



## RDF and RDF Schema

**Oscar Corcho, Raúl García Castro, Oscar Muñoz-García**  
{ocorcho,rgarcia}@fi.upm.es, omunoz@delicias.dia.fi.upm.es  
<http://www.oeg-upm.net/>

Ontological Engineering Group  
Laboratorio de Inteligencia Artificial  
Facultad de Informática  
Universidad Politécnica de Madrid  
Campus de Montegancedo sn,  
28660 Boadilla del Monte, Madrid, Spain

*Work distributed under the license Creative Commons Attribution-Noncommercial-Share Alike 3.0*

# Main References



Gómez-Pérez, A.; Fernández-López, M.; Corcho, O. **Ontological Engineering**. Springer Verlag. 2003

## Capítulo 4: Ontology languages



Brickley D, Guha RV (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation.

<http://www.w3.org/TR/PR-rdf-schema>

Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation.

<http://www.w3.org/TR/REC-rdf-syntax/>

Prud'hommeaux E, Seaborne A (2008) *SPARQL Query Language for RDF*. W3C Recommendation.

<http://www.w3.org/TR/rdf-sparql-query/>



Jena web site: <http://jena.sourceforge.net/>

Jena API: [http://jena.sourceforge.net/tutorial/RDF\\_API/](http://jena.sourceforge.net/tutorial/RDF_API/)

Jena tutorials: <http://www.ibm.com/developerworks/xml/library/j-jena/index.html>

<http://www.xml.com/pub/a/2001/05/23/jena.html>



SPARQL validator: <http://www.sparql.org/validator.html>

SPARQL implementations: <http://esw.w3.org/topic/SparqlImplementations>

SPARQL tutorials: <http://jena.sourceforge.net/ARQ/Tutorial/>

<http://www.w3.org/2004/Talks/17Dec-sparql/intro/all.html>

<http://www.cs.man.ac.uk/~bparsia/2006/row-tutorial/>

# Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. **RDF(S) management APIs**
  - 4.1 RDF(S) management APIs
  - 4.2 The Jena API, with a hands-on activity
5. RDF(S) query languages: SPARQL

# Sample RDF APIs

## RDF libraries for different languages:

- Java, Python, C, C++, C#, .Net, Javascript, Tcl/Tk, PHP, Lisp, Obj-C, Prolog, Perl, Ruby, Haskell
- List in <http://esw.w3.org/topic/SemanticWebTools>

## Usually related to a RDF repository

- **Multilanguage:**
  - Redland RDF Application Framework (C, Perl, PHP, Python and Ruby):  
<http://www.redland.opensource.ac.uk/>
- **Java:**
  - Jena: <http://jena.sourceforge.net/>
  - Sesame: <http://www.openrdf.org/>
- **PHP:**
  - RAP - RDF API for PHP: <http://www4.wiwiss.fu-berlin.de/bizer/rdfapi/>
- **Python:**
  - RDFLib: <http://rdflib.net/>
  - Pyrpale: <http://infomesh.net/pyrpale/>

# Jena

- **Java framework for building Semantic Web applications**
- **Open source software from HP Labs**
- **The Jena framework includes:**
  - A RDF API
  - An OWL API
  - Reading and writing RDF in RDF/XML, N3 and N-Triples
  - In-memory and persistent storage
  - A rule based inference engine
  - SPARQL query engine

