GRAPH DATA MODELING, STORAGE, RETRIEVAL AND PROCESSING

Analysis of Large Scale Social Networks

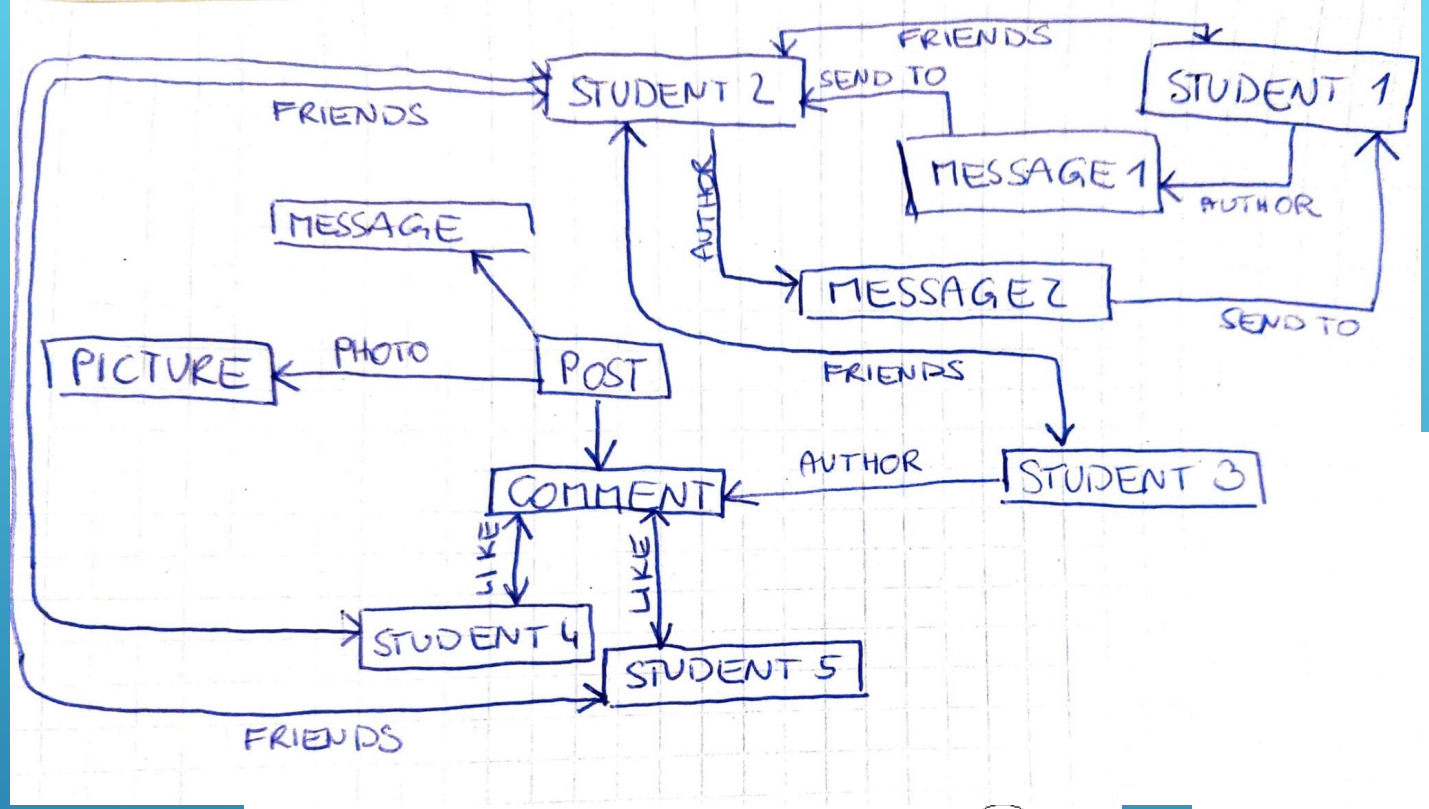
Bart Thijs

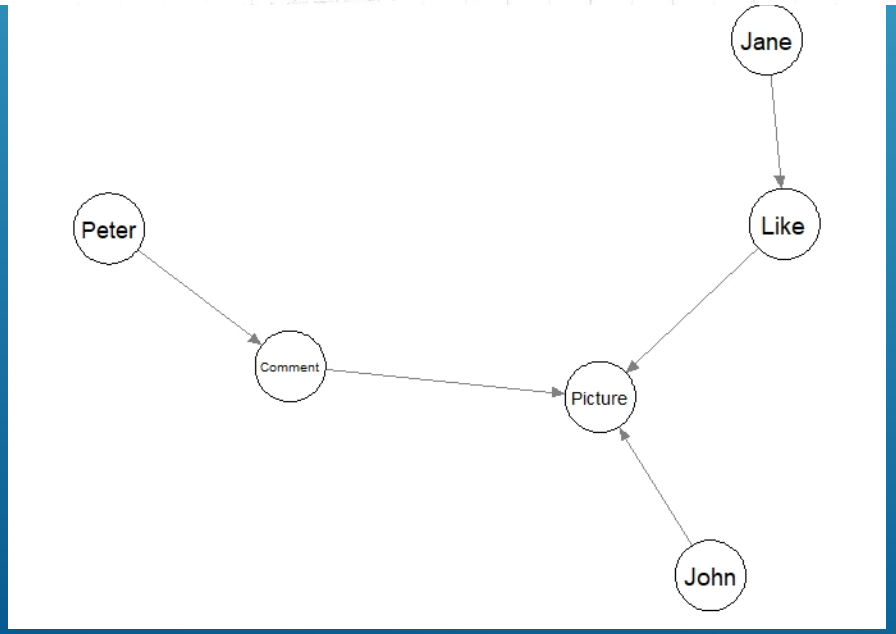
Test Information

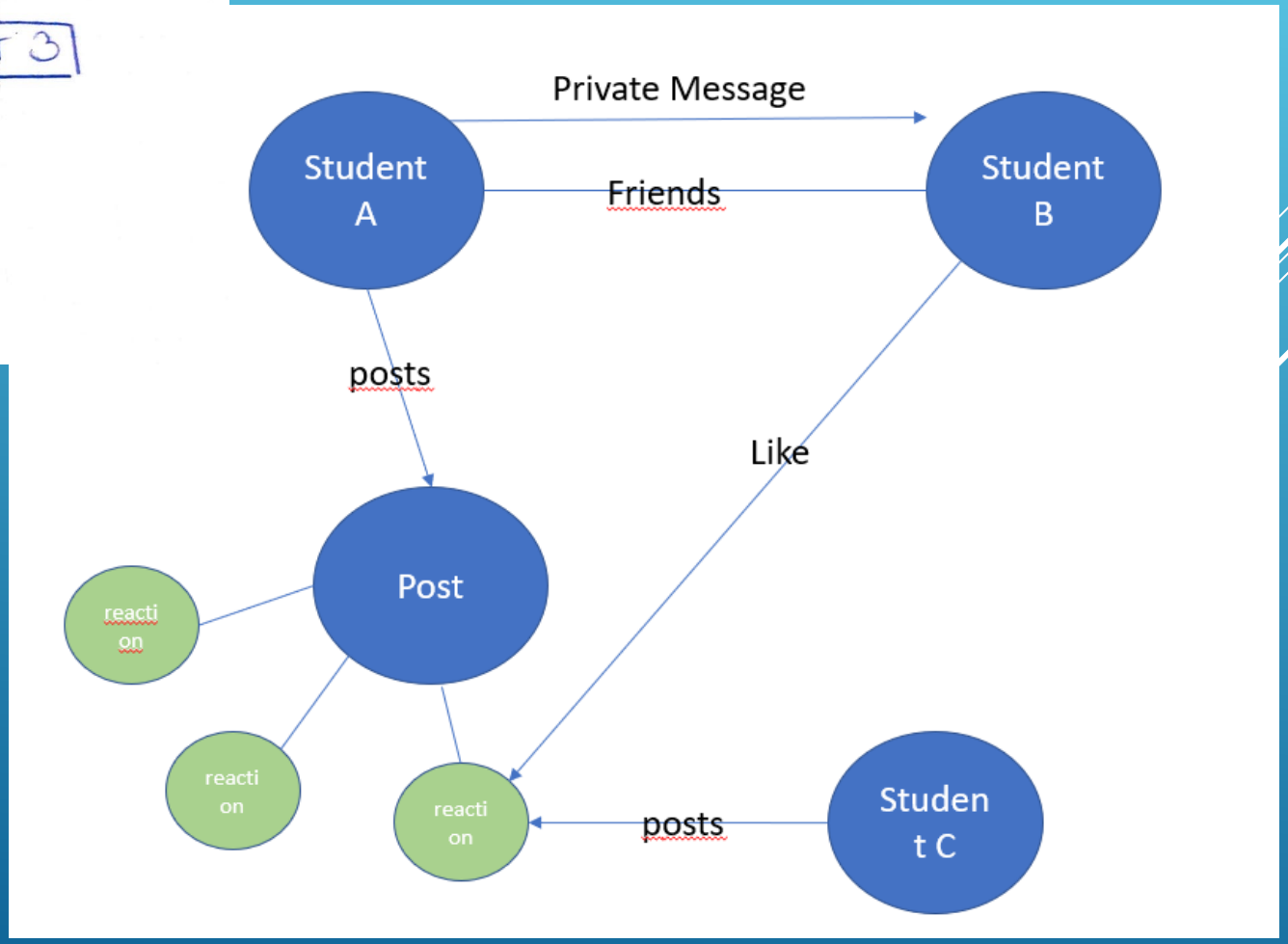
Description

