Lecture Notes

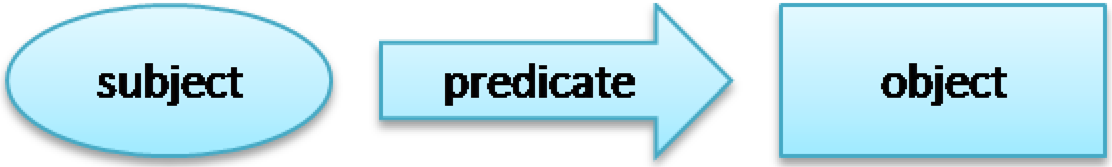
Big Data in Medical Informatics

Week 9:

Querying Biomedical Data in Semantic Web

Recall : RDF

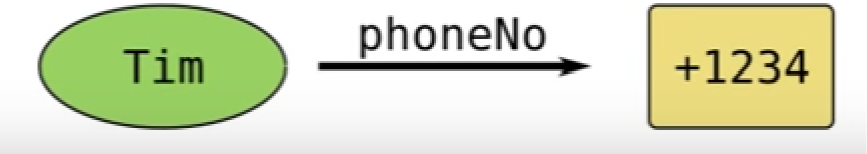
- RDF is a data model of graphs of subject, predicate, object triples.



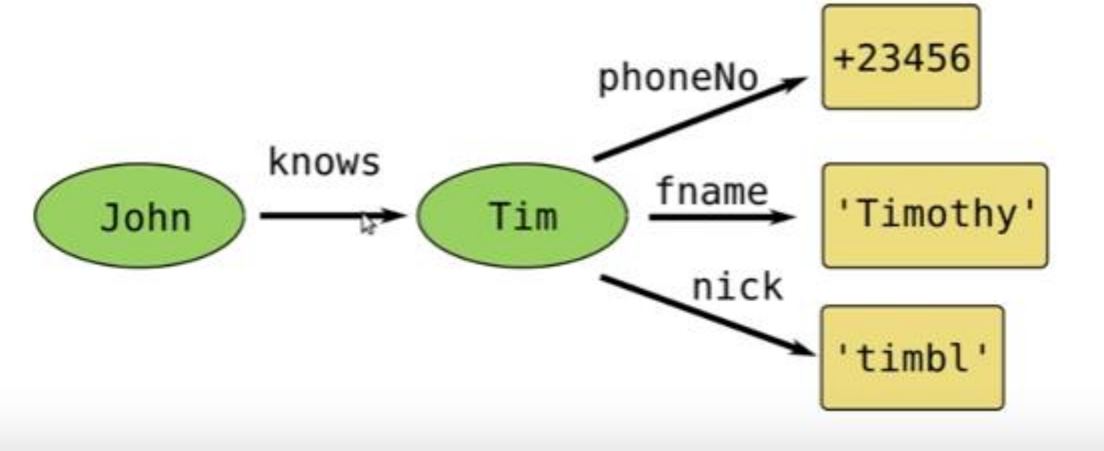
- Subjects, predicates, and objects are represented with URIs, which can abbreviated as prefixed names
- Objects can also be literals: strings, integers, Booleans, etc.

Triples and Graphs

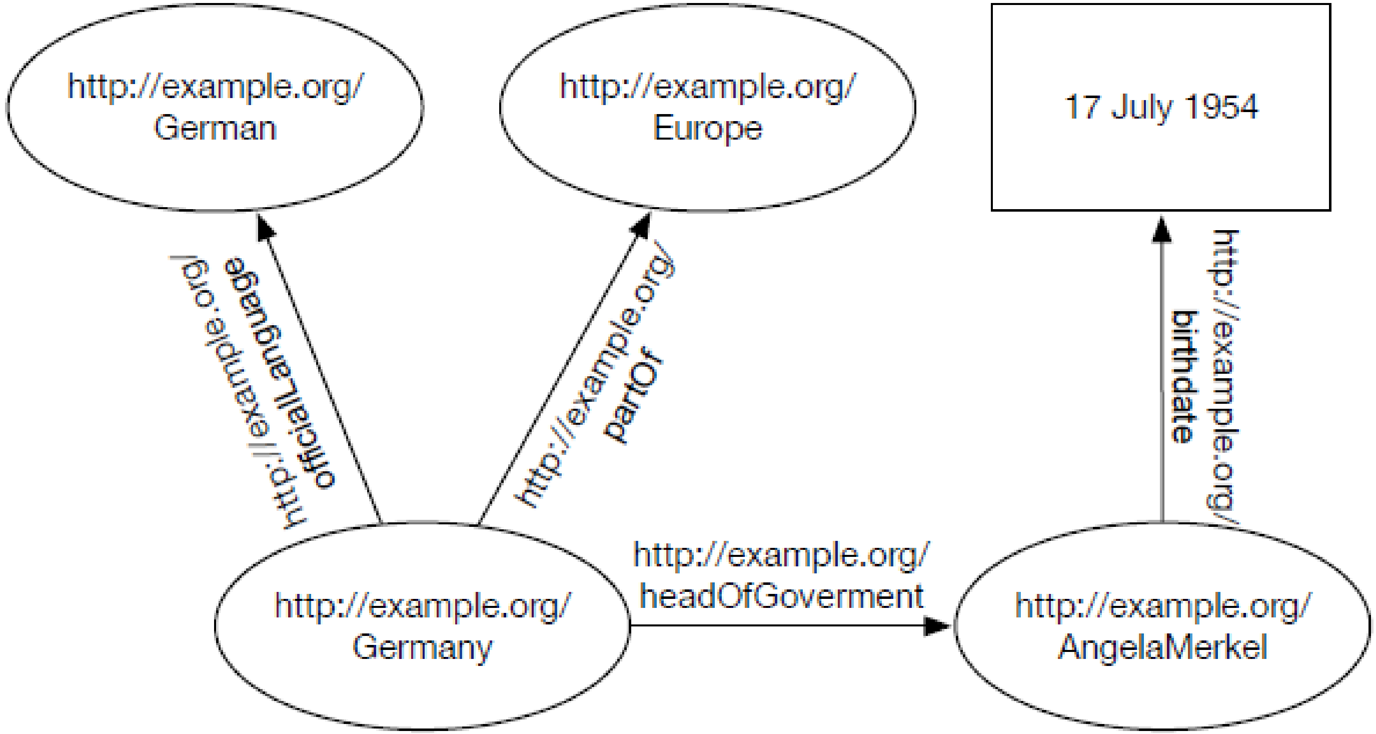
- Triples are the statements about resources using URIs and literal values



- Collection of triples creates graph



Example of an RDF graph

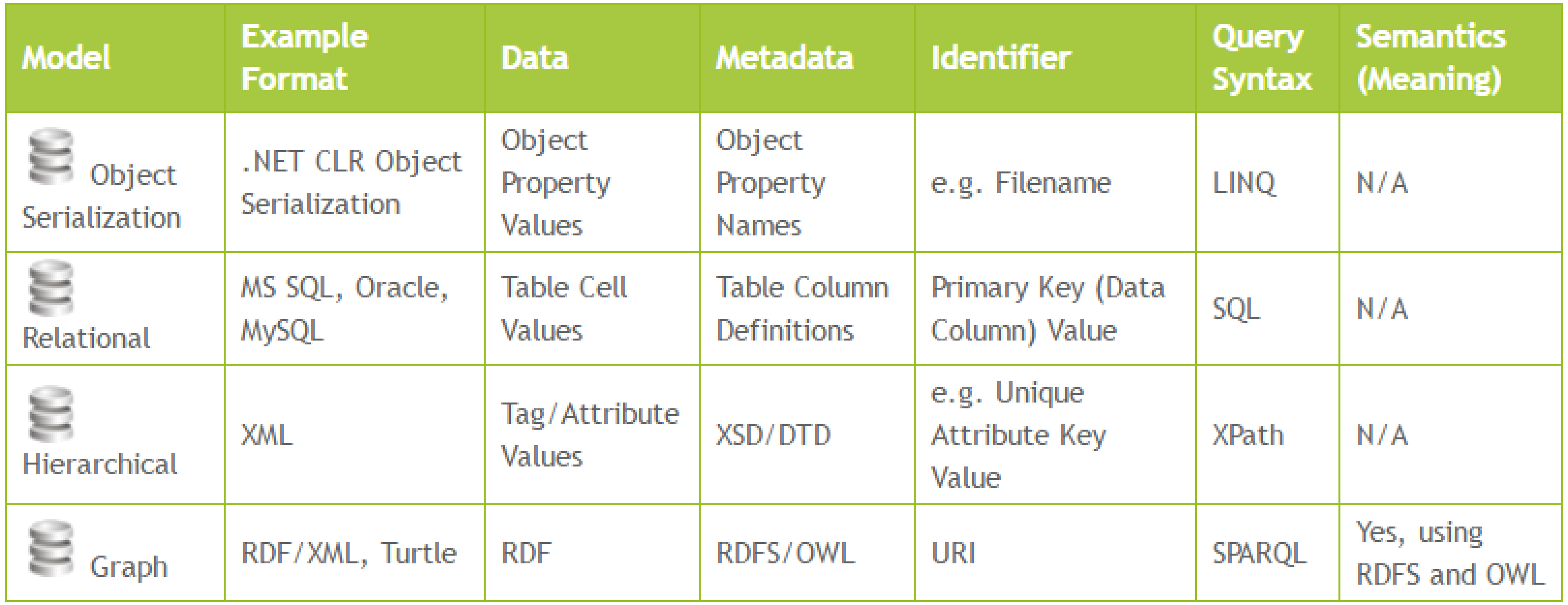

http://example.org/ German
http://example.org/ Europe
17 July 1954
http://example.org/ officialLanguage
http://example.org/ partOf
http://example.org/ birthdate
http://example.org/ Germany
http://example.org/ headOfGoverment
http://example.org/ AngelaMerkel

Source: Nicola Gatto.,: “RDF Data Model”

Semantic Data

- Comparing The Popular Data Models

Comparing the features of the mainstream ways of modeling data versus the semantic web model.



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	Primary Key (Data Column) Value	SQL	N/A
Hierarchical	XML	Tag/Attribute Values	XSD/DTD	e.g. Unique Attribute Key Value	XPath	N/A
Graph	RDF/XML, Turtle	RDF	RDFS/OWL	URI	SPARQL	Yes, using RDFS and OWL

Source: http://www.linkeddatatools.com/introducing-rdfs-owl

Introducing RDFS & OWL

- Vocabulary - A collection of terms given a well-defined meaning that is consistent across contexts.
- Ontology - Allows you to define contextual relationships behind a defined vocabulary. It is the cornerstone of defining a knowledge domain. A formal syntax for defining ontologies is OWL (Web Ontology Language) which is an extension to RDFS (RDF Schema)

- Metadata is as accessible to SPARQL queries as the data itself
- *Dublin Core Metadata Initiative* is a standard set of basic metadata itself

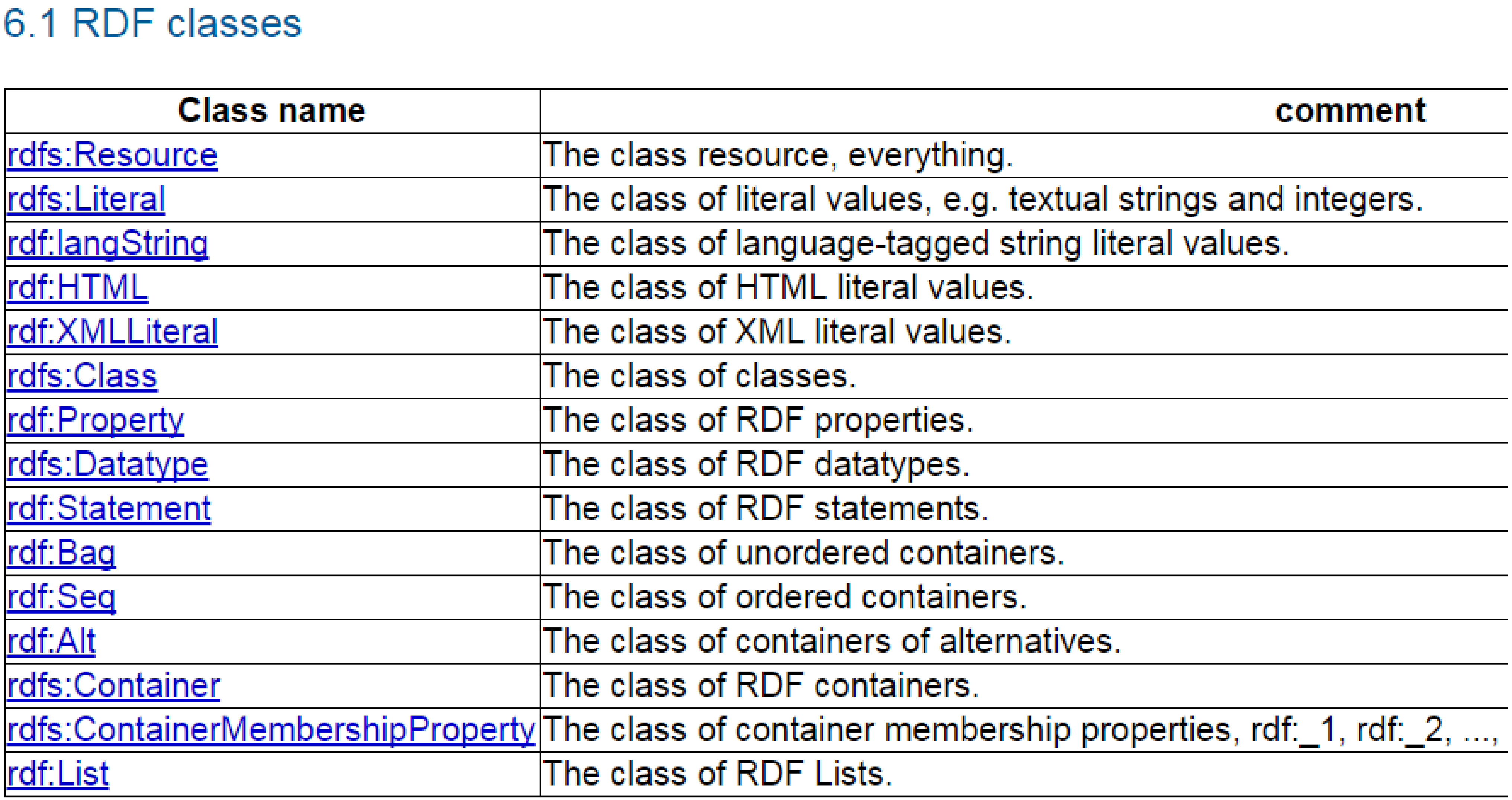
RDFS

- RDF Schema
- intended to structure RDF resources.
- set of classes with certain properties using the RDF extensible knowledge representation data model
- Provide basic elements for the description of ontologies, and RDF vocabularies,
- RDFS Syntax
  - **RDF:type**
  - Example:MyHondaCivic RDF:type Example:PassengerVehicle