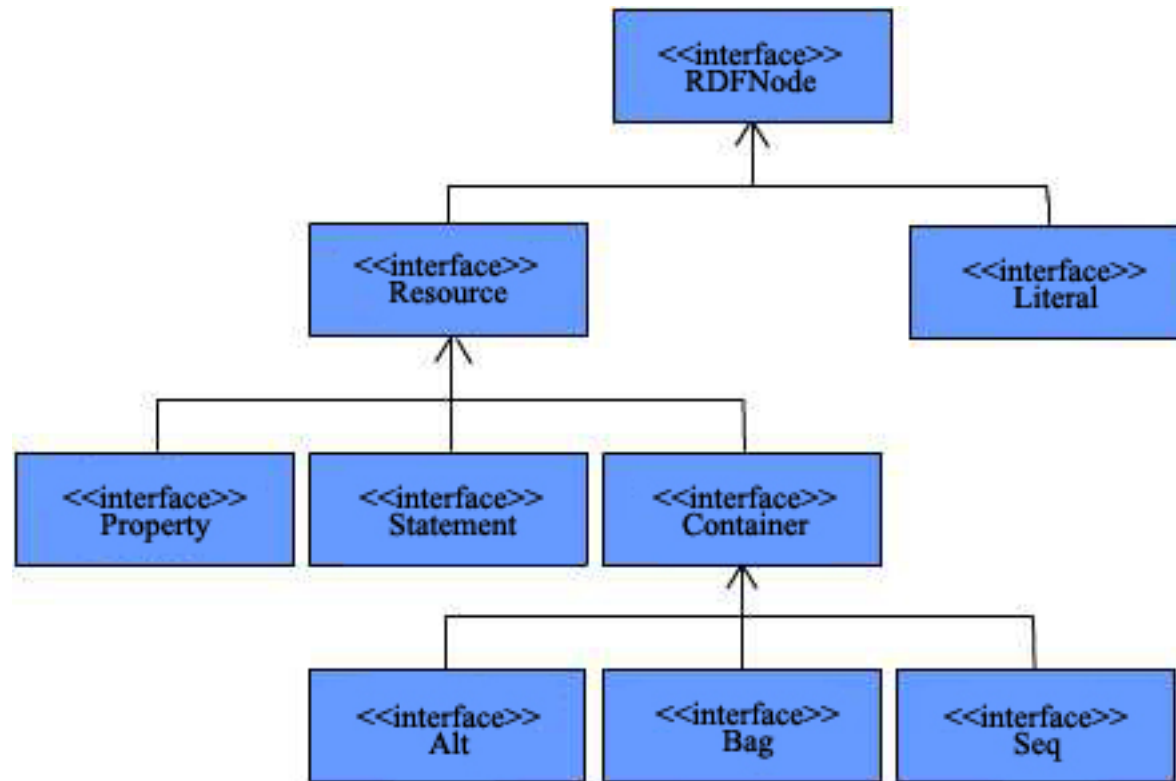
# Sesame

- **A framework for *storage, querying* and *inferencing* of RDF and RDF Schema**
- **A Java Library for handling RDF**
- **A Database Server for (remote) access to *repositories* of RDF data**
- **Highly expressive query and transformation languages**
  - SeRQL, SPARQL
- **Various backends**
  - Native Store
  - RDBMS (MySQL, Oracle 10, DB2, PostgreSQL)
  - main memory
- **Reasoning support**
  - RDF Schema reasoner
  - OWL DLP (OWLIM)
  - domain reasoning (custom rule engine)

# Table of Contents

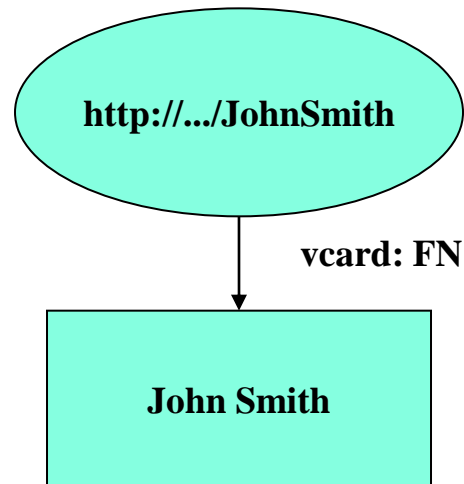
1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. **RDF(S) management APIs**
  - 4.1 RDF(S) management APIs
  - 4.2 **The Jena API, with a hands-on activity**
5. RDF(S) query languages: SPARQL

# Jena API Structure



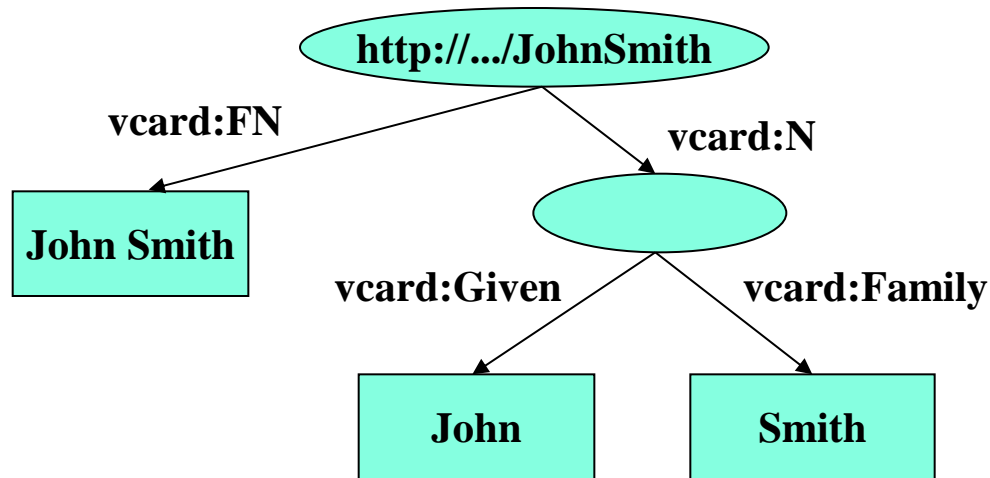


# Data Model



```
// some definitions
static String personURI = "http://somewhere/JohnSmith";
static String fullName = "John Smith";
// create an empty Model
Model model = ModelFactory.createDefaultModel();
// create the resource
Resource johnSmith = model.createResource(personURI);
// add the property
johnSmith.addProperty(VCARD.FN, fullName);
```

## Another data model



```
// some definitions
String personURI = "http://somewhere/JohnSmith";
String givenName = "John";
String familyName = "Smith";
String fullName = givenName + " " + familyName;
// create an empty
Model model = ModelFactory.createDefaultModel();
// create the resource
// and add the properties cascading style
Resource johnSmith = model.createResource(personURI)
    .addProperty(VCARD.FN, fullName)
    .addProperty(VCARD.N, model.createResource())
    .addProperty(VCARD.Given, givenName)
    .addProperty(VCARD.Family, familyName));
```

