INF3580 – Semantic Technologies – Spring 2011

Lecture 13: More SPARQL

Kjetil Kjernsmo

26th April 2011

Department of Informatics

University of Oslo

Today's Plan

1. Reminder: SPARQL

2. RDF Datasets

3. Functions and Operators

4. SPARQL 1.1

5. New Semantic Web Community

Oblig 4

- Oblig 4 is corrected.
- Results will be available in Devilry today.
- Due date for second attempt extended with one week: 09.05.2011 23:59.
- Students who did not handin first attempt are encouraged to try again!

Outline

1. Reminder: SPARQL
2. RDF Datasets
3. Functions and Operators
4. SPARQL 1.1
   - Update language
   - Property paths
   - Aggregates and negation
5. New Semantic Web Community

Query with Basic Graph Pattern

Titles of publications by people called “Martin Giese”

```
  SELECT ?title WHERE {
      ?mg foaf:name "Martin Giese" .
      ?pub dc:creator ?mg .
      ?pub dc:title ?title .
  }
```

Query with Basic Graph Pattern

Titles of publications by people called “Martin Giese”

```
SELECT ?title WHERE {
    ?mg foaf:name "Martin Giese" .
    ?pub dc:creator ?mg .
    ?pub dc:title ?title .
}
```

`PREFIX` declarations omitted from all examples, use `http://prefix.cc` to find!

Query with Basic Graph Pattern

Titles of publications by people called “Martin Giese”

```
SELECT ?title WHERE {
    ?mg foaf:name "Martin Giese" .
    ?pub dc:creator ?mg .
    ?pub dc:title ?title .
}
```

PREFIX declarations omitted from all examples, use http://prefix.cc to find!

Answer:

?title
"Incremental Closure of Free Variable Tableaux."^^xsd:string
"The KeY system 1.0 (Deduction Component)."^^xsd:string
"The KeY System: Integrating Object-Oriented Design and Formal Methods."^^xsd
"The KeY Approach: Integrating Object Oriented Design and Formal Verification."^^
"Saturation Up to Redundancy for Tableau and Sequent Calculi."^^xsd:strin
...

SPARQL Query with blank nodes

Names of people who have published with “Martin Giese”

```
SELECT DISTINCT ?name WHERE {
    _:mg foaf:name "Martin Giese" .
    _:pub dc:creator _:mg .
    _:pub dc:creator _:other .
    _:other foaf:name ?name.
}
```

The same with blank node syntax

```
SELECT DISTINCT ?name WHERE {
    [ dc:creator [foaf:name "Martin Giese"] ,
                 [foaf:name ?name]
    ]
}
```

Filters

E.g.

Places with more than a million inhabitants

```
{
  ?x a dbpedia-owl:Place ;
     dbpprop:population ?pop .
  FILTER (?pop > 1000000)
}
```

Optional Patterns

A match can leave some variables *unbound*.
E.g.

Places and their population, and Norwegian abstract if it exists

```
{
  ?x a dbpedia-owl:Place ;
     dbpprop:population ?pop .
  OPTIONAL {
    ?x dbpprop:abstract ?abs .
    FILTER (lang(?abs) = "no")
  }
}
```

Matching Alternatives

A UNION pattern matches if any of some alternatives matches
E.g.

Find the book and its author regardless of predicate

```
{
  { ?book dc:creator ?author . }
  UNION
  { ?book foaf:maker ?author . }
  UNION
  { ?author foaf:made ?book . }
}
```

Four Types of Queries

SELECT Compute table of bindings for variables

```
SELECT ?a ?b WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

Four Types of Queries

SELECT Compute table of bindings for variables

```
SELECT ?a ?b WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

CONSTRUCT Use bindings to construct a new RDF graph

```
CONSTRUCT {
  ?a foaf:knows ?b .
} WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

Four Types of Queries

SELECT Compute table of bindings for variables

