トリプルストア、RDF、SPARQL：今勉強しないと遅れますよ！

2015/04/21, Qcon Tokyo 2015
トルヴェ アントワン (九州大学)
trouve@soc.ait.kyushu-u.ac.jp
## Today’s Technologies

<table>
<thead>
<tr>
<th>Category</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Model</td>
<td><strong>RDF</strong> (Ressource-Description Framework)</td>
</tr>
<tr>
<td>Database Technology</td>
<td><strong>Triple</strong> (Graph) Store</td>
</tr>
<tr>
<td>Database Query Language</td>
<td><strong>SPARQL</strong> (SPARQL Protocol and RDF Language)</td>
</tr>
<tr>
<td>Model /Schema Description Language</td>
<td><strong>RDFS</strong> (RDF Schema), <strong>OWL</strong> (Web Ontology Language)</td>
</tr>
<tr>
<td>Development Paradigm</td>
<td><strong>Model-driven</strong> (development Ad-hoc, Fast Prototyping, Agile)</td>
</tr>
</tbody>
</table>
Disclaimer...

I don’t own any RDF company

I am not making this prevention for you to buy some of my product

I just want to talk about some technology I liked recently
The RDF Data Model
Inception of RDF: Context

- The www is a mine of information
- This is however mostly **unstructured information**
- Such www is useful to us human, because our brain is an outstanding information processor
- It is however **not amenable to machines**
How to Make the Web Amenable to Machines?

Solution 1
Make machines as smart as humans

Solution 2
Structure the information on the www for dumb machines to understand it
How to Make the Web Amenable to Machines?

Solution 1
Make machines as smart as humans

Solution 2
Structure the information on the www for dumb machines to understand it
Which Data Model for the Semantic Web?

Characteristics of the www

The “AAA” principle: Anyone says Anything about Any topic
→ It is vain to force people to use the same data model
Open world: there is someone else that is saying something about the same topic
→ It is important to merge data from different sources
Non unique naming
→ The data will be heterogenous, even when about the same resources
Dynamic, always in beta

x

Available Models

<table>
<thead>
<tr>
<th>Model 1</th>
<th>Model 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational</td>
<td>NoSQL</td>
</tr>
<tr>
<td>Schema</td>
<td>Fixed</td>
</tr>
<tr>
<td>Rigidity</td>
<td>Complex</td>
</tr>
<tr>
<td>Ease to merge</td>
<td>Complete</td>
</tr>
<tr>
<td>Homogeneity</td>
<td>Partial</td>
</tr>
</tbody>
</table>
Key-Value Metadata for the Information on the www

http://city.fukuoka.lg.jp

- **is about** Fukuoka city
- **is a** Web page
- **last seen** 2015-2-1

**Keys**

**Values**

Example of the webpage of Fukuoka city
Resource Description Framework

データはトリプルで表されます

Subject
The resource identifier

Predicate
The key, or property name

Object
The value

Example: from previous slide, as three triple

http://city.fukuoka.lg.jp is about Fukuoka city
http://city.fukuoka.lg.jp is a Web page
http://city.fukuoka.lg.jp last seen 2015-2-1
About the W3C

- Founded in 1994 by Tim Berners-Lee et al.
- Regulates the technologies used in the www
  - HTML, XML, Javascript, CSS, RDF
- RDF is a W3C standard
  - RDF 1.1 has been published in 2014/2/25
  - It consists of several sub-standards, part of which we will see today

- Tim Berners-Lee, head of the W3C.
  - He developed the early version of the www in 1989 (while working at CERN, France)
## A Real RDF Example

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

These addresses are **IRI**: Internationalized Resource Identifier (superset of URI)
### A Real RDF Example

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language/Type</th>
</tr>
</thead>
</table>

**Those IRIs are so hard to read!**
Qname and CURIE: make IRI more readable

http://www.w3.org/2000/01/rdf-schema#type
http://www.w3.org/2000/01/rdf-schema#label

Common prefix

Prefix

Local part
- CURIE: can contain slashes (used in RDFa - see slides later)
- Qname: no slash (used in RDF/XML)
### A Real RDF Example with CURIE

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:lastReviewed</td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“Fukuoka city official homepage”</td>
<td>en</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

I use well-used prefix here. In the real world one should define them before use.
## A Real RDF Example with CURIE

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:lastReviewed</td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“Fukuoka city official homepage”</td>
<td>en</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

I use well-used prefix here. In the real world one should define them before use.

This is much better!

