Aalto University
School of Science

SeCo

SPARQL
Query Language for the Web of Data

CS-E4410 Semantic Web, 27.1.2021

Eero Hyvönen
Aalto University, Semantic Computing Research Group (SeCo) http://seco.cs.aalto.fi
University of Helsinki, HELDIG http://heldig.fi

eero.hyvonen@aalto.fi

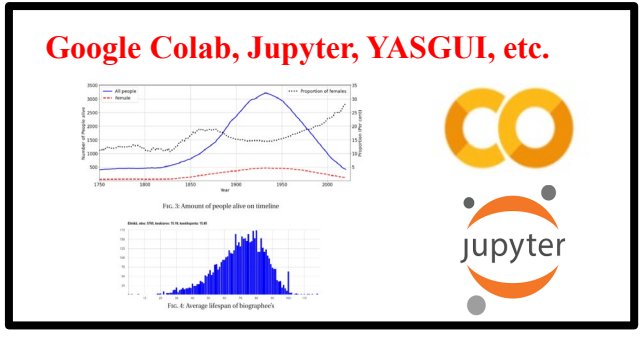
SPARQL

SPARQL
Protocol
and
RDF
Query
Language

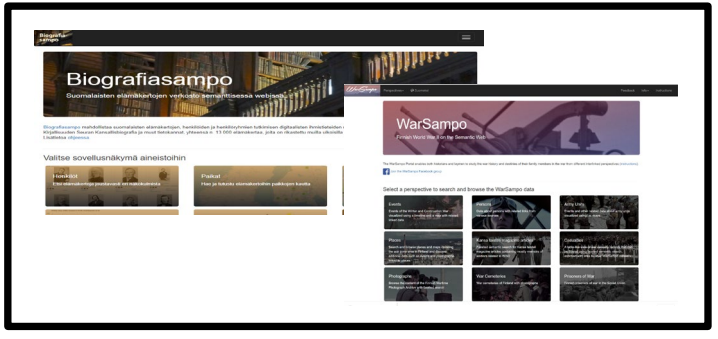
“sparkle”

SPARQL is used for querying Linked Data in SPARQL endpoints

Data analysis tools for research



Applications, such as semantic portals



queries/results

SPARQL {REST API}

SPARQL {REST API}

queries/results

SPARQL ENDPOINT containing RDF graphs

Learning objectives

Learn how to construct SPARQL queries

Learn how to use SPARQL for maintaining RDF graphs

Learn how to apply queries to linked data services

Outline

Using SPARQL for querying Linked Data

Using SPARQL for maintaining Linked Data

Examples of using SPARQL with YASGUI Tool and Google Colab

Introducing SPARQL for Querying Data

Aalto University
School of Science

Department of
Computer Science

SeCo

SPARQL basics

- Query language for 1) RDF data and 2) data management
- W3C recommendation
  - *SPARQL 1.0 recommendation 15.1.2008*
  - *SPARQL 1.1 recommendation 21.3.2013*
- Based on Turtle notation triple patterns, which are matched against the RDF data graph

Basic query form: SELECT

PREFIX

- Simplifies queries syntactically as in Turtle
  - *http://www.domain.com/namespace/property* → *ns:property*

SELECT

- Lists of the wanted result **variables**
- Variables are marked by either “?” or “$” character in front

WHERE

- Query as a graph pattern with variables

FROM

- Optionally: the graph for the query

Example: SELECT query

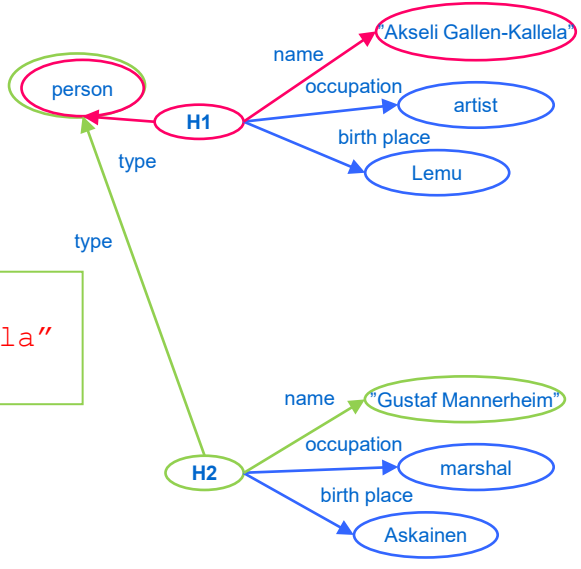
```
PREFIX ns:
   <http://www.domain.com/namespace/>
SELECT ?person ?name
WHERE {
     ?person ns:type ns:person .
     ?person ns:name ?name
}
```

Variables for pattern

Pattern matched

Query result = table of matches

?person	?name
H1	”Akseli Gallen-Kallela”
H2	”Gustav Mannerheim”



Writing query patterns

- Turtle notation with variables
- For each triple you can query for:
  - *Subject*
    - `?person rdf:type ns:Person .`
  - *Predicate*
    - `ns:person23 ?predicate ns:Helsinki .`
  - *Object*
    - `ns:person45 ns:age ?age .`

