INF3580 – Semantic Technologies – Spring 2010

Lecture 14: Presenting Relational Databases as RDF

Martin Giese

25th May 2010

Department of Informatics
University of Oslo

Today's Plan

Outline

Relational Database Management Systems

- "Relational" databases introduced in 1970
  - Replaced navigational and hierarchical systems
- Mostly used with query language SQL
- Most of the world's business data today is stored in relational databases
- Several freely available systems:
  - PostgreSQL
  - MySQL
  - SQLite
  - ...
- Many commercial systems:
  - Oracle
  - IBM DB2
  - Microsoft Access, SQL Server
  - ...

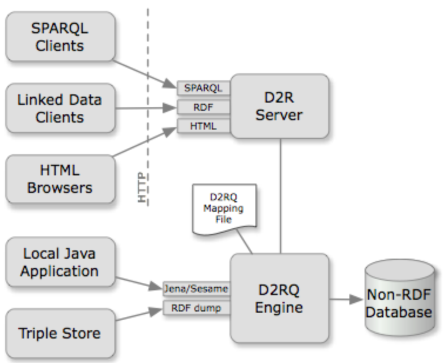
RDBMS to RDF

- Need a way to make data in RDBMS available as RDF
- First idea: RDF export
  - Read all records, export RDF
  - Bad idea: data replication. . .
  - Probably won't switch whole enterprise to RDF store
  - Need to convert to RDF regularly
- Often a better idea: RDF view
  - SPARQL endpoint translates incoming queries to SQL
  - Translates result to SPARQL SELECT result or RDF
  - Data remains where it is, no duplication
  - Drawback: need to keep "old-fashioned" DB backend

Outline

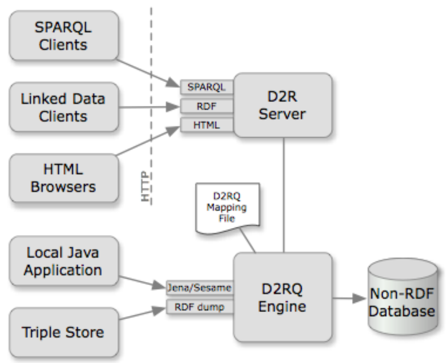
D2R/D2RQ

- Allows to treat relational databases as RDF
- Developed by FU Berlin
- Mapping describes relation between DB and RDF
- Can create SPARQL endpoint without transforming the whole database: *Virtual* RDF graph.
- Also on-demand RDF/HTML pages following LOD protocol



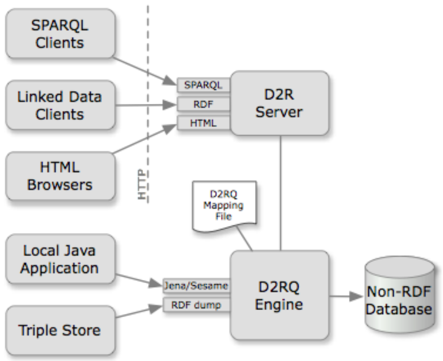
D2RQ Engine

- Reads a "Mapping File"
  - Table → Class
  - Row → Resource
  - Column → Property
  - RDF-encoded
- Translates SPARQL to SQL
- Can also act as Jena Graph
- Or the Sesame equivalent
- Can also export whole DB



D2R Server

- Provides WWW-frontend
- SPARQL Endpoint
- Serves RDF as linked open data
- Pages of data for HTTP browsers
- All requests translated to SPARQL



Example: World Database

- An example database from MySQL distribution
- Table City:
  - ID (key): a unique number
  - Name: the city's name
  - CountryCode: Code for the country the city lies in
  - ...
- Table Country:
  - Code (key): the code for a country
  - Name: the Country's name
  - Continent: the Continent the country lies in
  - Capital: the City ID of the country's capital
  - ...

Example: World Database (cont.)

- Table City:

ID	Name	CountryCode	...
		...	
2806	Kingston	NFK	...
2807	Oslo	NOR	...
2808	Bergen	NOR	...
		...	

- Table Country:

Code	Name	Continent	Capital	...
		...		
NLD	Netherlands	Europe	5	...
NOR	Norway	Europe	2807	...
NPL	Nepal	Asia	2729	...
		...		

