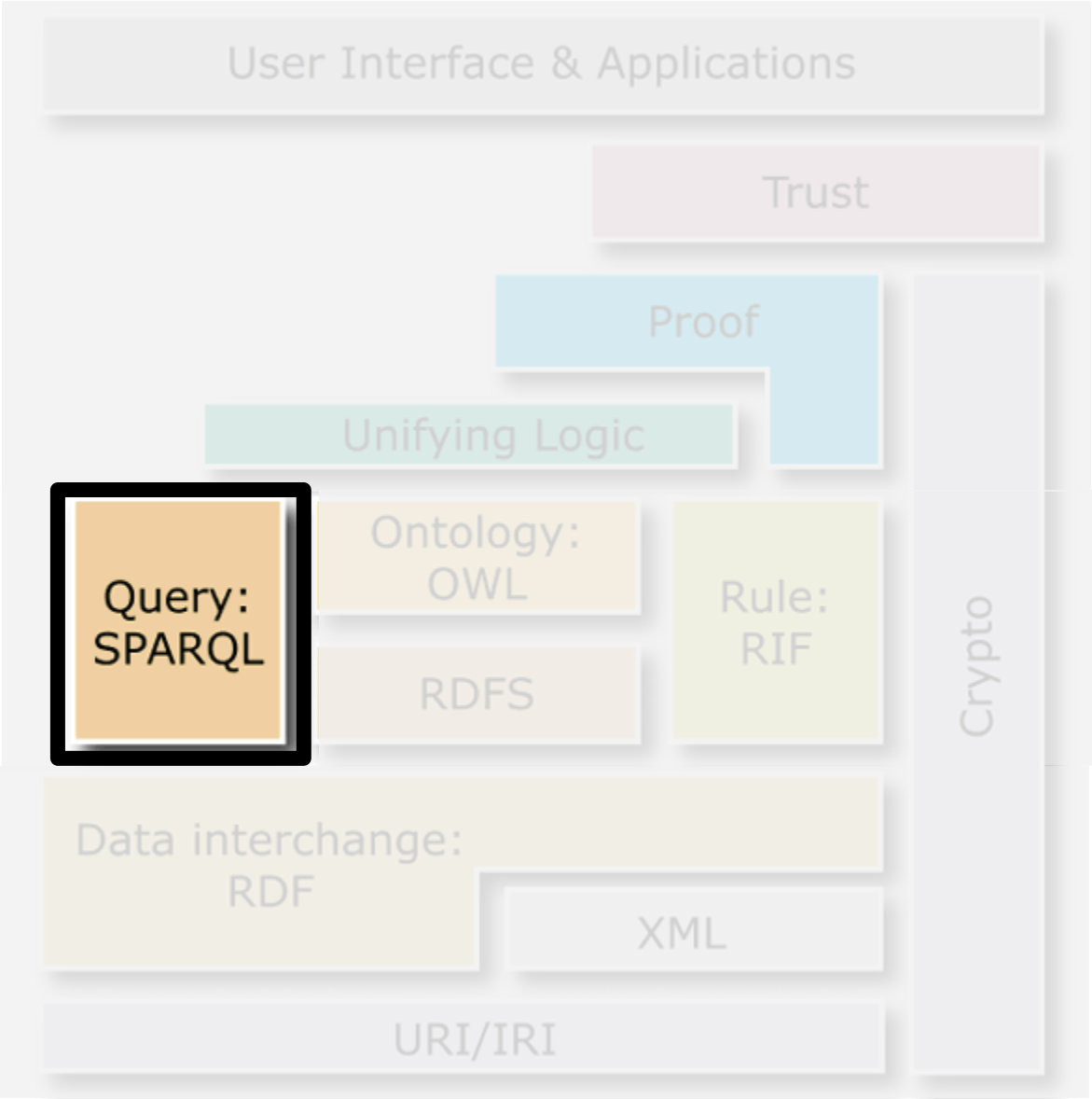
SPARQL

- These slides give an introduction to the language
- Queries against DBpedia SPARQL endpoint to exemplify

- These slides are partially based on “SPARQL By Example: The Cheat Sheet
by Lee Feigenbaum *<lee@cambridgesemantics.com>*

Query: SPARQL



What is SPARQL?

