Open Information Systems 2019-2020

Lecture 5: Querying with SPARQL

Christophe Debruyne

Recap

- Representing knowledge on the Web
  - XML and XML Schema
  - RDF
  - RDF(S)
  - OWL
- Important!
  - RDF/XML is merely a serialization of RDF, many others exist, such as N3, Turtle, etc.

SPARQL

- Stands for *SPARQL Protocol and RDF Query Language*
  - Recursive acronym
- SPARQL 1.1 is a W3C Recommendation since March 2013 (http://www.w3.org/TR/sparql11-overview/)
- SPARQL allows us to:
  - Pull values from structured and semi-structured data
  - Explore data by querying unknown relationships
  - Perform complex joins of disparate databases in a single, simple query
  - Transform RDF data from one vocabulary to another

Examples

- In this lecture, quite a few examples were adopted from
  - Feigenbaum and Prud'hommeaux (SPARQL by example)
  - Chapter 7 of Foundations of Semantic Web Technologies.

Structure of a SPARQL Query

A SPARQL query comprises, in order:

- **Prefix declarations**, for abbreviating URIs.
- A **result clause**, identifying what information to return from the query
- **Dataset definition**, stating what RDF graph(s) are being queried
- The **query pattern**, specifying what to query for in the underlying dataset
- **Query modifiers**, slicing, ordering, and otherwise rearranging query results

```
# prefix declarations
PREFIX foaf:
<http://xmlns.com/foaf/0.1/>
...
# result clause
SELECT ...
# dataset definition
FROM ...
# query pattern
WHERE {
     ...
}
# query modifiers
ORDER BY ...
```

SPARQL Architecture and Endpoints

- SPARQL queries are executed against **RDF datasets**
  - Consisting of one or more RDF graphs.
- A **SPARQL endpoint** accepts queries and returns results via HTTP.
- The results returned/rendered in a variety of formats:
  - SPARQL specifies an **XML** vocabulary for result sets.
  - A JSON "port" of the XML vocabulary (useful for RIA).
  - Certain SPARQL result clauses trigger **RDF** responses.
  - **HTML** often as an XSLT transformation of the XML.
  - **CSV**.

SPARQL

SPARQL 1.0

- January 2008
- SPARQL 1.0 Query Language
- SPARQL 1.0 Protocol
- Results in XML
- Results in JSON

SPARQL 1.1

- March 2013
- Several updates
- Addition of SPARQL 1.1
  - Update (insert, del, modify)
  - Graph Store HTTP protocol
  - Service Descriptions
  - Entailments
  - Basic Federated Query
  - Results in CSV, TSV

Example

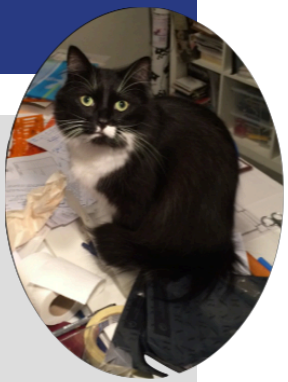
```
@base <http://foo.bar/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#Cat>    a rdfs:Class ; rdfs:label "Cat" .

<#owns>   a rdf:Property ;  rdfs:label "owns" ;
          rdfs:domain foaf:Person ;  rdfs:range <#Cat> .

<#victor>    a <#Cat> ; foaf:name "Victor" .
<#gaston>    a <#Cat> ; foaf:name "Gaston" .
<#bettina>   a <#Cat> ; foaf:name "Bettina" .

<#chrdebru> a foaf:Person ;    foaf:name "Christophe Debruyne" ;
          <#owns> <#victor> ; <#owns> <#gaston> ;
          <#owns> <#bettina> .
```



Simple SPARQL Queries

- Finding all cats and their names

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {  ?cat a ex:Cat.
         ?cat foaf:name ?name.   }
```

  - Variables start with a question mark and can match any term (resource or literal);
  - Triple patterns are just like triples, except that any of the parts of a triple can be replaced with a variable;
  - The SELECT result clause returns a table of variables and values that satisfy the query.
  - "a" is syntactic sugar for "rdf:type"

Simple SPARQL Queries

- Finding all cats and their names

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {  ?cat a ex:Cat ; foaf:name ?name. }
```

```
-----------------------------------------
| cat                          | name     |
=========================================
| <http://foo.bar/#bettina> | "Bettina" |
| <http://foo.bar/#gaston>  | "Gaston"  |
| <http://foo.bar/#victor>  | "Victor"  |
-----------------------------------------
```

