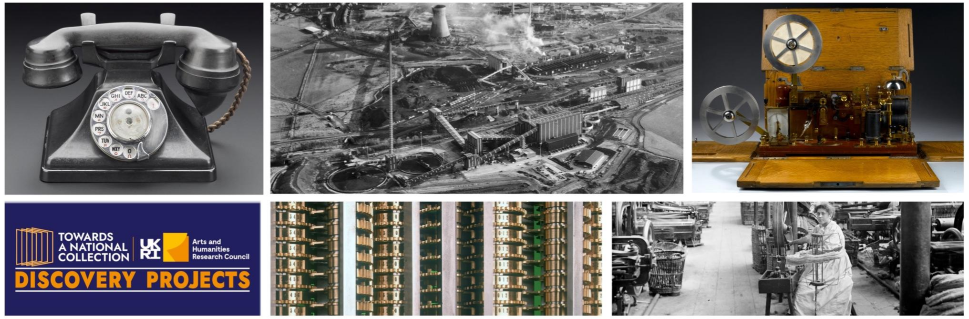
Congruence Engine - Training and Mutual Learning Working Group

Introduction to the Principles of Linked Open Data

Dr Anna-Maria Sichani
17 May 2023


TOWARDS A NATIONAL COLLECTION
UKRI
Arts and Humanities Research Council
DISCOVERY PROJECTS

SCIENCE MUSEUM GROUP

SCHOOL OF ADVANCED STUDY | UNIVERSITY OF LONDON

Content of the session

- core concepts of Linked Open Data - Key semantic Web technologies
- URIs
- Ontologies
- RDF formats
- gentle intro to the graph query language SPARQL

https://programminghistorian.org/en/lessons/intro-to-linked-data

5-star Linked Open data

★	Available on the web (whatever format) *but with an open licence, to be Open Data*
★★	Available as machine-readable structured data (e.g. excel instead of image scan of a table)
★★★	as (2) plus non-proprietary format (e.g. CSV instead of excel)
★★★★	All the above plus, Use open standards from W3C (RDF and SPARQL) to identify things, so that people can point at your stuff
★★★★★	All the above, plus: Link your data to other people's data to provide context

https://www.w3.org/DesignIssues/LinkedData.html

Linked open data: what is it?

Linked Open Data (LOD)

- is structured information in a format meant for machines.
- are inherently interoperable, helping to achieve a specific type of machine-executable interoperability known as semantic interoperability, which relies on linking data via common vocabularies or Knowledge Organisation Systems (KOS).
- have the potential to play a key role in implementing the "I" in FAIR.
- are machine-readable, based on a standard data representation (RDF - Resource Description Format) and are seen as epitomizing the ideals of open data (see https://5stardata.info/en/).

If all datasets were openly published, and used the same format for structuring information, it would be possible to interrogate all of the datasets at once. Analysing huge volumes of data is potentially much more powerful than everyone using their own individual datasets dotted around the web in what are known as information silos. These interoperable datasets are what LOD practitioners are working towards.

Principles when working with LOD

1. **Use a recognised LOD standard format**. In order for LOD to work, the data must be structured using recognised standards so that computers interrogating the data can process it consistently. There are a number of LOD formats.
2. **Refer to an entity the same way other people do**. If you have data about the same person/place/thing in two or more places, make sure you refer to the person/place/thing the same way in all instances.

   PIDs , URI, authority files

3. **Publish your data openly**. By openly I mean for anyone to use without paying a fee and in a format that does not require proprietary software.

Starting with LOD

Let’s create an example using **Jack Straw, as a person:**

1)a fourteenth-century English rebel and

2) a UK cabinet minister prominent in Tony Blair’s administration. It is clearly useful to be able to differentiate the two people who share a common name.

- using a **attribute-value pair**

  `person=number`

For the Jack Straw , ‘the enigmatic rebel leader’, `person=33059614`

For the UK minister Jack Straw, `person=64183282`

Assuming everyone uses these two numbers to refer to the respective Jack Straws, if you are searching in a linked open dataset , you can be confident that we are getting the right person.

The attribute-value pairs can also store information about other types of entities**: places,** for example.

Jack Straw the modern politician was a member of British Parliament, representing the seat of Blackburn. There’s more than one place in the UK called Blackburn, not to mention other Blackburns around the world. We can disambiguate between the

Let’s assign a unique identifier to the correct place: Blackburn in Lancashire, England.

`place=2655524`

Starting with LOD

! “that’s what a library catalogue does” !

authority file = a definitive list of terms which can be used in a particular context, for example when cataloguing a book

For both of the examples outlined above, we have used **authority files** to assign the numbers (the unique ids) to the Jacks and to Blackburn.

- The numbers we used for the two Jack Straws come from the Virtual International Authority File (VIAF), which is maintained by a consortium of libraries worldwide to try to address the problem of the myriad ways in which the same person might be referred to.
- The unique identifier we used for the Blackburn constituency came from GeoNames, a free geographical database.

Starting with LOD

But let's try to be more precise by what we mean by Blackburn in this instance. Jack Straw represented the parliamentary constituency (an area represented by a single member of parliament) of Blackburn, which has changed its boundaries over time. The '[Digging Into Linked Parliamentary Data]' (Dilipad) project produc*ed unique identifiers for party affiliations and constituencies for each member of parliament.* In this example, Jack Straw represented the constituency known as 'Blackburn' in its post-1955 incarnation:

`blackburn1955-current`

- As VIAF is a well respected and well looked after authority file of notable people, it was an obvious set of identifiers to use for Jack Straw.
- As the constituency represented by Straw was covered perfectly by the authority files created by the Dilipad project, it too was a logical authority file to use.
- Unfortunately, it is not always so obvious which of the published lists online is best to use. One might be more widely used than another, but the latter might be more comprehensive for a particular purpose.
- GeoNames would work better than the Dilipad identifiers in some cases.
- There will also be cases where you can't find a dataset with that information. For example, imagine if you wanted to write attribute-values pairs about yourself and your immediate family relationships. In this case you would have to **invent your own identifiers.**

This lack of consistent authority files is one of the major challenges LOD is facing at the moment.

From unique IDs to relationships - triples

Once all elements are assigned unique identifiers, the next key step in creating LOD is to have a way of ***describing*** the relationship between Jack Straw (64183282) and Blackburn (blackburn1955-current).