- SPARQL
  - is the query language of the Semantic Web
  - stays for **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage

A Query Language ...:

- *Find names and websites of contributors to PlanetRDF*:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?website
FROM <http://planetrdf.com/bloggers.rdf>
WHERE { ?person foaf:weblog ?website ;
                foaf:name ?name .
      ?website a foaf:Document }
```

... and a Protocol.

`http://.../qps?`

- `query-lang=http://www.w3.org/TR/rdf-sparql-query/ &graph-id=http://planetrdf.com/bloggers.rdf &query=PREFIX foaf: <http://xmlns.com/foaf/0.1/...`

Query: What do we do?

Input

- One (or many) RDF **Graphs**
- Typically a **SELECT** query with a query pattern to match in the graph

Output

- A subgraph (triples) that match the query pattern

Output formats

- JSON
- RDF/XML
- Turtle
- .....

Query Patterns

- SPARQL has a Turtle-like syntax for URIs, QNames, literals, …
- Variables in triple patterns
  - ?var
- Simple
  - `?s rdf:type <http://dbpedia.org/ontology/Boxer>.`
- complex

```
?s rdf:type <http://dbpedia.org/ontology/Boxer> .
?s dbpprop:birthDate “1980-01-01”^^xsd:date.
  – ?s foaf:nick “Mike”@en .
```

Simple query

“Get all boxer URIs from DBpedia”

```
SELECT  ?s
WHERE {
   ?s rdf:type <http://dbpedia.org/ontology/Boxer>.
}
```

?s: Variable to bind found subjects to.
All found triples (subgraph) have to match the given query pattern.

s
http://dbpedia.org/resource/Jack_Johnson_%28boxer%29
http://dbpedia.org/resource/Julio_C%C3%A9sar_Ch%C3%A1vez
http://dbpedia.org/resource/Ada_V%C3%A9lez
http://dbpedia.org/resource/Oscar_De_La_Hoya
http://dbpedia.org/resource/Jes%C3%BAs_Gonz%C3%A1lez
http://dbpedia.org/resource/Laura_Serrano
http://dbpedia.org/resource/Fernando_Vargas
http://dbpedia.org/resource/Michael_Carbajal
http://dbpedia.org/resource/Archie_Moore
http://dbpedia.org/resource/Jos%C3%A9_Torres
http://dbpedia.org/resource/Santos_Laciar
http://dbpedia.org/resource/Max_Schmeling
http://dbpedia.org/resource/Jake_LaMotta

…

…..

Exercise

“Get all football players from Dbpedia”
(SoccerPlayer)

Simple query

“Get all boxer URIs and their birthday and birthplace”

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?s ?bd ?bp
WHERE {
 ?s rdf:type <http://dbpedia.org/ontology/Boxer>.
 ?s dbo:birthPlace ?bp .
 ?s dbo:birthDate ?bd .
}
```

Simple query

"Get all boxer URIs and their birthday from Dbpedia which are from America"

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX :    <http://dbpedia.org/resource/>

SELECT ?s ?bd ?bp
WHERE {
 ?s  rdf:type       dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?s  dbo:birthDate  ?bd .
 ?bp dbo:country    :United_States .
}
```

Exercise 2

A) Get all Thai Boxers with nickname „Fierce Tiger“@en

B) Get the nicknames of all the Thai Boxers

C) Get all Russian novelists from Dbpedia.

D) Get the Russian novelists, and the title of their works

Anatomy of a Query

Declare prefix shortcuts (*optional*)

Define the dataset (*optional*)

Query result clause

Query pattern

Query modifiers (*optional*)

