Chapter 7
Linked Open Data – World Wide RDF Web – Web of Data

... slides in this section are work in progress ...

Traditional RDF Data Sources

- Web sources (Web services or “traditional” Web Servers) provide HTML (for browsers) or RDF (for data processing) data.
- simple, via physical resources of the form `filename.html`, `filename.rdf`, or `filename.ttl` etc.
- virtual URIs (and also URLs) are used inside of the the data.

Linked Open Data (LOD) Principles

Mandatory: access to RDF data

- There must be resolvable http:// (or https://) URIs.
- They must resolve, with or without content negotiation, to RDF data in one of the popular RDF formats (RDFa, RDF/XML, Turtle, N-Triples).
- Access of the entire dataset must be possible via RDF crawling, via an RDF dump, or via a SPARQL endpoint.
- There should be references (URL links) to other LOD datasets
  - properties whose object is an URL from another dataset, or
  - owl:sameAs triples that connect to URL from another dataset

Optional: access as HTML Web pages based on the RDF data

- pages about individual resources
- SPARQL Web interface
- References:
  `https://www.w3.org/DesignIssues/LinkedData.html`
  `http://www.lod-cloud.net/#about`

LOD: Access Options

- Example case: Mondial as LOD

RDF Data Access

- The URIs in Mondial are "real" URLs of the form `http://www.semwebtech.org/mondial/10/countries/D/` for Germany
- URL access with a (HTML) Web browser: Web page with formatted information about this object.
- URL access with an RDF client: receives RDF data related to this object (commonly sent as XML/RDF)
  - "rapper" tool: `rapper http://www.semwebtech.org/mondial/10/islands/Sicilia/` (shows the data in Turtle format)
  - ```
    # jena -q -if http://www.semwebtech.org/mondial/10/islands/Sicilia -qf alltriples.sparq
    select ?X ?P ?Y
    where {?X ?P ?Y}
    ```
    [Filename: RDF/alltriples.sparql]
    (this way, alltriples.sparql can be applied to any LOD URL)

LOD: SPARQL Endpoints

LOD sources provide also a SPARQL protocol endpoint.

- optionally, many LOD endpoints provide an interactive SPARQL (HTML) Web interface: Mondial: via `http://www.semwebtech.org/mondial/10/`

Recall that SPARQL is also a communication *protocol* (for exchanging RDF data):

- W3C Recommendation at `http://www.w3.org/TR/sparql11-protocol/`
- usually at the URL `http://.../sparql`; GET, POST via HTTP
- GET via browser (query must be URL-encoded) shows formatted HTML table:
  `http://www.semwebtech.org/mondial/10/sparql?query=select%20%3fX%20where%20%7b%3fX%20a%20%3chttp%3a%2f%2fwww.semwebtech.org%2fmondial%2f10%2fmeta%23Country%3e%7d`
  (URL-encoding service at `https://www.branah.com/unicode-converter`)
- GET/POST from an RDF consumer:
  - result in SPARQL Query Results XML Format (set of tuples of bindings in XML) (described at `https://www.w3.org/TR/rdf-sparql-XMLres/`)
  - example (via browser): `http://rdf.insee.fr/sparql`, set response format to "RDF/XML" (actually, it is not RDF/XML, but SPARQL Query Results XML Format).
    ```

LOD Example: DBpedia

- data extracted from Wikipedia (since 2007)
- research project for Web data extraction and natural language processing
- data correctness: rather bad
- `http://dbpedia.org/page/France` Web page describing the RDF data about France;
- `http://dbpedia.org/data/France` delivers RDF/XML data about France (even to the browser); this can easily be parsed and queried by tools.
- SPARQL endpoint and Web interface at `http://dbpedia.org/sparql/`
- The URIs in dbpedia are actually of the form `http://dbpedia.org/resource/France` which redirects browsers to the HTML, and RDF tools to the RDF/XML data URL.

```
select ?X ?P ?Y
from <http://dbpedia.org/resource/France>
where {?X ?P ?Y}
```
[Filename: RDF/dbpedia-france.sparql]

```
prefix owl: <http://www.w3.org/2002/07/owl#>
select ?X ?Y from <http://dbpedia.org/resource/France>
where {?X owl:sameAs ?Y}
```
[Filename: RDF/dbpedia-france-same-as.sparql]

- ... yields France owl:sameAs `http://sws.geonames.org/3017382/`.

LOD Example: Yago

"Yet Another Great Ontology"

- Project homepage: `https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`
- data extracted from Wikipedia (started 2007)
- research project for Web data extraction and natural language processing
- partially using and cleaning DBpedia base data
- quality/completeness: better than DBpedia
- resource URIs: `http://yago-knowledge.org/resource/Denmark` implemented (HTML) via redirection to SPARQL DESCRIBE queries of the form `http://lod.openlinksw.com/describe/?url=http%3A%2F%2Fdbpedia.org%2Fresource%2FDenmark`
- SPARQL endpoint and Web interface at `https://linkeddata1.calcul.u-psud.fr/sparql`
- Yago contains owl:sameAs-Links to Wikidata

LOD Example: geonames

- `http://sws.geonames.org/3017382/` in the browser: annotated google map with France;
- `http://sws.geonames.org/3017382/about.rdf` delivers RDF/XML data about France (For RDF, about.rdf is the equivalent to HTML's index.html)
- URL access to plain `http://sws.geonames.org/3017382/` with a *well-configured (see below)* RDF tool yields the triples:

```
select ?X ?P ?Y
from <http://sws.geonames.org/3017382/>
where {?X ?P ?Y}
```
[Filename: RDF/geonames-france.sparql]

- "rapper" tool: `rapper http://sws.geonames.org/3017382/`
- `http://sws.geonames.org/3017382/` is the resource describing France; while
- the triple <http://sws.geonames.org/3017382/about.rdf> <http://purl.org/dc/terms/modified> "2018-02-06"^^<http://www.w3.org/2001/XMLSchema#date> describes the *document* that describes France.

LOD Example: insee.fr

- many LOD sources are provided by e-government institutions.
- e.g., insee.fr (L'Institut national de la statistique et des études économiques, France):
  - `http://rdf.insee.fr/`
  - Browser: HTML table that shows RDF triples, e.g. `http://id.insee.fr/geo/departement/42`
  - RDF access: request triples `rapper http://id.insee.fr/geo/departement/42`
  - SPARQL endpoint with Web interface at `http://rdf.insee.fr/sparql`

Other LOD sources

- Wikidata: uses a specific modeling, see Slides 320
- overview of existing LOD data sources: `www.lod-cloud.net`

Aside: URL- “percent”-encoding

- whitespace → %20; some services allow “+” for whitespace.
- reserved symbols in URL structure: : / ? # [ ] @ ! $ & ( ) * + , ; =
  must be URL-encoded when occurring in a parameter string, if they could be misunderstood as URI structuring symbols
- currently for Mondial (and Insee.fr) , also `< > { }` must be encoded, “+” is not allowed for whitespace
- in SPARQL, there is the `ENCODE_FOR_URI`(*string*) function
  (I currently see no application for it *inside* SPARQL)

LOD: HTTP Accept Header

If an HTML interface is provided, LOD URLs are both “HTML URLs” and “LOD RDF URLs”:

- For HTML consumers (browsers), HTML should be delivered;
- For RDF consumers, RDF/XML, RDF/A (HTML with RDF annotations), Turtle unicode format etc. should be delivered.

⇒ HTTP Request contains appropriate information in the "accept" header:

- Browsers e.g.: text/html,application/xhtml+xml,
- XML consumers in general: application/xml,
- RDF consumers: application/rdf+xml, text/rdf, ... and some more.

- Technical observation
  - when running the jena-based tool from command line, it obtains triples;
  - when running it inside the Web Service, it obtained HTML (and threw an error)

  ⇒ don’t access LOD data via simple GET *url*, but explicitly opening of HTTP connections with explicit
  ACCEPT: {text/turtle|application/rdf+xml} header.

Aside: Technical details of Querying a SPARQL Endpoint by HTTP+SPARQL Protocol

- See `https://www.w3.org/TR/sparql11-protocol/#query-operation`:
- HTTP GET,
- HTTP POST with URL-encoded parameters in message body,
- HTTP POST with unencoded SPARQL query in message body;
- query,
- optionally default graph and named graphs may be specified;
- accept media types for answer:
  SELECT query (variable bindings): application/sparql-results+xml
  CONSTRUCT query (graph/triples): application/rdf+xml or text/turtle
  DESCRIBE query (graph/triples): application/rdf+xml or text/turtle
- Not every SPARQL server supports each of these access methods;
- wget and curl as quick-and-dirty UNIX tools for sending HTTP requests;
- Java sample code.

Samples and tests: HTTP GET with parameters

- GET URLs contain the parameters in url-encoded form:

```
## wget -O -   outputs to stdout.
## optionally add    --header='Accept: application/sparql-results+xml'

wget -O - \
  http://dbpedia.org/sparql?query=select+distinct+?X+where+{?Y+a+?X}
wget -O - \
  http://id.insee.fr/sparql?query=select+distinct+?X+where+{?Y+a+?X}
     ## insee: wget: ok; GET URL via firefox must be URL-encoded (same effect as mond
  http://id.insee.fr/sparql?query=
    select%20distinct%20%3fX%20where%20%7b%3fY%20a%20%3fX%7d

## curl: {a,b,c} has a special semantics for curl, so encode it or turn {..} off:
curl http://dbpedia.org/sparql?query=select+distinct+?X+where+%7B?Y+a+?X%7D
curl -g http://dbpedia.org/sparql?query=ask{}
curl http://id.insee.fr/sparql?query=select+distinct+?X+where+%7B?Y+a+?X%7D
curl -g http://id.insee.fr/sparql?query=ask{}
```

- results are in SPARQL Query Results XML Format.

Result: SPARQL Query Results XML Format

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
  <variable name="Concept"/>
 </head>
 <results distinct="false" ordered="true">
  <result>
   <binding name="Concept"><uri>http://www.w3.org/2002/07/owl#Thing</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://www.w3.org/2002/07/owl#Class</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://dbpedia.org/ontology/Person</uri></binding>
  </result>
  <result>
   <binding name="Concept"><uri>http://dbpedia.org/ontology/Country</uri></binding>
  </result>
   :
  and many more
   :
</sparql>
```


Samples: HTTP POST with URL-encoded parameters in message body

- POST: send parameters url-encoded in body

```
## wget: --method=POST must not be specified, if --post-data is used
## not necessary: --header='Content-Type: application/x-www-form-urlencoded' \

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://dbpedia.org/sparql

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://id.insee.fr/sparql

###curl -d: => POST;   -X POST not necessary
## curl optional: -H 'Content-Type: application/x-www-form-urlencoded'
curl -d 'query=select+distinct+?X+where+{?Y+a+?X}' \
     http://id.insee.fr/sparql
## equivalent:
curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     http://id.insee.fr/sparql
```

Samples: HTTP POST with URL-encoded parameters in message body (cont'd)

```
### lotico.com delivers result as [application/sparql-results+json]:
### (lotico.com hosts the SPARQL protocol interface of http://www.geosparql.org/)

wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     http://www.lotico.com:3030/lotico/sparql

### request results as xml:
wget -O - \
     --post-data='query=select+distinct+?X+where+{?Y+a+?X}' \
     --header='Accept: application/sparql-results+xml' \
     http://www.lotico.com:3030/lotico/sparql

curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     http://www.lotico.com:3030/lotico/sparql

curl --data-urlencode 'query=select distinct ?X where {?Y a ?X}' \
     --header 'Accept: application/sparql-results+xml' \
     http://www.lotico.com:3030/lotico/sparql
```


Samples: HTTP POST with unencoded SPARQL query in message body

- HTTP POST with unencoded SPARQL query in message body;
  (some services do not support this)
- parameters (default graph etc.) still as parameters

```
### not supported by http://dbpedia.org/sparql and http://id.insee.fr/sparql

wget -O - --method=POST \
     --header='Content-Type: application/sparql-query' \
     --body-data='select distinct ?X where {?Y a ?X}' \
     http://www.lotico.com:3030/lotico/sparql

curl -d 'select distinct ?X where {?Y a ?X}' \
   -H 'Content-Type: application/sparql-query' \
   http://www.lotico.com:3030/lotico/sparql
```

Querying a SPARQL Endpoint by Java (SPARQL Protocol) – GET

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class SparqlGET {
  public static void main(String[] args) { try {
    BufferedReader br = null;
    URL inputURL = new URL("http://dbpedia.org/sparql?query=" +
          "select+distinct+?X+where+{?X+a+<http://dbpedia.org/ontology/Country>}");
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("GET");
    con.setRequestProperty("Accept", "application/sparql-results+xml");
    con.connect();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();
    System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlGET.java]

