5 RDF and SPARQL

Objective:

Once data is stored in a database, it's an obvious fact that it must be queried, in order to retrieve the data and use it in an application. This chapter discusses about the data model of graphs, i.e. RDF, and SPARQL which is a querying language for RDF.

5.1 Introduction to RDF

RDF stands for Resource Description Framework. RDF is a family of World Wide Web Consortium (W3C) specifications, originally designed as a metadata data model. It is used as a general method for conceptual description or modeling of information that is implemented in web resources, using a variety of syntax notations and data serialization formats.

5.1.1 Explanation

RDF is the format in which the graph data model, used by the Semantic Web to store data, is written. RDF is a general method to decompose knowledge into small pieces, with some rules about the semantics, or meaning of those pieces. The point is to have a method so simple that it can express any fact, and yet so structured that computer applications can do useful things with knowledge expressed in RDF. It is termed as 'method' in particular rather than format, because one can write down those pieces in any number of ways and still preserve the original information and structure, just like how one can express the same meaning in different human languages or implement the same data structure in multiple ways.

RDF can be defined using three simple rules:

- A fact is expressed as a triple of the form (Subject, Predicate, and Object). It's almost like an English sentence.
- Subjects, predicates, and objects are names of entities, whether concrete or abstract, in the real world. Names are either:
  1) Global and refer to the same entity in any RDF document in which they appear, or
  2) Local and the entity it refers to cannot be directly referred, outside the RDF document.
- Objects can also be text values, called literal values.

5.1.2 Revisiting XML

XML stands for extensible Markup Language. XML is a specification for computer-readable documents. Markup means that certain sequences of characters in the document contain information indicating the role of the document's content. The markup describes the document's data layout and logical structure and makes the information self-describing, in a sense. It takes the form of words between the tags – for example, <name> or <age>.
In this aspect, XML looks very much like the well-known language HTML.

Example:
<?xml version="1.0"?>
<employees>
List of persons in company:
<person name="Nandini">
<phone>123456789</phone>
</person>
</employees>

XML does not imply a specific interpretation of the data. Of course, on account of the tag's names, the meaning of the previous piece of XML seems obvious to human users, but it is not formally specified.

The only legitimate interpretation is that XML code contains named entities with sub-entities and values; that is, every XML document forms an ordered, labeled tree. This generality is both XML's strength and its weakness. You can encode all kinds of data structures in an unambiguous syntax, but XML does not specify the data's use and semantics. The parties that use XML for their data exchange must agree beforehand on the vocabulary, its use and its meaning.

5.1.2.1 DTD and XML Schema

DTD stands for Document Type Definitions. DTD and XML Schemas partly answered the drawback of XML. Although DTDs and XML Schemas do not specify the data's meaning, they do specify the names of elements and attributes (the vocabulary) and their use in documents. Both are mechanisms with which you can specify the structure of XML documents. You can then validate specific documents against the structure prescription specified by a DTD or an XML Schema. DTDs specify the allowed nesting of elements, the element's possible attributes, and the locations where normal text is allowed. For example, a DTD might prescribe that every 'person' element must have a "name" attribute and may have a child element called 'phone' whose content must be text. A DTD's syntax looks a bit awkward, but it is actually quite simple.

XML Schemas are a proposed successor to DTDs. The XML Schema definition is still a candidate recommendation from the W3C (World Wide Web Consortium), which means that, although it is considered stable, it might still undergo small revisions. XML Schemas have several advantages over DTDs. First, the XML Schema mechanism provides a richer grammar for prescribing the structure of elements. For example, you can specify the exact number of allowed occurrences of child elements, you can specify default values, and you can put the elements in a choice group, which means that exactly one of the elements in that group is allowed at a specific location. Second, it provides data typing. A final difference from DTDs is that XML Schema prescriptions use XML as their encoding syntax. This simplifies tool development, because both the structure prescription and the prescribed documents use the same syntax. The XML Schema specification's developers exploited this feature by using an XML Schema document to define the class of XML Schema documents. After all, because an XML Schema prescription is an XML application, it must obey rules for its structure, which can be defined by another XML Schema prescription. However, this recursive definition can be a bit confusing.