Simple SPARQL Queries

- Use multiple triple patterns to retrieve multiple properties about a particular resource.
- **`SELECT *`** selects all variables mentioned in the query

Querying SPARQL Endpoints

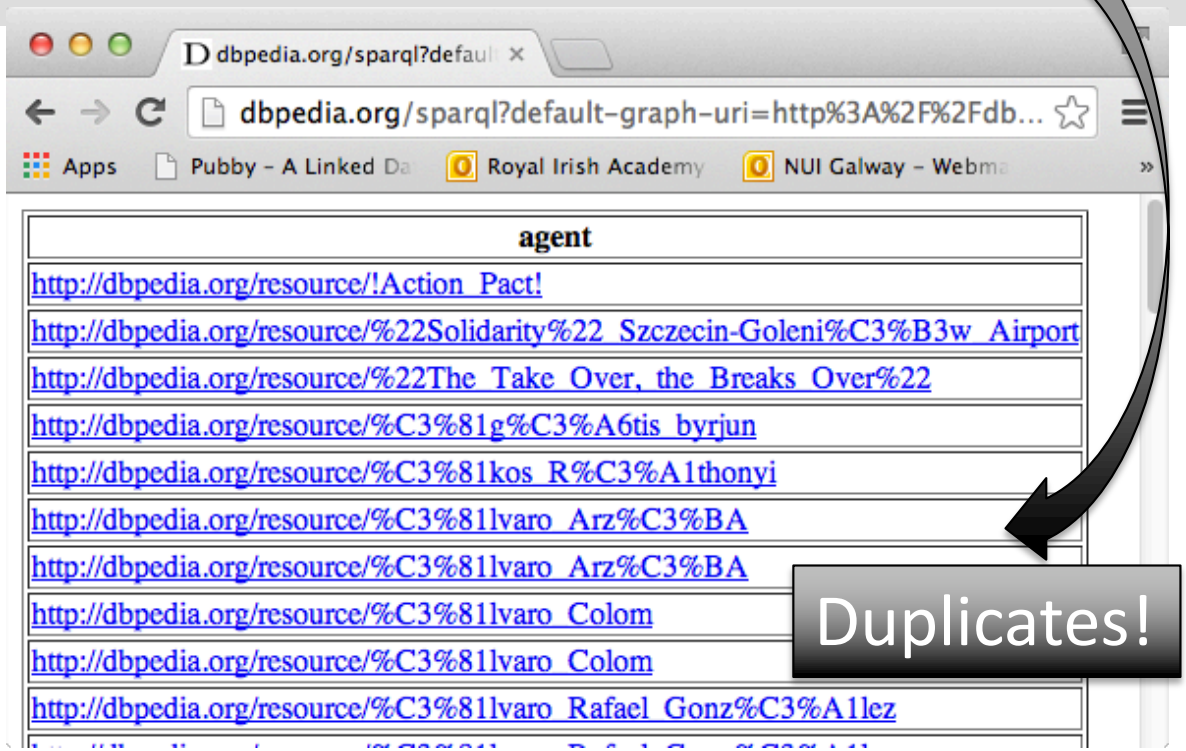
- DBpedia is an effort to “triplify” Wikipedia
- http://dbpedia.org/sparql
- “Find me 50 ‘things’ with names in DBPedia”

```
SELECT ?agent
WHERE {
     ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

Is there a problem with this query?

Querying SPARQL endpoints

```
SELECT ?agent
WHERE {
    ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```



Querying SPARQL endpoints

```
SELECT DISTINCT ?agent
WHERE {
    ?agent <http://xmlns.com/foaf/0.1/name> ?name .
} LIMIT 50
```