# Statements

```
// list the statements in the Model
StmtIterator iter = model.listStatements();
// print out the predicate, subject and object of each statement
while (iter.hasNext())
{
    Statement stmt = iter.nextStatement(); // get next statement
    Resource subject = stmt.getSubject(); // get the subject
    Property predicate = stmt.getPredicate(); // get the predicate
    RDFNode object = stmt.getObject(); // get the object
    System.out.print(subject.toString());
    System.out.print(" " + predicate.toString() + " ");
    if (object instanceof Resource) {
        System.out.print(object.toString());
    }
    else { // object is a literal
        System.out.print(" \"" + object.toString() + "\"");
    }
    System.out.println(" .");
} // end of while
```

```
http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#N anon:14df86:ecc3dee17b:-7fff
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Family "Smith"
anon:14df86:ecc3dee17b:-7fff http://www.w3.org/2001/vcard-rdf/3.0#Given "John"

http://somewhere/JohnSmith http://www.w3.org/2001/vcard-rdf/3.0#FN
"John Smith" .
```

# Writing RDF

```
Model model = ModelFactory.createDefaultModel();
Resource jsmith =
model.createResource("http://somewhere/johnsmith")
    .addProperty(VCARD.FN, "John Smith")
    .addProperty(VCARD.N, model.createResource())
    .addProperty(VCARD.Given, "John")
    .addProperty(VCARD.Family, "Smith"));
model.write(new PrintWriter(System.out));
```

```
model.write(System.out, "RDF/XML-ABBREV");
```

```
model.write(System.out, "N-TRIPLE");
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
>
  <rdf:Description rdf:nodeID='A0'>
    <vcard:Given>John</vcard:Given>
    <vcard:Family>Smith</vcard:Family>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/johnsmith'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID='A0' />
  </rdf:Description>
</rdf:RDF>
```

# Reading RDF

```
// create an empty model
Model model = ModelFactory.createDefaultModel();

// use the FileManager to find the input file
InputStream in = FileManager.get().open( inputFileName );
if (in == null) {
    throw new IllegalArgumentException("File not found");
}

// read the RDF/XML file
model.read(in, "");

// write it to standard out
model.write(System.out);
```

```
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:vcard='http://www.w3.org/2001/vcard-rdf/3.0#'
>
  <rdf:Description rdf:nodeID="A0">
    <vcard:Family>Smith</vcard:Family>
    <vcard:Given>John</vcard:Given>
  </rdf:Description>
  <rdf:Description rdf:about='http://somewhere/JohnSmith/'>
    <vcard:FN>John Smith</vcard:FN>
    <vcard:N rdf:nodeID="A0"/>
  </rdf:Description>
  ...
</rdf:RDF>
```

# Navigating a model

```
// retrieve the John Smith vcard resource from the model  
Resource vcard = model.getResource(johnSmithURI);
```

Three ways of retrieving property values:

```
// retrieve the value of the N property  
Resource name = (Resource) vcard.getProperty(VCARD.N).getObject();
```

```
// retrieve the value of the N property  
Resource name = vcard.getProperty(VCARD.N).getResource();
```

```
// retrieve the given name property  
String fullName = vcard.getProperty(VCARD.N).getString();
```

# Multiple values in properties

```
// add two nickname properties to vcard
vcard.addProperty(VCARD.NICKNAME, "Smithy")
    .addProperty(VCARD.NICKNAME, "Adman");

// set up the output
System.out.println("The nicknames of \"" + fullName + "\" are:");

// list the nicknames
StmtIterator iter = vcard.listProperties(VCARD.NICKNAME);
while (iter.hasNext()) {
    System.out.println(" " + iter.nextStatement().getObject().toString());
}
```

# Querying a model

```
// select all the resources with a VCARD.FN property
ResIterator iter = model.listSubjectsWithProperty(VCARD.FN);
if (iter.hasNext()) {
    System.out.println("The database contains vcards for:");
    while (iter.hasNext()) {
        System.out.println(" " + iter.nextStatement()
                           .getProperty(VCARD.FN).getString());
    }
} else {
    System.out.println("No vcards were found in the database");
}
```

```
The database contains vcards for:
Sarah Jones
John Smith
Matt Jones
Becky Smith
```



# Create resources

```
// URI declarations
String familyUri = "http://family/";
String relationshipUri = "http://purl.org/vocab/relationship/";

// Create an empty Model
Model model = ModelFactory.createDefaultModel();

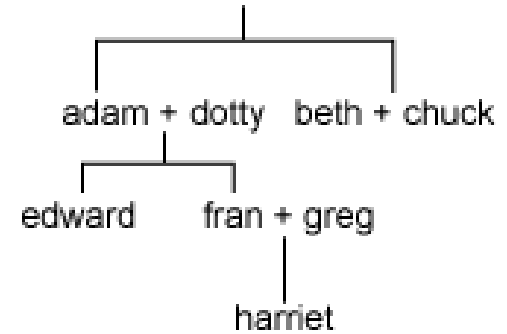
// Create a Resource for each family member, identified by their URI
Resource adam = model.createResource(familyUri+"adam");
Resource beth = model.createResource(familyUri+"beth");
Resource dotty = model.createResource(familyUri+"dotty");
// and so on for other family members

// Create properties for the different types of relationship to represent
Property childOf = model.createProperty(relationshipUri,"childOf");
Property parentOf = model.createProperty(relationshipUri,"parentOf");
Property siblingOf = model.createProperty(relationshipUri,"siblingOf");
Property spouseOf = model.createProperty(relationshipUri,"spouseOf");

// Add properties to adam describing relationships to other family members
adam.addProperty(siblingOf,beth);
adam.addProperty(spouseOf,dotty);
adam.addProperty(parentOf,edward);

// Can also create statements directly . . .
Statement statement = model.createStatement(adam,parentOf,fran);

// but remember to add the created statement to the model
model.add(statement);
```



