Graph Databases

bbilalli@essi.upc.edu

Knowledge Objectives

1. Describe what a graph database is
2. Explain the basics of the graph data model
3. Enumerate the best use cases for graph databases
4. Name two pros and cons of graph databases in front of relational databases
5. Name two pros and cons of graph databases in front of other NOSQL options

Understanding Objectives

1. Simulate traversing processing in a relational database and compare it with a graph database

Application Objectives

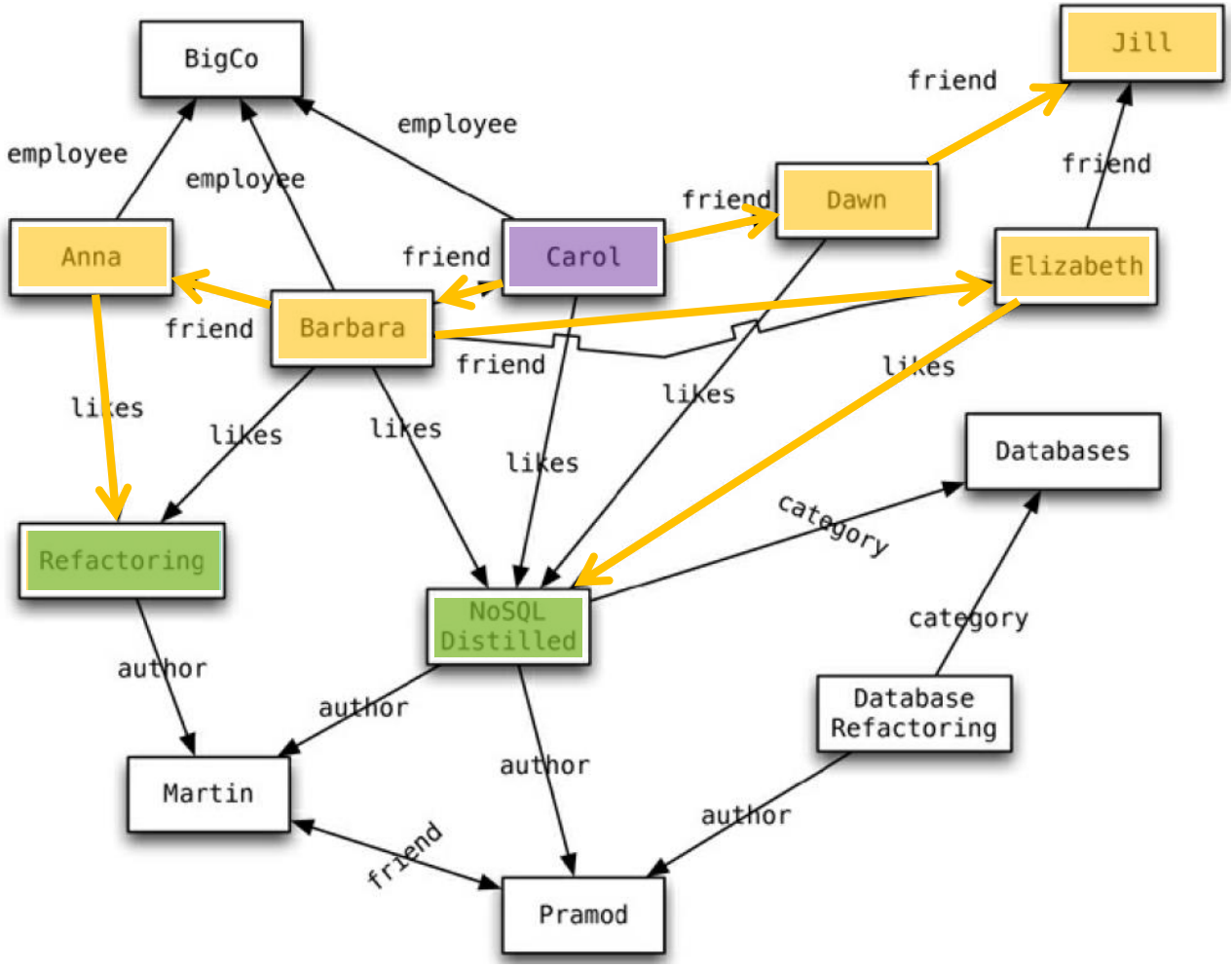
- Model simple graph databases following the property graph data model
- Implement graphs in Neo4J and use Cypher for traversing them

Graph Databases - motivation

- NoSQL DBs were inspired by the need to run on clusters -> this led to aggregate-oriented data models of **large records** with **simple connections**.
- Graph DBs are motivated by a different frustration with relational DBs and thus have an opposite model – **small records** with **complex interconnections**.
  - They typically run in a single server
  - They typically guarantee ACID properties

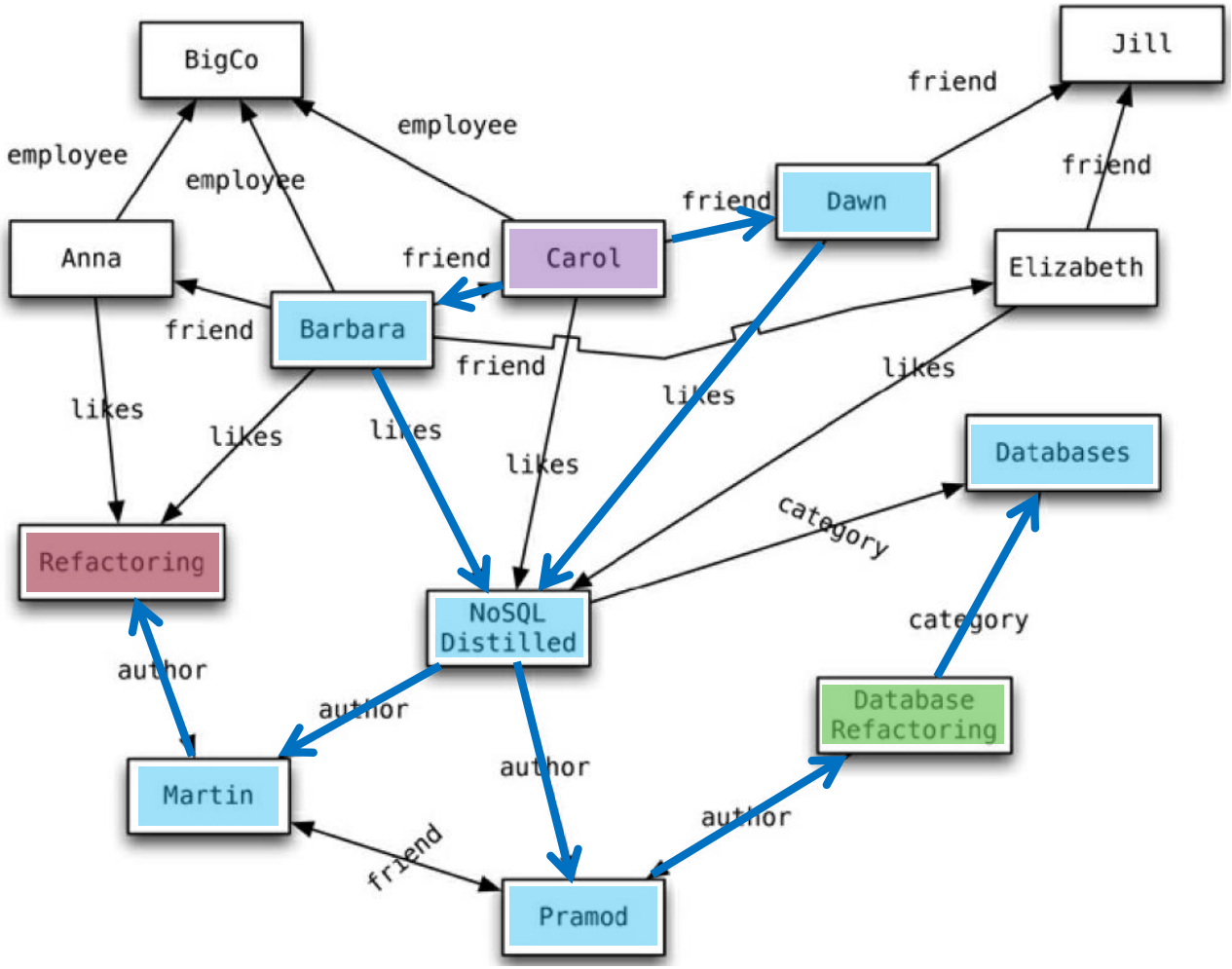
(Very different from the previous NoSQL aggregate-oriented databases. The only thing in common is the schemaless nature)

Example 1



- Find which books friends of my friends like.

Example 2



- Find the books of DB category that are written by someone whom a friend of mine likes.

Graph Databases in a Nutshell

- Occurrence-oriented
  - May contain millions of instances
    - Big Data!
  - It is a form of schemaless databases
    - There is no explicit database schema
    - Data (and its relationships) may quickly vary
  - Objects and relationships as first-class citizens
    - *An object o relates (through a relationship r) to another object o'*
    - Both objects and relationships may contain properties (in some GDBs)
  - Built on top of the graph theory
    - Euler (18th century)
    - More natural and intuitive than the relational model

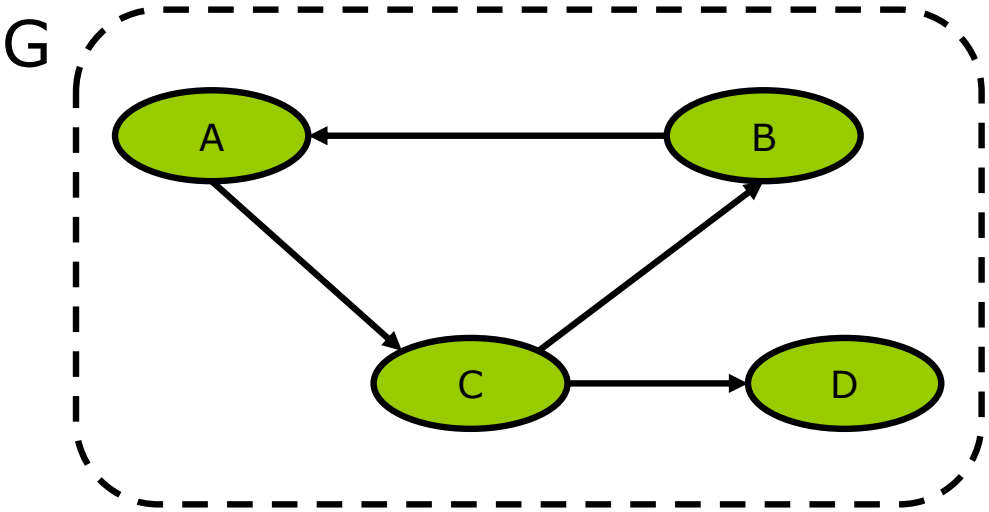
Notation (I)

- A **graph** *G* is a set of nodes and edges: *G (N, E)*
- *N* - **Nodes** (or vertices): n_1, n_2, … N_m
- *E* - **Edges** are represented as pairs of nodes: (n_1, n_2)
  - An edge is said to be **incident** to n_1 and n_2
  - Also, n_1 and n_2 are said to be **adjacent**
  - An edge is drawn as a line between n_1 *and* n_2
  - **Directed edges** entail direction: *from* n_1 *to* n_2
  - An edge is said to be **multiple** if there is another edge exactly relating the same nodes
  - An **hyperedge** is an edge inciding in more than 2 nodes.
- **Multigraph**: If it contains at least one multiple edge.
- **Simple graph**: If it does not contain multiple edges.
- **Hypergraph**: A graph allowing hyperedges (the edge can connect more than two nodes).

