Graph Databases

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 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
  - 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 relating exactly 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.

Notation (II)

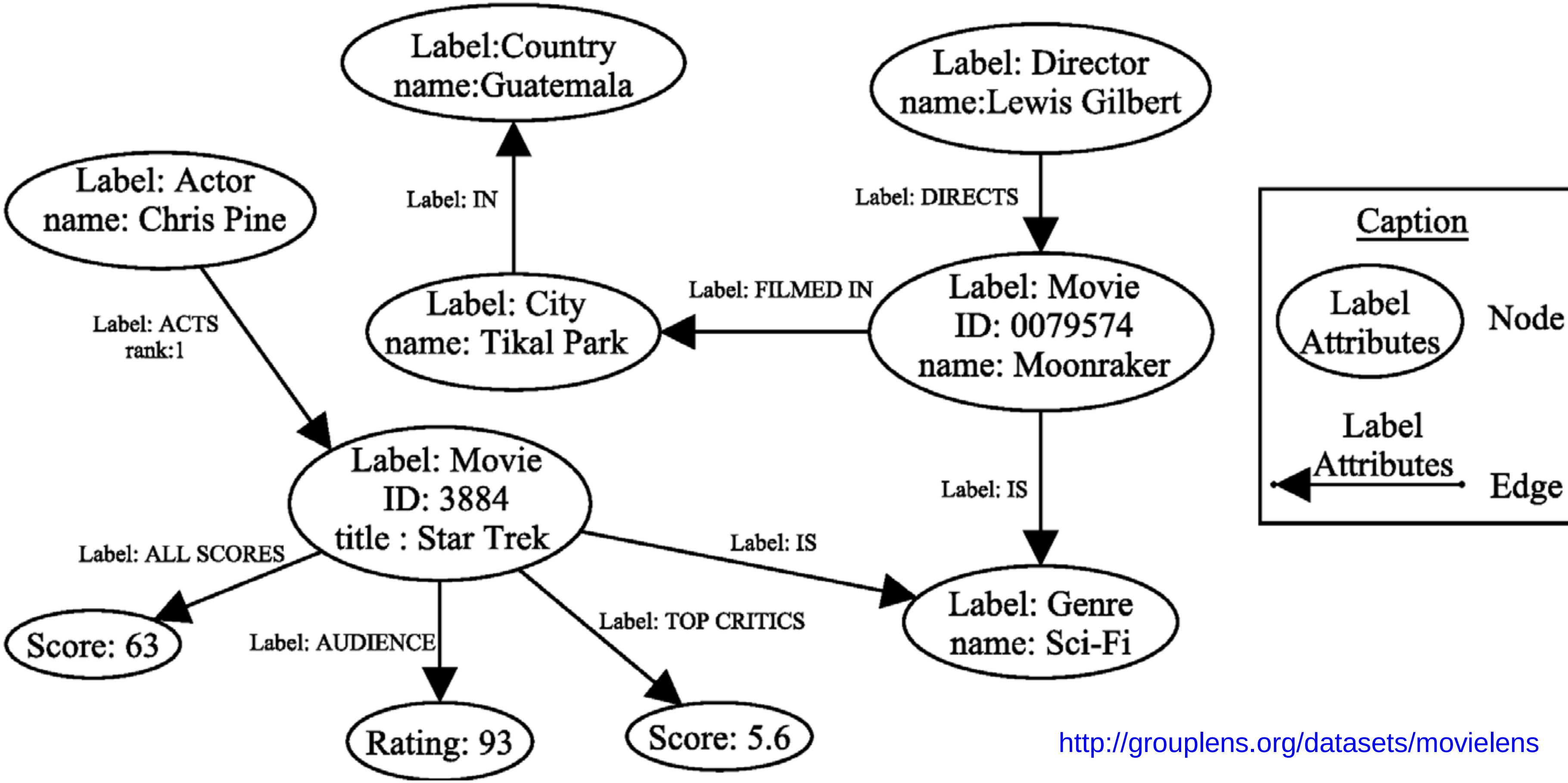
- **Size** (of a graph): #edges
- **Degree** (of a node): #(incident edges)
  - The degree of a node denotes the node adjacency
  - The neighbourhood of a node are all its adjacent nodes
- **Out-degree** (of a node): #(edges leaving the node)
  - Sink node: A node with 0 out-degree
- **In-degree** (of 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

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
  - 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

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

Activity

- *Objective: Understand the graph data model capabilities*
- *Tasks:*
  1. *(5') With a teammate think of the following:*
     1. *Think of three-four queries that naturally suit the graph data model*
     2. *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?*

