Lecture 3: Querying RDF data

TIES4520 Semantic Technologies for Developers
Autumn 2023

University of Jyväskylä

Khriyenko Oleksiy

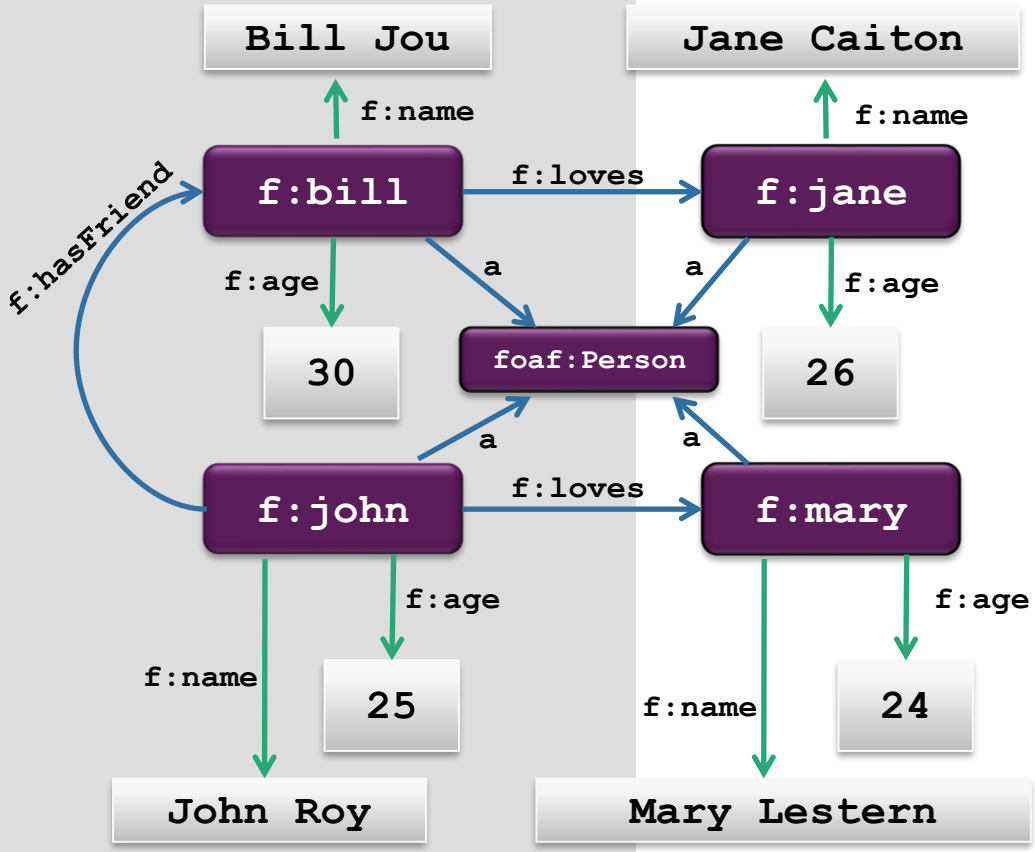
SPARQL: General Form

- SPARQL queries take the following general form

```
PREFIX (Namespace Prefixes)
e.g. PREFIX f: <http://example.org#>

SELECT (Result Set)
e.g. SELECT ?age

FROM and FROM NAMED (Dataset)
e.g. FROM <http://users.jyu.fi/~olkhriye/ties4520/rdf/people.rdf>

WHERE (Query Triple Pattern)
e.g. WHERE { f:mary f:age ?age }

ORDER BY, DISTINCT, HAVING, LIMIT, etc. (Modifiers)
e.g. ORDER BY ?age
```

Example data set

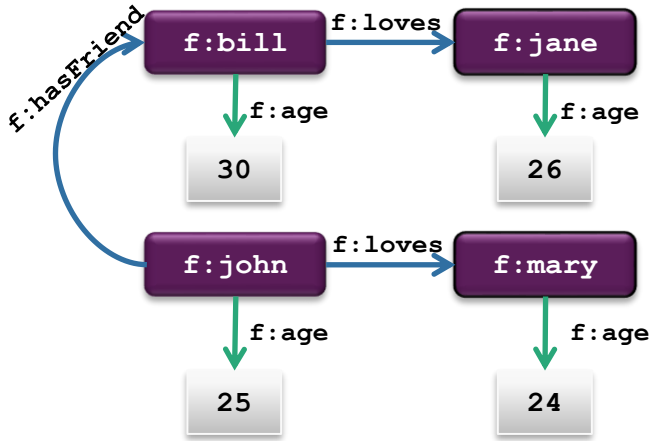
```
@prefix f: <http://example.org#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

f:john a foaf:Person .
f:bill a foaf:Person .
f:mary a foaf:Person .
f:jane a foaf:Person .
f:john f:age "25"^^xsd:int .
f:bill f:age "30"^^xsd:int .
f:mary f:age "24"^^xsd:int .
f:jane f:age "26"^^xsd:int .
f:john f:loves f:mary .
f:bill f:loves f:jane .
f:john f:hasFriend f:bill.
f:john f:name "John Roy" .
f:bill f:name "Bill Jou" .
f:mary f:name "Mary Lestern" .
f:jane f:name "Jane Caiton" .
f:bill foaf:name "Bill" .
f:john foaf:name "John" .
f:mary foaf:name "Mary" .
f:jane foaf:name "Jane" .
```