More on that later with turtle and SPARQL.
# Resource’s IRI

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- This IRI is **not a website**, it designate the **city of Fukuoka** itself.
- Everyone is free to decide its own set of unique IRI to designate resources (we use one from the dbpedia project here)
- It is common to use URL for IRI, the domain being used as namespace

I use as IRI the URL of the website
Vocabulary’s IRI

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>One shall use IRI for</td>
<td>schema:WebSite</td>
<td>This IRI designate a type of RDF resources (from schema.org)</td>
</tr>
<tr>
<td></td>
<td>predicates.</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:lastReviewed</td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“Fukuoka city official”</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“福岡市公式ウェブサイト”</td>
<td></td>
</tr>
</tbody>
</table>

- **IRI** are also used to designate well-defined **vocabulary** used to describe the meaning of the data (mostly **predicate**, **types**)
- It is possible to define one’s own vocabulary, but it is often a good idea to **re-use the well established ones**
Duplicated Predicates

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:lastReviewed</td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“Fukuoka city official homepage”</td>
<td>en</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

- It is possible to specify several times the same predicate
- This is a way to model unsorted data collections
Specify the Language and Datatype of Literals

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td>Example of type tag</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td></td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td></td>
<td>“Fukuoka city official homepage”</td>
<td>en</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td></td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

- The XML types are part of the RDF standard:
RDF Graph Representation

It is possible (and common usage) to represent RDF data graphically, as below:

Literals’ tags are represented with “^^” for datatypes, and @ for language, as below:
An Example of Graph

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:type</td>
<td>schema:WebSite</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:about</td>
<td>db:Fukuoka</td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>schema:lastReviewed</td>
<td>“2015-2-1”</td>
<td>xsd:date</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“Fukuoka city official homepage”</td>
<td>en</td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td>rdfs:label</td>
<td>“福岡市公式ホームページ”</td>
<td>ja</td>
</tr>
</tbody>
</table>

http://city.fukuoka.lg.jp

schema:WebSite

db:Fukuoka

“2015-2-1”^^xsd:date

"Fukuoka city official homepage"@en

"福岡市公式ホームページ"@ja
Linked RDF

- By re-using the same IRIs, it possible to express the relationship between resources

What is that supposed to mean?
Linked RDF on an Example (1)

Fukuoka City Homepage

- `http://city.fukuoka.lg.jp`
- `schema:WebSite`
- `schema:about` `db:Fukuoka`
- `schema:lastReviewed` "2015-2-1"^^xsd:date
- `rdfs:label` "Fukuoka city official homepage"@en
- `rdfs:label"福岡市公式ホームページ"@ja`

List of nurseries in Fukuoka

- `schema:WebSite`
- `schema:about` `db:Fukuoka`
- `schema:lastReviewed` "2015-2-1"^^xsd:date
- `rdfs:label` "List of nurseries in Fukuoka"@en
- `rdfs:label"福岡市保育園一覧"@ja""
Linked RDF on an Example (2)

Both pages are about the same resource db:Fukuoka
Linked RDF between Different Data Sources

These are data originated from dbpedia's database (includes longitude, latitude, and much more !!)
Real Example of Linked RDF

This example is from lod-cloud.net
Store RDF Data
How do I Serialize my RDF dataset?

- **N-Triple**
  - Naive representation
- **Turtle / N3**
  - Compact, human-readable
  - Used by SPARQL
- **RDF/XML** (often referred as RDF)
  - Verbose, hard to read but understood by web browser
- **MicroData / RDFa**: RDF annotation for XHTML pages
  - Well-used vocabulary: schema.org
  - Often used for search engine optimization (SEO)
How do I Serialize my RDF dataset?

- **N-Triple**
  - Naive representation

- **Turtle / N3**
  - Compact, human-readable
  - Used by SPARQL

- **RDF/XML** (often referred as RDF)
  - Verbose, hard to read but understood by web browser

- **MicroData / RDFa**: RDF annotation for XHTML pages
  - Well-used vocabulary: schema.org
  - Often used for search engine optimization (SEO)

Schema.org
Consortium of major search engine companies, including Google, Microsoft and Yahoo.
Microdata and RDFa

- Both are techniques to tag information contained in HTML
- Both consist in writing the information right in the HTML code as attributes to HTML elements
  - Microdata: supported by the schema.org consortium (including Google, Microsoft, Yahoo)
  - RDFa (especially RDFa Lite) is the W3C standard

```html
<p about="myself" vocab="http://schema.org/" typeof="Person">
  My name is 
  <span property="name">Antoine Trouvé</span>,
  my phone number is 
  <span property="telephone">xxx-xxxx-xxx</span> 
  and my homepage is 
  <a property="url" href="http://trouve.sakura.ne.jp/"></a>
</p>
```

