Triplestores

Oscar Romero
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Knowledge Graphs

- Knowledge graphs, also known as semantic graphs, refer to RDF, RDFS and OWL
  - They are typically used to share knowledge
  - Do not look at them as traditional databases!
- Unlike property graphs, they are grounded in logics and provide unambiguous semantics
- Formally speaking:
  - RDF provides basic constructs (IRIs-based, triples and the notion of graph), while RDFS and OWL provide means to express rich constraints
  - One say a knowledge graph is an ontology if:
    - Clearly distinguishes instances (ABOX) from terminology or model (TBOX)
    - Provides inference means

Following this definition, realize that RDF(S) graphs can also be considered ontologies

Storing Knowledge Graphs

- Semantic database modeling focuses on providing:
  - Graph structures,
  - Attached semantics to the graph structures
- Two main alternatives
  - Relational implementation (Triplestores)
  - Graph native representation (Graph databases)

Semantic Databases

TRIPLESTORES

Basic Components

- Triplestores (or semantic databases) store knowledge graphs
- A triplestore must contain:
  - A database to **store** knowledge graphs
  - A SPARQL engine to **query** the graphs
    - Typically, in the form of an endpoint (through HTTP)
  - A layer providing basic semantic services
    - Pre-loaded vocabularies (e.g., RDF, RDFS, OWL, etc.)
    - Graphical exploratory interfaces
    - Etc.

STORAGE IN TRIPLESTORES

Storage

- First triplestores stored data on relational databases
  - A single table with three fields
    - Subject, Predicate, Object
  - They do not scalate much

⟨ :Dupond :Leads :CSDept ⟩
⟨ :Dupond :TeachesIn :UE111 ⟩
⟨ :Dupond :TeachesTo :Pierre ⟩
⟨ :Pierre :EnrolledIn :CSDept ⟩
⟨ :Pierre :RegisteredTo :UE111 ⟩
⟨ :UE111 :OfferedBy :CSDept ⟩

Subject	**Predicate**	**Object**
:Dupond	:Leads	:CSDept
:Dupond	:TeachesIn	:UE111
:Dupond	:TeachesTo	:Pierre
:Pierre	:EnrolledIn	:CSDept
:Pierre	:RegisteredTo	:UE111
:UE111	:OfferedBy	:CSDept

Partitioning

- To avoid a extremely large triple-table, triplestores allow to partition it
- They introduce the notion of *graph name*
- A quadstore, is a system allowing to store quadtriples

  <graph name, subject, predicate, object>
- Internally, for each distinct graph name, a relational table is created. Thus, from a database point of view, quadstores boil down to using horizontal partitioning

