More SPARQL

The first exercise is to set up a real endpoint, and for this, we will use Fuseki, which is a SPARQL endpoint that uses Jena and ARQ and is developed by the same people. We will set it up using Eclipse.

1 Run Fuseki

- Download from `https://jena.apache.org/download/index.cgi`
- Unpack the file, e.g. unzip jena-fuseki-3.7.0.zip
- This will give you a directory named jena-fuseki-3.7.0/
- To start the server, run the fuseki-server script: `fuseki-server --update --mem /ds`
- You should now have a server that supports update queries, keeps the data set in memory and places the default dataset at `http://localhost:3030/ds`
- For more options on how to serve RDF data `https://jena.apache.org/documentation/serving_data/`
- Now go to `http://localhost:3030/sparql.html` and you should have a running SPARQL Endpoint! This page allows general purpose SPARQL queries for remote data sets.

2 RDF Dataset

Now, we will study the following query (from slide 17 from the 2015-lecture slides[1]):

```
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>
```

[1] `http://www.uio.no/studier/emner/matnat/ifi/INF3580/v15/undervisningsmateriale/lecture13-noanim.pdf`

```
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 .
  }
}
```

2.1 Exercise

Find necessary prefixes.

You need to begin your query with the namespace prefixes, which have been omitted from the slides for brevity. They look like this:

```
PREFIX gd: <http://vocab.lenka.no/geo-deling#>
```

You can use the service at `http://prefix.cc/`. What other prefix declarations do you need to run the query?

2.1.1 Solution

```
PREFIX pos: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX gn: <http://www.geonames.org/ontology#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
```

2.2 Exercise

Paste the query into the text field and run it. What's the result:

2.2.1 Solution

feature	lat	long	other
`http://data.lenka.no/geo/sted/143669`	61.11514	10.46628	`http://sws.geonames.org/3147474`
`http://data.lenka.no/geo/inndeling/05/0501`	61.11464	10.46743	`http://sws.geonames.org/645335(`
`http://sws.geonames.org/6453350/`			

2.3 Exercise

Why do you think you get three solutions to the query?

2.3.1 Solution

The named graph from Geonames only includes the Lillehammer municipality, whereas the other graphs include both the city of Lillehammer and the municipality.

2.4 Exercise

Why is the latitude and longitude only given for the two data.lenka.no sources?

2.4.1 Solution

That was the point of the query, since we assumed we could trust that source more than Geonames, we chose to query only the data.lenka.no datasource in the active graph using the GRAPH keyword.

2.5 Exercise

Why is the population only given in the Geonames solution?

2.5.1 Solution

That's not present in the data.lenka.no data, only in Geonames.

3 SPARQL Update

Now, we will turn to SPARQL Update. This cannot be done from the general purpose query form (at least not for the local datasets).

Go to: `http://localhost:3030/control-panel.tpl` and select the /ds dataset, or if you have started the server with multiple datasets, choose one that has update enabled.

You will now be taken to a page with three forms. One for read queries, one for update queries and one for uploading RDF files and storing them in a dataset.

For all SPARQL 1.1 update queries, you must use the second form. Update queries include `INSERT, DELETE, LOAD, CLEAR, CREATE, ADD, DROP, COPY, and MOVE`.

3.1 Exercise

Insert the data about Homer, Marge and Lisa Simpson, available at `http://sws.ifi.uio.no/inf3580/v14/oblig/3/simpsons.ttl` into the triple store. The endpoint may not give you any meaningful feedback the operation succeeded, so you may use the read endpoint (the first form) with `SELECT * WHERE { ?s ?p ?o }` to confirm that it worked.

3.1.1 Solution

```
LOAD <http://sws.ifi.uio.no/inf3580/v14/oblig/3/simpsons.ttl>
```

Also valid answer would be to paste all the data into an INSERT DATA query

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sim: <http://www.ifi.uio.no/INF3580/simpsons#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>

INSERT DATA {
sim:Homer
      a       foaf:Person ;
      foaf:age "36"^^xsd:int ;
      foaf:name "Homer Simpson" .
# Etc...
}
```

3.2 Exercise

Homer Simpson is actually 37 years old now. Update!

3.2.1 Solution

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX sim: <http://www.ifi.uio.no/INF3580/simpsons#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>

DELETE {
  sim:Homer foaf:age ?age .
}
INSERT {
  sim:Homer foaf:age "37"^^xsd:int .
}
WHERE {
  sim:Homer foaf:age ?age .
}
```

3.3 Exercise

Delete the fact that someone is 34 years old.