Example of RDFa information (use vocabulary from schema.org)

Equivalent RDF graph

myself

Antoine Trouvé

xxx-xxxx-xxx

http://trouve.sakura.ne.jp
Work with RDFa / Microdata

• Search engines support
  • Sindice is a search engine for such data, including SPARQL (see later)
  • Google, Yahoo, Bing are able to parse RDFa and Microdata
• Facebook uses RDFa (Open Graph API)
  • Extract information from websites
  • Used by features such as the “Like” button
• Browser support
  • Some browsers feature plugin that read RDFa data, and even allow to query them with SPARQL
• Conversion to/from RDFa and Microdata
  • http://rdf-translator.appspot.com
• Should I use RDFa or Microdata?
  • RDFa enables to express more complex information
  • But Microdata suffices for most tasks
  • Most tools support both, so it does not matter so much!
A First Turtle Example

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
<td>“福岡公式ホームページ”@ja</td>
<td></td>
</tr>
</tbody>
</table>

- Write the triples as it, separated by dots “.”
- IRI are enclosed within angle brackets “<…>”
- Literals are within quotes “”
- Type / Languages are specified with “^^” and “@”
- The types “xsd:integer” and “xsd:float” are automatically recognized by Turtle parsers (numbers without quotes)

<http://city.fukuoka.lg.jp>
<http://city.fukuoka.lg.jp>
<http://city.fukuoka.lg.jp>
<http://city.fukuoka.lg.jp>
<http://city.fukuoka.lg.jp>
  <http://www.w3.org/2000/01/rdf-schema#label> “福岡公式ホームページ”@ja .
A First Turtle Example

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Language / Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://city.fukuoka.lg.jp">http://city.fukuoka.lg.jp</a></td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#label">http://www.w3.org/2000/01/rdf-schema#label</a></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Wait, that’s it? It is so hard to read!

- Write the triples not separated by spaces.
- IRIs are enclosed within angle brackets ⟨⟩.
- Strings are enclosed within quotes “”.
- Type / Languages are specified with " arcane knowledge of Turtle..."
Use Qnames in Turtle

In fact, a slight superset of Qnames; but not CURIE!

```turtle
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> 
prefix schema: <http://schema.org/> 
prefix db: <http://dbpedia.org/> 
prefix xsd: <http://www.w3.org/2001/XMLSchema#> 

```
Make Turtle more Compact

Factorize a subject

```turtle
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix schema: <http://schema.org/>
prefix db: <http://dbpedia.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<http://city.fukuoka.lg.jp>
  rdfs:type schema:WebSite ;
  schema:about db:Fukuoka ;
  schema:lastReviewed "2015-2-1"^^xsd:date ;
  rdfs:label "Fukuoka city official homepage"@en ;
  rdfs:label "福岡市公式ホームページ"@ja .
```

Factorize subject with semi-colon ";"

The last triple ends with a dot "."

Factorize a predicate

```turtle
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix schema: <http://schema.org/>
prefix db: <http://dbpedia.org/>
prefix xsd: <http://www.w3.org/2001/XMLSchema#>

<http://city.fukuoka.lg.jp>
  rdfs:type schema:WebSite ;
  schema:about db:Fukuoka ;
  schema:lastReviewed "2015-2-1"^^xsd:date ;
  rdfs:label
    "Fukuoka city official homepage"@en ,
    "福岡市公式ホームページ"@ja .
```

Factorize predicates with colon ";"
Where do I store my RDF data?

- Small dataset: as a file
  - RDF files can be put on the web and build the semantic Web
  - Tool such as Apache Jena (ark) allows to query files with SPARQL
- Large dataset: in a database
  - RDF data model is different from the relational one
  - RDF database: Graph store or Triple store
  - A kind of NoSQL database

The Friend of Friend (FOAF) Project

- The FOAF vocabulary for RDF enables to express human relationships
- The FOAF project consists in building a social network based on distributed data
Where do I store my RDF data?