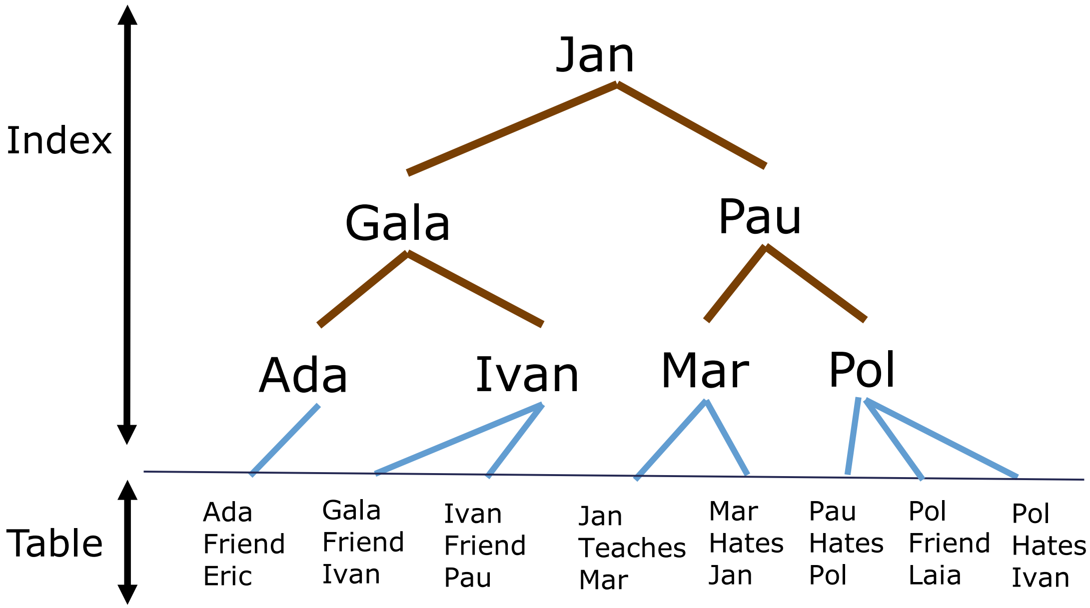
Quadstores: Example

GraphName	Subject	Predicate	Object
:Employer	:Ada	:Age	24
:Employer	:Gala	:From	:BCN
:Employer	:Jan	:WorksFor	:Ada
:Product	:Product1	:isProvided	:Provider6
:Product	:Product2	:isProvided	:Provider5
:Product	:Product3	:isProvided	:Provider4
:Product	:Product1	:SoldTo	:Client1
:Product	:Product1	owl:SameAs	:Product2
:Product	:Product2	:SoldIn	:China

Indexing in Triplestores

- Indexing by Subject (S)

Subject	Predicate	Object
:Ada	:Friend	:Eric
:Gala	:Friend	:Ivan
:Jan	:Teaches	:Mar
:Pol	:Hates	:Ivan
:Pau	:Hates	:Pol
:Mar	:Hates	:Jan
:Ivan	:Friend	:Pau
:Pol	:Friend	:Laia



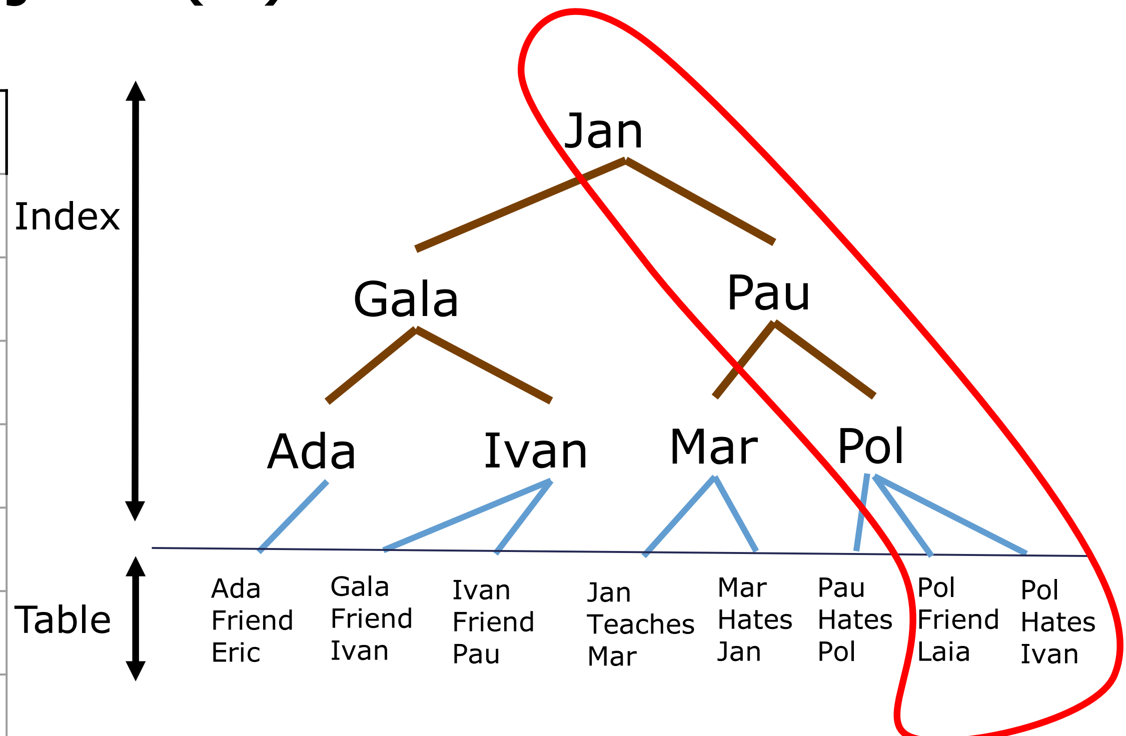
Query

SELECT ?p ?o
WHERE :Pol ?p ?o .