XML Entities

An XML entity can play several roles, such as a placeholder for repeatable characters (a type of shorthand), a section of external data (e.g., XML or other), or as a part of a declaration of elements.
For example, suppose that a document has several copyright notices that refer to the current year. Then it makes sense to declare an entity <!ENTITY thisyear "2007">.

Then, at each place where the current year needs to be included, we can use the entity reference &thisyear; instead. That way, updating the year value to "2008" for the whole document will only mean changing the entity declaration.

5.1.3 Why bother about XML in RDF

XML provides syntax to encode data**.** It means that the resource description framework is a mechanism to tell something about the data. As the name indicates, it is not a language, but a model for representing data about 'things on the web.' This type of data about data is called metadata. The 'things' are resources in RDF vocabulary. RDF's basic data model is simple: besides resources, it contains properties and statements. A property is a specific aspect, characteristic, attribute, or relation that describes a resource. A statement consists of a specific resource with a named property plus that property's value for that resource. This value can be another resource or a literal value, such as free text. Altogether, an RDF description is a list of triples: an object (a resource), an attribute (a property), and a value (a resource or free text).

Note:

People often feel that RDF and XML are the same. But the fact is that RDF is a data model which can be represented in XML (RDF/XML). RDF is a graph data model that makes use of URIs while XML is a tree data model and doesn't care about URIs.

5.1.4 RDF Schema

RDF Schema (RDFs) is an extension of RDF vocabulary to allow taxonomical description of classes and their properties. It also extends the definitions of some of the elements of RDF, for example it sets the domain and range of properties and relates the RDF classes and properties into taxonomies using the RDFS vocabulary.

5.2 RDF graph

RDF can be defined as a 2-Dimensional concept. It can be either perceived as a graph or as a set of statements. Let us begin with the former one.

Going by the classical definition, a graph is a representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by mathematical abstractions called vertices, and the links that connect some pairs of vertices are called edges.

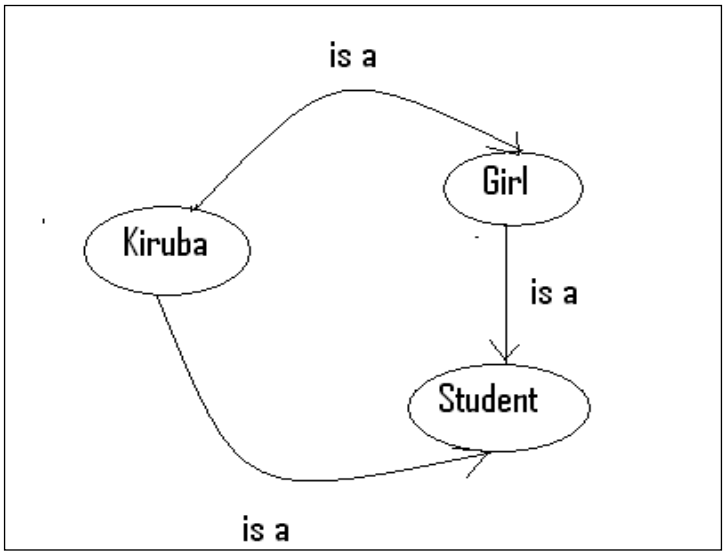

Figure 5.1: A simple graph

Refer figure 5.1. The name at the start of the arrow is the statement's subject, the name at the end of the arrow is the statement's object, and the name that labels the arrow is the predicate. RDF as a graph expresses exactly the same information as RDF written as triples, but the graph form makes it easier for us to see the structure in the data. Hence we can conclude:

Subject = (Kiruba, Girl)
Object = (Girl, Student)
Predicate = (is a)

5.2.1 RDF Basics

Resources

We can think of a resource as an object, a 'thing' we want to talk about. Resources may be authors, books, publishers, places, people, hotels, rooms, search queries, and so on. Every resource has a URI, a Uniform Resource Identifier. A URI can be a URL (Uniform Resource Locator, or Web address) or some other kind of unique identifier (note that an identifier does not necessarily enable access to a resource). URI schemes have been defined not only for web locations, but also for such diverse objects as telephone numbers, ISBN numbers, and geographic locations. There has been a long discussion about the nature of URIs, even touching philosophical questions (for example, what is an appropriate unique identifier for a person?), but we will not go into detail here. In general, we assume that a URI is the identifier of a Web resource.

