UNIVERSITY OF JYVÄSKYLÄ

Lecture 2: Storing and querying RDF data

TIES452 Practical Introduction to Semantic Technologies
Autumn 2014

University of Jyväskylä

Khriyenko Oleksiy

Part 1

Storing RDF data

Storing of RDF

- Small datasets (few triples)
  - RDF file published on the web or stored locally
  - Examples: `*.rdf`, `*.nt`, `*.ttl`, `*.n3`, etc.
- Large datasets (thousands to millions of triples)
  - Database-bases solution better
  - Usually in form of RDF storage
- Legacy data
  - Keep in original form
  - Provide mapping to RDF
  - Expose as RDF to the outer world

List of Triplestores: http://en.wikipedia.org/wiki/Triplestore

Native RDF Stores

RDF stores that implement their own database engine without reusing the storage and retrieval functionalities of other database management systems:

- ***4Store and 5Store*** (under GPL and commercial) are RDF databases developed by Garlik Inc. 4Store is available under GNU General Public License (GPL). Client connectors are available for PHP, Ruby, Python, and Java. 5Store (unlike 4Store) is commercial software and provides similar features as 4Store, but improved efficiency and scalability. Link: http://4store.org/ and http://4store.org/trac/wiki/5store
- ***AllegroGraph*** (commercial) is a commercial RDF graph database and application framework developed by Franz Inc. There are different editions of AllegroGraph and different clients: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure. Clients connectors are available for Java, Python, Lisp, Clojure, Ruby, Perl, C#, and Scala. Link: http://franz.com/agraph/allegrograph/
- ***Apache Jena TDB*** (open-source) is a component of the Jena Semantic Web framework and available as open-source software released under the BSD license. Link: http://jena.apache.org/
- ***Mulgara*** (open source, Open Software License) is the community-driven successor of Kowari and is described as a purely Java-based, scalable, and transaction-safe RDF database for the storage and retrieval of RDF-based metadata. Link: http://www.mulgara.org/
- ***GraphDB™*** (formerly ***OWLIM***) – An Enterprise Triplestore with Meaning (GNU LGPL license and commercial) It is a family of commercial RDF storage solutions, provided by Ontotext. There are three different editions: *GraphDB™ Lite*, *GraphDB™ Standard* and *GraphDB™ Enterprise*.
  Link: http://www.ontotext.com/products/ontotext-graphdb-owlim/
- ***etc.***

DBMS-backed Stores

RDF Stores that use the storage and retrieval functionality provided by another database management system:

- ***ARC2*** is a free, open-source semantic web framework for PHP applications released under the W3C Software License and the GNU GPL. It is designed as a PHP library and includes RDF parsers (RDF/XML, N-Triples, Turtle, SPARQL + SPOG, Legacy XML, HTML tag soup, RSS 2.0, Google Social Graph API JSON…) and serializers (N-Triples, RDF/JSON, RDF/XML, Turtle, SPOG dumps…), an RDBMS-backed (MySQL) RDF storage, and implements the SPARQL query and update specifications. Active code development has been discontinued due to lack of funds and the inability to efficiently implement the ever-growing stack of RDF specifications. Link: https://github.com/semsol/arc2/wiki

  More reading: http://tinman.cs.gsu.edu/~raj/8711/sp11/presentations/ARC_RDF_Store.pdf

- ***Apache Jena SDB*** (open-source) is another component of the Jena Semantic Web framework and provides storage and query for RDF datasets using conventional relational databases: Microsoft SQL Server, Oracle 10g, IBM DB2, PostgreSQL, MySQL, HSQLDB, H2, and Apache Derby. Link: http://jena.apache.org/

- ***Oracle Spatial and Graph: RDF Semantic Graph*** (formerly ***Oracle Semantic Technologies***) is a W3C standards-based, full-featured graph store in Oracle Database for Linked Data and Social Networks applications.

  Link: http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/rdfsemantic-graph-1902016.html

- ***Semantics Platform*** is a family of products for building medium and large scale semantics-based applications using the Microsoft .NET framework. It provides semantic technology for the storage, services and presentation layers of an application. Link: http://www.intellidimension.com/products/semantics-platform/

- ***RDFLib*** is a pure Python package working with RDF that contains most things you need to work with, including: parsers and serializers for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata; a Graph interface which can be backed by any one of a number of Store implementations; store implementations for in memory storage and persistent storage on top of the Berkeley DB; a SPARQL 1.1 implementation supporting Queries and Update statements. Link: https://rdflib.readthedocs.org/en/latest/

