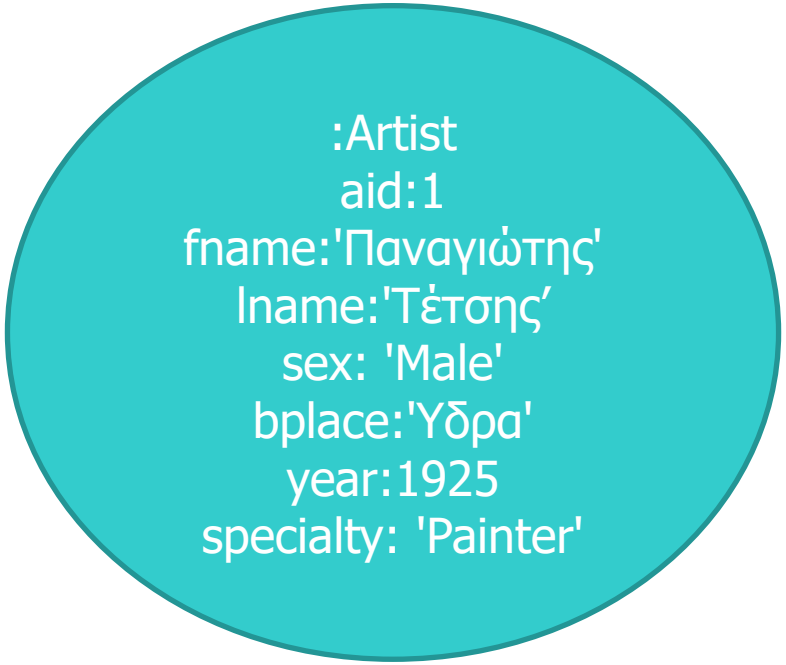
- Προτείνεται η χρήση ετικετών ώστε να διακρίνουμε στιγμιότυπα διαφορετικών οντοτήτων (πχ καλλιτέχνες από έργα τέχνης) στους κόμβους του γράφου

Ετικέτες

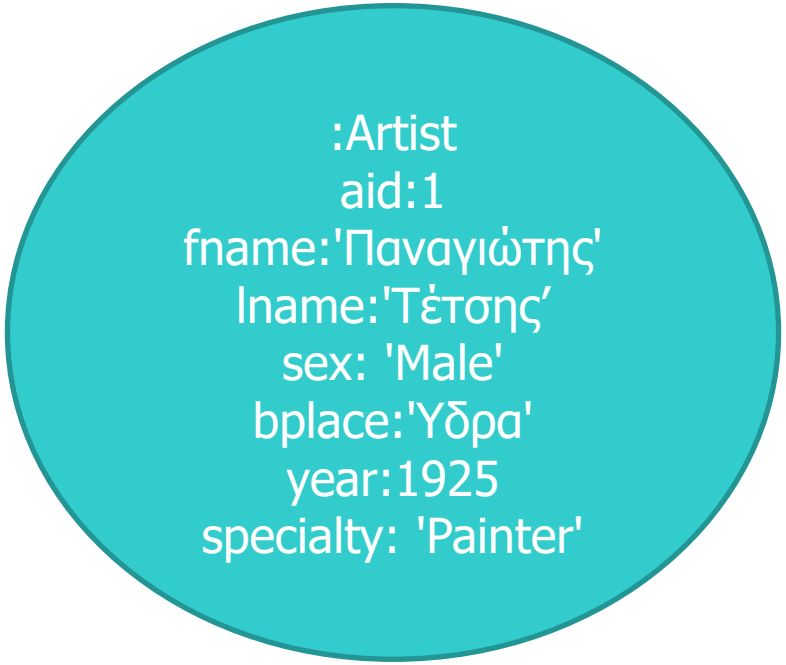
- Χρησιμοποιούνται για το διαχωρισμό των κόμβων της βάσης σε σύνολα

- 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:'Υδρα', year:1925,specialty: 'Painter'})

Περιορισμός κλειδιού

- 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?

- 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!

Γνώρισμα ή ετικέτα?

