SPARQL QUERY LANGUAGE
SPARQL query language

- W3C standard for querying RDF graphs
- Can be used to query not only native RDF data, but also any data that could be mapped to RDF
- This mapping could be done by making use of
  - (W3C) standard mapping languages such as **R2RML** that allow for transforming relational data to RDF
  - Various mapping tools such as those listed at: [http://www.w3.org/wiki/ConverterToRdf](http://www.w3.org/wiki/ConverterToRdf)
Let’s start with an example

Graphical representation of a small segment of the RDF graph given in:
http://www.ldodds.com/ldodds-knows.rdf

Metadata about Leigh Dodds’s relationships

Ethan Dodds

1bca73e5c6916c7c...

foaf:depiction

Edd Dumbill

T699kfs1djkskdf...

Graphical representation of a small segment of the RDF graph given in:
http://www.ldodds.com/ldodds-knows.rdf
Task 1: Find names of all mentioned persons

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  ?x rdf:type foaf:Person.
  ?x foaf:name ?name.
}
The basic structure of a SPARQL query

- **PREFIX**
  - the SPARQL equivalent of declaring an XML namespace

- **SELECT**
  - like its twin in an SQL query, it is used to define the data items that will be returned by the query

- **FROM**
  - identifies the data against which the query will be run
    - can be given in runtime as well

- **WHERE**
  - defines the part of RDF graph we are interested in
Some notes about the SPARQL syntax

- Variables are prefixed with either "?" or "$"
  - these two are interchangeable

- Blank nodes are indicated by:
  - the label form, such as "_:abc", or
  - the abbreviated form "[]"

- Dots (.) separate triple patterns

- Semi column (;) separates triple patterns with the common subject
Task 2: Find names and emails of the persons whom the author of the document knows

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?name ?email
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  ?doc rdf:type foaf:Document ;
    foaf:maker ?author .
}

Graph pattern
About graph patterns

- Graph pattern is a collection of triple patterns
- It identifies the shape of the (RDF) graph we want to match against
- Within one graph pattern each variable must have the same value no matter where and how (in the graph pattern) it is used
About graph patterns

- In SPARQL, one cannot SELECT a variable if it is not listed in the graph pattern (i.e., in the WHERE clause).

**Important:**
SPARQL query processor has NO data dictionary or schema that lists types and properties of a resource.
The only schema it has is the graph pattern (i.e., the WHERE part of the query).
The result set for the query from the previous example:

```
<sparql
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:xs="http://www.w3.org/2001/XMLSchema#"
 xmlns="http://www.w3.org/2005/sparql-results#" >

<head>
 <variable name="name"/>
 <variable name="email"/>
</head>

<results ordered="false" distinct="false">
 <result>
   <binding name="name">
     <literal>Edd Dunbill</literal>
   </binding>
   <binding name="email">
     <uri>mailto:edd@xml.com</uri>
   </binding>
 </result>
</results>
</sparql>
```
Optional Matching

• RDF often represents *semi-structured* data
  • this means that two resources of the same type may have different sets of properties
  • For instance,
    • a FOAF description of a person may consist only of an e-mail address;
    • alternatively, it can incorporate a real name, twitter nickname, URL of the photo depicting him/her, etc.

• SPARQL’s mechanism for *optional matching* allows for handling this heterogeneity
Task 3: Find all persons that the author of the document knows as well as their blogs if they have any

```sql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?person ?blog
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  OPTIONAL { ?person foaf:weblog ?blog. }
}
```
The OPTIONAL block

- If a query has multiple optional blocks
  - these act independently of one another
  - each block may be omitted from, or present in, a solution.

- Optional blocks can also be nested
  - the inner optional block is considered only when the outer optional block's pattern matches the graph.
Task 4: Find all persons that the author of the document knows as well as their blogs and emails, *if these are available*