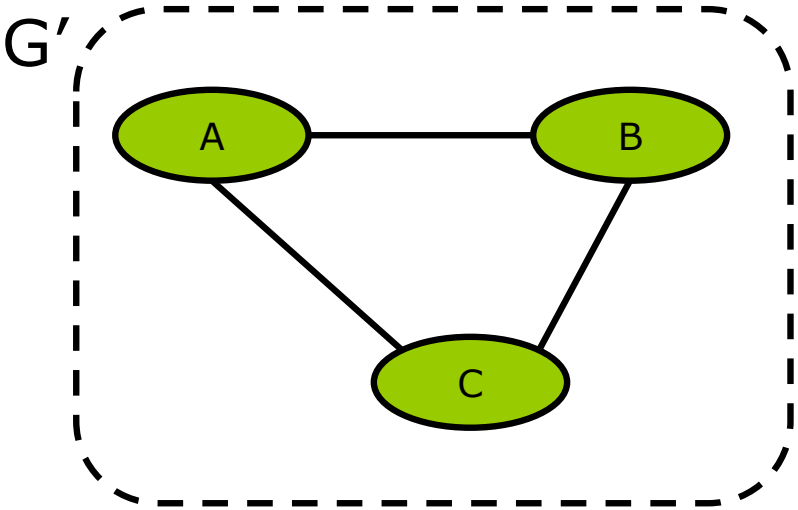
Notation (II)

- **Size** (of a graph): #edges
- **Degree** (of a node): #(incident edges)
  - The degree of a node denotes the node adjacency
  - The neighbourhood of a node are all its adjacent nodes
- **Out-degree** (of a node): #(edges leaving the node)
  - Sink node: A node with 0 out-degree
- **In-degree** (of a node): #(incoming edges reaching the node)
  - Source node: A node with 0 in-degree
- Cliques and trees are specific kinds of graphs
  - **Clique**: Every node is adjacent to every other node
  - **Tree**: A connected acyclic simple graph

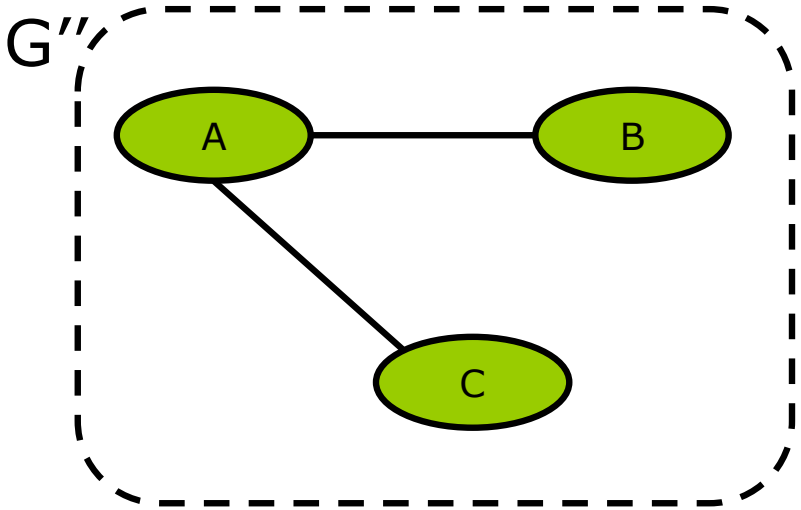
Notation (II) - example



- **Size** (G) = 4
- **Degree** (C): 3
- **Out-degree** (C): 2
- **In-degree** (C): 1



clique

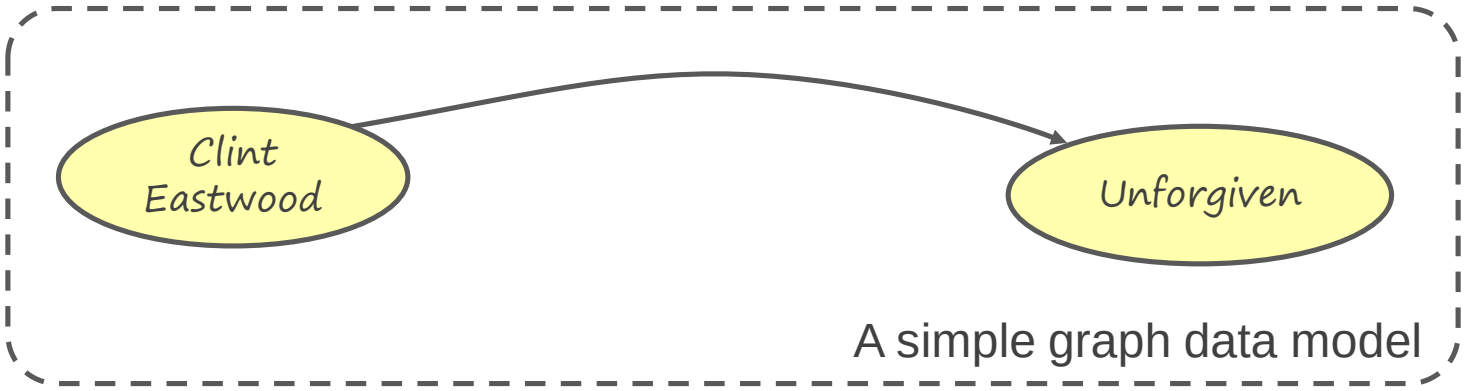


tree

Graph data models – simple graphs

Graphs can be used to encode data, whereby:

- **Nodes** represent objects in a domain
- **Edges** represent relationships between these objects

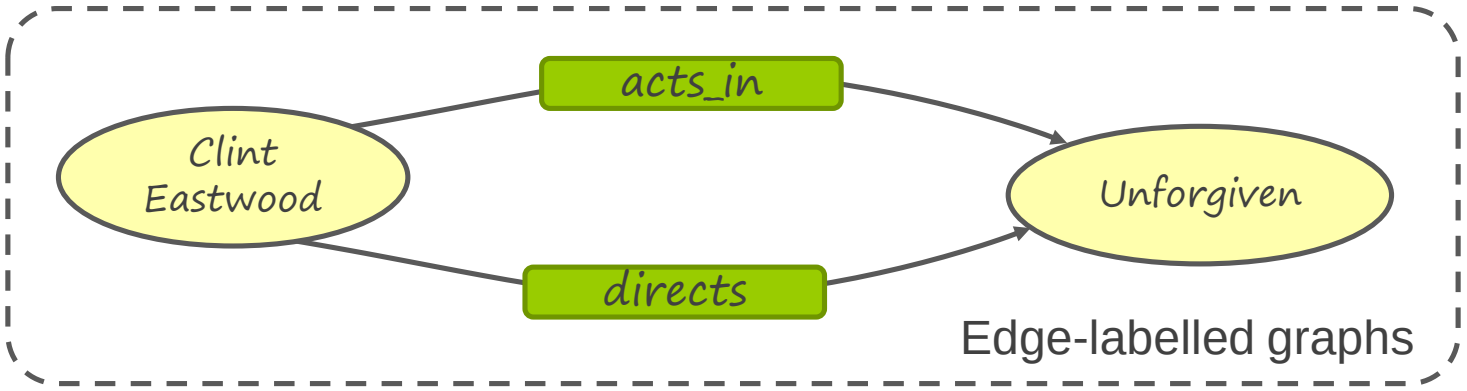


What if we have many types of relationships?

Graph data models – edge labelled

Graphs can be used to encode data, whereby:

- **Nodes** represent objects in a domain
- **Edges** represent relationships between these objects

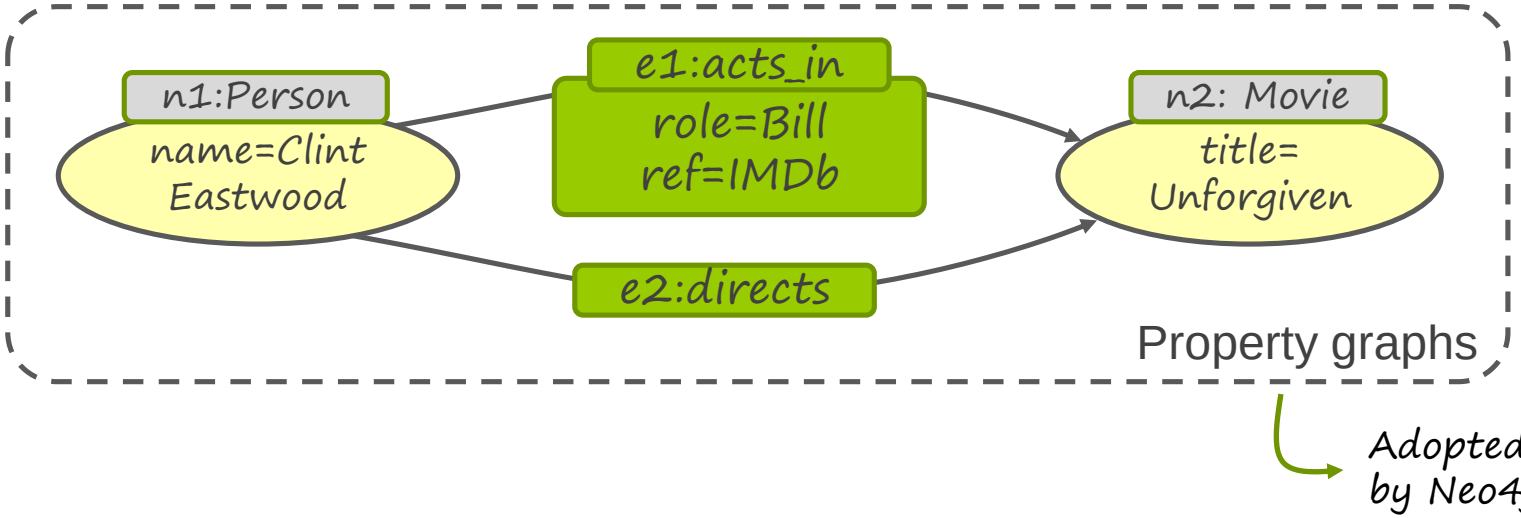


What if want to add new types of information to the edges?
e.g., acts_in relations are sourced from IMDB

Graph data models – property graphs

Graphs can be used to encode data, whereby:

- **Nodes** represent objects in a domain
- **Edges** represent relationships between these objects



