

Query Languages for the Semantic Web

- Introduction
 - What is SPARQL?
 - Functionality
- RDF
 - Triples
 - URI
- SPARQL Structure
 - Overall structure
 - Modifiers
- Usage of SPARQL queries

Query Languages for the Semantic Web

- **SPARQL Protocol And RDF Query Language**
 - W3C specification since Jan 15 2008
 - Query language for data from RDF documents
 - Extremely successful in practice
 - Update in progress (new SPARQL WG since Feb'13)

What is SPARQL?

- SPARQL is a programming language for querying ‘linked data’ stored on the web.
- It’s essentially a set of commands that allow you to find exactly the data you want.
- Learning SPARQL will allow you to query Wikidata, but also the countless other data sources offering a SPARQL query service
- Consists of two parts: query language and protocol.
 - extract information in the form of URIs, blank nodes, plain and typed literals.
 - extract RDF subgraphs.
 - construct new RDF graphs based on information in the queried graphs

SPARQL(cont.)

- SPARQL queries are not constrained to working within one database
- Queries can access multiple data stores (endpoints).
- Thus, Endpoint is a service that accepts SPARQL queries, and returns results, according to the details of the protocol.
- a SPARQL query retrieves information from a set of triples, and CONSTRUCT provides a set of triples

SPARQL Endpoints

- A SPARQL endpoint enables users (human or other) to query a knowledge base via the SPARQL language.
- A list of existing SPARQL end points:

<http://lists.w3.org/Archives/Public/public-lod/2010Oct/0001.html>

Example: **Dbpedia**

SPARQL Endpoint

<https://dbpedia.org/sparql>

Virtuoso SPARQL Query Editor

Default Data Set Name (Graph IRI)

Query Text

```
select distinct ?Concept where {[] a ?Concept} LIMIT 100
```

(Security restrictions of this server do not allow you to retrieve remote RDF data, see [details](#).)

Results Format:

(The CXML output is disabled, see [details](#))

Execution timeout:

milliseconds (values less than 1000 are ignored)

Options:



Strict checking of void variables

(The result can only be sent back to browser, not saved on the server, see [details](#))

Some Public SPARQL Endpoints

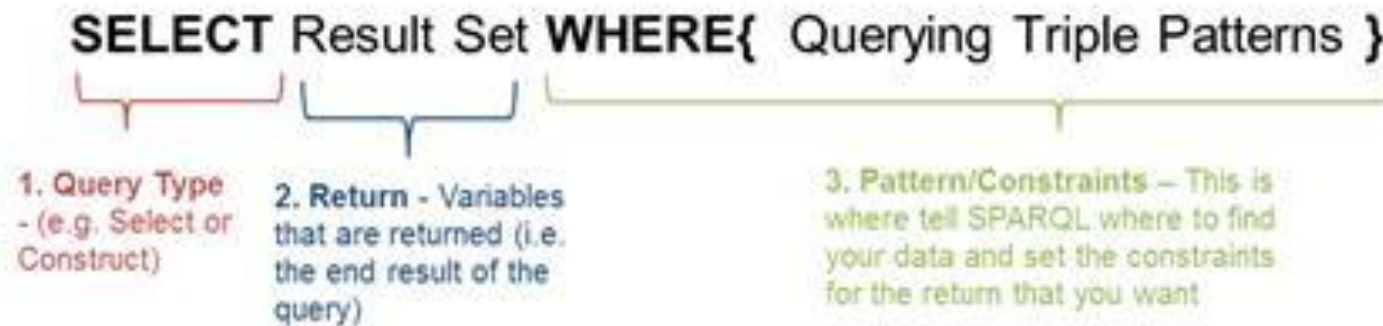
Name	URL	What's there?
SPARQLer	http://sparql.org/sparql.html	General-purpose query endpoint for Web-accessible data
DBPedia	http://dbpedia.org/sparql	Extensive RDF data from Wikipedia
DBLP	http://www4.wiwiss.fu-berlin.de/dblp/snorql/	Bibliographic data from computer science journals and conferences
LinkedMDB	http://data.linkedmdb.org/sparql	Films, actors, directors, writers, producers, etc.
World Factbook	http://www4.wiwiss.fu-berlin.de/factbook/snorql/	Country statistics from the CIA World Factbook
bio2rdf	http://bio2rdf.org/sparql	Bioinformatics data from around 40 public databases

SPARQL Fundamentals

- At its most basic, a SPARQL query is an RDF graph with variables.
- SPARQL queries are based on the concept of **graph pattern matching**.
- A basic SPARQL query is simply a **graph pattern** with some variables. Data that is returned via a query is said to **match the pattern**.

SPARQL

- Structure of a Sparql query



- Example

```
CONSTRUCT {?AssetID ?Accesses ?IPAddress} WHERE {{?IPAddress  
"@IPAddress-IPAddress@" ;}{?IPAddress ;}{?Accesses ;}{?AssetID  
;}{?AssetID ?Accesses ?IPAddress}}
```


SPARQL Fundamentals(cont.)

- Simple:

SELECT

some information

FROM

somewhere

WHERE

this match

AND

these constraints

USING

these vocabularies

SPARQL by Example

- Example:

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

SPARQL Fundamentals(cont.)

- A SPARQL query consists of three parts:
 - **Graph pattern**: Specifying a graph pattern, which is just RDF using some variables.
 - Pattern matching: optional, union, filtering, .
 - **Matching**: When RDF data matches a specific graph pattern.
Solution modifiers: projection, distinct, order, limit, offset, . . .
 - **Binding**: When a specific value in RDF is bound to a variable in a graph pattern.
 - Output part: construction of new triples, . . .