Querying a SPARQL Endpoint by Java (SPARQL Protocol) – encoded POST

```
import java.io.BufferedReader; import java.net.HttpURLConnection; import java.net.URL;
import java.io.InputStreamReader;  import java.io.OutputStreamWriter;
public class SparqlEncPOST {
  public static void main(String[] args) { try {  BufferedReader br = null;
    URL inputURL = new URL("http://dbpedia.org/sparql");
    String q="query=construct+{+?X+a+<foo:country>}" +
             "+where+{?X+a+<http://dbpedia.org/ontology/Country>}";
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("POST");
    con.setDoOutput(true);
    con.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
    con.setRequestProperty("Accept", "text/turtle");
    con.connect();
      OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
      wr.write(q); wr.flush(); wr.close();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();  System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlEncPOST.java]

Querying a SPARQL Endpoint by Java (SPARQL Protocol) Direct POST

```
import java.io.BufferedReader; import java.net.HttpURLConnection; import java.net.URL;
import java.io.InputStreamReader;  import java.io.OutputStreamWriter;
public class SparqlDirectPOST {
  public static void main(String[] args) { try {
    BufferedReader br = null;
    URL inputURL = new URL("http://www.lotico.com:3030/lotico/sparql");
    String q="select distinct ?C where {?X a ?C}";
    HttpURLConnection con = (HttpURLConnection) inputURL.openConnection();
    con.setRequestMethod("POST");
    con.setDoOutput(true);
    con.setRequestProperty("Content-Type", "application/sparql-query");
    con.setRequestProperty("Accept", "application/sparql-results+xml");
      OutputStreamWriter wr = new OutputStreamWriter(con.getOutputStream());
      wr.write(q); wr.flush(); wr.close();
    con.connect();
    String s = "";   StringBuffer res= new StringBuffer();
    br = new BufferedReader(new InputStreamReader(con.getInputStream(), "UTF-8"));
    while ((s = br.readLine()) != null) { res.append(s+ "\n"); }
    br.close();   System.out.println(res);  // or fill XML from Reader
  } catch (Exception e) { e.printStackTrace(); } }}
```

[Filename: RDF/SparqlDirectPOST.java]

LOD: Classes and Properties as Resources

- The RDFS and OWL metadata about classes and properties is also accessible.
- it is left to the owner of a service what is delievered then.
- Access to `http://dbpedia.org/ontology/Country`
  (or rapper http://dbpedia.org/ontology/Country)
  - owl:equivalentClass, rdfs:subClassOf statements belong here.
- Access to `http://dbpedia.org/ontology/name` or
  `http://rdf.insee.fr/def/demo#population`
  - owl:equivalentProperty, rdfs:domain, rdfs:range statements belong here.
- Mondial: there are a lot of (OWL) descriptions that cannot be described in a single triple and that use blank nodes
  - HTML access to classes or properties: triples about the resource and all its instances:
    `http://www.semwebtech.org/mondial/10/meta%23locatedAt`
    `http://www.semwebtech.org/mondial/10/meta%23Water`
  - RDF: all triples of the `mondial-meta.n3` file, including the blank nodes.
  - RDF access to base url `http://www.semwebtech.org/mondial/10/`:
    all is-a-triples.

LOD: SPARQL Service Descriptions

- The SPARQL Recomendation also defines "Service Descriptions" (of the SPARQL endpoints) at
  `https://www.w3.org/TR/2013/REC-sparql11-service-description-20130321/`:

```
> rapper http://dbpedia.org/sparql

<http://dbpedia.org/sparql> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
     <http://www.w3.org/ns/sparql-service-description#Service>;
  <http://www.w3.org/ns/sparql-service-description#endpoint> <http://dbpedia.org/sparql>;
  <http://www.w3.org/ns/sparql-service-description#feature>
     <http://www.w3.org/ns/sparql-service-description#UnionDefaultGraph>;
  <http://www.w3.org/ns/sparql-service-description#feature>
     <http://www.w3.org/ns/sparql-service-description#DereferencesURIs>;
  <http://www.w3.org/ns/sparql-service-description#resultFormat>
     <http://www.w3.org/ns/formats/SPARQL_Results_JSON>;
  <http://www.w3.org/ns/sparql-service-description#resultFormat>
     <http://www.w3.org/ns/formats/SPARQL_Results_XML>;
  <http://www.w3.org/ns/sparql-service-description#resultFormat> <http://www.w3.org/ns/formats/Turtle>;
  <http://www.w3.org/ns/sparql-service-description#resultFormat> <http://www.w3.org/ns/formats/N-Triples>
  <http://www.w3.org/ns/sparql-service-description#resultFormat> <http://www.w3.org/ns/formats/N3>;
  <http://www.w3.org/ns/sparql-service-description#resultFormat> <http://www.w3.org/ns/formats/RDF_XML>;
  <http://www.w3.org/ns/sparql-service-description#supportedLanguage>
     <http://www.w3.org/ns/sparql-service-description#SPARQL10Query>;
  <http://www.w3.org/ns/sparql-service-description#url> <http://dbpedia.org/sparql> .
```


SPARQL 1.1: Queries against LOD Services

- SPARQL subqueries against SPARQL Web Services:
  `SERVICE` *url* `{` *pattern* `}`
  `SERVICE` *url* `{ SELECT DISTINCT` *variables* `WHERE {` *pattern* `} }`
  - In the first case, *all* tuples of bindings of all variables bound in *pattern* are sent back.
  - The second variant applies a projection on the remote service first.
- see `https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/`
- error if the remote service does not answer;
- SERVICE SILENT: empty result in case of an error.

```
select distinct ?X
where { service <http://dbpedia.org/sparql>
        {?X a <http://dbpedia.org/ontology/Country> . } }
```
[Filename: RDF/lodsparql.sparql]

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select distinct ?X ?L
where { service <http://www.semwebtech.org/mondial/10/sparql>
        {?X a :Country; :localname ?L. } }
```
[Filename: RDF/lodmondial.sparql]

VALUES clause (mainly used internally)

Consider

```
select ?X ?Y ?Z
where { pattern1(?X,?Y)
        service <url> { remote_pattern/query(?Y,?Z) }
        pattern2(?X,?Y,?Z) }
```

(Possible) internal evaluation:

- evaluate *pattern*$_1$(?X,?Y),
- call *url* to evaluate *remote_pattern*(?Y,?Z) only for the values for ?Y that have been collected by *pattern*$_1$(?X,?Y), binds additionally ?Z,
  ("sideways information passing")
- evaluate *pattern*$_2$(?X,?Y,?Z)