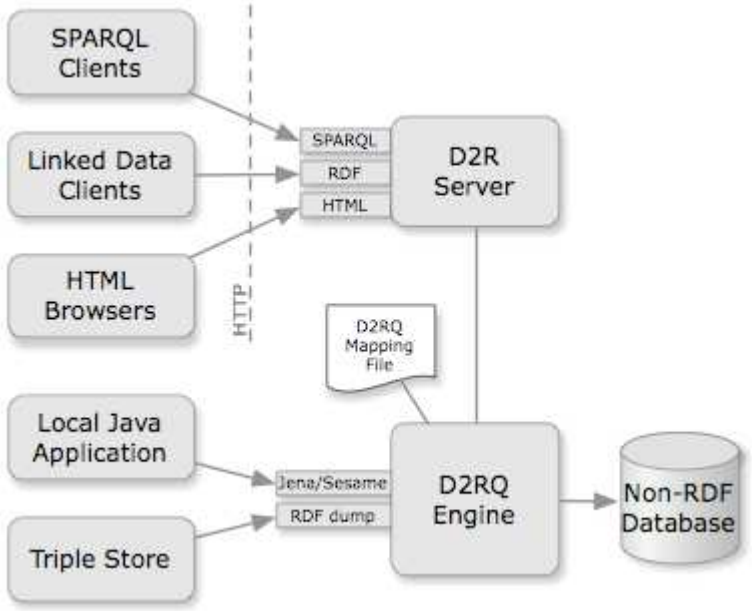
Non-RDF DB support

- ***D2RQ Platform*** is a system for accessing relational databases as virtual, read-only RDF graphs. It offers RDF-based access to the content of relational databases without having to replicate it into an RDF store. Using D2RQ you can: query a non-RDF database using SPARQL, access the content of the database as Linked Data over the Web, create custom dumps of the database in RDF formats for loading into an RDF store, access information in a non-RDF database using the Apache Jena API.

  Link: http://d2rq.org/

The D2RQ Platform consists of:

- **D2RQ Mapping Language**, a declarative mapping language for describing the relation between an ontology and an relational data model.
- **D2RQ Engine**, a plug-in for the Jena Semantic Web toolkit, which uses the mappings to rewrite Jena API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks.
- **D2R Server**, an HTTP server that provides a Linked Data view, a HTML view for debugging and a SPARQL Protocol endpoint over the database.



Hybrid Stores

RDF Stores that supports both architectural styles (native and DBMS-backed):

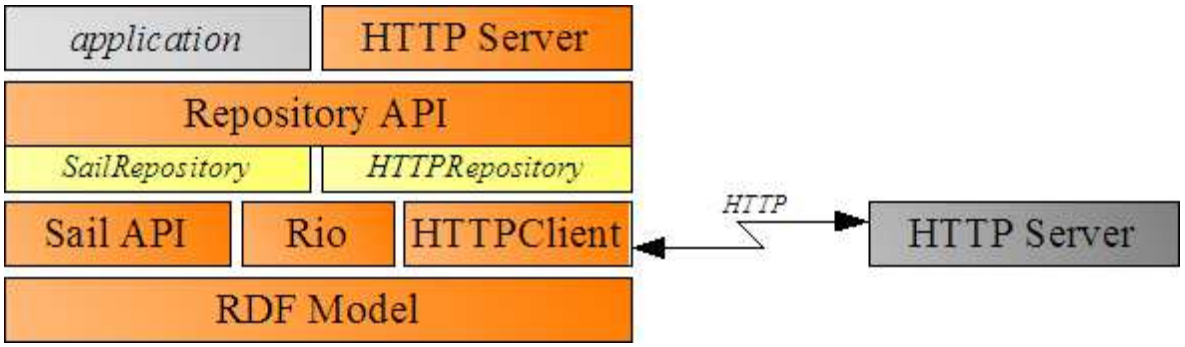
- ***RedStore*** is a lightweight RDF triplestore written in C using the Redland library. Link: http://www.aelius.com/njh/redstore/
- ***Sesame*** (open-source) is an open source framework for storage, inferencing and querying of RDF data. It is a library that is release under the Aduna BSD-style license and can be integrated in any Java application. Sesame includes RDF parsers and writers (Sesame Rio), a storage and inference layer (SAIL API) that abstracts from storage and inference details, a repository API for handling RDF data, and an HTTP Server for accessing Sesame repositories via HTTP. It operates in any Java-supporting environment and can be used by any Java application. Link: http://www.openrdf.org/
- ***OpenLink Virtuoso Universal Server*** is a hybrid storage solution for a range of data models, including relational data, RDF and XML, and free text documents. Through it unified storage it can be also seen as a mapping solution between RDF and other data formats, therefore it can serve as an integration point for data from different, heterogeneous sources. Virtuoso has gained significant interest since it is used to host many important Linked Data sets (e.g., DBpedia), and preconfigured snapshots with several important Linked Data sets are offered. Virtuoso is offered as an open-source version; for commercial purposes several license models exist. Link: http://virtuoso.openlinksw.com/
- ***Bigdata*** (GNU General Public License) is ultra-scalable, high-performance graph database supporting the RDF data model, which is a set of W3C standards for representation, interchange, and query of linked data. It is designed for UNIX systems (Linux) with client connectors available for Java (Sesame). Bigdata supports the standardized high-level query language for RDF called SPARQL, which is a powerful query language supporting update, complex joins, nested subqueries, aggregations, federation, custom functions and services, and much more. Bigdata also includes a soon-to-be standardized RDF extension called “RDR”, which provides a compact interchange representation and query syntax for link attributes.