GDBs Keystone: Traversal Navigation

*"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

- In the relational theory, it is equivalent to **joining** data (schema level) and select data (based on a value)

User	Address	Phone	Email	Alternate
Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
Bob	456 Bar Ave		bob@example.org	
…	…	…	…	…
Zach	99 South St.		zach@example.org	

User	OrderId
Alice	1234
Alice	5678
Alice	9012
…	…
Zach	5588

OrderId	LineItemId
1234	abcd
1234	efab
…	…
5588	bafd

LineItemId	Description	Handling
abcd	strawberry ice cream	freezer
efab	potatoes	
…	…	
bafd	dried spaghetti	

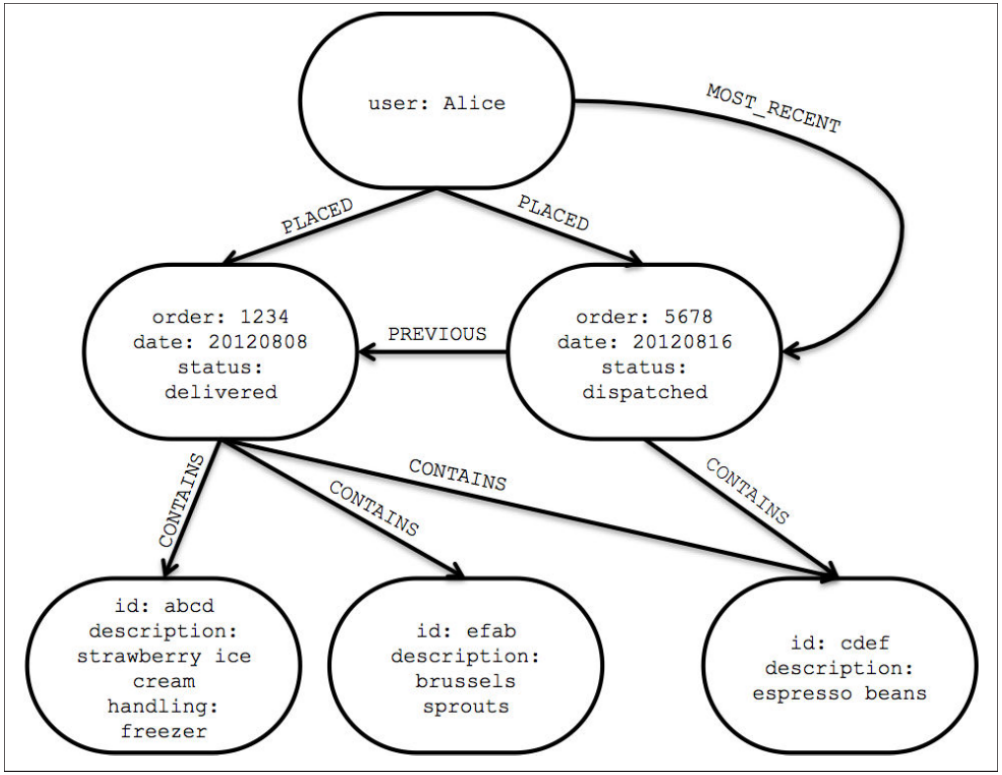
```
SELECT *
FROM user u, user_order uo,
orders o, items i
WHERE u.user = uo.user AND
uo.orderId = o.orderId AND
i.lineItemId = i.LineItemId
AND u.user = 'Alice'
```

Cardinalities:

|User|: 5.000.000
|UserOrder|: 100.000.000
|Orders|: 1.000.000.000
|Item|: 35.000

Query Cost?!

Traversing Data in a Graph Database



Cardinalities:

|User|: 5.000.000
|Orders|: 1.000.000.000
|Item|: 35.000

Query Cost?!

O(N)

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 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 queries

Implementation of the Operations

- Note that the operations presented are conceptual: agnostic of the technology
- The implementation of the ops depends on:
  - The graph database data structure
  - Typical exploration algorithms used
    - A*
    - Breath / depth-first search
    - Constraint programming

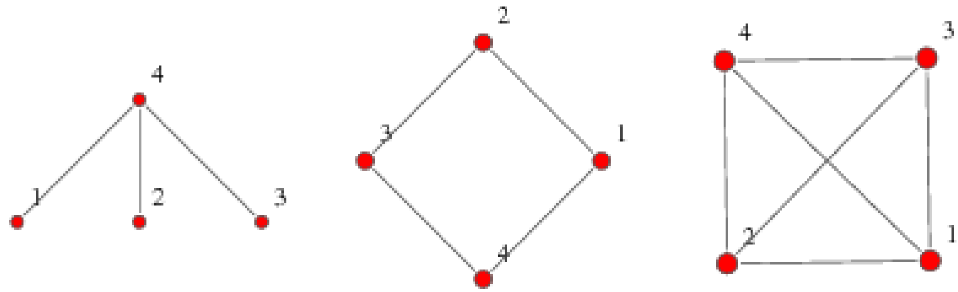
Implementation of Graphs (I)