In LOD, relationships are expressed using what's known as a (semantic) **'triple'**. Let's make a triple that represents the relationship between Jack Straw and his constituency:

```
person:64183282 role:representedInUKParliament constituency:"blackburn1955-current"
```

The triple, not surprisingly, has three parts (**structure**). The **syntax** will be discussed later on (RDF)

These are conventionally referred to as subject, predicate and object:

the subject	**the predicate**	**the object**
person 64183282	representedInUKParliament	"blackburn1955-current"

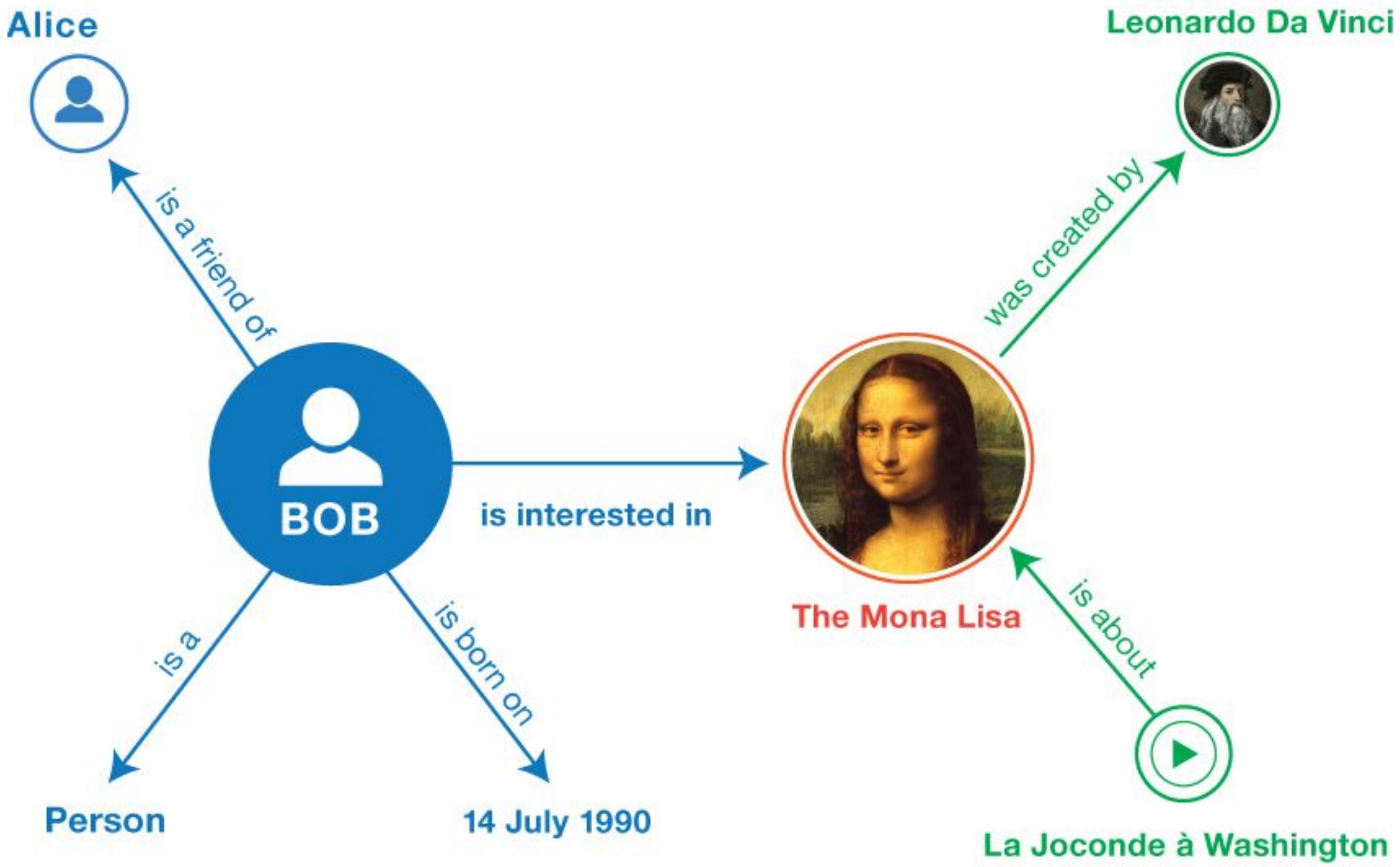
From unique IDs to relationships - triples

The traditional way to represent a triple in diagrammatic form is:

the classic way to represent a triple

So our Jack Straw triple, in more human-readable form, could be represented like this:

triple diagram showing that Jack Straw represented Blackburn



https://www.w3.org/TR/rdf11-primer/

Quick recap - key points about LOD

For now there are three key points to remember:

- LOD must be open and available to anyone on the internet (otherwise it isn’t ‘open’)
- LOD advocates aim to standardise ways of referring to unique entities
- LOD consists of triples which describe relationships between entities

The role of the Uniform Resource Identifier (URI)

An essential part of LOD is the Uniform Resource Identifier, or **URI.**
The URI is a reliably unique way of representing an entity (a person, an object, a relationship, etc), in a fashion that is **usable by everyone in the world.**
In the previous section we used two different numbers to identify our two different Jack Straws.

```
person="64183282"

person="33059614"
```

The problem is that around the world there are many databases that contain people with these numbers, and they're probably all different people. Outside of our immediate context these numbers don't identify unique individuals. Let's try to fix that. Here are these same identifiers but as URIs:

```
http://viaf.org/viaf/64183282/

http://viaf.org/viaf/33059614/
```

The role of the Uniform Resource Identifier (URI)

- Just as the unique number disambiguated our two Jack Straws, the full URI above helps us disambiguate between all of the different authority files out there. In this case, it’s clear that we are using **VIAF as our authority file.**
- **All** components of a triple could be expressed **via URIs (subject - predicate - object).**
- URIs also have to be unique.

URI VS URL

All URLs are URIs but not all URIs are URLs. A URI can describe anything at all, whereas URL describes the location of something on the web. So a URL tells you the location of a web page or a file or something similar. **A URI just does the job of identifying something**. Just as the International Standard Book Number, or ISBN 978-0-1-873354-6 uniquely identifies a hardback edition of *Baptism, Brotherhood and Belief in Reformation Germany* by Kat Hill, but doesn’t tell you where to get a copy.

The role of the Uniform Resource Identifier (URI)

URIs characteristics:

1. People talk about whether they are, or are not, **dereferenceable.** That just means *can it be turned from an abstract reference into something else.* For example, if you paste a URI into the address bar of a browser, will it return something? The VIAF URI for historian Simon Schama is:

http://viaf.org/viaf/46784579

If you put that into the browser you will get back a web page about Simon Schama which contains structured data about him and his publishing history. This is very handy - for one thing, it's not obvious from the URI who or even what is being referred to. But this is not essential. **Lots of URIs are not dereferenceable; it is a convention.**

2. Don't make them up unless you have to. People and organisations (like VIAF, Getty, Wikidata) have been making concerted efforts to construct good URI lists and LOD isn't going to work effectively if people duplicate that work by creating new URIs unnecessarily.

* If you can't find some people in VIAF, or other authority lists, only then might you need to make up your own.

How LOD organises knowledge: ontologies

LOD can answer complex questions. When you put triples together then they form a graph, because of the way that the triples interlink.