# Querying a model

```
// List everyone in the model who has a child:
ResIterator parents = model.listSubjectsWithProperty(parentOf);

// Because subjects of statements are Resources, the method returned a ResIterator
while (parents.hasNext()) {

    // ResIterator has a typed nextResource() method
    Resource person = parents.nextResource();

    // Print the URI of the resource
    System.out.println(person.getURI());
}

// Can also find all the parents by getting the objects of all "childOf" statements
// Objects of statements could be Resources or literals, so the Iterator returned
// contains RDFNodes
NodeIterator moreParents = model.listObjectsOfProperty(childOf);

// To find all the siblings of a specific person, the model itself can be queried
NodeIterator siblings = model.listObjectsOfProperty(edward, siblingOf);

// But it's more elegant to ask the Resource directly
// This method yields an iterator over Statements
StmtIterator moreSiblings = edward.listProperties(siblingOf);
```

# Using selectors to query a model

```
// Find the exact statement "adam is a spouse of dotty"
```

```
model.listStatements(adam, spouseOf, dotty);
```

```
// Find all statements with adam as the subject and dotty as the object
```

```
model.listStatements(adam, null, dotty);
```

```
// Find any statements made about adam
```

```
model.listStatements(adam, null, null);
```

```
// Find any statement with the siblingOf property
```

```
model.listStatements(null, siblingOf, null);
```

# Exercise



## •Objective

- Understand how to use an RDF(S) management API

## •Tasks

- Read an ontology in RDF(S) from two files:
  - GP\_Santiago.rdfs (conceptualization)
  - GP\_Santiago.rdf (instances)
- Write the class hierarchy of the ontology, including the instances of each class:

```
Class Practica2:MedioTransporte
  Class Practica2:Tren
  Class Practica2:Bicicleta
    Instance Practica2:GP_Santiago_Instance_70
  Class Practica2:Automovil
  Class Practica2:AutoBus
  Class Practica2:APie
Class Practica2:InfraEstructuraTransporte
  Class Practica2:ViaFerreá
  Class Practica2:Sendero
  Class Practica2:Carretera
    Instance Practica2:A6
...
```



# Set up

- **Requirements:**
    - Java JDK 5
    - Eclipse (optional)
    - Ant (optional)
    - Material at: <http://delicias.dia.fi.upm.es/wiki/index.php/Master09-10>
  - **Create a directory for your project**
  - **Copy Jena libraries:**
    - Unzip *Jena-2.6.2.zip/lib* in the project directory
  - **Copy the ontologies:**
    - Copy *ontologies/rdf* in the project directory
- } Or copy the JenaProjectTemplate directory in your computer
- **With Eclipse:**
    - Create a new Java project (from existing source)
    - Append the Jena libraries to your classpath if needed (check JDK libs)
    - Write Java code using the Jena API  
<http://jena.sourceforge.net/javadoc/index.html>
    - Compile
    - Run
  - **With Ant:**
    - Write Java code using the Jena API  
<http://jena.sourceforge.net/javadoc/index.html>
    - Run “ant”



# Hints

- **Create ontology model:**
  - `public static OntModel  
createOntologyModel (OntModelSpec spec)`
- **Read the ontology in the file**
  - `Model read (java.lang.String url)`
- **Add all the statements in another model to this model**
  - `Model add (Model m)`



## More hints

- **List root classes**
  - `ExtendedIterator listHierarchyRootClasses()`
- **List subclasses of a class**
  - `ExtendedIterator listSubClasses(boolean direct)`
- **List instances of a class**
  - `ExtendedIterator listInstances(boolean direct)`

