What's SPARQL and is there any reason why it might be useful for me?

1. What's an URI/IRI/HTTP IRI?

IRIs denote resources. The resource that is denoted is the "referent". Referents may be information resources (e.g. web pages) or non-information resources (e.g. a person, a play, a concept)

```
http://dbpedia.org/resource/Hamlet

http://viaf.org/viaf/96994048

http://www.w3.org/2000/01/rdf-schema#label

http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare
http://dbpedia.org/resource/Category %3APlays_by_William_Shakespeare
```

2. What's a namespace?

```
viaf: ="http://viaf.org/viaf/"
dbr: = "http://dbpedia.org/resource/"
rdfs: = "http://www.w3.org/2000/01/rdf-schema#"

viaf:96994048
dbr:Hamlet
rdfs:label
```

2. What's RDF/a triple?

RDF=resource description framework (an abstract model for describing relationships among resources)
Form of a triple:
[subject] [predicate] [object].

```
<http://viaf.org/viaf/96994048> <http://www.w3.org/2000/01/rdf-schema#label> "William
Shakespeare"@en.

viaf:96994048 rdfs:label "William Shakespeare"@en.
```

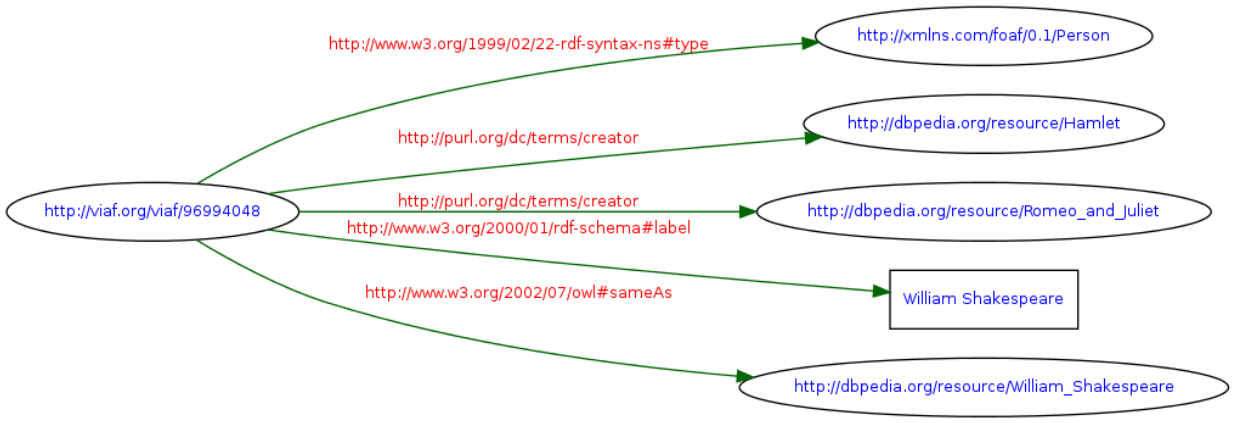
3. What's a graph?

A graph is a set of triples.

RDF/turtle (*Turtle* = Terse RDF Triple Language; a W3C standard) serialization of the graph:

```
viaf:96994048 rdfs:label "William Shakespeare"@en;
              owl:sameAs  dbres:William_Shakespeare;
              dcterms:creator dbres:Hamlet,
                              dbres:Romeo_and_Juliet;
              a foaf:Person.
```

Graphical representation of the graph:



4. What's Linked Data?

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL)
4. Include links to other URIs so that they can discover more things.

Tim Berners-Lee, 2006 http://www.w3.org/DesignIssues/LinkedData.html

LOD=Linked Open Data (linked data with an open license)
Linked Data client = a computer program for traversing the Web of Data and retrieving data.

5. Explore using these Linked Data clients:
EasyRdf Graph dumper (thanks Jacob Shelby, Parks Library, Iowa State University!):
http://jacobshelby.org/easyrdf/examples/dump.php
Explore: `http://viaf.org/viaf/96994048`

LodLive visualizer. Bubbles=IRI-identified resources, "text" bubble shows images/text/links
http://en.lodlive.it/
Explore:
`http://viaf.org/viaf/96994048`
`http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare`

5. What is a triple store?
It's a big collection of graphs, and therefore a big set of triples.

6. What SPARQL/a SPARQL endpoint?
SPARQL is a query language for RDF and W3C standard. (SPARQL=SPARQL Protocol and RDF Query Language; a recursive acronym)

A SPARQL endpoint is a web service that receives SPARQL queries, runs them against the graphs in a triplestore, and returns the result to a user. This interaction is the "protocol" part of SPARQL.