- Small dataset: as a file
  - RDF files can be put on the web and build the semantic Web
  - Tool such as Apache Jena (ark) allows to query with SPARQL
- Large dataset: in a database
  - RDF data model is different from the relational one
  - RDF database: Graph store or Triple store
  - A kind of NoSQL database

More on triple store later

Let’s talk SPARQL and Ontology first

Project
- The FOAF vocabulary for RDF enables to express human relationships
- The FOAF project consists in building a social network based on distributed data
The **SPARQL** Query Language for RDF Data
A bit of Background

**SPARQL Protocol And RDF Query Language**

- SPARQL is a W3C standard
  - SPARQL 1.0 (15/1/2008)
  - SPARQL 1.1 (21/3/2013)
- It is supported by most RDF stores and frameworks

Hum, this is a recursive acronym 😅

It appeared long after RDF itself (it was in 27/3/2000)
Comparison RDF vs. Relational

<table>
<thead>
<tr>
<th>Query language</th>
<th>Relational Database</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SQL</td>
<td>SPARQL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Database technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational database</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relational Model</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RDF 1.1 Abstract Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triple store (RDF store, graph store)</td>
</tr>
</tbody>
</table>
An Example of SPARQL Query

Definition of prefixes, same as Turtle

prefix db: <http://dbpedia.org/resource/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s
WHERE {
  ?s rdfs:label "Fukuoka"@en
}

Graph pattern
Same syntax as triples in Turtle
An Example of SPARQL Query

```sparql
prefix db: <http://dbpedia.org/resource/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s
WHERE {
  ?s rdfs:label "Fukuoka"@en
}
```
An Example of SPARQL Query

prefix db: <http://dbpedia.org/resource/>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s
WHERE {
  ?s rdfs:label "Fukuoka"@en
}

Names of the variables to return
Variable use a placeholder in a triple
Graph Pattern in SPARQL

- All SPARQL queries mainly use graph patterns to **match triples**
  - Constants (literals, IRIs) are constraints
  - Variables are placeholders
- A graph pattern consists of triples, using the Triple syntax
- It is evaluated in order
- **Examples:***

  All triples which English rdfs:label is "Fukuoka"
  ?s rdfs:label "Fukuoka"@en.
A more Complex Example of Graph Pattern

All triples which predicate is geo:lon, and which subject appears as subject in triples which rdfs:label is “福岡” in Japanese

?s rdfs:label "福岡"@ja ;
geo:lon ?longitude .

That is, the longitude of Fukuoka!

Two triples with the same subject
Graphs in SPARQL

- Graphs are collections of RDF triples
  - They define a logical partitioning of the global dataset
- Two types of graphs
  - Named graphs (identified by an IRI)
  - The default, anonymous graph
- It is possible to specify the graph in a SPARQL query
Specify the graph with the **GRAPH** Keyword

```
SELECT ?name ?lon ?lat
WHERE {
  GRAPH <graph IRI> {
    ?s rdfs:label ?name ;
    geo:lon ?lon ;
    geo:lat ?lat .
  }
}
```

- The default graph can not be specified with a GRAPH keyword
- To query the default graph, one has to do so outside any GRAPH block
**GRAPH IRI as part of the SPARQL Query**

**Example:** select the IRI of all the named graph in a RDF database

```
SELECT DISTINCT ?g
WHERE {
  GRAPH ?g {
    ?s ?p ?o
  }
}
```

- **DISTINCT keyword:** select only distinct values
- **SELECT DISTINCT ?g:** Stores graph IRI in variables
- **GRAPH ?g { ?s ?p ?o }** Does not match the default graph
- **This graph pattern matches any triples**


Construct the RDF Dataset with FROM and FROM NAMED

SELECT ?name
FROM <iri1>
FROM NAMED <iri2>
FROM NAMED <iri3>
WHERE {
  { ?s rdfs:label ?name }
  UNION
  { GRAPH ?g
    { ?s rdfs:label ?name } }
}

- The IRI specified in FROM /FROM NAMED designate serialized RDF file
- The IRI may designate local files or remote locations

Specifies the default graph
Specifies the named graphs
Allows remote query!
Construct the RDF Dataset with FROM and FROM NAMED

```
SELECT ?name
FROM <iri1>
FROM NAMED <iri2>
FROM NAMED <iri3>
WHERE {
  { ?s rdfs:label ?name }
  UNION
  { GRAPH ?g
    { ?s rdfs:label ?name } }
}
```
Federated Query with SERVICE

SELECT DISTINCT ?name
WHERE {
    SERVICE <http://dbpedia.org/sparql> {
        ?s rdfs:label ?name
    }
}

- SPARQL enables to distribute the process of a query between physically separated RDF datasets
  - This allows do mashup right inside a single SPARQL query
**Federated Query with**

**SERVICE**

**Collaboration between SPARQL endpoints**

**Select DISTINCT ?name**

**WHERE {**

* SERVICE <http://dbpedia.org/sparql>

?s rdfs:label ?name

**}**

- SPARQL enables to distribute the process of a query between physically separated RDF datasets.
- This allows doing mashup right inside a single SPARQL query.

