Axel Polleres

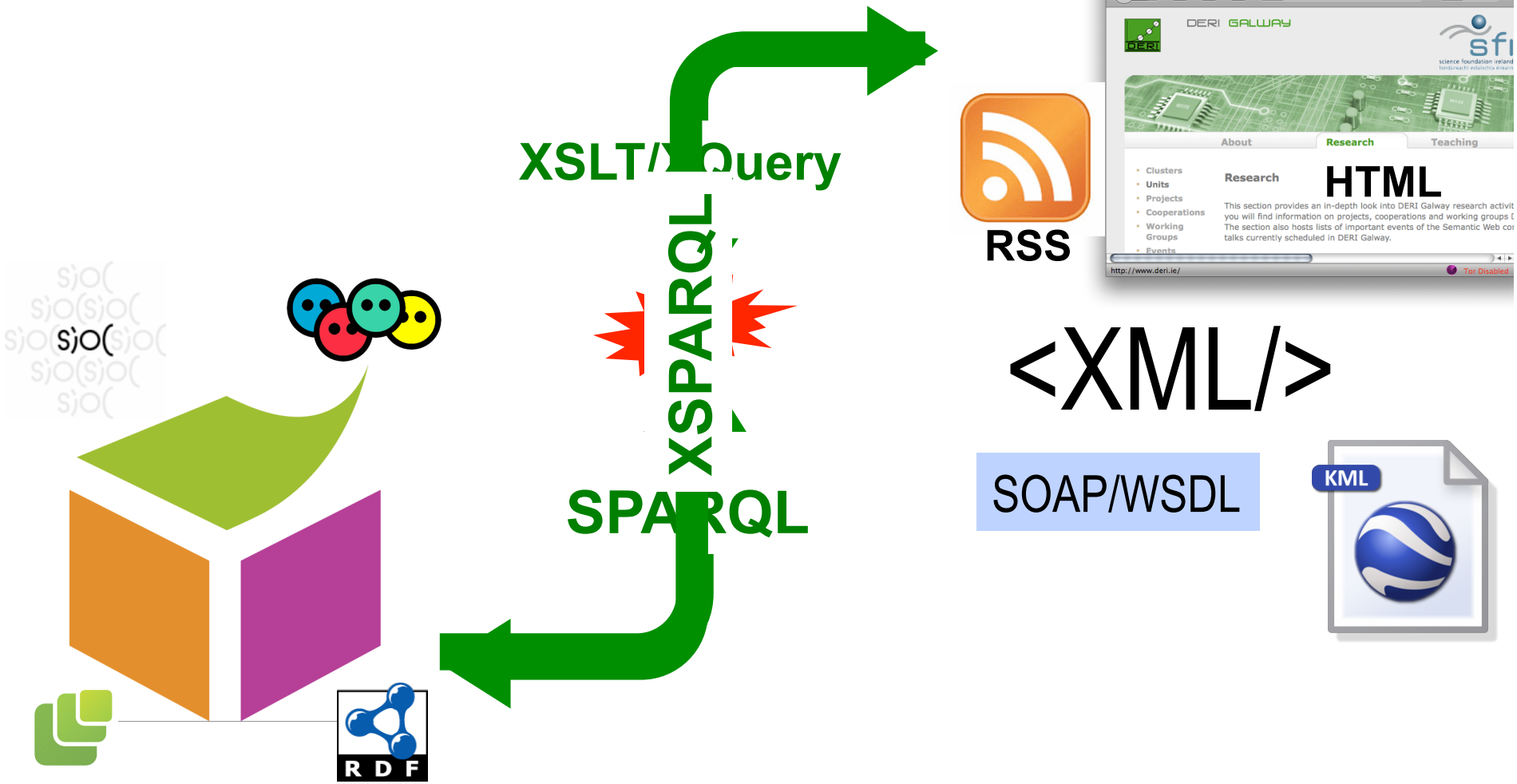
Querying and Exchanging XML and RDF on the Web with XSPARQL

(most slides taken from WWW'2012 Tutorial)

http://polleres.net/WWW2012Tutorial/

This tutorial presents partially joint work with: Nuno Lopes (DERI), Stefan Bischof (Siemens AG), Daniele Dell'Aglio (Politecnico Di Milano)…
… and of course the whole W3C SPARQL WG

XML & RDF: one Web – two formats



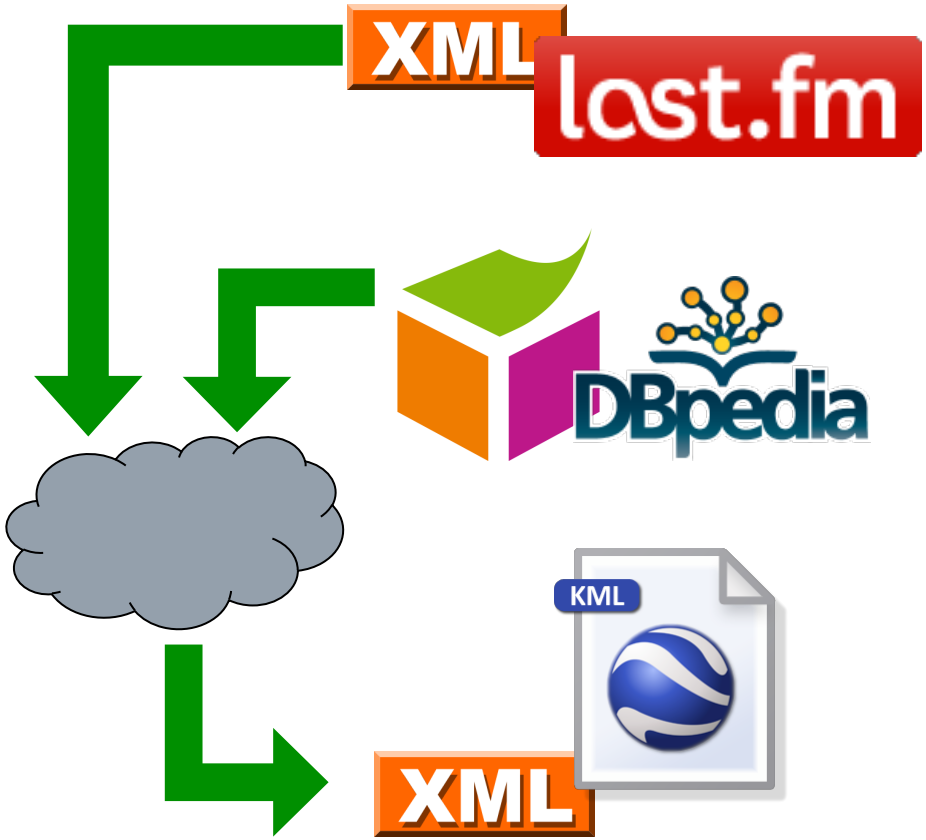
Who knows already about RDF?

Who knows already about SPARQL?

A Sample Scenario…

Example: Favourite artists location

Display information about your favourite artists on a map



Last.fm knows what music you listen to, your most played artists, etc. and provides an XML API.

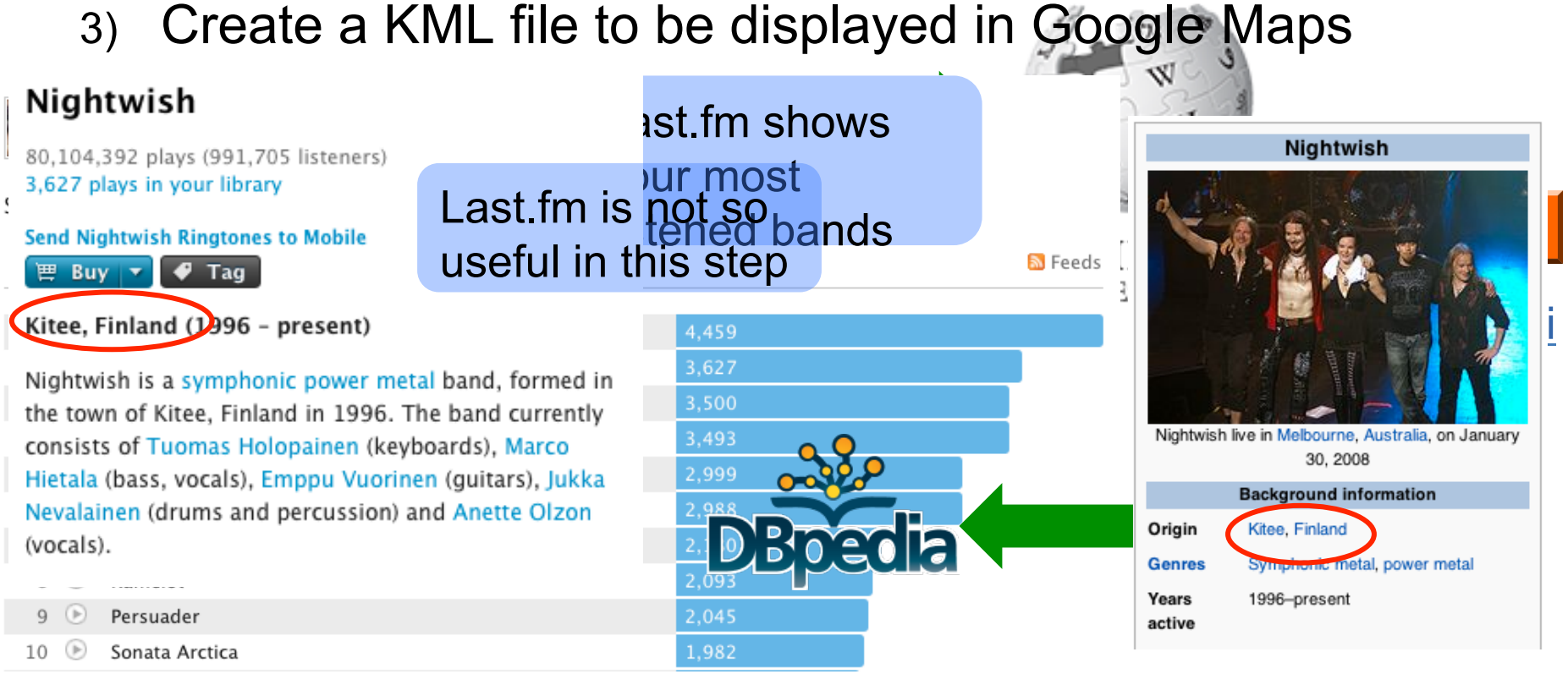
Using **RDF** allows to combine Last.fm info with other information on the web, e.g. location.

Show your top bands hometown in Google Maps, using KML – an XML format.

Example: Favourite artists location

How to implement this use case?