Suppose we want to find a list of all the people who were pupils of the composer Franz Liszt. If the information is in triples of linked data about pianists and their teachers, we can find out with a query on triples:

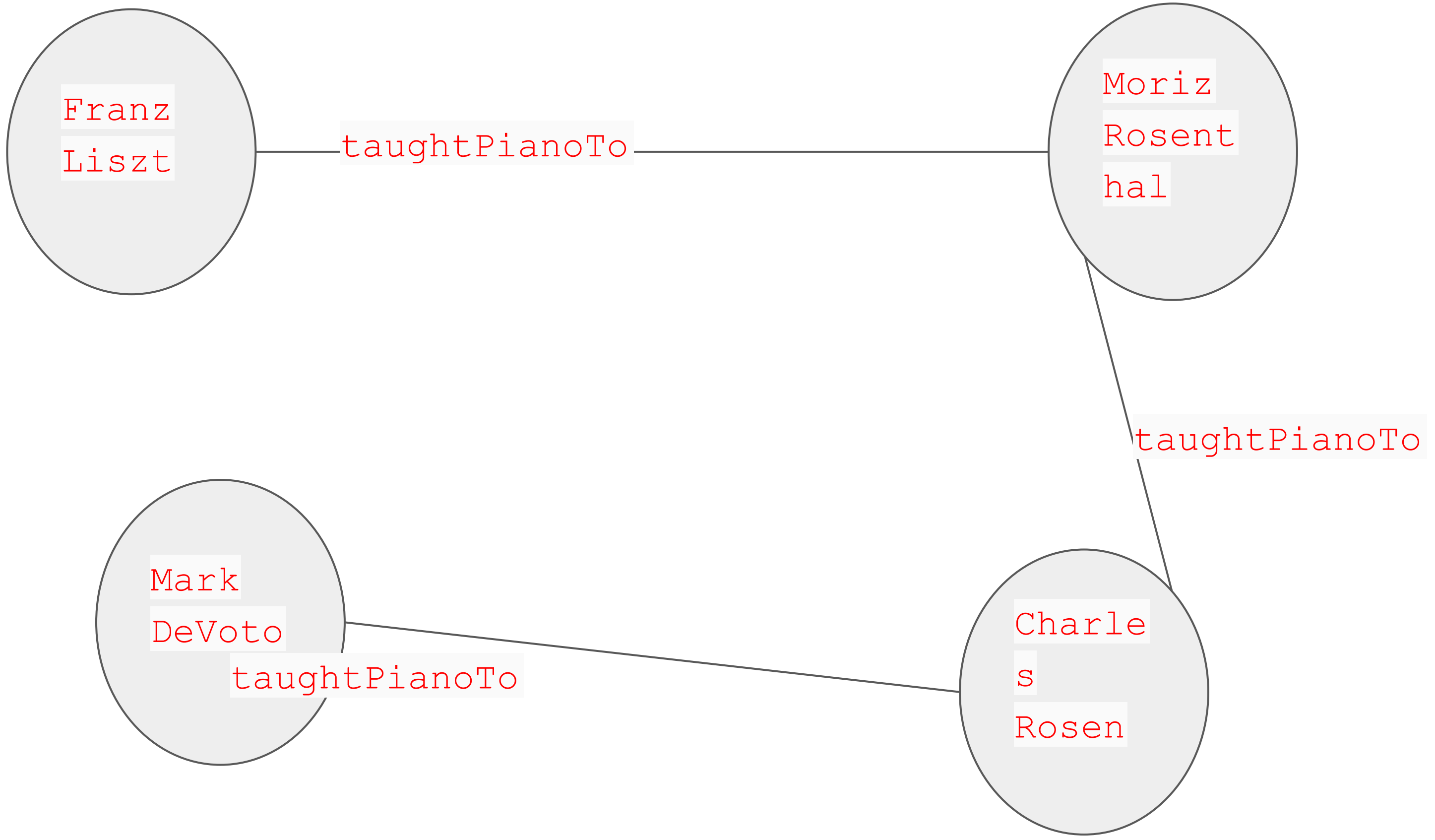
```
"Franz Liszt" taughtPianoTo "Moriz Rosenthal" .

"Moriz Rosenthal" taughtPianoTo "Charles Rosen"  .
```

or

```
"Charles Rosen" wasTaughtPianoBy "Moriz Rosenthal" ."Moriz Rosenthal"
wasTaughtPianoBy "Franz Liszt"
```

If you want to link your data to other datasets in the ‘linked data cloud’ you should look at what conventions are used in those datasets and do the same. People have spent a lot of time developing ways of modelling information within a particular area of study and thinking about how relationships within that area can be represented. These models are generally known as **ontologies.**


Franz Liszt
taughtPianoTo
Moriz Rosenthal
taughtPianoTo
Mark DeVoto
taughtPianoTo
Charles Rosen

How LOD organises knowledge: ontologies

An ontology is an abstraction that allows particular knowledge about the world to be represented.

- quite new as a concept in Information Science (VS hierarchical taxonomy)
- It aims to represent the fluidity of the real world
- flexible, non-hierarchical
- relationships not on a hierarchical tree-like structure, but in complex ways, more like a spider's web

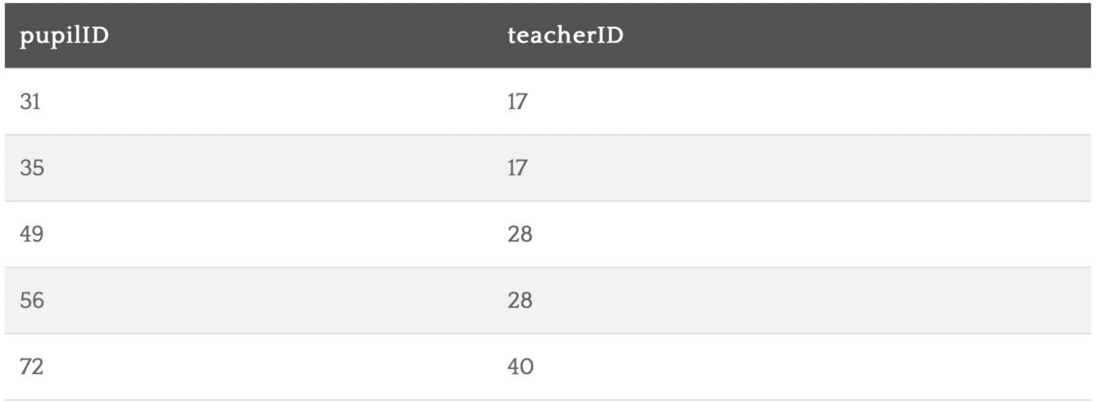
! find an existing vocabulary and use it, rather than try to write your own.

