SPARQL - SPARQL Protocol and RDF Query Language

These slides are partly inspired from the tutorial ARQ: http://jena.apache.org/tutorials/sparql.html

Philippe Genoud, UFR IM2AG, UGA
Manuel Atencia Arcas, UFR SHS, UGA
Querying RDF data: SPARQL

Linked Data: 3rd Principle
When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL).
SPARQL: Introduction

• **RDF: Resource Description Framework**
  – it provides a flexible and extensible way to represent information about resources of the web

• **SPARQL: SPARQL Protocol And RDF Query Language**
  – a query language to access RDF graphs (SPARQL Query Language Specification) inspired from SQL
  – a protocol to submit requests through HTTP GET, HTTP POST or SOAP (SPARQL Protocol for RDF Specification)
  – an XML format for the results (SPARQL Query XML Results Format), and now JSON
  – a W3C standard
    - SPARQL 1.0 recommendation – January 2008
    - SPARQL 1.1 recommendation – March 2013

http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/
SPARQL Protocol And RDF Query Language overview

a protocol to submit SPARQL requests over HTTP (SPARQL protocol for RDF Specification)

an XML output format (JSON, ...) for SPARQL query results (SPARQL Query XML Results Format)

interface utilisateur

Virtuoso SPARQL Query Editor

a request language to query RDF graphs (SPARQL RDF Query Language)
SPARQL Query Language

- find all restaurants that are less than 1km away from Fort Saint-Pierre (Théâtre de la Mer – Sète)

Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix ogc: <http://www.opengis.net/ont/geosparql#>
Prefix geom: <http://geovocab.org/geometry#>
Prefix lgdo: <http://linkededgeodata.org/ontology/>

Select ?s, ?l From <http://linkededgeodata.org>
Where
{
  ?s a lgdo:Restaurant ;
  rdfs:label ?l ;
  geom:geometry [ ogc:asWKT ?g ] .

  Filter(bif:st_intersects (?g, bif:st_point (3.692764, 43.393794), 1)) .
}
SPARQL Query Language

- SPARQL based on:
  - Turtle serialization of RDF
  - Graph Pattern Matching

Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix ogc: <http://www.opengis.net/ont/geosparql#>
Prefix geom: <http://geovocab.org/geometry#>
Prefix lgdo: <http://linkedgeodata.org/ontology/>

Select ?s, ?l From <http://linkedgeodata.org/>
Where
{ ?s a lgdo:Restaurant ;
  rdfs:label ?l ;
  geom:geometry [ ogc:asWKT ?g ] .
}

Filter(bif:st_intersects (?g, bif:st_point (3.692764, 43.393794), 1)) .

Graph Pattern: RDF triple containing one or more variables at any position (sujet, prédicat, objet)

Filter over the variable ?g which allows to restrict the results
SPARQL Protocol

- SPARQL query (SPARQL URI) has 4 components

SPARQL request over HTTP: SPARQL URI

the SPARQL endpoint URL
http://linkedgeodata.org/sparql

SPARQL Query Language

&query=PREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2000%2F01%2Frdf-schema%23%3E%0D%0APREFIX+ogc%3A+%3Chttp%3A%2F%2Fwww.opengis.net%2Font%2Fgeosparql%23%3E%0D%0APREFIX+geom%3A+%3Chttp%3A%2F%2Fgeovocab.org%2Fgeometry%23%3E%0D%0APREFIX+lgdo%3A+%3Chttp%3A%2F%2Flinkedgeodata.org%2Fontology%3E%0D%0A%0D%0ASELECT+%0D%0A+FROM+%3Chttp%3A%2F%2Flinkedgeodata.org%3E+%7B%0D%0A+?s+rdfs%3A+label+%3Fl++%0D%0A+geom%3A+geometry+?g++%0D%0A%7D%0D%0A+FILTER(bif%3A+st_intersects+%28?g%2C+bif%3A+st_point+%283.692764%2C+43.393794%29%29%29%29%0D%0A%0D%0AThe+RDF+graphs+to+request+(optional)%2C+
default-graph=http://linkedgeodata.org
+named-graph-uri=...