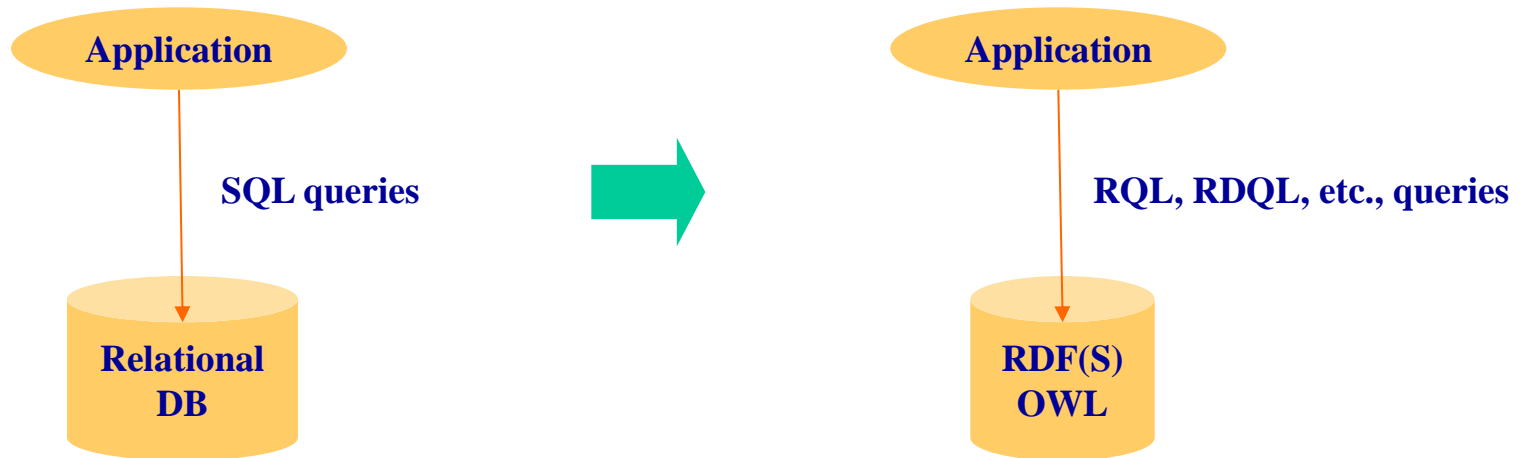
# Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
  - 5.1 RDF(S) query languages and SPARQL
  - 5.2 Turtle RDF syntax
  - 5.3 Graph patterns
  - 5.4 Restricting values and solutions
  - 5.5 SPARQL query forms
  - 5.6 Hands-on activity



# RDF(S) query languages

- **Languages developed to allow accessing datasets expressed in RDF(S) (and in some cases OWL)**

