Today’s Plan

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Today’s tip
Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Today's tip
The W3C representation of knowledge in the Semantic Web is RDF (Resource Description Framework)

RDF talks about resources identified by URIs.

In RDF, all knowledge is represented by triples

A triple consists of subject, predicate, and object

The subject maybe a resource or a blank node

The predicate must be a resource

The object can be a resource, a blank node, or a literal
objects of triples can also be literals
- I.e. nodes in an RDF graph can be resources or literals
- Subjects and predicates of triples can not be literals

Literals can be
- Plain, without language tag:
  geo:berlin geo:name "Berlin" .
- Plain, with language tag:
  geo:germany geo:name "Deutschland"@de .
  geo:germany geo:name "Germany"@en .
- Typed, with a URI indicating the type:
  geo:berlin geo:population "3431700"^^xsd:integer .
Blank nodes are like resources without a URI.

There is a city in Germany called Berlin:

```reason
_:x a geo:City .
_:x geo:containedIn geo:germany .
_:x geo:name "Berlin" .
```

Diagram:

- `geo:germany` is the parent of `geo:containedIn`.
- `geo:containedIn` is the parent of `geo:City`.
- `geo:City` is a resource with two properties:
  - `geo:name` with value "Berlin".
  - The resource type `geo:City`.

This diagram represents the RDF statement:

```
_:x a geo:City .
_:x geo:containedIn geo:germany .
_:x geo:name "Berlin" .
```
Jena is a semantic web programming framework
API has interfaces Resource, Property, Literal, Statement, Model
Need to create a Model first, using ModelFactory or ModelMaker.
Different kinds of models have different backing storage (memory, files, RDB)
Statements and Resources point back to the model they belong to
Retrieval of information via methods in Model and Resource
Simple pattern matching with null as wildcard possible
Reminder: Jena

Diagram of RDF concepts:
- **Model**
  - 1:1 relationship with **Statement**
- **Statement**
  - *:1 relationship with **Resource** and **Property**
  - Subject relationship with **Resource**
  - Predicate relationship with **Resource**
- **Resource**
  - +URI: String
  - Object relationship with **RDFNode**
  - Optional properties:
    - Literal
      - +lexicalForm: String
      - +language: String
      - +datatypeURI: String
- **Property**

**RDFNode**
Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Todays tip
The RDF Vocabulary

- Prefix `rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
- (needs to be declared like all others!)
- Important elements:
  - `type` links a resource to a type (can be abbreviated).
  - `Resource` type of all resources
  - `Property` type of all properties
- Examples:
  - `geo:containedIn a rdf:Property`.
  - `rdf:type a rdf:Property`.
Common Vocabularies

Friend Of A Friend

- People, personal information, friends, see http://www.foaf-project.org/
- Prefix foaf:<http://xmlns.com/foaf/0.1/>
- Important elements:
  - **Person** a person, alive, dead, real, imaginary
  - **name** name of a person (also firstName, familyName)
  - **mbox** mailbox URL of a person
  - **knows** a person knows another
- Examples:

  <http://heim.ifi.uio.no/martingi/foaf#me>
  a foaf:Person ;
  foaf:name "Martin Giese" ;
  foaf:mbox <mailto:martingi@ifi.uio.no> ;
  foaf:knows <http://.../martigie/foaf#me> .
Dublin Core

- Metadata for documents, see http://dublincore.org/.
- Prefix dct:<http://purl.org/dc/terms/>
- Use this instead of legacy dc:
- Important elements in dct:
  - creator a document’s main author
  - created the creation date
  - description a natural language description
  - replaces another document superseded by this
- Examples:

  <http://heim.ifi.uio.no/martingi/>
  dct:creator <http://.../foaf#me> ;
  dct:created "2007-08-01" ;
  dct:description "Martin Giese’s homepage"@en ;
Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Todays tip
SPARQL

- **SPARQL Protocol And RDF Query Language**
- **SPARQL 1.1** finalized 21st March 2013, this lecture is about SPARQL 1.0.
- **Documentation:**
  - Queries [http://www.w3.org/TR/rdf-sparql-query/](http://www.w3.org/TR/rdf-sparql-query/)  
    Language for submitting “graph pattern” queries
  - Protocol [http://www.w3.org/TR/rdf-sparql-protocol/](http://www.w3.org/TR/rdf-sparql-protocol/)  
    Protocol to submit queries to a server (“endpoint”)
  - Results [http://www.w3.org/TR/rdf-sparql-XMLres/](http://www.w3.org/TR/rdf-sparql-XMLres/)  
    XML format in which results are returned
- **Try it out:**
  - DBLP [http://dblp.l3s.de/d2r/snorql/](http://dblp.l3s.de/d2r/snorql/)
  - DBpedia [http://dbpedia.org/sparql](http://dbpedia.org/sparql)
  - Lenka [http://data.lenka.no/sparql](http://data.lenka.no/sparql)
Simple Examples

