Supplementary Material on SPARQL

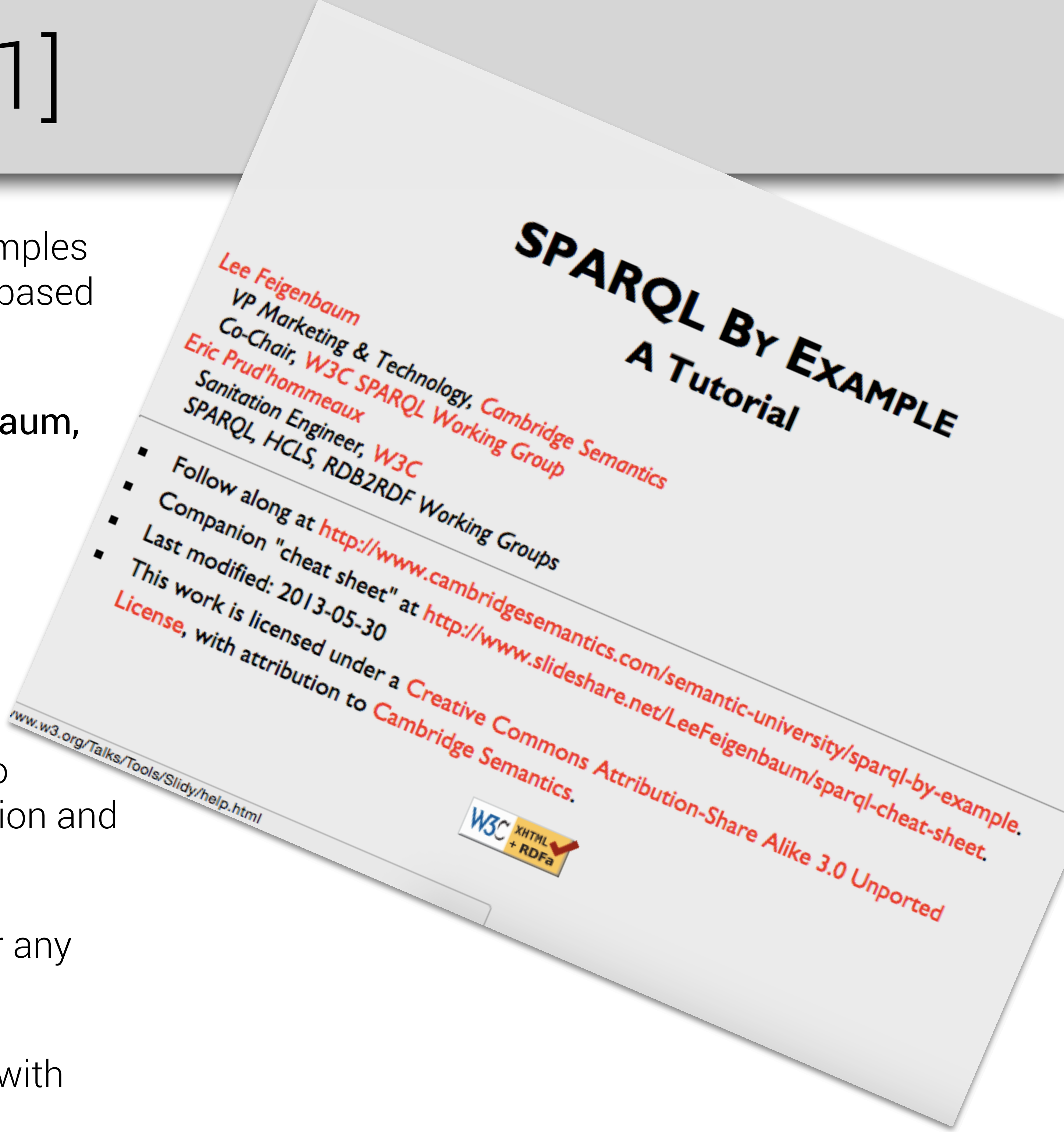
MANCHESTER
1824
The University of Manchester

SPARQL by Example

Alvaro A A Fernandes
School of Computer Science
University of Manchester

Acknowledgements [1]

- Some of these slides mostly contain text and examples that are most often taken *verbatim* from the web-based tutorial:
    - SPARQL By Example: A Tutorial. Lee Feigenbaum, Eric Prud'hommeaux.(2013) http://www.cambridgesemantics.com/semantic-university/sparql-by-example
- These slides were put together from the above publication for educational purposes only.
- Any changes made either reflect recent updates to SPARQL or are the result of minor editing, adaptation and extension for use in teaching.
- The author of these slides is solely responsible for any errrors they contain.
- No claims are made by the author of these slides with regards to any form of intellectual property rights wherever the above authors are acknowledged.
- The author of these slides is very grateful to the authors of the work above.



SPARQL By Example

A Tutorial

Lee Feigenbaum
VP Marketing & Technology, Cambridge Semantics
Co-Chair, W3C SPARQL Working Group
Eric Prud'hommeaux
Sanitation Engineer, W3C
SPARQL, HCLS, RDB2RDF Working Groups

- Follow along at http://www.cambridgesemantics.com/semantic-university/sparql-by-example.
- Companion "cheat sheet" at http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet.
- Last modified: 2013-05-30
- This work is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License, with attribution to Cambridge Semantics.

ww.w3.org/Talks/Tools/Slidy/help.html

Acknowledgements [2]

- Some of these slides mostly contain text and examples that are most often taken *verbatim* from W3C documents mentioned in the slide titled **W3C Documents**.

Why SPARQL?

- SPARQL is the query language of the Semantic Web.
- It lets us:
  - 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

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Basic Notions [1]

- The RDF data model is based on graphs, represented as sets of triples of the form <*subject, predicate, object*>.

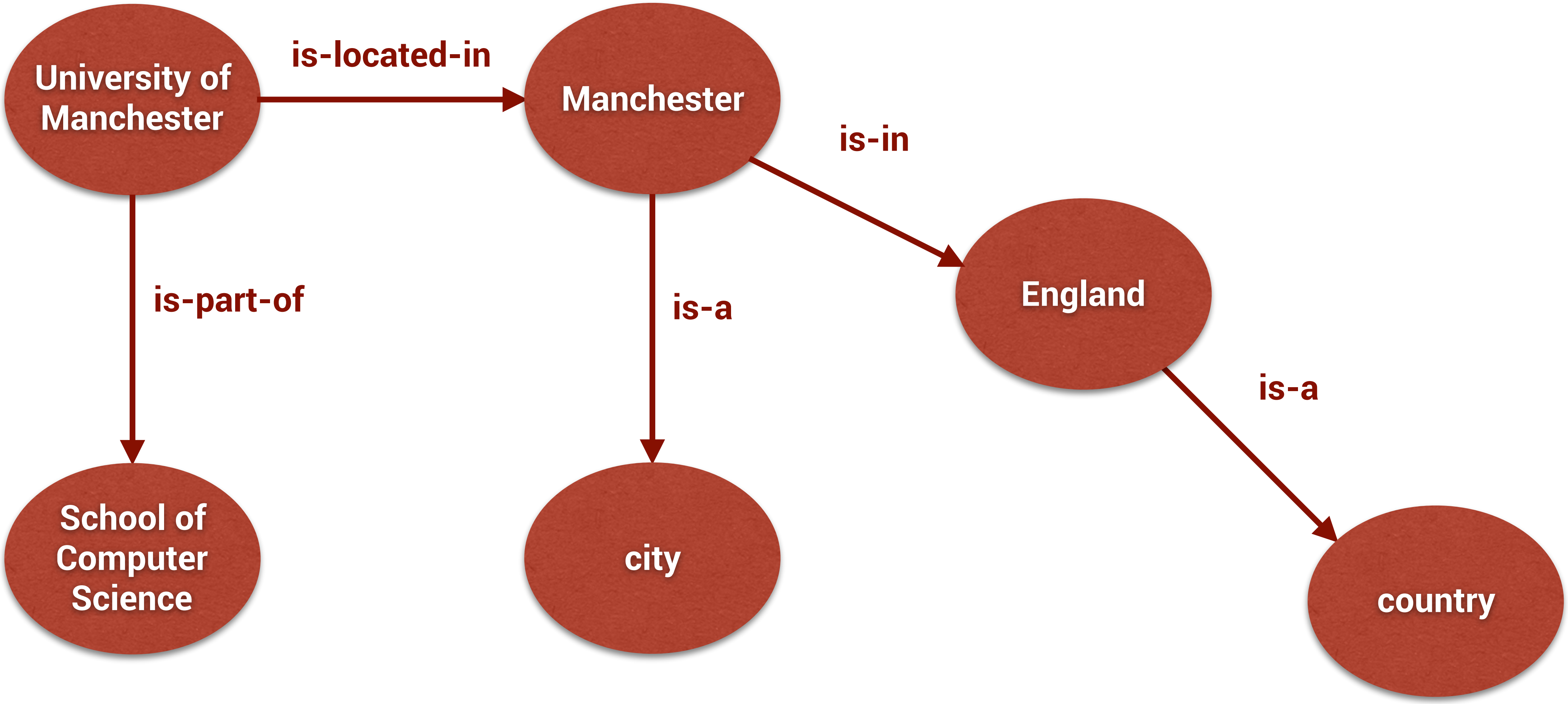
- Subjects, predicates, and objects are represented with URIs, which can be abbreviated as prefixed names.