```
SELECT ?a ?b WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

CONSTRUCT Use bindings to construct a new RDF graph

```
CONSTRUCT {
  ?a foaf:knows ?b .
} WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

ASK Answer (yes/no) whether there is ≥ 1 match

Four Types of Queries

SELECT Compute table of bindings for variables

```
SELECT ?a ?b WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

CONSTRUCT Use bindings to construct a new RDF graph

```
CONSTRUCT {
  ?a foaf:knows ?b .
} WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

ASK Answer (yes/no) whether there is ≥ 1 match

DESCRIBE Answer available information about matching resources

Solution Modifiers

- Patterns generate an unordered collection of solutions
- SELECT treats solutions as a sequence (solution sequence)
- *Sequence modifiers* can modify the solution sequence:
  - Order
  - Projection
  - Distinct
  - Reduce
  - Offset
  - Limit
- Applied in this order.

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*
- I.e. SPARQL queries can be submitted to a database server that sends back the results

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*
- I.e. SPARQL queries can be submitted to a database server that sends back the results
- Uses HTTP to submit URL-encoded queries to server
  `GET /sparql/?query=...  HTTP/1.1`

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*
- I.e. SPARQL queries can be submitted to a database server that sends back the results
- Uses HTTP to submit URL-encoded queries to server
  ```
  GET /sparql/?query=...  HTTP/1.1
  ```
- Actually defined via W3C Web Services, see

  `http://www.w3.org/TR/rdf-sparql-protocol/`

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*
- I.e. SPARQL queries can be submitted to a database server that sends back the results
- Uses HTTP to submit URL-encoded queries to server
  `GET /sparql/?query=...  HTTP/1.1`
- Actually defined via W3C Web Services, see

  `http://www.w3.org/TR/rdf-sparql-protocol/`
- For `SELECT` queries you get a XML or JSON result set, see

  `http://www.w3.org/TR/rdf-sparql-XMLres/`
  `http://www.w3.org/TR/rdf-sparql-json-res/`

SPARQL on the 'Net

- Many sites (DBLP, dbpedia, dbtunes,. . .) publish *SPARQL endpoints*
- I.e. SPARQL queries can be submitted to a database server that sends back the results
- Uses HTTP to submit URL-encoded queries to server
  `GET /sparql/?query=...  HTTP/1.1`
- Actually defined via W3C Web Services, see

  `http://www.w3.org/TR/rdf-sparql-protocol/`
- For `SELECT` queries you get a XML or JSON result set, see

  `http://www.w3.org/TR/rdf-sparql-XMLres/`
  `http://www.w3.org/TR/rdf-sparql-json-res/`
- Nothing you would want to do manually!

Outline

1. Reminder: SPARQL
2. RDF Datasets
3. Functions and Operators
4. SPARQL 1.1
   - Update language
   - Property paths
   - Aggregates and negation
5. New Semantic Web Community

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.
- URIs are used as names for the graphs.

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.
- URIs are used as names for the graphs.
- We have so far used the default unnamed graph as the *active graph*.

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.
- URIs are used as names for the graphs.
- We have so far used the default unnamed graph as the *active graph*.
- We may specify

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.
- URIs are used as names for the graphs.
- We have so far used the default unnamed graph as the *active graph*.
- We may specify
  - a new default graph for the query by an RDF Merge of named graphs.

SPARQL is used against an RDF Dataset

- The RDF Dataset is composed of:
  - one default unnamed graph
  - possibly one or more *named graphs*.
- URIs are used as names for the graphs.
- We have so far used the default unnamed graph as the *active graph*.
- We may specify
  - a new default graph for the query by an RDF Merge of named graphs.
  - a new active graph for parts of the query.

Motivations

Used to divide the data up in chunks.

- To improve performance.
- To track provenance.
- For access control.
- To return only specific data.
- Select only trusted data.
- . . .

Features to use named graphs

- Query language standard way is to use:

Features to use named graphs

- Query language standard way is to use:
  - FROM to add a graph to the default graph.

Features to use named graphs

- Query language standard way is to use:
  - `FROM` to add a graph to the default graph.
  - `FROM NAMED` and `GRAPH` to address an active graph.

Features to use named graphs

- Query language standard way is to use:
  - `FROM` to add a graph to the default graph.
  - `FROM NAMED` and `GRAPH` to address an active graph.
- Protocol standard way (takes precedence):

Features to use named graphs

- Query language standard way is to use:
  - `FROM` to add a graph to the default graph.
  - `FROM NAMED` and `GRAPH` to address an active graph.
- Protocol standard way (takes precedence):
  - `default-graph-uri` to add a graph to the default graph.

Features to use named graphs

- Query language standard way is to use:
  - `FROM` to add a graph to the default graph.
  - `FROM NAMED` and `GRAPH` to address an active graph.
- Protocol standard way (takes precedence):
  - `default-graph-uri` to add a graph to the default graph.
  - `named-graph-uri` to address a graph

Features to use named graphs

- Query language standard way is to use:
  - `FROM` to add a graph to the default graph.
  - `FROM NAMED` and `GRAPH` to address an active graph.
- Protocol standard way (takes precedence):
  - `default-graph-uri` to add a graph to the default graph.
  - `named-graph-uri` to address a graph
- Several non-standard extensions.

Default graph example

Add three Turtle files to default graph

