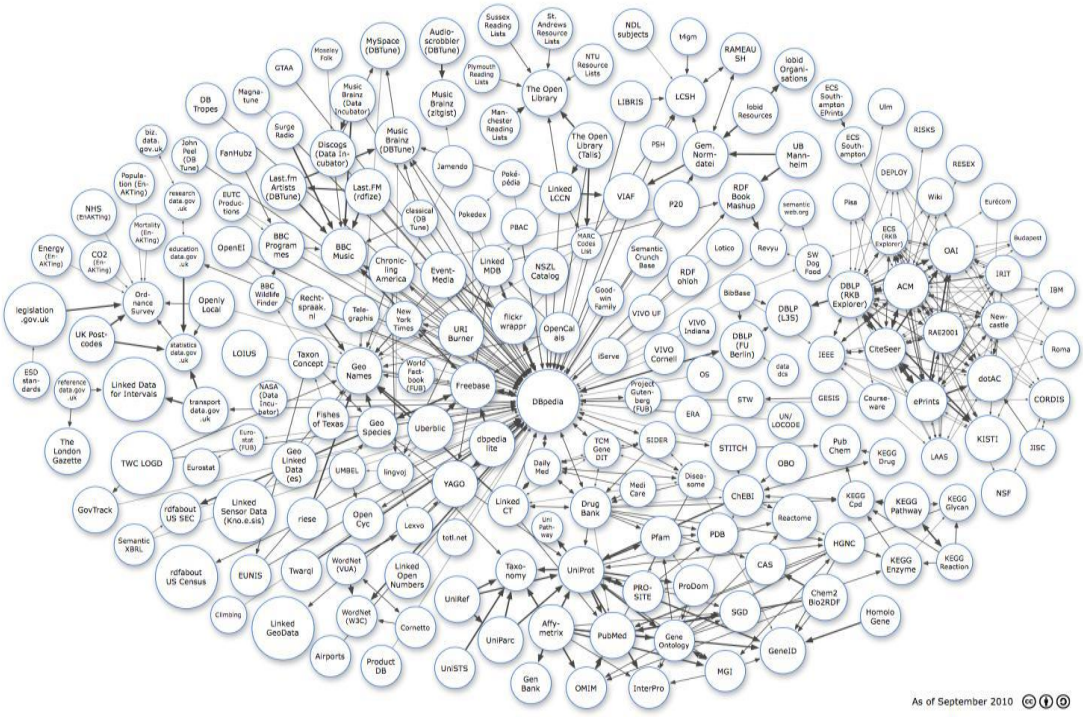
ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι. ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

# BACHELOR OF SCIENCE FINAL THESIS

## Linked Data on the Web




**Student**

**Anastasopoulos Theodoros**

**ID: 06/ 303**

**Supervisor**

**Dimitris Ach. Dervos**

## CONTENTS

# ABSTRACT

# 1. INTRODUCTION, LINKED DATA O THE WEB

The Web is increasingly understood as a global information space consisting not just of linked documents, but also of Linked Data. More than just a vision, the resulting Web of Data has been brought into being by the maturing of the Semantic Web technology stack, and by the publication of an increasing number of data sets according to the principles of Linked Data. It describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP and URIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried. The goal of Linked Data is to enable people to share structured data on the Web as easily as they can share documents today. In summary, Linked Data is simply about using the Web to create typed links between data from different sources.

The basic tenets of Linked Data are to:

- use the RDF data model to publish structured data on the Web
- use RDF links to interlink data from different data sources

Applying both principles leads to the creation of a data commons on the Web, a space where people and organizations can post and consume data about anything. This data commons is often called the Web of Data or Semantic Web.

The Web of Data can be accessed using Linked Data browsers, just as the traditional Web of documents is accessed using HTML browsers. However, instead of following links between HTML pages, Linked Data browsers enable users to navigate between different data sources by following RDF links. This allows the user to start with one data source and then move through a potentially endless Web of data sources connected by RDF links. For instance, while looking at data about a person from one source, a user might be interested in information about the person's home town. By following an RDF link, the user can navigate to information about that town contained in another dataset.

Just as the traditional document Web can be crawled by following hypertext links, the Web of Data can be crawled by following RDF links. Working on the crawled data, search engines can provide sophisticated query capabilities, similar to those provided by conventional relational databases. Because the query results themselves are structured data, not just links to HTML pages, they can be immediately processed, thus enabling a new class of applications based on the Web of Data.

The glue that holds together the traditional document Web is the hypertext links between HTML pages. The glue of the data web is RDF links. An RDF

link simply states that one piece of data has some kind of relationship to another piece of data. These relationships can have different types. For instance, an RDF link that connects data about people can state that two people know each other. An RDF link that connects information about a person with information about publications in a bibliographic database might state that a person is the author of a specific paper.

There is already a lot of structured data accessible on the Web through Web 2.0 APIs such as the eBay, Amazon, Yahoo, and Google Base APIs. Compared to these APIs, Linked Data has the advantage of providing a single, standardized access mechanism instead of relying on diverse interfaces and result formats. This allows data sources to be:

- more easily crawled by search engines,
- accessed using generic data browsers, and

Enables links between data from different data sources.

Having provided a background to Linked Data concepts, the rest of this document is structured as follows: **Section 2** outlines the basic principles of Linked Data. **Section 3** provides practical advice on how to name resources with URI references. **Section 4** discusses terms from well-known vocabularies and data sources which should be reused to represent information. **Section 5** explains what information should be included into RDF descriptions that are published on the Web. **Section 6** shows us the basic way to set RDF links to other data sources while in **Section 7** we focus on how we can serve the information as linked data. Finally in the **Sections 8** and **9** we figure the way we can handle this information from an endpoint using the well known language and basic tool SPARQL and in the same time we introduce the basic tool / server D2RQ Platform. We use the functions of the D2R Server in order to translate a relational DB to Linked Data view and upload this information in a website.

# 2. BASIC PRINCIPLES

This chapter describes the basic principles of Linked Data. As Linked Data is closely aligned to the general architecture of the Web, we first summarize the basic principles of this architecture. Then we give an overview of the RDF data model and recommend how the data model should be used in the Linked Data context.

## 2.1 Web Architecture

This section summarizes the basic principles of the Web Architecture and introduces terminology such as *resources* and *representation*.

## Resources

Information resources are resources, identified by URIs and whose essential characteristics can be conveyed in a message. The pages and documents familiar to users of the Web are information resources. Information resources typically have one or more representations that can be accessed using HTTP. It is these representations of the resource that flow in messages. The act of retrieving a representation of a resource identified by a URI is known as **dereferencing** that URI. Applications, such as browsers, render the retrieved representation so that it can be perceived by a user. Most Web users do not distinguish between a resource and the rendered representation they receive by accessing it.

Information resources make up the vast majority of the Web today. Their behavior is well understood. In particular, information resources have representations which are, in some sense, 'obvious'. The essence of an information resource is information. Consequently, the act of creating a representation is simply a transformation of that information into an appropriate form. Often that transformation will include formatting that allows the rendered representation to be used conveniently by a Web user.

As an example, let's consider the creation of a statement of activity for a particular month for a particular bank account. We'll suppose that a URI identifies the resource which, in this case, is a particular set of of binary data held in a relational database. To create a representation of the resource, the appropriate data is first extracted from the database and converted to textual form. Then it is embedded in a stream of HTML markup that also references appropriate styling information. This representation flows across the Web to a browser, where it is rendered. A user is able to perceive the rendered form and to understand the activity on the account for month in question.

The process of creating and rendering representations from information resources is so common that it is often either overlooked or considered to be completely ubiquitous. However, not all Web resources are necessarily associated with obvious representations.

## Representations

Information resources can have *representations*. A representation is a stream of bytes in a certain format, such as HTML, RDF/XML, or JPEG. For example, an invoice is an information resource. It could be represented as an HTML page, as a printable PDF document, or as an RDF document. A single information resource can have many different representations, e.g. in different formats, resolution qualities, or natural languages.

## Associating Information Resources with Other Resources

The representations of information resources associated with other kinds of resource can be extremely useful. However, it would be misleading to claim that they are representations of the resource itself.

Information resources associated with a non-information resource need to have their own URIs. They are themselves distinct resources and provide representations. They may have uses other than providing additional information about the non-information resource. However, the fact that they are associated with a non-information resource is important.

## W3C Technical Architecture Group (TAG)

The W3C Technical Architecture Group (TAG) distinguishes between two kinds of resources: information resources and non-information resources (also called 'other resources'). This distinction is quite important in a Linked Data context. All the resources we find on the traditional document Web, such as documents, images, and other media files, are information resources. But many of the things we want to share data about are not: People, physical products, places, proteins, scientific concepts, and so on. As a rule of thumb, all “real-world objects” that exist outside of the Web are non-information resources.

## Differencing HTTP URIs

URI Dereferencing is the process of looking up a URI on the Web in order to get information about the referenced resource. The W3C TAG draft finding about Dereferencing HTTP URIs introduced a distinction on how URIs identifying information resources and non-information resources are dereferenced:

- *Information Resources*: When a URI identifying an information resource is dereferenced, the server of the URI owner usually generates a new representation, a new snapshot of the information resource's current state, and sends it back to the client using the HTTP response code 200 OK.
- *Non-Information Resources* cannot be dereferenced directly. Therefore Web architecture uses a trick to enable URIs identifying non-information resources to be dereferenced: Instead of sending a representation of the resource, the server sends the client the URI of an information resource which describes the non-information resource using the HTTP response code 303. This is called a 303 redirect. In a

second step, the client dereferences this new URI and gets a representation describing the original non-information resource.

## Content Negotiation

HTML browsers usually display RDF representations as raw RDF code, or simply download them as RDF files without displaying them. This is not very helpful to the average user. Therefore, serving a proper HTML representation in addition to the RDF representation of a resource helps humans to figure out what a URI refers to.

This can be achieved using an HTTP mechanism called *content negotiation*. HTTP clients send HTTP headers with each request to indicate what kinds of representation they prefer. Servers can inspect those headers and select an appropriate response. If the headers indicate that the client prefers HTML, then the server can generate an HTML representation. If the client prefers RDF, then the server can generate RDF.

*Content negotiation for non-information resources* is usually implemented in the following way. When a URI identifying a non-information resource is dereferenced, the server sends a 303 redirect to an information resource appropriate for the client. Therefore, a data source often serves three URIs related to each non-information resource, for instance:

- http://www4.wiwiss.fu-berlin.de/factbook/resource/Russia (URI identifying the non-information resource Russia)
- http://www4.wiwiss.fuberlin.de/factbook/data/Russia (information resource with an RDF/XML representation describing Russia)
- http://www4.wiwiss.fuberlin.de/factbook/page/Russia (information resource with an HTML representation describing Russia)

Figure 1 below shows how dereferencing a HTTP URI identifying a non-information resource plays together with content negotiation:

1. The client performs an HTTP GET request on a URI identifying a non-information resource. In our case a vocabulary URI. If the client is a Linked Data browser and would prefer an RDF/XML representation of the resource, it sends an Accept: application/ rdf + xml header along with the request. HTML browsers would send an Accept: text / html header instead.

2. The server recognizes the URI to identify a non-information resource. As the server can not return a representation of this resource, it answers using the HTTP 303 See Other response code and sends the client the URI of an information resource describing the non-information resource. In the RDF case: RDF content location.
3. The client now asks the server to GET a representation of this information resource, requesting again application/ rdf + xml.
4. The server sends the client a RDF/XML document containing a description of the original resource vocabulary URI.

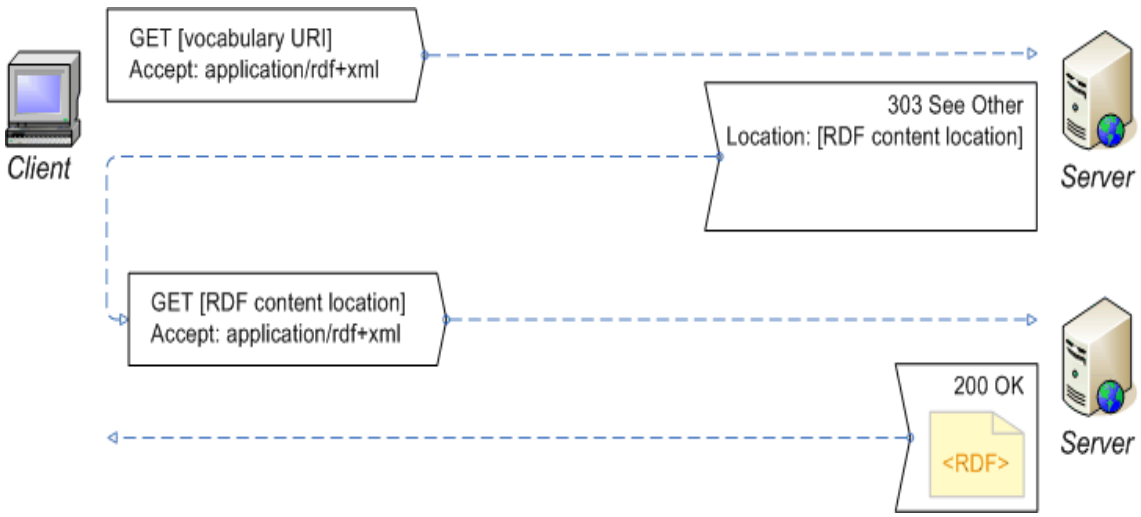



Figure 1: “Content Negotiation between server and client”

## URI Aliases

In an open environment like the Web it often happens that different information provider’s talk about the same non-information resource, for instance a geographic location or a famous person. As they do not know about each other, they introduce different URIs for identifying the same real-world object. For instance: DBpedia data source providing information that has been extracted from Wikipedia uses the URI http://dbpedia.org/resource/Berlin to identify Berlin. Geonames is a data source providing information about millions of geographic locations uses the URI http://sws.geonames.org/2950159/ to identify Berlin. As both URIs refer to the same non-information resource, they are called URI aliases. URI aliases are common on the Web of Data, as it cannot realistically be expected that all information providers agree on the same URIs to identify a non-

information resources. URI aliases provide an important social function to the Web of Data as they are dereferenced to different descriptions of the same non-information resource and thus allow different views and opinions to be expressed. In order to still be able to track that different information providers speak about the same non-information resource, it is common practice that information providers set owl: sameAs links to URI aliases they know about.

## 2.2. The RDF Data Model

When publishing Linked Data on the Web, we represent information about resources using the Resource Description Framework (RDF). RDF provides a data model that is extremely simple on the one hand but strictly tailored towards Web architecture on the other hand.

In RDF, a description of a resource is represented as a number of *triples*. The three parts of each triple are called its *subject*, *predicate*, and *object*. A triple mirrors the basic structure of a simple sentence, such as this one:

```
Chris          has the email address        chris@bizer.de.

(Subject)            (Predicate)                  (Object)
```

The subject of a triple is the URI identifying the described resource. The object can either be a simple *literal value*, like a string, number, or date or the URI of another resource that is somehow related to the subject. The predicate, in the middle, indicates what kind of relation exists between subject and object, e.g. this is the name or date of birth (in the case of a literal), or the employer or someone the person knows (in the case of another resource). The predicate is a URI too. These predicate URIs come from *vocabularies*, collections of URIs that can be used to represent information about a certain domain.

Some people like to imagine a set of RDF triples as an RDF graph. The URIs occurring as subject and object URIs are the nodes in the graph, and each triple is a directed arc (arrow) that connects the subject to the object.