- Objects can also be literals, i.e., strings, integers, Booleans, etc.

A URI (uniform resource identifier) is limited to a subset of the ASCII character set. An IRI (internationalized RI) may contain characters from the Universal Character Set. A URL (UR locator) is a URI that encodes the network access path to the resource. A URN (UR name) is a URI that is just a name.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Sets of RDF Triples are Graphs



Basic Notions [2]

- There are various way of writing down RDF.

Roughly, Turtle is more expressive than N-triples and less expressive than N3.

- One style is referred to as Turtle (for 'Terse RDF Triple Language'), the RDF part of N3.

```
<http://example.com/resource>
prefix:name
```

Alternative Turtle notations for URIs

```
"plain string"
"13.4"^^xsd:float
"string with language"@en
```

Different Turtle notations for literals

```
prefix:subject other_prefix:predicate "object" .
```

Turtle notations for s-p-o triples. Note the final dot.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Structure of a SPARQL Query [1]

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

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Structure of a SPARQL Query [2]

```
# prefix declarations
PREFIX foo:<http://example.com/resources/>
…

# dataset definition
FROM …

# result clause
SELECT …

# query pattern
WHERE {
  …
}

# query modifiers
ORDER BY …
```

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

SPARQL: Architecture and Endpoints [1]

- SPARQL queries are executed against RDF datasets, consisting of RDF graphs.
- A SPARQL endpoint accepts queries and returns results via HTTP (and hence can also lie behind browser-accessible webpage-based interfaces).

- There are generic endpoints that can query any Web-accessible RDF data.
- Specific endpoints are hardwired to query specific datasets.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

SPARQL: Architecture and Endpoints [2]

- The results of SPARQL queries can be returned (or be rendered) in a variety of formats:
  - **XML**: SPARQL specifies an XML vocabulary for returning results.
  - **JSON**: There is a JSON serialization of the XML vocabulary that is particularly useful for Web applications.
  - **RDF**. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)
  - **HTML**. This is useful when using an interactive form to work with SPARQL queries and is usually implemented by applying an XSL transform to XML results.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Example FOAF (Friend of a Friend) Data

- FOAF is a standard RDF vocabulary for describing people and relationships
- Take Tim Berners-Lee's FOAF information available (in RDF-XML) at http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf
- Using a notation converter (e.g., http://www.w3.org/RDF/Validator/uri), we can transform the more verbose RDF-XML into the more concise Turtle notation.
- An excerpt is shown in the next slide.
- The subsequent slide has our first SPARQL query: *find all the names of people mentioned in Tim's FOAF file*.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

An Excerpt of the Turtle Version of Tim Berners-Lee FOAF Card

```
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
@prefix cc: <http://creativecommons.org/ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix ns0: <http://www.w3.org/2000/10/swap/pim/contact#> .

<http://dig.csail.mit.edu/2008/2002/01/tr-automation/tr.rdf> dc11:title "W3C Standards and Technical Reports" .
<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
  cc:license <http://creativecommons.org/licenses/by-nc/3.0/> ;
  dc11:title "Tim Berners-Lee's FOAF file" ;
  a foaf:PersonalProfileDocument ;
  foaf:maker <http://www.w3.org/People/Berners-Lee/card#i> ;
  foaf:primaryTopic <http://www.w3.org/People/Berners-Lee/card#i> .

<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#cm>
  a foaf:Person ;
  rdfs:seeAlso <http://www.koalie.net/foaf.rdf> ;
  foaf:name "Coralie Mercier" .

<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#dj>
  a foaf:Person ;
  rdfs:seeAlso <http://www.grorg.org/dean/foaf.rdf> ;
  foaf:homepage <http://www.grorg.org/dean/> ;
  foaf:mbox <mailto:dean@w3.org>, <mailto:dino@grorg.org> ;
  foaf:mbox_sha1sum "6de4ff27ef927b9ba21ccc88257e41a2d7e7d293" ;
  foaf:name "Dean Jackson" .
…
```

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q1: SELECT, variables, and a triple pattern [1]

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

Return all names mentioned in Tim Berners-Lee's FOAF file.

In the graph http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf, find all subjects (?person) and objects (?name) linked with the foaf:name predicate, then return all the values of ?name.

q1.rq

- SPARQL variables start with a **?** and can match any node (i.e., either resource or literal) in the RDF dataset.
- A triple pattern also has a triple-form but one or more, and possibly all or none, of the element is replaced with a variable.
- The **SELECT** result clause defines a table of variables with the values that satisfy the query.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q1: SELECT, variables, and a triple pattern [2]

roqet is a simple SPARQL engine

http://librdf.org/

We can pass the URI of the graph as an invocation parameter

```
$ roqet --data dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf q1.rq
roqet: Running query from file q1.rq
roqet: Query has a variable bindings result
row: [name=string("Coralie Mercier")]
row: [name=string("Dean Jackson")]
row: [name=string("Edd Dumbill")]
row: [name=string("Libby Miller")]
row: [name=string("Susie Stephens")]
row: [name=string("Henry Story")]
…
row: [name=string("Daniel J Weitzner")]
row: [name=string("Karl Dubost")]
row: [name=string("World Wide Web Consortium")]
roqet: Query returned 52 results
```

- Dataset: http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf

Q1: SELECT, variables, and a triple pattern [3]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM   <file:timbl-foaf.rdf>
WHERE {
    ?person foaf:name ?name .
}
```

q1a.rq

If the data exists locally, in a file called 'timbl-foaf.rdf' in the current directory, this query would return the same result as q1.rq

- A **FROM** clause (and there can be more than one) points to data to be used in evaluating the query.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q1: SELECT, variables, and a triple pattern [4]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
    ?person foaf:name ?name .
}
```

Note the reference to a remote data source.

q1b.rq

- The data pointed to by a `FROM` clause need not be local.
- This make reference to the data source part of the query text, as opposed to an invocation parameter, as in a previous example.
- The result, once again, is the same as for q1.rq.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q2: SELECT with literals in a triple pattern [1]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?s ?p
FROM   <file:timbl-foaf.rdf>
WHERE {
    ?s ?p "Amy van der Hiel" .
}
```

Return all that is said about Amy van der Hiel in Tim Berners-Lee's FOAF file.

Find all subjects (?s) and predicates (?p) that have "Amy van der Hiel" as object, then return all the values of ?s and ?p.

q2.rq

- Triple patterns can contain literals.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q2: SELECT with literals in a triple pattern [2]

```
$ roqet q2.rq
roqet: Running query from file q2.rq
roqet: Query has a variable bindings result
row: [s=URI<http://www.w3.org/People/Berners-Lee/card#amy>,
p=URI<http://www.w3.org/2000/01/rdf-schema#label>]
row: [s=URI<http://www.w3.org/People/Berners-Lee/card#amy>,
p=URI<http://xmlns.com/foaf/0.1/name>]
roqet: Query returned 2 results
```

"Amy van der Hiel" appears as an RDFS label an a FOAF name in the card

- Dataset: http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf

Q3: Multiple triple patterns, project all [1]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT *
FROM   <file:timbl-foaf.rdf>
WHERE {
    ?person foaf:name ?name .
    ?person foaf:mbox ?email .
}
```

q3.rq

Find all the people in Tim Berners-Lee's FOAF file that have names and email addresses. Return each person's URI, name, and email address