  - **RDF:Class and RDFS:Property**
    - Example:UsedCarForSale RDF:type RDF:Class
    - Example:Price RDF:type RDFS:Property

  - **RDFS:Domain and RDFS:Range**
    - Example:Price RDFS:Domain Example:UsedCarForSale
    - Example:Price RDFS:Range XSD:int
  - **Other RDFS Basics**
    - **RDFS:Label** – A string of text describing the resource
    - **RDFS:Comment** – A potentially longer comment about the resource
    - **RDFS:SeeAlso –** Links to other "relevant" resources

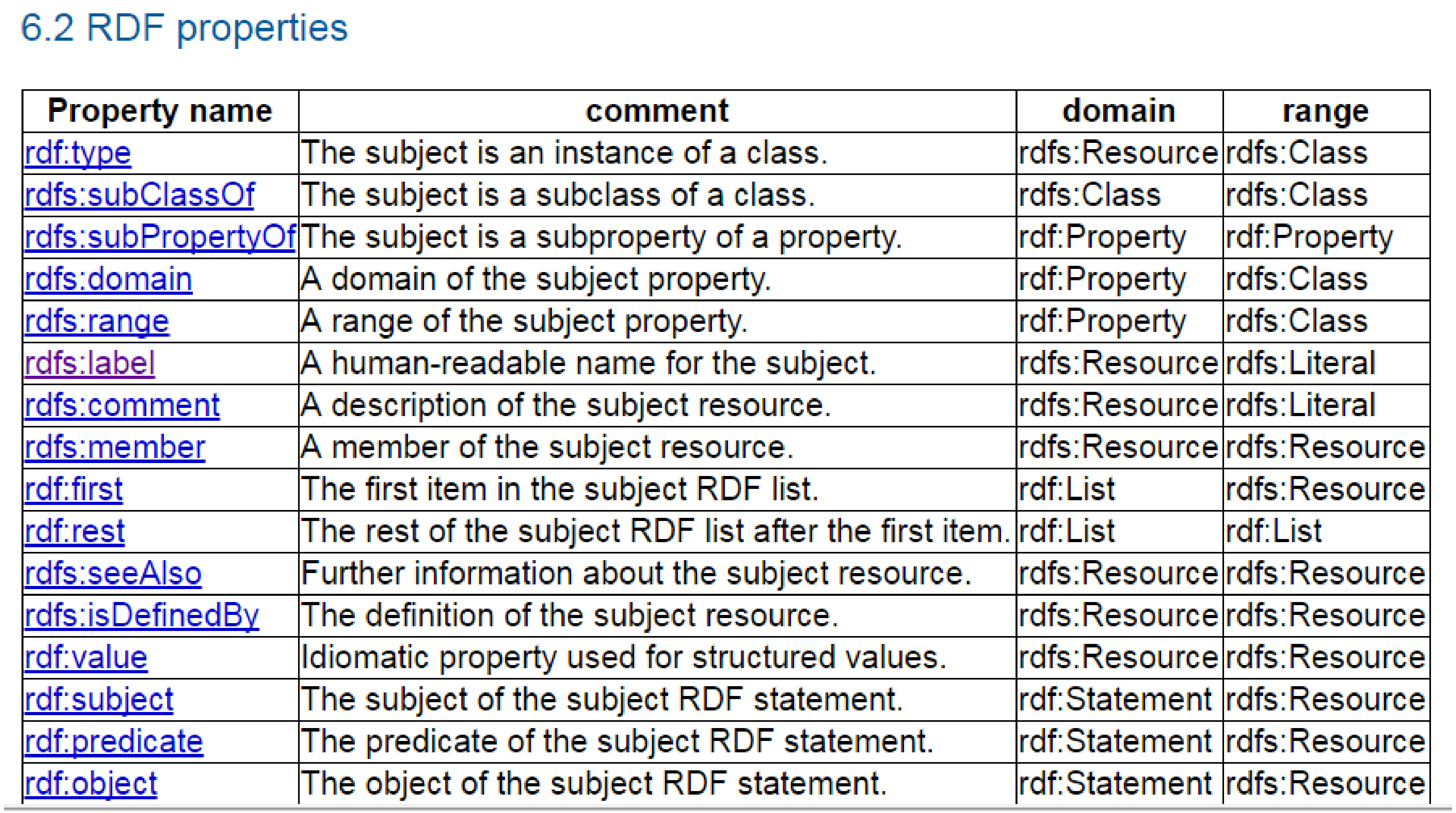
https://www.w3.org/TR/rdf-schema/



6.1 RDF classes

Class name	comment
rdfs:Resource	The class resource, everything.
rdfs:Literal	The class of literal values, e.g. textual strings and integers.
rdf:langString	The class of language-tagged string literal values.
rdf:HTML	The class of HTML literal values.
rdf:XMLLiteral	The class of XML literal values.
rdfs:Class	The class of classes.
rdf:Property	The class of RDF properties.
rdfs:Datatype	The class of RDF datatypes.
rdf:Statement	The class of RDF statements.
rdf:Bag	The class of unordered containers.
rdf:Seq	The class of ordered containers.
rdf:Alt	The class of containers of alternatives.
rdfs:Container	The class of RDF containers.
rdfs:ContainerMembershipProperty	The class of container membership properties, rdf:_1, rdf:_2, ...,
rdf:List	The class of RDF Lists.

https://www.w3.org/TR/rdf-schema/



6.2 RDF properties

Property name	comment	domain	range
rdf:type	The subject is an instance of a class.	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:member	A member of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:first	The first item in the subject RDF list.	rdf:List	rdfs:Resource
rdf:rest	The rest of the subject RDF list after the first item.	rdf:List	rdf:List
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values.	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

Metadata Initiatives

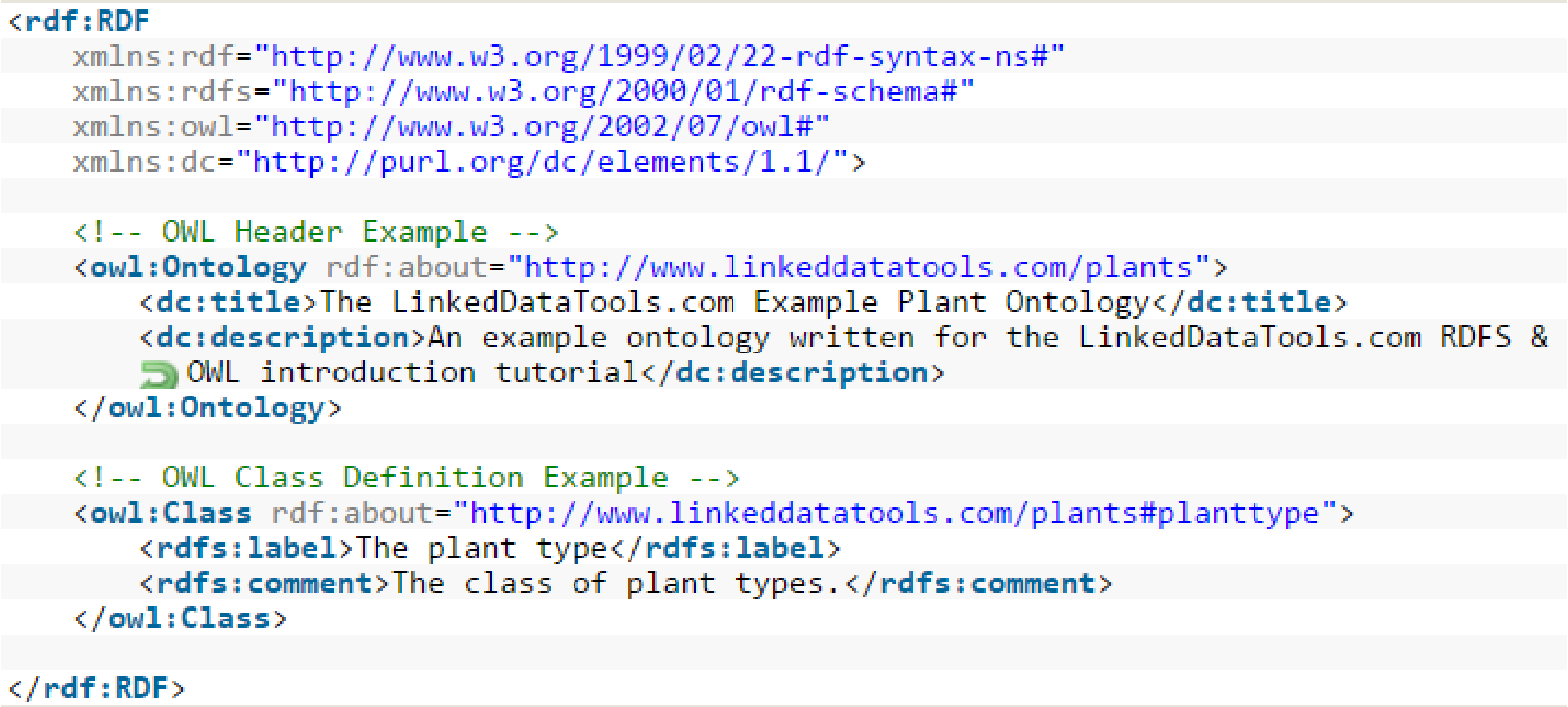
- Standard vocabularies, or formal ontologies representing terms within a domain of knowledge, are already available freely from various organisations dedicated to creating standard vocabularies for a range of subjects

Examples:

- Dublin Core Metadata Initiative (DCMI) - Creates ontologies for a range of subjects, particularly focusing on common, every day terms and terms important in media.

- Friend Of A Friend (FOAF) - focuses on developing a standard vocabulary/ontology for social networking purposes.

Example



```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:dc="http://purl.org/dc/elements/1.1/">

    <!-- OWL Header Example -->
    <owl:Ontology rdf:about="http://www.linkeddatatools.com/plants">
        <dc:title>The LinkedDataTools.com Example Plant Ontology</dc:title>
        <dc:description>An example ontology written for the LinkedDataTools.com RDFS &
        OWL introduction tutorial</dc:description>
    </owl:Ontology>

    <!-- OWL Class Definition Example -->
    <owl:Class rdf:about="http://www.linkeddatatools.com/plants#planttype">
        <rdfs:label>The plant type</rdfs:label>
        <rdfs:comment>The class of plant types.</rdfs:comment>
    </owl:Class>

</rdf:RDF>
```

Source: http://www.linkeddatatools.com/introducing-rdfs-owl

Querying RDF

SPARQL: The query language of the Semantic Web

- SPARQL: SPARQL Protocol And Query Language
- SPARQL is a W3C recommendation that is part of the semantic web stack
- A SPARQL query allows you to search linked data based on the structure of the triples it contains
- SPARQL can be used to explore the structure of RDF graphs and to transform linked data
- SPARQL queries are executed against RDF datasets, consisting of RDF graphs

SPARQL: The query language of the Semantic Web

SPARQL helps us to:

- Pull values from *structured and semi-structured data*
- Explore data by querying *unknown relationships*
- Perform *complex joins of disparate databases* in a single, simple query
- *Transform RDF data* from one vocabulary to another

SPARQL Landscape

SPARQL 1.0 became a standard in January, 2008, and included:

- **SPARQL 1.0 Query Language** - for matching patterns in RDF data
- **SPARQL 1.0 Protocol** - for sending queries over HTTP
- **SPARQL Results XML Format**
- **SPARQL Results JSON Format**

SPARQL 1.1 became a standard in March, 2013, and included:

- **Updated 1.1 versions of SPARQL Query and SPARQL Protocol**
- **SPARQL 1.1 Update** - for inserting, deleting, modifying RDF data
- **SPARQL 1.1 Graph Store HTTP Protocol** - RESTful access of RDF graphs
- **SPARQL 1.1 Service Descriptions** - describe capabilities of SPARQL endpoints
- **SPARQL 1.1 Entailments** - how to combine reasoning with SPARQL
- **SPARQL 1.1 Basic Federated Query** - querying multiple endpoints at once
- **SPARQL Results CSV/TSV Formats**

SPARQL Architecture & Endpoints

- A SPARQL endpoint accepts queries and returns results via HTTP.
  - Generic Endpoints will query any Web-accessible RDF data
  - Specific Endpoints are hardwired to query against particular datasets
- The results of SPARQL queries can be returned in a variety of formats:
  - XML
  - JSON
  - CSV/TSV
  - RDF
  - HTML