Two principal types of RDF triples can be distinguished, Literal Triples and RDF Links:

**Literal Triples**

Have an RDF literal such as a string, number, or date as the object. Literal triples are used to describe the properties of resources. For

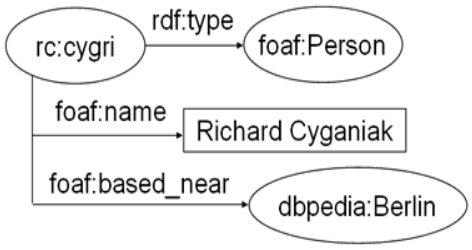
instance, literal triples are used to describe the name or date of birth of a person.

**RDF Links**

Represent typed links between two resources. RDF links consist of three URI references. The URIs in the subject and the object position of the link identify the interlinked resources. The URI in the predicate position defines the type of the link. For instance, an RDF link can state that a person is *employed* by an organization. Another RDF link can state that the persons *know* certain other people.

RDF links are the foundation for the Web of Data. Dereferencing the URI that appears as the destination of a link yields a description of the linked resource. This description will usually contain additional RDF links which point to other URIs that in turn can also be dereferenced, and so on. This is how individual resource descriptions are woven into the Web of Data. This is also how the Web of Data can be navigated using a Linked Data browser or crawled by the robot of a search engine.

Let's take an RDF browser like Disco or Tabulator as an example. The surfer uses the browser to display information about Richard from his FOAF profile. Richard has identified himself with the URI http://richard.cyganiak.de/foaf.rdf#cygri. When the surfer types this URI into the navigation bar, the browser dereferences this URI over the Web, asking for content type application / rdf + xml and displays the retrieved information. In his profile, Richard has stated that he is based near Berlin, using the DBpedia http://dbpedia.org/resource/Berlin as URI alias for the non-information resource Berlin. As the surfer is interested in Berlin, he instructs the browser to dereference this URI by clicking on it. The browser now dereferences this URI asking for application / rdf + xml [Figure 2].




GET /resource/Berlin HTTP/1.0
Accept: application/rdf+xml

Figure 2

After being redirected with a HTTP 303 response code, the browser retrieves an RDF graph describing Berlin in more detail. A part of this graph is shown below. The graph contains a literal triple stating that Berlin has 3.405.259 inhabitants and another RDF link to a resource representing a list of German cities [Figure 3].

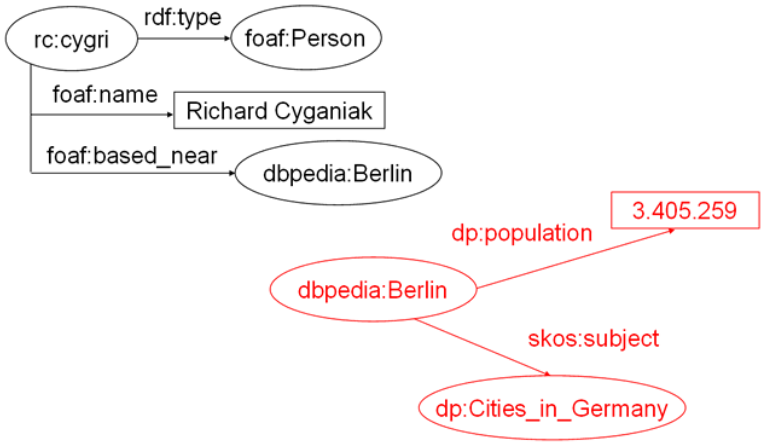



Figure 3

As both RDF graphs share the URI http://dbpedia.org/resource/Berlin, they naturally merge together, as shown below [Figure 4].

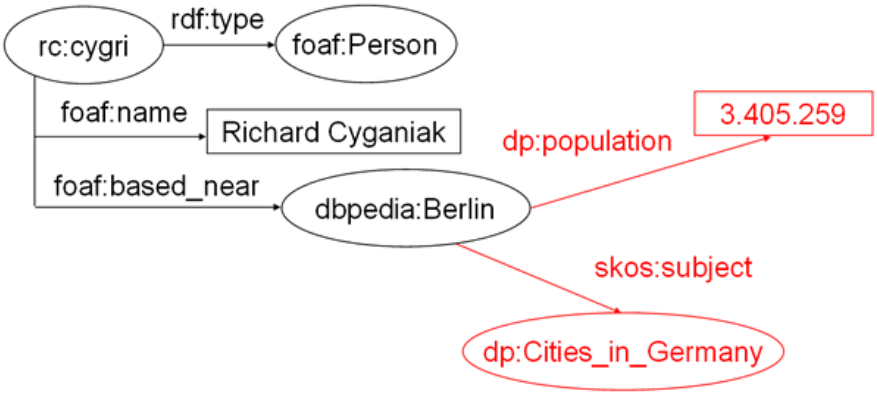



Figure 4

The surfer might also be interested in other German cities. Therefore he lets the browser dereference the URI identifying the list. The retrieved RDF graph contains more RDF links to German cities, for instance, Hamburg and München as shown below.

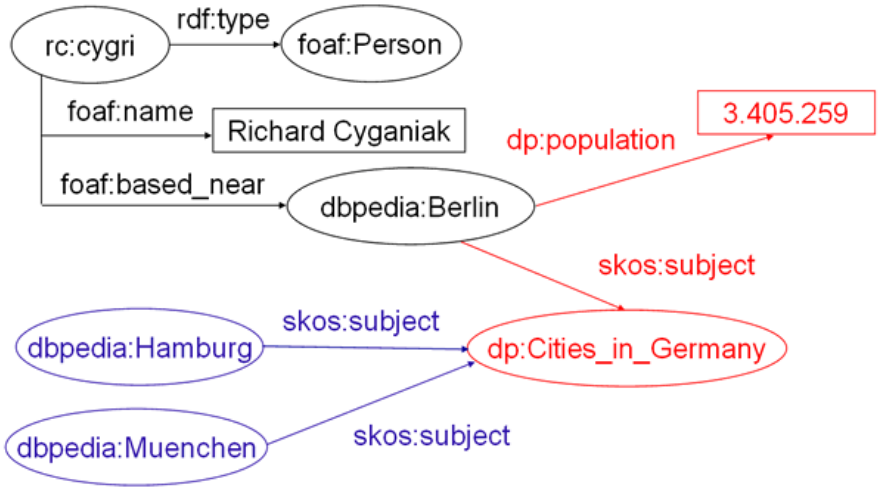



Figure 5

Seen from a Web perspective, the most valuable RDF links are those that connect a resource to external data published by other data sources, because they link up different islands of data into a Web. Technically, such an external RDF link is a RDF triple which has a subject URI from one data source and an object URI from another data source. The box below contains various external RDF links taken from different data sources on the Web.

## Examples of External RDF Links

**Two RDF links taken from DBpedia**

```
<http://dbpedia.org/resource/Berlin>

        Owl:sameAs <http://sws.geonames.org/2950159/> .

<http://dbpedia.org/resource/Tim_Berners-Lee>

        Owl: sameAs <http://www4.wiwiss.fu-
        berlin.de/dblp/resource/person/100007> .
```

**RDF links taken from Tim Berners-Lee's FOAF profile**

```
<http://www.w3.org/People/Berners-Lee/card#i>

        Owl:sameAs <http://dbpedia.org/resource/Tim_Berners-
        Lee>;

        foaf:knows <http://www.w3.org/People/Connolly/#me> .
```

### Benefits of using the RDF Data Model in the Linked Data Context

The main benefits of using the RDF data model in a Linked Data context are that:

- Clients can look up every URI in an RDF graph over the Web to retrieve additional information.
- Information from different sources merges naturally.

The data model enables you to set RDF links between data from different sources.

- The data model allows you to represent information that is expressed using different schemata in a single model.
- Combined with schema languages such as RDF-S or OWL, the data model allows you to use as much or as little structure as you need, meaning that you can represent tightly structured data as well as semi-structured data.

## 3. CHOOSING HTTP URIs

Resources are named with URI references. When publishing Linked Data, we should devote some effort to choosing good URIs for our resources.

On the one hand, they should be *good names* that other publishers can use confidently to link to your resources in their own data. On the other hand, we will have to put technical infrastructure in place to make them *dereferenceable*, and this may put some constraints on what we can do.

This section lists, in loose order, some things we have to keep in mind.

- Use HTTP URIs for everything. The http:// scheme is the only URI scheme that is widely supported in today's tools and infrastructure. All other schemes require extra effort for resolver web services, dealing with identifier registrars, and so on.
- Define our URIs in an HTTP namespace under our control, where we actually can make them dereferenceable. Do not define them in someone else's namespace.
- Keep implementation craft out of your URIs. Short, mnemonic names are better. Consider these two examples:

    - http://dbpedia.org/resource/Berlin
    - http://www4.wiwiss.fu-berlin.de:2020/demos/dbpedia/cgi-bin/resources.php?id=Berlin
- Try to keep our URIs stable and persistent. Changing our URIs later will break any already-established links, so it is advisable to devote some extra thought to them at an early stage.
- We often end up with three URIs related to a single non-information resource:

    1. an identifier for the resource,
    2. an identifier for a related information resource suitable to HTML browsers (with a web page representation)
    3. an identifier for a related information resource suitable to RDF browsers (with an RDF/XML representation).

    Here are several ideas for choosing these related URIs:

    4. http://dbpedia.org/resource/Berlin
    5. http://dbpedia.org/page/Berlin
    6. http://dbpedia.org/data/Berlin

    Or:

    7. http://id.dbpedia.org/Berlin
    8. http://pages.dbpedia.org/Berlin
    9. http://data.dbpedia.org/Berlin

    Or:

10. http://dbpedia.org/Berlin
11. http://dbpedia.org/Berlin.html
12. http://dbpedia.org/Berlin.rdf

We will often need to use some kind of primary key inside our URIs, to make sure that each one is unique. If we can, we use a key that is meaningful inside our domain. For example, when dealing with books, making the ISBN number part of the URI is better than using the primary key of an internal database table. This also makes equivalence mining to derive RDF links easier.

Examples of cool URIs:

- http://dbpedia.org/resource/Boston
- http://www4.wiwiss.fu-berlin.de/bookmashup/books/006251587X

# 4. WHICH VOCABULARIES SHOULD I USE TO REPRESENT INFORMATION?

In order to make it as easy as possible for client applications to process our data, we should reuse terms from well-known vocabularies wherever possible. We should only define new terms if we cannot find required terms in existing vocabularies.

## 4.1 Reusing existing terms

A set of well-known vocabularies has evolved in the Semantic Web community. We have to check whether our data can be represented using terms from these vocabularies before defining any new terms:

- Friend-of-a-Friend (FOAF), vocabulary for describing people.
- Dublin Core (DC) defines general metadata attributes.

- Semantically-Interlinked Online Communities (SIOC), vocabulary for representing online communities.
- Description of a Project (DOAP), vocabulary for describing projects.
- Simple Knowledge Organization System (SKOS), vocabulary for representing taxonomies and loosely structured knowledge.
- Music Ontology provides terms for describing artists, albums and tracks.
- Review Vocabulary, vocabulary for representing reviews.
- Creative Commons (CC), vocabulary for describing license terms.

It is common practice to mix terms from different vocabularies. We recommend the use of **rdfs:label** and **foaf:depiction** properties whenever possible as these terms are well-supported by client applications.

If we need URI references for geographic places, research areas, general topics, artists, books or CDs, we should consider using URIs from data sources within the **W3C SWEO Linking Open Data** community project, for instance Geonames, DBpedia, Musicbrainz, dbtune or the RDF Book Mashup. The two main benefits of using URIs from these data sources are:

1. The URIs are dereferenceable, meaning that a description of the concept can be retrieved from the Web. For instance, using the DBpedia URI http://dbpedia.org/page/Doom to identify the computer game Doom gives you an extensive description of the game including abstracts in 10 different languages and various classifications.
2. The URIs are already linked to URIs from other data sources. For instance, we can navigate from the DBPedia URI http://dbpedia.org/resource/Berlin to data about Berlin provides by Geonames and EuroStat. Therefor, by using concept URIs from these datasets, we interlink our data with a rich and fast-growing network of other data sources.

## 4.2 How to define terms?

When we cannot find good existing vocabularies that cover all the classes and properties we need, then we have to define our own terms. Defining new terms is not hard. RDF classes and properties are resources themselves, identified by URIs, and published on the Web, so everything we said about publishing Linked Data applies to them as well.

We can define vocabularies using the RDF Vocabulary Description Language 1.0: RDF Schema or the Web Ontology Language (OWL)

Here we give some guidelines for those who are familiar with these languages:

1. **Do not define new vocabularies from scratch,** but complement existing vocabularies with additional terms (in your own namespace) to represent our data as required.
2. **Provide for both humans and machines.** At this stage in the development of the Web of Data, more people will be coming across your code than machines, even though the Web of Data is meant for machines in the first instance. We cannot forget to add prose, e.g. rdfs:comments for each term invented. Always provide a label for each term using the rdfs:label property.
3. **Make term URIs dereferenceable.** It is essential that term URIs are dereferenceable so that clients can look up the definition of a term. Therefore we should make term URIs dereferenceable following the W3C Best Practice Recipes for Publishing RDF Vocabularies.
4. **Make use of other people's terms**. Using other people's terms, or providing mappings to them, helps to promote the level of data interchange on the Web of Data, in the same way that hypertext links built the traditional document Web.
5. **State all important information explicitly.** For example, state all ranges and domains explicitly. Remember: humans can often do guesswork, but machines can't. Don't leave important information out!

6. **Do not create over-constrained, brittle models, leave some flexibility for growth.** For instance, if we use full-featured OWL to define our vocabulary, we might state things that lead to unintended consequences and inconsistencies when somebody else references our term in a different vocabulary definition. Therefore, unless we know exactly what we are doing, use RDF-Schema to define vocabularies.

The following example contains a definition of a class and a property following the rules above. The example uses the Turtle syntax. Namespace declarations are omitted.

**Definition of the class "Lover"**

```
<http://sites.wiwiss.fu-
berlin.de/suhl/bizer/pub/LoveVocabulary#Lover>

    rdf:type rdfs:Class ;

    rdfs:label "Lover"@en ;

    rdfs:label "Liebender"@de ;

    rdfs:comment "A person who loves somebody."@en ;

    rdfs:comment "Eine Person die Jemanden liebt."@de ;

    rdfs:subClassOf foaf:Person .
```

**Definition of the property "loves"**

```
<http://sites.wiwiss.fu-
berlin.de/suhl/bizer/pub/LoveVocabublary#loves>

    rdf:type rdf:Property ;

    rdfs:label "loves"@en ;

    rdfs:label "liebt"@de ;
```

```
    rdfs:comment "Relation between a lover and a loved person."@en ;

    rdfs:comment "Beziehung zwischen einem Liebenden und einer geliebten Person."@de ;

    rdfs:subPropertyOf foaf:knows ;

    rdfs:domain <http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LoveVocabulary#Lover> ;

    rdfs:range foaf:Person .
```

## 5. WHAT SHOULD I RETURN AS RDF DESCRIPTION FOR A URI

So, assuming we have expressed all our data in RDF triples: What triples should go into the RDF representation that is returned (after a 303 redirect) in response to dereferencing a URI identifying a non-information resource?

1. **The description:** The representation should include all triples from our dataset that have the resource's URI as the subject. This is the immediate description of the resource.
2. **Backlinks:** The representation should also include all triples from our dataset that have the resource's URI as the object. This is redundant, as these triples can already be retrieved from their subject URIs, but it allows browsers and crawlers to traverse links in either direction.
3. **Related descriptions**: We may include any additional information about related resources that may be of interest in typical usage scenarios. For example, we may want to send information about the author along with information about a book, because many clients interested in the book may also be interested in the author. A moderate approach is recommended, returning a megabyte of RDF will be considered excessive in most cases.