- Analogously to SQL, SPARQL accepts the `*` shorthand in the `SELECT` clause to denote all the variables in the query.
- Very importantly, note that two or more occurrences of the same variable (in this case, `?person`) implicitly implies their identity, i.e., the `WHERE` clause above is only satisfied if the value that binds the first occurrence of `?person` is identical to (i.e., the same as) the one that binds the second occurrence.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q3: Multiple triple patterns, project all [2]

```
$ roqet q3.rq
roqet: Running query from file q3.rq
roqet: Query has a variable bindings result
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#dj>, name=string("Dean Jackson"),
email=URI<mailto:dean@w3.org>]
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#dj>, name=string("Dean Jackson"),
email=URI<mailto:dino@grorg.org>]
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#edd>, name=string("Edd Dumbill"),
email=URI<mailto:edd@usefulinc.com>]
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#edd>, name=string("Edd Dumbill"),
email=URI<mailto:edd@xml.com>]
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#edd>, name=string("Edd Dumbill"),
email=URI<mailto:edd@xmlhack.com>]
row: [person=URI<http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#libby>, name=string("Libby Miller"),
email=URI<mailto:libby.miller@bristol.ac.uk>]
row: [person=URI<http://www.aaronsw.com/about.xrdf#aaronsw>, name=string("Aaron Swartz"),
email=URI<mailto:me@aaronsw.com>]
row: [person=URI<http://www.w3.org/People/Berners-Lee/card#amy>, name=string("Amy van der Hiel"),
email=URI<mailto:amy@w3.org>]
row: [person=URI<http://www.w3.org/People/Connolly/#me>, name=string("Dan Connolly"),
email=URI<mailto:connolly@w3.org>]
row: [person=URI<http://www.w3.org/People/EM/contact#me>, name=string("Eric Miller"),
email=URI<mailto:em@w3.org>]
row: [person=URI<http://www.w3.org/People/karl/karl-foaf.xrdf#me>, name=string("Karl Dubost"),
email=URI<mailto:karl@w3.org>]
roqet: Query returned 11 results
```

- Dataset: http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf

Q4: Linking nodes, navigating paths [1]

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
FROM <file:timbl-foaf.rdf>
WHERE {
    card:i foaf:knows ?known .
    ?known foaf:homepage ?homepage .
}
```

Find the homepage of anyone known by Tim Berners-Lee.

q4.rq

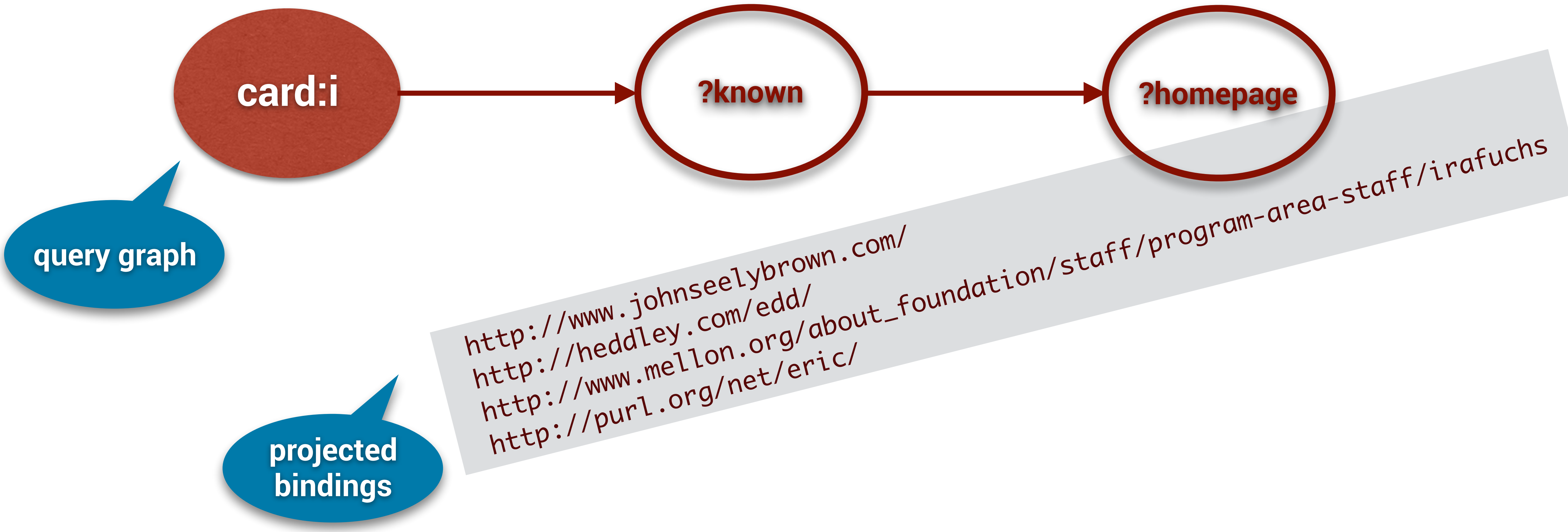
In the above, card:i denotes <http://www.w3.org/People/Berners-Lee/card#i>, who is asserted to be the foaf:maker of the data source being queried.

- Note how a variable (such as ?known above) that occurs in object position in one pattern and in subject position in another pattern allows us to link nodes and navigate paths, as depicted in the next slide.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q4: Linking nodes, navigating paths [2]

```
$ roqet q4.rq
roqet: Running query from file q4.rq
roqet: Query has a variable bindings result
row: [homepage=URI<http://www.johnseelybrown.com/>]
row: [homepage=URI<http://heddley.com/edd/>]
row: [homepage=URI<http://www.mellon.org/about_foundation/staff/program-area-staff/irafuchs>]
row: [homepage=URI<http://purl.org/net/eric/>]
roqet: Query returned 4 results
```



```
http://www.johnseelybrown.com/
http://heddley.com/edd/
http://www.mellon.org/about_foundation/staff/program-area-staff/irafuchs
http://purl.org/net/eric/
```

- Dataset: http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf

Q5: Exploring a data source [1]