Output format
format=application/sparql-results+xml
(text/html, json...)

the query string
http://linkedgeodata.org/sparql
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix ogc: <http://www.opengis.net/ont/geosparql#>
prefix geom: <http://geovocab.org/geometry#>
prefix lgdo: <http://linkedgeodata.org/ontology/>

<table>
<thead>
<tr>
<th>Prefix</th>
<th>URI</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
<td>RDF Schema</td>
</tr>
<tr>
<td>ogc</td>
<td><a href="http://www.opengis.net/ont/geosparql#">http://www.opengis.net/ont/geosparql#</a></td>
<td>OGC GeoSPARQL</td>
</tr>
<tr>
<td>geom</td>
<td><a href="http://geovocab.org/geometry#">http://geovocab.org/geometry#</a></td>
<td>GeoVocabulary</td>
</tr>
<tr>
<td>lgdo</td>
<td><a href="http://linkedgeodata.org/ontology/">http://linkedgeodata.org/ontology/</a></td>
<td>Ledge Data</td>
</tr>
</tbody>
</table>
**SPARQL : Output Format (XML)**

SPARQL request over HTTP : SPARQL URI

HTTP

SPARQL Protocol Layer

Serveur

Client

résultat de sortie Formaté (XML, JSON...)

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemalocation="http://www.w3.org/2001/sw/DataAccess/rf1/result2.xsd">
  <head>
    <variable name="s"/>
    <variable name="l"/>
  </head>
  <results distinct="false" ordered="true">
    <result>
      <binding name="s">
        <uri>http://linkedgeodata.org/triplify/node1304006380</uri>
      </binding>
      <binding name="l">
        <literal>Ls#39:Hostal</literal>
      </binding>
    </result>
    <result>
      <binding name="s">
        <uri>http://linkedgeodata.org/triplify/node1952213273</uri>
      </binding>
      <binding name="l">
        <literal>Ls#39:Ultima</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Variables from the SELECT clause

one result element for each row in the result set

(binding of one of the variables of this result element)
SPARQL Query Language (1.0)

Source: Pérez, Arenas and Gutierrez, Chapter 1: On the Semantics of SPARQL, Semantic Web Information Management: A Model Based Perspective, Springer 2010
A simple query with SPARQL

- the RDF data graph
A simple query with SPARQL

- the RDF data graph

http://www.w3.org/2001/vcard-rdf/3.0

an RDFS vocabulary corresponding to the vCard IETF specification (RFC6350) (http://tools.ietf.org/html/rfc6350)

vCard Format Specification

Abstract

This document defines the vCard data format for representing and exchanging a variety of information about individuals and other entities (e.g., formatted and structured name and delivery addresses, email address, multiple telephone numbers, photograph, logo, audio clips, etc.). This document obsoletes RFCs 2425, 2426, and 4770, and updates RFC 2739.

John Smith
A simple query with SPARQL

- RDF data graph in Turtle notation

```turtle
@prefix vCard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://somewhere/MattJones/> vCard:FN "Matt Jones";
  vCard:N [ vCard:Family "Jones";

<http://somewhere/RebeccaSmith/> vCard:FN "Becky Smith";
  vCard:N [ vCard:Family "Smith";
    vCard:Given "Rebecca" ] .

<http://somewhere/JohnSmith/> vCard:FN "John Smith";
  vCard:N [ vCard:Family "Smith";
    vCard:Given "John" ] .

<http://somewhere/SarahJones/> vCard:FN "Sarah Jones";
  vCard:N [ vCard:Family "Jones";
```
A simple query with SPARQL

- another way to represent blank nodes:

```
@prefix vCard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

<http://somewhere/MattJones/> vCard:N     _:b0 .
_:b0  vCard:Family "Jones" .
_:b0  vCard:Given  "Matthew" .

_:b1 vCard:Family "Smith" .
_:b1 vCard:Given  "Rebecca" .

_:b2 vCard:Family "Smith" .
_:b2 vCard:Given  "John" .

_:b3 vCard:Family "Jones" .
_:b3 vCard:Given  "Sarah" .
```
A simple query with SPARQL

```
SELECT ?x
WHERE {
  ?x <http://www.w3.org/2001/vcard-rdf/3.0#FN> "John Smith" .
}
```

- **variable**: RDF variable
- **triple pattern**: RDF triple containing one or more variables in any position (subject, property, object)
- **match the triple pattern in the WHERE clause against the triples in the RDF graph**: Possible bindings for variable x

<table>
<thead>
<tr>
<th>x</th>
<th><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></th>
</tr>
</thead>
</table>

15% match the triple pattern in the WHERE clause against the triples in the RDF graph.
Another simple query with SPARQL