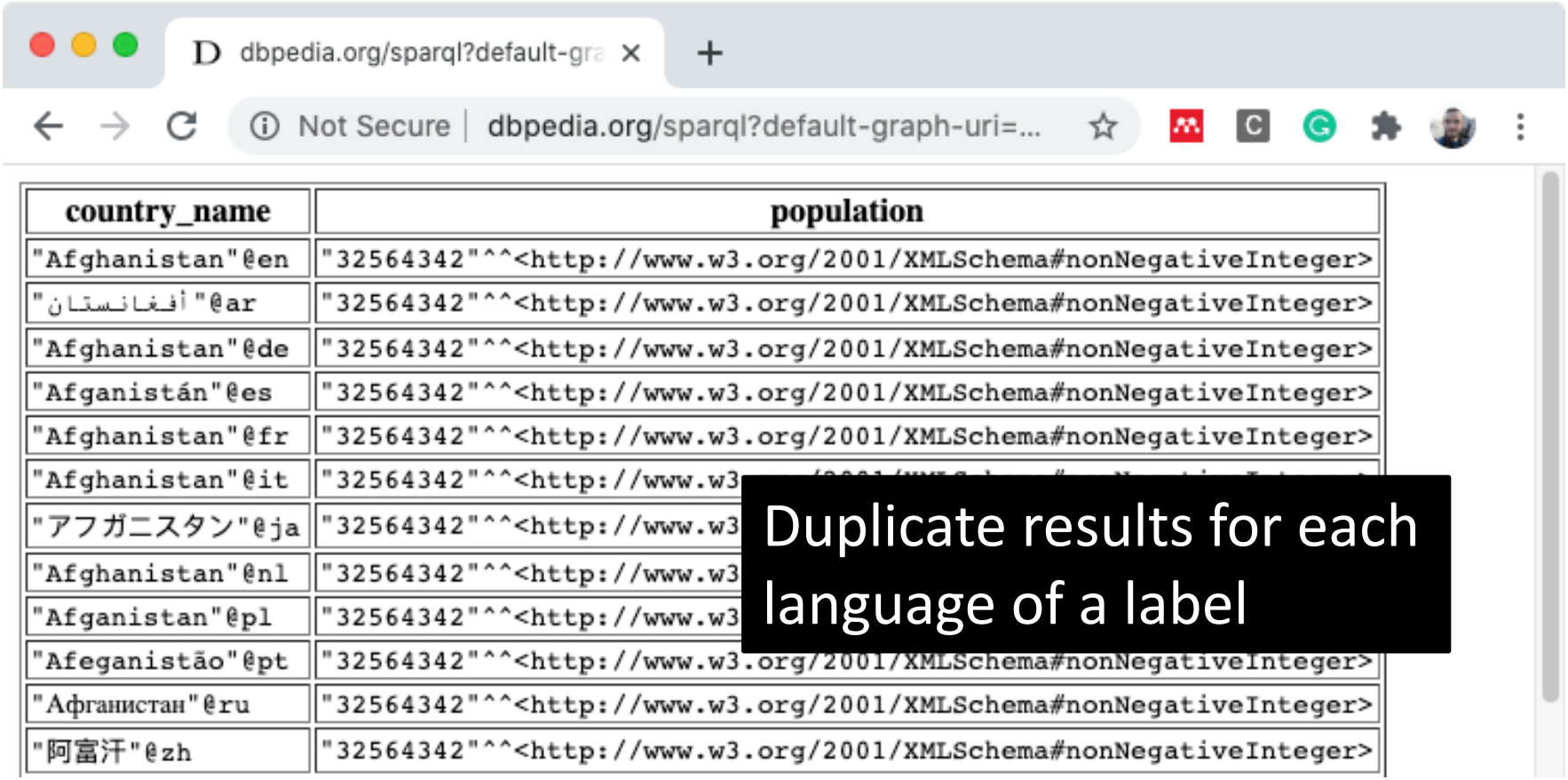
- Solution modifiers
  - **LIMIT** → Limit the number of rows returned
  - **ORDER BY** → Sorting
  - **OFFSET** → Used together with **LIMIT** and **ORDER BY** for *slicing*, e.g., for paging