Properties

Properties are a special kind of resources. They describe relations between resources, for example 'written by', 'age', 'title', and so on. Properties in RDF are also identified by URIs (and in practice by URLs). This idea of using URIs to identify 'things' and the relations between them is quite important.
This choice gives us in one stroke a global, worldwide, unique naming scheme. The use of such a scheme greatly reduces the homonym problem that has plagued distributed data representation until now.

Statements:

Statements assert the properties of resources. A statement is an object, attribute-value triple, consisting of a resource, a property, and a value. Values can either be resources or literals. Literals are atomic values (strings), the structure of which we do not discuss further.

5.3 Constructing RDF

RDF provides a general, flexible method to decompose any knowledge into small pieces, called triples, with some rules about the semantics (meaning) of those pieces. The foundation is laid by breaking the knowledge into basically what's called a labeled, directed graph, as discussed in the previous section.

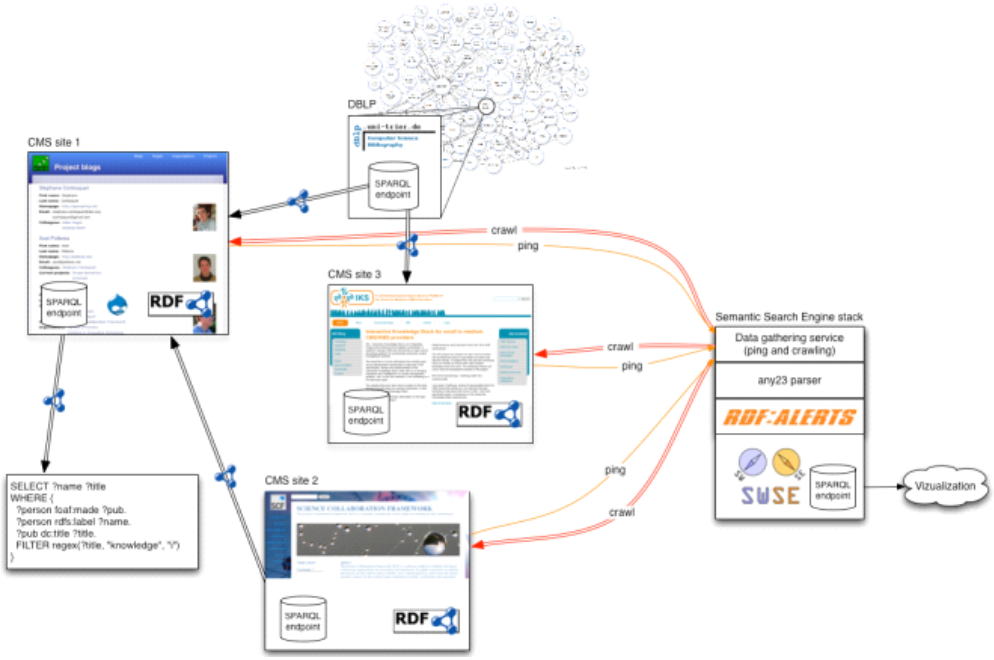

Figure 5.2: System implemented using RDF.