It is possible to have more than one variable in the triple pattern

```
SELECT ?x ?fname
WHERE {
  ?x <http://www.w3.org/2001/vcard-rdf/3.0#FN> ?fname
}
```
A third simple SPARQL request

Basic pattern: set of triple patterns

```
SELECT ?givenName
WHERE {
}
```

matches when the triple patterns all match with the same value used each time the variable with the same name is used

<table>
<thead>
<tr>
<th>givenName</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;John&quot;</td>
</tr>
<tr>
<td>&quot;Rebecca&quot;</td>
</tr>
</tbody>
</table>
A third simple SPARQL request

```sparql
SELECT ?givenName
WHERE {
}
```

results in XML format

```xml
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <head>
        <variable name="givenName"/>
    </head>
    <results>
        <result>
            <binding name="givenName">
                <literal>Rebecca</literal>
            </binding>
        </result>
        <result>
            <binding name="givenName">
                <literal>John</literal>
            </binding>
        </result>
    </results>
</sparql>
```
Q(ualified)Names

There is shorthand mechanism for writing long URIs using prefixes

```sparql
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?givenName
WHERE {
  ?y vcard:Given ?givenName .
}
```

other simplifications (similar to Turtle)

- Triples with same subject:
  ```sparql
  ?x c:firstName ?y .
  ?x c:lastName ?z .
  ```

- Triples with same subject and same predicate:
  ```sparql
  ?x c:name "Olivier" .
  ?x c:name "Laurent" .
  ```

- Prefix IRI mapping:

<table>
<thead>
<tr>
<th>Prefix</th>
<th>IRI</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:</td>
<td><a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a></td>
</tr>
<tr>
<td>rdfs:</td>
<td><a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a></td>
</tr>
<tr>
<td>xsd:</td>
<td><a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a></td>
</tr>
</tbody>
</table>
Exercise: Artemis' bookstore

- Artemis owns a bookstore and she has been convinced of the benefits of using semantic web technologies.
- Thanks to a semantic web developer, Artemis' data has been transformed into RDF (see artemisBookstoreData.ttl).
- She has a great idea: for next Valentine's day, she will offer a special discount for pairs of books written by married couples who are living writers, like Paul Auster and Siri Hustvedt, two of her favourite writers.
- Although "being married with" or "being alive" is not information that Artemis can find in her local data, she could find it in the Linked Open Data cloud using SPARQL. Let's see how.
Exercise: Artemis' bookstore

• We will start by running some SPARQL queries at DBpedia SPARQL endpoint (http://dbpedia.org/sparql)
  – Which are the properties of Paul Auster in DBpedia?
  – When and where was Paul Auster born?
  – Which are the classes Paul Auster is an instance of?
Filters

• Filters: allow to restrict the values in a solution
  – Boolean expression the request solution must satisfy
  – rich expression language based on Xpath, Xquery and special operators defined by SPARQL (see section 11 of SPARQL specification document [http://www.w3.org/TR/rdf-sparql-query/#tests](http://www.w3.org/TR/rdf-sparql-query/#tests))
    • Relational operators: <, >, =, <>, >=, !=
    • Boolean operators: &&, ||, !
    • Arithmetic operations: +, *, -, /
    • Variable binding tests: isURI(?x), isBlank(?x), isLiteral(?x), bound(?x)
    • Regular expressions: regex(?x, "^ali", "i")
    • Access to attributes/values: lang(), datatype(), str()
    • Casting (re-typing functions): xsd:integer(?x)
    • External functions/extensions
Filters: Testing Values

PREFIX info: <http://somewhere/peopleInfo#>

SELECT ?resource
WHERE {
  FILTER (?age >= 24)
}

The arithmetic expression must be in parentheses (round brackets)

let's add an age property to some persons

<http://somewhere/RebeccaSmith/> info:age "23"^^xsd:integer ;
  vCard:FN "Becky Smith" ;
    vCard:Given "Rebecca"
  ].
Filters: String matching

Regular expressions to test strings (like LIKE in SQL request)

Syntax:

```
FILTER regex(?x, "pattern" [, "flags"])
```

Note: The `FILTER` function is optional.