How LOD organises knowledge: ontologies

Now if you were studying the history of pianism you might want to identify many pianists who were taught by pupils of Liszt; with LOD you could (again, if the triples exist) write a query along the lines of:

```
Give me the names of all pianists taught by x
     where x was taught the piano by Liszt
```

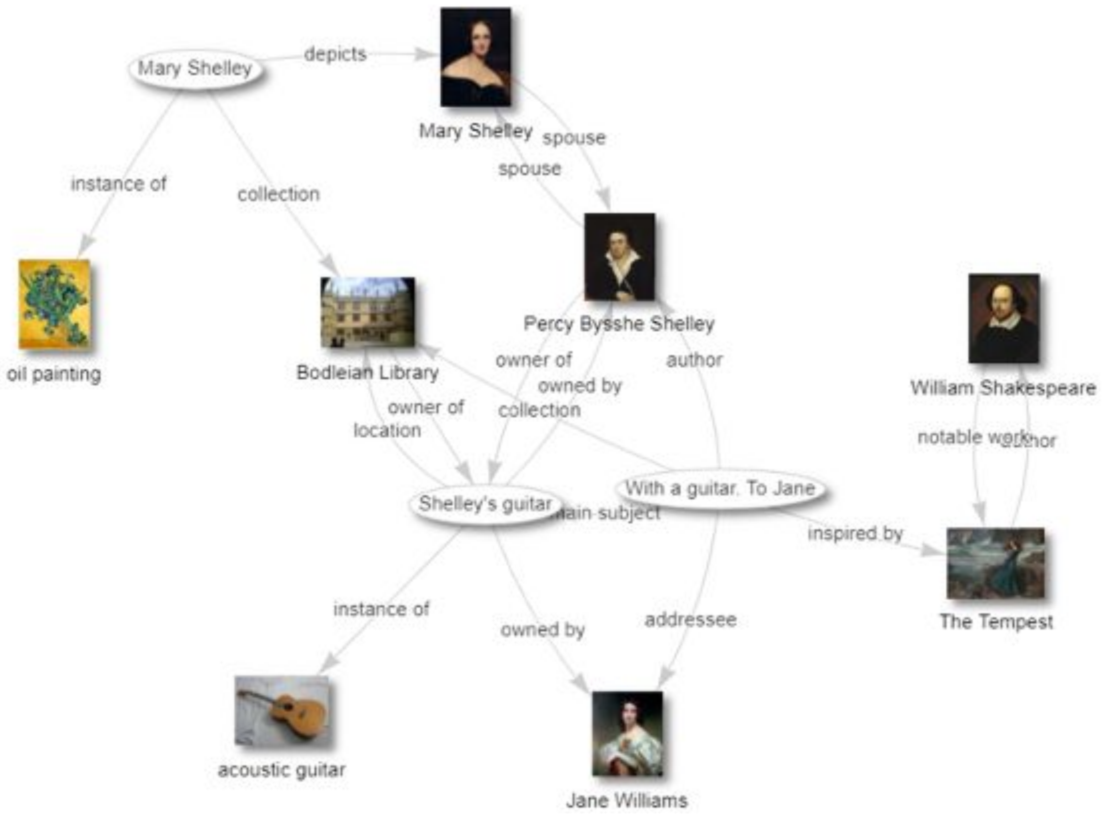
If you have used relational databases you might be thinking that they can perform the same function. In our Liszt case, the information about pianists described above might be organised in a database table called something like ‘Pupils’.



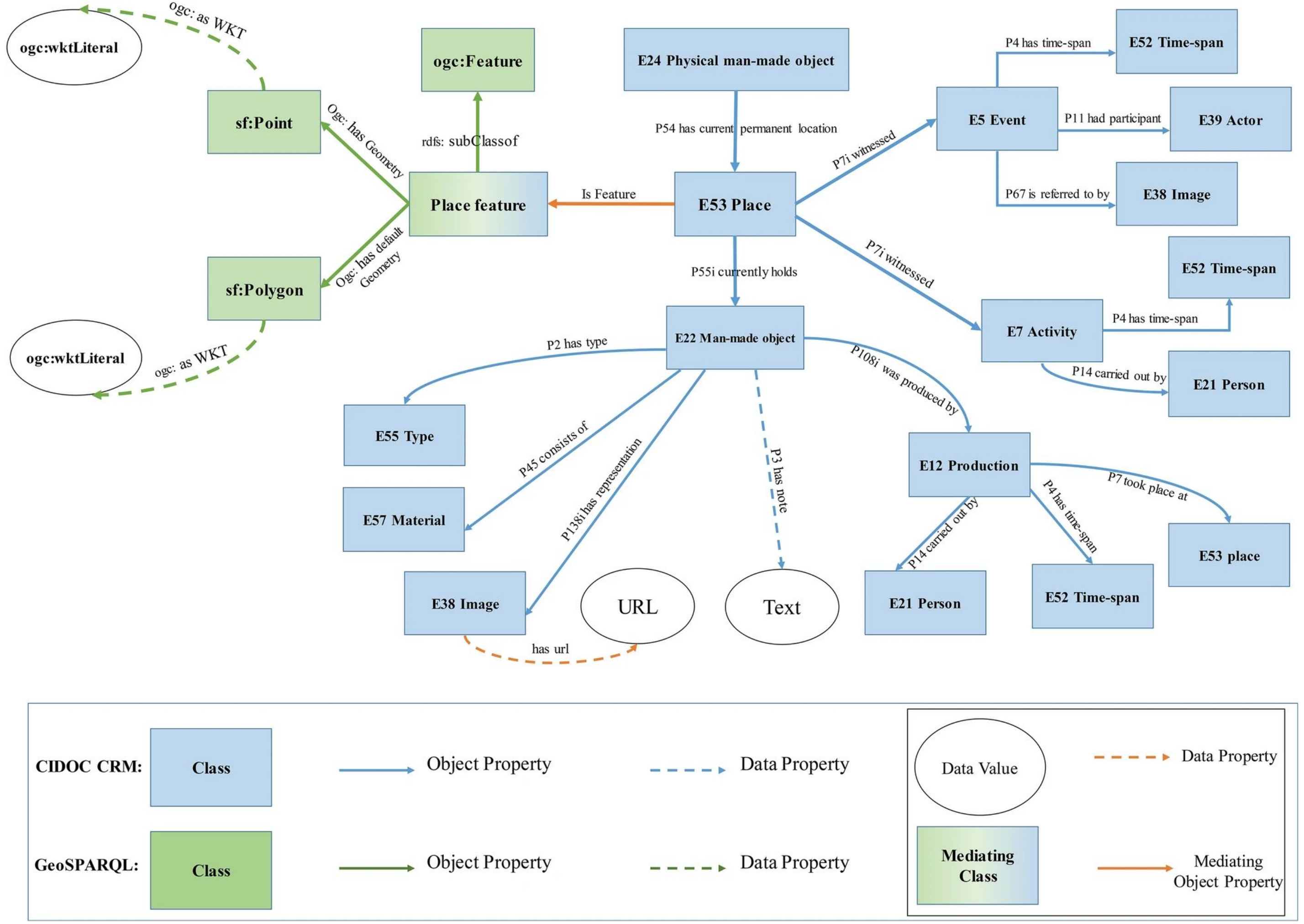
pupilID	teacherID
31	17
35	17
49	28
56	28
72	40

If Liszt is in this database table it would be fairly easy to extract the pupils of pupils of Liszt, using a **join.**