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?person ?email ?blog
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  OPTIONAL { ?person foaf:mbox_sha1sum ?email. }
  OPTIONAL { ?person foaf:weblog ?blog . }
}
```
Alternative Matching

• Let’s suppose that …
  • foaf:knows and rel:hasMet properties are used to represent somewhat similar information
  • we are interested in all persons that the author of the document either knows or has (ever) met

• In situations like this, you can use SPARQL's alternative matching feature to return whichever of the properties is available
Task 5: Find names of all persons that the author of the document either has met or knows

PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX rel: <http://purl.org/vocab/relationship/>  
SELECT ?name  
FROM <http://www.ldodds.com/ldodds-knows.rdf>  
WHERE {  
  { ?author foaf:knows [ foaf:name ?name] . }  
  UNION  
  {?author rel:hasMet [ foaf:name ?name] . }  
}
UNION

• In contrast with OPTIONAL graph patterns, in the case of UNION *at least one* of the alternatives must be matched by any query solution;

• If both branches of the UNION match, two solutions will be generated.
DISTINCT

• In the result set of the previous task some names appeared twice

• By adding the DISTINCT keyword in the SELECT clause, we exclude multiple appearance of the same values from the result set
  • Just like in SQL
Task 5a: Find names of all the persons that the author of the document either has met or knows (without name repetition)

PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX rel: <http://purl.org/vocab/relationship/>  
SELECT DISTINCT ?name  
FROM <http://www.ldodds.com/ldodds-knows.rdf>  
WHERE  
{}  
  {?author foaf:knows [ foaf:name ?name] . }  
  UNION  
  {?author rel:hasMet [ foaf:name ?name] . }  
}
The ORDER BY clause

• Indicates that the result set should be ordered by the specified variable

• It can list one or more variable names, indicating the variables that should be used to order the result set

• By default all sorting is done in the ascending order
  • this can be explicitly changed using the DESC (descending) and ASC (ascending) functions
Task 5b: Find names of all persons that the author of the document has either met or knows; sort the names in descending order

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rel: <http://purl.org/vocab/relationship/>
SELECT DISTINCT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
    { ?author foaf:knows [ foaf:name ?name] . }
    UNION
    {?author rel:hasMet [ foaf:name ?name] . }
}
ORDER BY DESC (?name)
SPARQL FILTERs

- SPARQL FILTERs restrict the solutions of a graph pattern match according to the given expressions.
- Expressions can be of different kinds, but they must evaluate in a boolean value (true or false).
- The following slides illustrate some of the functions that can be used for filtering the result set.
Task 6: Get names of all the persons whose birthday is unknown

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bio: <http://purl.org/vocab/bio/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  ?person foaf:name ?name .
  FILTER NOT EXISTS {
    ?ev rdf:type bio:Birth ; bio:date ?birthdate. }
}

Note: Function NOT EXISTS is introduced in SPARQL 1.1; if the query does not work, it means you are using a old SPARQL engine
Task 7: Find names of all members of the Dodds family

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
    ?person foaf:name ?name
    FILTER regex(?name, "dodds", "i")
}

Filtering with regular expressions Similar to SQL "LIKE"

Alternative:
FILTER strEnds(lcase(?name), "dodds")
Task 7a: Find names of all the persons who have Gmail email address

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
   FILTER regex( str(?mbox), "@gmail\.com$")
}

To learn more about the regular expression language visit:
http://www.w3.org/TR/xpath-functions/#regex-syntax
Task 8: Get all reviews with rating above 6 that were created by a person named Jim (filtering based on elements values)

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rev: <http://www.purl.org/stuff/rev#>

SELECT ?review
FROM <http://www.cs.umd.edu/~hendler/2003/foaf.rdf>
WHERE {
  ?someone rdf:type foaf:Person;
  foaf:name ?name FILTER regex(?name, "Jim", "i").
  ?review rev:rating ?rating
  FILTER (xsd:decimal(?rating) >= "6"^^xsd:decimal).
}
Grouping and aggregating data

- GROUP BY allows for grouping the items in the result set based on one or more variables and/or expressions.

- HAVING allows for selecting/filtering the query results at the group level:
  - It is analogous to a FILTER expression, but operates over groups, rather than individual solutions.

- There are various functions applicable at the group level: SUM, COUNT, AVG, MIN, MAX and the like.
Task 9: Find manufacturers who produce more than 10 different products and display the number of different products they produce

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia-owl: <http://dbpedia.org/ontology/>

SELECT ?manufacturer (COUNT(?product) AS ?count)
WHERE {
    dbpedia-owl:manufacturer ?manufacturer .
}
GROUP BY ?manufacturer
HAVING (COUNT(?product) > 10)
Other kinds of SPARQL queries

Besides the SELECT queries, SPARQL supports three other query types:

• ASK
• DESCRIBE
• CONSTRUCT
ASK query

• Aimed at testing whether a query pattern has a solution

• No information is returned about the possible query solutions, just whether a solution exists

• An example: have Natalie Portman and Scarlett Johansson ever played in the same movie?