Indexing in Triplestores

- Indexing by Subject (S)

Subject	Predicate	Object
:Ada	:Friend	:Eric
:Gala	:Friend	:Ivan
:Jan	:Teaches	:Mar
:Pol	:Hates	:Ivan
:Pau	:Hates	:Pol
:Mar	:Hates	:Jan
:Ivan	:Friend	:Pau
:Pol	:Friend	:Laia



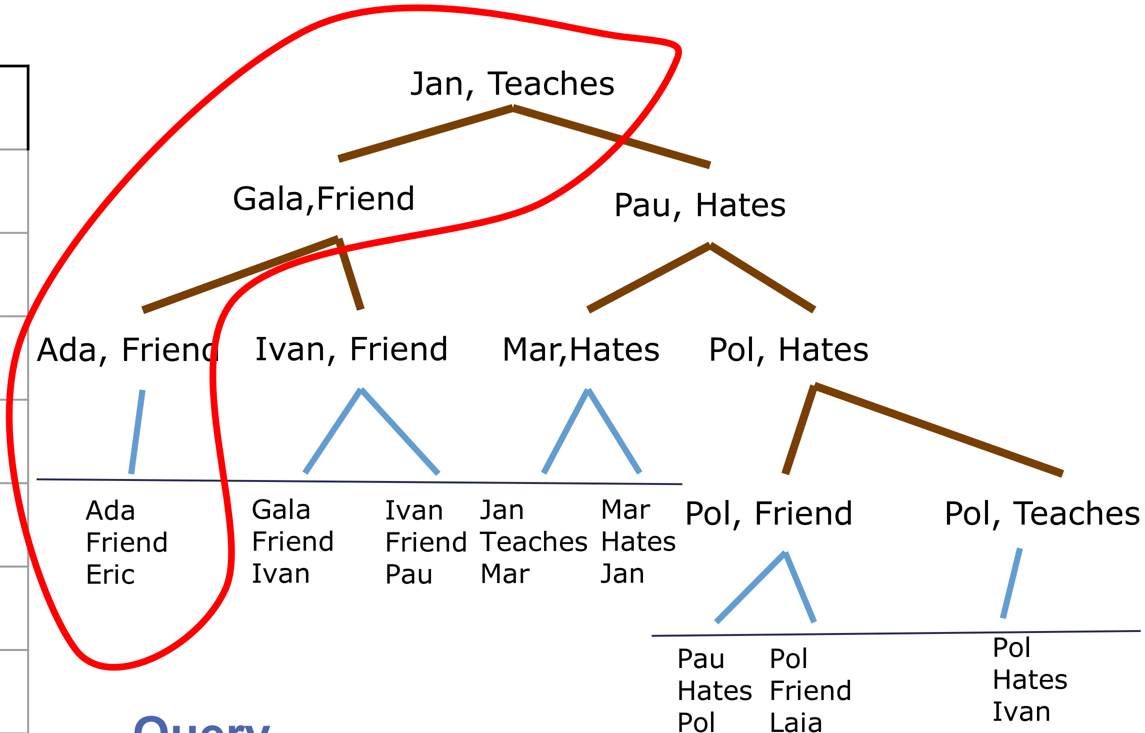
Query

SELECT ?p ?o
WHERE :Pol ?p ?o .

Indexing in Triplestores

- Indexing by Subject and Predicate (SP)

Subject	Predicate	Object
:Ada	:Friend	:Eric
:Gala	:Friend	:Ivan
:Jan	:Teaches	:Mar
:Pol	:Hates	:Ivan
:Pau	:Hates	:Pol
:Mar	:Hates	:Jan
:Ivan	:Friend	:Pau
:Pol	:Friend	:Laia



