Parallelle og distribuerte databaser – del IV

Today's topics:

- Resource Description Framework (RDF)
- SPARQL: RDF query language
- Demo

Semantic Graph DBs

- Also known as **triplestores** or **RDF stores**
- Store and retrieve “triples” (structures of the form subject-predicate-object, e.g. “John knows Per")
- Standardized NoSQL solution – built upon W3C's Linked Data technology stack
  - RDF: uniform standard data model
  - SPARQL: powerful standard query language
- Examples: AllegroGraph, Virtuoso, GraphDB, Jena TBD

  (more examples at https://www.w3.org/wiki/LargeTripleStores)

Intro to Resource Description Framework (RDF)

(Most of the examples in the upcoming slides are taken from: http://www.w3.org/TR/rdf-primer/)

- RDF is a language that enable to describe making statements on resources
  - John is father of Ann
- Statement (or triple) as a logical formula P(x, y), where the binary predicate P relates the object x to the object y
- Triple data model:

  <subject, predicate, object>

  - **Subject**: Resource or blank node
  - **Predicate**: Property
  - **Object**: Resource (or collection of resources), literal or blank node
- Example:

  <ex:john, ex:father-of, ex:ann>
- RDF offers only binary predicates (properties)

Resources

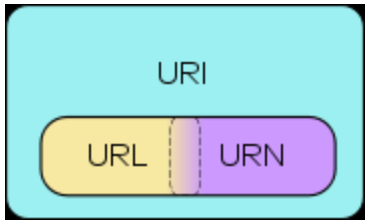
- A resource may be:
  - Web page (e.g. `http://www.w3.org`)
  - A person (e.g. `http://www.w3.org/People/Berners-Lee/`)
  - A book (e.g. `urn:isbn:4-534-34674-4`)
  - Anything denoted with a URI!

- A URI is an *identifier* and **not** a location on the Web

- RDF allows making statements about resources:
  - `http://www.w3.org` **has the format** text/html
  - `http://www.w3.org/People/Berners-Lee/` **has first name** Tim
  - `urn:isbn:0-345-33971-1` **has author** John

URI, URN, URL

- A Uniform Resource Identifier (URI) is a string of characters used to identify a name or a resource on the Internet

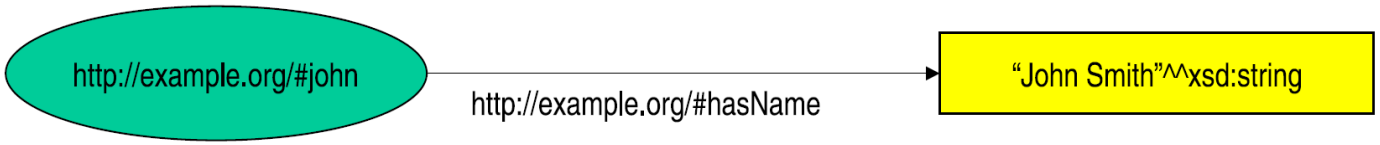


- A URI can be a URL or a URN
- A Uniform Resource Name (URN) defines an item's identity
  - the URN *urn:isbn:urn:isbn:4-534-34674-4* is a URI that specifies the identifier system, i.e. International Standard Book Number (ISBN), as well as the unique reference within that system and allows one to talk about a book, but doesn't suggest where and how to obtain an actual copy of it
- A Uniform Resource Locator (URL) provides a method for finding it
  - the URL *http://www.uio.no/studier/emner/matnat/ifi/INF3100/* identifies a resource (INF3100's home page) and implies that a representation of that resource (such as the home page's current HTML code, as encoded characters) is obtainable via HTTP from a network host named *https://www.uio.no*

Literals

- Plain literals
  - E.g. **`"any text"`**
  - Optional language tag, e.g. **`"Hello, how are you?"@en-GB`**
- Typed literals
  - E.g. **`"hello"^^xsd:string`**, **`"1"^^xsd:integer`**
  - Recommended datatypes:
    - XML Schema datatypes
- Only as *object* of a triple, e.g.:

```
⟨<http://example.org/#john>,
                  <http://example.org/#hasName>,
                                        "John Smith"^^xsd:string⟩
```



Datatypes

- One pre-defined datatype: **`rdf:XMLLiteral`**
  - Used for embedding XML in RDF

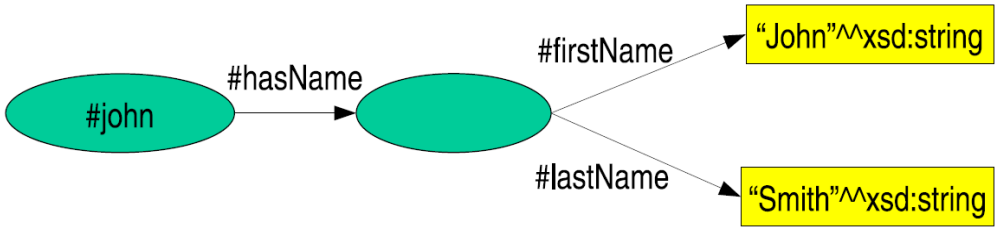
- Recommended datatypes are XML Schema datatypes, e.g.:
  - **`xsd:string`**
  - **`xsd:integer`**
  - **`xsd:float`**
  - **`xsd:anyURI`**
  - **`xsd:boolean`**

Blank Nodes I

- Blank nodes are nodes without a URI
  - Unnamed resources
  - More complex constructs
- Representation of blank nodes is syntax-dependent
  - *Blank node identifier*

- For example:

```
⟨<#john>, <#hasName>, _:johnsname⟩
⟨_:johnsname, <#firstName>, "John"^^xsd:string⟩
⟨_:johnsname, <#lastName>, "Smith"^^xsd:string⟩
```



Blank Nodes II

- **Representation of complex data**

  A blank node can be used to indirectly attach to a resource a consistent set of properties which together represent a complex data

- **Anonymous classes in OWL**

  The ontology language OWL uses blank nodes to represent anonymous classes such as unions or intersections of classes, or classes called restrictions, defined by a constraint on a property

RDF Containers

- Grouping property values:

"The lecture is attended by John, Mary and Chris" **`Bag`**

"[RDF-Concepts] is edited by Graham and Jeremy (in that order)" **`Seq`**

"The source code for the application may be found at ftp1.example.org, ftp2.example.org, ftp3.example.org" **`Alt`**

RDF Containers 2

- Three types of containers:
  - **`rdf:Bag`** - unordered set of items
  - **`rdf:Seq`** - ordered set of items
  - **`rdf:Alt`** - set of alternatives

- Every container has a triple declaring the **`rdf:type`**

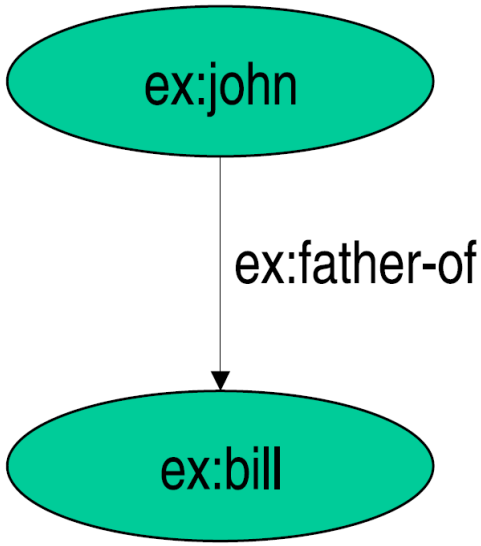
- Items in the container are denoted with
  - **`rdf:_1, rdf:_2, . . . ,rdf:_n`**

RDF Containers 2

- Three types of containers:
  - `rdf:Bag` - unordered set of items
  - `rdf:Seq` - ordered set of items
  - `rdf:Alt` - set of alternatives
- Every container has a triple declaring the `rdf:type`
- Items in the container are denoted with
  - `rdf:_1, rdf:_2, . . . ,rdf:_n`
- Limitations:
  - Semantics of the container is up to the application
  - What about closed sets?
    - How do we know whether Graham and Jeremy are the only editors of [RDF-Concepts]?