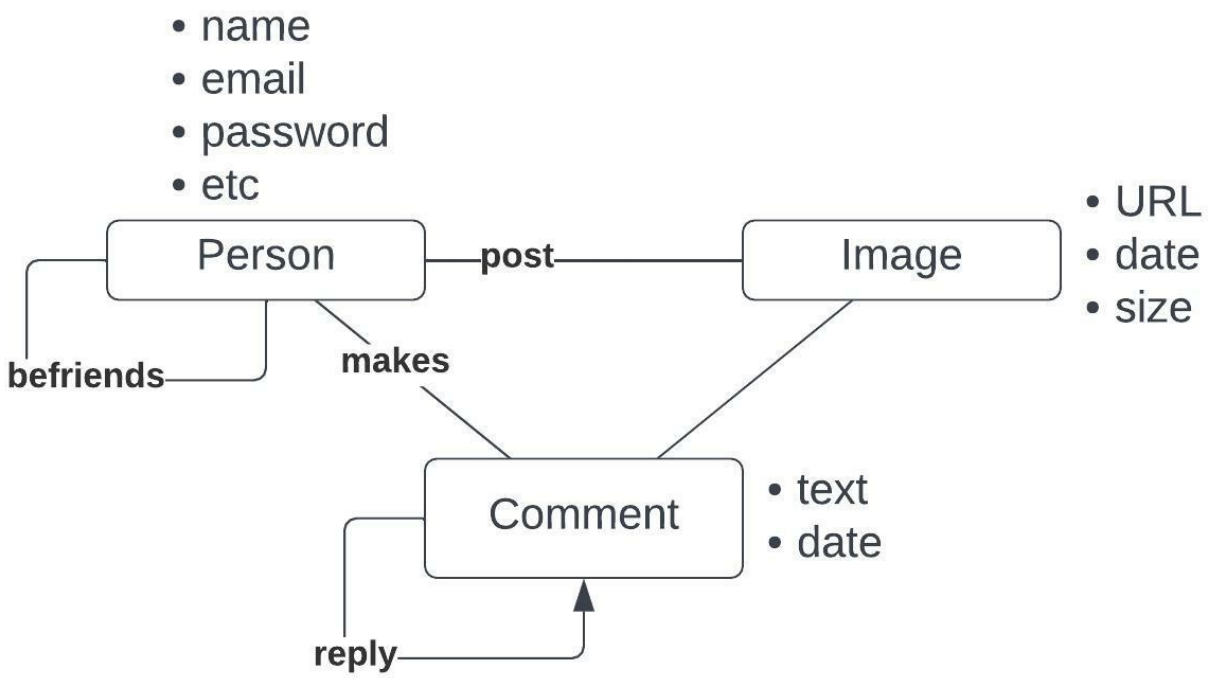
In this first task, you are asked to image a simple Facebook-like social network where users can connect with each other, send private messages, post messages including pictures, add comments on these messages and indicate that they like a comment or message.