Simple query

- Data (vocabulary)

```
<http://example.org/book/book1>  
<http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

- Query

```
SELECT ?title  
WHERE  
{  
  <http://example.org/book/book1>  
    <http://purl.org/dc/elements/1.1/title> ?title .  
}
```

- Result

title
"SPARQL Tutorial"

General Sparql Query

Declare prefix
shortcuts
(*optional*)

PREFIX **foo:** <...>
PREFIX **bar:** <...>

...

Define the
dataset (*optional*)

SELECT ...

FROM <...>

FROM NAMED <...>

WHERE {

...

}

GROUP BY ...

HAVING ...

ORDER BY ...

LIMIT ...

OFFSET ...

Query result
clause

Query pattern

Query modifiers
(*optional*)

Types of SPARQL queries

- SPARQL has four query forms
 - SELECT: return all, or a subset of, the variable bound in a query pattern match
 - CONSTRUCT: returns an RDF graph constructed by substituting variables in a set of triple templates
 - ASK: returns a boolean indicating whether a query pattern matches or not
 - DESCRIBE: Find all statements in the dataset that provide information about the following resource(s)

SPARQL by Example: PREFIX

- **PREFIX keyword:** describes prefix declarations for abbreviating URIs.
- Create a prefix by using a string (foaf) to reference a part of the URI (<http://xmlns.com/foaf/0.1/>).

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

```
SELECT ?name
```

```
WHERE { ?person foaf:name ?name }
```

When you use the abbreviation (foaf:name), it appends the string after the colon (:) to the URI that is referenced by the prefix string.

SPARQL by Example: SELECT query

- **SELECT keyword:** serves very much the same function in SPARQL as in SQL → to return data matching some conditions.
- SELECT queries return data represented in a simple table, where each matching result is a row, and each column is the value for a specific variable.
- Result of our previous example would be a table with one column (**SELECT** ?name) and as many rows as match the query.

SPARQL by Example: SELECT query

- **FROM keyword:** defines the RDF dataset which is being queried.

```
FROM <http://example.com/dataset.rdf>
```

- **WHERE clause:** specifies the query graph pattern to be matched. This is the heart of the query.

```
WHERE
```

```
{
```

```
  ?x foaf:name ?name.
```

```
}
```

Examples(cont.)

→ ↺ ↻ ⬆ ⓘ dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=PREFIX+foaf%3A+<http%3A%2F%2Fxmlns.com%2Ffoaf%2F0.1%2F>%0D%0A+

person	name
http://dbpedia.org/resource/Academy_Award_(radio)	"Academy Award"@en
http://dbpedia.org/resource/Aggregation_(magazine)	"Aggregation"@en
http://dbpedia.org/resource/Archive_(Magnum_album)	"Archive"@en
http://dbpedia.org/resource/Archive_(The_Specials_album)	"Archive"@en
http://dbpedia.org/resource/Archive_(band)	"Archive"@en
http://dbpedia.org/resource/Area_(journal)	"Area"@en
http://dbpedia.org/resource/Area_(band)	"Area"@en
http://dbpedia.org/resource/Band_Iran	"Band"@en
http://dbpedia.org/resource/Band_Mureş	"Band"@en
http://dbpedia.org/resource/Onyria	"Band"@en
http://dbpedia.org/resource/Rede_Bandeirantes	"Band"@en
http://dbpedia.org/resource/Believers_(film)	"Believers"@en
http://dbpedia.org/resource/Believers_(song)	"Believers"@en
http://dbpedia.org/resource/Believers_(A._A._Bondy_album)	"Believers"@en
http://dbpedia.org/resource/Believers_(Babylon_5)	"Believers"@en
http://dbpedia.org/resource/Believers_(Don_McLean_album)	"Believers"@en
http://dbpedia.org/resource/Believers_(Mayday!_album)	"Believers"@en
http://dbpedia.org/resource/British_Comedy_Awards	"British Comedy Awards"@en
http://dbpedia.org/resource/Capital_Ward	"Capital"@en
http://dbpedia.org/resource/Capital_Department,_Salta	"Capital"@en
http://dbpedia.org/resource/Capital_(sidewheeler)	"Capital"@en
http://dbpedia.org/resource/Capital_(BBC_adaptation)	"Capital"@en
http://dbpedia.org/resource/Capital_(French_magazine)	"Capital"@en
http://dbpedia.org/resource/Capital_(German_magazine)	"Capital"@en
http://dbpedia.org/resource/Capital_(film)	"Capital"@en
http://dbpedia.org/resource/Capital_(newspaper)	"Capital"@en
http://dbpedia.org/resource/Capital_(radio_network)	"Capital"@en
http://dbpedia.org/resource/Capital_Ethiopia	"Capital"@en
http://dbpedia.org/resource/Capital_Cvmru	"Capital"@en

General: graph patterns

- The fundamental idea: use graph patterns
 - the pattern contains unbound symbols
 - by binding the symbols, subgraphs of the RDF graph are selected
 - if there is such a selection, the query returns bound resources

Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

Patterns on Named Graphs - patterns are matched against named graphs

Basic Graph Pattern

- Set of Triple Patterns
 - **Triple Pattern** – similar to an RDF Triple (subject, predicate, object), but any component can be a query variable; literal subjects are allowed

`?book dc:title ?title`

- Matching a triple pattern to a graph: bindings between variables and RDF Terms
- Matching of Basic Graph Patterns
 - A **Pattern Solution** of Graph Pattern GP on graph G is any substitution S such that S(GP) is a subgraph of G.

`SELECT ?x ?v WHERE { ?x ?x ?v }`

x	v
rdf:type	rdf:Property

`rdf:type rdf:type rdf:Property`

Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

Patterns on Named Graphs - patterns are matched against named graphs

Group Pattern

A set of graph patterns must all match

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

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

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

Group Pattern

Data

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
_:a foaf:name "Johnny Lee Outlaw" .  
_:a foaf:mbox <mailto:jlow@example.com> .  
_:b foaf:name "Peter Goodguy" .  
_:b foaf:mbox <mailto:peter@example.org> .
```