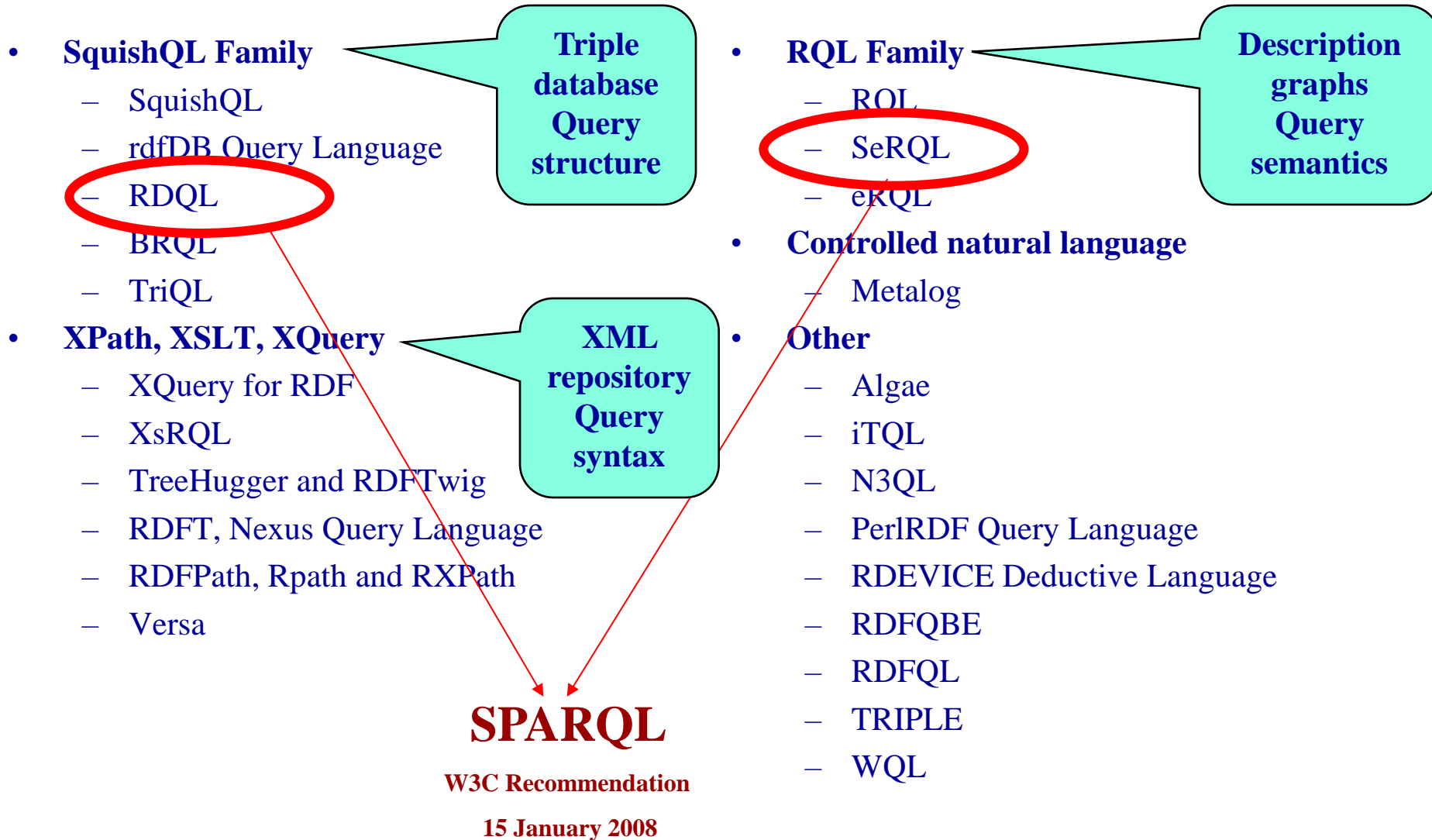


- **Supported by the most important language APIs**
  - Jena (HP labs)
  - Sesame (Aduna)
  - Boca (IBM)
  - ...
- **There are some differences wrt languages like SQL, such as**
  - Combination of different sources
  - Trust management
  - Open World Assumption

# Query types

- **Selection and extraction**
  - “Select all the essays, together with their authors and their authors’ names”.
  - “Select everything that is related to the book ‘Bellum Civile’”
- **Reduction**: we specify what it should not be returned
  - “Select everything except for the ontological information and the book translators”
- **Aggregation**
  - “Return all the essays together with the mean number of authors per essay”
- **Restructuring**: the original structure is changed in the final result
  - “Invert the relationship ‘has author’ by ‘is author of’”
- **Combination and inferences**
  - “Combine the information of a book called ‘La guerra civil’ and whose author is Julius Caesar with the book whose identifier is ‘Bellum Civile’”
  - “Select all the essays, together with its authors and author names”, *including also the instances of the subclasses of Essay*.
  - “Obtain the relationship ‘coauthor’ among persons who have written the same book”.

# RDF(S) query language families



# SPARQL

- **SPARQL Protocol and RDF Query Language**
- **Supported by:** Jena, Sesame, IBM Boca, etc.
- **Features**
  - It supports most of the aforementioned queries
  - It supports **datatype reasoning** (datatypes can be requested instead of actual values)
  - **The domain vocabulary and the knowledge representation vocabulary are** treated differently by the query interpreters.
  - It allows making queries over properties with multiple values, over multiple properties of a resource and over **reifications**
  - Queries can contain **optional statements**
  - Some implementations support **aggregation queries**
- **Limitations**
  - Neither **set operations** nor **existential or universal quantifiers** can be included in the queries
  - It does not support **recursive queries**

# SPARQL is also a protocol

- SPARQL is a Query Language ...:

*Find names and websites of contributors to PlanetRDF:*

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?website
FROM <http://planetrdf.com/bloggers.rdf>
WHERE {
    ?person foaf:weblog ?website .
    ?person foaf:name ?name .
    ?website a foaf:Document }
```

- ... and a Protocol.

```
http://.../qps?query-lang=http://www.w3.org/TR/rdf-sparql-query/
&graph-id=http://planetrdf.com/bloggers.rdf&query=PREFIXfoaf:
<http://xmlns.com/foaf/0.1/...
```

- Services running SPARQL queries over a set of graphs
- A transport protocol for invoking the service
- Based on ideas from earlier protocol work such as Joseki
- Describing the service with Web Service technologies

# SPARQL Endpoints

- **SPARQL protocol services**
  - Enables users (human or other) to query a knowledge base using SPARQL
  - Results are typically returned in one or more machine-processable formats.
- **List of SPARQL Endpoints**
  - <http://esw.w3.org/topic/SparqlEndpoints>
- **Programmatic access using libraries:**
  - ARC, RAP, Jena, Sesame, Javascript SPARQL, PySPARQL, etc.
- **Examples:**

Project	Endpoint
BBC Programmes and Music	<a href="http://bbc.openlinksw.com/sparql/">http://bbc.openlinksw.com/sparql/</a>
DBLP Bibliography Database	<a href="http://www4.wiwiss.fu-berlin.de/dblp/sparql">http://www4.wiwiss.fu-berlin.de/dblp/sparql</a>
DBpedia	<a href="http://dbpedia.org/sparql">http://dbpedia.org/sparql</a>
Musicbrainz	<a href="http://dbtune.org/musicbrainz/sparql">http://dbtune.org/musicbrainz/sparql</a>
U.S. Census	<a href="http://www.rdfabout.com/sparql">http://www.rdfabout.com/sparql</a>

# A simple SPARQL query

## Data:

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

## Query:

```
SELECT ?title  
WHERE  
{  
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .  
}
```

## Query result:

title
"SPARQL Tutorial"

- A pattern is *matched* against the RDF data
- Each way a pattern can be matched yields a solution
- The sequence of solutions is filtered by: Project, distinct, order, limit/offset
- One of the result forms is applied: SELECT, CONSTRUCT, DESCRIBE, ASK

# Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
  - 5.1 RDF(S) query languages and SPARQL
  - 5.2 Turtle RDF syntax
  - 5.3 **Graph patterns**
  - 5.4 Restricting values and solutions
  - 5.5 SPARQL query forms
  - 5.6 Hands-on activity



# Graph patterns

- **Basic Graph Patterns**, where a set of triple patterns must match
- **Group Graph Pattern**, where a set of graph patterns must all match
- **Optional Graph patterns**, where additional patterns may extend the solution
- **Alternative Graph Pattern**, where two or more possible patterns are tried
- **Patterns on Named Graphs**, where patterns are matched against named graphs

# Basic graph patterns: Multiple matches