- DBLP contains computer science publications
- vocabulary of RDF version:
  - author of a document: dc:creator
  - title of a document: dc:title
  - name of a person: foaf:name

People called "Martin Giese"

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?mg WHERE {
  ?mg foaf:name "Martin Giese" .
}
```

Answer:

<table>
<thead>
<tr>
<th>?mg</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Martin_Giese">http://dblp.l3s.de/d2r/resource/authors/Martin_Giese</a></td>
</tr>
</tbody>
</table>
Publications by people called “Martin Giese”

PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX dc: <http://purl.org/dc/elements/1.1/>  
SELECT ?pub WHERE {
   ?mg foaf:name "Martin Giese" .
   ?pub dc:creator ?mg .
}

Answer:

<table>
<thead>
<tr>
<th>pub</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/cade/Giese01">http://dblp.l3s.de/d2r/resource/publications/conf/cade/Giese01</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/cade/BeckertGHKRSS07">http://dblp.l3s.de/d2r/resource/publications/conf/cade/BeckertGHKRSS07</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/fase/AhrendtBBGHMHMS02">http://dblp.l3s.de/d2r/resource/publications/conf/fase/AhrendtBBGHMHMS02</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/jelia/AhrendtBBGHMHMS00">http://dblp.l3s.de/d2r/resource/publications/conf/jelia/AhrendtBBGHMHMS00</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/lpar/Giese06">http://dblp.l3s.de/d2r/resource/publications/conf/lpar/Giese06</a></td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Titles of publications by people called “Martin Giese”

SELECT ?title WHERE {
    ?mg foaf:name "Martin Giese" .
    ?pub dc:creator ?mg .
}

Answer:

<table>
<thead>
<tr>
<th>?title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Incremental Closure of Free Variable Tableaux.&quot;^^xsd:string</td>
</tr>
<tr>
<td>&quot;The KeY system 1.0 (Deduction Component).&quot;^^xsd:string</td>
</tr>
<tr>
<td>&quot;Saturation Up to Redundancy for Tableau and Sequent Calculi.&quot;^^xsd:string</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Names of people who have published with "Martin Giese"

```
SELECT DISTINCT ?name WHERE {
  ?mg foaf:name "Martin Giese" .
  ?pub dc:creator ?mg .
  ?pub dc:creator ?other .
  ?other foaf:name ?name.
}
```

Answer:

<table>
<thead>
<tr>
<th>?name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Martin Giese&quot;</td>
</tr>
<tr>
<td>&quot;Bernhard Beckert&quot;</td>
</tr>
<tr>
<td>&quot;Reiner Hähnle&quot;</td>
</tr>
<tr>
<td>&quot;Vladimir Klebanov&quot;</td>
</tr>
<tr>
<td>&quot;Philipp Rümmer&quot;</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Graph Patterns

The previous SPARQL query as a graph:

Assign values to variables to make this a sub-graph of the RDF graph!
Graph with blank nodes

Variables not SELECTed can equivalently be blank:

Assign values to variables and blank nodes to make this a sub-graph of the RDF graph!
SPARQL Query with blank nodes

Names of people who have published with “Martin Giese”

```sparql
SELECT DISTINCT ?name WHERE {
   _:mg foaf:name "Martin Giese" .
   _:pub dc:creator _:mg .
   _:pub dc:creator _:other .
   _:other foaf:name ?name.
}
```

The same with blank node syntax