Anatomy of SPARQL query

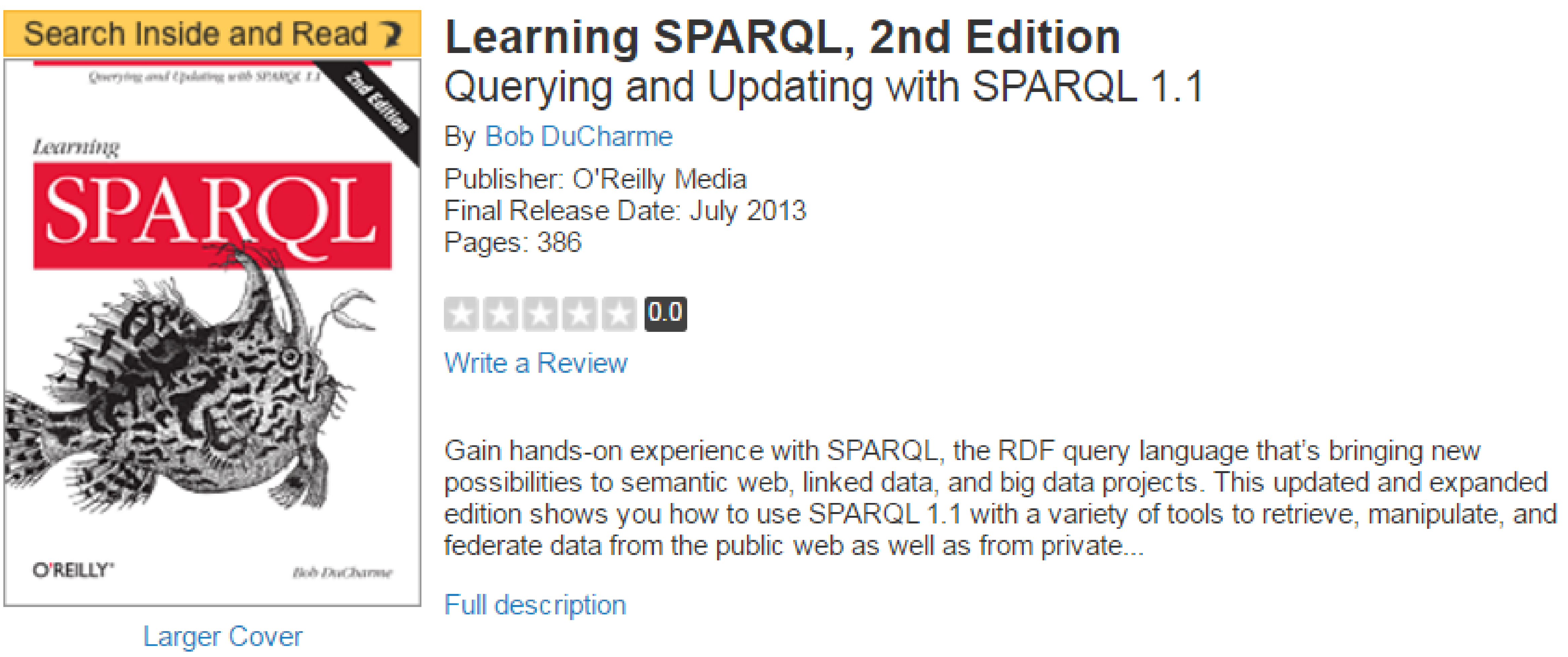
- Prefix Declarations: Shortcuts for URIs used in the query (e.g. rdf, rdfs, bio2rdf)
- Dataset Definition: RDF graph to query (support for this option is SPARQL endpoint engine dependent)
- Result Clause: Data returned by the query
- Query Pattern: Graph Pattern used to search the RDF data
- Query Modifier: Limiting, ordering, other forms of result rearrangements.

Anatomy of SPARQL query

```
#prefix declarations
PREFIX prefixA: <http://example.org/prefixA#>
PREFIX prefixB: <http://example.org/prefixB:>
#result clause
SELECT …
#dataset definition
FROM <http://example.org/myDataset>
#query pattern
WHERE{
    .....
}
#query modifiers
LIMIT 10
```

SPARQL queries over >1 endpoint use SERVICE keyword

```
#comments can be included
PREFIX prefixA: <http://example.org/prefixA#>
PREFIX prefixB: <http://example.org/prefixB:>
SELECT …
FROM <http://example.org/myDataset>
WHERE{
    SERVICE <http://somewhere.org/sparql> {
        .....
     }
}
LIMIT 10
```



Learning SPARQL, 2nd Edition

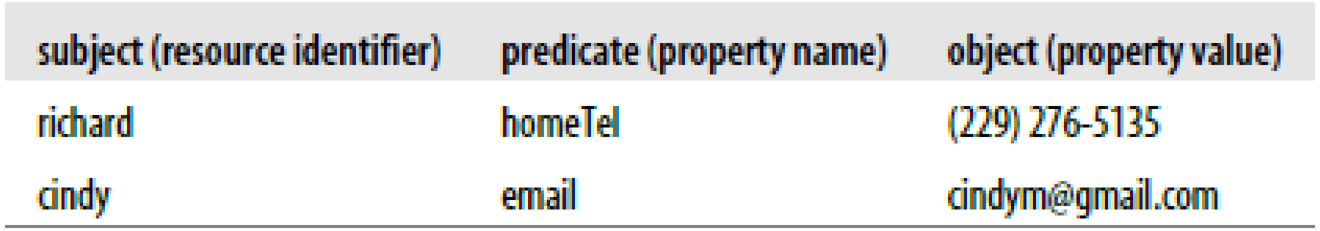
Querying and Updating with SPARQL 1.1

By Bob DuCharme

Publisher: O'Reilly Media
Final Release Date: July 2013
Pages: 386

Gain hands-on experience with SPARQL, the RDF query language that's bringing new possibilities to semantic web, linked data, and big data projects. This updated and expanded edition shows you how to use SPARQL 1.1 with a variety of tools to retrieve, manipulate, and federate data from the public web as well as from private...

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .

ab:richard ab:homeTel "(229) 276-5135" .
ab:richard ab:email   "richard49@hotmail.com" .

ab:cindy ab:homeTel "(245) 646-5488" .
ab:cindy ab:email   "cindym@gmail.com" .

ab:craig ab:homeTel "(194) 966-1505" .
ab:craig ab:email   "craigellis@yahoo.com" .
ab:craig ab:email   "c.ellis@usairwaysgroup.com" .
```



subject (resource identifier)	predicate (property name)	object (property value)
richard	homeTel	(229) 276-5135
cindy	email	cindym@gmail.com

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?craigEmail
WHERE
{ ab:craig ab:email ?craigEmail . }
```

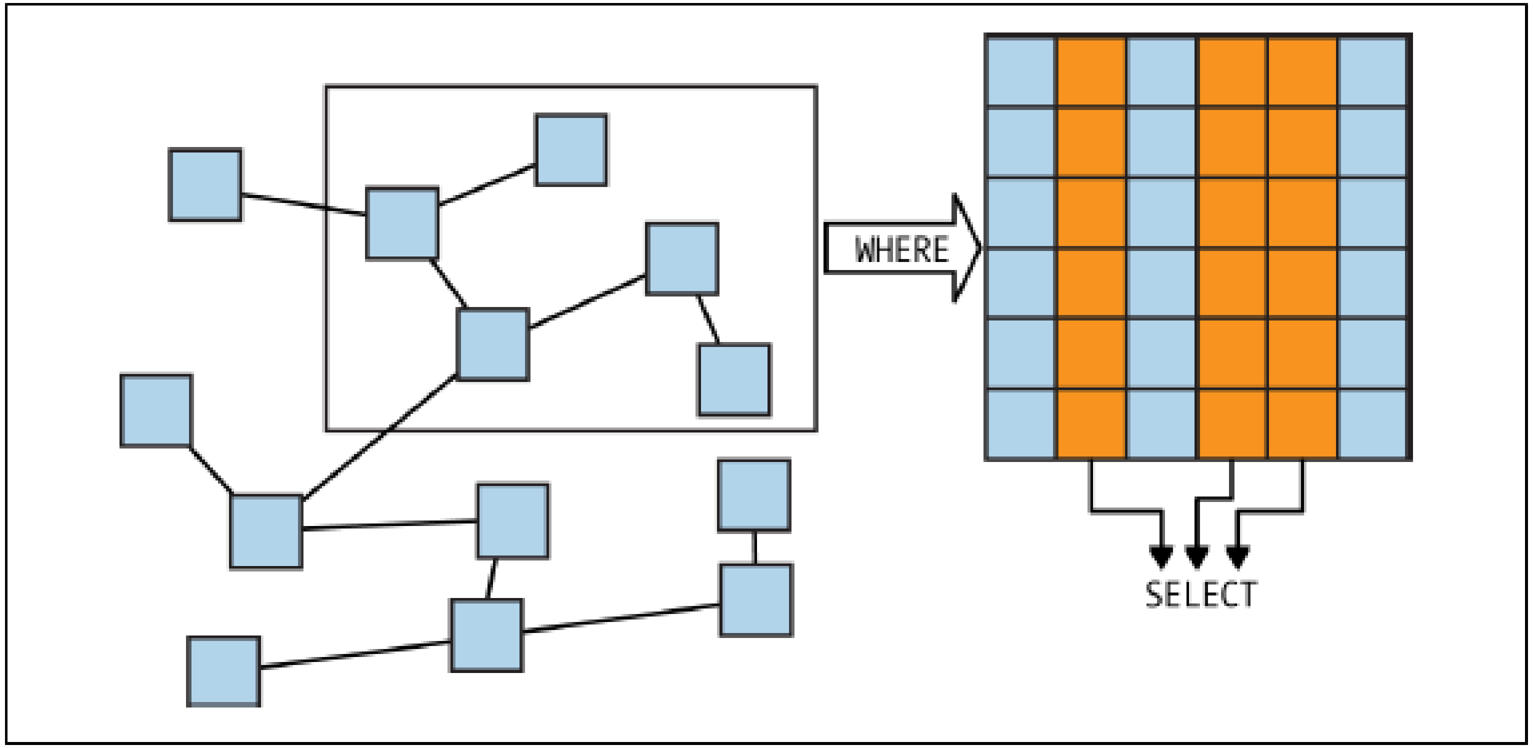
a subject of ab:craig, a predicate of ab:email, and a variable in the object position.

A variable is like a powerful wildcard.

- Tells the query engine that triples with any value at all in that position are OK to match this triple pattern
- the values that show up there get stored in the ?craigEmail variable so that we can use them elsewhere in the query

Basic Queries

- SPARQL : finds pieces of information from the subset of the data that meets these conditions



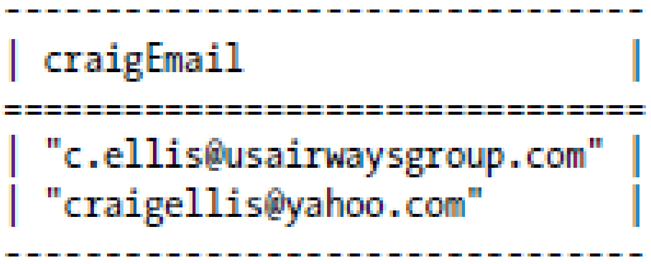
Data File

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .

d:i0432 ab:firstName "Richard" .
d:i0432 ab:lastName  "Mutt" .
d:i0432 ab:homeTel   "(229) 276-5135" .
d:i0432 ab:email     "richard49@hotmail.com" .

d:i9771 ab:firstName "Cindy" .
d:i9771 ab:lastName  "Marshall" .
d:i9771 ab:homeTel   "(245) 646-5488" .
d:i9771 ab:email     "cindym@gmail.com" .

d:i8301 ab:firstName "Craig" .
d:i8301 ab:lastName  "Ellis" .
d:i8301 ab:email     "craigellis@yahoo.com" .
d:i8301 ab:email     "c.ellis@usairwaysgroup.com" .
```

Basic Queries

Query: find Craig’s email addresses

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?craigEmail
WHERE
{
  ?person ab:firstName "Craig" .
  ?person ab:email ?craigEmail .
}
```



```
--------------------------------
| craigEmail                   |
================================
| "c.ellis@usairwaysgroup.com" |
| "craigellis@yahoo.com"       |
--------------------------------
```

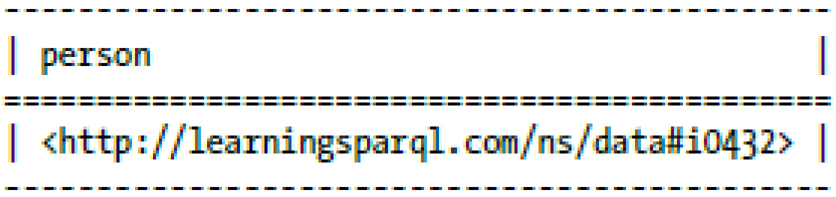
- Although the query uses a ?person variable, this variable isn’t in the list of variables to SELECT (a list of just one variable, ?craigEmail, in this query) because we’re not interested in the ?person variable’s value.
- We’re just using it to tie together the two triple patterns in the WHERE clause.
- If the SPARQL processor finds a triple with a predicate of ab:firstName and an object of “Craig”, it will assign (or bind) the URI in the subject of that triple to the variable ?person.
- Then, wherever else ?person appears in the query, it will look for triples that have that URI there.

