# SPARQL QUERY LANGUAGE

**JELENA JOVANOVIC**

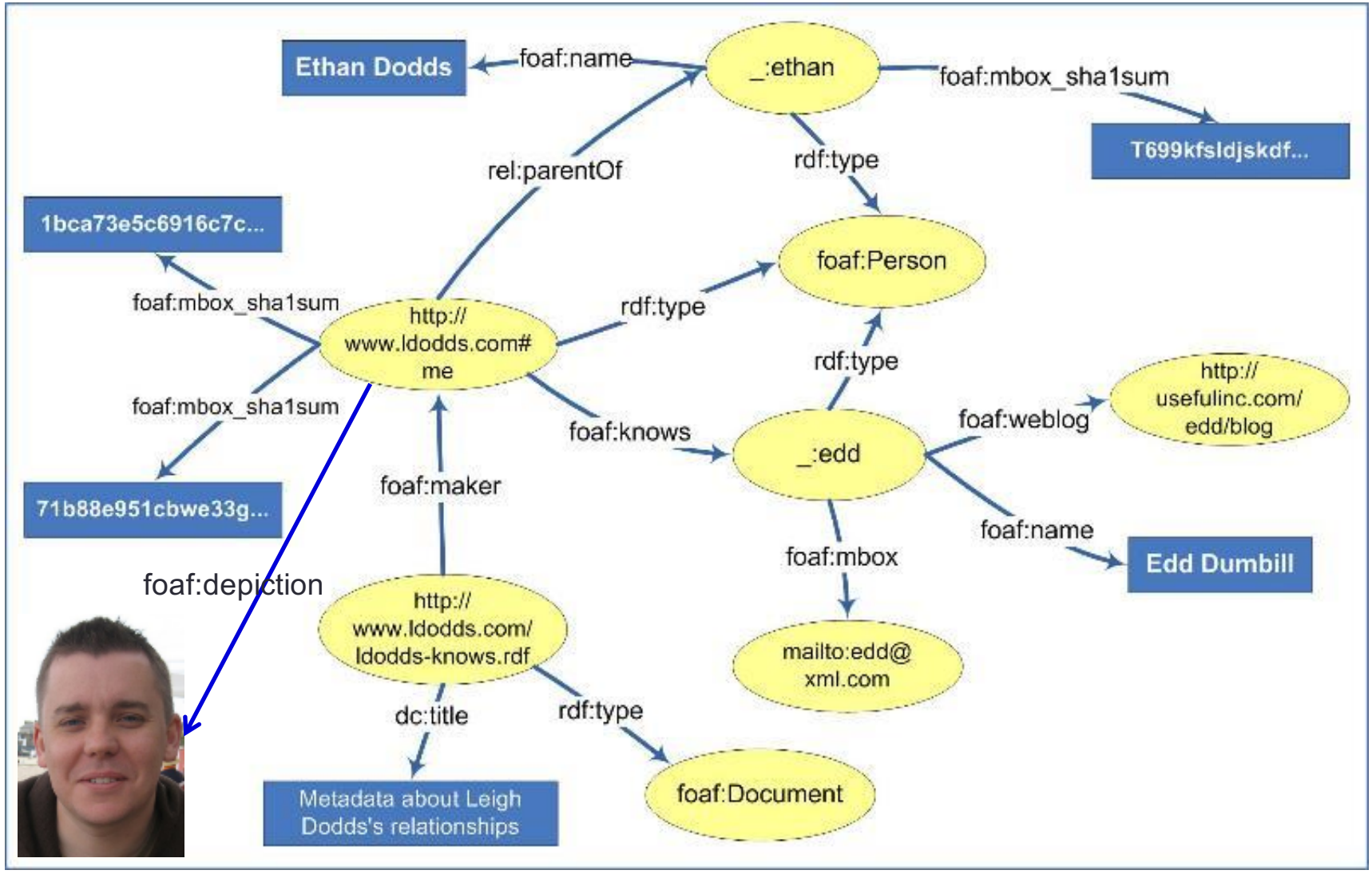
EMAIL: JELJOV@GMAIL.COM
WEB: HTTP://JELENAJOVANOVIC.NET