**XQuery regular expression language (similar to Perl)**

[http://www.w3.org/TR/xpath-functions/#regex-syntax](http://www.w3.org/TR/xpath-functions/#regex-syntax)

---

Prefix:

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
```

Query:

```
SELECT ?givenName
WHERE
{ ?y vcard:Given ?givenName .
  FILTER regex(?givenName, "r", "i")
}
```

given names with an "r" or "R" in them

Case insensitive

<table>
<thead>
<tr>
<th>givenName</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Rebecca&quot;</td>
</tr>
<tr>
<td>&quot;Sarah&quot;</td>
</tr>
</tbody>
</table>
Exercise: Artemis' bookstore

• Let's continue querying DBpedia data:
  – Which are the relations (if there is any) between Paul Auster and Siri Hustvedt?
  – Find married couples who are writers (remove duplicates).
OPTIONALs

- SPARQL has the ability to query for data but not to fail when that data does not exist.
- The **OPTIONAL** part of a query can extend the information found in the query solution but the non-optional information is returned anyway.

```sparql
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name ?age
WHERE
{ ?person vcard:FN ?name .
}
```

- The `info:age` property must be present in a solution.

```sparql
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name ?age
WHERE
{ ?person vcard:FN ?name .
  OPTIONAL { ?person info:age ?age } 
}
```

- The triple pattern for the age is **optional**, there is a pattern solution for the people who don't have age information.

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td>23</td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>
OPTIONALs

```sql
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
  ?person info:age ?age
}
```

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td>23</td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
</tbody>
</table>

the info:age property must be present in a solution
OPTIONALs

the triple pattern for the age is optional, there is a pattern solution for the people who don't have age information

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td>23</td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>

PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
OPTIONAL { ?person info:age ?age }
}
OPTIONAL is a binary operator that combines two graph patterns. If the group matches, the solution is extended, if not, the original solution is given.

PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
  OPTIONAL { ?person info:age ?age }
}

<table>
<thead>
<tr>
<th>person</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/RebeccaSmith/">http://somewhere/RebeccaSmith/</a></td>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/SarahJones/">http://somewhere/SarahJones/</a></td>
<td>&quot;Sarah Jones&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/MattJones/">http://somewhere/MattJones/</a></td>
<td>&quot;Matt Jones&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>person</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>25</td>
</tr>
<tr>
<td><a href="http://somewhere/RebeccaSmith/">http://somewhere/RebeccaSmith/</a></td>
<td>23</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td>23</td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>
OPTIONALs with FILTERs

- The optional pattern can be any group pattern and may involve any SPARQL pattern types, e.g. you can use OPTIONAL clauses with filters.

PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE {
  OPTIONAL { ?person info:age ?age . FILTER ( ?age > 24 ) }
}
OPTIONALs with FILTERs

- The optional pattern can be any group pattern and may involve any SPARQL pattern types, e.g. you can use OPTIONAL clauses with filters.

PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
  OPTIONAL { ?person info:age ?age . FILTER ( ?age > 24 ) }
}

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>

```sparql
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
  OPTIONAL { ?person info:age ?age . FILTER ( ?age > 24 ) }
}
```
OPTIONALs with FILTERs

- The optional pattern can be any group pattern and may involve any SPARQL pattern types, e.g. you can use **OPTIONAL** clauses with filters.

```
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
  OPTIONAL { ?person info:age ?age . FILTER ( ?age > 24 ) }
}
```

---

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td></td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>

---

OPTIONAL is a binary operator that combines two graph patterns. If the group matches, the solution is extended, if not, the original solution is given.

```
```

---

<table>
<thead>
<tr>
<th>person</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/RebeccaSmith/">http://somewhere/RebeccaSmith/</a></td>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/SarahJones/">http://somewhere/SarahJones/</a></td>
<td>&quot;Sarah Jones&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/MattJones/">http://somewhere/MattJones/</a></td>
<td>&quot;Matt Jones&quot;</td>
</tr>
</tbody>
</table>

---

```
```

---

<table>
<thead>
<tr>
<th>person</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>25</td>
</tr>
</tbody>
</table>

---

The age of "Becky Smith" is not included in the solution because it is less than 24.

---

```
OPTIONALs with FILTERs

PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE
{
OPTIONAL { ?person info:age ?age . FILTER (?age > 24) }
}