The remote service will receive a query

```
select *
where { remote_pattern(?Y,?Z) .
        values ?Y { y1 y2 ... yn } }
```


VALUES clause constrains bindings

- although intended for peer-to-peer communication, the VALUES clause is also allowed for user queries:

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select distinct ?C ?CN
from <file:mondial.n3>
where { ?C a :Country; :carCode ?CC; :name ?CN .
        values ?CC { 'D' 'F' 'B' } }
```

[Filename: RDF/values-in.sparql]

- note the similarity with SQL's "`WHERE ... IN (...)`":
  SELECT name FROM country WHERE code IN ('D', 'F', 'B')

Distributed Queries - Mondial and insee.fr

Note: Distributed queries work only with the LOD SPARQL service at `http://www.semwebtech.org/mondial/10`; in the old /semweb one, ARQ destroys the syntax of the embedded subquery!

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix insee: <http://rdf.insee.fr/def/geo#>
select distinct ?XN ?I ?DN ?PNL
from <file:mondial.n3>
where { ?C a :Country; :carCode 'F'; :name ?CN ; :hasCity ?X.
        ?P a :Province; :hasCity ?X .
        ?X :name ?XN.
        ?P :name ?PN .
        bind (strlang(?XN,"fr") as ?XNL) .
        bind (strlang(?PN,"fr") as ?PNL) .
        ### optional
        { service <http://rdf.insee.fr/sparql>
          { ?I a insee:Commune; insee:nom ?XNL; insee:subdivisionDe ?DEPT .
            ?DEPT a insee:Departement;  insee:nom ?DN; insee:subdivisionDe ?REG .
            ?REG insee:nom ?PNL
      } } }
```

[Filename: RDF/distributed-insee.sparql]

Distributed Queries: Allround Testcase

- SERVICE pattern can occur in classical inner join, optional/outer join and in FILTER NOT EXISTS/antijoin
- common variable is ?CN
- note: ?X produces ?CN-duplicates (that must be eliminated before submitting to remote service)

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select distinct ?C ?CN ?CC # ?X
where { ?C a :Country; :name ?CN ; :hasCity ?X .
        ## optional
        ## filter not exists
        { # service <http://localhost:8080/mondial/10/sparql>
          service <http://www.semwebtech.org/mondial/10/sparql>
          { ?CC a :City; :name ?CN }
      }
}
```

[Filename: RDF/distributed-testcase.sparql]

Distributed Queries - Mondial and insee.fr

- Variable in filter with equals
- note: (original) Jena does not communicate variables in a filter [3.2019]

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix insee: <http://rdf.insee.fr/def/geo#>
select distinct ?XN ?I ?DN ?PNL
from <file:mondial.n3>
where { ?C a :Country; :carCode 'F'; :name ?CN ; :hasCity ?X.
        ?P a :Province; :hasCity ?X .
        ?X :name ?XN.
        ?P :name ?PN .
        ### optional
        { service <http://rdf.insee.fr/sparql>
          { ?I a insee:Commune; insee:nom ?XNL; insee:subdivisionDe ?DEPT .
            ?DEPT a insee:Departement;  insee:nom ?DN; insee:subdivisionDe ?REG .
            ?REG insee:nom ?PNL .
            FILTER (str(?XNL) = ?XN && str(?PNL) = ?PN)
          }
      }
}
```

[Filename: RDF/distributed-filter-equals.sparql]

Distributed Queries Testcase: Variable in Filter

- Variable in filter with numeric comparison
- note: Jena does not communicate variables in a filter

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix insee: <http://rdf.insee.fr/def/geo#>
prefix inseeD: <http://rdf.insee.fr/def/demo#>
select distinct ?PN ?PN ?Pop ?XNL ?Pop2
from <file:mondial.n3>
where { ?C a :Country; :carCode 'F'; :name ?CN ; :hasProvince ?P .
        ?P :name ?PN; :population ?Pop . bind (strlang(?PN,"fr") as ?PNL) .
        { service <http://rdf.insee.fr/sparql>
          { ?I a insee:Commune; insee:nom ?XNL; insee:subdivisionDe ?DEPT .
            ?DEPT a insee:Departement;  insee:nom ?DN; insee:subdivisionDe ?REG .
            ?REG insee:nom ?PNL .
            ?I inseeD:population ?PopItem .
            ?PopItem inseeD:date ?D; inseeD:populationTotale ?Pop2 .
            FILTER (year(?D)=2015 && ?Pop2 > 200000 && ?Pop2 > 0.05 * ?Pop)
      } } }
```

[Filename: RDF/distributed-filter-comp.sparql]

- note: the filter is not “filter-safe” according to Slide 163
- “Push-into” to make a subquery self-contained is not applicable in the distributed case!

Distributed Queries Testcase: Service in NOT EXISTS

- Variable in filter with numeric comparison
- note: Jena 3.10 does not communicate variables in a filter

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
prefix insee: <http://rdf.insee.fr/def/geo#>
prefix inseeD: <http://rdf.insee.fr/def/demo#>
select distinct ?PN ?Pop
from <file:mondial.n3>
where { ?C a :Country; :carCode 'F'; :name ?CN ; :hasProvince ?P .
        ?P :name ?PN; :population ?Pop . bind (strlang(?PN,"fr") as ?PNL) .
        FILTER NOT EXISTS
        { service <http://rdf.insee.fr/sparql>
          { ?I a insee:Commune; insee:nom ?XNL; insee:subdivisionDe ?DEPT .
            ?DEPT a insee:Departement;  insee:nom ?DN; insee:subdivisionDe ?REG .
            ?REG insee:nom ?PNL .
            ?I inseeD:population ?PopItem .
            ?PopItem inseeD:date ?D; inseeD:populationTotale ?Pop2 .
            FILTER (year(?D)=2015 && ?Pop2 > 200000 && ?Pop2 > 0.05 * ?Pop)
      }}  }
```

[Filename: RDF/distributed-minus-service.sparql]

Distributed Queries: Trivial Testcase

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
SELECT ?c ?name
WHERE {
   ?c a :Continent.
   ## SERVICE <http://localhost:8080/mondial/10/sparql>{
   SERVICE <http://www.semwebtech.org/mondial/10/sparql>{
   ?c :name ?name.
   }
}
```

[Filename: RDF/distributed-trivial.sparql]

Distributed Queries: Variable Visibility Testcase