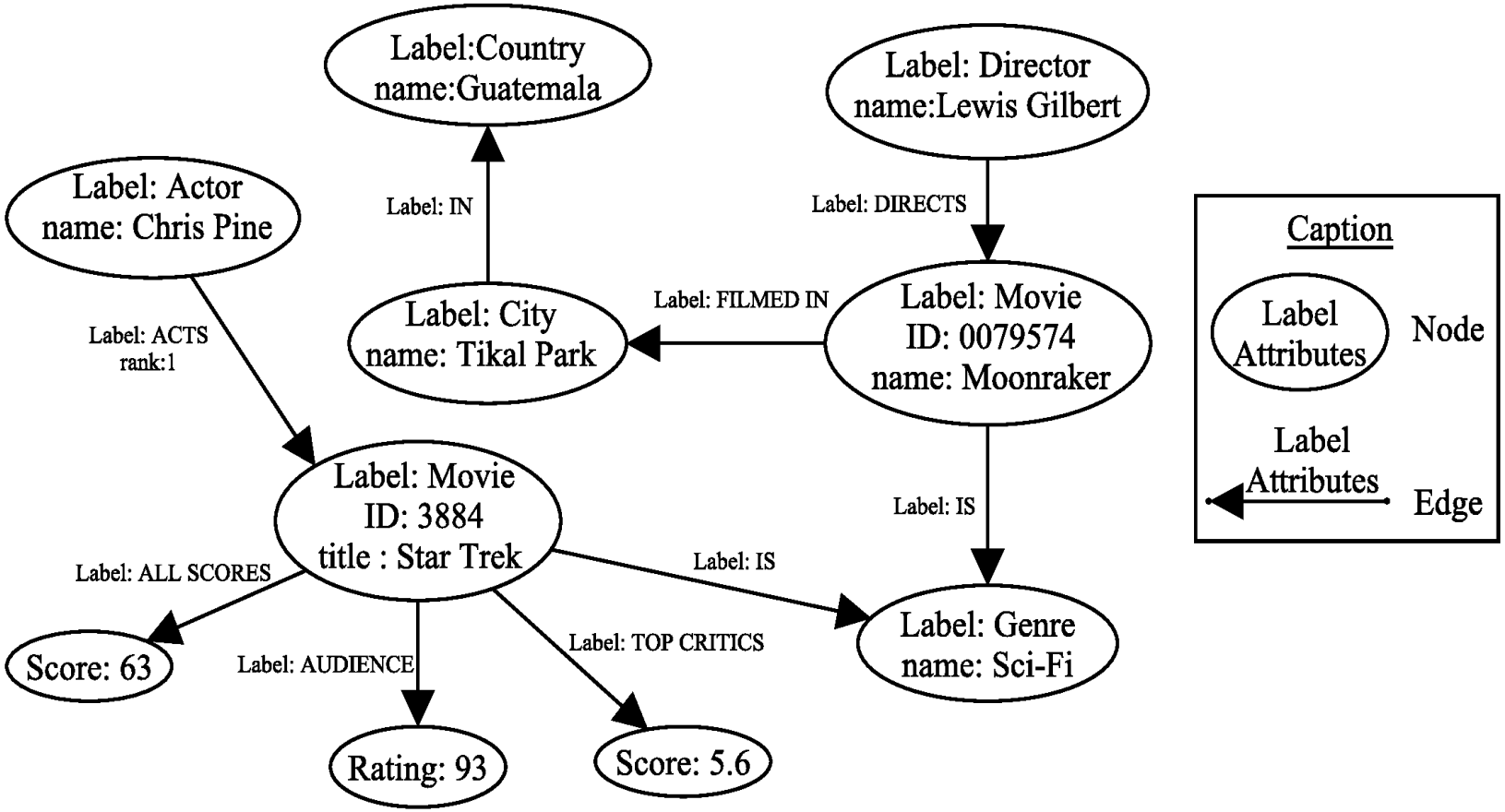
- **Properties** sets of property-value pairs called attributes.

The Property Graph Data Model

- Two main constructs: nodes and edges
  - Nodes represent entities,
  - Edges relate pairs of nodes, and may represent different types of relationships.
- Nodes and edges might be labeled, and may have a set of properties represented as attributes (key-value pairs)***
- Further assumptions:
  - Edges are directed,
  - Multi-graphs are allowed.

*** *Note: in some definitions edges are not allowed to have attributes*

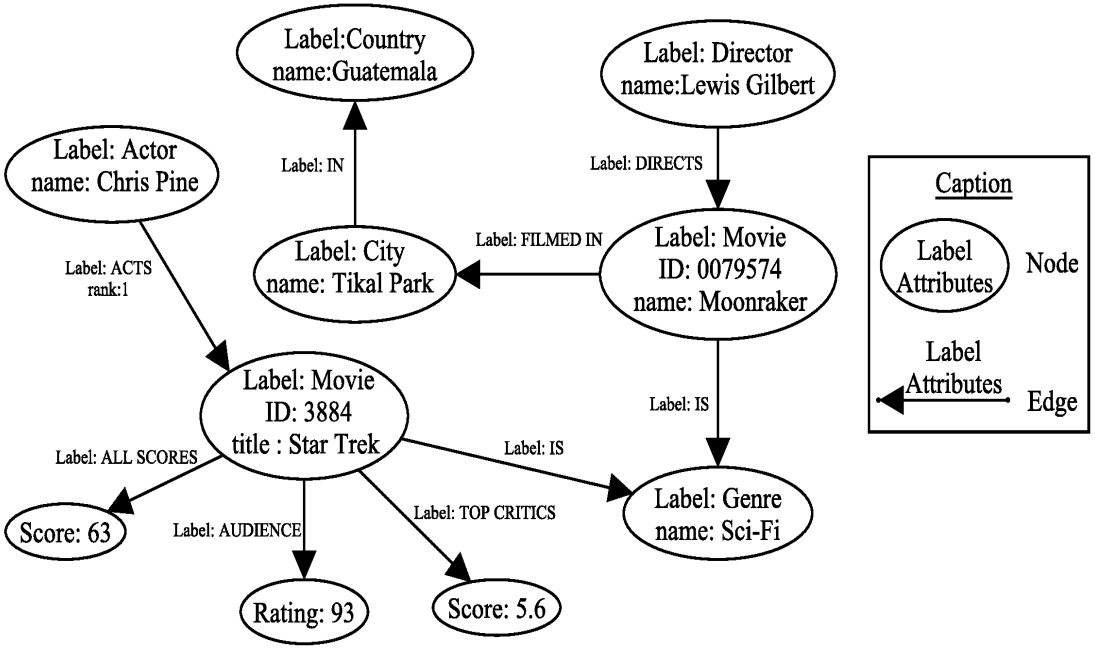
Example of Graph Database



http://grouplens.org/datasets/movielens/

Activity 1: Querying Graph Databases

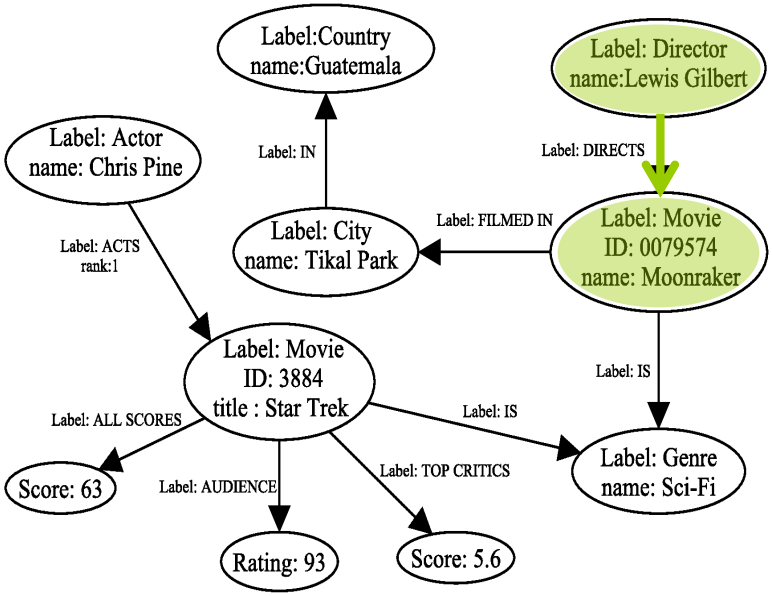
- *Objective: understand the main differences with regard to RDBMS*

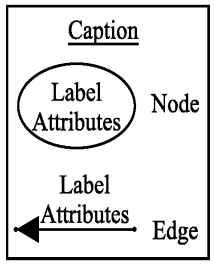


- What movies did Lewis Gilbert direct?
  - How would a RDBMS perform this query?

Activity 1 - discussion

- What movies did Lewis Gilbert direct?