```
SELECT DISTINCT ?Concept
WHERE {[] a ?Concept}
LIMIT 100
```

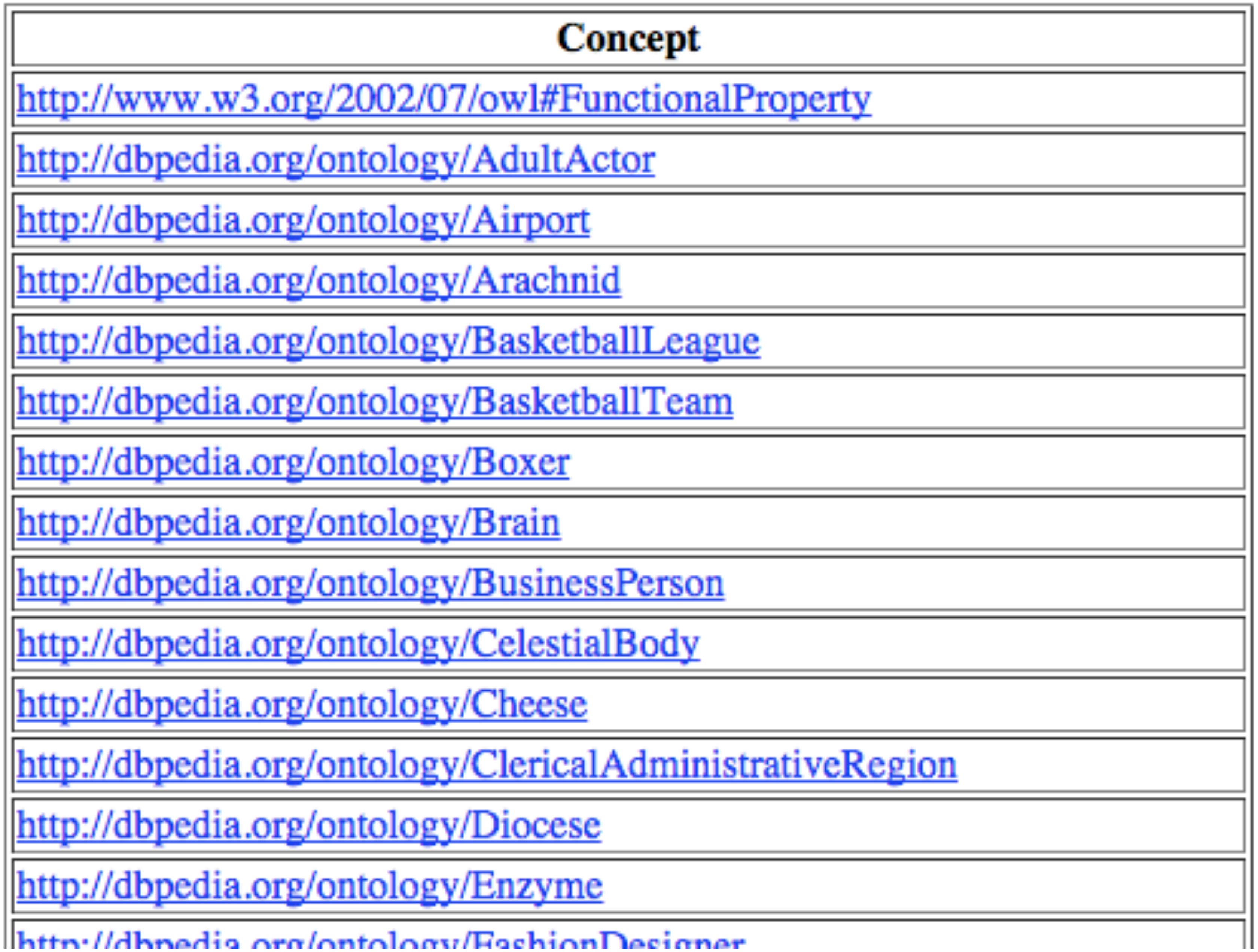
q5.rq

Which concepts are mentioned in the data?

- Note that, as in SQL, the keyword `DISTINCT` causes duplicates to be removed.
- Note that the subject position in the triple pattern is not of interest and, since the variable that would otherwise be used only occurs once, it can be denoted by `[]`, i.e., as a blank node.
- Note, finally, that `a` is a shorthand for `rdf:type`.
- Finally, note the `LIMIT` clause, which restricts the number of results returned.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q5: Exploring a data source [2]



Concept
http://www.w3.org/2002/07/owl#FunctionalProperty
http://dbpedia.org/ontology/AdultActor
http://dbpedia.org/ontology/Airport
http://dbpedia.org/ontology/Arachnid
http://dbpedia.org/ontology/BasketballLeague
http://dbpedia.org/ontology/BasketballTeam
http://dbpedia.org/ontology/Boxer
http://dbpedia.org/ontology/Brain
http://dbpedia.org/ontology/BusinessPerson
http://dbpedia.org/ontology/CelestialBody
http://dbpedia.org/ontology/Cheese
http://dbpedia.org/ontology/ClericalAdministrativeRegion
http://dbpedia.org/ontology/Diocese
http://dbpedia.org/ontology/Enzyme
http://dbpedia.org/ontology/FashionDesigner

Pasting the query text of Q5 in the webpage that front-ends the SPARQL endpoint (see below) with result format set to renderable HTML returns a table, the top of which is shown to the left.

- Endpoint: http://live.dbpedia.org/sparql

Using the SPARQL Protocol to Query a SPARQL Endpoint

- To encode/decode the text of a query in order to be able to send it to a SPARQL endpoint that conforms to the SPARQL Protocol, try:

  http://www.url-encode-decode.com/

- For example, recall Q5. Pasting and then encoding its text:

```
SELECT DISTINCT ?Concept
WHERE {[] a ?Concept}
LIMIT 100
```

- returns the percent-encoded string:

  SELECT+DISTINCT+%3FConcept%0D%0AWHERE+%7B%5B%5D+a+%3FConcept%7D%0D%0ALIMIT+100

- Then, passing this query string to an endpoint in RESTful style, e.g.:

  http://live.dbpedia.org/sparql?query=SELECT+DISTINCT+%3FConcept%0D%0AWHERE+%7B%5B%5D+a+%3FConcept%7D%0D%0ALIMIT+100

- returns renderable HTML-formatted results (and if you use a command-line tool like CURL instead of a browser, you get XML-formatted results).

Q6: Filtering the returned results [1]

Return the country name and population estimate of landlocked countries that are estimated to have more than 15M inhabitants?

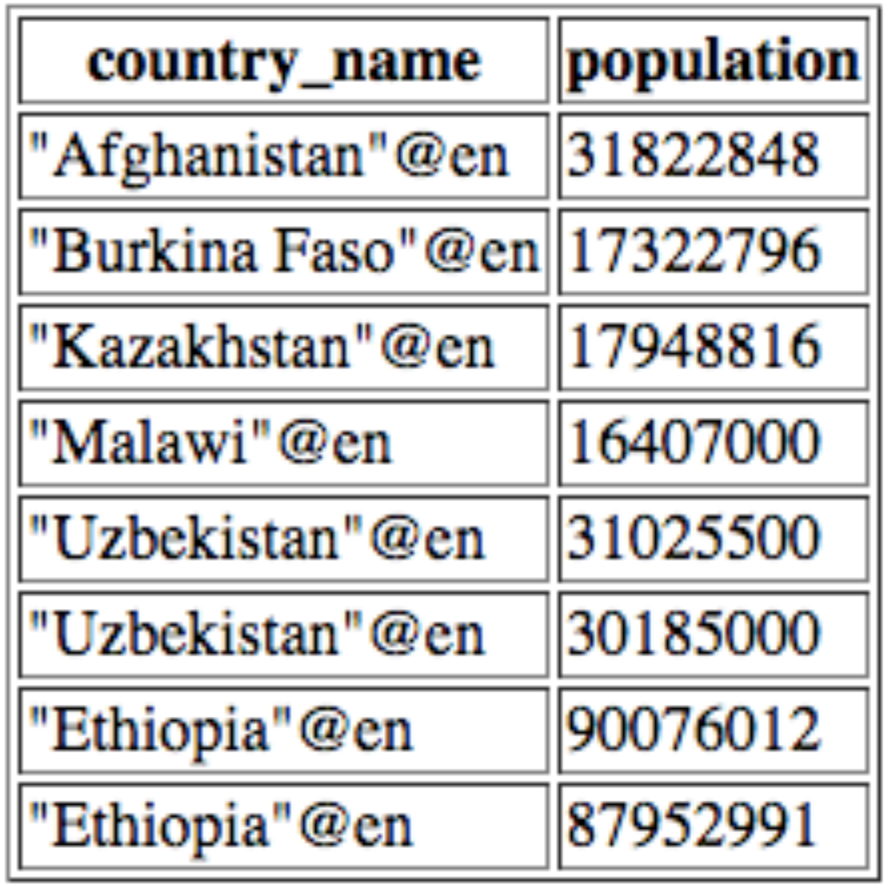
q6.rq

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

- A `FILTER` clause specifies a predicate that must be true of the returned bindings.
- Note that how the semicolon (`;`) can be used to abbreviate a query by allowing the omission of a common subject.
- A sequence of patterns that share the same subject, e.g., `{?x a ?y . ?x rdfs:label ?z .}`, can be written as `{?x a ?y ; rdfs:label ?z .}`

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q6: Filtering the returned results [2]



country_name	population
"Afghanistan"@en	31822848
"Burkina Faso"@en	17322796
"Kazakhstan"@en	17948816
"Malawi"@en	16407000
"Uzbekistan"@en	31025500
"Uzbekistan"@en	30185000
"Ethiopia"@en	90076012
"Ethiopia"@en	87952991

More than one value of population for the same country name indicates that there is more than one estimate for the population of the corresponding country in the dataset.

- Endpoint: http://live.dbpedia.org/sparql

Some SPARQL Built-In Functions

- Logical
  - `!`
  - `&&`
  - `||`
- Arithmetic
  - `+`
  - `-`
  - `*`
  - `/`
- Comparison
  - `=`
  - `!=`
  - `>`
  - `<`
  - ...
- Testing
  - `isURI`
  - `isBlank`
  - `isLiteral`
  - `bound`
- Access
  - `str`
  - `lang`
  - `datatype`
- Other
  - `sameTerm`
  - `langMatches`
  - `regex`

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q7: Renaming, mapping, ordering [1]

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT str(?country_name) AS ?name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000) .
}
ORDER BY DESC(?population)
```

