What's the big deal about big data?
NoSQL, Graphs, and Linked Data

Lecture 10
2ID35, Spring 2015

George Fletcher

Faculteit Wiskunde & Informatica
Technische Universiteit Eindhoven

5 June 2015

Agenda

1. Introduction and overview of linked and graph data, in the context of related research in database technology, and in general, data science

2. Models for graph and linked data
   - RDF and RDFS data model
   - graph database models

3. Introduction to querying over graph and linked data.
   - query languages for graphs
   - SPARQL 1.1
   - models for linked data
   - querying linked data

What's the big deal about big data?

Big Data & Data Science

Our ability to generate and capture data only continues to explode

Indeed, we live in a world that is

- increasingly driven by this generation and consumption of massive ever-growing data sets,
- which in turn is driven by both technology and a rapidly increasing "datafication" of all aspects of our public and private lives, and well, of most everything.

Managing this Big Data is now central to commerce, entertainment, government, education, science, ...

... and, is a core research issue in the emerging discipline of Data Science.

Big Data & Data Science

Due to datafication and technology trends, applications now must face data scale, heterogeneity, and distribution as the norm, rather than as exceptions, as in the past

- often captured as the four V's of "volume", "variety", "veracity", and "velocity"

Big Data & Data Science

Due to datafication and technology trends, applications now must face data scale, heterogeneity, and distribution as the norm, rather than as exceptions, as in the past

- often captured as the four V's of "volume", "variety", "veracity", and "velocity"

Furthermore, there is increased potential for extracting significant untapped value from data

- e.g., the successes of Google and Facebook are essentially built on extracting value from data

Big Data: NoSQL systems

However, there is increased difficulty in extracting "value" (a fifth V), due to the other four V's

In data management, embracing and confronting these challenges has led to an explosion of NoSQL systems, as alternatives to the dominant paradigm of structured data (i.e., "SQL" stores)

Big Data: NoSQL systems

Serious relaxations in "traditional" data management assumptions include

- uncertain data
- structural heterogeneity
  - semi-structured, denormalized data
  - query paradigms for such loosely structured data
- looser data consistency
- semantic heterogeneity

NoSQL systems typically embrace one or more of these

Big Data: NoSQL systems

Serious relaxations in "traditional" data management assumptions include

- uncertain data
- structural heterogeneity
  - semi-structured, denormalized data
  - query paradigms for such loosely structured data
- looser data consistency
- semantic heterogeneity

NoSQL systems typically embrace one or more of these

Note, though, that many of these relaxations predate NoSQL and even "SQL" systems (e.g., CODASYL and the network data model) ...

Big Data: NoSQL systems

Let's briefly look at relaxing consistency in distributed data management

ACID: the gold-standard for consistent data management

- updates to data should be atomic (A)
- transactions (i.e., blocks of read/write work) preserve all data consistency (C) rules, including having a single up-to-date copy of the data
- transactions should be performed in isolation (I) of one another
- updates should be durable (D), i.e., hard state, i.e., persistent

Big Data: NoSQL systems

Let's briefly look at relaxing consistency in distributed data management

ACID: the gold-standard for consistent data management

- updates to data should be atomic (A)
- transactions (i.e., blocks of read/write work) preserve all data consistency (C) rules, including having a single up-to-date copy of the data
- transactions should be performed in isolation (I) of one another
- updates should be durable (D), i.e., hard state, i.e., persistent

can be costly to enforce in a distributed context (i.e., using standard techniques, can lead to high latency)

Relaxed consistency

In the context of Big Data, data is often distributed and replicated, and the focus is on the *availability* of the data (i.e., low latency)

Hence, NoSQL systems often give no consistency guarantees, or only the guarantee of "eventual" consistency when transactions span nodes

- also know as BASE: Basically Available Soft-state services with Eventual-consistency

Justification for relaxed consistency

CAP Theorem. any networked shared-data system can have at most two of three desirable properties

- consistency (C) equivalent to having a single up-to-date copy of the data;
- high availability (A) of that data; and,
- tolerance to network partitions (P).

(Brewer, PODC 2000; Brewer, Computer, Feb 2012; Gilbert and Lynch, SIGACT News 2002)

Justification for relaxed consistency

CAP Theorem. any networked shared-data system can have at most two of three desirable properties

- consistency (C) equivalent to having a single up-to-date copy of the data;
- high availability (A) of that data; and,
- tolerance to network partitions (P).

(Brewer, PODC 2000; Brewer, Computer, Feb 2012; Gilbert and Lynch, SIGACT News 2002)

note that CAP only talks about partitioned systems, and not normal operation!

By explicitly handling partitions, designers can optimize consistency and availability, thereby achieving some tradeoff of all three.

Shades of relaxed consistency

PACELC. In case of partitions (P) does the system choose availability (A) or consistency (C)?; else (E), does it choose lower latency (L) or consistency (C)?

Shades of relaxed consistency

PACELC. In case of partitions (P) does the system choose availability (A) or consistency (C)?; else (E), does it choose lower latency (L) or consistency (C)?

- PA/EL: Dynamo (Amazon's key-value store), Cassandra (Apache's key-value store), Riak (key-value store)
- PA/EC: MongoDB (key-value/JSON store)
- PC/EL: PNUTS (Yahoo's key-value store)
- PC/EC (i.e., full ACID): VoltDB/H-Store (shared-nothing relational store)

(Abadi, Computer, Feb 2012)

Remarks

Consistency is not a binary choice, but rather lies on a continuum, based on context.

In practice, completely eliminating ACID consistency during normal operation is almost never justified by the CAP theorem.

Remarks

Consistency is not a binary choice, but rather lies on a continuum, based on context.

In practice, completely eliminating ACID consistency during normal operation is almost never justified by the CAP theorem.

For further discussion, see

- Lloyd et al. SOSP 2011
- Thomson et al. SIGMOD 2012
- Birman et al. Computer, Feb 2012

Big Data: NoSQL systems

Let's now consider data structural heterogeneity, driven by the need for so-called "polyglot persistence"

Much modern data doesn't cleanly map to a tight relational structure

- missing or multi-valued attributes
  - e.g., not all site visitors have exactly one phone number
- nested/hierarchical structure
  - ontologies, folksonomies
  - JSON, XML
- loose structure
  - social networks
  - chem-/bio- networks

Big Data: NoSQL systems

Four broad classes of approaches taken by "NoSQL" systems:

1. **key-value databases** (essentially scalable hash tables)
   - example systems: BerkelyDB, Tokyo Cabinet
2. **document databases** (generalization of key-value model to include nested structure, e.g., JSON and XML)
   - example systems: MongoDB, CouchDB, Couchbase
3. **column-family stores** (hybrid of key-value and relational model, where rows can have different schemas)
   - example systems: Cassandra, Amazon SimpleDB
4. **graph databases**
   - example systems: Neo4j, IBM DB2 RDF GraphStore

Big Data: NoSQL systems

Four broad classes of approaches taken by "NoSQL" systems:

1. **key-value databases** (essentially scalable hash tables)
   - example systems: BerkelyDB, Tokyo Cabinet
2. **document databases** (generalization of key-value model to include nested structure, e.g., JSON and XML)
   - example systems: MongoDB, CouchDB, Couchbase
3. **column-family stores** (hybrid of key-value and relational model, where rows can have different schemas)
   - example systems: Cassandra, Amazon SimpleDB
4. **graph databases**
   - example systems: Neo4j, IBM DB2 RDF GraphStore

All of these are firmly rooted in applications arising in web data management ...

Web Data

The explosion of the Web was both a precursor to and accelerant for the rise of Big Data.

Historically, web data has been modeled as

- hypertext and SGML
- text and HTML
- XML and JSON
- ...

Primarily focusing on the metaphor of the "document" ...

Web Data

The explosion of the Web was both a precursor to and accelerant for the rise of Big Data.

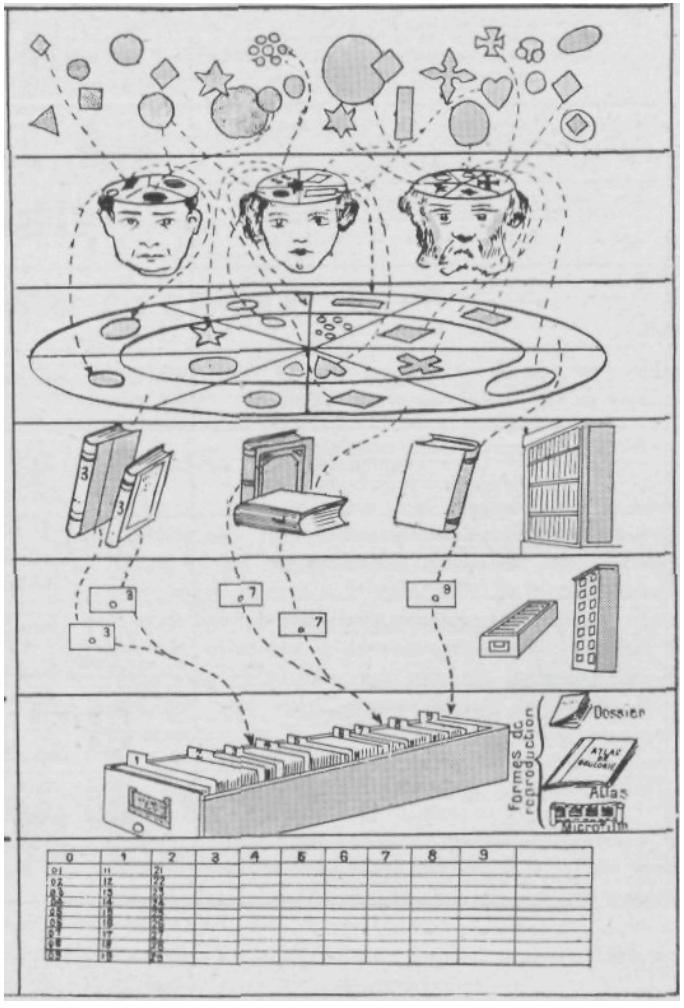
Historically, web data has been modeled as

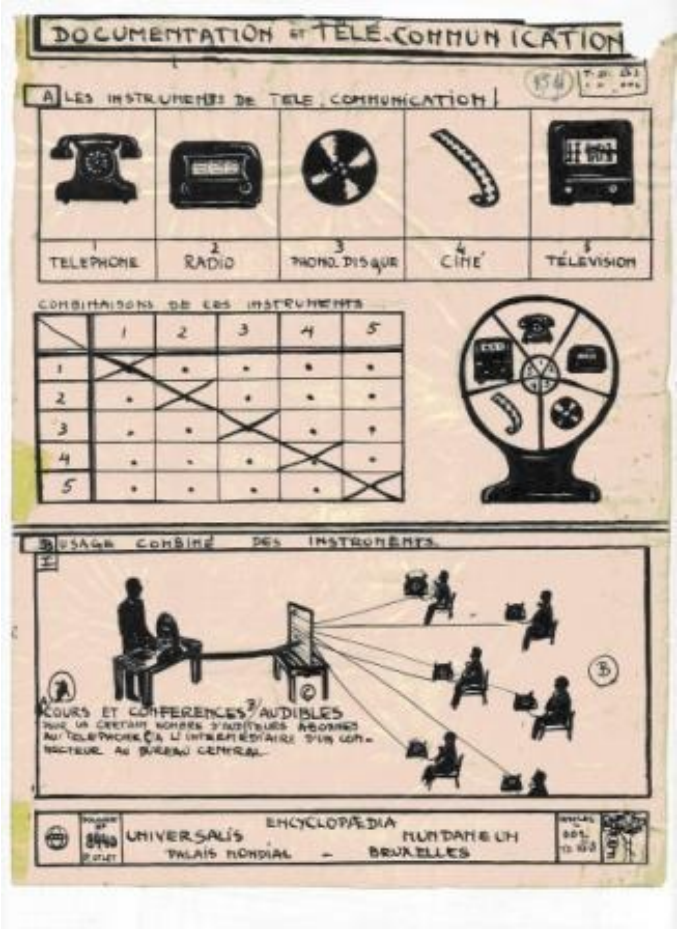
- hypertext and SGML
- text and HTML
- XML and JSON
- ...

Primarily focusing on the metaphor of the "document" ...

But early "web" scientists had even broader visions for human knowledge ...

Web Data





The Mundaneum

as conceived by the Belgian author and peace activist Paul Otlet (1868-1944)

Web Data: linked data

This vision has resurfaced recently in the Linked Data initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Web Data: linked data

This vision has resurfaced recently in the Linked Data initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Linked data principles:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things

e.g., BBC, NYTimes, Wikipedia, Uniprot, PubMed, DBLP, ...

Web Data: linked data

This vision has resurfaced recently in the Linked Data initiative, which is essentially a vision for modeling and sharing graph data on the web, using web standards

Linked data principles:

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things

e.g., BBC, NYTimes, Wikipedia, Uniprot, PubMed, DBLP, ...

Linked *open* data: public sector data as linked data

- e.g, dati.gov.it, data.gov.uk, data.gouv.fr, data.overheid.nl, data.gov, data.eu, datameti.go.jp, ...

Our focus today

We study the intersection of big data and web data, namely linked data and graph data

Our focus will be on linked and graph data management (and not information retrieval, although the distinction is increasingly fuzzy)

- i.e., scaleably implementing variations of first order logic and their application

Specifically, we will consider the challenges of modeling and querying linked and graph data

References & Credits, 1/2

- The end of theory: the data deluge makes the scientific method obsolete. Chris Anderson. *Wired Magazine* 16(7), 2008.

  `http://www.wired.com/science/discoveries/magazine/16-07/pb_theory`

- The second economy. W. Brian Arthur. *McKinsey Quarterly*, 2011.

  `http://www.mckinsey.com/insights/strategy/the_second_economy`

- The rise of big data. Kenneth Cukier and Viktor Mayer-Schoenberger. *Foreign Affairs*, May/June 2013.

  `http://www.foreignaffairs.com/articles/139104/kenneth-neil-cukier-and-viktor-mayer-schoenberger/the-rise-of-big-data`

- The unreasonable effectiveness of data. Alon Halevy, Peter Norvig, and Fernando Pereira. *IEEE Intelligent Systems*, March/April 2009.

  `http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/35179.pdf`

References & Credits, 2/2

- *Towards a Big Data reference architecture.* Markus Maier. MSc thesis, Eindhoven University of Technology, 2013.

  http://www.win.tue.nl/~gfletche/Maier_MSc_thesis.pdf

- *Frontiers in massive data analysis.* National Research Council of the National Academies. Washington, D.C., 2013.

  http://www.nap.edu/catalog.php?record_id=18374 (register for free PDF)

- *On being a data skeptic.* Cathy O'Neil. O'Reilly Media, 2013.

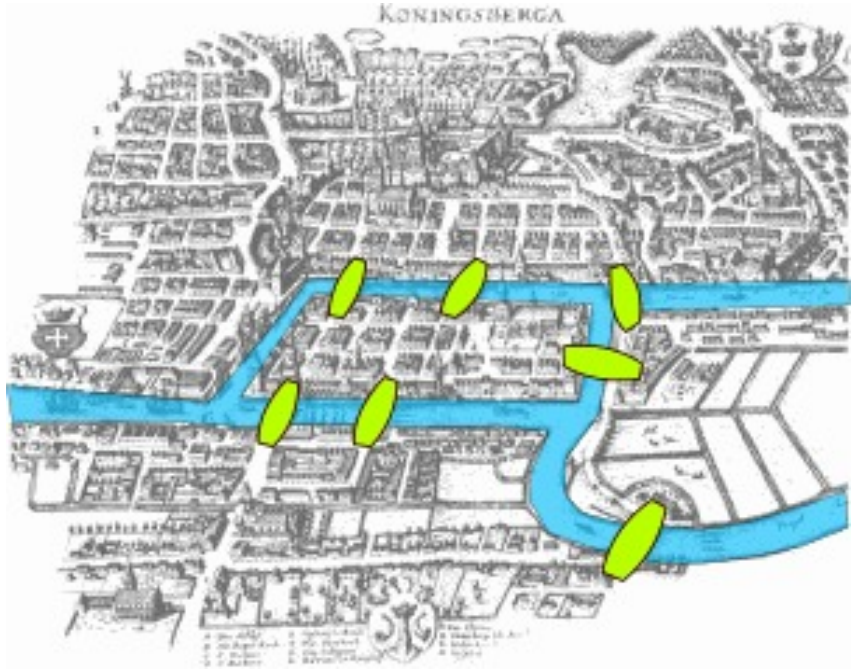
  http://cdn.oreillystatic.com/oreilly/radarreport/0636920032328/On_Being_a_Data_Skeptic.pdf

- *NoSQL distilled: a brief guide to the emerging world of polyglot persistence.* Pramod J. Sadalage and Martin Fowler. Addison-Wesley, 2013. http://martinfowler.com/nosql.html

graph data

Graphs (aka, Networks)

Euler (1735): the seven bridges of Königsberg (Kaliningrad)

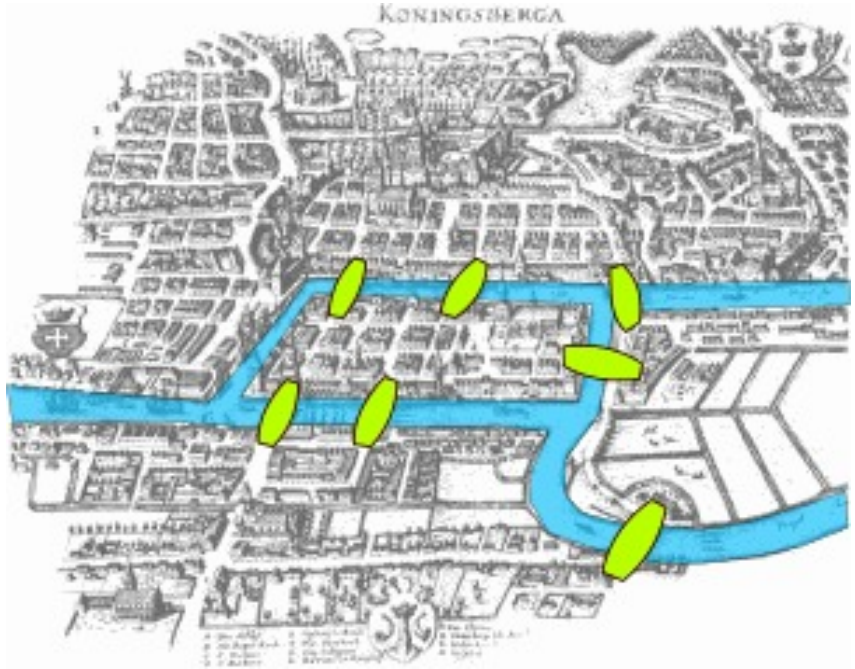


(wikipedia.org)

Can you walk through the city using each bridge exactly once?