Example from DBpedia (extracts structured information from Wikipedia):
http://dbpedia.org/snorql/

Queries can be run from the paste-in box. Starter query to find the value of the human-readable label (`rdfs:label`) for the resource:

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbres: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?name
WHERE {
      <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare> rdfs:label ?name.
      }
Limit 20
```

Whitespace is not important in the query, it just helps readability. The prefix declarations here are in addition to the defaults at the form. SELECT shows the variables you want to see. WHERE contains the graph pattern that screens the triples. Find predicate/object (i.e. property/value) pairs for the IRI when it's the subject of the triple:

```
SELECT ?predicate ?object
WHERE {
      <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare> ?predicate ?object.
      }
```

Try reversing the pattern:

```
SELECT ?subject ?predicate
WHERE {
      ?subject ?predicate <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
      }
```

Screen by more than one triple pattern to find the labels of categories that are narrower:

```
SELECT ?name
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?narrowCategory rdfs:label ?name.
}
```

Find plays whose Dublin Core subject is the narrower category and list their names:

```
SELECT ?name
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
}
```

Filter the play names to only English:

```
SELECT ?name
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="en")
}
```

Also find and display the abstracts:

```
SELECT ?name ?abstract
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="en")
?play dbo:abstract ?abstract.
}
```

Filter the abstracts, too:

```
SELECT ?name ?abstract
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="en")
?play dbo:abstract ?abstract.
FILTER (lang(?abstract)="en")
}
```

Find external links to Project Gutenberg etexts:

```
SELECT ?name ?abstract ?link
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="en")
?play dbo:abstract ?abstract.
FILTER (lang(?abstract)="en")
?play dbo:wikiPageExternalLink ?link.
FILTER (CONTAINS(str(?link),"http://www.gutenberg.org/etext"))
}
```

Note: results are returned only if all three variables are present. A more complex query with the OPTIONAL keyword is required for cases where only some items are present.

Translate to Japanese:

```
SELECT ?name ?abstract ?link
WHERE {
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="ja")
?play dbo:abstract ?abstract.
FILTER (lang(?abstract)="ja")
?play dbo:wikiPageExternalLink ?link.
FILTER (CONTAINS(str(?link),"http://www.gutenberg.org/etext"))
}
```

7. Using the Heard Library SPARQL endpoint

Go to the Heard Library SPARQL endpoint:
http://rdf.library.vanderbilt.edu/
Click on "click for search options".

Recall:

```
viaf:96994048 rdfs:label "William Shakespeare"@en;
              owl:sameAs  dbres:William_Shakespeare;
              dcterms:creator dbres:Hamlet,
                              dbres:Romeo_and_Juliet;
              a foaf:Person.
```

Stupidly, DBpedia never seems to have indicated who wrote the plays! We can fix that because these five triples have been added to our triplestore and so we can query them. Notice that there are no default namespaces shown, so you'll have to list them all. Note: DISTINCT keyword removes redundant solutions. Find the name of the person who wrote Romeo and Juliet:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbres: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?authorName
WHERE
{
?author dcterms:creator dbres:Romeo_and_Juliet.
?author rdfs:label ?authorName.
}
```

8. Federated query using a remote service (a real linked data application!)
Use the SERVICE keyword to specify that part of the graph pattern is at a remote endpoint:

```
SELECT DISTINCT ?name ?authorName ?abstract ?link
WHERE
{

{
?author dcterms:creator ?play.
?author rdfs:label ?authorName.
}

SERVICE <http://DBpedia.org/sparql>
{
?narrowCategory skos:broader <http://dbpedia.org/resource/Category:Plays_by_William_Shakespeare>.
?play dcterms:subject ?narrowCategory.
?play rdfs:label ?name.
FILTER (lang(?name)="en")
?play dbo:abstract ?abstract.
FILTER (lang(?abstract)="en")
?play dbo:wikiPageExternalLink ?link.
FILTER (CONTAINS(str(?link),"http://www.gutenberg.org/etext"))
}

}
Limit 5
```

Notice that the information for `?name`, `?abstract`, and `?link` come from DBpedia, but `?author` comes from the Heard Library endpoint.

9. Interactive use of SPARQL in a web page (Bioimages demo page)

Go to the Heard Library SPARQL endpoint:
http://rdf.library.vanderbilt.edu/
Click on "Explore the Bioimages data". View the page source. Notice that in the <head> element, it calls the Javascript "bioimages.js". To view it, go to:

http://www.library.vanderbilt.edu/webimages/Bioimages/bioimages.js

The script contains a number of functions that get called by various events that fire when the user interacts with the controls on the web page. Here's the general pattern:

A. Define the SPARQL query as a string, with the thing you want to vary inserted as a variable. For example:

```
var string = 'SELECT DISTINCT ?species WHERE {'
+'?identification <http://rs.tdwg.org/dwc/terms/genus> '+passedGenus+'.'
+'?identification <http://rs.tdwg.org/dwc/terms/specificEpithet> ?species.'
+'}'
+'ORDER BY ASC(?species)';
```

Remember that whitespace isn't important, so the string is just a long line of text. In this case the variable "passedGenus" gets stuck in the string in the right place.

B. URL encode the string so that nasty characters don't cause a problem:

```
var encodedQuery = encodeURIComponent(string);
```

C. Use AJAX via jQuery to send the encoded query to the Heard Library SPARQL endpoint by an HTTP GET command:

```
$.ajax({
    type: 'GET',
    url: 'http://rdf.library.vanderbilt.edu/sparql?query=' + encodedQuery,
    headers: {
             Accept: 'application/sparql-results+xml'
            },
    success: parseSpeciesXml
    });
```

D. The results get sent back to the browser from the endpoint in an XML formatted file as specified by the SPARQL W3C standard.

E. Various functions parse the results file and pulls the strings we want from the XML. The functions then stick the strings wherever we want them to be in the HTML. In the parseSpeciesXML function, the species that are part of the selected genus get added to the species dropdown list.

In the end, the user clicks on "Search" and a big, nasty SPARQL query finds the images that meet the screening requirements and loads the thumbnails and link URLs into an HTML table that is displayed to the user.