Contents

• Introduction to Semantic Technologies
• Semantic Databases – advantages, features and benchmarks
• Semantic Technologies and Triplestores for BI
INTRODUCTION TO SEMANTIC TECHNOLOGIES
The need for a smarter Web

• "The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation." (Tim Berners-Lee, 2001)
The need for a smarter Web (2)

• “PricewaterhouseCoopers believes a Web of data will develop that fully augments the document Web of today. You’ll be able to find and take pieces of data sets from different places, aggregate them without warehousing, and analyze them in a more straightforward, powerful way than you can now.” (PWC, May 2009)
The Semantic Web vision (W3C)

- Extend principles of the Web from documents to data
- Data should be accessed using the general Web architecture (e.g., URI-s, protocols, ...)
- Data should be related to one another just as documents are already
- Creation of a common framework that allows:
  - Data to be shared and reused across applications
  - Data to be processed automatically
  - New relationships between pieces of data to be inferred
The Semantic Web stack

The Semantic Web Technology Stack (not a piece of cake...)

- Most apps use only a subset of the stack
- Querying allows fine-grained data access
- Standardized information exchange is key
- Formats are necessary, but not too important
- The Semantic Web is based on the Web
- Linked Data uses a small selection of technologies

(c) Benjamin Nowack
The Semantic Web timeline

- RDF
- DAML+OIL
- OWL
- SPARQL
- RDFa
- RIF
- SAWSDL
- LOD
- SKOS
- HCLS
- RDB2RDF
- GLD
- PIL

Ontologies as data models on the Semantic Web

• An ontology is a *formal* specification that provides sharable and reusable knowledge representation
  – Examples – taxonomies, thesauri, topic maps, formal ontologies

• An ontology specification includes
  – Description of the *concepts* in some domain and their properties
  – Description of the possible *relationships* between concepts and the *constraints* on how the relationships can be used
  – Sometimes, the *individuals* (members of concepts)
Resource Description Framework (RDF)

- A simple data model for
  - Formally describing the *semantics* of information
  - representing meta-data (data about data)
- A set of representation syntaxes
  - RDF/XML (standard), N-Triples, N3
- Building blocks
  - *Resources* (with unique identifiers)
  - *Literals*
  - Named *relations* between pairs of resources (or a resource and a literal)
RDF (2)

- Everything is a triple
  - **Subject** (resource), **Predicate** (relation), **Object** (resource or literal)
- The RDF graph is a collection of triples

![Graph Diagram]

subject \(\rightarrow\) predicate \(\rightarrow\) object

École Centrale Paris \(\xrightarrow{\text{locatedIn}}\) Paris

Paris \(\xrightarrow{\text{hasPopulation}}\) 2193031
RDF graph example (3)

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Paris">http://dbpedia.org/resource/Paris</a></td>
<td>hasName</td>
<td>“Paris”</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Paris">http://dbpedia.org/resource/Paris</a></td>
<td>hasPopulation</td>
<td>2193031</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/%C3%89cole_centrale_Paris">http://dbpedia.org/resource/École_centrale_Paris</a></td>
<td>hasName</td>
<td>“École Centrale Paris”</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/%C3%89cole_centrale_Paris">http://dbpedia.org/resource/École_centrale_Paris</a></td>
<td>establishedIn</td>
<td>1829</td>
</tr>
</tbody>
</table>
RDF advantages

• Global identifiers of all resources (URIs)
  – Reduces ambiguity
  – Makes incremental data integration easier

• Graph data model
  – Suitable for sparse, unstructured and semi-structured data

• Inference of implicit facts

• Schema agility
  – Lowers the cost of schema evolution
RDF Schema (RDFS)

- RDFS provides means for:
  - Defining *Classes* and *Properties*
  - Defining hierarchies (of classes and properties)
  - Domain/range of a property
- Entailment rules (axioms)
  - Infer new triples from existing ones
RDFS entailment rules

