WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

web: www.polleres.net
twitter:@AxelPolleres

Querying the Web of Data with SPARQL and XSPARQL

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

Starting… ;-)

Querying the Web of Data with SPARQL and XSPARQL

(many slides taken from WWW'2012 Tutorial & from my Web Science Summer School Tutorial in St.Etienne)

http://polleres.net/WWW2012Tutorial/

http://polleres.net/20140826xsparql_st.etienne/

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

Which Data formats are popular on the Web?

RDF, XML, JSON

How to query and integrate data in these formats using **declarative query languages**?

SPARQL, XQuery, XSPARQL

RDF, XML & JSON: one Web of data – various formats

XSLT/XQuery

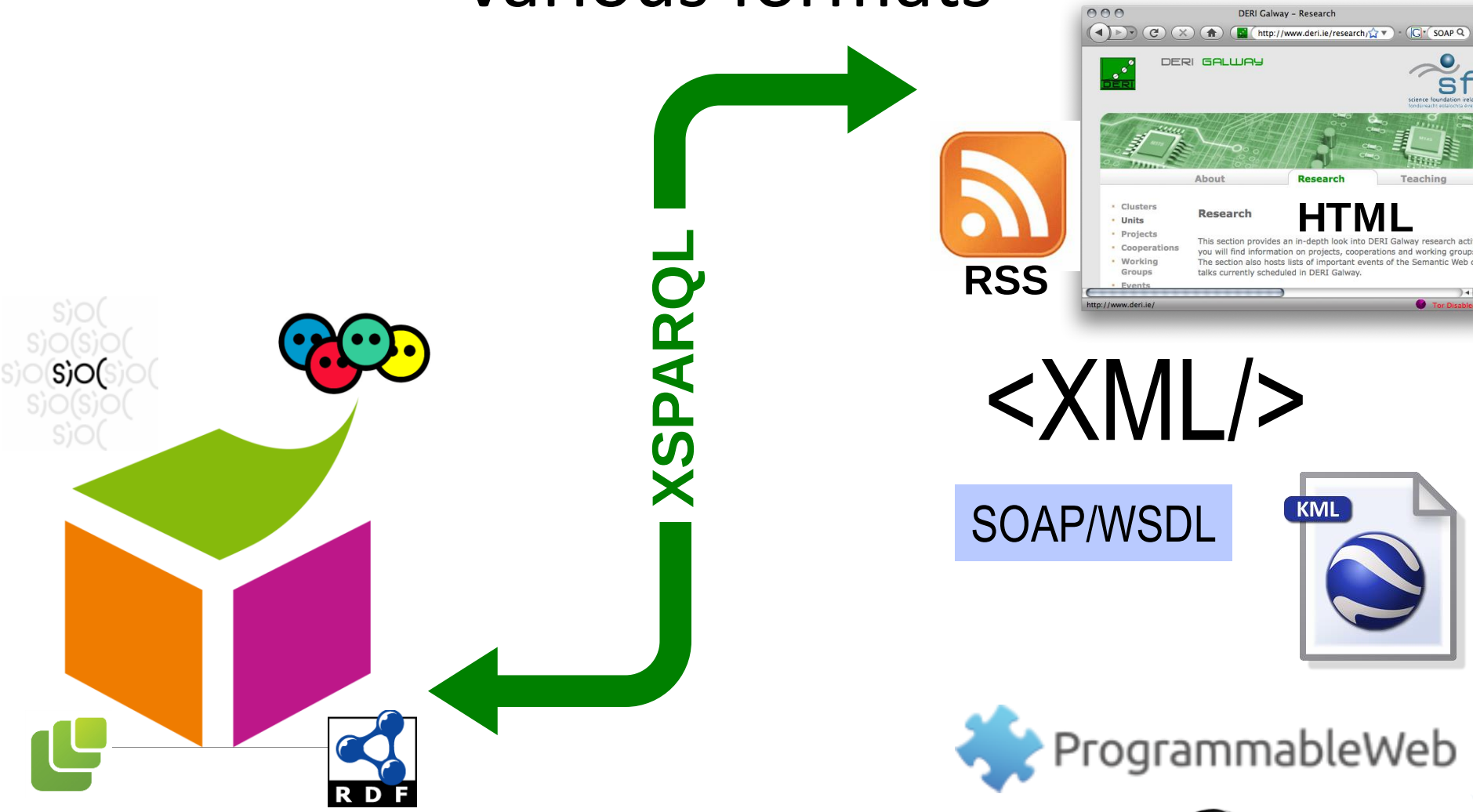
SPARQL

RSS

<XML/>

SOAP/WSDL

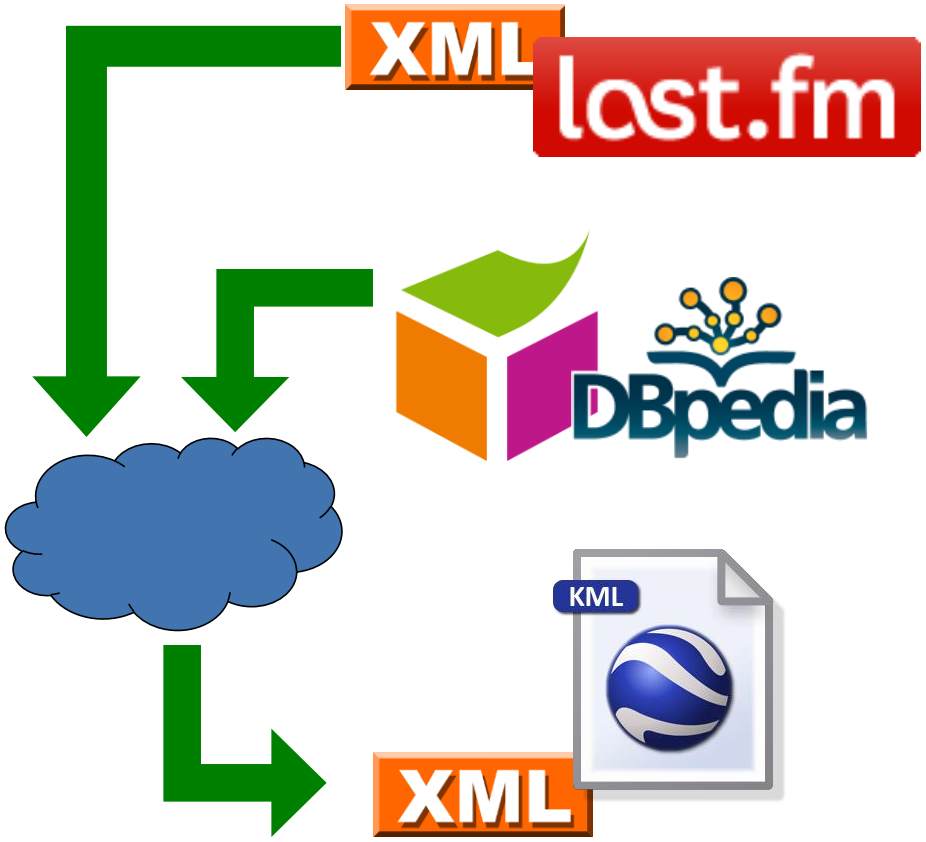
RDF, XML & JSON: one Web of data – various formats



A Sample Scenario…

Example: Favourite artists location

Display information about your favourite bands on a map



Last.fm knows what music you listen to, your most played artists, etc. and provides an **XML** (or **JSON**) 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

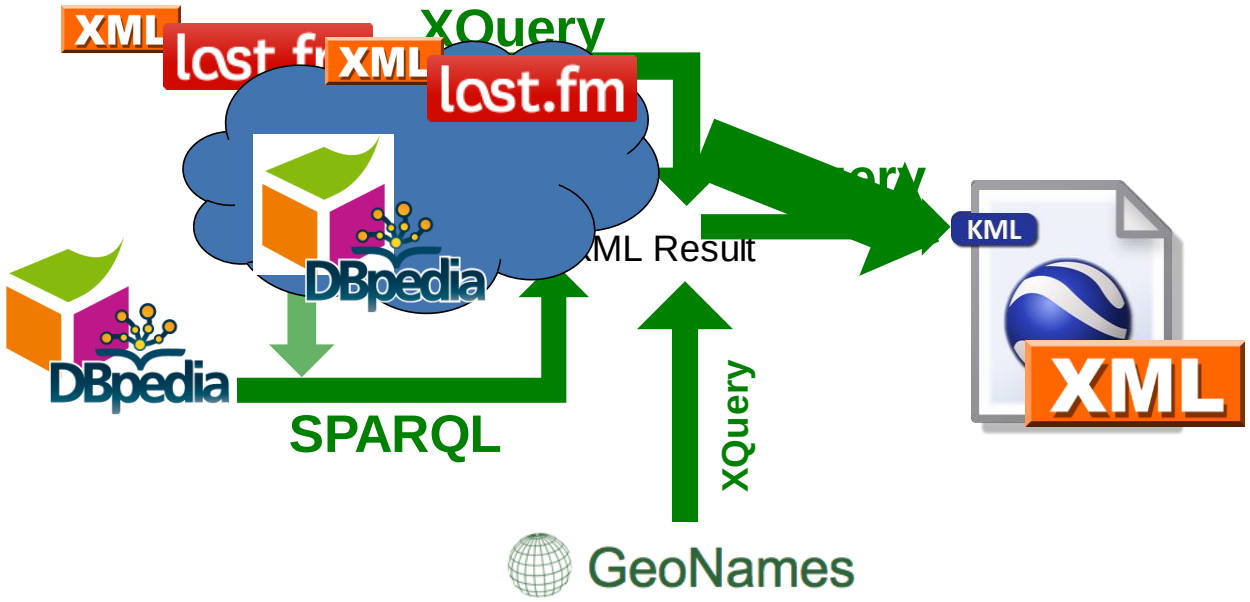
How to implement this use case?