4. **Metadata:** The representation should contain any metadata we want to attach to our published data, such as a URI identifying the author and licensing information. These should be recorded as RDF descriptions of the *information resource* that describes a non-information resource, that is, the subject of the RDF triples should be the URI of the information resource. Attaching meta-information to that information resource, rather than attaching it to the described resource itself or to specific RDF statements about the resource (as with RDF reification) plays nicely together with using Named Graphs and the SPARQL query language in Linked Data client applications. In order to enable information consumers to use our data under clear legal terms, each RDF document should contain a license under which the content can be used.
5. **Syntax:** There are various ways to serialize RDF descriptions. Our data source should at least provide RDF descriptions as RDF/XML which is the only official syntax for RDF, as RDF/XML is not very human-readable.

## 5.1 Authoritative Description

In the following, we give two examples of RDF descriptions following the rules above. The first example covers the case of an authoritative representation served by a URI owner. The second example covers the case of non-authoritative information served by somebody who is not the owner of the described URI.

**Metadata and Licensing Information**

```
<http://dbpedia.org/data/Alec_Empire>

    rdfs:label "RDF description of Alec Empire" ;
```

```
    rdf:type foaf:Document ;

    dc:publisher <http://dbpedia.org/resource/DBpedia> ;

    dc:date "2007-07-13"^^xsd:date ;

    dc:rights

        <http://en.wikipedia.org/wiki/WP:GFDL> .
```

### The description

```
<http://dbpedia.org/resource/Alec_Empire>

    foaf:name "Empire, Alec" ;

    rdf:type foaf:Person ;

    rdf:type <http://dbpedia.org/class/yago/musician> ;

    rdfs:comment

        "Alec Empire (born May 2, 1972) is a German musician who is
..."@en ;

    rdfs:comment

        "Alec Empire (eigentlich Alexander Wilke) ist ein deutscher
Musiker. ..."@de ;

    dbpedia:genre <http://dbpedia.org/resource/Techno> ;

    dbpedia:associatedActs
<http://dbpedia.org/resource/Atari Teenage Riot> ;

    foaf:page <http://en.wikipedia.org/wiki/Alec_Empire> ;

    foaf:page <http://dbpedia.org/page/Alec_Empire> ;
```

```
    rdfs:isDefinedBy <http://dbpedia.org/data/Alec_Empire> ;

    owl:sameAs    <http://zitgist.com/music/artist/d71ba53b-23b0-4870-
a429-cce6f345763b> .
```

**Backlinks**

```
<http://dbpedia.org/resource/60_Second_Wipeout>

    dbpedia:producer <http://dbpedia.org/resource/Alec_Empire> .

<http://dbpedia.org/resource/Limited_Editions_1990-1994>

    dbpedia:artist <http://dbpedia.org/resource/Alec_Empire> .
```

Note that the description contains an owl:sameAs Link stating that http://dbpedia.org/resource/Alec_Empire and http://zitgist.com/music/artist/d71ba53b-23b0-4870-a429-cce6f345763b are URI aliases referring to the same non-information resource.

In order to make it easier for Linked Data clients to understand the relation between:

http://dbpedia.org/resource/Alec_Empire

or

http://dbpedia.org/data/Alec_Empire

and

http://dbpedia.org/page/Alec_Empire

The URIs can be interlinked using the rdfs: isDefinedBy and the foaf: page property.

## *5.2* Non-Authoritative Description

The following example shows the representation of the information resource:

http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial/ChrisAboutRichard

It is published by Chris to provide information about Richard. Richard owns the URI http://richard.cyganiak.de/foaf.rdf#cygri and is therefore the only person who can provide an authoritative description for this URI. Thus using **Web Architecture** terminology, Chris is providing non-authoritative information about Richard.

**Metadata and Licensing Information**

```
<>

    rdf:type foaf:Document ;

    dc:author <http://www.bizer.de#chris> ;

    dc:date "2007-07-13"^^xsd:date ;

    cc:license <http://web.resource.org/cc/PublicDomain> .
```

**The description**

```
<http://richard.cyganiak.de/foaf.rdf#cygri>

    foaf:name "Richard Cyganiak" ;

    foaf:topic_interest
<http://dbpedia.org/resource/Category:Databases> ;
```

```
    foaf:topic_interest <http://dbpedia.org/resource/MacBook_Pro> ;

    rdfs:isDefinedBy <http://richard.cyganiak.de/foaf.rdf> ;

    rdf:seeAlso <> .
```

**Backlinks**

```
<http://www.bizer.de#chris>

    foaf:knows <http://richard.cyganiak.de/foaf.rdf#cygri> .

<http://www4.wiwiss.fu-berlin.de/is-group/resource/projects/Project3>

    doap:developer <http://richard.cyganiak.de/foaf.rdf#cygri>
```

# 6. HOW TO SET RDF LINKS TO OTHER DATA SOURCES

RDF links enable Linked Data browsers and crawlers to navigate between data sources and to discover additional data.

The application domain will determine which RDF properties are used as predicates. For instance, commonly used linking properties in the domain of describing people are :

- Foaf : based
- Foaf : based near
- Foaf : topic_interest

It is common practice to use the owl:sameAs property for stating that another data source also provides information about a specific non-information resource. An owl:sameAs link indicates that two URI references actually refer

to the same thing. Therefore, owl:sameAs is used to map between different URI aliases.

## 6.1 Setting RDF Links Manually

Before you can set RDF links manually, you need to know something about the datasets you want to link to. In order to get an overview of different datasets that can be used as linking targets please refer to the Linking Open Data Dataset list. Once you have identified particular datasets as suitable linking targets, you can manually search in these for the URI references you want to link to. If a data source doesn't provide a search interface, for instance a SPARQL endpoint or a HTML Web form, you can use Linked Data browsers like Tabulator or Disco to explore the dataset and find the right URIs.

You can also use services such as Uriqr or Sindice to search for existing URIs and to choose the most popular one if you find several candidates. Uriqr allows you to find URIs for people you know, simply by searching for their name. Results are ranked according to how heavily a particular URI is referenced in RDF documents on the Web, but you will need to apply a little bit of human intelligence in picking the most appropriate URI to use. Sindice indexes the Semantic Web and can tell you which sources mention a certain URI. Therefore the service can help you to choose the most popular URI for a concept.

Remember that data sources might use HTTP-303 redirects to redirect clients from URIs identifying non-information resources to URIs identifying information resources that describe the non-information resources. In this case, make sure that you link to the URI reference identifying the non-information resource, and not the document about it.

## 6.2 Auto-generating RDF Links

The approach described above does not scale to large datasets, for instance interlinking 70,000 places in DBpedia to their corresponding entries in Geonames. In such cases it makes sense to use an automated record linkage algorithm to generate RDF links between data sources.

Record Linkage is a well-known problem in the databases community. The Linking Open Data Project collects material related to using record linkage algorithms in the Linked Data context on the Equivalence Mining wiki page.

There is still a lack of good, easy-to-use tools to auto-generate RDF links. Therefore it is common practice to implement dataset-specific record linkage algorithms to generate RDF links between data sources. In the following we describe two classes of such algorithms:

### Pattern-based Algorithms

In various domains, there are generally accepted naming schemata. For instance, in the publication domain there are **ISBN** numbers, in the financial domain there are **ISIN** identifiers. If these identifiers are used as part of HTTP URIs identifying particular resources, it is possible to use simple pattern-based algorithms to generate RDF links between these resources.

An example of a data source using ISBN numbers as part of its URIs is the RDF Book Mashup. It uses the URI

http://www4.wiwiss.fu-berlin.d/bookmashup/books/0747581088

to identify the book 'Harry Potter and the Half-blood Prince'. Having the ISBN number in these URIs made it easy for DBpedia to generate owl:sameAs links between books within DBpedia and the Book Mashup. DBpedia uses the following pattern-based algorithm:

Having the ISBN number in these URIs made it easy for DBpedia to generate owl:sameAs links between books within DBpedia and the Book Mashup. DBpedia uses the following pattern-based algorithm:

1. Iterate over all books in DBpedia that have an ISBN number.
2. Create a owl:sameAs link between the URI of a book in DBpedia and the corresponding Book Mashup URI using the following pattern for Book Mashup URIs:

   http://www4.wiwiss.fu-berlin.de/bookmashup/books/**{ISBN number}**.

Running this algorithm against all books in DBpedia resulted in 9000 RDF links which were merged with the DBpedia dataset. For instance, the resulting link for the Harry Potter book is:

```
<http://dbpedia.org/resource/Harry_Potter_and_the_Half-Blood_Prince>

    owl:sameAs                                  <http://www4.wiwiss.fu-
berlin.de/bookmashup/books/0747581088>
```

## More complex property-based Algorithms

In cases where no common identifiers across datasets exist, it is necessary to employ more complex property-based linkage algorithms. We outline an algorithm below:

1. **Interlinking DBpedia and Geonames.** Information about geographic places appear in the Geonames database as well as in DBpedia. In order to identify places that appear in both datasets, the Geonames team uses a property-based heuristic that is based on article title together with semantic information like latitude and longitude, but also country, administrative division, feature type, population and categories. Running this heuristic against both data sources resulted in 70500 correspondences which were merged as Geonames

owl:sameAs links with the DBpedia dataset as well as with the Geonames dataset.

## 7. Recipes for Serving Information as Linked Data

This chapter provides practical recipes for publishing different types of information as Linked Data on the Web. Information has to fulfill the following minimal requirements to be considered "published as Linked Data on the Web":

- Things must be identified with dereferenceable HTTP URIs.
- If such a URI is dereferenced asking for the MIME-type application/rdf+xml, a data source must return an RDF/XML description of the identified resource.
- URIs that identify non-information resources must be set up in one of these ways: Either the data source must return an HTTP response containing an *HTTP 303 redirect* to an information resource describing the non-information resource, as discussed earlier in this document. Or the URI for the non-information resource must be formed by taking the URI of the related information resource and appending a fragment identifier.
- Besides RDF links to resources within the same data source, RDF descriptions should also contain RDF links to resources provided by other data sources, so that clients can navigate the Web of Data as a whole by following RDF links.

Which of the following recipes fits your needs depends on various factors, such as:

- **How much data do you want to serve?** If you only want to publish several hundred RDF triples, you might want to serve them as a static

RDF files using Recipe 7.1. If your dataset is larger, you might want to load it into a proper RDF store and put the Pubby Linked Data interface in front of it as described in Recipe 7.3.

- **How is your data currently stored?** If your information is stored in a relational database, you can use D2R Server as described in Recipe 7.2. If the information is available through an API, you might implement a wrapper around this API as described in Recipe 7.4. If your information is represented in some other format such as Microsoft Excel, CSV or BibTeX, you will have to convert it to RDF first as described in Recipe 7.3.
- **How often does your data change?** If your data changes frequently, you might prefer approaches which generate RDF views on your data, such as D2R Server (Recipe 7.2), or wrappers (Recipe 7.4).

After you have published your information as Linked Data, you should ensure that there are external RDF links pointing at URIs from your dataset, so that RDF browser and crawlers can find your data. There are two basic ways of doing this:

1. Add several RDF links to your FOAF profile that point at URIs identifying central resources within your dataset. Assuming that somebody else in the world knows you and references your FOAF profile, your new dataset is now reachable by following RDF links.
2. Convince the owners of related data sources to auto-generate RDF links to URIs from your dataset. Or to make it easier for the owner of the other dataset, create the RDF links yourself and send them to her so that she just has to merge them with her dataset. A project that is extremely open to setting RDF links to other data sources is the DBpedia community project. Just announce your data source on the DBpedia mailing list or send a set of RDF links to the list.

## What is the Pubby Linked Data Interface?

Many triple stores and other SPARQL endpoints can be accessed only by SPARQL client applications that use the SPARQL protocol. It cannot be accessed by the growing variety of Linked Data clients. **Pubby is designed to provide a Linked Data interface** to those RDF data sources [Figure 6].

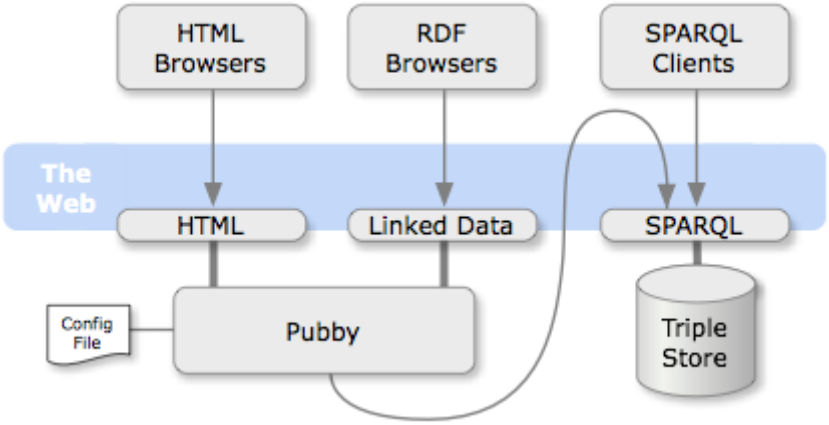



Figure 6 “Pubby”

**In RDF, resources are identified by URIs.** The URIs used in most SPARQL dataset are **not dereferenceable**, meaning they cannot be accessed in a Semantic Web browser, but return 404 Not Found errors instead, or use non-dereferenceable URI schemes, as in the fictional URI tag dbpedia.org, 2007:Berlin.

When setting up a Pubby server for a SPARQL endpoint, you will **configure a mapping** that translates those URIs to dereferenceable URIs handled by Pubby. If your server is running at

http://myserver.org:8080/pubby/

Then the Berlin URI above might be mapped to

http://myserver.org:8080/pubby/Berlin.

Pubby will **handle requests to the mapped URIs** by connecting to the SPARQL endpoint, asking it for information about the original URI, and passing back the results to the client. It also handles various **details of the**

**HTTP interaction**, such as the 303 redirect required by Web Architecture, and content negotiation between HTML, RDF/XML and Turtle descriptions of the same resource.

## 7.1 Serving Static RDF Files

The simplest way to serve Linked Data is to produce static RDF files, and upload them to a web server. This approach is typically chosen in situations where:

- The RDF files are created manually, e.g. when publishing personal FOAF files or RDF vocabularies or
- The RDF files are generated or exported by some piece of software that only outputs to files.

### Configuring the server for correct MIME types

Older web servers are sometimes not yet configured to return the correct MIME type when serving RDF/XML files. Linked Data browsers may not recognize RDF data served in this way because the server claims that it is not RDF/XML but plain text.

How to fix this depends on the web server. In the case of Apache, add this line to the httpd.conf configuration file, or to an .htaccess file in the web server's directory where the RDF files are placed:

```
AddType application/rdf+xml .rdf
```

This tells Apache to serve files with an .rdf extension using the correct MIME type for RDF/XML, application/rdf+xml. Note this means you have to name your files with the.rdf extension.

While you're at it, you can also add these lines to make your web server ready for other RDF syntaxes (N3 and Turtle):

```
AddType text/rdf+n3;charset=utf-8 .n3

AddType application/x-turtle .ttl
```

## File size

On the document Web, it's considered bad form to publish huge HTML pages, because they load very slowly in browsers and consume unnecessary bandwidth. The same is true when publishing Linked Data: Your RDF files shouldn't be larger than, say, a few hundred kilobytes. If your files are larger and describe multiple resources, you should break them up into several RDF files, or use Pubby as described in recipe 7.3 to serve them in chunks.

When you serve multiple RDF files, make sure they are linked to each other through RDF triples that involve resources described in different files.

## Choosing URIs for non-information resources

The static file approach doesn't support the 303 redirects required for the URIs of non-information resources. Fortunately there is another standards-compliant method of naming non-information resources, which works very well with static RDF files, but has a downside we will discuss later. This method relies on *hash URIs*.