**The URL of dbpedia SPARQL endpoint**

**Wait, SPARQL engines have URL?**
SPARQL Protocol

RDF Query Language

- SPARQL 1.1 defines two kinds of protocols
  - An HTTP REST API to submit SPARQL queries, with two urls
  - A protocol for SPARQL endpoints to discuss between each other
- Let’s take a look at the HTTP REST API

Not only a query language
The SPARQL HTTP REST API

- Given an API endpoint http://endpoint.org
- SPARQL queries
  - http://endpoint.org/sparql
    - Submit a read-only SPARQL query (SELECT/CONSTRUCT/ASK/DESCRIBE)
  - http://endpoint.org/update
    - Submit an update SPARQL query (LOAD/INSERT/DELETE/DROP/COPY/MOVE)
- Direct action on RDF data (at url http://endpoint.org?graph=graph_name)
  - **GET** request: returns a whole graph
  - **PUT** request: replaces a whole graph
  - **POST** request: adds triples to a graph
  - **DELETE** request: deletes a graph

More on CONSTRUCT/INSERT on next slide

Most triple stores organize the database in datasets, accessible via http://endpoint.org/dataset
CONSTRUCT and INSERT Queries

- The two following queries have similar syntax
  - **CONSTRUCT**: generates in output new triples derived from the current RDF dataset
  - **INSERT**: inserts to the RDF database new triples derived from the current RDF dataset
- They are often used for
  - ETL (Extract / Transform / Load)
  - Refactoring (e.g. change vocabulary)
  - Ontology inference (see next)
An Example of INSERT Query

```
INSERT
{
  ?a foaf:friend ?b ;
  foaf:knows ?b .
}
WHERE
{
  ?b foaf:friend ?a
}

- This query uses the vocabulary friend of a friend (foaf)
- This query stances that if ?a is friend with ?b then
  - the opposite is also true
  - they know each other

Use CONSTRUCT instead of INSERT for a construct query

A graph pattern to match triples and store data to variables

A graph pattern to construct new triples
```
Vocabulary, RDF Schema and Ontology
About Vocabulary

• Depending on the data you hold, you may need various vocabulary
  • You may create your own
  • But someone may have done the job for you!
• There are some W3C standard vocabularies
  • **RDF** and **RDF Schema** (RDFS)
  • **geo** for geographical data (e.g. longitude / latitude)
  • **SKOS**, the simple knowledge organization system
  • **XSD**, data types from the XML standards
• And some other well-established vocabularies
  • **Foaf** (Friend of a friend) to describe human relations
  • **Schema.org** to casually describe misc. resources such as public facilities, websites or drugs (aimed at being a general purpose vocabulary for RDFa and Microdata)
  • **Dublin Core** to describe bibliographical resources
  • **DBPedia, Yago**, two general-purpose vocabularies, used for online encyclopedia
About Vocabulary

• Depending on the data you hold, you may need various vocabulary
  • You may create your own
  • But someone may have done the job for you!

• There are some W3C standard vocabularies
  • RDF and RDF Schema (RDFS)
  • geo for geographical data (e.g. longitude/latitude)
  • SKOS, the simple knowledge organization system
  • XSD, data types from the XML standards

• And some other well-established vocabularies
  • Foaf (Friend of a friend) to describe human relations
  • Schema.org to describe misc. resources such as public facilities, websites, drugs (aimed at being a general purpose vocabulary for RDFa and Microdata)
  • Dublin Core to describe bibliographical resources
  • DBPedia, Yago, two general-purpose vocabularies, used for online encyclopedia
Find a vocabulary on prefix.cc

Example of RDFS

• Used to describe RDF schema (we’ll see later)
• W3C recommandation

prefix.cc shows us the ranking of most popular vocabularies
We get the turtle version of the vocabulary!
We get the turtle version of the vocabulary!