Graphs (aka, Networks)

Euler (1735): the seven bridges of Königsberg (Kaliningrad)

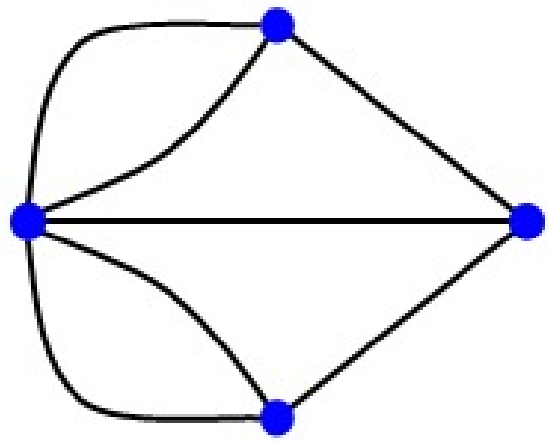


(wikipedia.org)

Can you walk through the city using each bridge exactly once?
Euler found that a solution does not exist. To prove this, he invented graph theory.

Graphs

Euler (1735): the seven bridges of Königsberg (Kaliningrad)



(wikipedia.org)

Graphs

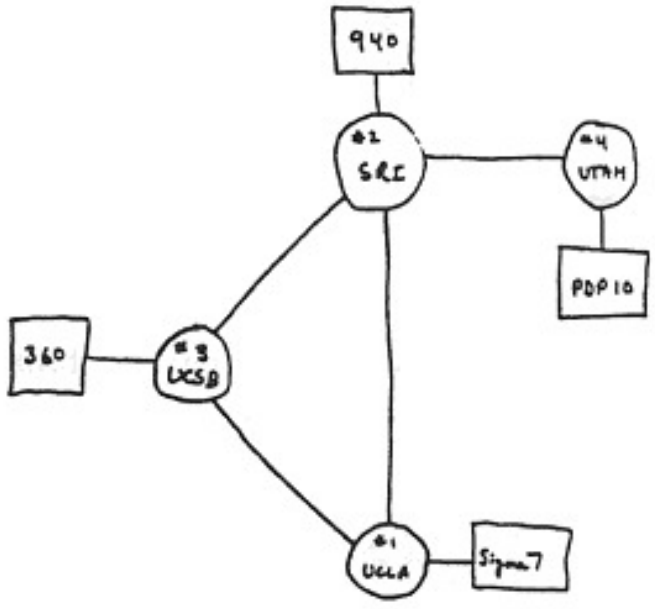
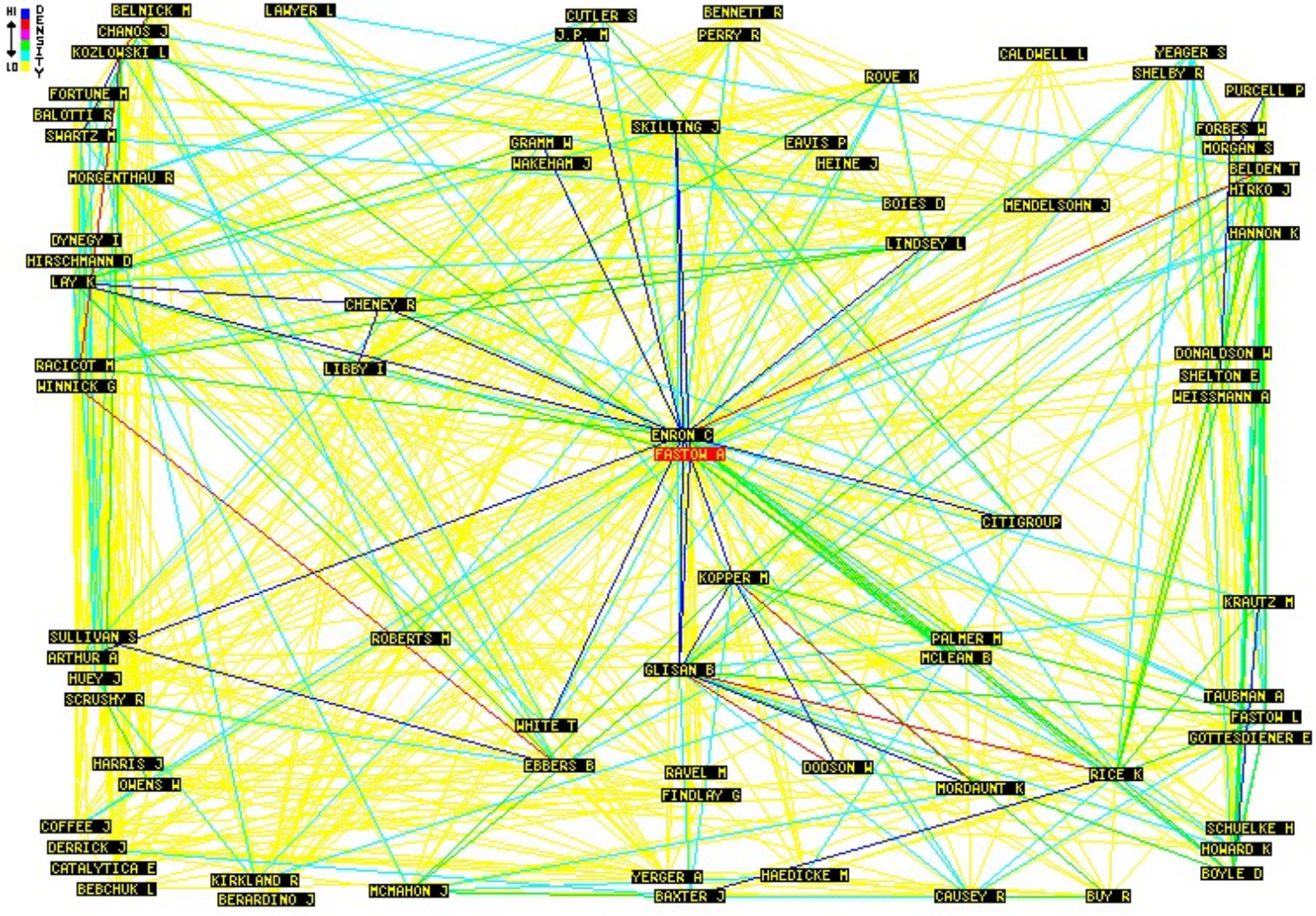


FIGURE 6.2 Drawing of 4 Node Network
(Courtesy of Alex McKenzie)

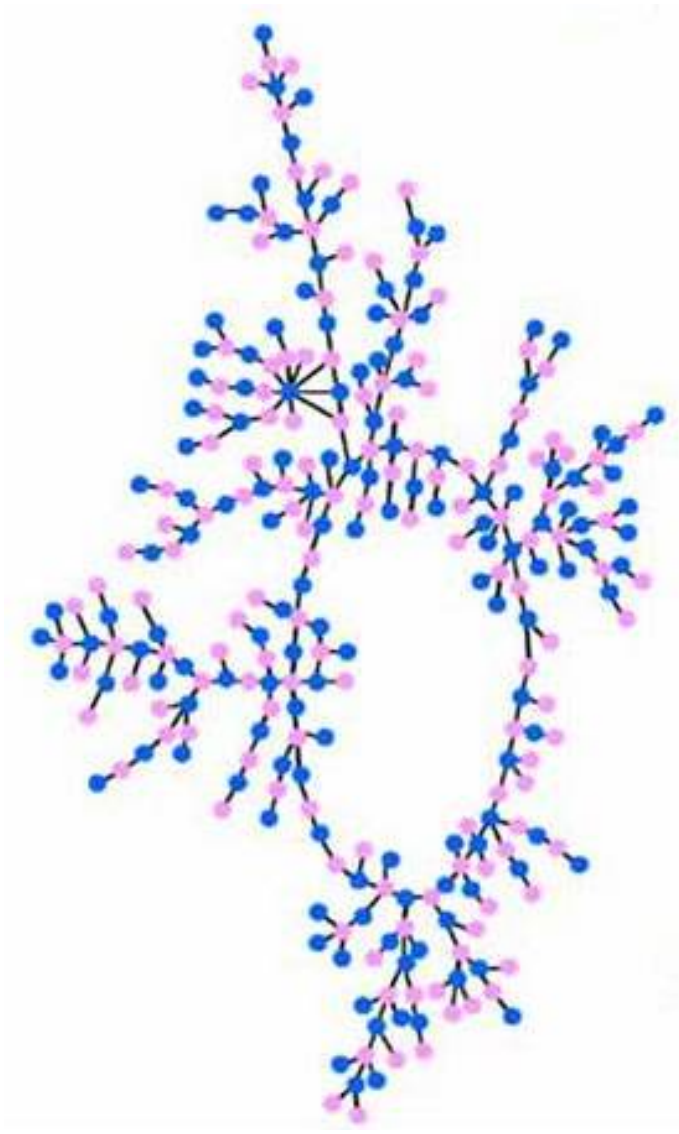
the Internet circa 1969 (Kearns, UPenn)

Graphs



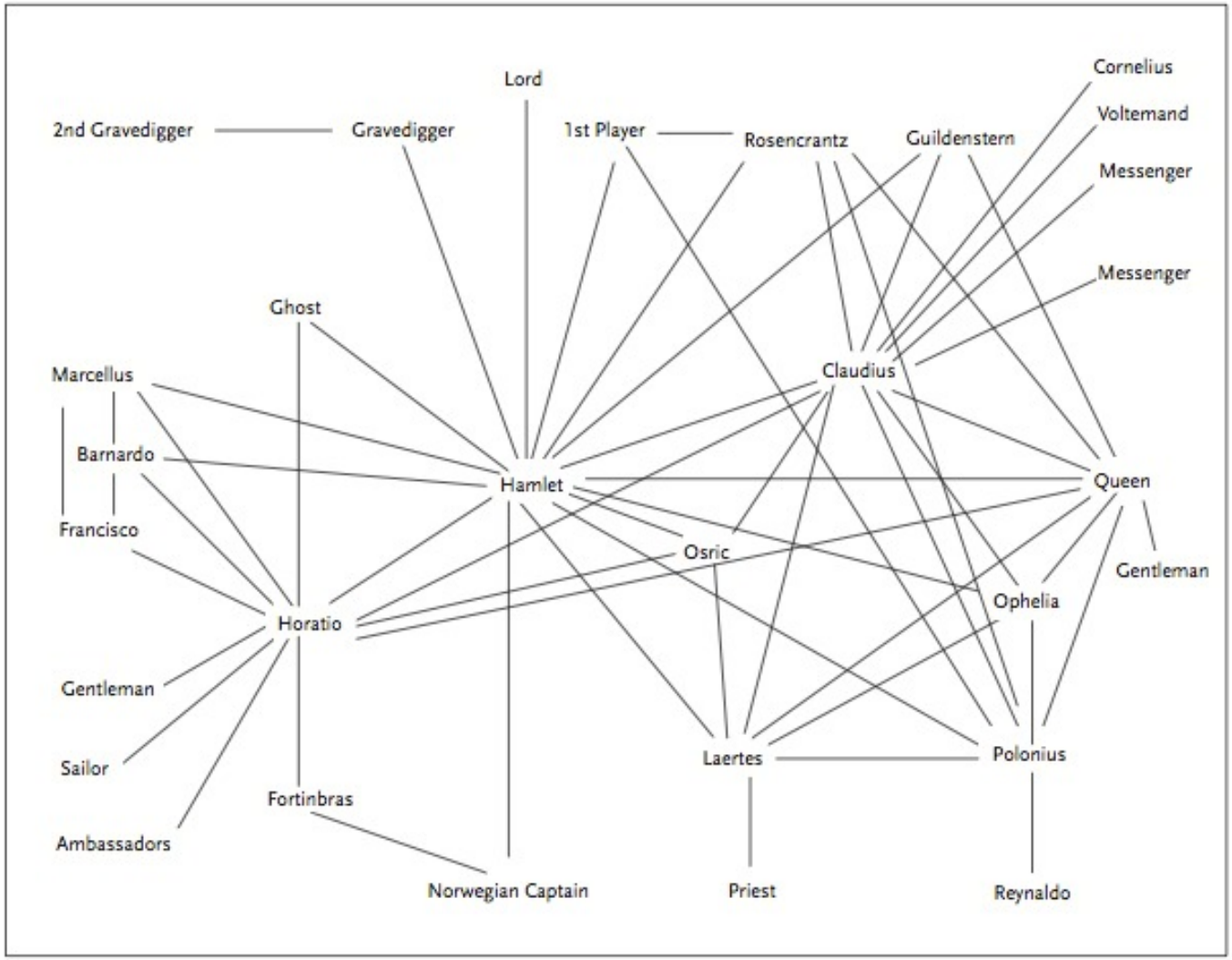
the Enron email network (Kearns, UPenn)

Graphs



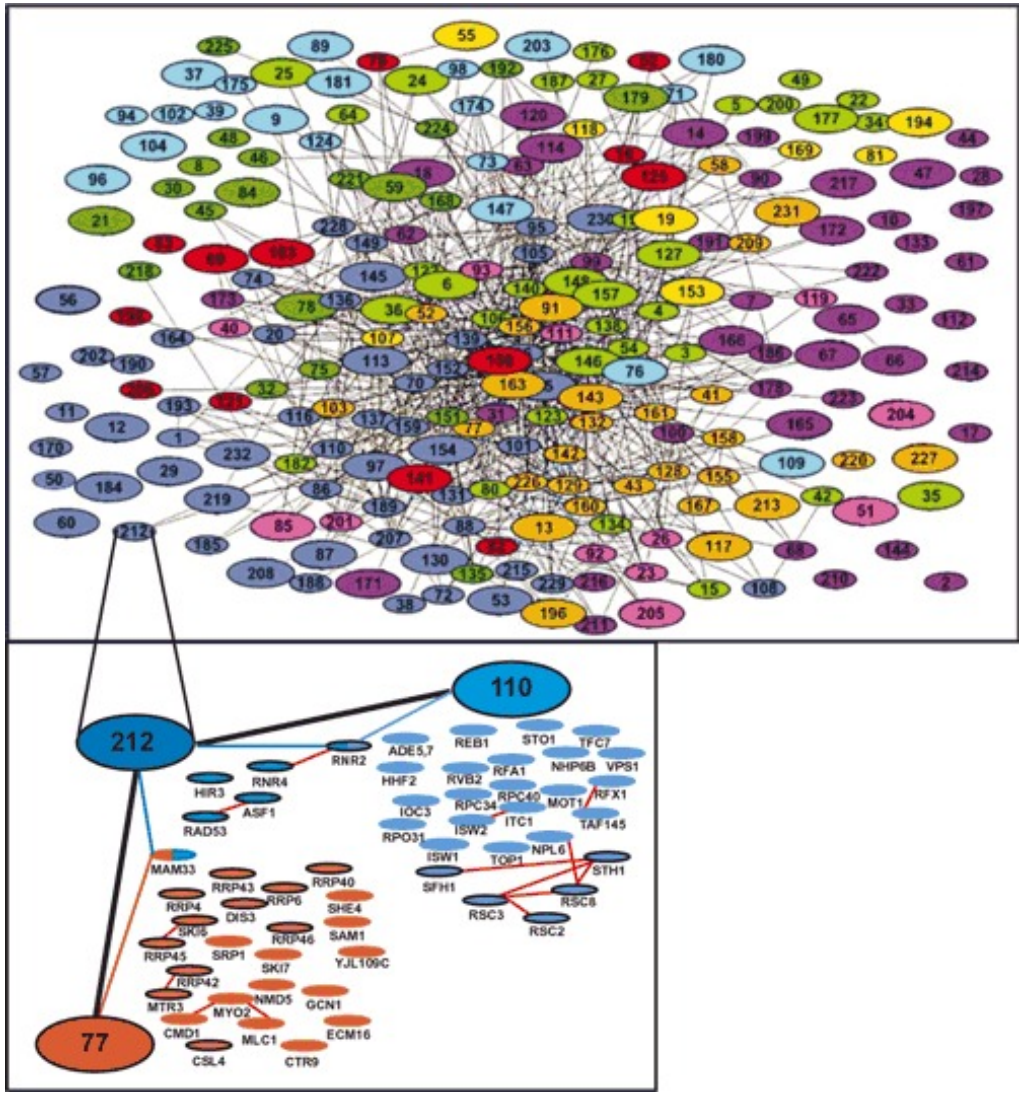
a highschool boyfriend-girlfriend network (Kearns, UPenn)