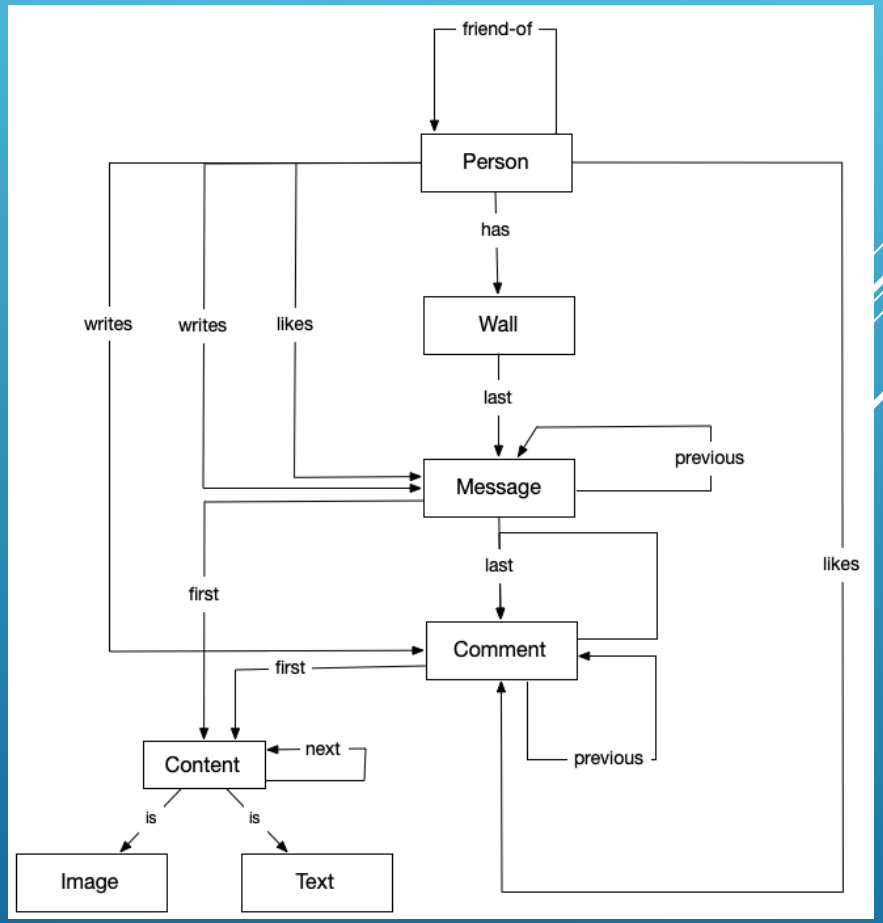
Suppose this situation:
Two students in Leuven who are using this social network app send private messages and plan to meet in the park. There, they take some pictures and post that online with a short message. Then, a friend of these students see the message and comments the post with 'I wish I could be there too'. Other friends like this comment.


FRIENDS
STUDENT 2
SEND TO
STUDENT 1
FRIENDS
MESSAGE 1
AUTHOR
MESSAGE
AUTHOR
MESSAGE 2
SEND TO
PICTURE
PHOTO
POST
FRIENDS
AUTHOR
COMMENT
STUDENT 3
LIKE
LIKE
STUDENT 4
STUDENT 5
FRIENDS


Jane
Peter
Like
Comment
Picture
John


Private Message
Student A
Friends
Student B
posts
Like
Post
reacti on
reacti on
reacti on
posts
Studen t C


• name
• email
• password
• etc
Person
post
Image
• URL
• date
• size
befriends
makes
Comment
• text
• date
reply


friend-of
Person
has
Wall
writes
writes
likes
last
previous
Message
last
likes
first
Comment
first
next
Content
previous
is
is
Image
Text

TAO: THE POWER OF THE GRAPH

Important from this presentation:

- Multiple types of nodes and edges.
- Properties of nodes are sometimes stored in separate nodes because of the possible undefined structure.
- Web rendering queries the graph multiple times at each request because of personalized content and time constraints.
- Each node has unique ID (64bit). Each edge refers to source and target ID and has time stamp
- High priority on reducing '*read latency*', less on *write*
- Consistency is reach 'at some point in time'

GRAPH DATABASES?

Observations:

https://en.wikipedia.org/wiki/Graph_database#List_of_graph_databases

- Microsoft included a graph database in its latest release of SQL-server, it released Graph Engine and introduced Azure Cosmos DB
- Amazon announced Neptune on AWS
- Datastax released the DS Enterprise Graph after the aquisition of Aurelius, the team behind Titan
- Neo4J launched their Native Graph Platform

QUESTION TO ANSWER?

- What is a graph database?
- Why use a graph database?
- How do they work?
- What is the differenence with traditional Relation Database Management Systems
- Are they better than traditional RDBMS for graph based data?