When you serve a static RDF file at, say, http://example.com/people.rdf, then you should name the non-information resources described in the file by appending a *fragment identifier* to the file's URI. The identifier must be unique within the file. That way, you end up with URIs like this for your non-information resources:

- http://example.com/people.rdf#alice
- http://example.com/people.rdf#bob

This works because HTTP clients dereference hash URIs by stripping off the part after the hash and dereferencing the resulting URI. A Linked Data browser will then look into the response (the RDF file in this case), and find triples that tell it more about the non-information resource, achieving an effect quite similar to the 303 redirect.

There's a reference to a specific representation format in the identifiers (the .rdf extension). And if you choose to rename the RDF file later on, or decide to split your data into several files, then all identifiers will change and existing links to them will break.

That's why you should use this approach only if the overall structure and size of the dataset are unlikely to change much in the future, or as a quick-and-dirty solution for transient data where link stability isn't so important.

## Extending the recipe for 303 redirects and content negotiation

This approach can be extended to use 303 redirects and even to support content negotiation, if you are willing to go through some extra hoops. Unfortunately this process is dependent on your web server and its configuration. The W3C has published several recipes that show how to do this for the Apache web server: Best Practice Recipes for Publishing RDF Vocabularies. The document is officially targeted at publishers of RDF vocabularies, but the recipes work for other kinds of RDF data served from static files. Note that at the time of writing there is still an issue with content negotiation in this document which might be solved by moving from Apache mod_rewrite to mod_negotiation.

## 7.2 Serving Relational Databases

If your data is stored in a relational database it is usually a good idea to leave it there and just publish a Linked Data view on your existing database.

A tool for serving Linked Data views on relational databases is D2R Server. D2R server relies on a declarative mapping between the schemata of the database and the target RDF terms. Based on this mapping, D2R Server serves a Linked Data view on your database and provides a SPARQL endpoint for the database [Figure 7].

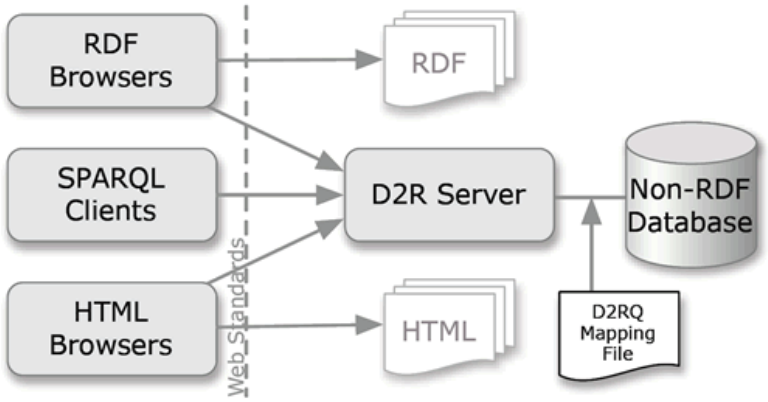



Figure 7 “Structure of DR2 Server”

There are several D2R Servers online, for example Berlin DBLP Bibliography Server, Hannover DBLP Bibliography Server, Web-based Systems @ FU Berlin Group Server or the EuroStat Countries and Regions Server.

Publishing a relational database as Linked Data typically involves the following steps:

1. Download and install the server software as described in the Quick Start section of the D2R Server homepage.

2. Have D2R Server auto-generate a D2RQ mapping from the schema of your database (see Quick Start).
3. Customize the mapping by replacing auto-generated terms with terms from well-known and publicly accessible RDF vocabularies.
4. Add your new data source to the ESW Wiki datasets list in the category Linked Data and SPARQL endpoint list and set several RDF links from your FOAF profile to the URIs of central resources within your new data source so that crawlers can discover your data.

Alternatively, you can also use:

1. OpenLink Virtuoso to publish your relational database as Linked Data.
    - Virtuoso RDF Views – Getting Started Guide on how to map your relational database to RDF and
    - Deploying Linked Data on how to get URI dereferencing and content negotiation into place.
2. Triplify, a small plugin for Web applications, which reveals the semantic structures encoded in relational databases by making database content available as RDF, JSON or Linked Data.

## 7.3 Serving Other Types of Information

If your information is currently represented in formats such as CSV, Microsoft Excel, or BibTEX and you want to serve the information as Linked Data on the Web it is usually a good idea to do the following:

- Convert your data into RDF using an RDFizing tool. There are two locations where such tools are listed: ConverterToRdf maintained in the ESW Wiki, and RDFizers maintained by the SIMILE team.
- After conversion, store your data in a RDF repository. A list of RDF repositories is maintained in the ESW Wiki.
- Ideally the chosen RDF repository should come with a Linked Data interface which takes care of making your data Web accessible. As many RDF repositories have not implemented Linked Data interfaces yet, you can also choose a repository that provides a SPARQL endpoint and put Pubby as a Linked Data interface in front of your SPARQL endpoint.

The approach described above is taken by the DBpedia project, among others. The project uses PHP scripts to extract structured data from Wikipedia pages. This data is then converted to RDF and stored in a OpenLink Virtuoso repository which provides a SPARQL endpoint. In order to get a Linked Data view, Pubby is put in front of the SPARQL endpoint.

If your dataset is sufficiently small to fit completely into the web server's main memory, then you can do without the RDF repository, and instead use Pubby's conf:loadRDF option to load the RDF data from an RDF file directly into Pubby. This might be simpler, but unlike a real RDF repository, Pubby will keep everything in main memory and doesn't offer a SPARQL endpoint.

## 7.4 Implementing Wrappers around existing Applications or Web APIs

Large numbers of Web applications have started to make their data available on the Web through Web APIs. Examples of data sources providing such APIs include eBay, Amazon, Yahoo, Google and Google Base. A more comprehensive API list is found at Programmable Web. Different APIs provide

diverse query and retrieval interfaces and return results using a number of different formats such as XML, JSON or ATOM. This leads to three general limitations of Web APIs:

- their content cannot be crawled by search engines
- Web APIs cannot be accessed using generic data browsers
- Mashups are implemented against a fixed number of data sources and cannot take advantage of new data sources that appear on the Web.

These limitations can be overcome by implementing Linked Data wrappers around APIs. In general, Linked Data wrappers do the following:

1. They assign HTTP URIs to the non-information resources about which the API provides data.
2. When one of these URIs is dereferenced asking for application/rdf+xml, the wrapper rewrites the client's request into a request against the underlying API.
3. The results of the API request are transformed to RDF and sent back to the client.

Examples of Linked Data Wrappers include:

## The RDF Book Mashup

The RDF Book Mashup makes information about books, their authors, reviews, and online bookstores available as RDF on the Web. The RDF Book Mashup assigns a HTTP URI to each book that has an ISBN number. Whenever one of these URIs is dereferenced, the Book Mashup requests data about the book, its author as well as reviews and sales offers from

the Amazon API and the Google Base API. This data is then transformed into RDF and returned to the client [Figure 8].

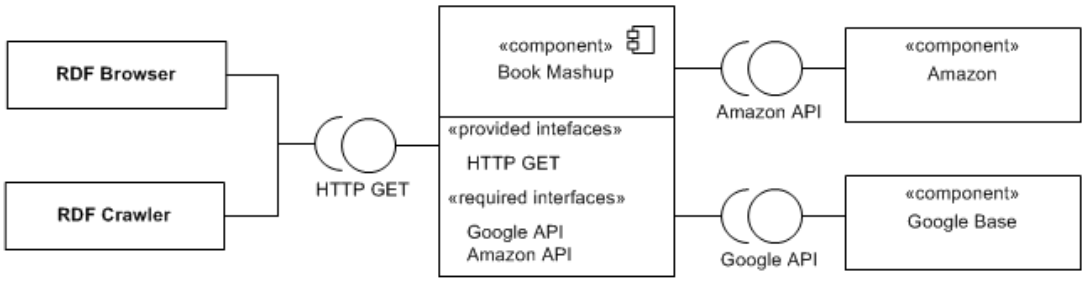



Figure 8 “RDF Book Mashup”

The RDF Book Mashup is implemented as a small PHP script which can be used as a template for implementing similar wrappers around other Web APIs. More information about the Book Mashup and the relationship of Web APIs to Linked Data in general is available in The RDF Book Mashup.

## 8. SPARQL LANGUAGE

SPARQL (pronounced “sparkle”, a recursive acronym for SPARQL Protocol and RDF Query Language) is an RDF query language, that is, a query language for databases, able to retrieve and manipulate data stored in Resource Description Framework format. It was made a standard by the RDF Data Access Working Group (DAWG) of the World Wide Web Consurtium, and considered as one of the key technologies of Semantic web. On 15 January 2008, SPARQL 1.0 became an official W3C Recommendation.

SPARQL allows for a query to consist of triple patters, conjunctions, disjunctions, and optional patters.

Implementations for multiple programming languages exist. There exist tools that allow one to connect and semi-automatically construct a SPARQL query for a SPARQL query for a SPARQL endpoint, for example ViziQuer. In addition, there exist tools that translate SPARQL queries to other languages, for example to SQL and XQuery.

## 8.1 SPARQL in General

- SPARQL Protocol and RDF Query Language
- SPARQL Query Language for RDF
  - Declarative
  - Based on the RDF data model (triple/graph)
  - Our focus
- SPARQL Query Results XML Format
  - Representation on the results of SPARQL queries
- SPARQL Protocol for RDF
  - Transmission of SPARQL queries and the results
- SPARQL endpoint: Web Service that implements the protocol

### Terminology

The following basic terms are defined in RDF Concepts and used in SPARQL:

- IRI (corresponds to the Concepts and Abstract Syntax RDF URI ref)
- Literal
- Lexical Form
- Plain Literal
- Language Tag
- Typed Literal
- Datatype IRI
- Blank node

## 8.2 Making Simple Queries

Most forms of SPARQL query contain a set of triple patterns called a *basic graph pattern*. Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable. A basic graph pattern *matches* a subgraph of the RDF data when RDF terms from that

subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph.

### Basic Query Forms

The SPARQL language specifies four different query variations for different purposes.

- **SELECT query**
  Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.
- **CONSTRUCT query**
  Used to extract information from the SPARQL endpoint and transform the results into valid RDF.
- **ASK query**
  Used to provide a simple True/False result for a query on a SPARQL endpoint.

- **DESCRIBE query**
  Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Each of these query forms takes a WHERE block to restrict the query although in the case of the DESCRIBE query the WHERE is optional.

## 8.2.1 Writing a Simple Query

The example below shows a SPARQL query to find the title of a book from the given data graph. The query consists of two parts, the SELECT clause identifies the variables to appear in the query results, and the WHERE clause provides the basic graph pattern to match against the data graph. The basic graph pattern in this example consists of a single triple pattern with a single variable (? title) in the object position.

**Data:**

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial".
```

**Query:**

```
SELECT ?title

WHERE

{

  <http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> ?title .

}
```

This query, on the data above, has one solution:

**Query Result:**

| title |
| --- |
| "SPARQL Tutorial" |

## 8.2.2 Multiple Matches

The result of a query is a solution sequence, corresponding to the ways in which the query's graph pattern matches the data. There may be zero, one or multiple solutions to a query.

**Data:**

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .



_:a  foaf:name   "Johnny Lee Outlaw" .
```

```
_:a  foaf:mbox   <mailto:jlow@example.com> .

_:b  foaf:name   "Peter Goodguy" .

_:b  foaf:mbox   <mailto:peter@example.org> .

_:c  foaf:mbox   <mailto:carol@example.org> .
```

**Query:**

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?name ?mbox

WHERE

  { ?x foaf:name ?name .

    ?x foaf:mbox ?mbox }
```

**Query Result:**

| name | mbox |
|---|---|
| "Johnny Lee Outlaw" | <mailto:jlow@example.com> |
| "Peter Goodguy" | <mailto:peter@example.org> |

Each solution gives one way in which the selected variables can be bound to RDF terms so that the query pattern matches the data. The result set gives all the possible solutions. In the above example, the following two subsets of the data provided the two matches.

## Multiple Optional Graph Patterns

Graph patterns are defined recursively. A graph pattern may have zero or more optional graph patterns, and any part of a query pattern may have an optional part. In this example, there are two optional graph patterns.

Data:

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
```

```
_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x foaf:mbox ?mbox } .
         OPTIONAL { ?x foaf:homepage ?hpage }
       }
```

Query result:

| name | mbox | hpage |
|---|---|---|
| "Alice" | | <http://work.example.org/alice/> |
| "Bob" | <mailto:bob@work.example> | |

### 8.2.3 Matching Literals with Numeric Types

Integers in a SPARQL query indicate an RDF typed literal with the datatype xsd:integer.

For example: 42 is a shortened form of "42"^^

<http://www.w3.org/2001/XMLSchema#integer>.

The pattern in the following query has a solution with variable v bound to :y.

```
SELECT ?v WHERE { ?v ?p 42 }
```

| v |
| --- |
| <http://example.org/ns#y> |

### 8.2.4 Blank Node Labels in Query Results

Query results can contain blank nodes. Blank nodes in the example result sets in this document are written in the form "_:" followed by a blank node label.

Blank node labels are scoped to a result or, for the CONSTRUCT query form, the result graph. Use of the same label within a result set indicates the same blank node.

Data:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .


_:a  foaf:name   "Alice" .

_:b  foaf:name   "Bob" .
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?x ?name

WHERE  { ?x foaf:name ?name }
```

| x | name |
| --- | --- |
| _:c | "Alice" |
| _:d | "Bob" |

The results above could equally be given with different blank node labels because the labels in the results only indicate whether RDF terms in the solutions are the same or different.

| x | name |
|---|---|
| _:r | "Alice" |
| _:s | "Bob" |

These two results have the same information: the blank nodes used to match the query are different in the two solutions. There need not be any relation between a label _:a in the result set and a blank node in the data graph with the same label.

An application writer should not expect blank node labels in a query to refer to a particular blank node in the data.

## Matching Alternatives

SPARQL provides a means of combining graph patterns so that one of several alternative graph patterns may match. If more than one of the alternatives matches, all the possible pattern solutions are found.

Pattern alternatives are syntactically specified with the UNION keyword.

Data:

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .

@prefix dc11:  <http://purl.org/dc/elements/1.1/> .


_:a  dc10:title     "SPARQL Query Language Tutorial" .

_:a  dc10:creator   "Alice" .


_:b  dc11:title     "SPARQL Protocol Tutorial" .

_:b  dc11:creator   "Bob" .


_:c  dc10:title     "SPARQL" .

_:c  dc11:title     "SPARQL (updated)" .
```

Query:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>

PREFIX dc11:  <http://purl.org/dc/elements/1.1/>


SELECT ?title

WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title  ?title } }
```

Query result:

| title |
|---|
| "SPARQL Protocol Tutorial" |
| "SPARQL" |
| "SPARQL (updated)" |
| "SPARQL Query Language Tutorial" |

This query finds titles of the books in the data.To determine exactly how the information was recorded, a query could use different variables for the two alternatives:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>

PREFIX dc11:  <http://purl.org/dc/elements/1.1/>


SELECT ?x ?y

WHERE  { { ?book dc10:title ?x } UNION { ?book dc11:title  ?y } }
```

| x | y |
|---|---|
| | "SPARQL (updated)" |

| | |
|---|---|
| | "SPARQL Protocol Tutorial" |
| "SPARQL" | |
| "SPARQL Query Language Tutorial" | |

This will return results with the variable x bound for solutions from the left branch of the UNION, and y bound for the solutions from the right branch. If neither part of the UNION pattern matched, then the graph pattern would not match.

