Applied Databases

Lecture 19
Querying RDF with SPARQL

Sebastian Maneth

University of Edinburgh - March 21st, 2016

Outline

1. RDF
2. Turtle RDF Syntax
3. SPARQL
4. RDF Schema

The Semantic Web

→ term was coined by Tim Berners-Lee (W3C's director) in a 2001 article in Scientific American (with Hendler and Lassila)

→ extension of the Web through standards by the W3C

→ Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries"

The Semantic Web

→ term was coined by Tim Berners-Lee (W3C's director) in a 2001 article in Scientific American (with Hendler and Lassila)

→ extension of the Web through standards by the W3C

→ Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries"

An evolutionary state of the Web in which automated software can
→ store
→ exchange and
→ use
machine-readable information on the Web,
in turn enabling users to deal with the information
with greater efficiency and certainty.

The Semantic Web

→ term was coined by Tim Berners-Lee (W3C's director) in a 2001 article in Scientific American (with Hendler and Lassila)

→ extension of the Web through standards by the W3C

→ Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries"

Linked Data (Berners-Lee 2006)

1. Use URIs to name (identify) things.
2. Use HTTP URIs so that these things can be looked up ("dereferenced").
3. Provide useful information about what a name identifies when it's looked up, using open standards such as RDF, SPARQL, etc.
4. Refer to other things using their HTTP URI-based names when publishing data on the Web.

Uniform Resource Identifier (URI)

→ a string of characters used to identify a resource

→ most common form of a URI is the
Uniform Resource Locator (URL) aka "*web address*"

→ another form is the Uniform Resource Name (URN)
a URN identifies a resource by name in a particular namespace
e.g. ISBN 0-486-27557-4 cites unambiguously a specific
edition of Shakespeare's play *Romeo and Juliet*.

URN for that edition would be urn:isbn:0-486-27557-4

Uniform Resource Identifier (URI)

→ a string of characters used to identify a resource

→ most common form of a URI is the
Uniform Resource Locator (URL) aka “*web address*”

→ another form is the Uniform Resource Name (URN)
a URN identifies a resource by name in a particular namespace
e.g. ISBN 0-486-27557-4 cites unambiguously a specific
edition of Shakespeare's play *Romeo and Juliet*.

URN for that edition would be urn:isbn:0-486-27557-4

IRI = Internationalized Resource Identifier
→ extension of URI’s to use Unicode

1. Resource Description Framework (RDF)

→ W3C Recommendation February 1998

→ Revised Recommendations Feb 2004 (version 1.0)

→ Revised again in 2014 (version 1.1)

→ designed as a metadata data model

→ build
"a vendor-neutral and operating system-independent system of metadata"

1. Resource Description Framework (RDF)

→ W3C Recommendation February 1998

→ Revised Recommendations Feb 2004 (version 1.0)

→ Revised again in 2014 (version 1.1)

→ designed as a metadata data model

→ general method for conceptual description
or modeling of information

→ allows to make statements about resources

(“the sky”, “has”, “the color blue”) ← an RDF Triple

subject predicate object

1. Resource Description Framework (RDF)

RDF Triple: (subject, predicate, object)

→ subject denotes a resource

→ predicate denotes a trait or aspect of the resource;
it expresses a *relationship* between the subject and object

A collection of RDF Triples represents a labeled directed multi-graph

---

RDF is a data model.
Many ways to serialize e.g.

→ RDF/XML
→ Turtle
→ Jason-LD
→ N-Triples
→ N-Quads

2. Turtle RDF Syntax

Turtle = Terse RDF Triple Language (Dave Beckett, Tim Berners-Lee)

> foo:bar expands to
> http://example.org/ns#bar

URIs

Enclosed in <>

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

in the style of XML Qnames as a shorthand for the full URI

Blank Nodes

_:name

> Node representing a resource for which no URI and no literal is given. (can *only* be used as subject or object)

> e.g. John has a friend born on April 21st
>
> ```
> ex:John foaf:knows _:p1
> _:p1 foaf:birthDate 04-21
> ```

Literals

"Literal"

"Literal"@language

"""long literal with
newlines"""

Datatyped Literals

"lexical form"^^datatype URI

e.g. `"10"^^xsd:integer`
`"true"^^xsd:boolean`

(values) maybe be object, but *not* subject or predicate.

```
@prefix eric:    <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
eric:me contact:mailbox <mailto:e.miller123(at)example> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

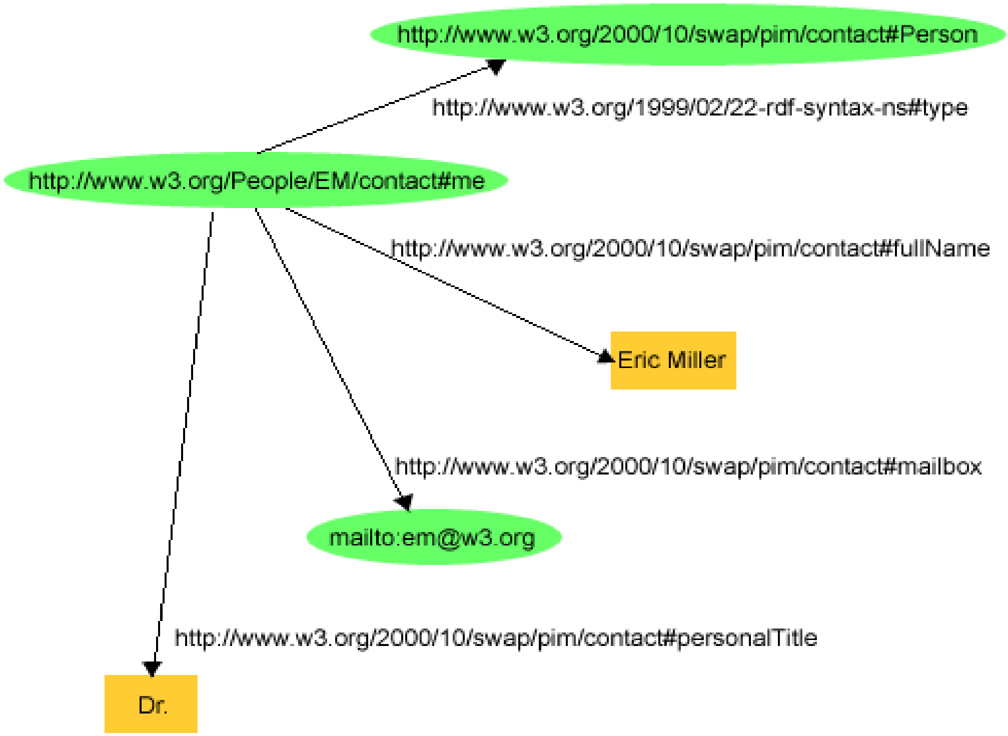
RDF Description of “Eric Miller” – in Turtle Syntax

literals