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

Query



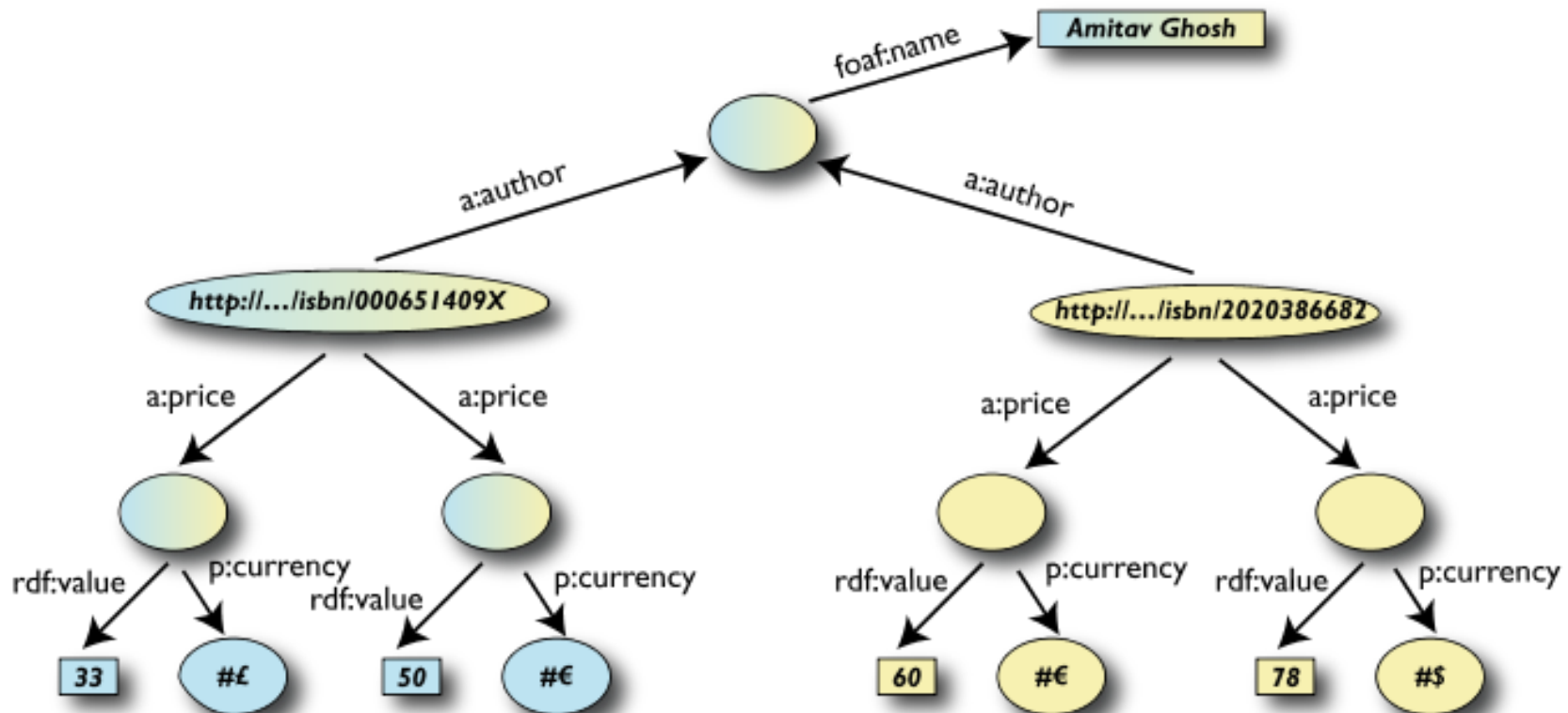
Group Graph Pattern
(set of graph patterns)
also!

Query Result

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Simple SPARQL example

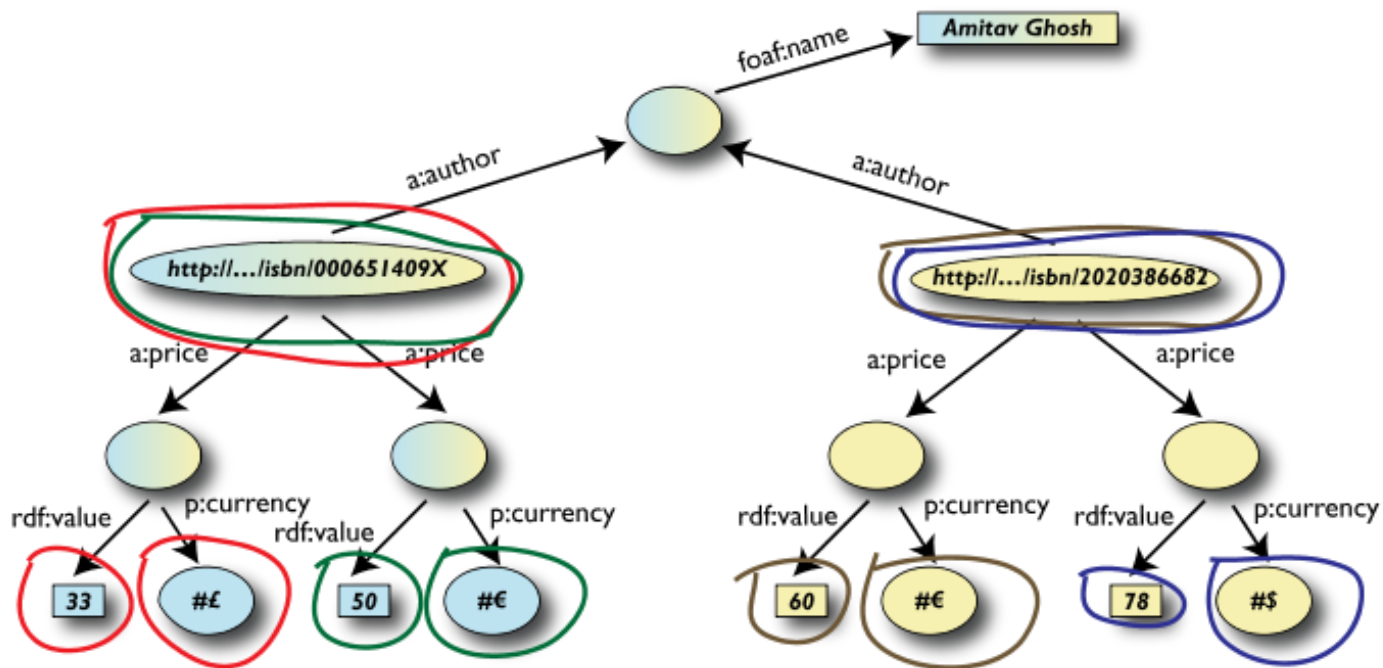
```
SELECT ?isbn ?price ?currency  
WHERE {?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```



Simple SPARQL example

```
SELECT ?isbn ?price ?currency {  
WHERE {?isbn a:price ?x. ?x rdf:value ?price. ?x p:currency ?currency.}
```

- Returns:
[[<..49X>,33,£], [<..49X>,50,€], [<..6682>,60,€],
[<..6682>,78,\$]]



Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

Patterns on Named Graphs - patterns are matched against named graphs

Value Constraints - FILTER

- **FILTERs** restrict the solutions of a graph pattern match according to a given expression.
- FILTERs eliminate any solutions that, when substituted into the expression, result in the **Boolean expression evaluated to false**.