```
SELECT ?kname ?fname
FROM <http://data.lenka.no/dumps/fylke-geonames.ttl>
FROM <http://data.lenka.no/dumps/kommune-navn.ttl>
FROM <http:// .../dumps/kommunesentre-geonames.ttl>
WHERE {
  ?fylke a gd:Fylke ;
         gn:officialName ?fname ;
         gn:childrenFeatures ?kommune .
  ?kommune a gd:Kommune ;
           gn:officialName ?kname ;
  FILTER (langMatches(lang(?fname), ’no’))
  FILTER (langMatches(lang(?kname), ’no’))
}
```

Named graph example

Take coordinates from one source only

```
SELECT *
FROM <http://data.lenka.no/dumps/kommune-navn.ttl>
FROM <http://data.lenka.no/dumps/kommunesentre-geonames.ttl>
FROM NAMED <http://data.lenka.no/dumps/kommunesentre-geonames.ttl>
FROM <http://sws.geonames.org/6453350/about.rdf>
WHERE {
  {
    ?feature gn:officialName "Lillehammer"@no .
  } UNION {
    ?feature gn:name "Lillehammer" .
  }
  OPTIONAL {
    GRAPH <http://data.lenka.no/dumps/kommunesentre-geonames.ttl> {
      ?feature pos:lat ?lat ;
               pos:long ?long ;
               owl:sameAs ?other .
    }
  }
  OPTIONAL {
    ?feature gn:population ?pop .
  }
}
```

Run this in the exercises!

Note

- Nothing compels the query engine to actually fetch the URIs!
- The examples from “Semantic Web Programming” doesn’t work anymore (but default Joseki config can do the same thing).
- Older frameworks (i.e. pre-SPARQL 1.0) sometimes use “contexts”.

Outline

1. Reminder: SPARQL
2. RDF Datasets
3. **Functions and Operators**
4. SPARQL 1.1
   - Update language
   - Property paths
   - Aggregates and negation
5. New Semantic Web Community

Overview

- Usual binary operators: `||`, `&&`, `=`, `!=`, `<`, `>`, `<=`, `>=`, `+`, `-`, `*`, `/`.
- Usual unary operators: `!`, `+`, `-`.
- Unary tests: `bound(?var)`, `isURI(?var)`, `isBlank(?var)`, `isLiteral(?var)`.
- Accessors: `str(?var)`, `lang(?var)`, `datatype(?var)`

Read the spec for details!

More tests

- Uses the concept of “Effective Boolean Value”.
- `sameTerm(?var)` is used with unsupported data types.
- `langMatches` is used with `lang` to test for language e.g. `langMatches( lang(?title), "no" )`.
- `regex` is to used to match a variable with a regular expression. *Always use with* `str(?var)`! E.g.: `regex(str(?name), "Os")`.
- Has extension mechanism for writing your own!
- SPARQL 1.1 brings more functions!

Outline

1 Reminder: SPARQL

2 RDF Datasets

3 Functions and Operators

4 SPARQL 1.1
- Update language
- Property paths
- Aggregates and negation

5 New Semantic Web Community

SPARQL 1.1 Status

- SPARQL 1.1 is work in progress!
- Some implementors are tracking it closely.
- Some features have been implemented for a while.
- Some are still in flux, thus things said in this lecture may change.
- Bugs abound!
- Subset lectured here.

The following (read-only) examples can be tried on `http://lod.kjernsmo.net/sparql`.

Whole graph operations

From the specification:

`LOAD [ SILENT ] <documentURI> [ INTO GRAPH <uri> ]` Loads the graph at documentURI into the specified graph, or the default graph if not given.

Whole graph operations

From the specification:

`LOAD [ SILENT ] <documentURI> [ INTO GRAPH <uri> ]` Loads the graph at documentURI into the specified graph, or the default graph if not given.