```
@prefix eric:    <http://www.w3.org/People/EM/contact#> .
@prefix contact: <http://www.w3.org/2000/10/swap/pim/contact#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

eric:me contact:fullName "Eric Miller" .
eric:me contact:mailbox <mailto:e.miller123(at)example> .
eric:me contact:personalTitle "Dr." .
eric:me rdf:type contact:Person .
```

RDF Description of “Eric Miller” – in Turtle Syntax



3. SPARQL

SPARQL Protocol and RDF Query Language

→ RDF Query Language

→ SPARQL 1.0, W3C Recommendation (2008)

→ SPARQL 1.1, W3C Recommendation (2013)

SPARQL query consists of

→ triple patterns
→ disjunctions
→ conjunctions
→ optional patterns

3. SPARQL by example

SPARQL queries consist of three parts:

1) Pattern matching part
   - → optional parts
   - → unions
   - → nesting
   - → filtering
2) Solution modifiers
   - → projection
   - → distinct
   - → order
   - → limit
   - → offset
3) Output
   - → yes/no
   - → selection of values
   - → construction of new triples
   - → description of resources

```
PREFIX

SELECT
    SELECT DISTINCT
    SELECT REDUCED
    CONSTRUCT

FROM
    FROM NAMED

WHERE

LIMIT
OFFSET
ORDER BY
```

3. SPARQL by example

Simplest query: ask for the existence of a single edge.

For instance, is there an edge (Amazon_River, length, ?x)
in the dbpedia RDF graph?

```
PREFIX prop: <http://dbpedia.org/property/>
ASK  {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
}
```

➔ Paste this query at http://dbpedia.org/sparql/

Answer:

→ true

3. SPARQL by example

Simplest query: ask for the existence of a single edge.

For instance, is there an edge (Amazon_River, length, ?x) in the dbpedia RDF graph?

Returns Boolean

"triple pattern"

```
PREFIX prop: <http://dbpedia.org/property/>
ASK {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
}
```

➔ Paste this query at http://dbpedia.org/sparql/

Answer:

→ true

3. SPARQL by example

"triple pattern"

```
PREFIX prop: <http://dbpedia.org/property/>
ASK  {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
}
```

A triple pattern P is a tuple of the form $(IL \cup V) \times (I \cup V) \times (IL \cup V)$
where $IL = I \cup L$ and
I = IRIs (Internationalized Resource Identifiers)
L = Literals
V = Variables

Let D be an RDF dataset.
$[[P]]_D = \{ \mu \mid dom(\mu) = var(P) \text{ and } \mu(P) \in D \}$
$[[(P1 \text{ UNION } P2)]]_D = [[P1]]_D \cup [[P2]]_D$

Note IRI's are the extension of URI's to use Unicode = "internationalized URI's"

3. SPARQL by example

Simplest query: ask for a particular value:

For instance, what is ?x for (Amazon_River, length, ?x)
in the dbpedia RDF graph?

```
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?x FROM {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
}
```

➔ Paste this query at http://dbpedia.org/sparql/

Answer:

→ "6800"^^<http://www.w3.org/2001/XMLSchema#int>

3. SPARQL by example

Simplest query: ask for a particular value:

For instance, what is ?x for (Amazon_River, length, ?x) in the dbpedia RDF graph?

```
PREFIX prop: <http://dbpedia.org/property/>
ASK {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
  <http://dbpedia.org/resource/Nile> prop:length ?y .
  FILTER(?x > ?y) .
}
```

Answer:

→ true

3. SPARQL by example

Simplest query: ask for a particular value:

For instance, what is ?x for (Amazon_River, length, ?x)
in the dbpedia RDF graph?

```
PREFIX prop: <http://dbpedia.org/property/>
ASK {
  <http://dbpedia.org/resource/Amazon_River> prop:length ?x .
  <http://dbpedia.org/resource/Nile> prop:length ?y .
  FILTER(?x > ?y) .
}
```

{ …. FILTER(..) …. } = Group Graph Pattern

→ Scope of FILTER is the group

→ FILTER can appear anywhere in group (same semantics)

3. SPARQL by example

Simplest query: ask for a particular value:
What properties/values are known about the Amazon river?

```
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?p ?x WHERE {
  <http://dbpedia.org/resource/Amazon_River> ?p ?x .
}
```

Answer:

Mozilla Firefox

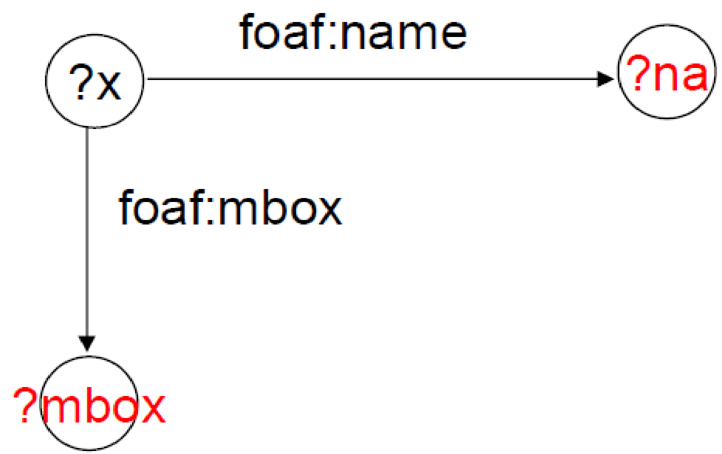
http://dbp...&debug=on

dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbp

p	x
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Thing
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/Place
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/Location
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.wikidata.org/entity/Q4022
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.wikidata.org/entity/Q47521
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/BodyOfWater
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/NaturalPlace
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/River
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/ontology/Stream
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/BodyOfWater
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/Place
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://schema.org/RiverBodyOfWater
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://umbel.org/umbel/rc/BodyOfWater
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://umbel.org/umbel/rc/Location_Underspecified
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://umbel.org/umbel/rc/River
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://umbel.org/umbel/rc/Stream
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/class/yago/BodyOfWater109225146
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/class/yago/River109411430
http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://dbpedia.org/class/yago/Stream109448361

→ Default semantics is CONJUNCTION:

```
PREFIX foaf: http://xmlns.com/foaf/0.10/
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```



$[[(P1 \text{ AND } P2]]_D = [[P1]]_D \text{ Join } [[P2]]_D$

$\Omega_1 \text{ Join } \Omega_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2$ are compatible mappings $\}$

$[[(P1 \text{ UNION } P2)]]_D = [[P1]]_D \cup [[P2]]_D$

Example: Arithmetic Filters

Data

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .
:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Query

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        FILTER (?price < 30.5)
        ?x dc:title ?title . }
```

Result

title	price
"The Semantic Web"	23

Example: String Filters

Data

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .
:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Query

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

Result

title	price
"SPARQL Tutorial"	42

3. SPARQL by example

Simplest query: ask for a particular value:
What properties/values are known about the Amazon river?

```
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?p ?x WHERE {
  <http://dbpedia.org/resource/Amazon_River> ?p ?x .
}
```

$$[[(P \text{ FILTER } R)]]_D = \{ \mu \in [[P]]_D \mid \mu \models R \}$$

R is an expression over AND, OR, NOT, =, and built-in conditions.
$\mu \models R$ means that μ satisfies R

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

OPT - Allows to add information to a mapping.

P1 = SELECT ?A, ?E, ?W WHERE
(?A email ?E) OPT (?A webPage ?W)

Select persons with email addresses, and, also include their web page, if it exists.

Ω_1 Join Ω = { $\mu_1 \cup \mu_2$ | $\mu_1 \in \Omega_1$, $\mu_2 \in \Omega_2$ are compatible mappings }

$\Omega_1 \setminus \Omega_2$ = { $\mu \in \Omega_1$ | for all $\mu' \in \Omega_2$, μ and μ' are not compatible }

Ω_1 # Ω_2 = (Ω_1 Join Ω_2) $\cup$ ($\Omega_1 \setminus \Omega_2$)

$[[(P1\ OPT\ P2)]]_D = [[P1]]_D\ \#\ [[P2]]_D$

OPT - Allows to add information to a mapping.

P1 = SELECT ?A, ?E, ?W WHERE
(?A email ?E) OPT (?A webPage ?W)

Select persons with email addresses, and, also include their web page, if it exists.

$$
\begin{aligned}
D = \{ \; & (B_1, \text{name}, \text{paul}), & & (B_1, \text{phone}, \text{777-3426}), \\
& (B_2, \text{name}, \text{john}), & & (B_2, \text{email}, \text{john@acd.edu}), \\
& (B_3, \text{name}, \text{george}), & & (B_3, \text{webPage}, \text{www.george.edu}), \\
& (B_4, \text{name}, \text{ringo}), & & (B_4, \text{email}, \text{ringo@acd.edu}), \\
& (B_4, \text{webPage}, \text{www.starr.edu}), & & (B_4, \text{phone}, \text{888-4537}), \quad \}
\end{aligned}
$$

$[\![P_1]\!]_D =$

	$?A$	$?E$	$?W$
μ_1 :	B_2	john@acd.edu	
μ_2 :	B_4	ringo@acd.edu	www.starr.edu

OPT - Allows to add information to a mapping.

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

Select all persons and includes their email,
then include web pages to those.

$$
\begin{aligned}
D = \{ \; &(B_1, \text{name}, \text{paul}), && (B_1, \text{phone}, \text{777-3426}), \\
&(B_2, \text{name}, \text{john}), && (B_2, \text{email}, \text{john@acd.edu}), \\
&(B_3, \text{name}, \text{george}), && (B_3, \text{webPage}, \text{www.george.edu}), \\
&(B_4, \text{name}, \text{ringo}), && (B_4, \text{email}, \text{ringo@acd.edu}), \\
&(B_4, \text{webPage}, \text{www.starr.edu}), && (B_4, \text{phone}, \text{888-4537}), \quad \}
\end{aligned}
$$

OPT - Allows to add information to a mapping.

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

Select all persons and includes their email,
then include web pages to those.

$$
\begin{array}{llll}
D = \{ & (B_1, \text{name}, \text{paul}), & (B_1, \text{phone}, \text{777-3426}), & \\
& (B_2, \text{name}, \text{john}), & (B_2, \text{email}, \text{john@acd.edu}), & \\
& (B_3, \text{name}, \text{george}), & (B_3, \text{webPage}, \text{www.george.edu}), & \\
& (B_4, \text{name}, \text{ringo}), & (B_4, \text{email}, \text{ringo@acd.edu}), & \\
& (B_4, \text{webPage}, \text{www.starr.edu}), & (B_4, \text{phone}, \text{888-4537}), & \}
\end{array}
$$

$[\![P_2]\!]_D =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		www.george.edu
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

OPT - Allows to add information to a mapping.

How is **P3** different from P2?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P3 = SELECT ?A, ?N, ?E, ?W WHERE
((?A name ?N) OPT ((?A email ?E) OPT (?A webPage ?W)))

$$
\begin{aligned}
D = \{\ & (B_1,\ \text{name},\ \text{paul}), && (B_1,\ \text{phone},\ \text{777-3426}),\\
& (B_2,\ \text{name},\ \text{john}), && (B_2,\ \text{email},\ \text{john@acd.edu}),\\
& (B_3,\ \text{name},\ \text{george}), && (B_3,\ \text{webPage},\ \text{www.george.edu}),\\
& (B_4,\ \text{name},\ \text{ringo}), && (B_4,\ \text{email},\ \text{ringo@acd.edu}),\\
& (B_4,\ \text{webPage},\ \text{www.starr.edu}), && (B_4,\ \text{phone},\ \text{888-4537}),\ \}
\end{aligned}
$$

$[\![P_2]\!]_D =$

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		www.george.edu
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

OPT - Allows to add information to a mapping.

How is **P3** different from P2?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P3 = SELECT ?A, ?N, ?E, ?W WHERE