```
@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> .
```

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

```

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

# Basic graph patterns: Matching RDF literals

```
@prefix dt:    <http://example.org/datatype#> .
@prefix ns:    <http://example.org/ns#> .
@prefix :      <http://example.org/ns#> .
@prefix xsd:   <http://www.w3.org/2001/XMLSchema#> .

:x    ns:p      "cat"@en .
:y    ns:p      "42"^^xsd:integer .
:z    ns:p      "abc"^^dt:specialDatatype .
```

```
SELECT ?v WHERE { ?v ?p "cat" }
```

v

```
SELECT ?v WHERE { ?v ?p "cat"@en }
```

v

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

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

v

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

```
SELECT ?v WHERE { ?v ?p "abc"^^<http://example.org/datatype#specialDatatype> }
```

v

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

# Basic graph patterns: Blank node labels in query results

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

```
_:a foaf:name "Alice" .
```

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

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

```
SELECT ?x ?name
```

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

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

=

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

# Group graph pattern

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

```
SELECT ?x
WHERE {}
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
        { ?x foaf:mbox ?mbox . FILTER regex(?name, "Smith") }
      }
```

# Optional graph patterns

```
@prefix foaf:      <http://xmlns.com/foaf/0.1/> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a  rdf:type      foaf:Person .
_:a  foaf:name     "Alice" .
_:a  foaf:mbox     <mailto:alice@example.com> .
_:a  foaf:mbox     <mailto:alice@work.example> .

_:b  rdf:type      foaf:Person .
_:b  foaf:name     "Bob" .
```

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

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

# Multiple optional graph patterns

```
@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/>
SELECT ?name ?mbox ?hpage
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox } .
        OPTIONAL { ?x foaf:homepage ?hpage }
}
```

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

# Alternative graph patterns

```
@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)" .
```

```
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 } }
```

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

```
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"	

```
SELECT ?title ?author
WHERE
{ { ?book dc10:title ?title . ?book dc10:creator ?author }
  UNION
  { ?book dc11:title ?title . ?book dc11:creator ?author } }
```

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



# Patterns on named graphs

```
# Named graph: http://example.org/foaf/aliceFoaf
@prefix foaf:<http://.../foaf/0.1/> .
@prefix rdf:<http://.../1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://.../2000/01/rdf-schema#> .

_:a foaf:name      "Alice" .
_:a foaf:mbox      <mailto:alice@work.example> .
_:a foaf:knows     _:b .

_:b foaf:name      "Bob" .
_:b foaf:mbox      <mailto:bob@work.example> .
_:b foaf:nick      "Bobby" .
_:b rdfs:seeAlso   <http://example.org/foaf/bobFoaf> .

<http://example.org/foaf/bobFoaf>
  rdf:type         foaf:PersonalProfileDocument .
```

```
# Named graph: http://example.org/foaf/bobFoaf
@prefix foaf:<http://.../foaf/0.1/> .
@prefix rdf:<http://.../1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:<http://.../2000/01/rdf-schema#> .

_:z foaf:mbox      <mailto:bob@work.example> .
_:z rdfs:seeAlso   <http://example.org/foaf/bobFoaf> .
_:z foaf:nick      "Robert" .

<http://example.org/foaf/bobFoaf>
  rdf:type         foaf:PersonalProfileDocument .
```

# Patterns on named graphs II

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

```
SELECT ?src ?bobNick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH ?src
  { ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?bobNick
  }
}
```

src	bobNick
<http://example.org/foaf/aliceFoaf>	"Bobby"
<http://example.org/foaf/bobFoaf>	"Robert"

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

```
SELECT ?nick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH data: bobFoaf {
    ?x foaf:mbox <mailto:bob@work.example> .
    ?x foaf:nick ?nick
  }
}
```

nick
"Robert"

# Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
  - 5.1 RDF(S) query languages and SPARQL
  - 5.2 Turtle RDF syntax
  - 5.3 Graph patterns
  - 5.4 **Restricting values and solutions**
  - 5.5 SPARQL query forms
  - 5.6 Hands-on activity

# Restricting values

```
@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 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
       FILTER regex(?title, "^SPARQL")
}
```

title
-------

"SPARQL Tutorial"
-------------------

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title
       FILTER regex(?title, "web", "i" )
}
```

title
-------

"The Semantic Web"
--------------------

```
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 . }
```

title	price
-------	-------

"The Semantic Web"	23
--------------------	----

## Value tests

- Based on XQuery 1.0 and XPath 2.0 Function and Operators
- XSD boolean, string, integer, decimal, float, double, dateTime
- Notation <, >, =, <=, >= and != for value comparison  
Apply to any type
- BOUND, isURI, isBLANK, isLITERAL
- REGEX, LANG, DATATYPE, STR (lexical form)
- Function call for casting and extensions functions