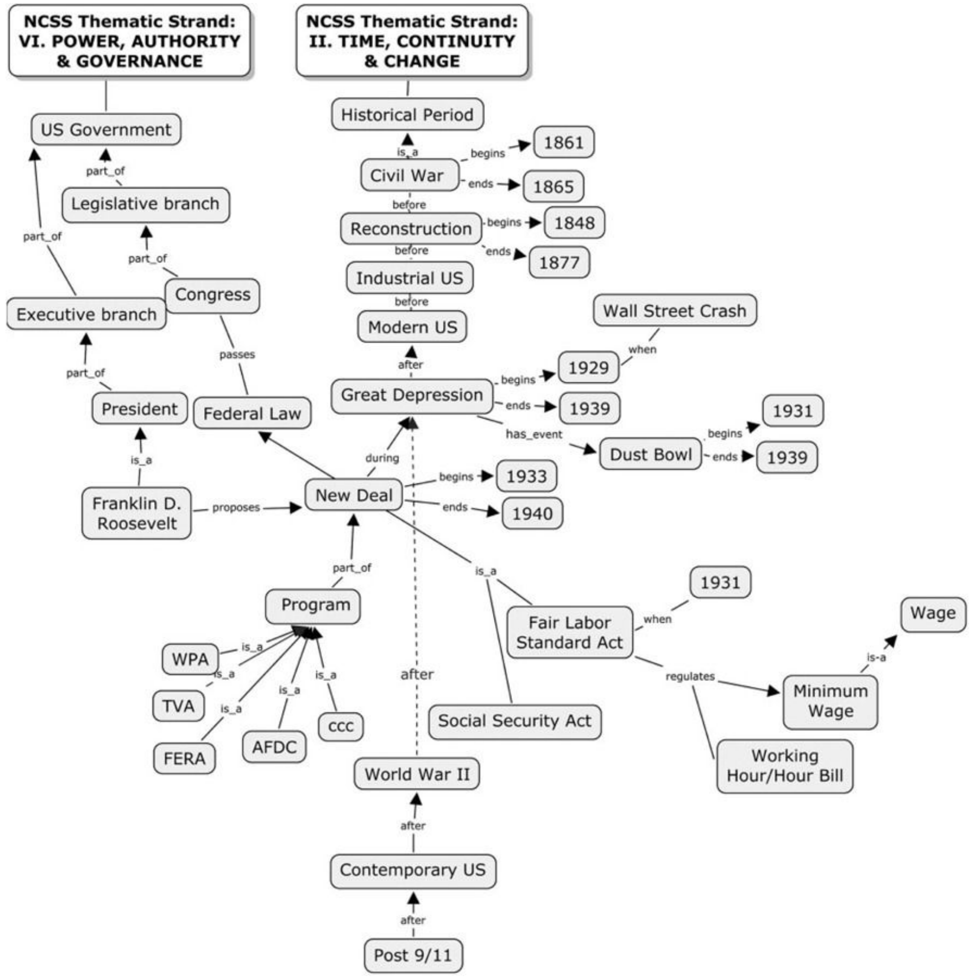
Indeed, relational databases can offer similar results to LOD. The big difference is that LOD can go further: it can link datasets that were created with no explicit intention to link them together. **The use of Resource Description Framework (RDF) and URIs allows this to happen.**



http://blogs.bodleian.ox.ac.uk/digital/2019/01/23/making-wikidata-visible/



Ranjgar, B. et al., An ontological data model for points of interest (POI) in a cultural heritage site. *Herit Sci* 10, 13 (2022). https://doi.org/10.1186/s40494-021-00635-9



Pattuelli, M.C. (2011), Modeling a domain ontology for cultural heritage resources: A user-centered approach. J. Am. Soc. Inf. Sci., 62: 314-342. https://doi.org/10.1002/asi.21453

RDF and data formats

- LOD uses a standard, defined by the World Wide Web Consortium called *Resource Description Framework* (RDF). **RDF has been widely adopted as the LOD standard.**
- The RDF is a graph-based representation format for data publishing and interchange on the Web developed by the W3C. It is also used in **semantic graph databases** (also known as **RDF triplestores**) – a technology developed for storing interconnected data and inferring new facts out of existing ones.
- RDF is a **data model** that describes how data is structured on a theoretical level. So the insistence on using triples is a rule in RDF. RDF tells you what you have to do but not exactly how you have to do it. These choices break down into two areas: how you write things down (serialisation) and the relationships your triples describe.
- **Serialisation** is the technical term for 'how you write things down'. There are different formats for creating RDF. Popular formats include RDFa, RDF/XML, Turtle and N-Triples. Although these formats are slightly different, the meaning of the RDF statements written with them remains the same.

Recognising what serialisation you are looking at means that you can then choose appropriate tools designed for that format.

RDF and data formats

Turtle

- Turtle is a pleasantly simple way of writing triples.
- Turtle uses aliases or a shortcuts known as prefixes, which saves us having to write out full URIs every time.

Let' s recall a triple's basic structure : `Subject - predicate - object`

Let's create an RDF using the Turtle syntax for describing some stuff about William Shakespeare's works. We'll use the following authority files : the Library of Congress Control Number authority file for the work in question (Macbeth), VIAF for the person, and Dublin Core as one of our prefixes for the relationship. Together these three authority files provide unique identifiers for all of the entities :

```
@prefix lccn: <http://id.loc.gov/authorities/names> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix viaf: <http://viaf.org/viaf> .

lccn:n82011242 dc:creator viaf:96994048 .
```

In the above example, lccn:n82011242 represents Macbeth; dc:creator links Macbeth to its author; viaf:96994048 represents William Shakespeare.

RDF and data formats

RDF/XML

By contrast with Turtle, RDF/XML can look a bit weighty. To begin with, let's just convert one triple from the Turtle above, the one that says that Shakespeare was the creator of *The Two Noble Kinsmen*:

```
lccn:n82011242 dc:creator viaf:96994048 .
```