((?A name ?N) OPT ((?A email ?E) OPT (?A webPage ?W)))

$$D = \{ (B_1, \text{name}, \text{paul}), (B_2, \text{name}, \text{john}), (B_3, \text{name}, \text{george}), (B_4, \text{name}, \text{ringo}), (B_4, \text{webPage}, \text{www.starr.edu}),$$
$$(B_1, \text{phone}, \text{777-3426}), (B_2, \text{email}, \text{john@acd.edu}), (B_3, \text{webPage}, \text{www.george.edu}), (B_4, \text{email}, \text{ringo@acd.edu}), (B_4, \text{phone}, \text{888-4537}), \}$$

P3

~~$[[P2]]_D$~~ =

	$?A$	$?N$	$?E$	$?W$
μ_1 :	B_1	paul		
μ_2 :	B_2	john	john@acd.edu	
μ_3 :	B_3	george		~~www.george.edu~~
μ_4 :	B_4	ringo	ringo@acd.edu	www.starr.edu

OPT - Allows to add information to a mapping.

What is the result for **P4**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P4 = SELECT ?A, ?N, ?E, ?W WHERE
((?A name ?N) AND ((?A email ?E) UNION (?A webPage ?W)))

$$
\begin{aligned}
D = \{\ & (B_1, \text{name}, \text{paul}), && (B_1, \text{phone}, \text{777-3426}), \\
& (B_2, \text{name}, \text{john}), && (B_2, \text{email}, \text{john@acd.edu}), \\
& (B_3, \text{name}, \text{george}), && (B_3, \text{webPage}, \text{www.george.edu}), \\
& (B_4, \text{name}, \text{ringo}), && (B_4, \text{email}, \text{ringo@acd.edu}), \\
& (B_4, \text{webPage}, \text{www.starr.edu}), && (B_4, \text{phone}, \text{888-4537}), \quad \}
\end{aligned}
$$

OPT - Allows to add information to a mapping.

What is the result for **P4**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P4 = SELECT ?A, ?N, ?E, ?W WHERE
((?A name ?N) AND ((?A email ?E) UNION (?A webPage ?W)))

$$
\begin{array}{llll}
D = \{ & (B_1, \text{name}, \text{paul}), & (B_1, \text{phone}, \text{777-3426}), & \\
& (B_2, \text{name}, \text{john}), & (B_2, \text{email}, \text{john@acd.edu}), & \\
& (B_3, \text{name}, \text{george}), & (B_3, \text{webPage}, \text{www.george.edu}), & \\
& (B_4, \text{name}, \text{ringo}), & (B_4, \text{email}, \text{ringo@acd.edu}), & \\
& (B_4, \text{webPage}, \text{www.starr.edu}), & (B_4, \text{phone}, \text{888-4537}), & \}
\end{array}
$$

$[\![P_4]\!]_D =$

	?A	?N	?E	?W
μ_1 :	B_2	john	john@acd.edu	
μ_2 :	B_3	george		www.george.edu
μ_3 :	B_4	ringo	ringo@acd.edu	
μ_4 :	B_4	ringo		www.starr.edu

OPT - Allows to add information to a mapping.

What is the result for **P4**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P4 = SELECT ?A, ?N, WHERE
((?A name ?N) AND ((?A email ?E) UNION (?A webPage ?W)))

$$D = \{ (B_1, \text{name}, \text{paul}), (B_1, \text{phone}, \text{777-3426}), (B_2, \text{name}, \text{john}), (B_2, \text{email}, \text{john@acd.edu}), (B_3, \text{name}, \text{george}), (B_3, \text{webPage}, \text{www.george.edu}), (B_4, \text{name}, \text{ringo}), (B_4, \text{email}, \text{ringo@acd.edu}), (B_4, \text{webPage}, \text{www.starr.edu}), (B_4, \text{phone}, \text{888-4537}), \}$$

$[\![P_4]\!]_D =$

	$?A$	$?N$
μ_1 :	B_2	john
μ_2 :	B_3	george
μ_3 :	B_4	ringo
μ_4 :	B_4	ringo

OPT - Allows to add information to a mapping.

What is the result for **P4**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P41 = SELECT **DISTINCT** ?A, ?N, WHERE
((?A name ?N) AND ((?A email ?E) UNION (?A webPage ?W)))

$$D = \{ (B_1, \text{name}, \text{paul}), (B_1, \text{phone}, \text{777-3426}), (B_2, \text{name}, \text{john}), (B_2, \text{email}, \text{john@acd.edu}), (B_3, \text{name}, \text{george}), (B_3, \text{webPage}, \text{www.george.edu}), (B_4, \text{name}, \text{ringo}), (B_4, \text{email}, \text{ringo@acd.edu}), (B_4, \text{webPage}, \text{www.starr.edu}), (B_4, \text{phone}, \text{888-4537}), \}$$

$[[P_4]]_D =$

	?A	?N
μ_1 :	B_2	john
μ_2 :	B_3	george
μ_3 :	B_4	ringo
μ_4 :	B_4	ringo

[[**P41**]]$_D$ =

	?A	?N
μ_1 :	B_2	john
μ_2 :	B_3	george
μ_3 :	B_4	ringo

OPT - Allows to add information to a mapping.

What is the result for **P5**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P5 = SELECT ?A, ?N, ?P WHERE
((?A name ?N) OPT ((?A phone ?P)) FILTER NOT(bound(?P)))

$$
\begin{aligned}
D = \{ \; & (B_1, \text{name}, \text{paul}), && (B_1, \text{phone}, \text{777-3426}), \\
& (B_2, \text{name}, \text{john}), && (B_2, \text{email}, \text{john@acd.edu}), \\
& (B_3, \text{name}, \text{george}), && (B_3, \text{webPage}, \text{www.george.edu}), \\
& (B_4, \text{name}, \text{ringo}), && (B_4, \text{email}, \text{ringo@acd.edu}), \\
& (B_4, \text{webPage}, \text{www.starr.edu}), && (B_4, \text{phone}, \text{888-4537}), \quad \}
\end{aligned}
$$

$\mu \models$ bound(?X) if ?X $\in$ dom(μ)

OPT - Allows to add information to a mapping.

What is the result for **P5**?

P2 = SELECT ?A, ?N, ?E, ?W WHERE
(((?A name ?N) OPT (?A email ?E)) OPT (?A webPage ?W))

P5 = SELECT ?A, ?N, ?P WHERE
((?A name ?N) OPT ((?A phone ?P)) FILTER NOT(bound(?P)))

$$D = \{ (B_1, \text{name}, \text{paul}), (B_1, \text{phone}, \text{777-3426}), (B_2, \text{name}, \text{john}), (B_2, \text{email}, \text{john@acd.edu}), (B_3, \text{name}, \text{george}), (B_3, \text{webPage}, \text{www.george.edu}), (B_4, \text{name}, \text{ringo}), (B_4, \text{email}, \text{ringo@acd.edu}), (B_4, \text{webPage}, \text{www.starr.edu}), (B_4, \text{phone}, \text{888-4537}), \}$$

$[\![P_5]\!]_D =$

	$?A$	$?N$	$?P$
μ_1 :	B_2	john	
μ_2 :	B_3	george	