The UNION pattern combines graph patterns, each alternative possibility can contain more than one triple pattern:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>

PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title ?author

WHERE  { { ?book dc10:title ?title .  ?book dc10:creator ?author }

         UNION

         { ?book dc11:title ?title .  ?book dc11:creator ?author }

       }
```

| title | author |
|---|---|
| "SPARQL Query Language Tutorial" | "Alice" |
| "SPARQL Protocol Tutorial" | "Bob" |

This query will only match a book if it has both a title and creator predicate from the same version of Dublin Core.

## 8.3 Query Forms

### 8.3.1 SELECT

The SELECT form of results returns variables and their bindings directly. It combines the operations of projecting the required variables with introducing new variable bindings into a query solution.

### *Projection*

Specific variables and their bindings are returned when a list of variable names is given in the SELECT clause. The syntax SELECT * is an abbreviation that selects all of the variables that are in-scope at that point in the query. It excludes variables only used in FILTER, in the right-hand side of MINUS, and takes account of subqueries.

Use of SELECT * is only permitted when the query does not have a GROUP BY clause.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .


_:a    foaf:name   "Alice" .

_:a    foaf:knows  _:b .

_:a    foaf:knows  _:c .


_:b    foaf:name   "Bob" .


_:c    foaf:name   "Clare" .

_:c    foaf:nick   "CT" .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?nameX ?nameY ?nickY

WHERE

  { ?x foaf:knows ?y ;

       foaf:name ?nameX .

    ?y foaf:name ?nameY .

    OPTIONAL { ?y foaf:nick ?nickY }

  }
```

| nameX | nameY | nickY |
|---|---|---|
| "Alice" | "Bob" | |
| "Alice" | "Clare" | "CT" |

Result sets can be accessed by a local API but also can be serialized into either JSON, XML, CSV or TSV.

**SPARQL 1.1 Query Results JSON Format:**

```
{
  "head": {
    "vars": [ "nameX" , "nameY" , "nickY" ]
  } ,
  "results": {
    "bindings": [
      {
        "nameX": { "type": "literal" , "value": "Alice" } ,
        "nameY": { "type": "literal" , "value": "Bob" }
      } ,
      {
        "nameX": { "type": "literal" , "value": "Alice" } ,
        "nameY": { "type": "literal" , "value": "Clare" } ,
        "nickY": { "type": "literal" , "value": "CT" }
      }
    ]
  }
}
```

**SPARQL Query Results XML Format:**

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="nameX"/>
    <variable name="nameY"/>
    <variable name="nickY"/>
  </head>
  <results>
    <result>
      <binding name="nameX">
        <literal>Alice</literal>
      </binding>
      <binding name="nameY">
        <literal>Bob</literal>
```

```
    </binding>
  </result>
   <result>
    <binding name="nameX">
      <literal>Alice</literal>
    </binding>
    <binding name="nameY">
      <literal>Clare</literal>
    </binding>
    <binding name="nickY">
      <literal>CT</literal>
    </binding>
   </result>
  </results>
</sparql>
```

***SELECT Expressions***

As well as choosing which variables from the pattern matching are included in the results, the SELECT clause can also introduce new variables. The rules of assignment in SELECT expression are the same as for assignment in BIND. The expression combines variable bindings already in the query solution, or defined earlier in the SELECT clause, to produce a binding in the query solution.

The scoping for (expr AS v) applies immediately. In SELECT expressions, the variable may be used in an expression later in the same SELECT clause and may not be be assigned again in the same SELECT clause.

Example:

Data:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .

@prefix :     <http://example.org/book/> .

@prefix ns:   <http://example.org/ns#> .



:book1  dc:title  "SPARQL Tutorial" .

:book1  ns:price  42 .

:book1  ns:discount 0.2 .
```

```
:book2  dc:title  "The Semantic Web" .

:book2  ns:price  23 .

:book2  ns:discount 0.25 .
```

Query:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>

PREFIX  ns:  <http://example.org/ns#>

SELECT  ?title (?p*(1-?discount) AS ?price)

{ ?x ns:price ?p .

  ?x dc:title ?title .

  ?x ns:discount ?discount

}
```

Results:

| title | price |
|---|---|
| "The Semantic Web" | 17.25 |
| "SPARQL Tutorial" | 33.6 |

New variables can also be used in expressions if they are introduced earlier, syntactically, in the same SELECT clause:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>

PREFIX  ns:  <http://example.org/ns#>

SELECT   ?title (?p AS ?fullPrice) (?fullPrice*(1-?discount) AS
?customerPrice)

{ ?x ns:price ?p .

   ?x dc:title ?title .

   ?x ns:discount ?discount

}
```

Results:

| title | fullPrice | customerPrice |
|---|---|---|
| "The Semantic Web" | 23 | 17.25 |
| "SPARQL Tutorial" | 42 | 33.6 |

### 8.3.2 CONSTRUCT

The CONSTRUCT query form returns a single RDF graph specified by a graph template. The result is an RDF graph formed by taking each query solution in the solution sequence, substituting for the variables in the graph template, and combining the triples into a single RDF graph by set union.

If any such instantiation produces a triple containing an unbound variable or an illegal RDF construct, such as a literal in subject or predicate position, then that triple is not included in the output RDF graph. The graph template can contain triples with no variables (known as ground or explicit triples), and these also appear in the output RDF graph returned by the CONSTRUCT query form.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .


_:a    foaf:name   "Alice" .

_:a    foaf:mbox   <mailto:alice@example.org> .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT       { <http://example.org/person#Alice> vcard:FN
?name }

WHERE      { ?x foaf:name ?name }
```

creates vcard properties from the FOAF information:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://example.org/person#Alice> vcard:FN "Alice" .
```

***Templates with Blank Nodes***

A template can create an RDF graph containing blank nodes. The blank node labels are scoped to the template for each solution. If the same label occurs twice in a template, then there will be one blank node created for each query solution, but there will be different blank nodes for triples generated by different query solutions.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .


_:a    foaf:givenname   "Alice" .

_:a    foaf:family_name "Hacker" .


_:b    foaf:firstname   "Bob" .

_:b    foaf:surname     "Hacker" .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>


CONSTRUCT { ?x  vcard:N _:v .

          _:v vcard:givenName ?gname .

          _:v vcard:familyName ?fname }

WHERE

 {

   { ?x  foaf:firstname  ?gname  } UNION    { ?x  foaf:givenname
?gname } .

   { ?x  foaf:surname     ?fname  } UNION   { ?x  foaf:family_name
?fname } .

 }
```

creates vcard properties corresponding to the FOAF information:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:v1 vcard:N        _:x .
_:x vcard:givenName  "Alice" .
_:x vcard:familyName "Hacker" .

_:v2 vcard:N        _:z .
_:z vcard:givenName  "Bob" .
_:z vcard:familyName "Hacker" .
```

The use of variable x in the template, which in this example will be bound to blank nodes with labels _:a and _:b in the data, causes different blank node labels (_:v1 and _:v2) in the resulting RDF graph.

***Accessing Graphs in the RDF Dataset***

Using CONSTRUCT, it is possible to extract parts or the whole of graphs from the target RDF dataset. This first example returns the graph (if it is in the dataset) with IRI labelhttp://example.org/aGraph; otherwise, it returns an empty graph.

```
CONSTRUCT { ?s ?p ?o } WHERE { GRAPH <http://example.org/aGraph> { ?s ?p ?o } . }
```

The access to the graph can be conditional on other information. For example, if the default graph contains metadata about the named graphs in the dataset, then a query like the following one can extract one graph based on information about the named graph:

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>

PREFIX app: <http://example.org/ns#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>



CONSTRUCT { ?s ?p ?o } WHERE

 {

   GRAPH ?g { ?s ?p ?o } .

   ?g dc:publisher <http://www.w3.org/> .

   ?g dc:date ?date .
```

```
  FILTER ( app:customDate(?date) > "2005-02-28T00:00:00Z"^^xsd:dateTime ) .

 }
```

where app:customDate identifies an extension function to turn the date format into an xsd:dateTime RDF term.

***Solution Modifiers and CONSTRUCT***

The solution modifiers of a query affect the results of a CONSTRUCT query. In this example, the output graph from the CONSTRUCT template is formed from just two of the solutions from graph pattern matching. The query outputs a graph with the names of the people with the top two sites, rated by hits. The triples in the RDF graph are not ordered.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

@prefix site: <http://example.org/stats#> .



_:a foaf:name "Alice" .

_:a site:hits 2349 .



_:b foaf:name "Bob" .

_:b site:hits 105 .



_:c foaf:name "Eve" .

_:c site:hits 181 .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

PREFIX site: <http://example.org/stats#>



CONSTRUCT { [] foaf:name ?name }

WHERE

{ [] foaf:name ?name ;
```

```
    site:hits ?hits .

}

ORDER BY desc(?hits)

LIMIT 2
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:x foaf:name "Alice" .
_:y foaf:name "Eve" .
```

***CONSTRUCT WHERE***

A short form for the CONSTRUCT query form is provided for the case where the template and the pattern are the same and the pattern is just a basic graph pattern (no FILTERs and no complex graph patterns are allowed in the short form). The keyword WHERE is required in the short form.

The following two queries are the same; the first is a short form of the second.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT WHERE { ?x foaf:name ?name }

PREFIX foaf: <http://xmlns.com/foaf/0.1/>



CONSTRUCT { ?x foaf:name ?name }

WHERE

{ ?x foaf:name ?name }
```

## 8.3.3 ASK

Applications can use the ASK form to test whether or not a query pattern has a solution. No information is returned about the possible query solutions, just whether or not a solution exists.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
```

```
_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

ASK  { ?x foaf:name  "Alice" }
```

```
true
```

The **SPARQL Query Results XML Format** form of this result set gives:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head></head>
  <boolean>true</boolean>
</sparql>
```

On the same data, the following returns no match because Alice's mbox is not mentioned.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

ASK  { ?x foaf:name  "Alice" ;

         foaf:mbox  <mailto:alice@work.example> }
```

```
false
```

### 8.3.4 DESCRIBE (Informative)

The DESCRIBE form returns a single result RDF graph containing RDF data about resources. This data is not prescribed by a SPARQL query, where the query client would need to know the structure of the RDF in the data source, but, instead, is determined by the SPARQL query processor. The query pattern is used to create a result set. The DESCRIBE form takes each of the resources identified in a solution, together with any resources directly named by IRI, and assembles a single RDF graph by taking a "description" which can come from any information available including the target RDF Dataset. The

description is determined by the query service. The syntax DESCRIBE * is an abbreviation that describes all of the variables in a query.

***Explicit IRIs***

The DESCRIBE clause itself can take IRIs to identify the resources. The simplest DESCRIBE query is just an IRI in the DESCRIBE clause:

```
DESCRIBE <http://example.org/>
```

***Identifying Resources***

The resources to be described can also be taken from the bindings to a query variable in a result set. This enables description of resources whether they are identified by IRI or by blank node in the dataset:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

DESCRIBE ?x

WHERE    { ?x foaf:mbox <mailto:alice@org> }
```

The property foaf:mbox is defined as being an inverse functional property in the FOAF vocabulary. If treated as such, this query will return information about at most one person. If, however, the query pattern has multiple solutions, the RDF data for each is the union of all RDF graph descriptions.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

DESCRIBE ?x

WHERE    { ?x foaf:name "Alice" }
```

More than one IRI or variable can be given:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

DESCRIBE ?x ?y <http://example.org/>

WHERE    {?x foaf:knows ?y}
```

***Descriptions of Resources***

The RDF returned is determined by the information publisher. It may be information the service deems relevant to the resources being described. It

may include information about other resources: for example, the RDF data for a book may also include details about the author.

A simple query such as

```
PREFIX ent:  <http://org.example.com/employees#>

DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

might return a description of the employee and some other potentially useful details:

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg:  <http://org.example.com/employees#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#>

_:a     exOrg:employeeId    "1234" ;

       foaf:mbox_sha1sum     "bee135d3af1e418104bc42904596fe148e90f033"
;
       vcard:N
        [ vcard:Family       "Smith" ;
          vcard:Given        "John"  ] .

foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```

which includes the blank node closure for the vcard vocabulary vcard,N. Other possible mechanisms for deciding what information to return include Concise Bounded Descriptions [CBD].

For a vocabulary such as FOAF, where the resources are typically blank nodes, returning sufficient information to identify a node such as the InverseFunctionalPropertyfoaf:mbox_sha1sum as well as information like name and other details recorded would be appropriate. In the example, the match to the WHERE clause was returned, but this is not required.

## 8.4 Subqueries

Subqueries are a way to embed SPARQL queries within other queries, normally to achieve results which cannot otherwise be achieved, such as limiting the number of results from some sub-expression within the query.

Due to the bottom-up nature of SPARQL query evaluation, the subqueries are evaluated logically first, and the results are projected up to the outer query.

Note that only variables projected out of the subquery will be visible to the outer query.

## Example

Data:

```
@prefix : <http://people.example/> .


:alice :name "Alice", "Alice Foo", "A. Foo" .

:alice :knows :bob, :carol .

:bob :name "Bob", "Bob Bar", "B. Bar" .

:carol :name "Carol", "Carol Baz", "C. Baz" .
```

Return a name (the one with the lowest sort order) for all the people that know Alice and have a name.

Query:

```
PREFIX : <http://people.example/>

PREFIX : <http://people.example/>

SELECT ?y ?minName

WHERE {

  :alice :knows ?y .

  {

    SELECT ?y (MIN(?name) AS ?minName)

    WHERE {

      ?y :name ?name .

    } GROUP BY ?y
```

```
  }
}
```

Results:

| y | minName |
|---|---|
| :bob | "B. Bar" |
| :carol | "C. Baz" |

This result is achieved by first evaluating the inner query:

```
SELECT ?y (MIN(?name) AS ?minName)
WHERE {
  ?y :name ?name .
} GROUP BY ?y
```

This produces the following solution sequence:

| y | minName |
|---|---|
| :alice | "A. Foo" |
| :bob | "B. Bar" |
| :carol | "C. Baz" |

Which is joined with the results of the outer query:

| y |
|---|
| :bob |
| :carol |

## 8.5 Building RDF Graphs

The SELECT query form returns variable bindings. The CONSTRUCT query form returns an RDF graph. The graph is built based on a template which is used to generate RDF triples based on the results of matching the graph pattern of the query.

**Data:**

```
@prefix org:    <http://example.com/ns#> .


_:a  org:employeeName   "Alice" .
_:a  org:employeeId     12345 .


_:b  org:employeeName   "Bob" .
_:b  org:employeeId     67890 .
```

**Query:**

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>
CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

**Results:**

```
@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

**Which can be serialized in** RDF/XML **as:**

```
<rdf:RDF
```

```
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   >
  <rdf:Description>
   <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
   <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

## 8.6 SPARQL Filters

Graph pattern matching produces a solution sequence, where each solution has a set of bindings of variables to RDF terms. SPARQL FILTERs restrict solutions to those for which the filter expression evaluates to TRUE.

### Restricting Numeric Values

SPARQL FILTERS can restrict on arithmetic expressions.

**Query:**

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>

PREFIX  ns:  <http://example.org/ns#>

SELECT  ?title ?price

WHERE   { ?x ns:price ?price .