  Link: http://www.systap.com/bigdata

Sesame

- ***Sesame*** is a framework for processing RDF data
- Home: http://www.openrdf.org/
- Features:
  - Parsing
    - Supports all major notations
  - Storing
    - In-memory, RDBS-backed, file-based
  - Inferencing
    - Rule-based, Ontology-based
  - Querying
    - SPARQL, SeRQL
- Java-based API + tools
- Elmo framework: allows Java applications to access RDF stores and map a Java object model to the RDF (http://semanticweb.org/wiki/Elmo)

Sesame architecture



- Rio (RDF I/O)
  - Parsers and writers for various notations
- Sail (Storage And Inference Layer)
  - Low level System API
  - Abstraction for storage and inference
- Repository API
  - Higher level API
  - Developer-oriented methods for handling RDF data
- HTTP Server
  - Accessing Sesame through HTTP

Image source: http://www.openrdf.org/doc/sesame2/users/ch03.html

Environmental variables

practical

- OS stores simple variables in its memory
- Applications can ask OS for the value
- Each variable has a name and a simple textual value
- To see variables and their values run (Win):
  - See all: `set`
  - See a concrete variable: `set VARNAME`
- Scope:
  - Global (valid for the whole OS)
  - Local (valid in the current command line window)
- To change or set new local variable run:

```
set VARNAME=some text here
```

Installing Sesame workbench

- Simple web interface for storing and querying RDF data
- Install steps (no admin rights needed): practical
  1. Download and unzip newest Sesame and Tomcat
  2. Copy all **`*.war`** files from Sesame's **`war`** folder to Tomcat's **`webapps`** folder
  3. Start Tomcat
     - From bin folder by running **`startup.sh`** (UNIX) or **`startup.bat`** (Win)
     - You may need to set **`JAVA_HOME`** variable (it should point to JDK or JRE main folder)
  4. Go to http://localhost:8080/openrdf-workbench

Apache Jena

- ***Apache Jena*** is a Java framework (collection of tools and Java libraries) to simplify the development of Semantic Web and Linked Data applications.
- Home: http://jena.apache.org/index.html
- Includes:
  - RDF API for processing RDF data in various notations
  - Ontology API for OWL and RDFS
  - Rule-based inference engine and Inference API
  - TDB – a native triple store
  - SPARQL query processor (called *ARQ*)
  - Servers for publishing RDF data to other applications

ARQ

practical

- Installation:
  - Download and unzip Jena
  - Set **`JENAROOT`** environmental variable to folder where you unzipped Jena
  - In **`bat`** (Win) or **`bin`** (UNIX) you find **`arq`** executable
- Usage:
  - Prepare a SPARQL query and save it into a file (here: **`query.sparql`**)
  - Prepare some data file (if needed) – e.g. **`data.rdf`**(.**`ttl`**)
  - Execute the query on top of the data by running:

```
arq --query=query.sparql --data=data.rdf
```

- In case of problems use **`arq --help`**

Joseki

- ***Joseki*** - SPARQL server for Jena that lets to separate application code and data store into separate server processes while communicating through SPARQL.
- Features:
  - RDF Data from files and databases
  - HTTP (GET&POST) implementation of the SPARQL protocol
- Installation: **practical**
  1. Download (http://www.joseki.org/download.html) and unzip Joseki
  2. Check if you can run java from command line

```
java -cp "%CLASSPATH%;lib\*" joseki.rdfserver joseki-config.ttl
```

  3. Start Joseki server from Joseki folder by running:
  4. Open http://localhost:2020/
  5. Installation tutorial (Jena, TDB and Joseki): http://www.milanmarkovic.org/tutorials/jenaTDBjoseki.html
- Joseki is no longer being developed. It is replaced by ***Apache Jena Fuseki***.

Apache Jena Fuseki

- ***Fuseki*** - is a SPARQL server that provides REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the SPARQL protocol over HTTP.
- Installation: **practical**
  1. Download (http://jena.apache.org/download/ or from course webpage) and unzip Fuseki
  2. Start server from Fuseki folder by running:
     ```
     fuseki-server --update --mem /ds
     ```
     ```
     fuseki-server --update --loc=c:/dir… /ds
     ```
  3. Open http://localhost:3030/
  4. Documentation: http://jena.apache.org/documentation/serving_data

Mulgara

- ***Mulgara*** – is a RDF database (*successor of Kowari RDF database*): http://www.mulgara.org/
- Written entirely in Java
- Querying language – SPARQL and own TQL
- TQL language:
  - Interpreted querying and command language
  - To manage Mulgara storage (upload, etc.)
- SPARQL – query-only language:
- REST interface for TQL and SPARQL
- Starting the server: **practical**
  - Download and unzip the binary package
  - Inside the *Mulgara* folder run: `java -jar mulgara-2.1.13.jar`
  - Web interface: http://localhost:8080/webui/
  - SPARQL REST interface: http://localhost:8080/sparql/