```
PREFIX uni: <http://example.org/uni/>
SELECT ?name
FROM <http://example.org/personal>
WHERE { ?s uni:name ?name. ?s rdf:type uni:lecturer }
```

PREFIX
- Prefix mechanism for abbreviating URIs

SELECT
- Identifies the variables to be returned in the query answer
- SELECT DISTINCT
- SELECT REDUCED

FROM
- Name of the graph to be queried
- FROM NAMED

WHERE
- Query pattern as a list of triple patterns

LIMIT

OFFSET

ORDER BY

```
SELECT DISTINCT ?Author
WHERE
{
   ?Book rdf:type swrc:Book .
   ?Book dc:creator ?Author .
   ?Paper swrc:journal ?Journal .
   ?Paper dc:creator ?Author .
}
```

Select all authors that wrote a book and a journal.

Aggregates,
Expressions in the SELECT clause
Example

Data:

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

:org1 :affiliates :auth1, :auth2 .
:auth1 :writesBook :book1, :book2 .
:book1 :price 9 .
:book2 :price 5 .
:auth2 :writesBook :book3 .
:book3 :price 7 .
:org2 :affiliates :auth3 .
:auth3 :writesBook :book4 .
:book4 :price 7 .
```

Query:

```
PREFIX : <http://books.example/>
SELECT (SUM(?lprice) AS ?totalPrice)
WHERE {
  ?org :affiliates ?auth .
  ?auth :writesBook ?book .
  ?book :price ?lprice .
}
GROUP BY ?org
HAVING (SUM(?lprice) > 10)
```

Results:

totalPrice
21

Negation, Example

Data:

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

:alice  rdf:type   foaf:Person .
:alice  foaf:name  "Alice" .
:bob    rdf:type   foaf:Person .
```

Query:

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

SELECT ?person
WHERE
{
    ?person rdf:type  foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

Query Result:

person
<http://example/bob>

Property Paths

→ similar to XPath and regular expressions

`foaf:knows/foaf:name`	names of friends
`foaf:knows/foaf:knows/foaf:name`	names of friends of friends
`foaf:knows*`	
`foaf:knows{5,7}`	

path syntax constructs.

Property Paths

<table>
<tr><th>Syntax Form</th><th>Matches</th></tr>
<tr><td>iri</td><td>An IRI. A path of length one.</td></tr>
<tr><td>^elt</td><td>Inverse path (object to subject).</td></tr>
<tr><td>!iri or !(iri_1| ...|iri_n)</td><td>Negated property set. An IRI which is not one of iri_i. !iri is short for !(iri).</td></tr>
<tr><td>!^iri or !(iri_1| ...|iri_j|^iri_{j+1}| ...|^iri_n)</td><td>Negated property set with some inverse properties. An IRi which is n
iri_{j+1}...iri_n as reverse paths. !^iri is short for !(^iri).</td></tr>
<tr><td>(elt)</td><td>A group path elt, brackets control precedence.</td></tr>
<tr><td>elt1 / elt2</td><td>A sequence path of elt1 followed by elt2.</td></tr>
<tr><td>elt1 | elt2</td><td>A alternative path of elt1 or elt2 (all possibilities are tried).</td></tr>
<tr><td>elt*</td><td>A path of zero or more occurrences of elt.</td></tr>
<tr><td>elt+</td><td>A path of one or more occurrences of elt.</td></tr>
<tr><td>elt?</td><td>A path of zero or one occurrences of elt.</td></tr>
<tr><td>elt{n,m}</td><td>A path of between n and m occurrences of elt.</td></tr>
<tr><td>elt{n}</td><td>A path of exactly n occurrences of elt.</td></tr>
<tr><td>elt{n,}</td><td>A path of n or more occurrences of elt.</td></tr>
<tr><td>elt{,n}</td><td>A path of between 0 and n occurrences of elt.</td></tr>
</table>

4. RDF Schema

→ The **purpose** of RDF Schema is to provide an XML vocabulary to:

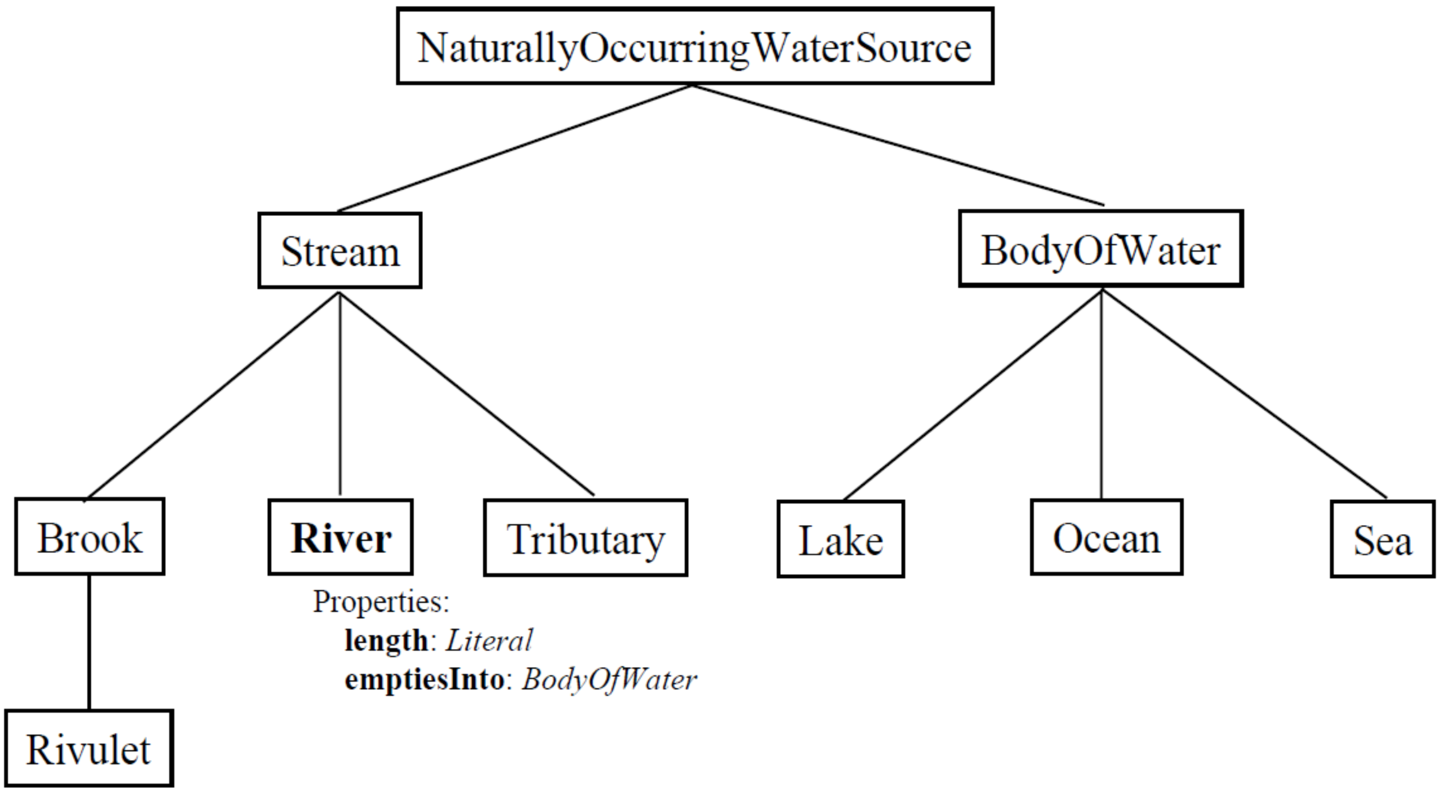
-- express classes and their (subclass) relationships.

-- define properties and associate them with classes.

→ The **benefit** of an RDF Schema is that it facilitates inferencing on your data, and enhanced searching.

4. RDF Schema

Is about generating Taxonomies! (class hieararchies)



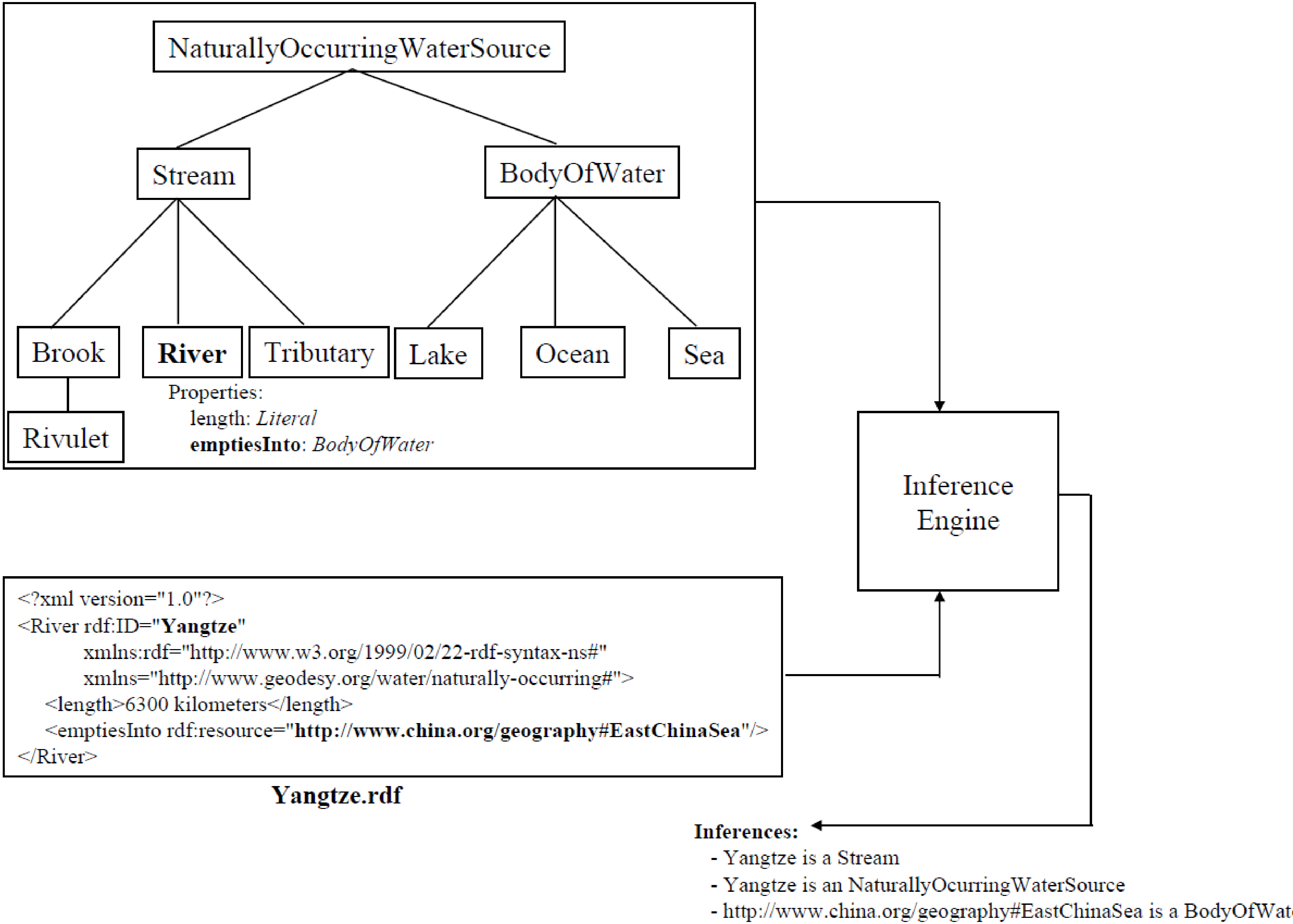
4. RDF Schema

What inferences can be made with this data?
Using the taxonomy of the previous slide.