Movie				
ID	name	scoreA	scoreA	…
1	Moonr aker			

FilmedIn	
movieID	City
1	Tikal Park

Actor		
actID	Name	Age
10	Chris	48

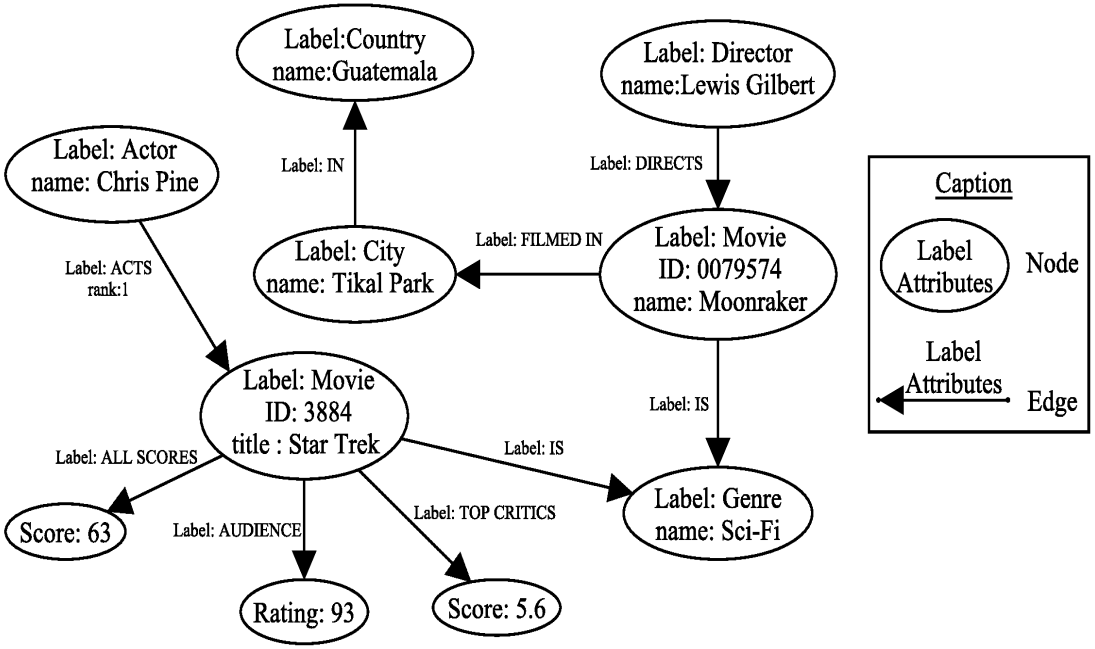
ActedIn	
actorID	movieID
10	4

Director		
dirID	Name	Age
5	Lewis	60

Directed		
dirID	movID	Year
5	1	1995

Activity 2: Querying Graph Databases

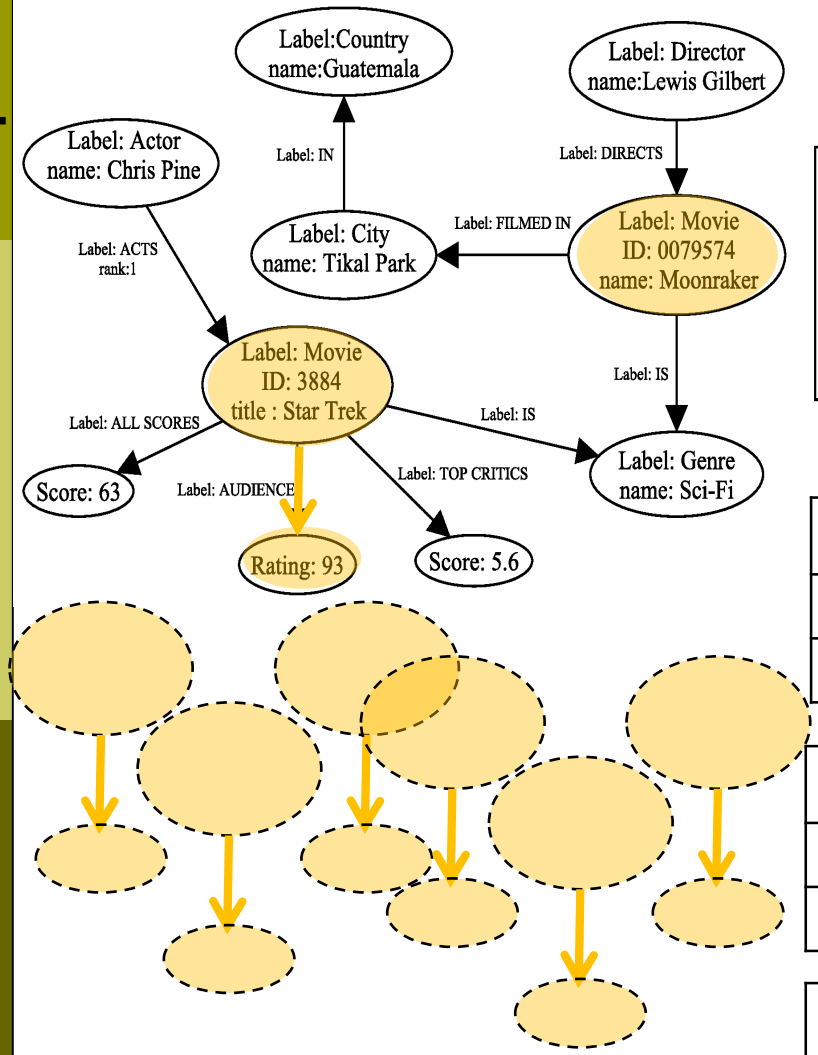
- *Objective: understand the main differences with regard to RDBMS*

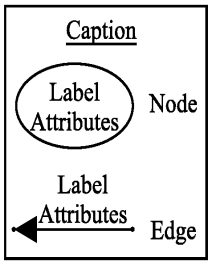


- What movies did receive a rating lower than 60 by the audience?
  - How would a RDBMS perform this query?

Activity 2 - discussion

- What movies did receive a rating lower than 60 by the audience?





Movie	
ID	name
1	Moonraker

MovieScores	
movID	AudSco
1	9.0

FilmedIn	
movieID	City
1	Tikal Park

Actor		
actID	Name	Age
10	Chris	48

ActedIn	
actorID	movieID
10	4

Director		
dirID	Name	Age
5	Lewis	60

Directed		
dirID	movID	Year
5	1	1995

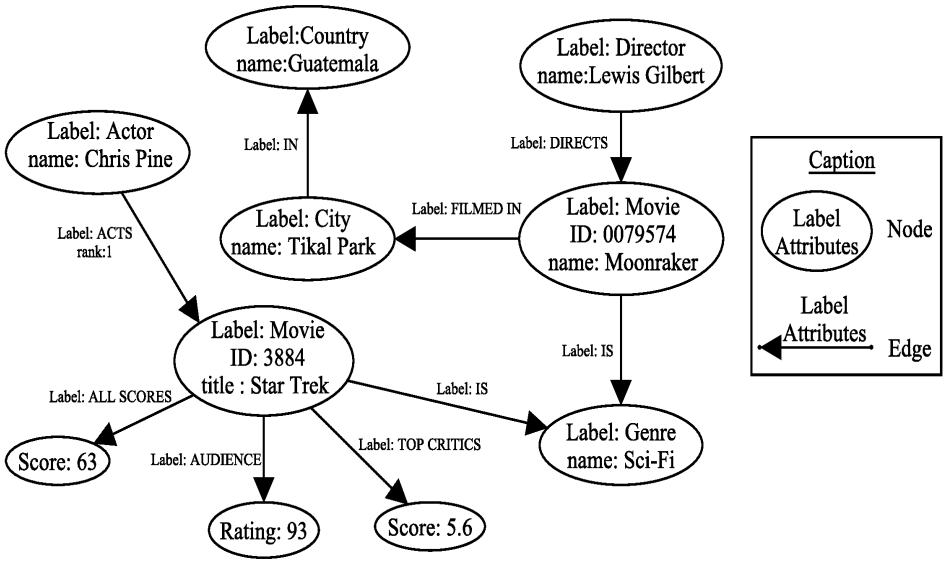
Activity: The Graph Data Model

- *Objective: Understand the graph data model*
- *Tasks:*
  1. *(5') With a teammate think of the following:*
     I. *Think of three-four queries that naturally suit the graph data model*
     II. *Think of three-four queries that do not suit the graph data model that nicely*
  2. *(5') Think tank: So, what kind of queries graph databases are thought for?*

Activity: The Graph Data Model - discussion

- Find all the directors that have filmed a movie in Guatemala?
- Find the co-actors of a movie and the number of movies they act in together.
- Find the actors who acted in in the same movie as my co-actors.

Queries that exploit the graph structure (traverse the graph) from a starting data point are **good**.



- Find all the actors that act in a movie starting with letter "S".
- Find the genre that has the highest average rating.
- Find the movie that the audience liked the most.

Queries that do not start from a known data point and require scans, or queries where relationships do not matter are **bad**.

GDBs Keystone: Traversal Navigation

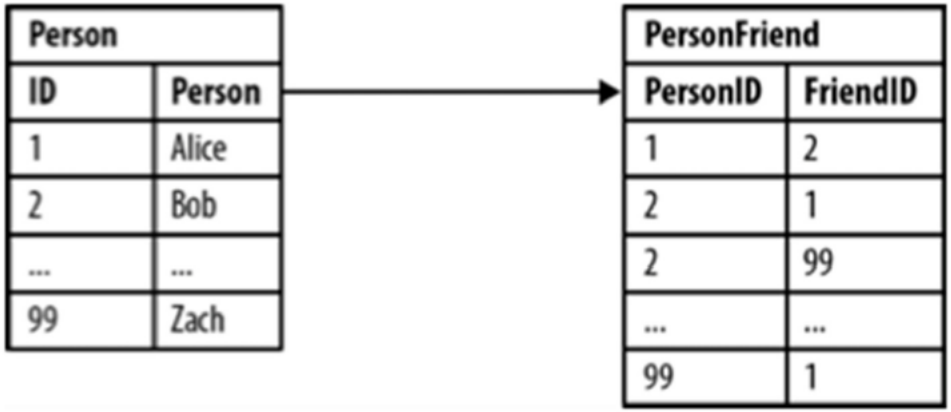
- We define the graph traversal pattern as: *"the ability to rapidly traverse structures to an arbitrary depth (e.g., tree structures, cyclic structures) and with an arbitrary path description (e.g. friends that work together, roads below a certain congestion threshold)"* [Marko Rodriguez]

GDBs Keystone: Traversal Navigation

- We define the graph traversal pattern as: *"the ability to **rapidly** traverse structures to an **arbitrary depth** (e.g., tree structures, cyclic structures) and with an **arbitrary path description** (e.g. friends that work together, roads below a certain congestion threshold)"* [Marko Rodriguez]
- Totally opposite to set theory (on which relational databases are based on)
  - Sets of elements are operated by means of the relational algebra

Traversing Data in a RDBMS

- Modeling friends and friends of friends in a relational database



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

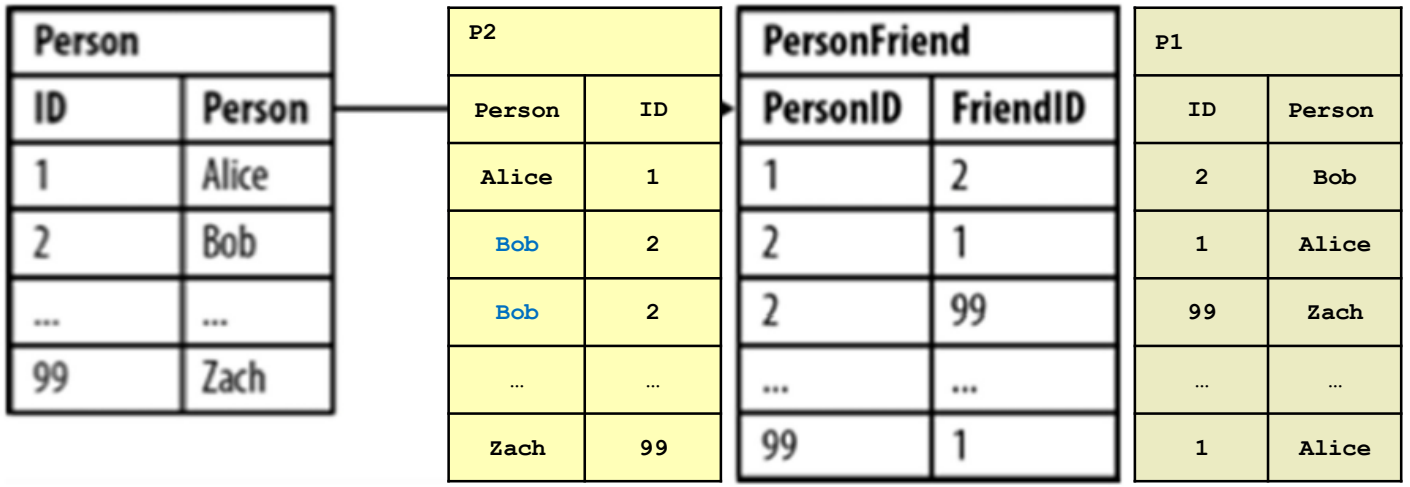
- *PersonFriend does not have to be considered symmetric: Bob may consider Zach to be his friend, but the converse does not necessarily hold

Example taken from the Slides of Domenico Lembo and Ricardo Rosati: "Graph Databases", Sapienza University of Rome

Traversing Data in a RDBMS

means JOINS

- Who are Bob's friends? (i.e., those that Bob considers friends)



Person	
ID	Person
1	Alice
2	Bob
…	…
99	Zach

P2	
Person	ID
Alice	1
Bob	2
Bob	2
…	…
Zach	99

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
…	…
99	1

P1	
ID	Person
2	Bob
1	Alice
99	Zach
…	…
1	Alice

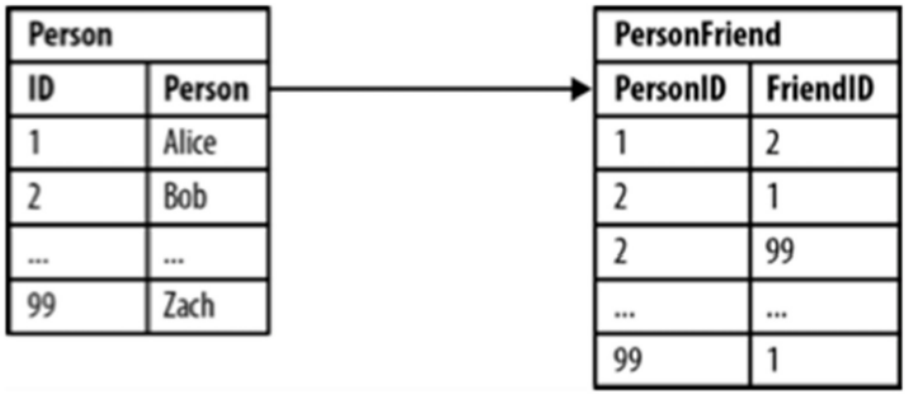
```
SELECT p1.Person
FROM Person p1 JOIN PersonFriend pf
ON pf.FriendID = p1.ID
JOIN Person p2 ON pf.PersonID = p2.ID
WHERE p2.Person = 'Bob'
```

Join to find the names of the friends

Join to find the name of Bob

Traversing Data in a RDBMS – more joins

- Who are Alices's friends-of-friends?