Graphs



the Hamlet social network (crookedtimber.org)

Graphs



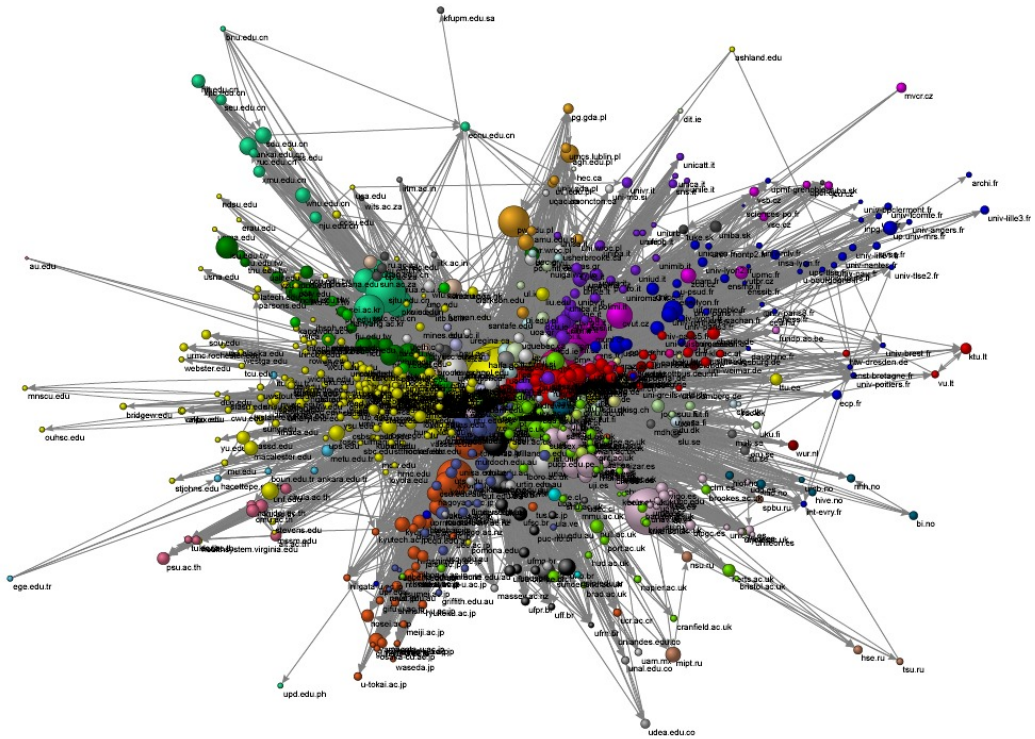
a yeast protein network (genomenewsnetwork.org)

Web as a graph

The web as an object of study: a global macro-structure perspective on the web

- resources as nodes
- edges between nodes which link to each other

in contrast to the local perspective of one resource/document



(Ortega and Aguillo 2009)

Web as a graph

The web as an object of study

- the study of the structure of web and online social networks is a major theme of data/web science, e.g.,
  - connectivity and path length
  - degree of nodes

Web as a graph

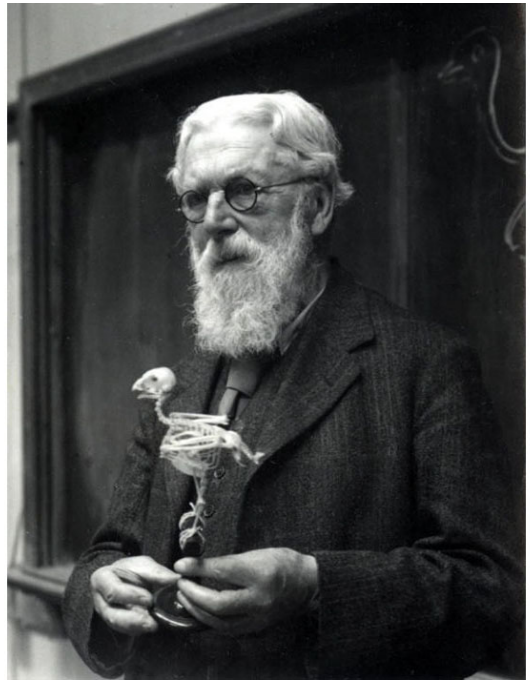
The web as an object of study

- the study of the structure of web and online social networks is a major theme of data/web science, e.g.,
  - connectivity and path length
  - degree of nodes
- also, the study of processes on this structure is a major theme, e.g.,
  - searching
  - ranking, popularity
  - information dissemination
  - protection against destruction, attack
  - linked data query processing and optimization

Graphs

Many, many applications of graph theory

- physical, biological, social, chemical
- communication (e.g., phone)
- data/relationships (e.g., XML)
- travel, transportation
- chip design
- linguistics
- mathematics
- ...

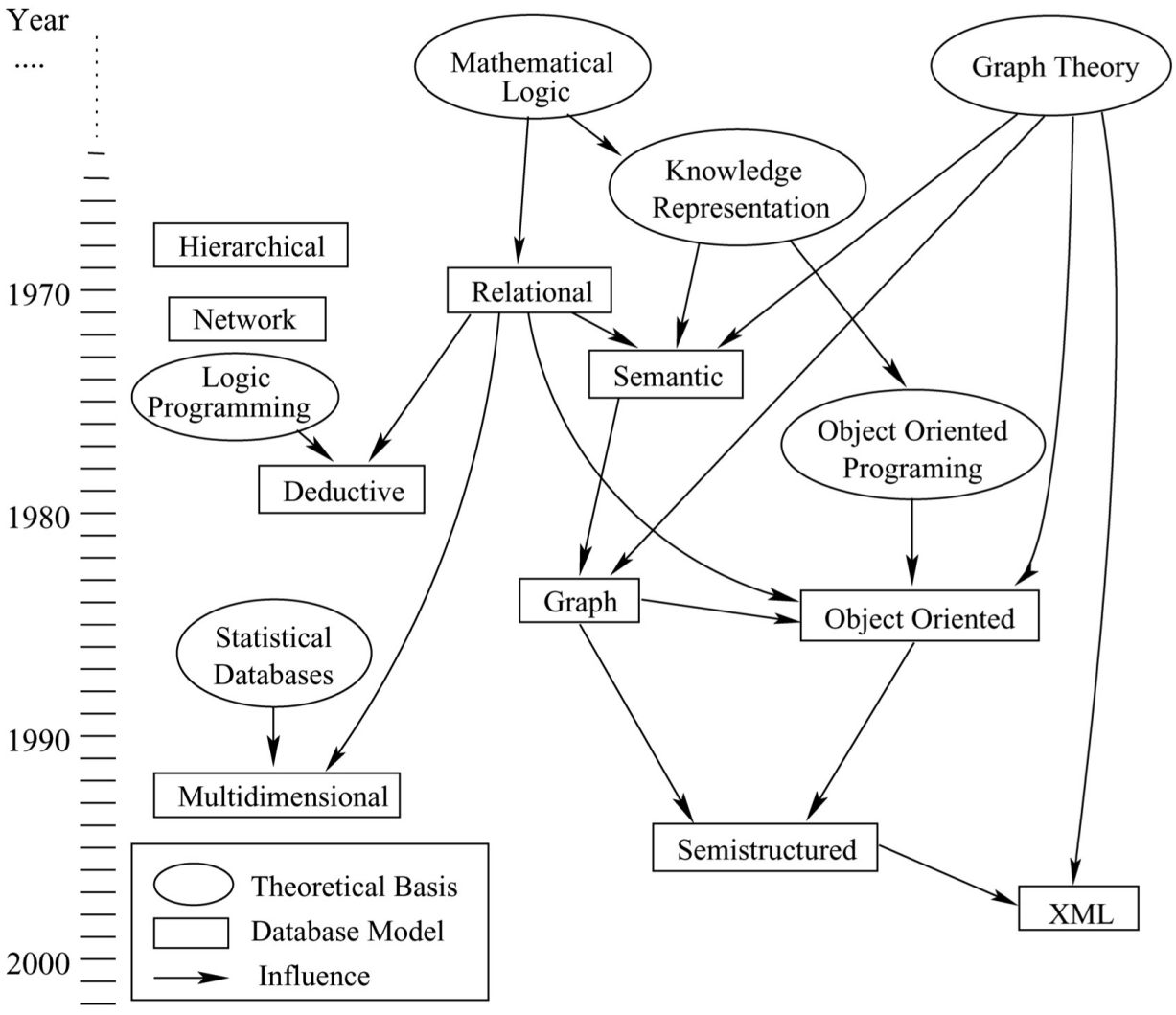


Things are interesting only in so far as they relate themselves to other things; only then can you put two and two together and tell stories about them. Such is science itself and such is all the knowledge that interests mankind.

– D'Arcy Wentworth Thompson (1860-1948)

graph database models

Historical context: database models



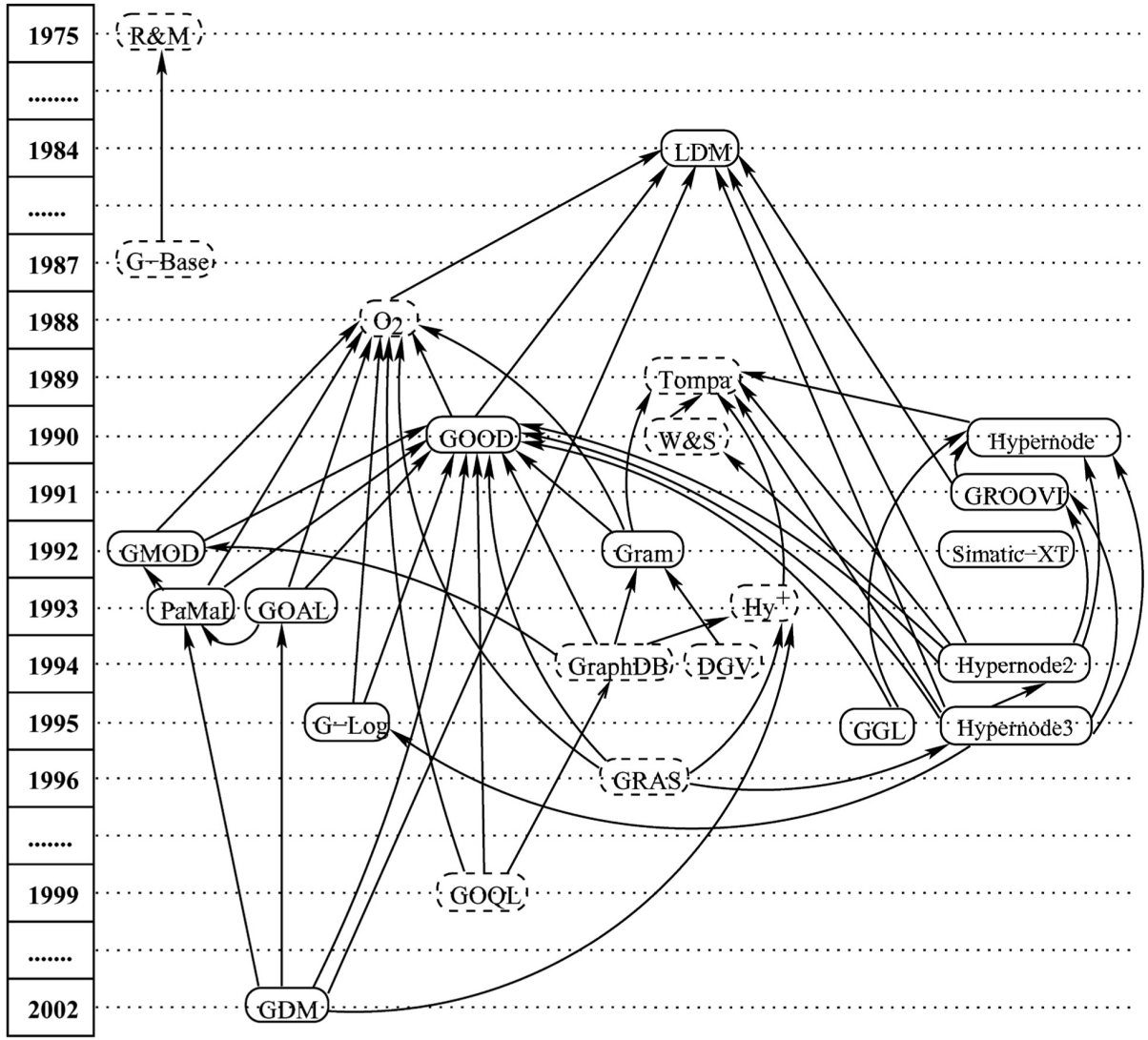
(We follow Angles and Gutierrez 2008 in this section)

What makes a database a graph database?

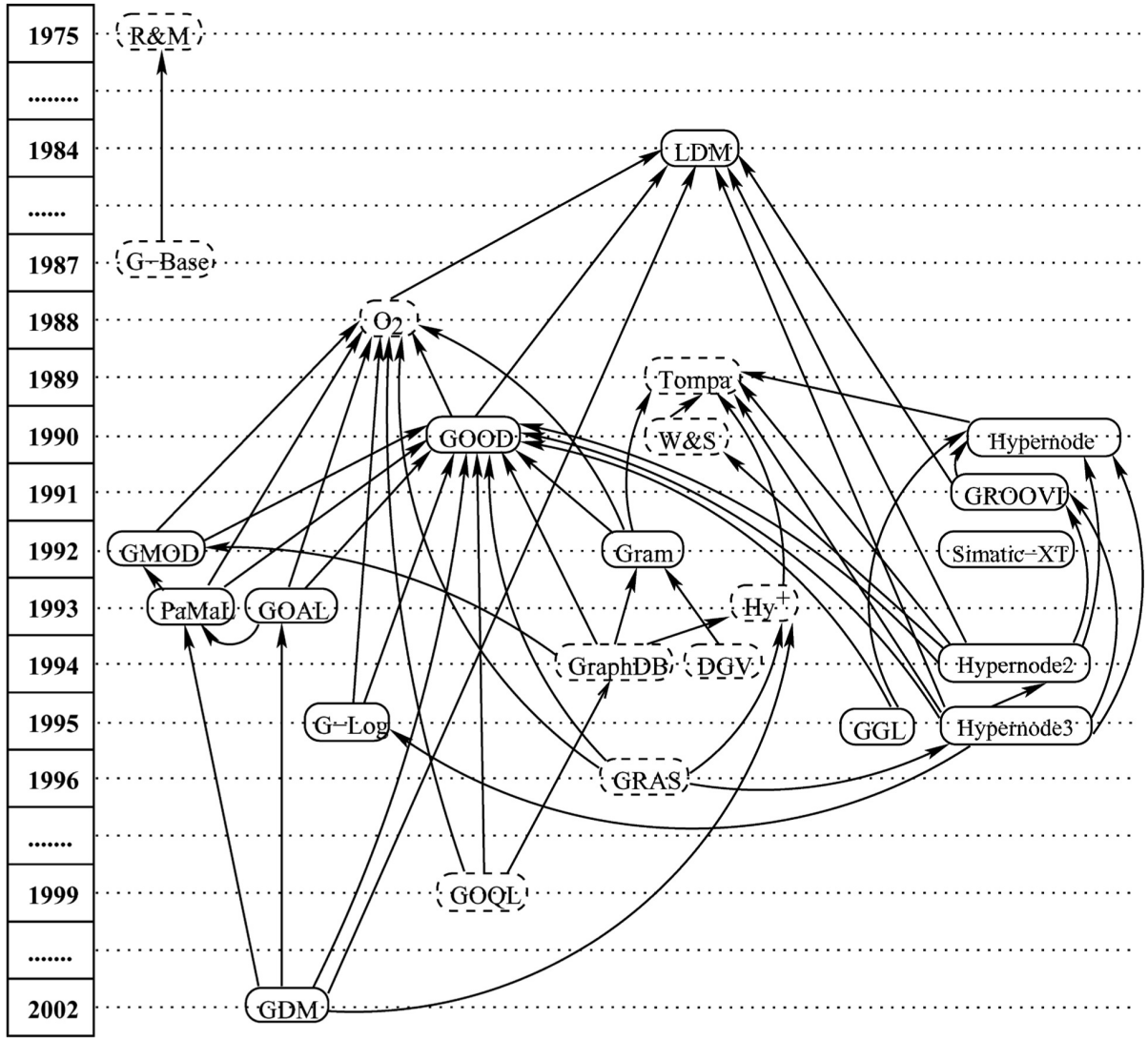
We can loosely identify three characteristics

1. Data, schemas, queries, and/or updates are represented by graphs, or their generalization.
2. Data manipulation is by graph transformations or operations on graph features such as paths, neighborhoods, diameter, etc.
3. There are possibly facilities for expressing appropriate integrity constraints such as label typing and domain/range restrictions.