```sparql
SELECT DISTINCT ?name WHERE {
   [ dc:creator [foaf:name "Martin Giese"] ,
     [foaf:name ?name]
   ]
}
```
Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Todays tip
Basic Graph Patterns

- A *Basic Graph Pattern* is a set of triple patterns.
- e.g.
  
  ```sparql
  ?mg foaf:name "Martin Giese" .
  _:pub dc:creator ?mg .
  _:pub dc:creator ?other .
  ```

- Scope of blank node labels is the basic graph pattern
- Matching is defined via *entailment*, see next lecture
- Basically: A match is a function that maps
  - every variable and every blank node in the pattern
  - to a resource, a blank node, or a literal in the RDF graph (an “RDF term”)

INF3580/4580 :: Spring 2015
Lecture 4 :: 9th February
Group Graph Patterns

- Group several patterns with { and }.
- A group containing one basic graph pattern:
  
  ```
  { 
    _:pub dc:creator ?mg .
    _:pub dc:creator ?other .
  }
  ```

- A group containing two groups:
  
  ```
  { 
    { _:pub1 dc:creator ?mg . } 
    { _:pub2 dc:creator ?other . } 
  }
  ```

- Note: Same name for two different blank nodes not allowed!
- Match is a function from variables to RDF terms
- Need to match all the patterns in the group.
Filters

- Groups may include *constraints or filters*
  
  E.g.
  ```sparql
  {  
    ?x a dbpedia-owl:Place ;
    dbpprop:population ?pop .
    FILTER (?pop > 1000000)
  }
  ```

  E.g.
  ```sparql
  {  
    ?x a dbpedia-owl:Place ;
    dbpprop:abstract ?abs .
    FILTER (lang(?abs) = "no")
  }
  ```

- Numerical functions, string operations, reg. exp. matching, etc.
- Reduces matches of surrounding group to those where filter applies
Optional Patterns

- A match can leave some variables *unbound*
- A *partial* function from variables to RDF terms
- Groups may include *optional parts*
- E.g.
  ```sparql
  { 
    ?x a dbpedia-owl:Place ;
    dbpprop:population ?pop .
    OPTIONAL { 
      FILTER (lang(?abs) = "no")
    }
  }
  ```
- ?x and ?pop bound in every match, ?abs bound if there is a Norwegian abstract
- Groups can contain several optional parts, evaluated separately
Matching Alternatives

- A UNION pattern matches if any of some alternatives matches
- E.g.
  
  ```
  { 
       dc:created ?date . } 
     UNION 
     { ?book foaf:maker ?author . } 
     UNION 
     { ?author foaf:made ?book . } 
  }
  ```

  - Variables in matches union of variables in sub-patterns
  - Match of one pattern leaves rest of variables unbound
Four Types of Queries

**SELECT** Compute table of bindings for variables

```
SELECT ?a ?b WHERE {
[ dc:creator ?a ;
  dc:creator ?b ]
}
```

**CONSTRUCT** Use bindings to construct a new RDF graph

```
CONSTRUCT {
} WHERE {
  [ dc:creator ?a ;
    dc:creator ?b ]
}
```

**ASK** Answer (yes/no) whether there is ≥ 1 match

**DESCRIBE** Answer available information about matching resources
Solution Modifiers

- Patterns generate an unordered collection of solutions
- Each solution is a partial function from variables to RDF terms
- SELECT treats solutions as a sequence (solution sequence)
- *Sequence modifiers* can modify the solution sequence:
  - Order
  - Projection
  - Distinct
  - Reduce
  - Offset
  - Limit
- Applied in this order.
ORDER BY

- Used to sort the solution sequence in a given way:
  
  SELECT ... WHERE ... ORDER BY ...

- E.g.

  ```sparql
  SELECT ?country ?city ?pop WHERE {
    ?city geo:containedIn ?country ;
    geo:population ?pop .
  } ORDER BY ?country DESC(?pop)
  ```

- Standard defines sorting conventions for literals, URIs, etc.
Projection, DISTINCT, REDUCED

- Projection means that only some variables are part of the solution
  - Done with `SELECT ?x ?y WHERE {?x ?y ?z...}`
- DISTINCT eliminates duplicate solutions:
  - Done with `SELECT DISTINCT ?x ?y WHERE {?x ?y ?z...}`
  - A solution is duplicate if it assigns the same RDF terms to all variables as another solution.
- REDUCE *allows* to remove some or all duplicate solutions
  - Done with `SELECT REDUCED ?x ?y WHERE {?x ?y ?z...}`
  - Can be expensive to find and remove all duplicates
  - Leaves amount of removal to implementation
OFFSET and LIMIT

- Useful for paging through a large set of solutions
- ...but not useful for implementing paging in applications.
- Can compute solutions number 51 to 60
- Done with
  
  SELECT ... WHERE {...} ORDER BY ...
  LIMIT 10 OFFSET 50

- LIMIT and OFFSET can be used separately
- OFFSET not meaningful without ORDER BY.
Executing SPARQL Queries

Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Today's tip
**SPARQL in Jena**

- SPARQL functionality bundled with Jena has separate Javadocs:
  
  http://jena.apache.org/documentation/javadoc/arq/

- Main classes in package `com.hp.hpl.jena.query`
  
  - Query a SPARQL query
  - QueryFactory for creating queries in various ways
  - QueryExecution for the execution state of a query
  - QueryExecutionFactory for creating query executions
  - ResultSet for results of a SELECT

- `CONSTRUCT` and `DESCRIBE` return Models, `ASK` a Java boolean.

- SPARQL with ARQ is generally slow due to triple pattern breakdown.
Constructing a Query and a QueryExecution

- Query objects are usually constructed by parsing:
  
  ```java
  String qStr =
  "PREFIX foaf: <" + foafNS + ">
  + "SELECT ?a ?b WHERE {
  + " ?a foaf:knows ?b ."
  + "} ORDER BY ?a ?b";
  
