

# Introducing Linked Data And The Semantic Web

<<http://www.linkeddatatools.com/semantic-web-basics>>

Linked Data와 Semantic Web이란 무엇인가? 원칙적으로 Semantic Web은 Web 3.0 이다; 즉, 시스템들 또는 엔티티(entities) 끼리 서로 데이터를 링크하는 방법이다. 따라서 웹 환경에서 이것을 사용하여 데이터끼리 풍부하고, 자아-기술적(self-describing)인 관계성(interrelations)을 갖는다.

사실, SW는 HTML 문서에 포함되어 있는 데이터를 사람이 읽는 것이 아니라, 컴퓨터가 읽을 수 있어야 한다는 사고의 전환에서 비롯되었다. 다시 말해서, 컴퓨터가 인간을 위해 좀 더 많은 사고적(thinking) 업무를 수행할 수 있어야 한다는 생각에서 출발하였다.

## 1) Semantic search란?

Seeks to improve search accuracy by understanding the searcher's intent and the contextual meaning of terms as they appear in the searchable dataspace, whether on the Web or within a closed system, to generate more relevant results. Semantic search systems consider various points including context of search, location, intent, variation of words, synonyms, generalized and specialized queries, concept matching and natural language queries to provide relevant search results. Major web search engines like Google and Bing incorporate some elements of semantic search.

Guha et al. distinguish two major forms of search: navigational and research. In navigational search, the user is using the search engine as a navigation tool to navigate to a particular intended document. Semantic search is not applicable to navigational searches. In research search, the user provides the search engine with a phrase which is intended to denote an object about which the user is trying to gather/research information. There is no particular document which the user knows about and is trying to get to. Rather, the user is trying to locate a number of documents which together will provide the desired information. Semantic search lends itself well with this approach that is closely related with exploratory search.

## How Does It Differ From The Web As It Is Today?

오늘날 우리가 웹에서 얻는 많은 데이터는 웹 페이지의 형태로 우리에게 전달되는데, 이것들은 HTML 문서들이며, 서로 하이퍼링크로 연결되어 있다. 사람이나 기계 모두 이 같은 문서를 읽을 수 있다. 그렇지만, 사람은 전형적으로 페이지에서 키워드를 찾을 수 있지만, 기계가 이 문서에 들어 있는 의미를 스스로 발췌하기란 결코 쉽지 않다.

## Enter Linked Data - Liberating Web Databases From Their Old Chains

웹에는 많은 정보가 들어 있지만, 전형적으로 raw data 그 자체를 이용할 수는 없다. 만일 데이터베이스의 웹 사이트가 만들어 한다면, 그것의 HTML 문서들은 해당 데이터로만만 들어져야 한다.

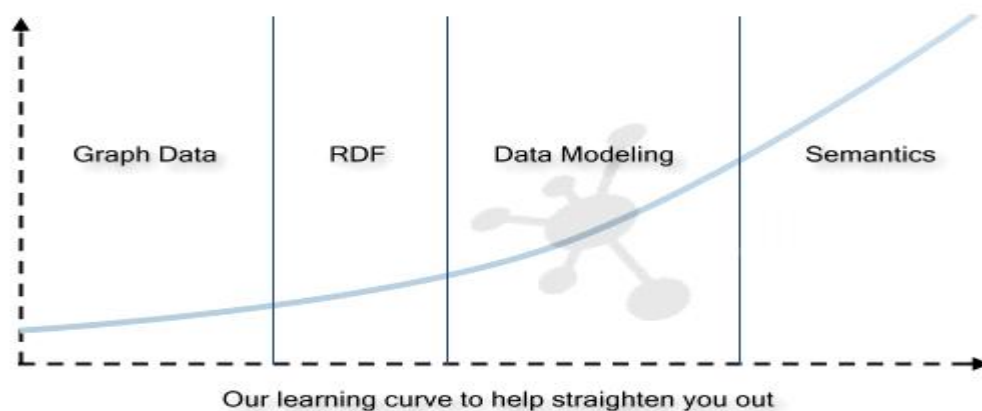
따라서 SW은 이 같은 문제를 해결하기 위하여, 다음과 같은 여러 가지 방법으로 현재의 인터넷 모습을 변화시키고 있다:

- 인공지능처리를 위하여 web of data를 개방하고 있다.(웹으로 하여금 우리를 위해 약간 좀 생각하도록 한다.)
- 회사, 조직, 개인들로 하여금 공개된 표준 포맷으로 자유롭게 자신들의 데이터를 출판하도록 한다.
- 기업에서 이미 웹에 있는 데이터를 이용할 수 있도록 한다(데이터 주고받기)

실제로, SW는 여러 곳에서 출판된 모든 HTML 정보를 수집한 다음, 그것을 마치 전에는 하나의 데이터베이스였던 것처럼, 다루고 조사하는 데이터 모델의 한 유형이다.

오늘날의 다양한 관련 도구와 소프트웨어를 비교해 볼 때, 이것이 인터넷을 사용하는 모든 data humanity 분야의 자동화 연구에 끼치는 이익은 엄청나다.

**But Where Do I Start?**



이 강의는 복잡하지 않다. 우리는 초보자의 시작을 돕기 위하여 개론으로 시작한 다음에, SW를 정의하고, SW과 현재의 웹과의 차이를 알아보며, 마지막으로 SW의 제작 기술에 대하여 깊이 있게 알아볼 것이다:

- ▶ Tutorial 1 Introduction To Graph Databases - gives a brief overview of the way in which the semantic web stores data.
- ▶ Tutorial 2 RDF - A Quick Start - an introductory look at Resource Description Framework (RDF), the format the semantic web uses to store data in graph databases.

- ▶ Tutorial 3 Semantic Modeling - introduces the key aspects of describing data with meaning, or semantics - and the tremendous advantages this can offer.
- ▶ Tutorial 4 Introduction To RDFS & OWL - the key syntax the semantic web uses to encode semantic meaning into data.
- ▶ Tutorial 5 Querying Semantic Data - how to query published semantic data using SPARQL protocol - the means to harness the immense discovery capabilities of the semantic web.

## Tutorial 1: Introducing Graph Data

SW은 그렇게 잘 알려진 분야가 아니다. 초보자가 SW의 정의와 그것의 작동원리를 열심히 이해하려 한다면, 먼저 그것의 데이터 저장방법부터 이해해야 한다. 따라서 먼저, SW의 데이터 저장 모델인 graph database부터 알아보다.

After this tutorial, you should be able to:

- Describe in basic terms what the semantic web is.
- Experience the paradigm-shift of storing information as a graph database, rather than a hierarchical or relational database.
- Understand that the semantic web of data is defined using Resource Description Framework (RDF).
- Understand the basic principles of RDF statements and how they can define data graphs.

여러분이 전통적인 IT 분야에 지식을 갖고 있고, 계층형(for example XML) 또는 관계형 데이터베이스(for example MySQL, MS SQL)의 데이터저장방법에 대하여 알고 있겠지만, Resource Description Framework(RDF)에 대해선 잘 알지 못할 수도 있다.

RDF란 SW 커뮤니티에서 사용하는 일반적인 두문자어이며, 시멘틱 데이터의 웹을 제작하는데 필요한 기본적인 빌딩블록들 중의 한 요소이다. 또한 이것이 define(정의)하는 것은 여러분이 익숙하지 않은 데이터베이스의 유형인 **graph database** 이다.

2) define이란?

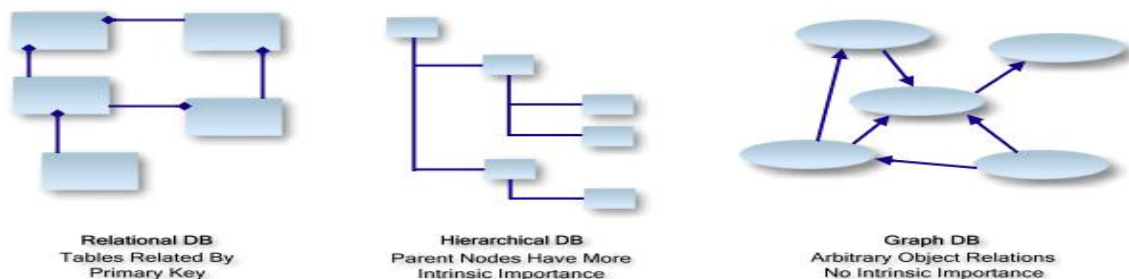
- To state the precise meaning of (a word or sense of a word, for example).
- To describe the nature or basic qualities of: explain:

비록 이 데이터베이스에 대해 여러분이 생소하더라도, 이것이 바로 전 세계적으로 SW을 제작하는데 사용하는 데이터베이스의 한 부류 이다.

이제 graph database가 무엇이고, 어떻게 RDF가 그것을 정의하는지, 그리고 graph database를 시각화하는 방법이 무엇인지에 대하여 알아보기로 한다.

먼저 hierarchical, relational, 그리고 graph databases를 비교하여 이것들이 서로 어떻게 다른지를 살펴보기로 하자:

## 1.1 Introducing The Graph Database



대부분의 데이터 저장 유형들에서는, 자신들의 여러 가지 데이터 요소(elements) 중에는 다른 것보다 보다 더 많은 precedence나 importance를 갖고 있는 몇 가지의 요소들(예를 들어, data nodes 또는 data tables)에 대한 개념을 정의하고 있다.

예를 들어, XML 다큐먼트를 보자. 전형적으로 XML 다큐먼트에는 한 개의 parent node와 함께 여러 개의 정보 노드를 사용하고 있다. 따라서 이러한 다큐먼트의 root는 최상위의 노드이며, 이 노드는 어떠한 부모 노드도 갖지 않는다.

위의 그림을 살펴보자. 맨 오른쪽의 data graph에는 어떠한 roots(또는 계층)도 존재하지 않으며, 서로 연결된 자원들로만 구성되어 있다. 또한 이 그래프의 어떠한 자원도 특별하게 다른 자원보다 본질적으로(intrinsic) 더 중요하지 않다는 것을 나타내고 있다.

### An Example Of A Data Graph

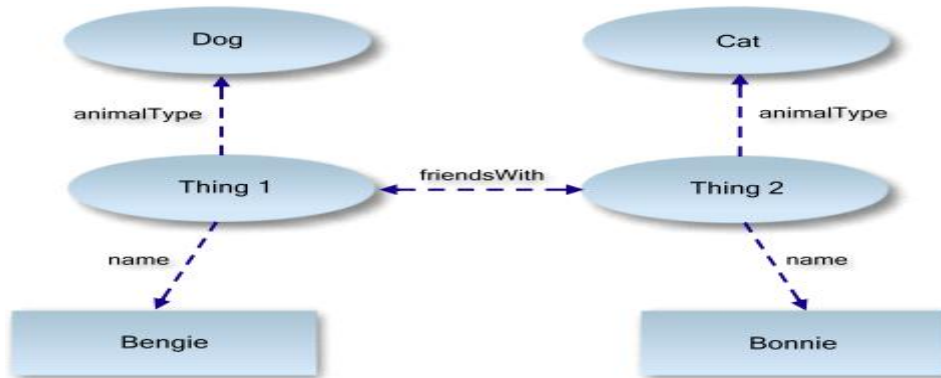
사물들(things)이 서로 어떤 관계를 갖고 있는지를 기술하고 있는 지시문(statements)과 그 속에 있는 관계들을 RDF로 표현하는 방법을 알아보기에 앞서서, 먼저 그래프로 이들 사물간의 관계를 시각화해 보자. 이것은 매우 쉽다.