Filtering

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?country_name ?population
WHERE {
    ?country a yago:WikicatLandlockedCountries ;
             rdfs:label ?country_name  ;
             dbo:populationTotal ?population .
    FILTER (?population > 15000000)
}
```

- **`FILTER`** constraints use boolean conditions to filter out unwanted query results.
- Shortcut: a semicolon (;) for abbreviating some triples.

Filtering



country_name	population
"Afghanistan"@en	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"أفغانستان"@ar	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Afghanistan"@de	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Afganistán"@es	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Afghanistan"@fr	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Afghanistan"@it	"32564342"^^<http://www.w3
"アフガニスタン"@ja	"32564342"^^<http://www.w3
"Afghanistan"@nl	"32564342"^^<http://www.w3
"Afganistan"@pl	"32564342"^^<http://www.w3
"Afeganistão"@pt	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Афганистан"@ru	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"阿富汗"@zh	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Duplicate results for each language of a label

Filtering

SPARQL 1.0	
Logical	!, &&, ||
Math	+, -, *, /
Comparison:	=, !=, >, <, IN, NOT IN, …
SPARQL tests	isURI, isBlank, isLiteral, isNumeric, bound
SPARQL accessors	str, lang, datatype
Other	sameTerm, langMatches, regex, REPLACE

Filtering

SPARQL 1.1	
Conditionals	IF, COALESCE, EXISTS, NOT EXISTS
Constructors	URI, BNODE, STRLANG, UUID, …
Strings	STRLEN, SUBSTR, UCASE, …
Extra math functions	abs, round, ceil, floor, RAND
Date and Time	now, year, month, day, hours, minutes, …
Hashing	MD5, SHA1, SHA256, SHA384, SHA512

Filtering

```
# Namespaces omitted
SELECT ?country_name ?population
WHERE {
    ?country a yago:WikicatLandlockedCountries ;
             rdfs:label ?country_name  ;
             dbo:populationTotal ?population .
    FILTER (?population > 15000000)
    FILTER (langMatches(lang(?country_name), "EN"))
}
```

country_name	population
"Afghanistan"@en	"32564342"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Burkina Faso"@en	"17322796"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Kazakhstan"@en	"17693500"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Malawi"@en	"16407000"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Uzbekistan"@en	"31576400"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Zambia"@en	"16212000"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Ethiopia"@en	"99465819"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Filtering

```
# Namespaces omitted
SELECT ?country_name ?population
WHERE {
    ?country a yago:WikicatLandlockedCountries ;
             rdfs:label ?country_name  ;
             dbo:populationTotal ?population .
    FILTER (?population > 15000000 &&
            langMatches(lang(?country_name), "EN"))
}
```

This query, however, assumes a couple of things:

- All landlocked countries are typed
- All landlocked countries have an rdfs:label with the English language tag
- All landlocked countries have a population that is numerical via dbo:populationTotal

Union and Optional

```
{
 { ?book ex:author ?a . }
 UNION
 { ?book ex:writer ?a . }
}
```

- **UNION** is useful to match alternatives. By combining two or more *graph patterns*. It is thus not restricted to triples patterns.

Union and Optional

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

```
SELECT ?title
WHERE  {
   { ?book dc10:title  ?title } UNION
   { ?book dc11:title  ?title }
}
```

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Union and Optional

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

```
SELECT ?x ?y
WHERE  {
  { ?book dc10:title ?x } UNION
  { ?book dc11:title ?y }
}
```

x	y
	"SPARQL (updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

Union and Optional

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox } }
```

- **OPTIONAL** tries to match a graph pattern, but doesn't fail the whole query if the optional match fails.
- If an **OPTIONAL** pattern fails to match for a particular solution, any variables in that pattern remain unbound for that solution.

Union and Optional

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:a   rdf:type       foaf:Person .
_:a   foaf:name      "Alice" .
_:a   foaf:mbox      <mailto:alice@example.com> .
_:a   foaf:mbox      <mailto:alice@work.example> .
_:b   rdf:type       foaf:Person .
_:b   foaf:name      "Bob" .
```

```
SELECT ?name ?mbox
WHERE  { ?x foaf:name ?name .
  OPTIONAL { ?x foaf:mbox ?mbox }
}
```

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Union and Optional