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

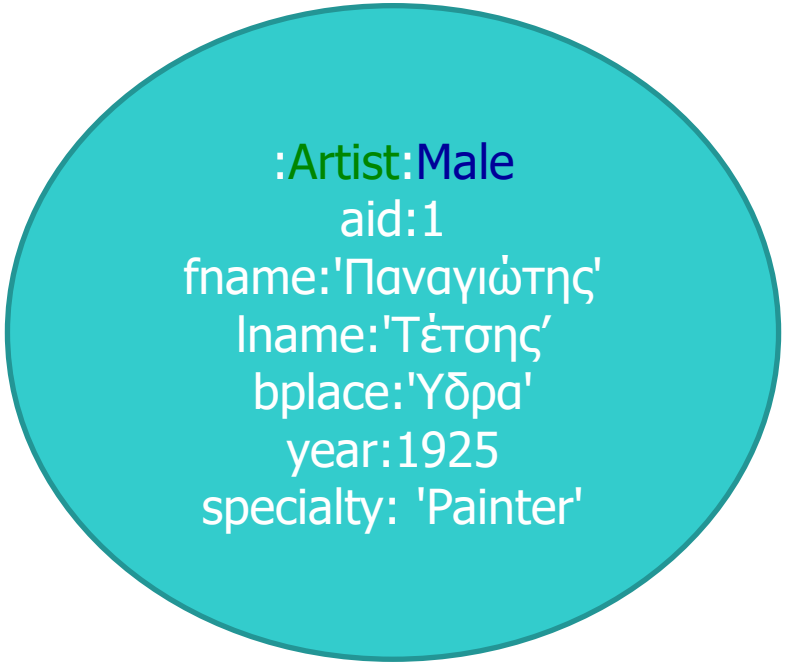
Γνώρισμα ή ετικέτα?

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

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

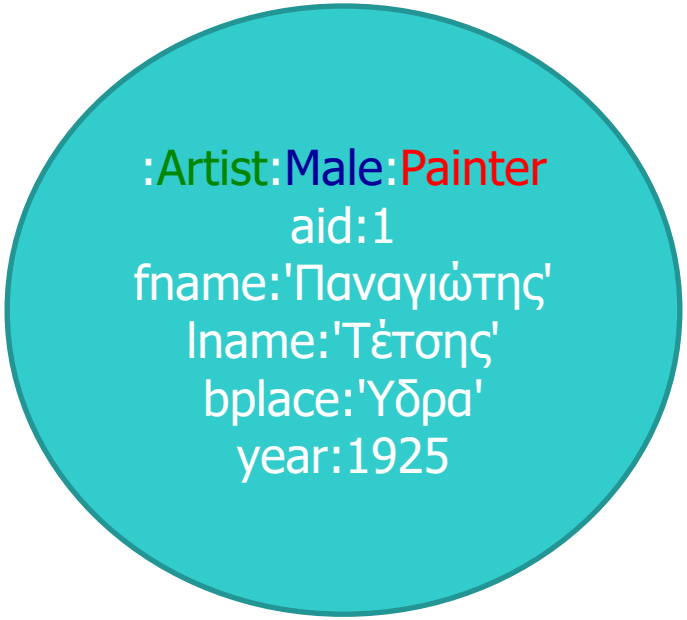
- Φυσικά θα πρέπει να αξιολογήσουμε αν μας ενδιαφέρουν τέτοια ερωτήματα

Ποιο άλλο γνώρισμα θα μπορούσε να αναβαθμιστεί ως ετικέτα?



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

Καλλιτέχνες ως «κόμβοι/nodes» - λήψη 2



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

Οντότητα «Artwork»