먼저 아래의 개(벤지)와 고양이(보니) 간의 관계를 설명하는 지시문을 살펴보자:

- Bengie는 개다.
- Bonnie는 양이다.

■Bengie와 Bonnie는 친구이다.

위의 간단한 3개의 지시문을 데이터 그래프로 그려보면, 아래와 같다:



위의 그래프에 나타나 있는 관계는 매우 직관적이므로, 우리는 이 관계들을 완전하게 이해할 수 있다: 즉, 우리는 "Thing 1" 과 "Thing 2" 라는 두 개의 사물이 있고, 이 사물들의 속성 이름이 하나는 "animalType"이고 나머지는 "friendsWith"라는 것을 알 수 있다.

또한, 우리는 "Thing 1"의 속성 "name"의 값이 "Bengie"이고, "Thing 2"의 속성 "name"의 값이 "Bonnie"라는 것도 알 수 있고, 물론 "Thing 1"은 개를, "Thing 2"는 고양이를 의미한다는 것도 알 수 있다. 끝으로, 이 두 개의 사물은 서로 친구(속성 "friendsWith"가 화살표로 양 방향 모두를 가리키고 있음)라는 것도 알 수 있다.

**핵심 포인트** - 위의 그래프에서 화살표는 속성(properties)을 나타낸다: 때론, 이것을 RDF 용어(terminology)로는 predicates라 한다. 이 두 용어는 호환적으로 사용되며, 그래프에서 표시된 화살표는 속성을 의미한다.

#### <Graph Model의 예: 지시문>

[http://dbpedia.org/resource/Billie\\_Jean](http://dbpedia.org/resource/Billie_Jean) has a **singer** whose value is **Michael Jackson**

- ▶ Subject: [http://dbpedia.org/resource/Billie\\_Jean](http://dbpedia.org/resource/Billie_Jean) (URI)
- ▶ Predicate: <http://www.example.com/terms/singer> (URI)
- ▶ Object: Michael\_Jackson (Literal)

#### <URI란?>

정보기술에서, Uniform Resource Identifier (URI)는 자원을 식별하기 위하여 사용되는 문자열(a string of characters)이다. 이러한 식별 기능을 통하여 WWW과 같은 네트워크에 있는 자원의 표현물(representations) 간의 상호작용이 이루어질 수 있다. 가장 잘 알려진 URI의 일반적 형태는 종종 비공식적으로 웹 어드레스로 불리우는 Uniform Resource Locator (URL)이다. 또한 거의 사용되지는 않고 있는 Uniform Resource Name (URN)이 있는데, 이것은 특별한 namespaces에 있는 자원들을 식별하기 위한 메커니즘을 제공함으로써 URLs을 보완하도록 다

자인된 것이다.

예를 들어, Uniform Resource Name (URN)을 사람의 이름이라면, Uniform Resource Locator (URL)은 그들이 사는 거리의 어드레스라 비유할 수 있다. 다시 말해서, URN은 아이템을 식별하기 위한 것이고, URL은 그것을 찾는 방법을 제공하는 것이다.

공식적으로 RDF에 대해 알아보기 전에, 다음과 같은 간단한 예를 먼저 맛보기로 하자:

### 1.3 A Starting Example Of RDF

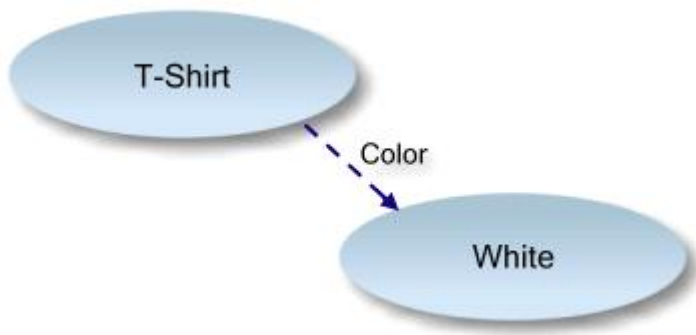
```
01. <?xml version="1.0" encoding="UTF-8"?>
02.
03. <rdf:RDF
04.     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.     xmlns:dc="http://purl.org/dc/elements/1.1/"
06.     xmlns:region="http://www.country-regions.fake/">
07.
08.   <rdf:Description rdf:about="http://en.wikipedia.org/wiki/Oxford">
09.     <dc:title>Oxford</dc:title>
10.     <dc:coverage>Oxfordshire</dc:coverage>
11.     <dc:publisher>Wikipedia</dc:publisher>
12.     <region:population>10000</region:population>
13.     <region:principaltown rdf:resource="http://www.country-regions.fake/oxford"/>
14.   </rdf:Description>
15.
16. </rdf:RDF>
```

지금 위의 내용에 대해 걱정하지 마라. 우리는 나중에 이것을 다시 검토할 것이다. 지금은 단지 위와 같은 것이 RDF/XML - the XML form of RDF라는 것만을 이해하면 된다. RDF를 레코딩하는 여러 가지 방법들이 있지만, 우리는 단지 RDF/XML만을 살펴볼 것이다.

### 1.3 The RDF Statement (Triple)

위의 콘텐츠에서 붉은 색으로 표시한 RDF/XML(<rdf:Description> tags 사이)를 RDF 지시문, 또는 때때로 RDF triple이라 부른다. 이 두 가지 이름 중에서 triple이 우리가 RDF를 이해하는데 가장 도움이 되는 용어인데, 그 이유는 이것이 지시문을 3가지의 요소로 지시문이 구성되어 있다는 것을 의미하기 때문이다: 즉, 지시문의 subject, predicate, 그리고 object.

그래프의 형태로 이 3가지의 구성 용어들을 나타내 보자. 예를 들어, 티-셔츠의 색깔을 표현하고 있는 데이터에 대한 다음의 그래프를 살펴보자:



위의 간단한 그래프에서는 3개의 구성요소가 표현되어 있다:

- Subject: T-shirt;
- Predicate(또는 property): color;
- Object: white.

**핵심 포인트** - RDF는 SW의 데이터 구조를 정의하는 토대이지만, 데이터에 숨어있는 어의나 의미를 스스로 기술하지는 못한다. 이것은 나중에 RDFS (RDF Schema) 와 OWL (Web Ontology Language)을 배울 때 다루기로 한다. 지금은 이것들에 대하여 걱정하지 마라. 먼저, RDF가 데이터간의 관계구축방법이 이미 잘 알려져 있는 기존의 데이터저장방법과 어떻게 다른지에 대해 배워야 한다. 또한 여러분은 계층형이나 관계형 데이터 모델에서 벗어나 graph model로 추세가 바뀌고 있음을 깨닫는 것이 중요하다.

간단한 RDF/XML 지시문을 통해, 이 구성요소들에 대하여 알아보자:

```
01 .<?xml version="1.0" encoding="UTF-8"?>
02.
03. <rdf:RDF
04.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.   xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
06.
07.   <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
08.
09.     <feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
10.
11.   </rdf:Description>
12.
13. </rdf:RDF>
```

공식적으로 우리는 이 지시문을 다음 강의에서 분석하겠지만, 여러분은 먼저 위의 RDF에서 subject, predicate, object를 어떻게 기술하는지에 대하여 감을 잡아야 한다.

이제 이장을 마감한다. 이제 여러분은 다음과 같은 것을 이해하고 있을 것이다:

- What a data graph is.
- That the semantic web is a giant, global data graph defined in RDF (Resource Description Framework).
- The all-important shift in thinking from storing data in relational, or hierarchical models to a storing in graph models.
- The subject, predicate and object in terms of basic data graphs and RDF statements.
- A basic familiarity with the layout of an RDF document.

## Tutorial 2: Introducing RDF/XML

graph data model이 SW에서 데이터를 저장하는 모델이라면, RDF는 그것을 만드는 포맷이다. 앞 장에서, 우리는 graph data와 RDF를 살펴봤다. 이제 우리는 RDF/XML - 웹에서 가장 인기 있는 RDF 포맷들 중의 하나 - 을 사용하여 그래프 데이터를 작성하는데 필요한 기본 개념을 설명할 것이다.

이미 여러분은 graph database의 개념을 살펴봤고, 계층형과 관계형 데이터 모델과 같은 전통적인 데이터 저장 형태와 이것을 비교도 해 보았다.

이제 간단하게 RDF (Resource format)을 살펴보고, 주체(subject), 술어(predicate 또는 property), 그리고 객체(object)로 이루어진 지시문을 어떻게 정의하는지에 대해서도 알아봤으며, subject->predicate->object relationship를 triple이라 부른다는 것도 알았다. 또한 여러분은 RDF가 시멘틱 데이터의 웹을 구축하는 기본 포맷이라는 것도 배웠다.

이번 강좌에서, 여러분은 한 단계씩 여러분 자신의 RDF 지시문을 구축하는 방법을 배울 것이고, 그래프를 사용하여 그것들을 시각적으로 이해할 수 있게 될 것이다. 그런 다음에, 데이터에 어의나 의미를 추가하는 것에 대해 생각해 볼 것이고, 끝으로 이것이 제공하는 커다란 장점에 대해서도 이해하게 될 것이다.

위의 내용을 가장 잘 수행할 수 있도록, 먼저 간단한 RDF 문서의 예를 가지고 한 단계씩 알아보기로 한다.

### 2.1 Building An RDF document

#### >Add The RDF document Root Tag



먼저, RDF root node를 다음과 같이 추가해 보자:

```
1. <rdf:RDF
2.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
3.
4.   <!-- Body Code Omitted -->
5.
6. </rdf:RDF>
```

위의 예에 있는 두 번째 줄에서, 여러분은 표준화된 W3.org namespace인

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

라는 URI를 보게 될 것이다. 이 namespace는 다른 컴퓨터 리더(reader)에게 이 태그로 봉해져 있는(enclosing) 다큐먼트가 RDF 다큐먼트라는 것과 여기서 사용된 rdf:RDF tag들이 이 namespace에 포함되어 있다는 것을 알려주는 것이다.

1) 컴퓨팅에서 namespace란 여러 종류의 사물을 조직하기 위하여 사용되는 심볼의 세트를 말한다. 따라서 이러한 사물들은 이름으로 참조(referred to)될 수 있다: 잘 알려진 예로는 다음과 같은 것이 있다:

- file systems은 파일들에 이름을 할당한 namespaces 이다;
- programming languages는 namespaces에 있는 자신들의 변수와 하위-루틴을 조직한다;
- 컴퓨터 네트워크와 분산 시스템은 computers, printers, websites, (remote) files, 등과 같은 자원에 이름을 할당한다.