Simple SPARQL queries (1)

- Show me the property ***f:age*** of resource ***f:mary***

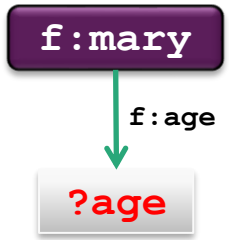
Data



Query

```
SELECT ?age
WHERE { <http://example.org#mary>
<http://example.org#age>  ?age }
```

```
PREFIX f: <http://example.org#>
SELECT ?age
WHERE { f:mary f:age ?age }
```



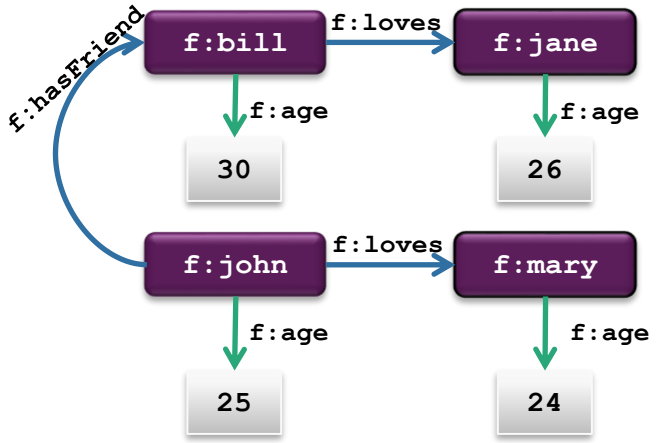
Result

age
24

Simple SPARQL queries (2)

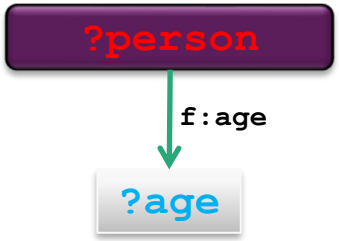
- Show me ***f:age*** of all resources

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE { ?person f:age ?age }
```



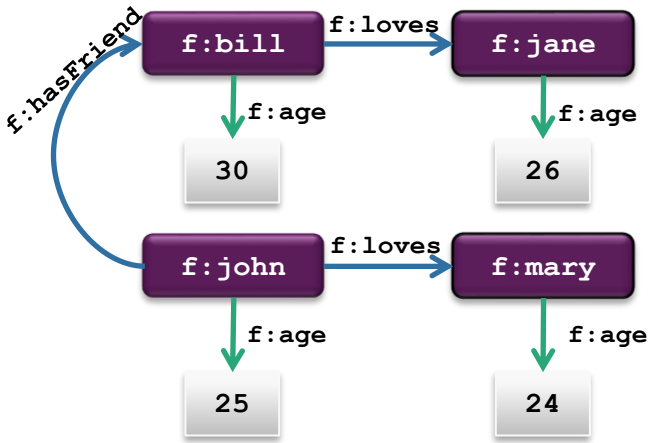
Result

person	age
f:bill	30
f:jane	26
f:john	25
f:mary	24

Simple SPARQL queries (3)

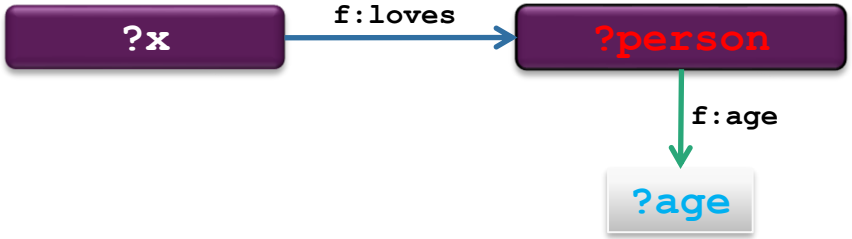
- Show me all things that are loved. Also show me their age (***f:age***)

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {
   ?x f:loves ?person .
   ?person f:age ?age
}
```



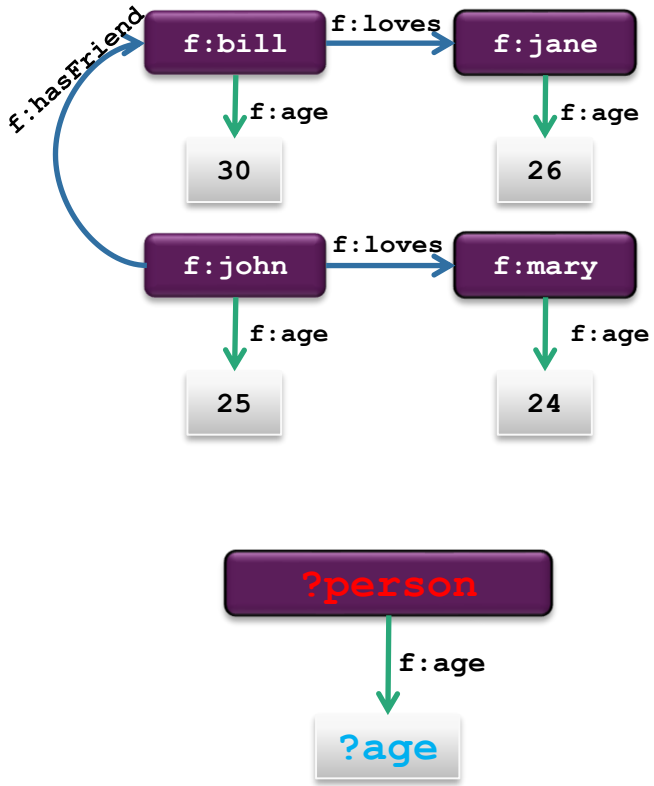
Result

person	age
f:jane	26
f:mary	24

SPARQL: FILTER (testing values)