Mulgara

practical

- Create a model: `create <http://localhost/server#myModel>;`
- Insert some RDF triples:

```
insert <http://someServer.com/somePage.html>
<http://purl.org/dc/elements/1.1/title> ‘Some Cool Webpage‘
into <http://localhost/server#myModel>;
```

```
alias <http://purl.org/dc/elements/1.1/> as dc;
insert <http://someServer.com/someOtherPage.html> <dc:title>
  ‘Some Other Cool Webpage' into <http://localhost/server#myModel>;
```

- Upload RDF triples from file
- Query RDF store with TQL or SPARQL query languages:

```
select $subject $predicate $object from <http://localhost/server#myModel>
where $subject $predicate $object;
```

```
alias <http://purl.org/dc/elements/1.1/> as dc;
select $subject $object from <http://localhost/server#myModel>
where $subject <dc:title> $object;
```

AllegroGraph

- ***AllegroGraph*** is a high-performance persistent graph database
- Editions of AllegroGraph: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure.
- Supports SPARQL, RDFS++, and Prolog reasoning
- Supports REST Protocol clients: Java Sesame, Java Jena, Python, C#, Clojure, Perl, Ruby, Scala and Lisp clients.
- Link: http://www.franz.com/agraph/allegrograph/
- ***AllegroGraph Web View (AGWebView)*** is a graphical user interface for exploring, querying, and managing AllegroGraph triple stores. It uses HTTP interface to provide the services through a web browser.
- ***Gruff*** - a graph-based triple-store browser for AllegroGraph.

  practical

  – Download and unzip the package: http://www.franz.com/agraph/gruff/
  – Run the browser, create new triple store
  – Load sample data from own data files or files available at the same download page or the course webpage.

GraphDB™

- ***GraphDB™*** (formerly *OWLIM*) – An Enterprise Triplestore with Meaning.

It is a family of commercial RDF storage solutions, provided by Ontotext. There are three different editions: *GraphDB™ Lite*, *GraphDB™ Standard* and *GraphDB™ Enterprise*.

- Link: http://www.ontotext.com/products/ontotext-graphdb-owlim/

- *GraphDB™ Lite* is a free RDF triplestore that allows you to store up to 100 million triples on a desktop computer, perform SPARQL 1.1 queries in memory (not using files based indices), and supports reasoning operations for inferencing.

- *GraphDB™ Standard* allows organizations to manage tens of billions of semantic triples on one commodity server, load and query RDF statements at scale simultaneously, performs querying and reasoning operations using file based indices, and has optimized performance using "Same As" technology.

- *GraphDB™ Enterprise* has all the features of our Standard Edition with the added advantage that it has been architected to run on enterprise replication clusters with backup, recovery and failover ensuring you are always up. "Enterprise" offers industrial strength resilience and linearly scalable parallel query performance with support for load-balancing across any number of servers.