Example

```
PREFIX ns:
   <http://www.domain.com/namespace/>
SELECT ?person ?name
WHERE {
     ?person rdf:type ns:Person .
     ?person ns:name ?name
}
```

person	name
<ns:person5>	"Matti"
<ns:person2>	"Teppo"
<ns:person13>	"Liisa"

Matching triples with literal values

- Data in SPARQL endpoint
-
```
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
:animal-3 ns:label "cat"@en .
:answer-1 ns:value "42"^^xsd:integer .
:measure-2 ns:datatype "abc"^^dt:specialDatatype .
```
- Language Tags
  - **SELECT ?subject WHERE {?subject ?predicate "cat" }** → empty result
  - **SELECT ?subject WHERE {?subject ?predicate "cat"@en }** → :animal-3
- Numbers
  - **SELECT ?subject WHERE {?subject ?predicate 42}** → :answer-1
- Datatypes
  - **SELECT ?subject**

    WHERE {?subject ?subject "abc"^^dt:specialDatatype } -> :measure-2

Creating new variable values from matched values: CONCAT, BIND

Data:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:givenName   "John" .
_:a  foaf:surname  "Doe" .
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ( CONCAT(?G, " ", ?S) AS ?name )
WHERE  { ?P foaf:givenName ?G ; foaf:surname ?S }
```

Query:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE  {
   ?P foaf:givenName ?G ;
      foaf:surname ?S
   BIND(CONCAT(?G, " ", ?S) AS ?name)
}
```

name
"John Doe"

New variable ?name value is concatenated from matched variable values

Here variable ?name value is concatenated using BIND with the same result

(Example from SPARQL 1.1. W3C Specification)

Restricting solutions in graph patterns using additional FILTER conditions

```
?person ns:age ?age . FILTER(xsd:integer(?age) > 18)
```

```
?object rdf:type ?type . FILTER(?type != ns:Car)
```

```
?book ns:title ?title . FILTER regex(str(?title), "sparql", "i")
```

"i" = case-insensitive

Note:

Type conversions (cast) are needed for strings and numbers

- E.g., str(...), xsd:decimal(...)

Literals can be filtered using regular expressions

- Regex()

Numeric values can be compared

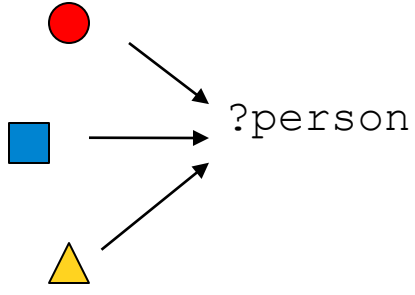
- "<", ">", "!=", "="

There is support for datatypes (xsd:boolean, xsd:dateTime,…)

Combining alternatives: UNION

Matching alternative values for the same variable

```
{?person ns:nationality ns:Finnish}
UNION
{?person ns:nationality ns:Swedish}
UNION
{?person ns:name "John"}
```



OPTIONAL matching of triples

Optional triples that are used only if they match but may not match, too

```
?person rdf:type ns:Person .
?person ns:name ?name .
OPTIONAL {?person ns:age ?age}
```

name	age
” Matti”	
” Teppo”	34
” Liisa”	52

Missing optinal value ?age for “Matti”

Modifying the result set before or after matching

DISTINCT

- Removes duplicate results

ORDER BY

- Defines the order of the result set
- DESC()
- ASC()

LIMIT

- Limits the number of results returned

OFFSET

- Defines the result set to start after the specified number of results

Example

```
PREFIX ns: <http://www.domain.com/namespace/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?field ?length ?width
WHERE {
    ?field rdf:type ns:Field ;
     ns:length ?length ;
     ns:width ?width .
}
ORDER BY ASC(xsd:decimal(?length) *
  xsd:decimal(?width))
```

Results are ordered by their area in an ascending order

Example

```
PREFIX ns: <http://www.domain.com/namespace/>
SELECT DISTINCT ?person ?name ?age
WHERE {
        {?person ns:gender ns:Female ;
         ns:nationality ns:Finnish ;
        ns:name ?name .
         OPTIONAL { ?person ns:age ?age . FILTER(xsd:integer(?age) > 30) }
        }
        UNION
        {?person ns:gender ns:Male ;
        ns:nationality ns:Swedish ;
        ns:name ?name .
        OPTIONAL { ?person ns:age ?age . FILTER(xsd:integer(?age) < 30) }
        }}
ORDER BY ?name
LIMIT 7
```

Example result set

person	name	age
<ns:person86>	"Daniel"	28
<ns:person145>	"Håkan"	
<ns:person23>	"Jaana"	34
<ns:person125>	"Lars"	
<ns:person231>	"Marjatta"	42
<ns:person48>	"Mikael"	23
<ns:person6>	"Tiina"	

• Aggregating values over **solution sets: GROUP BY**

Two organizations affiliate three authors who write four books that have a price

Data:

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

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

Query:

?totalPrices, i.e., sums of ?price for each ?org are returned

Matches prices of books of authors in organizations

The query results are grouped for each ?org separately

Only groups with ?lprice sum > 10 are selected in the final result

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

org1 prices sum: 9+5+7=21
org2 prices sum: 7

Results:

totalPrice
21

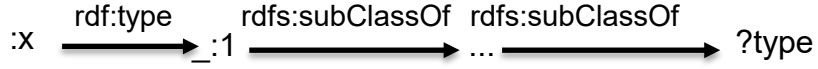
(Example from SPARQL 1.1. W3C Specification)

Many other SPARQL 1.1 features available…

- Path expressions
- Hierarchical subqueries
- Focusing query parts to different graphs in a dataset…
- Federated queries
  - *For querying multiple SPARQL endpoints in one query*

All resources and all their inferred types:

```
{ ?x rdf:type/rdfs:subClassOf* ?type }
```



(Examples from SPARQL 1.1. W3C Specification)

Representing the SELECT query result set: XML format

nameX	nameY	nickY
"Alice"	"Bob"	
"Alice"	"Clare"	"CT"

Result sets can be accessed by a local API but also can be serialized into either XML or an RDF graph. An XML format is described in SPARQL Query Results XML Format, and gives for this example:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="nameX"/>
    <variable name="nameY"/>
    <variable name="nickY"/>
  </head>
  <results>
    <result>
      <binding name="nameX">
        <literal>Alice</literal>
      </binding>
      <binding name="nameY">
        <literal>Bob</literal>
      </binding>
   </result>
    <result>
      <binding name="nameX">
        <literal>Alice</literal>
      </binding>
      <binding name="nameY">
        <literal>Clare</literal>
      </binding>
      <binding name="nickY">
        <literal>CT</literal>
      </binding>
    </result>
  </results>
</sparql>
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
WHERE
  { ?x foaf:knows ?y ;
       foaf:name ?nameX .
    ?y foaf:name ?nameY .
    OPTIONAL { ?y foaf:nick ?nickY }
  }
```

(Example from SPARQL 1.1. W3C Specification)

Other formats: JSON, CSV, TSV

Maintaining RDF graphs with SPARQL

Aalto University
School of Science

Department of
Computer Science

SeCo

SPARQL query (+basic update) forms

- **SELECT**
  - *Presented on previous slides*
- **CONSTRUCT**
  - *Returns an RDF graph specified by a graph template*
- **ASK**
  - *Test if a query pattern has solutions without returning the result set (boolean)*
- **DESCRIBE**
  - *Returns RDF descriptions of the resources found*
- **INSERT** adds new triples into an RDF graph
- **DELETE** removes triples from an RDF graph

Building RDF graphs: CONSTRUCT

Data:

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

_:a  org:employeeName   "Alice" .
_:a  org:employeeId     12345 .

_:b  org:employeeName   "Bob" .
_:b  org:employeeId     67890 .
```

Query:

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

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

Results:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

(Example from SPARQL 1.1. W3C Specification)

Full SPARQL Specification:

https://www.w3.org/TR/sparql11-query/

W3C Recommendation

SPARQL 1.1 Query Language

W3C Recommendation 21 March 2013

This version:
http://www.w3.org/TR/2013/REC-sparql11-query-20130321/
Latest version:
http://www.w3.org/TR/sparql11-query/
Previous version:
http://www.w3.org/TR/2012/PR-sparql11-query-20121108/
Editors:
Steve Harris, Garlik, a part of Experian
Andy Seaborne, The Apache Software Foundation
Previous Editor:
Eric Prud'hommeaux, W3C

Please refer to the **errata** for this document, which may include some normative corrections.

See also **translations**.

Abstract

RDF is a directed, labeled graph data format for representing information in the Web. This specification defines the syntax and semantics of the SPARQL query language for RDF. SPARQL can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. SPARQL contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. SPARQL also supports aggregation, subqueries, negation, creating values by expressions, extensible value testing, and constraining queries by source RDF graph. The results of SPARQL queries can be result sets or RDF graphs.

Status of This Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is one of eleven SPARQL 1.1 Recommendations produced by the SPARQL Working Group:

1. SPARQL 1.1 Overview
2. SPARQL 1.1 Query Language (this document)
3. SPARQL 1.1 Update
4. SPARQL1.1 Service Description
5. SPARQL 1.1 Federated Query
6. SPARQL 1.1 Query Results JSON Format
7. SPARQL 1.1 Query Results CSV and TSV Formats
8. SPARQL Query Results XML Format (Second Edition)
9. SPARQL 1.1 Entailment Regimes
10. SPARQL 1.1 Protocol
11. SPARQL 1.1 Graph Store HTTP Protocol

No Substantive Changes

There have been no substantive changes to this document since the previous version. Minor editorial changes, if any, are detailed in the change log and visible in the color-coded diff.

Please Send Comments

Please send any comments to public-rdf-dawg-comments@w3.org (public archive). Although work on this document by the SPARQL Working Group is complete, comments may be addressed in the errata or in future revisions. Open discussion is welcome at public-sparql-dev@w3.org (public archive).