  Query q = QueryFactory.create(qStr);
  ```

- Programming interface deprecated and badly documented

- A Query can be used several times, on multiple models

- For each execution, a new QueryExecution is needed

- To produce a QueryExecution for a given Query and Model:
  ```java
  QueryExecution qe =
  QueryExecutionFactory.create(q, model);
  ```
Executing a Query

- QueryExecution contains methods to execute different kinds of queries (SELECT, CONSTRUCT, etc.)
- E.g. for a SELECT query:
  ```java
  ResultSet res = qe.execSelect();
  ```
- ResultSet is a sub-interface of Iterator<QuerySolution>
- Also has methods to get list of variables
- QuerySolution has methods to get list of variables, value of single variables, etc.
- Important to call close() on query executions when no longer needed.
Example: SPARQL in Jena

String qStr = "SELECT ?a ?b ...";
Query q = QueryFactory.create(qStr);

QueryExecution qe =
    QueryExecutionFactory.create(q, model);

try {
    res = qe.execSelect();
    while( res.hasNext()) {
        QuerySolution soln = res.next();
        RDFNode a = soln.get("?a");
        RDFNode b = soln.get("?b");
        System.out.println(""+a+" knows "+b);
    }
} finally {
    qe.close();
}
Many sites (DBLP, dbpedia, dbtunes, ...) publish SPARQL endpoints. I.e. SPARQL queries can be submitted to a database server that sends back the results. Uses HTTP to submit URL-encoded queries to server:
GET /sparql/?query=... HTTP/1.1
Actually defined via W3C Web Services, see:
http://www.w3.org/TR/rdf-sparql-protocol/
For SELECT queries you get a XML or JSON result set, see:
http://www.w3.org/TR/rdf-sparql-XMLres/
http://www.w3.org/TR/rdf-sparql-json-res/
Nothing you would want to do manually!
Remote SPARQL with Jena

- Jena can send SPARQL queries to a remote endpoint!
- Use one of the `sparqlService` in `QueryExecutionFactory`
- E.g.
  ```java
  String endpoint = "http://dblp.l3s.de/d2r/sparql";
  String qStr = "SELECT ?a ?b ...";
  Query q = QueryFactory.create(qStr);

  QueryExecution qe =
      QueryExecutionFactory.sparqlService(endpoint,q);

  try {
      res = qe.execSelect();
      ...
  } finally {
      qe.close();
  }
  ```
Summary

- SPARQL is a W3C-standardised query language for RDF graphs
- It is built about “graph patterns”
- Only queries compatible with “open world assumption”
- Comes with a protocol to communicate with “endpoints”
- Can be conveniently used with Jena and tens of other systems.
Outline

1. Repetition: RDF
2. Common Vocabularies
3. SPARQL By Example
4. SPARQL Systematically
5. Executing SPARQL Queries
6. More to come!
7. Todays tip
SPARQL contains a mechanism for named RDF graphs

Collections of named graphs are called “RDF datasets”

Syntax for declaring named graphs in SPARQL

Syntax for matching graph patterns in a given graph
SPARQL 1.1 became W3C Recommendations 21 March 2013.

- Updates (add delete triples)
- Service Descriptions
- Basic Federated query
- Subqueries.
- Property paths (to shorten common queries)
- Aggregate functions (count, sum, average,...)
- Negation, set difference, i.e. something is not in a graph
- Entailment
Todays tip

Command line tools

From the Redland-library

   rapper data.rdf

This will parse and print N-Triples og what it finds if it validates.

   rapper -i turtle data.ttl

will do the same for Turtle

Install yourself on Debian/Ubuntu:

   apt-get install raptor-utils

May also suggest:

   apt-get install redland-utils rasqal-utils
The Perl-stack

- Attean revolutionary new low-level traits based API.
- RDF::Trine low-level API, like Jena.
- RDF::Query full SPARQL 1.1-implementation, like ARQ.
- perlrdf comprehensive command line tool.
- RDF::Endpoint, SPARQL Endpoint, like Fuseki.
- RDF::LinkedData, servers data from a triple-store based on Linked Data principles, optionally with extra good practices.
- Test::RDF, testing RDF-data.
- RDF::RDFa::Parser, RDFa parser.

Can also be found in Debian (perlrdf only in testing):

```
apt-get install librdf-query-perl librdf-linkeddata-perl perlrdf
```