Person	
ID	Person
1	Alice
2	Bob
...	...
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
...	...
99	1

```
SELECT p1.Person AS PERSON, p2.Person AS
FRIEND_OF_FRIEND FROM
PersonFriend pf1 JOIN Person p1 ON
pf1.PersonID = p1.ID JOIN PersonFriend pf2 ON
pf2.PersonID = pf1.FriendID JOIN Person p2 ON
pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Find names of personID

Find friends of friends (friendID)

Find names of fof's

Performance deteriorates as we go more in depth into the network of friends. The cost depends on the depth and the number of rows that need to be joined

Traversing Data in a RDBMS - COST

- Who are Alices's friends-of-friends?

```
SELECT p1.Person AS PERSON, p2.Person AS
FRIEND_OF_FRIEND FROM
PersonFriend pf1 JOIN Person p1 ON
pf1.PersonID = p1.ID JOIN PersonFriend pf2 ON
pf2.PersonID = pf1.FriendID JOIN Person p2 ON
pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
```

Find names of personID

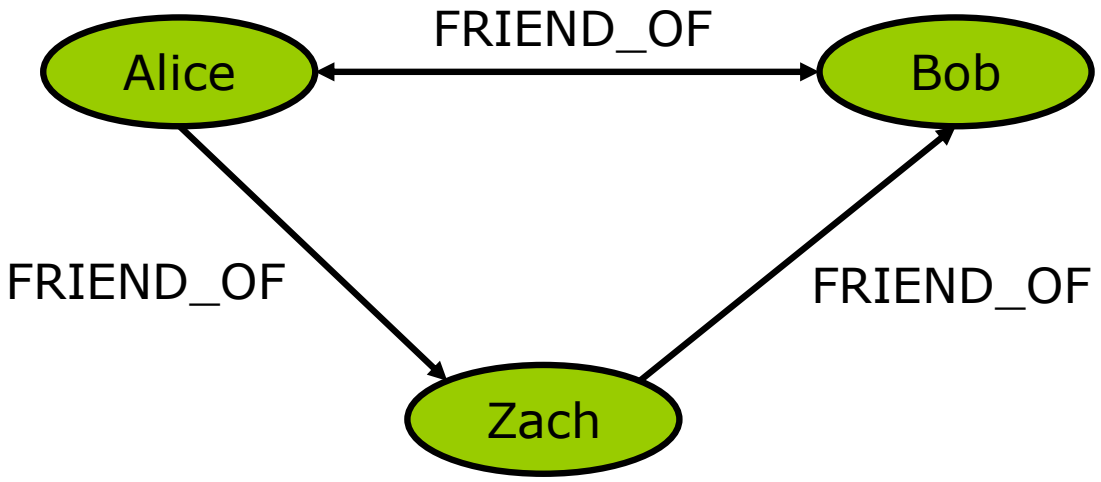
Find friends of friends (friendID)

Find names of fof's

- Indeed, in the SQL query we need *3 joins* (each table is joined twice).
- If n is the number of persons and m is the number of pairs of friends, this means a cost of $O(n^2m^2)$.
- Indexes reduce this cost, since they allow us to avoid linear search over a column.
- Assuming that the structure of the index is a binary search tree, the cost is $O((\log_2 n)^2(\log_2 m)^2)$

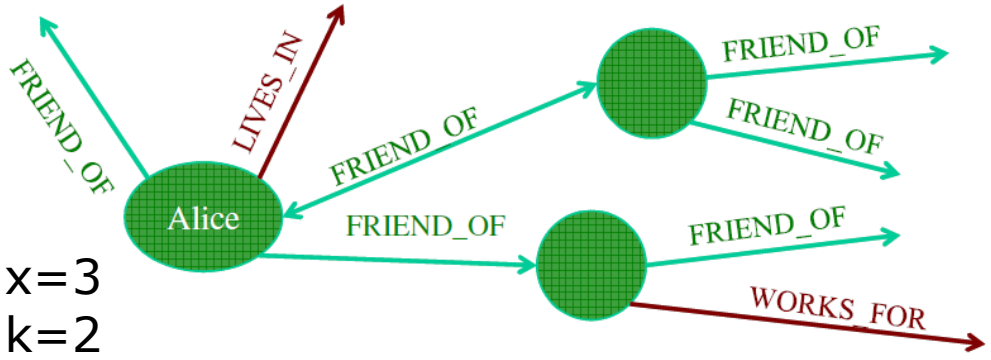
Traversing Data in Graph databases

- Modeling friends and friends-of-friends in a graph database



- Relationships in a graph naturally form paths. **Querying means actually traversing the graph**, i.e., following paths. Because of the fundamentally path-oriented nature of the data model, the majority of path-based graph database operations are extremely efficient (but other operations may be however more difficult)

Traversing Data in Graph DBs - COST



- Query the vertex name O(log2n).
- Starting from a node, we have to scan all outgoing edges to identify FRIEND_OF edges O(x).
- We then have to traverse the edges and repeat the search in all the reached nodes O(k).
- If *x* bounds the number of outgoing edges (assuming $x<<n$), and *k* bounds the number of FRIEND_OF edges outgoing from a node ($k<<m$), then the cost here is *O(x)+O(kx)* (the cost of traversal is constant)
- Local indexes are normally used to speed up local search

Graph DBs vs Relational DBs (experim.)

- The following table reports result of an experiment aimed to finding friends-of-friends in a social network, to a maximum depth of five, for a social network containing 1,000,000 people, each with approximately 50 friends.
- Given any two persons chosen at random, is there a path that connects them that is at most five relationships long?

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

- From *Neo4j in Action. Jonas Partner, Aleksa Vukotic, and Nicki Watt. MEAP. 2012*

Graph DBs vs Relational DBs (Queries)

Relational Databases (querying is through joins)

- In effect, the join operation forms a graph that is dynamically constructed as one table is linked to another table. While having the benefit of being able to dynamically construct graphs, the limitation is that this graph is not explicit in the relational structure, but instead must be inferred through a series of index-intensive operations.
- Moreover, while only a particular subset of the data in the database may be desired (e.g. only Alice's friend's), all data in all queried tables must be examined in order to extract the desired subset

Graph Databases (querying is through traversal paths)

- There is no explicit join operation because vertices maintain direct references to their adjacent edges. In many ways, the edges of the graph serve as explicit, "hard-wired" join structures (i.e., structures that are not computed at query time as in a relational database).
- What makes this more efficient in a graph database is that traversing from one vertex to another is a constant time operation.

Marko A. Rodriguez, Peter Neubauer: The Graph Traversal Pattern. Graph Data Management 2011: 29-46

REFRESHING SOME BASICS ON GRAPHS

TYPICAL GRAPH OPERATIONS

Basic operations

Given a graph G, the following are operations over G:

- **`AddNode(G,x)`**: adds node x to the graph G.
- **`DeleteNode(G,x)`**: deletes the node x from graph G.
- **`Adjacent(G,x,y)`**: tests if there is an edge from x to y.
- **`Neighbors(G,x)`**: nodes y s.t. there is an edge from x to y.
- **`AdjacentEdges(G,x,y)`**: set of labels of edges from x to y.
- **`Add(G,x,y,l)`**: adds an edge between x and y with label l.
- **`Delete(G,x,y,l)`**: deletes an edge between x and y with label l.
- **`Reach(G,x,y)`**: tests if there a path from x to y.
- **`Path(G,x,y)`**: a (shortest) path from x to y.
- **`2-hop(G,x)`**: set of nodes y s.t. there is a path of length 2 from x to y, or from y to x.
- **`n-hop(G,x)`**: set of nodes y s.t. there is a path of length n from x to y, or from y to x.

Typical Graph Operations

- Content-based queries
  - The value is relevant
    - Get a node, get the value of a node / edge attribute, etc.
    - A typical case are summarization queries (i.e., aggregations)
- Topological queries
  - Only the graph topology is considered
  - Typically, several business problems (such as fraud detection, trend prediction, product recommendation, network routing or route optimization) are solved using graph algorithms exploring the graph topology
    - Computing the betweenness centrality of a node in a social network an analyst can detect influential people or groups for targeting a marketing campaign audience.
    - For a telecommunication operator, being able to detect central nodes of an antenna network helps optimizing the routing and load balancing across the infrastructure.
- Hybrid approaches

Topological Queries (I)

- Categories of queries
  - Adjacency queries
    - Basic node / edge adjacency
    - K-neighbourhood of a node

    Linear cost (on the number of edges to explore)

    Examples: Return all the friends of a person