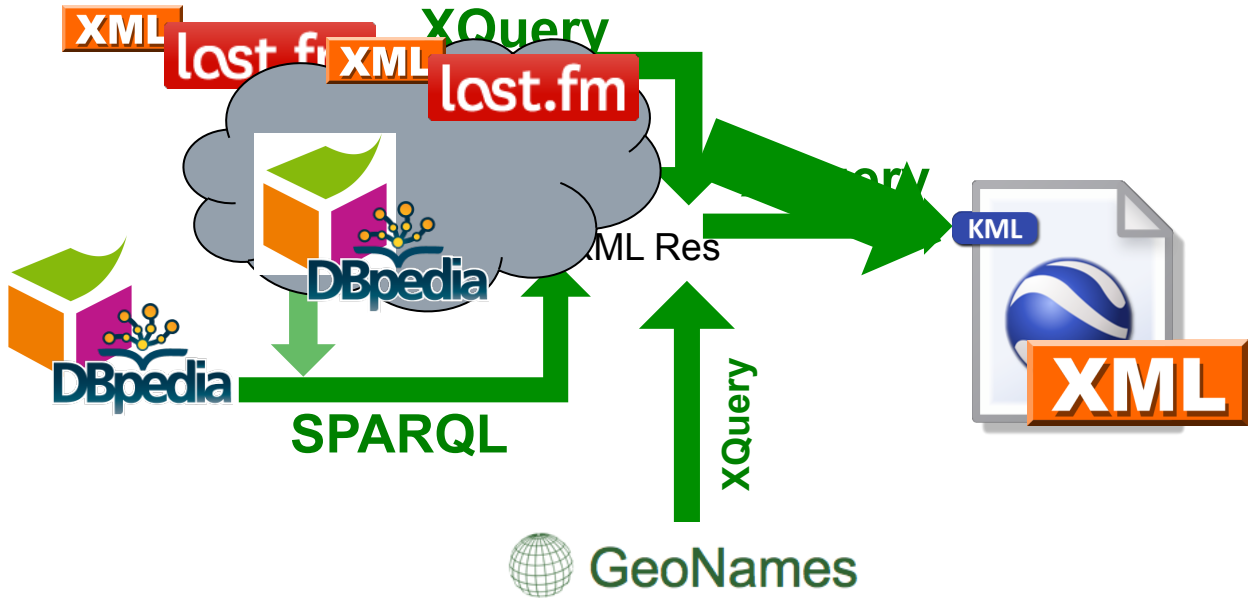
1) Get your favourite bands – from lastfm
2) Get the hometown of the bands – from Dbpedia
3) Create a KML file to be displayed in Google Maps



Example: Favourite artists location

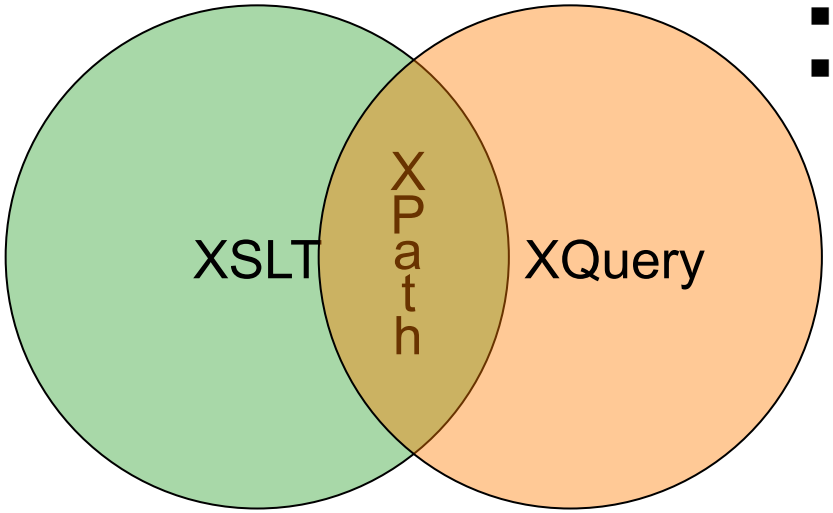
How to implement the visualisation?

1) Get your favourite bands
2) Get the hometown of the bands
3) Create a KML file to be displayed in Google Maps



Transformation and Query Languages

XML Transformation Language
Syntax: XML



- XPath is the common core
- Mostly used to select nodes from an XML doc

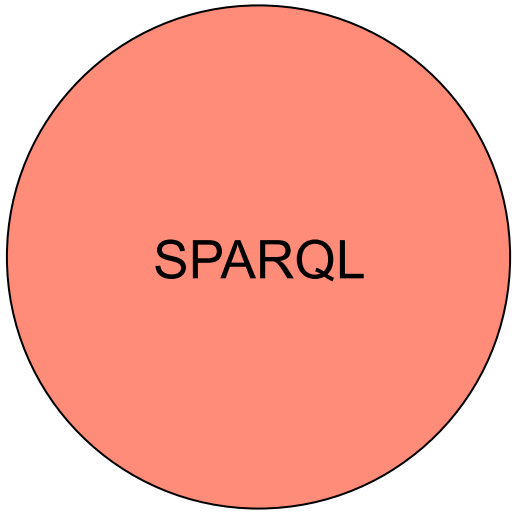
XML world / RDF world

- XML Query Language
- non-XML syntax

RDF/XML… ambiguous

- Query Language for RDF
- Pattern based
- declarative

SPARQL XML Result format



Lecture Overview

Part 1: **RDF & SPARQL Overview**

Part 2: **XSPARQL: a combined language integrating SPARQL with XQuery**

Assumption: you all know Xquery ☺

XQuery: Back to our Sample Scenario…

Querying XML Data from Last.fm 1/2

```
<lfm status="ok">
  <topartists type="overall">
    <artist rank="1">
      <name>Therion</name>
      <playcount>4459</playcount>
      <url>http://www.last.fm/music/Therion</url>
    </artist>
    <artist rank="2">
      <name>Nightwish</name>
      <playcount>3627</playcount>
      <url>http://www.last.fm/music/Nightwish</url>
    </artist>
  </topartists>
</lfm>
```

Last.fm API format:

- root element: “lfm”, then “topartists”
- sequence of “artist”

Querying this document with XPath:

XPath steps: `/lfm` Selects the “lfm” root element

`//artist` Selects all the “artist” elements

XPath Predicates `//artist[@rank = 1]` Selects the “artist” with rank 1

Querying XML Data from Last.fm 2/2

iterate over sequences

assign values to variables

filter expressions

create XML elements

Prolog:	**P**	`declare namespace` *prefix="namespace-URI"*
Body:	**F**	`for` *var* `in` *XPath-expression*
	L	`let` *var* `:=` *XPath-expression*
	W	`where` *XPath-expression*
	O	`order by` *XPath-expression*
Head:	**R**	`return` *XML + nested XQuery*

Query:

Retrieve information regarding a users' 2nd top artist from the

Last.fm API

```
let $doc := "http://ws.audioscrobbler.com/2.0/user.gettopartist"
for $artist in doc($doc)//artist
where $artist[@rank = 2]
return <artistData>{$artist}</artistData>
```

```
<artistData>
   <artist rank="2">
      <name>Nightwish</name>
            <playcount>3850</playcount>
            <mbid>00a9f935-ba93-4fc8-a33a-993abe9c936b</mbid>
                  <url>http://www.last.fm/music/Nightwish</url>
      <streamable>1</streamable>
            <image size="small">http://userserve-ak.last.fm/serve/34/149929.jpg</image>
        <image size="medium">http://userserve-ak.last.fm/serve/64/149929.jpg</image>
        <image size="large">http://userserve-ak.last.fm/serve/126/149929.jpg</image>
        <image size="extralarge">http://userserve-ak.last.fm/serve/252/149929.jpg</image>
        <image size="mega">http://userserve-ak.last.fm/serve/500/149929/Nightwish.jpg</image>
    </artist>
</artistData>
```

Result for user “jacktrades”

Query:

Retrieve information regarding a users' 2nd top artists from the

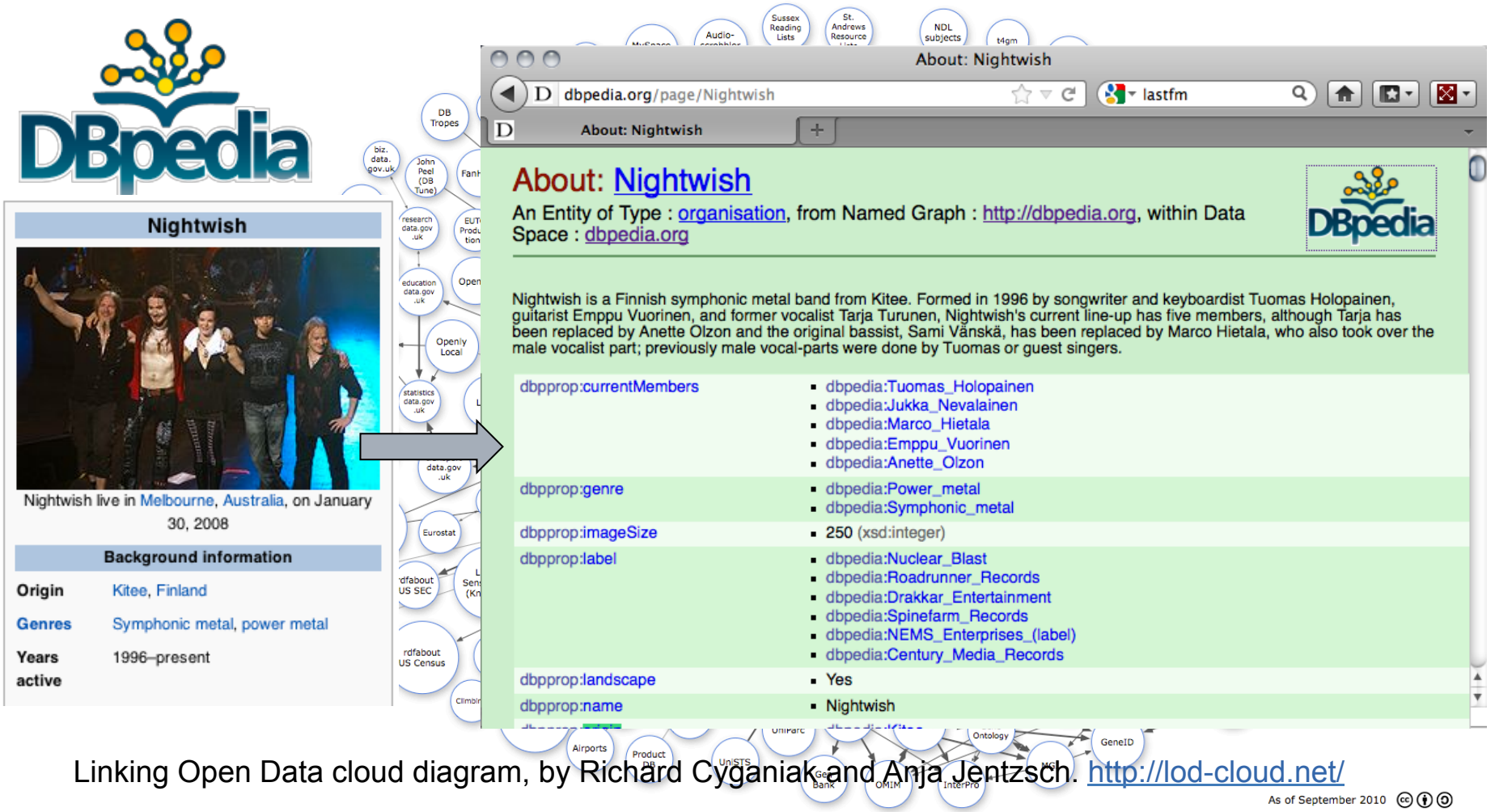
Last.fm API

```
let $doc := "http://ws.audioscrobbler.com/2.0/user.gettopartist"
for $artist in doc($doc)//artist
where $artist[@rank = 2]
return <artistData>{$artist}</artistData>
```

Now what about RDF Data?