- Show me people and their age for people older than 25.

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {
   ?person f:age ?age .
   FILTER (?age > 25)
}
```

If ?age is not a number, then it will not work

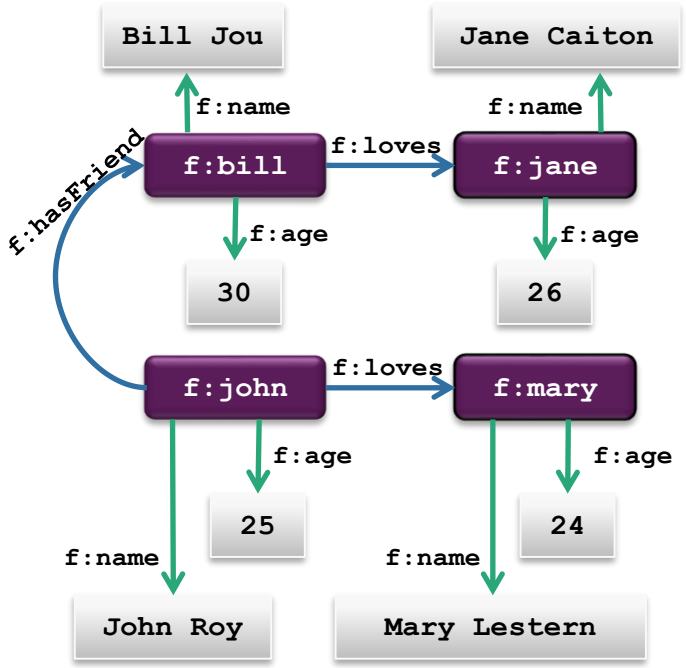
Result

person	age
f:bill	30
f:jane	26

SPARQL: FILTER (string matching)

- Show me people and their name if name has "***r***" or "***R***" in it.

Data



Syntax

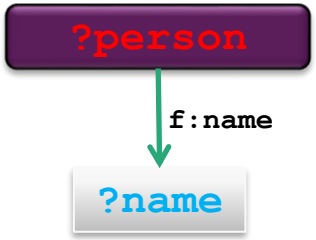
```
FILTER regex(?x, "pattern"[, "flags"])
```

*Flag "**i**" means a case-insensitive pattern*

Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?name
WHERE {
   ?person f:name ?name .
   FILTER regex(?name, "r", "i" )
}
```

Result

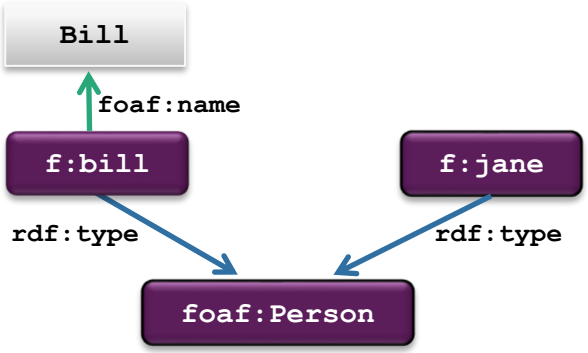


person	name
f:john	John Roy
f:mary	Mary Lestern

SPARQL: FILTER (EXISTS / NOT EXISTS)

- EXISTS expression tests whether the pattern can be found in the data.
- NOT EXISTS expression tests whether the pattern does not match the dataset.

Data



Result

person
f:bill

person
f:jane

Syntax

```
FILTER EXISTS {"pattern"}
```

```
FILTER NOT EXISTS {"pattern"}
```

Query

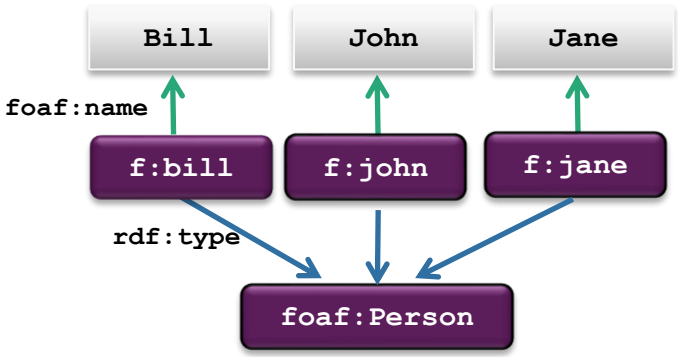
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX f: <http://example.org#>