1: $s p o$ (if $o$ is a literal)  \Rightarrow \_n \text{ rdf:type rdfs:Literal}
2: $p \text{ rdfs:domain } x$ & $s p o$  \Rightarrow s \text{ rdf:type } x
3: $p \text{ rdfs:range } x$ & $s p o$  \Rightarrow o \text{ rdf:type } x
4a: $s p o$  \Rightarrow s \text{ rdf:type rdfs:Resource}
4b: $s p o$  \Rightarrow o \text{ rdf:type rdfs:Resource}
5: $p \text{ rdfs:subPropertyOf } q \& q \text{ rdfs:subPropertyOf } r$  \Rightarrow p \text{ rdfs:subPropertyOf } r
6: $p \text{ rdf:type rdfs:Property}$  \Rightarrow p \text{ rdfs:subPropertyOf } p
7: $s p o$ & $p \text{ rdfs:subPropertyOf } q$  \Rightarrow s q o
8: $s \text{ rdf:type rdfs:Class}$  \Rightarrow s \text{ rdfs:subClassOf rdfs:Resource}
9: $s \text{ rdf:type } x$ & $x \text{ rdfs:subClassOf } y$  \Rightarrow s \text{ rdf:type } y
10: $s \text{ rdf:type rdfs:Class}$  \Rightarrow s \text{ rdfs:subClassOf } s
11: $x \text{ rdfs:subClassOf } y$ & $y \text{ rdfs:subClassof } z$  \Rightarrow x \text{ rdfs:subClassOf } z
12: $p \text{ rdf:type rdfs:ContainerMembershipProperty}$  \Rightarrow p \text{ rdfs:subPropertyOf rdfs:member}
13: $o \text{ rdf:type rdfs:Datatype}$  \Rightarrow o \text{ rdfs:subClassOf rdfs:Literal}
RDF entailment rules (2)

- **Class/Property hierarchies**
  - R5, R7, R9, R11

  ```
  :human rdfs:subClassOf :mammal .
  :man rdfs:subClassOf :human .
  ➞ :man rdfs:subClassOf :mammal .
  :John a :man .
  ➞ :John a :human .
  ➞ :John a :mammal .
  :hasSpouse rdfs:subPropertyOf :hasRelative .
  :John :hasSpouse :Merry .
  ➞ :John :hasRelative :Merry .
  :hasSpouse rdfs:domain :human ;
  rdfs:range :human .
  :Adam :hasSpouse :Eve .
  ➞ :Adam a :human .
  ➞ :Eve a :human .
  ```

- **Inferring types (domain/range restrictions)**
  - R2, R3
Web Ontology Language (OWL)

• More expressive than RDFS
  – Identity equivalence/difference
    • sameAs, differentFrom, equivalentClass/Property

• Complex class expressions
  – Class intersection, union, complement, disjointness

• More expressive property definitions
  – Object/Datatype properties
  – Cardinality restrictions
  – Transitive, functional, symmetric, inverse properties
• Identity equivalence
  \[\text{db1:Paris} : \text{hasPopulation} 2913031.\]
  \[\text{db1:Paris} = \text{db2:Paris} .\]
  \[\Rightarrow \text{db2:Paris} : \text{hasPopulation} 2193031 .\]

• Transitive properties
  \[\text{:locatedIn} \text{a owl:TransitiveProperty} .\]
  \[\text{:ECP :locatedIn :Paris} .\]
  \[\text{:Paris :locatedIn :France} .\]
  \[\Rightarrow \text{:ECP :locatedIn :France} .\]

• Symmetric properties
  \[\text{:hasSpouse} \text{a owl:SymmetricProperty} .\]
  \[\text{:John :hasSpouse :Merry} .\]
  \[\Rightarrow \text{:Merry :hasSpouse :John} .\]

• Inverse properties
  \[\text{:hasParent owl:inverseOf :hasChild} .\]
  \[\text{:John :hasChild :Jane} .\]
  \[\Rightarrow \text{:Jane :hasParent :John} .\]