Part 2

Querying RDF data

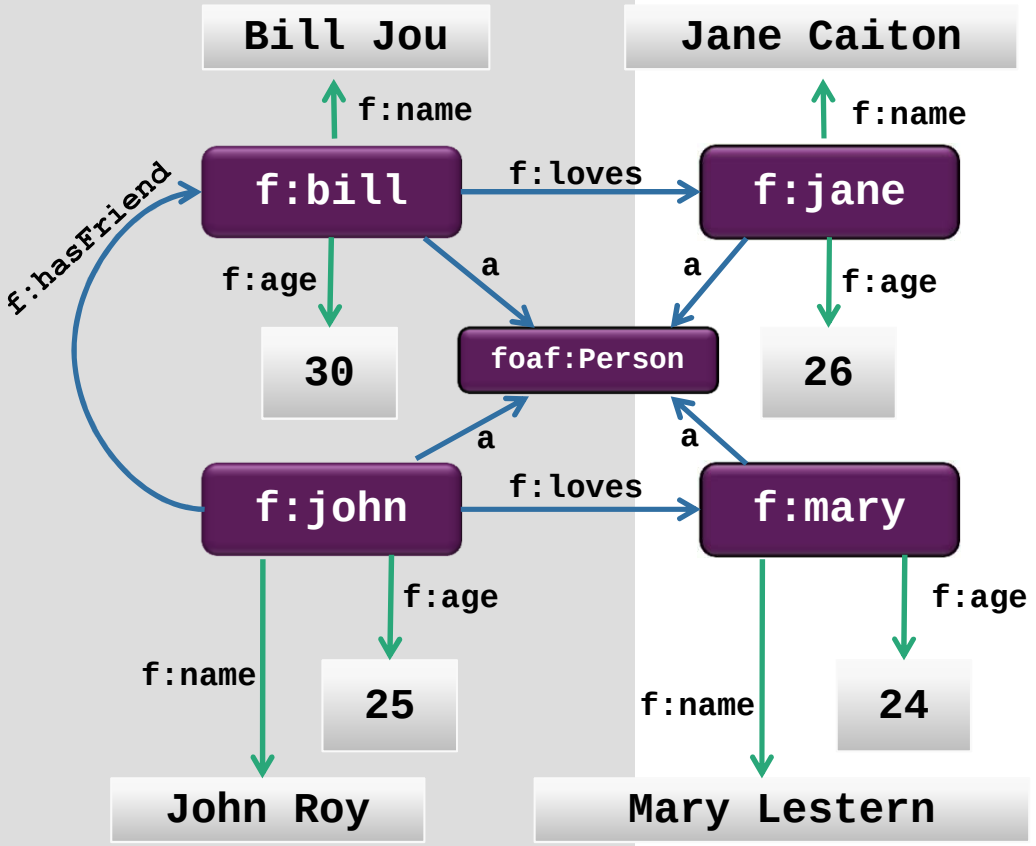
SPARQL: General Form

- 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 (Data Set)
e.g. FROM <http://users.jyu.fi/~olkhriye/itks544/rdf/people.rdf>

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

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

Example data set

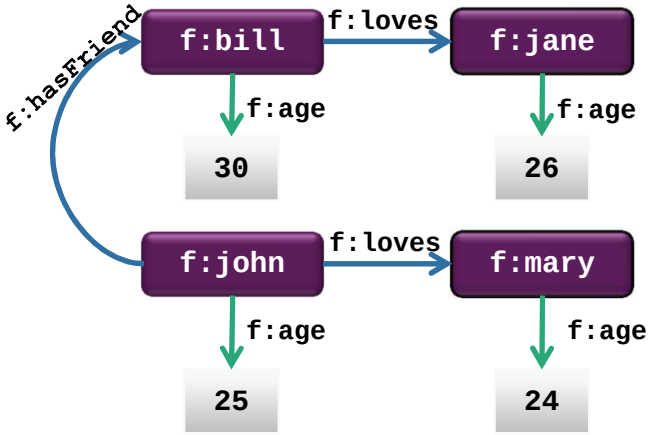
```
@prefix f: <http://example.org#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

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***

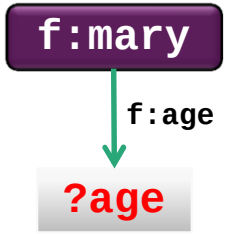
Data



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 }
```



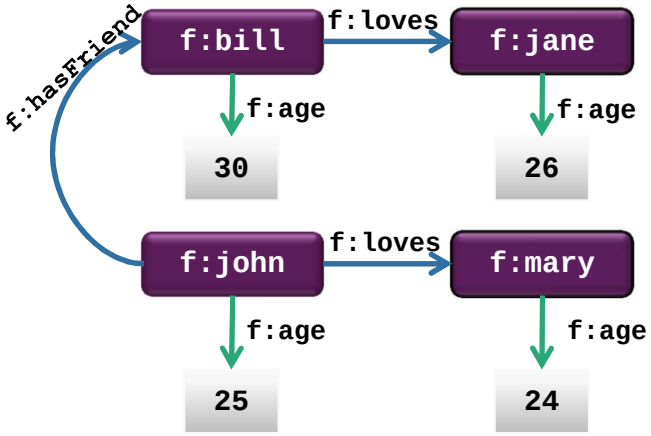
Result

age
24

Simple SPARQL queries (2)

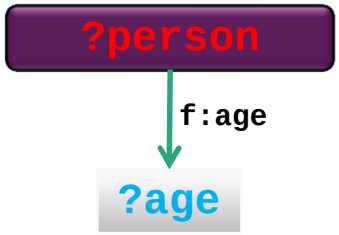
- Show me ***f:age*** of all resources

Data



Query

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



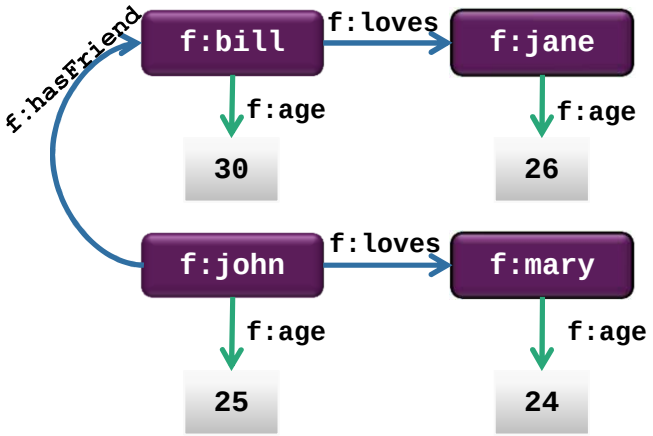
Result

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

Simple SPARQL queries (3)