SELECT ?person
WHERE {
  ?person rdf:type foaf:Person .
  FILTER EXISTS {?person foaf:name ?name}
}
```

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

SPARQL: FILTER (MINUS)

- MINUS removes matches based on the evaluation of two patterns.

Data



Query

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

SELECT DISTINCT ?s
WHERE { ?s ?p ?o .
        MINUS { ?s foaf:name "John" . }
}
```

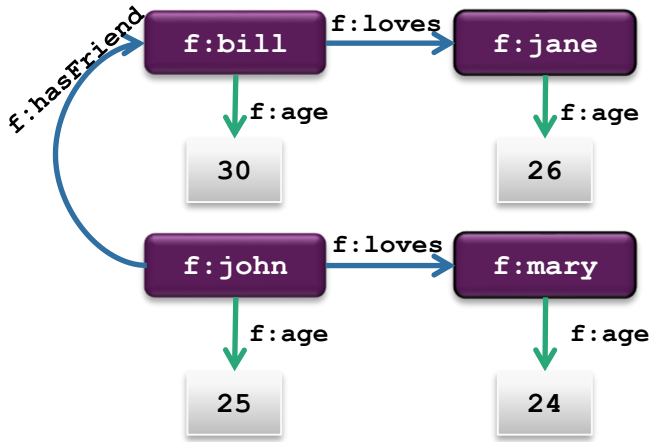
Result

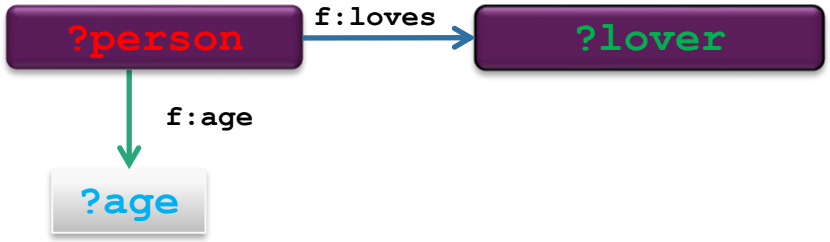
s
f:bill
f:jane

SPARQL: OPTIONAL

- Show me the person and its age (***f:age***). If you have information that person loves somebody, then show it as well.

Data





Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
  ?person f:age ?age .
  OPTIONAL {?person f:loves ?lover}
}
```

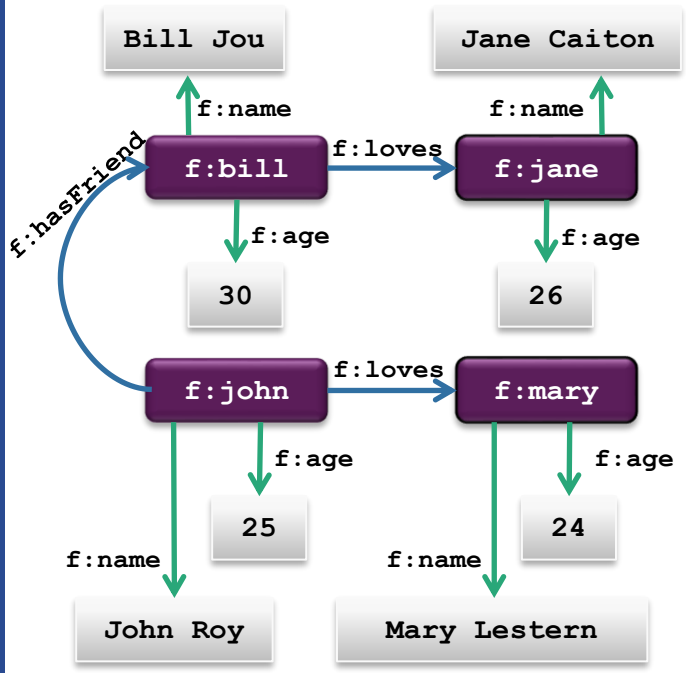
Result

person	age	lover
f:bill	30	f:jane
f:john	25	f:mary
f:mary	24	
f:jane	26	

SPARQL: OPTIONAL with FILTER

- Show me the person and its age (***f:age***). If you have information about that person loving somebody, then show that person if his/her name contains " ***r*** ".

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
  ?person f:age ?age .
  OPTIONAL {?person f:loves ?lover .
            ?lover f:name ?loverName .
           FILTER regex(?loverName, "r", "i")}
}
```

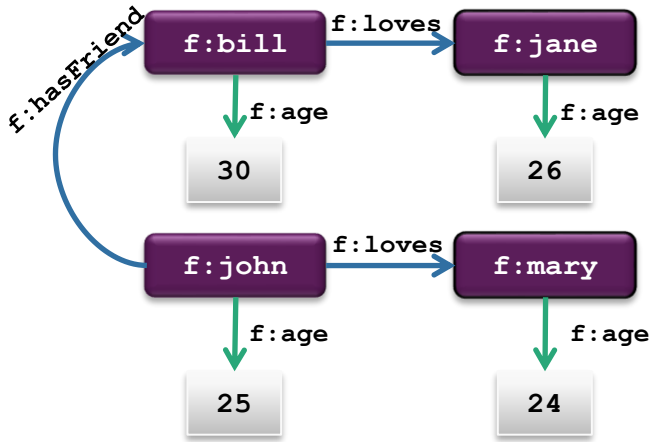
Result

person	age	lover
f:bill	30	
f:john	25	f:mary
f:mary	24	
f:jane	26	

SPARQL: Logical OR (UNION)