```
{
 ?book ex:published_by <http://foo/#bar>.
 { ?book ex:author ?a . }
 UNION
 { ?book ex:writer ?a . }
}
```

- Optional and Union are left associative.
- Curly braces improve “readability”, but cause overhead.

Watch out!

- Apart from bound, all functions and operators that operate on RDF will produce a type error if any arguments are unbound.
- Simple tests such as ?c != ‘M’ are not sufficient!
- We thus have a three-valued logic.

A	B	A V B	A ^ B	A	B	A V B	A ^ B
T	T	T	T	T	E	T	E
T	F	T	F	E	T	T	E
F	T	T	F	F	E	E	F
F	F	F	F	E	F	E	F
				E	E	E	F

Named Graphs

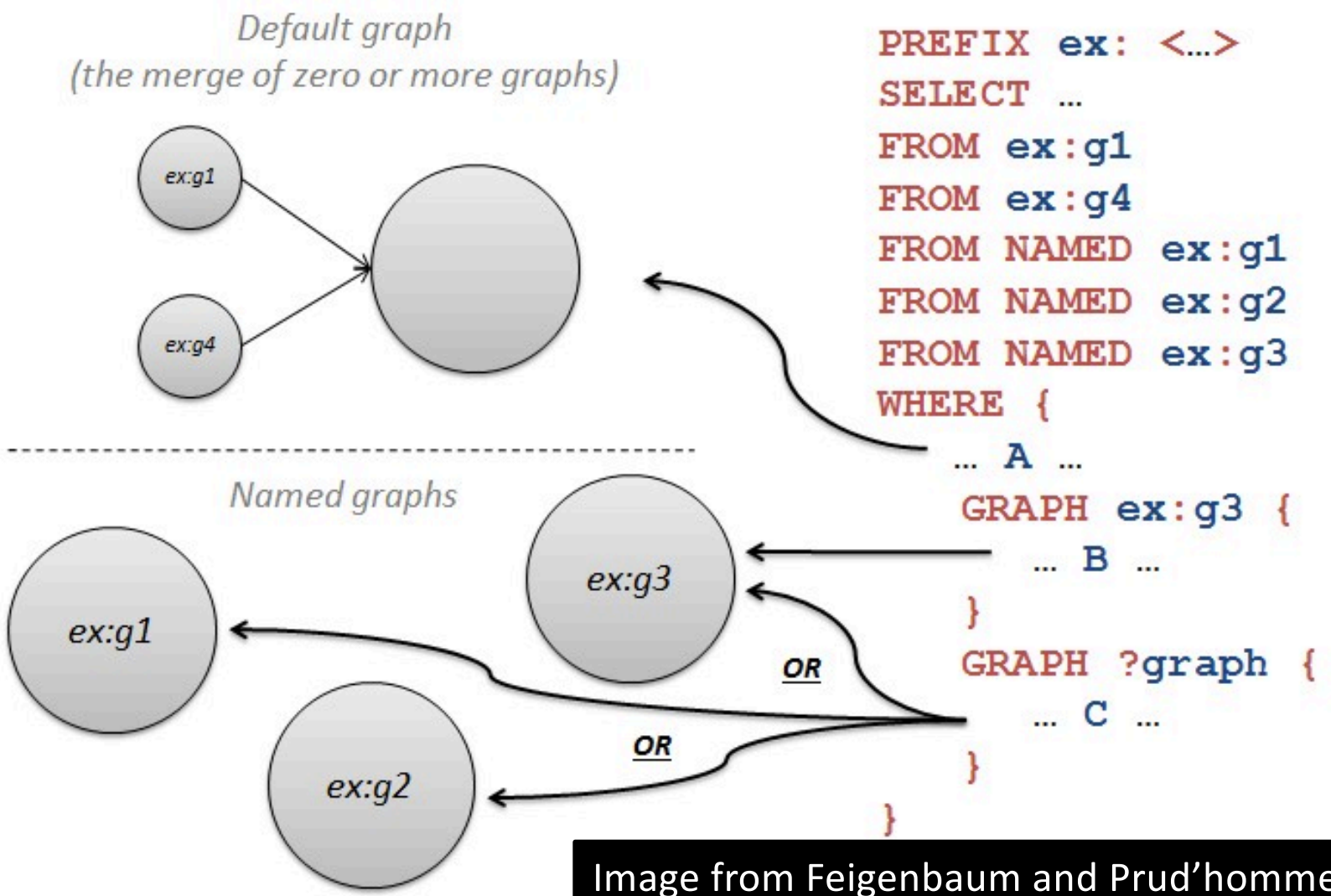
```
SELECT DISTINCT ?name
WHERE { ?person foaf:name ?name .
        GRAPH ?g1 { ?person a foaf:Person }
        GRAPH ?g2 { ?person a foaf:Person }
        GRAPH ?g3 { ?person a foaf:Person }
        FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3).}
```

- http://data.semanticweb.org/snorql/
  - Data set of ISWC and ESWC events, containing information about authors, etc.
  - Each graph represents a particular ISWC or ESWC conference
- Find persons that occur in (at least) three different conferences

Named Graphs

- Instead of triples, we have quads
- Graph can serve as ‘context’
- Queries may specify the datasets to be used for matching
  - **`FROM`** clauses to refer to default graphs
  - **`FROM NAMED`** clauses to refer to named graphs
  - If **`FROM NAMED`** clauses are provided without a **`FROM`** clause, the default empty graph is assumed to be used.

Named Graphs