RDF is an increasingly popular format for Data on the Web:

… lots of RDF Data is out there, ready to “query the Web”, e.g.:



Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

As of September 2010

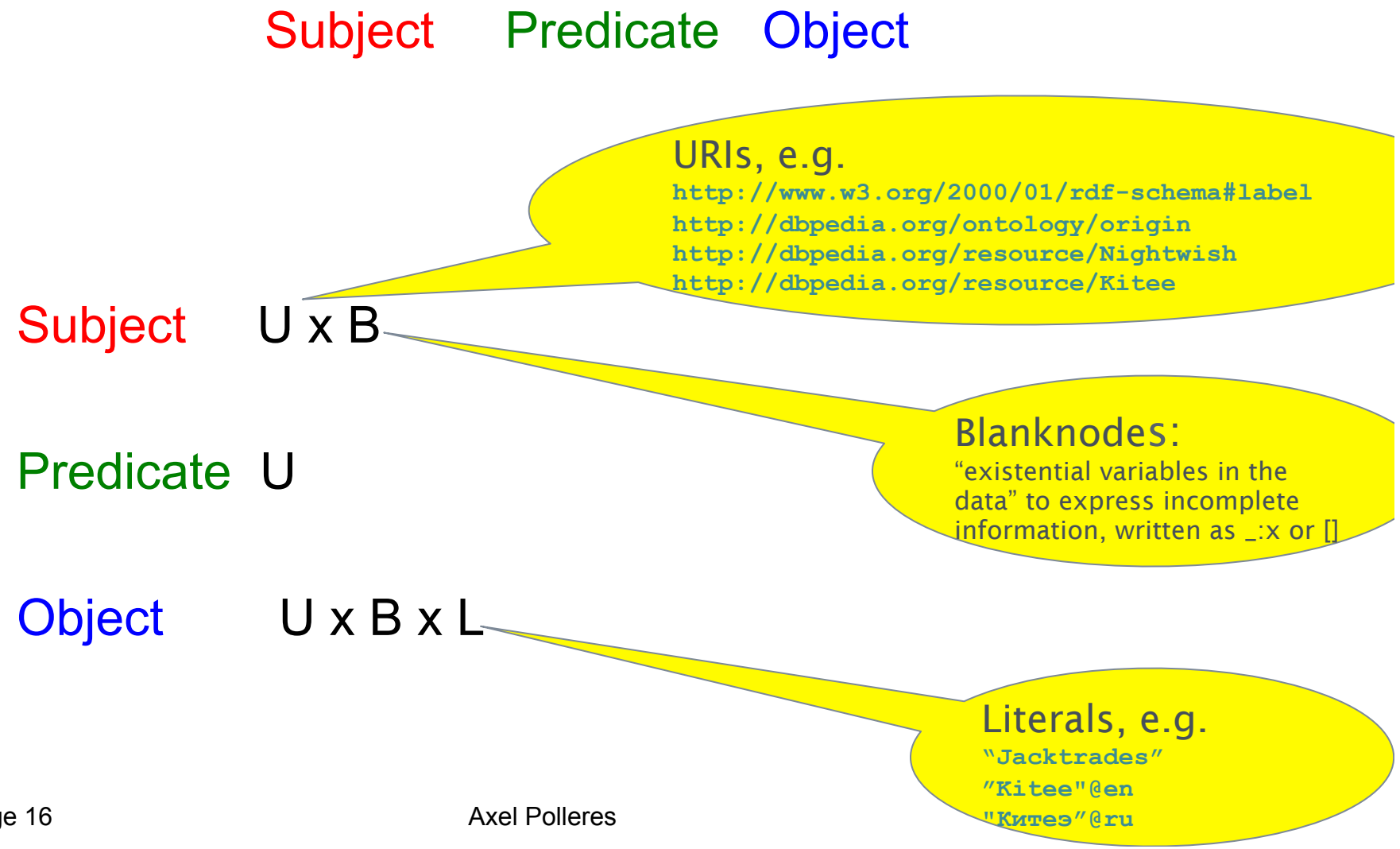
XML vs. RDF

XML: *"treelike" semi-structured Data (mostly schema-less, but "implicit" schema by tree structure… not easy to combine, e.g. how to combine lastfm data with wikipedia data?*

```
<table>
   <tr>
      <th colspan="2">Background information</th>
   </tr>
   <tr>
      <th>Origin</th>
      <td>
         <a title="Kitee" href="/wiki/Kitee">Kitee</a>, <a title="Finland" href="/wiki/Finland">Finland</a>
      </td>
   </tr>
   <tr>
      <th>
         <a title="Music genre" href="/wiki/Music_genre">Genres</a>
      </th>
      <td>
         <a title="Symphonic metal" href="/wiki/Symphonic_metal">Symphonic metal</a>, <a title="Gothic metal"
/Gothic_metal">gothic metal</a>
      </td>
   </tr>
   <tr>
      <th>Years active</th>
      <td>1996-present</td>
   </tr>
</table>
```

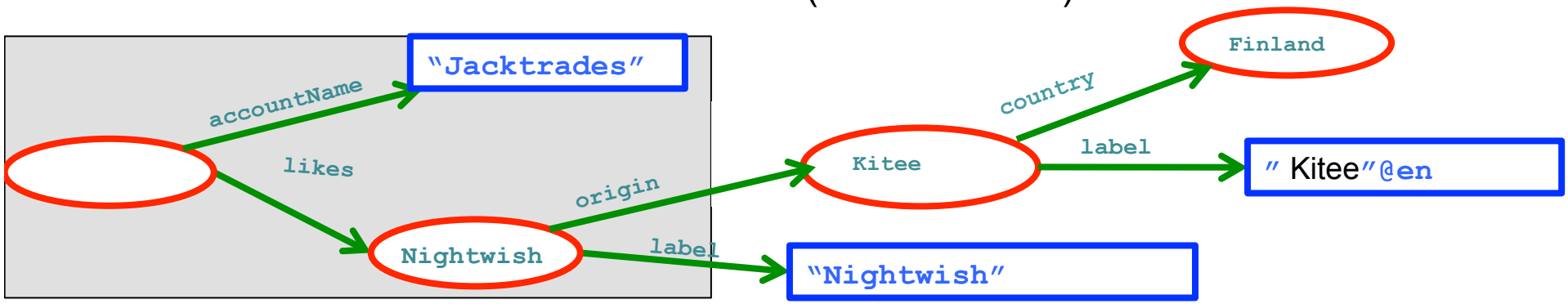
```
<artistData>
   <artist rank="2">
      <name>Nightwish</name>
            <playcount>3850</playcount>
            <mbid>00a9f935-ba93-4fc8-a33a-993abe9c936b</mbid>
                  <url>http://www.last.fm/music/Nightwish</url>
      <streamable>1</streamable>
            <image size="small">http://userserve-ak.last.fm/serve/34/149929.jpg</image>
        <image size="medium">http://userserve-ak.last.fm/serve/64/149929.jpg</image>
        <image size="large">http://userserve-ak.last.fm/serve/126/149929.jpg</image>
        <image size="extralarge">http://userserve-ak.last.fm/serve/252/149929.jpg</image>
        <image size="mega">http://userserve-ak.last.fm/serve/500/149929/Nightwish.jpg</image>
    </artist>
</artistData>
```

RDF Simple, declarative, graph-style format
based on dereferenceable URIs (= Linked Data)



RDF

Simple, declarative, graph-style format
based on dereferenceable URIs (= Linked Data)



Various syntaxes, RDF/XML, Turtle, N3, RDFa,...

```
<http://dbpedia.org/resource/Nightwish> <http://dbpedia.org/property/origin>
                                <http://dbpedia.org/resource/Kitee> .
<http://dbpedia.org/resource/Kitee> <http://www.w3.org/2000/01/rdf-schema#label>
                               "Kitee"@es .
```

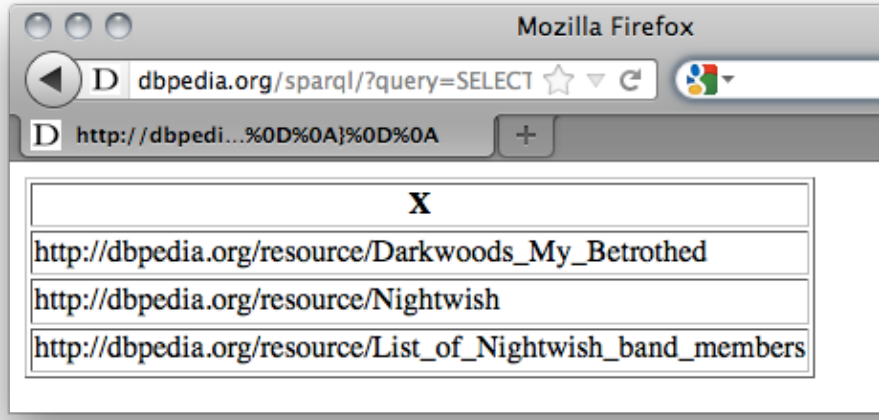
```
_:x <http://xmlns.com/foaf/0.1/accountName> "Jacktrades" .
_:x <http://graph.facebook.com/likes> <http://dbpedia.org/resource/Nightwish> .
```

How to query RDF?

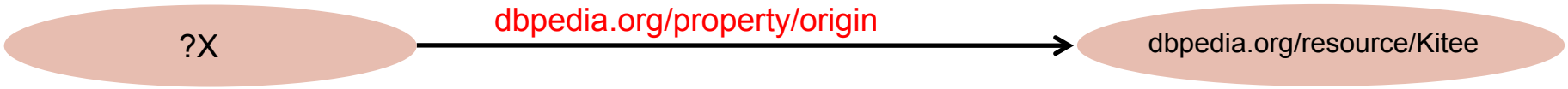
SPARQL in a Nutshell...