<table>
<thead>
<tr>
<th>person</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/RebeccaSmith/">http://somewhere/RebeccaSmith/</a></td>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/SarahJones/">http://somewhere/SarahJones/</a></td>
<td>&quot;Sarah Jones&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td><a href="http://somewhere/MattJones/">http://somewhere/MattJones/</a></td>
<td>&quot;Matt Jones&quot;</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>person</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://somewhere/RebeccaSmith/">http://somewhere/RebeccaSmith/</a></td>
<td>23</td>
</tr>
<tr>
<td><a href="http://somewhere/JohnSmith/">http://somewhere/JohnSmith/</a></td>
<td>25</td>
</tr>
</tbody>
</table>

FILTER is performed after the OPTIONAL

( ?age > 24 )

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Becky Smith&quot;</td>
<td>23</td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;John Smith&quot;</td>
<td>25</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
<td>25</td>
</tr>
</tbody>
</table>

variable age is unbound
→evaluation exception
FAILURES!
OPTIONALS with FILTERs

- **bound** operator (like PHP `isset`) allows the use of unbound variables in FILTERs

```
PREFIX info: <http://somewhere/peopleInfo#>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name ?age
WHERE {
              FILTER (!bound(?age) || ?age > 24) }
}
```

Filtering names of all people whose age is undefined or older than 24
OPTIONALs and Order Dependent Queries

if you use the same variable in two or more optional clauses order is relevant

Not the same result!
OPTIONALs and Order Dependent Queries

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name
WHERE
{
  ?x a foaf:Person .
  OPTIONAL { ?x foaf:name ?name } 
  OPTIONAL { ?x vCard:FN ?name }
}

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td>&quot;Sarah Jones&quot;</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
</tr>
</tbody>
</table>

?x a foaf:Person .
OPTIONAL { ?x foaf:name ?name }

| x | name          |
|---|--|---------------|
| A11556 | |---------------|
| A11555 | |---------------|
| A11554 | "Sarah Jones"|---------------|
| A11553 | "Matt Jones" |---------------|

?x vCard:FN ?name

<table>
<thead>
<tr>
<th>x</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11556</td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td>A11555</td>
<td>&quot;Becky Smith&quot;</td>
</tr>
</tbody>
</table>
OPTIONALs and Order Dependent Queries

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name
WHERE
{
  ?x a foaf:Person .
  OPTIONAL { ?x foaf:name ?name }
  OPTIONAL { ?x vCard:FN ?name }
}

<table>
<thead>
<tr>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td>&quot;S. Jones&quot;</td>
</tr>
<tr>
<td>&quot;Matt Jones&quot;</td>
</tr>
</tbody>
</table>
Union provides a means of combining graph patterns so that one of several alternative graph patterns may match.

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?name
WHERE {
    { [] foaf:name ?name } UNION { [] vCard:FN ?name }
}
If different variables are used in each branch, the application can discover which sub-pattern caused the match.
### Union ≠ OPTIONALs

**PREFIX** foaf: <http://xmlns.com/foaf/0.1/>

**PREFIX** vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

```
SELECT ?name1 ?name2
WHERE {
  ?x a foaf:Person
  OPTIONAL { ?x foaf:name ?name1 }
  OPTIONAL { ?x vCard:FN ?name2 }
}
```

<table>
<thead>
<tr>
<th>x</th>
<th>name1</th>
<th>name2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A11556</td>
<td></td>
<td>&quot;John Smith&quot;</td>
</tr>
<tr>
<td>A11555</td>
<td></td>
<td>&quot;Becky Smith&quot;</td>
</tr>
<tr>
<td>A11554</td>
<td>&quot;Sarah Jones&quot;</td>
<td>&quot;S. Jones&quot;</td>
</tr>
<tr>
<td>A11553</td>
<td>&quot;Matt Jones&quot;</td>
<td></td>
</tr>
</tbody>
</table>
Solution Modifiers

• The set of solutions produced by graph pattern matching can be modified in various ways:
  – Projection - keep only selected variables
  – OFFSET/LIMIT - chop the number solutions (best used with ORDER BY)
    • OFFSET the start index,
    • LIMIT the number of solutions to be returned.
      – Using LIMIT alone useful to ensure not too many solutions are returned, to restrict the effect of some unexpected situation
  – ORDER BY - sorted results
  – DISTINCT - yield only one row for one combination of variables and values.

• The solution modifiers OFFSET/LIMIT and ORDER BY always apply to all result forms

example

```sql
prefix foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT * where {
  ?x foaf:name ?name;
  foaf:age ?age.
}
ORDER BY ?name DESC(?age)
```
The RDF data-store service can handle one or more RDF graphs, the SPARQL query is executed against a data set (RDF Dataset) that represents a collection of one or more graphs.

Prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
Prefix ogc: <http://www.opengis.net/ont/geosparql#>
Prefix geom: <http://geovocab.org/geometry#>
Prefix lgdo: <http://linkedgeodata.org/ontology/>

Select ?s, ?l From <http://linkedgeodata.org>
Where
{
  ?s a lgdo:Restaurant ;
  rdfs:label ?l ;
  geom:geometry [ ogc:asWKT ?g ] .

  Filter(bif:st_intersects (?g, bif:st_point (3.692764, 43.393794), 1)) .
}
SPARQL Query Language

Select clause - DataSets

- A SPARQL request can associate different graph patterns to different named graphs.
  - `FROM` defines the default graph (this may be the result of merging several graphs)
  - `FROM NAMED` defines graphs which then can be explicitly named in the `WHERE` part of the query through the `GRAPH` keyword

PREFIX p1: ... 
SELECT ... 
FROM p1:g1 
FROM p2:g2 
FROM NAMED hpi:g3 
FROM NAMED hpi:g4 
WHERE {
    .......
    GRAPH hpi:g3 {
        .......
    }
    GRAPH ?g {
        .......
    }
}
Datasets

- an **RDF Dataset** is the unit that is queried by a SPARQL query. It consists of a default graph, and a number of named graphs.
- graph matching operations (basic patterns, OPTIONALs, and UNIONS) work on one RDF graph.
  - by default …. the default graph 😊
  - it can be changed by the GRAPH keyword.

```sparql
prefix foaf: <http://xmlns.com/foaf/0.1/>
SELECT * where {
  ?x rdf:type foaf:Person.
  GRAPH <http://www.ujf-grenoble.fr/data> {
    ?x foaf:name ?name
  }
  GRAPH ?g { ?x foaf:knows ?y }
}
```

- `uri`: the pattern will be matched against the graph in the dataset with that name
- `?var`: all the named graphs (not the default graph) are tried
  The variable may be used elsewhere so that if, during execution, it's value is already known for a solution, only the specific named graph is tried.

names must come from the graph <http://www.ujf-grenoble.fr/> to know the source of the result for each contact found
Datasets

- a (very simple) dataset that might occur for an RDF aggregator of book details
  - two small graphs describing some books
  - a default graph which records when these graphs were last read

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.

```

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.

[] dc:title "Harry Potter and the Philosopher's Stone".
[] dc:title "Harry Potter and the Chamber of Secrets".
```

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.

[] dc:title "Harry Potter and the Sorcerer's Stone".
[] dc:title "Harry Potter and the Chamber of Secrets".
```
Datasets

**Default graph**

- <ds-ng-1.ttl> `dc:date`: 2005-07-14 ...
- <ds-ng-2.ttl> `dc:date`: 2005-09-22 ...

**Named graph <ds-ng-1.ttl>**

- `dc:title` Harry Potter and the Philosopher's Stone

**Named graph <ds-ng-2.ttl>**

- `dc:title` Harry Potter and the Chamber of Secrets

PREFIX `xsd: <http://www.w3.org/2001/XMLSchema#>`
PREFIX `dc: <http://purl.org/dc/elements/1.1/>`

```
SELECT * WHERE { ?s ?p ?o }
```

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ds-ng-2.ttl</td>
<td><code>dc:date</code></td>
<td>&quot;2005-09-22T05:53:05+01:00&quot;^^xsd:dateTime</td>
</tr>
<tr>
<td>:ds-ng-1.ttl</td>
<td><code>dc:date</code></td>
<td>&quot;2005-07-14T03:18:56+01:00&quot;^^xsd:dateTime</td>
</tr>
</tbody>
</table>
Datasets

The default graph and the named graphs are queried.

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT * WHERE{
  { ?s ?p ?o } UNION { GRAPH ?g { ?s ?p ?o } }
}
```
Datasets

Dataset

Default graph

<ds-ng-1.ttl> dc:date 2005-07-14 ...

<ds-ng-2.ttl> dc:date 2005-09-22 ...