Historical context: graph database models



Historical context: graph database models



As a representative sample, let's consider the thread *LDM* $\leftarrow$ *GOOD* $\leftarrow$ *GDM*

One thread: the Logical Data Model

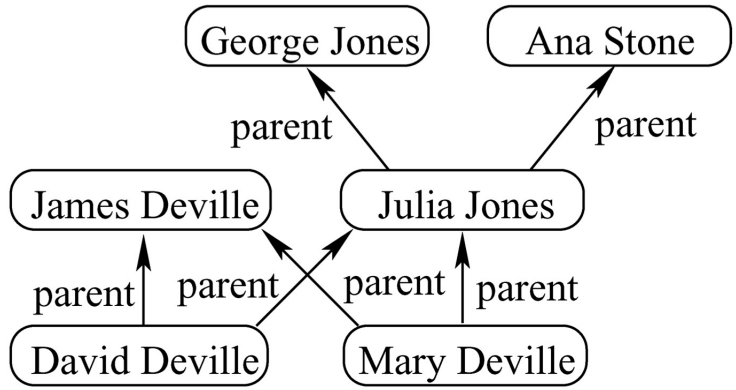
LDM. (Kuper and Vardi, 1984)

- unifies relational, hierarchical, and network models
- provides constructs for physical addressing
- schemas and instances are node-labeled graphs
- three node types: basic (terminal nodes with atomic data), composition (tuples built from children), and collection (sets with elements taken from child)
- algebraic and logical querying

A graph ...

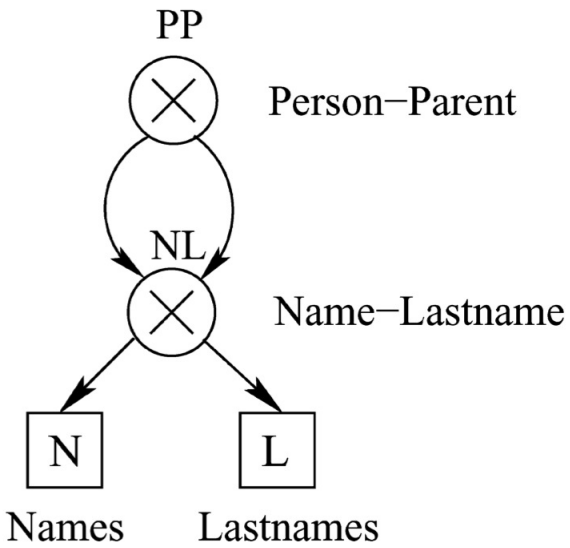
NAME	LASTNAME
George	Jones
Ana	Stone
Julia	Jones
James	Deville
David	Deville
Mary	Deville

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



One thread: LDM

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)

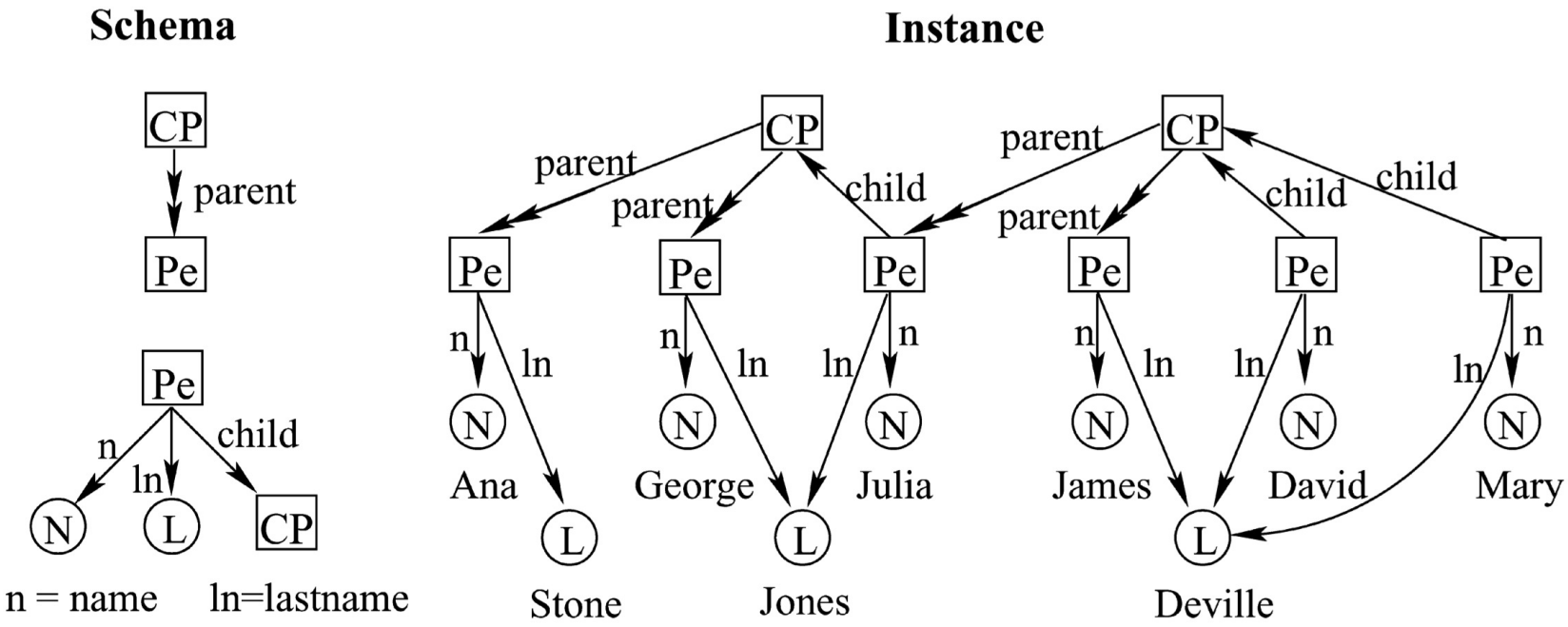
basic nodes (squares, labeled with atomic type), composition (circle with $\times$), collection (circle with $*$, not shown here)

One thread: the Graph-Oriented Object Database model

GOOD. (Gyssens, Paredaens, and Van Gucht, 1990)

- purely graph model, inspired and motivated by object databases
- schemas, instances, and queries are node- and edge-labeled graphs
- two node types: "printable" and "non-printable"

One thread: GOOD



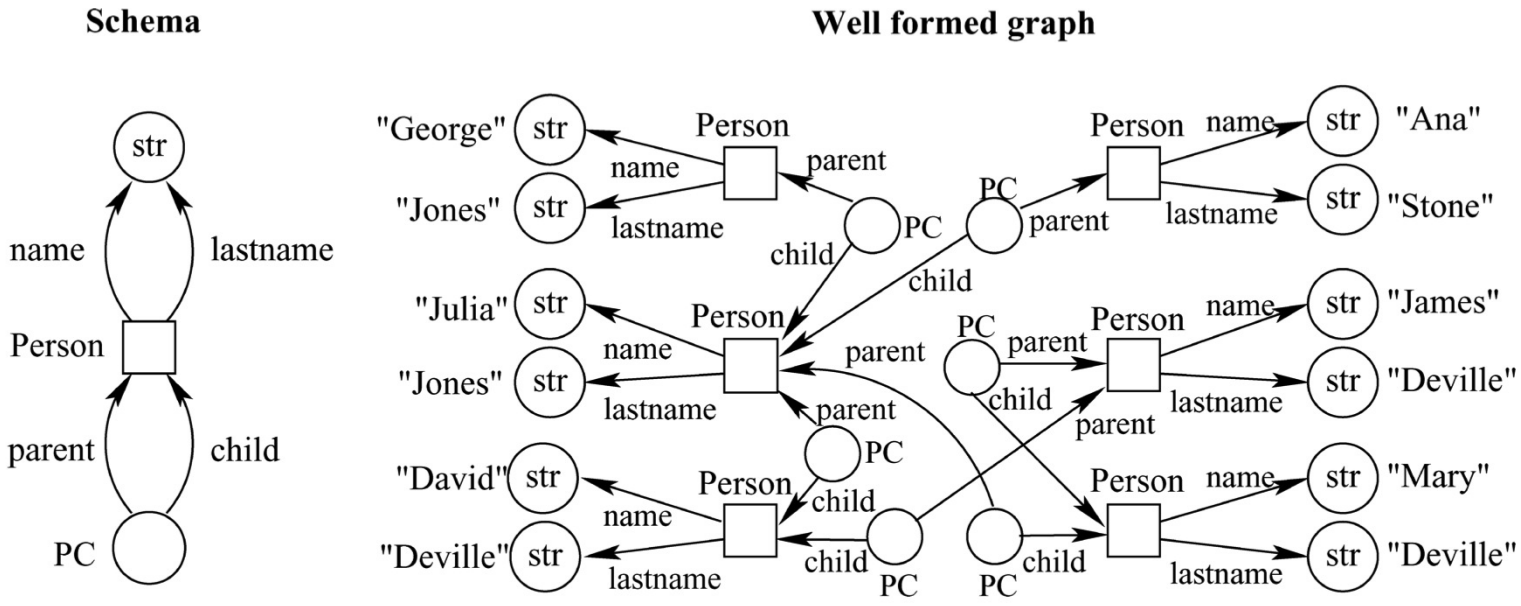
"printable" nodes (circle), "non-printable" nodes (square), functional (i.e., having a unique value) edges (single arrow), non-functional edges (double arrow)

One thread: the Graph Data Model

GDM. (Hidders 2002)

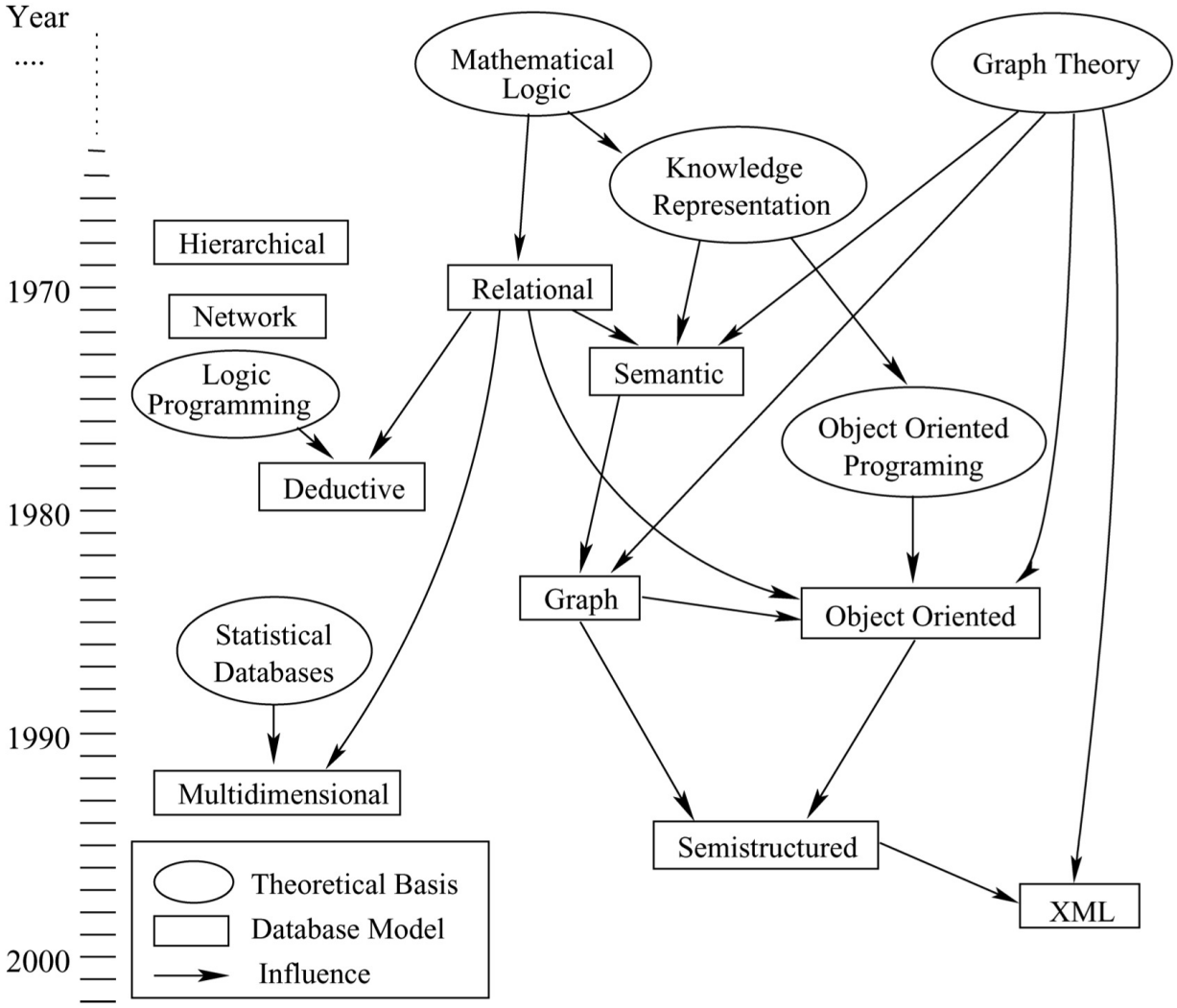
- builds on GOOD to include: complex objects, n-ary relations, inheritance, and other advanced modeling and querying features
- also includes a pattern-based update language (GUL)
- three node types: objects, composite-valued, basic-valued

One thread: GDM



object nodes (square, with class label(s)), composite-value nodes (round empty), basic value nodes (round with type label)

Related models: OO, OEM, and XML data



Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- OEM: self-describing, "semi-structured" model with nesting and node-identity

Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- OEM: self-describing, "semi-structured" model with nesting and node-identity

Extensible Markup Language (XML)

- ordered tree structured data exchange format, with OEM-like identity and references, also self describing

Related models: OO, OEM, XML, ...

Object exchange model (OEM) and object-oriented (OO) data

- OO: flurry of research and industry activity in the 80's and 90's, much work wrapped up into SQL standard
- OEM: self-describing, "semi-structured" model with nesting and node-identity

Extensible Markup Language (XML)

- ordered tree structured data exchange format, with OEM-like identity and references, also self describing

Resource Description Framework (RDF) ...

RDF

Towards triples

A data model for web data?

Some major problems with classical data models

- evolving schemas
- data heterogeneity

Lessons learned in the past decades:

- metadata is data
- it's all about relationships

Design for change

Towards triples

Basic requirements:

1. flexibly capture "things" and their relationships
2. don't force artificial data/metadata distinctions
3. support full spectrum between structured and unstructured data

The RDF data model

Resource description framework (RDF)

- W3C recommendation data model for (semantic) web data
- Graph-based data model, embodying lessons learned from previous data models
- Uses web identifiers (URIs) to identify resources ("things")
- Uses triples to state relationships between things
- serialization formats: N-triples, N3, RDF/XML

... an old idea (circa 1870)

Charles S. Peirce
(1839-1914)

DESCRIPTION

OF A

NOTATION FOR THE LOGIC OF RELATIVES,

RESULTING FROM AN AMPLIFICATION OF THE
CONCEPTIONS OF

BOOLE'S CALCULUS OF LOGIC.

Charles Sanders
By C. S. PEIRCE.

Extracted from the Memoirs of the American Academy, Vol. IX.

CAMBRIDGE:
WELCH, BIGELOW, AND COMPANY,
Printers to the University.
1870.

The RDF data model: triples

Relationships: Triples

- Basic data structure
- has form `(subject, predicate, object)`
- "`subject` *has relationship* `predicate` *to* `object`"

The RDF data model: triples

Relationships: Triples

- Basic data structure
- has form `(subject, predicate, object)`
- "`subject` *has relationship* `predicate` *to* `object`"

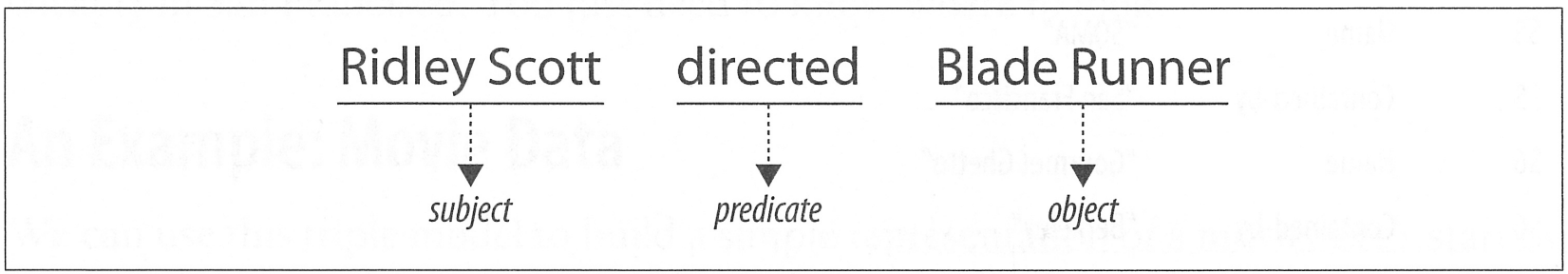


Figure 2-1. Sentence diagram showing a subject-predicate-object relationship

The RDF data model: triples

Relationships: Triples

- Basic data structure
- has form `(subject, predicate, object)`
- "`subject` *has relationship* `predicate` *to* `object`"

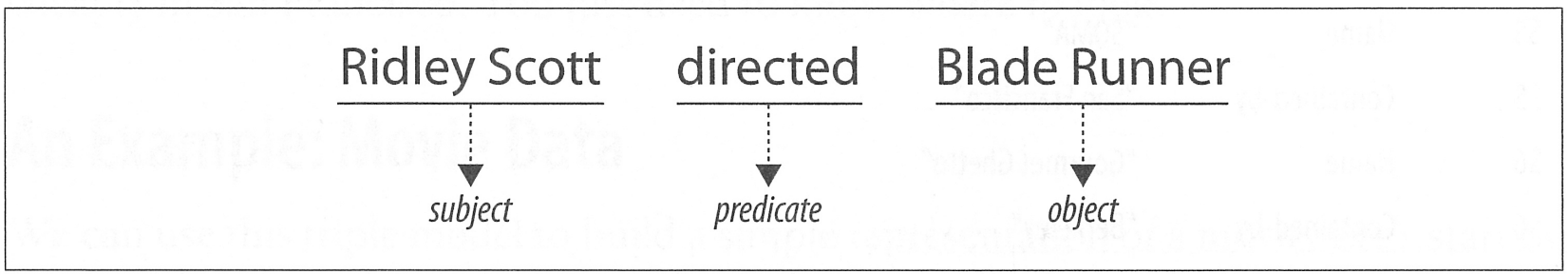


Figure 2-1. Sentence diagram showing a subject-predicate-object relationship

An RDF graph is a finite set of triples.