# SPARQL query language

- W3C standard for querying RDF graphs
- Can be used to query not only native RDF data, but also any data that can be mapped to RDF
- This mapping could be done by making use of
  - (W3C) standard mapping languages such as R2RML that allow for transforming relational data to RDF
  - Various mapping tools such as those listed at:

    http://www.w3.org/wiki/ConverterToRdf

# Let's start with an example

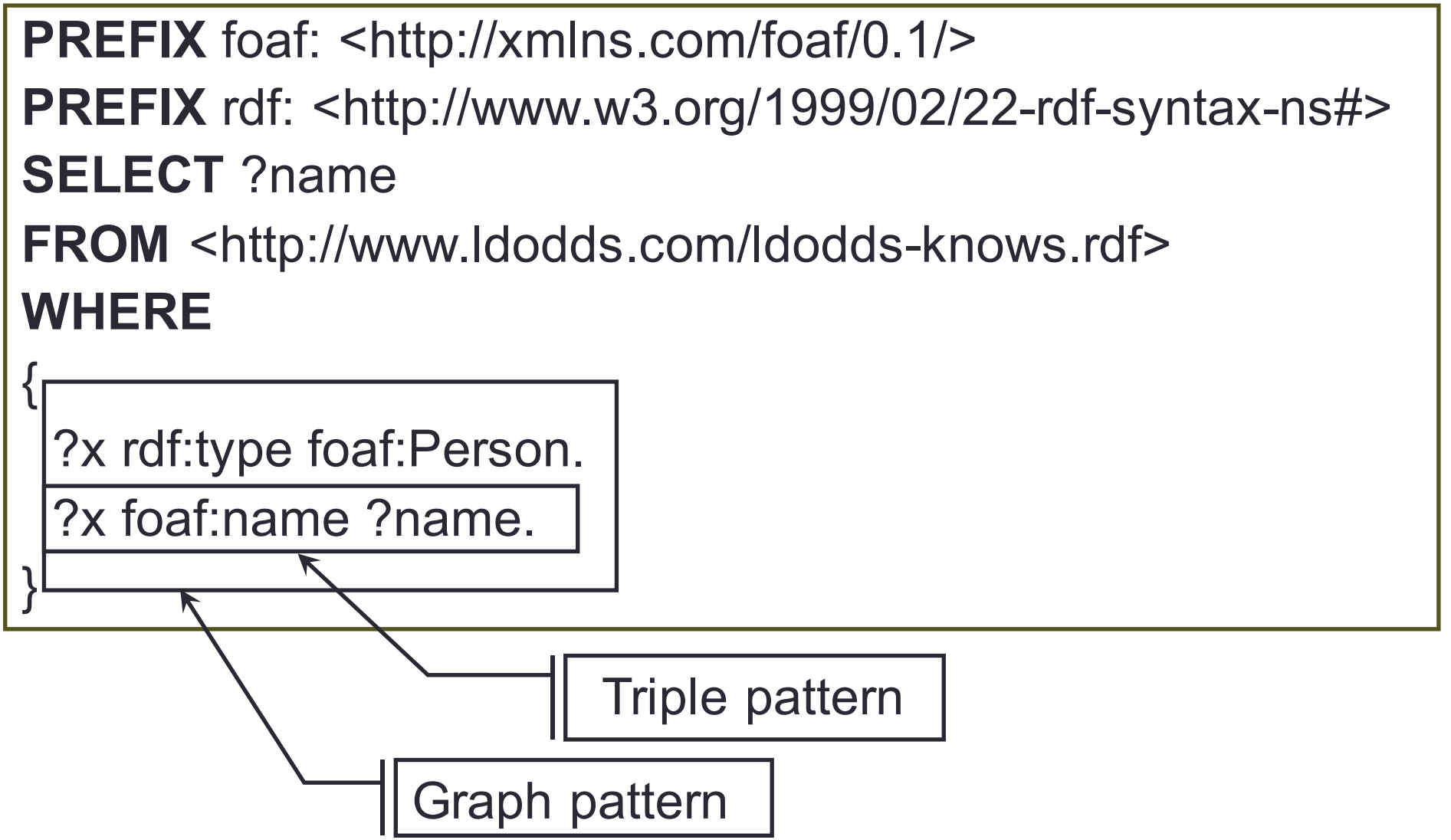



Graphical representation of a small segment of the RDF graph stored in:

http://www.ldodds.com/ldodds-knows.rdf

# Task 1: Find names of all mentioned persons

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
  ?x rdf:type foaf:Person.
  ?x foaf:name ?name.
}
```



# The basic structure of a SPARQL query

- PREFIX
  - the SPARQL equivalent of declaring an XML namespace
- SELECT
  - like its twin in an SQL query, it is used to define the data items that will be returned by the query
- FROM
  - identifies the data against which the query will be run
  - can be given in runtime as well
- WHERE
  - defines the part of RDF graph we are interested in

# Some notes about the SPARQL syntax

- Variables are prefixed with either "? " or "$“
  - these two are interchangeable
- Blank nodes are indicated by:
  - the label form, such as "_:abc", or
  - the abbreviated form "[]"
- Dots (.) separate triple patterns
- Semi column (;) separates triple patterns with the common subject

# About graph patterns

- In SPARQL, one cannot SELECT a variable if it is not listed in the graph pattern (i.e., in the WHERE clause).

- **Important**:

  SPARQL query processor

  has NO data dictionary or schema

  that lists types and properties of resources

  The only schema it has is the graph pattern

  (i.e., the WHERE part of the query)

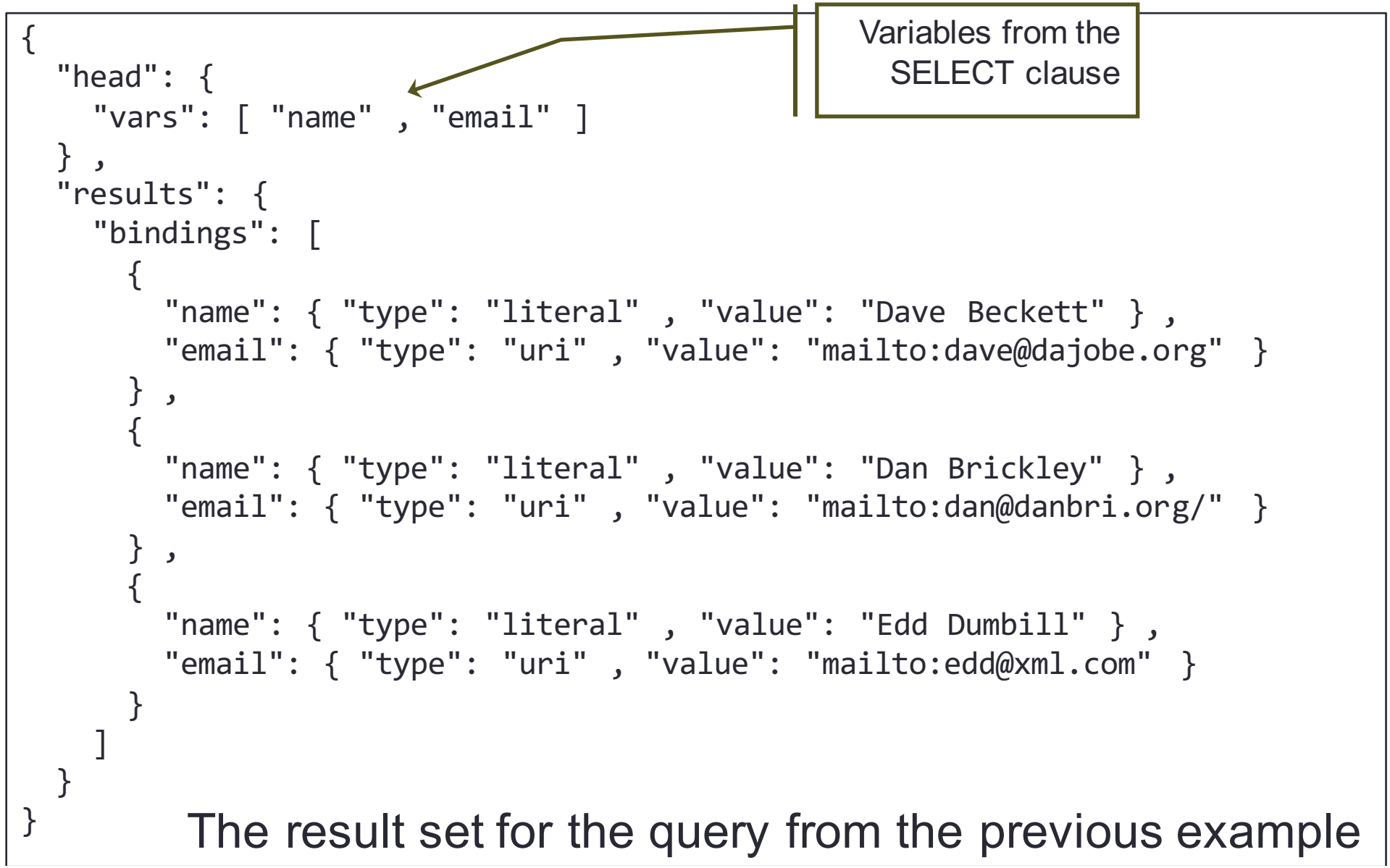
# About graph patterns

- Graph pattern is a collection of triple patterns
- It defines the shape of the (RDF) graph we want to match against
- Within one graph pattern each variable must have the same value

# Task 2: Find names and emails of the persons whom the author of the document knows

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?email
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE
{
   ?doc rdf:type foaf:Document ;
            foaf:maker ?author .
   ?author foaf:knows ?someone .
   ?someone foaf:name ?name ; foaf:mbox ?email .
 }
```