• Functional properties
  \[\text{:hasSpouse a owl:FunctionalProperty} .\]
  \[\text{:Merry :hasSpouse :John} .\]
  \[\Rightarrow \text{:JohnSmith = :John} .\]
OWL sublanguages

• OWL Lite
  – low expressivity / low formal complexity
  – Logical decidability & completeness
  – All RDFS features
  – `sameAs/differentFrom`, equivalent class/property
  – inverse/symmetric/transitive/functional properties
  – cardinality restriction (only 0 or 1)
  – class intersection
OWL sublanguages (2)

- **OWL DL**
  - high expressivity / efficient DL reasoning
  - Logical decidability & completeness
  - All OWL Lite features
  - Class disjointness
  - Complex class expressions
  - Class union & complement

- **OWL Full**
  - max expressivity / no efficient reasoning
  - No guarantees for completeness & decidability
OWL 2 profiles

• Goals
  – sublanguages that trade expressiveness for efficiency of reasoning
  – Cover specific important application areas
  – Easier to understand by non-experts

• **OWL 2 EL**
  – Best for large ontologies / small instance data (TBox reasoning)
  – Computationally optimal
    • PTime reasoning complexity
OWL 2 profiles (2)

• **OWL 2 QL**
  - Quite limited expressive power, but very efficient for query answering with large instance data
  - Can exploit query rewriting techniques
    • Data storage & query evaluation can be delegated to a RDBMS

• **OWL 2 RL**
  - Balance between scalable reasoning and expressive power
  - Suitable for rule-based reasoning
OWL 2 profiles (3)

OWL 2 Full

OWL 2 DL

OWL 2 Full

Undecidable

2NExpTime-Complete

OWL 2 DL

NExpTime-Complete

SROIQ

OWL 1 DL

PWTime-Complete

SHOIN

In AC^0

OWL 2 QL

EL++

DL-Lite

OWL 2 RL

OWL 2 EL

(c) Axel Polleres
SPARQL Protocol and RDF Query Language (SPARQL)

- SQL-like query language for RDF data
- Simple protocol for querying remote databases over HTTP
- Query types
  - `select` – query data by complex graph patterns
  - `ask` – whether a query returns results (result is true/false)
  - `describe` – returns all triples about a particular resource
  - `construct` – create new triples based on query results
Graph patterns

- Whitespace separated list of **Subject, Predicate, Object**
  - \(?x\) `dbp-ont:city dbpedia:Paris`
  - `dbpedia:École_centrale_Paris db-ont:city ?y`

- **Group Graph Pattern**
  - A group of 1+ graph patterns
  - FILTERs can constrain the whole group

```{?
?uni a dbpedia:University ;
  dbp-ont:city dbpedia:Paris ;
  dbp-ont:numberOfStudents ?students .
FILTER (?students > 5000)
}
```
Graph Patterns (2)

- **Optional Graph Pattern**
  - Optional parts of a pattern
  - *pattern OPTIONAL {pattern}*

```sparql
SELECT ?uni ?students
WHERE {
  ?uni a dbpedia:University ;
  OPTIONAL {
    ?uni dbp-ont:numberOfStudents ?students
  }
}
```
Graph Patterns (3)

- **Alternative Graph Pattern**
  - Combine results of several alternative patterns
  - `{pattern} UNION {pattern}

```sparql
SELECT ?uni
WHERE {
  ?uni a dbpedia:University .
  {
    { ?uni dbp-ont:city dbpedia:Paris }
    UNION
    { ?uni dbp-ont:city dbpedia:Lyon }
  }
}
```
Anatomy of a SPARQL query

• List of namespace prefixes
  – PREFIX xyz: <URI>

• Query result clause (variables)
  – ?x, $y

• Datasets

• Graph patterns + filters
  – Simple / group / alternative / optional

• Modifiers
  – ORDER BY, DISTINCT, OFFSET/LIMIT
Linked Data

• “To make the Semantic Web a reality, it is necessary to have a large volume of data available on the Web in a standard, reachable and manageable format. In addition the relationships among data also need to be made available. This collection of interrelated data on the Web can also be referred to as Linked Data. Linked Data lies at the heart of the Semantic Web: large scale integration of, and reasoning on, data on the Web.” (W3C)