Sources of RDF data

As part of the LOD vision, a wide variety of data becoming available in RDF:

- Friend of a friend
- Wikipedia
- Uniprot gene database
- Bio2RDF
- BBC, New York Times
- open government data (UK, Australia, US, NL, Japan, ...)
- ...

Richer data modeling

RDF allows us to easily express facts

- John is the father of Mary
  $\{(\mathit{john}, \mathit{fatherOf}, \mathit{mary})\}$
- Geoffrey knows that John is the father of Mary
  $\{(\mathit{geoff}, \mathit{knows}, S), (S, \mathit{subject}, \mathit{john}), (S, \mathit{predicate}, \mathit{fatherOf}), (S, \mathit{object}, \mathit{mary})\}$

Richer data modeling

RDF allows us to easily express facts

- John is the father of Mary
  $\{(\mathit{john}, \mathit{fatherOf}, \mathit{mary})\}$
- Geoffrey knows that John is the father of Mary
  $\{(\mathit{geoff}, \mathit{knows}, S), (S, \mathit{subject}, \mathit{john}), (S, \mathit{predicate}, \mathit{fatherOf}), (S, \mathit{object}, \mathit{mary})\}$

But, we'd like to be able to express more generic knowledge

- All fathers are male
- If a person has a daughter, then they are a parent

This kind of knowledge is often referred to as *schematic*, *ontological*, or *terminological* knowledge

RDF Schema

RDFS

- part of the W3C recommendation for RDF, for schema knowledge
- uses a vocabulary, with pre-defined semantics

RDF Schema

RDFS

- part of the W3C recommendation for RDF, for schema knowledge
- uses a vocabulary, with pre-defined semantics
- every RDFS document is an RDF document
  - models are data: data and "meta" data live side by side. RDFS triples are just data with no intrinsic behavior, other than that provided by associated semantics of keywords

RDF Schema

RDFS

- part of the W3C recommendation for RDF, for schema knowledge
- uses a vocabulary, with pre-defined semantics
- every RDFS document is an RDF document
  - models are data: data and "meta" data live side by side. RDFS triples are just data with no intrinsic behavior, other than that provided by associated semantics of keywords
- vocabulary is generic, not bound to a specific application area
  - allows to specify semantics of user-defined vocabularies (its a kind of meta vocabulary)
  - implementations are responsible for interpreting the vocabulary defined using RDF Schema

RDF Schema: classes

Classes

- classes stand for sets of things (i.e., sets of URIs)

RDF Schema: classes

Classes

- classes stand for sets of things (i.e., sets of URIs)
- "book:uri is a member of the class ex:textbook"

  (book:uri, rdf:type, ex:textbook)

RDF Schema: classes

Classes

- classes stand for sets of things (i.e., sets of URIs)
- "book:uri is a member of the class ex:textbook"

  (book:uri, rdf:type, ex:textbook)
- a URI can belong to several classes

  (book:uri, rdf:type, ex:textbook)
  (book:uri, rdf:type, ex:popularbook)

RDF Schema: classes

Classes

- classes stand for sets of things (i.e., sets of URIs)
- "book:uri is a member of the class ex:textbook"

  (book:uri, rdf:type, ex:textbook)
- a URI can belong to several classes

  (book:uri, rdf:type, ex:textbook)
  (book:uri, rdf:type, ex:popularbook)
- classes can be arranged in hierarchies: "each textbook is a book"

  (ex:textbook, rdfs:subClassOf, ex:book)

RDF Schema: properties

An RDF property is a relation between subject resources and object resources

(ex:John, ex:isMarriedTo, ex:Sally)
(ex:isMarriedTo, rdf:type, rdf:Property)

RDF Schema: properties

An RDF property is a relation between subject resources and object resources

(ex:John, ex:isMarriedTo, ex:Sally)
(ex:isMarriedTo, rdf:type, rdf:Property)

property restrictions

- Allows us to state that a certain property can only be between things of a certain rdf:type
  - E.g. when *a* is married to *b*, then both *a* and *b* are Persons
- Expressed by rdfs:domain and rdfs:range, which are instances of rdf:Property

(ex:isMarriedTo, rdfs:domain, ex:Person)
(ex:isMarriedTo, rdfs:range, ex:Person)

References & Credits

- Towards well-behaved schema evolution. Rada Chirkova and George H. L. Fletcher. *WebDB*, Providence, Rhode Island, 2009.

  `http://www.win.tue.nl/~gfletche/papers-final/ChirkovaFletcherWebDB09.pdf`

- Foundations of semantic web databases. Claudio Gutierrez, Carlos A. Hurtado, Alberto O. Mendelzon, and Jorge Pérez. *J. Comput. Syst. Sci.* 77(3): 520-541, 2011. `http://users.dcc.uchile.cl/~cgutierr/papers/ghmp10.pdf`

- *Linked data: evolving the web into a global data space*. Tom Heath and Christian Bizer. Synthesis Lectures on the Semantic Web, Morgan & Claypool, 2011. `http://linkeddatabook.com/editions/1.0/`

- *Foundations of semantic web technologies.* Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph. Chapman & Hall/CRC, 2009.

  `http://www.semantic-web-book.org/page/Foundations_of_Semantic_Web_Technologies`

- *Programming the semantic web.* Toby Segaran, Colin Evans, and Jamie Taylor. O'Reilly Media, 2009.

graph queries

Graphs

For simplicity, we consider next navigating over directed graphs whose edges are labeled by symbols from a finite, nonempty set of labels Λ.

Formally, then, a graph is a relational structure G, consisting of

- a set of nodes N and,
- for every $R \in \Lambda$, a relation $G(R) \subseteq N \times N$, the set of edges with label R.

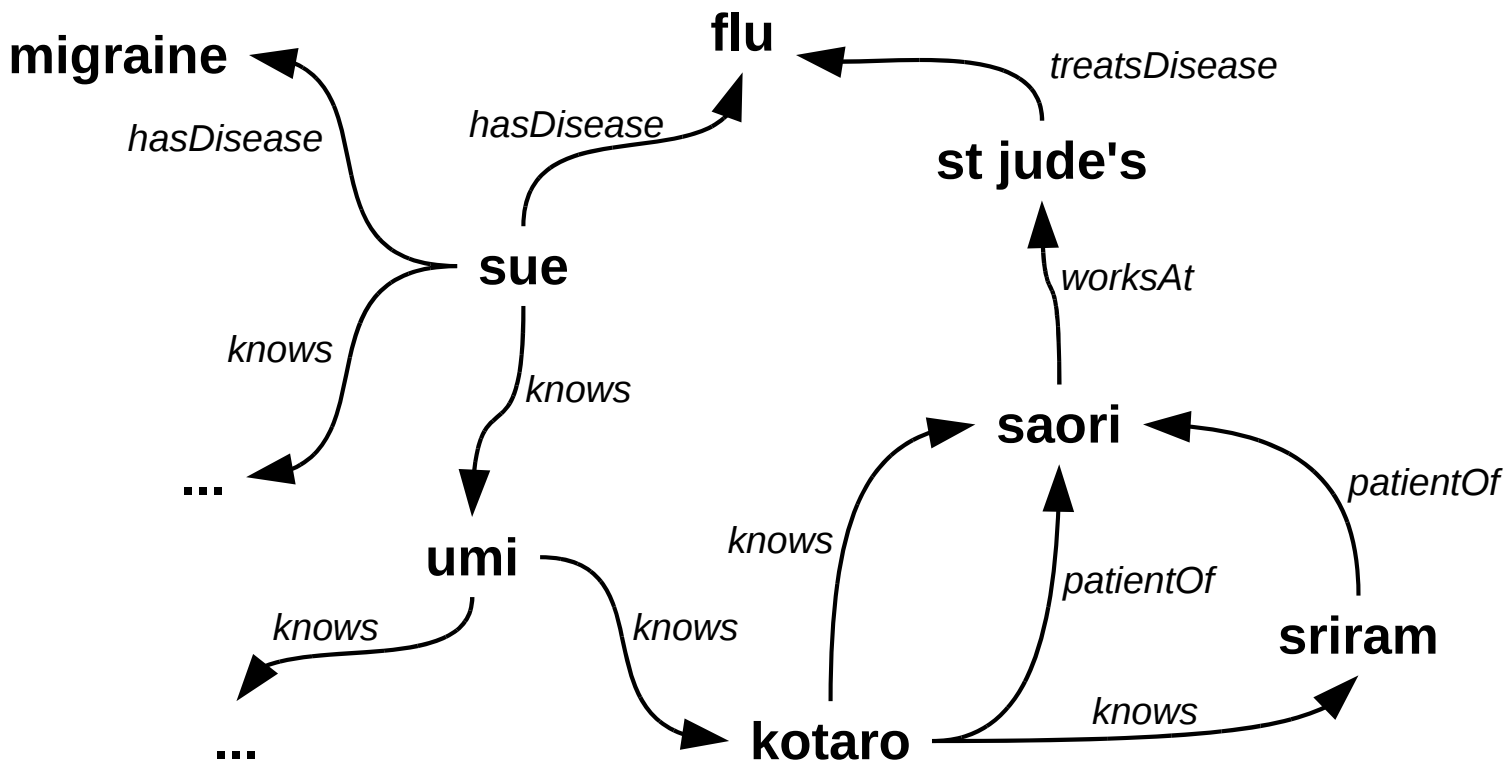
Graphs

For example, suppose our domain is people, hospitals, and diseases, and we have relationships (i.e., edge labels)

$$\Lambda = \{\text{knows}, \text{worksAt}, \text{patientOf}, \text{hasDisease}, \text{treatsDisease}\}.$$

Graphs

A small fragment of such a graph



Query language capabilities

Graph query languages typically feature one or both of the following basic capabilities

- subgraph matching
- finding nodes connected by paths

and possibly one or more advanced features such as *approximate matching* and *comparing paths*

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$
- a query is then a pattern $head \leftarrow body$ where *head* and *body* are sets of edge patterns such that every variable occurring in *head* occurs in *body*
  - alternatively, *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

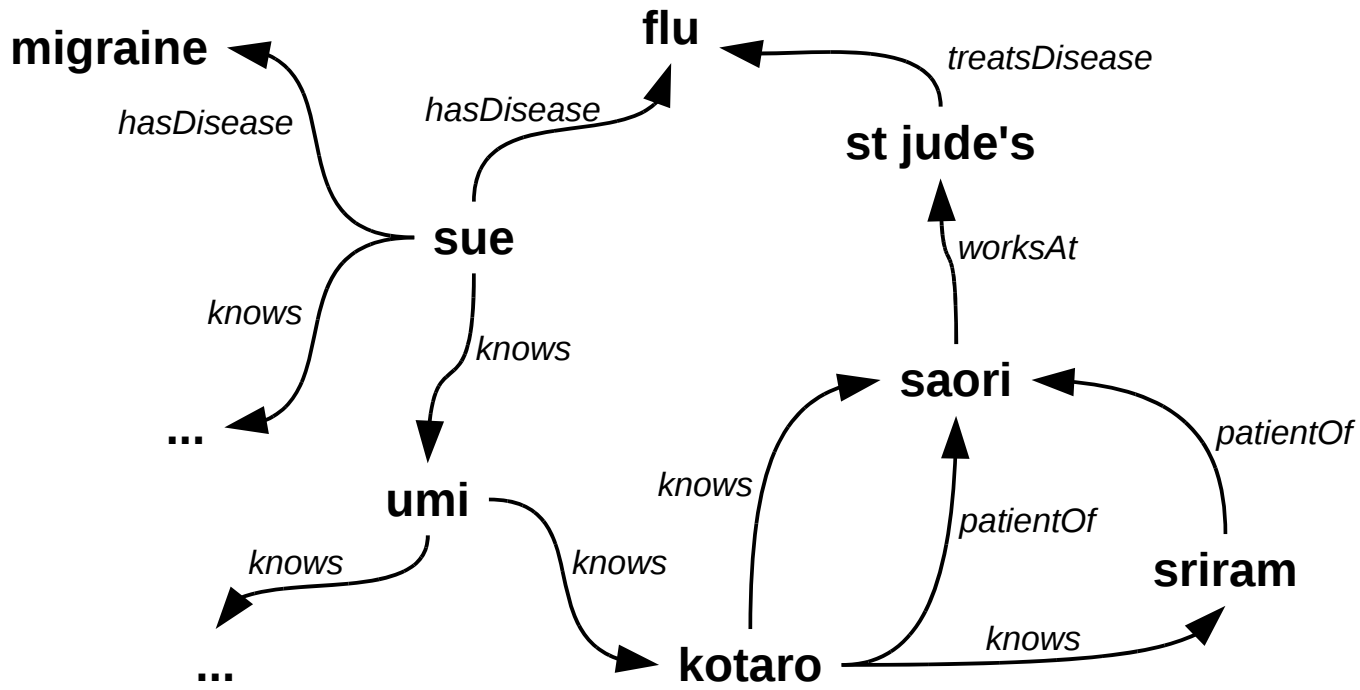
Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (n_1, ℓ, n_2) where n_1 and n_2 can be either constants $n \in N$ or variables, and $\ell \in \Lambda$
- a query is then a pattern $head \leftarrow body$ where *head* and *body* are sets of edge patterns such that every variable occurring in *head* occurs in *body*
  - alternatively, *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*
- the semantics $Q(G)$ of evaluating query Q on graph G is based on embeddings of *body* in G

$$Q(G) = \{h(head) \mid h(body) \subseteq G\}$$

where h is a homomorphism, i.e., a function with domain $N \cup Variables$ and range N that is the identity on N

Subgraph matching

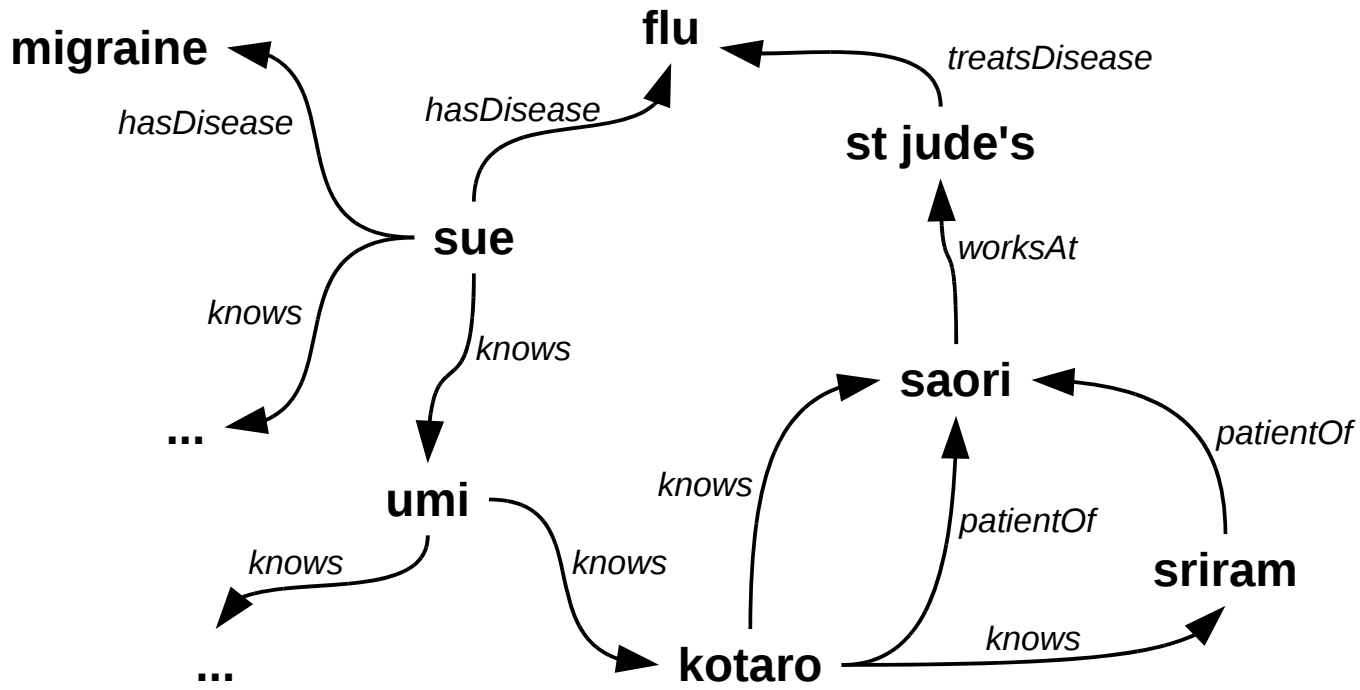


Example: People and the doctors of their friends

$$Q = (?p, \mathit{friendDoctor}, ?d) \leftarrow (?p, \mathit{knows}, ?f), (?f, \mathit{patientOf}, ?d)$$

$$Q(G) = \{(\mathit{umi}, \mathit{friendDoctor}, \mathit{saori}), (\mathit{kotaro}, \mathit{friendDoctor}, \mathit{saori}), \ldots\}$$

Subgraph matching



Example: People who know someone who knows a doctor.

$$Q = \langle ?p \rangle \leftarrow (?p, knows, ?f), (?f, knows, ?d), (?po, patientOf, ?d)$$

$$Q(G) = \{\langle umi \rangle, \ldots\}$$

Subgraph matching

Recall that the combined complexity of conjunctive queries is NP-complete. This is due to the homomorphic semantics of query evaluation.