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

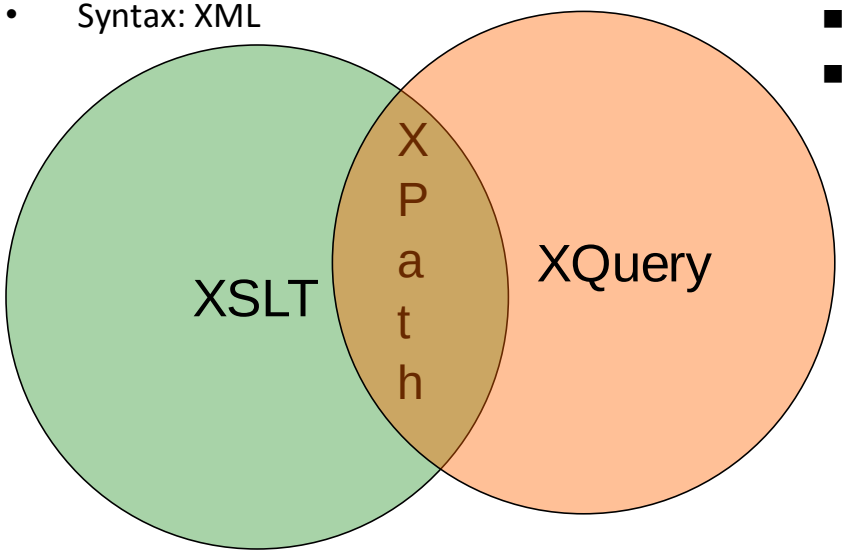
How to implement this use case?

1) Get your favourite bands
2) Get the hometown of the bands, and the geo locations
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

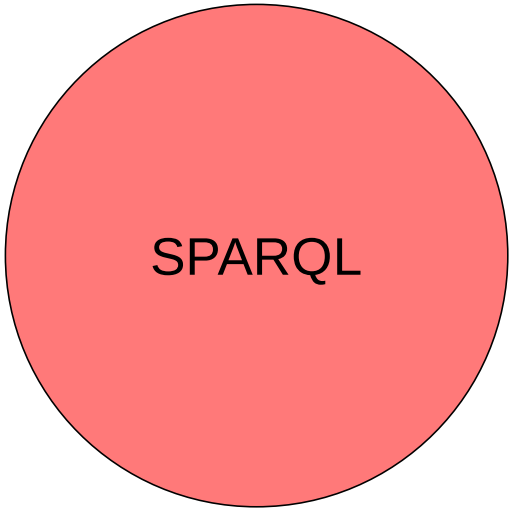
XML world / RDF world

- XML Query Language
- non-XML syntax

RDF/XML… ambiguous

- Query Language for RDF
- Pattern based
- declarative

SPARQL XML Result format



Lecture Overview

- Part 1: **Data Formats – quick recap**
  - **XML**
  - **JSON**
  - **XPath & Xquery in a nutshell**
- Part 2: **SPARQL-by-examples (as needed)**
- Part 3: **XSPARQL: a combined language integrating SPARQL with XQuery**
- Part 4: more examples and where to find further info…

XML & JSON: Back to our Sample Scenario…

Example: Favourite artists location

Last.fm knows what music you listen to, your most played artists, etc. and provides an XML API, which you can access if you have an account.

http://www.last.fm/api

Last.fm Web Services

user.getTopArtists

Get the top artists listened to by a user. You can stipulate a time period. Sends the overall chart by default.

Params

user (Required) : The user name to fetch top artists for.
period (Optional) : overall | 7day | 1month | 3month | 6month | 12month – The time period over which to retrieve top artists for.
limit (Optional) : The number of results to fetch per page. Defaults to 50.
page (Optional) : The page number to fetch. Defaults to first page.
api_key (Required) : A Last.fm API key.

Unfortunately, doesn't work anymore… ☹ (API seems changed)

Sample Call:

http://ws.audioscrobbler.com/2.0/method=user.gettopartists&user=jacktrades&api_key=…

Example: Favourite artists location

Find a sample result here:
http://polleres.net/20140826xsparql_st.etienne/xsparql/lastfm_user_sample.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<lfm status="ok">
 <topartists user="jacktrades" type="overall" page="1" perPage="50" totalPages="16" total="767">
  <artist rank="1">
   <name>Nightwish</name>
   <playcount>4958</playcount>
   <mbid>00a9f935-ba93-4fc8-a33a-993abe9c936b</mbid>
   <url>http://www.last.fm/music/Nightwish</url>
   <streamable>0</streamable>
   <image size="small">http://userserve-ak.last.fm/serve/34/84310519.png</image>
   <image size="medium">http://userserve-ak.last.fm/serve/64/84310519.png</image>
   <image size="large">http://userserve-ak.last.fm/serve/126/84310519.png</image>
</artist>
  <artist rank="2">
   <name>Therion</name>
   <playcount>4947</playcount>
   <mbid>c6b0db5a-d750-4ed8-9caa-ddcfb75dcb0a</mbid>
   ...
  </artist>
  ...
 </topartists>
</lfm>
```

JSON

JavaScript Object Notation

- Recently becoming even more popular than XML in the context of Web Data APIs
- More compact than XML
- Directly accessible for Javascript
- JSON Objects support simple types (string, number, arrays, boolean)

… if you want a bit like "Turtle" for XML (or tree-shaped, nested data in General)

except: no Namespaces or URIs per se

JSON

JavaScript Object Notation

Syntax

- **unordered** Set of attribute-value pairs.
- Each Object enclosed in '{' '}'.
- Attribute names followed by ':'
- Attribute-Value pairs separated by ' , '
- Like elements in XML, JSON Objects can be nested
- Arrays as ordered collections of values enclosed in '[' ']'

JSON Example

```
{
 "first": "Jimmy",
 "last": "James",
 "age": 29,
 "sex": "male",
 "salary": 63000,
 "department": {"id": 1, "name" : "Sales"},
 "registered": false,
 "lucky numbers": [ 2, 3, 11, 23],
 "listofCustomers":[ {"name": "Customer1" },
                     {"name": "Customer2" } ]
}
```

Example: Favourite artists location

Last.fm also provides its API in JSON… many other data services nowadays only provide JSON APIs!

http://www.last.fm/api

Last.fm Web Services

user.getTopArtists

Get the top artists listened to by a user. You can stipulate a time period. Sends the overall chart by default.

Params

user (Required) : The user name to fetch top artists for.
period (Optional) : overall | 7day | 1month | 3month | 6month | 12month – The time period over which to retrieve top artists for.
limit (Optional) : The number of results to fetch per page. Defaults to 50.
page (Optional) : The page number to fetch. Defaults to first page.
api_key (Required) : A Last.fm API key.

Sample Call for JSON:

http://ws.audioscrobbler.com/2.0/method=user.gettopartists&user=jacktrades&format=json&api_key=…

Example: Favourite artists location

XML last.fm

Find a sample result here:
http://polleres.net/20140826xsparql_st.etienne/xsparql/lastfm_user_sample.json


```
{ "topartists": {
    "@attr": { "total": "767",
      "user": "jacktrades"},
    "artist": [
      { "@attr": { "rank": "1" },
        "image": [ { "#text": "http://userserve-ak.last.fm/serve/34/84310519.png", "size": "small" },
          { "#text": "http://userserve-ak.last.fm/serve/64/84310519.png","size": "medium" },
          { "#text": "http://userserve-ak.last.fm/serve/126/84310519.png", "size": "large" } ],
        "mbid": "00a9f935-ba93-4fc8-a33a-993abe9c936b",
        "name": "Nightwish", "playcount": "4958", "streamable": "0", "url": "http://www.last.fm/music/Nightwish"},
      { "@attr": { "rank": "2" },
        "image": [ { "#text": "http://userserve-ak.last.fm/serve/34/2202944.jpg", "size": "small" },
          { "#text": "http://userserve-ak.last.fm/serve/64/2202944.jpg", "size": "medium" },
          { "#text": "http://userserve-ak.last.fm/serve/126/2202944.jpg", "size": "large" } ],
        "mbid": "c6b0db5a-d750-4ed8-9caa-ddcfb75dcb0a",
        "name": "Therion", "playcount": "4947", "streamable": "0", "url": "http://www.last.fm/music/Therion },
      ...
    ]
  }
}
```

Getting back to our goal: How to query that data?

XPath & Xquery in a nutshell…

Querying XML Data from Last.fm: XPath & XQuery 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 = 2]` Selects the “artist” with rank 2

Note: each XPath query is an XQuery... You can execute this:

```
java -cp /Users/apollere/software/SaxonHE9-5-1-7J/saxon9he.jar net.sf.saxon.Query -q:query2.xq -
s:lastfm_user_sample.xml
```

Querying XML Data from Last.fm: XPath & XQuery 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” looks something like this…

Query:

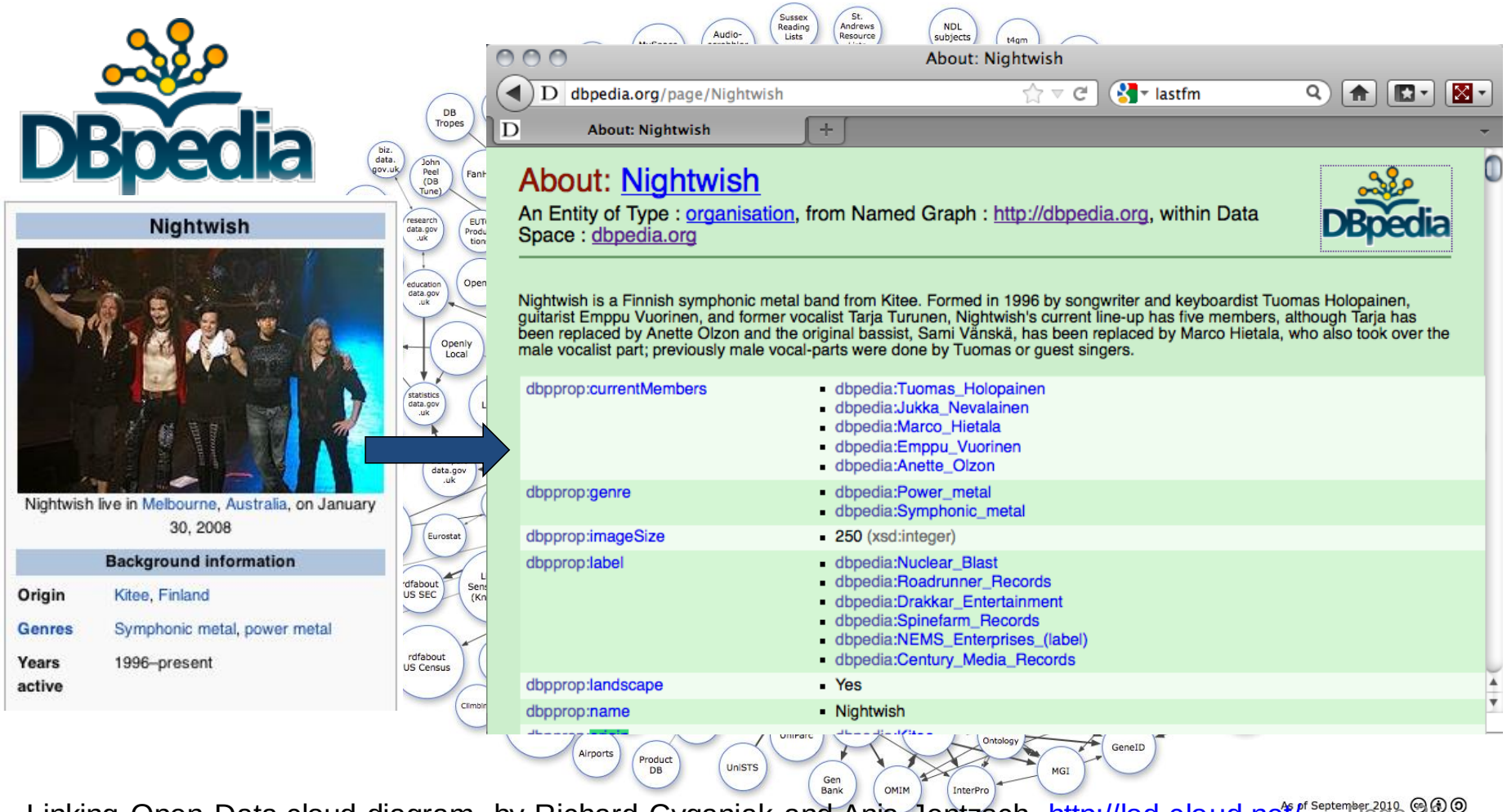
Retrieve information regarding a users' 2nd top artists from the

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:*** *"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?*

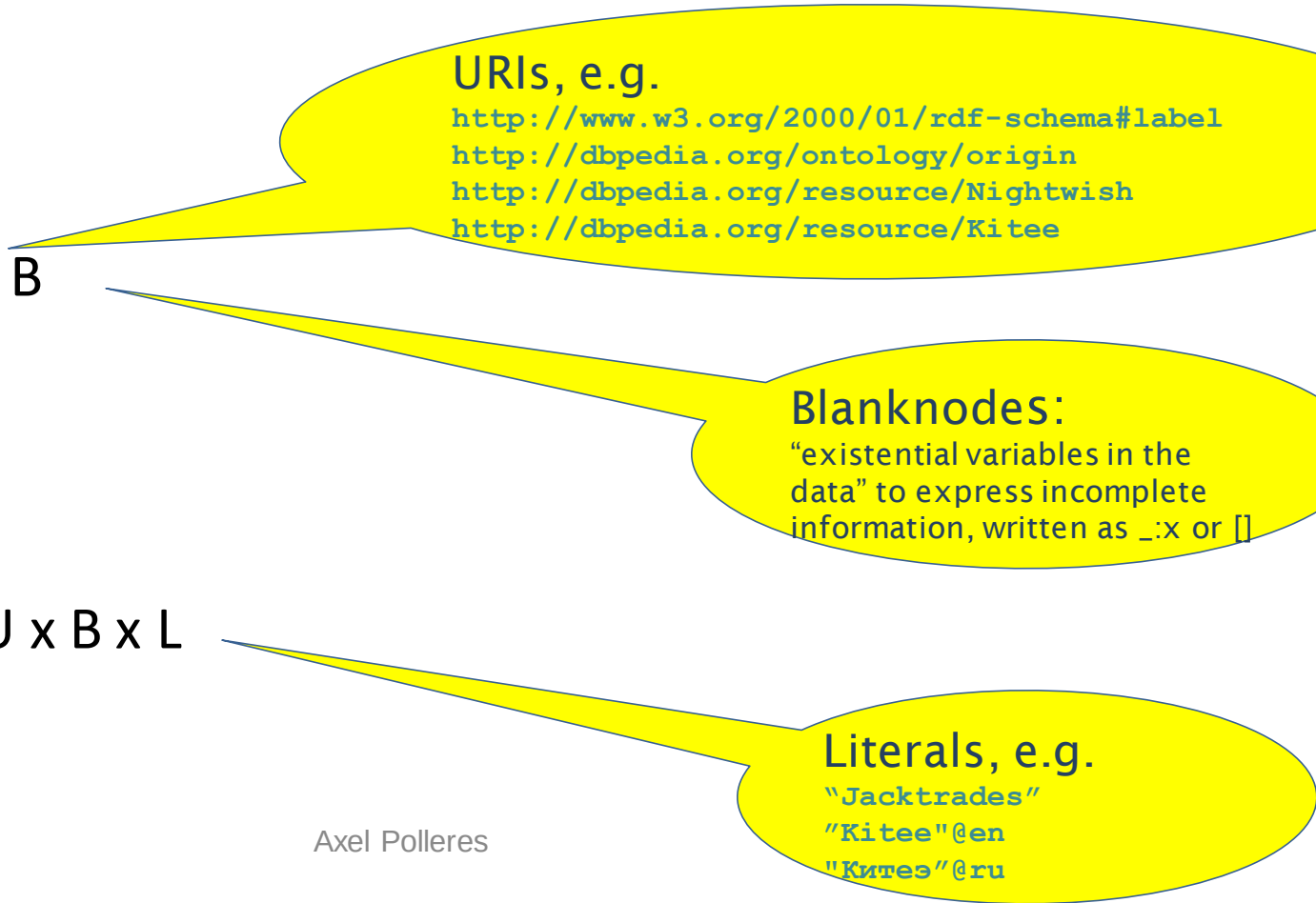
```
<table>
   <tr>
      <th colspan="2">Background information</th>
   </tr>
   <tr>
      <th>Origin</th>
      <td>
         <a title="Kitee" href="/wiki/Kitee">Kitee</a>, <a title="Finland" href="/wiki/Finland">Finland</a>
      </td>
   </tr>
   <tr>
      <th>
         <a title="Music genre" href="/wiki/Music_genre">Genres</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>
```

What's the advantages of RDF against XML (and JSON)?

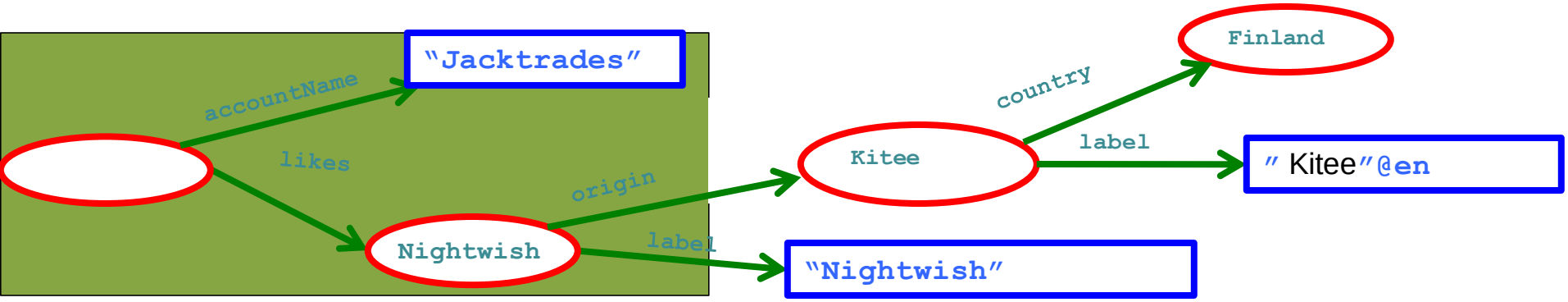
- Simple, declarative, graph-style format
- based on dereferenceable URIs (= Linked Data)

Subject Predicate Object



What's the advantages of RDF against XML (and JSON)?

- **Easily combinable! RDF data can simply be merged!**

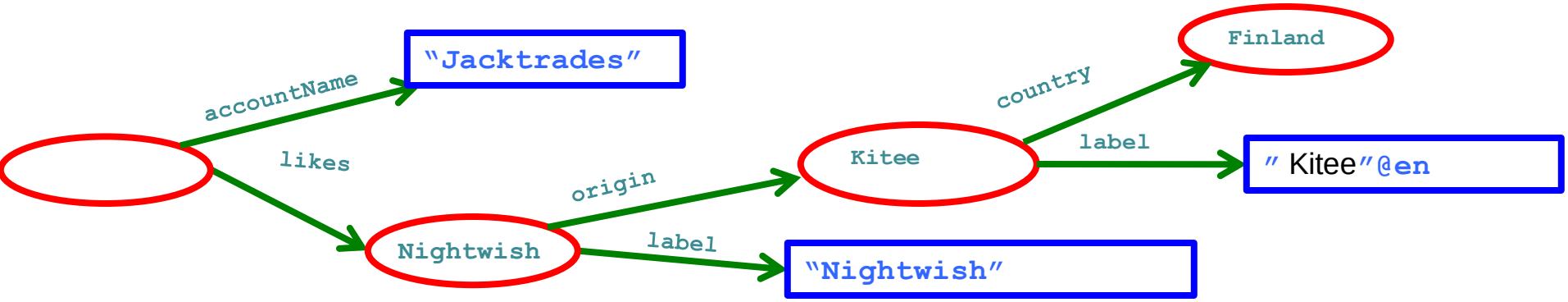


Various syntaxes, RDF/XML, Turtle, N3, RDFa,...