• Linked Data is a set of principles that allows publishing, querying and browsing of RDF data, distributed across different servers
  • similar to the way HTML is currently published & consumed
Linked Data design principles

1. Unambiguous identifiers for objects (resources)
   – Use URIs as names for things

2. Use the structure of the web
   – Use HTTP URIs so that people can look up the names

3. Make it easy to discover information about an object (resource)
   – When someone lookups a URI, provide useful information

4. Link the object (resource) to related objects
   – Include links to other URIs
Linked Data evolution – Sep 2008

(c) R. Cyganiak & A. Jentzsch

As of September 2008
Linked Data evolution – Jul 2010

Semantic Technologies & Triplestores for BI (eBISS 2011)

(c) R. Cyganiak & A. Jentzsch
Why use Linked Data?

- Facilitate data integration
  - Use LOD as an “interlingua” for EDI
    - Additional public information can help alignment and linking
- Add value to proprietary data
  - Public data can allow enhanced content and more analytics on top of proprietary data
    - E.g. linking to spatial data from GeoNames, search for images
  - Better description and access to content
- Make enterprise data more open & accessible
  - Public identifiers and vocabularies can be used to access them
SEMANTIC DATABASES (TRIPLESTORES)
Triplestores

• RDF databases
  – Store data according to the RDF data model
  – Provide inference of implicit triples (either at data loading time, or at query time)
  – SPARQL as a query language
• Many similarities to traditional DBMS approaches
  – ... and many differences too
## Triplestores vs. traditional DBMS

<table>
<thead>
<tr>
<th>Feature</th>
<th>Triplestore</th>
<th>OLTP</th>
<th>OLAP</th>
<th>NoSQL</th>
<th>Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update performance</td>
<td>+/-</td>
<td>+</td>
<td>+/-</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Complex Queries</td>
<td>+</td>
<td>+/-</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
</tr>
<tr>
<td>Inference</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sparse data</td>
<td>+</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Semi-structured / unstructured data</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+</td>
</tr>
<tr>
<td>Dynamic schema</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+/-</td>
<td>+/-</td>
</tr>
</tbody>
</table>
Triplestore advantages

• Global identifiers of resources (entities)
  – Lowers the cost of data integration
• Inference of implicit facts
• Graph data model
  – Suitable for sparse, semi-structured and unstructured data
• Agile schema
  – New relations between entities may be easily added
• Exploratory queries against unknown schema
  – Query and data vocabulary may differ
• Compliance to standards (RDF, SPARQL)
Design & Implementation

- **Storage engine**
  - Native
  - on top of a RDBMS
  - on top of NoSQL engine

- **Triple density**
  - The in-memory “footprint” per triple may differ x10 between different triplestores
  - Impact on TCO

- **Compression**
  - Improves I/O on multi-core systems
• Reasoning strategy
  - *Forward-chaining* – at data loading time, start from the explicit facts and apply the inference rules until the complete closure is inferred
  - *Backward-chaining* – at runtime, start with a query and decompose recursively into smaller requests that can be matched to explicit facts
  - *Hybrid strategy* – partial materialization at data loading time + partial query pattern matching at runtime
Pros and cons of forward-chaining based materialization

- Relatively **slow addition** of new facts
  - inferred closure is extended after each transaction
- **Deletion** of facts is **slow**
  - facts that are no longer true are removed from the inferred closure
- The **maintenance of the inferred closure** requires more resources
- **Querying and retrieval** are fast
  - no reasoning is required at query time
  - RDBMS-like query evaluation & optimisation techniques are applicable
Design & Implementation (3)

• Invalidation strategy
  – Truth maintenance is not trivial
  – huge overhead for keeping meta-data about inference dependencies
  – Trivial approach: just re-compute the complete inferred closure after a deletion
  – Advanced approach: detect which parts of the inferred closure are invalid and need to be deleted as well
Design & Implementation (4)