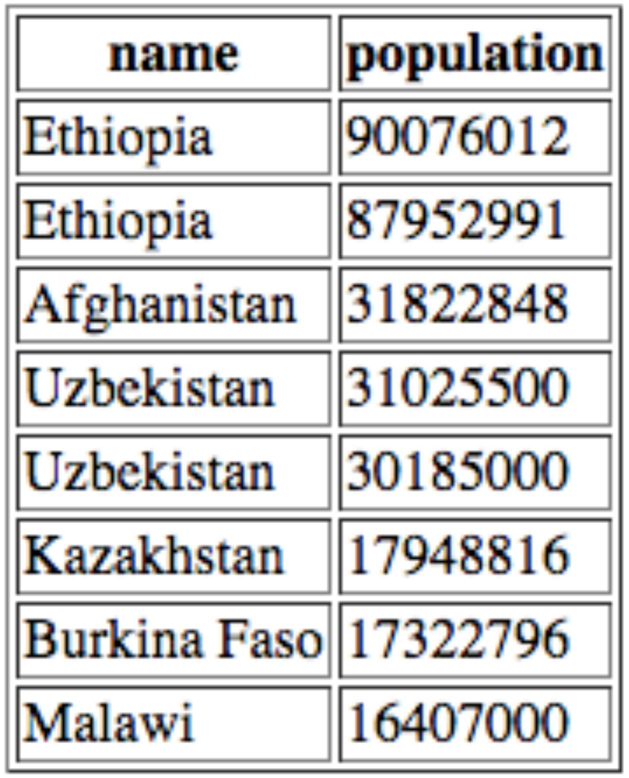
Return the name and population estimate (in descending order of the latter) of landlocked countries that are estimated to have more than 15M inhabitants.

q7.rq

- As in SQL, an **AS** keyword is available for renaming, and functions (such as str can be used to map values before emission into the result stream).
- Again, as in SQL, an **ORDER BY** clause is also available in SPARQL.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q7: Renaming, mapping, ordering [2]



name	population
Ethiopia	90076012
Ethiopia	87952991
Afghanistan	31822848
Uzbekistan	31025500
Uzbekistan	30185000
Kazakhstan	17948816
Burkina Faso	17322796
Malawi	16407000

This is in descending order of population and the name is stripped of the language qualifier.

- Endpoint: http://live.dbpedia.org/sparql

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q8: Dealing with Missing Information [1]

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?a
WHERE {
  ?a a mo:MusicArtist .
}
```

q8a.rq

Return the music artists in Jamendo.

Data is from DBTune, which serves data from (among others) the Jamendo music repository.

3,505 rows!

Q8: Dealing with Missing Information [2]

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name ;
     foaf:img ?img ;
     foaf:homepage ?hp ;
     foaf:based_near ?loc .
}
```

q8b.rq

Return the name, image, homepage and location of the music artists in Jamendo.

2,667 rows!

Some artists don't have image, homepage or location recorded!

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q8: Dealing with Missing Information [1]

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name .
  OPTIONAL { ?a foaf:img ?img }
  OPTIONAL { ?a foaf:homepage ?hp }
  OPTIONAL { ?a foaf:based_near ?loc }
}
```

Return the name, and, *if possible*, the image, homepage and location of the music artists in Jamendo.

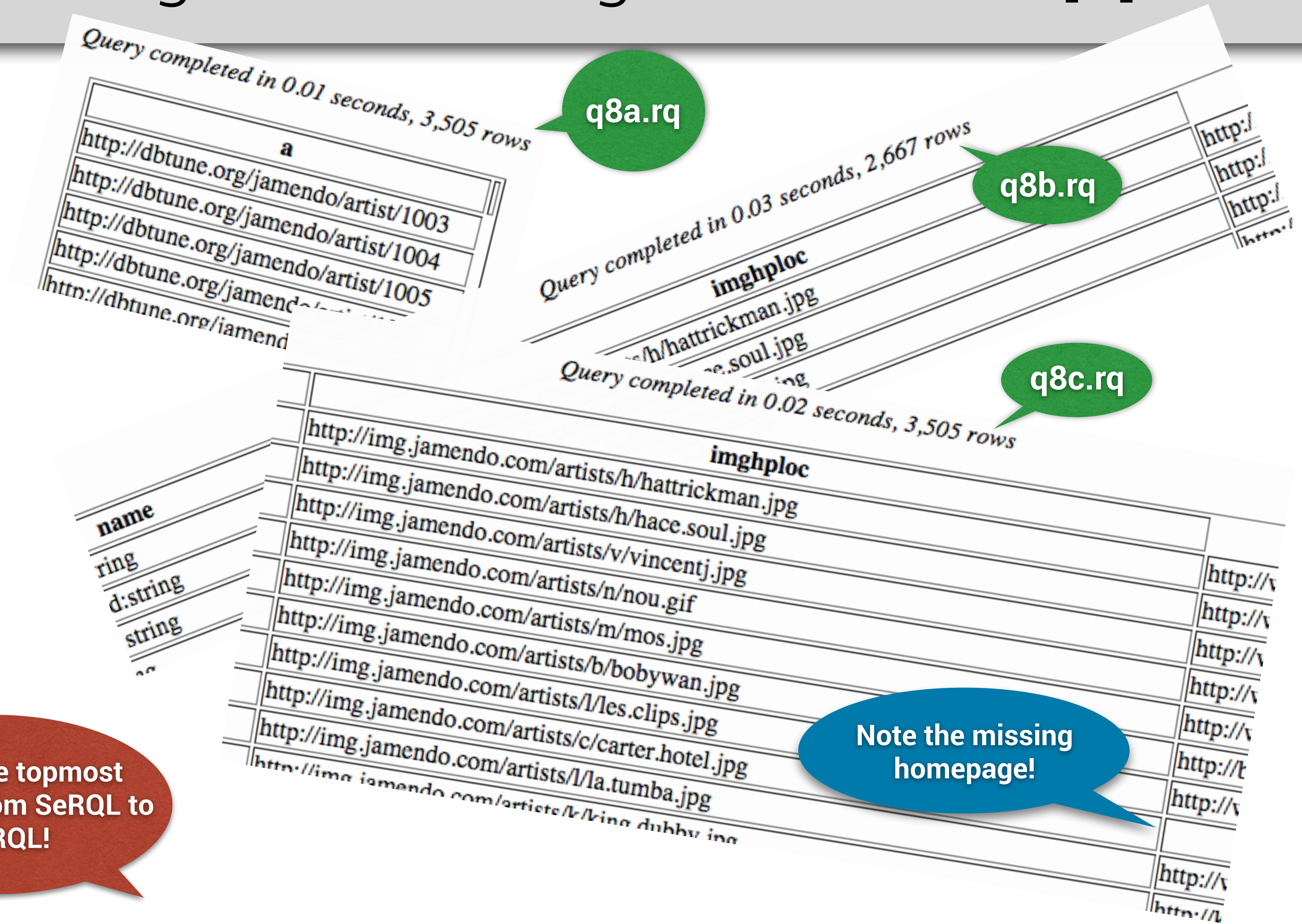
q8c.rq

3,505 rows!

- The `OPTIONAL` clause 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 (no value) for that solution.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q8: Dealing with Missing Information [2]



Query completed in 0.01 seconds, 3,505 rows

a
http://dbtune.org/jamendo/artist/1003
http://dbtune.org/jamendo/artist/1004
http://dbtune.org/jamendo/artist/1005
http://dbtune.org/jamend

Query completed in 0.03 seconds, 2,667 rows

name	imghploc
ring	/h/hattrickman.jpg
d:string	e.soul.jpg
string	

Query completed in 0.02 seconds, 3,505 rows

imghploc
http://img.jamendo.com/artists/h/hattrickman.jpg
http://img.jamendo.com/artists/h/hace.soul.jpg
http://img.jamendo.com/artists/v/vincentj.jpg
http://img.jamendo.com/artists/n/nou.gif
http://img.jamendo.com/artists/m/mos.jpg
http://img.jamendo.com/artists/b/bobywan.jpg
http://img.jamendo.com/artists/l/les.clips.jpg
http://img.jamendo.com/artists/c/carter.hotel.jpg
http://img.jamendo.com/artists/l/la.tumba.jpg
http://img.jamendo.com/artists/k/king.dubby.jpg

q8a.rq

q8b.rq

q8c.rq

Note the missing homepage!

Change the topmost pull-down from SeRQL to SPARQL!

- Endpoint: http://dbtune.org/jamendo/store/user/query

Q9: Retrieving Alternatives [1]