More information

W3C SPARQL recommendation family

Set of Documents

This document is one of eleven SPARQL 1.1 Recommendations produced by the SPARQL Working Group:

1. SPARQL 1.1 Overview (this document)
2. SPARQL 1.1 Query Language
3. SPARQL 1.1 Update
4. SPARQL1.1 Service Description
5. SPARQL 1.1 Federated Query
6. SPARQL 1.1 Query Results JSON Format
7. SPARQL 1.1 Query Results CSV and TSV Formats
8. SPARQL Query Results XML Format (Second Edition)
9. SPARQL 1.1 Entailment Regimes
10. SPARQL 1.1 Protocol
11. SPARQL 1.1 Graph Store HTTP Protocol

Learn SPARQL by using SPARQL endpoints: just type in queries in forms

Recommended query interface: YASGUI http://yasgui.org

- Syntax highlighting; prefix, property, and class autocompletion

Many endpoints have SPARQL query forms of their own

There are SPARQL endpoints around

https://www.w3.org/wiki/SparqlEndpoints

Page | Discussion | Read | View source | View history | Search

Main Page
Browse categories
Recent changes
Tools
What links here
Related changes
Special pages
Printable version
Permanent link
Page information

SparqlEndpoints

see also: SPARQL

In addition to the list below, Mondeca provides a SPARQL endpoint uptime service which monitors the availability of all SPARQL endpoints that are cataloged in CKAN. A similar service is provided by Vienna University.

Currently Alive SPARQL Endpoints

(alphabetical. let's avoid PoorMansHypertext and in-your-face URIs, please)

Project	status	SPARQL endpoint	Webform	comment
Wikidata	(2017-02-23) alive	endpoint	GUI	See also SPARQL federation input
BBC Programmes and Music	(2010-06-29) alive	endpoint	Ajax based Visual Query Builder	Powered by OpenLink Virtuoso; also supports Faceted Browsing and Exploration
Bio2RDF	(2010-01-07) alive	List of 40 SPARQL endpoints	n/a	uses OpenLink Virtuoso
BioGateway	(2010-01-07) timeout	endpoint	webform	BioGateway provides many parameterizable SPARQL queries, both biological as ontological, on RDF graphs that were optimized for querying. The graphs have relational closures. Empowered by OpenLink Virtuoso.
BBC Backstage (HP Labs)	(2010-01-07) server not responding	endpoint	webform	uses joseki 3
BBC John Peel sessions from DBTune (Centre for Digital Music, Queen Mary, University of London)	(2010-01-07) alive	endpoint	n/a	dbtune aims to gather and interlink music-related information.
BBC playcount data from DBTune (Centre for Digital Music, Queen Mary, University of London)	(2010-01-07) alive	endpoint	n/a	dbtune aims to gather and interlink music-related information.
DailyMed	(2010-01-07) alive	endpoint		a **D2R** endpoint
data.gov	(2010-05-22) alive	endpoint	webform	uses OpenLink Virtuoso
data.gov.uk	(2010-02-04) alive	endpoint	webform	The **data.gov.uk** endpoint
DBLP Bibliography Database published through D2R Server (Freie Universität Berlin)	(2010-01-07) alive	endpoint	webform (Maybe only Firefox)	The DBLP database provides bibliographic information on major computer science journals and conference proceedings.
DBpedia (University of Mannheim, Universität Leipzig, OpenLink Software)	(2010-01-07) alive	endpoint	SNORQL webform (Firefox/Safari/Opera); Ajax based Visual Query Builder	dbpedia.org is a community effort to extract structured information from periodic Wikipedia dumps and to make this information available on the Web. It is served to the public via a live instance of OpenLink Virtuoso, and also offers Faceted Browsing and Exploration
DBpedia-live (Universität Leipzig, University of Mannheim, OpenLink Software)	(2010-01-07) alive	endpoint	webform	Based on, now parallel to, and soon to replace the existing dbpedia.org data sets, DBpedia-Live is constantly updated, based on Wikipedia change-feeds. It is served to the public via a live instance of OpenLink Virtuoso, and also offers Faceted Browsing and Exploration
German DBpedia (AG Corporate Semantic Web, Freie Universität Berlin)	(2008-10-15) alive	endpoint	site	de.dbpedia.org is the German language chapter of DBpedia
DBpedia Live German (AG Corporate Semantic Web, Freie Universität Berlin)	(2008-10-15) alive	endpoint	site	de.dbpedia.org is the German language chapter of DBpedia
Spanish DBpedia (Universidad Autónoma de Madrid, Universidad Politécnica de Madrid, OpenLink Software)	(2011-04-04) alive	endpoint	site	es.dbpedia.org is the Spanish chapter of DBpedia
Diseasome	(2010-01-07) alive	endpoint		a **D2R** endpoint
DoapSpace	(2010-01-07) away	endpoint	webform	This is a highly experimental TurboGears with rdflib triplestore (mysql) SPARQL endpoint.
DrugBank	(2010-01-07) alive	endpoint		a **D2R** endpoint
EEA (European Environment Agency) Semantic				