```
<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> .
```

Could be stored more or less straightforwardly (and naively ;)) stored into a relational DB!



- Query: Bands from Finland that user "Jacktrades" likes?

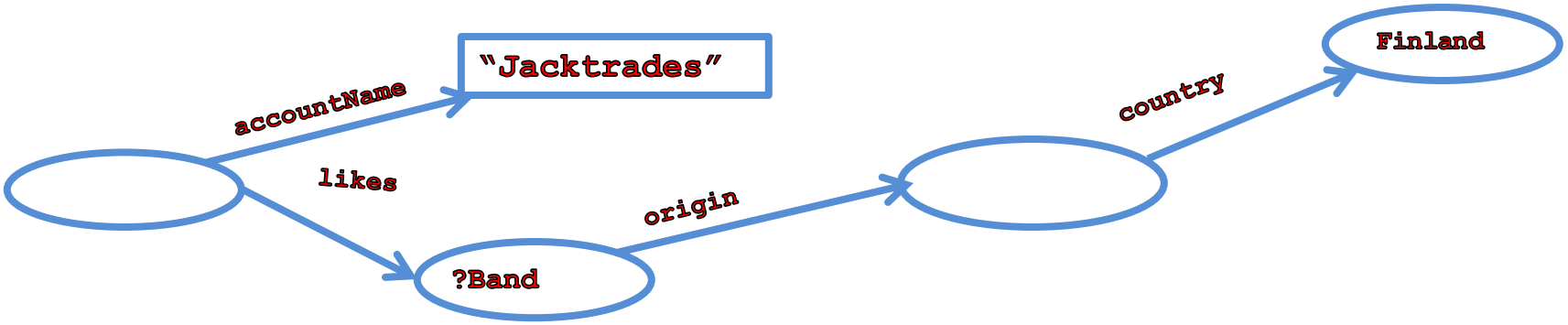
RDF Store

Subj	Pred	Obj
_:b	accountname	"Jacktrades"
_:b	likes	Nightwish
Nightwish	origin	Kitee
Nightwish	Label	"Nightwish"
Kitee	Country	Finland
…	…	…

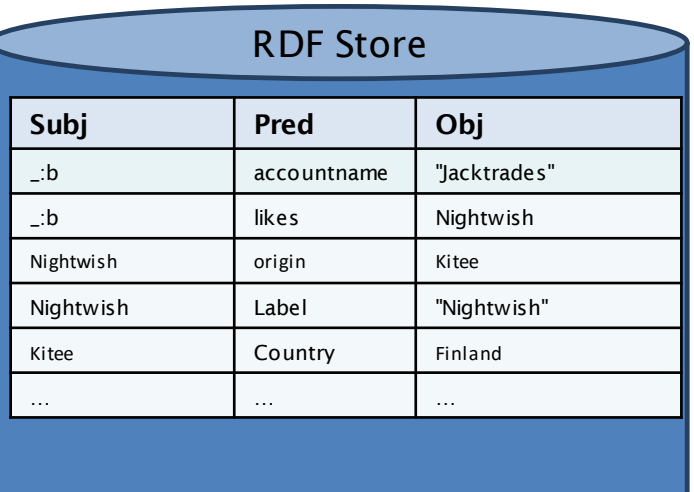
```
SELECT T2.Obj
FROM triples T1, triples T2, triples T3, triples T4
WHERE
 T1.Obj = "Jacktrades" AND T1.Pred = accountname AND
 T1.Subj = T2.Subj AND T2.Pred = likes AND
 T3.Obj = T4.Subj AND T3.Pred = origin AND
 T4.Pred = country AND T4.Obj = Finland
```

SQL is not the best solution for this… Fortunately there's something better: SPARQL!

Could be stored more or less straightforwardly (and naively ;)) stored into a relational DB!



- Query: Bands from Finland that user "Jacktrades" likes?



Subj	Pred	Obj
_:b	accountname	"Jacktrades"
_:b	likes	Nightwish
Nightwish	origin	Kitee
Nightwish	Label	"Nightwish"
Kitee	Country	Finland
…	…	…

SPARQL core Idea: formulate queries as **graph patterns**

…where basic graph patterns are just "Turtle with variables":

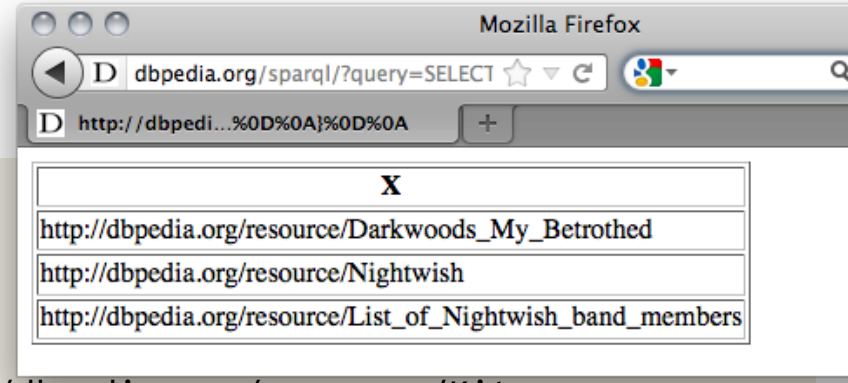
```
SELECT ?Band
WHERE { ?Band :origin [ :country :Finland ] .
        [] :accountName "jacktrades" ;
           :likes ?Band . }
```

How to query RDF?

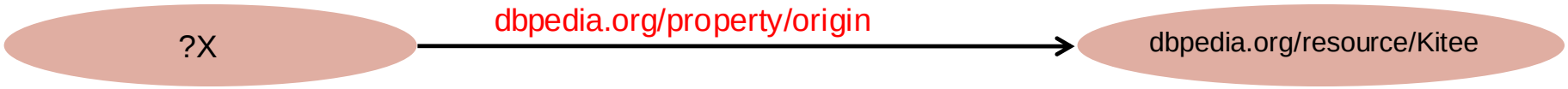
SPARQL in a Nutshell…

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>
}
```