```
PREFIX dbpedia:  <http://dbpedia.org/resource/>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>
PREFIX dbpprop:  <http://dbpedia.org/property/>
SELECT DISTINCT ?s
WHERE {
    {?s dbpprop:nationality dbpedia:United_States}
  UNION
    {?s dbpprop:nationality dbpedia:United_Kingdom} .
  ?s dbpprop:birthDate ?d .
  FILTER (
     DATATYPE(?d) = xsd:date
  &&
     ?d >= "2001-01-01"^^xsd:date)
}
```

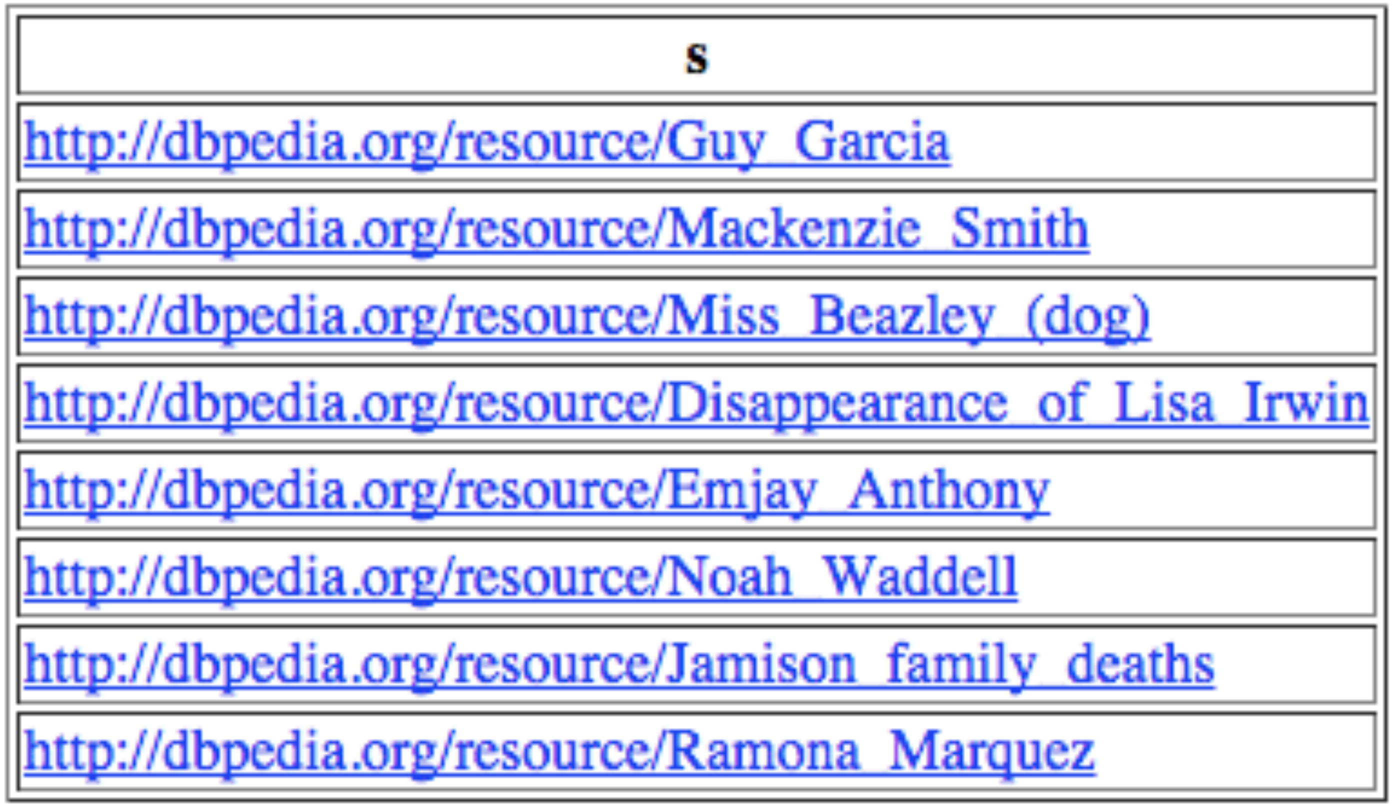
q9.rq

Return subjects who were born in the US or the UK on or after 1st Jan 2001 and have a DBpedia entry.

- The **UNION** operation expresses a disjunction of two graph patterns, i.e., a solution to either the left or the right operand (or to both) is included in the result stream.
- Note the use of **DATATYPE** to avoid comparing a string that is not an instance of the XML Schema date type and one that is.
- Note also how the comparison is expressed with explicit typing of the string.
- Note, finally, that in a FILTER clause, we can compose predicates in the normal way using **!**, **&&** and **||** (for *not*, *and* and *or*).

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q9: Retrieving Alternatives [2]



s
http://dbpedia.org/resource/Guy_Garcia
http://dbpedia.org/resource/Mackenzie_Smith
http://dbpedia.org/resource/Miss_Beazley_(dog)
http://dbpedia.org/resource/Disappearance_of_Lisa_Irwin
http://dbpedia.org/resource/Emjay_Anthony
http://dbpedia.org/resource/Noah_Waddell
http://dbpedia.org/resource/Jamison_family_deaths
http://dbpedia.org/resource/Ramona_Marquez

- Endpoint: http://live.dbpedia.org/sparql

RDF Datasets: Composition

- As was said earlier, SPARQL queries are executed against an RDF dataset, comprising one or more graphs.
- So far, all queries have been against a single graph.
- In SPARQL, this is known as *the default graph*.
- RDF datasets are composed of the default graph and zero or more *named graphs*, each identified by a URI.
- Named graphs can be specified with one or more **`FROM NAMED`** clauses, or they can be hardwired into a particular SPARQL endpoint.
- The **`GRAPH`** keyword allows portions of a query to match against a specific named graph in the RDF dataset.
- Anything outside a **`GRAPH`** clause matches against the default graph.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

RDF Datasets: Example

- http://data.semanticweb.org hosts RDF data regarding workshops, schedules, and presenters for the International Semantic Web (ISWC) and European Semantic Web Conference (ESWC) series of events.

- The data uses the FOAF, SWRC, and iCal ontologies.

- The data for each individual ISWC or ESWC event is stored in its own named graph, i.e., there is one named graph per conference event contained in this dataset.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q10: Querying Named Graphs [1]

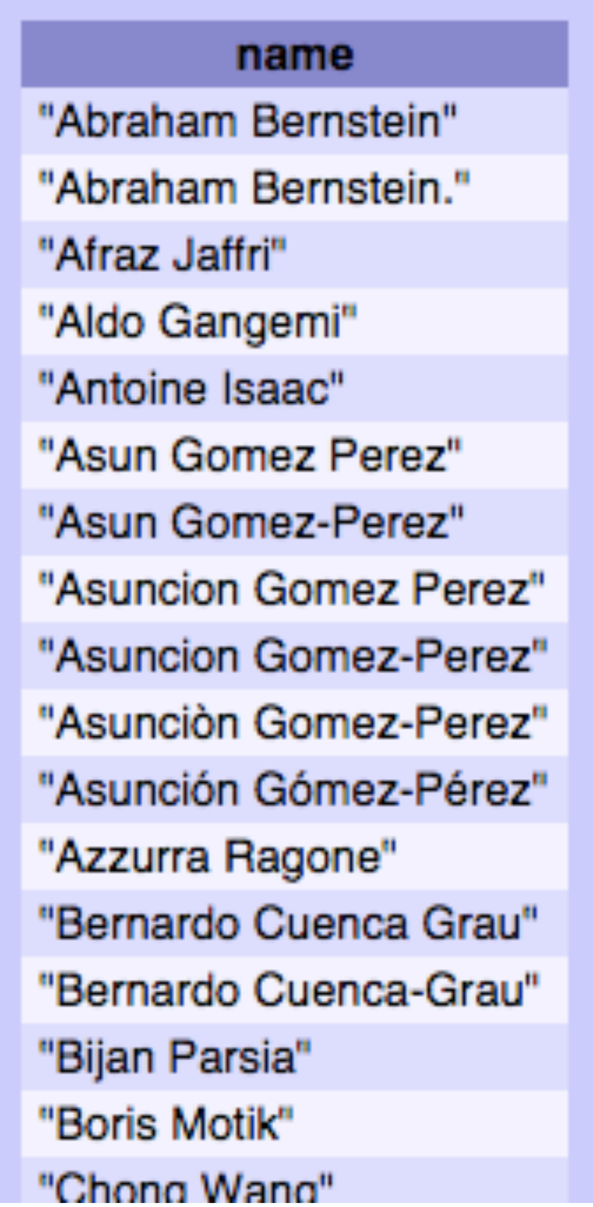
```
SELECT DISTINCT ?name
WHERE {
    ?person foaf:name ?name .
    GRAPH <http://data.semanticweb.org/conference/iswc-aswc/2007/complete>
      { ?person a foaf:Person }.
    GRAPH <http://data.semanticweb.org/conference/eswc/2007/complete>
      { ?person a foaf:Person }.
}
ORDER BY ?name
```

q10.rq

Return the name of every person that attended ISWC and ESWC in 2007.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q10: Querying Named Graphs [2]



name
"Abraham Bernstein"
"Abraham Bernstein."
"Afraz Jaffri"
"Aldo Gangemi"
"Antoine Isaac"
"Asun Gomez Perez"
"Asun Gomez-Perez"
"Asuncion Gomez Perez"
"Asuncion Gomez-Perez"
"Asunciòn Gomez-Perez"
"Asunción Gómez-Pérez"
"Azzurra Ragone"
"Bernardo Cuenca Grau"
"Bernardo Cuenca-Grau"
"Bijan Parsia"
"Boris Motik"
"Chong Wang"

Note how, in spite of DISTINCT, minor syntactic variations produce many semantic duplicates

- Endpoint: http://data.semanticweb.org/snorql/

Q11: Constructing New Graphs [1]

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?X vCard:FN ?name .
  ?X vCard:URL ?url .
  ?X vCard:TITLE ?title .
}
FROM   <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
  OPTIONAL { ?X foaf:name ?name . FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:homepage ?url . FILTER isURI(?url) . }
  OPTIONAL { ?X foaf:title ?title . FILTER isLiteral(?title) . }
}
```