```
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
    <length>6300 kilometers</length>
    <emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
```

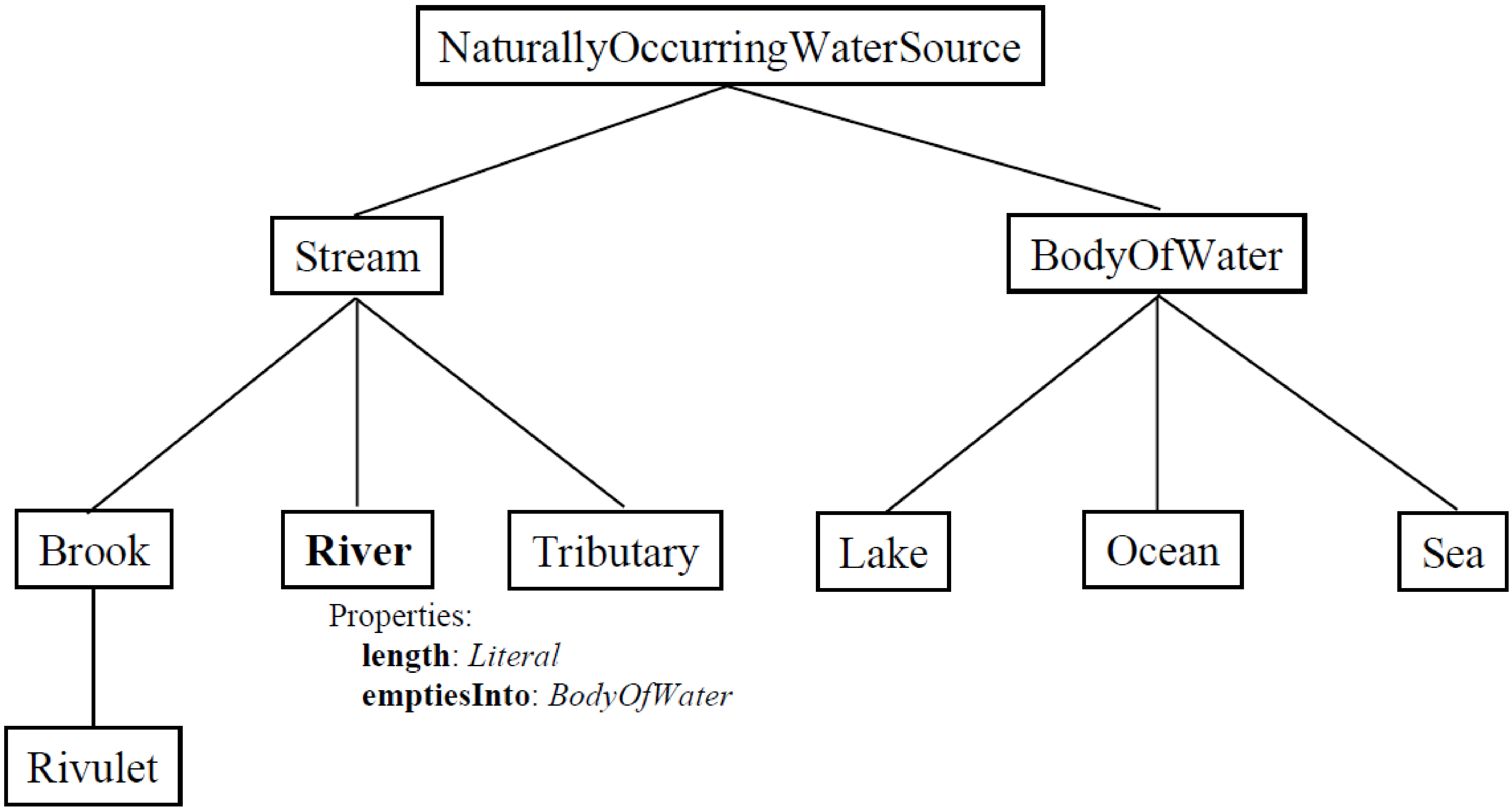
Yangtze.rdf

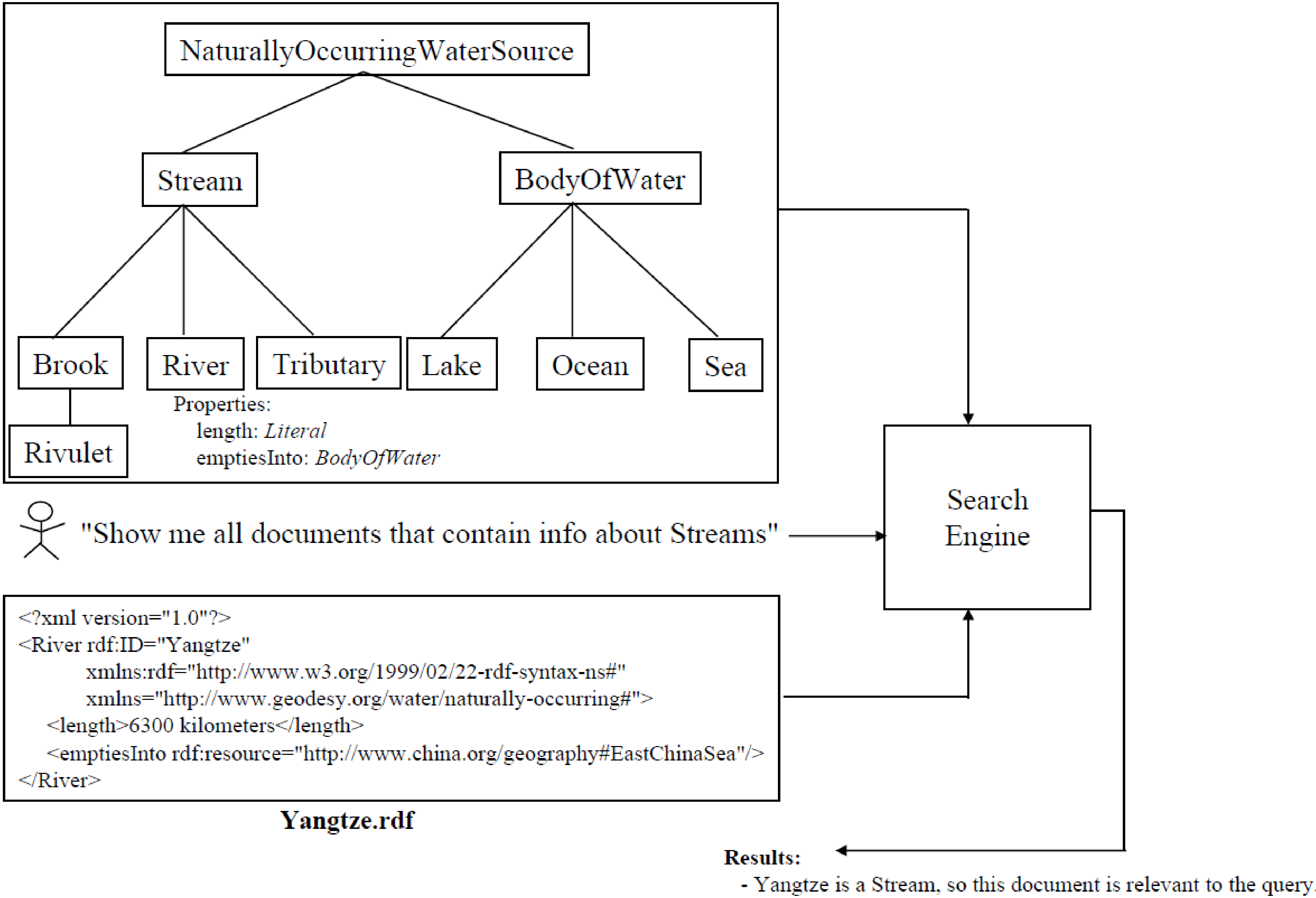
Inferences are made by examining a taxonomy that contains River.
See next slide.



NaturallyOccurringWaterSource
Stream
BodyOfWater
Brook
River
Tributary
Lake
Ocean
Sea
Rivulet
Properties:
length: Literal
emptiesInto: BodyOfWater
Inference Engine
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.geodesy.org/water/naturally-occurring#">
<length>6300 kilometers</length>
<emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
Yangtze.rdf
Inferences:
- Yangtze is a Stream
- Yangtze is an NaturallyOcurringWaterSource
- http://www.china.org/geography#EastChinaSea is a BodyOfWat

How does a taxonomy facilitate searching?




NaturallyOccurringWaterSource
Stream
BodyOfWater
Brook
River
Tributary
Lake
Ocean
Sea
Rivulet
Properties:
length: *Literal*
emptiesInto: *BodyOfWater*
"Show me all documents that contain info about Streams"