In RDF/XML, with the prefixes declared inside the XML snippet, this is:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/terms/">
  <rdf:Description rdf:about="http://id.loc.gov/authorities/names/n82011242">
    <dc:creator rdf:resource="http://viaf.org/viaf/96994048"/>
  </rdf:Description>
</rdf:RDF>
```

The RDF/XML format has the same basic information as Turtle, but is written very differently, drawing on the principles of nested XML tags.

Querying RDF with SPARQL

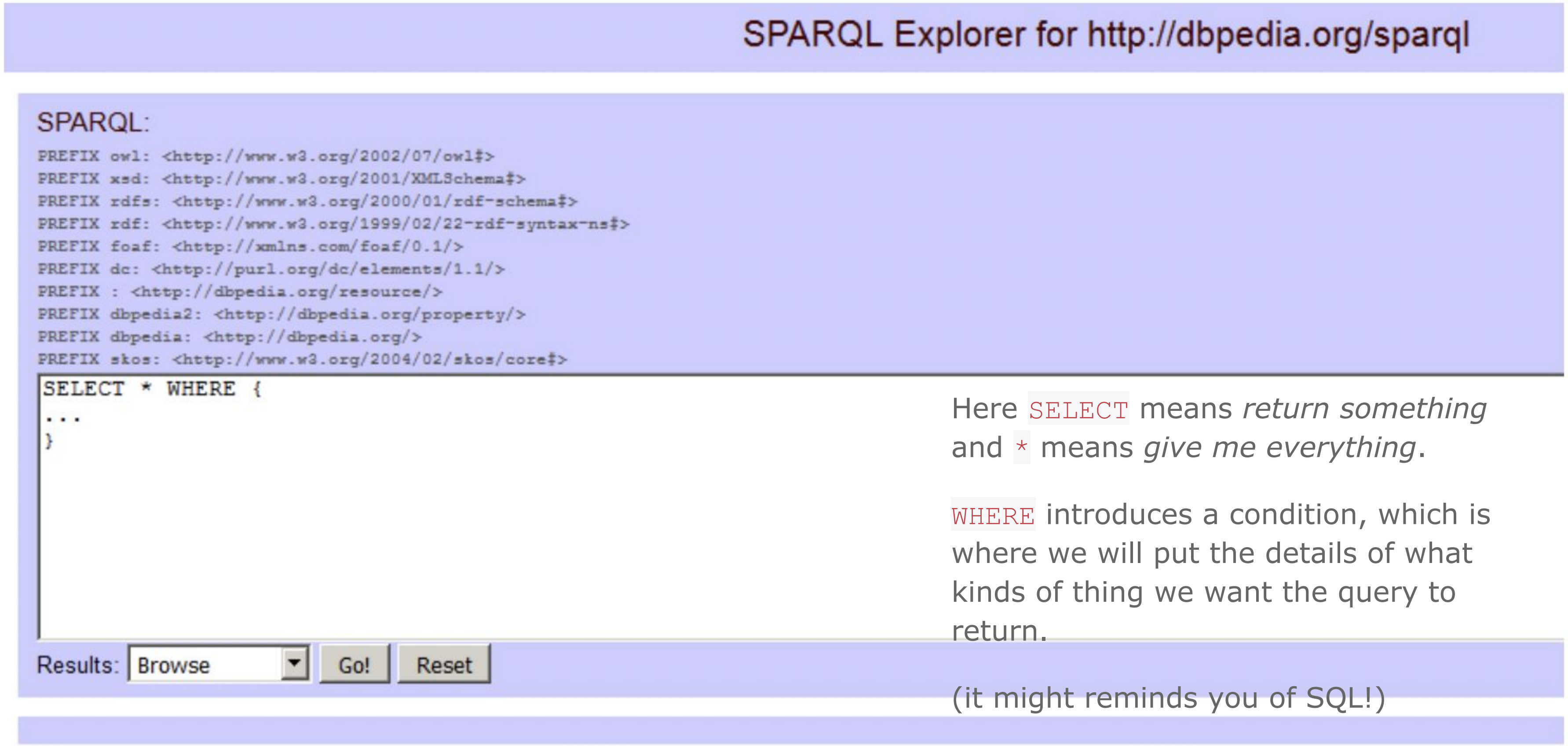
The query language we use for LOD is called SPARQL.

It’s the W3C-standardized query language for retrieving and manipulating data stored in RDF format.

We’re going to run our SPARQL queries on DBpedia, which is a huge LOD set derived from Wikipedia. As well as being full of information that is very difficult to find through the usual Wikipedia interface, it has several SPARQL “end points” - interfaces where you can type in SPARQL queries and get results from DBpedia’s triples.

We are going to use the https://dbpedia.org/snorql/ SPARQL query end point.

*Check also *Programming Historian* on using SPARQL.



SPARQL Explorer for http://dbpedia.org/sparql

SPARQL:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT * WHERE {
...
}
```

Here `SELECT` means *return something* and `*` means *give me everything*.

`WHERE` introduces a condition, which is where we will put the details of what kinds of thing we want the query to return.

(it might reminds you of SQL!)

snorql's default query box, with some prefixes declared for you

Querying RDF with SPARQL

- Let's start with something simple to see how this works. Paste (or, better, type out) this into the query box:

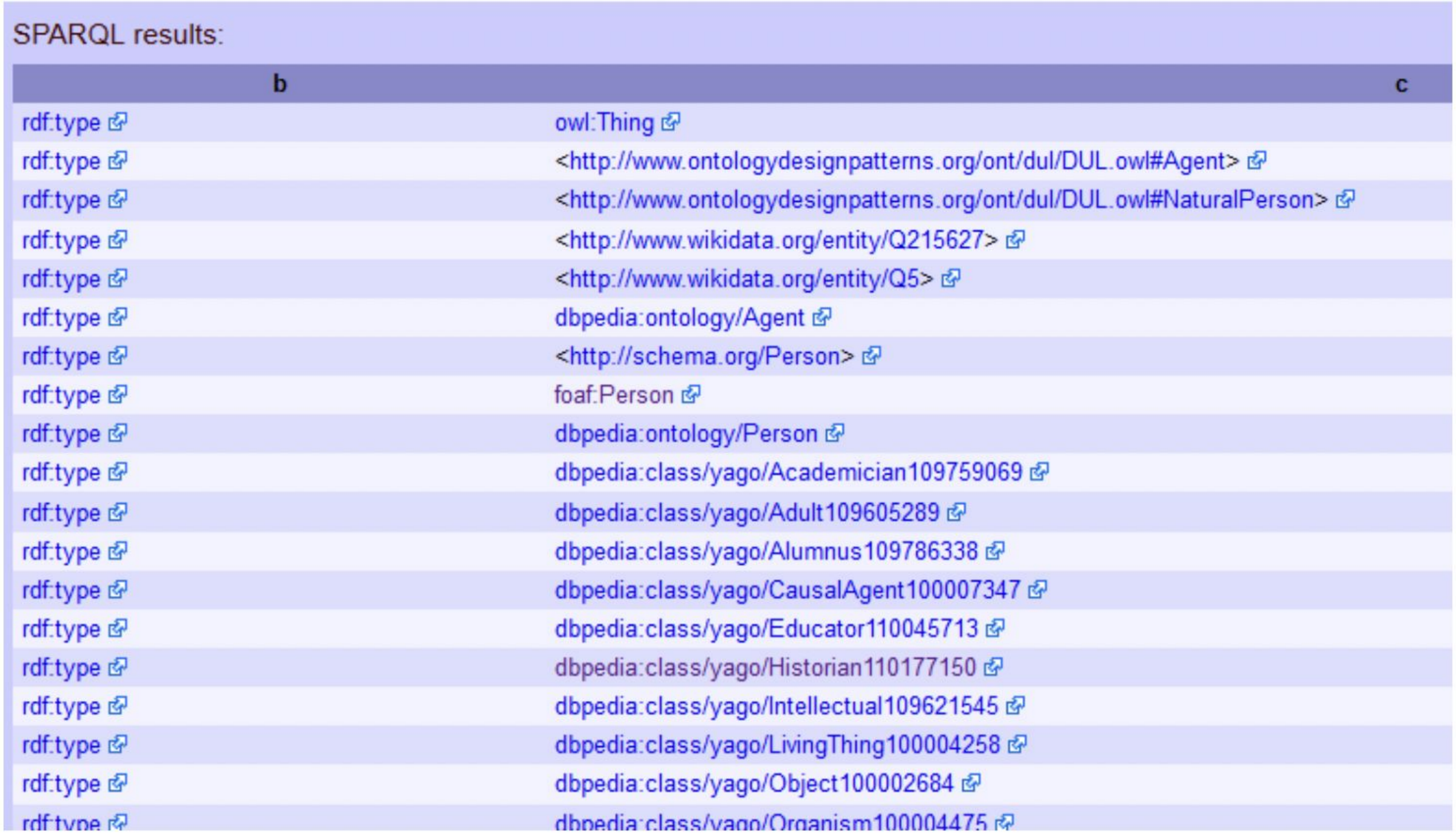
```
SELECT * WHERE {
:Lyndal_Roper ?b ?c
}
```