q11.rq

Retrieve Tim Berners-Lee FOAF card and create the vCard version of the FOAF information in it.

- `CONSTRUCT` is an alternative SPARQL result clause to `SELECT`: instead of returning a table of result values, it returns an RDF graph.
- The result RDF graph is created by taking the results of the equivalent `SELECT` query and filling in the values of variables that occur in the template that is the argument of the `CONSTRUCT` clause.
- Triples are not created in the result graph for template patterns that involve an unbound variable.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q11: Constructing New Graphs [2]

```
$ roqet --results rdfxml q11.rq
roqet: Running query from file q11.rq
roqet: Query has a graph result:
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#" xml:base="file:///[...]/q11.rq">
  <rdf:Description rdf:about="http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#cm">
    <vCard:FN>Coralie Mercier</vCard:FN>
  </rdf:Description>
  <rdf:Description rdf:about="http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#dj">
    <vCard:FN>Dean Jackson</vCard:FN>
  </rdf:Description>
  <rdf:Description rdf:about="http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf#dj">
    <vCard:URL rdf:resource="http://www.grorg.org/dean/"/>
  </rdf:Description>
…
  <rdf:Description rdf:about="http://www.w3.org/data#W3C">
    <vCard:FN>World Wide Web Consortium</vCard:FN>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/data#W3C">
    <vCard:URL rdf:resource="http://dig.csail.mit.edu/2008/"/>
  </rdf:Description>
</rdf:RDF>
roqet: Total 58 triples
```

- Dataset: http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q12: Asking True-or-False Questions [1]

```
PREFIX prop: <http://dbpedia.org/property/>
ASK # returns a Boolean
{
  <http://dbpedia.org/resource/Amazon_River> prop:length ?amazon .
  <http://dbpedia.org/resource/Nile> prop:length ?nile .
  FILTER(?amazon > ?nile) .
}
```

Is the Amazon river longer than the Nile River?

q12.rq

- The **ASK** result clause simply returns true or false depending on whether or not the query pattern has any matches in the dataset.
- As with **SELECT** queries, the Boolean result is (by default) encoded in an SPARQL Results Format XML document.
- Note that the **WHERE** keyword is optional, not only here but in any SPARQL query.
- Note also that the hash character is the up-to-the-end-of-the-line comment character in SPARQL.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q12: Asking True-or-False Questions [2]

```
false
```

This is the terse reply in the web front-end.

- Endpoint: http://live.dbpedia.org/sparql

Q13: Learning about a Resource [1]

```
DESCRIBE <http://data.linkedmdb.org/resource/actor/29362>
```

LinkedMDB (i.e., the linked data version of the Movie Databases), describe the actor identified by the URI http://data.linkedmdb.org/resource/actor/29362

q13.rq

- The `DESCRIBE` query result clause allows the query server to return whatever RDF it uses to describe a given resource(s).
- Because the server is free to interpret the request as it sees fit, `DESCRIBE` queries are not interoperable.
- Common implementations include concise-bounded descriptions, named graphs, minimum self-contained graphs, etc.
- Of course, we could have used graph patterns, with filters, etc., finally projecting out a variable in the head, as if it were a `SELECT` query.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q13: Learning about a Resource [2]

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:dbpedia="http://dbpedia.org/property/"
    xmlns:skos="http://www.w3.org/2004/02/skos/core#"
…
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <rdf:Description rdf:about="http://xmlns.com/foaf/0.1/Person">
    <rdfs:seeAlso rdf:resource="http://data.linkedmdb.org/sparql?query=DESCRIBE+%3Chttp%3A%2F%2Fxmlns.com%2Ffoaf
%2F0.1%2FPerson%3E"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.freebase.com/view/guid/9202a8c04000641f8000000000003c35">
    <rdfs:seeAlso rdf:resource="http://data.linkedmdb.org/sparql?query=DESCRIBE+%3Chttp%3A%2F%2Fwww.freebase.com%2Fview
%2Fguid%2F9202a8c04000641f8000000000003c35%3E"/>
  </rdf:Description>
  <foaf:Person rdf:about="http://data.linkedmdb.org/resource/actor/29362">
    <movie:performance rdf:resource="http://data.linkedmdb.org/resource/performance/113885"/>
    <movie:actor_name>Ang Lee</movie:actor_name>
    <movie:actor_nytimes_id></movie:actor_nytimes_id>
    <movie:actor_netflix_id></movie:actor_netflix_id>
    <movie:actor_actorid rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
    >29362</movie:actor_actorid>
    <rdfs:label>Ang Lee (Actor)</rdfs:label>
    <rdf:type rdf:resource="http://data.linkedmdb.org/resource/movie/actor"/>
    <foaf:page rdf:resource="http://www.freebase.com/view/guid/9202a8c04000641f8000000000003c35"/>
  </foaf:Person>
…
```

This server replies with an RDF/XML file.

This server lacks a web front-end, so we use the SPARQL Protocol.

http://data.linkedmdb.org/sparql?query=DESCRIBE+%3Chttp%3A%2F%2Fdata.linkedmdb.org%2Fresource%2Factor%2F29362%3E

Q14: Aggregation [1]

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT COUNT(?country) AS ?how_many
WHERE
{
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000) .
}
```

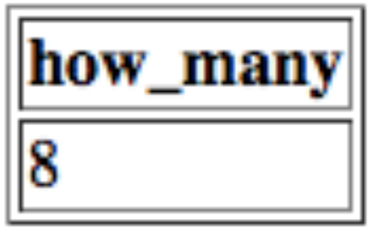
Compute how many landlocked countries are estimated to have more than 15M inhabitants.

q14.rq

- In SPARQL 1.1, aggregation functions can be used as in SQL.

from: SPARQL By Example: A Tutorial. Lee Feigenbaum and Eric Prud'hommeaux (2013)

Q14: Aggregation [2]



how_many
8

- Endpoint: http://live.dbpedia.org/sparql

Supplementary Material on SPARQL Semantics

Acknowledgements

- Some of these slides mostly contain text and examples that are most often taken *verbatim* from the tutorial:
  - SPARQL Formalization. Marcelo Arenas, Claudio Gutierrez, Jorge Pérez. ESWC 2007 Tutorial. https://ai.wu.ac.at/~polleres/sparqltutorial/
- These slides were put together from the above publication for educational purposes only.
- Any changes made either reflect recent updates to SPARQL or are the result of minor editing, adaptation and extension for use in teaching.
- The author of these slides is solely responsible for any errrors they contain.
- No claims are made by the author of these slides with regards to any form of intellectual property rights wherever the above authors are acknowledged.
- The author of these slides is very grateful to the authors of the work above.