# Result of the SELECT query (in JSON syntax)




```
{
  "head": {
    "vars": [ "name" , "email" ]
  } ,
  "results": {
    "bindings": [
      {
        "name": { "type": "literal" , "value": "Dave Beckett" } ,
        "email": { "type": "uri" , "value": "mailto:dave@dajobe.org" }
      } ,
      {
        "name": { "type": "literal" , "value": "Dan Brickley" } ,
        "email": { "type": "uri" , "value": "mailto:dan@danbri.org/" }
      } ,
      {
        "name": { "type": "literal" , "value": "Edd Dumbill" } ,
        "email": { "type": "uri" , "value": "mailto:edd@xml.com" }
      }
    ]
  }
}
```

The result set for the query from the previous example

# Optional Matching

- RDF often represents *semi-structured* data
  - this means that two resources of the same type may have different sets of properties
  - For instance,
    - a FOAF description of a person may consist only of an e-mail address;
    - alternatively, it can incorporate a real name, twitter nickname, URL of the photo depicting him/her, etc.
- SPARQL's mechanism for *optional matching* allows for handling this heterogeneity

# Task 3: Find names of all persons that the author of the document knows, as well as their blogs *if they have any*

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?blog
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
   ?doc rdf:type foaf:Document; foaf:maker ?author .
   ?author foaf:knows ?person.
   ?person foaf:name ?name .
   OPTIONAL { ?person foaf:weblog ?blog. }
 }
```

# The OPTIONAL block

- If a query has multiple optional blocks
  - these act independently of one another
  - each block may be omitted from, or present in, a solution.
- Optional blocks can also be nested
  - the inner optional block is considered only when the outer optional block's pattern matches the graph.

# Task 4: Find names of all persons that the author of the document knows as well as their blogs and emails, *if these are available*

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?email ?blog
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
   ?doc rdf:type foaf:Document; foaf:maker ?author .
   ?author foaf:knows ?person.
   ?person foaf:name ?name .
   OPTIONAL { ?person foaf:mbox_sha1sum ?email. }
   OPTIONAL { ?person foaf:weblog ?blog . }
 }
```

# Alternative Matching

- Let's suppose that …
  - *foaf:knows* and *rel:hasMet* properties are used to represent somewhat similar information
  - we are interested in all persons that the author of the document either knows or has (ever) met
- In situations like this, you can use SPARQL's *alternative matching* feature to return whichever of the properties is available

# Task 5: Find names of all persons that the author of the document either has met or knows

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rel: <http://purl.org/vocab/relationship/>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  ?doc foaf:maker ?author .
  { ?author foaf:knows  ?someone . }
                    UNION
   {?author rel:hasMet    ?someone . }
   ?someone  foaf:name ?name .
}
```

# UNION

- In contrast with OPTIONAL graph patterns, in the case of UNION *at least one* of the alternatives must be matched by any query solution;
- If both branches of the UNION match, two solutions will be generated.

# DISTINCT

- In the result set of the previous task some names appeared twice
- By adding the DISTINCT keyword in the SELECT clause, we exclude multiple appearance of the same values from the result set
  - Just like in SQL

# Task 5a: Find names of all the persons that the author of the document either has met or knows, but without name repetition

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rel: <http://purl.org/vocab/relationship/>

SELECT DISTINCT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
   ?doc foaf:maker ?author .
   { ?author foaf:knows  ?someone . }
                        UNION
    {?author rel:hasMet    ?someone . }
    ?someone  foaf:name ?name .
 }
```