SPARQL + Linked Data give you Semantic search almost “for free”

Which bands origin from Kitee?




```
SELECT ?X
WHERE
{
 ?X  <http://dbpedia.org/property/origin> <http://dbpedia.org/resource/Kitee>
}
```



Try it out at http://live.dbpedia.org/sparql/

SPARQL – Standard RDF Query Language and Protocol

SPARQL (2008):

```
SELECT ?X
WHERE
{
 ?X  <http://dbpedia.org/property/origin> <http://dbpedia.org/resource/Kitee>
}
```

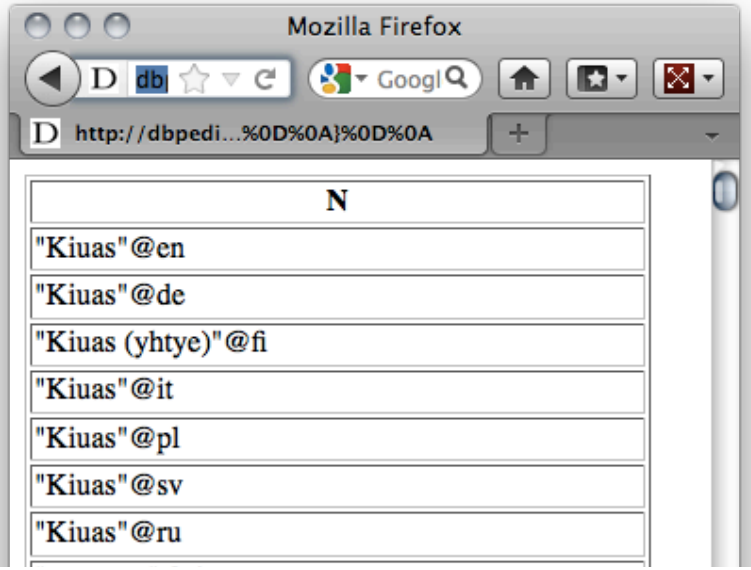
- SQL “Look-and-feel” for the Web
- Essentially “graph matching” by *triple patterns*
- Allows conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)
- Construct new RDF from existing RDF
- Solution modifiers (DISTINCT, ORDER BY, LIMIT, …)
- A **standardized** HTTP based protocol:

Conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)

***Names** of **bands** from **cities in Finland**?*

```
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX dbont: <http://dbpedia.org/ontology/>
PREFIX category: <http://dbpedia.org/resource/Category:>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dcterms: <http://purl.org/dc/terms/>

SELECT ?N
WHERE
{
 ?X a dbont:Band ; rdfs:label ?N ;
    dbprop:origin [ dcterms:subject category:Cities_and_towns_in_Finland] .
}
```

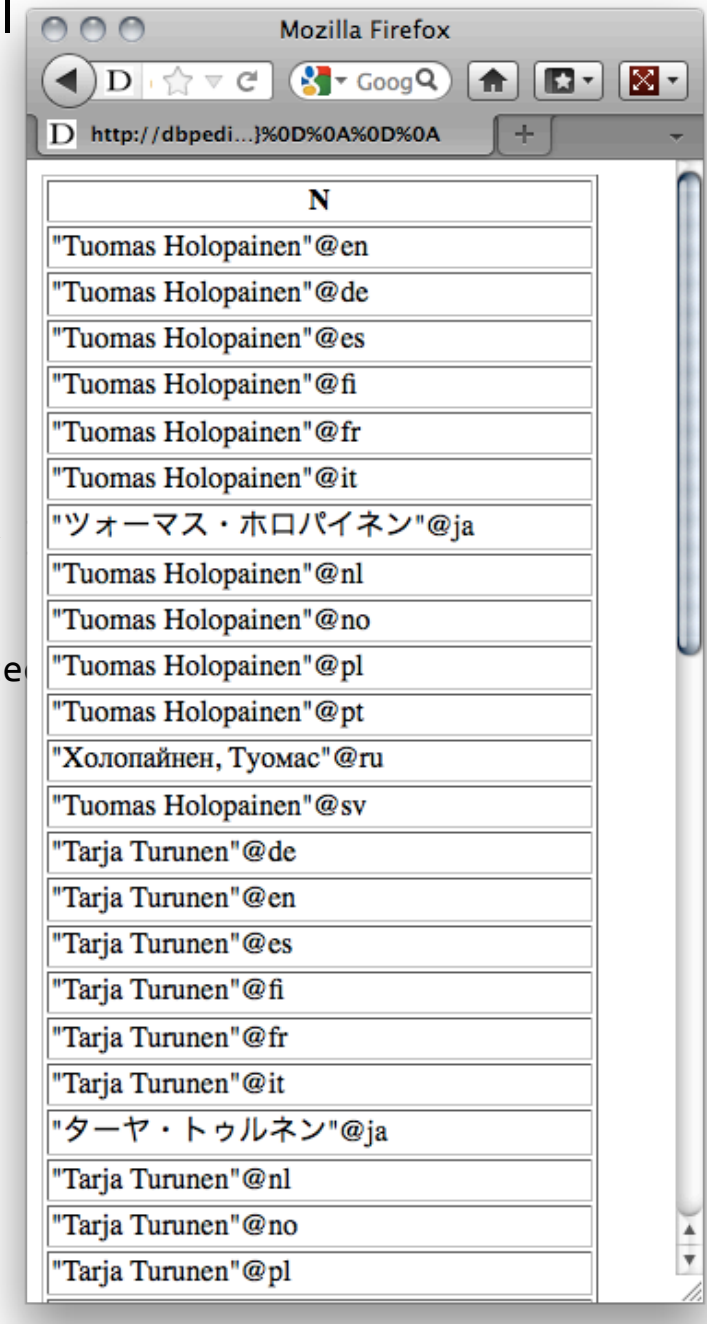
- *Shortcuts for namespace prefixes and to group triple patterns*



Conjunction (.) , **disjunction (UNION)**, optional (OPTIONAL) patterns and filters (FILTER)

*Names of things **that origin or were born in** Kitee?*

```
SELECT ?N
WHERE
{
 { ?X  dbprop:origin <http://dbpedia.org/resource/Kitee>
 UNION
 { ?X  dbont:birthPlace <http://dbpedia.org/resource/Kite

?X rdfs:label ?N
}
```



N
"Tuomas Holopainen"@en
"Tuomas Holopainen"@de
"Tuomas Holopainen"@es
"Tuomas Holopainen"@fi
"Tuomas Holopainen"@fr
"Tuomas Holopainen"@it
"ツォーマス・ホロパイネン"@ja
"Tuomas Holopainen"@nl
"Tuomas Holopainen"@no
"Tuomas Holopainen"@pl
"Tuomas Holopainen"@pt
"Холопайнен, Туомас"@ru
"Tuomas Holopainen"@sv
"Tarja Turunen"@de
"Tarja Turunen"@en
"Tarja Turunen"@es
"Tarja Turunen"@fi
"Tarja Turunen"@fr
"Tarja Turunen"@it
"ターヤ・トゥルネン"@ja
"Tarja Turunen"@nl
"Tarja Turunen"@no
"Tarja Turunen"@pl

Conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and **filters (FILTER)**

Cites Finland with a

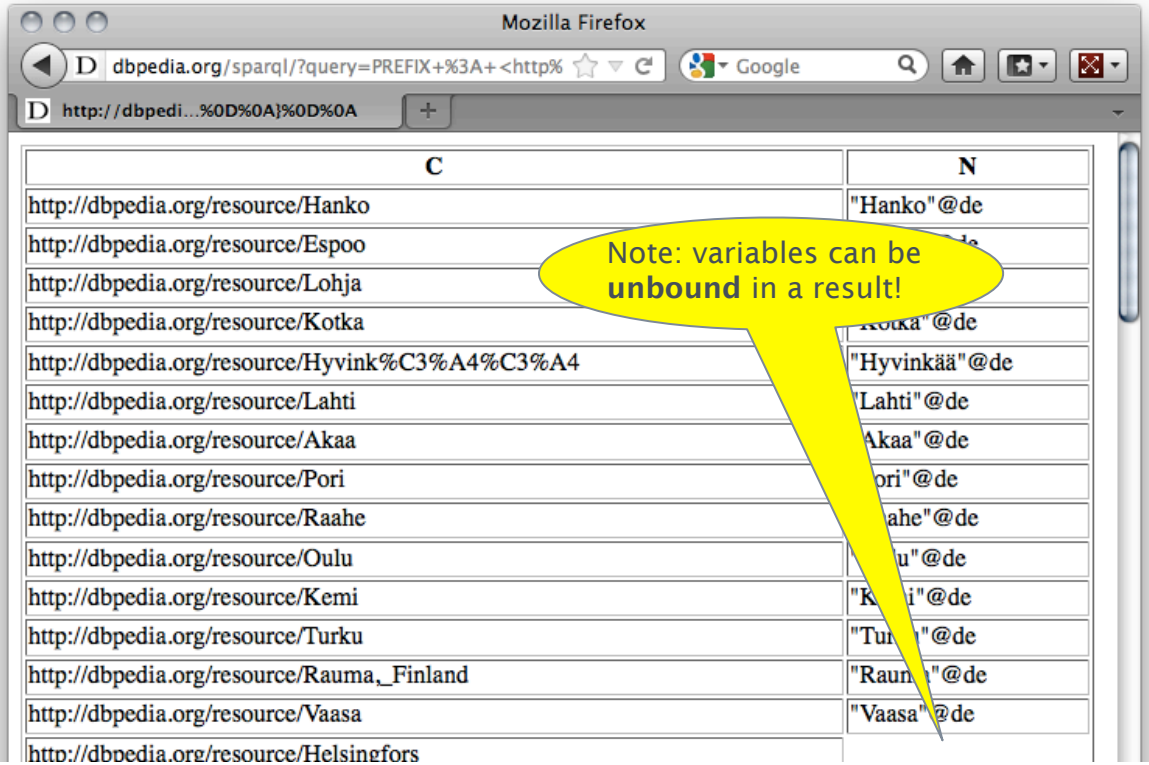
```
SELECT ?C ?N
WHERE
{
 ?C  dcterms:subject category:Cities_and_towns_in_Finland ;
     rdfs:label ?N .
 FILTER( LANG(?N) = "de" )

}
```

SPARQL has lots of FILTER functions to filter text with regular expressions (REGEX), filter numerics (<,>,=,+,-…), dates, etc.)