          FILTER (?price < 30.5)

          ?x dc:title ?title . }
```

**Query Result:**

| title | price |
|---|---|
| "The Semantic Web" | 23 |

By constraining the price variable, only:book2 matches the query because only :book2 has a price less than 30.5, as the filter condition requires.

## 8.7 SPARQL endpoint

A SPARQL **endpoint** is a conformant SPARQL protocol service as defined in the SPROT specification. A SPARQL endpoint enables users (human or other) to query a knowledge base via the SPARQL language. Results are typically returned in one or more machine-processable formats. Therefore, a SPARQL endpoint is mostly conceived as a machine-friendly interface towards a knowledge base. Both the formulation of the queries and the human-readable presentation of the results should typically be implemented by the calling software, and not be done manually by human users.

The term **endpoint** has a more general meaning. In the "normative definitions" section of the Web Services Description Requirements document we find the End Point (AKA Port), Definition:

An association between a fully-specified InterfaceBinding and a network address, specified by a URI, that may be used to communicate with an instance of a Web Service. An EndPoint indicates a specific location for accessing a Web Service using a specific protocol and data format (SPARQL, RDF and XML).

# 9. D2RQ PLATFORM

The D2RQ Platform is a system for accessing relational databases as virtual, read-only RDF graphs. It offers RDF-based access to the content of relational databases without having to replicate it into an RDF store. Using D2RQ you can:

- query a non-RDF database using SPARQL
- access the content of the database as Linked Data over the Web
- create custom dumps of the database in RDF formats for loading into an RDF store
- access information in a non-RDF database using the Apache Jena API

## 9.1 D2R Server: Accessing databases with SPARQL and as Linked Data

D2R Server is a tool for publishing relational databases on the Semantic Web. It enables RDF and HTML browsers to navigate the content of the database, and allows querying the database using the SPARQL query language. It is part of the D2RQ Platform.

### About D2R Server

D2R Server is a tool for publishing the content of relational databases on the Semantic Web, a global information space consisting of Linked Data [Figure 9].

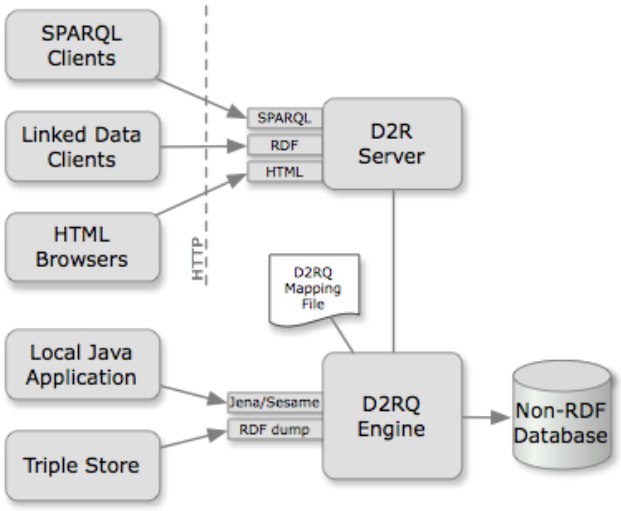



Figure 9 “D2R Server Structure”

Data on the Semantic Web is modelled and represented in RDF. D2R Server uses a customizable D2RQ mapping to map database content into this format, and allows the RDF data to be *browsed* and *searched* – the two main access paradigms to the Semantic Web.

Requests from the Web are rewritten into SQL queries via the mapping. This on-the-fly translation allows publishing of RDF from large live databases and eliminates the need for replicating the data into a dedicated RDF triple store.

## Features

**Browsing database contents**

A simple web interface allows navigation through the database's contents and gives users of the RDF data a “human-readable” preview.

**Resolvable URIs**

Following the Linked Data principles, D2R Server assigns a URI to each entity that is described in the database, and makes those URIs resolvable – that is, an RDF description can be retrieved simply by accessing the entity's URI over the Web. Semantic Web browsers like Marbles or LinkSailor can follow links from one entity to the next, surfing the Web of Data.

**Content negotiation**

Following best practices, the regular web interface and the browsable RDF graph share the same URIs.

**SPARQL endpoint and explorer**

The SPARQL interface enables applications to query the database using the SPARQL 1.1 query language over the SPARQL protocol.

**Downloading contents of BLOBs/CLOBs**

D2R Server can be configured to serve files stored in the database in BLOBs or CLOBs.

**Serving the vocabulary**

If new classes and properties are introduced for a D2R deployment, the server can make their URIs resolvable in the spirit of Linked Data, and allows configuration of their labels, comments, and additional properties.

**Publishing metadata**

Metadata such as licensing and provenance information can be attached to every RDF document and web page published by D2R Server.

### 9.1.1 Getting started with D2R Server

You need:

- **Java 1.5** or newer on the path (check with java -version if you're not sure),
- **A supported database**, such as Oracle, SQL Server, PostgreSQL, MySQL or HSQLDB.
- Optionally, a **J2EE servlet container** as a deployment target. D2R Server can be run either as a stand-alone web server or inside an existing servlet container.

What to do:

1. **Download and extract the archive** into a suitable location.
2. **Download a JDBC driver** from your database vendor. Place the driver's JAR file into D2R Server's /lib directory. A **list of JDBC drivers** from different vendors is maintained by Sun. Also take note of the **driver class name** (e.g. org.postgresql.Driver for PostgreSQL or oracle.jdbc.driver.OracleDriver for Oracle) and **JDBC URL pattern** (e.g. jdbc:mysql://*servername*/*database* for MySQL) from the driver's documentation. Drivers for MySQL and PostgreSQL are already included with D2R Server.
3. **Generate a mapping file** for your database schema using the generate-mapping tool. Change into the D2R Server directory and run:

4. generate-mapping -o *mapping.ttl* -d *driver.class.name* -u *db-user* -p *db-password jdbc:url:...*

*mapping.ttl* is the name for the new mapping file. -d can be skipped for MySQL.

5. **Start D2R Server:**

```
d2r-server mapping.ttl
```

6. **Test the Server:** Open **http://localhost:2020/** in a web browser.


You can browse the database content or use the SPARQL Explorer to execute queries and display results in a number of formats.

7. **Run a SPARQL query from the command line** using **the d2r-query** tool:

```
d2r-query mapping.ttl "SELECT * { ?s ?p ?o } LIMIT 10"
```

Or load the query from a file, here called query.sparql:

```
d2r-query mapping.ttl @query.sparql
```

8. **Generate an RDF dump** using the **dump-rdf** tool:

```
dump-rdf mapping.ttl -o dump.nt
```

After you're up and running, the next step is typically to refine the RDF output by customizing the D2RQ mapping.

The D2RQ Platform uses the **D2RQ Mapping Language** to map the content of a relational database to RDF. A D2RQ mapping specifies how resources are identified and which properties are used to describe the resources.

The generate-mapping **script** automatically generates a D2RQ mapping from the table structure of a database. The tool generates a new RDF vocabulary for each database, using table names as class names and column names as property names. Semantic Web client applications will understand more of your data if you customize the mapping and replace the auto-generated terms with terms from well-known and publicly accessible RDF vocabularies.

The mapping file can be edited with any text editor. Its syntax is described in the **D2RQ language specification**.

D2R Server will automatically detect changes to the mapping file and reload appropriately when you hit the browser's refresh button.

**Note:** The HTML and RDF browser interfaces only work for URI patterns that are *relative* and *do not contain the hash (#) character*. For example, a URI pattern such asentries/@@mytable.id@@ is browsable,

but http://example.com/entries#@@mytable.id@@ is not. The mapping generator only creates browsable patterns. Non-browsable patterns still work in the SPARQL interface and in RDF dumps.

### 9.1.2 Running D2R Server from the command line

D2R Server can be run as a stand-alone server application that includes its own web server. This is recommended for testing and development.

```
d2r-server [-p port] [-b serverBaseURI]
    [--fast] [--verbose] [--debug]
    mapping-file.ttl
```

**mapping-file.ttl**

The name of the D2RQ mapping file to use.

If no mapping file is provided, then the database connection must be specified on the command line using the same connection parameters as for the generate-mapping tool, and a default mapping will be used.

**-p *port***

D2R Server will be started on this port. Defaults to 2020.

**-b *serverBaseURI***

The base URI where D2R Server is running.

Defaults to http://localhost:2020/.

Must be specified if the D2R Server is to be accessible from other machines or if it is supposed to be run on a port other than 2020.

**--fast**

Enables bleeding-edge optimizations that result in better performance but may not be as well-tested. Generally we recommend the use of this option unless problems are observed.

**--verbose**

Output more logging information.

**--debug**

Output much more logging information.

### 9.1.3 Running D2R Server in a servlet container

D2R Server can be run as a J2EE web application inside an existing servlet container, such as Apache Tomcat or Jetty. This is recommended for production use.

1. Make sure that your mapping file includes a configuration block, as described in the server configuration section. Set the base URI to something like http://*servername*/*webappname*/.
2. Change the configFile param in /webapp/WEB-INF/web.xml to the name of your configuration file. For deployment, we recommend placing the mapping file into the /webapp/WEB-INF/ directory.
3. In D2R Server's main directory, run ant war. This creates the d2rq.war file. You need Apache Ant for this step.
4. Optionally, if you want a different name for your web application, rename the file to *webappname*.war
5. Deploy the war file into your servlet container, e.g. by copying it into Tomcat's webapps directory.

### 9.1.4 D2R Server Configuration

The server can be configured by adding a **configuration block** to the mapping file. This consists of a **d2r:Server** instance with configuration properties. An example follows:

```
@prefix d2r: <http://sites.wiwiss.fu-berlin.de/suhl/bizer/d2r-server/config.rdf#> .
@prefix meta: <http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/metadata#> .


<> a d2r:Server;
  rdfs:label "My D2R Server";
  d2r:baseURI <http://localhost:2020/>;
  d2r:port 2020;
  d2r:vocabularyIncludeInstances true;

  d2r:sparqlTimeout 300;
  d2r:pageTimeout 5;

  meta:datasetTitle "My dataset" ;
  meta:datasetDescription "My dataset contains many nice resources."
;
  meta:datasetSource "This other dataset" ;

  meta:operatorName "John Doe" ;
  meta:operatorHomepage  ;
  .
```

### 9.1.5 Server level Configuration Options

The following configuration properties can be set for the d2r:Server instance:

| | |
|---|---|
| **rdfs:label** | The server name displayed throughout the HTML interface. |

| | |
|---|---|
| **d2r:baseURI** | Base URI of the server. Same as -b command line parameter. |
| **d2r:port** | Port of the server. Same as -p command line parameter. |
| **d2r:vocabularyIncludeInstances** | Controls whether the RDF and HTML representations of vocabulary classes will also list instances, and whether the representations of properties also list triples using the property (defaults to true). |
| **d2r:autoReloadMapping** | Specifies whether changes to the mapping file should be detected automatically (defaults to true). This feature is convenient for development, but has performance implications, so this value should be set to false for production systems. |
| **d2r:limitPerClassMap** | Specifies a maximum for the number of entities per class map that will be displayed in the “directory” pages of the web interface. This stops pages from getting too large, but prevents users from exploring the full data through the web interface. This setting does not affect the RDF output or SPARQL queries. The default is 50. Use false to disable the limit. |
| **d2r:limitPerPropertyBridge** | Specifies a maximum for the number of values from each property bridge that will |

| | |
|---|---|
| | be displayed in the web interface. This stops pages from getting too large, but prevents users from exploring the full data through the web interface. This setting does not affect RDF representations or SPARQL queries. The default is 50. Use false to disable the limit. |
| **d2r:sparqlTimeout** | Specifies a timeout in seconds for the server's SPARQL endpoint. A value of 0 or a negative value disables the timeout. |
| **d2r:pageTimeout** | Specifies a timeout in seconds for generating resource description pages. A value of 0 or a negative value disables the timeout. |
| **d2r:metadataTemplate** | Overrides the default *resource* metadata template, refers to a TTL-encoded RDF file. The literal value specifies a path name either absolute or relative to the location of the server configuration file. |
| **d2r:datasetMetadataTemplate** | Overrides the default *dataset* metadata template, refers to a TTL-encoded RDF file. The literal value specifies a path name either absolute or relative to the location of the server configuration file. |
| **d2r:disableMetadata** | Enables the automatic creation and publication of all dataset and resource |

| | |
|---|---|
| | metadata.<br>Possible values are "true" and "false".<br>Note that "true" is assumed if this flag is missing. |
| **d2r:documentMetadata** | A simpler alternative to d2r:metadataTemplate: The value should be a blank node. Any statements involving this blank node will be copied as metadata into any RDF documents generated by D2R Server, with the blank node replaced with the document's URL. # |

Note that further configuration options can be set elsewhere in the mapping file. Database-level configuration is specified on the d2rq:Database instances, and configuration of the D2RQ query engine is specified on a d2rq:Configuration instance.

### 9.1.6 Dataset and Resource Metadata

Often, providing additional information for the served resources is desirable. Main areas for this additional information include licensing, proveance, and general dataset descriptions. D2R Server has comprehensive support for this so-called metadata. First, metadata can be provided on two levels: Every served resource can have metadata assigned, and the entire dataset served by the D2R server installation can also have metadata assigned. Metadata templates are RDF documents, that can contain placeholders, which are replaced with user-specified information, configuration values, or run-time information. The resource metadata are added to the RDF and HTML responses for each requested resource, while the dataset metdata is served at a single URL which is created by appending /dataset to the configuration valued2r:baseURI. Most of the dataset metadata is auto-generated from the

mapping file, with the user-specified dataset metadata then being mixed into the RDF and HTML representation of the dataset metadata.

### 9.1.7 Optimizing Performance

Here are some simple hints to improve D2R's performance:

- Define primary keys whenever you can and create indexes where applicable (e.g. on foreign keys) – besides optimizing database performance, these will be picked up and used by various optimizations within D2RQ.
- Use the latest optimizations by launching D2R Server with --fast (or activate d2rq:useAllOptimizations).
- Provide hint properties.
- Indicate directions in d2rq:joins to enable join optimizations.
- Give D2R Server more heap space by means of Java's -Xmx parameter in d2r-server or d2r-server.bat (default: -Xmx1G).
- To prevent excessively large pages in the web interface, consider changing the values of

  d2r:limitPerClassMap and d2r:limitPerPropertyBridge.

- Consider changing the default timeout values for SPARQL queries and for page generation.
- To speed up the generation of large pages, consider setting d2rq:resultSizeLimit, d2rq:limit,
  and d2r:vocabularyIncludeInstances.
- Set d2rq:autoReloadMapping to false where it is not required.
- Databases often ship with development configurations that are designed for a small footprint rather than performance. For instance, some good pointers for optimizing MySQL can be the key buffer size, the additional buffer memory.

## 9.2 Auto-generating D2RQ mapping files

The **generate-mapping** tool creates a D2RQ mapping file by analyzing the schema of an existing database. This mapping file, called the default mapping, maps each table to a new RDFS class that is based on the table's name, and maps each column to a property based on the column's name. This mapping file can be used as-is or can be customized.

### Usage