```
prefix : <http://www.semwebtech.org/mondial/10/meta#>
select ?C ?N
where { ?C a :Country ; :name ?N .
        service <http://localhost:8080/mondial/10/sparql>
          { select distinct ?C   ## ?N
            where
            { ?C :hasCity ?X . ?X :name ?N
          } }
}
```

[Filename: RDF/service-projection-test-mondial.sparql

- the inner ?N is logically local, *not* to bei a join condition with the outer one.
- correct eval: 244 countries.
- wrong eval: 10 countries (having a city name = country name)
- original Jena 3.10: correct eval.

Distributed Queries - Mondial and wikidata

```
prefix owl: <http://www.w3.org/2002/07/owl#>
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
prefix : <http://www.semwebtech.org/mondial/10/meta#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
select distinct ?C ?CN ?label
from <file:mondial.n3>
from <file:mondial-sameas.n3>
where { ?C a :Country;  :name ?CN ; :hasCity ?X.
        ?X :name ?XN; owl:sameAs ?URL
        SERVICE <https://query.wikidata.org/sparql>
          {
            ?URL wdt:P31/wdt:P279* wd:Q7930989 .
            ?URL wdt:P17 ?wc .    ### (in) country
            ?wc rdfs:label "France"@en .
            ?URL rdfs:label ?label.
            FILTER (langMatches( lang(?label), "fr")) .
          }
       ### values ?url ( ...)
}
```

[Filename: RDF/distributed-wikidata.sparql]

- inner query (wikidata) 91 results (in France; 49000 city/towns worldwide)
- outer query (mondial cities with sameAs): 1300/1800
- intersection: 16 cities
  - eval outer (1ms), get 1300 results, state 1300 individual queries with ?URL replaced (each 0.3s) $\rightarrow$ 400sec.
  - eval outer, get 1300 results, state inner query with 1300 values (??sec)
  - eval outer (1ms), eval inner (1.4 sec), join (1ms)

Chapter 8
Statements about Statements

Sometimes, statements/triples need annotations, (temporal) qualifications, references etc.

- Reification as a concept in RDF: Statements
- Complex representation based on plain RDF

8.1 Further RDF Vocabulary: Reification

Take statements (=triples) as resources and make statements about them:

- `rdf:Statement` which has properties `rdf:subject`, `rdf:predicate`, `rdf:object`, that yield a resource.
- XML: give an ID to the statement.
  "The statement "Germany had 83536115 inhabitants" was valid in year 1997":

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <mon:Country rdf:about="countries/D">
    <mon:name>Germany</mon:name>
    <mon:population rdf:ID="de-pop">83536115</mon:population>
  </mon:Country>
  <rdf:Description rdf:about="#de-pop">
    <mon:year>1997</mon:year>
  </rdf:Description>
</rdf:RDF>
```

[Filename: RDF/reification.rdf]

REIFICATION: EXAMPLE

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <mon:Country rdf:about="countries/D">
    <mon:name>Germany</mon:name>
    <mon:population rdf:ID="de-pop">83536115</mon:population>
  </mon:Country>
  <rdf:Description rdf:about="#de-pop">
    <mon:year>1997</mon:year>
  </rdf:Description>
</rdf:RDF>
```

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?X ?P ?V
from <file:reification.rdf>
where {?X a rdf:Statement; ?P ?V}
```

[Filename: RDF/reification.sparql]

Triples (added automatically by the RDF semantics)
(use `jena -t -if reification.rdf` or see RDF validator):
(<http://.../mondial/10/#de-pop> rdf:type rdf:Statement)
(<http://.../mondial/10/#de-pop> rdf:subject <http://.../mondial/10/countries/D>)
(<http://.../mondial/10/#de-pop> rdf:predicate <http://.../mondial/10/meta#population>)
(<http://.../mondial/10/#de-pop> rdf:object "83536115")
(<http://.../mondial/10/#de-pop> <http://.../mondial/10/meta#year> "1997")

... the above annotates a statement that is assumed to hold.

Reification in the Semantic Web

Annotating Statements:

- with probabilities, trust, “who says ...”, even negation!
- annotations can be in different files than the annotated statements (cf. out-of-line XLink arcs),
- can be used for reasoning.

Note:

- information about a *predicate* (which describes the predicate “name” (e.g., the source where all this data is taken from, transitivity or symmetry) or the set of all instances (cardinalities), or each its (range, domain))

is different

- from describing/annotating a *statement* (i.e. one instance of a predicate).

Reification and Annotation

- Statements that do *not* hold can also by annotated:

```
<!DOCTYPE rdf:RDF [ <!ENTITY mon "http://www.semwebtech.org/mondial/10/meta#"> ] >
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:mon="http://www.semwebtech.org/mondial/10/meta#"
     xml:base="http://www.semwebtech.org/mondial/10/">
  <rdf:Statement rdf:ID="cap-d-bonn">  <!-- ******************** BONN ***************** -->
    <rdf:subject rdf:resource="countries/D/"/>
    <rdf:predicate rdf:resource="&mon;capital"/>
    <rdf:object rdf:resource="countries/D/provinces/NordrheinWestfalen/cities/Bonn"/>
    <mon:from>1949</mon:from>
    <mon:until>1990</mon:until>
  </rdf:Statement>
  <rdf:Description rdf:about="countries/D/"> <!-- *************** BERLIN *************** -->
   <mon:capital rdf:ID="cap-d-berlin"
     rdf:resource="countries/D/provinces/Berlin/cities/Berlin"/>
  </rdf:Description>
  <rdf:Description rdf:about="#cap-d-berlin">
    <mon:from>1990</mon:from>
  </rdf:Description>
</rdf:RDF>
```

[Filename: RDF/reification-2.rdf]

Reification and Annotation (Cont'd)

- queries against the above information

```
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix mon: <http://www.semwebtech.org/mondial/10/meta#>
select ?S ?X ?Y ?F ?U
from <file:reification-2.rdf>
where {{?X mon:capital ?Y} UNION
       {?S rdf:subject ?X ; rdf:predicate mon:capital; rdf:object ?Y
        OPTIONAL {?S mon:from ?F} . OPTIONAL {?S mon:until ?U}}}
```

[Filename: RDF/reification-2.sparql]

- (?X capital ?Y) contains only (Germany, Berlin), the *triple* that actually holds.
- The statement (Germany, capital, Bonn) is not a triple, but there is only annotation about it.

Annotated Knowledge Representation

- A knowledge base in a setting where annotations are important, thus would not contain any explicit triples but only described statements
- Applications must then interpret the semantics of the annotations:
  - heuristics to generate those triples that are assume to hold actually:
  - annotation of probabilities, opinion, provenance, trust, ...
  - if an RDF source contains a statement that is somewhere else annotated that it held only in earlier times, discard it,
  - if some statement does not hold, but is e.g. annotated as believed by a trusted person, consider it to be true.
- example: Wikidata, see next section.