BookSampo SPARQL endpoint in LDF.fi:

http://www.ldf.fi/dataset/kirjasampo

SPARQL Endpoint: http://ldf.fi/kirjasampo/sparql

Queries are represented using the URI template: `http://ldf.fi/SERVICE/sparql?query=QUERY`

Query Test Form

```
# Find novels written by somebody with name "Waltari"

PREFIX k: <http://www.yso.fi/onto/kaunokki#>
prefix s: <http://www.w3.org/2004/02/skos/core#>
select ?name ?title where {
  ?x a k:romaani .
  ?x k:tekija ?y .
  ?y s:prefLabel ?name .
  FILTER (regex(?name,".*Waltari.*"))
  ?x s:prefLabel ?title .
}
```

Format: XML/HTML

☐ View result in original form in browser (content-type = `text/plain`).

By default (XML/HTML and the "text/plain" check box not checked) the result is presented as an HTML table with links to related resources.

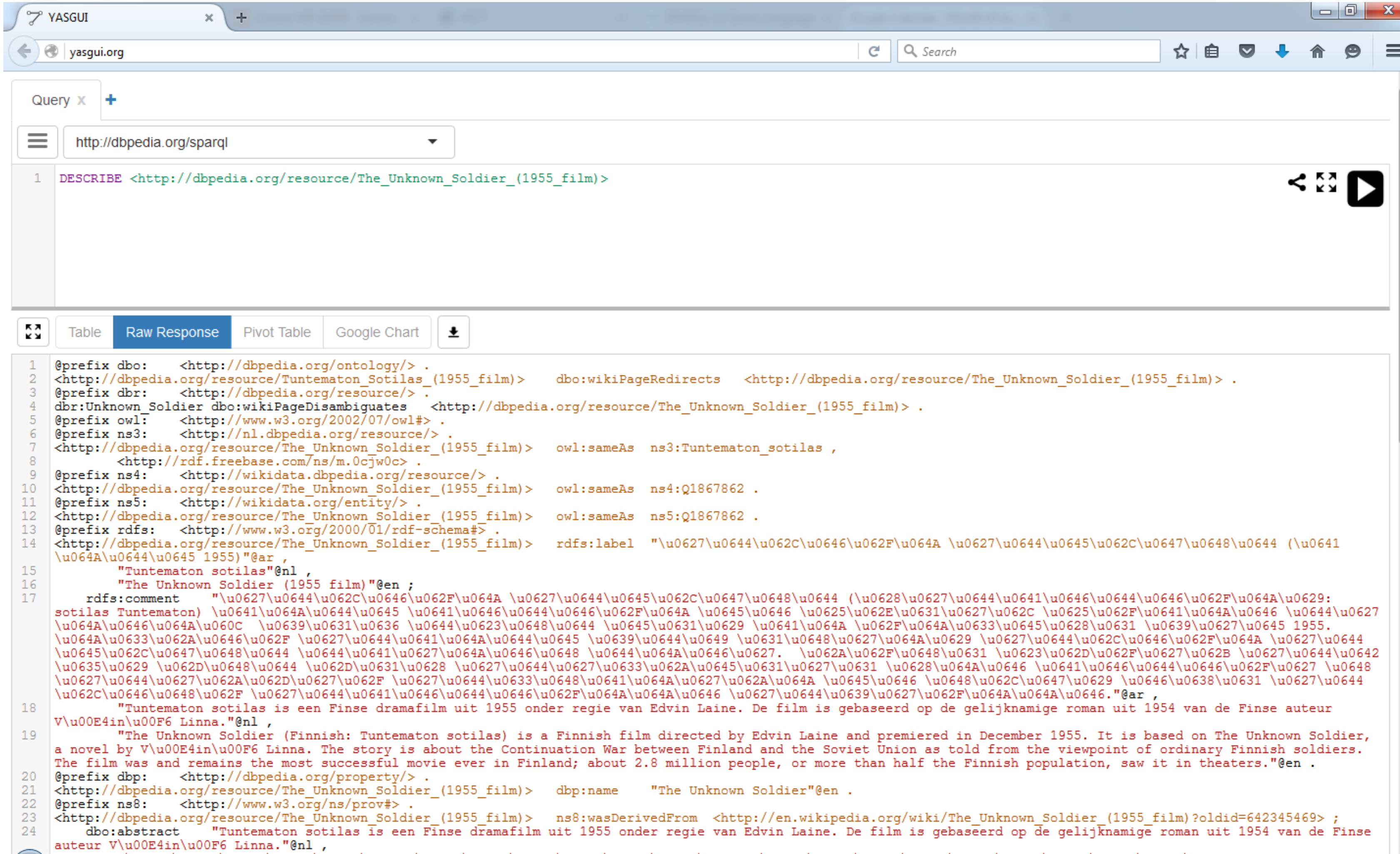
Note: With DESCRIBE/CONSTRUCT queries "Text" format means Turtle, "CSV" and "TSV" formats cannot be used, and checking the "text/plain" check box presents the result in N-Triples.