```
generate-mapping [-u user] [-p password] [-d driver]
        [-l script.sql] [--[skip-](schemas|tables|columns) list]
        [--w3c] [-v] [-b baseURI] [-o outfile.ttl]
        [--verbose] [--debug]
        jdbcURL
```

### Connection Parameters

**jdbcURL**

JDBC connection URL for the database. Refer to your JDBC driver documentation for the format for your database engine. Examples:

**MySQL**: jdbc:mysql://*servername*/*databasename*
**PostgreSQL**: jdbc:postgresql://*servername*/*databasename*
**Oracle**: jdbc:oracle:thin:@*servername*:1521:*databasename*
**HSQLDB**: jdbc:hsqldb:mem:*databasename* (in-memory database)
**MSQLServer**: jdbc:sqlserver://*servername*;databaseName=*databasename* (due to the semicolon, the URL must be put in quotes when passed as a command-line argument in Linux/Unix shells)

If -l is present, then the JDBC URL can be omitted to load a SQL script into an in-memory HSQLDB database.

**-u user**

The login name of the database user.

**-p password**

The password of the database user.

**-d driver**

The fully qualified Java class name of the database driver. For MySQL, PostgreSQL, and HSQLDB, this argument can be omitted as drivers are already included with D2RQ. For other databases, a driver has to be downloaded from the vendor or a third party. The jar file containing the JDBC driver class has to be in D2RQ's /lib/db-drivers/ directory. To find the driver class name, consult the driver documentation. Examples:

Oracle: oracle.jdbc.OracleDriver
MSQL Server: com.microsoft.sqlserver.jdbc.SQLServerDriver

**-l script.sql**

Load a SQL script before running the tool. Useful for initializing the connection and testing. The d2rq:startupSQLScript property of the database in the generated mapping will be initialized with the same value.

**--schemas, --tables, --columns, --skip-schemas, --skip-tables, --skip-columns**

Only map the specified schemas, tables or columns. The value of each argument is a comma-separated list of names. Schema names are of the form schema, table names of the form table or schema. table, and column names are

table.column or schema.table.column.

Each dot-separated segment can be specified as a regular expression encloded between slashes.

If the value starts with “@”, then it is interpreted as a file name, and the list of names is loaded from the file. The file contains one name per line or comma-separated names. Examples follow:

- --schema SCOTT (maps only tables in the SCOTT schema)
- --tables PERSONS,ORGS (maps only the PERSONS and ORGS tables)
- --skip-table TEMP_CACHE (skips the TEMP_CACHE table)
- --skip-columns /.*/.CHECKSUM (skips the CHECKSUM column of each table, if present)
- --skip-tables /BACKUP.*/i (skips tables backup1, BACKUP_2 and so on)
- --skip-tables @exclude.txt (reads a list of excluded table names from a file

## Output parameters

**--w3c**

Generate a mapping file that is compatible with W3C's Direct Mapping of Relational Data to RDF. **This is an experimental feature and work in progress.**

**-v**

Generate an RDF Schema description of the vocabulary instead of a mapping file.

**-o outfile.ttl**

The generated mapping (or vocabulary if -v is used) will be stored in this file in Turtle syntax. If this parameter is omitted, the result will be written to standard out.

**-b baseURI**

The base URI is used to construct a vocabulary namespace that will automatically be served as Linked Data by D2R Server, following the convention http://*baseURI*/vocab/resource/. This should be the same base URI that is used when invoking the server. Defaults to http://localhost:2020/.

**--verbose**

Print extra progress log information.

**--debug**

Print all debug log information.

## Examples

Local MySQL database

```
generate-mapping -u root jdbc:mysql:///iswc
```

Remote Oracle database

```
generate-mapping -u riccyg -p password -d oracle.jdbc.OracleDriver
    -o                                          staffdb-mapping.ttl
jdbc:oracle:thin:@ora.intranet.deri.ie:1521:staffdb
```

### 9.2.1 Direct Mapping Description

The direct mapping defines an RDF Graph [RDF-concepts] representation of the data in a relational database. The direct mapping takes as input a relational database (data and schema), and generates an RDF graph that is called the **direct graph**. The algorithms in this document compose a graph of relative IRIs which must be resolved against a **base** IRI [RFC3987] to form an RDF graph.

Foreign keys in relational databases establish a reference from any row in a table to exactly one row in a (potentially different) table. The direct graph conveys these references, as well as each value in the row.

#### Direct Mapping Example

The concepts in direct mapping can be introduced with an example RDF graph produced by a relational database. Following is SQL (DDL) to create a simple example with two tables with single-column primary keys and one foreign key reference between them:

```
CREATE TABLE "Addresses" (
    "ID" INT, PRIMARY KEY("ID"),
    "city" CHAR(10),
    "state" CHAR(2)
)

CREATE TABLE "People" (
    "ID" INT, PRIMARY KEY("ID"),
    "fname" CHAR(10),
    "addr" INT,
    FOREIGN KEY("addr") REFERENCES "Addresses"("ID")
)

INSERT INTO "Addresses" ("ID", "city", "state") VALUES (18, 'Cambridge',
'MA')
INSERT INTO "People" ("ID", "fname", "addr") VALUES (7, 'Bob', 18)
INSERT INTO "People" ("ID", "fname", "addr") VALUES (8, 'Sue', NULL)
```

HTML tables will be used in this document to convey SQL tables. The primary key of these tables will be marked with the PK class to convey an SQL primary key such as ID in

CREATE TABLE "Addresses" ("ID" INT, ... PRIMARY KEY("ID")).

Foreign keys will be illustrated with a notation like "→ Address(ID)" to convey an SQL foreign key such as:

CREATE TABLE "People" (... "addr" INT, FOREIGN KEY("addr") REFERENCES "Addresses"("ID")).

| People | | |
|---|---|---|
| *PK* | | *→ Address(ID)* |
| **ID** | **fname** | **addr** |
| 7 | Bob | 18 |
| 8 | Sue | **NULL** |

| Addresses | | |
|---|---|---|
| *PK* | | |
| **ID** | **city** | **state** |
| 18 | Cambridge | MA |

Given a base IRI `http://foo.example/DB/`, the direct mapping of this database produces a direct graph:

```
@base <http://foo.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

```
<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18> .
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .
```

In this expression, each row, e.g. (7, "Bob", 18), produces a set of triples with a common subject. The subject is an IRI formed from the concatenation of the base IRI, table name (People), primary key column name (ID) and primary key value (7). The predicate for each column is an IRI formed from the concatenation of the base IRI, table name and the column name. The values are RDF literals formed from the lexical form of the column value. Each foreign key produces a triple with a predicate composed from the foreign key column names, the referenced table, and the referenced column names. The object of these triples is the row identifier (<Addresses/ID=18>) for the referenced triple. Note that these reference row identifiers must coincide with the subject used for the triples generated from the referenced row. The direct mapping does not generate triples for NULL values. Note that it is not known how to relate the behavior of the obtained RDF graph with the standard SQL semantics of the NULL values of the source RDB.

## Foreign Keys referencing candidate keys

More complex schemas include composite keys. In this example, the columns **deptName** and **deptCity** in the **People** table reference **name** and **city** in the **Department** table:

```
CREATE TABLE "Addresses" (
    "ID" INT,
    "city" CHAR(10),
    "state" CHAR(2),
    PRIMARY KEY("ID")
)

CREATE TABLE "Department" (
```

```
    "ID" INT,
    "name" CHAR(10),
    "city" CHAR(10),
    "manager" INT,
    PRIMARY KEY("ID"),
    UNIQUE ("name", "city")
)

CREATE TABLE "People" (
    "ID" INT,
    "fname" CHAR(10),
    "addr" INT,
    "deptName" CHAR(10),
    "deptCity" CHAR(10),
    PRIMARY KEY("ID"),
    FOREIGN KEY("addr") REFERENCES "Addresses"("ID"),
    FOREIGN      KEY("deptName",      "deptCity")      REFERENCES
"Department"("name", "city")
)

ALTER TABLE "Department" ADD FOREIGN KEY("manager") REFERENCES
"People"("ID")
```

Following is an instance of this schema:

| People | | | | |
|---|---|---|---|---|
| *PK* | | *→ Addresses(ID)* | *→ Department(name, city)* | |
| **ID** | **fname** | **addr** | **deptName** | **deptCity** |
| 7 | Bob | 18 | accounting | Cambridge |
| 8 | Sue | **NULL** | **NULL** | **NULL** |

| Addresses | | |
|---|---|---|
| *PK* | | |
| **ID** | **city** | **state** |
| 18 | Cambridge | MA |

| Department | | | |
|---|---|---|---|
| *PK* | *Unique Key* | | *→ People(ID)* |
| **ID** | **name** | **City** | **manager** |
| 23 | accounting | Cambridge | 8 |

Per the **People** table's compound foreign key to Department:

- The rowin **People** with deptName="accounting" and deptCity="Cambridge" references a row in **Department** with a primary key of ID=23.
- The predicate for this key is formed from "deptName" and "deptCity", reflecting the order of the column names in the foreign key.
- The object of the above predicate is formed from the base IRI, the table name "Department" and the primary key value "ID=23".

Note: The order of a primary key constraint's columns is determined by the DDL statement used to create it. In SQL implementations that support the information schema, this order can be accessed through the INFORMATION_SCHEMA.KEY_COLUMN_USAGE.ORDINAL_POSITIO N column.

In this example, the direct mapping generates the following triples:

```
@base <http://foo.example/DB/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<People/ID=7> rdf:type <People> .
<People/ID=7> <People#ID> 7 .
<People/ID=7> <People#fname> "Bob" .
<People/ID=7> <People#addr> 18 .
<People/ID=7> <People#ref-addr> <Addresses/ID=18> .
<People/ID=7> <People#deptName> "accounting" .
<People/ID=7> <People#deptCity> "Cambridge" .
<People/ID=7> <People#ref-deptName;deptCity> <Department/ID=23> .
<People/ID=8> rdf:type <People> .
<People/ID=8> <People#ID> 8 .
<People/ID=8> <People#fname> "Sue" .

<Addresses/ID=18> rdf:type <Addresses> .
<Addresses/ID=18> <Addresses#ID> 18 .
<Addresses/ID=18> <Addresses#city> "Cambridge" .
<Addresses/ID=18> <Addresses#state> "MA" .

<Department/ID=23> rdf:type <Department> .
<Department/ID=23> <Department#ID> 23 .
<Department/ID=23> <Department#name> "accounting" .
<Department/ID=23> <Department#city> "Cambridge" .
<Department/ID=23> <Department#manager> 8 .
<Department/ID=23> <Department#ref-manager> <People#ID=8> .
```

The green triples above are generated by considering the new elements in the augmented database. Note:

- The Reference Triple

  <People/ID=7><People#ref-deptName;deptCity> <Department/ID=23>

  is generated by considering a foreign key referencing a candidate key (different from the primary key).

## Multi - column Primary Keys

Primary keys may also be composite. If, in the above example, the primary key for **Department** were (**name**, **city**) instead of **ID**, the identifier for the only row in this table would be

<Department/name=accounting;city=Cambridge>.

The triples involving <Department/ID=23> would be replaced with the following triples:

```
<People/ID=7>                          <People#ref-deptName;deptCity>
<Department/name=accounting;city=Cambridge> .
<Department/name=accounting;city=Cambridge> rdf:type <Department>
.
<Department/name=accounting;city=Cambridge> <Department#ID> 23 .
<Department/name=accounting;city=Cambridge>     <Department#name>
"accounting" .
<Department/name=accounting;city=Cambridge>     <Department#city>
"Cambridge" .
```

## Empty (Non Existent) Primary Keys

If there is no primary key, each row implies a set of triples with a shared subject, but that subject is a blank node. A **Tweets** table can be added to the above example to keep track of employees' tweets in Twitter:

```
CREATE TABLE "Tweets" (
    "tweeter" INT,
    "when" TIMESTAMP,
    "text" CHAR(140),
    FOREIGN KEY("tweeter") REFERENCES "People"("ID")
)
```

The following is an instance of table **Tweets**:

| Tweets | | |
|---|---|---|
| → *People(ID)* | | |
| **tweeter** | **when** | **text** |
| 7 | 2010-08-30T01:33 | I really like lolcats. |
| 7 | 2010-08-30T09:01 | I take it back. |

Given that table **Tweets** does not have a primary key, each row in this table is identified by a Blank Node. In fact, when translating the above table the direct mapping generates the following triples:

```
@base <http://foo.example/DB/>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:a rdf:type <Tweets> .
_:a <Tweets#tweeter> "7" .
_:a <Tweets#ref-tweeter> <People/ID=7> .
_:a <Tweets#when> "2010-08-30T01:33"^^xsd:dateTime .
_:a <Tweets#text> "I really like lolcats." .

_:b rdf:type <Tweets> .
_:b <Tweets#tweeter> "7" .
_:b <Tweets#ref-tweeter> <People/ID=7> .
_:b <Tweets#when> "2010-08-30T09:01"^^xsd:dateTime .
_:b <Tweets#text> "I take it back." .
```

## Referencing Tables with Empty Primary Keys

Rows in tables with no primary key may still be referenced by foreign keys. (Relational database theory tells us that these rows must be unique as foreign keys reference candidate keys and candidate keys are unique across all the rows in a table.) References to rows in tables with no primary key are expressed as RDF triples with blank nodes for objects, where that blank node is the same node used for the subject in the referenced row.

Here is DDL for a schema with references to a Projects table which has no primary key:

```
CREATE TABLE "Projects" (
    "lead" INT,
    FOREIGN KEY ("lead") REFERENCES "People"("ID"),
    "name" VARCHAR(50),
    UNIQUE ("lead", "name"),
    "deptName" VARCHAR(50),
    "deptCity" VARCHAR(50),
    UNIQUE ("name", "deptName", "deptCity"),
```

```
    FOREIGN     KEY     ("deptName",     "deptCity")     REFERENCES
"Department"("name", "city")
)

CREATE TABLE "TaskAssignments" (
    "worker" INT,
    FOREIGN KEY ("worker") REFERENCES "People"("ID"),
    "project" VARCHAR(50),
    PRIMARY KEY ("worker", "project"),
    "deptName" VARCHAR(50),
    "deptCity" VARCHAR(50),
    FOREIGN KEY ("worker") REFERENCES "People"("ID"),
    FOREIGN  KEY  ("project",  "deptName",  "deptCity")  REFERENCES
"Projects"("name", "deptName", "deptCity"),
    FOREIGN     KEY     ("deptName",     "deptCity")     REFERENCES
"Department"("name", "city")
)
```

The following is an instance of the preceding schema:

| Projects | | | |
|---|---|---|---|
| *Unique key* | | | |
| | *Unique key* | | |
| *→ People(ID)* | | *→ Department(name, city)* | |
| **lead** | **name** | **deptName** | **deptCity** |
| 8 | pencil survey | accounting | Cambridge |
| 8 | eraser survey | accounting | Cambridge |

| TaskAssignments | | | |
|---|---|---|---|
| *PK* | | | |
| | *→ Projects(name, deptName, deptCity)* | | |
| *→ People(ID)* | | *→ Departments(name, city)* | |
| **worker** | **project** | **deptName** | **deptCity** |
| 7 | pencil survey | accounting | Cambridge |

In this case, the direct mapping generates the following triples from the preceding tables:

```
@base <http://foo.example/DB/>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:c rdf:type <Projects> .
_:c <Projects#lead> <People/ID=8> .
_:c <Projects#name> "pencil survey" .
_:c <Projects#deptName> "accounting" .
_:c <Projects#deptCity> "Cambridge" .
_:c <Projects#ref-deptName;deptCity> <Department/ID=23> .

_:d rdf:type <Projects> .
_:d <Projects#lead> <People/ID=8> .
_:d <Projects#name> "eraser survey" .
_:d <Projects#deptName> "accounting" .
_:d <Projects#deptCity> "Cambridge" .
_:d <Projects#ref-deptName;deptCity> <Department/ID=23> .

<TaskAssignments/worker=7.project=pencil%20survey> rdf:type
<TaskAssignments> .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#worker> 7 .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#ref-worker> <People/ID=7> .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#project> "pencil survey" .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#deptName> "accounting" .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#deptCity> "Cambridge" .
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#ref-deptName;deptCity>  <Department/ID=23>
.
<TaskAssignments/worker=7.project=pencil%20survey>
<TaskAssignments#ref-project;deptName;deptCity>  _:c .
```

The absence of a primary key forces the generation of blank nodes, but does not change the structure of the direct graph or names of the predicates in that graph.

## 9.3 d2r-query: Running SPARQL queries against a database

The d2r-query tool allows executing SPARQL queries against a D2RQ mapped relational database from the command line. This can be done with or without a D2RQ mapping file. If a mapping file is specified, then the tool will query the virtual RDF graph defined by the mapping. If no mapping file is specified, then the tool will use the default mapping of generate-mapping for the translation.