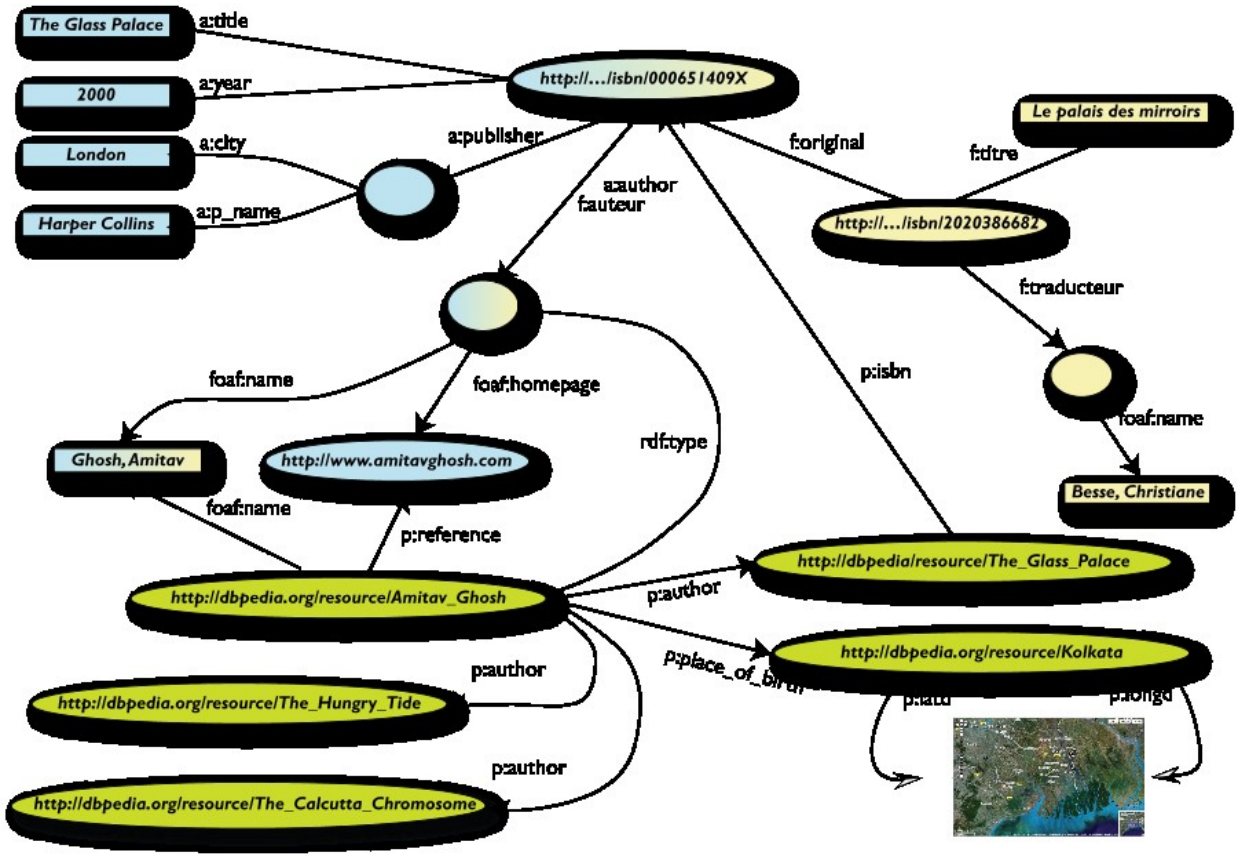
RDF Triple Graph Representation

- The triple data model can be represented as a graph
- Such graph is called in the Artificial Intelligence community a **semantic net**
- Labeled, directed graphs
  - **Nodes**: resources, literals
  - **Labels**: properties
  - **Edges**: statements



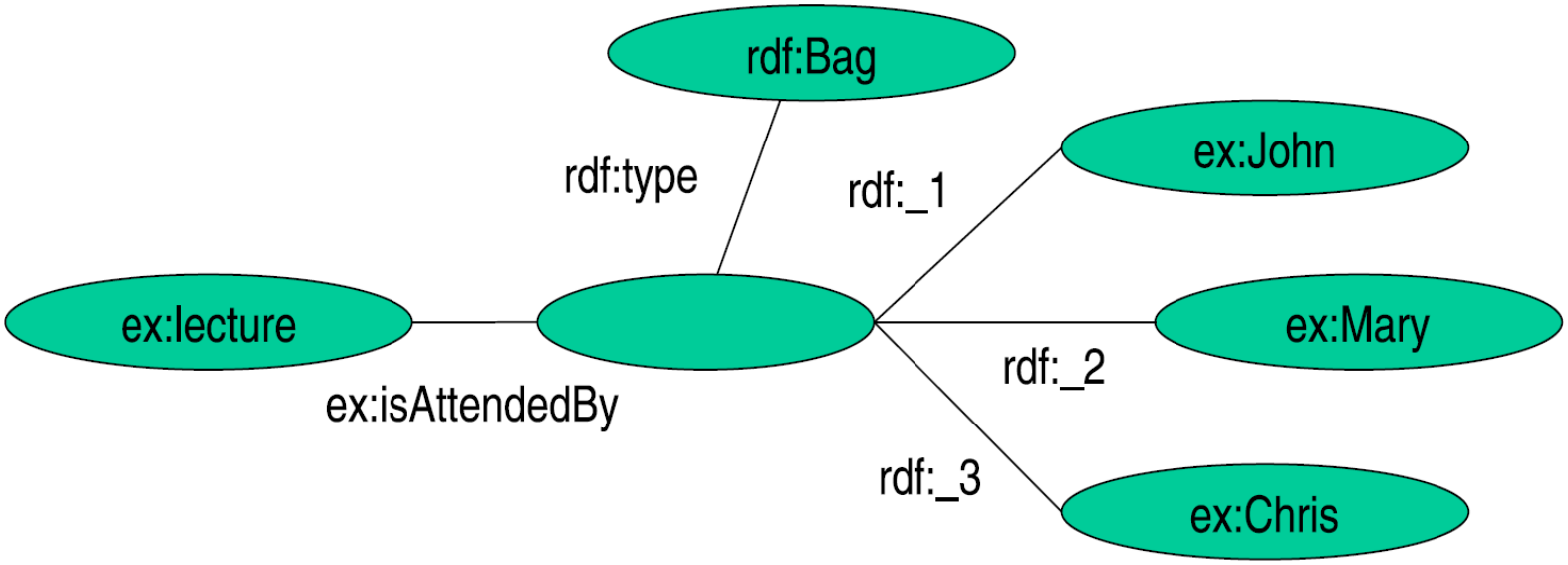
RDF: a Direct Connected Graph based Model

- Different interconnected triples lead to a more complex graphic model
- Basically a RDF document is a direct connect graph
  - http://en.wikipedia.org/wiki/Connectivity_%28graph_theory%29



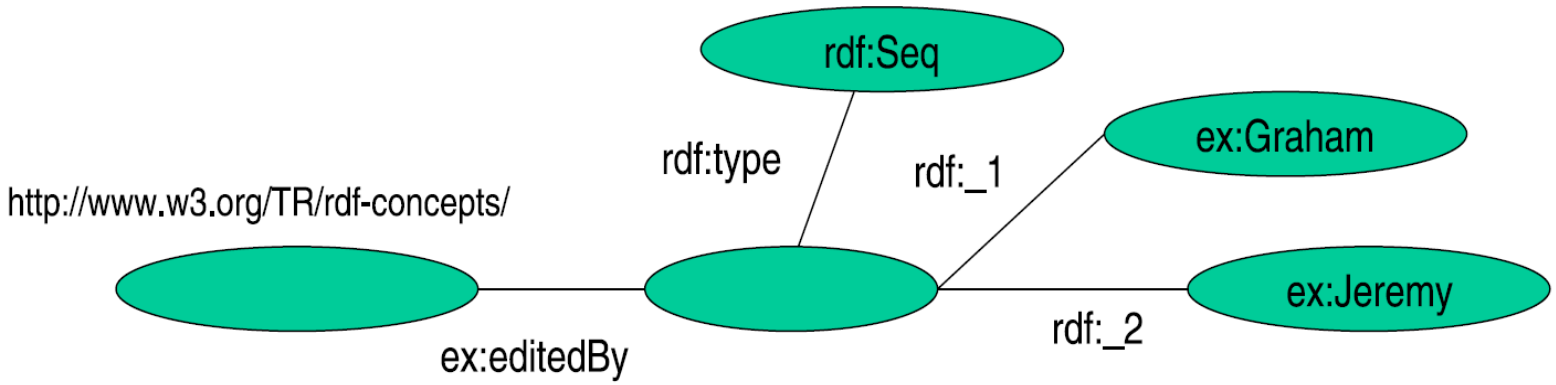
RDF Containers Graph Representation: Bag

"The lecture is attended by John, Mary and Chris"



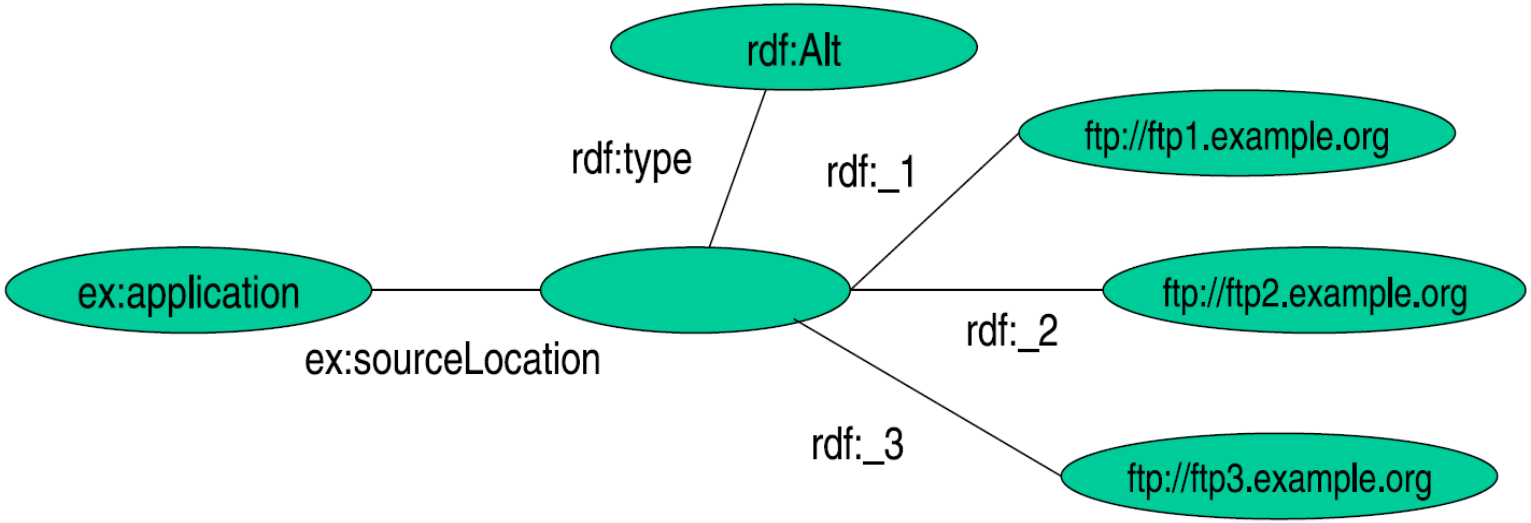
RDF Containers Graph Representation: Seq

"[RDF-Concepts] is edited by Graham and Jeremy (in that order)"



