



# SPARQL By Example: The Cheat Sheet

Accompanies slides at:

<http://www.cambridgesemantics.com/2008/09/sparql-by-example/>

Comments & questions to:

*Lee Feigenbaum <lee@cambridgesemantics.com>*

*VP Technology & Standards, Cambridge Semantics*

*Co-chair, W3C SPARQL Working Group*

# Conventions

Red text means:

*“This is a core part of the SPARQL syntax or language.”*

Blue text means:

*“This is an example of query-specific text or values that might go into a SPARQL query.”*

# 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 .
```

# 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>

# Anatomy of a Query

Declare prefix  
shortcuts  
(*optional*)

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

Define the  
dataset (*optional*)

{  
  **SELECT** ...  
  **FROM** <...>  
  **FROM NAMED** <...>  
  **WHERE** {

← Query result  
clause

  ...  
  }  
  **GROUP BY** ...  
  **HAVING** ...  
  **ORDER BY** ...  
  **LIMIT** ...  
  **OFFSET** ...  
  **BINDINGS** ...

← Query pattern

Query modifiers  
(*optional*)

→ {  
  **GROUP BY** ...  
  **HAVING** ...  
  **ORDER BY** ...  
  **LIMIT** ...  
  **OFFSET** ...  
  **BINDINGS** ...

# 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> ;  
    ...
```

# 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

# 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.



# SPARQL Subqueries (*SPARQL 1.1*)

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

```
A .  
{  
  SELECT ...  
  WHERE {  
    B  
  }  
}  
C .
```

⇒ Join the results of the subquery with the results of solving A and C.

# SPARQL Filters

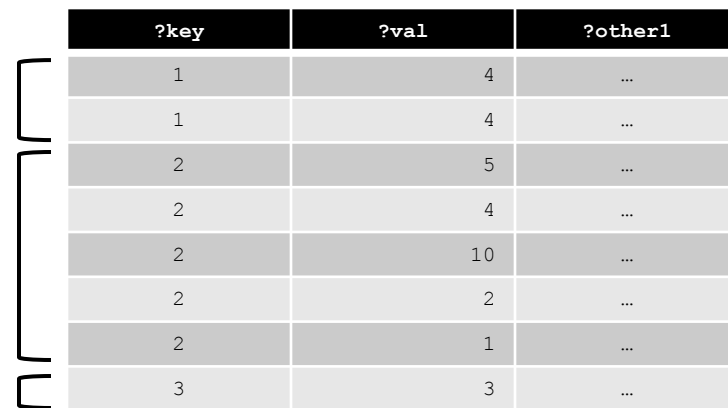
- SPARQL **FILTER**s eliminate solutions that do not cause an expression to evaluate to true.
- Place **FILTER**s in a query inline within a basic graph pattern

**A . B . FILTER ( ...expr... )**


Category	Functions / Operators	Examples
Logical	!, &&,   , =, !=, <, <=, >, >=	?hasPermit    ?age < 25
Math	+, -, *, /	?decimal * 10 > ?minPercent
Existence (SPARQL 1.1)	EXISTS, NOT EXISTS	NOT EXISTS { ?p foaf:mbox ?email }
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}")

# 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 result per group
3. Filter aggregated results via the **HAVING** clause



?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**

# Property Paths (*SPARQL 1.1*)

- Property paths allow triple patterns to match arbitrary-length paths through a graph
- Predicates are combined with regular-expression-like operators:

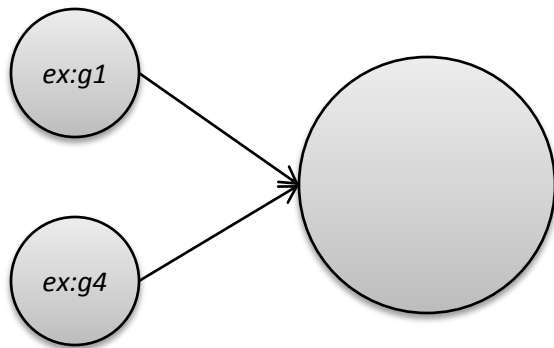
Construct	Meaning
<code>path1/path2</code>	Forwards path ( <code>path1</code> followed by <code>path2</code> )
<code>^path1</code>	Backwards path (object to subject)
<code>path1 path2</code>	Either <code>path1</code> or <code>path2</code>
<code>path1*</code>	<code>path1</code> , repeated zero or more times
<code>path1+</code>	<code>path1</code> , repeated one or more times
<code>path1?</code>	<code>path1</code> , optionally
<code>path1{m,n}</code>	At least <code>m</code> and no more than <code>n</code> occurrences of <code>path1</code>
<code>path1{n}</code>	Exactly <code>n</code> occurrences of <code>path1</code>
<code>path1{m,}</code>	At least <code>m</code> occurrences of <code>path1</code>
<code>path1{,n}</code>	At most <code>n</code> occurrences of <code>path1</code>