Conjunction (.) , disjunction (UNION), **optional (OPTIONAL)** patterns and filters (FILTER**)**

Cites Finland with optionally their German (@de) name

```
SELECT ?C ?N
WHERE
{
 ?C  dcterms:subject category:Cities_and_towns_in_Finland .
 OPTIONAL { ?C rdfs:label ?N . FILTER( LANG(?N) = "de" ) }
}
```



C	N
http://dbpedia.org/resource/Hanko	"Hanko"@de
http://dbpedia.org/resource/Espoo	
http://dbpedia.org/resource/Lohja	
http://dbpedia.org/resource/Kotka	"@de
http://dbpedia.org/resource/Hyvink%C3%A4%C3%A4	"Hyvinkää"@de
http://dbpedia.org/resource/Lahti	"Lahti"@de
http://dbpedia.org/resource/Akaa	"Akaa"@de
http://dbpedia.org/resource/Pori	ori"@de
http://dbpedia.org/resource/Raahe	ahe"@de
http://dbpedia.org/resource/Oulu	u"@de
http://dbpedia.org/resource/Kemi	"K i"@de
http://dbpedia.org/resource/Turku	"Tur "@de
http://dbpedia.org/resource/Rauma,_Finland	"Raun "@de
http://dbpedia.org/resource/Vaasa	"Vaasa" @de
http://dbpedia.org/resource/Helsingfors	

Missing features in SPARQL1.0 (and why SPARQL1.1 was needed)

Based on implementation experience, in 2009 new W3C SPARQL WG founded to address common feature requirements requested urgently by the community: ***http://www.w3.org/2009/sparql/wiki/Main_Page***

1. **Negation**
2. **Assignment/Project Expressions**
3. **Aggregate functions (SUM, AVG, MIN, MAX, COUNT, …)**
4. **Subqueries**
5. **Property paths**

6. **Updates**
7. **Entailment Regimes**

- Other issues for wider usability:
  - Result formats (JSON, CSV, TSV),
  - Graph Store Protocol (REST operations on graph stores)

  - ***SPARQL 1.1 is a W3C Recommendation since 21 March 2013***

1. Negation: MINUS and NOT EXISTS

Select Persons without a homepage:

```
SELECT ?X
WHERE{ ?X rdf:type foaf:Person
          FILTER ( NOT EXISTS { ?X foaf:homepage ?H } ) }
```

***SPARQL1.1** has two alternatives to do negation*

- *NOT EXISTS in FILTERs*
  - *detect non-existence*

1. Negation: MINUS and NOT EXISTS

Select Persons without a homepage:

```
SELECT ?X
WHERE{ ?X rdf:type foaf:Person
       MINUS { ?X foaf:homepage ?H } ) }
```

***SPARQL1.1** has two alternatives to do negation*

- *NOT EXISTS in FILTERs*
  - *detect non-existence*
- *(P1 MINUS P2) as a new binary operator*
  - *“Remove rows with matching bindings”*
  - *only effective when P1 and P2 share variables*

2. Assignment/Project Expressions

Assignments, Creating new values… now available in SPARQL1.1

```
PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr * 1.1 AS ?NewP )
WHERE { ?Item ex:price ?Pr }
```

Data:

```
@prefix ex: <http://example.org/> .

ex:lemonade1     ex:price 3 .
ex:beer1         ex:price 3 .
ex:wine1         ex:price 3.50 .
```

Results:

?Item	?NewP
lemonade1	3.3
beer1	3.3
wine1	3.85

3. Aggregates

"Count items per categories"

```
PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T
```

Data:

```
@prefix ex: <http://example.org/> .

ex:lemonade1    ex:price 3 ;
                rdf:type ex:Softdrink.
ex:beer1        ex:price 3;
                rdf:type ex:Beer.
ex:wine1        ex:price 3.50 ;
                rdf:type ex:Wine.
ex:wine2        ex:price 4 .
                rdf:type ex:Wine.
ex:wine3        ex:price "n/a";
                rdf:type ex:Wine.
```

Results:

?T	?C
Softdrink	1
Beer	1
Wine	3

4. Subqueries

- How to create new triples that concatenate first name and last name?
- Possible with SELECT sub-queries or BIND

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

CONSTRUCT{ ?P foaf:name ?FullName }
WHERE {
  SELECT ?P ( fn:concat(?F, " ", ?L) AS ?FullName )
  WHERE { ?P foaf:firstName ?F ; foaf:lastName ?L. }
}
```

4. Subqueries

- How to create new triples that concatenate first name and last name?
- Possible with SELECT sub-queries or BIND

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX fn: <http://www.w3.org/2005/xpath-functions#>

CONSTRUCT{ ?P foaf:name ?FullName }
WHERE {
  ?P foaf:firstName ?F ; foaf:lastName ?L.
  BIND ( fn:concat(?F, " ", ?L) AS ?FullName )
}
```

5. Property Path expressions

Arbitrary Length paths, Concatenate property paths, etc.
E.g. names of people Tim Berners-Lee transitively co-authored papers with…

```
SELECT DISTINCT ?N
 WHERE {  <http://dblp…/Tim_Berners-Lee>
        (^foaf:maker/foaf:maker)+/foaf:name ?N
       }
```

Path expressions full list of operators

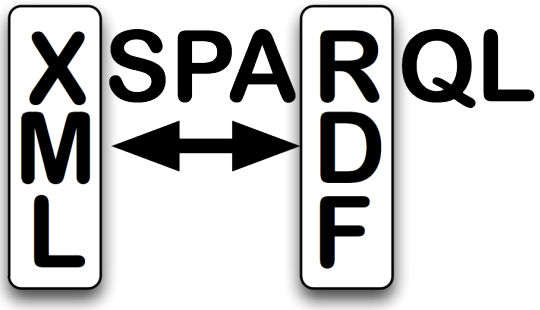
- elt … Path Element

Syntax Form	Matches
uri	A URI or a prefixed name. A path of length one.
^elt	Inverse path (object to subject).
!uri or *!(uri$_1$/ .../uri$_n$)*	Negated property set. A URI which is not one of *uri$_i$*
!^uri and *!(uri$_1$/ .../uri$_j$/^uri$_{j+1}$/ .../^uri$_n$)*	Negated property set. A URI which is not one of *uri$_i$*, nor uri$_{j+1}$...^uri$_n$ as reverse paths
(elt)	A group path *elt*, brackets control precedence.
elt1 / elt2	A sequence path of *elt1*, followed by *elt2*
elt1 | elt2	A alternative path of *elt1*, or *elt2* (all possibilities are tried).
*elt**	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.
elt?	A path of zero or one *elt*.

- Details: http://www.w3.org/TR/sparql11-query/#propertypaths

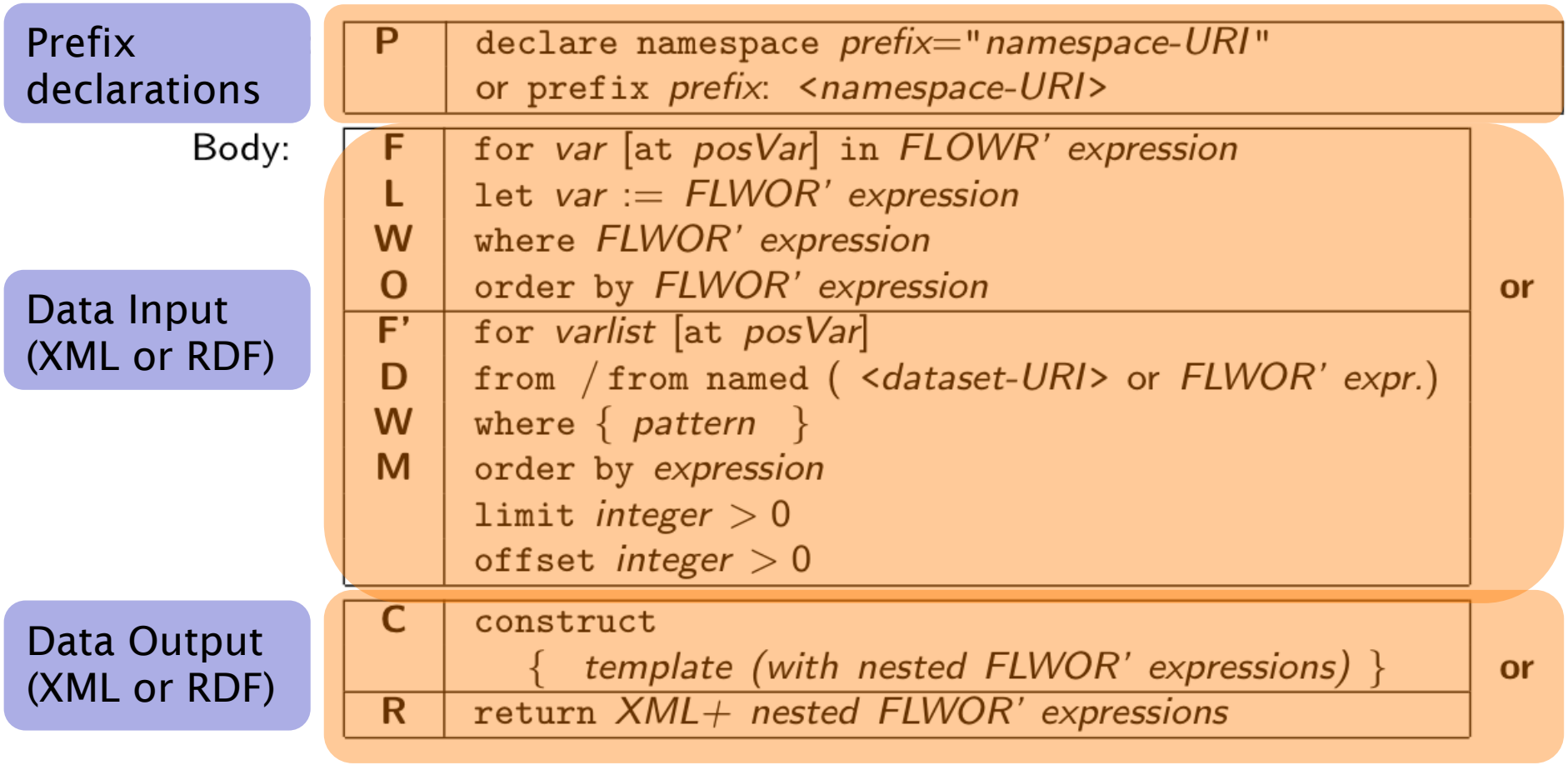
XSPARQL

Idea: One approach to conveniently query XML and RDF side-by-side: XSPARQL



- Transformation language
- Consume and generate XML and RDF
- Syntactic extension of XQuery, ie.
  XSPARQL = XQuery + SPARQL

XSPARQL: Syntax overview (I)



XSPARQL Syntax overview (II)