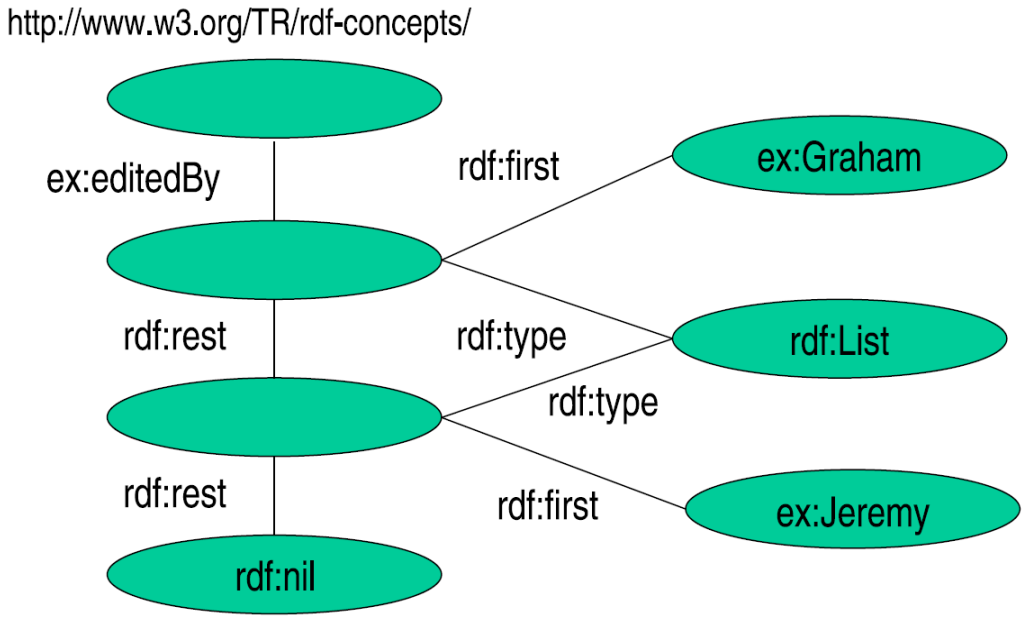
RDF Containers Graph Representation: Alt

"The source code for the application may be found at ***ftp1.example.org, ftp2.example.org, ftp3.example.org****"*



RDF Collections

"[RDF-Concepts] is edited by Graham and Jeremy (in that order) and nobody else"



RDF provides support for describing groups containing only the specified members, in the form of RDF collections.

Reification I

- Reification: statements about statements

Mary claims that John's name is "John Smith".

```
⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩
```

This kind of statement can be used to describe belief or trust in other statements, which is important in some kinds of applications

Necessary because there are only triples in RDF: we cannot add an identifier directly to a triple (then it would be a quadruple)

Reification II

- Reification: statements about statements

Mary claims that John's name is "John Smith".

⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩

⟨<#john>, <#hasName>, "John Smith"⟩

In such a way we attached a label to the statement.

Reification III

- Reification: statements about statements

Mary claims that John's name is "John Smith".

```
⟨<#myStatement>, rdf:type, rdf:Statement⟩
⟨<#myStatement>, rdf:subject, <#john>⟩
⟨<#myStatement>, rdf:predicate, <#hasName>⟩
⟨<#myStatement>, rdf:object, "John Smith"⟩

⟨<#mary>, <#claims>, <#myStatement>⟩
```

RDF uses only binary properties. This restriction seems quite serious because often we use predicates with more than two arguments. Luckily, such predicates can be simulated by a number of binary predicates.

RDF Vocabulary

- RDF defines a number of resources and properties
- We have already seen: **`rdf:XMLLiteral`**, **`rdf:type`**, . . .
- RDF vocabulary is defined in the namespace:
  `http://www.w3.org/1999/02/22-rdf-syntax-ns#`

- Classes:
  - **`rdf:Property`**, **`rdf:Statement`**, **`rdf:XMLLiteral`**
  - **`rdf:Seq`**, **`rdf:Bag`**, **`rdf:Alt`**, **`rdf:List`**
- Properties:
  - **`rdf:type`**, **`rdf:subject`**, **`rdf:predicate`**, **`rdf:object`**,
  - **`rdf:first`**, **`rdf:rest`**, **`rdf:_n`**
  - **`rdf:value`**
- Resources:
  - **`rdf:nil`**

RDF Vocabulary

- Typing using **`rdf:type`**:

  `<A, rdf:type, B>`

  "A belongs to class B"

- All properties belong to class **`rdf:Property`**:

  `<P, rdf:type, rdf:Property>`

  "P is a property"

  `<rdf:type, rdf:type, rdf:Property>`

  "rdf:type is a property"

RDF Schema (RDFS)

- Types in RDF:

  <#john, rdf:type, #Student>

- What is a "**#Student**"?

- RFD is not defining a vocabulary about the statements, but only to express statements

- We know that "**#Student**" identifies a category (a concept or a class), but this is only implicitly defined in RDF

RDF Schema (RDFS)

- We need a language for defining RDF types:
  - Define classes:
    - *"**`#Student`** is a class"*
  - Relationships between classes:
    - *"**`#Student`** is a sub-class of **`#Person`**"*
  - Properties of classes:
    - *"**`#Person`** has a property **`hasName`**"*

- RDF Schema is such a language

RDF Schema (RDFS)

- Classes:
  `<#Student, rdf:type, #rdfs:Class>`
- Class hierarchies:
  `<#Student, rdfs:subClassOf, #Person>`

- Properties:
  `<#hasName, rdf:type, rdf:Property>`
- Property hierarchies:
  `<#hasMother, rdfs:subPropertyOf, #hasParent>`

- Associating properties with classes (a):
  - "The property `#hasName` only applies to `#Person`"
    `<#hasName, rdfs:domain, #Person>`

- Associating properties with classes (b):
  - "The type of the property `#hasName` is `#xsd:string`"
    `<#hasName, rdfs:range, xsd:string>`

RDFS Vocabulary

- RDFS Extends the RDF Vocabulary
- RDFS vocabulary is defined in the namespace:

`http://www.w3.org/2000/01/rdf-schema#`

RDFS Classes

- `rdfs:Resource`
- `rdfs:Class`
- `rdfs:Literal`
- `rdfs:Datatype`
- `rdfs:Container`
- `rdfs:ContainerMembershipProperty`

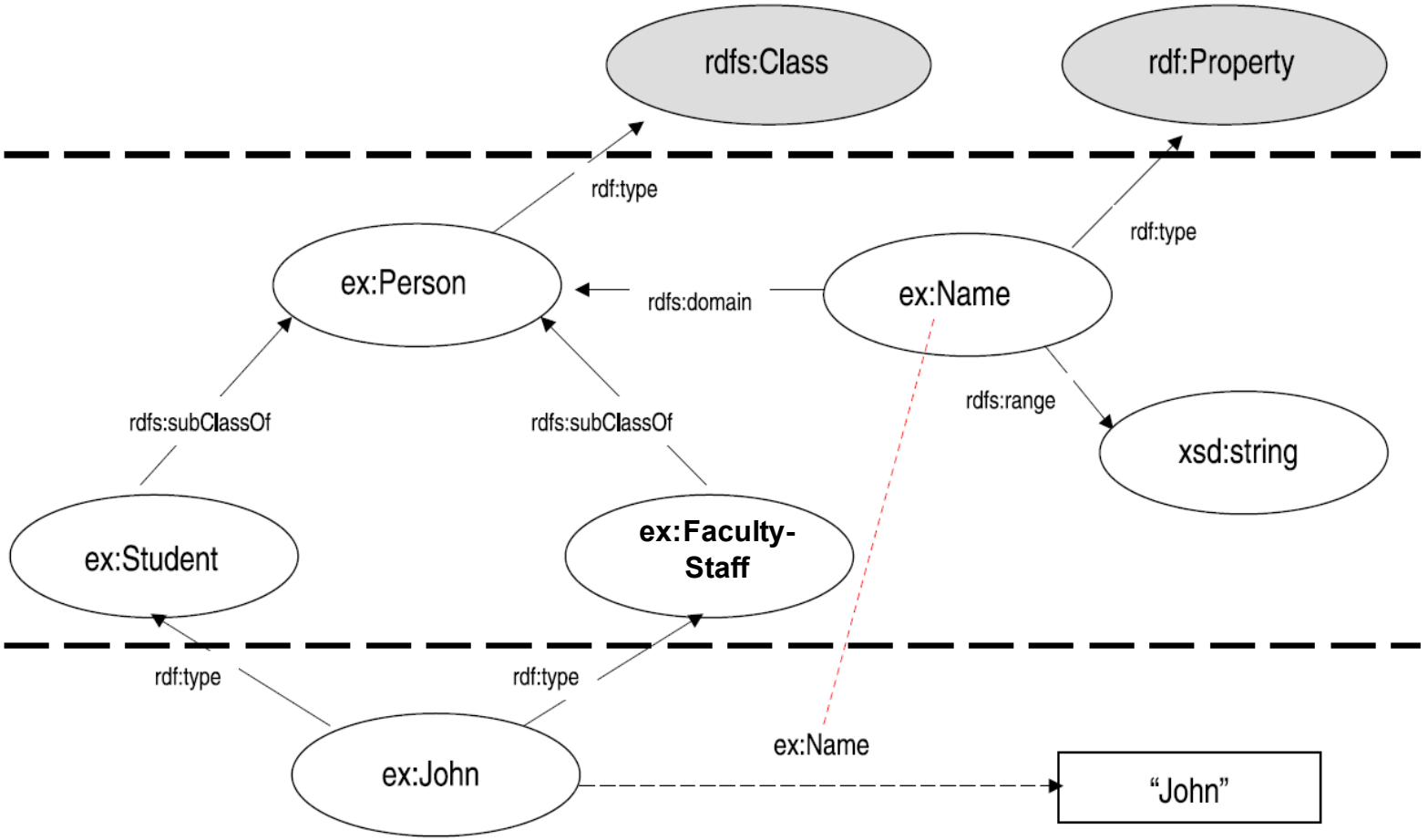
RDFS Properties

- `rdfs:domain`
- `rdfs:range`
- `rdfs:subPropertyOf`
- `rdfs:subClassOf`
- `rdfs:member`
- `rdfs:seeAlso`
- `rdfs:isDefinedBy`
- `rdfs:comment`
- `rdfs:label`

RDFS Principles

- **Resource**
  - All resources are implicitly instances of `rdfs:Resource`
- **Class**
  - Describe sets of resources
  - Classes are resources themselves - e.g. Webpages, people, document types
    - Class hierarchy can be defined through `rdfs:subClassOf`
    - Every class is a member of `rdfs:Class`
- **Property**
  - Subset of RDFS Resources that are properties
    - **Domain**: class associated with property: `rdfs:domain`
    - **Range**: type of the property values: `rdfs:range`
    - Property hierarchy defined through: `rdfs:subPropertyOf`

RDFS Example



RDFS Metadata Properties

- Metadata is “data about data”
- Any meta-data can be attached to a resource, using:
  - **`rdfs:comment`**
    - Human-readable description of the resource, e.g.
      - ⟨**`<ex:Person>, rdfs:comment, "A person is any human being"`**⟩
  - **`rdfs:label`**
    - Human-readable version of the resource name, e.g.
      - ⟨**`<ex:Person>, rdfs:label, "Human being"`**⟩
  - **`rdfs:seeAlso`**
    - Indicate additional information about the resource, e.g.
      - ⟨**`<ex:Person>, rdfs:seeAlso, <http://xmlns.com/wordnet/1.6/Human>`**⟩
  - **`rdfs:isDefinedBy`**
    - A special kind of rdfs:seeAlso, e.g.
      - ⟨**`<ex:Person>,rdfs:isDefinedBy,<http://xmlns.com/wordnet/1.6/Human>`**⟩

Databases and RDF (cont')

- Relational database are a well established technology to store information and provide query support (SQL)
- Relational database have been designed and implemented to store concepts in a predefined (not frequently alterable) schema.