# Solution sequences and modifiers

- **Order modifier:** put the solutions in order
- **Projection modifier:** choose certain variables
- **Distinct modifier:** ensure solutions in the sequence are unique
- **Reduced modifier:** permit elimination of some non-unique solutions
- **Limit modifier:** restrict the number of solutions
- **Offset modifier:** control where the solutions start from in the overall sequence of solutions

```
SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC (?emp)
```

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

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

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

```
SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

```
SELECT ?name WHERE { ?x foaf:name ?name }
ORDER BY ?name
LIMIT 5
OFFSET 10
```

# Table of Contents

1. An introduction to knowledge representation formalisms
2. Resource Description Framework (RDF)
3. RDF Schema
4. RDF(S) management APIs
5. **RDF(S) query languages: SPARQL**
  - 5.1 RDF(S) query languages and SPARQL
  - 5.2 Turtle RDF syntax
  - 5.3 Graph patterns
  - 5.4 Restricting values and solutions
  - 5.5 **SPARQL query forms**
  - 5.6 Hands-on activity

# SPARQL query forms

- **SELECT**
  - Returns all, or a subset of, the variables bound in a query pattern match.
- **CONSTRUCT**
  - Returns an RDF graph constructed by substituting variables in a set of triple templates.
- **ASK**
  - Returns a boolean indicating whether a query pattern matches or not.
- **DESCRIBE**
  - Returns an RDF graph that describes the resources found.



# SPARQL query forms: SELECT

```
@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"

# SPARQL query forms: CONSTRUCT

```
@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 }
```

## Query result:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .  
  
<http://example.org/person#Alice> vcard:FN "Alice" .
```

# SPARQL query forms: ASK

```
@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" }
```

Query result:

yes

# SPARQL query forms: DESCRIBE

```
PREFIX ent: <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

## Query result:

```
@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_shalsum      "ABCD1234" ;
         vcard:N
           [ vcard:Family        "Smith" ;
             vcard:Given         "John" ] .

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

# Exercise



## •Objective

- Understand how to perform SPARQL queries

## •Tasks

- Perform a set of SPARQL queries over the sample ontology.
  - Browse to:
    - <http://my.computer.ip:8080/openrdf-workbench>
  - Select repository GP\_InMemoryRDFS
  - Select the Query option from the left menu





# Queries on the model

## 1) Get all the classes

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x WHERE { ?x a rdfs:Class. }
```

## 2) Get the subclasses of the class *Establecimiento*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x rdfs:subClassOf pr:Establecimiento. }
```

## 3) Get the instances of the class *Ciudad*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x a pr:Ciudad. }
```



# Queries on the instances

## 4) Get the number of inhabitants of *Santiago de Compostela*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { pr:Santiago_de_Compostela pr:NumeroHabitantes ?x. }
```

## 5) Get the number of inhabitants of *Santiago de Compostela* and of *Arzua*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE {
    {pr:Santiago_de_Compostela pr:NumeroHabitantes ?x.}
    UNION
    {pr:Arzua pr:NumeroHabitantes ?x.}
}
```

## 6) Get different places with the inhabitants number, ordering the results by the name of the place (ascending)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio pr:NumeroHabitantes ?y;
                      rdfs:label ?x.}

ORDER BY ASC(?x)
```





## Queries on the instances II

### 7) Get all the instances of *Localidad* with their inhabitant number (if it exists)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio a pr:Localidad;
                        rdfs:label ?x.
                        OPTIONAL {$sitio pr:NumeroHabitantes ?y.} }
```

### 8) Get all the places with more than 200.000 inhabitants

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x ?y WHERE { $sitio pr:NumeroHabitantes ?y;
                        rdfs:label ?x.
                        FILTER(?y > 200000) }
```

### 9) Get postal data of *Pazo de Breogan* (calle, número, localidad, provincia)

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?calle ?numero ?poblacion ?provincia
WHERE { pr:Pazo_Breogan pr:SituadoEn $pob;
        pr:Direccion $dir.
        $pob rdfs:label ?poblacion;
        pr:Provincia ?provincia.
        $dir pr:Calle ?calle;
        pr:Numero ?numero.}
```



# Queries with inference

## 10) Get the subclasses of class *Lugar*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x rdfs:subClassOf pr:Lugar. }
```

## 11) Get the instances of class *Localidad*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT ?x WHERE { ?x a pr:Localidad. }
```

## Special query (SELECT \*)

## 12) Get the values of all the variables in the query

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
SELECT * WHERE { ?x pr:NumeroHabitantes ?y. }
```



# Different query forms

## 13) Describe the resource with *rdfs:label* "Madrid"

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
DESCRIBE ?x WHERE { ?x rdfs:label "Madrid". }
```

## 14) Construct the RDF(S) graph that directly relates all the touristic places with their respective provinces, using a new property called "estaEn".

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
CONSTRUCT {?x pr:estaEn ?y}
WHERE {
    ?x a pr:LugarTuristico;
        pr:SituadoEn $pob.
    $pob pr:Provincia ?y. }
```

## 15) Ask if there is some instance of *Pueblo*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
ASK WHERE {?a a pr:Pueblo}
```

## 16) Ask if there is some instance of *Ermita*

```
PREFIX pr: <http://GP-onto.fi.upm.es/Practica2#>
ASK WHERE {?a a pr:Ermita}
```