8.2 Wikidata: a Linked Open Data Source

Wikidata uses an own modeling:

- query interface at `https://query.wikidata.org/`
- fun factor: low – wikidata poses a time limit on the evaluation of queries that can hardly be met even for rather simple examples
- not a "simple database" but a comprehensive, multilingual "knowledge graph".
- uses just plain RDF data + rdfs:label, no rdfs:subClassOf, no OWL, no reasoning at all.
- cf. paradoxes (Slide 224) whenever something is both an instance and a class (e.g. "penguin" as a the class of all penguins, and an instance "penguin is a species")
- everything (things/classes and also properties) has an rdfs:label in most of the wikipedia/wikidata languages
  $\Rightarrow$ properties/classes/things are not identified by their name (in any language), but just by a number
  (e.g. P36 is "has capital", P31 means "instance of", and P279 means "subclass of")
- statements are intensively annotated
- class hierarchy *without inherent transitivity* (no reasoning, partially redundant storage)

Wikidata Things and Properties

The query interface `https://query.wikidata.org/` has predefined prefixes for a set of wikidata namespaces:

- wd: is the namespace for all wikidata things (entities and/or classes)

  ```
  PREFIX wd: <http://www.wikidata.org/entity/>
  ```

  URI of Germany: `https://www.wikidata.org/entity/Q183`
  When browsing, this is automatically redirected to
  `https://www.wikidata.org/wiki/Q183`

- wdt: is the wikidata namespace for "direct" properties (i.e., the application-level properties in real-world triples):

  ```
  PREFIX wdt: <http://www.wikidata.org/prop/direct/>
  ```

  URI of the "capital" property: `http://www.wikidata.org/prop/direct/P36`
  When browsing, this is automatically redirected to
  `https://www.wikidata.org/wiki/Property:P36`

- everything has rdfs:label-s in many languages.

Wikidata Classes and Class Hierarchy

```
SELECT DISTINCT ?c ?cn ?capn
WHERE {
   ?c wdt:P31 wd:Q6256.          ## instanceOf country
   filter not exists {?c wdt:P576 ?X}  ## dissolved date
   ?c rdfs:label ?cn.
   ?c wdt:P36  ?cap . ?cap rdfs:label ?capn  ## capital
   FILTER ((langMatches(lang(?cn),"en") && langMatches(lang(?capn),"en")) ||
           (langMatches(lang(?cn),"ru") && langMatches(lang(?capn),"ru"))    )
   ## FILTER (contains(str(?cn),'Germany'))
}                                  [Filename: RDF/wikidatacountrycaps1.sparql]
```

- wd:Q6256 (see `https://www.wikidata.org/wiki/Q6256`) has broad meaning, Q3624078 means “souvereign state” (still including ancient ones). (but with this, it runs into a server error; 14.1.2019)
- Germany does not belong (directly) to Q3624078.

Wikidata Classes and Class Hierarchy

- use P31 (instance of) and P279 (subclass of) in a path expression

```
SELECT DISTINCT ?c ?cn ?capn
WHERE {
   ?c wdt:P31/wdt:P279* wd:Q6256.   ## path: isa/subclass*
   filter not exists {?c wdt:P576 ?X}  ## dissolved date
   ?c rdfs:label ?cn.
   ?c wdt:P36  ?cap . ?cap rdfs:label ?capn  ## capital
   FILTER ((langMatches(lang(?cn),"en") && langMatches(lang(?capn),"en")) ||
           (langMatches(lang(?cn),"ru") && langMatches(lang(?capn),"ru"))    )
   ## FILTER (contains(str(?cn),'Germany'))
}                                  [Filename: RDF/wikidatacountrycaps2.sparql]
```

Basic Knowledge Graph + Multilingual labels

- the basic graph of the resources and statements exists only once, based on the wd:Qxxx (things), wdt:Pyyy (properties) and wds:zzz (statements),
- Qxxx and Pyyy have labels in each/most of the languages.

Wikidata: Built-In Handling for Labels and Languages

The service supports several extensions to standard SPARQL capabilities:
(see `https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual`

- for objects bound to a variable ?X, automatically bind ?XLabel (if SELECTed) to the label(s) in selected languages:

```
PREFIX wikibase: <http://wikiba.se/ontology#>
 SERVICE wikibase:label {
     bd:serviceParam wikibase:language "de,en" .
 }
```

```
SELECT DISTINCT ?c ?cLabel ?capLabel
WHERE {
   ?c wdt:P31/wdt:P279* wd:Q6256.   ## path: isa/subclass*; Q6256=country
   filter not exists {?c wdt:P31/wdt:P279* wd:Q3024240}  ## historical country
   filter not exists {?c wdt:P576 ?x}  ## dissolved date
   ?c wdt:P36  ?cap .  ## capital
SERVICE wikibase:label { bd:serviceParam wikibase:language "en,fr" }
}
```

[Filename: RDF/wikidatacountrycaps3.sparql]

Combination of Wikidata and Mondial/ SPARQL SERVICE clause

```
PREFIX : <http://www.semwebtech.org/mondial/10/meta#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?mcont ?wdcont ?labelWithoutLanguageTag
FROM <http://www.dbis.informatik.uni-goettingen.de/Mondial/Mondial-RDF/mondial.n3>
WHERE {
    SERVICE <https://query.wikidata.org/sparql>{
        ?wdcont wdt:P31 wd:Q5107.     ### instanceOf continent
        ?wdcont rdfs:label ?label.
        FILTER (langMatches( lang(?label), "*" ) )
        BIND(STR(?label) AS ?labelWithoutLanguageTag)
    }
    ?mcont a :Continent.
    ?mcont :name ?labelWithoutLanguageTag.
}
```