- How can we store the following RDF data in a relational database?

```
<rdf:Description rdf:about="12345">
      <rdf:type rdf:resource="&uni;lecturer"/>
      <uni:name>Joe Doe</uni:name>
      <uni:title>University Professor</uni:title>
</rdf:Description>
```

Databases and RDF

- Possible approach: Relational "Traditional" approach

Lecturer		
id	**name**	**title**
12345	Joe Doe	University Professor

- We can create a table "Lecturer" to store information about the "Lecturer" RDF Class.
- Query: Find the names of all the lecturers

```
SELECT NAME FROM LECTURER
```

- Drawbacks: Every time we need to add new content we have to create a new table -> Not scalable, not dynamic, not based on the RDF principles (triples)

Databases and RDF

- Another possible approach: Relational "Triple" based approach

Statement			
Subject	**Predicate**	**ObjectURI**	**ObjectLiteral**
101	102	103	null
101	104		201
101	105		202
103	…	…	null

Resources	
Id	**URI**
101	21345
102	rdf:type
103	uni:lecturer
104	…

Literals	
Id	**Value**
201	Joe Doe
202	University Professor
203	…
…	…

- We can create a table to maintain all the triples S P O (and distinguish between URI objects and literals objects)
- Drawbacks: We are flexible w.r.t. adding new statements dynamically without any change to the database structure…but what about querying?
  - Query: Find the names of all the lecturers
  - The query is quite complex: 5 JOINS!
  - This require a lot of optimization specific for RDF and triple data storage, that it is not included in the DB
  - For achieving efficiency a layer on top of a database is required
  - SQL is not appropriate to extract RDF fragments

```
SELECT L.Value FROM Literals AS L
INNER JOIN Statement AS S ON S.ObjectLiteral=L.ID
INNER JOIN Resources AS R ON R.ID=S.Predicate
INNER JOIN Statement AS S1 ON
S1.Predicate=S.Predicate
INNER JOIN Resources AS R1 ON R1.ID=S1.Predicate
INNER JOIN Resources AS R2 ON R2.ID=S1.ObjectURI
WHERE R.URI = "uni:name"
AND R1.URI = "rdf:type"
AND R2.URI = "uni:lecturer"
```

SPARQL: RDF Query language

- SPARQL
  - RDF Query language
  - Uses SQL-like syntax

- Example:

```
PREFIX uni: <http://example.org/uni/>

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

SPARQL Queries

```
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

SPARQL Query keywords

- PREFIX: based on namespaces
- DISTINCT: The DISTINCT solution modifier eliminates duplicate solutions. Specifically, each solution that binds the same variables to the same RDF terms as another solution is eliminated from the solution set.
- REDUCED: While the DISTINCT modifier ensures that duplicate solutions are eliminated from the solution set, REDUCED simply permits them to be eliminated. The cardinality of any set of variable bindings in an REDUCED solution set is at least one and not more than the cardinality of the solution set with no DISTINCT or REDUCED modifier.
- LIMIT: The LIMIT clause puts an upper bound on the number of solutions returned. If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned.

SPARQL Query keywords

- OFFSET: OFFSET causes the solutions generated to start after the specified number of solutions. An OFFSET of zero has no effect.
- ORDER BY: The ORDER BY clause establishes the order of a solution sequence.
- Following the ORDER BY clause is a sequence of order comparators, composed of an expression and an optional order modifier (either ASC() or DESC()). Each ordering comparator is either ascending (indicated by the ASC() modifier or by no modifier) or descending (indicated by the DESC() modifier).

Example RDF Graph

```
<http://example.org/#john> <http://.../vcard-rdf/3.0#FN> "John Smith"

<http://example.org/#john> <http://.../vcard-rdf/3.0#N> :_X1
_:X1 <http://.../vcard-rdf/3.0#Given> "John"
_:X1 <http://.../vcard-rdf/3.0#Family> "Smith"

<http://example.org/#john> <http://example.org/#hasAge> "32"

<http://example.org/#john> <http://example.org/#marriedTo> <#mary>

<http://example.org/#mary> <http://.../vcard-rdf/3.0#FN> "Mary Smith"

<http://example.org/#mary> <http://.../vcard-rdf/3.0#N> :_X2
_:X2 <http://.../vcard-rdf/3.0#Given> "Mary"
_:X2 <http://.../vcard-rdf/3.0#Family> "Smith"

<http://example.org/#mary> <http://example.org/#hasAge> "29"
```

SPARQL Queries: All Full Names

"Return the full names of all people in the graph"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?fullName
WHERE {?x vCard:FN ?fullName}
```

result:

```
fullName
================
"John Smith"
"Mary Smith"
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Properties

"Return the relation between John and Mary"

```
PREFIX ex: <http://example.org/#>
SELECT ?p
WHERE {ex:john ?p ex:mary}
```

result:

```
p
==================
<http://example.org/#marriedTo>
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Complex Patterns

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ex: <http://example.org/#>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
       ?x ex:marriedTo ?y}
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Complex Patterns

"Return the spouse of a person by the name of John Smith"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ex: <http://example.org/#>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
       ?x ex:marriedTo ?y}
```

result:

```
y
==================
<http://example.org/#mary>
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Blank Nodes

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
       ?name vCard:Given ?firstName}
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Blank Nodes

"Return the first name of all people in the KB"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
       ?name vCard:Given ?firstName}
```

result:

```
name firstName
==============
_:a "John"
_:b "Mary"
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Building RDF Graph

"Rewrite the naming information in original graph by using the **`foaf:name`**"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x vCard:FN ?name }
```

result:

```
#john foaf:name "John Smith"
#marry foaf:name "Marry Smith"
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Building RDF Graph

"Rewrite the naming information in original graph by using the **`foaf:name`**"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x vCard:FN ?name }
```

result:

```
#john foaf:name "John Smith"
#marry foaf:name "Marry Smith"
```

```
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
                 syntax-ns#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:ex="http://example.org">
  <rdf:Description rdf:about=ex:john>
       <foaf:name>John Smith</foaf:name>
  </rdf:Description>
  <rdf:Description rdf:about=ex:marry>
       <foaf:name>Marry Smith</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Testing if the Solution Exists

"Are there any married persons in the KB?"

```
PREFIX ex: <http://example.org/#>
ASK { ?person ex:marriedTo ?spouse }
```

result:

yes

==========

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Constraints (Filters)

"Return all people over 30 in the KB"

```
PREFIX ex: <http://example.org/#>
SELECT ?x
WHERE {?x hasAge ?age .
FILTER(?age > 30)}
```

result:

```
x
==================
<http://example.org/#john>
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Optional Patterns

"Return all people and (optionally) their spouse"

```
PREFIX ex: <http://example.org/#>
SELECT ?person, ?spouse
WHERE {?person ex:hasAge ?age .
OPTIONAL { ?person ex:marriedTo ?spouse } }
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

result:

```
?person ?spouse
============================
<http://example.org/#mary>
<http://example.org/#john> <http://example.org/#mary>
```

SPARQL Queries: Traversing Class-Property hierarchy

"Return all people and their ancestors of all levels in KB"

```
PREFIX ex: <http://example.org/#>
SELECT ?person, ?ancestor
WHERE {?person (ex:hasParent)* ?ancestor .
}
```

result:

```
?person ?ancestor
============================
<http://example.org/#john> <http://example.org/#john>
<http://example.org/#john> <http://example.org/#bob>
<http://example.org/#john> <http://example.org/#mary>
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasParent :bob .
ex:bob
  vcard:FN "Bob Smith" ;
  vcard:N [
    vcard:Given "Bob" ;
    vcard:Family "Smith" ] ;
  ex:hasParent :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
```

SPARQL Queries: Functions

1. Expressions

   BIND, IF

2. Functions

   a) General functions

      STR, IRI, BOUND

   b) Functions on strings

      SUBSTR, REGEX, REPLACE

   c) Functions on numerics

      ABS, ROUND

   d) Functions on dates and times

      NOW, YEAR, MONTH, DAY

SPARQL Queries: Functions

"Return all people and their and (optionally) their spouse. If no spouse – return "Unmarried" "

```
PREFIX ex: http://example.org/#
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?fullName, ?spouseFN
WHERE {
?person vCard:FN ?fullName ;
OPTIONAL { ?person ex:marriedTo ?spouse .
?spouse vCard:FN ?spouseName .}
bind(if(bound ?spouseName),
 ?spouseName, "Unmarried") as ?spouseFN)
}
```

result:

```
?fullName ?spouseFN
============================
"John Smith" "Mary Smith"
"Mary Smith" "Unmarried"
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

SPARQL Queries: Querying multiple datasets

"Return all people and their work places"

```
PREFIX ex: http://example.org/#
PREFIX vCard: http://www.w3.org/2001/vcard-rdf/3.0#
PREFIX ex-work: <http://example.org/#work/>
```

```
SELECT ?fullName ?workplace WHERE {
?person vCard:FN ?fullName ;
SERVICE http://example.org/sparql {
?person ex-work:worksAt ?workplace .
}
```

result:

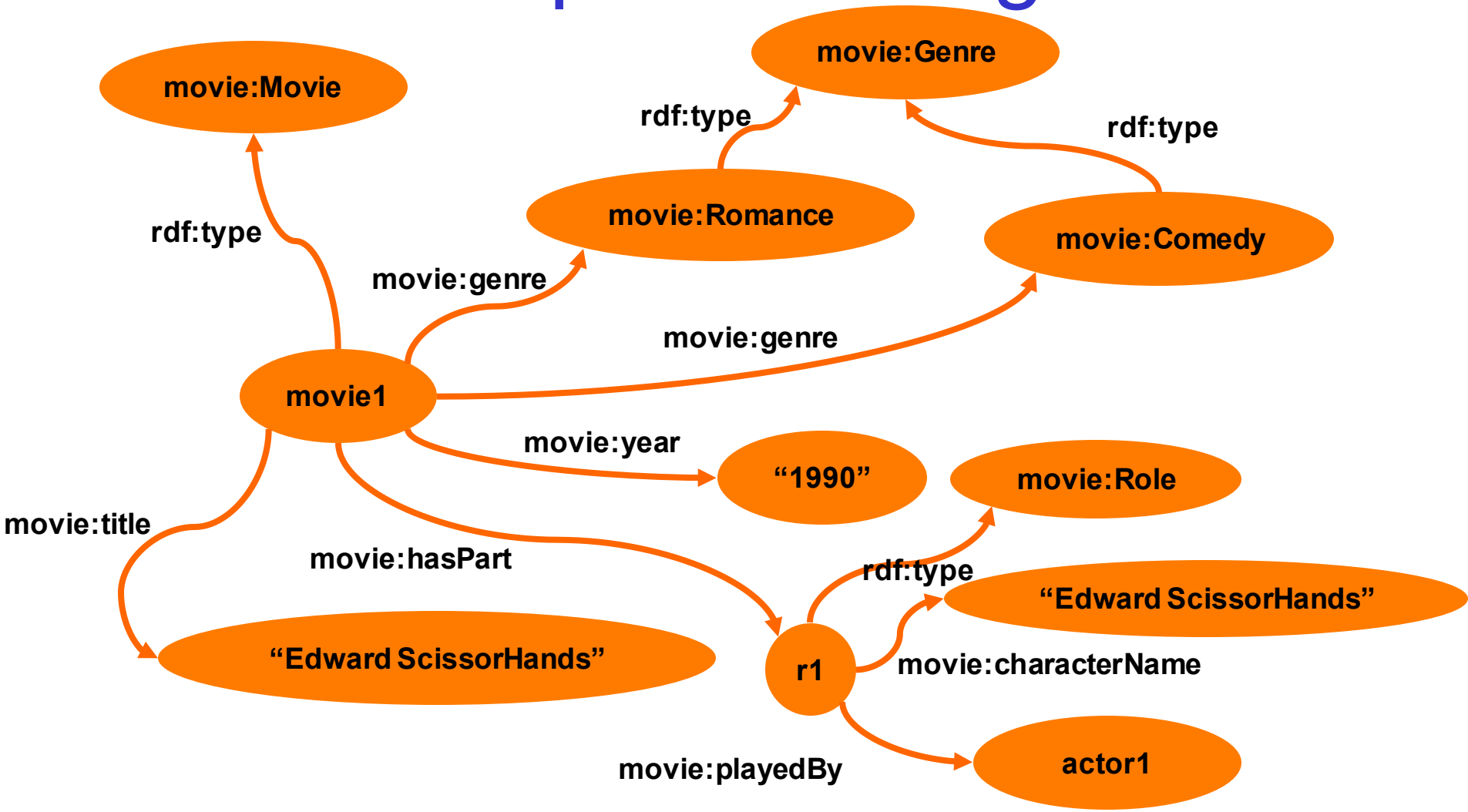
```
?fullName ?workplace
============================
"John Smith" "UiO"
"Mary Smith" " UiO"
```

```
@prefix ex: <http://example.org/#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
ex:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 32 ;
  ex:marriedTo :mary .
ex:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ex:hasAge 29 .
```

Remote endpoint:

```
@prefix ex-work: <http://example.org/#work/> .
ex:john
 ex-work:worksAt "UiO" .
ex:mary
 ex-work:worksAt "UiO" .
```

A RDF Graph Modeling Movies



[http://www.openrdf.org/conferences/eswc2006/Sesame-tutorial-eswc2006.ppt]

Example Query 1

- Select the movies that has a character called "Edward Scissorhands"

```
PREFIX movie: <http://example.org/movies/>

SELECT DISTINCT ?x ?t
WHERE {
                    ?x movie:title ?t ;
                    movie:hasPart ?y .
                    ?y movie:characterName ?z .
                    FILTER (?z = "Edward Scissorhands"@en)
      }
```

Example Query 1

```
PREFIX movie: <http://example.org/movies/>

SELECT DISTINCT ?x ?t
WHERE {
                    ?x movie:title ?t ;
                    movie:hasPart ?y .
                    ?y movie:characterName ?z .
                    FILTER (?z = "Edward Scissorhands"@en)
      }
```

- Note the use of ";" This allows to create triples referring to the previous triple pattern (extended version would be **?x movie:hasPart ?y**)
- Note as well the use of the language speciation in the filter **@en**

Example Query 2

- Create a graph of actors and relate them to the movies they play in (through a new ‘playsInMovie’ relation)

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

CONSTRUCT {
        ?x foaf:firstName ?fname.
        ?x foaf:lastName ?lname.
        ?x movie:playInMovie ?m
        }
WHERE   {
                ?m movie:title ?t ;
                movie:hasPart ?y .
                ?y movie:playedBy ?x .
        ?x foaf:firstName ?fname.
        ?x foaf:lastName ?lname.
}
```

Example Query 3

- Find all movies which share at least one genre with “Gone with the Wind”