- Show me all people who have a friend together with all the people that are younger than 25

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person
WHERE {
      {?person f:age ?age . FILTER (?age < 25)}
  UNION
      {?person f:hasFriend ?friend}
}
```

```
PREFIX f: <http://example.org#>
SELECT ?person
WHERE {?person  f:age ?age . FILTER (?age < 25)}
```

+

```
PREFIX f: <http://example.org#>
SELECT ?person
WHERE {?person  f:hasFriend ?friend}
```

Result

person
f:mary
f:john

SPARQL: Solution set modifiers

- Example:

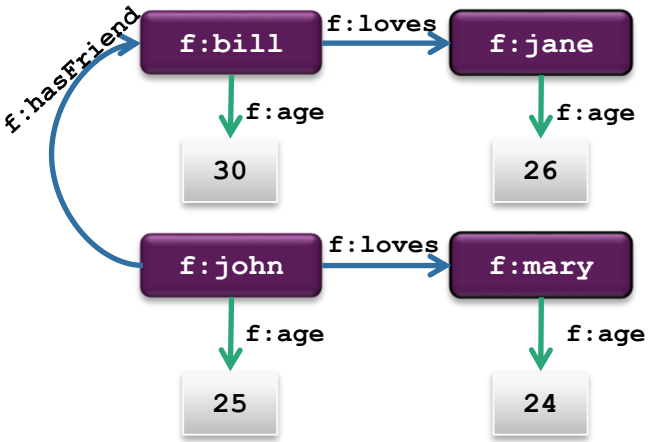
```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {?person f:age ?age }
ORDER BY ?age
```

- Others:
  - ORDER BY DESC(*?x*)
    - Arrange in descending order
  - LIMIT *n*
    - Include only first *n* solutions
  - OFFSET *n*
    - Include solutions starting from index *n+1*
  - SELECT DISTINCT
    - Do not duplicate solutions
  - SELECT *
    - Return all named variables
  - ASK
    - Yes/No answer (whether or not a solution exists)

SPARQL: Constructing graphs

- Annotate people with age below 26 as young people

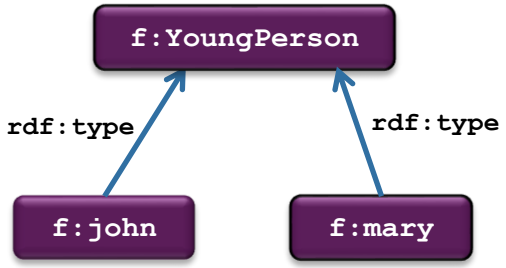
Data



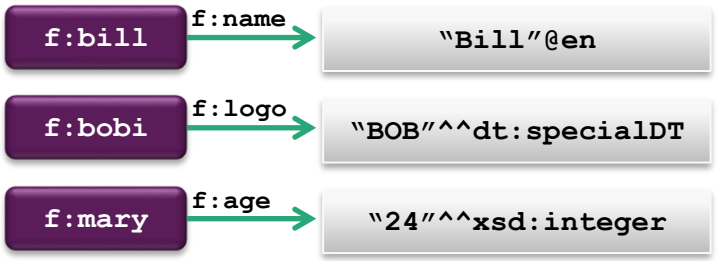
Query

```
PREFIX f: <http://example.org#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

CONSTRUCT { ?person rdf:type f:YoungPerson }

WHERE {?person f:age ?age . FILTER (?age < 26) }
```

Result



Simple SPARQL (matching RDF Literals)



Data

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

f:bill f:name "Bill"@en .
f:bobi f:logo "BOB"^^dt:specialDT .
f:mary f:age  "24"^^xsd:integer .
```

- Literals with Language Tags (e.g. **`@en`**)

Query

```
SELECT ?s
WHERE { ?s ?p "Bill" }
```

Result

s

"Bill" is not the same RDF literal as "Bill"@en

```
SELECT ?s
WHERE { ?s ?p "Bill"@en }
```

s
f:bill

- Literals with Numeric Types

```
SELECT ?s
WHERE { ?s ?p 24 }
```

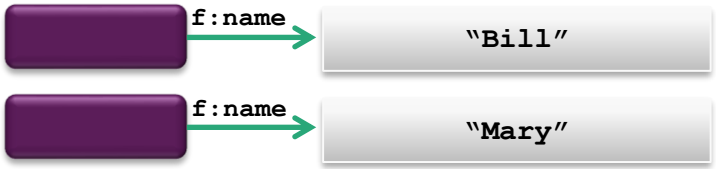
s
f:mary

- Literals with Arbitrary Datatypes

```
PREFIX dt: <http://example.org/datatype#>
SELECT ?s
WHERE { ?s ?p "BOB"^^dt:specialDT }
```

s
f:bobi

Simple SPARQL (Blank Node Labels)



Data

```
@prefix   f: <http://example.org/ont#> .

_:a f:name "Bill" .
_:b f:name "Mary" .
```