```
PREFIX foo: <…>
PREFIX bar: <…>
…
SELECT …
FROM <…>
FROM NAMED <…>
WHERE {
    …
}
GROUP BY …
HAVING …
ORDER BY …
LIMIT …
OFFSET …
BINDINGS …
```

Source: *Lee Feigenbaum lee@cambridgesemantics.com*, *http://www.cambridgesemantics.com/2008/09/sparql-by-example/*

Common Prefixes

prefix...	...stands for
rdf:	http://xmlns.com/foaf/0.1/
rdfs:	http://www.w3.org/2000/01/rdf-schema#
owl:	http://www.w3.org/2002/07/owl#
xsd:	http://www.w3.org/2001/XMLSchema#
dc:	http://purl.org/dc/elements/1.1/
foaf:	http://xmlns.com/foaf/0.1/

More common prefixes at http://prefix.cc

Source: *Lee Feigenbaum lee@cambridgesemantics.com*, *http://www.cambridgesemantics.com/2008/09/sparql-by-example/*

What does this query do??

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX :    <http://dbpedia.org/resource/>

SELECT ?s ?bd ?bp
WHERE {
 ?s  rdf:type dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?s  dbo:birthDate  ?bd .
}
```

```
ORDER BY DESC(?bd)
LIMIT 15
OFFSET 30
```

Aggregates *(SPARQL 1.1)*

1. Partition results into groups based on the expression(s) in the **GROUP BY** clause
2. Evaluate projections and aggregate functions in **SELECT** clause to get one

?key	?val	?other1
1	4	…
1	4	…
2	5	…
2	4	…
2	10	…
2	2	…
2	1	…
3	3	…

?key	?sum_of_val
1	8
2	22
3	3

?key	?sum_of_val
1	8
3	3

SPARQL 1.1 includes: **COUNT**, **SUM**, **AVG**, **MIN**, **MAX**, **SAMPLE**, **GROUP_CONCAT**

3. Filter aggregated

What does this do???

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT count(?s) as ?num_boxers ?country
WHERE {
 ?s  rdf:type dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?bp dbo:country ?country .
}

GROUP by ?country
```

Show one randomly pick example!

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT count(?s) as ?num_boxers ?country sample(?s)
WHERE {
 ?s  rdf:type dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?bp dbo:country ?country .
}

GROUP by ?country
```

Put all terms of the group into one line!

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT count(?s) as ?num_boxers ?country GROUP_CONCAT(?s;
   SEPARATOR=", ")
WHERE {
 ?s  rdf:type dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?bp dbo:country ?country .
}

GROUP by ?country
```

What does this do???

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX : <http://dbpedia.org/resource/>

SELECT count(?s) as ?num_boxers ?country
WHERE {
 ?s  rdf:type dbo:Boxer .
 ?s  dbo:birthPlace ?bp .
 ?bp dbo:country ?country .
}

GROUP by ?country
```

Exercise 3

A) Get the number of works written by Russian novelists

B) All writers with more than 5 works

What does this do???

What happens without UNION?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX :    <http://dbpedia.org/resource/>

SELECT ?s ?bd ?bp

FROM <http://dbpedia.org>

WHERE {
 { ?s  rdf:type dbo:Writer . } UNION { ?s  rdf:type
dbo:Boxer . }
 ?s  dbo:birthPlace ?bp .
 ?s  dbo:birthDate  ?bd .
 ?bp dbo:country :United_States .
}  LIMIT 10
```

The OPTIONAL keyword: similar to a left join in a database, pattern binds if it is found, but it does not have to match to be in the result graph (can be „NULL“).

What does this do???

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX :    <http://dbpedia.org/resource/>

SELECT ?s ?bd ?bp ?dpl

WHERE {
 ?s  rdf:type dbo:Writer .
 ?s  dbo:birthPlace ?bp .
 ?s  dbo:birthDate  ?bd .
 OPTIONAL {?s dbo:deathPlace ?dpl }
 ?bp dbo:country :Russia .
}
LIMIT 100
```

Exercise 4

A) Get the names, and the nicknames (if they exist) and their genre (dbo:genre) if existing, for Russian writers.

B) Get the first 200 writers from Germany and Austria

Combining SPARQL Graph Patterns

*Consider **A** and **B** as graph patterns.*

A Basic Graph Pattern – one or more triple patterns

A . B

Conjunction. Join together the results of solving A and B by matching the values of any variables in common.