```
PREFIX movie: <http://example.org/movies/>

SELECT DISTINCT ?x2 ?t2
WHERE {
                        ?x1 movie:title ?t1.
                        ?x1 movie:genre ?g1.
                        ?x2 movie:genre ?g2.
                        ?x2 movie:title ?t2.
                        FILTER (?t1 = “Gone with the Wind”@en &&
                        ?x1!=?x2 && ?g1=?g2)
      }
```

Demo

- Input data sample (CSV)
- Vocabulary for annotation of the input data
- RDF data generation from the input data
- Data storage in the triplestore (GraphDB)
- Querying data via SPARQL

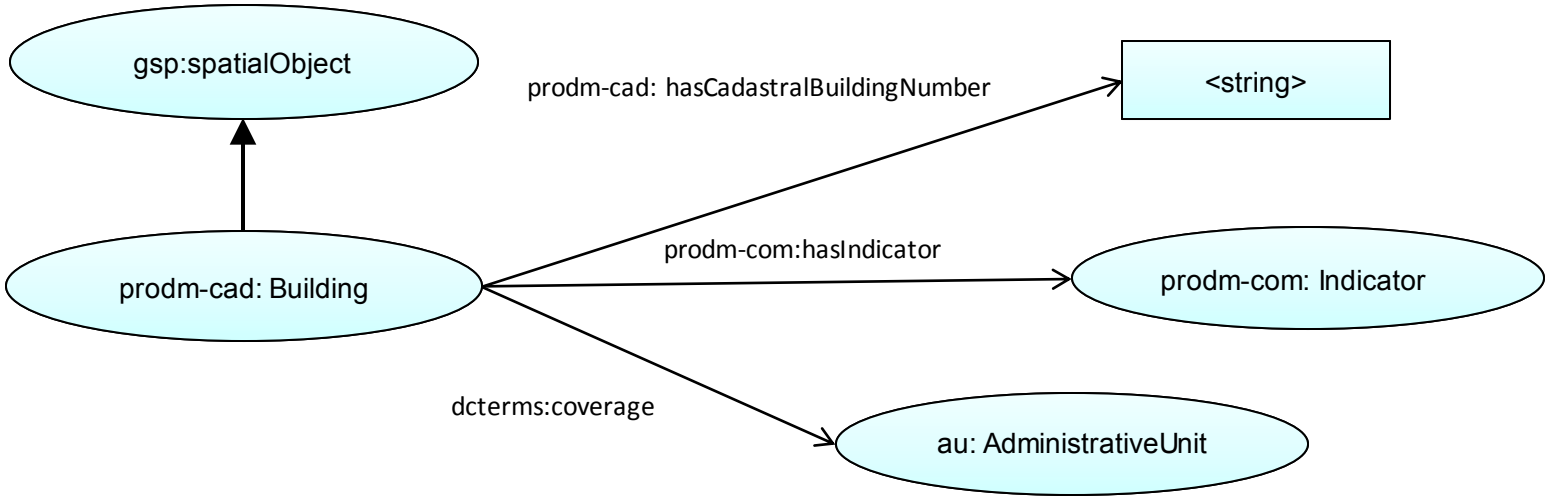
Input data

GABNR	WKT	FLOOD	STORM	ORGNR	NAVN	HOVEDORG	KOMM	KOMM_NAVN
300459280	POINT (7.93059368574362 58.1823924970903)		0.0	971033533	JERNBANEVERKET	972417904	1001	Kristiansand
153461260	POINT (12.114963268343 60.1104550302724)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461279	POINT (12.1147926402154 60.1109352954992)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461287	POINT (12.114640985043 60.1112353987486)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461295	POINT (12.1146160764906 60.1113528004179)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461309	POINT (12.1144927160267 60.1115713641566)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461317	POINT (12.1138165721977 60.1129890672545)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
153461767	POINT (12.116014214123 60.1107178207337)		0.0	971033533	JERNBANEVERKET	972417904	420	Eidskog
12735545	POINT (7.17912087057991 60.7302794862984)		1.3	971033533	JERNBANEVERKET	972417904	1421	Aurland
12735596	POINT (7.20277950026191 60.7174511263063)	1	0.8	971033533	JERNBANEVERKET	972417904	1421	Aurland
177253855	POINT (7.14068695270539 60.7387686654623)		0.8	971033533	JERNBANEVERKET	972417904	1421	Aurland
177253871	POINT (7.14548739646934 60.7337789936212)		0.6	971033533	JERNBANEVERKET	972417904	1421	Aurland
177266973	POINT (7.18541133744704 60.7286494688793)		1.0	971033533	JERNBANEVERKET	972417904	1421	Aurland
159247473	POINT (10.0934745939107 60.2065635945719)		0.0	971033533	JERNBANEVERKET	972417904	605	Ringerike
160147334	POINT (7.77374434346912 60.5352572719904)		0.7	971033533	JERNBANEVERKET	972417904	620	Hol
160147342	POINT (7.77360284097347 60.5350592476609)		0.7	971033533	JERNBANEVERKET	972417904	620	Hol
160489480	POINT (10.0161976017982 59.9857712065448)	1	0.0	971033533	JERNBANEVERKET	972417904	623	Modum
160489499	POINT (10.0165970246855 59.9859387517609)	1	0.0	971033533	JERNBANEVERKET	972417904	623	Modum
185640264	POINT (11.113658316783 63.6666740208529)		0.0	971033533	JERNBANEVERKET	972417904	1719	Levanger
18266687	POINT (10.2061562311985 59.6027785755979)		0.0	971033533	JERNBANEVERKET	972417904	713	Sande
18266695	POINT (10.2079364886206 59.6025196951238)		0.0	971033533	JERNBANEVERKET	972417904	713	Sande
163403439	POINT (10.208768322561 59.6024940628082)		0.0	971033533	JERNBANEVERKET	972417904	713	Sande
163403447	POINT (10.2083966875307 59.6025064713491)		0.0	971033533	JERNBANEVERKET	972417904	713	Sande