Basic Queries

PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?person
WHERE
{ ?person ab:homeTel "(229) 276-5135" . }

- The result would shows subject of the triple that had ab:homeTel as a predicate and “(229) 276-5135” as an object

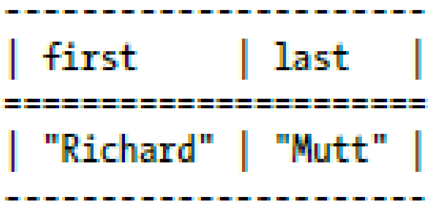


```
------------------------------------------------
| person                                       |
================================================
| <http://learningsparql.com/ns/data#i0432>    |
------------------------------------------------
```

Basic Queries

Find the first and last name of the person with a specific phone number

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last
WHERE
{
  ?person ab:homeTel "(229) 276-5135" .
  ?person ab:firstName ?first .
  ?person ab:lastName  ?last .
}
```



```
-----------------------
| first     | last    |
=======================
| "Richard" | "Mutt"  |
-----------------------
```

Basic Queries

Using semicolons to shorten the statements with same subject.

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last
WHERE
{
  ?person ab:homeTel "(229) 276-5135" ;
                ab:firstName ?first;
                ab:lastName  ?last .
}
```

Basic Queries

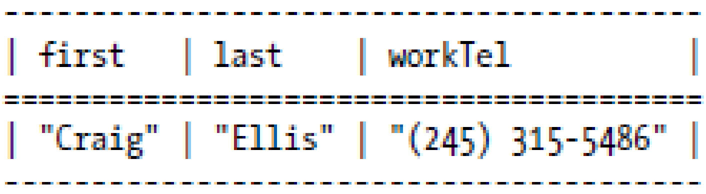
```
@prefix ab:
<http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .
d:i0432 ab:firstName "Richard" .
d:i0432 ab:lastName  "Mutt" .
d:i0432 ab:homeTel   "(229) 276-5135" .
d:i0432 ab:nick      "Dick" .
d:i0432 ab:email     "richard49@hotmail.com" .

d:i9771 ab:firstName "Cindy" .
d:i9771 ab:lastName  "Marshall" .
d:i9771 ab:homeTel   "(245) 646-5488" .
d:i9771 ab:email     "cindym@gmail.com" .

d:i8301 ab:firstName "Craig" .
d:i8301 ab:lastName  "Ellis" .
d:i8301 ab:workTel   "(245) 315-5486" .
d:i8301 ab:email     "craigellis@yahoo.com" .
d:i8301 ab:email     "c.ellis@usairwaysgroup.com" .
```

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?workTel
WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last ;
     ab:workTel ?workTel .
}
```

What is it returns ?

Basic Queries

- Just Craig, but for no one else



```
----------------------------------------
| first   | last    | workTel          |
========================================
| "Craig" | "Ellis" | "(245) 315-5486" |
----------------------------------------
```

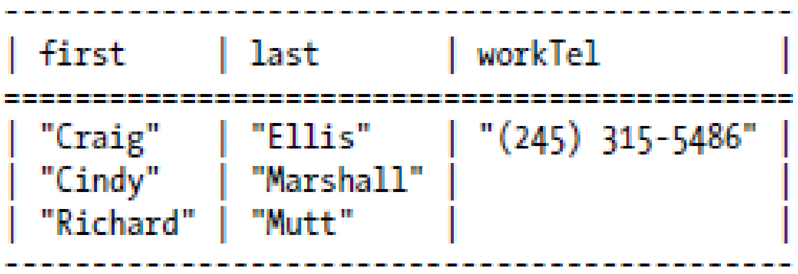
- Why?
- Because the triples in the pattern work together as a unit, or, as the SPARQL specification puts it, as a graph pattern.
- This **graph pattern** asks for someone who has an **ab:firstName value**, an **ab:lastName** value, and an **ab:workTel** value, and
- Craig is the only one who does.

Optional

A graph pattern is one or more triple patterns inside of curly braces

Putting the triple pattern about the work phone number in an OPTIONAL graph pattern lets your query say “show me this value, if it’s there”:

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?workTel
WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .
  OPTIONAL
  { ?s ab:workTel ?workTel . }
}
```



```
---------------------------------------------
| first     | last       | workTel          |
=============================================
| "Craig"   | "Ellis"    | "(245) 315-5486" |
| "Cindy"   | "Marshall" |                  |
| "Richard" | "Mutt"     |                  |
---------------------------------------------
```

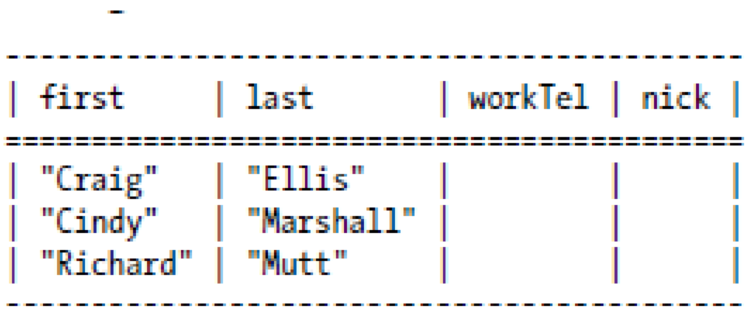
Optional

Richard has a nickname value stored with the ab:nick property, and no one else does.
What happens if we ask for that and put the triple pattern inside the OPTIONAL graph pattern that we just added?

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?workTel ?nick
WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .
  OPTIONAL
  {
     ?s ab:workTel ?workTel ;
        ab:nick ?nick .
  }
}
```

Optional

- We get everyone’s first and last names, but no one’s nickname or work phone number,
- even though we have Richard’s nickname and Craig’s work phone number

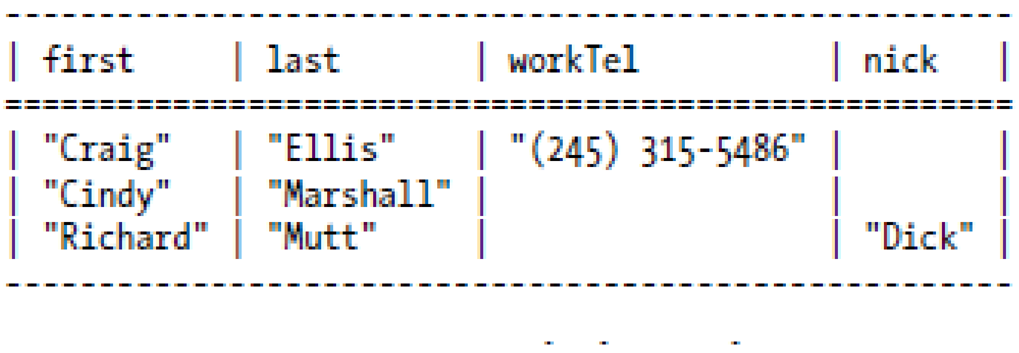


```
--------------------------------------------
| first     | last       | workTel | nick |
============================================
| "Craig"   | "Ellis"    |         |      |
| "Cindy"   | "Marshall" |         |      |
| "Richard" | "Mutt"     |         |      |
--------------------------------------------
```

- Why? Because the OPTIONAL graph pattern is just that: a pattern,
- no subjects in our data fit that pattern
  - that is, no subjects have both a nickname and a work phone number.

Optional

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?workTel ?nick
WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .
  OPTIONAL { ?s ab:workTel ?workTel . }
  OPTIONAL { ?s ab:nick ?nick .  }

}
```



```
------------------------------------------------------------
| first     | last       | workTel          | nick   |
============================================================
| "Craig"   | "Ellis"    | "(245) 315-5486" |        |
| "Cindy"   | "Marshall" |                  |        |
| "Richard" | "Mutt"     |                  | "Dick" |
------------------------------------------------------------
```

Finding Data That Doesn’t Meet Certain Conditions

- Query: List everyone whose work number is missing

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last
WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .

  OPTIONAL { ?s ab:workTel ?workNum . }
  FILTER (!bound(?workNum))
}
```

- query asks for each person’s first and last names and work phone number if they have it,
- but it has a filter to pass along a subset of the retrieved triples

Finding Data That Doesn’t Meet Certain Conditions

FILTER (!bound(?workNum))

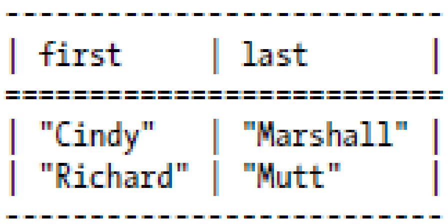
- FILTER to retrieve only labels with specific language tags assigned to them.

- the boolean bound() function to decide what the FILTER statement should pass along.

- This function returns true if the variable passed as a parameter is bound
  - if it’s been assigned a value)
- false otherwise.

- the exclamation point is a “not” operator

- so !bound(?workNum) will be true if the ?workNum variable is not bound
- dataset will pass along the first and last names of everyone who didn’t have an ab:workTel value to assign to the ?workNum variable



```
---------------------------
| first     | last        |
===========================
| "Cindy"   | "Marshall"  |
| "Richard" | "Mutt"      |
---------------------------
```

Finding Data That Doesn’t Meet Certain Conditions

- FILTER NOT EXISTS, is a FILTER condition that
  returns a boolean true if the specified graph pattern does not exist

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last

WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .
  FILTER NOT EXISTS { ?s ab:workTel ?workNum }
}
```

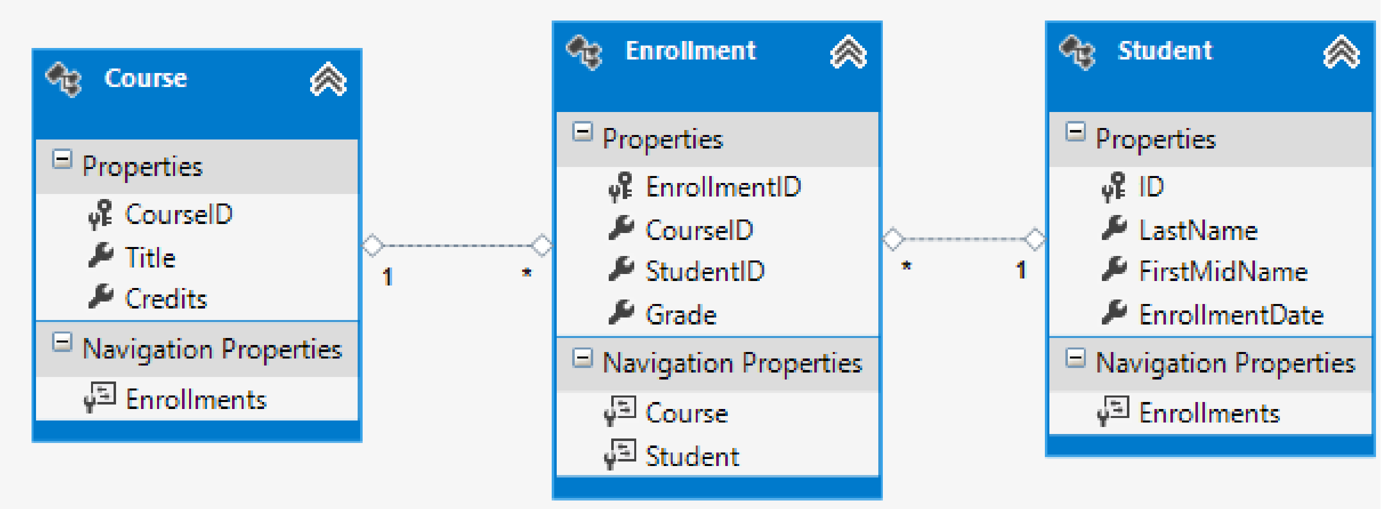
- MINUS removes patterns

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last

WHERE
{
  ?s ab:firstName ?first ;
     ab:lastName ?last .
  MINUS { ?s ab:workTel ?workNum }
}
```

- finds all the subjects with an ab:firstName and an ab:lastName value
- but uses the MINUS keyword to subtract those that have an ab:workTel value

Querying Combination of Multiple Data Sets

- Which students are taking which courses ?



- Relational Data base- each row of each table have a unique identifier within that table - so that you could cross-reference between the tables
- SQL : Join

@prefix ab: <http://learningsparql.com/ns/addressbook#> . @prefix d: <http://learningsparql.com/ns/data#> . # Courses d:course34 ab:courseTitle "Modeling Data with OWL" . d:course71 ab:courseTitle "Enhancing We

Querying Combination of Multiple Data Sets

RDF data: People

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .
d:i0432 ab:firstName "Richard" ;
        ab:lastName  "Mutt" ;
        ab:email     "richard49@hotmail.com" .

d:i9771 ab:firstName "Cindy" ;
        ab:lastName  "Marshall" ;
        ab:email     "cindym@gmail.com" .

d:i8301 ab:firstName "Craig" ;
        ab:lastName  "Ellis" ;
        ab:email     "c.ellis@usairwaysgroup.com" .
```

- the subject of each triple about a person (for example, d:i0432) is the unique identifier for that person

Querying Combination of Multiple Data Sets

RDF data: Courses

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .

d:course34 ab:courseTitle "Modeling Data with OWL" .
d:course71 ab:courseTitle "Enhancing Websites with RDFa" .
d:course59 ab:courseTitle "Using SPARQL with non-RDF Data" .
d:course85 ab:courseTitle "Updating Data with SPARQL" .
```

RDF data: Who's taking which courses

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .

d:i8301 ab:takingCourse d:course59 .
d:i9771 ab:takingCourse d:course34 .
d:i0432 ab:takingCourse d:course85 .
d:i0432 ab:takingCourse d:course59 .
d:i9771 ab:takingCourse d:course59 .
```

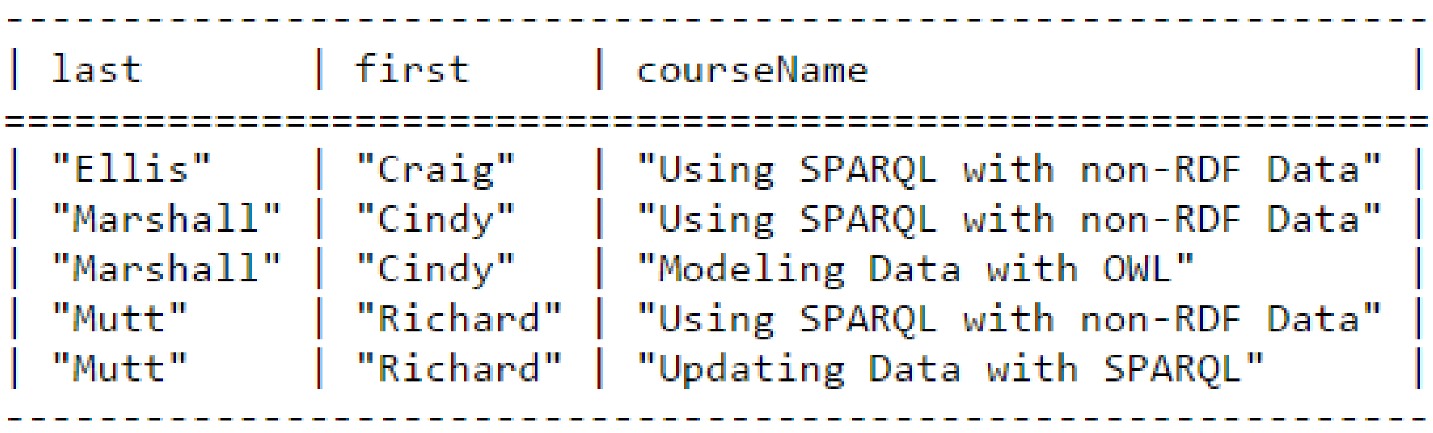
Querying Combination of Multiple Data Sets

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
SELECT ?last ?first ?courseName
WHERE
{
    ?s ab:firstName ?first ;
       ab:lastName ?last ;
       ab:takingCourse ?course .