3.3.1 Solution

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>

DELETE WHERE {
  ?person foaf:age "34"^^xsd:int .
}
```

3.4 Exercise

Delete persons aged less than 10 years.

3.4.1 Solution

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>

DELETE {
  ?person ?p ?o .
} WHERE {
  ?person a foaf:Person ;
  foaf:age ?age ;
  ?p ?o .
  FILTER (?age < 10)
}
```

3.5 Exercise

Delete all the Simpsons triples.

3.5.1 Solution

```
DROP ALL
```

3.6 Exercise

Insert the following triples (look up the prefix):

```
<http://example.org/dahut> rdfs:label "Dahut"@en_US, "Le Dahu"@fr .
```

3.6.1 Solution

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

INSERT DATA {
  <http://example.org/dahut> rdfs:label "Dahut"@en-EN, "Le Dahu"@fr .
}
```

3.7 Exercise

Find the English-language labels irrespective of locale.

3.7.1 Solution

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?label WHERE {
  [ rdfs:label ?label ]
  FILTER (langMatches(lang(?label), 'en'))
}
```

Using just

```
FILTER (lang(?label) = 'en')
```

fails because it tries to match the whole language string.

4 Property paths

In this exercise we will use the family vocabulary we have used in previous exercises. Write SPARQL queries which use property path primarily to solve the following questions. Note that some of these queries will not return any results if executed on the simpsons data at `http://sws.ifi.uio.no/inf3580/v14/oblig/3/simpsons.ttl`.

4.1 Exercise

Find all pairs (x,y) where y is the grandmother on the mother's side of x (mormor in Norwegian).

4.1.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x ?y
WHERE
  { ?x fam:hasMother{2} ?y }
```

4.2 Exercise

Find all pairs (x,y) where y is the grandparent of x.

4.2.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x ?y
WHERE
  { ?x (fam:hasMother | fam:hasFather | fam:hasParent){2}  ?y }
```

4.3 Exercise

Find all pairs (x,y) where y is an ancestor of x.

4.3.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x ?y
WHERE
  { ?x (fam:hasMother | fam:hasFather | fam:hasParent)+  ?y }
```

4.4 Exercise

Find all pairs (x,y) where y is the uncle of x.

4.4.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x ?y
WHERE
  { ?x (fam:hasMother | fam:hasFather | fam:hasParent) / fam:hasBrother  ?y }
```

4.5 Exercise

Find everyone who is married. List them in one column, and remember, do not use `UNION` but paths solve this.

4.5.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x
WHERE
  { ?x fam:hasSpouse | ^fam:hasSpouse [] }
```

4.6 Exercise

Find all pairs (x,y) where y is the second cousin (norsk: tremenning) of x.

4.6.1 Solution

This solution is not correct. Why?

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
SELECT ?x ?y
WHERE
  { ?x (fam:hasFather | fam:hasMother){3} / (^fam:hasFather | ^fam:hasMother){3}  ?y }
```

5 Aggregate functions

Using the same vocabulary as in the previous exercise, solve these exercises by using aggregate functions.

5.1 Exercise

List everyone and the number of siblings they have.

5.1.1 Solution

```
PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x (count(?y) AS ?ycount)
WHERE
  { ?x a foaf:Person .
    OPTIONAL { ?x (fam:hasBrother | fam:hasSister) ?y }
  }
GROUP BY ?x
```

5.2 Exercise

Find the oldest and youngest person.

5.2.1 Solution

Youngest:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?my
WHERE {
  ?x foaf:age ?my
  {
     SELECT (min(?y) as ?my)
     WHERE { ?x foaf:age ?y }
  }
}
```

Oldest:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?my
WHERE {
  ?x foaf:age ?my
  {
     SELECT (max(?y) as ?my)
     WHERE { ?x foaf:age ?y }
  }
}
```