# RDF Datasets

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

*Default graph*

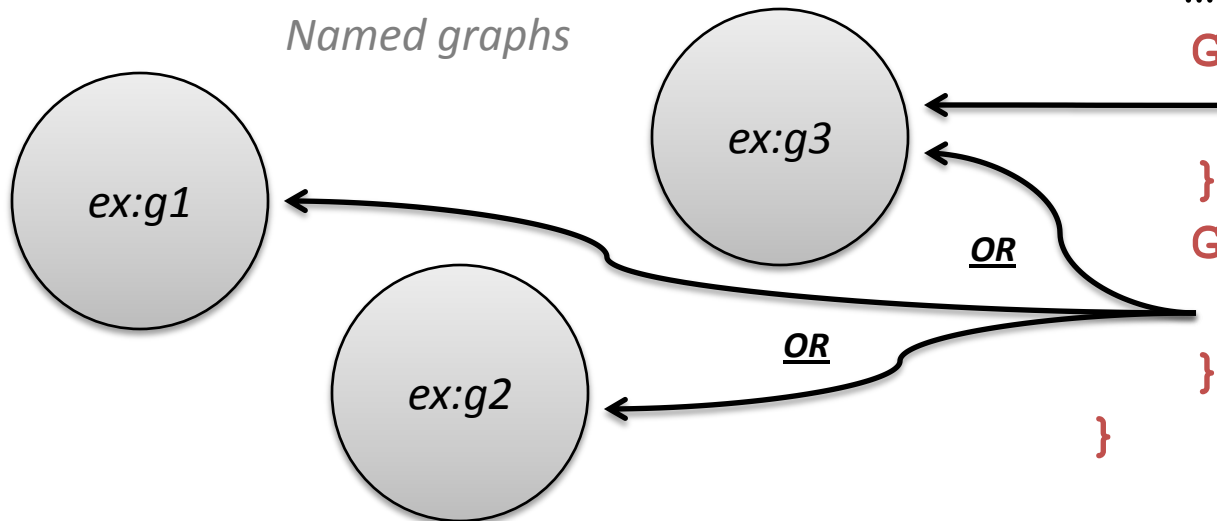
(the merge of zero or more graphs)



```
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*



# 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%2Fexample.com%2Ffoo...`

n.b. zero or more occurrences of `default-graph-uri`

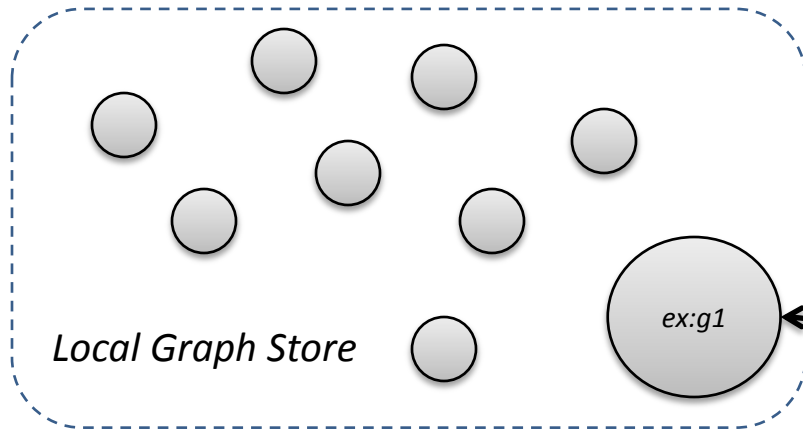
`named-graph-uri=<encoded graph URI>`

e.g. `http%3A%2F%2Fexample.com%2Fbar...`

n.b. zero or more occurrences of `named-graph-uri`

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

# Federated Query (*SPARQL 1.1*)



**PREFIX** **ex:** <...>

**SELECT** ...

**FROM** **ex:g1**

**WHERE** {

... **A** ...

**SERVICE** **ex:s1** {

... **B** ...

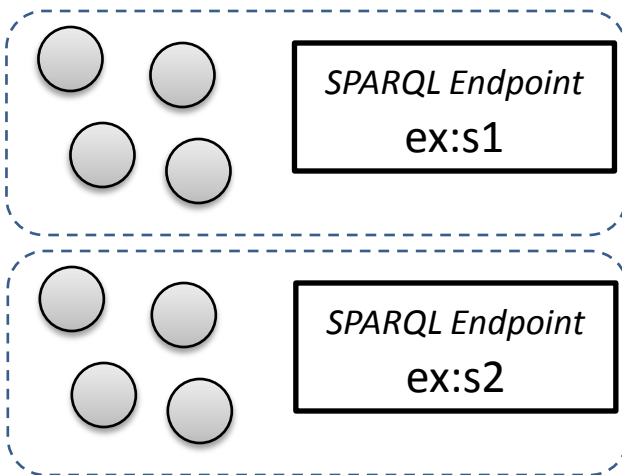
}

**SERVICE** **ex:s2** {

... **C** ...

}

}



# 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>
```

```
CREATAE GRAPH <uri>
```

```
DROP GRAPH <uri>
```

*[ ... ] denotes optional parts of SPARQL 1.1 Update syntax*



# Some Public SPARQL Endpoints

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