XQuery or SPARQL prefix declarations

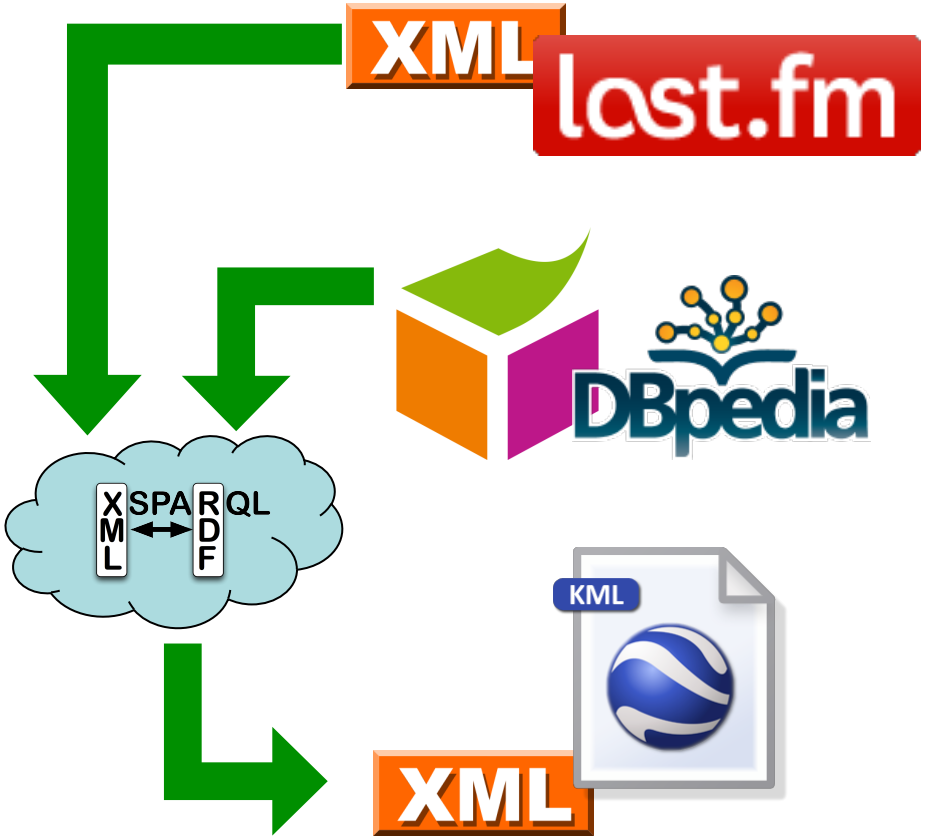
Any XQuery query

SPARQLFOR Clause represents a SPARQL query

construct allows to create RDF

P	`declare namespace` *prefix="namespace-URI"* or `prefix` *prefix*: <*namespace-URI*>	
F L W O	`for` *var* [`at` *posVar*] `in` *FLOWR' expression* `let` *var* := *FLWOR' expression* `where` *FLWOR' expression* `order by` *FLWOR' expression*	**or**
F' D W M	`for` *varlist* [`at` *posVar*] `from` / `from named` (<*dataset-URI*> or *FLWOR' expr.*) `where` { *pattern* } `order by` *expression* `limit` *integer* > 0 `offset` *integer* > 0	
C	`construct` { *template (with nested FLWOR' expressions)* }	**or**
R	`return` *XML+ nested FLWOR' expressions*	

Use case



XSPARQL: Convert XML to RDF

Query:

Convert Last.fm top artists of a user into RDF

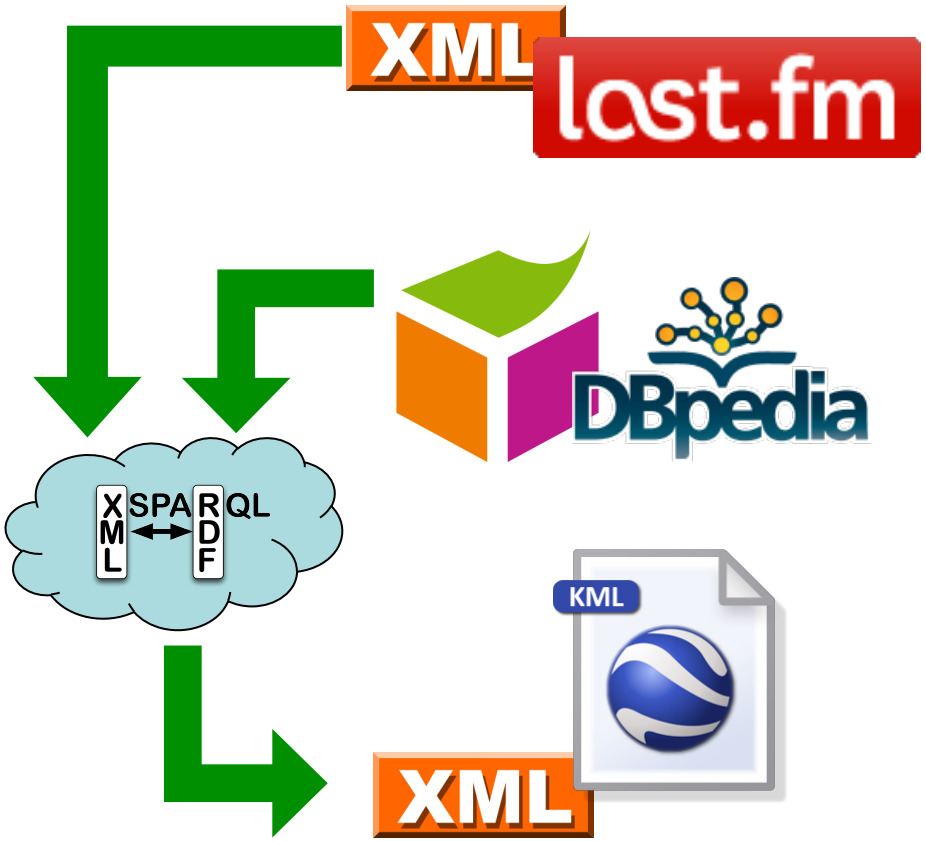
```
prefix lastfm: <http://xsparql.deri.org/lastfm#>

let $doc := "http://ws.audioscrobbler.com/2.0/?method=user.gettopartists"
for $artist in doc($doc)//artist
where $artist[@rank < 6]
construct { [] lastfm:topArtist {$artist//name};
               lastfm:artistRank {$artist//@rank} . }
```

Result:

```
@prefix lastfm: <http://xsparql.deri.org/lastfm#> .

[ lastfm:topArtist "Therion" ;  lastfm:artistRank "1" ] .
[ lastfm:topArtist "Nightwish" ;  lastfm:artistRank "2" ] .
[ lastfm:topArtist "Blind Guardian" ;  lastfm:artistRank "3" ] .
[ lastfm:topArtist "Rhapsody of Fire" ;  lastfm:artistRank "4" ] .
[ lastfm:topArtist "Iced Earth" ;  lastfm:artistRank "5" ] .
```

XSPARQL construct generates valid Turtle RDF

Use case



XSPARQL: Integrate RDF sources

Query:
Retrieve the origin of an artist from DBPedia: Same as the SPARQL query

```
prefix dbprop: <http://dbpedia.org/property/>
prefix foaf:   <http://xmlns.com/foaf/0.1/>

construct { $artist foaf:based_near $origin }
from <http://dbpedia.org/resource/Nightwish>
where { $artist dbprop:origin $origin }
```

Issue: determining the artist identifiers

DBPedia does not have the map coordinates

XSPARQL: Integrate RDF sources

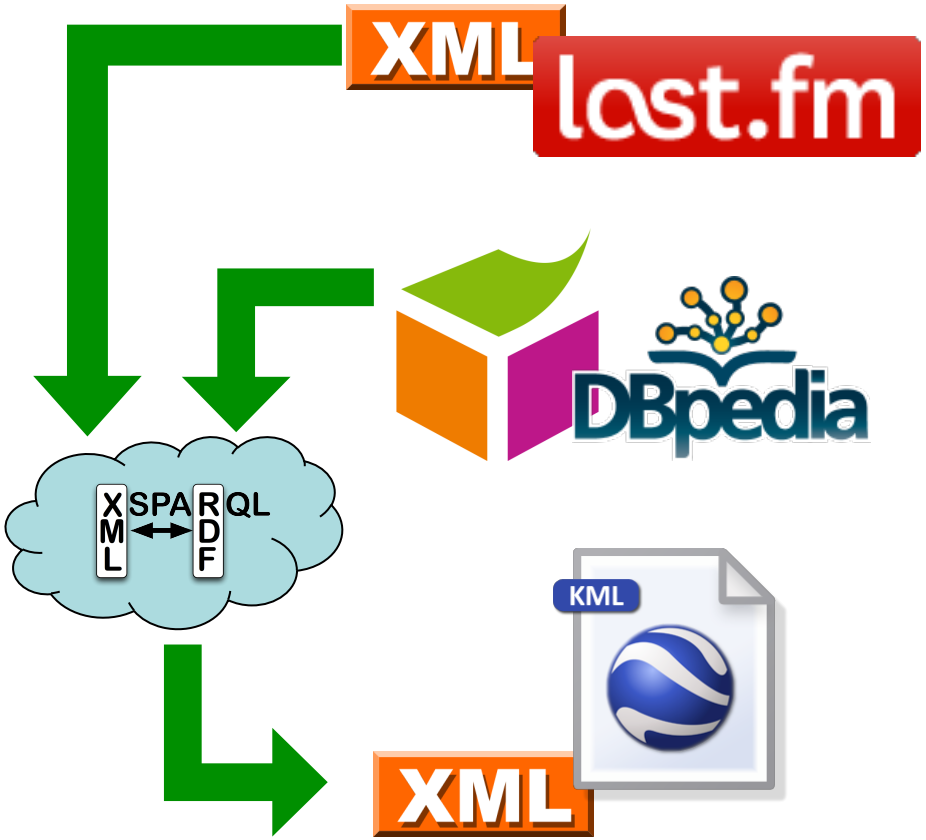
Query:
Retrieve the origin of an artist from DBPedia ***including map coordinates***

```
prefix wgs84_pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix dbprop: <http://dbpedia.org/property/>

for * from <http://dbpedia.org/resource/Nightwish>
where { $artist dbprop:origin $origin }
return
let $hometown :=
  fn:concat("http://api.geonames.org/search?type=rdf&q=",fn:encode-for-uri($origin))
for * from $hometown
where { [] wgs84_pos:lat $lat; wgs84_pos:long $long }
limit 1
construct { $artist wgs84_pos:lat $lat; wgs84_pos:long $long }
```

DBPedia does not have the map coordinates

GeoNames

Use case



Output: KML XML format

```
<kml xmlns="http://www.opengis.net/kml/2.2">
   <Document>
      <Placemark>
         <name>Hometown of Nightwish</name>
         <Point>
            <coordinates>
              30.15,62.1,0
            </coordinates>
         </Point>
      </Placemark>
 </Document>
</kml>
```

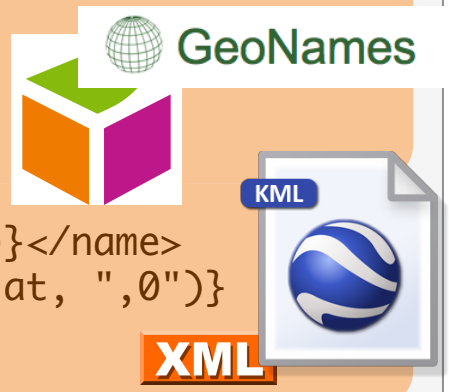
KML format:

- root element: “kml”, then “Document”
- sequence of “Placemark”
- Each “Placemark” contains:
  - “Name” element
  - “Point” element with the “coordinates”