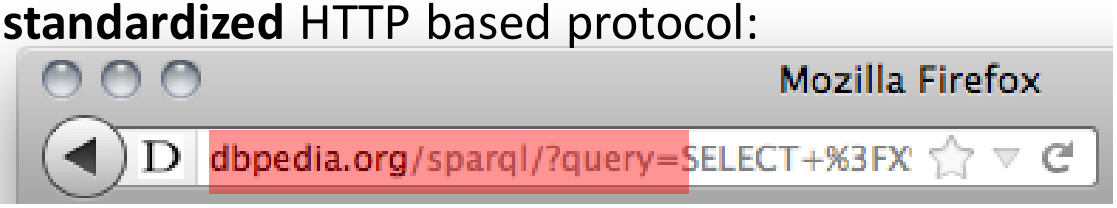
- *Try it out at http://live.dbpedia.org/sparql/*

SPARQL – Standard RDF Query Language and Protocol

- SPARQL 1.0 (standard since 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:

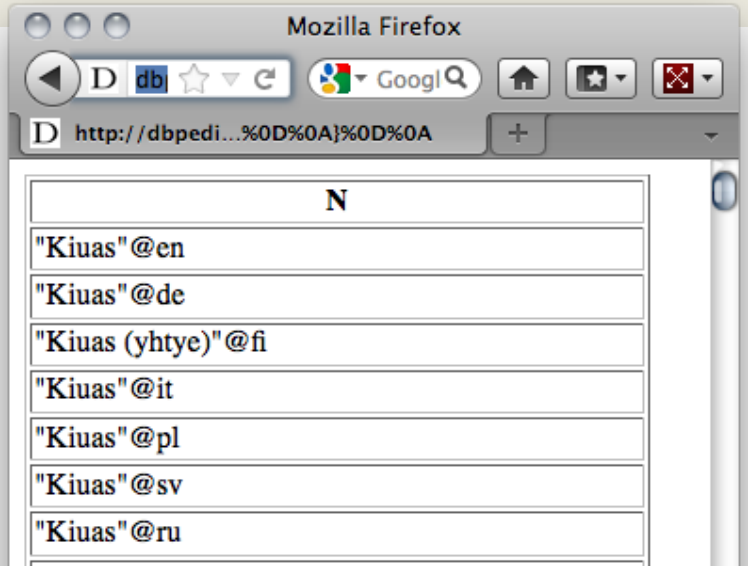


Conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and filters (FILTER)

***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] .
}
```

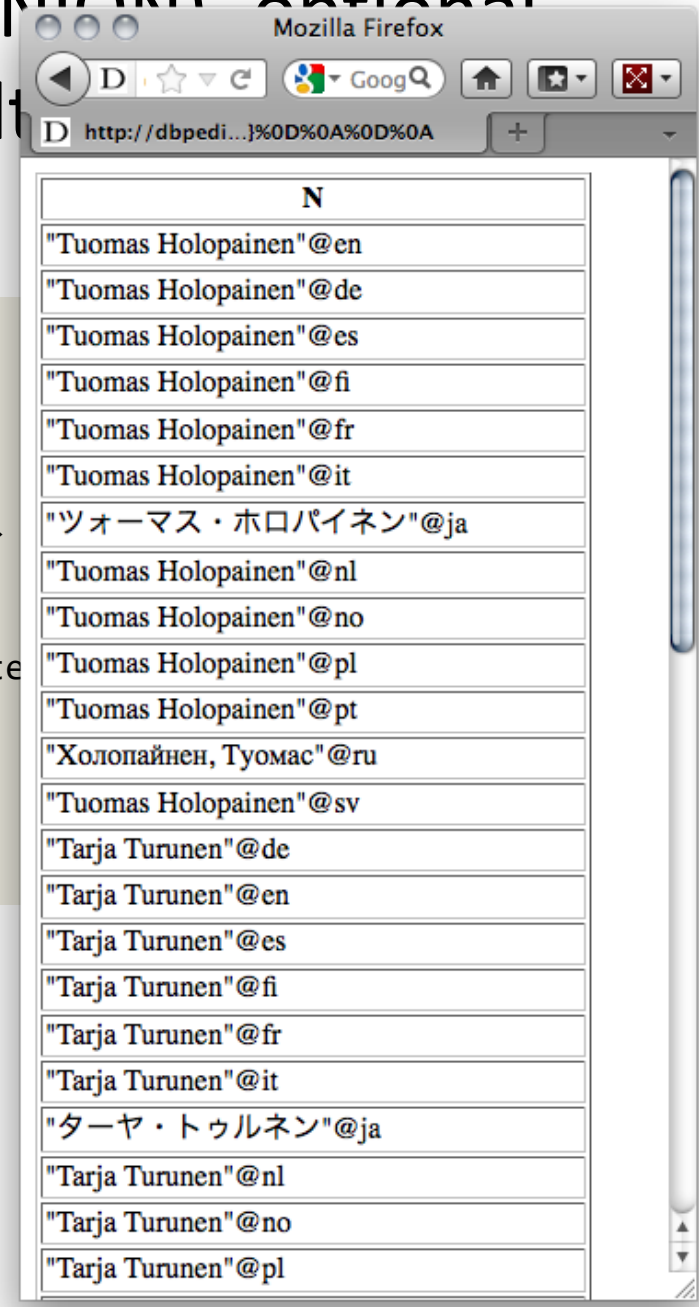
- *Shortcuts for namespace prefixes and to group triple patterns*



Conjunction (.) , disjunction (UNION), optional (OPTIONAL) patterns and filt

*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 German (@de) name…

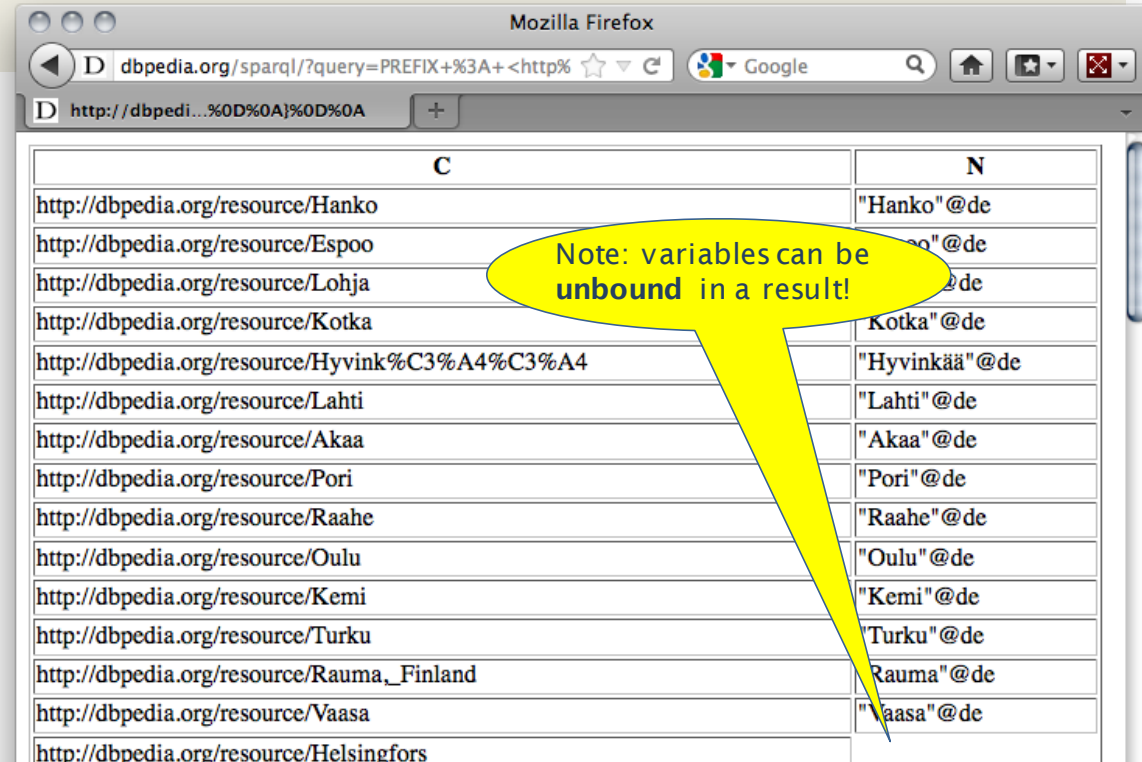
```
SELECT ?C ?N
WHERE
{
 ?C  dcterms:subject category:Cities_and_towns_in_Finland ;
     rdfs:label ?N .
 FILTER( LANG(?N) = "de" )

}
```

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" ) }
}
```

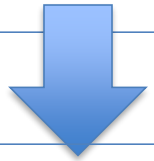


C	N
http://dbpedia.org/resource/Hanko	"Hanko"@de
http://dbpedia.org/resource/Espoo	"@de
http://dbpedia.org/resource/Lohja	@de
http://dbpedia.org/resource/Kotka	Kotka"@de
http://dbpedia.org/resource/Hyvink%C3%A4%C3%A4	"Hyvinkää"@de
http://dbpedia.org/resource/Lahti	"Lahti"@de
http://dbpedia.org/resource/Akaa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

CONSTRUCT Queries to create new triples (or to transform one RDF Graph to another)