Query

Examples of Using SPARQL with YASGUI Tool:

https://yasgui.triply.cc/

Getting a Resource Description



SPARQL result in JSON

YASGUI

yasgui.org

Search

Query x +

http://dbpedia.org/sparql

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?name ?title
WHERE {
  ?person a dbo:Writer ;
          rdfs:label ?name .
  FILTER (regex(?name, "Waltari") && langMatches(lang(?name), "en"))
  ?book dbp:author ?person ;
        rdfs:label ?title .
}
```

Table | Raw Response | Pivot Table | Google Chart


```
{ "head": { "link": [], "vars": ["name", "title"] },
  "results": { "distinct": false, "ordered": true, "bindings": [
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "en", "value": "The Egyptian" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "ar", "value": "\u0633\u0646\u0648\u062D\u0649
\u0627\u0644\u0645\u0635\u0631\u064A" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "de", "value": "Sinuhe der \u00C4gypter" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "es", "value": "Sinuh\u00E9, el egipcio" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "fr", "value": "Sinouh\u00E9 l'\u00C9gyptien"
}},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "ja", "value": "\u30A8\u30B8\u30D7\u30C8\u4EBA
(\u5C0F\u8AAC)" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "it", "value": "Sinuhe l'egiziano (romanzo)"
}},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "pl", "value": "Egipcjanin Sinuhe" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "pt", "value": "O Eg\u00EDpcio" }},
    { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }  , "title": { "type": "literal", "xml:lang": "ru", "value": "\u0421\u0438\u043D\u0443\u0445
\u0435, \u0435\u0433\u0438\u043F\u0442\u044F\u043D\u0438\u043D" }} ] } }
```

Same result output as a table

YASGUI

yasgui.org

Search

Query x +

http://dbpedia.org/sparql