```
@PREFIX dc:      <http://purl.org/dc/elements/1.1/> .
@PREFIX :        <http://example.org/book/> .
@PREFIX ns:      <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Value Constraints - FILTER

- FILTER functions
 - Can restrict on using arithmetic comparisons or logical expressions
 - Query

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
PREFIX ns:      <http://example.org/ns#>

SELECT ?title ?price
WHERE { ?x ns:price ?price .
       FILTER (?price < 30 )
       ?x dc:title ?title . }
```

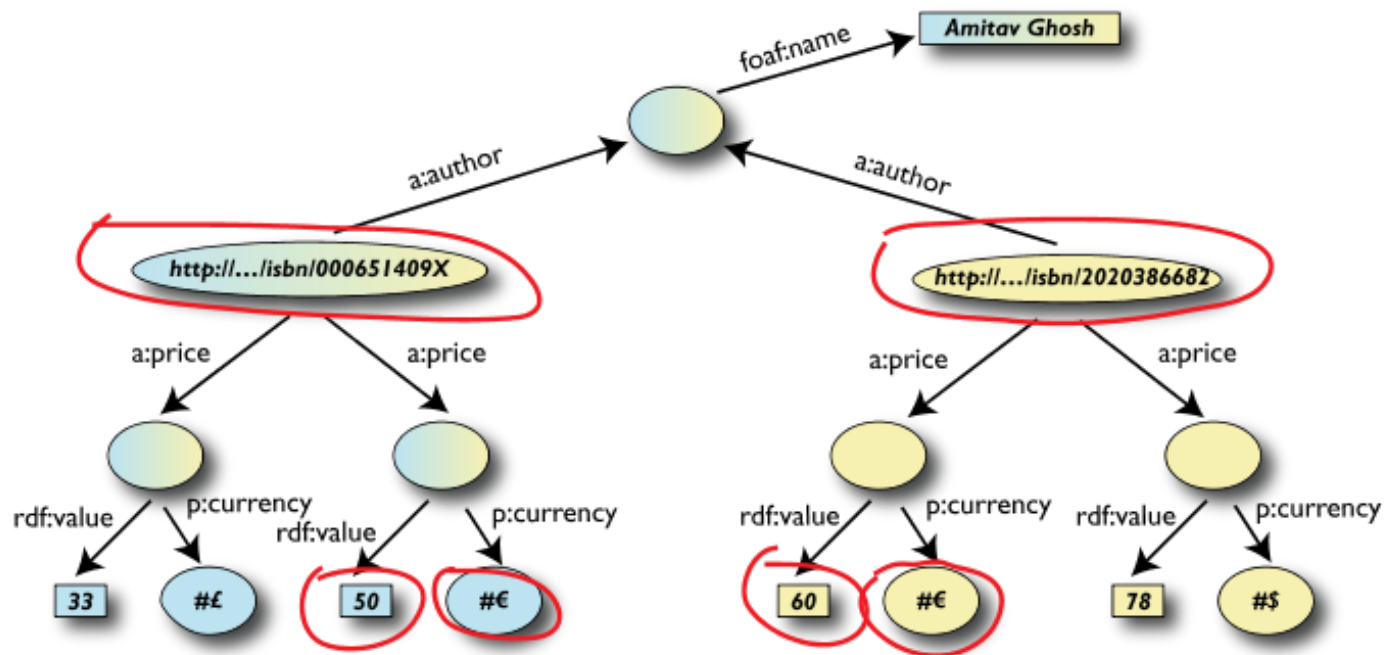
– Result

title	price
"The Semantic Web"	23

Pattern constraints Example

```
SELECT ?isbn ?price ?currency  
WHERE {  
  ?isbn a:price ?x.  
  ?x rdf:value ?price.  
  ?x p:currency ?currency.  
  FILTER(?currency == € )  
}
```

- Returns: [[<..409X>,50,€], [<..6682>,60,€]]



Value Constraints - FILTER

- Persons with younger siblings
 - { ?p1 :siblingOf ?p2 .
 - ?p1 :age ?a1 .
 - ?p2 :age ?a2 .
 - FILTER(?a2 < ?a1)}
- Persons that have both younger and older siblings
 - { ?p1 :siblingOf ?p2,p3 .
 - ?p1 :age ?a1 .
 - ?p2 :age ?a2 .
 - ?p3 :age ?a3 .
 - FILTER(?a2 < ?a1 && ?a3 > ?a1)}

Value Constraints - FILTER

- a person with two children
- $\{ ?p : \text{hasChild } ?c1, ?c2 . \text{FILTER}(?c1 \neq ?c2) \}$
- • A slight improvement
- \rightarrow Variables are now bound to different resources
- • But: we still have the Non-Unique Naming Assumption
- \rightarrow i.e., given that
- $\text{:Peter} : \text{hasChild} \text{:Julia} .$
- $\text{:Peter} : \text{hasChild} \text{:Stefan} .$
- we still cannot conclude that Peter has two children!
- • Furthermore, there is still the Open World Assumption
- \rightarrow i.e., Peter could also have more children

Value Constraints - FILTER

- FILTER functions
 - regex: match only plain literals with no language tag.
 - Query

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "^SPARQL")
      }
```

– Result

title
"SPARQL Tutorial"

Value Constraints - FILTER

- FILTER functions
 - regex: match only plain literals with no language tag.
 - Query

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "^SPARQL")
      }
```

– Result

title
"SPARQL Tutorial"

Value Constraints - FILTER

- FILTER functions
 - Regular expression matches may be made case-insensitive with the “i” flag.
 - Query

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "web", "i" )
      }
```

– Result

title
"The Semantic Web"

Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

Patterns on Named Graphs - patterns are matched against named graphs

Optional graph patterns - Optional matching

- If optional part does not match, it creates no bindings but **does not eliminate the solution.**
- For example, for the following data

```
@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" .
```

Optional matching (con.)

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
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"	

Optional matching(con.)