• **owl:sameAs** optimization
  – It is a transitive, reflexive and symmetric relationship
  – _owl:sameAs_ induced inference can “inflate” the number of statements and deteriorate inference/query performance
  – Specific optimizations allow that a compact representation of equivalent resources is used
  – Query results can be expanded through backward-chaining at query time
Popular triplestores

- 4store
  - http://4store.org
  - *Open source*, distributed cluster (up to 32 nodes), data fully partitioned, no inference (external reasoner, backward chaining)

- AllegroGraph
  - http://www.franz.com
  - ACID transactions, RDF and limited OWL reasoning, full-text indexing, compression, replication cluster, backward chaining
Popular triplestores (2)

- **Bigdata**
  - [http://www.systap.com](http://www.systap.com)
  - Open source, data partitioning, hybrid materialization, RDF and limited OWL reasoning
- **Dydra**
  - [http://dydra.com](http://dydra.com)
  - SaaS, SPARQL endpoint + REST API, no reasoning
- **Jena TDB**
  - [http://www.openjena.org/TDB](http://www.openjena.org/TDB)
  - Open source, RDF and limited OWL reasoning
Popular triplestores (3)

- **Oracle**
  - RDF and limited OWL reasoning, data partitioning & compression (RAC), `owl:sameAs` optimization, security & versioning; geo-spatial extensions

- **OWLIM**
  - [http://www.ontotext.com](http://www.ontotext.com)
  - Forward-chaining, RDF / OWL 2 RL / OWL 2 QL and limited OWL Lite / OWL DL reasoning; replication cluster; `owl:sameAs` optimization; full-text indexing; geo-spatial extensions; scalable RDF Rank
Popular triplestores (4)

- **Sesame**
  - [http://www.openrdf.org](http://www.openrdf.org)
  - Open source; plugable storage & inference layer

- **StarDog**
  - [http://stardog.com](http://stardog.com)
  - backward-chaining; OWL DL and all OWL 2 profiles; full-text indexing; compression

- **Virtuoso**
  - [http://virtuoso.openlinksw.com](http://virtuoso.openlinksw.com)
  - Universal server (RDF, XML, RDBMS); backward chaining; geo-spatial extensions; full-text indexing
Benchmarking

- Tasks
  - Data loading
  - Query evaluation
  - Data modification

- Performance factors
  - Forward-chaining vs. backward-chaining
  - Data model complexity
  - Query complexity
  - Result set size
  - Number of concurrent clients
Popular benchmarks for triplestores

- **LUBM**
  - Storing & query performance benchmark
  - 14 predefined queries + data generator tool

- **BSBM**
  - SPARQL benchmark
  - 3 use cases (12/17/8 distinct queries)

- **SP²Bench**
  - SPARQL benchmark
  - 12 queries for most common SPARQL constructs & access patterns
SEMANTIC TECHNOLOGIES & TRIPLESTORES FOR BI
Data integration & querying (HCLS)

distributed querying at present

distributed querying with RDF and SPARQL

(c) HCLS @ W3C
Data integration & querying (HCLS)
Data integration cost (PwC)

Cost of using data sources

Number of data sources

High

Low

Low

High

Traditional data integration process

Ontology-driven process

(c) PriceWaterhouseCooper

Semantic Technologies & Triplestores for BI (eBISS 2011)
Semantic Technologies & Triplestores for BI

- Speed-up data integration
  - RDF based ETL is more agile

- Lower the cost of data integration
  - Initial cost of using ontologies is higher
  - But the cost of ad-hoc ETL will be higher in the long term (too many data sources)

- Align & integrate legacy data silos
  - Querying & consuming data from disparate sources is easier with SPARQL
Semantic Technologies & Triplestores for BI (2)

- Infer implicit & hidden knowledge
  - Custom, user-defined rules as well

- Efficiently manage unstructured & semi-structured data together
  - Graph data model

- Improve the quality of query results
  - Inference of implicit facts
  - SPARQL query vocabulary may differ from data vocabulary
  - Exploratory queries
Questions?

@ontotext