WHAT? (ANGLES & GUTIERREZ, 2008)

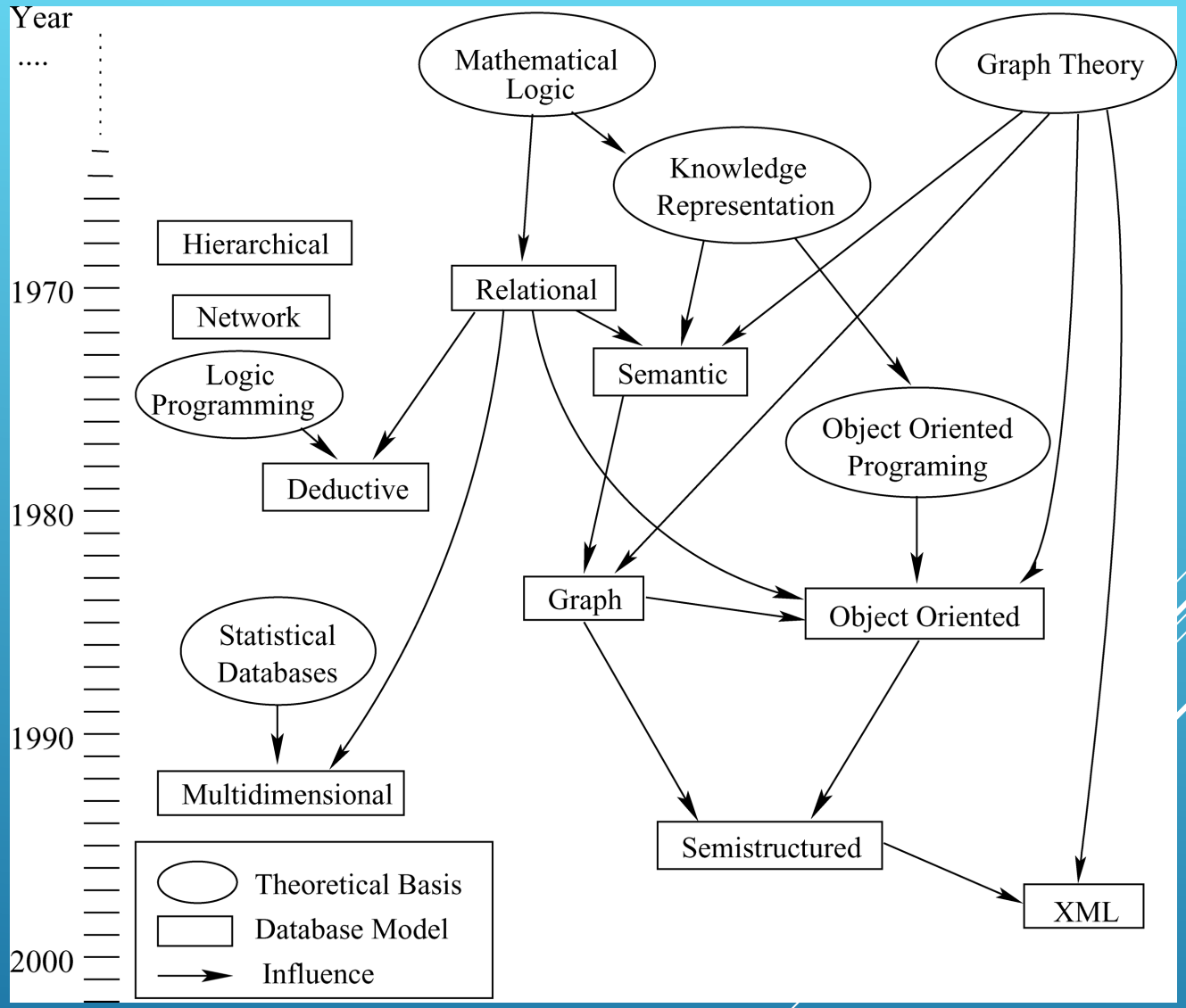
Three components describes a data model

- *Set of Data Structure Types.*
- *Set of Data Operations and Inference Rules.*
- *Set of Integrity Rules*.

However, several proposed db-models only define the data structure types.

There is a long evolution and diversity in db-models.

1. Strong emphasis on physical level
2. Relational: separation between physical and logical (Codd)
3. Semantic: clear and natural representation



WHAT? (ANGLES & GUTIERREZ, 2008)

WHAT?

Definition of(Angles & Gutierrez, 2008)

3 Basic characteristics

- *Data structure* type: (hyper)graph (directed, labelled), leaves and interconnections. Relationships are explicit
- *Data operations or manipulations*: expressed as graph operations: paths, neighbourhoods, sub-graphs, graph statistics (degree, centrality)
- *Integrity constraints*: Schema-Instance consistency, identity and referential integrity.

Definition (Angles & Gutierrez, 2008)

In summary, a graph db-model is a model in which the data structures for the schema and/or instances are modeled as a directed, possibly labeled, graph, or generalizations of the graph data structure, where data manipulation is expressed by graph-oriented operations and type constructors, and appropriate integrity constraints can be defined over the graph structure.

WHAT?

- adjacent(*G*, x, *y*): tests whether there is an edge from the vertices x to *y*;
- neighbors(*G*, x): lists all vertices *y* such that there is an edge from the vertices x to *y*;
- add_vertex(*G*, x): adds the vertex x, if it is not there;
- remove_vertex(*G*, x): removes the vertex x, if it is there;
- add_edge(*G*, x, *y*): adds the edge from the vertices x to *y*, if it is not there;
- remove_edge(*G*, x, *y*): removes the edge from the vertices x to *y*, if it is there;

Weighted Network
- get_edge_value(*G*, x, *y*): returns the value associated with the edge (x, *y*);
- set_edge_value(*G*, x, *y*, *v*): sets the value associated with the edge (x, *y*) to *v*.