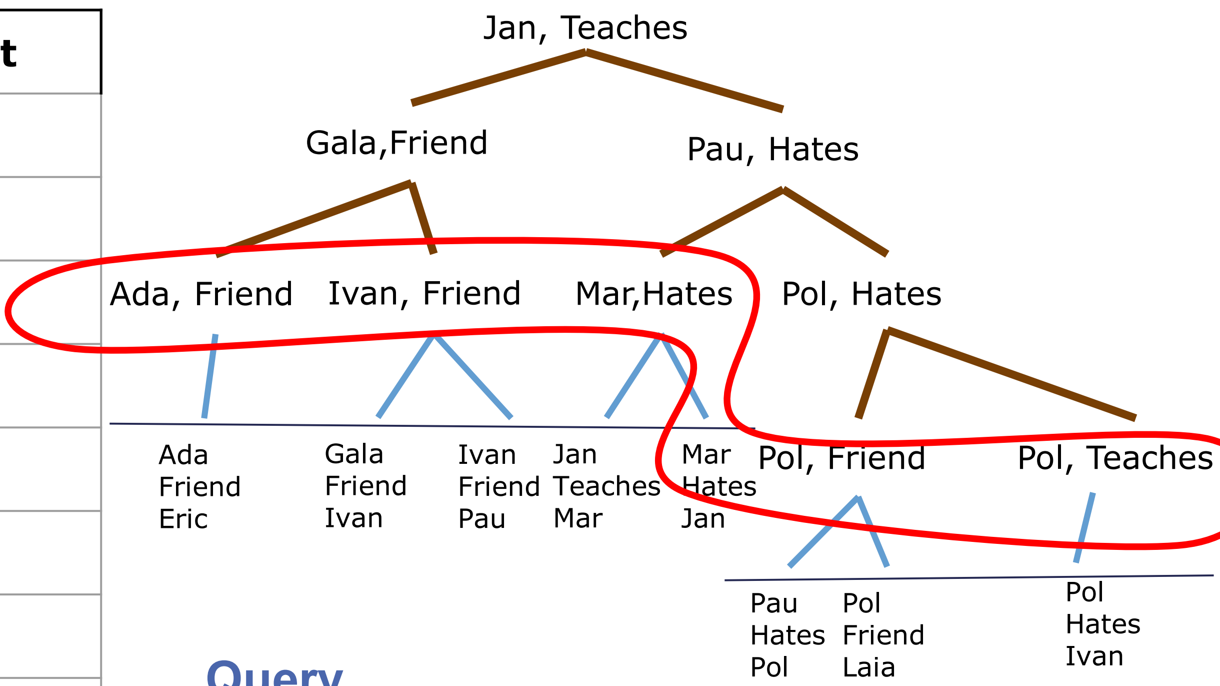
Query

SELECT ?o
WHERE :Ada :Friend ?o .

Indexing in Triplestores

- Indexing by Subject and Predicate (SP)

Subject	Predicate	Object
:Ada	:Friend	:Eric
:Gala	:Friend	:Ivan
:Jan	:Teaches	:Mar
:Pol	:Hates	:Ivan
:Pau	:Hates	:Pol
:Mar	:Hates	:Jan
:Ivan	:Friend	:Pau
:Pol	:Friend	:Laia



Query

SELECT ?s ?o
WHERE ?s :Friend ?o .

Indexing in Triplestores

- Following the same idea we can perform IOQA for any query by indexing the following seven combinations and **intersecting their results**:
  - S
  - O
  - P
  - SO
  - SP
  - OP
  - SOP

Multi-attribute indexes

IOQA (Index-Only Query Answering)

- **Pros**: read-only queries are extremely fast (specially IOQA)
- **Cons**: Maintenance of the indexes (inserts, updates, deletes plus additional memory required)

SPO and Index-Only Query Answering

- A well-known technique were queries are answered with indexes without accessing the table
  - **Principle**: full or partial index scans are better than full or partial table scans

- IOQA Example:

TABLE: *SURVEY(<u>id</u>, city, age, answer1, answer2, answer3)*
INDEX: CREATE INDEX i1 ON survey(city, age) -- this creates a B+