Namespaces는 일반적으로 다른 환경에서도 그 이름을 재사용할 수 있도록 계층구조를 갖는다. 하나의 추론으로, 각각의 사람은 올바른 이름뿐만 아니라 자신들의 친척과 공유하고 있는 가족이름을 갖고 있는 인명 시스템을 생각해 보자. 만일, 각 가족에서 가족 이름의 고유한 것이라면, 각각의 사람은 이름과 가족이름의 결합에 의해 유일하게 식별될 수 있다. 다시 말해서, 비록 많은 Janes이 있더라도 Jane Doe는 단지 한명 뿐이다: Doe라는 성의 namespaces에서, 단지 "Jane"은 이 사람을 확실하게 설명하는데 충분하지만, 모든 사람의 "global" namespace에서는 full name이 사용되어야만 한다.

비슷한 방법으로, 계층적 파일 시스템은 디렉토리로 파일을 조직한다. 각 디렉토리는 하나의 독립된 namespace이다. 그러므로 디렉토리 "letters" 와 "invoices"는 둘 다 "to\_jane" 파일을 포함할 수도 있다.

컴퓨터 프로그래밍에서, namespaces는 전형적으로 특별한 기능성과 관련된 symbols 과 identifiers를 그룹핑할 목적으로, 그리고 동일한 이름을 공유하는 복수의 identifiers 간의 충돌을 피할 목적으로 사용된다.

네트워킹에서, **Domain Name System**은 websites와 기타 자원들을 namespaces로 조직화 한다. 예를 들어, "org"는 비영리기관용 namespace이다. 예를 들어, "wikipedia.org"는 Wikimedia Foundation에 할당된 하위 namespace이며, "en.wikipedia.org"는 이 스페이스에 있는 영어판 위키피디어의 이름이다.

위의 다큐먼트에서 namespace를 표현하고 있는 RDF node가 바로 이 RDF 다큐먼트의 roots node이다.

>Add A Statement

RDF 문서에는 하나 이상의 지시문이 포함될 수 있다. 간단하게 우리는 하나를 추가할 것이다. RDF/XML에서 주체(subject)를 정의하는 방법은 <rdf:Description> tag를 사용하는 것이다. 다음과 같이 붉은 색 지시문을 추가해 보자:

```
01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.
04.   <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
05.
06.       <!-- Statement Code Omitted -->
07.
08.   </rdf:Description>
09.
10. </rdf:RDF>
```

위에서 붉은 색으로 표현된 <rdf:Description> tag가 의미하는 것은 간단하게 말해서 다음과 같다:

"나는 객체 t-shirt에 대하여(about) 정의(describe)하며, 그것의 유일한 ID는 <http://www.linkeddatatools.com/clothes#t-shirt> 이다."

## 주목!!!

<rdf:Description>은 about 속성에 의해 지정된 resource에 대한 URI 정보를 갖고 있는 container 이다. 다음의 예에서 각각의 resources는 books이고, 그것의 식별자(URI)는 서명이다. 그리고 각각의 책에는 한명의 저자와 페이지 수만 표현되어 있다.

## Source

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:lib="http://www.zvon.org/library">

  <rdf:Description about="Matilda">
    <lib:creator>Roald Dahl</lib:creator>
    <lib:pages>240</lib:pages>
  </rdf:Description>

  !!! 위의 <rdf:Description />의 또 다른 표기법:
  <rdf:Description about="Matilda" lib:creator = "Roald Dahl" lib:pages="240" />

  <rdf:Description about="The BFG">
    <lib:creator>Roald Dahl</lib:creator>
    <lib:pages>208</lib:pages>
  </rdf:Description>
```

```

<rdf:Description about="Heart of Darkness">
  <lib:creator>Joseph Conrad</lib:creator>
  <lib:pages>110</lib:pages>
</rdf:Description>

<rdf:Description about="Lord Jim">
  <lib:creator>Joseph Conrad</lib:creator>
  <lib:pages>314</lib:pages>
</rdf:Description>

<rdf:Description about="The Secret Agent">
  <lib:creator>Joseph Conrad</lib:creator>
  <lib:pages>249</lib:pages>
</rdf:Description>
</rdf:RDF>

```

## Output

Author	Title	Pages
Roald Dahl	Matilda	240
Roald Dahl	The BFG	208
Joseph Conrad	Heart of Darkness	110
Joseph Conrad	Lord Jim	314
Joseph Conrad	The Secret Agent	249

좀 이해가 됐나요? 계속 배워봅시다.

## >Add Predicates

여러분이 주체에 대해 정의하면서 그것의 고유한 ID를 지정해 놓았지만, 그 주체의 특성에 대해서는 어떠한 것도 정의하지 않았다. RDF 지시문에서는 RDF 전문용어로 속성을 의미하는 properties나 predicates를 사용하여 해당 주체의 특성들을 정의한다.

간단하게, T-shirt의 속성 중 하나인 size를 다음과 같이 추가해 보자:

```

01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.   xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.   <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.     <feature:size>12</feature:size>
08.
09.   </rdf:Description>
10.
11. </rdf:RDF>

```

위의 7번째 줄을 보자. 간단하게 말해서 이 주체는 <feature:size> 태그이름으로 기술된 한 개의 속성을 갖고 있으며, 이 속성의 문자 값은 12이다. RDF 전문용어로 이것이 바로 지시문(statement)이다.

그리고 3번째 줄에 이 속성이름뿐만 아니라 이 객체에서 사용되는 속성이름의 namespace에 대한 URI가 표시되어 있다는 것을 확인하라.

끝으로, T-shirt의 색깔인 color 속성을 하나 더 다음과 같이 추가해 보자:

```

01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.   xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.   <rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.     <feature:size>12</feature:size>
08.     <feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
09.
10.   </rdf:Description>
11.
12. </rdf:RDF>

```

이 <feature:color> 속성의 표현은 <feature:size>와 다르다는 것을 알았을 것이다. 지난 번 속성에서는 문자 값 12가 있었던 반면에, 이번 것에는 또 다른 지시문의 subject(ID)를 참조 하도록 지정해 놓았다. 이러한 표현이 옳은 것이다. RDF에서 object는 다른 지시문에 있는 subject를 참조할 수 있다.

!!! 속성 **rdf:resource**는 다른 **rdf:Description element**에서 정의하고 있는 resource에 about한 정보를 입력하는데 사용될 수 있다.

#### Book Sample

```

<rdf:Description about="RD">
<lib:firstName>Roald</lib:firstName>
<lib:surname>Dahl</lib:surname>

```

```
</rdf:Description>
```

```
<rdf:Description about="JC">  
<lib:firstName>Joseph</lib:firstName>  
<lib:surname>Conrad</lib:surname>  
</rdf:Description>
```

```
<rdf:Description about="Matilda">  
<lib:creator rdf:resource='RD' />  
<lib:pages>240</lib:pages>  
</rdf:Description>
```

```
<rdf:Description about="The BFG">  
<lib:creator rdf:resource='RD' />  
<lib:pages>208</lib:pages>  
</rdf:Description>
```

```
<rdf:Description about="Heart of Darkness">  
<lib:creator rdf:resource='JC' />  
<lib:pages>110</lib:pages>  
</rdf:Description>
```

```
<rdf:Description about="Lord Jim">  
<lib:creator rdf:resource='JC' />  
<lib:pages>314</lib:pages>  
</rdf:Description>
```

다시 본론으로 돌아가서, <feature:color>의 속성은 3번째 줄에 있는 xmlns:feature에 이미 포함되어 있다는 것도 이해하여야 한다. 다시 말해서, xmlns:feature의 이름리스트에는 이미 size와 color라는 속성이름이 포함되어 있다는 것이다.

!!! RDF 다큐먼트에 있는 주체 또한 다른 RDF 지시문에서 속성의 객체처럼 참조될 수 있다. 이것은 RDF 초보자들에게는 개념적으로 혼란스러울 수 있다.

따라서 이것은 간단하게 "이 주체는 ID가 <http://www.linkeddatatools.com/colors#white>인 지시문을 참조하고 있는 객체와 더불어 feature:color이라는 이름의 한 개의 속성을 가지고 있다"라고 말하고 있다.

## 2.2 Breaking Down The Statement

이제 간단한 예의 RDF 다큐먼트를 살펴보고, 우리가 배운 것을 근거로 지시문의 구성부분을 분석해 보자:

1. <rdf:Description rdf:about="subject">
2.     <predicate rdf:resource="object" />
3.     <predicate>literal value</predicate>

#### 4. <rdf:Description>

이미 여러분은 지시문의 주체(what the statement is about), 그리고 두 가지 형태의 속성 (다른 RDF 지시문을 참고하도록 하는 resources 그리고 문자값과 예 대하여 알게 되었다.

주목 - `rdf:Description` element는 단일 container 안에 하나 이상의 지시문을 집단화할 수 있게 허용하고 있다. 위와 같은 일반적인 형태에는 사실상 한 개의 속성과 한 개의 객체 즉, literal과 resource, 두 가지가 함께 동일한 주체를 참조하는 두 개의 지시문이 포함되어 있다.

### 2.3 A More Thorough Example

앞에서 그래프 데이터에서 다루었던 보다 복잡한 예로 되돌아가 보자:

```
01.<?xml version="1.0" encoding="UTF-8"?>
02.
03.<rdf:RDF
04.xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
05.xmlns:dc="http://purl.org/dc/elements/1.1/"
06.xmlns:region="http://www.country-regions.fake/">
07.
08.<rdf:Description rdf:about="http://en.wikipedia.org/wiki/Oxford">
09.<dc:title>Oxford</dc:title>
10.<dc:coverage>Oxfordshire</dc:coverage>
11.<dc:publisher>Wikipedia</dc:publisher>
12.<region:population>10000</region:population>
13.<region:principaltown rdf:resource="http://www.country-regions.fake/oxford"/>
14.</rdf:Description>
15.
16.</rdf:RDF>
```

여러분의 이해력을 테스트하고 보완할 분야를 알기 위하여, 위의 RDF 다큐먼트를 아래와 같이 구분할 수 있는지를 스스로 확인해 보라:

- Subject of the statement?
- Predicates of the statement(including whether they are resources or literals)?
- Objects referenced by the **resource** predicates?

이제 RDF 다큐먼트s에 대해 이해했고, 그것이 데이터 그래프와 어떤 관계가 있는지를 알았다면, 여러분은 RDF graph data로 시맨틱즈를 모델링하는 방법에 대하여 알 준비가 끝난 것이다.

## 2.4 A Quick Recap Of URIs And XML Namespaces

앞에서 사용된 `http://www.linkeddatatools.com/clothes#t-shirt`처럼 우리가 이제까지 사용해 왔던 고유한 IDs를 Uniform Resource Identifiers, 또는 줄여서 URIs라 부른다. 우리는 이제까지 아무런 설명없이 지시문의 주체, 속성, 객체에 고유한 IDs를 제공하기 위하여 URIs를 사용해 왔다.

URIs는 전 세계적으로 데이터 교환을 가능하게 만들자는 RDF의 목적을 달성하는데 너무나 중요하기 때문에, 우리는 이제 신속하게 URIs에 대해 다시 살펴보자.

### >XML Namespace URIs

앞에 있던 RDF 다크먼트 예로 다시 가 보자. T-shirt size 속성은 아래의 7번째 줄에서 `<feature:size>`란 이름을 가지고 있다는 것을 알 수 있다:

```
01.<rdf:RDF
02.xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.xmlns:feature="http://www.linkeddatatools.com/clothing-features#">
04.
05.<rdf:Description rdf:about="http://www.linkeddatatools.com/clothes#t-shirt">
06.
07.<feature:size>12</feature:size>
08.<feature:color rdf:resource="http://www.linkeddatatools.com/colors#white"/>
09.
10.</rdf:Description>
11.
12.</rdf:RDF>
```

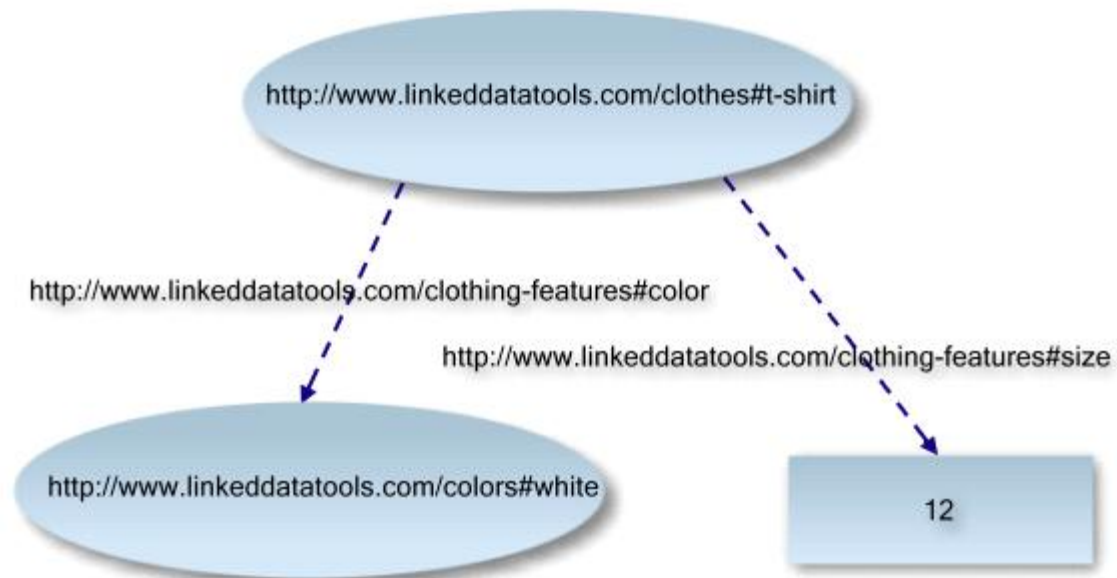
그리고 3번째 줄에서, 여러분은 우리가 XML namespace feature를 정의하기 위하여 그것에 URI `http://www.linkeddatatools.com/clothing-features#`라는 namespace를 지정하였다는 것도 주목해야 한다.

이 namespace의 목적은 단지 동일한 이름의 태그로 인하여 발생하는 이름간의 충돌을 피하기 위한 것이다: 그렇지만 “size”란 이름의 태그가 다른 namespace URIs로 정의된다면, RDF reader는 비록 그것들이 동일한 태그이름이라 하더라도 서로 다른 속성들이라는 것을 알게 된다.

또한 추가로 `feature:size`용으로 완전한 자격을 갖춘 URI를 얻기 위하여, 간단하게 prefix feature를 그것의 완전 이름인 namespace URI인 `http://www.linkeddatatools.com/clothing-features#size`로 대체할 수 있다.

Note - RDF에서 XML namespace URIs는 동일한 (태그) 이름을 가진 속성들을 구분하는데 사용된다. 충분히 자격이 있는 URI를 얻기 위하여, 간단하게 namespace prefix를 그것의 namespace URI로 대체할 수 있다.

이제 우리는 완전하게 자격을 갖춘 URIs에 대해 말할 수 있다:



보시다시피, 이것을 RDF graph diagrams으로 완전하게 URIs를 표현하는 것이 항상 쉽지도 편리하지도 않다. 종종 간략 버전(shorthand versions)이 대신 사용된다(다시 말해서, namespace prefix만을 사용하는 것).

이 번 강의를 마치자. 여러분은 다음의 방법에 대해 이해하여야 한다:

- How to write your own basic RDF 다큐먼트s in RDF/XML
- Understand how and why RDF uses URIs to identify subjects, predicates and objects
- How to relate an RDF 다큐먼트 to a corresponding data graph with fully qualified URIs

### Tutorial 3: Semantic Modeling

RDF가 전 세계적으로 교환 가능한 데이터를 레코딩하기 위하여 유연하고 그래프-중심의 모델을 제공하는 반면에, 그것이 어떤 어의나 의미를 제공하지는 않는다. 이제 일반적으로 이용



가능한 데이터 모델을 검토해서 문제점(fuss)이 무엇인지에 대해 알아보자.

데이터를 모델화하는 많은 인기 있고 주류적인 방법들이 있다. 이것들 중에서 어떤 것들은 다른 것보다 최신식이다. RDF 모델의 장점을 조사하기 전에, 이미 기존의 데이터 모델 방법들 중에서 몇 가지를 살펴보기로 하자:

아래의 테이블에서 시멘틱 데이터 모델의 고유한 특성에 초점을 맞추어, 몇 가지의 방법들을 비교해 보았다.

### 3.1 Comparing The Popular Data Models

데이터 모델링의 주류 방법과 SW 모델 간의 특징(features) 비교

Model	Example Format	Data	Metadata	Identifier	Query Syntax	Semantics (Meaning)
Object Serialization	.NET CLR Object Serialization	Object Property Values	Object Property Names	e.g. Filename	LINQ	N/A
Relational	MS SQL, Oracle, MySQL	Table Cell Values	Table Column Definitions	P r i m a r y K e y ( D a t a Column) Value	SQL	N/A
Hierachical	XML	Tag/Attribute Values	XSD/DTD	e.g. Unique Attribute Key Value	XPath	N/A
Graph	RDF/XML, Tutle	RDF	RDFS/OWL	URI	SPARQL	Yes, using RDFS and OWI

#### \*serialization?

In the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.

#### \*XSD?

XSD (XML Schema Definition), a recommendation of the World Wide Web Consortium (W3C), specifies how to formally describe the elements in an Extensible Markup Language (XML) 다큐먼트.

#### \*DTD?

A 다큐먼트 type definition (DTD) is a set of markup declarations that define a 다큐먼트 type for an SGML-family markup language (SGML, XML, HTML). A 다큐먼트 Type Definition (DTD) defines the legal building blocks of an XML 다큐먼트. It defines the 다큐먼트 structure with a list of legal elements and attributes. A DTD can be declared inline inside an XML 다큐먼트, or as an external reference.

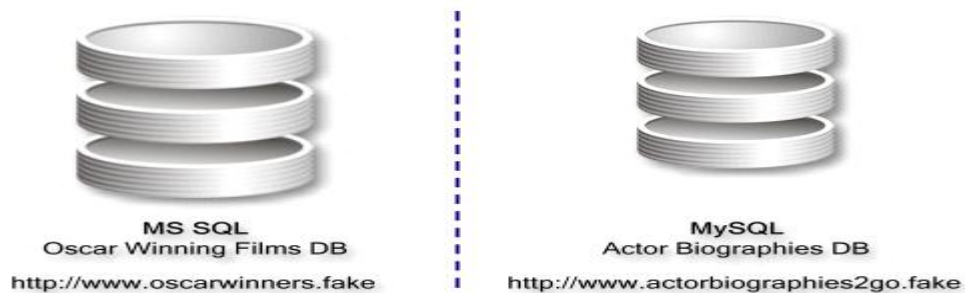
#### \*XPath?

XPath, the XML Path Language, is a query language for selecting nodes from an XML 다큐먼트. In addition, XPath may be used to compute values (e.g., strings, numbers, or Boolean values) from the content of an XML 다큐먼트. XPath was defined by the World Wide Web Consortium (W3C).

### 3.2 Why Include Semantics In Data? Knowledge Integration

만일 시멘틱이 커다란 이익을 주지 못한다면, 우리는 우리의 데이터에 시멘틱을 추가할 필요

가 없다. 데이터에 시멘틱 의미를 추가하는 가장 중요한 이익들 중의 하나는 자동적으로 다양한 지식의 영역에 가치를 뻗칠 수 있다는 것이다. 무엇이 지식의 도메인(domains of knowledge)인가? 간단한 예를 살펴보자.



위의 예에서, 두 개의 웹 사이트들은 서로 독립적으로 시작되었다. 한 사이트는 현재와 과거의 Oscar 수상 영화에 대한 정보를 호스트하고 있으며, 다른 사이트는 할리우드 남녀 영화배우의 전기에 대한 대규모의 데이터베이스 이다.

그렇지만 둘 다 자신들의 웹 사이트 데이터베이스에 보완할 정보가 있다. 우리는 먼저 이들 사이트 간에 시멘틱의 사용 없이 어떻게 정보공유가 발생하는지를 살펴보고 난 다음에, 시멘틱스를 사용하여 동일한 정보를 두 사이트가 서로 공유하는 방법에 대하여 알아볼 것이다.

## 1) Sharing Without Semantic Modeling

두 사이트 중에서 하나는 모든 오스카 수상영화에 대한 MS SQL database이고, 또 하나는 할리우드 배우에 대한 MySQL database이다. 이것들은 <http://www.oscarwinners.fake> 그리고 <http://www.actorbiographies2go.fake>에 각각 포함되어 있다. 그리고 이 두 사이트는 독립적으로 시작하였고 협력하지도 않고 있다.

Oscar Winners site는 그 이름처럼 과거에 제작된 모든 오스카 수상영화를 리스트하고 있으며, 또한 그것들에서 주연을 맡은 남녀 영화배우의 리스트도 갖고 있다. 그렇지만, 그 콘텐츠에는 이들의 이름과 생일날짜 이외의 다른 정보는 소장하고 있지 않다.

Actor Biographies 사이트는 현재뿐만 아니라 과거의 많은 할리우드 배우들에 대하여 완전한 전기 리스트뿐만 아니라 그들이 출연한 영화 리스트도 가지고 있다. 그러나 이것의 콘텐츠에는 해당 영화에 대한 어떠한 film-plots(영화구성), 또는 screen-shot(image)을 가지고 있지 않다.

이제 이 두 사이트가 최신의 모델보다 기존의 전통적 데이터 모델에서 협력할 수 있는 방법에 대하여 살펴보자:

- 분명하게 말해서, <http://www.oscarwinners.fake>의 이용자는 출연배우의 이름을 클릭하

면, 그들에 대해 더 많은 정보를 찾을 수 있다는 이점이 있다. - 이 정보는 <http://www.actorbiographies2go.fake>의 MySQL database에 저장되어 있다.

■ 똑같이, <http://www.actorbiographies2go.fake>의 이용자가 출연배우들의 영화이름을 클릭하여 더 많은 정보를 얻을 수 있다. 이것은 <http://www.oscarwinners.fake>에 있는 MS SQL database에 저장되어 있다.

■ 두 사이트간의 어떠한 데이터 공유도 자신들의 데이터베이스에 있는 테이블을 결합(joining)시키지 못하고 있다. 먼저, 이것들은 처음부터 독립적으로 설계되었으며, 따라서 두 데이터베이스에 있는 각각의 영화배우나 영화를 참조하는 primary key를 동기화(synchronized)시킬 수 없다. 이 문제를 해결하기 위하여 이것들은 mapped되어야 할 것이다. 그러나 또하나의 문제는 이것들이 서로 호환성이 없는 데이터베이스 서버 시스템을 사용하고 있다는 것이다.