Search Engine
<?xml version="1.0"?>
<River rdf:ID="Yangtze"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.geodesy.org/water/naturally-occurring#">
<length>6300 kilometers</length>
<emptiesInto rdf:resource="http://www.china.org/geography#EastChinaSea"/>
</River>
Yangtze.rdf
Results:
- Yangtze is a Stream, so this document is relevant to the query.

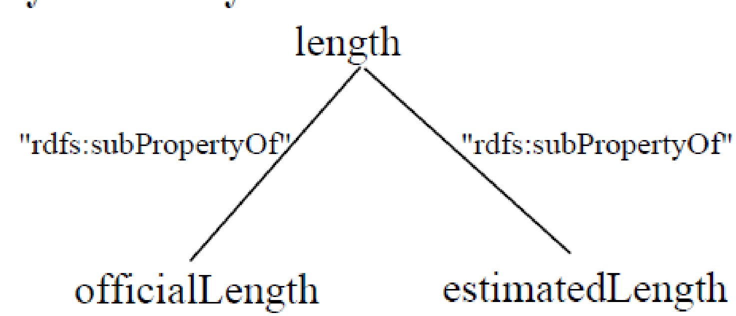
Classes have Properties.
Properties may have Subproperties.

Classes →

	Stream	Brook	Rivulet	River	Tributary
length	X	X	X	X	X
emptiesInto				X	
obstacle				X	
estimatedLength	X	X	X	X	X
officialLength	X	X	X	X	X

↑ Properties

Property Hierarchy:



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:XMLLiteral	The class of XML literals 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, ..., all of which are sub-properties of 'member'.
rdf:List	The class of RDF Lists.

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 (see the RDF Primer for an example of its usage).	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

END

Lecture 19