[Sakr and Pardede 2012]

- **Adjacency List**
  - For each node a list of neighbors.
  - If the graph is directed, adjacency list of i contains only the outgoing nodes of i.
  - Cheaper for obtaining the neighbors of a node.
  - Not suitable for checking if there is an edge between two nodes.
- **Incidence List**
  - Vertices and edges are stored as records of objects.
  - Each vertex stores incident edges.
  - Each edge stores incident nodes.
- **Compressed Adjacency List**
  - Differential encoding between two consecutive nodes
- **Adjacency Matrix**
  - Bidimensional graph representation.
  - Rows represent source vertices.
  - Columns represent destination vertices.
  - Each non-null entry represents that there is an edge from the source node to the destination node.
- **Incidence Matrix**
  - Bidimensional graph representation.
  - Rows represent Vertices.
  - Columns represent edges
  - A non-null entry represents that the source vertex is incident to the Edge.

Implementation of Graphs (II)

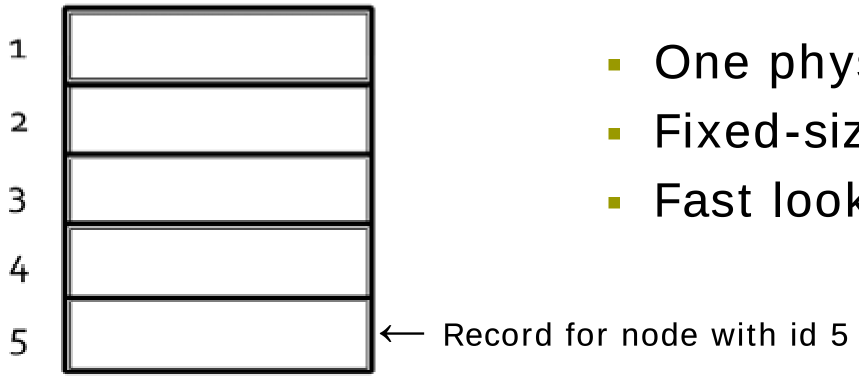
- Adjacency matrix (baseline)



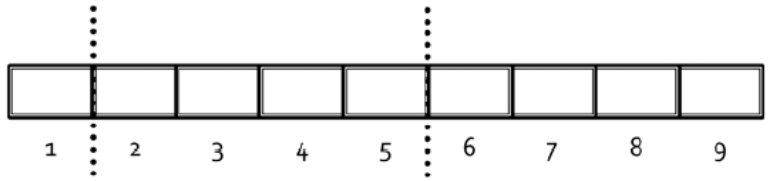
$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Implementation of Graphs (III)

- Linked Lists (adjacency list) – Neo4J



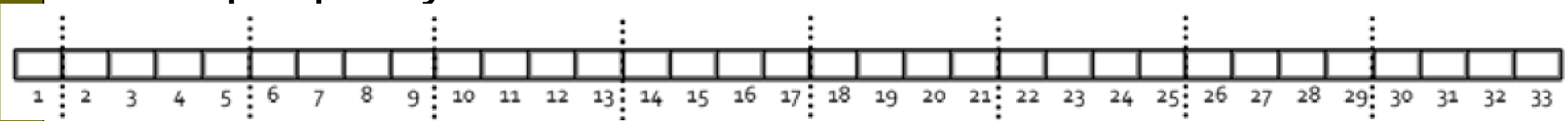
- One physical file to store all nodes (in-memory)
- Fixed-size record: 9 bytes in length
- Fast look-up: O(1)