- 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

- 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

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

Παρατήρηση 2

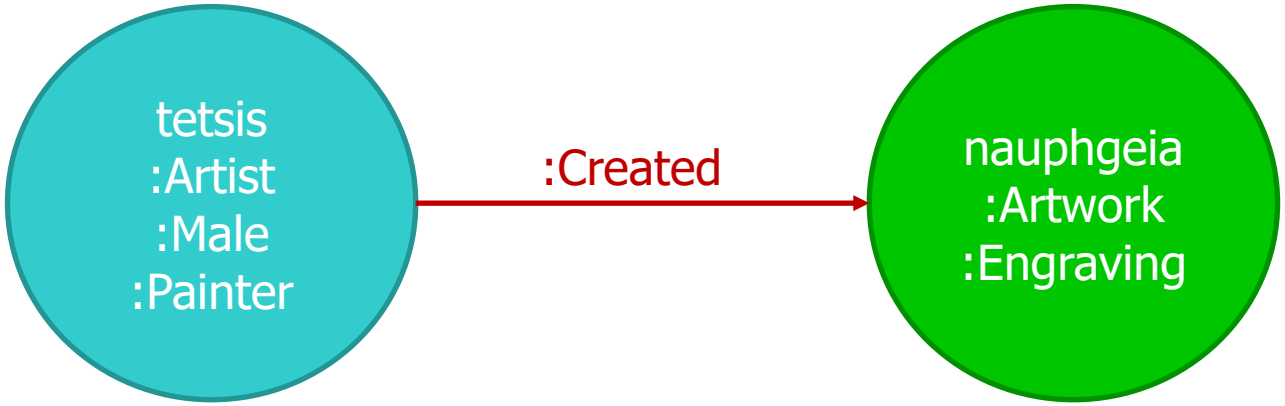
```
CREATE TABLE artwork
    ( id INT PRIMARY KEY,  title VARCHAR(200) NOT NULL,  cyear  INT,