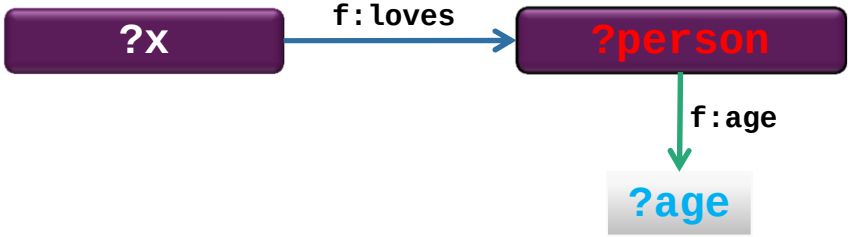
- Show me all things that are loved. Also show me their age (***f:age***)

Data



Query

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



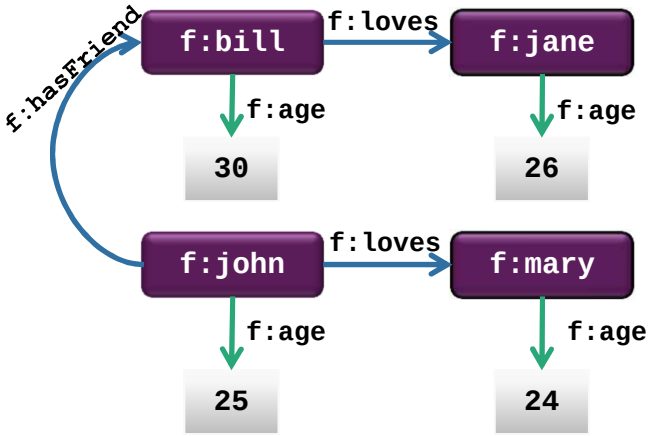
Result

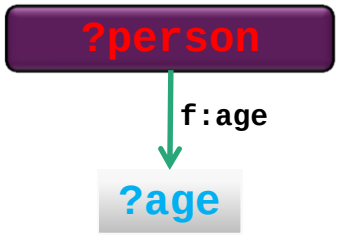
person	age
f:jane	26
f:mary	24

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

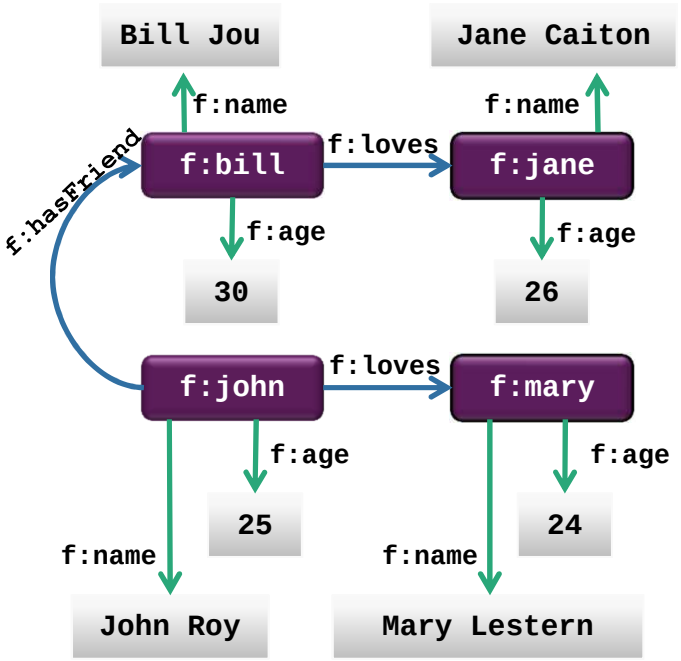
Result

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

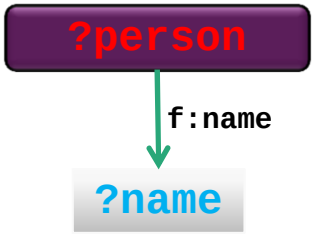
```
FILTER regex(?x, "pattern"[, "flags"])
```