Hit 'go' and, if you left the drop-down box as 'browse' you should get two columns labelled "b" and "c". (Note that here, searching for a string, case does matter: lyndal_roper will get you no results.)

- When getting to know a dataset you have to try things and find out what terms are used. Because this comes from *Wikipedia*, and I was interested in what information on historians I could find, I went to the *Wikipedia* page for the historian Lyndal Roper.

The part at the end of the URL is `Lyndal_Roper` and I concluded that this string is likely to be how Roper is referred to in DBpedia. Because I don't know what else might be in triples that mention Roper I use `?a` and `?b:` these are just place-holders: I could equally well have typed `?whatever` and `?you_like` and the columns would have had those headings. When you want to be more precise about what you are returning, it will be more important to label columns meaningfully.

- Try your own SPARQL query now: choose a *Wikipedia* page and copy the end part of the URL, after the final slash, and put it in place of Lyndal_Roper. Then hit 'go'.



SPARQL results:

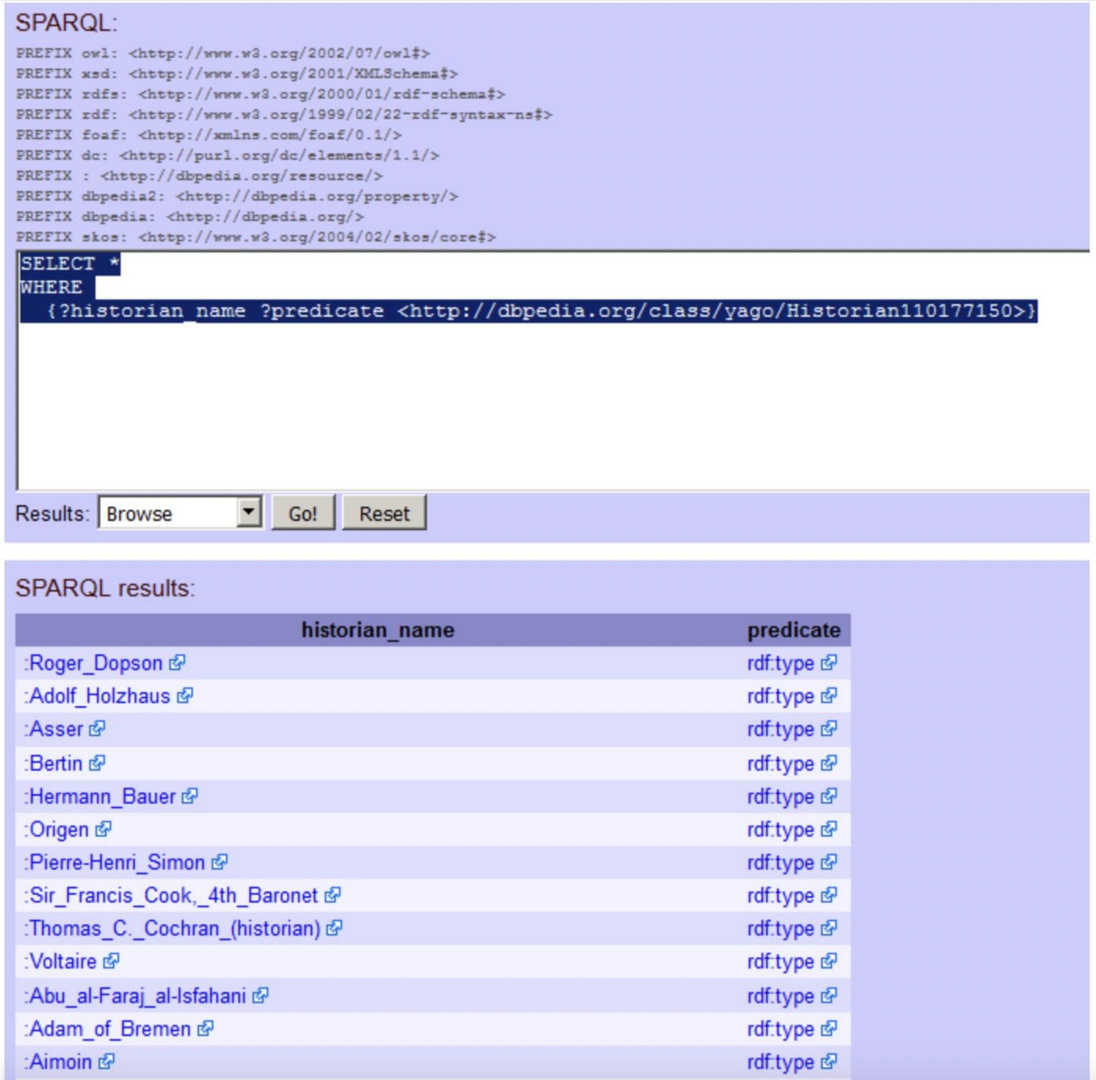
b	c
rdf:type	owl:Thing
rdf:type	<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Agent>
rdf:type	<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#NaturalPerson>
rdf:type	<http://www.wikidata.org/entity/Q215627>
rdf:type	<http://www.wikidata.org/entity/Q5>
rdf:type	dbpedia:ontology/Agent
rdf:type	<http://schema.org/Person>
rdf:type	foaf:Person
rdf:type	dbpedia:ontology/Person
rdf:type	dbpedia:class/yago/Academician109759069
rdf:type	dbpedia:class/yago/Adult109605289
rdf:type	dbpedia:class/yago/Alumnus109786338
rdf:type	dbpedia:class/yago/CausalAgent100007347
rdf:type	dbpedia:class/yago/Educator110045713
rdf:type	dbpedia:class/yago/Historian110177150
rdf:type	dbpedia:class/yago/Intellectual109621545
rdf:type	dbpedia:class/yago/LivingThing100004258
rdf:type	dbpedia:class/yago/Object100002684
rdf:type	dbpedia:class/yago/Organism100004475

top of results lists for a query for all triples with 'Lyndal_Roper' as subject in the column labelled *c* are all the attributes Roper has in *DBpedia* and will help us to find other people with these attributes. For example I can see `http://dbpedia.org/class/yago/Historian110177150`.