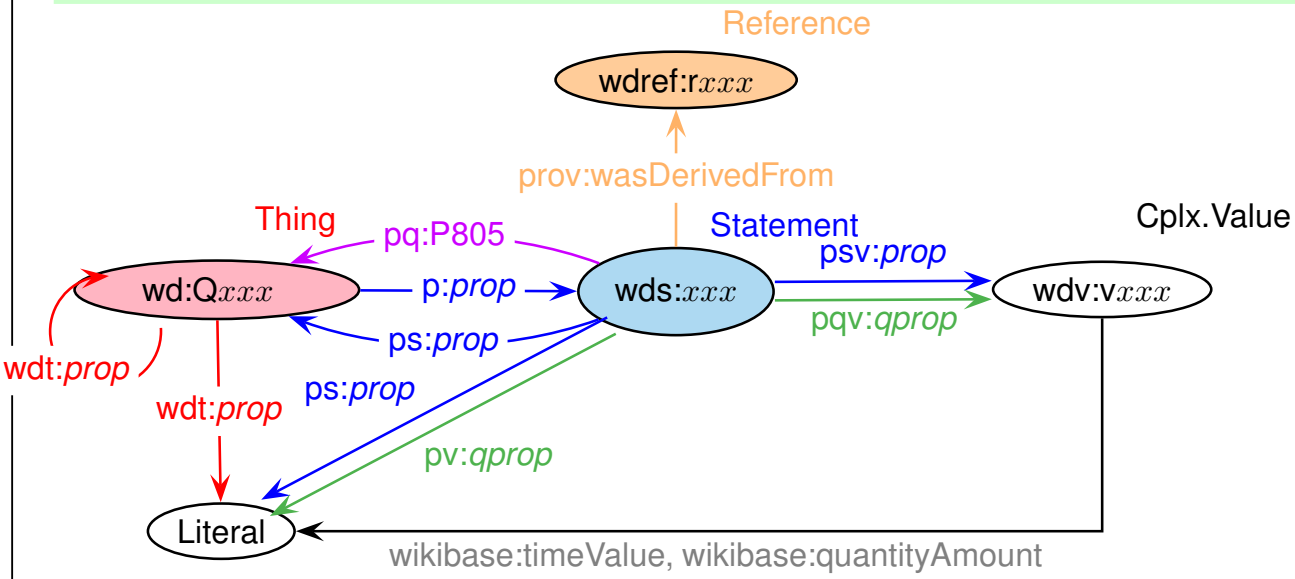
[Filename: RDF/lodwikidata.sparql]

Wikidata: Prefixes and types of edges and nodes (Graphics see next slide)

(see `https://www.mediawiki.org/wiki/Wikibase/Indexing/RDF_Dump_Format`)

- wdt:*prop*: "real-world" edges: object $\overset{prop}{\rightarrow}$ things | literals | cplxValues
- p:*prop*: thing → *prop*-statements (one statement for each filler of property *prop* of a thing)
- ps:*prop*: *prop*-statement → thing | literal
- psv:*prop*: *prop*-statement → cplxValue if the value is a complex one
- wikibase:timeValue (→ xsd:date) and wikibase:quantityAmount (→ numeric): cplxValue → literal
  wikibase:timeCalendarModel, wikibase:quantityUnit (units),
  wikibase:geoLatitude/geoLongitude (for type wikibase:GlobecoordinateValue)
- pq:*qprop*: qualifier properties: statement → literal
  (start, end, timepoint ... as xsd:date)
  pqv:*qprop*: qualifier properties: statement → cplxValue (time values)
- pq:P805: statement → Thing – "isSubjectOf", better name would be "is reified as"
  ⇒ Thing p:*prop*/pq:P805 → Thing leads to reified real-world things.
- prov:wasDerivedFrom: statement → Reference – *Provenance*: references where the statement can be derived from.

Wikidata: Prefixes and types of edges and nodes



- wdt.*prop* = p:*prop* ∘ ps:*prop*
- wdt.*prop* = p:*prop* ∘ psv:*prop* ∘ (wikibase:timeValue | wikibase:quantityAmount)
  (complex values)
- properties pointing to objects that represent reified properties:
  wdt.*prop'* = p:*prop* ∘ pq:P805 ?? (Country: hasMembership = p:isMember ∘ pq:P805)

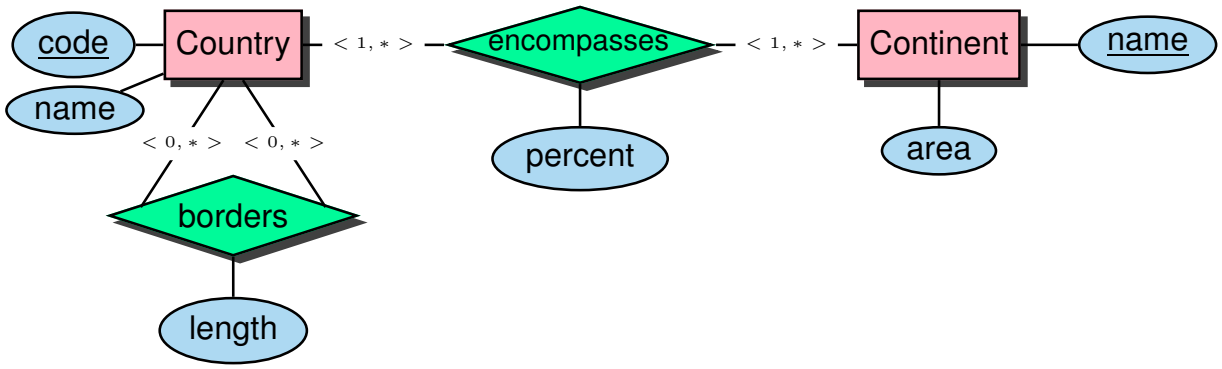
Wikidata: Temporal qualifiers: capitals

```
SELECT ?cl ?capl ?from ?fr ?until ?un
WHERE
{ ?c wdt:P31/wdt:P279* wd:Q3624078 . ## sovereign state
  FILTER NOT EXISTS { ?c wdt:P576 ?X }  ### not dissolved date
  ?c p:P36 ?stmt .       ## P36: capital
    ?stmt ps:P36 ?cap ;
          optional { ?stmt pq:P580 ?from }
          optional { ?stmt pq:P582 ?until } .
  ?c rdfs:label ?cl .
  ?cap rdfs:label ?capl .
  filter (lang(?cl)="en")
  filter (lang(?capl)="en")
  bind (str(?from) as ?fr) . bind (str(?until) as ?un)  ### as xsd:dates
  # filter (str(?cl) = "Germany")
}
order by ?cl ?fr
```

[Filename: RDF/wikidata-capital.sparql]

Wikidata: Properties with Properties – Reification

- Recall from DB lectures: the issue of *reification* of attributes occurs in many data models (UML, XML, ... and also in RDF)



- Consider that Germany borders Austria with a length of 815km.
- Wikidata: the Wikidata-Statement-Object has a “isSubjectOf” (better name would be “is reified as”) (pq:P805) property that points to a Wikidata resource (here, Q1991986) which is the “Austria-Germany border” and has P2043 (length) with a value of 815.