    ?course ab:courseTitle ?courseName .
}
```

- use the same variable in the object position of one triple pattern and the subject position of another
- when the query processor looks for the course that a student is taking, it assigns the course's identifying URI to the ?course variable,
- then it looks for an ab:courseTitle value for that course and assigns it to the ?courseName variable.

Querying Combination of Multiple Data Sets

- Result set:



```
--------------------------------------------------------------
| last       | first     | courseName                       |
==============================================================
| "Ellis"    | "Craig"   | "Using SPARQL with non-RDF Data" |
| "Marshall" | "Cindy"   | "Using SPARQL with non-RDF Data" |
| "Marshall" | "Cindy"   | "Modeling Data with OWL"         |
| "Mutt"     | "Richard" | "Using SPARQL with non-RDF Data" |
| "Mutt"     | "Richard" | "Updating Data with SPARQL"      |
--------------------------------------------------------------
```

Eliminating Redundant Output

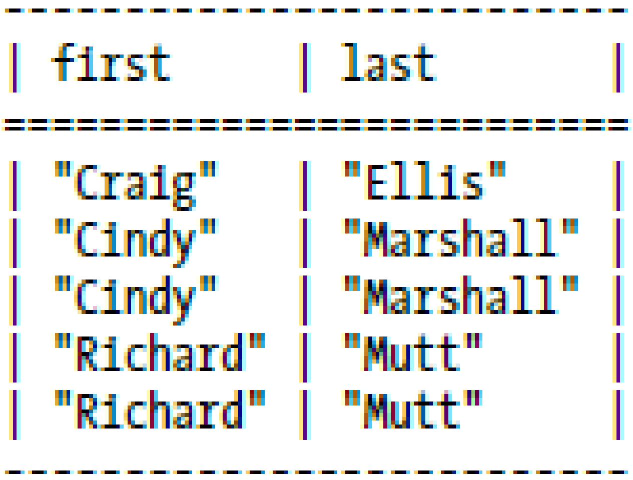
```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:  <http://learningsparql.com/ns/data#> .
# People
d:i0432 ab:firstName "Richard" ;
        ab:lastName  "Mutt" ;
        ab:email     "richard49@hotmail.com" .

d:i9771 ab:firstName "Cindy" ;
        ab:lastName  "Marshall" ;
        ab:email     "cindym@gmail.com" .

d:i8301 ab:firstName "Craig" ;
        ab:lastName  "Ellis" ;
        ab:email     "c.ellis@usairwaysgroup.com" .
# Courses
d:course34 ab:courseTitle "Modeling Data with OWL" .
d:course71 ab:courseTitle "Enhancing Websites with RDFa" .
d:course59 ab:courseTitle "Using SPARQL with non-RDF Data" .
d:course85 ab:courseTitle "Updating Data with SPARQL" .

# Who's taking which courses
d:i8301 ab:takingCourse d:course59 .
d:i9771 ab:takingCourse d:course34 .
d:i0432 ab:takingCourse d:course85 .
d:i0432 ab:takingCourse d:course59 .
d:i9771 ab:takingCourse d:course59 .
```

I want to search the name of students who are taking any course, but not the duplicate results

the DISTINCT keyword - Like it does in SQL

Eliminating Redundant Output

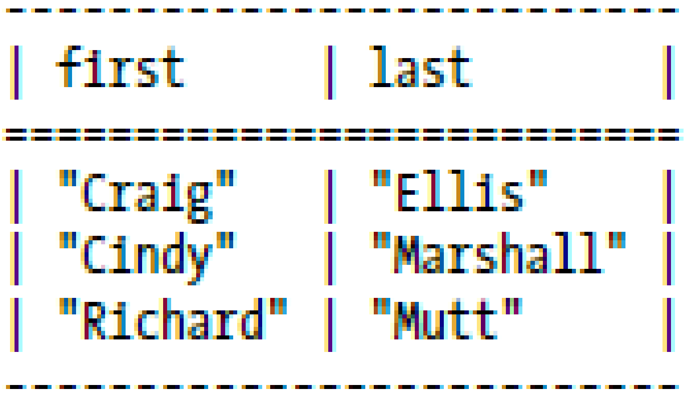
```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT  ?first ?last
WHERE
{
  ?s ab:takingCourse ?class ;
     ab:firstName ?first ;
     ab:lastName ?last .
}
```



```
---------------------------
| first     | last        |
===========================
| "Craig"   | "Ellis"     |
| "Cindy"   | "Marshall"  |
| "Cindy"   | "Marshall"  |
| "Richard" | "Mutt"      |
| "Richard" | "Mutt"      |
---------------------------
```

Eliminating Redundant Output

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT  DISTINCT ?first ?last
WHERE
{
  ?s ab:takingCourse ?class ;
     ab:firstName ?first ;
     ab:lastName ?last .
}
```



```
--------------------------
| first     | last       |
==========================
| "Craig"   | "Ellis"    |
| "Cindy"   | "Marshall" |
| "Richard" | "Mutt"     |
--------------------------
```

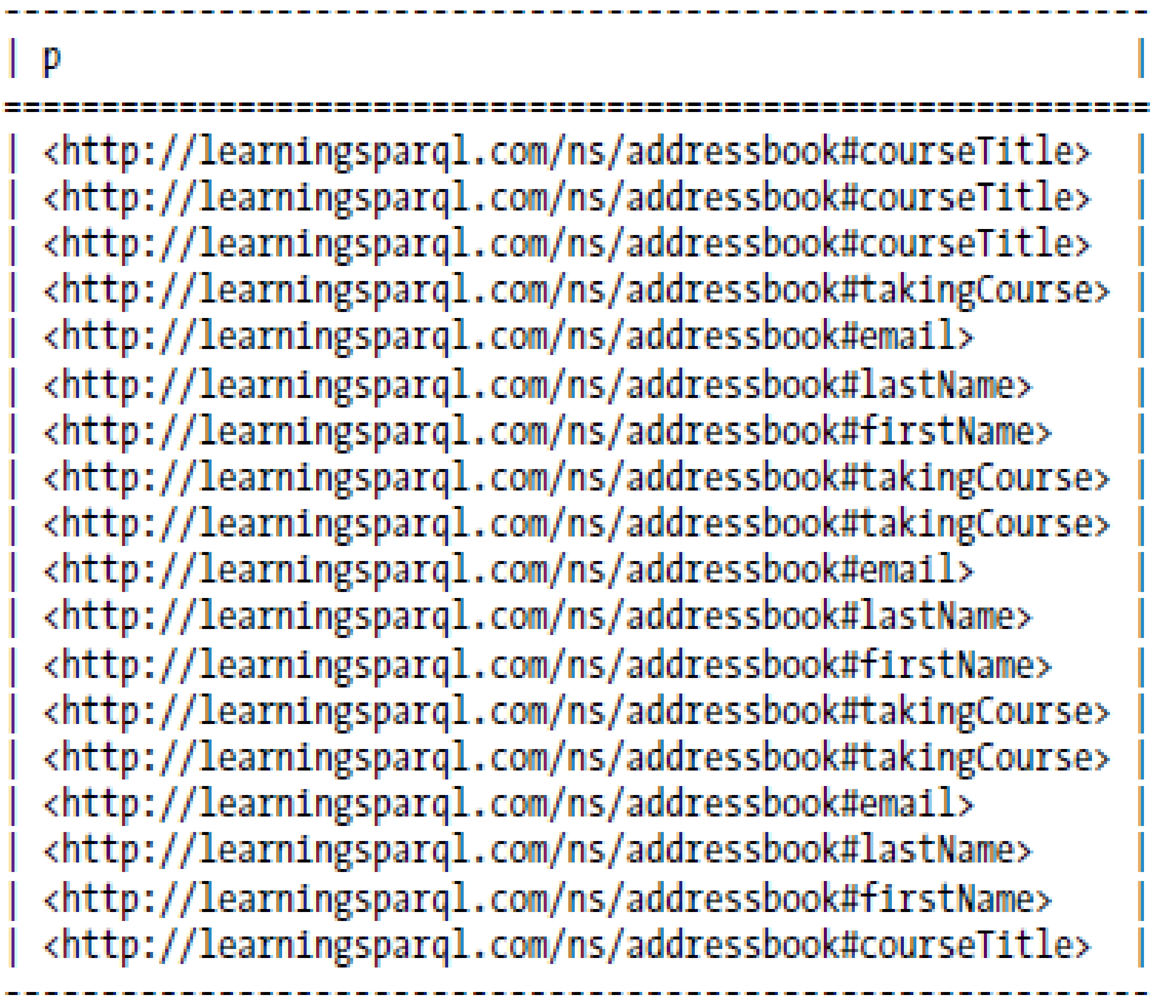
- With the DISTINCT keyword, the query lists the name of each person taking a course with no repeats:

Eliminating Redundant Output

- Retrieve all the triples in your data set

```
SELECT ?p
  WHERE
    { ?s ?p ?o . }
    LIMIT 10
```

There are repeats . Why ?

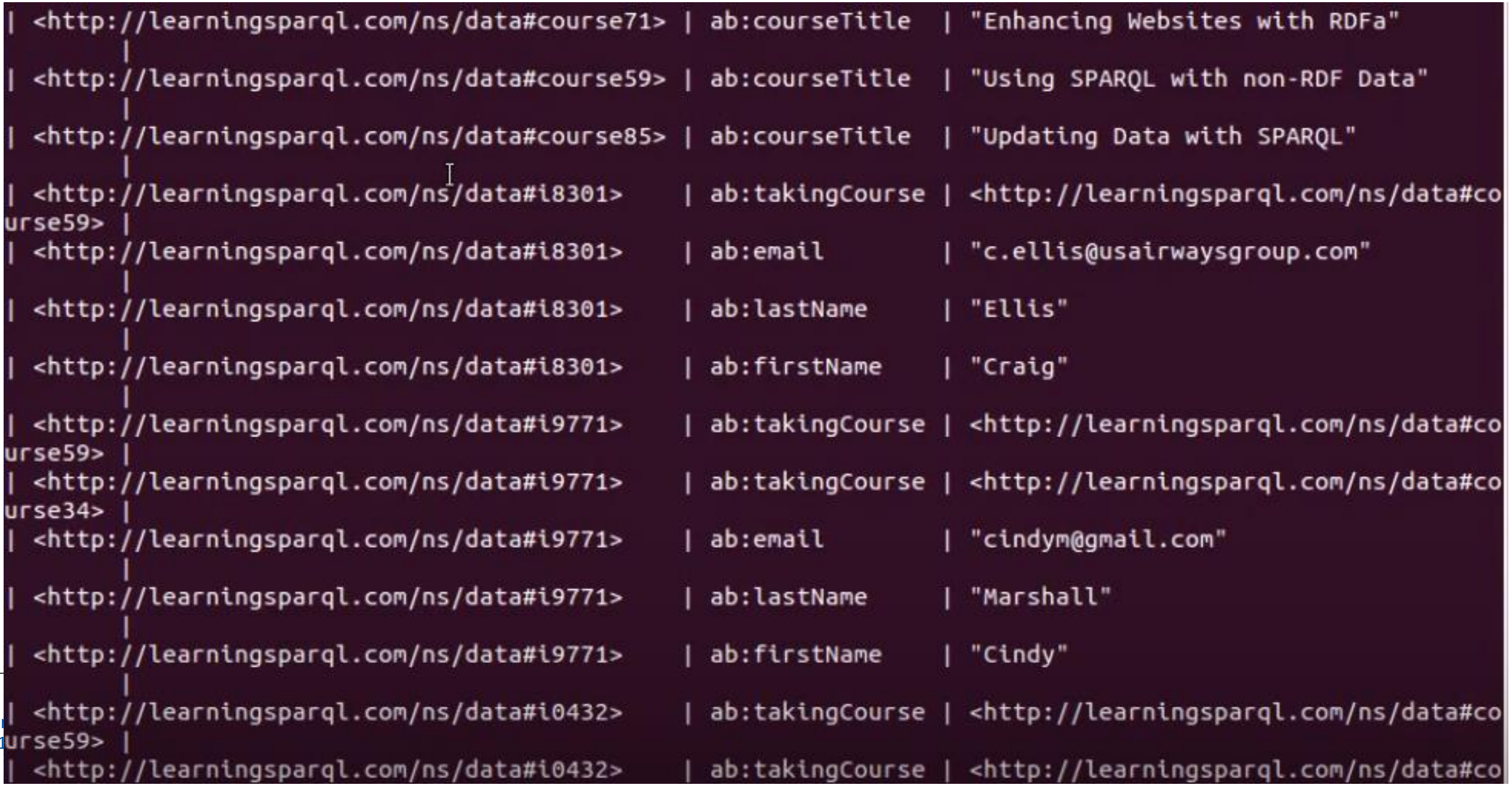


```
-------------------------------------------------------------
| p                                                         |
=============================================================
| <http://learningsparql.com/ns/addressbook#courseTitle>    |
| <http://learningsparql.com/ns/addressbook#courseTitle>    |
| <http://learningsparql.com/ns/addressbook#courseTitle>    |
| <http://learningsparql.com/ns/addressbook#takingCourse>   |
| <http://learningsparql.com/ns/addressbook#email>          |
| <http://learningsparql.com/ns/addressbook#lastName>       |
| <http://learningsparql.com/ns/addressbook#firstName>      |
| <http://learningsparql.com/ns/addressbook#takingCourse>   |
| <http://learningsparql.com/ns/addressbook#takingCourse>   |
| <http://learningsparql.com/ns/addressbook#email>          |
| <http://learningsparql.com/ns/addressbook#lastName>       |
| <http://learningsparql.com/ns/addressbook#firstName>      |
| <http://learningsparql.com/ns/addressbook#takingCourse>   |
| <http://learningsparql.com/ns/addressbook#takingCourse>   |
| <http://learningsparql.com/ns/addressbook#email>          |
| <http://learningsparql.com/ns/addressbook#lastName>       |
| <http://learningsparql.com/ns/addressbook#firstName>      |
| <http://learningsparql.com/ns/addressbook#courseTitle>    |
-------------------------------------------------------------
```