XSPARQL: Putting it all together

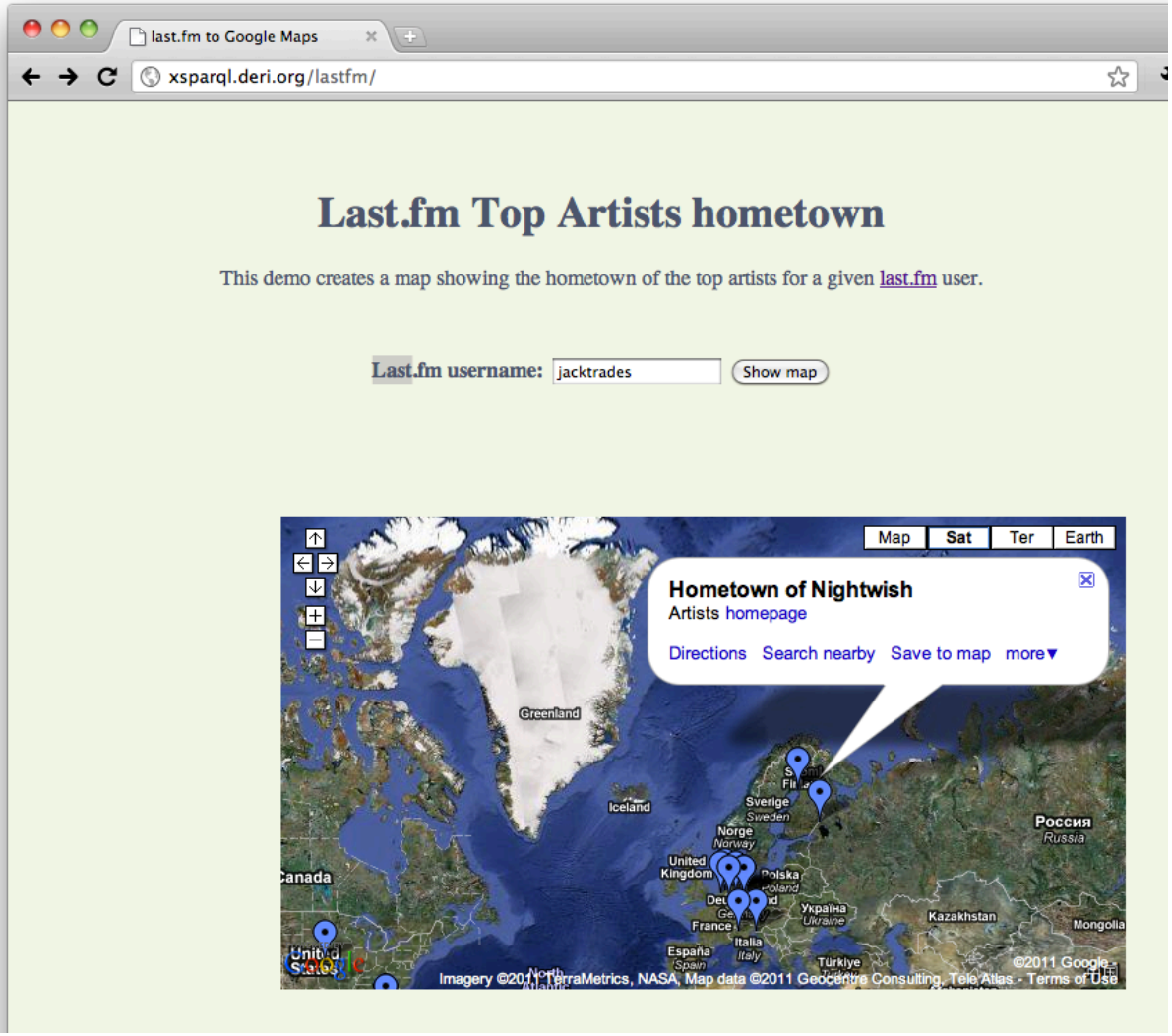
Query: Display top artists origin in a map

```
prefix dbprop: <http://dbpedia.org/property/>

<kml><Document>{
let $doc := "http://ws.audioscrobbler.com/2.0/?method=user.gettopartists"
for $artist in doc($doc)//artist
return let $artistName := fn:data($artist//name)
  let $uri := fn:concat("http://dbpedia.org/resource/", $artistName)
  for $origin from $uri
  where { [] dbprop:origin $origin }
  return
   let $hometown := fn:concat("http://api.geonames.org/search?type=rdf&q=",
                    fn:encode-for-uri($origin))
   for * from $hometown
   where { [] wgs84_pos:lat $lat; wgs84_pos:long $long }
   limit 1
   return <Placemark>
            <name>{fn:concat("Hometown of ", $artistName)}</name>
            <Point><coordinates>{fn:concat($long, ",", $lat, ",0")}
            </coordinates></Point>
           </Placemark>
}</Document></kml>
```



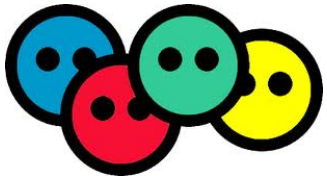
XSPARQL: Demo

http://xsparql.deri.org/lastfm



XSPARQL: more examples

XSPARQL: Convert FOAF to KML



RDF (FOAF) data representing your location … *in different ways*

Show this information in a Google Map embedded in your webpage

XSPARQL: Convert FOAF to KML

```
 <foaf:based_near>                    http://nunolopes.org/foaf.rdf
      <geo:Point>
        <geo:lat>53.289881</geo:lat><geo:long>-9.073849</geo:long>
      </geo:Point>
    </foaf:based_near>
```

Display location in Google Maps based on your FOAF file

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>

<kml xmlns="http://www.opengis.net/kml/2.2">{
 for $name $long $lat
 from <http://nunolopes.org/foaf.rdf>
 where { $person a foaf:Person; foaf:name $name;
         foaf:based_near [ a geo:Point; geo:long $long;
                           geo:lat $lat ] }
return <Placemark>
         <name>{fn:concat("Location of ", $name)}</name>
         <Point>
           <coordinates>{fn:concat($long, ",", $lat, ",0")}
           </coordinates>
         </Point>
       </Placemark>
}</kml>
```

XSPARQL: Convert FOAF to KML

Different location representation in different foaf files…

http://polleres.net/foaf.rdf

```
<foaf:based_near rdf:resource="http://dbpedia.org/resource/Galway"/>
```

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix georss: <http://www.georss.org/georss/>

<kml><Document>{
  for * from <http://polleres.net/foaf.rdf>
  where { $person a foaf:Person; foaf:name $name;
          foaf:based_near $point. }
  return for * from $point
         where { $c georss:point $latLong }
         return let $coordinates := fn:tokenize($latLong, " ")
            let $lat1 := $coordinates[1]
            let $long1 := $coordinates[2]
            return <Placemark>
              <name>{fn:concat("Location of ", $name)}</name>
                <Point><coordinates>{fn:concat($long1, ",", $lat1, ",0")}
             </coordinates></Point>
            </Placemark>
}</Document></kml>
```

We can handle different representations of locations in the FOAF files

XSPARQL: Convert FOAF to KML

you can cater for different representations in one query…

http://polleres.net/foaf.rdf

```
<foaf:based_near rdf:resource="http://dbpedia.org/resource/Galway"/>
```

http://nunolopes.org/foaf.rdf

```
<foaf:based_near>
     <geo:Point>
       <geo:lat>53.289881</geo:lat><geo:long>-9.073849</geo:long>
     </geo:Point>
   </foaf:based_near>
```

- Previous 2 queries can be easily combined into one… see:
  http://xsparql.deri.org/foaf2kml/foaf2kml.xsparql

Obtaining locations in RDF

- Update or enhance your FOAF file with your current location based on a Google Maps search:

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix kml: <http://earth.google.com/kml/2.0>

let $loc := "Hilton San Francisco Union Square, San Francisco, CA"
for $place in doc(fn:concat("http://maps.google.com/?q=",
                  fn:encode-for-uri($loc),
                  ”&num=1&output=kml"))
let $geo := fn:tokenize($place//kml:coordinates, ",")
construct { <nunolopes> foaf:based_near [ geo:long {$geo[1]};
                                          geo:lat {$geo[2]} ] }
```

Find the location in Google Maps and get the result as KML

Result:

```
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix kml: <http://earth.google.com/kml/2.0> .

<nunolopes> foaf:based_near [ geo:long "-122.411116" ;
                              geo:lat "37.786000" ] .
```

XSPARQL vs. SPARQL for “pure RDF” queries

Extending SPARQL1.0: Computing values

Computing values is not possible in SPARQL 1.0:

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix : <http://xsparql.deri.org/geo#>

construct { $person :latLong $lat; :latLong $long }
from <http://nunolopes.org/foaf.rdf>
where { $person a foaf:Person; foaf:name $name;
        foaf:based_near [ geo:long $long;
                          geo:lat $lat ] }
```

While XSPARQL allows to use all the XPath functions:

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
prefix : <http://xsparql.deri.org/geo#>

construct { $person :latLong {fn:concat($lat, " ", $long)}}
from <http://nunolopes.org/foaf.rdf>
where { $person a foaf:Person; foaf:name $name;
        foaf:based_near [ geo:long $long;
                          geo:lat $lat ] }
```

Note: SPARQL1.1 allow that (BIND)

Federated Queries in SPARQL1.1

Find which persons in DBPedia have the same birthday as Axel (foaf-file):

SPARQL 1.1 has new feature SERVICE to query remote endpoints

```
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?N ?MyB
FROM <http://polleres.net/foaf.rdf>
{ [ foaf:birthday ?MyB ].

  SERVICE <http://dbpedia.org/sparql> { SELECT ?N WHERE {
    [ dbpedia2:born ?B; foaf:name ?N ]. FILTER ( Regex(str(?B),str(?MyB)) ) } }
}
```

Doesn't work!!! **?MyB** unbound in SERVICE query

Federated Queries in SPARQL1.1

Find which persons in DBPedia have the same birthday as Axel (foaf-file):

SPARQL 1.1 has new feature SERVICE to query remote endpoints

```
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?N ?MyB
FROM <http://polleres.net/foaf.rdf>
{ [ foaf:birthday ?MyB ].