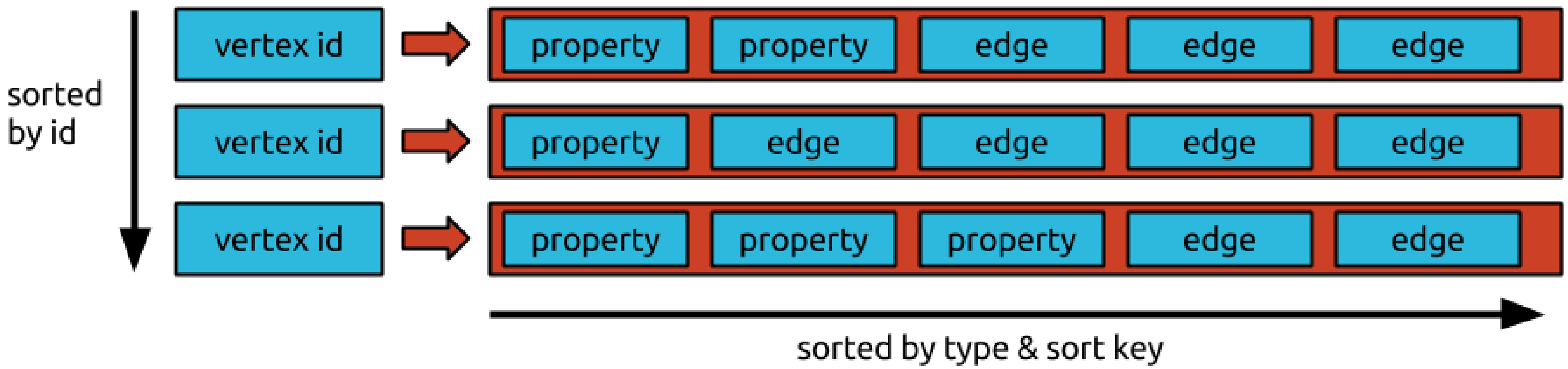
- Each record is as follows:
- 1 byte (metadata; e.g., in-use?)
- 2-5 bytes: id first relationship
- 6-9 bytes: id first property

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



Linked Lists - Titan



- Works on top of Cassandra or Hbase
- Properties and edges are stored as column:value
  - Sort key ~ key design

Linked Lists - Sparksee

- Each node and edge identified by a oid
- The graph is represented as a set of lists
  - Two lists, Heads (e1, v1) and Tails (e1, v2) to represent edges and nodes
  - One list, Labels (e1, ACTS), to represent the type of each element
  - Lists of attributes. A list per *kind of* attribute: Atitle (v1, 'The Big Bang Theory')
- The lists are transformed into value sets (value, <values>)
  - The first element is always a node or an edge

L	(v_1, ARTICLE), (v_2, ARTICLE), (v_3, ARTICLE), (v_4, ARTICLE), (v_5, IMAGE), (v_6, IMAGE), (e_1, BABEL), (e_2, BABEL), (e_3, REF), (e_4, REF), (e_5, CONTAINS), (e_6, CONTAINS), (e_7, CONTAINS)	(ARTICLE, {v_1, v_2, v_3, v_4}), (BABEL, {e_1, e_2}), (CONTAINS, {e_5, e_6, e_7}), (IMAGE, {v_5, v_6}), (REF, {e_3, e_4})
T	(e_1, v_1), (e_2, v_2), (e_3, v_4), (e_4, v_4), (e_5, v_3), (e_6, v_3), (e_7, v_4)	(v_1, {e_1}), (v_2, {e_2}), (v_3, {e_5, e_6}), (v_4, {e_3, e_4, e_7})
H	(e_1, v_3), (e_2, v_3), (e_3, v_3), (e_4, v_3), (e_5, v_5), (e_6, v_6), (e_7, v_6)	(v_3, {e_1, e_2, e_3, e_4}), (v_5, {e_5}), (v_6, {e_6, e_7})
Aid	(v_1, 1), (v_2, 2), (v_3, 3), (v_4, 4), (v_5, 1), (v_6, 2)	(1, {v_1, v_5}), (2, {v_2, v_6}), (3, {v_3}), (4, {v_4})
Atitle	(v_1, Europa), (v_2, Europe), (v_3, Europe), (v_4, Barcelona)	(Barcelona, {v_4}), (Europa, {v_1}), (Europe, {v_2, v_3})

Other popular graph databases

- Pregel
  - Parallel graph processing
- Giraph
  - Extends Pregel with several new features, including sharded aggr.
    - Similar to vertical fragmentation
- OrientDB
  - Mixed document-graph DB
- ArangoDB
  - Mixed document-graph DB

What Is Still Missing?

- Graph data management is getting mature enough as to be used
- However, RDF, RDFS and OWL are something else than graph data management
  - Semantics related to properties
  - Inference allowed by means of exploiting such semantics
- There is no current framework on top of graph data management addressing such problem
  - Many academic prototypes

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
- 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
- *Resource Description Framework (RDF)*. W3C Recommendation.
  - http://www.w3.org/TR/rdf-concepts
- *OWL 2 Web Ontology Language (OWL)*. W3C Recommendation.
  - http://www.w3.org/TR/owl2-overview

Resources

- http://linkeddata.org
- http://www.neo4j.org
- http://www.sparsity-technologies.com
- http://thinkaurelius.github.io/titan