      height  INT, width  INT, depth  INT, price DECIMAL (8,2) CHECK(price >=0),
      cid  INT, aid INT,
      CONSTRAINT fk_cid FOREIGN KEY (cid) REFERENCES categories(cid),
      CONSTRAINT fk_aid  FOREIGN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

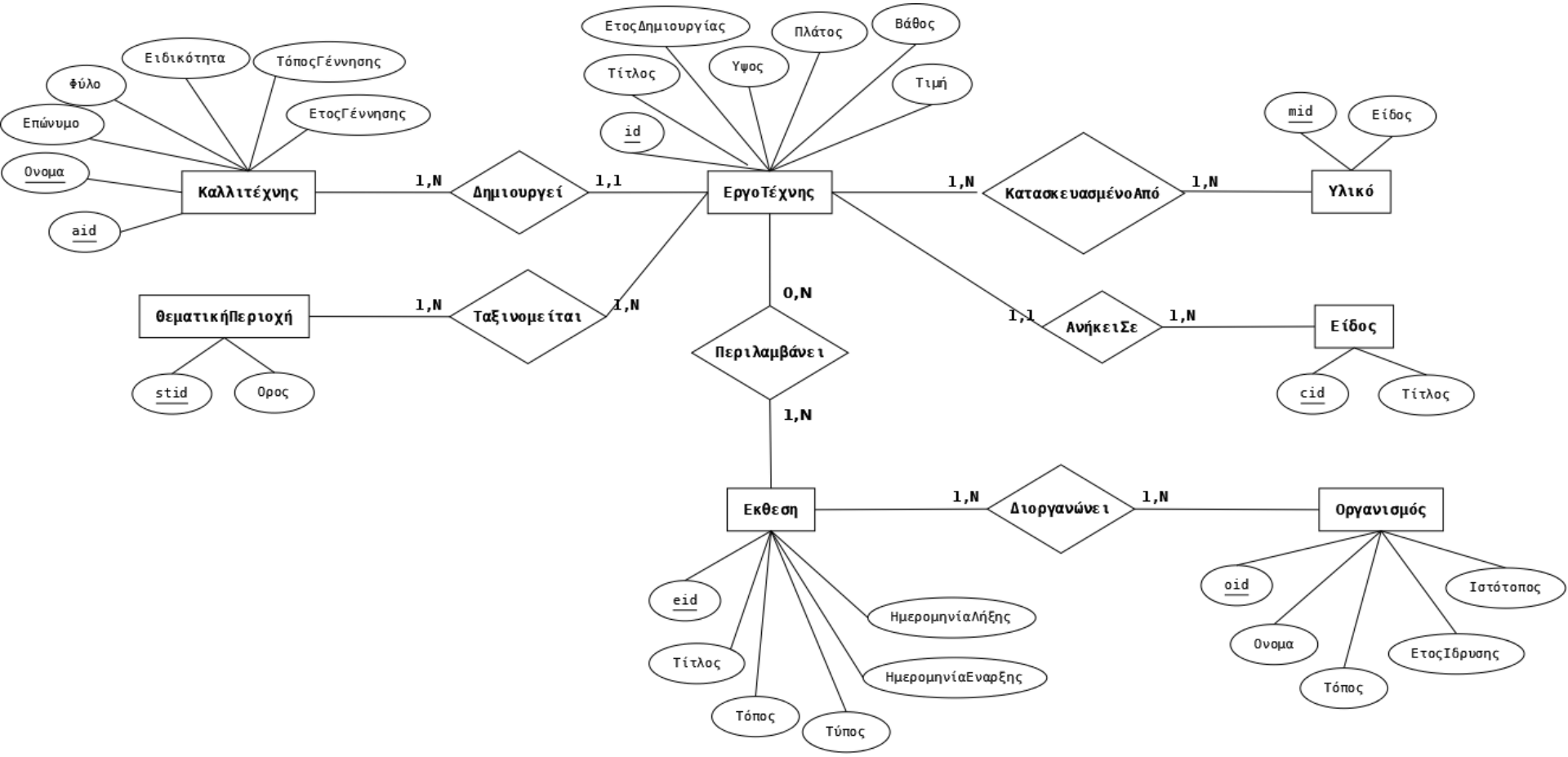
Αν το κάναμε όπως σε μία σχεσιακή βάση…

Όμως σε ένα γράφο η αναφορά μπορεί να ορισθεί άμεσα!

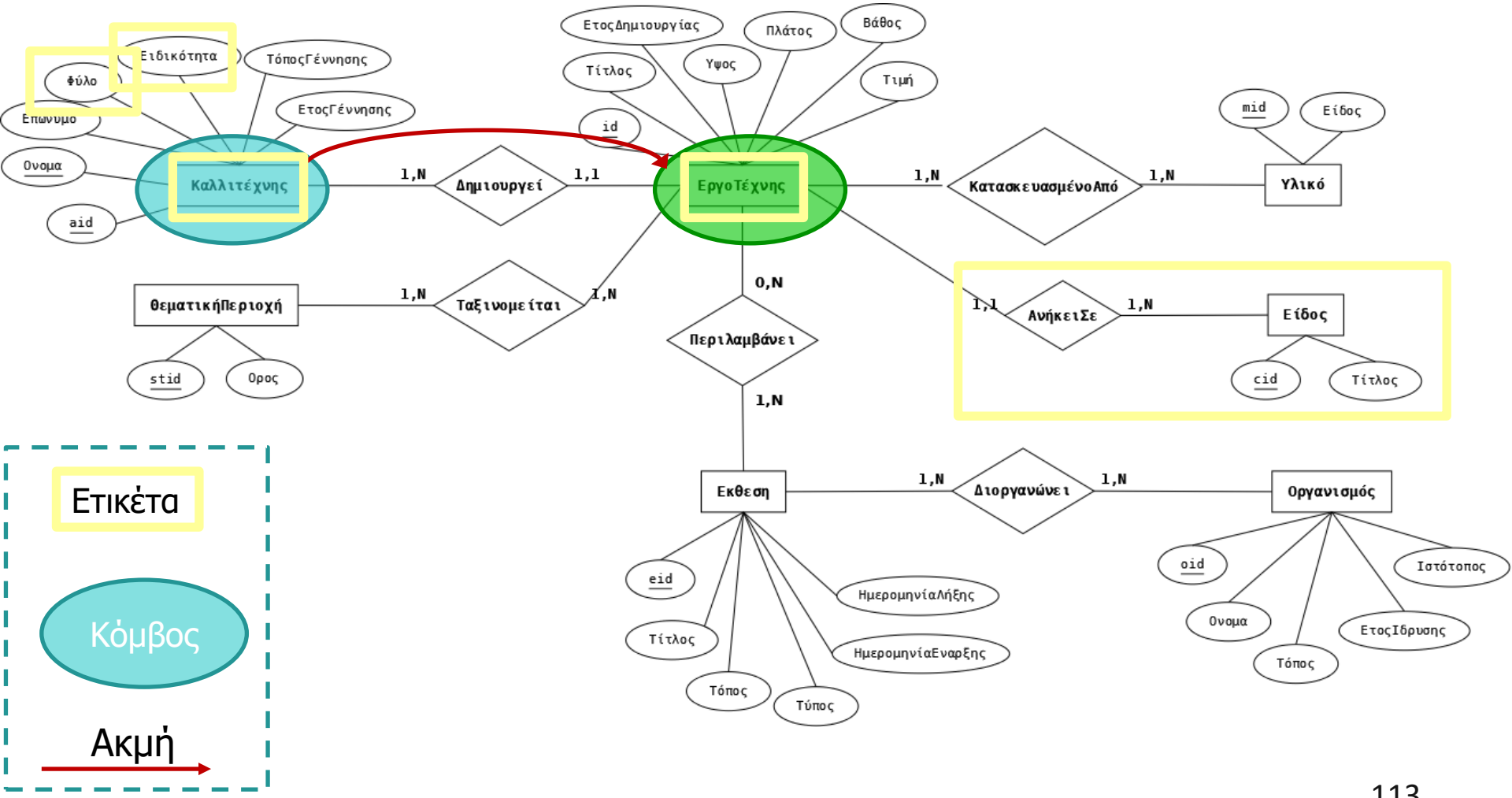
- create (tetsis)-[:Created]->(nauphgeia)



Διάγραμμα ER



Property Graph Model



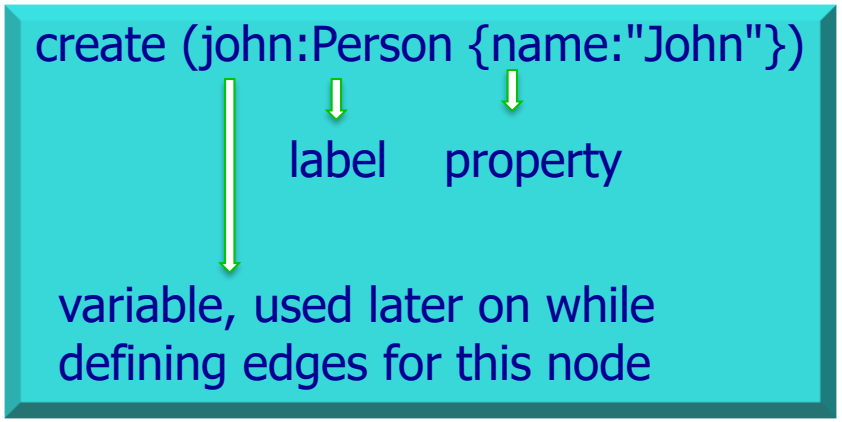
