These slides introduce SPARQL, the ‘SELECT’ query in SPARQL, and show how you can use relatively straightforward SELECT queries on the British Library’s BNB SPARQL endpoint.
SPARQL

- Pronounced “Sparkle”
- Recursive acronym for “SPARQL Protocol and RDF Query Language”
- A W3C Standard
- A way of querying RDF in a triple store

Monday, 25 November 13

One of those recursive acronyms beloved of some of the IT community (e.g. see also GNU = Gnu’s Not Linux) Basically a standard way of querying RDF stored in a triple store. A triple store is a database designed specifically to store RDF triples - the equivalent of a relational database management system (RDBMS) that are widely used for library systems.

Just as RDBMS where there are different products (proprietary and open source) such as Oracle, MySQL, SQL Server etc. there are different triple stores. Popular ones include:

OWLIM
Jena
4Store and 5Store (currently - November 2013 - the British Library BNB runs on 5Store hosted by TSO http://openup.tso.co.uk/openup-platform#RDF%20Store)
Virtuoso

Just as with RDBMS systems, each product has it’s own strengths/weaknesses and may implement standards in different ways etc. Most triple stores will support SPARQL in some way, but different implementations may not support the full range of the latest SPARQL spec (1.1)
A reminder - Triples are made of Subject, Predicate, Object
A reminder - an English form of a triple (sort of)
Basic form of a SPARQL Query

```sparql
SELECT *
WHERE {
    ?subject ?predicate ?object
}
LIMIT 10
```

The simplest SPARQL query you can do:
Essentially means:
“Give me the first 10 triples (in the queried store)” - since no order is specified it is up to the store how it does this

The terms ‘?subject’, ‘?predicate’, and ‘?object’ are just placeholders (known as variables) - you can use whatever names you like for these as long as they start with a ‘?’. So this general query will more often be written:

```sparql
SELECT *
WHERE {
    ?s ?p ?o
}
LIMIT 10
```
Basic form of a SPARQL Query II

SELECT *
WHERE {
    ?s ?p ?o
}
LIMIT 10

https://gist.github.com/ostephens/7633423

Monday, 25 November 13
As previous slide, but with single letter variables to show how this works
A reminder – an English form of a triple
SPARQL select queries are essentially about giving patterns to the triple store and getting back triples that match this pattern.

So this query will find all triples that have the literal “Austen, Jane” as an object. However, returns no results against the BNB due to the literal not being a precise match.
Basic SPARQL Query using a literal II

SELECT *
WHERE {
    ?subject ?predicate "Austen, Jane, 1775-1817"
}
LIMIT 10

https://gist.github.com/ostephens/7639249

SPARQL select queries are essentially about giving patterns to the triple store and getting back triples that match this pattern.

So this query will find all triples that have the literal “Austen, Jane, 1775-1817” as an object. This finds results, but note this does not find books authored by Jane Austen - due to the data structure used by BNB - which assigns a URI, and then uses the authorised form of the name as an RDFS label on a URI representing
Monday, 25 November 13

Generally statements in the BNB about Jane Austen will use the URI for Jane Austen, rather than a string. The BNB for Jane Austen the person who wrote Pride and Prejudice is <http://bnb.data.bl.uk/id/person/AustenJane1775-1817> so we would need to use this in our query.
Basic SPARQL Query using a URI

```
SELECT *
WHERE {
  ?subject ?predicate <http://bnb.data.bl.uk/id/person/AustenJane1775-1817>
}
LIMIT 10
```

https://gist.github.com/ostephens/7633236

Monday, 25 November 13

SPARQL select queries are essentially about giving patterns to the triple store and getting back triples that match this pattern.

So this query will find all triples that have the URI “http://bnb.data.bl.uk/id/person/AustenJane1775-1817” as an object. This URI represents Jane Austen, so this query will find any statements where Jane Austen is the object. A typical statement of this type might be (in sort-of English rather than RDF):

Pride and Prejudice - has the author - Jane Austen

So - this would find all the books authored by Jane Austen. However, since we haven’t specified the type of relationship to Jane Austen it would also find any other triples where the URI for Jane Austen was the object.
If we want to be more specific and retrieve books (or strictly, URIs which identify books) where Jane Austen is the dc:creator we need to specify dc:creator as the predicate in the query pattern.

http://bnb.data.bl.uk/id/person/AustenJane1775-1817

Austen, Jane, 1775-1817
Basic SPARQL Query using a URI II

SELECT *

WHERE {
}

LIMIT 10

https://gist.github.com/ostephens/7633258

Monday, 25 November 13

This is a stricter pattern - insists not only that <http://bnb.data.bl.uk/id/person/AustenJane1775-1817> is the object, but that the predicate is http://purl.org/dc/terms/creator

This finds all books where Jane Austen is the ‘dc:creator’ which I think in terms of the BNB data essentially means Austen, Jane was the entry in the MARC 100 field

Does not include books where Jane Austen is ‘dc:contributor’ (in MARC 700 field)

Perhaps more accurately: This finds the URIs for books where Jane Austen is the dc:creator - don’t confuse the URIs for the books themselves - what you get in response to this queries is a list of URIs, not a list of titles.