■ 자신들이 현재 사용하고 있는 데이터베이스 간의 협력을 위하여, 각 사이트의 소유자들은 공동으로 영화 및 배우에 대한 고유한 ID scheme을 만들어 정보를 공유할 수 있는 공동의 데이터 포맷을 결정해야 할 것이다. 예를 들어, 자신들의 웹사이트에서 서로의 필요에 따라 정보를 요구할 수 있는 안전한 XML 콘텐츠를 만든다면, 이것이 가능할 것이다. 이런 방식으로 자신들의 정보 공유를 발전시킬 수 있다.

**Important Point** - 비호환적이고 독립적인 데이터 시스템 간에 정보를 교환하는 것은 시간, 돈, 그리고 서로 다른 데이터 세트에 대한 상황판단에 의한 인간적 해석을 필요로 한다. 또한 이러한 두 개의 웹사이트의 데이터 도메인으로만 제한되어 있어서 다른 곳에서 지식을 추가하는 데도 비슷한 노력이 요구된다. 따라서 이것은 인간이 데이터의 의미를 이해하도록 두 데이터베이스를 올바르게 통합시키는 공동 포맷을 필요로 한다.

이제 RDF와 semantics을 소개하기로 한다. RDF와 semantic web을 사용하여 이러한 문제를 해결할 수 있는 방법에 대하여 조사해 보자 - 모든 것은 자동적으로 이루어지며, 수동적으로 이루어지진 않는다.

## 2) Sharing With The Semantic Web Model

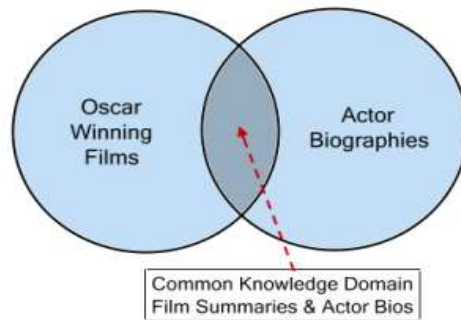
시멘틱 모델링에서, 먼저 다음과 같은 중요한 용어에 대해 이해하여야 한다:

- Vocabulary - contexts 간에 정의가 잘 일치하는 용어들의 집합이다.
- Ontology - 우리로 하여금 정의된 vocabulary 뒤에 숨어있는 contextual relationships를 정의한다. 이것은 지식의 영역을 정의하는 cornerstone이다. 온톨로지를 정의하기 위한 공식적 syntax는 OWL (Web Ontology Language)이며, 이것은 RDFS (RDF Schema)를 확장시켜 놓은 것으로, 다음 레슨에서 설명하기로 한다.

\*온톨로지란?

정보시스템의 대상이 되는 분야에 존재하는 개체와 개념에 대한 명세로서, 사람과 컴퓨터간에 공유되는 지식을 개념화한 구체적인 형식이며, 개념화와 개념화간의 관계를 표현하는 것이다. 온톨로지는 단어와 관계들로 구성된 일종의 사전으로서 생각할 수 있으며, 그 속에는 특정 도메인에 관련된 단어들이 계층적으로 표현되어 있고, 추가적으로 이를 확장할 수 있는 연관관계가 포함되어 있어, 웹 기반의 지식 처리나 응용프로그램 사이의 지식 공유 등이 가능하도록 되어 있다. 즉, 시맨틱 웹의 목적인 자동적인 실행과 추론을 하기 위해 온톨로지는 가장 핵심적인 개념이라고 할 수 있다.

시맨틱 모델링을 사용하여 우리가 두 개의 사이트의 시나리오를 어떻게 모델 하는가?



첫째, 두 사이트들은 공동의 표준적인 vocabulary를 개발하여, 이것을 문맥상에서 (contextually) 일치하는 자신들의 데이터를 기술하는 적용시켜야 한다. 예를 들어, 용어 'film title'은 두 사이트 모두에서 동일한 사물(thing)을 의미하여야 하며, 용어 'actor name' 과 'actor birthdate'도 마찬가지 이다.

이러한 활동으로 vocabulary로 사용된 용어들을 가지고 데이터의 암시적 의미를 표현할 수 있으며, 또한 쿼리에 의한 동일한 데이터를 생산할 수 있다. 이러한 결과는 똑같은 base ontology, 또는 common vocabulary를 채택하고 있는 두 사이트에 의해 얻어질 수 있다. 결과적으로 이들 두 사이트는 웹에서 서로 통신(communicate)할 수 있다.

만일 이러한 표준 vocabulary가 적재적소에 사용된다면(in place):

- 두 사이트는 동일한 용어에 의해 서로 질의할 수 있다.
- The Oscar Winning Movies site는 on-demand 방식으로 the Actor Biographies site의 배우 이름을 쿼리할 수 있으며, 그 영화에 출연한 특별한 남녀배우에 대하여 보다 자세한 정보를 얻을 수 있다.
- The Actor Biographies site 역시 on-demand 방식으로 Oscar Winning Movies site에 있는 film plots을 이제 쿼리할 수 있으며, 배우가 출연했던 영화들에 대하여 더 많은 자세한 정보를 얻을 수 있다.
- 공식적인 웹 온톨로지에서 정의된 contextual relationships(문맥전후 연관성)와 더불어, 촬영 장소와 같은 영화배우나 영화에 대한 추가적 연관 정보, 영화촬영과 동일한 날짜에 발생한 여러 가지 뉴스, 배우의 생년월일, 또는 동일한 제작자에 의해 제작된 영화 등등, 처음부터 그러한 정보가 존재했다고 상상하지 못하더라도, 이용자는 이러한 링크용 표준용어(linked

standard terminology)를 사용하여 부수적 정보를 얻을 수도 있다.

■ 이것은 두 사이트간에 존재하는 transformation, mapping, 또는 contracts와 같은 필요성과 상관없이 이루어져야 한다. 모든 정보가 바로 이 의미론(semantics)을 통해 생산된다.

우리는 다음 강의에서 SW 온톨로지의 구성(makeup)이 무엇이고, 여러분이 시멘틱 데이터베이스에 어떻게 쿼리하는지, 그리고 심지어 그것에서 기계적 추론(machine inference)을 어떻게 형성하는지를 보여줄 것이다.

**Point Of Interest** - 좋은 뉴스는 여러분이 관심을 갖는 특별한 지식영역에 대하여 여러분 스스로 온톨로지를 정의하고 공유하려는 노력을 기울일 필요가 없다는 것이다. 웹에는 이미 여러분이 채택할 수 있는 수 많은 인기있는 표준 온톨로지가 존재하고 있다. 그리고 필요하다면 여러분 스스로 이것을 확대할 수도 있다. 다음 섹션에서 이들 중 몇 가지를 소개하고자 한다.

여기서 논의된 도메인간의 지식공유(cross-domain knowledge sharing)는 단지 웹 사이트에만 적용되는 것이 아니라, 기관에서 구축한 지식 base에서도 적용된다. 따라서 SW 테크놀로지는 웹에서 출판된 어플이나 정보에만 제한되지 않아야 한다.

비록 시멘틱 데이터베이스를 처음 구축할 때, 좀 더 많은 기초 작업이 필요하다 하더라도, 전 세계에 걸쳐서 도메인 간의 통합의 용이성, 절약된 시간, 그렇게 하여 얻어진 아이디어에 대한 이점들은 잠재적으로 매우 귀중한 것이다.

### 3.3 Metadata Initiatives

지식의 도메인에서 용어를 표현하는 표준 vocabularies, 또는 공식적 ontologies는 이미 다양한 주제 - 예를 들어, media terms, biomedical terms, scientific terms - 에 대한 표준 vocabularies를 만드는 여러 전문기관으로부터 무료로 이용할 수 있다. 몇 가지 예를 살펴보면 다음과 같다:

■ Dublin Core Metadata Initiative (DCMI) - 특히 공통적이고 일상적인 용어 그리고 미디어의 주요 용어에 특별하게 초점을 맞추면서 여러 주제에 대한 온톨로지를 만들고 있다.

■ Friend Of A Friend (FOAF) - social networking purposes에 맞는 표준 vocabulary/ontology의 개발에 초점을 맞추고 있다.

■ OpenCyc(오픈사이크) - 일상적이고 상식적인 지식을 수집해 놓은 온톨로지.

The OpenCyc Platform is your gateway to the full power of Cyc, the world's largest and most complete general knowledge base and commonsense reasoning engine. OpenCyc contains hundreds of thousands of Cyc terms organized in a carefully designed ontology.

이번 강의는 끝났다. 다음 강의에서, 우리는 SW에서 온톨로지를 정의하는 방법과 SW 데이터베이스에서 정보를 쿼리하는 방법에 대한 보다 완벽한 기술적 내용을 알아볼 것이다.

You should now understand the following:

- The main benefits of semantic data over traditional data models.
- How a semantic web application might function to automatically link data between independent data sources covering the same base of knowledge.
- That open standards for common, everyday vocabulary currently exist.

## Tutorial 4: Introducing RDFS & OWL

앞장의 데이터 모델에서 vocabulary와 semantics의 모델링에 따른 장점을 소개하였고, 이제 의미론과 함께 RDF data models을 다루는데 필요한 실질적인 기술을 소개하기로 한다. RDF data는 RDFS 그리고 OWL과 같은 두 가지의 syntaxes를 사용하여 시멘틱 메타데이터와 함께 encoded 될 수 있다:

지난 강의에서, 우리는 시멘틱 모델과 함께 데이터를 모델화하는 보다 인기있고 전통적인 형태의 몇 가지를 비교하였다. 그리고 또한 SW 방식을 사용함으로써, 데이터 공유를 보다 분명하게 그리고 쉽게 확대시키고 상황(situation)을 소개하였다.

그리고 샘플을 사용하여, 두 개의 독립된 웹사이트가 서로 데이터를 공유함으로써 이익을 얻는 상황도 보여주었다. 이번 강의에서, 우리는 semantic metadata와 함께 RDF 데이터를 표현(annotate)하는데 사용되는 공식적인 syntax를 조사할 것이다. 그리고 그 다음 강의에서, 우리는 이런 데이터를 출판하고 쿼리하는 방법을 알아볼 것이다.

RDF 데이터는 두 가지의 중요한 신택스인 RDFS와 OWL을 사용하여 시멘틱 메타데이터와 함께 표현된다. RDFS 와 OWL 둘 다 W3C specifications 이다.

### 4.1 A Starting Example

여러분이 보고 있는 것이 OWL의 예이다:

```
01. <rdf:RDF
02. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03. xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04. xmlns:owl="http://www.w3.org/2002/07/owl#"
05. xmlns:dc="http://purl.org/dc/elements/1.1/">
06.
07. <!-- OWL Header Example -->
08. <owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
09. <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
10. <dc:description>An example ontology</dc:description>
```

```

11. </owl:Ontology>
12.
13. <!-- OWL Class Definition Example -->
14. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
15. < rdfs:label>The plant type</rdfs:label>
16. <rdfs:comment>The class of plant types.</rdfs:comment>
17. </owl:Class>
18.
19. </rdf:RDF>

```

지금 자세히 알려고 하지마라. 나중에 살펴보기로 한다. 그렇지만, 주목해야 하는 것은 예제인 RDF 다큐먼트의 header에 전에는 없었던 두 개의 새로운 namespaces가 포함되어 있다는 것이다: 3번째 줄의 RDFS (RDF Schema, <http://www.w3.org/2000/01/rdf-schema#>) 그리고 4번째 줄의 OWL (Web Ontology Language, <http://www.w3.org/2002/07/owl#>)의 namespaces 이다.

한 가지 더 주목할 것은 RDF에서 우리의 온톨로지를 정의하는 방법이다. 그래, 온톨로지 또한 RDF 다큐먼트 이다.

전통적인 온톨로지 문서를 분해해 보자. 예로서, plant varieties(식물의 종류)를 정의하고 있는 간단한 온톨로지를 살펴보자.

Point Of Interest - Why OWL, not WOL? When the acronym for Web Ontology Language (OWL) was first proposed by the working group, OWL was adopted instead of WOL as it is easily remembered, and suggested wisdom. But confusingly enough, it is still an acronym that should strictly speaking be WOL.

## 4.2 OWL Header

```

01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04.   xmlns:owl="http://www.w3.org/2002/07/owl#"
05.   xmlns:dc="http://purl.org/dc/elements/1.1/">
06.
07. <!-- OWL Header Example -->
08. <owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
09. <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
10. <dc:description>An example ontology written for the LinkedDataTools.com RDFS &OWL introduction
    tutorial</dc:description>
11. </owl:Ontology>
12.
13.<!-- Remainder Of 다큐먼트 Omitted For Brevity... -->
14.
15.</rdf:RDF>

```

비록 온톨로지가 헤더에 포함되지 않아야 하더라도, 이 곳은 여러분의 온톨로지에 포함되어 있는 것을 다른 사람에게 이해하는데 도움이 될 정보를 포함시키기에 좋은 장소이다.

위에서처럼, 우리는 온톨로지용 한 개의 title과 한 개의 description을 포함시켰다. 그렇지만, 이곳은 또한 올바른 갱신을 나타내는 version information을 포함시켜야 하는 장소이며, 여러분의 온톨로지를 다른 온톨로지에 수출(import)한다고 진술(state)할 수 있는 장소이다.

만일 여러분의 온톨로지가 다른 온톨로지에서 온 elements를 사용한다면, 이것은 여러분의 온톨로지가 다른 것에 의존하고 있다는 것을 알리는 tools나 frameworks를 위해 절대적으로 필요한 것이다.

### !!! Point:

**dc:** prefix, <http://purl.org/dc/elements/1.1/>라는 namespace를 정의한 5번째 줄을 보라. 이것은 Dublin Core Metadata Initiative라는 namespace를 참조하며, machine readers에게 **dc:title** and **dc:description** 와 같은 엘리먼트들은 이 온톨로지서 정의하고 있다고 알려주고 있다. Remember - we mentioned this often used ontology at the end of the previous tutorial. And, because it's an OWL ontology, it is also defined in RDF. Just point your browser to its namespace URI to download it.

## 4.3 OWL Classes, Subclasses & Individuals

온톨로지의 제 1차적 목적은 things를 semantics나 meaning에 따라 분류하는 것이다. OWL에서, 이런 목적은 this is achieved through the use of *classes* 그리고 *subclasses*를 사용하여 이룰 수 있다. 예를 들어 OWL에서는 이것을 *individuals(instance)* 라 부른다. 특정한 OWL class의 멤버들인 individuals는 그것의 *class extension*라 부른다.

OWL에서 *class* 는 individuals를 분류하여 공통의 성질을 공유하고 있는 그룹이다. 만일 하나의 individual이 한 class의 한 멤버라면, 그것은 machine reader에게 그것이 OWL class에 의해 주어진 어의적 분류에 속한다고 알려준다.

### An Example

다시 OWL ontology의 예를 보자. 이번에는 몇 가지의 classes와 subclasses가 추가 되었다. 우리는 3가지의 plant classes를 정의한다: the **flowering plants** class, **shrubs** class, 그리고 이 두가지 클래스를 superclass로 가지고 있는 **planttype** class. 따라서 **planttype** class는 모든 plant types의 최상위 클래스이다.



```

01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04.   xmlns:owl="http://www.w3.org/2002/07/owl#"
05.   xmlns:dc="http://purl.org/dc/elements/1.1/"
06.   xmlns:plants="http://www.linkeddatatools.com/plants#">
07.
08. <!-- OWL Header Omitted For Brevity -->
09.
10. <!-- OWL Class Definition - Plant Type -->
11. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
12.
13.   <rdfs:label>The plant type</rdfs:label>
14.   <rdfs:comment>The class of all plant types.</rdfs:comment>
15.
16. </owl:Class>
17.
18. <!-- OWL Subclass Definition - Flower -->
19. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#flowers">
20.
21.   <!-- Flowers is a subclassification of planttype -->
22.   <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
23.
24.   <rdfs:label>Flowering plants</rdfs:label>
25.   <rdfs:comment>Flowering plants, also known as angiosperms.</rdfs:comment>
26.
27. </owl:Class>
28.
29. <!-- OWL Subclass Definition - Shrub -->
30. <owl:Class rdf:about="http://www.linkeddatatools.com/plants#shrubs">
31.
32.   <!-- Shrubs is a subclassification of planttype -->
33.   <rdfs:subClassOf rdf:resource="http://www.linkeddatatools.com/plants#planttype"/>
34.
35.   <rdfs:label>Shrubbery</rdfs:label>
36.   <rdfs:comment>Shrubs, a type of plant which branches from the base.</rdfs:comment>
37.
38. </owl:Class>
39.
40. <!-- Individual (Instance) Example RDF Statement -->
41. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
42.
43.   <!-- Magnolia is a type (instance) of the flowers classification -->
44.   <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
45.
46. </rdf:Description>
47.
48. </rdf:RDF>

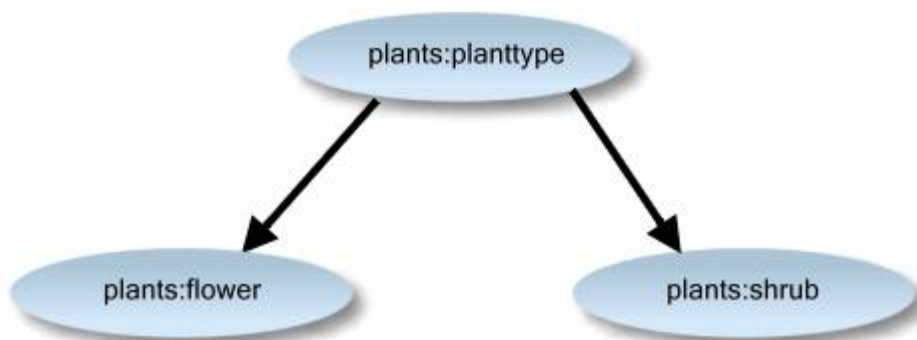
```

Point Of Interest(RDF 다큐먼트 의 타당성 조사) - You can investigate this RDF 다큐먼트 further by passing it through W3C's [RDF validator](#), which automatically tabulates the 다큐먼트's RDF subjects, predicates and objects for you. This can

help significantly improve your understanding of RDF. Just click the 'copy to clipboard' button on the top right of the code excerpt and paste into the validator.

### Taxonomy - A Hierarchy Of Terms

우리가 한 것은 우리의 어의적 용어나 클래스를 계층적으로 정의한 것이다. SW 세계에서, 이러한 용어의 계층을 *taxonomy*라 부른다. 다음은 우리가 정의한 taxonomy hierarchy을 그래프로 나타낸 것이다:



An example of a taxonomy hierarchy

Note - 우리는 magnolis(목련)라 부르는 flower 클래스의 또다른 섹클래스를 만들지 않았다. 그것보다는 magnolia는 flower 클래스의 individual(instance) 이다. 왜 이렇가? magnolia는 flower classification의 한 멤버이지만, 그것은 더 이상 flower subclassification이 아니다. 이것은 당연히 magnolia의 어의적 관점에서 보면 이것은 flower 클래스의 individuals(instances)이지 subclassification 즉, 다른 꽃이 아니라는 의미이다.

### 4.4 OWL Properties

OWL에서 Individuals은 *properties*와 관련 있다. OWL에는 두 종류의 property가 있다:

- **Object properties** (owl:ObjectProperty): 두 가지 OWL classes의 individuals (instances)와 관련이 있다.
- **Datatype properties** (owl:DatatypeProperty): OWL classes의 individuals (instances)의 literal values과 관련 있다.

### An Example

먼저 a *data type* property (one which links an instance to a literal value)을 추가한

다음에, Magnolia가 속해 있는 *species family* 의 이름을 추가해 보자.