Automatic Mapping

- Call D2R program `generate-mapping`
  - (Requires access information for database)
- Generates a mapping file for:
  - one `rdfs:Class` for each table
  - one resource per DB row
  - one data-property per column (ie. literal objects)
  - plus one `rdfs:label` for every resource
- Uses automatically generated class and property names

Generated RDF for Automatic Mapping

```
<http://.../City/2807> a vocab:City ;
        rdfs:label "City #2807" ;
        vocab:City_Name "Oslo" ;
        vocab:City_CountryCode "NOR" .

<http://.../Country/NOR> a vocab:Country ;
        rdfs:label "Country #NOR" ;
        vocab:Country_Name "Norway" ;
        vocab:Country_Continent "Europe" ;
        vocab:Country_Capital "2807"
```

- Only literals, no URI-links between Oslo and Norway
- No attempt to introduce a class for continents
- Solution: refine the generated mapping file manually

Outline

Where Classes Come From

- From the generated mapping file:

```
map:City a d2rq:ClassMap ;
    d2rq:dataStorage map:database ;
    d2rq:uriPattern "City/@@City.ID@@" ;
    d2rq:class vocab:City ;
    d2rq:classDefinitionLabel "City" .
```

- identify a "class mapping"
- link to a resource describing the DB connection
- give the pattern for resources of this class
  - contains placeholder with DB table and column
- give the RDFS class for those resources
- give the label for that class.

Resources for Continents

- Add to mapping file:

```
map:Continent a d2rq:ClassMap ;
    d2rq:dataStorage map:database ;
    d2rq:uriPattern "Continent/@@Country.Continent|urlify@@";
    d2rq:class vocab:Continent ;
    d2rq:classDefinitionLabel "Continent" .
```

- For everything in the `Continent` column of `Country`...
- ...generate a resource with URI `.../Continent/...`
- ...removing spaces from "North America", etc.
- E.g. `http://.../resource/Continent/North_America`

Where Properties Go To

- In original mapping file:

```
map:City_CountryCode a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap map:City ;
    d2rq:property vocab:City_CountryCode ;
    d2rq:propertyDefinitionLabel "City CountryCode" ;
    d2rq:column "City.CountryCode" .
```

- Identify a “property bridge”
- that adds properties to the resources described in `map:City`
- give the predicate
- give a label to the predicate
- the object is a *literal* taken from this column
- Also possible to define literals with patterns containing columns

Linking Cities to Countries

- Replace the previous property bridge with:

```
map:City_CountryCode a d2rq:PropertyBridge ;
    d2rq:belongsToClassMap map:City ;
    d2rq:property vocab:City_Country ;
    d2rq:propertyDefinitionLabel "City Country" ;
    d2rq:refersToClassMap map:Country ;
    d2rq:join "City.CountryCode=>Country.Code" .
```

- Foreign key: link to resource from another class map
- Say how columns for `map:City` correspond to those for `map:Country`

Linking Countries to Capitals

- Replace:

```
map:Country_Capital a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Country;
    d2rq:property vocab:Country_Capital;
    d2rq:propertyDefinitionLabel "Country Capital";
    d2rq:column "Country.Capital" .
```

- By:

```
map:Country_Capital a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Country;
    d2rq:property vocab:capital;
    d2rq:propertyDefinitionLabel "Country Capital";
    d2rq:refersToClassMap map:City;
    d2rq:join "Country.Capital=>City.ID";
```


Resulting Graph

```
<http://.../City/2807> a vocab:City ;
        rdfs:label "City #2807" ;
        vocab:City_Name "Oslo" ;
        vocab:City_Country <http://.../Country/NOR> .

<http://.../Country/NOR> a vocab:Country ;
        rdfs:label "Country #NOR" ;
        vocab:Country_Name "Norway" ;
        vocab:Country_Continent "Europe" ;
        vocab:Country_Capital <http://.../City/2807> .
```

Linking to DBpedia

- Add property bridge:

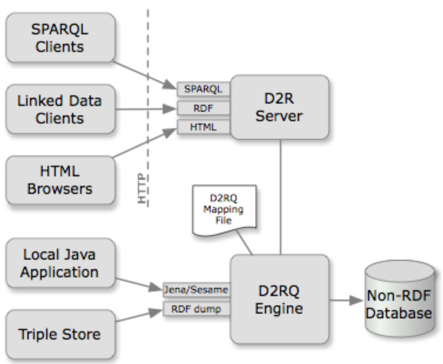
```
map:Country_DBpedia a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Country;
    d2rq:property owl:sameAs;
    d2rq:uriPattern
"http://dbpedia.org/resource/@@Country.Name|urlify@@" .
```

- No problem to use "external" properties or classes
- No problem to link to "external" URIs.
- Careful: Generating links like this often fails for some cases:
  - World DB country name: `Sao Tome and Principe`
  - DBpedia URI: `http://.../São_Tomé_and_Príncipe`
- Better in general to have a DB table with corresponding URIs