Wait, how come a RDF vocabulary is described in RDF?
Data Schema in RDF

- It is possible to describe the Schema of RDF data
  - We call it an **Ontology**
- The schema itself is stored in RDF, using some standard vocabulary (W3C recommendation)
  - **RDFS**: The simplest vocabulary
  - **OWL**: Very complex, and complete
  - **SPIN**: express rules using SPARQL
- These Ontology languages are real language
  - Toward model-driven development
- It is important to define the ontology in your RDF database so that **anyone can understand your data**

It is possible to express a large part of programs right in the ontology!
Basics of RDF Schema

- Similar to object-oriented languages
  - RDF resource have classes (typing system)
  - Types are organized in hierarchy of subclass / superclass
  - The kind of properties that an object of a given class can accept is well defined
- But with some differences
  - An RDF resource may have more than one class
  - Properties are first-class objects, that is, the properties of an RDF resource define its type
- Yet, this makes object mapping super-easy
  - For example the library “dotnetrdf” enables direct mapping between RDF objects and C# classes
How Ontologies are used by Triple Stores?

- SPARQL engine do not (usually) check the Ontology on the fly
- Instead, one use Ontology reasoner to generate extra RDF triples
- This is called **inference**
- Inference rules can also be expressed in SPARQL (CONSTRUCT query)

SPIN is a vocabulary that enable to use ontology rules written in SPARQL

The inferred triples are part of the RDF database!
Some Triple Stores
Sesame (rdf4j.org)

- Open-source, written in Java
- Supports plugins
- Several functionalities
  - Java RDF framework to programmatically work with RDF data
  - Triple Store Server (Java weblet for servers such as Tomcat or Jetty)
  - Inference in RDFS (not OWL)
- Originally developed as a research project
  - European Union project On-To-Knowledge (2000-2002)
  - Developed by the company Aduna (Dutch) for the
  - Distributed as Java weblet (war)
Apache Jena (jena.apache.org)

- Open source, written in Java
- Several functionalities
  - Java framework to manipulate RDF data
  - Triple store server
  - Inference in RDFS ans OWL
- Research project
  - From Hewlett-Packard’s Semantic Web Research Lab
  - The most popular project among researcher, therefore supports several cutting-edge plugins
AllegroGraph (franz.com)

- Closed-source, written in LISP
- Bindings in most language
- Commercial database from Franz.inc
- High performance
- Powerful inference (RDFS, RDFS++)
Virtuoso
(virtuoso.openlinksw.com)

- Open source, written in C
- Originated from the Finish database ecosystem in 1998
- Not only for RDF, also supports relational data
  - Supports RDF and SPARQL through mapping to relational model and SQL
- Multi-purpose server, notably:
  - Database (based on object-relational model)
  - Web application server
  - Web content management system
- Usually seen as the fastest and most scalable triple store (used by dbpedia)
- However it lacks powerful inference functionality like Apache Jena and Sesame
Conclusion
The RDF Developer Toolbox

- **A triple store**
  - Example: “Gruff” from the company Franz

- A RDF graph visualizer
  - In most triple stores

- A RDF Library
  - Enable to programmatically manipulate RDF data.
    - Example: “dotnetrdf” for C#,
      “C RDF lirabry” for C,
      “RDFLib” for Python

- An ontology reasoner
  - Example: “Protégé” from Stanford University

- An ontology debugger
Example of System
from teapot at BODIC.org

nginx × node.js × Apache Jena × RDF Lib

マッシュアップアプリケーション
(コンテストのアプリ実証実験アプリ等)

HTTP API

公共機関等

Open Data of FUKUOKA CITY
福岡市 CKAN Portal
(http://ckan.open-governmentdata.org)

• 自動バージョン確認
• 手動加工が必要なファイルもある

生データ
(CSV,XLS)

クラシフィング/
LOD化

RDF File
(Turtle)

JSON File

Triple Store

Geographic Data

Event

Data

User Data

DB

MongoDB

GeoJSON API

SPARQL API (読み込み専用)

Apache Fuseki

Apache TDB

Mongo Node

アクセスログ

ロードバランサー
(nginxを利用)

APIエンドポイント
カスタムサーバー
(node.jsを利用)

イベント・コメント公開サーバー

イベント・コメント管理サーバー

ユーザー管理・認証サーバー

斯ケーラビリティ対策
• 元長サーバーを立ち上げる
• ロードバランサーを扱う
(APIエンドポイントを利用)

スケーラビリティ対策
MongoDBのスケーラビリティ機能を利用（冗長、シャーディング）

ユーザー認証
• APIキーの確認

ロードバランサー
(nginxを利用)
General conclusion... 

Give it a try!
Thank you very much