- Data

```
@PREFIX dc:      <http://purl.org/dc/elements/1.1/> .
@PREFIX :        <http://example.org/book/> .
@PREFIX ns:      <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Optional patterns may result in unbound variables

Optional matching (con.)

- Query

```
PREFIX dc:      <http://purl.org/dc/elements/1.1/>
PREFIX ns:      <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x dc:title ?title .
       OPTIONAL { ?x ns:price ?price .
                 FILTER (?price < 30) }
}
```

- Result

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

Multiple optional matching

- Data

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name          "Alice" .
_:a foaf:homepage      <http://work.example.org/alice/> .

_:b foaf:name          "Bob" .
_:b foaf:mbox          <mailto:bob@work.example> .
```

Multiple optional matching

- Query

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

- Result

name	mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

Graph Patterns

Basic Graph Pattern – set of *Triple Patterns*

Group Pattern - a set of graph patterns must all match

Value Constraints - restrict RDF terms in a solution

Optional Graph Patterns .- additional patterns may extend the solution

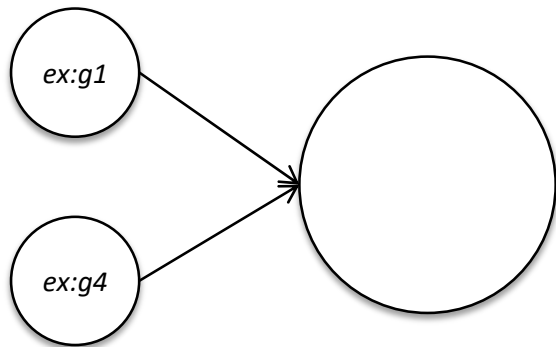
Patterns on Named Graphs - patterns are matched against named graphs

RDF Datasets

A SPARQL queries a *default graph* (normally) and zero or more *named graphs* (when inside a **GRAPH** clause).

Default graph

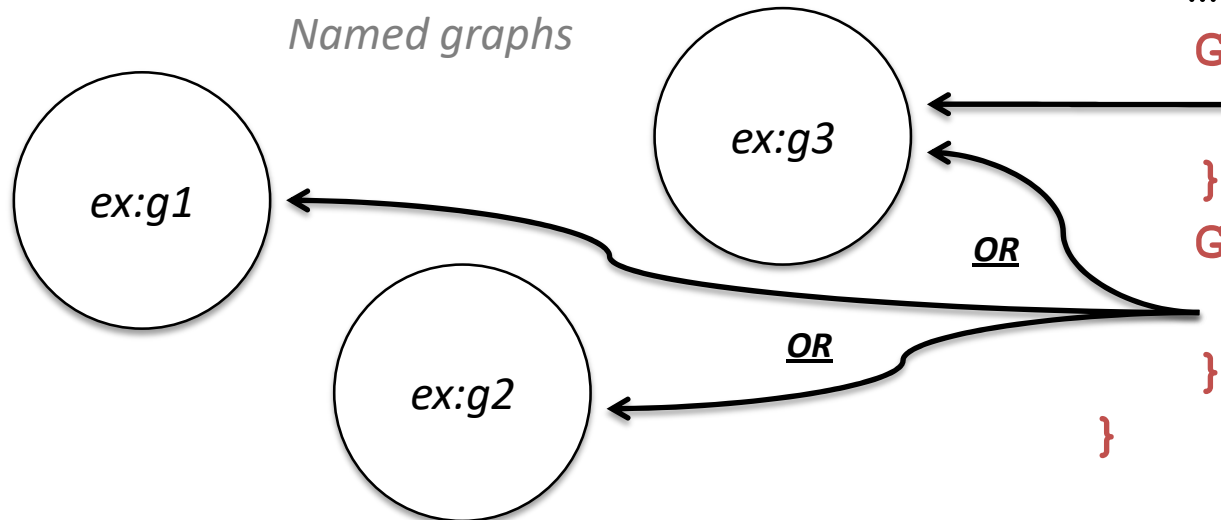
(the merge of zero or more graphs)



```
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 ?g {
... C ...
}
}
```

Named graphs



Specifying RDF datasets

- An RDF Dataset comprises
 - one graph, the default graph, which does not have a name, and
 - zero or more named graphs, where each named graph can be identified by an URIs.
- A SPARQL query may specify the dataset to be used for matching by using the FROM clause and the FROM NAMED clause.
- The FROM and FROM NAMED keywords allow a query to specify an RDF dataset by reference:
 - A default graph consisting of the RDF merge of the graphs referred to in the FROM clauses, and
 - A set of named graph pairs, one from each FROM NAMED clause
 - If there is no FROM clause, but one or more FROM NAMED clauses, then the dataset includes an empty graph for the default graph.

RDF Dataset- The Relationship between Named and Default Graphs

Default graph

```
@PREFIX dc: <http://purl.org/dc/elements/1.1/> .  
  
<http://example.org/bob> dc:publisher "Bob" .  
<http://example.org/alice> dc:publisher "Alice" .
```

Named graph: http://example.org/bob

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1> .  
  
_:a foaf:name "Bob" .  
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

Named graph: http://example.org/alice

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1> .  
  
_:a foaf:name "Alice" .  
_:a foaf:mbox <mailto:alice@work.example.org> .
```

FROM

- Data (stored at <http://example.org/foaf/aliceFoaf>)

```
@PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
```

Default Graph

```
_:a foaf:name "Alice" .
```

```
_:a foaf:mbox <mailto:alice@work.example> .
```

- Query

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

```
SELECT ?name
```

```
FROM <http://example.org/foaf/aliceFoaf>
```

```
WHERE { ?x foaf:name ?name }
```

- Result

name
"Alice"

Example: Google scholar dataset

Author's information	Paper's information
name	title
affiliation	authors (name, not links to authors' ids in Scholar)
domain	publication type (conference, journal, patent)
citations and publications time series	publication name
total number of citations (after 2008 and all)	publication date
h index (after 2008 and all)	publisher
i_{10} index (after 2008 and all)	citations time series
areas of interest	total number of citations

doi:10.1371/journal.pone.0114302.t001

Solution Modifiers

- **Solution Modifiers:** used to rearrange the query results.
- **ORDER BY:** used to sort the results (ASC or DESC).
- **LIMIT:** indicates the maximum number of rows that should be returned.
 - **Zero** will return no results;
 - **Value > the size of the result set**, then all rows will be returned.

ORDER BY DESC (?name)

LIMIT 10

Query Ordering

- A solution modifier contain also
 - Distinct: ensure solutions in the sequence are **unique**
 - Offset: control where the solutions start from in the overall sequence of solutions

ORDER BY

- ASC() – ascending
- DESC() - descending

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