```
1  PREFIX dbo: <http://dbpedia.org/ontology/>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX dbp: <http://dbpedia.org/property/>
4  SELECT ?name ?title
5  WHERE {
6    ?person a dbo:Writer ;
7            rdfs:label ?name .
8    FILTER (regex(?name, "Waltari") && langMatches(lang(?name), "en"))
9    ?book dbp:author ?person ;
10         rdfs:label ?title .
11 }
```

Table | Raw Response | Pivot Table | Google Chart

Showing 1 to 10 of 10 entries (in 0.93 seconds)

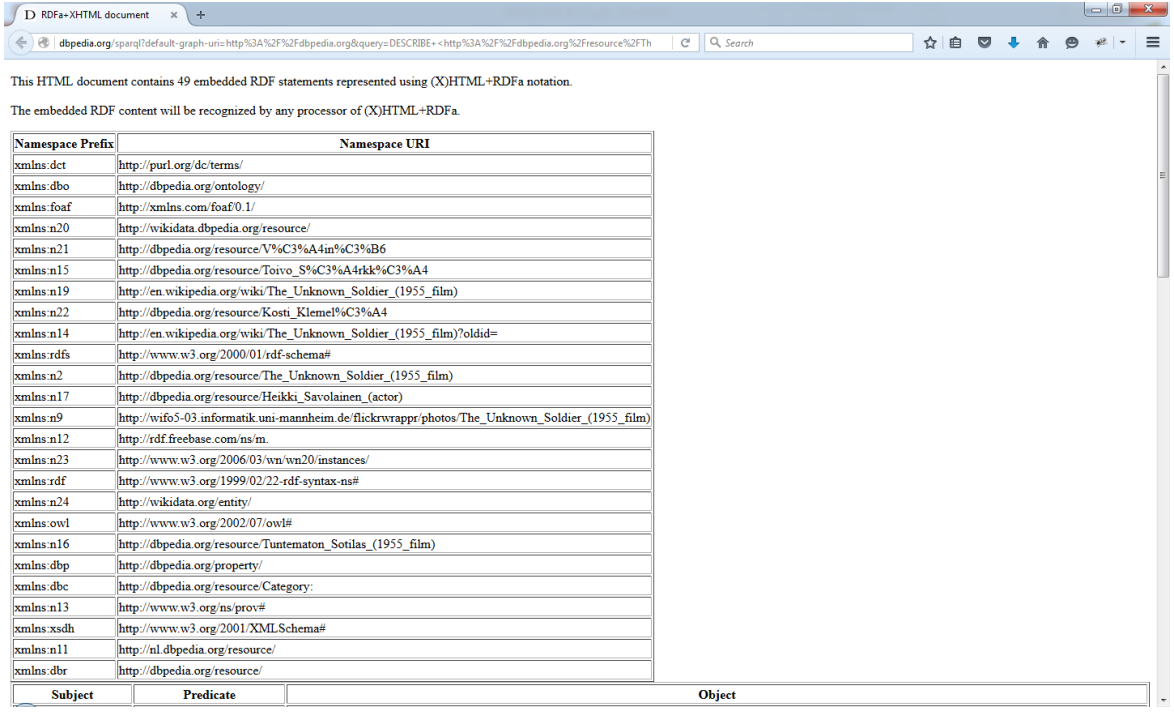
Search: Show 50 entries

	name	title
1	"Mika Waltari"@en	"The Egyptian"@en
2	"Mika Waltari"@en	"سنوحى المصري"@ar
3	"Mika Waltari"@en	"Sinuhe der Ägypter"@de
4	"Mika Waltari"@en	"Sinuhé, el egipcio"@es
5	"Mika Waltari"@en	"Sinouhé l'Égyptien"@fr
6	"Mika Waltari"@en	"エジプト人 (小説)"@ja
7	"Mika Waltari"@en	"Sinuhe l'egiziano (romanzo)"@it
8	"Mika Waltari"@en	"Egipcjanin Sinuhe"@pl
9	"Mika Waltari"@en	"O Egípcio"@pt
10	"Mika Waltari"@en	"Синухе, египтянин"@ru

Using a SPARQL endpoint in JavaScript, Python etc. programs

HTTP API using the endpoint URL, e.g., http://dbpedia.org/sparql

- Query and other parameters as GET/POST parameters
  - *Use URL encoding*