```PREFIX db: <http://dbpedia.org/ontology/>
ASK { ?movie db:starring <http://dbpedia.org/resource/Natalie_Portman> ;
}
```
Results of an ASK query:
  • Possible values: true/false
  • XML format of the results:

```xml
<sparql xmlns="http://www.w3.org/2001/sw/DataAccess/rf1/result2">
  <head/>
  <results>
    <boolean>true</boolean>
  </results>
</sparql>
```
DESCRIBE query

• **Returns a graph** comprising all the available triplets about the resource matched in the graph pattern (that is, in the WHERE part of the query)

• Example:

```
PREFIX db: <http://dbpedia.org/ontology/>
DESCRIBE ?movie
WHERE {
    ?movie
        db:starring <http://dbpedia.org/resource/Natalie_Portman> ;
}
```

The query returns a graph comprising all the available triplets about the movie(s) starred by both actresses
CONSTRUCT query

- It is used for creating a new RDF graph from an existing one
- It is for RDF graph
  somewhat the same as XSLT for XML data
Task 10: Map the data about musicians’ date and place of birth from DBpedia to Bio vocabulary

PREFIX db-ont: <http://dbpedia.org/ontology/>
PREFIX bio: <http://purl.org/vocab/bio/0.1/>
PREFIX dcterm: <http://purl.org/dc/terms/>

CONSTRUCT {
  ?someone bio:event [ 
    rdf:type bio:Birth ; 
    bio:place ?birthplace ; 
    dcterms:date ?birthdate ].
}

WHERE {
  ?someone rdf:type db-ont:MusicalArtist ;
  db-ont:birthDate ?birthdate ;
  db-ont:birthPlace ?birthplace .
}
Task 11: Establish aunt relationship

PREFIX schema: <http://schema.org/>
PREFIX rel: <http://purl.org/vocab/relationship/>
CONSTRUCT {
    ?aunt rel:hasAunt ?child.
} WHERE {
    ?aunt schema:parent ?grandparent ;
    schema:gender ?gender
}
Queries over multiple distributed data sources

- All the queries we’ve seen so far were executed over data originating from one data source (one RDF graph)

- However, queries could be executed over multiple data sources
  - In that case, we talk about *federated queries*
  - SPARQL 1.1 introduces the SERVICE keyword for defining additional data sources
Task 11: Find all the acquaintances of Leigh Dodds who have the same surname as the pioneers of computing

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?person
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?someone rdf:type yago:ComputerPioneers ;
    foaf:surname ?surname .
  }
}
Learn more about SPARQL

- SPARQL Query Language for RDF - specification
  - http://www.w3.org/TR/rdf-sparql-query/

- SPARQL 1.1 Query Language – specification
  - http://www.w3.org/TR/sparql11-query/

- Search RDF data with SPARQL

- SPARQL by Example
  - http://www.cambridgesemantics.com/semantic-university/sparql-by-example

- A detailed SPARQL tutorial
  - http://www.w3.org/2004/Talks/17Dec-sparql/

- Bring existing data to the Semantic Web
Learn more about SPARQL

- SPARQL screencast
  - http://linkeddata.deri.ie/node/58

- RDF as self-describing data
  - http://goo.gl/Gdr5LG

- SPARQL at the movies

- Bart (Simpson) blackboard queries
  - http://goo.gl/aM9mcd; http://goo.gl/z9qOiH

- Example SPARQL queries over 10+ different RDF datasets
  - http://openuplabs.tso.co.uk/datasets

- SPARQL queries over **Europeana** repository
  - http://europeana.ontotext.com/sparql
Some handy tools for learning SPARQL

- YASGUI – Yet Another SPARQL GUI
  - [http://yasgui.laurensrietveld.nl/](http://yasgui.laurensrietveld.nl/)

- Flint SPARQL Editor
  - [http://openuplabs.tso.co.uk/demos/sparqleditor](http://openuplabs.tso.co.uk/demos/sparqleditor)

- SPARQLer - an online SPARQL query tool
  - [http://www.sparql.org/sparql.html](http://www.sparql.org/sparql.html)

- ARQ, a SPARQL processor for Jena framework
  - [http://jena.sourceforge.net/ARQ/](http://jena.sourceforge.net/ARQ/)