Wikidata: Reification qualifiers: border length

```
SELECT ?cl ?neighl ?border ?bl ?len ?len2
WHERE
{ ?c wdt:P31/wdt:P279* wd:Q3624078 . ## sovereign state
  FILTER NOT EXISTS { ?c wdt:P576 ?X }  ### not dissolved date
  ?c p:P47 ?stmt .       ## P47: shares border with
    ?stmt ps:P47 ?neighb .
  FILTER NOT EXISTS { ?neighb wdt:P576 ?Y }  ### not dissolved date
  ?c rdfs:label ?cl .
  ?neighb rdfs:label ?neighl .
  ?stmt pq:P805 ?border .   ### P805: isSubjectOf - "Reification"
  ?border rdfs:label ?bl ;
          wdt:P2043 ?len; p:P2043/ps:P2043 ?len2.  ### P2043: length
  filter (lang(?cl)="en")
  filter (lang(?neighl)="en")
  filter (lang(?bl)="en")
  ## filter (str(?cl) = "Germany")
}
order by ?cl ?neighl
```

[Filename: RDF/wikidata-borderlength.sparql]

Wikidata: countries encompassed by continents

```
SELECT DISTINCT ?cl ?contl ?PL ?VL ?reif
WHERE
{ ?c wdt:P31/wdt:P279* wd:Q3624078 . ## sovereign state
  FILTER NOT EXISTS { ?c wdt:P576 ?X }  ### not dissolved date
  ?c p:P30 ?stmt .       ## P30: continent
    ?stmt ps:P30 ?cont .
  ?c rdfs:label ?cl .
  ?cont rdfs:label ?contl .
  optional { ?stmt ?P ?V . ?P rdfs:label ?PL . ?V rdfs:label ?VL .
             filter (lang(?PL)="en")
             filter (lang(?VL)="en") }
  optional {?stmt pq:P805 ?reif} .   ### P805: isSubjectOf - "Reification"
  filter (lang(?cl)="en")
  filter (lang(?contl)="en")
}
order by ?cl ?contl
```

[Filename: RDF/wikidata-encompassed.sparql]

- no bindings for ?P, ?V, or ?reif.

$\Rightarrow$ no information apart from the pure “X is (partly) located on continent Y”

Wikidata: ethnic groups in countries

```
SELECT DISTINCT ?cl ?ethl ?PL ?VL ?reif
WHERE
{ ?c wdt:P31/wdt:P279* wd:Q3624078 . ## sovereign state
  FILTER NOT EXISTS { ?c wdt:P576 ?X }  ### not dissolved date
  ?c p:P172 ?stmt .       ## P172: ethnic group
      ?stmt ps:P172 ?eth .
  ?c rdfs:label ?cl .
  ?eth rdfs:label ?ethl .
  optional { ?stmt ?P ?V . ?P rdfs:label ?PL . ?V rdfs:label ?VL .
             filter (lang(?PL)="en")
             filter (lang(?VL)="en") }
  optional {?stmt pq:P805 ?reif} .   ### P805: isSubjectOf - "Reification"
  filter (lang(?cl)="en")
  filter (lang(?ethl)="en")
}
order by ?cl
```

[Filename: RDF/wikidata-ethnics.sparql]

- no bindings for ?P, ?V, or ?reif $\rightarrow$ no percentages.
- no information about languages and religions in a country

Wikidata: asking for property labels

Each property Pxxx "exists" in many different ways:

- wd:Pxxx is the property as an object – it has an rdfs:label
- all usages as wdt:Pxxx, p:Pxxx, ps:Pxxx etc do not have labels.
- for querying "return all properties (with names) of an object", the following connections between wd:Pxxx and the other usages must be used:

```
wd:Pxxx a wikibase:Property ;
    wikibase:directClaim wdt:Pxxx ;
    wikibase:claim p:Pxxx ;
    wikibase:statementProperty ps:Pxxx ;
    wikibase:statementValue psv:Pxxx ;
    wikibase:qualifier pq:Pxxx ;
    wikibase:qualifierValue pqv:Pxxx ;
    wikibase:reference pr:Pxxx ;
    wikibase:referenceValue prv:Pxxx ;
    wikibase:novalue wdno:Pxxx .
```

Wikidata: reified things

- this query lists all properties PL of countries whose statements are reified as objects (YL), and optionally gives the propertyname (x RPL y)

```
SELECT DISTINCT ?cl ?PL ?YL ?RPL1 ?RPL2  WHERE
{ ?c wdt:P31/wdt:P279* wd:Q3624078 . ## sovereign state
  FILTER NOT EXISTS { ?c wdt:P576 ?X }  ### not dissolved date
  ?c rdfs:label ?cl .  filter (lang(?cl)="en")
  filter (str(?cl) = "Germany")
  ?c ?P ?stmt . ?stmt pq:P805 ?Y .  ## p:P-stmt reified as Y
  ?Prop wikibase:claim ?P; rdfs:label ?PL . filter (lang(?PL)="en")
  ?Y rdfs:label ?YL .   filter (lang(?YL)="en")
  ### link between ?c and ?Y or back: must be in wdt:rprop
  optional { ?c ?rp1 ?Y .
             ?Rprop wikibase:directClaim ?rp1 .
             ?Rprop rdfs:label ?RPL1 . filter (lang(?RPL1)="en") }
  optional { ?Y ?rp2 ?c .
             ?Rprop wikibase:directClaim ?rp2 .
             ?Rprop rdfs:label ?RPL2 . filter (lang(?RPL2)="en") }
} order by ?cl
```

[Filename: RDF/wikidata-reified.sparql]

Answer e.g. : cl/Germany, PL/shares border with, YL/Austria-Germany border, RPL2/country

Wikidata: Temporal qualifiers/data quality

- quality is better than DBpedia, but not perfect:

```
SELECT *
WHERE
{ ?uni rdfs:label ?val .
  ?uni wdt:P31/wdt:P279* wd:Q3918 .     ## Q3918: University
  ?uni p:P17 ?stmt .       ## P17: located in country
    ?stmt ps:P17 ?C ;
          optional { ?stmt pq:P580 ?start }
          optional { ?stmt pq:P582 ?end } .
  ?C rdfs:label ?cl
  filter (lang(?cl)="en")
  filter (lang(?val)="en")
  filter (str(?val) = "Moscow State University")
}
```

[Filename: RDF/lomonossow.sparql]

- Countries: Russia, Soviet Union, Russian Empire
  the “start” and “end” entries are often missing.