Labeled Vertices
- get_vertex_value(*G*, x): returns the value associated with the vertex x;
- set_vertex_value(*G*, x, *v*): sets the value associated with the vertex x to *v*.

WHAT?

Information about the **data interconnectivity** or topology is more important or as important as the data itself.

Advantages:

- More natural modelling of the data which allows visualization of db-model
- Queries refer to the graph structure and allow high level of abstraction without complex query rewrites
- Browsing, graphical and visual interface

WHY?

WHY?

- Classical application
  - User wants to see **data connectivity**
  - Complexity exceeds capabilities of relation db-models
  - Need for **flexible knowledge representations**
  - Improvements in OO-models (database and programming)
- Complex networks (Newman, 2003; Albert & Barabasi 2002)
  - Social networks (eg Facebook, LinkedIn)
  - Information networks (Citation Network, Hypertext)
  - Technological networks (Power grid, Transportation)
  - Biological networks (Genomics, neural networks)

NAME	LASTNAME
George	Jones
Ana	Stone
Julia	Jones
James	Deville
David	Deville
Mary	Deville

PERSON	PARENT
Julia	George
Julia	Ana
David	James
David	Julia
Mary	James
Mary	Julia

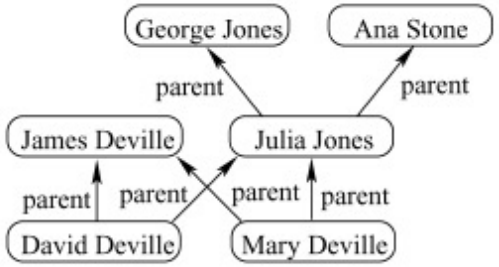
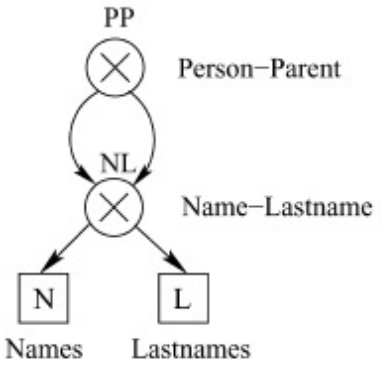


Fig. 5. A genealogy diagram (right-hand side) represented as two tables (left-hand side) NAME-LASTNAME and PERSON-PARENT (Children inherit the lastname of the father just for modeling purposes).

Schema



Instance

I (N)	
l	*val (l)*
1	George
2	Ana
3	Julia
4	James
5	David
6	Mary

I (L)	
l	*val (l)*
7	Jones
8	Stone
9	Deville

I (NL)	
l	*val (l)*
10	(1,7)
11	(2,8)
12	(3,7)
13	(4,9)
14	(5,9)
15	(6,9)

I (PP)	
l	*val (l)*
16	(12,10)
17	(12,11)
18	(14,13)
19	(14,12)
20	(15,13)
21	(15,12)

Fig. 6. Logical Data Model. The schema (on the left) uses two basic type nodes for representing data values (N and L), and two product type nodes (NL and PP) to establish relations among data values in a relational style. The instance (on the right) is a collection of tables, one for each node of the schema. Note that internal nodes use pointers (names) to make reference to basic and set data values defined by other nodes.

HOW? EXAMPLE

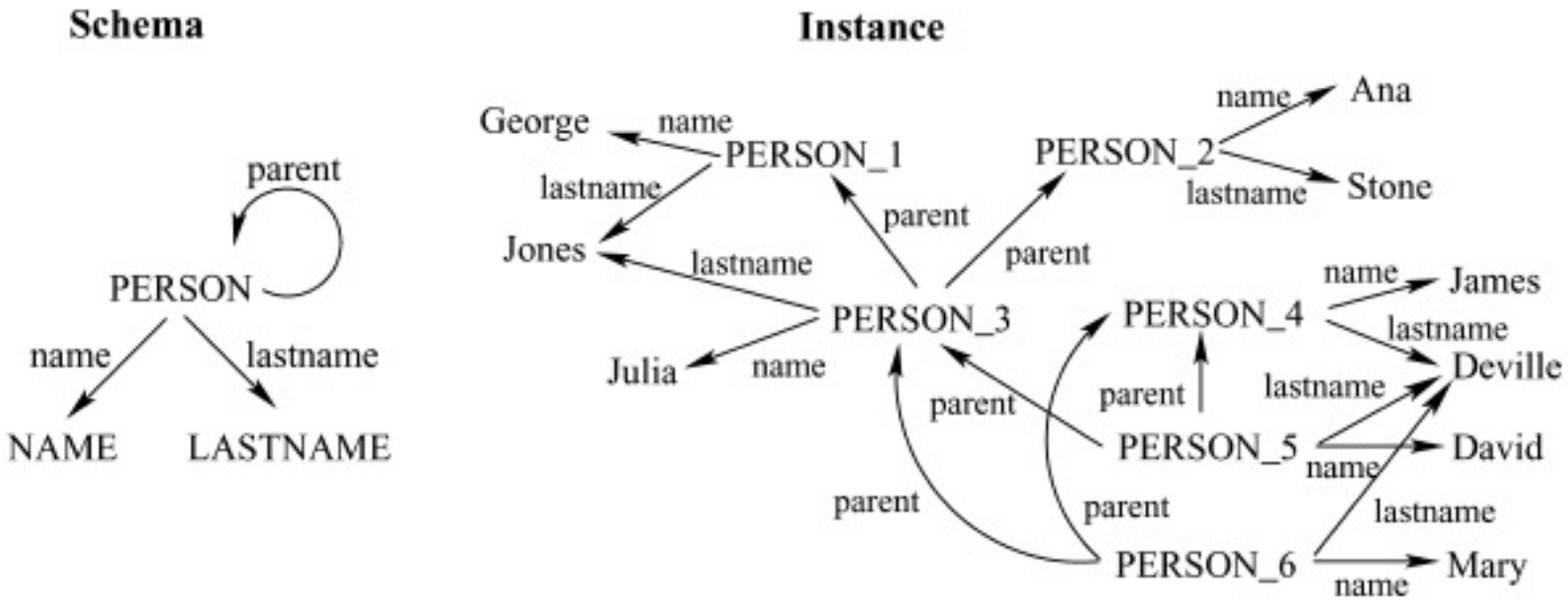


Fig. 16. Gram. At the schema level we use generalized names for definition of entities and relations. At the instance level, we create instance labels (e.g. PERSON_1) to represent entities, and use the edges (defined in the schema) to express relations between data and entities.