Query

```
PREFIX f: <http://example.org/ont#>
SELECT ?s ?:name
WHERE { ?s f:name ?:name }
```

Result

s	name
_:c	"Bill"
_:d	"Mary"

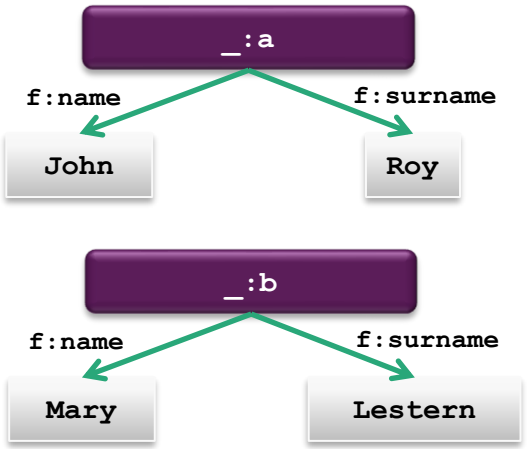
s	name
_:e	"Bill"
_:f	"Mary"

The blank node labels in the result ***could be different***, because they only indicate whether RDF terms in the solutions are the same or different.

SPARQL: Functions

- Return the names of the resources as concatenation of ***f:name*** and ***f:surname*** properties' values.

Data



Query

```
PREFIX f: <http://example.org#>

SELECT ?fullName
WHERE {?resource f:name ?name ;
                 f:surname ?surname
       BIND(CONCAT(?name," ",?surname) AS ?fullName)
}
```

Result

fullName
"John Roy"
"Mary Lestern"

SPARQL: Functions

- SPARQL Query language has a set of functions:
  - Functional Forms (BOUND, IF, COALESCE, NOT EXISTS / EXISTS, IN / NOT IN, etc.)
  - Functions on RDF Terms
  - Functions on Strings
  - Functions on Numerics
  - Functions on Dates and Times
  - Hash Functions

- Documentation: *http://www.w3.org/TR/sparql11-query/#SparqlOps*

TriG notation for RDF1.1

- ***TriG*** is an extension of the *Turtle* format, extended to support representing a complete *RDF Dataset* in a compact and natural text form. Link: *http://www.w3.org/TR/trig/*

N-Quads

```
<http://example.org/John> <http://example.org/hasName> "John" <http://example.org/graph1> .
          <…>                      <…>                 "…"   <http://example.org/graph1> .
<http://example.org/Mary> <http://example.org/hasName> "Mary" <http://example.org/graph2> .
          <…>                      <…>                 "…"   <http://example.org/graph2> .
…
```

TriX

```
<TriX xmlns="http://www.w3.org/2004/03/trix/trix-1/">
  <graph>
    <uri>http://example.org/graph1</uri>
    <triple>
      <uri>http://example.org/John</uri>
      <uri>http://example.org/hasName</uri>
      <plainLiteral>John</plainLiteral>
    </triple>
    <triple>
    …
    </triple>
  </graph>
  <graph>
    <uri>http://example.org/graph2</uri>
    <triple>
      <uri>http://example.org/Mary</uri>
      <uri>http://example.org/hasName</uri>
      <plainLiteral>Mary</plainLiteral>
    </triple>
    <triple>
    …
    </triple>
  </graph>
</TriX>
```

TriG

```
@prefix ex: <http://example.org/>.

# default graph
    {    …    }

<http://example.org/graph1>
{ ex:John ex:hasName  "John" .
  …   }

<http://example.org/graph2>
{ ex:Mary ex:hasName  "Mary" .
  …   }
```

```
@prefix ex: <http://example.org/>.

# default graph
    {    …    }

ex:graph1 { ex:John ex:hasName  "John" .
            …
          }
ex:graph2 { ex:Mary ex:hasName  "Mary" .
            …
          }
```

SPARQL: RDF Dataset

- Finding matches in a specific graph.

```
#Default graph (located at http://users.jyu.fi/graphs.ttl)
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example.org/bob>    dc:publisher  "Bob Hacker" .
<http://example.org/alice>  dc:publisher  "Alice Hacker" .
```

```
#Named graph: http://example.org/bob
@prefix f: <http://example.org#> .
_:a f:name "Bob" .
_:a f:age “25” .
```

```
#Named graph: http://example.org/alice
@prefix f: <http://example.org#> .
_:a f:name “Alice" .
_:a f:age “22” .
```

Query

```
PREFIX f: <http://example.org#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?graph ?name ?age
WHERE {
  ?graph dc:publisher ?who .
  GRAPH ?graph {?person f:name ?name .
                ?person f:age ?age }
}
```

Result

graph	name	age
<http://example.org/bob>	Bob	25
<http://example.org/alice>	Alice	22