*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

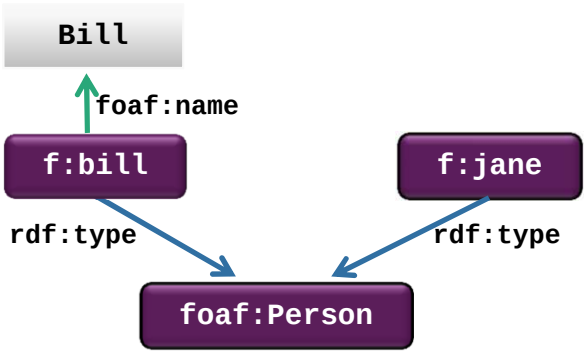


person	name
f:john	John Roy
f:mary	Mary Lestern

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



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}
}
```

Result

person
f:bill

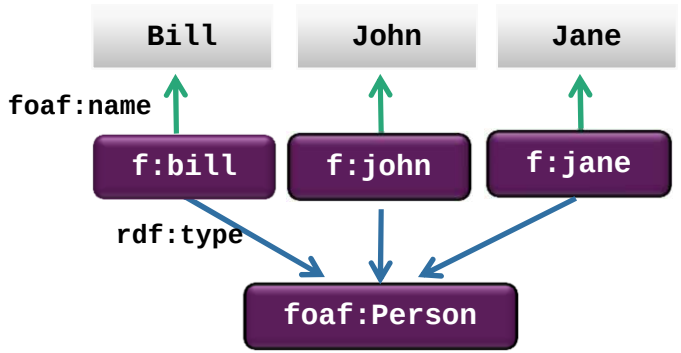
```
SELECT ?person
WHERE {
  ?person rdf:type foaf:Person .
  FILTER NOT EXISTS {?person foaf:name ?name}
}
```

person
f:jane

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" . }
}
```

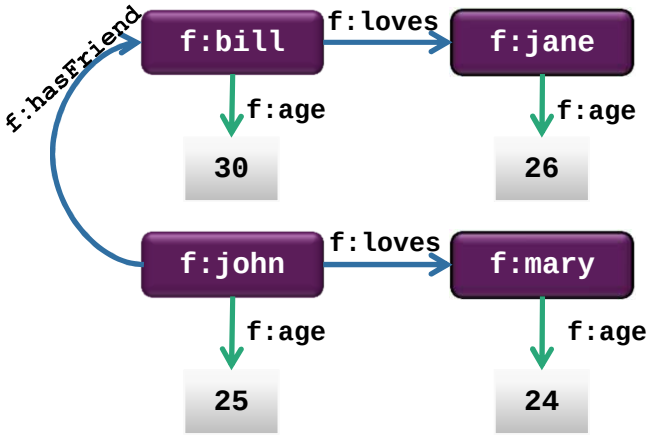
Result

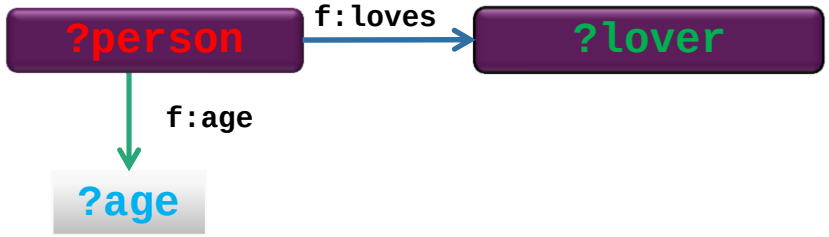
s
f:bill
f:jane

SPARQL: OPTIONAL

- Show me the person and its age (***f:age***). If you have information about that person loving 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}
}
```

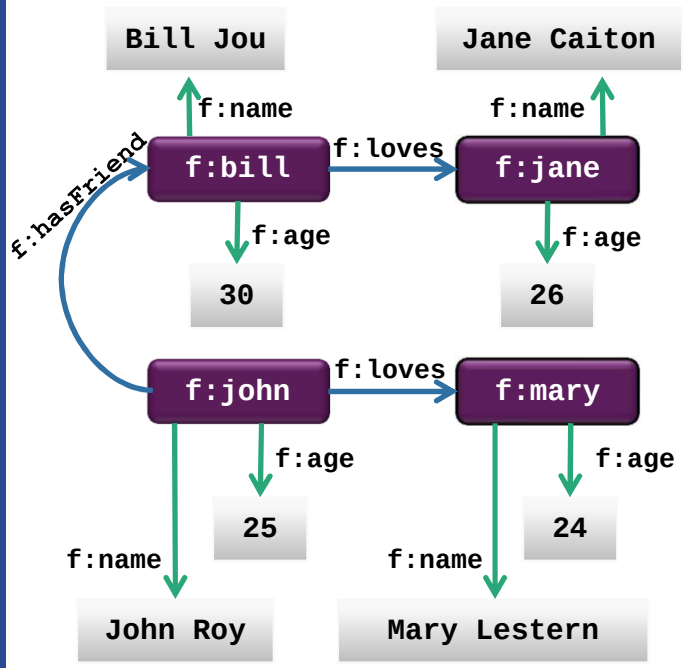
Result

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")}
}
```

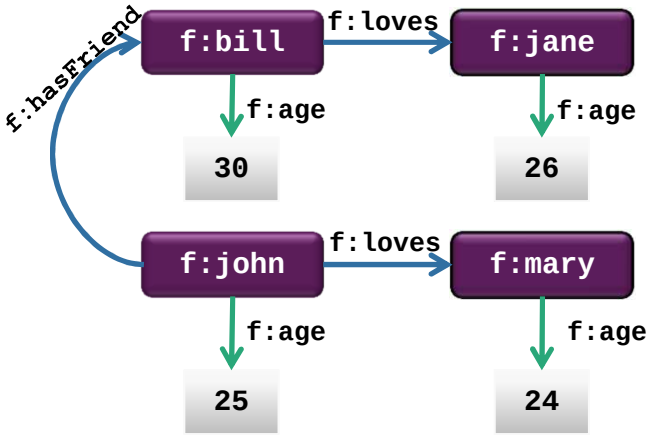
Result

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:

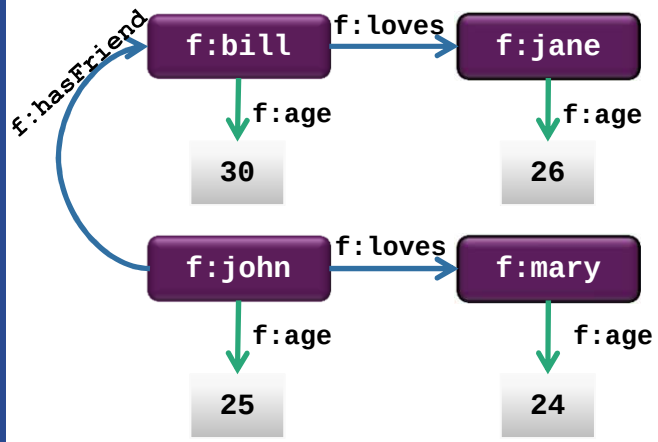
```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {?person f:age ?age }
ORDER BY ?age
```

- 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

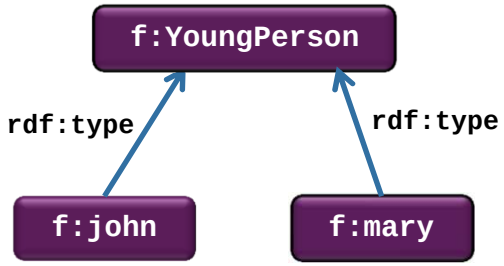
Data



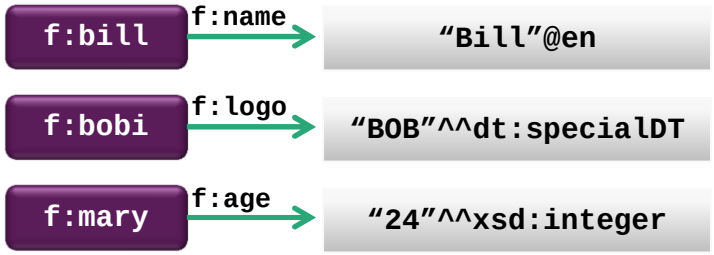
Query

```
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:bill 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 }
```

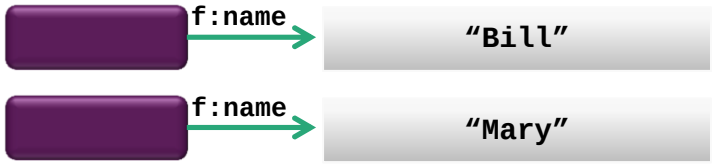
s
f:mary

- 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"

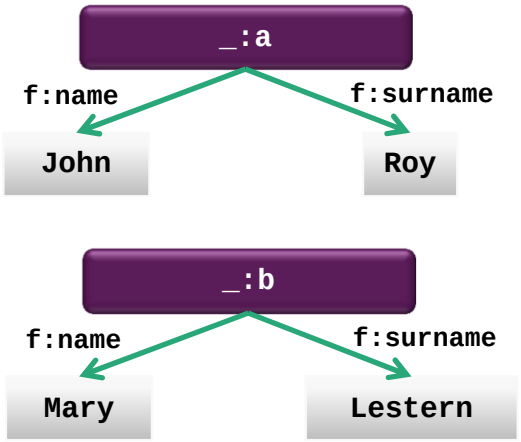
s	name
_:e	"Bill"
_:f	"Mary"

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)
  - 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

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/ties452/rdf/people.rdf

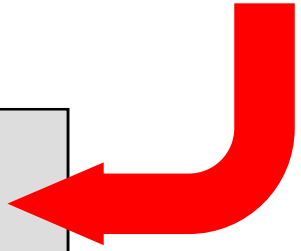
```
j:teacher rdf:type j:EducationJob .
j:seniorResearcher rdf:type j:ResearchJob .
j:juniorResearcher rdf:type j:ResearchJob .
j:professor rdf:type j:ResearchJob, j:EducationJob .
```

http://users.jyu.fi/~olkhriye/ties452/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/ties452/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/ties452/rdf/employment.rdf>
FROM <http://users.jyu.fi/~olkhriye/ties452/rdf/people.rdf>
FROM <http://users.jyu.fi/~olkhriye/ties452/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

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

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/
- Tutorials:
  - http://jena.sourceforge.net/ARQ/Tutorial/index.html
- Validator:
  - http://www.w3.org/RDF/Validator/

Homework

1. Download and start Sesame yourself
2. Create some RDF file
3. Import it to Sesame
4. Try a few queries
5. Play with other tools as well