Optional Graph Patterns

A OPTIONAL { B }

Left join. Join together the results of solving A and B by matching the values of any variables in common, if possible. Keep all solutions from A whether or not there's a matching solution in B

Source: *Lee Feigenbaum lee@cambridgesemantics.com*,
http://www.cambridgesemantics.com/2008/09/sparql-by-example/

Nuts & Bolts

URIs

Write full URIs:

```
<http://this.is.a/full/URI/written#out>
```

Abbreviate URIs with prefixes:

```
PREFIX foo: <http://this.is.a/URI/prefix#>
… foo:bar …
```

 http://this.is.a/URI/prefix#bar

Shortcuts:

a  rdf:type

Literals

Plain literals:

```
"a plain literal"
```

Plain literal with language tag:

```
"bonjour"@fr
```

Typed literal:

```
"13"^^xsd:integer
```

Shortcuts:

true  "true"^^xsd:boolean
3  "3"^^xsd:integer
4.2  "4.2"^^xsd:decimal

Variables

Variables:

```
?var1, ?anotherVar, ?and_one_more
```

Comments

Comments:

```
# Comments start with a '#'
# continue to the end of the line
```

Triple Patterns

Match an exact RDF triple:

```
ex:myWidget ex:partNumber "XY24Z1" .
```

Match one variable:

```
?person foaf:name "Lee Feigenbaum" .
```

Match multiple variables:

```
conf:SemTech2009 ?property ?value .
```

Datatypes and literals

- Turtle style.
- New `PREFIX xsd:`
- `"20" != "20"^^xsd:int`

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?b
WHERE {
   ?s a dbo:Boxer;
      dbo:birthDate ?b;
      dbp:losses "20"^^xsd:int .
}
```

Remarks:

- Abbreviated triples (with „;“)
- „`a`“ as short form for „`rdf:type`“

SPARQL Filters

- *SPARQL* **FILTER***s eliminate solutions that do not cause an expression to evaluate to true.*
- *Place* **FILTER***s in a query inline within a basic graph pattern*

```
A . B . FILTER ( …expr… )
```

Category	Functions / Operators	Examples
Logical	`!`, `&&`, `||`, `=`, `!=`, `<`, `<=`, `>`, `>=`	`?hasPermit || ?age < 25`
Math	`+`, `-`, `*`, `/`	`?decimal * 10 > ?minPercent`
SPARQL tests	`isURI`, `isBlank`, `isLiteral`, `bound`	`isURI(?person) || !bound(?person)`
Accessors	`str`, `lang`, `datatype`	`lang(?title) = "en"`
Miscellaneous	`sameTerm`, `langMatches`, `regex`	`regex(?ssn, "\\d{3}-\\d{2}-\\d{4}")`

What is this query doing?

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?b
WHERE {
   ?s a dbo:Boxer;
      dbo:birthDate ?b;
      dbp:losses "20"^^xsd:int .

  FILTER (REGEX(STR(?b),"[0-9]{4}-[0-9]{2}-[0-9]{2}")).
}
```

Try this query, what’s happing?

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?label
WHERE {
   ?s a dbo:Writer;
      rdfs:label ?label.
}
LIMIT 50
```

Get only English labels

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?label
WHERE {
   ?s a dbo:Writer;
      rdfs:label ?label.
     FILTER(lang(?label) = 'en')
}
LIMIT 50
```

Get only English labels

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?label
WHERE {
   ?s a dbo:Writer;
      rdfs:label ?label.
     FILTER(lang(?label) = 'en')
}
LIMIT 50
```

What does this do??

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?label ?dd
WHERE {
   ?s a dbo:Writer;
      rdfs:label ?label.
     FILTER(lang(?label) = 'en')

   OPTIONAL {?s dbo:deathDate ?dd}
   FILTER (!BOUND(?dd))

}
LIMIT 50
```

Does the same!

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT ?s ?label
WHERE {
   ?s a dbo:Writer;
      rdfs:label ?label.
     FILTER(lang(?label) = 'en')

   FILTER NOT EXISTS {?s dbo:deathDate ?dd .}


}
LIMIT 50
```

What does this do??

```
SELECT count(DISTINCT ?s) AS ?num_writers
WHERE {
   ?s    a    dbo:Writer .