Named graph <ds-ng-1.ttl>

dc:title Harry Potter and the Philosopher's Stone

dc:title Harry Potter and the Chamber of Secrets

Named graph <ds-ng-2.ttl>

dc:title Harry Potter and the Sorcerer's Stone

dc:title Harry Potter and the Chamber of Secrets

PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?title
{
  GRAPH :ds-ng-2.ttl
  { ?b dc:title ?title }
}

only the named graph ds-ng-2.ttl is queried

<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Harry Potter and the Sorcerer's Stone&quot;</td>
</tr>
<tr>
<td>&quot;Harry Potter and the Chamber of Secrets&quot;</td>
</tr>
</tbody>
</table>
The name of the graph to be queried is determined with the query itself.

<table>
<thead>
<tr>
<th>date</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;2005-09-22T05:53:05+01:00&quot;^^xsd:dateTime</td>
<td>&quot;Harry Potter and the Sorcerer's Stone&quot;</td>
</tr>
<tr>
<td>&quot;2005-09-22T05:53:05+01:00&quot;^^xsd:dateTime</td>
<td>&quot;Harry Potter and the Chamber of Secrets&quot;</td>
</tr>
</tbody>
</table>
FROM and FROM NAMED

• FROM and FROM NAMED allow to describe the RDF dataset used for a query
  
  – FROM url → the graphs defining the default data set
    • There can be more than one FROM clause and the default graph is the RDF merge of the individual graphs
  
  – FROM NAMED url → the named graphs used to solve the request
    • The graph is given the name url and the data is read from that location.
    • Multiple FROM NAMED clauses cause multiple graphs to be added to the dataset.


RDF 1.1 provides a standard way to name graphs
SPARQL Query Language (1.0)

returns an RDF graph by substituting the variables in a set of triple templates

returns an RDF graph describing found resources

tests if there exists at least one result

Source: Pérez, Arenas and Gutierrez, Chapter 1: On the Semantics of SPARQL, Semantic Web Information Management: A Model Based Perspective, Springer 2010
ASK

- ASK – ask a boolean query.
  - to verify that there is at least one response.
  - Is there a student over 30 years?

```
PREFIX ufrimag: <http://www.ufrimag.fr#>
ASK {
  FILTER (?age > 30)
}
```

the boolean result

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head> ... </head>
  <boolean> true </boolean>
</sparql>
```
Exercise: Artemis' bookstore

- Let's go back to DBpedia and Paul Auster:
  - Is Paul Auster dead?
• **SELECT** returns a flat list of variables bindings
  – the application program is in charge of processing these bindings
    (often by converting solution tuples into triples and adding them to an RDF graph).

• **CONSTRUCT** allows you to directly product a RDF graph containing the variables values
  – the WHERE and FILTER clause works the same way as the SELECT form
  – bindings of the variables are inserted into a new graph constructed from template triples specified in the CONSTRUCT clause (which replace the SELECT clause).
CONSTRUCT avec un graph-gabarit à un seul sujet et deux triplets

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x foaf:firstName ?y .
          ?x foaf:lastName ?z .
    }

FROM <vca-db-1rd.rdf>
WHERE
{ ?x vcard:N ?u .
  ?u vcard:Given ?y .
  ?u vcard:Family ?z.}

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://somewhere/RebeccaSmith/>
  foaf:firstName "Rebecca" ;
  foaf:lastName  "Smith" .

<http://somewhere/MattJones/>
  foaf:firstName  "Matthew" ;
  foaf:lastName  "Jones" .

<http://somewhere/SarahJones/>
  foaf:firstName  "Sarah" ;
  foaf:lastName  "Jones" .

<http://somewhere/JohnSmith/>
  foaf:firstName  "John" ;
  foaf:lastName  "Smith" .

Construct

The graph we have

The graph we want
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?auteur foaf:pastProject _:oeuvre .
   ?auteur foaf:lastName ?n .
}

WHERE
?vc vcard:Family ?n .
}

@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://inalco/M2-Trad/poetes#Théodore_de_BANVILLE>
   foaf:firstName "Théodore" ;
   foaf:lastName "de BANVILLE" ;
   foaf:pastProject
        dc:title "Odes funambulesques"
      ].
**DESCRIBE**

- **DESCRIBE** – Returns an RDF graph, based on what the query processor is configured to return.
  - SPARQL specification says: "the useful information the service has about a resource"
  - in theory this should help you understand the context of the resources returned... but there is no warranty.

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.com/>
DESCRIBE ex:karen ?friend {
}
```