5.3.1 RDF Triples

Let us start with the standard syntax of RDF triples. Stated below is the standard syntax-

<rdf:Description rdf:about="subject">
<predicate rdf:resource="object" />
<predicate>literal value</predicate>
<rdf:Description>

In order to understand this concept properly, let us start with an example. Consider the following example,

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:feature="http://www.bookboon.com/cartoon-features#">
<rdf:Description rdf:about="http://www.bookboon.com/cartoon#mickey-mouse">
<feature:character rdf:resource="http://www.bookboon.com/character#funny"/>
</rdf:Description>
</rdf:RDF>
```

Explanation:

RDF triple

In example 1, the RDF between the <rdf: Description> tags is called an RDF statement, or an RDF triple. In RDF triple, as the name suggests, the statement is broken into three parts:

- Subject
- Object
- Predicate of the statement

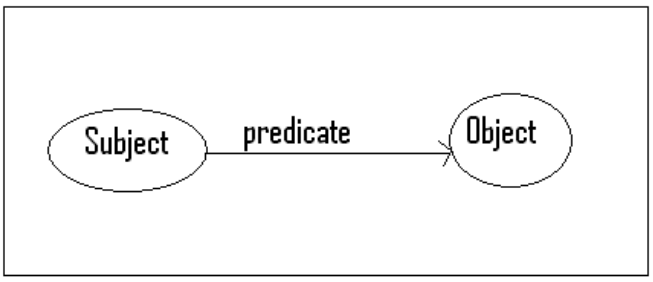

Figure 5.3: Graphical representation of triples

Refer example 1,

The namespace "**//www.w3.org/1999/02/22-rdf-syntax-ns#**" that we find, is the standard W3.org namespace. This namespace tells any machine reader that the enclosing document is an RDF document, and that the rdf:RDF tag resides in this namespace. This namespace, and the RDF node, form the root of all RDF documents.

Constructing a statement in RDF:

An RDF document can contain more than one statement. The rdf:Description tag describes the **subject** and gives it a unique identifier namely, **http://www.bookboon.com/cartoon#mickey-mouse**. Hence, here the subject is 'Mickey-mouse'.

Constructing a predicate in RDF:

RDF statements describe the characteristics of their subjects using properties, or predicates in RDF terminology. The subject has a property with name **feature:character** which has the literal value 'funny', i.e. the cartoon character Mickey-mouse has a funny characteristic. So, here the predicate is 'character'.

Constructing an object in RDF:

The **feature** i.e. character, in

"**<feature:character rdf:resource="http://www.bookboon.com/character#funny**"/>", is referring to the subject (ID) of another statement. It implies that objects in RDF can refer the subjects of other statements. The object here is 'funny'.

In graphical format Example 1 can be represented as follows:

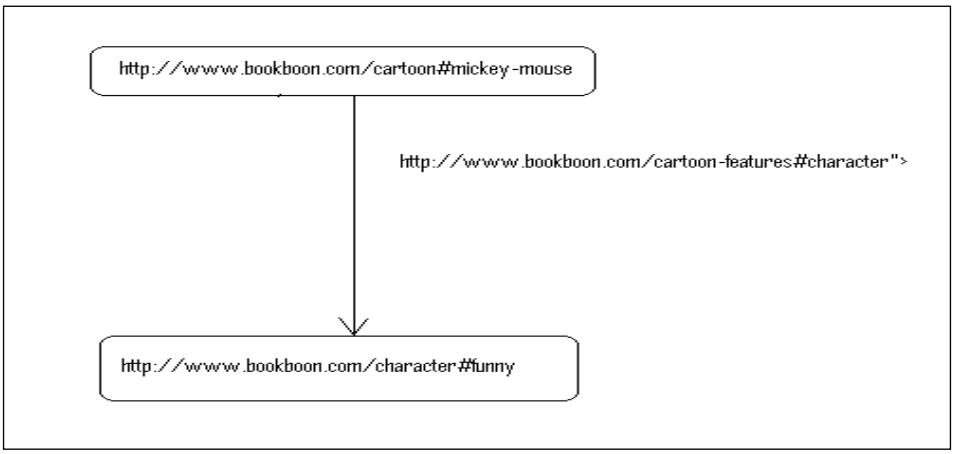

Figure 5.4: RDF-Graph for example 1

5.4 Introduction to SPARQL

SPARQL stands for 'SPARQL Protocol and RDF Query Language'. Like the tables of a relational database are queried using SQL, the triples of RDF data are queried using SPARQL. SPARQL is a query language designed specifically to query RDF databases. SPARQL queries are sent from a client to a service known as a SPARQL endpoint using the HTTP protocol. The interaction between the client and the endpoint is defined in a machine-friendly protocol that is not intended to be interpreted by humans, so use of SPARQL requires an interface that allows the user to enter the queries and to display the results in a meaningful way. As with traditional database languages such as SQL, interfaces are commonly constructed so that the queries are constructed and launched through forms that do not require the human user to have any knowledge of RDF and SPARQL.

In addition to the language, W3C has also defined:

- The **SPARQL Protocol** for RDF specification: it defines the remote protocol for issuing SPARQL queries and receiving the results.
- The **SPARQL Query Results XML** Format specification: it defines an XML document format for representing the results of SPARQL queries.

5.5 SQL

SQL stands for Structured Query Language. SQL is a computer language for storing, manipulating and retrieving data stored in relational database. A Computer engineering student will find this term too familiar. This is because, most of the existing applications use SQL to query database. SQL is the standard language for a Relational Database System. All relational database management systems like MySQL, MS Access, Oracle, Sybase, Informix, postgres and SQL Server use SQL as the standard database language. SQL is followed by a unique set of rules and guidelines called Syntax. All the SQL statements start with a keyword like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and the entire statements end with a semicolon.

Let us discuss some of the important keywords in SQL that will be useful in understanding SPARQL:

CREATE

Creating a basic table involves naming the table and defining its columns and each column's data type. The SQL **CREATE TABLE** statement is used to create a new table.

Example:

```
Create table cust (

Emp_no int not null,

Name varchar(25)

);
```

SELECT

SQL **SELECT** statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

Example:

```
Select * from customer_information;
```

This will retrieve all the data from the table "customer_information".

INSERT

SQL **INSERT** statement is used to insert the data into a database table.

Example:

Insert into customer_information values (A, B, C);

This will insert the data into the table "customer_information".

SPARQL Vs SQL

Many people ask what can be done using SPARQL that can't be done using SQL. Both, SQL and SPARQL, give access to the user to create, combine, and consume structured data. SQL does this by accessing tables in relational databases, and SPARQL does this by accessing a web of Linked Data. RDF captures both entity attributes and relationships between entities, as statements of the form '**entity1** has **property A** relationship to **entity2'** using a language called Turtle. We can say that there is a person named "Mickey" with an address in Disney, USA, as follows:

<PersonA>a<Person>.
<AddressB>a<Address>.
<PersonA><Person#fname>"Mickey".
<AddressB><Address#city>"Disney".
<PersonA><Person#addr><AddressB>.
<AddressB> <Address#state> "USA" .

Also, we can have another person, "Pluto", but we won't say anything about her address because we don't know it. Hence, we represent as follows:

<PersonF>a<Person>.
<Person> <Person#fname> "Pluto".

The terms used in the statements above are relative URLs in angle brackets (<>s), literals in quotation marks (""s) and the keyword 'a' which is just a shortcut for the URL used to identify a 'has type' relationship. There is no concept in RDF corresponding to SQL's NULL as there is no RDF requirement corresponding to SQL's structural constraint that every row in a relational database must conform to the same schema. The object of one assertion, e.g.<AddressB> above, may be the subject or object of other assertions. In this way, a set of RDF statements connects up to create a "graph" (in the mathematical sense). You will frequently hear the term 'RDF graph'. These graphs may be cyclic, for example, stating that Mickey lives in someplace, for which, he is also the owner:

<PersonC><Person#homeAddress><AddressK>.
<PerconC><Person#fname>"Mickey".
<AddressK><Address#owner><PersonC>.

The examples above illustrate some of the structural similarities and differences between RDF and relational data. A core philosophical difference is that RDF is a post-Web language; that is, it allows one to use web identifiers for the entities we want to describe, and for the attributes and relationships we use to describe them. If I trust the publishers not to lie to me, I can merge information from different parties trivially.

Finally, consider the following example:

- SELECT Person.fname, Address.city FROM Person, Address WHERE Person.addr=Address.ID AND Address.state="Mumbai"

Conceptually, we are selecting a list of attributes from a set of tables, where certain constraints are met. These constraints capture the relationships implicit in the scheme, Person.addr=Addresses.ID, and the selection criteria, e.g. Address.state="Mumbai".

A SPARQL query of the same data could look like –