HOW? EXAMPLE

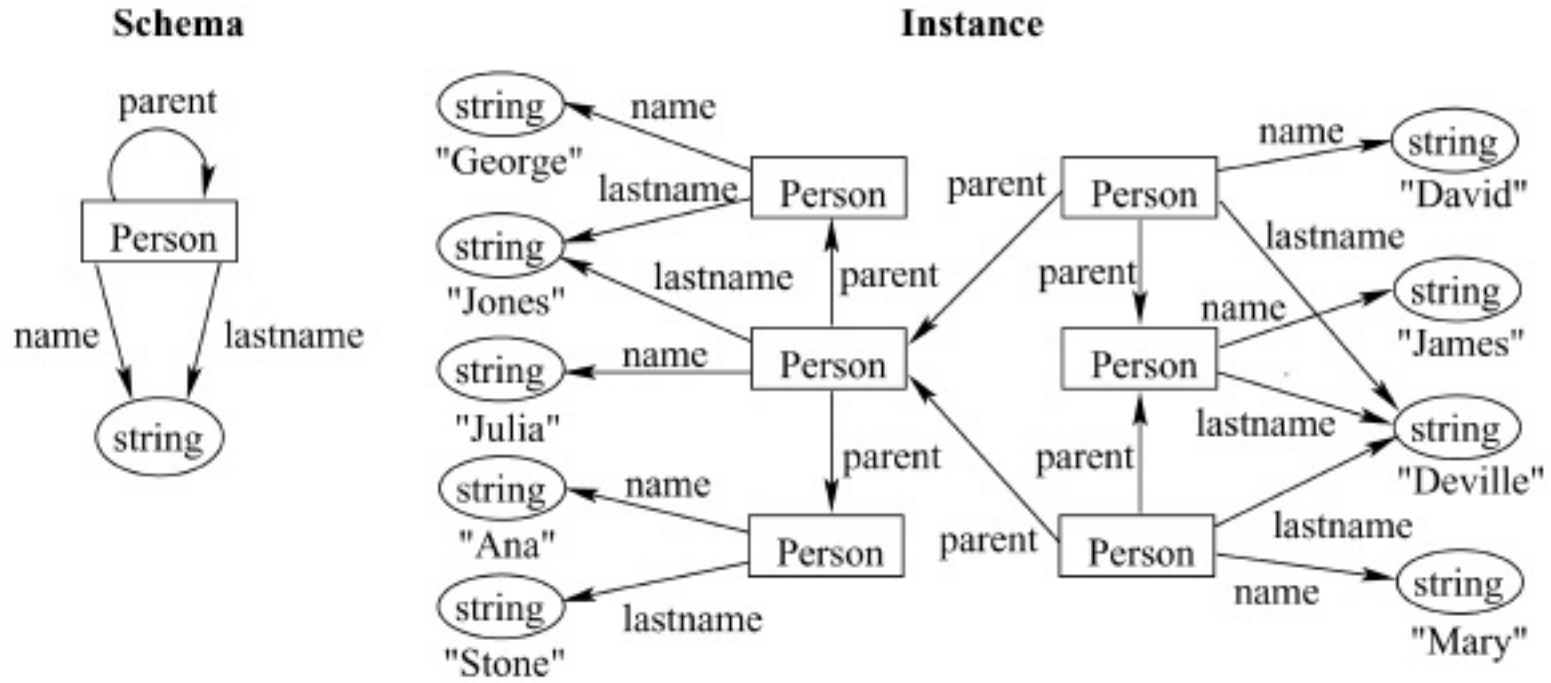


Fig. 12. GMOD. In the schema, nodes represent abstract objects (Person) and labeled edges establish relations with primitive objects (properties name and lastname), and other abstract objects (parent relation). For building an instance, we instantiate the schema for each person by assigning values to oval nodes.