13664668	POINT (11.0273611042444 59.2684919939837)	1	0.0	971033533	JERNBANEVERKET	972417904	105	Sarpsborg
146017258	POINT (11.0265763857844 59.2682796412608)	1	0.0	971033533	JERNBANEVERKET	972417904	105	Sarpsborg
146017320	POINT (11.0271512589572 59.2685042457323)	1	0.0	971033533	JERNBANEVERKET	972417904	105	Sarpsborg
146017339	POINT (11.0269158830315 59.2683821496692)	1	0.0	971033533	JERNBANEVERKET	972417904	105	Sarpsborg
10964806	POINT (12.028136844097 64.4845344752087)		0.0	971033533	JERNBANEVERKET	972417904	1744	Overhalla
186581644	POINT (12.0293246601821 64.4832256842343)		0.0	971033533	JERNBANEVERKET	972417904	1744	Overhalla

Vocabulary

The proDataMarket cadaster vocabulary

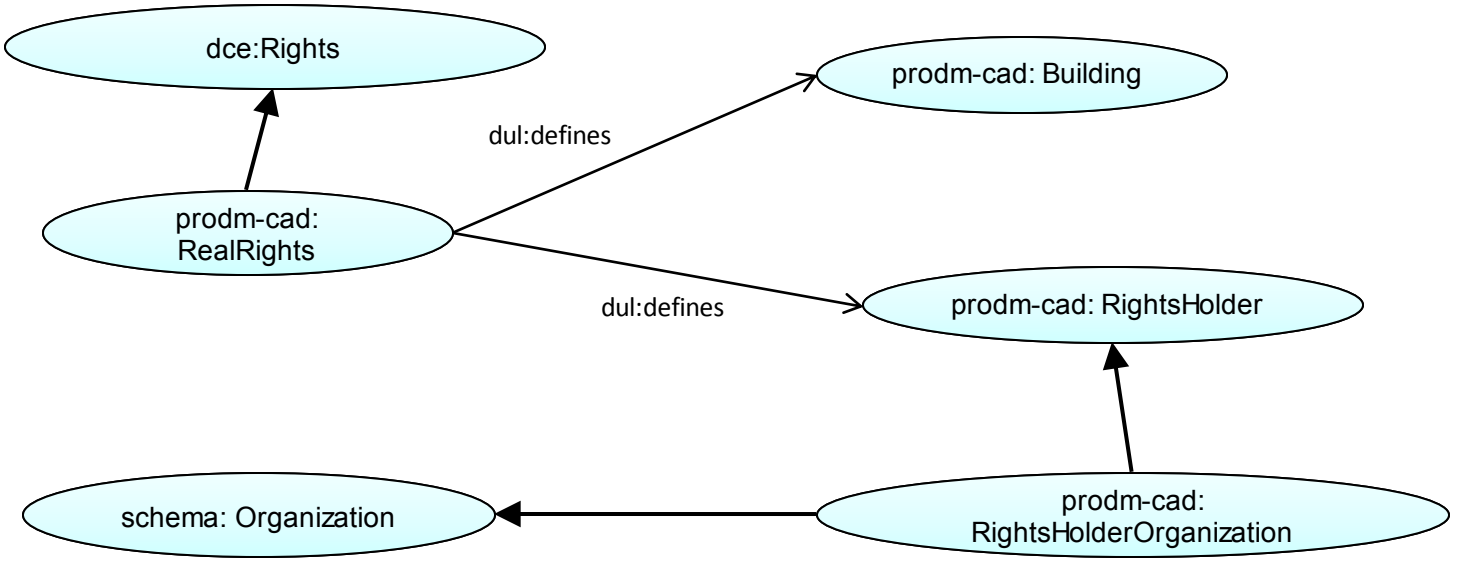
@prefix gsp: http://www.opengis.net/ont/geosparql#
@prefix dcterms: http://purl.org/dc/terms/
@prefix dbpedia-owl: http://dbpedia.org/ontology/
@prefix prodm-cad: http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#
@prefix prodm-com: http://vocabs.datagraft.net/proDataMarket/0.1/Common#



Vocabulary (cont')

The proDataMarket cadaster vocabulary

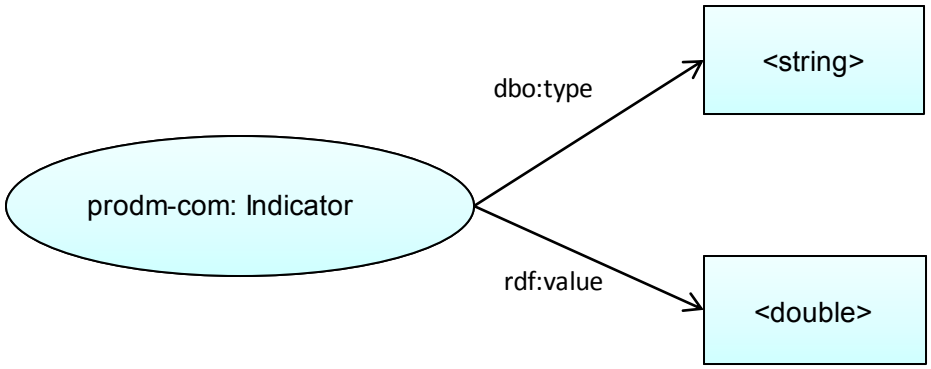
@prefix dul: http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#
@prefix dce: http://purl.org/dc/elements/1.1/
@prefix prodm-cad: http://vocabs.datagraft.net/proDataMarket/0.1/Cadastre#
@prefix scema: http://schema.org/



Vocabulary (cont')

The proDataMarket common vocabulary

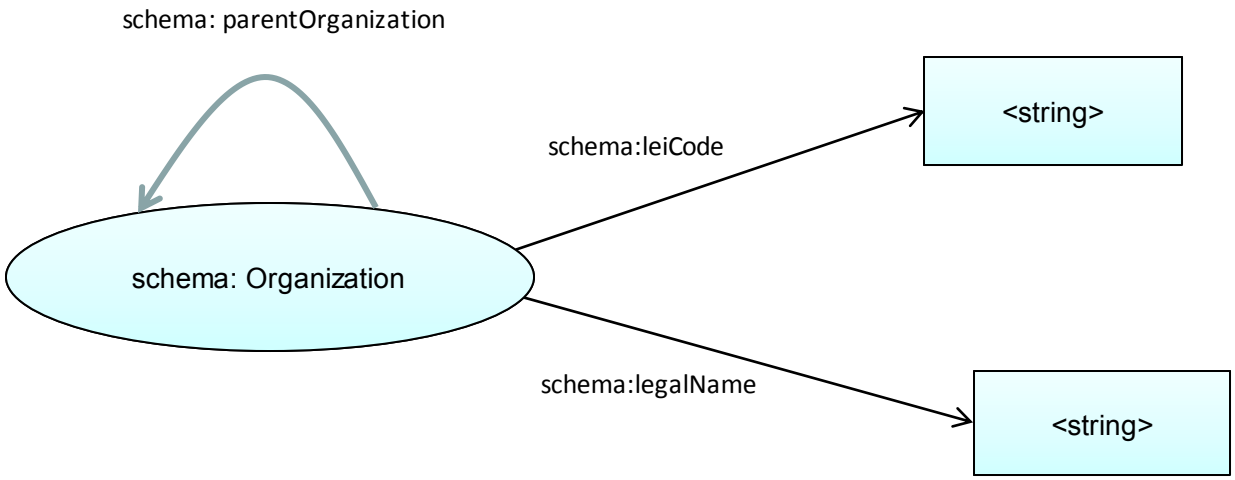
@prefix dbo: http://dbpedia.org/ontology/
@prefix prodm-com: http://vocabs.datagraft.net/proDataMarket/0.1/Common#



Vocabulary (cont')

The schema organization vocabulary

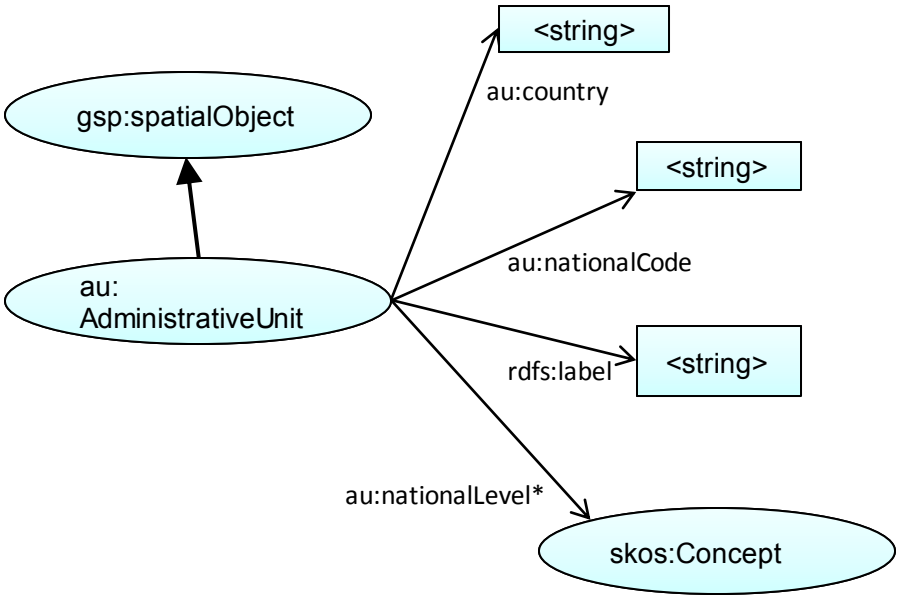
@prefix schema :http://schema.org/



Vocabulary (cont')

The administrative units vocabulary

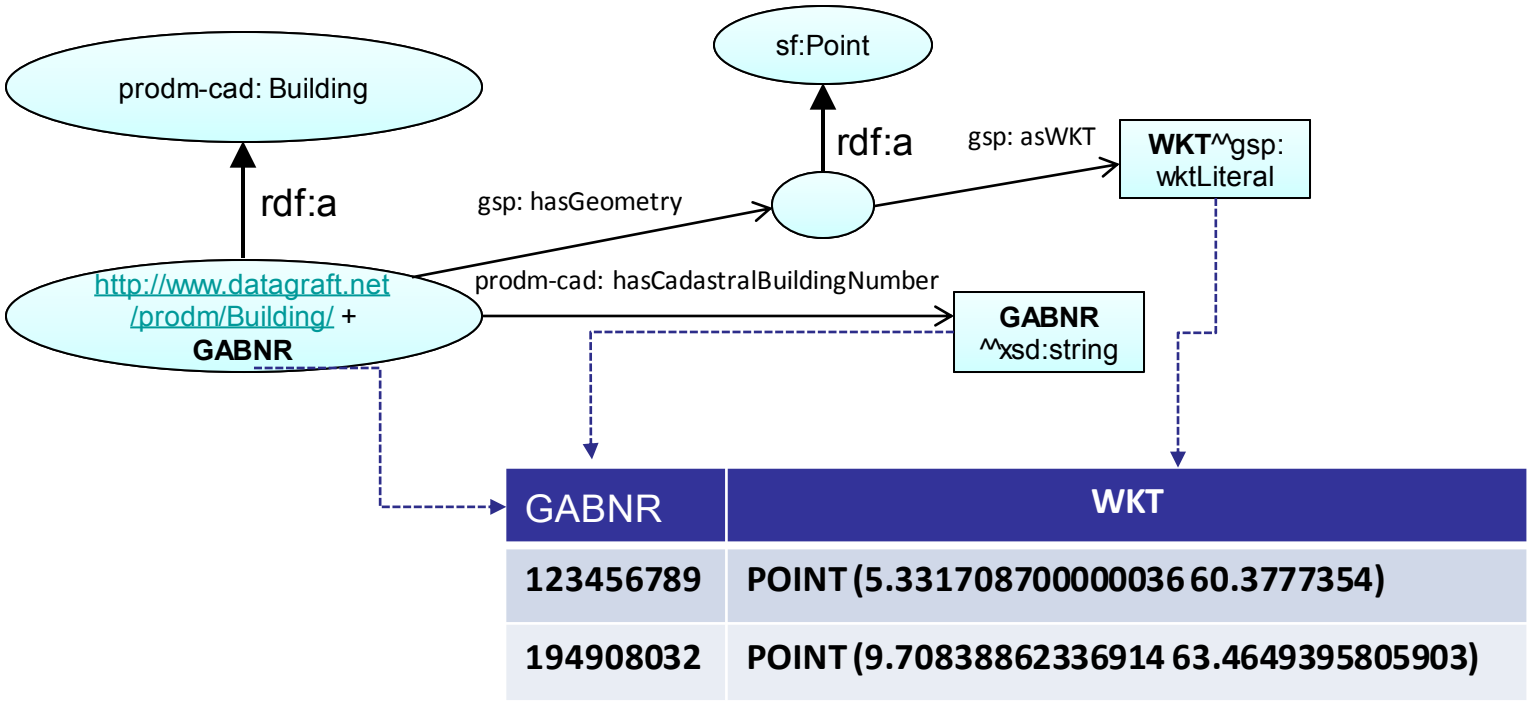
@prefix au: http://www.w3.org/2015/03/inspire/au#
@prefix gsp: http://www.opengis.net/ont/geosparql#
@prefix skos: http://www.w3.org/2008/05/skos#



*au:nationalLevel property links the Administrative Unit to the level in the national administrative hierarchy, at which the Unit is established. The property takes values from the INSPIRE SKOS concept scheme for the Administrative_Hierarchy Level.
National levels for the administrative units in different countries can be found here: https://en.wikipedia.org/wiki/List_of_administrative_divisions_by_country.
Municipalities in Norway have national level 3

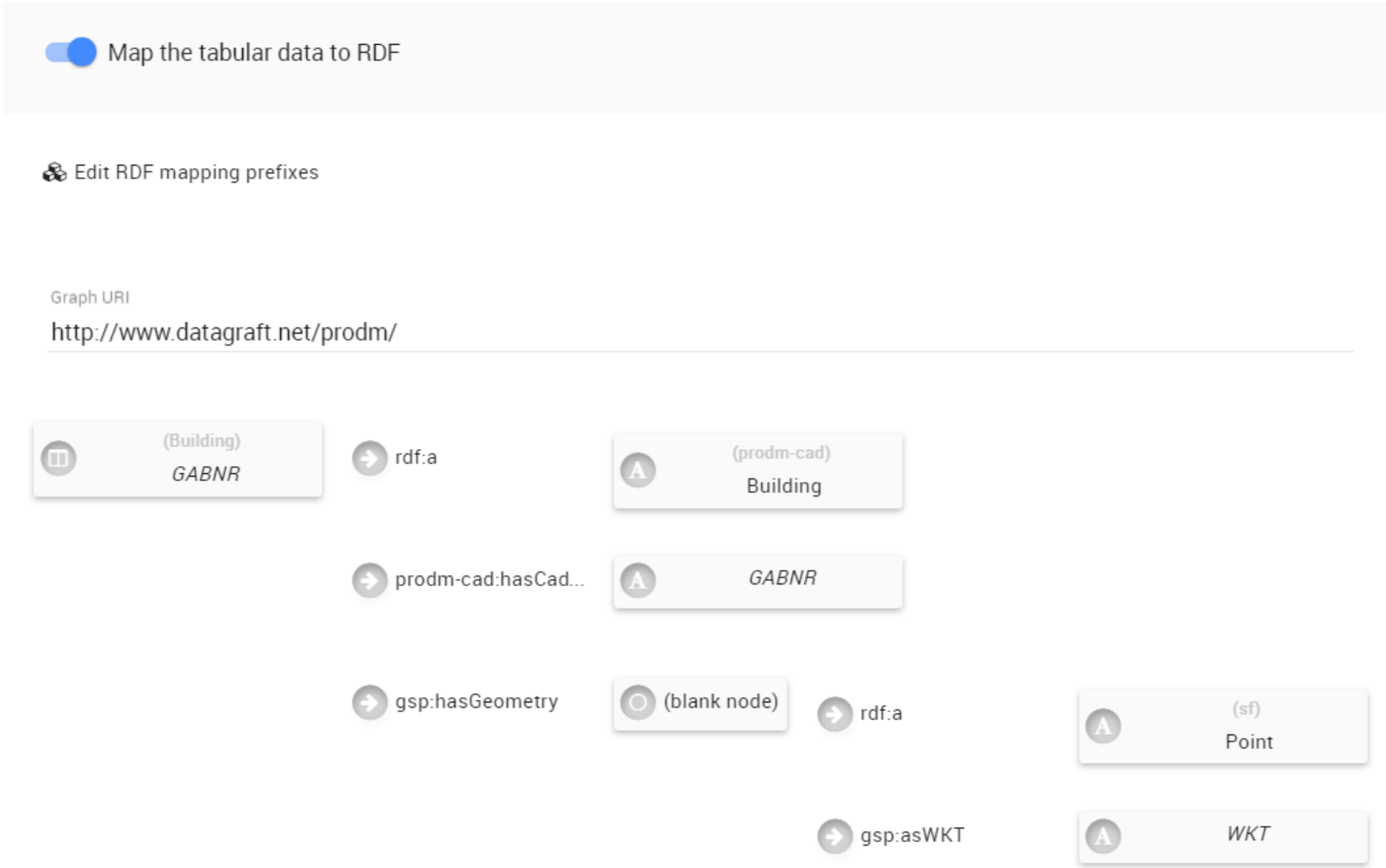
RDF mapping

- Map column names to elements in the ontology
- RDF is generated according to the mapping for each row

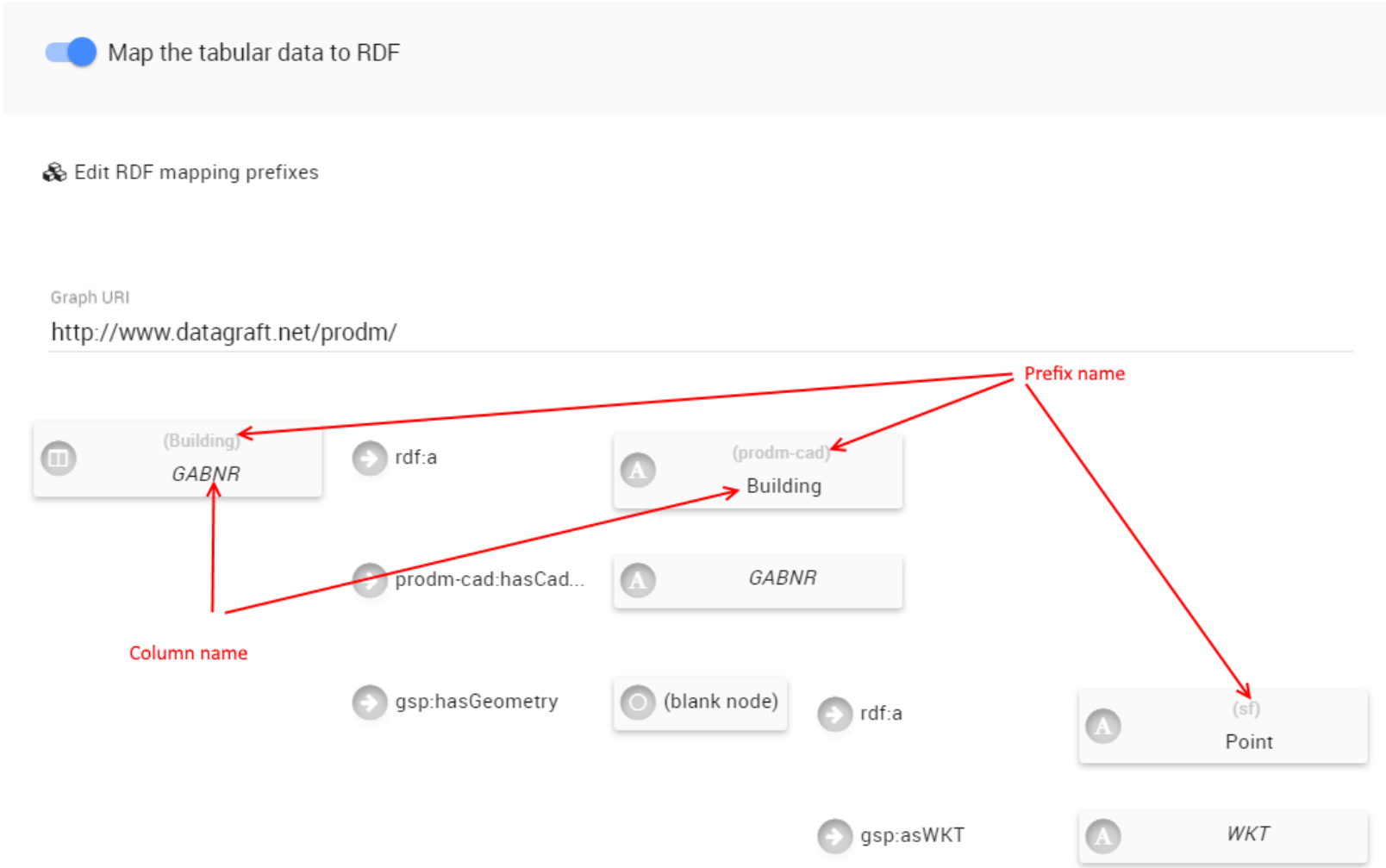


GABNR	WKT
123456789	POINT (5.3317087000003660.3777354)
194908032	POINT (9.70838862336914 63.4649395805903)

RDF mapping (cont')



RDF mapping (cont')



Query #1: Retrieve coordinates of state-owned buildings having flood risk

https://datagraft.io/demo_publisher/queries/demo_query1-buildings-with-flood-risk

- Each building has indicator "FloodRisk" having values within [0.0, 1.0] that marks flood risk or no flood risk on building