  - Reachability queries (formalized as a traversal)
    - Fixed-length paths (fixed #edges and nodes)
    - Regular simple paths (restrictions as regular expressions) – Hybrid if the restriction is in the *content*
    - Shortest path

    Hard, to compute, in general, e.g., finding simple paths with desired properties in direct graphs is NP-complete

    Examples: Friend-of-a-friend

Topological Queries (II)

- Categories of queries (cont'd)
  - Pattern matching queries (formalized as the graph isomorphism problem)

    Hard, to compute, in general, NP-complete

    Examples: people without telephone
  - Graph metrics
    - Compute the graph / node order, the min / max degree in the graph, the length of a path, the graph diameter, the graph density, closeness / betwenness of a node, the pageRank of a node, etc.

    Depends on the metric to be computed.

    Examples: Compute business processes bottlenecks (betweenness)

Topological Queries

- Support provided by current GBDs

	Adjacency		Reachability				
Graph Database	Node/edge adjacency	k-neighborhood	Fixed-length paths	Regular simple paths	Shortest path	Pattern matching	Summarization
Allegro	•		•			•	
DEX	•		•	•	•	•	
Filament	•		•			•	
G-Store	•		•	•	•	•	
HyperGraph	•					•	
Infinite	•		•	•	•	•	
Neo4j	•		•	•	•	•	
Sones	•					•	
vertexDB	•		•	•		•	

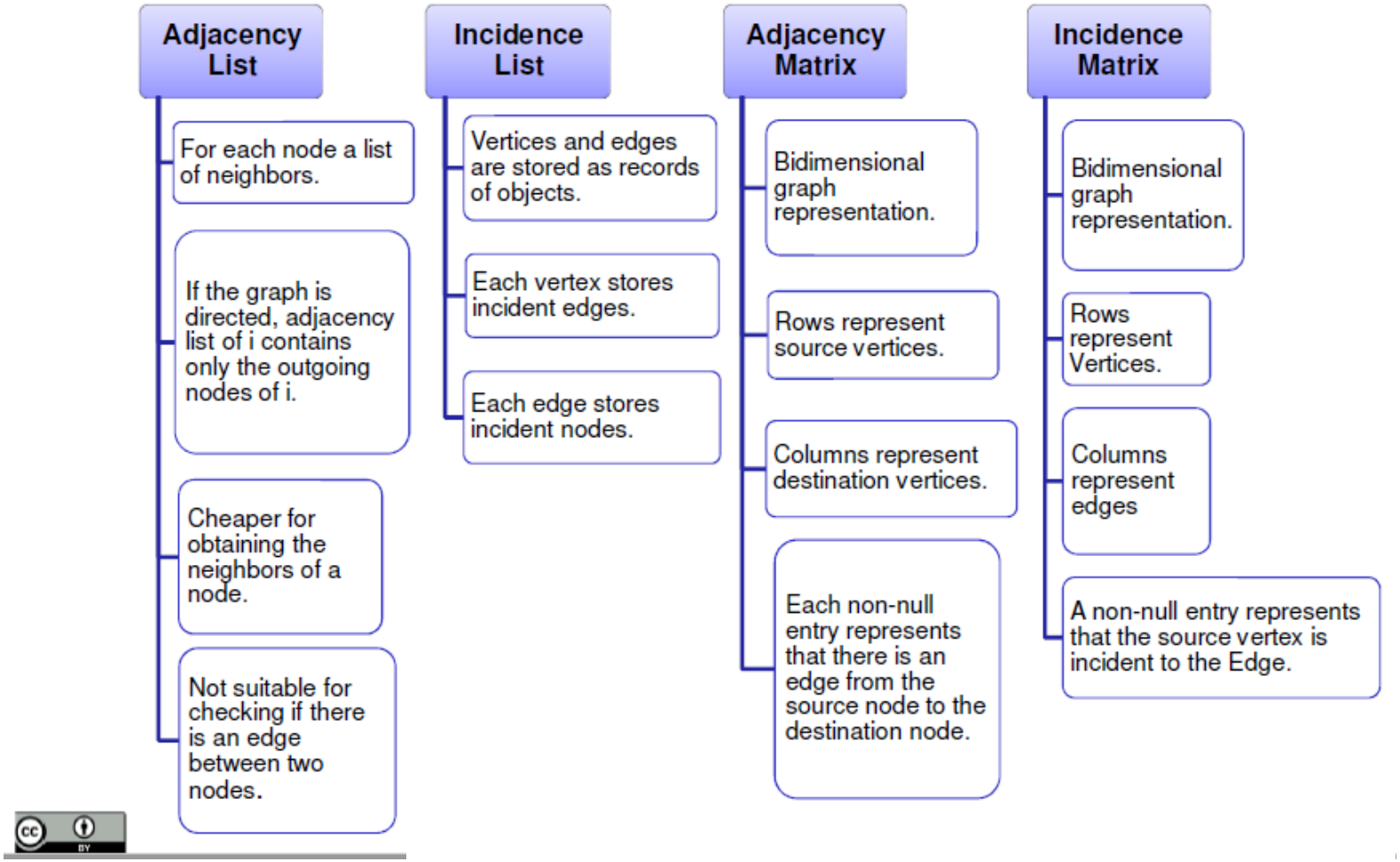
R. Angles. A Comparison of Current Graph Database Models (as of 2012)

REFRESHING SOME BASICS ON GRAPHS

INTERNALS OF GRAPH DATABASES

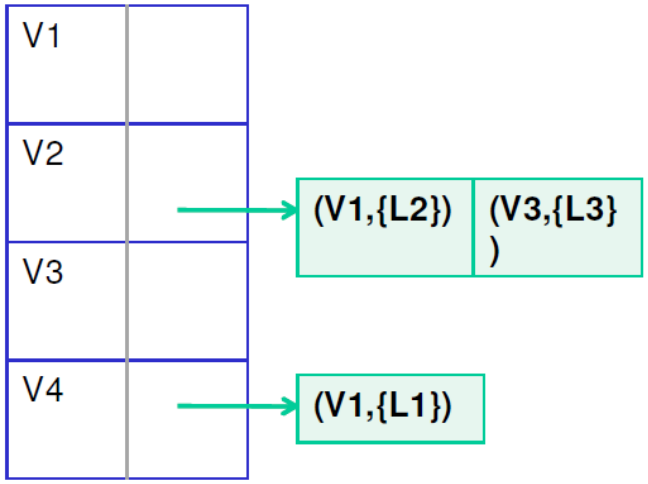
Implementation of Graphs

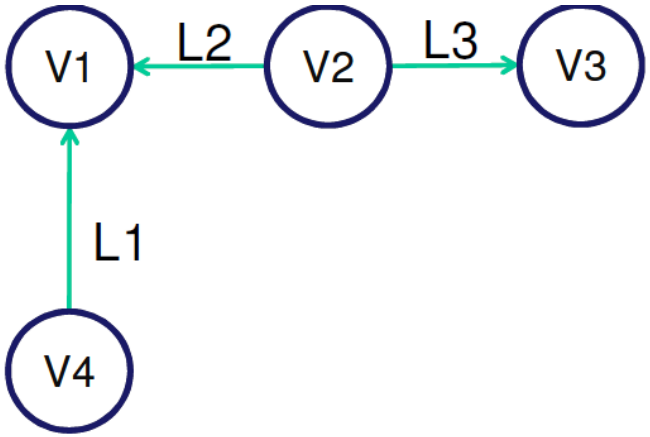
[Sakr and Pardede 2012]



Adjacency List

For each vertex a list of neighbors



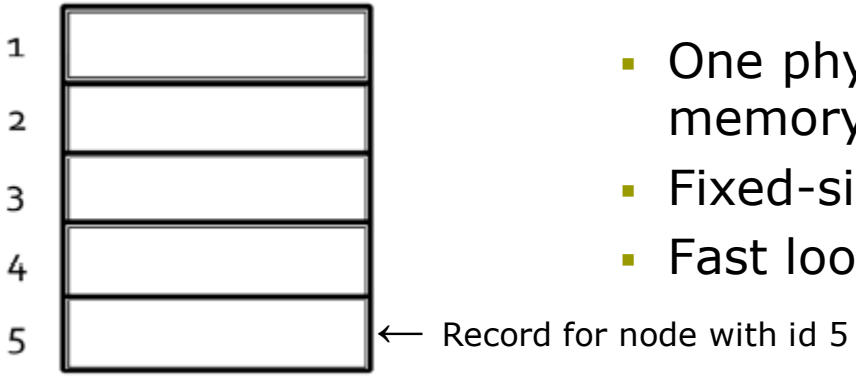


Properties:

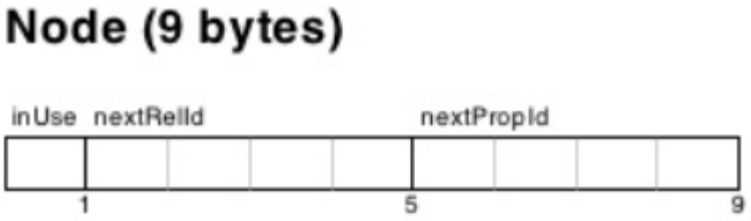
- Storage: O(|V|+|E|+|L|)
- Adjacent(G,x,y): O(|V|)
- Neighbors(G,x): O(|V|)
- AdjacentEdges(G,x,y): O(|V|+|E|)
- Add(G,x,y,l): O(|V|+|E|)
- Delete(G,x,y,l): O(|V|+|E|) 21

Graph Databases Architectures

- Linked Lists (adjacency list) – Neo4J



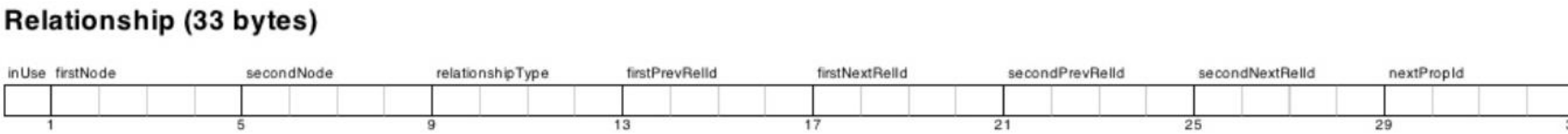
- One physical file to store all nodes (in-memory)
- Fixed-size record: 9 bytes in length
- Fast look-up: O(1)



- Each record is as follows:
- 1 byte (metadata; e.g., in-use?)
- 2-5 bytes: id first relationship
- 6-9 bytes: id first property

Linked Lists – Neo4J

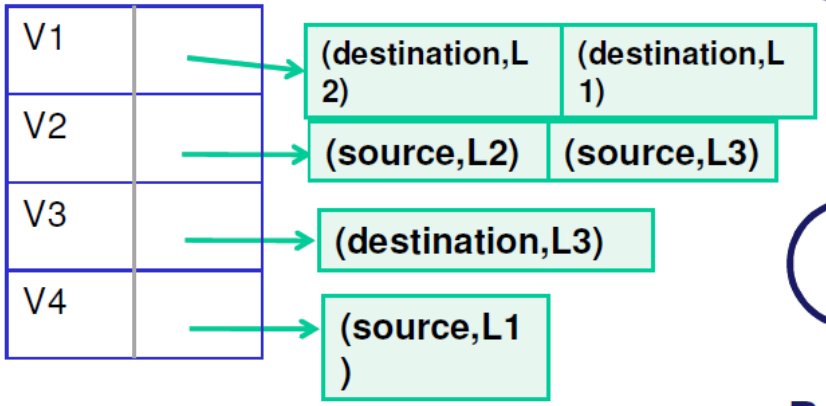
- Two files: relationship and property files.
  - Both contain records of fixed size
  - Cache with Least Frequently Used policy
- Relationship file (similarly for properties)
  - Metadata, id starting node, id end node, id type, ids of the previous and following relationship of the starting node and ending node, id first property

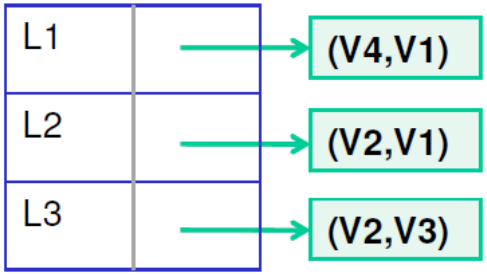


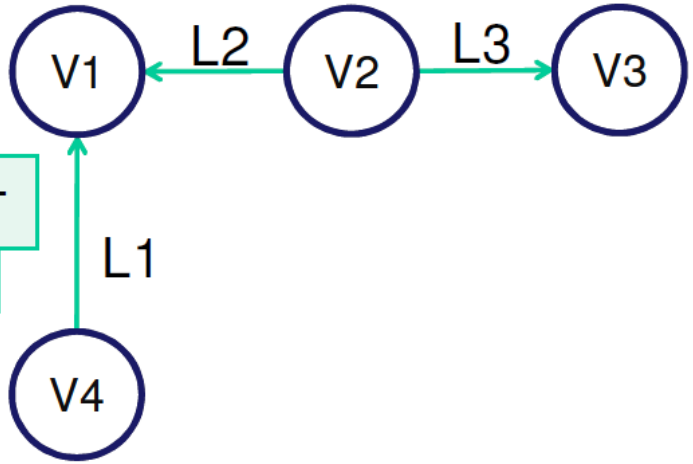
Incidence List*

For each vertex a list of incident edges
For each edge a list of incident vertices

** Simplified version: each edge has a different label*







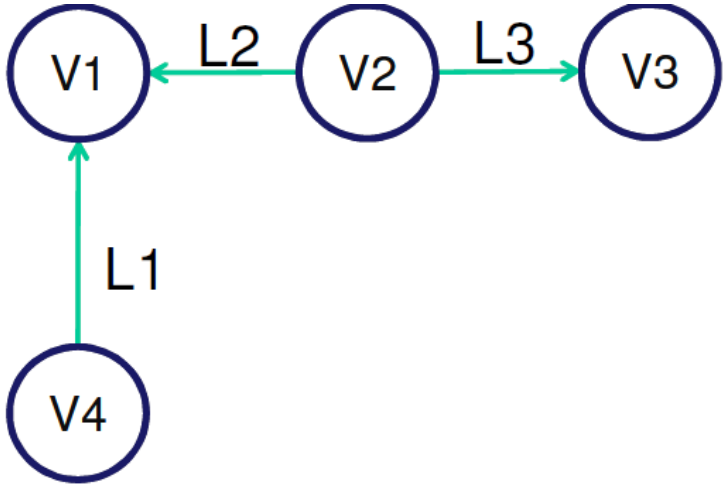
Properties:

- Storage: O(|V|+|E|+|L|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

Adjacency Matrix

Rows represent source vertices, columns the destination vertices

	V1	V2	V3	V4
V1				
V2	{L2}		{L3}	
V3				
V4	{L1}			



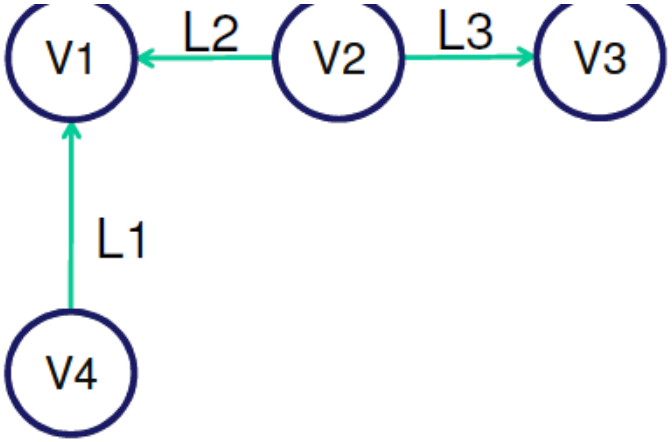
Properties:

- Storage: $O(|V|^2)$
- Adjacent(G,x,y): O(1)
- Neighbors(G,x): O(|V|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|E|)
- Delete(G,x,y,l): O(|E|)

Incidence Matrix

Rows represent source vertices, columns represent edges

	L1	L2	L3
V1	destination	destination	
V2		source	source
V3			destination
V4	source		



Properties:

- Storage: O(|V|x|E|)
- Adjacent(G,x,y): O(|E|)
- Neighbors(G,x): O(|V|x|E|)
- AdjacentEdges(G,x,y): O(|E|)
- Add(G,x,y,l): O(|V|)
- Delete(G,x,y,l): O(|V|)

27

Other Issues

- Distributed Graph Databases
  - Graph databases, by nature, do not distribute well (need to identify graph components)
    - http://www3.nd.edu/~rmccune/papers/Think_Like_a_Vertex_MWM.pdf
  - Most popular prototypes:
    - GraphX (Pregel on top of Spark): **Discontinued**
    - Titan
- Graphs on top of column-oriented databases
  - Vertexica (based on Vertica): http://people.csail.mit.edu/alekh/vertexica.html
  - SynopSys (on top of SAP HANA): http://event.cwi.nl/grades2013/16-Rudolf.pdf

An Example of Graph Database

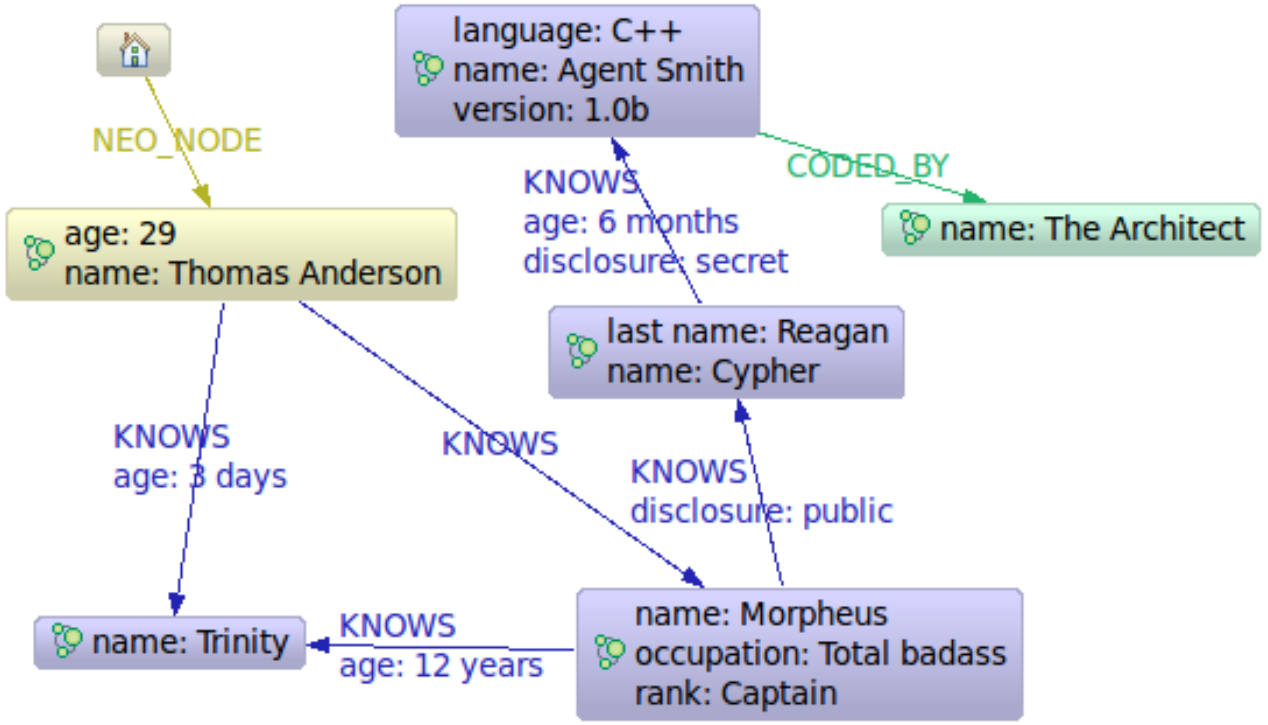
NEO4J

Neo4J Data Model

- It is a graph!
  - Use nodes to represent entities
  - Use relationships to represent the semantic connection between entities
  - Use node properties to represent entity attributes plus any necessary entity metadata such as timestamps, version numbers, etc.
  - Use relationship properties to represent connection attributes plus any necessary relationship metadata, such as timestamps, version numbers, etc.
- Unique constraints (~PK) can be asserted
  - `CREATE CONSTRAINT ON (book:Book) ASSERT book.isbn IS UNIQUE`

Neo4J: Data Model

- Example:

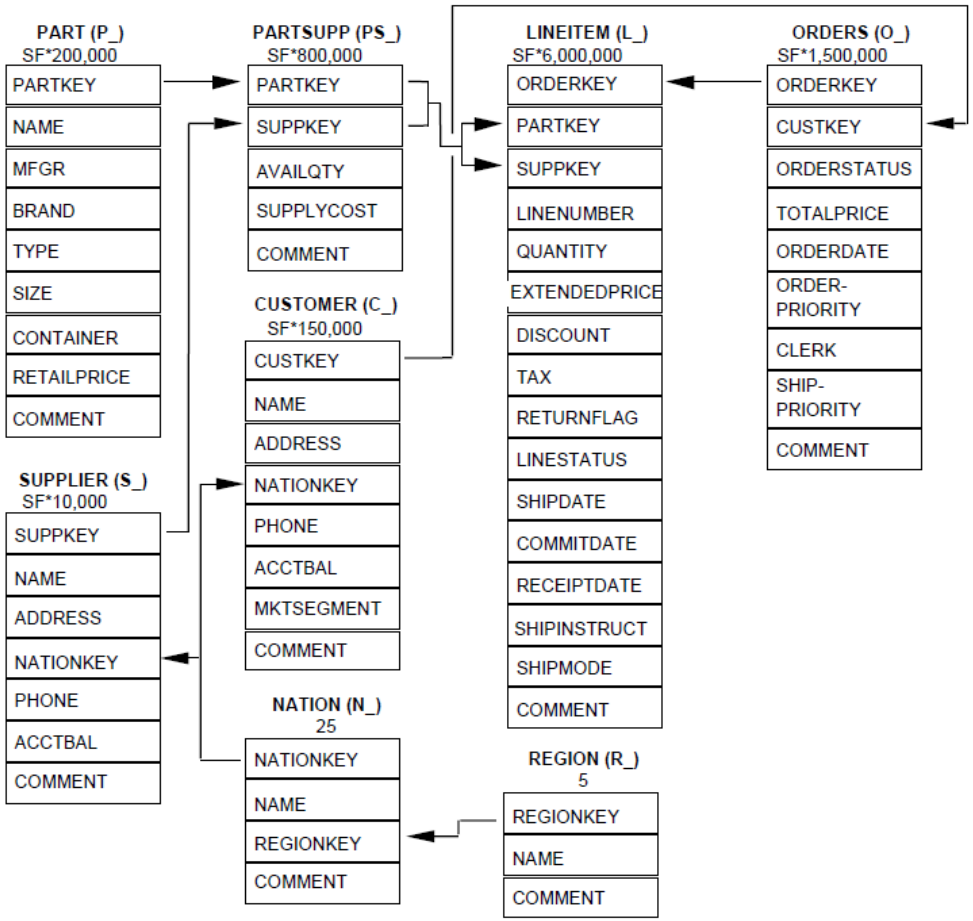


Source: Neo4J Java Tutorial

Activity: Modeling in Neo4J

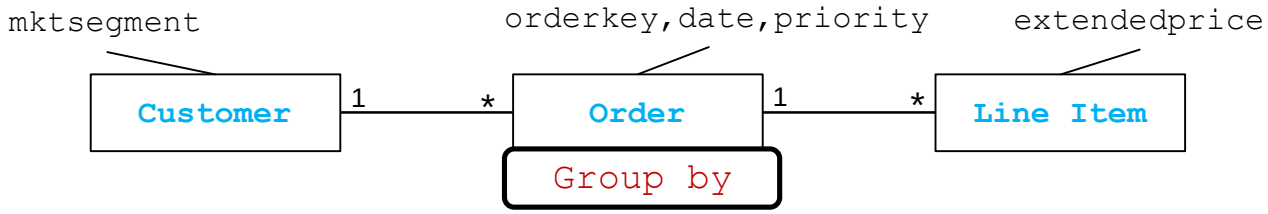
- *Objective: Learn how to model graphs*
- *Tasks:*
  1. *(15') Model the TPC-H database as a graph*
  2. *(5') Discussion*

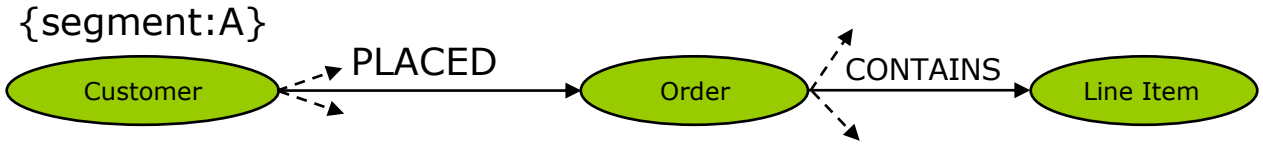
```
SELECT l_orderkey,
sum(l_extendedprice*(1-
l_discount)) as revenue,
o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]'
AND c_custkey = o_custkey AND
l_orderkey = o_orderkey AND
o_orderdate < '[DATE]' AND
l_shipdate > '[DATE]'
GROUP BY l_orderkey,
o_orderdate, o_shippriority
ORDER BY revenue desc,
o_orderdate;
```



Activity: Modeling in MongoDB (II) - discussion

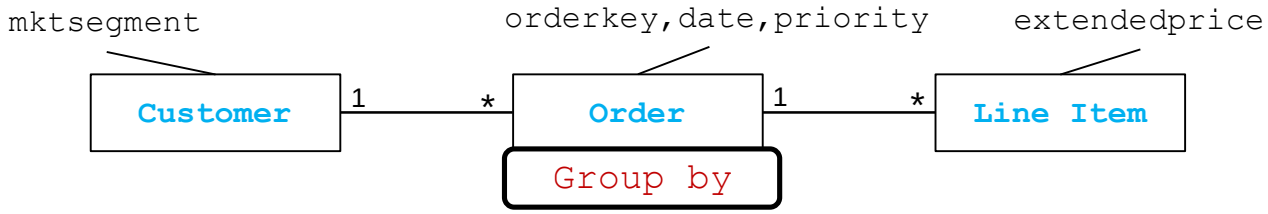
```
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o.priority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]' AND c_custkey = o_custkey AND l_orderkey = o_orderkey AND
o_orderdate < '[DATE]' AND l_shipdate > '[DATE]'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate;
```

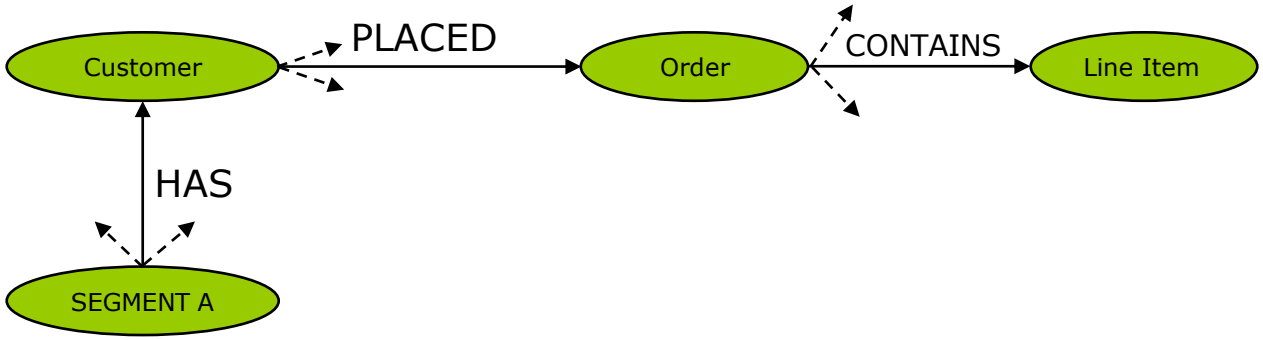




Activity: Modeling in MongoDB (II) - discussion