SPARQL: several sources

```
f:john f:age "25"^^xsd:integer .
f:bill f:age "30"^^xsd:integer .
f:mary f:age "24"^^xsd:integer .
f:jane f:age "26"^^xsd:integer .
f:john f:loves f:mary .
f:bill f:loves f:jane .
f:john f:hasFriend f:bill
```

http://users.jyu.fi/~olkhriye/ties4520/rdf/people.rdf

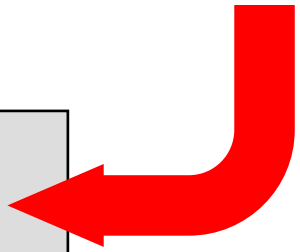
```
j:teacher rdf:type j:EducationJob .
j:seniorResearcher rdf:type j:ResearchJob .
j:juniorResearcher rdf:type j:ResearchJob .
j:professor rdf:type j:ResearchJob, j:EducationJob .
```

http://users.jyu.fi/~olkhriye/ties4520/rdf/jobs.rdf

```
f:john e:worksAs j:teacher .
f:mary e:worksAs j:seniorResearcher .
f:jane e:worksAs j:juniorResearcher .
f:bill e:worksAs j:professor .
```

http://users.jyu.fi/~olkhriye/ties4520/rdf/employment.rdf

From now on prefixes will be omitted to save space



```
@prefix j: <http://jyu.fi/jobs#> .
@prefix e: <http://jyu.fi/employment#> .
@prefix f: <http://example.org#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

Prefixes

SPARQL: several sources

- Show me people, their age and their job, if a job is related to education.

Query

```
…
SELECT ?person ?age ?job
FROM <http://users.jyu.fi/~olkhriye/ties4520/rdf/employment.rdf>
FROM <http://users.jyu.fi/~olkhriye/ties4520/rdf/people.rdf>
FROM <http://users.jyu.fi/~olkhriye/ties4520/rdf/jobs.rdf>
WHERE {
  ?person f:age ?age .
  ?person e:worksAs ?job .
  ?job rdf:type j:EducationJob .
}
```

Result

person	age	job
f:bill	30	j:professor
f:john	25	j:teacher

```
FROM < URI >
FROM NAMED < URI >
```

- is used to identify the contents to be in the default graph
- is used to identify a named graph

URI refers a Graph ID…

It *is not* a distributed query among separate RDF files. It is used just to define more specific (sub) dataset for the particular query.

SPARQL: several sources

Use ***SERVICE*** clause for federated queries.

- The *SERVICE* keyword instructs a federated query processor to invoke a portion of a SPARQL query against a *remote SPARQL endpoint (not a RDF file)*.

Query

```
…
PREFIX ex: <http://users.jyu.fi/~olkhriye/ties4520/rdf/>
SELECT ?person ?age ?job
WHERE {
  SERVICE ex:people {?person f:age ?age .}
  SERVICE ex:employment {?person e:worksAs ?job .}
  SERVICE ex:jobs {?job rdf:type j:EducationJob .}
}
```

Result

person	age	job
f:bill	30	j:professor
f:john	25	j:teacher

Some Public SPARQL Endpoints

Name	URL	What's there?
SPARQLer	http://sparql.org/sparql.html	General-purpose query endpoint for Web-accessible data
DBPedia	http://dbpedia.org/sparql	Extensive RDF data from Wikipedia
DBLP	http://www4.wiwiss.fu-berlin.de/dblp/snorql/	Bibliographic data from computer science journals and conferences
LinkedMDB	http://data.linkedmdb.org/sparql	Films, actors, directors, writers, producers, etc.
World Factbook	http://www4.wiwiss.fu-berlin.de/factbook/snorql/	Country statistics from the CIA World Factbook
bio2rdf	http://bio2rdf.org/sparql	Bioinformatics data from around 40 public databases

Triple store SPARQL Endpoints

RDF4J workbench:

- *http://localhost:8080/rdf4j-server/repositories/ <repository name>*

GraphDB:

- *http://localhost:7200/repositories/ <repository name>*

Apache Jena Fuseki:

- *http://localhost:8080/fuseki/ <repository name> /query or sparql*
- *http://localhost:3030/ <repository name> /query or sparql*

SPARQL1.1: Update

- INSERT DATA and DELETE DATA

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
INSERT DATA { <http://example/book1> dc:title "A new book" ;
                                     dc:creator "A.N.Other" .
            }
```

```
DELETE DATA { GRAPH <http://example/bookStore>
                  { <http://example/book2> dc:title "David Copperfield";
                                           dc:creator "Edmund Wells" .
                  }
            }
```

- Documentation: *http://www.w3.org/TR/sparql11-update/*

SPARQL1.1: Update

- DELETE / INSERT

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
DELETE { ?book ?p ?v }
WHERE  { ?book dc:date ?date .
         FILTER ( ?date > "1970-01-01T00:00:00-02:00"^^xsd:dateTime )
         ?book ?p ?v
       }
```

```
WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
WHERE  { ?person foaf:givenName 'Bill' }
```

```
INSERT { GRAPH <http://example/bookStore2> { ?book ?p ?v } }
WHERE  { GRAPH  <http://example/bookStore>
           { ?book dc:date ?date .
             FILTER ( ?date < "2000-01-01T00:00:00-02:00"^^xsd:dateTime )
             ?book ?p ?v } } ;
WITH <http://example/bookStore>
DELETE { ?book ?p ?v }
WHERE  { ?book dc:date ?date ;
                 dc:type dcmitype:PhysicalObject .
         FILTER ( ?date < "2000-01-01T00:00:00-02:00"^^xsd:dateTime )
         ?book ?p ?v }
```

- Documentation: *http://www.w3.org/TR/sparql11-update/*

SPARQL1.1: Update

- DELETE WHERE

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
```

```
DELETE WHERE { ?person foaf:givenName 'Fred';
                       ?property      ?value }
```

```
DELETE WHERE { GRAPH <http://example.com/names>
                     { ?person foaf:givenName 'Fred' ;
                               ?property1     ?value1
                     }
               GRAPH <http://example.com/addresses>
                     { ?person ?property2 ?value2  }
             }
```