```
SELECT ?name  
WHERE { ?x foaf:name ?name }  
ORDER BY ?name
```

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

```
SELECT ?name  
WHERE { ?x foaf:name ?name }  
ORDER BY DESC(?name)
```

OFFSET

- OFFSET tells from which number the solutions should be generated from
- position/index of the first returned mapping
- (useful only in combination with ORDER BY)
- LIMIT sets an upper bound on the number of solutions returned.

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

```
SELECT ?name  
WHERE { ?x foaf:name ?name }  
ORDER BY ?name  
LIMIT 5  
OFFSET 10
```

Returned Solutions are generated from the no. 11th solutions with no more than 5 solutions in total

Duplicate solutions

- Using DISTINCT or REDUCED to deal with duplicate solutions

```
@Prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:x foaf:name      "Alice" .
_:x foaf:mbox      <mailto:alice@example.com> .

_:y foaf:name      "Alice" .
_:y foaf:mbox      <mailto:asmith@example.com> .

_:z foaf:name      "Alice" .
_:z foaf:mbox      <mailto:alice.smith@example.com> .
```

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

```
SELECT ?name
WHERE { ?x foaf:name ?name }
```

name
"Alice"
"Alice"
"Alice"

Duplicate solutions

- Using DISTINCT/REDUCT to deal with duplicate solutions

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

```
SELECT DISTINCT ?name  
WHERE { ?x foaf:name ?name }
```

name
"Alice"

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

```
SELECT REDUCED ?name  
WHERE { ?x foaf:name ?name }
```

name
"Alice"

name
"Alice"
"Alice"

name
"Alice"
"Alice"
"Alice"

Combining Patterns

- Find the private and work phone number

{ ?p :privatePhone ?nr }

UNION

{ ?p :workPhone ?nr }

UNION creates a set union

?p = :peter, ?nr = 123;

?p = :john, ?nr = 234;

?p = :john, ?nr = 345;

Union

- Question: Which volcanos are located in the Italy or in Norway?

```
dbpedia:Mount_Etna rdf:type umbel-sc:Volcano ;  
  rdfs:label "Etna" p:location dbpedia:Italy .  
dbpedia:Mount_Baker rdf:type umbel-sc:Volcano ;  
  p:location dbpedia:United_States .  
dbpedia:Beerenberg rdf:type umbel-sc:Volcano ;  
  rdfs:label "Beerenberg" p:location dbpedia:Norway
```

```
SELECT ?v WHERE {  
  ?v rdf:type umbel-sc:Volcano ;  
  p:location ? . }
```

```
SELECT ?v WHERE {  
  ?v rdf:type umbel-sc:Volcano  
  { ?v p:location dbpedia:Italy }  
  UNION  
  { ?v p:location dbpedia:Norway }  
}
```


“Select full names of all scholars studying in University of Jyvaskyla and provide information about period of their study.”

PREFIX rdf: <<http://www.w3.org/1999/02/22-rdf-syntax-ns#>>

PREFIX onto: <<http://www.cs.jyu.fi/ai/vagan/ontologies/2016/Cars.owl#>>

PREFIX db: <<http://dbpedia.org/page/>>

SELECT **DISTINCT** ?name ?familyname ?begins ?ends

WHERE {

 ?x rdf:type onto:Scholar .

 ?x onto:hasFirstName ?name .

 ?x onto:hasFamilyName ?familyname .

 ?x onto:hasStudyRecord ?record .

 ?record onto:hasReferenceToSchool db:University_of_Jyvaskyla .

 ?record onto:hasValidityInterval ?interval .

 ?interval onto:hasStartingTime ?begins .

 ?interval onto:hasFinishingTime ?ends .

}

ORDER BY ?name

Query:

SPARQL query result

Redland Rasqal RDF Query Demonstration

by [Dave Beckett](#)

This is a demonstration of using [Rasqal](#) to perform queries in either of the SPARQL language results accessed via the [Redland Perl](#) language binding.

The query must be in SPARQL and needs a URI of some RDF content to query and run it. There are other example queries further down the page.

Some RDF content you can use could be my FOAF file at: <http://www.dajobe.org/foaf.rdf> or the W3C's RSS 1.0 file at: <http://www.w3.org/2000/08/w3c-synd/home.rss>

RDF content URIs

<http://www.cs.jyu.fi/ai/vagan/ontologies/2016/Cars.owl>

Query

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX onto: <http://www.cs.jyu.fi/ai/vagan/ontologies/2016/Cars.owl#>
PREFIX db: <http://dbpedia.org/page/>
SELECT DISTINCT ?name ?familyname ?begins ?ends
WHERE {
  ?x rdf:type onto:Scholar .
  ?x onto:hasFirstName ?name .
  ?x onto:hasFamilyName ?familyname .
  ?x onto:hasStudyRecord ?record .
  ?record onto:hasReferenceToSchool db:University_of_Jyvaskyla .
```



Results

Variable bindings result format

Count	name	familyname	begins	ends
1	Vitalii^^<http://www.w3.org/2001/XMLSchema#string>	Tsybulko^^<http://www.w3.org/2001/XMLSchema#string>	2016-09-01^^<http://www.w3.org/2001/XMLSchema#date>	2016-10-10^^<http://www.w3.org/2001/XMLSchema#date>

Found 1 result

“Select full name, birthday, city and county of residence with its capital of a person who owns pet Tom”

PREFIX tutorial: <<http://www.cs.jyu.fi/ai/vagan/ontologies/2019/tutorial.owl#>>

SELECT DISTINCT ?firstname ?familyname ?birthday ?country ?city ?capital

WHERE {

?someone tutorial:hasPet tutorial:Tom .

?someone tutorial:hasFirstName ?firstname .

?someone tutorial:hasFamilyName ?familyname .

?someone tutorial:hasBirthday ?birthday .

?someone tutorial:hasCityOfResidence ?city .

?city tutorial:isCityOf ?country .