```
PREFIX ex: <…>
SELECT …
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
  … A …
  GRAPH ex:g3 {
    … B …
  }
  GRAPH ?graph {
    … C …
  }
}
```

Image from Feigenbaum and Prud'hommeaux

SPARQL CONSTRUCT

- Construct graphs from graphs based on a query
- Useful for – one-step – transformation between vocabularies

SPARQL CONSTRUCT

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT { ?agent a foaf:Agent . }
WHERE {
   { ?agent a ex:Cat. }
   UNION
   { ?agent a foaf:Person. }
}
```

CONSTRUCT template.

```
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:    <http://foo.bar/#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:bettina     a  foaf:Agent .
ex:victor      a  foaf:Agent .
ex:gaston      a  foaf:Agent .
ex:chrdebru    a  foaf:Agent .
```

SPARQL ASK

- Return True or False
- Do we have cat owners?

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK WHERE { ?person ex:owns ?cat . }
```

SPARQL DESCRIBE

- Server decides whatever RDF to return describing a resource.
- One needs to be careful, as server decides how to interpret the query: “The description is determined by the query service.”

SPARQL DESCRIBE

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DESCRIBE ?cat
WHERE { ?cat foaf:name "Victor" . }
```

```
...
@prefix ex:    <http://foo.bar/#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
ex:victor  a        ex:Cat ;
        foaf:name   "Victor" .
```

Projected Expressions

```
PREFIX : <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?element ?protons
(ROUND(?weight) - ?protons AS ?neutrons)
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE {
  [] a :Element ;
    :atomicNumber ?protons ;
    :atomicWeight ?weight ;
    :name ?element .
} ORDER BY ?protons
```

[ ] is used for unnamed variables; cfr. '_' in prolog

Assignment

```
PREFIX : <http://www.daml.org/2003/01/periodictable/PeriodicTable#>
SELECT ?element ?protons ?neutrons
FROM <http://www.daml.org/2003/01/periodictable/PeriodicTable.owl>
WHERE {
  [] a :Element ;
    :atomicNumber ?protons ;
    :atomicWeight ?weight ;
    :name ?element .
  BIND(ROUND(?weight) - ?protons AS ?neutrons)
} ORDER BY ?protons
```

Aggregates

```
PREFIX roads: <http://transport.data.gov.uk/def/traffic/>
SELECT ?cat (COUNT(DISTINCT ?thing) AS ?roads)
WHERE {
   ?thing a roads:Road .
   ?thing roads:countPointRoadCategory ?cat .
} GROUP BY ?cat
```

Aggregates

- Similar to SQL
- COUNT, MIN, MAX, SUM, AVG, GROUP_CONCAT, SAMPLE
- Also the introduction of the HAVING clause to filter the results of the query after applying aggregates.

Subqueries

- Retrieve the “second page” of names and emails of people in Tim Berners-Lee's FOAF file, given that each page has 10 people.

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
    ?person foaf:name ?name .
    OPTIONAL { ?person foaf:mbox ?email }
} ORDER BY ?name LIMIT 10 OFFSET 10
```

Subqueries

- Problem?
  - People with n > 1 email addresses are appearing n times in the result set
  - How do we show ten people, AND their n > 1 email addresses?

name	email
"Dave Beckett"	<mailto:dave@dajobe.org>
"Dean Jackson"	<mailto:dean@w3.org>
"Dean Jackson"	<mailto:dino@grorg.org>
"Edd Dumbill"	<mailto:edd@usefulinc.com>
"Edd Dumbill"	<mailto:edd@xml.com>
"Edd Dumbill"	<mailto:edd@xmlhack.com>
"Eric Miller"	<mailto:em@w3.org>
"Henrik Nielsen"	
"Henry Story"	
"Håkon Wium Lie"	

Subqueries

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
FROM <http://…/webdav/timbl/foaf.rdf>
WHERE {
    {
        SELECT DISTINCT ?person ?name WHERE {
            ?person foaf:name ?name
        } ORDER BY ?name LIMIT 10 OFFSET 10
    }
    OPTIONAL { ?person foaf:mbox ?email }
}
```

Negation in SPARQL 1.0

Find the people that do not contain a URL with the rdfs:seeAlso predicate:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  OPTIONAL { ?person rdfs:seeAlso ?url }
  FILTER(!bound(?url))
}
```

Negation in SPARQL 1.1

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  MINUS { ?person rdfs:seeAlso ?url }
}
```

SPARQL 1.1 MINUS graph pattern clause is a binary operator that removes bindings that match the right-hand side.