Asks for a description of karen and her friends

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:karen</td>
<td>foaf:knows</td>
<td>ex:alex</td>
</tr>
<tr>
<td>ex:karen</td>
<td>foaf:name</td>
<td>&quot;Karen&quot;</td>
</tr>
<tr>
<td>ex:alex</td>
<td>foaf:name</td>
<td>&quot;Alex&quot;</td>
</tr>
</tbody>
</table>

```sparql
DESCRIBE <http://example.com/fish> ?x WHERE {
    ?x ?y <http://example.com/fish>
}
```

Asks for a description of fish, and any resource directly related to fish.
SPARQL 1.1: new functionalities

- W3C Recommendation, March 2013, 21
  - [http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/](http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/)

- Query
  - New aggregation functions for results (count, min, max, group by, etc.)
  - Variable assignment
    - SELECT (COUNT(DISTINCT ?s)) AS ?num number of distinct restaurants
  - Negation
    - NOT EXISTS, EXISTS: filtering results depending on whether a graph pattern does, or does not match in the context of the query solution being filtered

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person WHERE {
  FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

- MINUS: removing solutions related to another pattern

*Persons who don’t have a name*  
See: [http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#negation](http://www.w3.org/TR/2013/REC-sparql11-query-20130321/#negation)
Exercise: Artemis' bookstore

• Let's go back to DBpedia, Paul Auster and Siri Hustvedt:
  – Find married couples who are living writers.
SPARQL 1.1: new functionalities

- Query...
  - subqueries: possibility to embed SPARQL queries within other queries
    - e.g. for limiting the number of results from some sub-expression within the query
  - subqueries are evaluated logically first, and the results are projected up to the outer query

*Example: Return a name (the one with the lowest sort order) for all the people that know Alice and they have a name*

```
PREFIX : <http://people.example/> .
:alice :name "Alice", "Alice Foo", "A. Foo" .
:carol :name "Carol", "Carol Baz", "C. Baz"

PREFIX : <http://people.example/>
SELECT ?y ?minName
WHERE {
  {
    SELECT ?y (MIN(?name) AS ?minName)
    WHERE {
      ?y :name ?name .
    }
  }
  GROUP BY ?y
}
```

<table>
<thead>
<tr>
<th>y</th>
<th>minName</th>
</tr>
</thead>
<tbody>
<tr>
<td>bob</td>
<td>&quot;B. Bar&quot;</td>
</tr>
<tr>
<td>carol</td>
<td>&quot;C. Baz&quot;</td>
</tr>
<tr>
<td>alice</td>
<td>&quot;A. Foo&quot;</td>
</tr>
<tr>
<td>bob</td>
<td>&quot;B. Bar&quot;</td>
</tr>
<tr>
<td>carol</td>
<td>&quot;C. Baz&quot;</td>
</tr>
</tbody>
</table>

1: inner query evaluation  
2: outer query evaluation  
3: results of 1 are joined with results of 2
SPARQL 1.1: new functionalities

- **Query**
  - Basic federated queries (**SERVICE, BINDING**)
    - To execute requests distributed over different SPARQL endpoints

  *Example: Is there anyone among Alice’s friends with the same name as the resource identified by the URI <http://dbpedia.org/resource/Snoopy> at DBpedia?*

  ```sparql
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?name
  WHERE {
    SERVICE <http://dbpedia.org/sparql> {
      <http://dbpedia.org/resource/Snoopy> foaf:name ?name
    }
  }
  ```

  See: [http://www.w3.org/TR/sparql11-overview/](http://www.w3.org/TR/sparql11-overview/)
Exercise: Artemis' bookstore

• Final query:
  – Find, in Artemis' bookstore data, pairs of books written by married couples who are living writers.

• One way to do this is:
  – upload Artemis' data in a fuseki server:
    https://jena.apache.org/documentation/serving_data/
  – submit a federated query to this SPARQL endpoint.
SPARQL 1.1: new functionalities

• New serialization formats for query results (JSON...)
• CRUD operations
  – Graph update: INSERT, INSERT DATA, DELETE DATA, DELETE WHERE, LOAD, CLEAR)
  – Graph management: CREATE, DROP, COPY, MOVE, ADD
• Entailments
  – RDF, RDFS, OWL, RIF
• ...
SPARQL 1.1 as a unifying point

- Triple store
- SPARQL Endpoint
- SPARQL Construct
- SPARQL Update
- RDF Graph
- RDF
- RDFa
- NLP Techniques
- GRDDL, RDFa
- SQL/RDF
- Relational Database
- HTML
- Unstructured Text
- XML/XHTML

Application

SPARQL Processor