```
SELECT l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue, o_orderdate, o.priority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]' AND c_custkey = o_custkey AND l_orderkey = o_orderkey AND
o_orderdate < '[DATE]' AND l_shipdate > '[DATE]'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate;
```



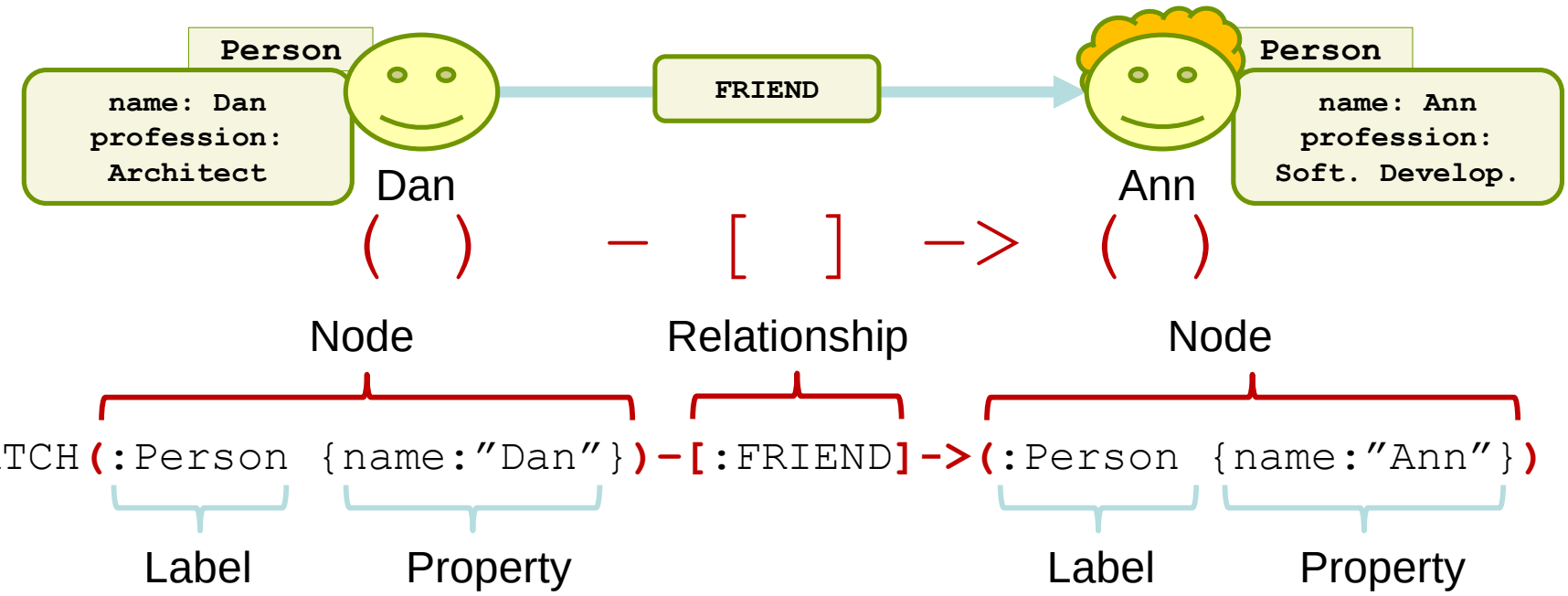


Neo4J: Querying

- The Traversal Framework (from any Neo4J API)
- Cypher
  - High-level, declarative language. It is both a DDL and a DML
    - Opposite idea to Gremlin
    - Query optimizer, welcome!
  - Internally, it uses the traversal framework
  - It can be used from the Shell
  - Contains several clauses:
    - START: Starting points in the graph, obtained via index lookups or by element IDs. [DEPRECATED]
    - MATCH: The graph pattern to match, bound to the starting points in START.
    - WHERE: Filtering criteria.
    - RETURN: What to return.
    - CREATE: Creates nodes and relationships.
    - DELETE: Removes nodes, relationships and properties.
    - SET: Set values to properties.
    - FOREACH: Performs updating actions once per element in a list.
    - WITH: Divides a query into multiple, distinct parts.

http://docs.neo4j.org/chunked/milestone/cypher-introduction.html
http://docs.neo4j.org/chunked/milestone/introduction-pattern.html

Cypher – powerful and expressive lang