   FILTER NOT EXISTS {?s dbo:deathDate ?dd .}
}
```

Exercise 5

A) Get all Russian writers, and their names in Russian.

B) Same as above, but show their Russian names if available, and their description in German (de).

C) How many Russian writers are there.

D) Advanced: how many Russian writers by gender.

4 Types of SPARQL Queries

SELECT queries

Project out specific variables and expressions:

```
SELECT ?c ?cap (1000 * ?people AS ?pop)
```

Project out all variables:

```
SELECT *
```

Project out distinct combinations only:

```
SELECT DISTINCT ?country
```

Results in a table of values (in XML or JSON):

?c	?cap	?pop
ex:France	ex:Paris	63,500,000
ex:Canada	ex:Ottawa	32,900,000
ex:Italy	ex:Rome	58,900,000

CONSTRUCT queries

Construct RDF triples/graphs:

```
CONSTRUCT {
   ?country a ex:HolidayDestination ;
       ex:arrive_at ?capital ;
       ex:population ?population .
}
```

Results in RDF triples (in any RDF serialization):

```
ex:France a ex:HolidayDestination ;
    ex:arrive_at ex:Paris ;
    ex:population 635000000 .
ex:Canada a ex:HolidayDestination ;
    ex:arrive_at ex:Ottawa ;
    ex:population 329000000 .
```

ASK queries

Ask whether or not there are any matches:

```
ASK
```

Result is either "true" or "false" (in XML or JSON):

```
true, false
```

DESCRIBE queries

Describe the resources matched by the given variables:

```
DESCRIBE ?country
```

Result is RDF triples (in any RDF serialization) :

```
ex:France a geo:Country ;
  ex:continent geo:Europe ;
  ex:flag <http://…/flag-france.png> ;
  …
```

Construct ..

```
PREFIX dbpedia-owl:
  <http://dbpedia.org/ontology/>
PREFIX dbpprop:
  <http://dbpedia.org/property/>
PREFIX dbres: <http://dbpedia.org/
  resource/>
PREFIX xsd:
  <http://www.w3.org/2001/XMLSchem
  a#>

CONSTRUCT {
```

When selecting N3/Turtle as output format

```
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .
@prefix dbpprop:        <http://dbpedia.org/property/> .
@prefix dbpedia:        <http://dbpedia.org/resource/> .

dbpedia:Laura_Serrano           dbpprop:birthDate       "1967-10-20"^^xsd:date .
dbpedia:Michael_Carbajal        dbpprop:birthDate       "1967-09-17"^^xsd:date .
dbpedia:Archie_Moore            dbpprop:birthDate       "1916-12-13"^^xsd:date .
<http://dbpedia.org/resource/Jos%C3%A9_Torres>          dbpprop:birthDate       "1936-05-03"^^xsd:date .
<http://dbpedia.org/resource/Carlos_Z%C3%A1rate_Serna>  dbpprop:birthDate       "1951-05-23"^^xsd:date .
<http://dbpedia.org/resource/James_Douglas_%28boxer%29> dbpprop:birthDate       "1960-04-07"^^xsd:date .
dbpedia:Bob_Fitzsimmons dbpprop:birthDate       "1863-05-26"^^xsd:date .
dbpedia:Clifford_Etienne        dbpprop:birthDate       "1972-03-09"^^xsd:date .
dbpedia:Michael_Moorer          dbpprop:birthDate       "1967-11-12"^^xsd:date .
```

Combining SPARQL Graph Patterns

*Consider **A** and **B** as graph patterns.*

Either-or Graph Patterns

{ A } UNION { B }

Disjunction. Include both the results of solving A and the results of solving B.

"Subtracted" Graph Patterns (SPARQL 1.1)

A MINUS { B }

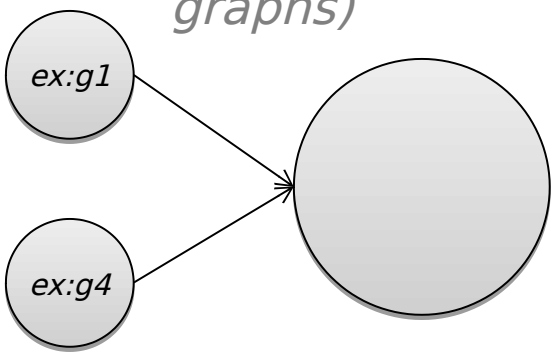
Negation. Solve A. Solve B. Include only those results from solving A that are *not compatible* with any of the results from B.

Source: *Lee Feigenbaum lee@cambridgesemantics.com*,
http://www.cambridgesemantics.com/2008/09/sparql-by-example/

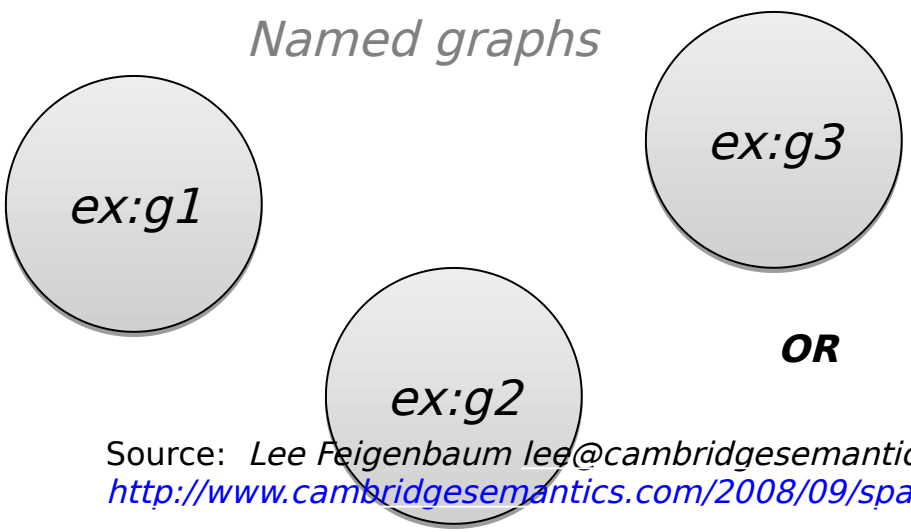
RDF Datasets

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)



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 ?g {
    … C …
  }
}
```