```
SELECT?name?city
WHERE{

?who<Person#fname>?name;
<Person#addr>?adr.
?adr<Address#city>?city;
 <Address#state>“Mumbai”
}
```

For better or worse, SPARQL reuses some key words familiar to SQL users: SELECT, FROM, WHERE, UNION, GROUP BY, HAVING and most aggregate function names.

5.6 Constructing a SPARQL query

The SPARQL query language is based on matching graph patterns. The simplest graph pattern is the triple pattern, which is like an RDF triple but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions. Combining triple patterns give a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern.

As a simple example, consider the following query:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
?c rdf:type rdfs:Class .
}
```

This query retrieves all triple patterns where the property is rdf:type and the object is rdfs:Class. In other words, this query when executed will retrieve all classes.

Format for constructing a SPARQL query:

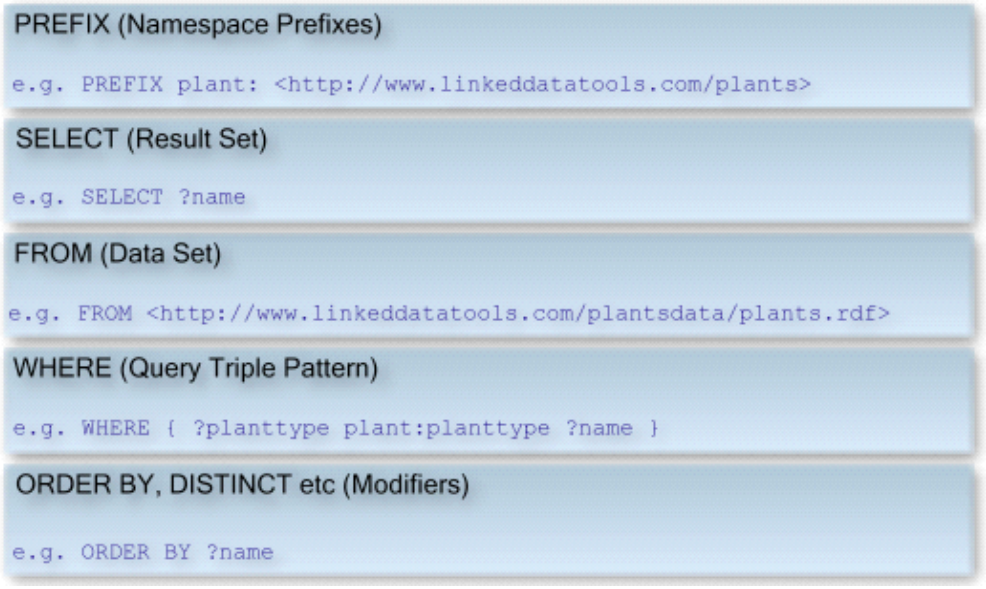

Figure 5.1: Format of SQL query

Example:

PREFIX school-info: <http://education.data.gov.india/def/school/>

SELECT ?name WHERE {

?school a school-info:School.

?school school-info:establishmentName ?name.

?school school-info:districtAdministrative <http://statistics.data.gov.uk/id/local-authority-district/Mumbai>.

}

ORDER BY ?name

This query, when executed, will return the names of all the schools in India, in administrative district "Mumbai", and orders the results in alphabetical order.

The familiarity with SQL will be very useful in understanding the format and construction of SPARQL. The **SELECT** statement requests the variable **?name** to be returned. **?name** returns all the names of the schools which match the three search patterns given in the query.