`CLEAR [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the triples from the specified graph, the default graph, all named graps or all graphs respectively. Some implementations may remove the whole graph.

Whole graph operations

From the specification:

`LOAD [ SILENT ] <documentURI> [ INTO GRAPH <uri> ]` Loads the graph at documentURI into the specified graph, or the default graph if not given.

`CLEAR [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the triples from the specified graph, the default graph, all named graps or all graphs respectively. Some implementations may remove the whole graph.

`CREATE [ SILENT ] GRAPH <uri>` Creates a new graph in stores that record empty graphs.

Whole graph operations

From the specification:

`LOAD [ SILENT ] <documentURI> [ INTO GRAPH <uri> ]` Loads the graph at documentURI into the specified graph, or the default graph if not given.

`CLEAR [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the triples from the specified graph, the default graph, all named graps or all graphs respectively. Some implementations may remove the whole graph.

`CREATE [ SILENT ] GRAPH <uri>` Creates a new graph in stores that record empty graphs.

`DROP [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the specified graph, the default graph, all named graps or all graphs respectively. It also removes all triples of those graphs.

Whole graph operations

From the specification:

`LOAD [ SILENT ] <documentURI> [ INTO GRAPH <uri> ]` Loads the graph at documentURI into the specified graph, or the default graph if not given.

`CLEAR [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the triples from the specified graph, the default graph, all named graps or all graphs respectively. Some implementations may remove the whole graph.

`CREATE [ SILENT ] GRAPH <uri>` Creates a new graph in stores that record empty graphs.

`DROP [ SILENT ] (GRAPH <uri> | DEFAULT | NAMED | ALL )` Removes the specified graph, the default graph, all named graps or all graphs respectively. It also removes all triples of those graphs.

Usually, `LOAD` and `DROP` are what you want.

Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/kjetil> .
    <staff/kjetil> foaf:name "Kjetil Kjernsmo" ;
                   owl:sameAs <http:// ...> .
} }
```

Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/kjetil> .
    <staff/kjetil> foaf:name "Kjetil Kjernsmo" ;
                   owl:sameAs <http:// ...> .
} }
```

Deleting triples from a graph

```
DELETE DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:oblig <exercise/oblig6> .
    <exercise/oblig6> rdfs:label "Mandatory Exercise 6" .
} }
```

Inserting and deleting triples

Inserting triples in a graph

```
INSERT DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:taughtBy <staff/kjetil> .
    <staff/kjetil> foaf:name "Kjetil Kjernsmo" ;
                   owl:sameAs <http:// ...> .
} }
```

Deleting triples from a graph

```
DELETE DATA {
  GRAPH </graph/courses/> {
    <course/inf3580> ex:oblig <exercise/oblig6> .
    <exercise/oblig6> rdfs:label "Mandatory Exercise 6" .
} }
```

If no `GRAPH` is given, default graph is used.

Inserting conditionally

Most useful when inserting statements that you already have, but hold true for something else.

Inserting triples for another subject