- *The members of a Band know each other:*

```
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX prop: <http://dbpedia.org/property/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?M1 foaf:knows ?M2 }
WHERE { <http://dbpedia.org/resource/Nightwish> <http://dbpedia.org/ontology/bandMember> ?M1, ?M2
  FILTER( ?M1 != ?M2 ) }
```



```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix dbpedia:    <http://dbpedia.org/resource/> .

dbpedia:Jukka_Nevalainen foaf:knows dbpedia:Emppu_Vuorinen , dbpedia:Troy_Donockley ,
dbpedia:Floor_Jansen , dbpedia:Marco_Hietala , dbpedia:Tuomas_Holopainen .
dbpedia:Emppu_Vuorinen foaf:knows dbpedia:Jukka_Nevalainen , dbpedia:Troy_Donockley ,
dbpedia:Floor_Jansen , dbpedia:Marco_Hietala , dbpedia:Tuomas_Holopainen .
dbpedia:Troy_Donockley foaf:knows dbpedia:Jukka_Nevalainen , dbpedia:Emppu_Vuorinen ,
dbpedia:Floor_Jansen , dbpedia:Marco_Hietala , dbpedia:Tuomas_Holopainen .
dbpedia:Floor_Jansen foaf:knows dbpedia:Jukka_Nevalainen , dbpedia:Emppu_Vuorinen ,
dbpedia:Troy_Donockley , dbpedia:Marco_Hietala , dbpedia:Tuomas_Holopainen .
dbpedia:Marco_Hietala foaf:knows dbpedia:Jukka_Nevalainen , dbpedia:Emppu_Vuorinen ,
dbpedia:Troy_Donockley , dbpedia:Floor_Jansen , dbpedia:Tuomas_Holopainen .
dbpedia:Tuomas_Holopainen foaf:knows dbpedia:Jukka_Nevalainen , dbpedia:Emppu_Vuorinen ,
dbpedia:Troy_Donockley , dbpedia:Floor_Jansen , dbpedia:Marco_Hietala .
```

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 : <http://www.example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT (strbefore(?Name," ") AS ?firstname)
       (strafter(?Name," ") AS ?lastname)
WHERE  { ?X foaf:name ?Name . }
```

Data:

```
:klaus foaf:knows :karl ;
       foaf:nickname "Niki".
:alice foaf:knows :bob , :karl ;
       foaf:name "Alice Wonderland" .
:karl foaf:name "Karl Mustermann" ;
      foaf:knows :joan.
:bob foaf:name "Robert Mustermann" ;
     foaf:nickname "Bobby" .
```

Results:

?firstname	?lastname
Alice	Wonderland
Karl	Mustermann
Bob	Mustermann

2. Assignment/Project Expressions

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

```
PREFIX : <http://www.example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?firstname ?lastname
WHERE  { ?X foaf:name ?Name .
         BIND (strbefore(?Name," ") AS ?firstname)
         BIND (strafter(?Name," ") AS ?lastname) }
```

Data:

```
:klaus foaf:knows :karl ;
       foaf:nickname "Niki".
:alice foaf:knows :bob , :karl ;
       foaf:name "Alice Wonderland" .
:karl foaf:name "Karl Mustermann" ;
      foaf:knows :joan.
:bob foaf:name "Robert Mustermann" ;
  44 foaf:nickname "Bobby" .
```

Results:

?firstname	?lastname
Alice	Wonderland
Karl	Mustermann
Bob	Mustermann

3. Aggregates

- *"How many different names exist?"*

```
PREFIX ex: <http://example.org/>
SELECT (Count(DISTINCT ?Name) as ?NamesCnt)
WHERE { ?P foaf:name ?Name }
```

Data:

```
:klaus foaf:knows :karl ;
       foaf:nickname "Niki".
:alice foaf:knows :bob , :karl ;
       foaf:name "Alice Wonderland" .
:karl foaf:name "Karl Mustermann" ;
      foaf:knows :joan.
:bob foaf:name "Robert Mustermann" ;
     foaf:nickname "Bobby" .
```

Result:

? NamesCnt
3

3. Aggregates

- *"How many people share the same lastname?"*

```
SELECT ?lastname (count(?lastname) AS ?count)
WHERE {
    ?X foaf:name ?Name .
    BIND (strbefore(?Name," ") AS ?firstname)
    BIND (strafter(?Name," ") AS ?lastname)
}
GROUP BY ?lastname
```

Data:

```
:klaus foaf:knows :karl ;
       foaf:nickname "Niki".
:alice foaf:knows :bob , :karl ;
       foaf:name "Alice Wonderland" .
:karl foaf:name "Karl Mustermann" ;
      foaf:knows :joan.
:bob foaf:name "Robert Mustermann" ;
     foaf:nickname "Bobby" .
```

Result:

?lastname	?count
"Mustermann"	2
"Wonderland"	1

3. Aggregates

- *"How many people share the same lastname?"*

```
SELECT ?lastname (count(?lastname) AS ?count)
WHERE {
    ?X foaf:name ?Name .
    BIND (strbefore(?Name," ") AS ?firstname)
    BIND (strafter(?Name," ") AS ?lastname)
}
GROUP BY ?lastname
HAVING (?count > 1 )
```

Data:

```
:klaus foaf:knows :karl ;
       foaf:nickname "Niki".
:alice foaf:knows :bob , :karl ;
       foaf:name "Alice Wonderland" .
:karl foaf:name "Karl Mustermann" ;
      foaf:knows :joan.
:bob foaf:name "Robert Mustermann" ;
     foaf:nickname "Bobby" .
```

Result:

?lastname	?count
"Mustermann"	2

4. Subqueries

- *How to create new triples that concatenate first name and last name?*

```
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?

```
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. *transitive closure of foaf:knows:*

```
SELECT *
WHERE {
   ?X foaf:knows* ?Y .
}
```

- if 0-length paths should not be considered, use '+':

```
SELECT *
WHERE {
   ?X foaf:knows+ ?Y .
}
```

5. Property Path expressions

- Arbitrary Length paths, Concatenate property paths, etc.
- E.g. *Implement RDFS reasoning: All employees (using rdfs:subClassOf reasoning) that alice knows (using rdfs:subPropertyOf reasoning)?*

```
PREFIX : <http://www.example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT *
    WHERE {
        :alice ?P ?X .
        ?P rdfs:subPropertyOf* foaf:knows .
        ?X rdf:type/rdfs:subClassOf* :Employee .
    }
```

- For details on the limits of this approach, cf.
  S. Bischof, M. Krötzsch, A. Polleres, S. Rudolph. Schema-agnostic query rewriting in SPARQL 1.1. ISWC2014

Small detail: We found out that the DBPedia "ontology" is inconsistent: every library is inferred to belong to the mutually disjoint classes “Place” and “Agent”…. Cf. http://stefanbischof.at/publications/iswc14/

Hands-on?
If you want to try this out:

- http://www.polleres.net/20140826xsparql_st.etienne/sparql/

- Quick run through some very simple example queries to recap the concepts... SPARQL_simple_step-by-step/

- Sample queries to a remote SPARQL endpoint ... http://live.dbpedia.org/sparql
  - Sample Queries: SPARQL_dbpedia_various_examples/

Hands-on?

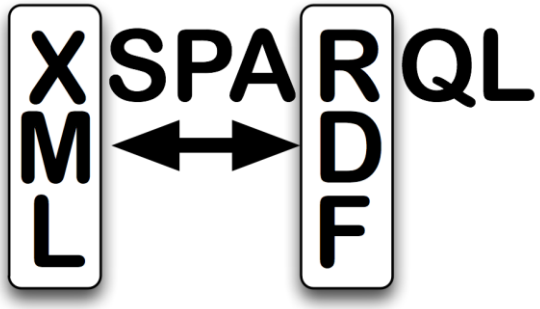
- Let's first quickly run through some very simple example queries to recap the concepts...

- DBPedia SPARQL endpoint ...
- http://live.dbpedia.org/sparql

- E.g.: Bands that origin in Vienna and their members?
  - How do we proceed building such a query?
  - What can we observe on the result?

XSPARQL

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



- Transformation language
- Consume and generate XML and RDF
- Syntactic extension of XQuery, ie.

  XSPARQL = XQuery + SPARQL

Recall: XQuery 2/2

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*

Example 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>
```

XSPARQL: Syntax overview (I)

Prefix declarations

Body:

Data Input (XML or RDF)

Data Output (XML or 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*	

XSPARQL Syntax overview (II)