Neo4j Examples

Yannis Kotidis

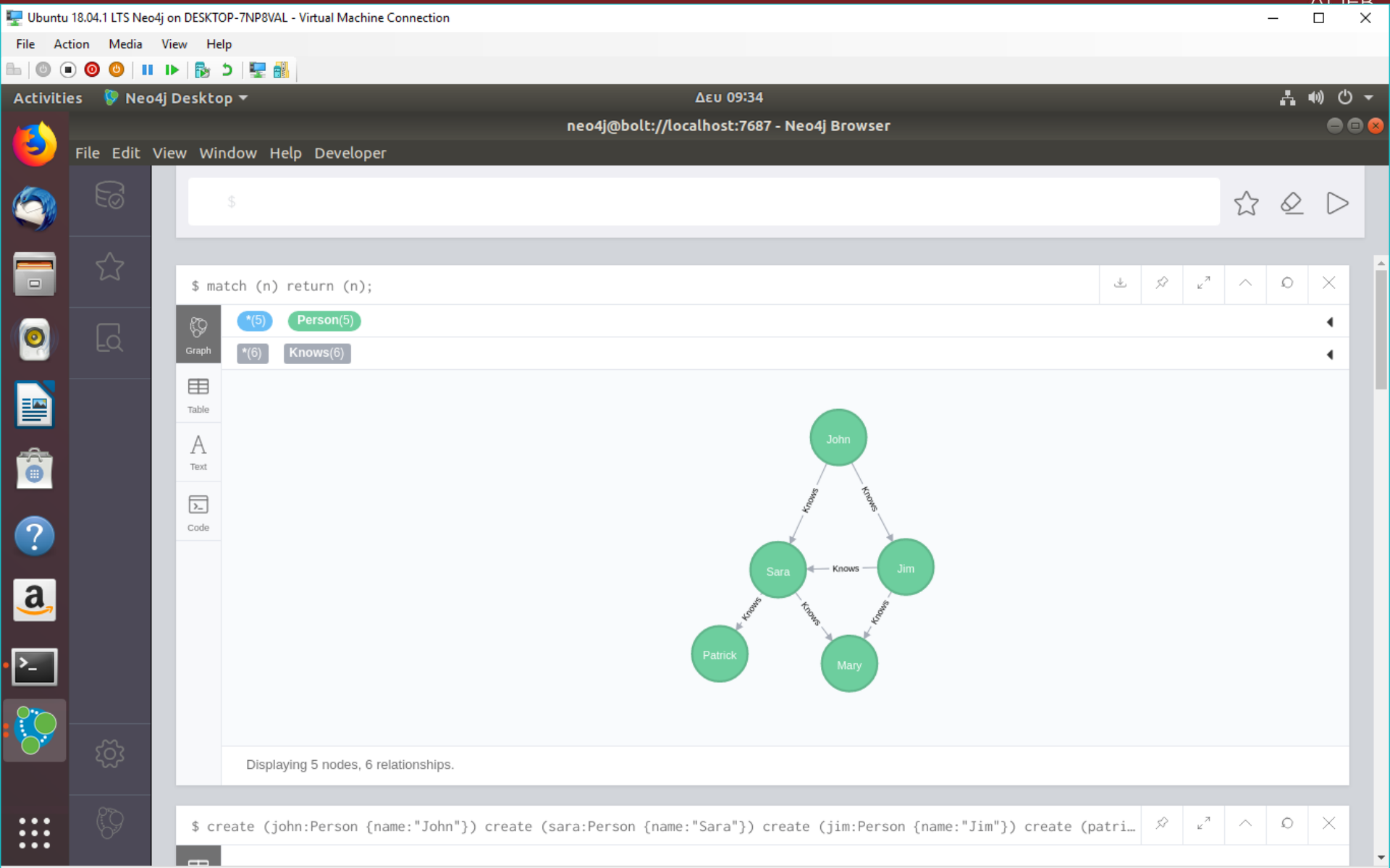
http://pages.cs.aueb.gr/~kotidis/

Create Social Graph script

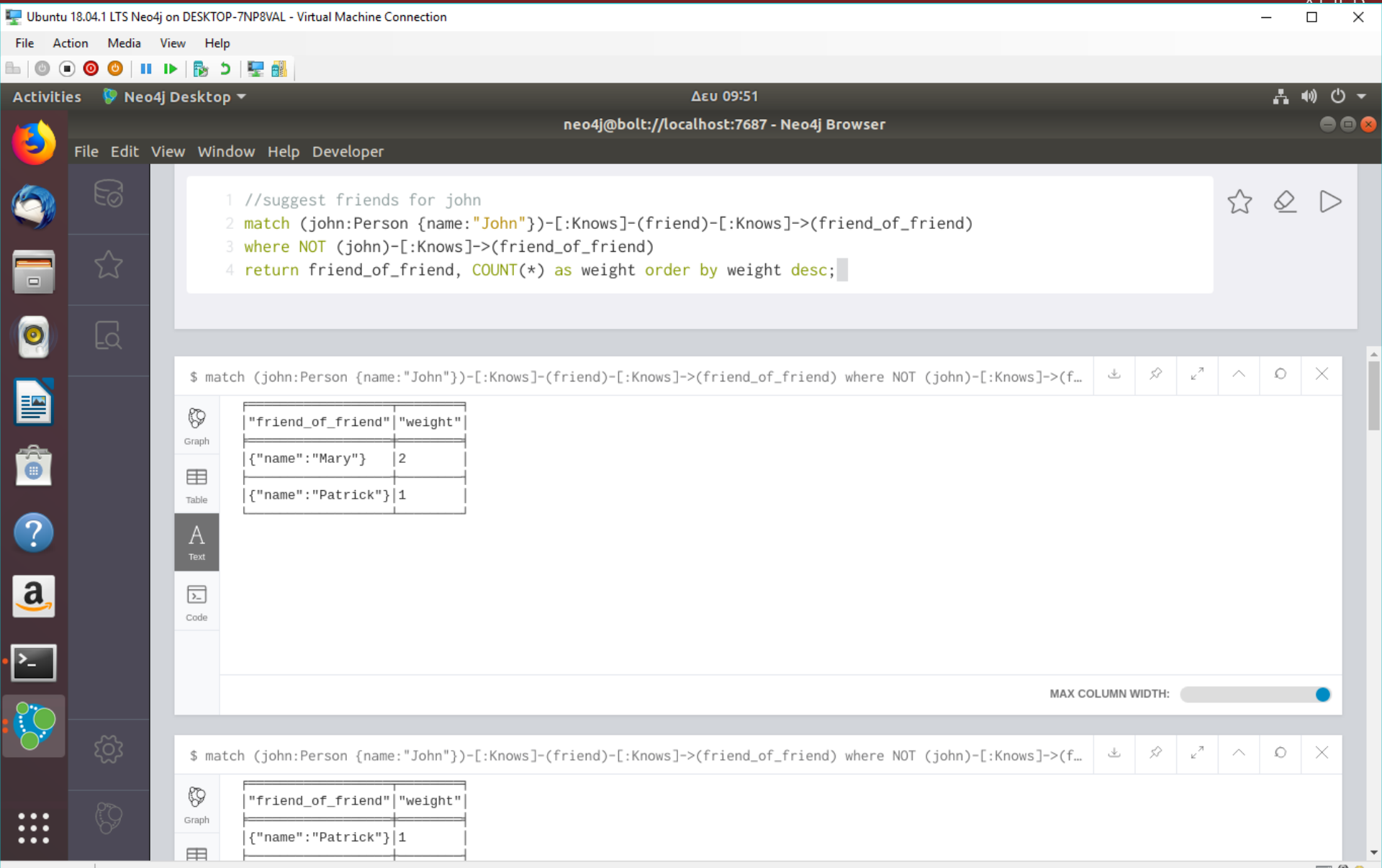
```
//cleanup
//DETACH removes all relationships of a node, DELETE removes the node
MATCH (n) DETACH DELETE n;