  SERVICE <http://dbpedia.org/sparql> { SELECT ?N WHERE {
    [ dbpedia2:born ?B; foaf:name ?N ]. } }

 FILTER ( Regex(Str(?B),str(?MyB)) )
}
```

Doesn't work either in practice ☹ as SERVICE endpoints often only returns limited results…

Federated Queries

Find which persons in DBPedia have the same birthday as Axel (foaf-file):

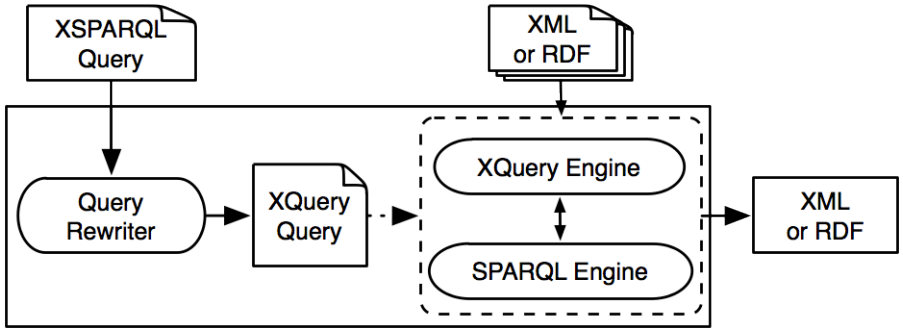
In XSPARQL:

```
prefix dbprop: <http://dbpedia.org/property/>
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix : <http://xsparql.deri.org/bday#>

let $MyB := for * from <http://polleres.net/foaf.rdf>
            where { [ foaf:birthday $B ]. }
            return $B

for * from <http://dbpedia.org/> endpoint <http://dbpedia.org/sparql>
where { [ dbprop:born $B; foaf:name $N ].
          filter ( regex(str($B),str($MyB)) )  }
construct { :axel :sameBirthDayAs $N }
```

Specifies the endpoint to perform the query, similar to SERVICE in SPARQL1.1

Works! In XSPARQL bound values (**?MyDB**) are **injected** into the SPARQL subquery
→ More direct control over "query execution plan"

XSPARQL Implementation



- Each XSPARQL query is translated into a native XQuery
- SPARQLForClauses are translated into SPARQL SELECT clauses
- Uses *off the shelf* components:
  - XQuery engine: Saxon
  - SPARQL engine: Jena / ARQ

Example:

relations.rdf

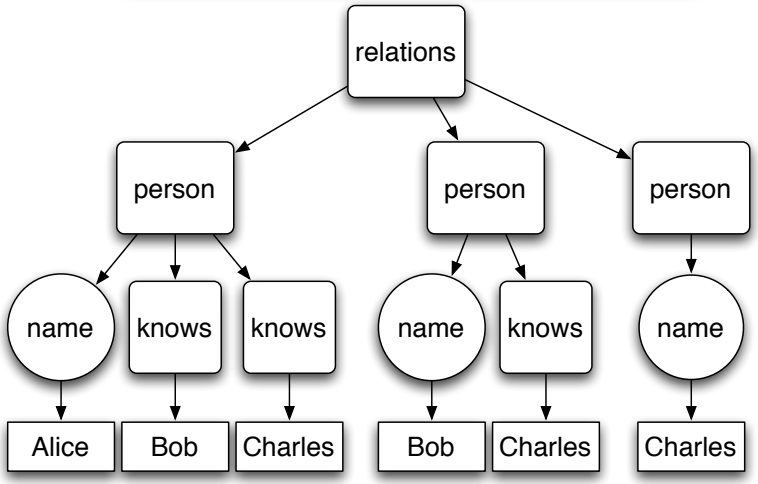
```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
     foaf:name "Alice";
     foaf:knows _:b2;
     foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```

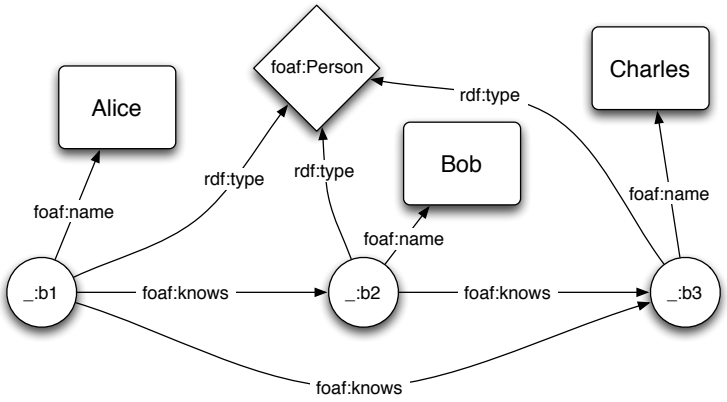
Lowering

Lifting

relations.xml

```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```





Example: Mapping from RDF to XML

```
<relations>
{ for $Person $Name
  from <relations.rdf>
  where { $Person foaf:name $Name }
  order by $Name
  return
      <person name="{$Name}">
      {for $FName
       from <relations.rdf>
       where {
            $Person foaf:knows $Friend .
            $Person foaf:name $Name .
            $Friend foaf:name $Fname }
       return <knows>{$FName}</knows>
    } </person>
}</relations>
```

```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

Example: Adding value generating functions to SPARQL (using XSPARQL to emulate a SPARQL1.1 feature)

```
construct { :me foaf:knows _:b .
            _:b foaf:name {fn:concat("""",?N," ",?F,"""")} }
from <MyAddrBookVCard.rdf>
where {
       ?ADDR vc:Given ?N .
       ?ADDR vc:Family ?F .
}
```

…

```
:me foaf:knows _:b1. _:b1 foaf:name "Peter Patel-Schneider" .
:me foaf:knows _:b2. _:b2 foaf:name "Stefan Decker" .
:me foaf:knows _:b3. _:b3 foaf:name "Thomas Eiter" .
```

…

XSPARQL Implementation … very simplified…
Rewriting XSPARQL to XQuery…

```
construct { _:b foaf:name {fn:concat($N," ",$F)} } from
<vcard.rdf>
where { $P vc:Given $N . $P vc:Family $F . }
```

```
let $aux_query := fn:concat("http://localhost:2020/sparql?query=",
                                       fn:encode-for-uri(
                  "select $P $N $F from <vcard.rdf>
                   where {$P vc:Given $N. $P vc:Family $F.}"))

for $aux_resu
                in doc($aux_query)//sparql_result:result

let $P_Node :                                :binding[@name="P"]
let $N_Node :                                :binding[@name="N"]
let $F_Node := $aux_result/sparql_result:binding[@name="F"]
let $N := data($N_Node/*)let $N_NodeType := name($N_Node/*)
let $N_RDFTerm := local:rdf_term($N_NodeType,$N)
...
return ( fn:c                                        "),
             ( f                                     """") ), "." )
```

1. Encode SPARQL in HTTP call SELECT Query

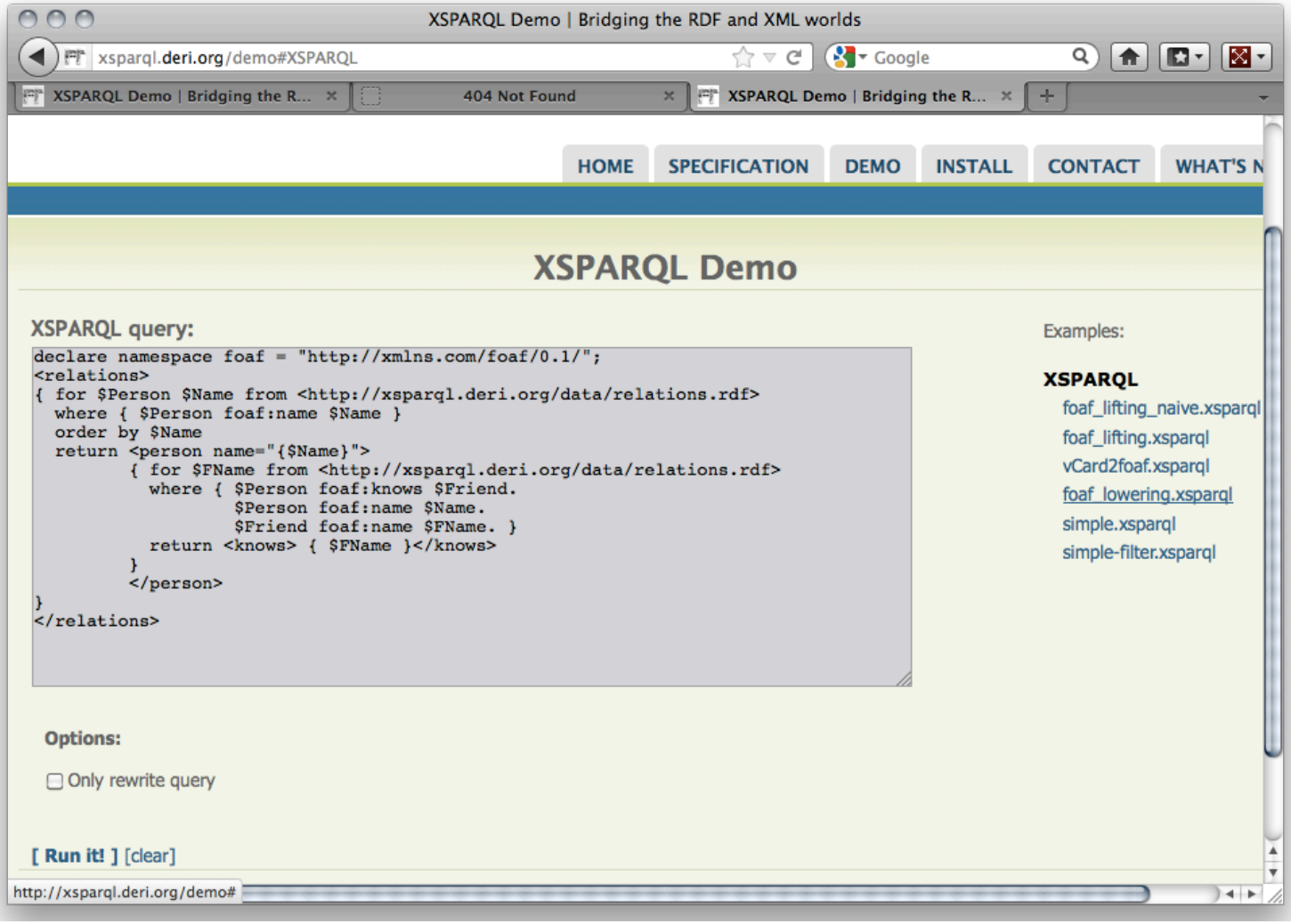
2. Execute call, via fn:doc function

3. Collect results from SPARQL result format (XML)

4. `construct` becomes `return` that outputs triples (slightly simplified)

Test Queries and show rewriting…

http://xsparql.deri.org/demo



Details about XSPARQL semantics and implementation (also about some optimizations)

Check our Journal paper:

Stefan Bischof, Stefan Decker, Thomas Krennwallner, Nuno Lopes, Axel Polleres: Mapping between RDF and XML with XSPARQL. J. Data Semantics 1(3): 147-185 (2012)
http://link.springer.com/article/10.1007%2Fs13740-012-0008-7

Demo time of the new "fresh from the oven" XSPARQL1.1 release!!!!

http://www.polleres.net/20140605xsparql1.1-sneak-preview/

(release should be made available on sourceforge by next week, hopefully… check: http://sourceforge.net/projects/xsparql/

BSc, MSc, PhD topics!!!! Please check: http://www.polleres.net/ or talk to me after the lecture!

We're looking for interested students to work on various exciting projects with partners in industry and public administration that involve

- integration tasks using XSPARQL
- Querying Linked Data
- foundations and extensions of SPARQL

Particularly:

- Integrating Open Data and making it available as Linked Data
- Integrating and Querying Processes and Data (FFG Project SHAPE)
  - Combining Answer Set Programming, Data Integration, Business Process Management