- LOAD operation reads an RDF document from a IRI and inserts its triples into the specified graph in the Graph Store.

```
LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?
```

- CLEAR operation removes all the triples in the specified graph(s) in the Graph Store.

```
CLEAR  ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )
```

- Documentation: *http://www.w3.org/TR/sparql11-update/*

SPARQL1.1: Update (Graph Management)

- CREATE creates a new graph in stores that support empty graphs. The contents of already existing graphs remain unchanged.

```
CREATE ( SILENT )? GRAPH IRIref
```

- DROP removes a graph and all of its contents (DROP DEFAULT is equivalent to CLEAR DEFAULT).

```
DROP  ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )
```

- COPY modifies a graph to contain a copy of another. It inserts all data from an input graph into a destination graph. Data from the destination graph, if any, is removed before insertion.

```
COPY (SILENT)? ((GRAPH)? IRIref_from | DEFAULT) TO ((GRAPH)? IRIref_to | DEFAULT)
```

```
DROP SILENT (GRAPH IRIref_to | DEFAULT);
INSERT {(GRAPH IRIref_to)? {?s ?p ?o}} WHERE {(GRAPH IRIref_from)? {?s ?p ?o}}
```

- MOVE moves all of the data from one graph into another. The input graph is removed after insertion and data from the destination graph, if any, is removed before insertion.

```
MOVE (SILENT)? ((GRAPH)? IRIref_from | DEFAULT) TO ((GRAPH)? IRIref_to | DEFAULT)
```

```
DROP SILENT (GRAPH IRIref_to | DEFAULT);
INSERT {(GRAPH IRIref_to)? {?s ?p ?o}} WHERE {(GRAPH IRIref_from)? {?s ?p ?o}};
DROP ( GRAPH IRIref_from | DEFAULT)
```

- ADD reproduces all data from one graph into another. Data from the input graph is not affected, and initial data from the destination graph, if any, is kept intact.

```
ADD (SILENT)? ((GRAPH)? IRIref_from | DEFAULT) TO ((GRAPH)? IRIref_to | DEFAULT)
```

```
INSERT {(GRAPH IRIref_to)? {?s ?p ?o}} WHERE {(GRAPH IRIref_from)? {?s ?p ?o}}
```

- Documentation: *http://www.w3.org/TR/sparql11-update/*

More on SPARQL

- Official specification:
  - *http://www.w3.org/TR/sparql11-query/*
- SPARQL update
  - *http://www.w3.org/TR/sparql11-update/*
- SPARQL 1.1 Federated Query
  - *https://www.w3.org/TR/2013/REC-sparql11-federated-query-20130321/#simpleService*
  - *https://www.w3.org/TR/sparql11-federated-query/*
  - *http://www.snee.com/bobdc.blog/2010/01/federated-sparql-queries.html*
- Other SPARQL tutorials:
  - *https://jena.apache.org/tutorials/sparql.html*
- SPARQL endpoints:
  - *https://www.w3.org/wiki/SparqlEndpoints*
- SPARQL extensions:
  - *https://www.w3.org/wiki/SPARQL/Extensions*

Apache Jena Fuseki

- Since Fuseki v3 stars to support TriG format as well, you can upload data in TriG format directly from the file. But, there is a possibility to define named graphs in the structure of repository in other way. If you noticed, there is a possibility to provide Graph Id while uploading a file. So… you may create 2 Turtle files. First one should contain triples from default graph of initial TriG document, second file should contain triples from the named graph. Specify corresponding graph ID while uploading the files ("default" for the first file, *<id of the named graph>* for the second).
- Fuseki ***does not support "#"*** character in the graph URI (drops out the rest starting form #). Use "***/***" character instead of "#".
- Fuseki has following URLs of SPARQL Endpoints:
  - http://localhost:3030/repName/query - for SPARQL queries;
  - http://localhost:3030/repName/update - for SPARQL UPDATE queries.
- Fuseki distinguishes content of default and named graphs. If you do not use ***GRAPH <grophID>*** in the query pattern and specify only triples, query engine associates the triples only with content of default graph.

In contrast, ***RDF4J (Sesame)*** "ignores" association with graphs, if you use just triples in the query pattern, and treats quads (quadruples) as triples. It looks like triples of all the named graphs are considered as triples of default graph.

Empty Graphs

- SPARQL UPDATE query *CREATE GRAPH <graphID>* executes properly, but does not actually creates an empty graph.
- You can use new graph IDs in SPARQL UPDATE queries and corresponding graph will be created automatically.
- If the content of the named graph become empty (is deleted), such empty graph disappears from the repository.

In contrast to ***Fuseki***, ***RDF4J (Sesame)*** considers any empty graph as existing graph (any empty subset is a part of a set).

Task 2