Eliminating Redundant Output

- Retrieve all the triples in a dataset

SELECT *
WHERE
{ ?s ?p ?o . }

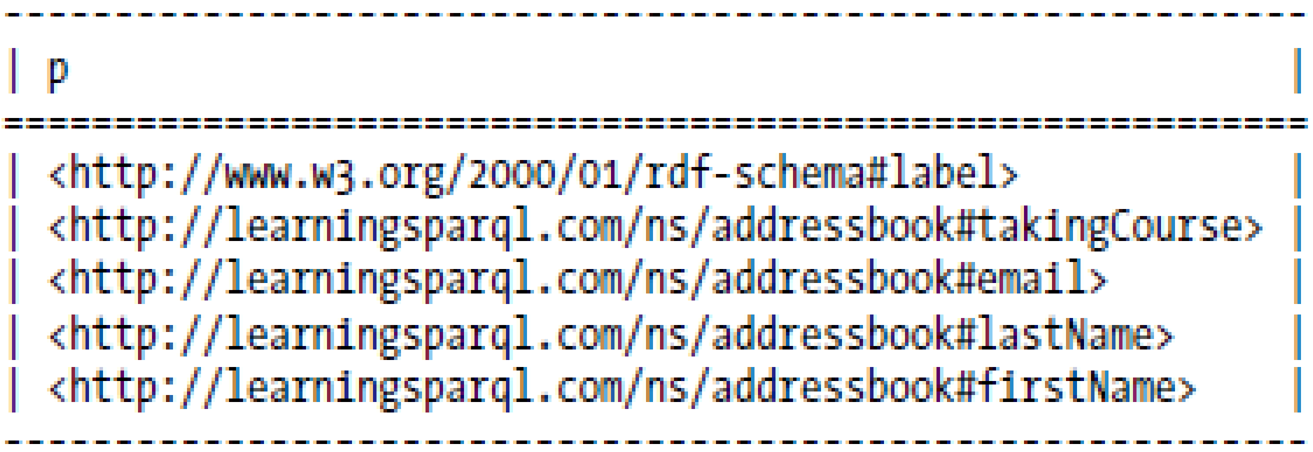


```
| <http://learningsparql.com/ns/data#course71> | ab:courseTitle   | "Enhancing Websites with RDFa"
        |
| <http://learningsparql.com/ns/data#course59> | ab:courseTitle   | "Using SPARQL with non-RDF Data"
        |
| <http://learningsparql.com/ns/data#course85> | ab:courseTitle   | "Updating Data with SPARQL"
        |
| <http://learningsparql.com/ns/data#i8301>    | ab:takingCourse | <http://learningsparql.com/ns/data#co
urse59> |
| <http://learningsparql.com/ns/data#i8301>    | ab:email        | "c.ellis@usairwaysgroup.com"
        |
| <http://learningsparql.com/ns/data#i8301>    | ab:lastName     | "Ellis"
        |
| <http://learningsparql.com/ns/data#i8301>    | ab:firstName    | "Craig"
        |
| <http://learningsparql.com/ns/data#i9771>    | ab:takingCourse | <http://learningsparql.com/ns/data#co
urse59> |
| <http://learningsparql.com/ns/data#i9771>    | ab:takingCourse | <http://learningsparql.com/ns/data#co
urse34> |
| <http://learningsparql.com/ns/data#i9771>    | ab:email        | "cindym@gmail.com"
        |
| <http://learningsparql.com/ns/data#i9771>    | ab:lastName     | "Marshall"
        |
| <http://learningsparql.com/ns/data#i9771>    | ab:firstName    | "Cindy"
        |
| <http://learningsparql.com/ns/data#i0432>    | ab:takingCourse | <http://learningsparql.com/ns/data#co
urse59> |
| <http://learningsparql.com/ns/data#i0432>    | ab:takingCourse | <http://learningsparql.com/ns/data#co
```

Eliminating Redundant Output

- Retrieve all the distinct predicates in your data set

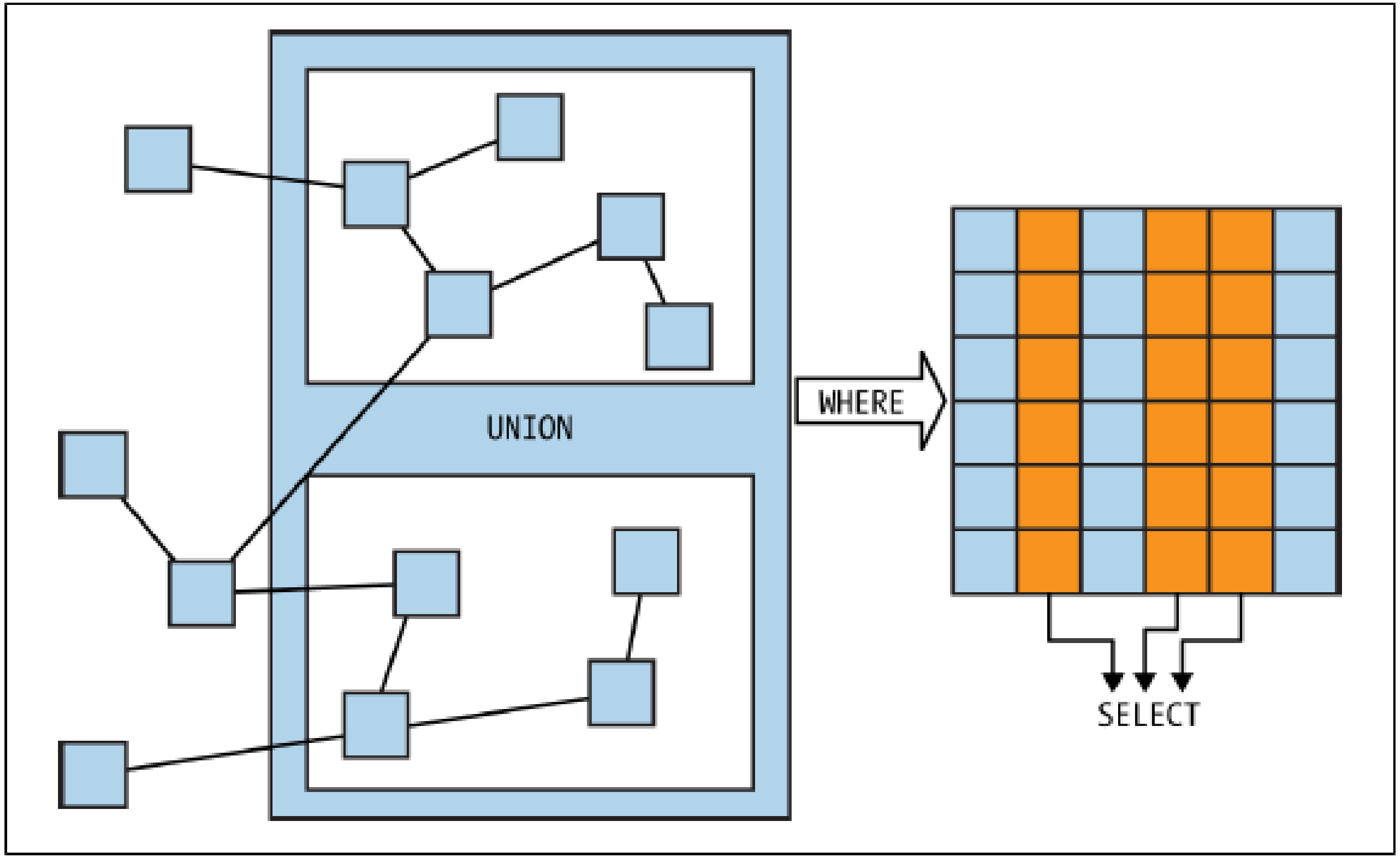
SELECT **DISTINCT** ?p
   WHERE
    { ?s ?p ?o . }



```
-------------------------------------------------------
| p                                                   |
=======================================================
| <http://www.w3.org/2000/01/rdf-schema#label>        |
| <http://learningsparql.com/ns/addressbook#takingCourse> |
| <http://learningsparql.com/ns/addressbook#email>    |
| <http://learningsparql.com/ns/addressbook#lastName> |
| <http://learningsparql.com/ns/addressbook#firstName> |
-------------------------------------------------------
```

Combining Different Search Conditions

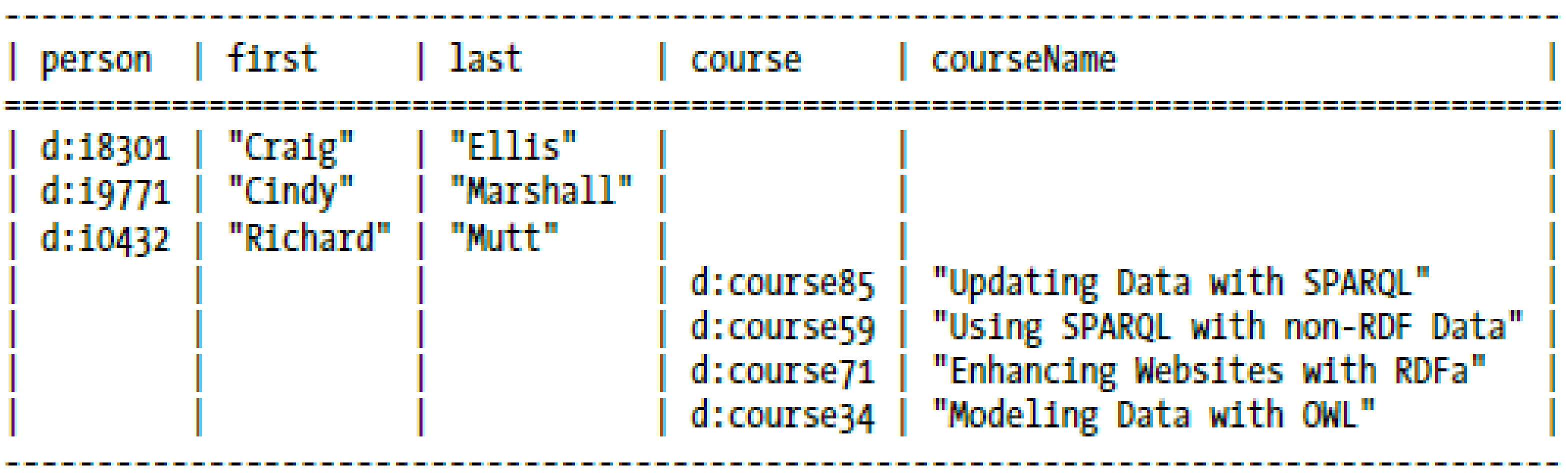
- SPARQL’s UNION keyword lets you specify multiple different graph patterns and then ask for a combination of all the data that fits any of those patterns.



Combining Different Search Conditions

list the people and the courses
– all courses and all people (not the one who takes courses)

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
PREFIX d: <http://learningsparql.com/ns/data#>
SELECT *
WHERE
    {
        { ?person ab:firstName ?first ; ab:lastName ?last . } – Graph pattern 1
    UNION
        { ?course ab:courseTitle ?courseName . } – Graph pattern 2
}
```

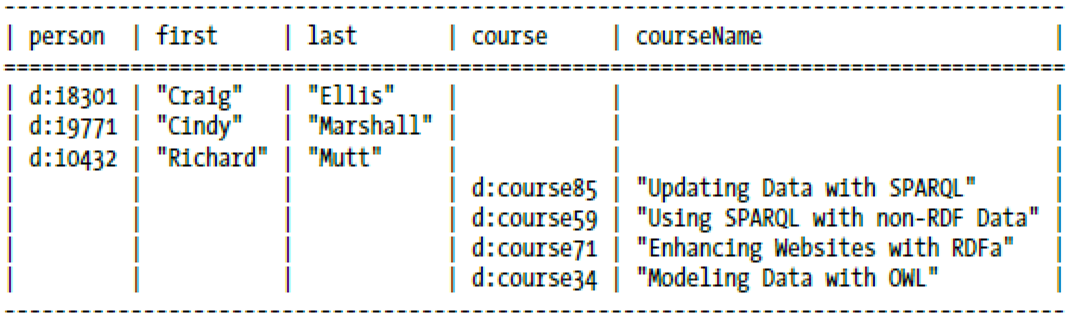


```
--------------------------------------------------------------------------------
| person  | first     | last       | course     | courseName                   |
================================================================================
| d:i8301 | "Craig"   | "Ellis"    |            |                              |
| d:i9771 | "Cindy"   | "Marshall" |            |                              |
| d:i0432 | "Richard" | "Mutt"     |            |                              |
|         |           |            | d:course85 | "Updating Data with SPARQL"  |
|         |           |            | d:course59 | "Using SPARQL with non-RDF Data" |
|         |           |            | d:course71 | "Enhancing Websites with RDFa" |
|         |           |            | d:course34 | "Modeling Data with OWL"     |
--------------------------------------------------------------------------------
```

Combining Different Search Conditions

list the people and the courses
– all courses and all people (not the one who takes courses)

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>
PREFIX d: <http://learningsparql.com/ns/data#>
SELECT *
WHERE
    {
        { ?person ab:firstName
    UNION
        { ?course ab:courseTitl
}
```



```
-----------------------------------------------------------------------------------
| person   | first     | last       | course      | courseName                     |
===================================================================================
| d:i8301  | "Craig"   | "Ellis"    |             |                                |
| d:i9771  | "Cindy"   | "Marshall" |             |                                |
| d:i0432  | "Richard" | "Mutt"     |             |                                |
|          |           |            | d:course85  | "Updating Data with SPARQL"    |
|          |           |            | d:course59  | "Using SPARQL with non-RDF Data" |
|          |           |            | d:course71  | "Enhancing Websites with RDFa" |
|          |           |            | d:course34  | "Modeling Data with OWL"       |
-----------------------------------------------------------------------------------
```