Hence, relaxations have been proposed recently based on a *simulation*-based semantics for subgraph matching (Fan et al. 2012, 2013)

- quadratic complexity (instead of intractable)

Path matching

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

where r is a regular expression over Λ

Path matching

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

  where r is a regular expression over Λ
- for example, the "knowing" social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, \mathsf{knows}^+, ?y)$$

  and the general social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, (\mathsf{knows} \cup \mathsf{patientOf})^+, ?y)$$

Path matching

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- i.e., queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

  where r is a regular expression over Λ
- for example, the "knowing" social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, \mathsf{knows}^+, ?y)$$

  and the general social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, (\mathsf{knows} \cup \mathsf{patientOf})^+, ?y)$$

- polynomial time complexity
- many variations, such as Conjunctive RPQs and Extended CRPQs, have been intensively studied

References & Credits

- Survey of graph database models. Renzo Angles and Claudio Gutiérrez. *ACM Comput. Surv.* 40(1), 2008. http://users.dcc.uchile.cl/~cgutierr/papers/surveyGDB.pdf
- Querying graph databases. Pablo Barceló Baeza. *PODS 2013*. http://users.dcc.uchile.cl/~pbarcelo/pods001i-barcelo.pdf
- Graph pattern matching revised for social network analysis. Wenfei Fan. *ICDT 2012*. http://homepages.inf.ed.ac.uk/wenfei/qsx/reading/icdt12.pdf
- Incremental graph pattern matching. Wenfei Fan, Xin Wang, and Yinghui Wu. *ACM Trans. Database Syst.* 38(3):18, 2013. http://tods.acm.org/accepted/2013/FanIncremental.pdf
- Query languages for graph databases. Peter T. Wood. *SIGMOD Record* 41(1):50-60, 2012. http://www.sigmod.org/publications/sigmod-record/1203/pdfs/08.principles.wood.pdf

SPARQL

SPARQL

- SPARQL: SPARQL Protocol And RDF Query Language
  - Query language for data from RDF documents
  - W3C specification since January 2008
  - Extremely successful in practice
    - Key technology in the Linked Data framework

SPARQL

- SPARQL: SPARQL Protocol And RDF Query Language
  - Query language for data from RDF documents
  - W3C specification since January 2008
  - Extremely successful in practice
    - Key technology in the Linked Data framework
  - (Backwards compatible) revision finished in March 2013

SPARQL

- SPARQL: SPARQL Protocol And RDF Query Language
  - Query language for data from RDF documents
  - W3C specification since January 2008
  - Extremely successful in practice
    - Key technology in the Linked Data framework
  - (Backwards compatible) revision finished in March 2013
- Parts of the SPARQL specification:
  - Query language: our focus
  - Result format: encode results in XML
  - Query protocol: transmitting queries and results

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

- Main part is a query pattern (WHERE)
  - Patterns use RDF Turtle syntax
  - Variables can be used, even in predicate positions (?variable)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

- Main part is a query pattern (WHERE)
  - Patterns use RDF Turtle syntax
  - Variables can be used, even in predicate positions (?variable)
- Abbreviations for URIs (PREFIX)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

- Main part is a query pattern (WHERE)
  - Patterns use RDF Turtle syntax
  - Variables can be used, even in predicate positions (?variable)
- Abbreviations for URIs (PREFIX)
- Graph(s) to be queried (FROM)

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

- Main part is a query pattern (WHERE)
  - Patterns use RDF Turtle syntax
  - Variables can be used, even in predicate positions (?variable)
- Abbreviations for URIs (PREFIX)
- Graph(s) to be queried (FROM)
- Query result based on selected variables (SELECT)
  - other query forms are CONSTRUCT (output new graph), ASK (boolean), and DESCRIBE (informative description)

SPARQL: Basic graph patterns

$$\{\langle \text{ex:book1}, \text{ex:title}, \text{“title 1”}\rangle,$$
$$\langle \text{ex:book1}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle,$$
$$\langle \text{ex:book1}, \text{ex:author}, \text{johndoe}\rangle,$$
$$\langle \text{ex:book2}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle,$$
$$\langle \text{ex:book2}, \text{ex:title}, \text{“title 1”}\rangle,$$
$$\langle \text{ex:book3}, \text{ex:publishedBy}, \text{http://springer.com}\rangle,$$
$$\langle \text{ex:book3}, \text{ex:author}, \text{janedoe}\rangle,$$
$$\langle \text{ex:book4}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle, \ldots\}$$

Basic graph patterns (BGP) form the core of SPARQL (the WHERE clause). In essence, a BGP is a finite set of triples, wherein elements of the triples may be atoms or variables. The semantics of a BGP P with respect to a graph G is the set of all bindings β of the variables in P to atoms such that $\beta(P) \subseteq G$.

SPARQL: Basic graph patterns

$$\{\langle \text{ex:book1}, \text{ex:title}, \text{“title 1”}\rangle,$$
$$\langle \text{ex:book1}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle,$$
$$\langle \text{ex:book1}, \text{ex:author}, \text{johndoe}\rangle,$$
$$\langle \text{ex:book2}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle,$$
$$\langle \text{ex:book2}, \text{ex:title}, \text{“title 1”}\rangle,$$
$$\langle \text{ex:book3}, \text{ex:publishedBy}, \text{http://springer.com}\rangle,$$
$$\langle \text{ex:book3}, \text{ex:author}, \text{janedoe}\rangle,$$
$$\langle \text{ex:book4}, \text{ex:publishedBy}, \text{http://elsevier.com}\rangle, \ldots\}$$

For example, in

$$(?book, publishedBy, elsevier), (?book, title, ?title),$$
$$(?book, author, ?author)$$

variable $?book$ binds to all those Elsevier books with a known title and author. In the database G above, the only valid binding for $?book$ is “ex:book1”.

SPARQL: SELECT versus CONSTRUCT

The SELECT query form generates vectors of variable bindings

$$\langle ?title, ?author \rangle \leftarrow (?book, publishedBy, elsevier),$$
$$(?book, title, ?title), (?book, author, ?author)$$

SPARQL: SELECT versus CONSTRUCT

The SELECT query form generates vectors of variable bindings

$$\langle ?title, ?author \rangle \leftarrow (?book, publishedBy, elsevier), \\ (?book, title, ?title), (?book, author, ?author)$$

The CONSTRUCT query form, on the other hand, generates an RDF graph

$$(?book, rdf : type, fullBook) \leftarrow (?book, publishedBy, elsevier), \\ (?book, title, ?title), (?book, author, ?author)$$

SPARQL: SELECT versus CONSTRUCT

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?book rdf:type ex:fullBook }
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author . }
```

SPARQL

Further features

- OPTIONAL clauses (introduces null values)
- FILTER clauses, for specifying filter conditions
- UNION of BGPs
- Modifiers: postprocess query result set, e.g.: ORDER BY ?age LIMIT 10 OFFSET 5

SPARQL

Further features

- OPTIONAL clauses (introduces null values)
- FILTER clauses, for specifying filter conditions
- UNION of BGPs
- Modifiers: postprocess query result set, e.g.: ORDER BY ?age LIMIT 10 OFFSET 5

Further features (SPARQL 1.1)

- aggregates, negation, subqueries, expressions in SELECT, path recursion, federation
- we will explore these in the following

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book   ex:publishedBy <http://elsevier.com> .
  ?book   ex:title       ?title .
  ?book   ex:author      ?author .
  ?author foaf:knows/foaf:name ?name
}
```

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book   ex:publishedBy <http://elsevier.com> .
  ?book   ex:title       ?title .
  ?book   ex:author      ?author .
  ?author foaf:knows/foaf:knows/foaf:name ?name
}
```

SPARQL 1.1

Property paths. Allows us to introduce RPQs into BGPs.

```
PREFIX ex: <http://example.org/>
SELECT ?name
FROM <http://example.org/example>
WHERE
{ ?book   ex:publishedBy <http://elsevier.com> .
  ?book   ex:title       ?title .
  ?book   ex:author      ?author .
  ?author foaf:knows+/foaf:name ?name
}
```

SPARQL 1.1

Federation. Allows for evaluating BGPs on non-local service points, via the SERVICE keyword.

```
PREFIX ex: <http://example.org/>
SELECT ?title ?phone
FROM <http://example.org/example>
WHERE
{ ?book ex:publishedBy <http://elsevier.com> .
  ?book ex:title       ?title .
  ?book ex:author      ?author .
  SERVICE <http://people.example.org/sparql>
    { ?author foaf:phone ?phone . }
}
```

SPARQL 1.1

Federation. Note that SERVICE point can be dynamically retrieved