SPARQL Formalization

Marcelo Arenas, Claudio Gutierrez, Jorge Pérez

Department of Computer Science
Pontificia Universidad Católica de Chile
Universidad de Chile

Center for Web Research
http://www.cwr.cl

M. Arenas, C. Gutierrez, J. Pérez – SPARQL Formalization

SPARQL Datasets

- One of the interesting features of SPARQL is that a query may retrieve data from different sources.
- A SPARQL *dataset* is a set
  - $D = \{G_0, \langle u_1, G_1 \rangle, \langle u_2, G_2 \rangle, \ldots, \langle u_n, G_n \rangle\}$
  - where G_0 is the *default* graph,
  - $\langle u_i, G_i \rangle$ are *named* graphs,
  - the set of graph names in D is $\mathrm{name}(D) = \{u_1, u_2, \ldots, u_n\}$,
  - d^D is a function such that $d^D(u_i) = G_i$.

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The GRAPH Clause

- If u is a URI, ?X is a variable and P is a graph pattern, then
    - (u GRAPH P) is a graph pattern
    - (?X GRAPH P) is a graph pattern
- The GRAPH clause allows us to dynamically change the graph against which our pattern is evaluated.

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The GRAPH Operator: Semantics

- The *evaluation* of a general pattern P against a dataset D, denoted by $[\![P]\!]_D$, is the set $[\![P]\!]_{G0}$ where G_0 is the default graph in D.
- Given a dataset D, a named graph u, and a graph pattern P, the *evaluation* of a GRAPH graph pattern expression is defined as follows:
  - $[\![(u \text{ GRAPH } P)]\!]_G = [\![P]\!]_{d^D(u)}$
  - $[\![(?X \text{ GRAPH } P)]\!]_G = \bigcup_{u \in name(D)}([\![P]\!]_{d^D(u)} \bowtie \{\{?X \to u\}\})$

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The GRAPH Operator: Example [1]

D = { G0: { },
<tb, G1: { (R1, name, john),
(R1, email, J@ed.ex),
(R2, name, paul) }>,
<trs, G2: { (R4, name, mick),
(R4, email, M@ed.ex),
(R5, name, keith),
(R5, email, K@ed.ex) }> }

$[\![(\text{ trs GRAPH } \{(?X, \text{name}, ?N)\})]\!]_D$

$\rightsquigarrow [\![(\text{ trs GRAPH } \{(?X, \text{name}, ?N)\})]\!]_{G2}$

$\rightsquigarrow [\![\{(?X, \text{name}, ?N)\})]\!]_{G2} \rightsquigarrow$

	?X	?N
μ1	R4	mick
μ2	R5	keith

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The GRAPH Operator: Example [2]

same graph as previous slide

$[\![(\ ?G\ \text{GRAPH}\ \{(?X, \text{name}, ?N)\})]\!]_D$

$\rightsquigarrow [\![\{(?X, \text{name}, ?N)\})]\!]_{G1} \bowtie \{\{?G \rightarrow \text{tb}\ \}\} \cup$

$[\![\{(?X, \text{name}, ?N)\})]\!]_{G2} \bowtie \{\{?G \rightarrow \text{trs}\}\}$

$\rightsquigarrow$

	?X	?N
μ1	R1	john
μ2	R2	paul

$\bowtie \{\{?G \rightarrow \text{tb}\ \}\} \cup$

	?X	?N
μ1	R4	mick
μ2	R5	keith

$\bowtie \{\{?G \rightarrow \text{trs}\ \}\}$

$\rightsquigarrow$

?G	?X	?N
tb	R1	john
tb	R2	paul
trs	R4	mick
trs	R5	keith

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The SELECT Query Form [1]

- Up to this point we have concentrated in the body of a SPARQL query, i.e. in the graph pattern matching expression.
- A query can also post-process the mappings.
- The simplest post-processing operation is the selection of variables to appear in the result.

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The SELECT Query Form [2]

- A SELECT query is a tuple (W, P) where P is a graph pattern and W is a set of variables.
- Let $\mu|_W$ denote the restriction of μ to domain W.
- The *evaluation* of a SELECT against a dataset D is defined as:
  - $[\![\text{SELECT}(W,P)]\!]_D = \pi_W([\![P]\!]_D) = \{\mu|_W \mid \mu \in [\![P]\!]_D\}$

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The CONSTRUCT Query Form [1]

- A query can also output an RDF graph.
- The construction of the output graph is based on a template.
- A template is a set of triple patterns possibly with bnodes.
- A CONSTRUCT query is a tuple (T,P) where P is a graph pattern and T is a template.
- For example, let B be a bnode. Then an example template T is:
  - {(?X, name, ?Y), (?X, info, ?I), (?X, addr, B)}

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

The CONSTRUCT Query Form [2]

- The evaluation of a CONSTRUCT query (T,P) against a dataset D is defined as the result of the following procedure:

  I. for every $\mu \in [\![P]\!]_D$,create a template T_μ with fresh bnodes;

  II. take the union of $\mu(T_\mu)$ for every $\mu \in [\![P]\!]_{D;}$

  III. discard any invalid RDF triples, i.e., those that have

     I. any uninstantiated variable

     II. a bnode in predicate position

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

Blank Nodes in Graph Patterns [1]

- We allow now bnodes in triple patterns.
- Bnodes act as existentials scoped to the basic graph pattern.
- This leads to a natural extension of BGPs without bnodes.
- The algebra remains the same.

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

Blank Nodes in Graph Patterns [2]

- The *evaluation* of the BGP P with bnodes over the graph G, denoted by $[\![P]\!]_G$, is the set of all mappings μ such that:
  - dom(μ) is exactly the set of variables occurring in P,
  - there exists a function θ from the bnodes of P to G such that $\mu(\theta(P)) \subseteq G$.

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

Set v. Bag Semantics

- Like SQL, the default value of a SPARQL query is a bag, i.e., the semantics given so far is set-based, which essentially presumes an implicit DISTINCT query modifier that causes duplicates to be removed.
- It is not difficult, but out of scope here, to give the above a bag-based semantics (see Arenas et al. 2007 for a brief account).

from: SPARQL Formalization. M Arenas, C Gutierrez, J Pérez (2007)

Supplementary Material on Indices

Acknowledgements

- Some of these slides mostly contain text and examples that are most often taken *verbatim* from the online documentation of Microsoft's SQL Server DBMS at
  - https://msdn.microsoft.com/en-gb/library/ms190457.aspx
- These slides were put together from the above publication for educational purposes only.
- Any changes made either reflect recent updates to SPARQL or are the result of minor editing, adaptation and extension for use in teaching.
- The author of these slides is solely responsible for any errrors they contain.
- No claims are made by the author of these slides with regards to any form of intellectual property rights wherever the above authors are acknowledged.
- The author of these slides is very grateful to the authors of the work above.

Clustered and Unclustered Indexes

- An index is an on-disk structure associated with a table or view that speeds retrieval of rows from the table or view.

- An index contains keys built from one or more columns in the table or view.

- These keys are stored in a structure (B-tree) that enables the DBMS to find the row or rows associated with the key values quickly and efficiently.

- A table or view can contain clustered or unclustered indexes.

from: https://msdn.microsoft.com/en-gb/library/ms190457.aspx

Clustered and Unclustered Indexes

- Clustered indexes sort and store the data rows in the table or view based on their key values.
- These are the columns included in the index definition.
- There can be only one clustered index per table, because the data rows themselves can be sorted in only one order.
- The only time the data rows in a table are stored in sorted order is when the table contains a clustered index.
- When a table has a clustered index, the table is called a clustered table.
- If a table has no clustered index, its data rows are stored in an unordered structure called a heap.

from: https://msdn.microsoft.com/en-gb/library/ms190457.aspx

Clustered and Unclustered Indexes

- Unclustered indexes have a structure separate from the data rows.
- An unclustered index contains the unclustered index key values and each key value entry has a pointer to the data row that contains the key value.
- The pointer from an index row in a nonclustered index to a data row is called a row locator.
- The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table.
- For a heap, a row locator is a pointer to the row.
- For a clustered table, the row locator is the clustered index key.

from: https://msdn.microsoft.com/en-gb/library/ms190457.aspx

Clustered and Unclustered Indexes

- Both clustered and nonclustered indexes can be unique.
- This means no two rows can have the same value for the index key.
- Otherwise, the index is not unique and multiple rows can share the same key value.
- Indexes are automatically maintained for a table or view whenever the table data is modified.

from: https://msdn.microsoft.com/en-gb/library/ms190457.aspx