To query a D2RQ-mapped database using a web-based interface, use D2R Server.

### Usage

```
d2r-query [-f format] [-b baseURI] [-t timeout] [--verbose] [--debug]
mapping-file.ttl query
```

**mapping-file.ttl**

The filename of a D2RQ mapping file that contains a database mapping.

If no mapping file is provided, then the database connection must be specified on the command line using the same connection parameters as for the generate-mappingtool, and a default mapping will be used.

**query**

A SPARQL query. All prefixes defined in the mapping file are available without being declared. The query can also be read from a file by using the syntax @filename.

**-f format**

The output format. Supported formats include text

(the default), xml, json, csv, tsv, srb, and ttl.

**-b baseURI**

The base URI for turning relative URIs and URI patterns into absolute URIs. It is used both for the data and for the query.

**-t timeout**

Query timeout in seconds.

**--verbose**

Print extra progress log information.

**--debug**

Print all debug log information.

## Examples

### Invocation using a mapping file

```
d2r-query mapping-iswc.ttl "SELECT * { ?s ?p ?o } LIMIT 10"
```

### Writing results to a CSV file

```
d2r-query -f csv mapping-iswc.ttl "SELECT * { ?paper dc:title ?title }" > papers.csv
```

**Invocation with default mapping**

```
d2r-query -u root jdbc:mysql:///iswc "SELECT * { ?s ?p ?o } LIMIT 10"
```

**Querying a SQL dump using a temporary in-memory database**

```
dump-rdf -l db_dump.sql -o output.nt "SELECT * { ?s ?p ?o } LIMIT 10"
```

**Reading the query from a file**

```
dump-rdf mapping.ttl @my-query.sparql
```

## 9.4 dump-rdf: Dumping the database to an RDF file

The **dump-rdf** tool uses D2RQ to **dump the contents of the whole database into a single RDF file**. This can be done with or without a D2RQ mapping file. If a mapping file is specified, then the tool will use it to translate the database contents to RDF. If no mapping file is specified, then the tool will use the default mapping of generate-mapping for the translation.

### Usage

```
dump-rdf [-f format] [-b baseURI] [-o outfile.ttl]
        [--verbose] [--debug]
        mapping-file.ttl
```

**mapping-file.ttl**

The filename of a D2RQ mapping file that contains a database mapping.

If no mapping file is provided, then the database connection must be specified on the command line using the same connection parameters as for the generate-mapping tool, and a default mapping will be used.

**-f format**

The RDF syntax to use for output. Supported syntaxes are “TURTLE”, “RDF/XML”, “RDF/XML-ABBREV”, “N3”, and “N-TRIPLE” (the default). “N-TRIPLE” works best for large databases.

**-b baseURI**

The base URI for turning relative URIs and URI patterns into absolute URIs.

**-o outfile**

Name of the destination file. Defaults to standard output.

**--verbose**

Print extra progress log information.

**--debug**

Print all debug log information.

## Examples

**Dump using a mapping file**

```
dump-rdf -f N-TRIPLE -b http://localhost:2020/ mapping-iswc.ttl > iswc.nt
```

**Dump with default mapping**

```
dump-rdf -u root -f RDF/XML-ABBREV -o iswc-dump.rdf jdbc:mysql:///iswc
```

This dumps to RDF/XML format and writes the output to a file iswc-dump.rdf.

# 10. IMPLEMENTATION

## 10.1 Relational DB to Linked Data

1. Create the database iswc in MySQL (iswc.sql file has given).
2. Feed the downloaded iswc-mysql.sql into mysql by running (you should specify the full path of the downladed iswc-mysql.sql if you run mysql in another directory):

```
$ mysql -u [username] -p[password] < iswc-mysql.sql
```

3. Check that the database was created OK. In MySql run, for example:

```
show tables;
```

And

```
select FirstName, LastName, email from persons;
```

```
mysql> show tables;
+-------------------------+
| Tables_in_iswc          |
+-------------------------+
| conferences             |
| organizations           |
| papers                  |
| persons                 |
| rel_paper_topic         |
| rel_person_organization |
| rel_person_paper        |
| rel_person_topic        |
| topics                  |
+-------------------------+
9 rows in set (0.01 sec)

mysql> select FirstName, LastName, email from persons;
+-----------+-------------+---------------------------+
| FirstName | LastName    | email                     |
+-----------+-------------+---------------------------+
| Yolanda   | Gil         | gil@isi.edu               |
| Varun     | Ratnakar    | varunr@isi.edu            |
| Jim       | Blythe      | blythe@isi.edu            |
| Andreas   | Eberhart    | eberhart@i-u.de           |
| Borys     | Omelayenko  | borys@cs.vu.nl            |
| Andy      | Seaborne    | andy.seaborne@hpl.hp.com  |
| Alberto   | Reggiori    | areggiori@webweaving.org  |
| Sonia     | Bergamaschi | bergamaschi.sonia@unimo.it|
| Francesco | Guerra      | guerra.francesco@unimo.it |
| Christian | Bizer       | chris@bizer.de            |
+-----------+-------------+---------------------------+
10 rows in set (0.00 sec)
```

4. Download the mapping file **mapping-iswc.ttl** (file has given).
5. Change the username in the downloaded mapping file to be your mysql username and add a password of your mysql (if the password is required for your mysql)

```
File Edit Options Buffers Tools Help
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix iswc: <http://annotation.semanticweb.org/iswc/iswc.daml#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix jdbc: <http://d2rq.org/terms/jdbc/> .

map:database a d2rq:Database;
        d2rq:jdbcDriver "com.mysql.jdbc.Driver";
        d2rq:jdbcDSN "jdbc:mysql://127.0.0.1/iswc?autoReconnect=true";
        d2rq:username "username";
        d2rq:password "password";
        jdbc:keepAlive "3600";                          # sends noop-query ever\
y 3600 seconds
#       jdbc:keepAliveQuery "SELECT 1";         # optional custom noop-query
        .

# Table conferences
map:Conferences a d2rq:ClassMap;
        d2rq:dataStorage map:database;
        d2rq:
        d2rq:class iswc:Conference;
```

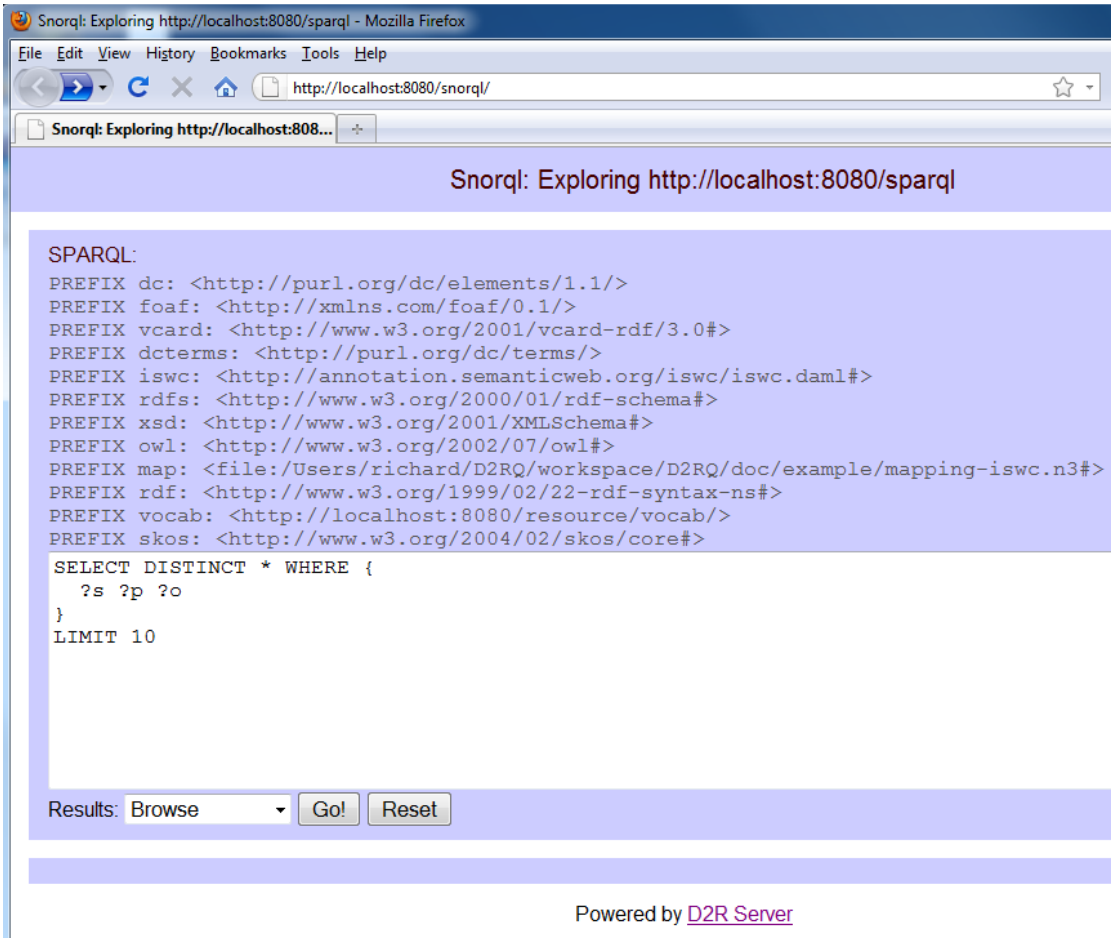
6. Run d2r-server in the directory you uncompressed it in the step 1 (you should specify the full path of the mapping-iswc.ttl if you run mysql in another directory):

```
d2r-server -p 8080 mapping-iswc.ttl
```

```
C:\D2RQtutorials\D2RQManual\d2r-server-0.7>d2r-server -p 8080 mapping-iswc.n3
06:35:56 INFO  log              :: Logging to org.slf4j.impl.Log4jLoggerAdapter(org.mortbay.log) via org.mortbay.log.Slf4jLog
06:35:56 INFO  log              :: jetty-6.1.10
06:35:56 INFO  log              :: NO JSP Support for , did not find org.apache.jasper.servlet.JspServlet
06:35:56 INFO  D2RServer        :: using port 8080
06:35:56 INFO  D2RServer        :: using config file: file:/C:/D2RQtutorials/D2RQManual/d2r-server-0.7/mapping-iswc.n3
06:35:57 INFO  ConnectedDB      :: Keep alive agent is enabled (interval: 3600 seconds, noop query: 'SELECT 1').
06:35:58 INFO  D2RServer        :: Safe mode (launch using --fast to use all optimizations)
06:35:59 INFO  log              :: Started SocketConnector@0.0.0.0:8080
06:35:59 INFO  server           :: [[[ Server started at http://localhost:8080/ ]]]
```

7. Open the following URL with your web browser:

**http://localhost:8080/snorql/**



8. Run an example SPARQL query. Write the query into the text of the *SPARQL* section and push the *"Go!"* button. You will see the results of the query in the *SPARQL results*section. The following query returns the names and the email addresses of all the people in the database. You can see that the results match the SQL query ***select FirstName, LastName, email from persons;*** that you run in mysql

```
SELECT DISTINCT ?name ?email WHERE {
  ?person rdf:type foaf:Person.
  ?person foaf:name ?name ;
          foaf:mbox ?email
}
```

SPARQL:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX iswc: <http://annotation.semanticweb.org/iswc/iswc.daml#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX map: <file:/Users/richard/D2RQ/workspace/D2RQ/doc/example/mapping-iswc.n3#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX vocab: <http://localhost:8080/resource/vocab/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT DISTINCT ?name ?email WHERE {
  ?person rdf:type foaf:Person.
  ?person foaf:name ?name ;
          foaf:mbox ?email
}
LIMIT 10
```

Results: Browse Go! Reset

SPARQL results:

| name | email |
|---|---|
| "Yolanda Gil" | <mailto:gil@isi.edu> |
| "Varun Ratnakar" | <mailto:varunr@isi.edu> |
| "Jim Blythe" | <mailto:blythe@isi.edu> |
| "Andreas Eberhart" | <mailto:eberhart@i-u.de> |
| "Borys Omelayenko" | <mailto:borys@cs.vu.nl> |
| "Andy Seaborne" | <mailto:andy.seaborne@hpl.hp.com> |
| "Alberto Reggiori" | <mailto:areggiori@webweaving.org> |
| "Sonia Bergamaschi" | <mailto:bergamaschi.sonia@unimo.it> |
| "Francesco Guerra" | <mailto:guerra.francesco@unimo.it> |
| "Christian Bizer" | <mailto:chris@bizer.de> |

Powered by D2R Server

9. Examples of additional SPARQL queries

- Select people and topics of their interest:

```
SELECT DISTINCT ?personName ?topicName  WHERE {
        ?person rdf:type foaf:Person.
        ?person foaf:name ?personName.
        ?person iswc:research_interests ?topic.
        ?topic rdfs:label ?topicName .
        }
```

- Select persons whose research interests include Semantic Web:

```
SELECT DISTINCT ?personName ?topicName  WHERE {
        ?person rdf:type foaf:Person.
        ?person foaf:name ?personName.
        ?person iswc:research_interests ?topic.
        ?topic rdfs:label ?topicName .
        FILTER (?topicName = "Semantic Web")
        }
```

- Select all organizations where the people interested in Semantic Web work:

```
SELECT DISTINCT ?organizationName ?personName  WHERE {
        ?person rdf:type foaf:Person.
        ?person foaf:name ?personName.
        ?person iswc:research_interests ?topic.
        ?topic rdfs:label ?topicName .
        FILTER (?topicName = "Semantic Web").
        ?person iswc:has_affiliation ?organization .
        ?organization rdfs:label ?organizationName
        }
```

- Select people who wrote papers on the topic "Semantic Web"

```
SELECT DISTINCT ?personName ?paperTitle  WHERE {
        ?paper dc:creator ?person .
        ?person foaf:name ?personName.
        ?paper dc:title ?paperTitle  .
        ?paper skos:subject ?topic.
        ?topic rdfs:label ?topicName .
        FILTER (?topicName = "Semantic Web")
        }
```

- Select people who wrote papers on topic that is not of their interest:

```
    SELECT  DISTINCT  ?personName  ?paperTitle    ?paperTopicName
WHERE {
          ?paper dc:creator ?person .
          ?person foaf:name ?personName.
          ?paper dc:title ?paperTitle  .
          ?paper skos:subject ?paperTopic.
          ?paperTopic rdfs:label ?paperTopicName .
          OPTIONAL    {    ?person    iswc:research_interests
   ?personTopic .
          FILTER (?personTopic = ?paperTopic) }
          FILTER ( !BOUND(?personTopic) )
    }
```

- Select papers and their authors and organizations:

```
    SELECT  DISTINCT  ?paperTitle  ?authorName  ?organizationName
WHERE {
          ?paper dc:creator ?author .
          ?author foaf:name ?authorName.
          ?paper dc:title ?paperTitle  .
          ?author iswc:has_affiliation ?organization .
          ?organization rdfs:label ?organizationName
          }
```

- Select papers that were written by authors from different organizations:

```
    SELECT  DISTINCT  ?paperTitle  ?authorName  ?organizationName
WHERE {
          ?paper dc:creator ?author .
          ?author foaf:name ?authorName.
          ?paper dc:title ?paperTitle  .
          ?author iswc:has_affiliation ?organization .
          ?organization rdfs:label ?organizationName
          OPTIONAL { ?paper dc:creator ?anotherAuthor .
           ?anotherAuthor                  iswc:has_affiliation
   ?anotherOrganization .
           FILTER(?anotherAuthor != ?author &&
                  ?anotherOrganization    !=    ?organization)
   }FILTER ( BOUND(?anotherAuthor))       }
```