```
PREFIX ex: <http://example.org/>
SELECT ?title ?phone
FROM <http://example.org/example>
WHERE
{ ?book   ex:publishedBy <http://elsevier.com> .
  ?book   ex:title       ?title .
  ?book   ex:author      ?author .
  ?author ex:authority   ?source .
  SERVICE ?source
    { ?author foaf:phone ?phone . }
}
```

SPARQL

SPARQL: Based on matching simple graph patterns

- simple SQL-like syntax
- flexible and expressive, for loosely structured RDF data
  - Grouping, optionals, and alternatives
  - Filters: extra-logical result restrictions
  - Variety of query forms
- key standard in linked data initiative
  - try out the DBPedia end point `http://live.dbpedia.org/sparql`

Basic graph pattern processing: in brief

$$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$$

Basic graph pattern processing: in brief

$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$

- each triple pattern evaluates in a graph G to the set of matching triples in G

  $(?person, knows, mary) \Rightarrow_G \{(john, knows, mary), (sue, knows, mary), \ldots\}$

Basic graph pattern processing: in brief

$$(?person, knows, mary), (?person, livesIn, ?city), (?city, locatedIn, Italy)$$

- each triple pattern evaluates in a graph G to the set of matching triples in G

  $$(?person, knows, mary) \Rightarrow_G \{(john, knows, mary), (sue, knows, mary), \ldots\}$$

- BGP is evaluated by *joining* these sets on common variables

  $$\{(john, knows, mary), (sue, knows, mary), \ldots\}$$
  $$\bowtie$$
  $$\{(john, livesIn, Eindhoven), (sue, livesIn, Trento), \ldots\}$$
  $$\bowtie$$
  $$\{(Rome, locatedIn, Italy), (Trento, locatedIn, Italy), \ldots\}$$

Storage in external memory: in brief

Two basic storage alternatives

1. flat file of triples, one per line
   - ... sorted on some subset of positions (e.g., by subject values)

Storage in external memory: in brief

Two basic storage alternatives

1. flat file of triples, one per line
   - ... sorted on some subset of positions (e.g., by subject values)
2. index data structure: maps a search key to a set of matching triples
   - B+tree: logarithmic lookup cost, sorted access
   - hash table: essentially constant lookup cost, point access

Storage: on the varieties of triple indexes

state of the art in the market: value-based indexing

- MAP (2007)
- HexTree (2008)
- TripleT (2009)

Storage: on the varieties of triple indexes

state of the art in the market: value-based indexing

- MAP (2007)
- HexTree (2008)
- TripleT (2009)

still in the research lab: structure-based indexing

- group triples both by the values they share and by the *structure* they share in the graph (e.g., similar neighborhood topology)
- structure-based indexing also successful in XML data management

Storage: on the varieties of (value-based) triple indexes

Let's focus on value-based indexes

For graph G, let

$$\begin{aligned}
\mathcal{S}(G) &= \{s \mid (s,p,o) \in G\} \\
\mathcal{P}(G) &= \{p \mid (s,p,o) \in G\} \\
\mathcal{O}(G) &= \{o \mid (s,p,o) \in G\} \\
\mathcal{A}(G) &= \mathcal{S}(G) \cup \mathcal{P}(G) \cup \mathcal{O}(G)
\end{aligned}$$

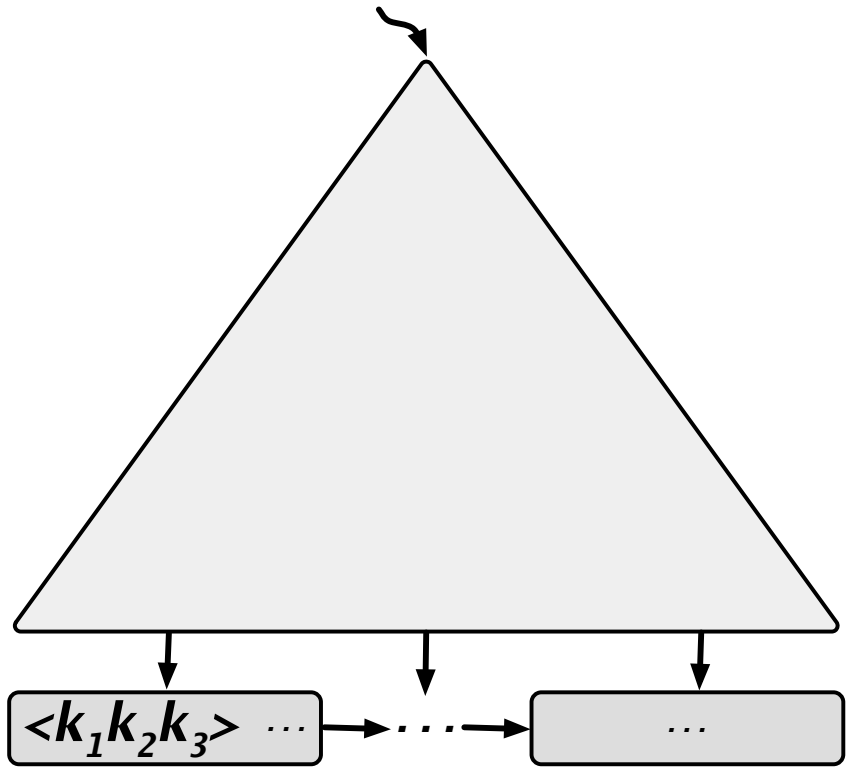
Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O

$$
\begin{aligned}
SPO &= \{s\#p\#o \mid (s,p,o) \in G\} \\
SOP &= \{s\#o\#p \mid (s,p,o) \in G\} \\
PSO &= \{p\#s\#o \mid (s,p,o) \in G\} \\
POS &= \{p\#o\#s \mid (s,p,o) \in G\} \\
OSP &= \{o\#s\#p \mid (s,p,o) \in G\} \\
OPS &= \{o\#p\#s \mid (s,p,o) \in G\}
\end{aligned}
$$

Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O



... times six

Storage: on the varieties of triple indexes

MAP: index all permutations of S, P, and O

Most popular approach to triple storage in practice

- RDF-3X: open source triple store
  - `http://code.google.com/p/rdf3x/`
- Virtuoso: open source and commercial industrial-strength triple store
  - `http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/`
- Sesame/BigData: open source and commercial industrial-strength triple store
  - `http://www.openrdf.org/`
  - `http://www.systap.com/bigdata.htm`

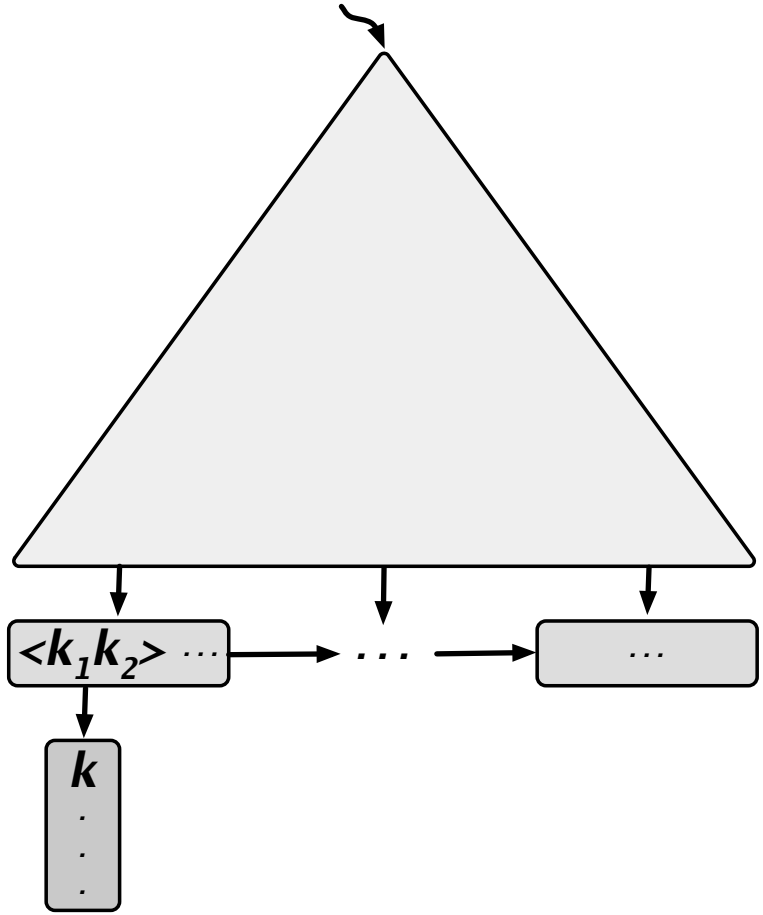
Storage: on the varieties of triple indexes

HexTree: index all pairs of S, P, and O

$$
\begin{aligned}
SP &= \{s\#p \mid (s,p,o) \in G\} \\
SO &= \{s\#o \mid (s,p,o) \in G\} \\
PS &= \{p\#s \mid (s,p,o) \in G\} \\
PO &= \{p\#o \mid (s,p,o) \in G\} \\
OS &= \{o\#s \mid (s,p,o) \in G\} \\
OP &= \{o\#p \mid (s,p,o) \in G\}
\end{aligned}
$$

Storage: on the varieties of triple indexes

HexTree: index all pairs of S, P, and O



... times six

Storage: on the varieties of triple indexes

HexTree: partners share payload

$$
\begin{aligned}
SP &\rightarrow \{o \in \mathcal{O}(G) \mid (s, p, o) \in G\} \\
PS &\rightarrow \{o \in \mathcal{O}(G) \mid (s, p, o) \in G\}
\end{aligned}
$$

Storage: on the varieties of triple indexes

MAP and HexTree

- *strengths:* support all native joins expressible in the SPARQL WHERE clause, using fast "merge" joins, with no constraints on "schema"

Storage: on the varieties of triple indexes

MAP and HexTree

- *strengths:* support all native joins expressible in the SPARQL WHERE clause, using fast "merge" joins, with no constraints on "schema"
- *weaknesses*
  - lots of (redundant) storage space
  - access to multiple data structures to perform joins

Storage: on the varieties of triple indexes

MAP and HexTree

- *strengths:* support all native joins expressible in the SPARQL WHERE clause, using fast "merge" joins, with no constraints on "schema"
- *weaknesses*
  - lots of (redundant) storage space
  - access to multiple data structures to perform joins
  - in general, weak data locality

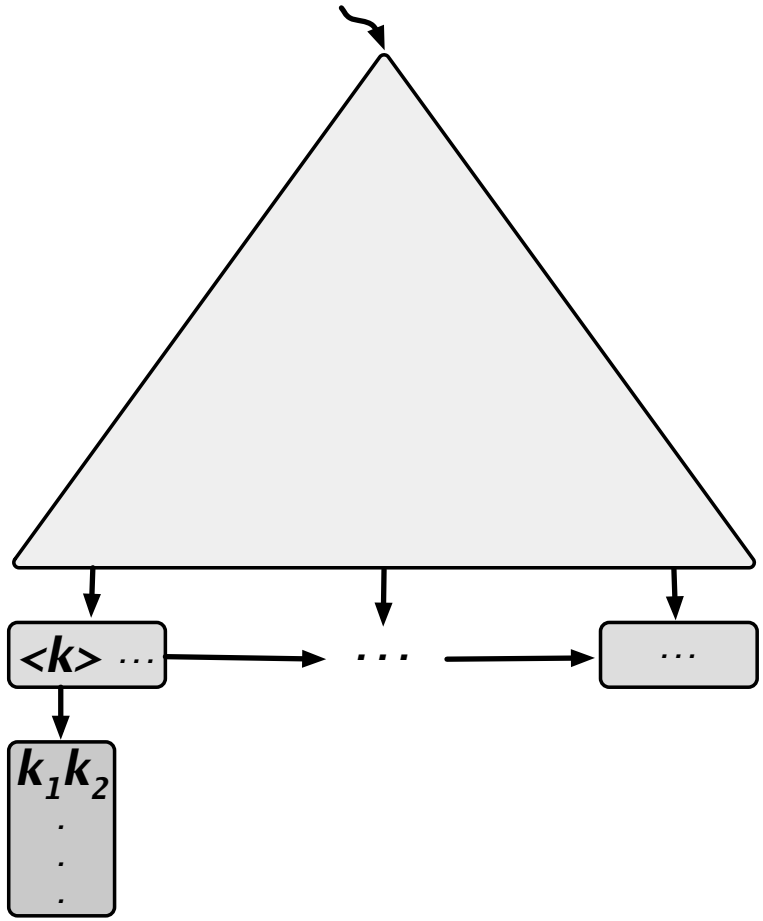
Storage: on the varieties of triple indexes

TripleT

- index $\mathcal{A}(G)$
- just one data structure – a three-way triple tree

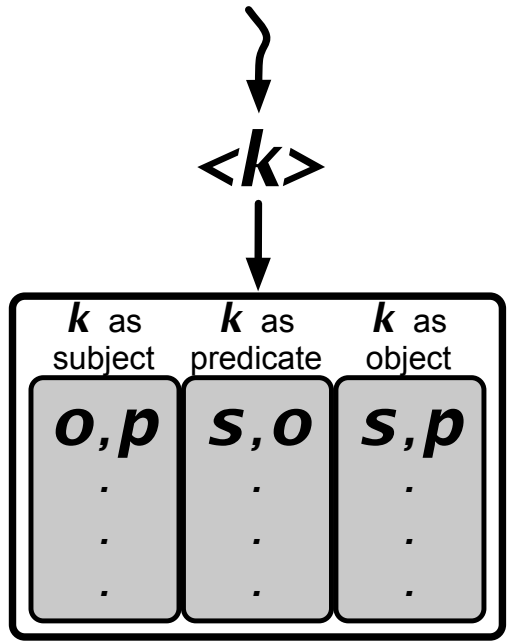
Storage: on the varieties of triple indexes

TripleT



Storage: on the varieties of triple indexes

TripleT



payload

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree
- reduced storage space
- reduced key size (i.e., shallower trees)

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree
- reduced storage space
- reduced key size (i.e., shallower trees)
- single data structure to perform joins

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree
- reduced storage space
- reduced key size (i.e., shallower trees)
- single data structure to perform joins
- in general, stronger data locality

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree
- reduced storage space
- reduced key size (i.e., shallower trees)
- single data structure to perform joins
- in general, stronger data locality

Leads to significantly reduced storage and query processing costs

Storage: on the varieties of triple indexes

TripleT advantages

- retains strengths of MAP and HexTree
- reduced storage space
- reduced key size (i.e., shallower trees)
- single data structure to perform joins
- in general, stronger data locality

Leads to significantly reduced storage and query processing costs

idea appears in various guises in products

- open source TU/e implementation `https://github.com/b-w/TripleT`
- open source 4store engine `http://4store.org`
- and storage of RDF in so-called column stores
  - open source MonetDB `http://www.monetdb.org/`
  - commercial Vertica `http://www.vertica.com/`

Storage: on the varieties of triple indexes

Also, relational DB-based RDF solutions are available in industrial-strength IBM DB2 and Oracle Database 11G products. See:

- `http://www.ibm.com/developerworks/data/tutorials/dm-1205rdfdb210/index.html`
- `http://www.oracle.com/technetwork/database/options/semantic-tech/index.html`

Both use various hybrids of the MAP, HexTree, and TripleT indexes

References & Credits

- An extensible framework for query optimization on TripleT-based RDF stores. Bart Wolff, George Fletcher, James Lu. LWDM 2015, Brussels.

  `http://www.win.tue.nl/~gfletche/papers-final/wolff-camera.pdf`

- Querying semantic data on the web. Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Perez, and Juan Sequeda. *SIGMOD Record* 41(4): 6-17, 2012.

  `http://www.sigmod.org/publications/sigmod-record/1212/pdfs/03.principles.arenas.pdf`

- *Linked data: evolving the web into a global data space*. Tom Heath and Christian Bizer. Synthesis Lectures on the Semantic Web, Morgan & Claypool, 2011.

  `http://linkeddatabook.com/editions/1.0/`

Models and queries for linked data

Formal models for querying the web of data

What does it even mean to query the web?

Formal models for querying the web of data

What does it even mean to query the web?

There have been (relatively) few investigations which attempt directly address this question

- Mendelzon and Milo (1998)
- Abiteboul and Vianu (2000)
- Bouquet, Ghidini, and Serafini (2010)
- Hartig and Freytag (2012)

We next take a brief look at each of these major efforts, each of which help us demarcate the borders of practical LOD query processing solutions

Mendelzon and Milo

(MM) The web of data is modeled as a *finite* virtual database with schema

$$Node(id, \overline{A_N}),$$
$$Link(source, destination, \overline{A_L}),$$
$$R_1(\overline{A_1}), \ldots, R_n(\overline{A_n})$$

such that (1) *id* is a key for *Node*, (2) the inclusion dependency $\pi_{source}(Link) \subseteq \pi_{id}(Node)$ holds; and, (3) the R_i's are normal arbitrary relational schemas, which are materialized locally.

The *id* field of *Node* and the *source*, *destination* fields of *Link* contain URIs.
The other fields contain node content and edge/node labels, etc.

Mendelzon and Milo

The *Node* and *Link* tables are never fully available to the user. The only way to access *Node* is by specifying a URI. The only way to access *Link* is by specifying a source URI.

Mendelzon and Milo

The *Node* and *Link* tables are never fully available to the user. The only way to access *Node* is by specifying a URI. The only way to access *Link* is by specifying a source URI.

- some "seed" nodes may be stored in one of the R'_i, e.g., as bookmarks or results from a search engine
- Since URI's are recursively enumerable, users can also try to "guess" valid Node ID's (but never know when to stop ...)

Mendelzon and Milo

The *Node* and *Link* tables are never fully available to the user. The only way to access *Node* is by specifying a URI. The only way to access *Link* is by specifying a source URI.

- some "seed" nodes may be stored in one of the R'_i, e.g., as bookmarks or results from a search engine
- Since URI's are recursively enumerable, users can also try to "guess" valid Node ID's (but never know when to stop ...)

Hence, access to the web of data is by navigation from known nodes to new nodes, following *Link*s

Mendelzon and Milo

The query model is arbitrary computable queries, with a *Node*/*Link* oracle

Mendelzon and Milo

The query model is arbitrary computable queries, with a *Node*/*Link* oracle

MM establish that the following queries are computable:

a. Find all nodes reachable from the node with URI u
b. Find all nodes on a cycle of length at most 3 involving node with URI u

while the following are not computable:

c. Find all nodes
d. Find all nodes with no incoming links
e. Find all nodes referencing the node with URI u (though eventually computable)

Mendelzon and Milo

The query model is arbitrary computable queries, with a *Node*/*Link* oracle

MM establish that the following queries are computable:

a. Find all nodes reachable from the node with URI u
b. Find all nodes on a cycle of length at most 3 involving node with URI u

while the following are not computable:

c. Find all nodes
d. Find all nodes with no incoming links
e. Find all nodes referencing the node with URI u (though eventually computable)

MM also study the subtle impact of updates on computability (e.g., (a.) becomes "eventually" computable, depending on how fast the query engine is ...)

Abiteboul and Vianu

(AV) The web of data is modeled as Mendelzon and Milo except with the crucial difference that instances are infinite.

> *We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)*