?country tutorial:hasCapital ?capital

}

Query:

SPARQL query result

Redland Rasqal RDF Query Demonstration

by [Dave Beckett](#)

RDF content URIs

<http://www.cs.jyu.fi/ai/vagan/ontologies/2019/tutorial.owl>

Query

```
PREFIX tutorial: <http://www.cs.jyu.fi/ai/vagan/ontologies/2019/tutorial.owl#>
SELECT DISTINCT ?firstname ?familyname ?birthday ?capital
WHERE {
  ?someone tutorial:hasPet tutorial:Tom .
  ?someone tutorial:hasFirstName ?firstname .
  ?someone tutorial:hasFamilyName ?familyname .
  ?someone tutorial:hasBirthday ?birthday .
  ?someone tutorial:hasCityOfResidence ?city .
  ?city tutorial:isCityOf ?country .
  ?country tutorial:hasCapital ?capital
}
```

in SPARQL with an ☐ XML result ☐ JSON result

Run Query

Clear Query



Results

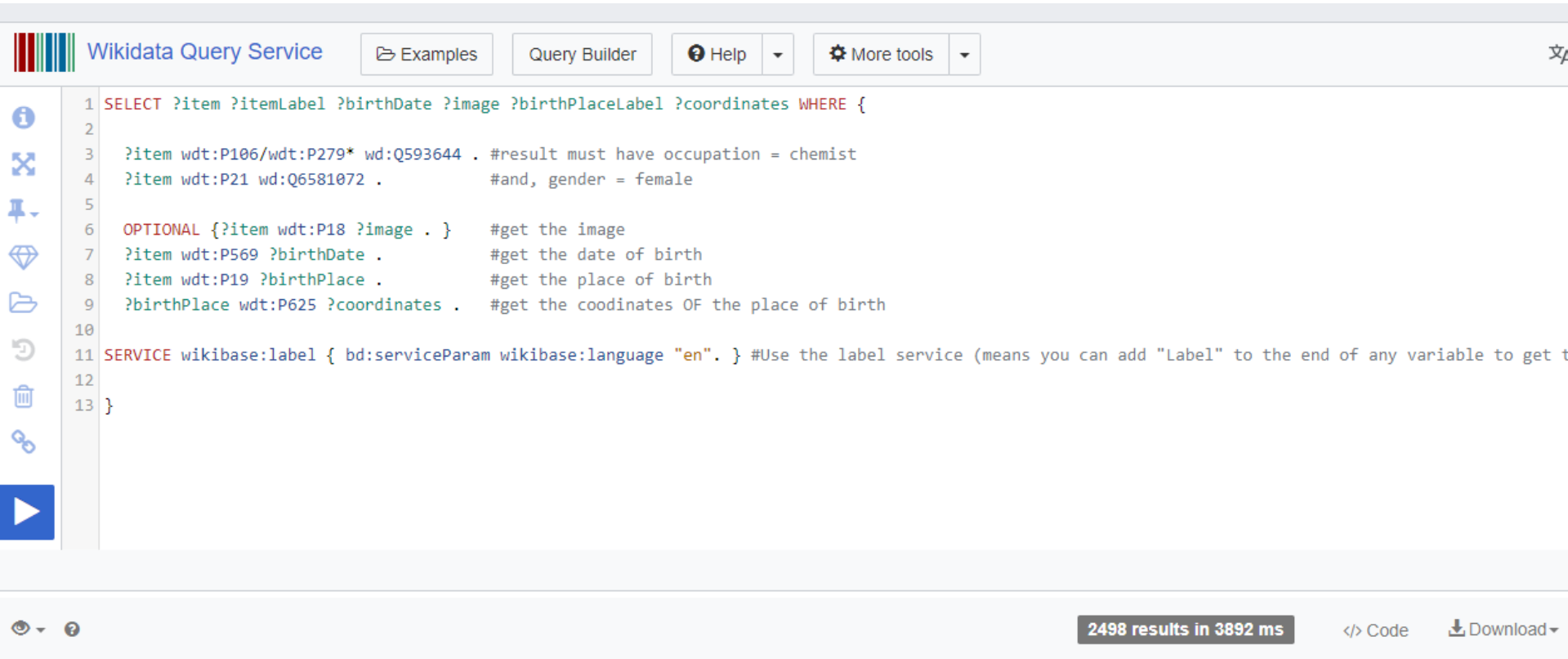
Variable bindings result format

Count	firstname	familyname	birthday	capital
1	Vagan^^<http://www.w3.org/2001/XMLSchema#string>	Terziyan^^<http://www.w3.org/2001/XMLSchema#string>	1958-12-27T00:00:00^^<http://www.w3.org/2001/XMLSchema#dateTime>	http://dbpedia.org/page/Helsinki

Found 1 result

Wikidata Query Example(cont.)

E.g. Show me a list of all Female Chemists, along with their date of birth, place of birth, and the map coordinates of their place of birth.



The screenshot shows the Wikidata Query Service interface. At the top, there's a header with the Wikidata logo, the text "Wikidata Query Service", and navigation buttons: "Examples", "Query Builder", "Help", and "More tools". Below the header is a text area containing a SPARQL query. The query is as follows:

```
1 SELECT ?item ?itemLabel ?birthDate ?image ?birthPlaceLabel ?coordinates WHERE {
2
3   ?item wdt:P106/wdt:P279* wd:Q593644 . #result must have occupation = chemist
4   ?item wdt:P21 wd:Q6581072 .          #and, gender = female
5
6   OPTIONAL { ?item wdt:P18 ?image . }  #get the image
7   ?item wdt:P569 ?birthDate .          #get the date of birth
8   ?item wdt:P19 ?birthPlace .          #get the place of birth
9   ?birthPlace wdt:P625 ?coordinates .  #get the coordinates OF the place of birth
10
11 SERVICE wikibase:label { bd:serviceParam wikibase:language "en". } #Use the label service (means you can add "Label" to the end of any variable to get t
12
13 }
```

At the bottom right of the interface, there's a status bar showing "2498 results in 3892 ms", a "Code" button, and a "Download" button.