//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

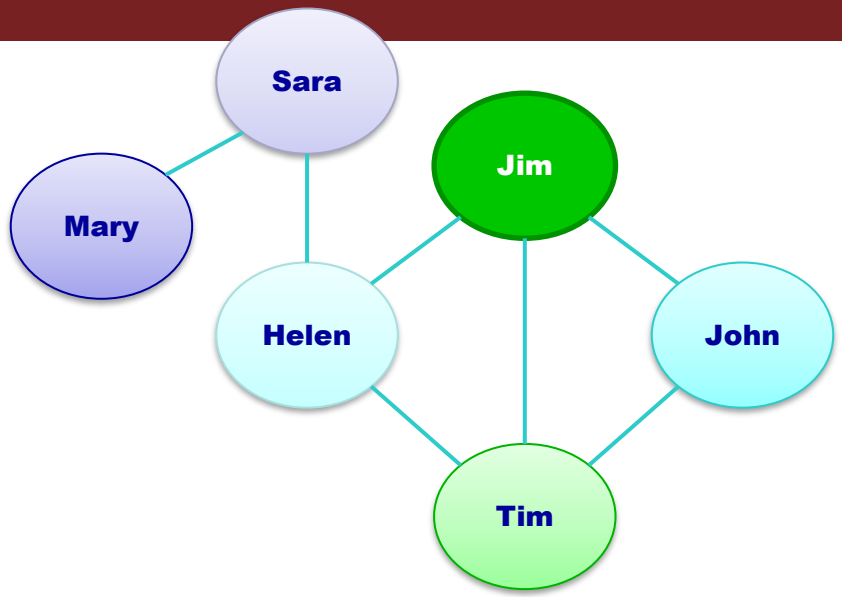


Friend Suggestion

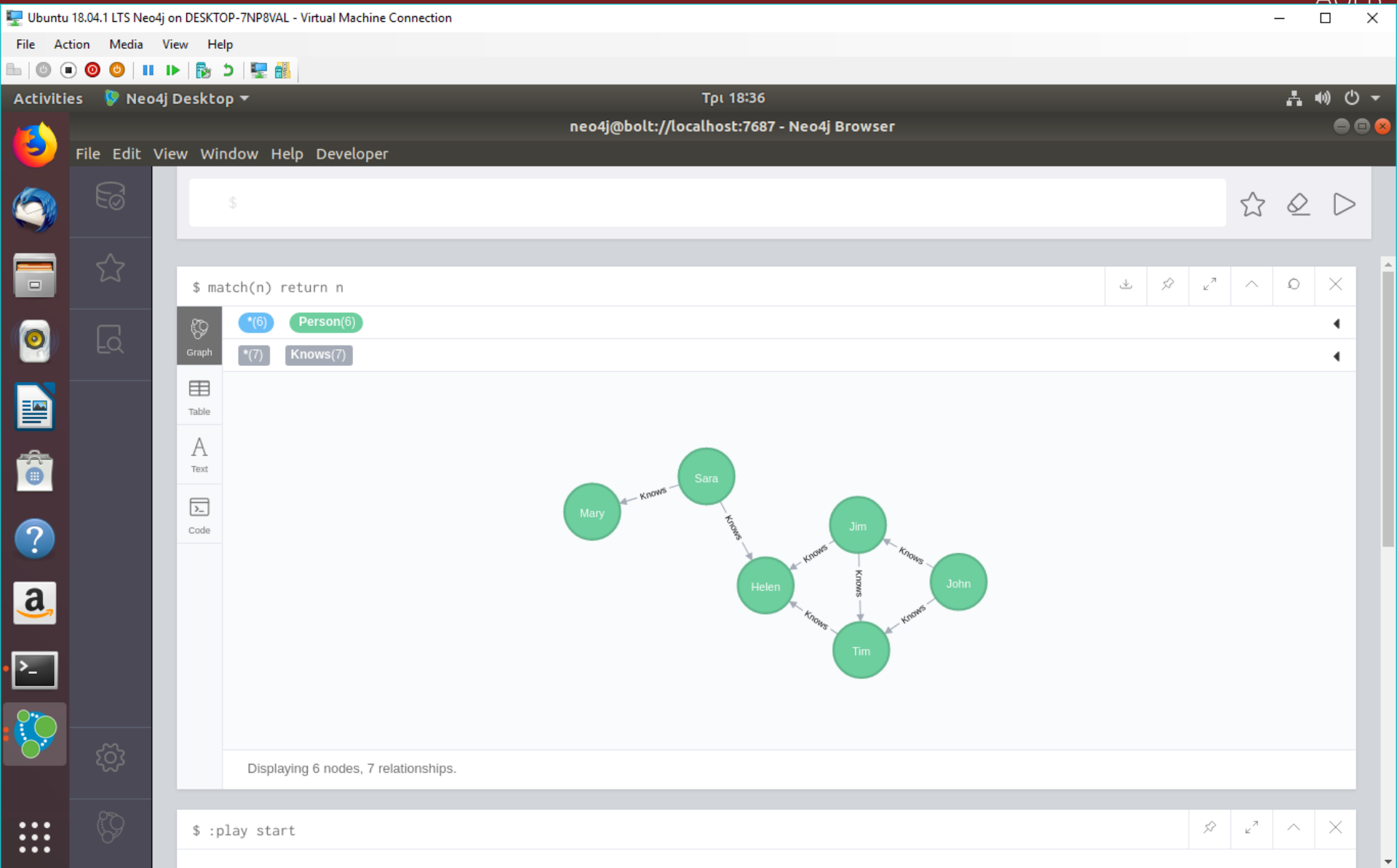


Social Graph 2

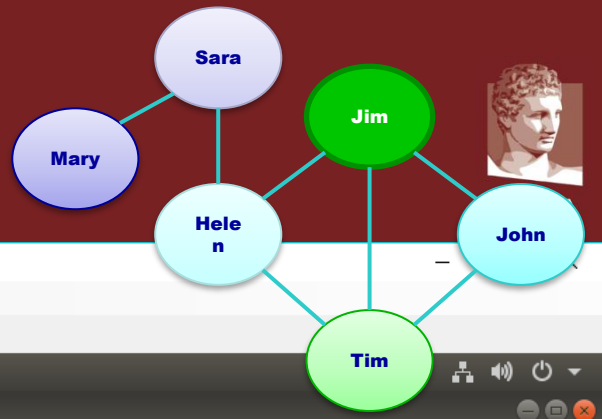
```
//create social graph
merge (mary:Person {name:"Mary"})
merge (sara:Person {name:"Sara"})
merge (jim:Person {name:"Jim"})
merge (helen:Person {name:"Helen"})
merge (tim:Person {name:"Tim"})
merge (john:Person {name:"John"})
merge (john)-[:Knows]->(jim)
merge (john)-[:Knows]->(tim)
merge (jim)-[:Knows]->(tim)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



Resulting Graph



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

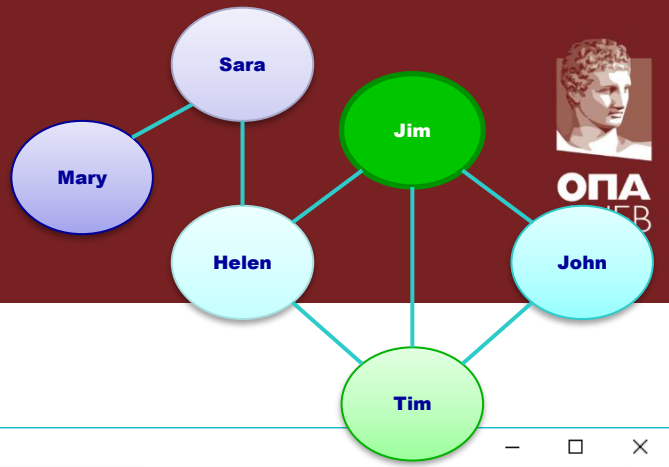
"SOURCE"	"TARGET"	"ShortestPath"
{"name":"John"}	{"name":"Mary"}	[{"name":"John"},{},{"name":"Tim"},{"name":"Tim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"},{"name":"Sara"},{},{"name":"Mary"}]