Outline

The Jena Adapter

- No direct way of adding reasoning to D2R
- An RDF view of a database can be made available as a Jena Model
- Requires mapping file and d2rq.jar
- Add reasoning to that model



The Jena Adapter: Example

`Model m = new ModelD2RQ("file:mapping.n3");`

- Create a model backed by a DB through D2R
- No data is read into memory

`OntModel om = ModelFactory.createOntologyModel();`

`om.read("file:world.owl");`

- Create model with ontology, e.g.
- `vocab:City rdfs:subClassOf vocab:Place`
- `vocab:Country rdfs:subClassOf vocab:Place`

`Model infm = ModelFactory.createRDFSModel(om, m);`

- Asking `infm` for all objects of type `vocab:Place`...
- ...gives all cities...
- ...and all countries!
- Can use Jena query engine for SPARQL queries with reasoning
- But does it still not read data into memory?

Forward Chaining vs. Backward Chaining

- Given: reasoning rules, like e.g.:

$$\frac{x \ \texttt{rdf:type} \ C \qquad C \ \texttt{rdfs:subClassOf} \ D}{x \ \texttt{rdf:type} \ D}$$

- Forward Chaining:
  - Add all consequences of rules to the model
  - Queries can be answered using the expanded model
- Backward Chaining:
  - Leave model as it is
  - Answer queries by applying rules "backwards"
  - A bit like Prolog!

Example for Forward Chaining

- Given triples:

```
:City rdfs:subClassOf :Place
:Oslo rdf:type :City
```

- Inferred triples:

```
:Oslo rdf:type :Place
:Place rdf:type rdfs:Class
:Place rdfs:subClassOf rdfs:Resource
...
```

- To answer x `rdf:type :Place`:
  - Simply look in model:
  - $x \to$ `:Oslo`

Example for Backward Chaining

- Given triples:

```
:City rdfs:subClassOf :Place
:Oslo rdf:type :City
```

- To answer x `rdf:type :Place`:
  - Look for direct occurrences: none
  - Look for instances of:
    - C `rdf:subClassOf :Place`
    - x `rdf:type` C
  - E.g. $C \to$ `:City`, $x \to$ `:Oslo`
- In general, need to backward-chain over many rules!
  - E.g. C `rdf:subClassOf :Place` could come from other rules

Forward Chaining vs. Backward Chaining

Forward	**Backward**
reason once	repeated computation
diffuse	goal-oriented
adds to data	data unchanged
much space	little space
expensive up-front	cheap up-front
fast queries	slow queries
possibly non-terminating expansion	possibly non-terminating backward chaining

- "Hybrid" approaches possible, e.g. Jena RDFS reasoner
  - Forward chaining for sub-class/prop. hierarchy, ranges, domains
  - Backward chaining for `rdf:type`
- Forward chaining difficult for data in databases
  - RDFS reasoner OK for databases
  - Pellet etc. in general not

OWL 2 Profiles

OWL QL Based on "DL-Lite$_\mathcal{A}$". Allows query answering by "query rewriting", i.e. backward chaining. Same data-efficiency as SQL.

OWL RL Based on "pD*" semantics for OWL. Allows terminating exhaustive forward chaining.

OWL EL Based on "$\mathcal{EL}^{++}$". Shown to allow query answering by query rewriting after some amount of preprocessing.

- QL and RL "maximal" with these properties. EL originally defined for efficient classification.
- Query processors for these profiles still academic.
  - Google for "QuOnto" for work on OWL QL/DL-Lite.

Outline

Topics Covered

- RDF, principles, Turtle syntax
- The Jena API for RDF
- The SPARQL Query Language
- Basics of the RDFS and OWL ontology languages
- Basics of model semantics and reasoning
- Linked Open Data, RDFa
- Publishing Databases as RDF

Topics *Not* Covered

- Rule Languages (SWRL, RIF, Jena rules, etc.)
- SW application structures
- Semantic Web Services
- Details of RDF/RDFS model semantics
- Some details of OWL
- Details of OWL 2 profiles
- Logical theory: Soundness, Completeness,. . .
  - (You ain't seen nothing yet :-)
- And many more!

Help! I Can't Get Enough!

- For more information on theory:
  - Book on Foundations of SW Technologies
  - Take a course in logic or automated reasoning
- For more information on practical questions:
  - Book on Semantic Web Programming
  - Standards texts on W3C Web pages
  - Google
- Still not enough?
  - Contact us for possible MSc topics!