```
SELECT DISTINCT ?coords ?floodRisk
WHERE {
?bygg a prodm-cad:Building ;
            gsp:hasGeometry [gsp:asWKT ?coords ;].
OPTIONAL {
?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
                              dbo:type "FloodRisk" ;
                              rdf:value ?flood ;]; }
bind(if(bound(?flood), ?flood, "0.0"^^xsd:double) as
                              ?floodRisk)
}
```

Query #2: Retrieve coordinates of state-owned buildings having storm risk

https://datagraft.io/demo_publisher/queries/demo_query2-buildings-with-storm-risk

- Each building has indicator "StormRisk" having values within [0.0 … 1.0] that marks storm risk on building

```
SELECT DISTINCT ?coords ?stormRisk
WHERE {
?bygg a prodm-cad:Building ;
            gsp:hasGeometry [gsp:asWKT ?coords ;].
OPTIONAL {
?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
                              dbo:type "StormRisk" ;
                              rdf:value ?storm ;]; }
bind(if(bound(?storm), ?storm, "0.0"^^xsd:double) as
                              ?stormRisk)
}
```

Query #3: Retrieve the total risk value for the state-owned buildings (0 risks, 1 risk or 2 risks)

https://datagraft.io/demo_publisher/queries/demo_query3-the-total-risk-value-for-the-state-owned-buildings

```
SELECT DISTINCT ?coords ?risk
WHERE {
?bygg a prodm-cad:Building ;
          gsp:hasGeometry [gsp:asWKT ?coords ;].
OPTIONAL {
?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
                                  dbo:type "StormRisk" ;
                                  rdf:value ?storm ;]; }
OPTIONAL {
?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
                                  dbo:type "FloodRisk" ;
                                  rdf:value ?flood ;]; }
bind(if(bound(?storm), if
(?storm>0,"1"^^xsd:integer,"0"^^xsd:integer),"0"^^xsd:integer) as
?stormRisk )
bind(if(?stormRisk > 0 && bound(?flood), "2", if(?stormRisk > 0 ||
bound(?flood), "1", "0")) as ?risk)
}
```

Query #4: Retrieve the exposure index for flood for each municipality

https://datagraft.io/demo_publisher/queries/demo_query4-show-exsposure-index-flood-per-each-municipality

The exposure index is calculated as the amount of state owned buildings with flood risk against the total amount of state owned buildings in each municipality:

- EIF = (SUM*B*inHZF/SUM_B) *100
  - SUM*B*inHZF -- amount of state owned buildings in Hazard Zone Flood
  - SUM_B -- total amount of state owned buildings

```
SELECT ?name ?code (?underFloodRisk/?totalAmount*100 as ?eif) WHERE {
SELECT DISTINCT ?code ?name (count(?bygg) as ?totalAmount) (sum(?floodRisk)
as ?underFloodRisk) WHERE
{
?bygg a prodm-cad:Building ;
            dcterms:coverage ?au .
OPTIONAL {?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
dbo:type "FloodRisk" ;
rdf:value ?flood ;]; }

bind(if(bound(?flood), "1"^^xsd:integer, "0"^^xsd:integer) as ?floodRisk)

?au a au:AdministrativeUnit;
        au:nationalCode ?code;
        rdfs:label ?name ;
}
GROUP BY ?code ?name
}
```

Query #5: Retrieve the exposure index for storm for each municipality

https://datagraft.io/demo_publisher/queries/demo_query5-show-exsposure-index-storm-per-each-municipality

The exposure index for storm is calculated similarly to the exposing index for flood (see previous slide)

```
SELECT ?name ?code (?underStormRisk/?totalAmount*100 as ?eif) WHERE {
SELECT DISTINCT ?code ?name (count(?bygg) as ?totalAmount) (sum(?stormRisk)
as ?underStormRisk) WHERE
{
?bygg a prodm-cad:Building ;
            dcterms:coverage ?au .
OPTIONAL {
?bygg prodm-com:hasIndicator [a prodm-com:Indicator ;
                                    dbo:type "StormRisk" ;
                                    rdf:value ?storm;]; }

bind(if(bound(?storm), if
(?storm>0,"1"^^xsd:integer,"0"^^xsd:integer),"0"^^xsd:integer) as ?stormRisk
)


?au a au:AdministrativeUnit;
        au:nationalCode ?code;
        rdfs:label ?name ;
}
GROUP BY ?code ?name
}
```

Query #6: Retrieve the combined exposure index for storm and flood for each municipality

The Combined exposure index for storm and flood is calculated with weights reflecting the potential damage from storm (0.64) and (flood 0.28).

- Find the highest and lowest values from the two risk results
  - EIF100=((EIF - minimum)/(maximum-minimum))100
  - EIS100=((EIS - minimum)/(maximum-minimum))100
- The exposure index (EI) is derived from the former values:
  - EI = (0.64 * |EIS100|) + (0.28 * |EIF100|)

Query #6: Retrieve the combined exposure index for storm and flood for each municipality (cont')

https://datagraft.io/demo_publisher/queries/demo_query6-show-exsposure-index-to-risk-for-both-storm-or-flood

```
SELECT ?name ?code ?impact WHERE
{{
SELECT (min(?eif) as ?minEIF) (max(?eif) as ?maxEIF) (min(?eis) as
?minEIS) (max(?eis) as ?maxEIS) WHERE {
[...Select EIF and EIS per municipality...]
}}
{
[...Select EIF and EIS per municipality...]
}
bind(((?eif -?minEIF)/(?maxEIF-?minEIF))*100 as ?eif100)
bind(((?eis -?minEIS)/(?maxEIS-?minEIS))*100 as ?eis100)
bind((0.64 * ABS (?eis100)) + (0.28 * ABS (?eif100)) as ?impact)
}
```

Query #7: Retrieve the number of state owned buildings for each owner per municipality (group by ministry and owner)

https://datagraft.io/demo_publisher/queries/demo_query7-show-the-amount-of-state-owned-buildings-for-each-owner-per-municipality-group-by-ministry-and-owner

```
…
WHERE {
?bygg a prodm-cad:Building ;
dcterms:coverage ?au.
?rr a prodm-cad:RealRights;
dul:defines ?bygg;
dul:defines ?org.
?org a prodm-cad:RightsHolderOrganization;
schema:legalName ?owner ;
schema:parentOrganization* ?parent .
?parent a prodm-cad:RightsHolderOrganization;
schema:legalName ?min .
…
FILTER NOT EXISTS {?parent schema:parentOrganization ?parentLast }
}
…
```

Query #8: Calculate buildings-to-population ratio for each municipality

https://datagraft.io/demo_publisher/queries/demo_query8-calculate-buildings-to-population-ratio-for-each-municipality

```
…
WHERE {?au a au:AdministrativeUnit ;
                        au:nationalCode ?code ;
                        rdfs:label ?navn .

SERVICE SILENT <http://dbpedia.org/sparql> {
SELECT ?muni ?navn1 WHERE {
?muni a dbo:PopulatedPlace ;
a yago:WikicatMunicipalitiesOfNorway ;
dbp:name ?navn1 ;
dbo:abstract ?about .

}}
FILTER (str(?navn1) = str(?navn))
…
```

How to run the queries

- Use a tool such as the Sesame Windows Client
  - https://sourceforge.net/projects/sesamewinclient/
- Connection:
  - Database: https://rdf.datagraft.net/4036477454/db
  - Repository: state-owned-buildings-risk
  - Username:Password: s4c5796ik8k4:fvhsmsk57h7g1ce