```
INSERT {
  <http:// .../geo/inndeling/03> a gd:Fylke ;
       gn:name "Oslo" ;
       ?p ?o .
}
WHERE {
  <http:// .../geo/inndeling/03/0301> a gd:Kommune ;
            ?p ?o .
}
```

Deleting conditionally

From specification:

Deleting old books

```
DELETE {
   ?book ?p ?v .
}
WHERE {
  ?book dc:date ?date .
  FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
  ?book ?p ?v .
}
```

Deleting conditionally, common shortform

Deleting exactly what's matched by the `WHERE` clause.

Deleting in SMIL

```
DELETE WHERE {
  ?s a skos:Concept .
  ?s ?p <http://smil.uio.no/topic/betennelse-i-bihuler> .
}
```

Most common update query in the Sublima and Media Zone projects.

Delete/Insert full syntax

In most cases, you would delete some triples first, then add new, possibly in the same or other graphs.
From specification:

All the possibilities offered by DELETE/INSERT

```
[ WITH <uri> ]
DELETE {modify_template [ modify_template ]* }
INSERT {modify_template [ modify_template ]* }
[ USING [NAMED] <uri> ]*
[ WHERE ] GroupGraphPattern
```

Delete/Insert simple example

Update user information query from Sublima

```
DELETE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
INSERT {
  <http:// .../user/larshvermannsen> a sioc:User ;
    rdfs:label """Lars Hvermannsen"""@no ;
    sioc:email <mailto:lars@hvermannsen.no> ;
    sioc:has_function <http:// .../role/Administrator> ;
    wdr:describedBy status:inaktiv .
}
WHERE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
```

Delete/Insert example with named graphs

Update user information query from Sublima

```
DELETE {
  GRAPH </graphs/users/> {
    <http:// .../user/larshvermannsen> ?p ?o .
  }
}
INSERT {
  GRAPH </graphs/users/> {
    <http:// .../user/larshvermannsen> a sioc:User ;
          rdfs:label """Lars Hvermannsen"""@no .
  }
}
USING </graphs/users/>  WHERE {
    <http:// .../user/larshvermannsenno> ?p ?o .
}
```

Delete/Insert example explained

- `USING` plays the same role as `FROM`.

Delete/Insert example explained

- `USING` plays the same role as `FROM`.
- `GRAPH` says where to insert or delete.

Delete/Insert example explained

- `USING` plays the same role as `FROM`.
- `GRAPH` says where to insert or delete.
- This makes it possible to delete, insert and match against different graphs.

Delete/Insert example with single named graphs

Update user information query from Sublima

```
WITH </graphs/users/>
DELETE {
  <http:// .../user/larshvermannsen> ?p ?o .
}
INSERT {
  <http:// .../user/larshvermannsen> a sioc:User ;
        rdfs:label """Lars Hvermannsen"""@no .
}
WHERE {
  <http:// .../user/larshvermannsenno> ?p ?o .
}
```

Equivalent to the previous query!

Basic motivation for Property paths

- Some queries get needlessly complex.
- Sometimes write `foaf:maker|dct:creator` instead of `UNION`.
- To get friend's name, go { `_:me foaf:knows/foaf:name ?friendsname` }.
- etc.
- Adds a small property-oriented query language inside the language.

Longer example

Friends of Kjetil Kjernsmo, including subproperties

```
SELECT ?label ?name WHERE {
  ?rel rdfs:subPropertyOf?  foaf:knows ;
       rdfs:label ?label .
  <http://www.kjetil.kjernsmo.net/foaf#me> ?rel ?friend .
  ?friend foaf:name|foaf:firstName ?name .
}
```

Longer example

Friends of Kjetil Kjernsmo, including subproperties