Let's put this into our query but in third place (because that's where it was when we found it in the Lyndal Roper results.

```
SELECT * WHERE {

?historian_name ?predicate
<http://dbpedia.org/class/yago/
Historian110177150>
```



SPARQL:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT *
WHERE
  {?historian_name ?predicate <http://dbpedia.org/class/yago/Historian110177150>}
```

Results: Browse Go! Reset

SPARQL results:

historian_name	predicate
:Roger_Dopson	rdf:type
:Adolf_Holzhaus	rdf:type
:Asser	rdf:type
:Bertin	rdf:type
:Hermann_Bauer	rdf:type
:Origen	rdf:type
:Pierre-Henri_Simon	rdf:type
:Sir_Francis_Cook,_4th_Baronet	rdf:type
:Thomas_C._Cochran_(historian)	rdf:type
:Voltaire	rdf:type
:Abu_al-Faraj_al-Isfahani	rdf:type
:Adam_of_Bremen	rdf:type
:Aimoin	rdf:type

Querying RDF with SPARQL

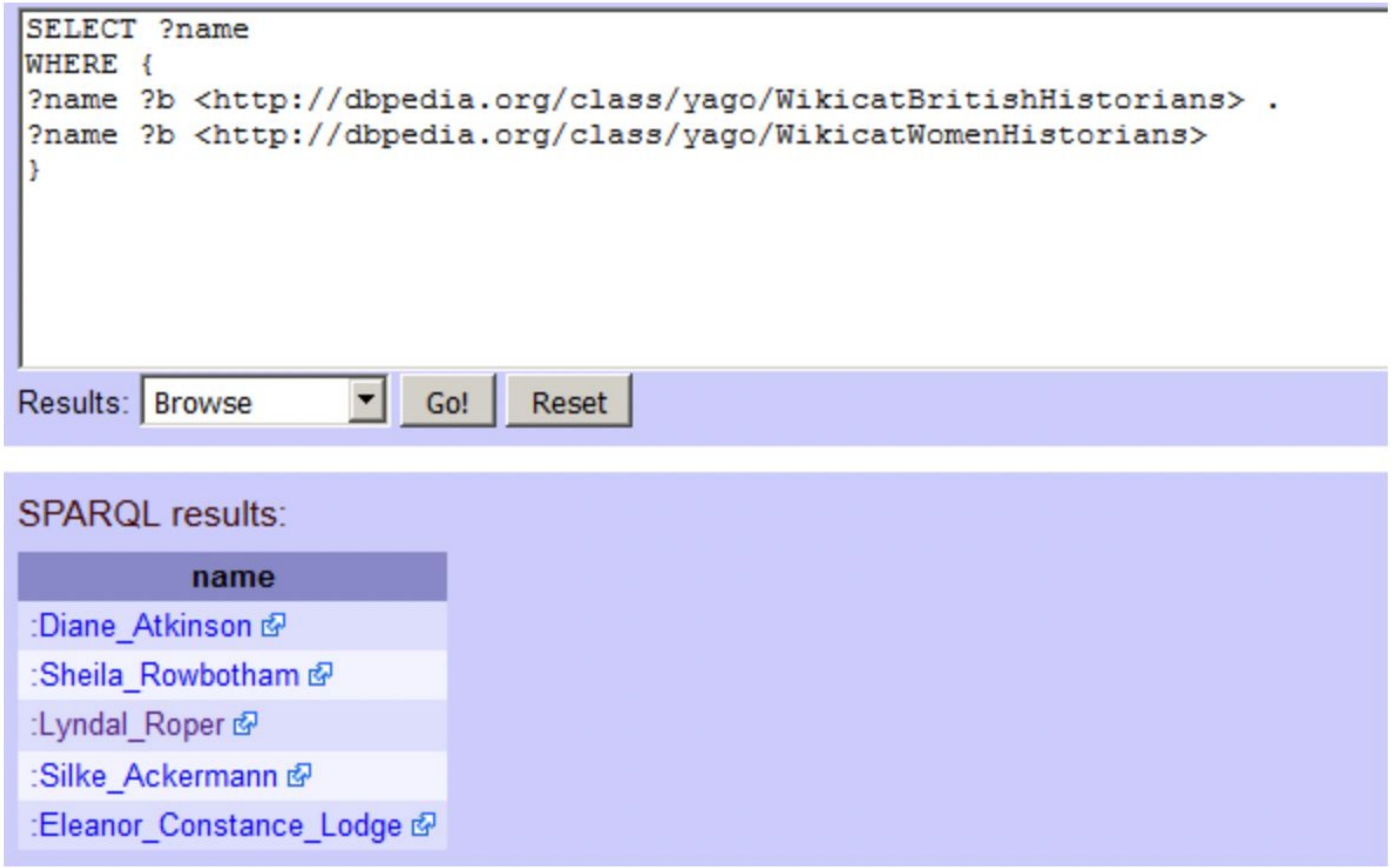
- You can combine lists, to get intersections of sets.

It might be interesting to query in Lyndal Roper's DBpedia attributes: http://dbpedia.org/class/yago/WikicatBritishHistorians and http://dbpedia.org/class/yago/WikicatWomenHistorians.

It's very easy to combine these by asking for a variable to be returned (in our case this is `?name`) and then using that in multiple lines of a query. Note as well the space and full point at the end of the first line beginning with `?name`:

```
SELECT ?name
WHERE {
?name ?b <http://dbpedia.org/class/yago/WikicatBritishHistorians> .
?name ?b <http://dbpedia.org/class/yago/WikicatWomenHistorians>
}
```

- Do try your own queries combining DBpedia attributes.
- ! With SPARQL on *DBpedia* you have to be careful of the inconsistencies of crowd-sourced material. You could use SPARQL in exactly the same way on a more curated dataset, for example the UK government data: https://data-gov.tw.rpi.edu//sparql and expect to get more robust results (there is a brief tutorial for this dataset here: https://data-gov.tw.rpi.edu/wiki/A_crash_course_in_SPARQL).
- However, despite its inconsistencies, *DBpedia* is a great place to learn SPARQL. This has only been an a brief introduction but there is much more in Using SPARQL to access Linked Open Data.



```
SELECT ?name
WHERE {
?name ?b <http://dbpedia.org/class/yago/WikicatBritishHistorians> .
?name ?b <http://dbpedia.org/class/yago/WikicatWomenHistorians>
}
```

Results: Browse

SPARQL results:

name
:Diane_Atkinson
:Sheila_Rowbotham
:Lyndal_Roper
:Silke_Ackermann
:Eleanor_Constance_Lodge

British historians who are women, according to DBpedia

If you have any question, please drop me a line

annamaria.sichani@sas.ac.uk

ps:Training material will be soon uploaded at CE GitHub repo