```
MATCH (john {name: 'John'})-[:friend]->()-[:friend]->(fof)
RETURN john, fof
```

Cypher main clauses

- `MATCH` - allows you to specify the **patterns** Neo4j will search for in the database
- `WHERE` - adds **constraints** to the patterns in a MATCH, or **filters** the results of a WITH clause
- `RETURN` - defines **what to include** in the query result set
- `WITH` - allows query parts to be **chained together**, **piping** the results from one to be used as starting points or criteria in the next

Documentation: https://neo4j.com/docs/cypher-manual/current/clauses/

Cypher - variables

- When you reference parts of a pattern or a query, you do so by naming them. These names are called **variables**
- ```
  MATCH (n) --> (b)
  RETURN b;
  ```

  The variables are n and b
- ```
  MATCH (jenn:Person{name:Jennifer})
  RETURN jenn;
  ```

  jenn is a variable

Cypher - examples

- `MATCH (n) RETURN n;`

  Returns all the nodes in the graph
- `MATCH (n:T) RETURN n;`

  Returns all nodes with label of type T
- `MATCH (n) RETURN n.p;`

  Returns the property p of all the nodes in the graph
- `MATCH (n)-[r:R1|R2]->(m) RETURN n,m;`

  Returns all the pairs of nodes that have a directed relationship of type R1 or R2 between them.

Summary

- A graph database is a database, thus every GDBMS (graph database management system) must choose its:
  - Concurrency control
  - Transactions (ACID Vs. BASE)
  - Replication / fragmentation strategy
  - Position with regard to the CAP theorem
  - Etc.
- Graph databases do not scale well
  - Since there is no schema, edges are implemented as pointers and thus affected by data distribution
    - Algorithms to fragment graphs and distribute
- Many graph databases but Neo4j is the most popular (http://www.neo4j.org/)
  - Declarative language (Cypher) – **VERY interesting!**
  - Traversal package
  - Basic graph package
- Be aware of graph databases in the near future. They are de facto standard for Linked Data
- Becoming more and more popular for Open Data

Bibliography

- Ian Robinson et al. Graph Databases. O'Reilly. 2013 (http://graphdatabases.com/)
- http://linkeddata.org/
- Resource Description Framework (RDF). W3C Recommendation. Latest version is available at http://www.w3.org/TR/rdf-concepts/
- OWL 2 Web Ontology Language (OWL). W3C Recommendation. Latest version is available at http://www.w3.org/TR/owl2-overview/