```
SELECT ?label ?name WHERE {
  ?rel rdfs:subPropertyOf?  foaf:knows ;
       rdfs:label ?label .
  <http://www.kjetil.kjernsmo.net/foaf#me> ?rel ?friend .
  ?friend foaf:name|foaf:firstName ?name .
}
```

Answer (manual excerpt):

?label	?name
"Child Of"@en	"Ragnhild Kjernsmo"
"Child Of"@en	"Dag Kjernsmo"
"knows"	"Gregory Todd Williams"
"Parent Of"@en	"Eivind"
"Parent Of"@en	"Synne"
"Spouse Of"@en	"Hege Prestrud"
...	...

From the specification

Syntax Form	Matches
uri	A URI or a prefixed name. A path of length one.
^elt	Inverse path (object to subject).
(elt)	A group path elt, brackets control precedence.
elt1 / elt2	A sequence path of elt1, followed by elt2
elt1 ^ elt2	Like elt1 / ^elt2, (elt1 and the inverse of elt2).
elt1 | elt2	A alternative path of elt1, or elt2.
elt*	A path of zero or more occurrences of elt.
elt+	A path of one or more occurrences of elt.
elt?	A path of zero or one elt.
elt{n,m}	A path between n and m occurrences of elt.
elt{n}	Exactly n occurrences of elt. A fixed length path.
elt{n,}	n or more occurrences of elt.
elt{,n}	Between 0 and n occurrences of elt.

Aggregate functions: Set functions

- Flatten is a function which is used to collapse multisets of lists into a multiset, so for example $\{(1,2),(3,4)\}$ becomes $\{1,2,3,4\}$.
- Count counts the number of times a variable has been bound.
- Sum sums numerical values of bound variables.
- Avg finds the average of numerical values of bound variables.
- Min finds the minimum of the numerical values of bound variables.
- Max finds the maximum of the numerical values of bound variables.
- Group_Concat creates a string with the values concatenated, separated by some optional character.
- Sample just returns a sample of the values.

Already implemented in most frameworks!

Aggregate functions: Grouping

- Solutions can optionally be grouped according to one or more expressions.

Aggregate functions: Grouping

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.

Aggregate functions: Grouping

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.
- To filter solutions resulting from grouping, use `HAVING`.

Example

Counties of Norway with less than 15 municipalities

```
SELECT ?name (count(?kommune) AS ?kcount)
WHERE {
  ?fylke a gd:Fylke ;
           gn:officialName ?name ;
           gn:childrenFeatures ?kommune .
  ?kommune a gd:Kommune .
  FILTER (langMatches(lang(?name),'no'))
} GROUP BY ?name HAVING (?kcount < 15)
```

Also uses *projection*!

Answer:

name	kcount
"Vest-Agder"@no	14
"Oslo"@no	1
"Vestfold"@no	13

Negation

Two ways to do negation:

People without names

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    MINUS { ?person foaf:name ?name }
}
```

People without names, take II

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

`FILTER NOT EXISTS` filters based on bindings whereas `MINUS` removes solutions that matches the pattern.

Open World Assumption

Aggregates and negation assume Closed World and Unique names!
The answers are only true with respect to the current dataset.

- “As far as we know, there are 13 municipalities in Vestfold.”
- Can’t say: “they don’t have names”, can say: “we don’t know their names”.
- “As far as we know, no-one has climbed that mountain.”
- “Based on the available data, the average fuel price is 13.37 NOK/l.”

This is like the Real World!

Outline

1. Reminder: SPARQL
2. RDF Datasets
3. Functions and Operators
4. SPARQL 1.1
   - Update language
   - Property paths
   - Aggregates and negation
5. New Semantic Web Community

Lenka.no

- A community site for Linked Data in Norway.
- A site to just do stuff instead of waiting for “official” projects.
- Lenka.no isn’t up yet, but these are:
  - `http://lists.lenka.no/listinfo/data`
  - `http://vocab.lenka.no/`
  - E.g. `http://data.lenka.no/geo/inndeling/03/0301`
- Next up: Yr.no, a database of places in Norway, etc.