```
01. <rdf:RDF
02. xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03. xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
04. xmlns:owl="http://www.w3.org/2002/07/owl#"
05. xmlns:dc="http://purl.org/dc/elements/1.1/"
06. xmlns:plants="http://www.linkeddatatools.com/plants#">
07.
08. <!-- OWL Header Omitted For Brevity -->
09.
10. <!-- OWL Classes Omitted For Brevity -->
11.
12. <!-- Define the family property -->
13. <owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"/>
14.
15. <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
16.
17. <!-- Magnolia is a type (instance) of the flowers class -->
18. <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
19.
20. <!-- The magnolia is part of the 'Magnoliaceae' family -->
21. <plants:family>Magnoliaceae</plants:family>
22.
23. </rdf:Description>
24.
25. </rdf:RDF>
```

Important Point - 여기서, 만일 여러분이 object oriented programmer라면, 여러분은 마 음으로부터 programmatic object classes and their associated properties를 생각해 낸 다음에, 그것들을 OWL classes에 대해 배운 것과도 비교하려 할 것이다. 그러지 마세요 - 정말 달라요. 위의 예제를 주목해 보자. 'family' property는 어떠한 class type과도 독립적 이며, class flower (magnolia)의 instance에 할당된다. 똑같은 클래스의 또다른 instance 는 이 property를 갖지 않을 수 있다. OWL에서는 그러하다. instance를 가지고 있는 properties가 그것들의 class types이 아니라 그것들의 instance를 기술한다는 것을 주목하 라. 이 경우에, 여러분은 완전히 서로 다른 클래스용으로 동일한 'family' property를 사용할 수 있다.

끝으로, an *object* property (one which links an instance to another instance)를 추가 해 보자. 우리가 상점을 운영하고 있고 이 식물(Magnolia)을 상점주인과 마찬가지로 우리도 인기가 있는 다른 식물에 링크하길 원한다고 하자( Let's say we're running a shop, and we want to link this plant (Magnolia) to another plant which we know as the shop owner is equally as popular). "similarlyPopularTo"라는 property를 추가해 보자:

```

01. <rdf:RDF
02.   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
03.   xmlns:owl="http://www.w3.org/2002/07/owl#"
04.   xmlns:dc="http://purl.org/dc/elements/1.1/"
05.   xmlns:plants="http://www.linkeddatatools.com/plants#">
06.
07.   <!-- OWL Header Omitted For Brevity -->
08.
09.   <!-- OWL Classes Omitted For Brevity -->
10.
11.   <!-- Define the family property -->
12.   <owl:DatatypeProperty rdf:about="http://www.linkeddatatools.com/plants#family"/>
13.
14.   <!-- Define the similarlyPopularTo property -->
15.   <owl:ObjectProperty rdf:about="http://www.linkeddatatools.com/plants#similarlyPopularTo"/>
16.
17.   <!-- Define the Orchid class instance -->
18.   <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#orchid">
19.
20.     <!-- Orchid is an individual (instance) of the flowers class -->
21.     <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
22.
23.     <!-- The orchid is part of the 'Orchidaceae' family -->
24.     <plants:family>Orchidaceae</plants:family>
25.
26.     <!-- The orchid is similarly popular to the magnolia -->
27.     <plants:similarlyPopularTo rdf:resource="http://www.linkeddatatools.com/plants#magnolia"/>
28.
29.   </rdf:Description>
30.
31.   <!-- Define the Magnolia class instance -->
32.   <rdf:Description rdf:about="http://www.linkeddatatools.com/plants#magnolia">
33.
34.     <!-- Magnolia is an individual (instance) of the flowers class -->
35.     <rdf:type rdf:resource="http://www.linkeddatatools.com/plants#flowers"/>
36.
37.     <!-- The magnolia is part of the 'Magnoliaceae' family -->
38.     <plants:family>Magnoliaceae</plants:family>
39.
40.     <!-- The magnolia is similarly popular to the orchid -->
41.     <plants:similarlyPopularTo rdf:resource="http://www.linkeddatatools.com/plants#orchid"/>
42.
43.   </rdf:Description>
44.
45. </rdf:RDF>

```

예를 들어, 우리는 URI <http://www.linkeddatatools.com/plants#orchid>와 더불어 Orchid를 표현하는 flowers 클래스의 새로운 individual(instance)을 정의하였다. 여러분이 우리가 동일한 클래스의 individual인 Mangnolia instance를 정의한 방법으로부터 이것을 이해할 수 있는지 확인해 보라. 이제, 우리의 첫 번째 두 개의 tutorials에서처럼, 여러분이 위에서 정의한 RDF graph에 따라 Orchid 그리고 Magnolia class instances와 이것들의

predicates를 보여주는 그래프를 그릴 수 있는지 확인해 보라.

*Hint:* 여러분은 양쪽 the *family* 그리고 *similarlyPopularTo* properties을 나타내는 화살표를 그려야 할 것이다. *family* property와 관련해서, 각각의 plant 용으로 서로 다른 family type literals을 포인트(point)해야 할 것이다. 그리고 *similarlyPopularTo* property와 관련해서는 the instances가 서로서로 포인트해야 할 것이다.

#### Important - Point Note

위의 예에서, 우리는 *similarlyPopularTo* object property를 통해 똑같은 OWL 클래스의 2 instances간을 링크하는 a two way를 갖는다(Orchid와 Magnolia 둘 다 똑같은 클래스의 individuals 이다). 그렇지만 주목할 것은 object properties는 two way가 아닐 수도 있다 - 그것들은 they may be one way일 수도 있다는 것이다. 그리고 중요한 것은 똑같은 OWL 클래스의 instances 간에 일어나지 않아야 한다(need not be between instances of the same OWL class). 그것들은 완전히 다른 OWL classes일 수 있다.

이제 이장을 마친다. 여러분은 다음과 같은 것을 이해하고 있어야 한다:

- That RDFS and OWL are W3C specifications with their own standard W3C namespaces.
- How OWL headers are constructed and some example uses.
- How to implement your own OWL classes, subclasses, individuals and properties.
- How to build your own basic ontology.

### Tutorial 5: Querying Semantic Data

이제 여러분은 RDF에 시멘텍스를 추가하는 방법, 그것을 출판하고 쿼리하는 방법을 알고 있다. 관계형 데이터베이스의 테이블에서 SQL을 사용하여 쿼리하는 것처럼, RDF 데이터의 triples는 SPARQL을 사용하여 쿼리한다. 우리는 몇 가지 기본적인 쿼리를 조사하여 SPARQL과 SQL을 비교해 본다.

After this tutorial, you should be able to:

- Build your own basic SPARQL queries, step-by-step
- Understand that SPARQL is not just a query language, it is also a protocol
- Understand the XML format in which SPARQL protocol returns the results of queries
- Try executing some SPARQL queries yourself on some UK government RDF data
- Look ahead to SPARQL+ which has add, modify and delete capabilities

MS SQL or MySQL과 같은 관계형 데이터베이스로부터 데이터를 검색하는데 사용하는.

만일 여러분이 전통적인 IT에 대한 배경을 갖고 있다면, 여러분은 이미 MS SQL or MySQL 과 같은 관계형 데이터베이스에서 데이터를 검색하는데 사용하는 SQL(Structured Query Language, pronounced "sequel")에 대하여 잘 알고 있을 것이다.

비슷하게, RDF data stores 역시 자신들의 쿼리 언어인 SPARQL (SPARQL Protocol and RDF Query Language, pronounced "sparkle")을 사용하여 쿼리한다. 그렇지만 SPARQL은 좀 더 고급스럽다.

SPARQL은 W3C standard 이며, 현재 버전은 1.1 이다.

## 5.1 A Starting Example

이제 예를 통해 SPARQL의 모습을 보여주기로 한다:

```
1. PREFIX sch-ont: <http://education.data.gov.uk/def/school/>
2. SELECT ?name WHERE {
3. ?school a sch-ont:School.
4. ?school sch-ont:establishmentName ?name.
5. ?school sch-ont:districtAdministrative
   <http://statistics.data.gov.uk/id/local-authority-district/00AA>.
6. }
7. ORDER BY ?name
```

여러분은 이 모습에 익숙치않지만 걱정하지 마라.

이 쿼리가 만일 UK government's Open Semantic Database에서 수행된다면, 영국에서 행정구역 00AA에 있는 모든 학교의 이름을 얻게 될 것이다. 간단하게 쿼리를 살펴본 다음에, 이것을 정밀하게 조사하기 전에 여러분은 왜 이것에 대해 알아야 하는지를 생각해 보라.

**Try It Yourself** - Copy and paste the above query into the UK government's SPARQL query endpoint at <http://education.data.gov.uk/sparql/education/query.html> and then see the results. Such an URL is called a SPARQL endpoint in the semantic web world - and is the way in which the data on the semantic web is published to the outside world in a query enabled form.

## SPARQL Is Similar To SQL

SQL처럼, SPARQL은 선택된 데이터의 어떠한 하위집합을 얻을 것인지를 결정하기 위하여 a SELECT statement를 사용하여 query data set로부터 데이터를 선택한다. 또한 SPARQL은 쿼리 데이터 세트에서 매칭되는 것을 찾기 위한 그래프 패턴(graph pattern)을 정의하기

위하여 uses a WHERE clause을 사용한다.

SPARQL WHERE clause에서 graph pattern은 데이터에서 매치되는 것을 찾기 위하여, subject, predicate 그리고 object triple로 구성된다. 이제 위의 예를 좀 더 자세히 조사해보자.

SELECT statement는 변수 ?name의 결과(returned)를 얻도록 한다.

**Note** - SPARQL에서, 변수 이름(variable names)은 question mark ("?) symbol을 접두사(prefix)로 갖는다. query graph pattern에서, 이것들은 어떤 node와 match 한다 - 그것이 resource 또는 literal이든 상관없다.

주목할 것은 이 변수 또한 WHERE clause search pattern에 사용될 수 있다 - second query search pattern의 object로. 그러나 또한 주목할 것은 ?school 변수이다. Because a 특정한 URI가 매치가 아니라 변수용이기 때문에, 어떠한 matching subject URI도 이 부분의 쿼리 패턴과 관련해서 리턴될 수 있으며 그 결과는 그 변수 이름과 mapped될 것이다.

그러므로, 위의 SPARQL query에서, ?name은 쿼리에서 지정한 3가지 탐색패턴과 매치되는 모든 학교이름을 리턴 시킨다. 우리가 원한다면, 우리는 추가적으로 매치 criteria를 추가함으로써 이 쿼리를 좀 더 전문화할 수 있다. 그렇지 않다면, 행정지역의 값 "00AA"과 매치되는 학교만을 요구하는 마지막 탐색 패턴을 제거함으로써, 우리는 위의 예에서 좀 더 포괄적인 결과를 얻을 수 있다.

마지막으로, 주목할 것은 ?school 변수는 모두 3가지인 탐색패턴과 관련해서, 어떤 subject가 탐색패턴과 매치되면 이 변수의 결과를 리턴시킬 것을 의미하고 있다. 그러나 이것이 이번 쿼리의 SELECT statement에 언급되지 않았으므로, ?school가 mapped되더라도, 그 결과 세트를 리턴하지는 않는다.

위에서 제시한 정부 데이터베이스에 대하여 쿼리해 보면 여러분 스스로 이것을 알게될 것이다.

## 5.2 SPARQL General Form

SPARQL queries 는 아래와 같은 일반적 행태를 가지고 있다. 이것은 하나의 쿼리는 여러 섹션으로 세분될 수도 있으며, 그 섹션을 정의하는 clause or keyword가 있다는 것을 보여주고 있다.

<b>PREFIX (Namespace Prefixes)</b>
e.g. PREFIX plant: <http://www.linkeddatatools.com/plants>
<b>SELECT (Result Set)</b>
e.g. SELECT ?name
<b>FROM (Data Set)</b>
e.g. FROM <http://www.linkeddatatools.com/plantsdata/plants.rdf>
<b>WHERE (Query Triple Pattern)</b>
e.g. WHERE { ?planttype plant:planttype ?name }
<b>ORDER BY, DISTINCT etc (Modifiers)</b>
e.g. ORDER BY ?name

또는, 위에서 나타난 각 섹션의 예를 근거로, 우리는 다음과 같은 쿼리를 작성할 수 있다:

1. PREFIX plant: <http://www.linkeddatatools.com/plants>
2. FROM <http://www.linkeddatatools.com/plantsdata/plants.rdf>
3. SELECT ?name WHERE {
4. ?planttype plant:planttype ?name.
5. }
6. ORDER BY ?name

한 단계 한 단계 SPARQL query를 형성하는 방법을 쉽게 이해해 보자.

### 5.3 Building A SPARQL Query

어떤 트리플 데이터와 관련해서 SPARQL 쿼리를 구축하는 방법을 배우기 위하여, 하나의 예제 데이터 세트로 시작해 보자: 우리가 이미 앞 강의에서 정의했던 plants & shrubs 용 온톨로지를 근거로 한다.

Example triple data containing a variety of shrubs and plants, and their family names

Subject	Predicate	Object
http://www.linkeddatatools.com/plants#magnolia	http://www.linkeddatatools.com/plants#family	Magnoliaceae
http://www.linkeddatatools.com/plants#african_lilly	http://www.linkeddatatools.com/plants#family	Liliaceae
http://www.linkeddatatools.com/plants#silvertop	http://www.linkeddatatools.com/plants#family	Aralianae
http://www.linkeddatatools.com/plants#velvetleaf	http://www.linkeddatatools.com/plants#family	Malvaceae
http://www.linkeddatatools.com/plants#manglietia	http://www.linkeddatatools.com/plants#family	Magnoliaceae

우리의 example data set에는 5가지의 plants & shrubs가 들어있다. subject는 그 식물을 표현하는 URI이다(가독성을 위하여 URI는 식물이나 관목의 일반 이름을 사용하였다). The predicate는 family name이다 - 이것은 우리가 정의했던 온톨로지에서 가져왔다. 그런 다음에



우리는 literal objects를 사용한다 - 이것은 식물이나 관목의 과학적 과이름(family name)인 문자열로 구성되어 있다.

이 데이터와 관련해서 몇 가지 간단한 쿼리를 작성해서 그 결과를 살펴보자.

## Select All Data

먼저, 위의 데이터로부터 모든 plant URIs (subjects) 그리고 plant family names (literal-type objects)을 선택할 쿼리를 작성해 보자.

1. PREFIX plants: <http://www.linkeddatatools.com/plants>
2. SELECT \* WHERE
3. {
4. ?name plants:family ?family
5. }

SQL과 마찬가지로 SPARQL에서 wildcard '\*'는 모든 매치된 데이터를 결과세트로 리턴시킬 것이다. 이것이 무엇을 의미하는가? 우리가 두 개의 변수 ?name 그리고 ?family를 언급했으므로, 그 쿼리는 우리의 결과 세트에 이렇게 선언된 두 가지 쿼리변수가 mapped 되는 subjects, predicates or objects에 따라, 그 결과를 리턴시킬 것이다.

이렇게 우리는 쿼리를 정의하였고, 이제 그것을 실행해 보자. SPARQL은 단지 query language만이 아니며, 그것은 또한 프로토콜이며 여러분의 소프트웨어가 읽을 수 있는 특별한 schema에 결과를 리턴 시킨다. 결과 세트 중에서 가장 널리 사용되는 형태들 중의 하나가 XML result format 이다. 아래는 우리가 되돌아갈 XML format result set 이다.

01. <?xml version="1.0" ?>
02. <sparql xmlns="http://www.w3.org/2005/sparql-results#">
03. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
04. <head>
05. <variable name="name"/>
06. <variable name="family"/>
07. </head>
08. <results>
09. <result>
10. <binding name="name">
11. <uri>http://www.linkeddatatools.com/plants#magnolia</uri>
12. </binding>
13. <binding name="family">
14. <literal>Magnoliaceae</literal>
15. </binding>
16. </result>
17. <result>
18. <binding name="name">
19. <uri>http://www.linkeddatatools.com/plants#african\_lilly</uri>
20. </binding>