Wikidata Query Example(cont.)

item	itemLabel	birthDate	image	birthPlaceLabel	coordinates
Q7186 Marie Curie		7 November 1867	commons:Marie Curie c. 1920s.jpg	Warsaw	Point(21.01111111 52.23)
Q7195 Elizabeth Ann Nalley		1 January 2000	commons:Elizabeth Ann Nalley Innovation Day 2006 crop.jpg	Catron	Point(-89.7047 36.6122)
Q7212 Pauline Newman		20 June 1927	commons:Circuit judge newman.png	New York City	Point(-74.0 40.7)
Q7229 Mary-Anne Paulze Lavoisier		20 January 1758	commons:David - Portrait of Monsieur Lavoisier and His Wife (cropped).jpg	Montbrison	Point(4.065277777 45.6075)
Q7236 Lucy Weston Pickett		19 January 1904	commons:LucyWestonPickett1925.png	Beverly	Point(-70.88 42.55833333)
Q7249 Agnes Pockels		14 February 1862	commons:Agnes Pockels.jpg	Venice	Point(12.33194444 45.43972222)
Q7249 Agnes Pockels		14 February 1862	commons:Agnes Pockels ca1922.jpg	Venice	Point(12.33194444 45.43972222)
Q7255 Darshan Ranganathan		4 June 1941		Delhi	Point(77.21666666 28.66666666)
Q7267 Hazel Alden Reason		1 April 1901		London	Point(-0.1275 51.50722222)
		3 December			Point(-71.48333333)

2- Ask query

- ASK allows for yes/no queries:
- – e.g., are there persons with siblings?
- ASK {?p :hasSibling ?s . }
- Often faster than SELECT queries
- The answer is true or false

false means: there are no matching sub graphs

ASK – example

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
_:a foaf:name "Alice" .  
_:a foaf:homepage <http://work.example.org/alice/> .  
_:b foaf:name "Bob" .  
_:b foaf:mbox <mailto:bob@work.example> .
```

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

```
ASK { ?x foaf:name "Alice" }
```

TRUE.

3- CONSTRUCT

- CONSTRUCT: returns an RDF graph constructed by substituting variables in a set of triple templates
- A SPARQL query processor returns the data for a CONSTRUCT query as actual triple, not as a formatted report
- It allow to specify template of new information based on the graph pattern at you find in old information
- Predicates could be combined

Constructing an Output Graph

•Data:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:givenname "Alice" .

_:a foaf:family_name "Hacker" .

_:b foaf:firstname "Bob" .

_:b foaf:surname "Hacker" .

•Query:

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

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

CONSTRUCT

```
{
  ?x vcard:N _:v .
  _:v vcard:givenName ?gname .
  _:v vcard:familyName ?fname
}
```

WHERE

```
{
  { ?x foaf:firstname ?gname } UNION { ?x foaf:givenname ?gname } .
  { ?x foaf:surname ?fname } UNION { ?x foaf:family_name ?fname } .
}
```

Result:

@prefix vcard:

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

_:v1 vcard:N _:x .

_:x vcard:givenName "Alice" .

_:x vcard:familyName "Hacker" .

_:v2 vcard:N _:z .

_:z vcard:givenName "Bob" .

_:z vcard:familyName "Hacker" .

Usage of SPARQL queries to explore the content of a given dataset

- Examples:

What properties are used in the data?

- ```
SELECT DISTINCT ?p WHERE {
 ?s ?p ?o
}
```

What classes are used?

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
SELECT DISTINCT ?t WHERE {
 ?s rdf:type ?t
}
```

# Sparql examples(cont.)

- **List a few example instances of a particular class**

```
PREFIX rdf: <http://www.w3.org/1999/02/22rdfsyntaxns#>
```

```
PREFIX ssn: <http://purl.oclc.org/NET/ssnx/ssn#>
```

```
SELECT ?s WHERE {
 ?s rdf:type ssn:SensingDevice
}
```

```
LIMIT 10
```

# A query to ask about no more than 10 names and associated locations

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

select ?name ?latitude ?longitude
from <http://3roundstones.com/dave/me.rdf>
from <http://semanticweb.org/wiki/Special:ExportRDF/Michael_Hausenblas>
where {
 ?person foaf:name ?name ;
 foaf:based_near ?near .
 ?near pos:lat ?latitude ;
 pos:long ?longitude .
}
LIMIT 10
```

**Namespace prefixes.**

**Requesting three fields be retrieved.**

**Results will be retrieved from two sources.!**

**Criteria described in the form of a triple pattern.**

**Items preceded by ? represent variables in the results.**

**Only the first 10 results will be returned.**

# Summary :

## SPARQL Structure

- A **SPARQL query** consists of the following points in order:
  - **Prefix declarations** for abbreviating URIs
  - **Database definition** stating which RDF graph are being queried.
  - A **result clause** , identifying the what to return from the query
  - The **query pattern**, specifying what to query
  - Query **modifiers**

# Summary :

## SPARQL Structure(cont.)

- **Modifiers**

- Some common modifiers to use in SPARQL

- **DISTINCT**

- Modifier that eliminates duplicate query results.

- **LIMIT**

- Solution modifier that limits the number of results returned

- Other two modifiers ORDER and OFFSET

- **FILTER**

- Uses Boolean conditions to filter out unwanted results

# Lecture Question

- **Data:**

@prefix ex: <http://www.snee.com/ns/demo#> .

|         |               |             |
|---------|---------------|-------------|
| ex:jane | ex:hasParent  | ex:gene .   |
| ex:jane | ex:hasSibling | ex:mary .   |
| ex:gene | ex:hasParent  | ex:pat      |
| ex:gene | ex:gender     | ex:female . |
| ex:joan | ex:hasParent  | ex:pat .    |
| ex:joan | ex:gender     | ex:female . |
| ex:pat  | ex:gender     | ex:male .   |
| ex:mike | ex:hasParent  | ex:joan .   |



# Lecture Question

- **Query:**

```
PREFIX ex: http://www.snee.com/ns/demo#
CONSTRUCT { ?p ex:unknownProperty ?g . }
WHERE {
 ?p ex:hasParent ?parent .
 ?parent ex:hasParent ?g .
 ?g ex:gender ex:male .
}
```

# Lecture Question

1. What is the relation between Jane and Mike?
2. What is the relation between Joan and Gene?

## Answer:

```
@prefix ans:
```

```
<http://www.snee.com/ns/demo#> .
```

```
ans:jane ans:hasGrandfather ans:pat .
```

```
ans:mike ans:hasGrandfather ans:pat .
```