# The ORDER BY clause

- Indicates that the result set should be ordered by the specified variable
- It can list one or more variable names, indicating the variables that should be used to order the result set
- By default all sorting is done in the ascending order
  - this can be explicitly changed using the DESC (descending) and ASC (ascending) functions

# Task 5b: Find names of all persons that the author of the document has either met or knows; sort the names in descending order

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rel: <http://purl.org/vocab/relationship/>

SELECT DISTINCT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
   ?doc foaf:maker ?author .
    { ?author foaf:knows  ?someone . }
                        UNION
    {?author rel:hasMet    ?someone . }
    ?someone  foaf:name ?name .
 }
ORDER BY DESC (?name)
```

# SPARQL FILTERs

- SPARQL FILTERs restrict the solutions of a graph pattern match according to the given expressions
- Expressions can be of different kinds, but they must evaluate in a boolean value (true or false)
- The following slides illustrate some of the functions that can be used for filtering the result set

# Task 6: Find names of all the persons whose birthday is unknown

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bio: <http://purl.org/vocab/bio/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
FROM <http://www.ldodds.com/ldodds-knows.rdf>
SELECT ?name
WHERE {
  ?person rdf:type foaf:Person ; foaf:name ?name .
   FILTER NOT EXISTS {
            ?person bio:event ?event .
            ?event rdf:type bio:Birth ; bio:date ?birthdate. }
}
```

Note: Function NOT EXISTS is introduced in SPARQL 1.1; if the query does not work, it means you are using an old SPARQL engine

# Task 7: Find names of all members of the Dodds family

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  ?person foaf:name ?name
  FILTER regex(?name, "dodds", "i")
}
```

Filtering with regular expressions
Similar to SQL "LIKE"

Alternative:
FILTER strEnds(lcase(?name), “dodds”)

# Task 7a: Find names of all the persons who have Gmail email address

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  ?person foaf:name ?name ; foaf:mbox ?mbox
  FILTER regex( str(?mbox), "@gmail\\.com$" )
}
```

To learn more about the regular expression language check this tutorial: http://regex.bastardsbook.com/

# Task 8: Get all reviews with rating above 6 that were created by a person named Jim (filtering based on elements values)

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rev: <http://www.purl.org/stuff/rev#>

SELECT ?review
FROM  <http://www.cs.umd.edu/~hendler/2003/foaf.rdf>
WHERE {
     ?someone rdf:type foaf:Person;
          foaf:name ?name FILTER regex(?name, "Jim", "i").
     ?someone foaf:made ?review .
     ?review rev:rating ?rating
               FILTER (xsd:int(?rating) >= 6) .
  }
```

SPARQL type casting

# Grouping and aggregating data

- GROUP BY allows for grouping the items in the result set based on one or more variables and/or expressions
- Having grouped the results, we can apply various functions at the group level: SUM, COUNT, AVG, MIN, MAX and the like
- We can also use HAVING clause to select / filter the query results at the group level
  - it is analogous to a FILTER expression, but operates over groups, rather than individual solutions

# Task 9: Find manufacturers who produce more than 10 different products and display the number of different products they produce

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX schema: <http://schema.org/>

SELECT ?manufacturer (COUNT(?product) AS ?count)
WHERE {
   ?product rdf:type schema:Product ;
                dbo:manufacturer ?manufacturer .
}
GROUP BY ?manufacturer
HAVING (COUNT(?product) > 10)
ORDER BY  ?count
```

# Other kinds of SPARQL queries

Besides the SELECT queries,

SPARQL supports three other query types:

- ASK
- DESCRIBE
- CONSTRUCT

# ASK query

- Aimed at testing whether a query pattern has a solution
- No information is returned about the possible query solutions, just whether a solution exists
- An example: have Natalie Portman and Scarlett Johansson ever played in the same movie?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
ASK  {
    ?movie
            dbo:starring dbr:Natalie_Portman ;
            dbo:starring dbr:Scarlett_Johansson .
}
```

# ASK query

Results of an ASK query:

- Possible values: true/false
- JSON format of the results:

```
{
   "head" : { } ,
   "boolean" : true
}
```

# DESCRIBE query

- **Returns a graph** comprising all the available triplets about the resource matched in the graph pattern (that is, in the WHERE part of the query)
- Example:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbr: <http://dbpedia.org/resource/>
DESCRIBE ?movie
WHERE {
    ?movie dbo:starring dbr:Natalie_Portman ;
                dbo:starring dbr:Scarlett_Johansson .
 }
```

The query returns a graph comprising all the available triplets about the movie(s) starred by both actresses

# CONSTRUCT query

- It is used for creating a new RDF graph from an existing one
- It is for RDF graph

  somewhat the same as XSLT for XML data

# Task 10: Map the data about musicians' date and place of birth from DBpedia to Schema.org vocabulary

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX schema: <http://schema.org/>
CONSTRUCT {
        ?someone a schema:Person ;
                schema:birthPlace ?birthplace ;
                schema:birthDate ?birthdate ;
                schema:jobTitle "Musician".
} WHERE {
   ?someone  a  dbo:MusicalArtist ;
        dbo:birthDate ?birthdate ;
        dbo:birthPlace ?birthplace .
 }
```

abbreviated form for rdf:type

# Task 11: Establish aunt relationship

```
PREFIX schema: <http://schema.org/>
PREFIX rel: <http://purl.org/vocab/relationship/>
CONSTRUCT {
        ?child rel:hasAunt ?aunt.
} WHERE {
        ?child  schema:parent  ?parent .
        ?parent schema:parent ?grandparent .
        ?aunt   schema:parent  ?grandparent ;
                schema:gender  ?gender
        FILTER (?parent != ?aunt && regex(?gender, “female”, “i”)) .
}
```

# Queries over multiple distributed data sources

- All the queries we've seen so far were executed over data originating from one data source (one RDF graph)
- However, queries can be executed over multiple data sources
  - In that case, we talk about *federated queries*
  - SPARQL 1.1 introduces the SERVICE keyword for defining additional data sources

# Task 12: Find all the acquaintances of Leigh Dodds who have the same surname as well known scientists

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX db: <http://dbpedia.org/ontology/>
SELECT ?person
FROM <http://www.ldodds.com/ldodds-knows.rdf>
WHERE {
  <http://www.ldodds.com#me> foaf:knows ?person .
   ?person foaf:surname ?surname .
    SERVICE <http://dbpedia.org/sparql> {
            ?someone  a  db:Scientist ;
                        foaf:surname ?surname .
   }
}
```

Unique identifier (IRI) for Leigh Dodds as given in the used data source (see FROM)

# Learn SPARQL through examples

- Search RDF data with SPARQL
  - https://www.ibm.com/developerworks/library/j-sparql/
- SPARQL by Example
  - http://www.cambridgesemantics.com/semantic-university/sparql-by-example
- A detailed SPARQL tutorial
  - http://www.w3.org/2004/Talks/17Dec-sparql/
- Bring existing data to the Semantic Web
  - http://www.ibm.com/developerworks/library/x-semweb/

# Learn SPARQL through examples

- RDF as self-describing data
  - http://goo.gl/Gdr5LG
- SPARQL at the movies
  - http://www.snee.com/bobdc.blog/2008/11/sparql-at-the-movies.html
- Bart (Simpson) blackboard queries
  - http://goo.gl/aM9mcd ; http://goo.gl/z9qOlH
- Example SPARQL queries over 10+ different RDF datasets
  - http://openuplabs.tso.co.uk/datasets
- SPARQL queries over Europeana repository
  - http://labs.europeana.eu/api/linked-open-data-sparql-endpoint

# Some handy tools for learning SPARQL

- Flint SPARQL Editor
  - http://openuplabs.tso.co.uk/demos/sparqleditor
- YASGUI – Yet Another SPARQL GUI
  - http://yasgui.laurensrietveld.nl/
- SPARQLer - an online SPARQL query tool
  - http://www.sparql.org/sparql.html
- ARQ, a SPARQL processor for Jena framework
  - http://jena.sourceforge.net/ARQ/