Abiteboul and Vianu

(AV) The web of data is modeled as Mendelzon and Milo except with the crucial difference that instances are infinite.

> *We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)*

Hence, "find all nodes reachable from the node with URI u" moves from being computable to being eventually computable ...

Abiteboul and Vianu

(AV) The web of data is modeled as Mendelzon and Milo except with the crucial difference that instances are infinite.

> *We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)*

Hence, "find all nodes reachable from the node with URI *u*" moves from being computable to being eventually computable ...

AV study various weaker query languages as well, showing, e.g., that all positive FO queries are eventually computable, even if recursion is introduced.

Abiteboul and Vianu

(AV) The web of data is modeled as Mendelzon and Milo except with the crucial difference that instances are infinite.

> *We believe this captures the intuition that exhaustive exploration of the Web is – or will soon become – prohibitively expensive ... our model draws a sharp distinction between exhaustive exploration of the Web and more controlled types of computation. (AV)*

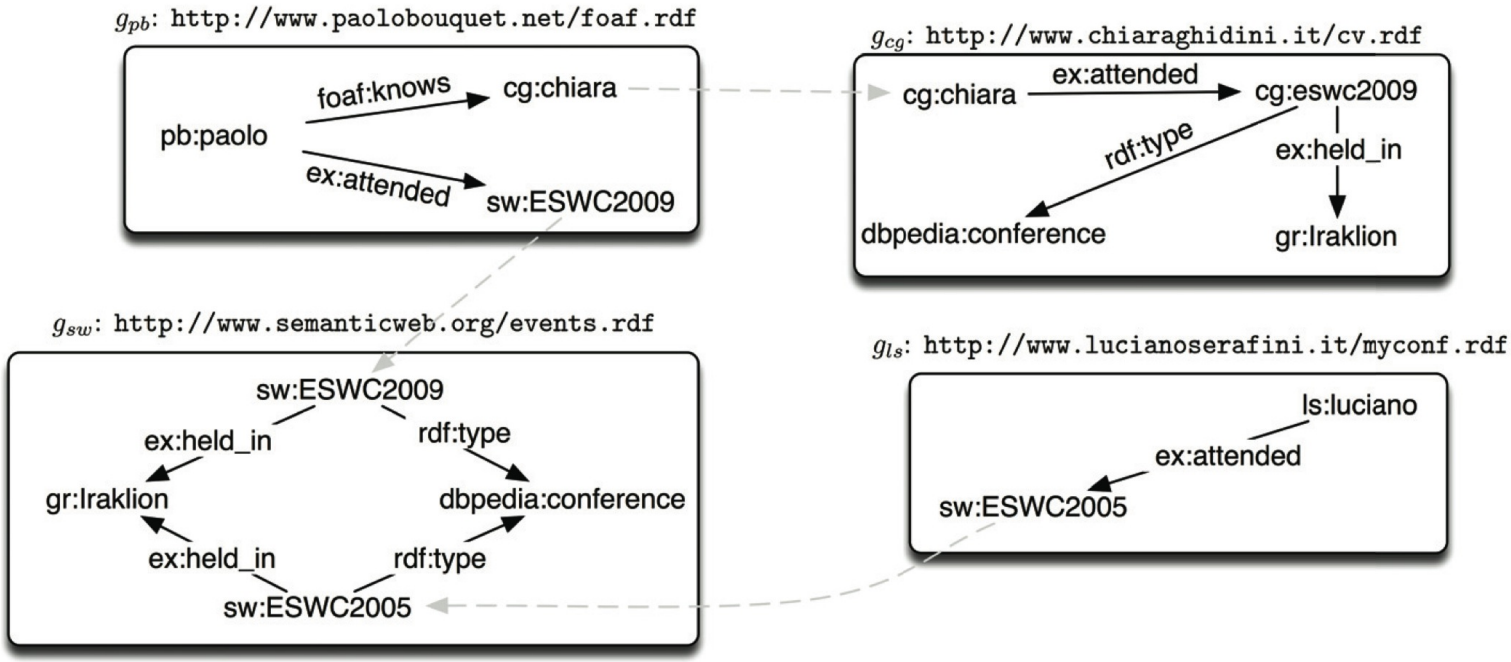
Hence, "find all nodes reachable from the node with URI *u*" moves from being computable to being eventually computable ...

AV study various weaker query languages as well, showing, e.g., that all positive FO queries are eventually computable, even if recursion is introduced.

No study of updates has been made in this model, although it has been extended by Spielmann et al. PODS 2002, to study efficient distributed processing of queries

Bouquet, Ghidini, and Serafini

BGS give formal semantics for two models of query processing on a *finite* web of data, using the RDF standard. BGS do not study the impact of updates.

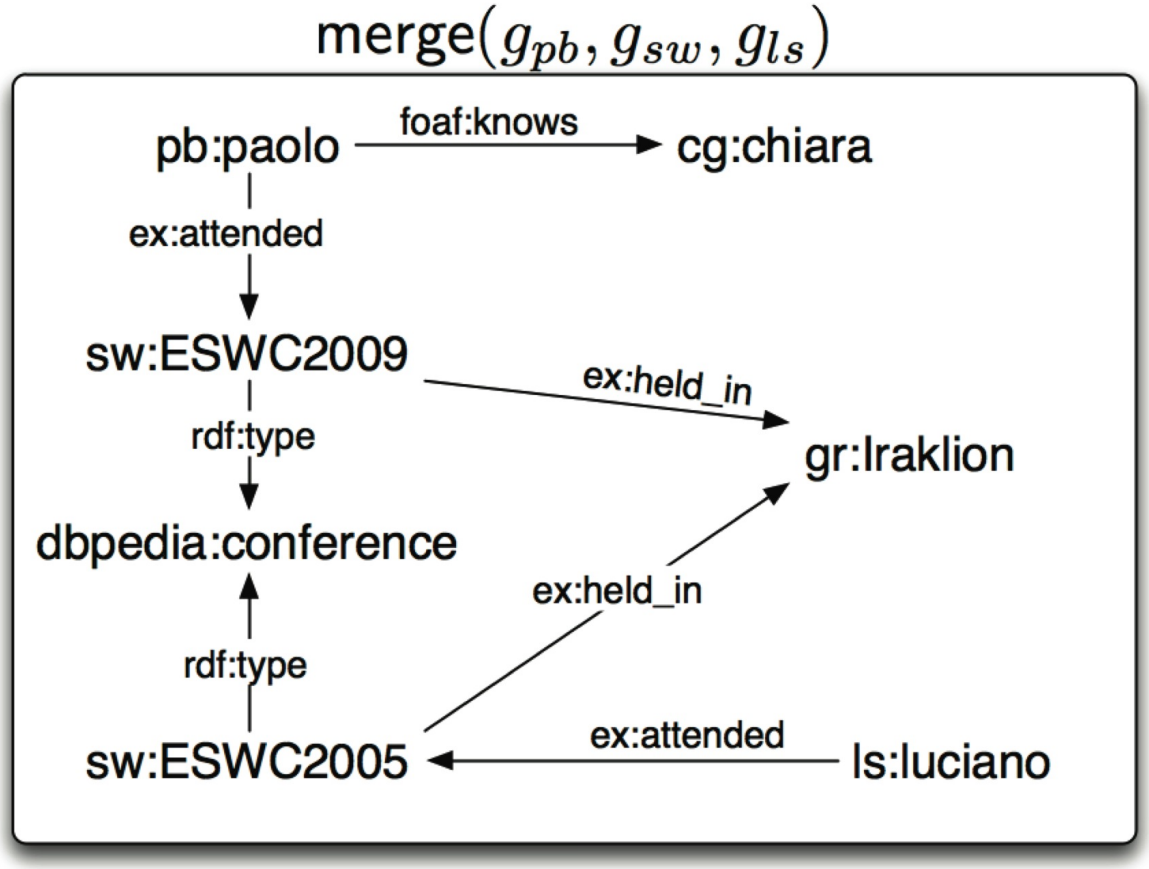


four LOD sources g_{pb}, g_{ls}, g_{cg}, and g_{sw}

In particular, URI's resolve to finite sets of triples. Then, query evaluation is over the "merge" (i.e., union, with BNode renaming) of a set of graphs identified by the model.

Bouquet, Ghidini, and Serafini

In the direct access query model, queries are evaluated over the merge of graphs resolved from some known finite set of URI's.

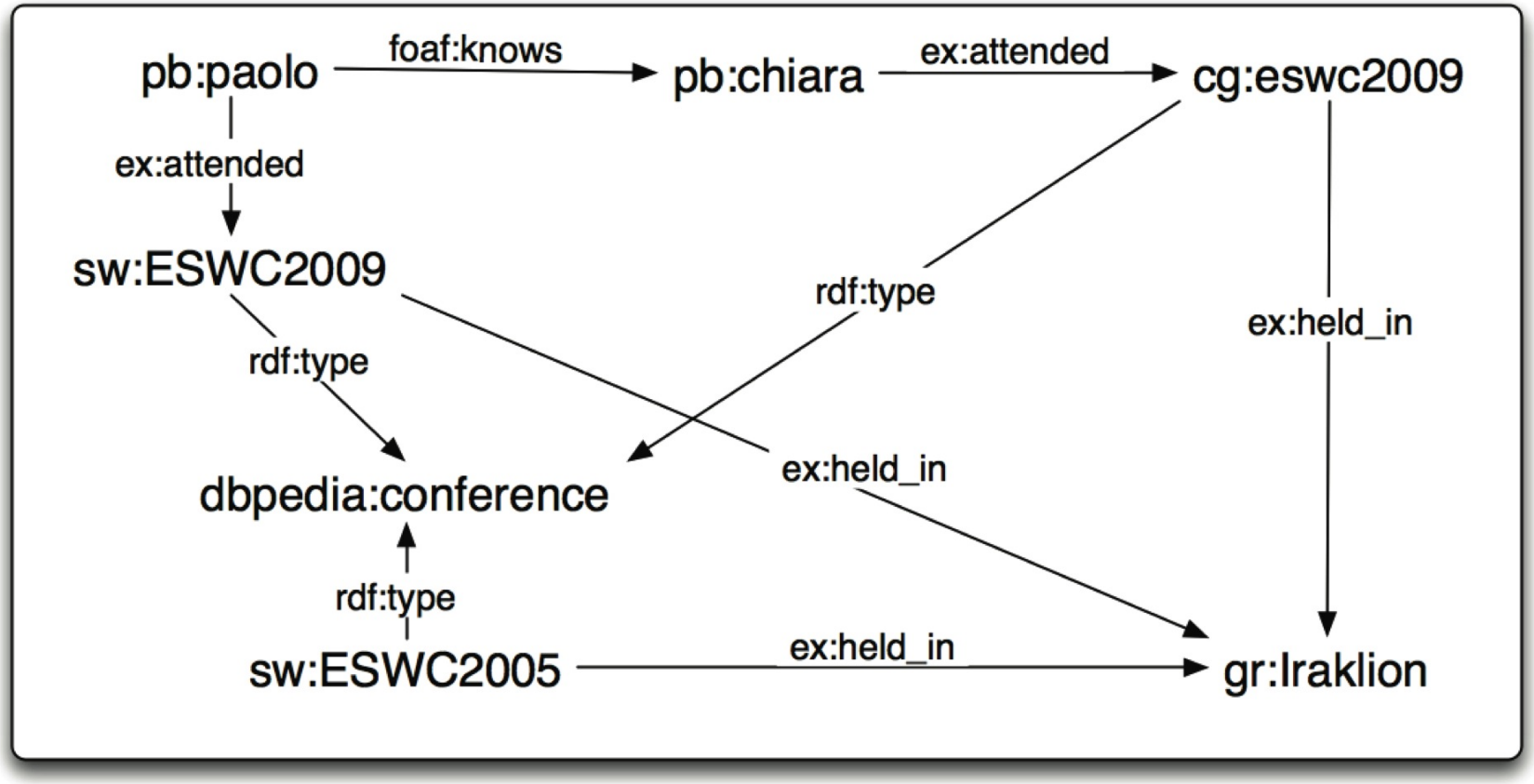


direct access on g_{pb}, g_{ls}, and g_{sw}

Bouquet, Ghidini, and Serafini

In the navigational query model, queries are evaluated over the merge of graphs resolved by following all outgoing links, starting from some known finite set of URI's.

$$\text{merge}(g_{pb}, g_{cg}, g_{sw})$$



navigation from g_{pb}

Hartig and Freytag

HF give a formal semantics for a model of query processing on an *infinite* web of data. HF do not study the impact of updates.

In essence, they follow the AV framework, with the navigational query semantics of BGS, abstracting away from the LOD standards, and also concrete models of navigation

- e.g., they consider, in addition to the BGS model of resolving all outgoing links, resolving only those links in current partial query result bindings

They also adapt the computability notions of AV to this setting.

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- Whether or not the web of data is bounded or unbounded
  - *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
  - *unbounded*: Abiteboul and Vianu; Hartig and Freytag

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- Whether or not the web of data is bounded or unbounded
  - *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
  - *unbounded*: Abiteboul and Vianu; Hartig and Freytag
- Whether or not the web of data is treated as static or dynamic
  - *static*: first model of Mendelzon and Milo; models of Bouquet et al.; Abiteboul and Vianu; Hartig and Freytag
  - *dynamic*: second model of Mendelzon and Milo

Formal models for querying the web of data

We see that these general models can be categorized along two axes

- Whether or not the web of data is bounded or unbounded
  - *bounded*: models of Mendelzon and Milo; models of Bouquet et al.
  - *unbounded*: Abiteboul and Vianu; Hartig and Freytag
- Whether or not the web of data is treated as static or dynamic
  - *static*: first model of Mendelzon and Milo; models of Bouquet et al.; Abiteboul and Vianu; Hartig and Freytag
  - *dynamic*: second model of Mendelzon and Milo

Note that there has been no formal study of a dynamic unbounded model for the web! Clearly, the real web as we know it satisfies both of these conditions ...

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- Universal cost models cannot be maintained

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- Universal cost models cannot be maintained
- Query execution is non-deterministic

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- Universal cost models cannot be maintained
- Query execution is non-deterministic
- Standards do not guarantee interoperability (i.e., homogeneity)

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- Universal cost models cannot be maintained
- Query execution is non-deterministic
- Standards do not guarantee interoperability (i.e., homogeneity)
- Data sources are not coordinated centrally, both in alignment of URI references and in vocabulary

Querying the (real) web of data

More generally, based on initial practical experiences with deploying the LOD vision, several fundamental challenges have been identified in the community (Hartig 2013, Umbrich et al. 2013)

- Data sources are not reliable
- Consumer behavior cannot be anticipated
- Data sources are not always trustworthy
- You can never know the complete state of the web (i.e., the web of data is a virtually unbounded space)
- Universal cost models cannot be maintained
- Query execution is non-deterministic
- Standards do not guarantee interoperability (i.e., homogeneity)
- Data sources are not coordinated centrally, both in alignment of URI references and in vocabulary
- You can guarantee at most two of the following properties at a time: alignment (i.e., currency), coverage, efficiency

Practical models for querying the web of data

Towards overcoming these challenges for practical LOD query processing solutions, three basic approaches can be identified (Hartig and Langegger 2010, Rakhmawati et al. 2013)

1. traditional search engines
2. federation of sources
3. discovery-driven query engines

Let's next discuss and highlight the pros/cons of these approaches

Traditional search engines

There are several applications of the search paradigm to discovery of LOD (e.g., Sindice and Falcons). Here, the web of data is crawled and indexed, with various limited query interfaces provided to clients

Traditional search engines

There are several applications of the search paradigm to discovery of LOD (e.g., Sindice and Falcons). Here, the web of data is crawled and indexed, with various limited query interfaces provided to clients

pros.

- coverage is quite good, potentially including a significant portion of the web of data
- response times and throughput are quite high
- data is relatively up to date

cons.

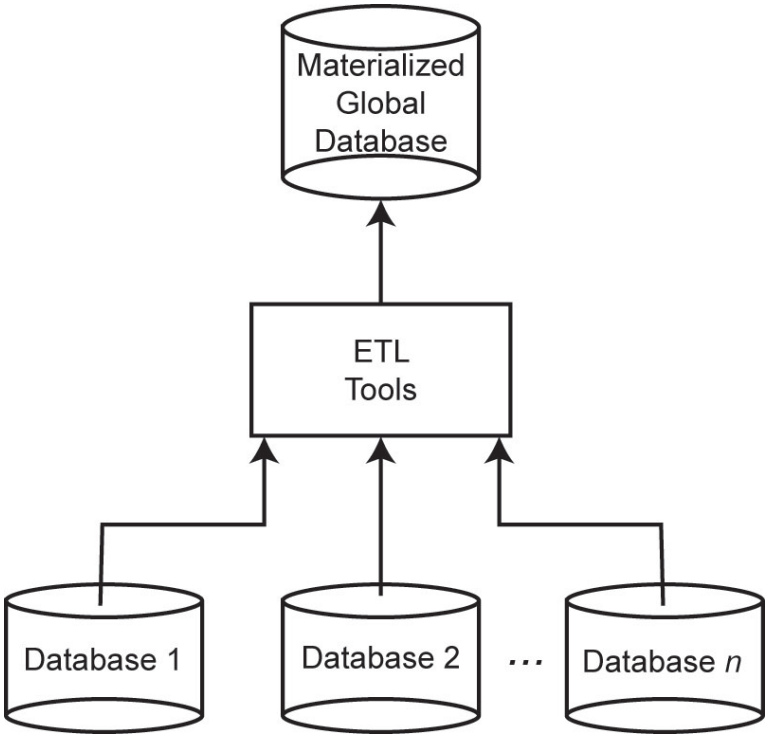
- precision and recall can both be too low for some applications
- dynamic data sources can grow stale, depending on frequency of crawls
- limited query access, client must do query processing

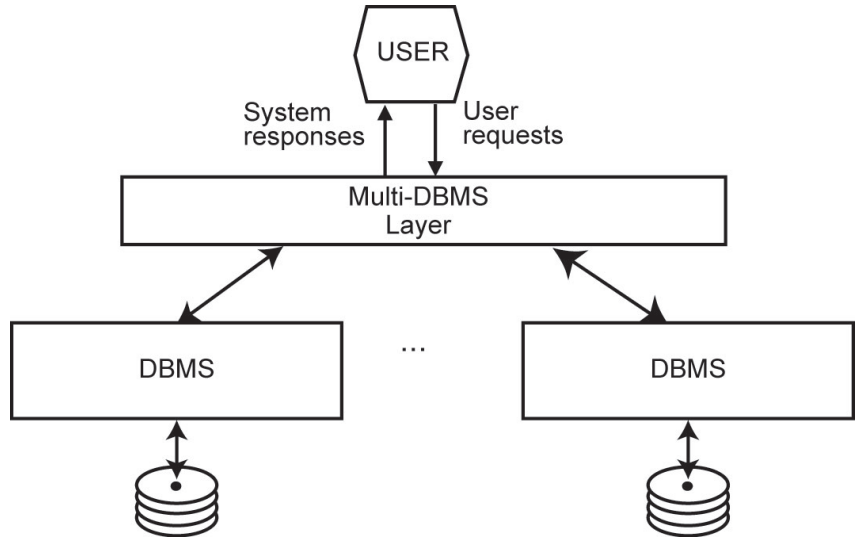
Federation of sources

Also traditionally known as distributed heterogeneous- or multi-databases.

Two basic classes

- data warehouses (materialized federation)
- mediator systems (logical federation)





Federation of sources

pros.

- response times and throughput are quite high (for materialized federations)
- data is up to date (for mediated federations)
- precision and recall are both high
- richer query access languages

cons.

- response times and throughput can be quite low (for mediated federations)
- data can be out of date (for materialized federations)
- coverage is quite poor, limited to known and integrated sources
- query evaluation costs can grow (for mediated federations)

Link-traversal-based query execution

The LTBQE approach closely follows the online navigational models of HF.

This is a new approach that is still in an early stage of study. The basic idea is to interleave query pattern matching with navigation and retrieval of LOD resources

- resources might be seeded, or just based on URIs in the query

Link-traversal-based query execution

The LTBQE approach closely follows the online navigational models of HF.

This is a new approach that is still in an early stage of study. The basic idea is to interleave query pattern matching with navigation and retrieval of LOD resources

- resources might be seeded, or just based on URIs in the query

In general, solution strategies of first research proposals focus engineering effort along three axes (Hartig 2013)

- how data sources are selected (live/cold-start or index-driven)
- how data sources are ranked (scoring, and selecting top-k)
- how data retrieval and result construction are intertwined (separate retrieval/construction, or fully intertwined)

Link-traversal-based query execution

pros.

- data is up to date
- coverage can be quite high
- precision is high
- richer query access languages

cons.

- response times and throughput can be quite low
- query evaluation costs can grow

References & Credits, 1/2

- Querying semantic data on the web. Marcelo Arenas, Claudio Gutierrez, Daniel P. Miranker, Jorge Perez, and Juan Sequeda. *SIGMOD Record* 41(4): 6-17, 2012.

  http://www.sigmod.org/publications/sigmod-record/1212/pdfs/03.principles.arenas.pdf

- Queries and computation on the web. Serge Abiteboul and Victor Vianu. *Theor. Comput. Sci.* 239(2):231-255, 2000. http://dx.doi.org/10.1016/S0304-3975(99)00221-2

- A formal model of queries on interlinked RDF graphs. Paolo Bouquet, Chiara Ghidini, and Luciano Serafini. *AAAI Spring Symposium 2010.*

  http://www.aaai.org/ocs/index.php/SSS/SSS10/paper/viewFile/1185/1441

- An overview on execution strategies for linked data queries. Olaf Hartig. *Datenbank-Spektrum* 13(2):89-99, 2013.

  https://cs.uwaterloo.ca/~ohartig/files/Hartig_LDQueryExec_DBSpektrum2013_Preprint.pdf

References & Credits, 2/2

- A database perspective on consuming linked data on the web. Olaf Hartig and Andreas Langegger. *Datenbank-Spektrum* 10(2):57-66, 2010.

  https://cs.uwaterloo.ca/~ohartig/files/Hartig_QueryingLD_DBSpektrum_Preprint.pdf

- Foundations of traversal based query execution over linked data. Olaf Hartig and Johann-Christoph Freytag. *HT 2012*. http://arxiv.org/abs/1108.6328

- Formal models of web queries. Alberto O. Mendelzon and Tova Milo. *Inf. Syst.* 23(8):615-637, 1998. ftp://ftp.db.toronto.edu/db/papers/infosysMM.ps.gz

- Querying over federated SPARQL endpoints - a state of the art survey. Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, and Michael Hausenblas. DERI Technical Report 2013-06-07, 2013.

  http://arxiv.org/abs/1306.1723

- Eight fallacies when querying the web of data. Jürgen Umbrich, Claudio Gutierrez, Aidan Hogan, Marcel Karnstedt, and Josiane Xavier Parreira. *DESWEB 2013*. http://www.deri.ie/sites/default/files/publications/fallacies.pdf

- The ACE theorem for querying the web of data. Jürgen Umbrich, Claudio Gutierrez, Aidan Hogan, Marcel Karnstedt, and Josiane Xavier Parreira. *WWW 2013*. http://www.deri.ie/sites/default/files/publications/www13-poster.pdf

Recap

1. Introduction and overview of linked and graph data, in the context of related research in data science

2. Models for graph and linked data
   - RDF and RDFS data model
   - graph database models

3. Introduction to querying over graph and linked data.
   - query languages for graphs
   - SPARQL 1.1
   - models for linked data
   - querying linked data

Looking ahead

Next week

- Performance tuning, course summary, and exam review (Wednesday)
- First batch of project presentations: teams 1-8 (Friday)
- Physical hand-in of written assignment (Friday)