Combining Different Search Conditions

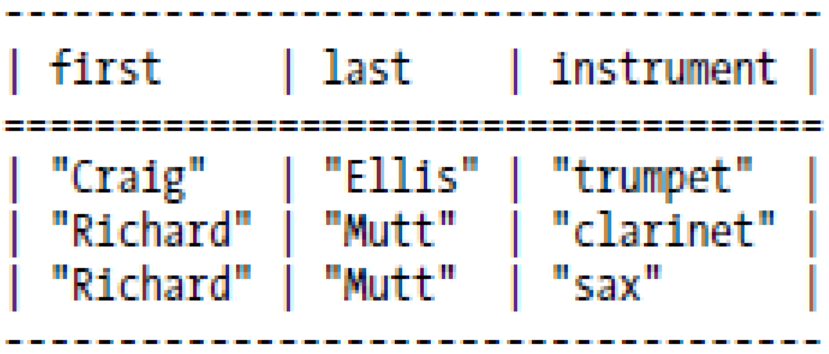
- Data: sample address book data stored information about musical instruments that each person plays

```
@prefix ab: <http://learningsparql.com/ns/addressbook#> .
@prefix d:   <http://learningsparql.com/ns/data#> .

d:i0432 ab:firstName  "Richard" ;
        ab:lastName   "Mutt" ;
        ab:instrument "sax" ;
        ab:instrument "clarinet" .

d:i9771 ab:firstName  "Cindy" ;
        ab:lastName   "Marshall" ;
        ab:instrument "drums" .

d:i8301 ab:firstName  "Craig" ;
        ab:lastName   "Ellis" ;
        ab:instrument "trumpet" .
```

Combining Different Search Conditions

- Task: to retrieve the first names, last names, and instrument names of the Trumpet and Sax players

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?instrument
WHERE
{
  { ?person ab:firstName ?first ;
            ab:lastName ?last ;
            ab:instrument "trumpet" ;
            ab:instrument ?instrument .
  }

  UNION

  { ?person ab:firstName ?first ;
            ab:lastName ?last ;
            ab:instrument "sax" ;
            ab:instrument ?instrument .
  }

}
```



```
------------------------------------
| first     | last    | instrument |
====================================
| "Craig"   | "Ellis" | "trumpet"  |
| "Richard" | "Mutt"  | "clarinet" |
| "Richard" | "Mutt"  | "sax"      |
------------------------------------
```

- Why do we have clarinet in the result set ?

Combining Different Search Conditions

```
PREFIX ab: <http://learningsparql.com/ns/addressbook#>

SELECT ?first ?last ?instrument
WHERE
{
   ?person ab:firstName ?first ;
          ab:lastName ?last ;
          ab:instrument ?instrument .

   { ?person ab:instrument "sax" . }

   UNION

   { ?person ab:instrument "trumpet" . }

}
```

AND

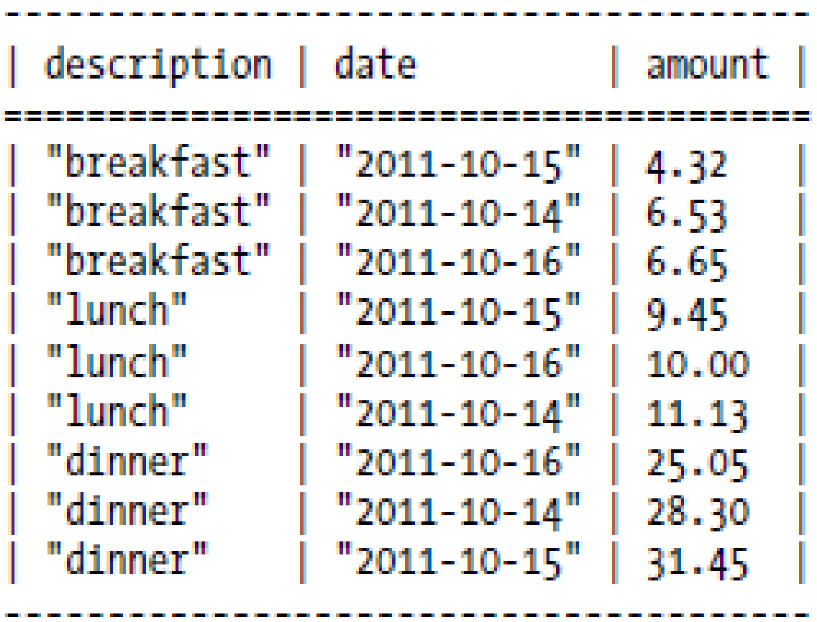
OR

Problem fixed this by using the UNION keyword to unite smaller graph patterns and add them to the part that the last query’s two graph patterns had in common.

```
@prefix e: <http://learningsparql.com/ns/expenses#> .
@prefix d: <http://learningsparql.com/ns/data#> .

d:m40392 e:description "breakfast" ;
        e:date "2011-10-14" ;
        e:amount 6.53 .

d:m40393 e:description "lunch" ;
        e:date "2011-10-14" ;
        e:amount 11.13 .

d:m40394 e:description "dinner" ;
        e:date "2011-10-14" ;
        e:amount 28.30 .

d:m40395 e:description "breakfast" ;
        e:date "2011-10-15" ;
        e:amount 4.32 .

d:m40396 e:description "lunch" ;
        e:date "2011-10-15" ;
        e:amount 9.45 .

d:m40397 e:description "dinner" ;
        e:date "2011-10-15" ;
        e:amount 31.45 .

d:m40398 e:description "breakfast" ;
        e:date "2011-10-16" ;
        e:amount 6.65 .

d:m40399 e:description "lunch" ;
        e:date "2011-10-16" ;
        e:amount 10.00 .

d:m40400 e:description "dinner" ;
        e:date "2011-10-16" ;
        e:amount 25.05 .
```

Sorting Data

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT ?description ?date ?amount
WHERE
{
  ?meal e:description ?description ;
        e:date ?date ;
        e:amount ?amount .
}

ORDER BY ?amount
```



```
---------------------------------------
| description | date         | amount |
=======================================
| "breakfast" | "2011-10-15" | 4.32   |
| "breakfast" | "2011-10-14" | 6.53   |
| "breakfast" | "2011-10-16" | 6.65   |
| "lunch"     | "2011-10-15" | 9.45   |
| "lunch"     | "2011-10-16" | 10.00  |
| "lunch"     | "2011-10-14" | 11.13  |
| "dinner"    | "2011-10-16" | 25.05  |
| "dinner"    | "2011-10-14" | 28.30  |
| "dinner"    | "2011-10-15" | 31.45  |
---------------------------------------
```

Finding the Smallest, the Biggest, the Count, the Average

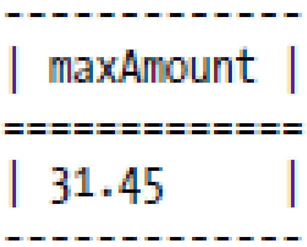
What is the most expensive meal

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT ?description ?date ?amount
WHERE
{
  ?meal e:description ?description ;
        e:date ?date ;
        e:amount ?amount .
}

ORDER BY DESC(?amount)
LIMIT 1
```

Finding the Smallest, the Biggest, the Count, the Average

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT (MAX(?amount) as ?maxAmount)  -- assigns max value of amount to variable maxAmount
WHERE { ?meal e:amount ?amount . }
```

This query’s SELECT clause stores the maximum value bound to the ?amount variable in the ?maxAmount variable



```
-------------
| maxAmount |
=============
| 31.45     |
-------------
```

Finding the Smallest, the Biggest, the Count, the Average

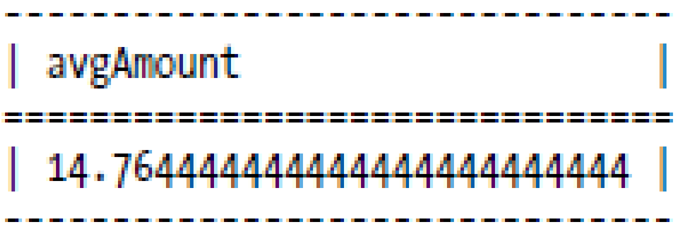
- find the description and date values associated with the maximum amount identified with the MAX() function

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT ?description ?date ?maxAmount
WHERE
{
  {
    SELECT (MAX(?amount) as ?maxAmount)
    WHERE { ?meal e:amount ?amount . }
  }
  {
    ?meal e:description ?description ;
          e:date ?date ;
          e:amount ?maxAmount .
  }
}
```

Finding the Smallest, the Biggest, the Count, the Average

Find the average cost of all the meals

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT (AVG(?amount) as ?avgAmount)
WHERE { ?meal e:amount ?amount . }
```



```
-------------------------------
| avgAmount                   |
===============================
| 14.764444444444444444444444 |
-------------------------------
```

Finding the Smallest, the Biggest, the Count, the Average

Sum the total cost of all the meals

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT (SUM (?amount) as ?avgAmount)
WHERE { ?meal e:amount ?amount . }
```

Finding the Smallest, the Biggest, the Count, the Average

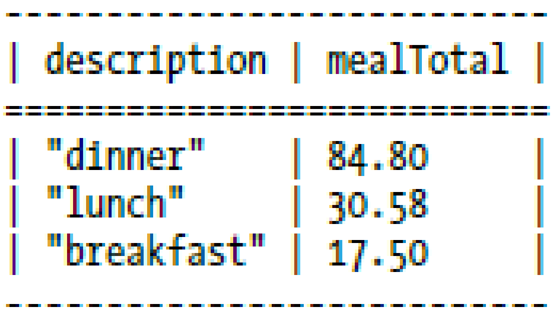
Find how many values got bound to amount variable—in other words, how many e:amount values were retrieved.

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT (COUNT(?amount) as ?avgAmount)
WHERE { ?meal e:amount ?amount . }
```

Grouping Data and Finding Aggregate Values within Groups

- group sets of data together to perform aggregate functions such as subtotal calculation on each group.

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT ?description (SUM(?amount) AS ?mealTotal)
WHERE
{
  ?meal e:description ?description ;
        e:amount ?amount .
}
GROUP BY ?description
```

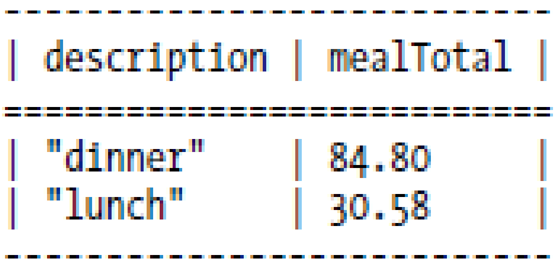


```
----------------------------
| description | mealTotal |
============================
| "dinner"    | 84.80     |
| "lunch"     | 30.58     |
| "breakfast" | 17.50     |
----------------------------
```

Grouping Data and Finding Aggregate Values within Groups

- The HAVING keyword specifies a condition that lets you restrict which values you want to appear in the results.
  - It does for aggregate values what FILTER does for individual values

```
PREFIX e: <http://learningsparql.com/ns/expenses#>

SELECT ?description (SUM(?amount) AS ?mealTotal)
WHERE
{
  ?meal e:description ?description ;
        e:amount ?amount .
}
GROUP BY ?description
HAVING (SUM(?amount) > 20)
```



```
----------------------------
| description | mealTotal |
============================
| "dinner"    | 84.80     |
| "lunch"     | 30.58     |
----------------------------
```

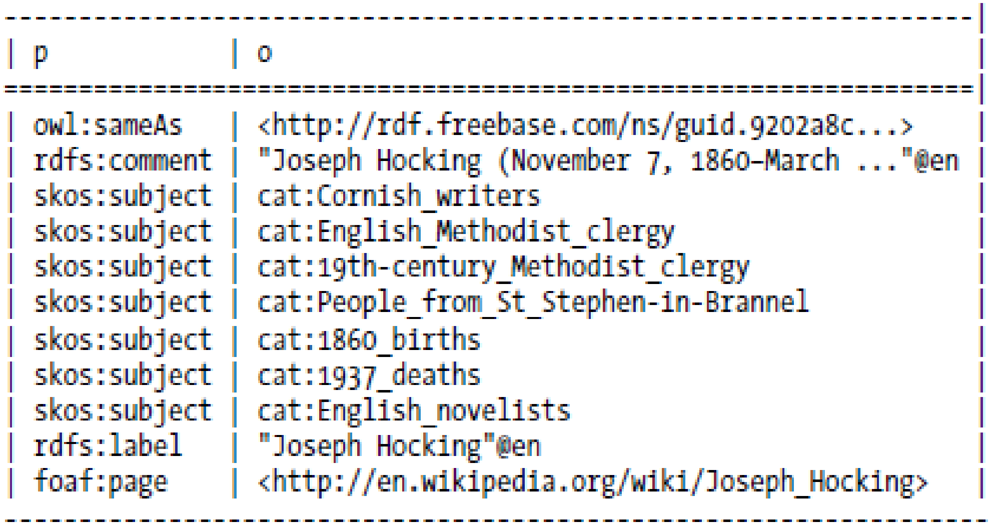
Querying a Remote SPARQL Service

- How to query remote data: The SERVICE keyword

- Points the query at a SPARQL endpoint

```
PREFIX cat:     <http://dbpedia.org/resource/Category:>
PREFIX skos:    <http://www.w3.org/2004/02/skos/core#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:     <http://www.w3.org/2002/07/owl#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>

SELECT ?p ?o
WHERE
{
  SERVICE <http://DBpedia.org/sparql>
  { SELECT ?p ?o
    WHERE { <http://dbpedia.org/resource/Joseph_Hocking> ?p ?o . }
  }
}
```

asks for the predicates and objects of all the DBpedia triples that have *http://dbpedia.org/resource/Joseph_Hocking* as their subject