OR

OR

Source: *Lee Feigenbaum lee@cambridgesemantics.com*, *http://www.cambridgesemantics.com/2008/09/sparql-by-example/*

SPARQL Over HTTP (the SPARQL Protocol)

`http://host.domain.com/sparql/endpoint?`*<parameters>*

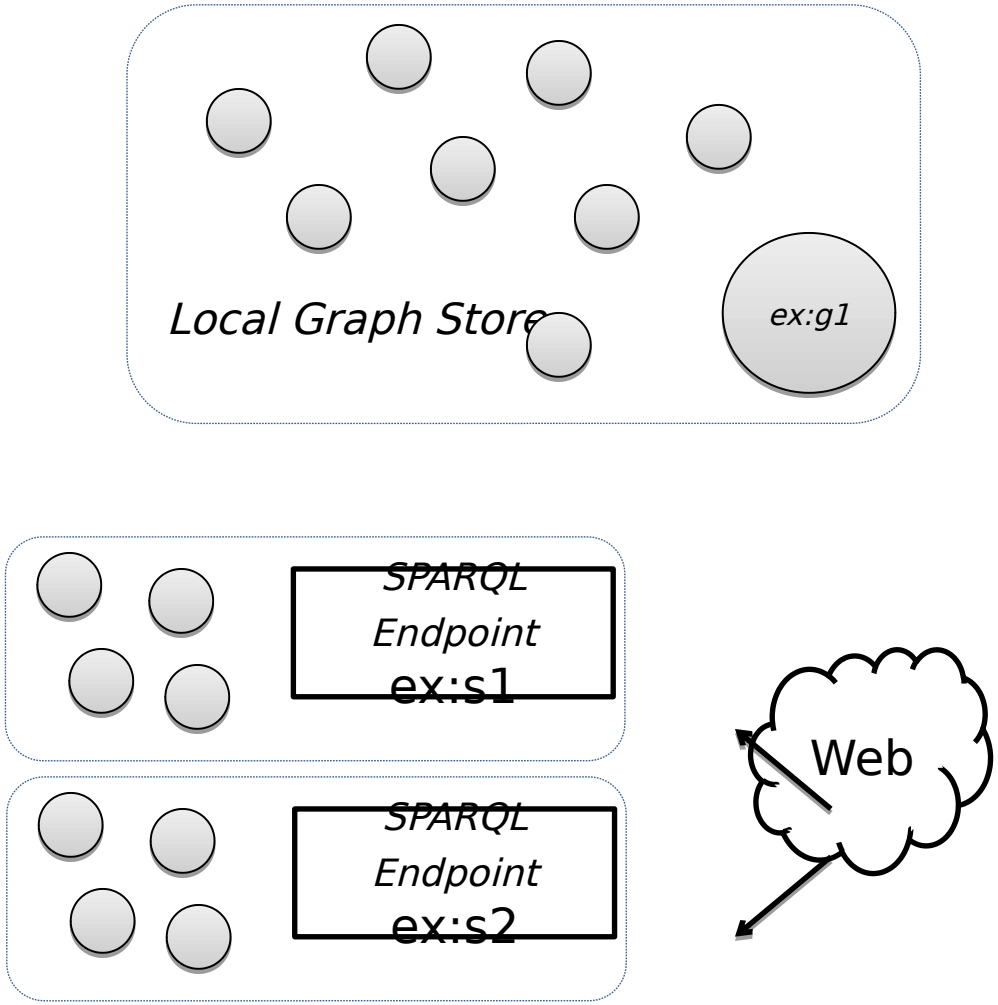
where *<parameters>* can include:

`query=`*<encoded query string>*
e.g. `SELECT+*%0DWHERE+{…`
`default-graph-uri=`*<encoded graph URI>*
e.g. `http%3A%2F%2Fexmaple.com%2Ffoo…`
n.b. zero of more occurrences of `default-graph-uri`
`named-graph-uri=`*<encoded graph URI>*
e.g. `http%3A%2F%2Fexmaple.com%2Fbar…`
n.b. zero of more occurrences of `named-graph-uri`

HTTP `GET` or `POST`. Graphs given in the protocol override graphs given in the query.

Source: *Lee Feigenbaum lee@cambridgesemantics.com*,
http://www.cambridgesemantics.com/2008/09/sparql-by-example/

Federated Query *(SPARQL 1.1)*
(informational only)




```
PREFIX ex: <…>
SELECT …
FROM ex:g1
WHERE {
   … A …
   SERVICE ex:s1 {
     … B …
   }
   SERVICE ex:s2 {
     … C …
   }
}
```

Source: *Lee Feigenbaum lee@cambridgesemantics.com,*
http://www.cambridgesemantics.com/2008/09/sparql-

SPARQL 1.1 Update

SPARQL Update Language Statements
`INSERT DATA { triples }`
`DELETE DATA {triples}`
`[ DELETE { template } ] [ INSERT { template } ] WHERE { pattern }`
`LOAD <uri> [ INTO GRAPH <uri> ]`
`CLEAR GRAPH <uri>`
`CREATE GRAPH <uri>`
`DROP GRAPH <uri>`

[…] denotes optional parts of SPARQL 1.1 Update syntax

Last Exercise

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .


<http://example.org/bobsBlog> dc:publisher <http://example.org/bob> .
<http://example.org/alicesBlog> dc:publisher <http://example.org/alice> .
<http://example.org/bob>   foaf:name "Bob"@en .
<http://example.org/bob>   foaf:age 25 .
<http://example.org/bob>   foaf:mbox <mailto:bob@oldcorp.example.org> .
<http://example.org/alice> foaf:name "Alice"@en  .
<http://example.org/alice> foaf:mbox "alice@wunderland.org" .
```

Questions:

- What is the email adress, name and age of the person who publishes <http://example.org/bobsBlog> ? The age is optional.
- How many distinct email adresses (foaf:mbox) are mentioned in the dataset?
- Select name and mailbox of persons older than 20 years where the mailbox is a URI (not a string). Sort the results by name, descending.

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 Resources

- The SPARQL Specification
  - http://www.w3.org/TR/rdf-sparql-query/
- SPARQL implementations
  - http://esw.w3.org/topic/SparqlImplementations
- SPARQL endpoints
  - http://esw.w3.org/topic/SparqlEndpoints
- SPARQL Frequently Asked Questions
  - http://www.thefigtrees.net/lee/sw/sparql-faq
- SPARQL Working Group
  - http://www.w3.org/2009/sparql/wiki/
- Common SPARQL extensions
  - http://esw.w3.org/topic/SPARQL/Extensions

Source: *Lee Feigenbaum lee@cambridgesemantics.com, http://www.cambridgesemantics.com/2008/09/sparql-by-example/*