HOW? EXAMPLE

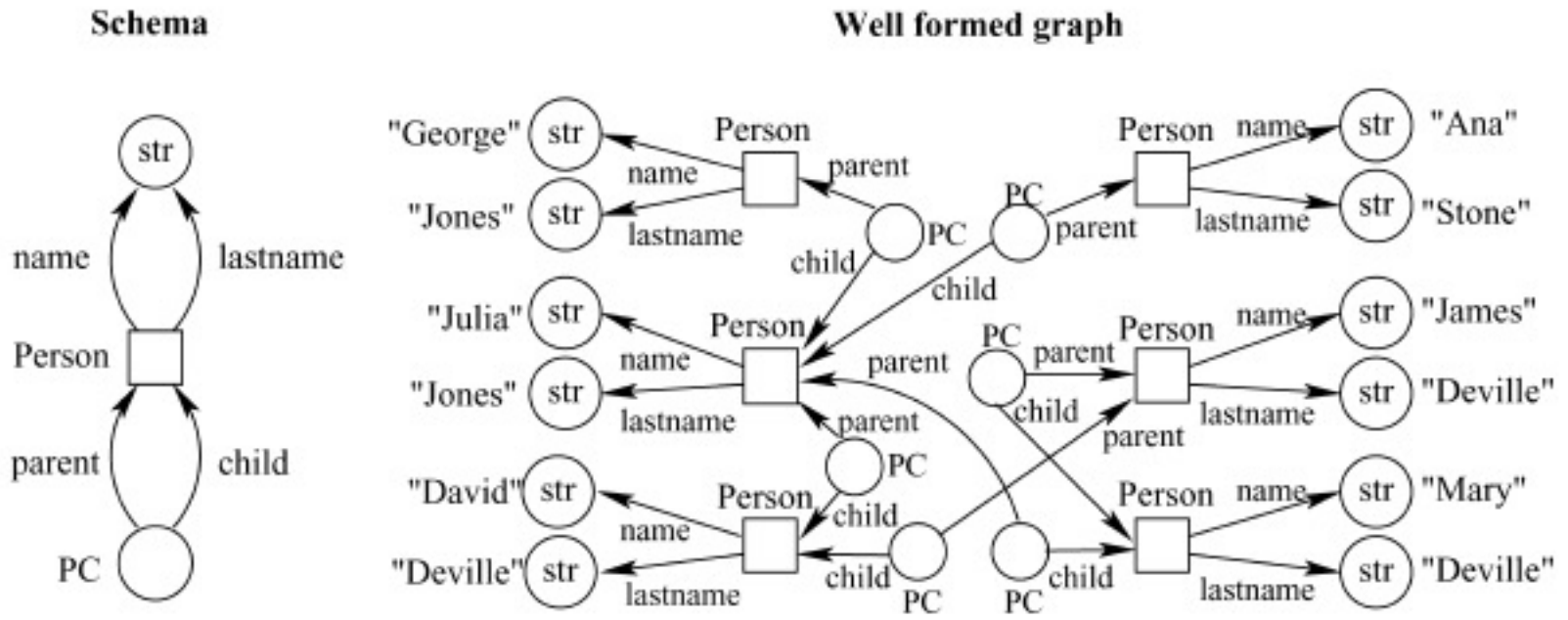


Fig. 15. GDM. In the schema each entity Person (object node represented as a square) has assigned the attributes *name* and *lastname* (basic value nodes represented round and labeled str). We use the composite-value node PC to establish the relationship Parent-Child. Note the redundancy introduced by the node PC. The instance is built by instantiating the schema for each person.

HOW? EXAMPLE

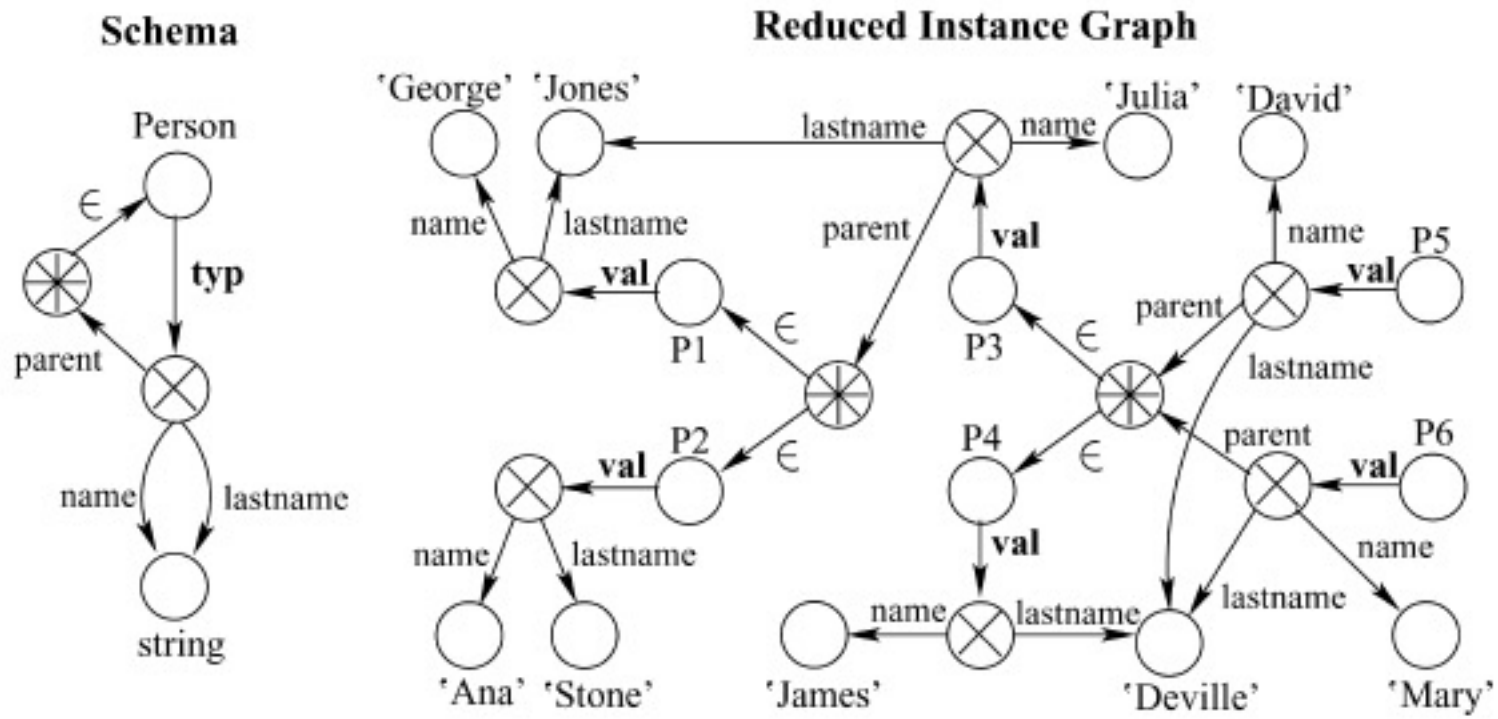


Fig. 13. PaMaL. The example shows all the nodes defined in PaMaL: basic type (string), class (Person), tuple (⊗), set (⊛) nodes for the schema level, and atomic (George, Ana, etc.), instance (*P1, P2*, etc), tuple and set nodes for the instance level. Note the use of edges ∈ to indicate elements in a set, and the edge **typ** to indicate the type of class Person (these edges are changed to **val** in the instance level).