```
----------------------------------------------------------------|
| p              | o                                             |
================================================================|
| owl:sameAs     | <http://rdf.freebase.com/ns/guid.9202a8c...>  |
| rdfs:comment   | "Joseph Hocking (November 7, 1860-March ..."@en |
| skos:subject   | cat:Cornish_writers                           |
| skos:subject   | cat:English_Methodist_clergy                  |
| skos:subject   | cat:19th-century_Methodist_clergy             |
| skos:subject   | cat:People_from_St_Stephen-in-Brannel         |
| skos:subject   | cat:1860_births                               |
| skos:subject   | cat:1937_deaths                               |
| skos:subject   | cat:English_novelists                         |
| rdfs:label     | "Joseph Hocking"@en                           |
| foaf:page      | <http://en.wikipedia.org/wiki/Joseph_Hocking> |
-----------------------------------------------------------------
```

Query Forms: SELECT, DESCRIBE, ASK, and CONSTRUCT

CONSTRUCT

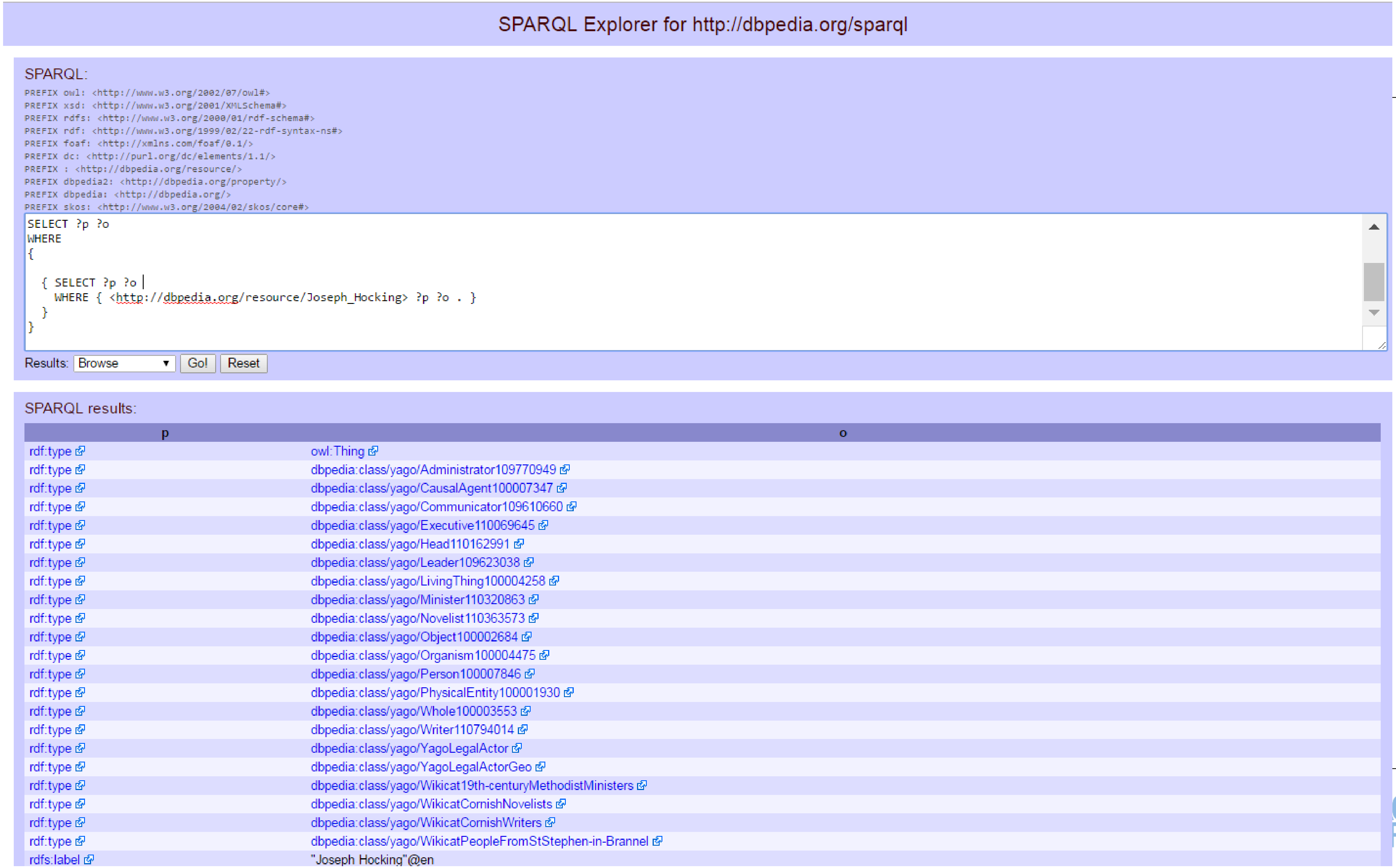
- returns triples.
- You can pull triples directly out of a data source without changing them,
- or you can pull values out and use those values to create new triples.
- This lets you copy, create, and convert RDF data

ASK

- asks a query processor whether a given graph pattern describes a set of triples in a particular dataset or not,
- the processor returns a boolean true or false.

DESCRIBE

- asks for triples that describe a particular resource.



SPARQL Explorer for http://dbpedia.org/sparql

SPARQL:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
```

```
SELECT ?p ?o
WHERE
{

  { SELECT ?p ?o
    WHERE { <http://dbpedia.org/resource/Joseph_Hocking> ?p ?o . }
  }
}
```

SPARQL results:

p	o
rdf:type	owl:Thing
rdf:type	dbpedia:class/yago/Administrator109770949
rdf:type	dbpedia:class/yago/CausalAgent100007347
rdf:type	dbpedia:class/yago/Communicator109610660
rdf:type	dbpedia:class/yago/Executive110069645
rdf:type	dbpedia:class/yago/Head110162991
rdf:type	dbpedia:class/yago/Leader109623038
rdf:type	dbpedia:class/yago/LivingThing100004258
rdf:type	dbpedia:class/yago/Minister110320863
rdf:type	dbpedia:class/yago/Novelist110363573
rdf:type	dbpedia:class/yago/Object100002684
rdf:type	dbpedia:class/yago/Organism100004475
rdf:type	dbpedia:class/yago/Person100007846
rdf:type	dbpedia:class/yago/PhysicalEntity100001930
rdf:type	dbpedia:class/yago/Whole100003553
rdf:type	dbpedia:class/yago/Writer110794014
rdf:type	dbpedia:class/yago/YagoLegalActor
rdf:type	dbpedia:class/yago/YagoLegalActorGeo
rdf:type	dbpedia:class/yago/Wikicat19th-centuryMethodistMinisters
rdf:type	dbpedia:class/yago/WikicatCornishNovelists
rdf:type	dbpedia:class/yago/WikicatCornishWriters
rdf:type	dbpedia:class/yago/WikicatPeopleFromStStephen-in-Brannel
rdfs:label	"Joseph Hocking"@en

Biomedical Data

Linked Data in the Life Sciences

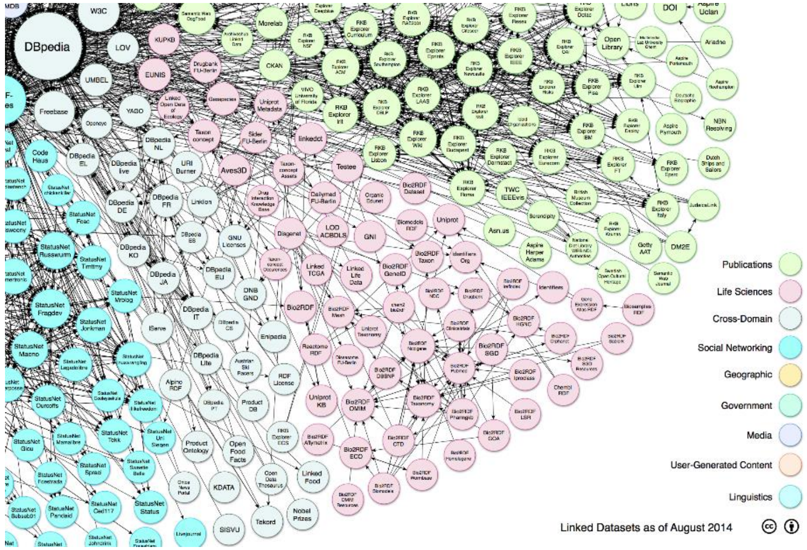
- Life Science Data in the Linked Open Data Cloud (LOD)

- LOD project aims to identify data resources, convert them to RDF and publish them as Linked Data

- 2009: Life Science data occupies the second biggest domain of the LOD
  - 36,1 % of the total LOD triples
  - 89,4 % of all LOD links in life science triples

- 2011:
  - 9,6 % of the total LOD triples
  - 13,9 % of the total LOD datasets

- 2014:
  - 7,8 % of the total LOD datasets



Linked Datasets as of August 2014

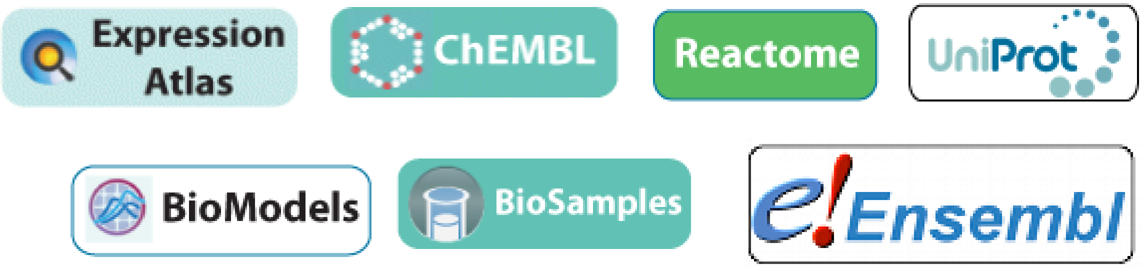
Linked Data in the Life Sciences

- EBI RDF Platform

- EBI grants access to their data via interfaces, web services, data download or direct database access

- EBI developed an RDF platform, where data is published following the Linked Data principles
  - Meet increasing demands to use Semantic Web technologies
  - Additional data access to EBI data resources

- 7 EBI resources published as SPARQL endpoints



- BioMedBridges tested scalability of Semantic Web on the EBI RDF platform

Linked Data in the Life Sciences

- OpenPHACTS (Open Pharmacological Concept Triple Store)

- Partnership between organisations like academia, enterprises, publishers, pharmaceutical companies
- Aims to reduce barriers to drug discovery in industry and academia
- Create a sustainable data infrastructure that provides an open pharmacological space
- Builds upon modern Semantic Web standards to support innovation in drug discovery

- Further develop and improve Linked Data ontologies and tooling software

Linked Data in the Life Sciences

- Bio2RDF

- Apply Semantic Web technologies to publicy available databases
  - Convert databases‘ content into RDF and link between them
  - Create freely available knowledge space
  - Support scientists in life science research

- Development of software tools for *rdfizing*

- Early contributor to distribution of Linked Data for the life sciences
  - Many lessons learned
  - Many new releases of Bio2RDF
  - Scientific papers that report the issues and solutions

SPARQL in BIO2RDF Project

- BIO2RDF is an open source framework that makes biological data available on the emerging semantic web using a set of simple conventions.
- 10 billion triples
- 29 datasets:

http://download.bio2rdf.org/release/3/release.html

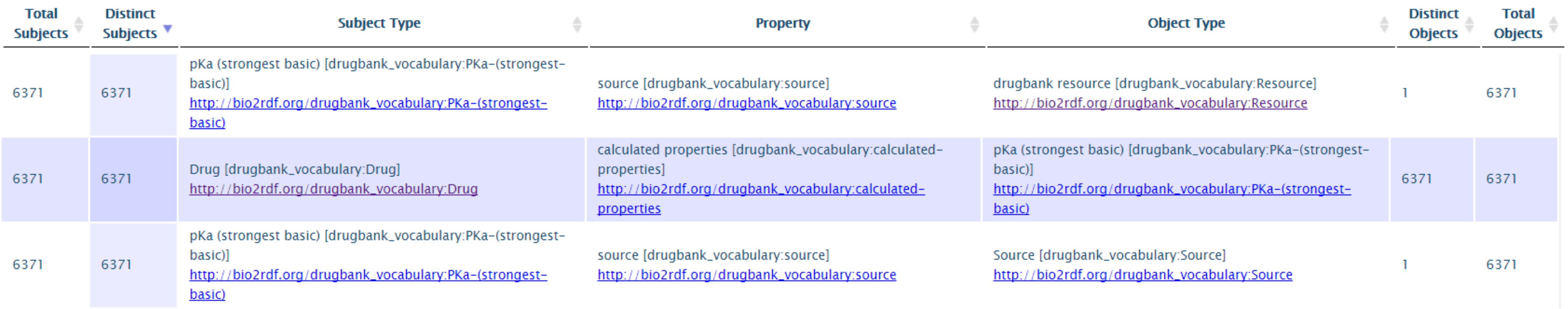
- Every URI is typed as an instance of an owl:Class, owl:ObjectProperty, or owl:DatatypeProperty
- SPARQL 1.1 endpoints using Virtuoso 7.1.0 or Yasgui

http://bio2rdf.org/sparql
http://legacy.yasgui.org/

BIO2RDF metrics can be used to develop SPARQL query

Each BIO2RDF database contains summary metrics about the dataset:

- Unique subject type-predicate-object type links and their frenquencies
- Unique subject type-predicate-literal links and their frenquencies



Total Subjects	Distinct Subjects ▼	Subject Type	Property	Object Type	Distinct Objects	Total Objects
6371	6371	pKa (strongest basic) [drugbank_vocabulary:PKa-(strongest-basic)] http://bio2rdf.org/drugbank_vocabulary:PKa-(strongest-basic)	source [drugbank_vocabulary:source] http://bio2rdf.org/drugbank_vocabulary:source	drugbank resource [drugbank_vocabulary:Resource] http://bio2rdf.org/drugbank_vocabulary:Resource	1	6371
6371	6371	Drug [drugbank_vocabulary:Drug] http://bio2rdf.org/drugbank_vocabulary:Drug	calculated properties [drugbank_vocabulary:calculated-properties] http://bio2rdf.org/drugbank_vocabulary:calculated-properties	pKa (strongest basic) [drugbank_vocabulary:PKa-(strongest-basic)] http://bio2rdf.org/drugbank_vocabulary:PKa-(strongest-basic)	6371	6371
6371	6371	pKa (strongest basic) [drugbank_vocabulary:PKa-(strongest-basic)] http://bio2rdf.org/drugbank_vocabulary:PKa-(strongest-basic)	source [drugbank_vocabulary:source] http://bio2rdf.org/drugbank_vocabulary:source	Source [drugbank_vocabulary:Source] http://bio2rdf.org/drugbank_vocabulary:Source	1	6371

Source: http://download.bio2rdf.org/release/3/drugbank/drugbank.html