{"name":"John"}	{"name":"Mary"}	[{"name":"John"},{},{"name":"Jim"},{"name":"Jim"},{},{"name":"Helen"},{"name":"Helen"},{},{"name":"Sara"},{"name":"Sara"},{},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

```
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

File Action Media View Help

Activities Neo4j Desktop Τρι 09:53

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

File Edit View Window Help Developer

```
1 CALL algo.closeness.stream('Person', 'Knows')
2 YIELD nodeId, centrality
3
4 RETURN algo.getNodeById(nodeId).name AS node, centrality
5 ORDER BY centrality DESC
6 LIMIT 20;
```

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

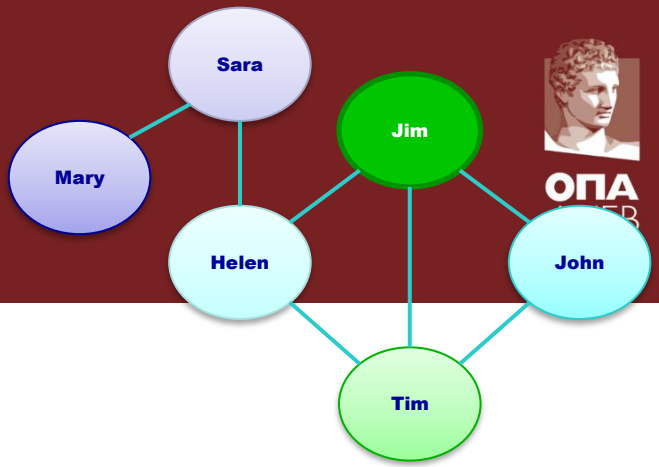
Table Text Code

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

```
1 CALL algo.betweenness.stream('Person','Knows',{direction:'Both'})
2 YIELD nodeId, centrality
3
4 RETURN algo.getNodeById(nodeId).name  AS name, centrality
5 ORDER BY centrality DESC;
```

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

Table Text Code

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