Negation in SPARQL 1.1

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name ?url
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
  ?person a foaf:Person ; foaf:name ?name .
  FILTER(NOT EXISTS { ?person rdfs:seeAlso ?url })
}
```

SPARQL 1.1 NOT EXISTS filter uses the bindings from a solution to test whether a given pattern exists.

Negation in SPARQL 1.1

NOT EXISTS and MINUS represent two ways of thinking about negation, one based on testing whether a pattern exists in the data, given the bindings already determined by the query pattern, and one based on removing matches based on the evaluation of two patterns. In some cases they can produce different answers.

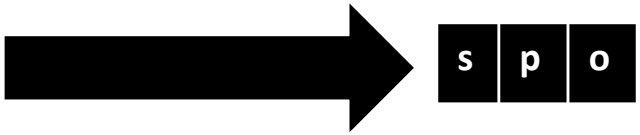
MINUS vs. FILTER NOT EXISTS?

Given:

```
@prefix : <http://foo/> .
:a :b :c .
```

```
SELECT * {
  ?s ?p ?o
  FILTER(NOT EXISTS{ ?x ?y ?z })
}
```

{ ?x ?y ?z } matches given any ?s ?p ?o, so NOT EXISTS { ?x ?y ?z } eliminates any solutions.




```
SELECT * {
   ?s ?p ?o
   MINUS { ?x ?y ?z }
}
```

s	p	o
http://foo/a	http://foo/b	http://foo/c

No shared variable between the first part (?s ?p ?o) and the second (?x ?y ?z) so no bindings are eliminated.

Property Paths

Property paths for querying arbitrary-length paths through the dataset graphs.

Find all instances of beer:

- `?beer rdf:type beer:Beer .`

Find all instances of beer, or instances of subclasses of beer

- `?beer rdf:type/rdfs:subClassOf* beer:Beer .`

This assumes that the ontology is also available via the endpoint, but it allows us to "reason" over the graph.

Property Paths

Match one or both possibilities

```
SELECT *
{ :book1 dc:title|rdfs:label ?displayString }
```

Find the name of any people that Alice knows (sequence).

```
SELECT ?x ?name {
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows/foaf:name ?name .
}
```

Property Paths

Find the name of any people known by someone that Alice knows (2 foaf:knows links away).

```
SELECT ?x ?name {
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

This is equivalent to…

```
SELECT ?x ?name {
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows ?a1 .
 ?a1 foaf:knows ?a2 .
 ?a2 foaf:name ?name .
}
```

Property Paths

Find the name of any people known by someone that Alice knows (2 foaf:knows links away).

```
SELECT ?x ?name {
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

This is equivalent to… *(with no explicit variables)*.

```
SELECT ?x ?name {
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows [ foaf:knows [ foaf:name ?name ]].
}
```

Federated Queries

- Example 1. We have books and their titles in our endpoint. We want to retrieve the authors of books. They are available in another endpoint and we join the books on their title.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?a {
   ?b dc:title ?title .
   SERVICE <http://sparql.org/books> {
      ?s dc:title ?title .
      ?s dc:creator ?a
   }
}
```

Federated Query

- Example 2. SPARQL Endpoint of Ordnance Survey UK allows federated queries to be executed.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT * WHERE {
  ?something rdfs:label ?label .
  SERVICE <http://dbpedia.org/sparql> {
    ?other rdfs:label ?label2 .
    FILTER(STR(?label2) = ?label)
  }
} LIMIT 2
```

- What does this query do? What problems can occur?

Summary

- Different SPARQL queries
  - SELECT, CONSTRUCT, DESCRIBE, ASK
- Use of named graphs

- Not covered
  - SPARQL Update (to be applied in your project)
  - Graph Store HTTP Protocol.
  - Service descriptions
  - Entailment regimes

References

- Feigenbaum and Prud'hommeaux. SPARQL by Example. http://www.cambridgesemantics.com/semantic-university/sparql-by-example.
- Chapter 7 of Foundations of Semantic Web Technologies.