XQuery or SPARQL prefix declarations

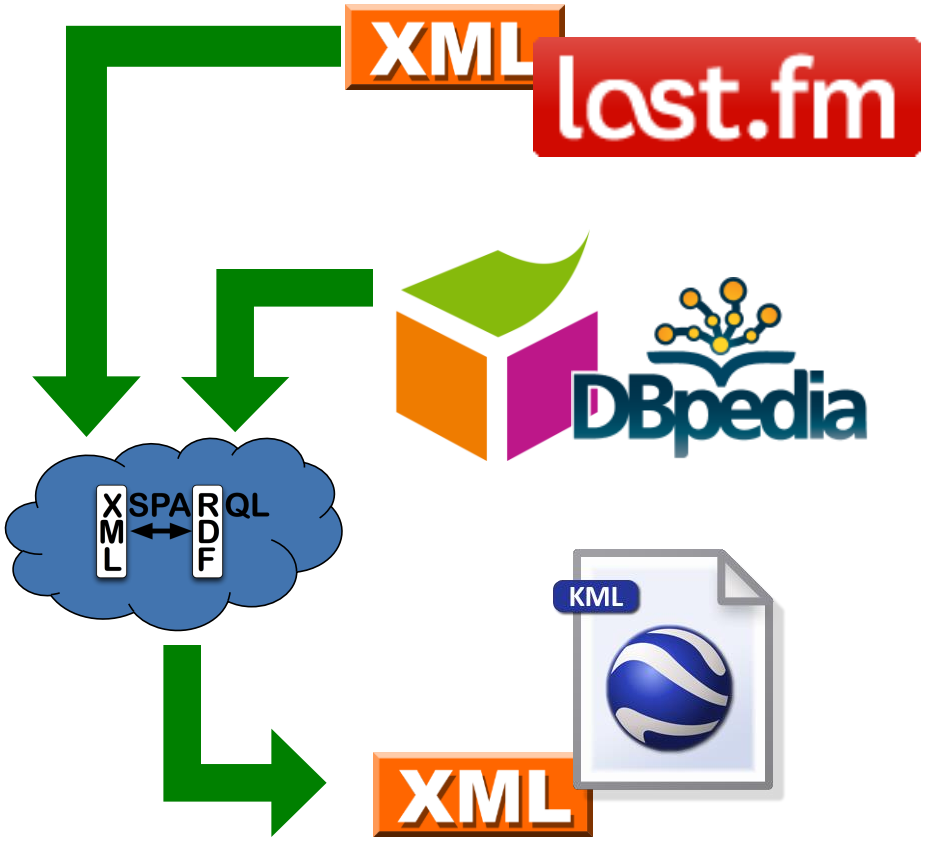
Any XQuery query

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*	

Back to our original use case



XSPARQL: Convert XML to RDF

Query:

Convert Last.fm top artists of a user into RDF

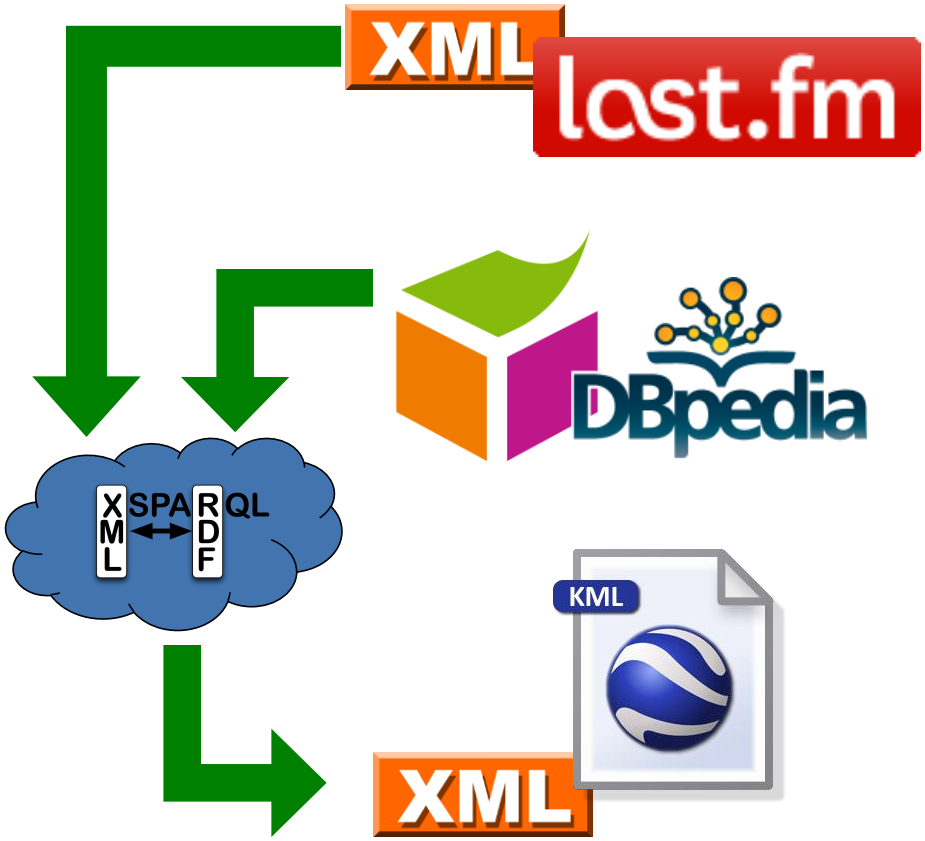
```
prefix lastfm: <http://xsparql.deri.org/lastfm#>

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

Back to our original 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

GeoNames

XSPARQL: Integrate RDF sources

Query:

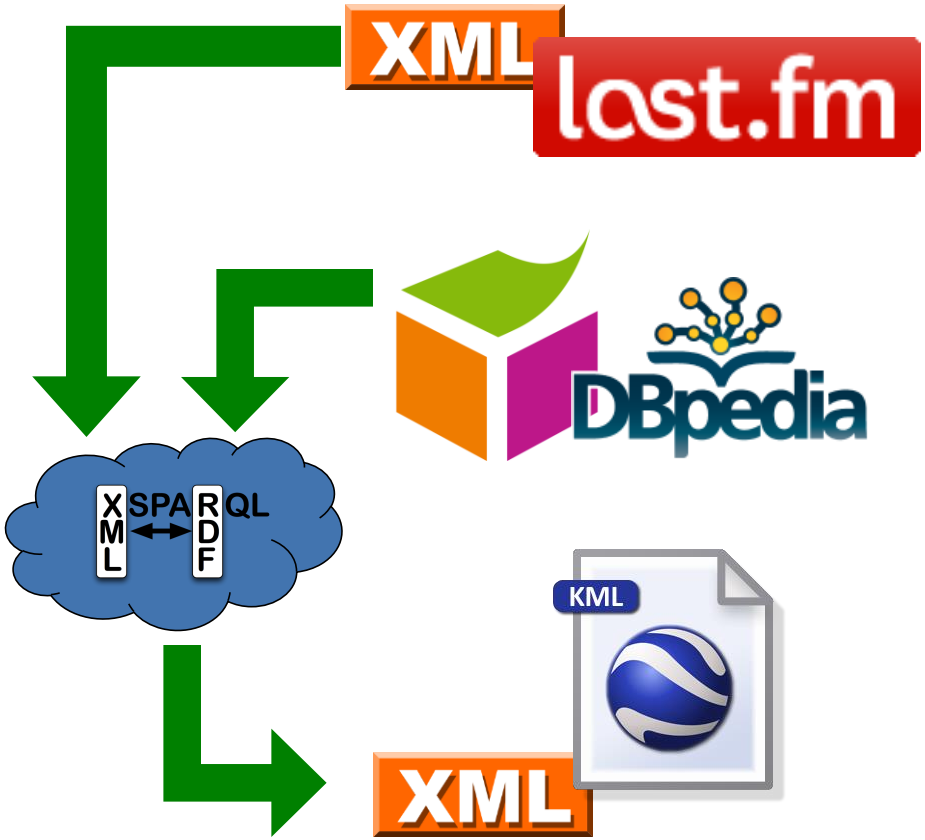
Retrieve the origin of an artist from DBPedia ***including map coordinates***

```
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:

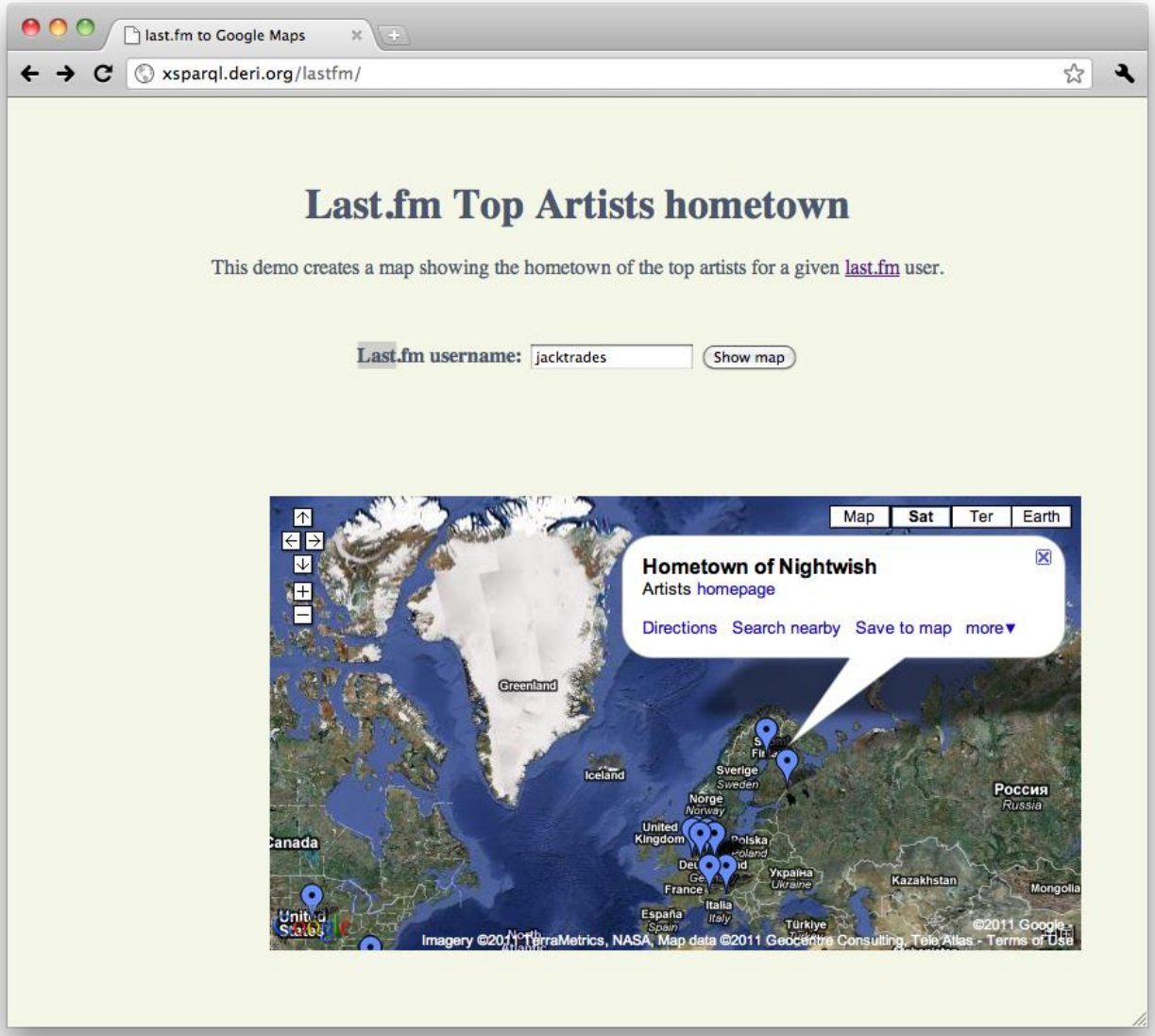
- root element: “kml”, then “Document”
- sequence of “Placemark”
- Each “Placemark” contains:
  - “Name” element
  - “Point” element with the “coordinates”

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



Last, but not least: Consuming JSON with XSPARQL:

- XSPARQL can handle JSON by transforming it to a canonical XML format using the custom XSPARQL function:

`xsparql:json-doc( URI-to-json-file )`

- Example: return names of bands user jacktrades likes fromlastfm (json):

```
declare namespace rdfs="http://www.w3.org/2000/01/rdf-schema#";

for $m in
xsparql:json-doc("http://polleres.net/20140826xsparql_st.etienne/xsparql/lastfm_user_sample.json")//artist
return $m//name
```

Producing Json with XSPARQL

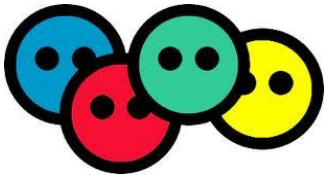
- No syntactic sugar specifically for that, but can be done with "onboard" means of Xquery and some special functions of XSPARQL

```
xsparql:isBlank()
xsparql:isIRI()
xsparql:isLiteral()
```

- Example: convert RDF to JSON-LD:
- http:www.polleres.net/20140826xsparql_st.etienne/xsparql/query1_json-ld.xsparql

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 RDF files

Obtaining locations in RDF

- Update or enhance your RDF 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 allows that, but more verbose (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 ?B 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… This has to do with the compositionality of SPARQL

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 *
where{ service <http://live.dbpedia.org/sparql>{[ dbprop:birthDate $B; foaf:name $N ].
         filter ( regex(str($B),str($MyB)) ) } }
construct { :me :sameBirthDayAs $N }
```

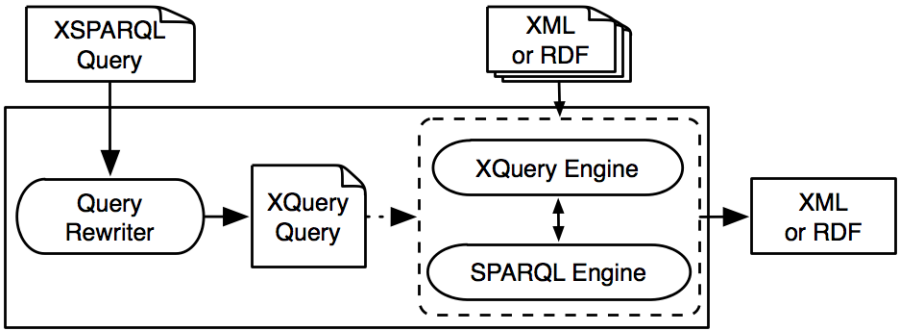
You can use SERVICE from SPARQL1.1 in a for loop!

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

What's missing?

- No full control flow:
  - XQuery/XSPARQL e.g. don't allow you to specify politeness (e.g. crawl delays between doc() calls.
- Only doc() function, but no custom HTTP request
  - E.g. PUT,POST ...
  - Some Xquery implementations have additional built-in functions for that (e.g. MarkLogic)
- Bottomline:
  - For many practical use cases you'll still be ending up doing scripting, but declarative Query languages help you to get the necessary data for these scripts!
  - **And: it‘s extensible! Would be happy to talk to interested students to extend our current prototype!**

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

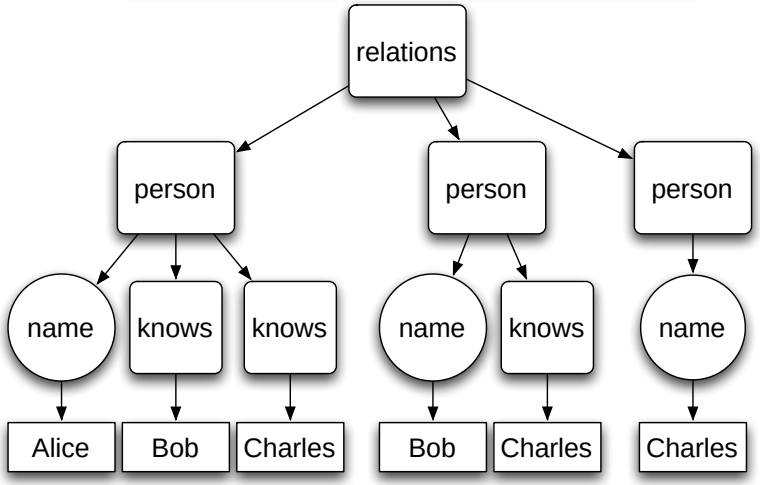
```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
     foaf:name "Alice";
     foaf:knows _:b2;
     foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```

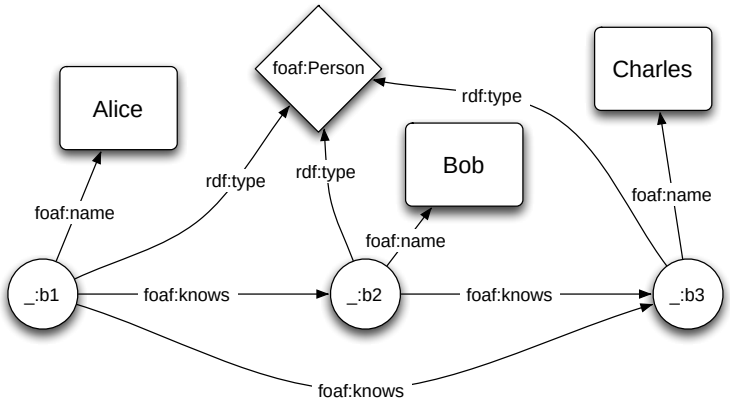
Lowering

relations.xml

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

Lifting





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 $aux_resu
                    fn:doc($aux_query)//sparql_result:result
let $P_Node :=  $aux_result/sparql_result:binding[@name="P"]
let $N_Node :=  $aux_result/sparql_result: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)

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

http://xsparql.sourceforge.net/

- 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

- Adding JSON support and SPARQL1.1 features:

Daniele Dell'Aglio, Axel Polleres, Nuno Lopes, and Stefan Bischof. Querying the web of data with XSPARQL 1.1. In *ISWC2014 Developers Workshop*, volume 1268 of *CEUR Workshop Proceedings*. CEUR-WS.org, October 2014.
http://www.polleres.net/publications/dell-etal-2014iswc-dev.pdf

- Demo/Hand-on: Some more XSPARQL examples
- https://ai.wu.ac.at/~polleres/20140826xsparql_st.etienne/xsparql/


"That's all Folks!"

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

- We're always looking for interested students for internships or to work on various exciting projects with partners in industry and public administration that involve:
- Solving Data Integration tasks using (X)SPARQL
- Querying Linked Data and Open Data
  - Integrating Open Data and making it available as Linked Data
  - Linked and Open Data Quality
- Foundations and extensions of SPARQL
  - Extending XSPARQL
  - SPARQL and Entailments, etc.

eres