```

21. <binding name="family">
22. <literal>Liliaceae</literal>
23. </binding>
24. </result>
25. <result>
26. <binding name="name">
27. <uri>http://www.linkeddatatools.com/plants#silvertop</uri>
28. </binding>
29. <binding name="family">
30. <literal>Aralianae</literal>
31. </binding>
32. </result>
33. <result>
34. <binding name="name">
35. <uri>http://www.linkeddatatools.com/plants#velvetleaf</uri>
36. </binding>
37. <binding name="family">
38. <literal>Malvaceae</literal>
39. </binding>
40. </result>
41. <result>
42. <binding name="name">
43. <uri>http://www.linkeddatatools.com/plants#manglietia</uri>
44. </binding>
45. <binding name="family">
46. <literal>Magnoliaceae</literal>
47. </binding>
48. </result>
49. </results>
50. </sparql>

```

이것은 매우 읽기 쉽고 양이 많다. 우리는 결과 세트가 사용하는 어떤 XML namespaces를 정의하는 사용할 수 있는 root <sparql> node를 갖고 있다. 그런 다음, 우리의 SELECT statement에서 wildcard (\*)를 사용했기 때문에, WHERE clause 에 있는 쿼리 그래프 패턴에서 선언된 두 개의 변수인 ?name 과 ?family를 검색한다. 알다시피, 이러한 것들은 결과 세트의 <head> section에서 정의된다. 즉, 이것들을 쿼리에 의해 리턴되는 변수들이다.

이제 우리는 이들 쿼리 변수에 묶여진 결과를 얻는다. 각각의 패턴 매치는 그것 자체의 <result> node에 리턴된다. 주목할 것은 우리의 쿼리의 결과에서 어떠한 매치도 이루어지지 않았다면 우리는 우리의 XML에 리턴되는 어떠한 결과도 없다는 것이다.

여러분은 우리의 result XML이 데이터를 enclose하기 위하여 <uri> and <literal> tags를 사용함으로써 URI type values과 literal values을 구분하는 방법을 알 수 있다. 이것은 여러분이 필요로 할 때 이러한 구분법을 이용하여 이러한 데이터를 조사(parse)하도록 만들어진 소프트웨어에 의존하게 될 것이다. 그러나 그것들이 이러한 방법을 타이프한다는 사실은 리턴된 결과가 사용하는 시스템이 무엇이든지 간에 매우 유용하다.

## Select Only "Magnoliaceae" Family

보다 정밀하게 우리의 데이터 세트로부터 Magnoliaceae family에 속하는 식물만을 리턴시켜 보자. 어떻게 할까?

```
1. PREFIX plants: <http://www.linkeddatatools.com/plants>
2. SELECT * WHERE
3. {
4. ?name plants:family "Magnoliaceae"
5. }
```

이것은 간단하게 이전의 쿼리에서 ?family 변수를 제거한 다음에, 그것 대신에 literal "Magnoliaceae"를 대치시키면 된다. 이제 우리의 쿼리는 어떤 object match보다 이 문자에 맞는 정확한 매치를 수행할 것이다.

쿼리 수행 결과는 다음과 같다:

```
01. <?xml version="1.0" ?>
02. <sparql xmlns="http://www.w3.org/2005/sparql-results#"
03. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
04. <head>
05. <variable name="name"/>
06. </head>
07. <results>
08. <result>
09. <binding name="name">
10. <uri>http://www.linkeddatatools.com/plants#magnolia</uri>
11. </binding>
12. </result>
13. <result>
14. <binding name="name">
15. <uri>http://www.linkeddatatools.com/plants#manglietia</uri>
16. </binding>
17. </result>
18. </results>
19. </sparql>
```

WHERE 조건절에서 ?family 변수를 제거했기 때문에, 이것은 결과에 더 이상 존재하지 않으며, 우리의 <head> section에는 단지 return variable인 ?name, 그리고 우리의 두 가지 matching triple patterns만이 포함 된다 - object value으로 문자값 "Magnoliaceae"를 가지고 있는 두 멤버 모두 다 Magnoliaceae family 이다.

## 5.4 Further Exploration

이번에는 여러분에게 무엇이 querying triple data와 같은지에 대하여 알아보자. Building SPARQL queries를 구축하는 것은 다소 실무적이지만, 알아야할 가치가 있다. 이제 여러분은 SPARQL query가 어떤 형태로 되어 있는지에 대해 잘 알았을 것이다. 추가로 더 많은 자료

를 읽을으로써 여러분이 실습하도록 노력해라.

다행스럽게도, W3C에서 이 강의 내용을 넘어서서 이 주제에 대한 훌륭한 개론을 소개하고 있다. 여러분이 이것을 배우길 추천한다. 여러분은 W3C의 SPARQL Query Language For RDF 페이지에서 이것을 찾을 수 있다.

**Note - You may like to try SPARQL queries of your own using our linked data tool suite RDF Studio. RDF Studio has full SPARQL help (including autocomplete of your queries) to help you write queries as you type.**

## 5.5 What About CREATE, INSERT, UPDATE?

여러분이 triple data store로부터 데이터의 subsets을 얻기 위하여 SPARQL의 SELECT-WHERE 형태에 대한 기본들을 배우는 동안, 우리는 아직까지 CREATE, INSERT or UPDATE 방법들을 다루지 않았다. 이렇게 한 이유는 바로 그것들은 현재 실행되지 (implemented) 않기 때문이다.

만일 여러분이 그것에 대하여 생각한다면, 어느 정도 이해하여야 한다. 질의가능하고, 공식적인 SPARQL endpoint에서 데이터를 출판하는 단체들은 아마도 대부분의 경우에 자신들의 가치있는 데이터를 기술하거나 변화시키는 것을 원치 않을 것이다. 또한 어느 단계에서 그것은 missing requirement 같이 여겨지고 있다 - 특히 만일 여러분이 웹을 통해서 공식적이라기 보다는 로컬리하게 triple data stores를 사용하고 있고 쿼리언어(SQL 데이터베이스에서처럼)를 사용하여 변경하고자 할 경우에. 어떻게 이것이 이뤄질 수 있을까?

현재 new specifications such as SPARUL (SPARQL/Update) and SPARQL+와 같은 새로운 스펙이 이런 문제를 해결하기 위하여 개발 중에 있지만, 기능적으로 이러한 문제를 해결할 수 있는 확실한 것(solid contender)는 아직 이 글을 쓰고 있는 기간에는 나타나지 않았다.

As RDF data가 그렇더라도 널리 채택되고 있더라도, Semantic Web frameworks에 의해 설치되도록 등장할 a winning contender를 기대하자. 우리는 물론 관심을 기울일 것이다.

이 장을 마치면서, 여러분은 다음과 같은 것을 이해할 수 있어야 한다:

- That SPARQL can query RDF datasets, rather like SQL can query a relational database.
- That SPARQL endpoints are used to query and return data from the semantic web.
- How you can build and execute simple SPARQL queries yourself.
- Have a starting knowledge of the form in which SPARQL queries return results.

FIN