HOW? EXAMPLE

Possible native implementation (not the only one):

Index-free adjacency

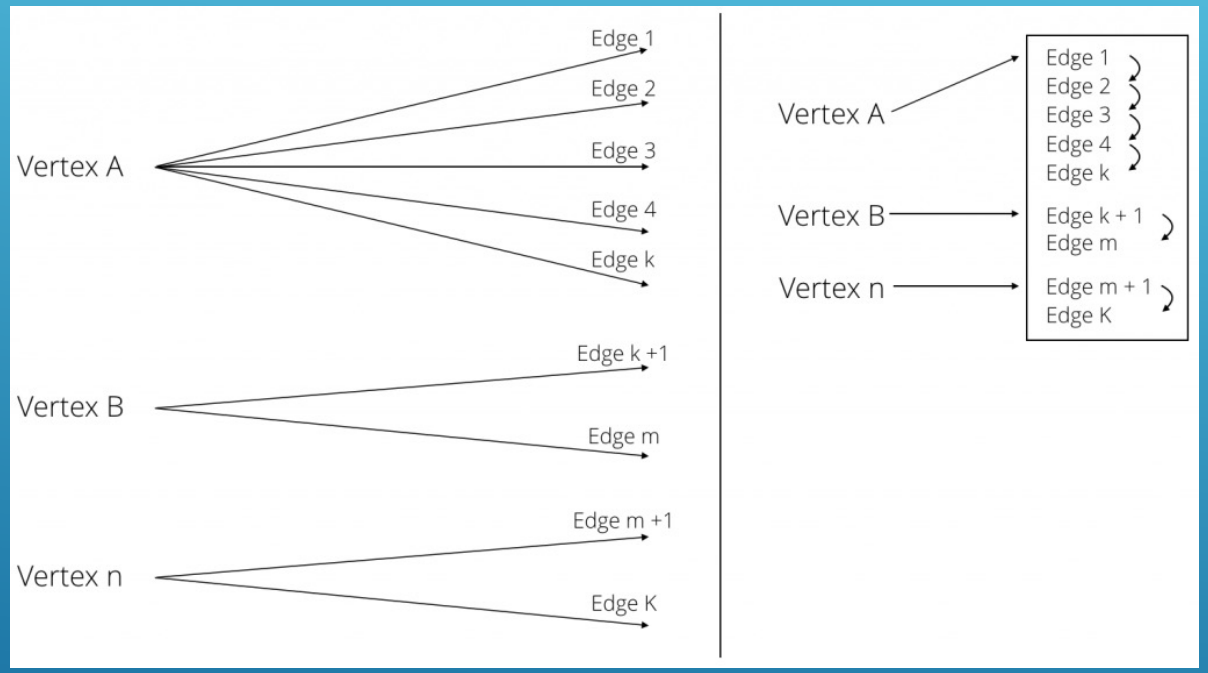
- Nodes holds pointers or references to its edges
- Traversal can be very fast
- Pointers need to be present at each end of edge
- Inserts can be fast (if…) and add no cost to traversals
- Presence of high-degree nodes makes other operations very costly
- Lower efficiency for non-graph operations

TAO's alternative: MySQL and MEMCache

HOW?

Hybrid implementation:

Hash Table and (Doubly) Linked List



https://www.arangodb.com/2016/04/index-free-adjacency-hybrid-indexes-graph-databases/

HOW?

Comparison to other databases

- Physical db-models (hierarchical / Network)
  - Lack abstraction level and not flexible but navigation at record level
- Relation db-models (Codd)
  - Abstraction level, modeling has mathematical foundation
- Semantic db-models
  - More expressive set of semantics from users viewpoint
- Object-Oriented db-models
  - Complex data objects related to OO-Programming (predefined)

DIFFERENCE?

ACID-compliance vs BASE model:

- **Atomic**: a set of operations or transactions are grouped and execution is completely successful or completely unsuccessful (rolled back)
- **Consistent**: After completion of transaction, the database is consistent with all constraints
- **Isolated**: Transactions are executed in isolation.
- **Durable**: Once executed, the outcome of the operations or transactions become permanent in the database across all instances

DIFFERENCE?

ACID-compliance vs **BASE model**:

- **Basic Availability**: The database is accessible most of the time without delay
- **Soft-State**: Stores don't have to be write-consistent, synchronization across instances is not guaranteed
- **Eventually consistent**: Lazy implementation of consistency, to be achieved at some point in the future

Availability having priority over consistency leads to improvements in scalability (Horizontal scaling)
=> Within partition consistency: clustering, community detecion

DIFFERENCE?

Retrieve data through specific query language:

- Cypher
  - Created by Neo4J
  - Based on their Property Graph Model

- Gremlin
  - Developed by Apache Tinkerpop
  - Supported by many vendors

DIFFERENCE?

Three papers:

Vicknair, 2010; Welc, 2013 & Khan 2018;

1. What is the main conclusion of the paper?
2. What are the properties of the networks used in the comparison? Can you think about other properties that were not described?
3. What are the criteria used in the paper to come to the conclusion?
4. What is your critical opinion about this paper?

ARE THEY BETTER?

Discussion in three groups.

Each group discusses one paper.

The answers are added to the discussion forum and presented in the general session after the discussion.

TASK