Also note uses the slightly more readable ‘?book’ as variable instead of the generic ‘?subject’
PREFIX dct: <http://purl.org/dc/terms/>

SELECT *

WHERE {
    ?subject dct:creator <http://bnb.data.bl.uk/id/person/AustenJane1775-1817>
}

LIMIT 10

https://gist.github.com/ostephens/7633289

Monday, 25 November 13

The PREFIX command is just a timesaver - essentially allows you to substitute part of a URI with an abbreviation. The PREFIX is defined only in the query, although there are some common prefixes that are used - as long as you are consistent in your use you can use what you want. PREFIX banana: <http://purl.org/dc/terms/> works just as effectively as PREFIX dct: <http://purl.org/dc/terms/> but the latter is much more common! (see https://gist.github.com/ostephens/7633297 for use of ‘banana’ as PREFIX)

Common PREFIX statements are given at https://gist.github.com/ostephens/7633203
Basic SPARQL Query

PREFIX dct: <http://purl.org/dc/terms/>

SELECT *

WHERE {
  <http://bnb.data.bl.uk/id/resource/009724890> ?predicate ?object
}

LIMIT 10

https://gist.github.com/ostephens/7633258

Pattern for finding statements about a particular book (in this case an edition of Pride and Prejudice)
Query with 2 patterns

```
PREFIX dct:   <http://purl.org/dc/terms/>

SELECT ?title
WHERE {
}
LIMIT 10
```

This query joins together the two patterns we’ve see in the previous slides.
First finds URIs where Jane Austen is dct:creator
Then for each of the URIs finds what the dct:title is

That is - the second pattern applies to the URIs found in the first pattern.

This will get us a title list of books authored by Jane Austen

Note that the two patterns are joined with a ‘period’ - ‘.’
Note also that here the ‘Select’ statement only asks for ?title to ‘*’ - what do you think you’d get if you used ‘*’ here
Try amending this to find titles of books where Jane Austen is a ‘contributor’ - a much more interesting title list!
A 3 pattern query
Note use of ‘Distinct’ to avoid duplicates in results
This finds titles of books authored by the person identified by the VIAF URI http://viaf.org/viaf/102333412 - which happens to be ... Jane Austen. Starts to show how sharing identifiers can be very powerful
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct:   <http://purl.org/dc/terms/>
PREFIX owl:   <http://www.w3.org/2002/07/owl#>
PREFIX bio:   <http://purl.org/vocab/bio/0.1/> 
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?name
WHERE {
  ?person bio:event ?event .
  ?e2 bio:date ?dob .
  ?e2 rdf:type bio:Birth .
  ?p2 foaf:name ?name
}
LIMIT 1000

https://gist.github.com/ostephens/7633750

Monday, 25 November 13
Complex select - finds the names of all the people in BNB with the same birth year as person identified by http://viaf.org/viaf/102333412 (Jane Austen)
Example application using SPARQL

Built using SPARQL statement of previous page
Possible because Wikipedia uses VIAF URIs to identify people...
See http://www.meanboyfriend.com/overdue_ideas/2013/04/contemporaneous-part-one/
Write your own SPARQL

• BNB has a SPARQL endpoint and online SPARQL editor + sample queries

• http://bnb.data.bl.uk/flint

Many SPARQL endpoints provide a web based SPARQL editor - here are a few:
BNB = http://bnb.data.bl.uk/flint
British Museum = http://collection.britishmuseum.org/sparql
Cambridge University Library (beta) = http://data.lib.cam.ac.uk/endpoint.php
Europeana = http://europeana.ontotext.com/sparql
...
Important things I haven’t mentioned

• Using ‘filters’ in queries
• Data types
• Language tags
• much more...

See next slide for some further reading!
Further reading

• Learning SPARQL by Bob DuCharme (http://www.learningsparql.com/bio.html)


• http://www.meanboyfriend.com/overdue_ideas/2013/04/contemporaneous-part-one/

• Tutorial from Cambridge University Library Linked Data project: http://data.lib.cam.ac.uk/sparql.php