Q: *SELECT city, MIN(age), MAX(age), COUNT(*) FROM survey GROUP BY city*

<u>This query can be accessed by means of a full-scan on i1</u>

QUERY ENGINES IN TRIPLESTORES

Querying Knowledge Graphs

- Triplestores must provide a semantic query engine
  - SPARQL as *de facto* standard to query RDF(S) and OWL nowadays
    - The engine is exposed as an endpoint through the HTTP protocol
  - Some triplestores allow to plug reasoners for querying

Querying Knowledge Graphs

SPARQL

- Works under the closed-world assumption unless a regime entailment is activated
- Two main regimes:
  - RDFS
  - OWL 2 DL / OWL 2 QL*
- Even if you activate a regime entailment, SPARQL does the following trick:
  - Materializes all the inferences unfolding all implicit knowledge entailed by the activated regime **entailment rules**
  - Queries using the closed-world assumption

Reasoners

Querying Knowledge Graphs

SPARQL

- Works under the closed-world assumption unless a regime entailment is activated
- Two main regimes:
  - RDFS
  - OWL 2 DL
- Even if you activate a regime entailment, SPARQL does the following trick:
  - Materializes all the inferences unfolding all implicit knowledge entailed by the activated regime **entailment rules**
  - Queries using the closed-world assumption

Reasoners

- Works under the open-world assumption
- Grounded in logics
- Performs **ontology reasoning**
  - Uses tableaux and similar pruning algorithms to infer **logical consequences**
- If you want to unfold all the inference power of Description Logics when using OWL you need a reasoner!

Constraining Knowledge Graphs

- Shape Constraining Language (SHACL) https://www.w3.org/TR/shacl/ allows to state constraints over graph data (equivalent to ASSERTIONS, CHECK, PK, FK, procedures and triggers in the relational model).
- Constraints are declared in the form of triples and constraint the shape of the graph (i.e., **conformance checking**). Examples: https://www.w3.org/TR/shacl/#shacl-example
- Importantly, reasoning and inference unfold new knowledge or identify contradictions, but do not provide means for conformance checking
  - SHACL together with inference has not been fully studied (ongoing work)
- Already supported by some tools such as Jena and GraphDB

Current state of the art of the market

KNOWLEDGE GRAPH RELATED SW TOOLS

Tools

More at: https://www.w3.org/wiki/SemanticWebTools

- We may distinguish:
  - **Triplestores**: Openlink Virtuoso, Ontotext GraphDB, Apache Rya, GRAKN.AI, Oracle graph DB, Stardog, Amazon Neptune, etc.
  - **Processing frameworks**: Apache Jena (it has a simple built-in triplestore: TDB), RDF4J, RDFLib, Protégé, Pool Party, etc.
    - Provide libraries to manipulate RDF(S) and OWL at the application level. They have connectors to databases, reasoners, etc.
  - **ETL tools**: Data Lens, data.world, AWS DMS, Amazon Comprehend, Pool Party, etc.
  - **Reasoners**:
    - Stand-alone: Ontop (OWL and RDFS), Hermit, etc.
    - Plug-ins or libraries: Pellet, Racer, Fact++, etc.

Future Trends

- Native semantic graph databases (i.e., not storing graphs as relational databases) are a current trend
  - They benefit from all pros and cons of graph databases (see slides on graph databases)
  - A promising tool in this direction is ontotext GraphDB
- Combining analytics and reasoning in the same tools
  - However, only academic prototypes or discussions up to now
  - Amazon Neptune combines both worlds but, up to now, without graph algorithms

Expected kind of Questions

- Consider a knowledge graph with 5 milion nodes and 25 milion edges. We know that:
  - The average node order is 5
  - Most queries to be performed are point-queries starting from 1 to 50 nodes and *hoping* 2-3 times in the most usual pattern matching queries
- Briefly justify if you would choose a relational (e.g., Openlink Virtuoso) or native graph-based triplestore (e.g., Ontotext GraphDB)
- In case you choose a relational triplestore, what indexing strategy you would follow for a given set of queries?

Summary

- Triplestores
  - Storage
    - Graph-oriented vs. relational
    - Partitioning (i.e., named graphs and quadstores)
    - Indexing strategies in relational triplestores
  - Querying
    - SPARQL Rengime Entailments Vs. Reasoners
    - SHACL
- Current tool market
  - Relational Vs. Graph-based implementations