RDFa+XHTML document

dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=DESCRIBE+<http%3A%2F%2Fdbpedia.org%2Fresource%2FTh

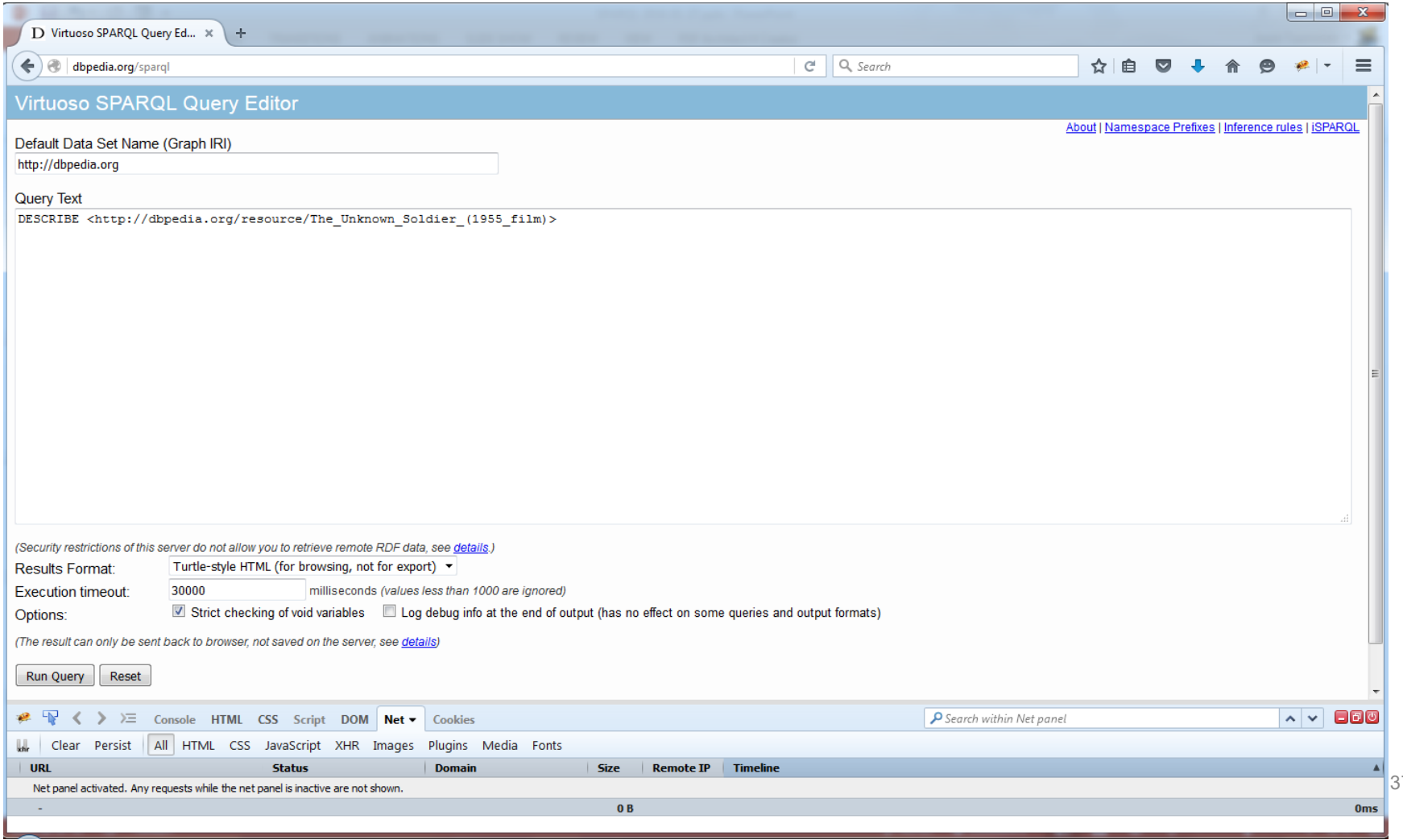
This HTML document contains 49 embedded RDF statements represented using (X)HTML+RDFa notation.

The embedded RDF content will be recognized by any processor of (X)HTML+RDFa.

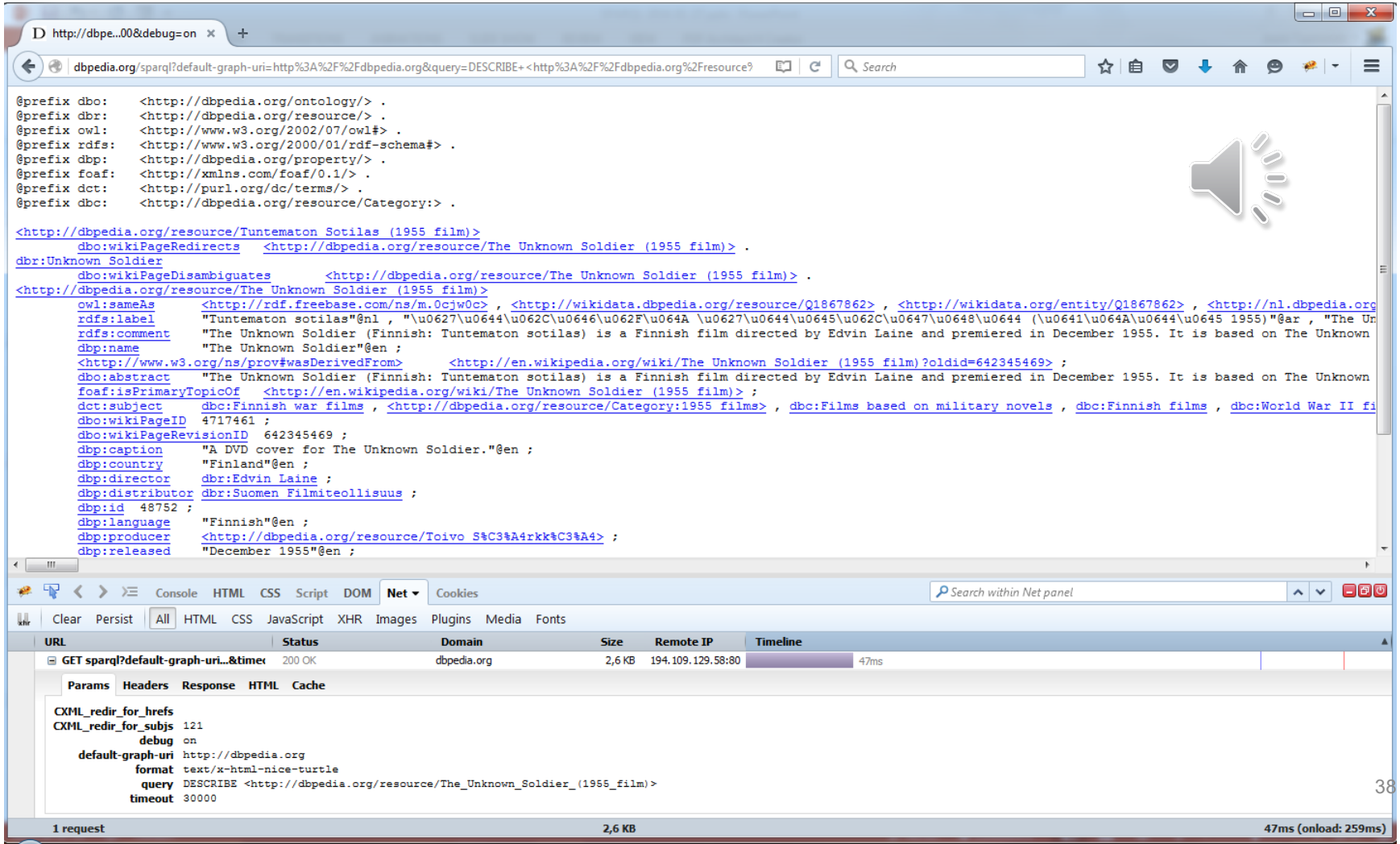
Namespace Prefix	Namespace URI
xmlns:dct	http://purl.org/dc/terms/
xmlns:dbo	http://dbpedia.org/ontology/
xmlns:foaf	http://xmlns.com/foaf/0.1/
xmlns:n20	http://wikidata.dbpedia.org/resource/
xmlns:n21	http://dbpedia.org/resource/V%C3%A4in%C3%B6
xmlns:n15	http://dbpedia.org/resource/Toivo_S%C3%A4rkk%C3%A4
xmlns:n19	http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)
xmlns:n22	http://dbpedia.org/resource/Kosti_Klemel%C3%A4
xmlns:n14	http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)?oldid=
xmlns:rdfs	http://www.w3.org/2000/01/rdf-schema#
xmlns:n2	http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)
xmlns:n17	http://dbpedia.org/resource/Heikki_Savolainen_(actor)
xmlns:n9	http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/photos/The_Unknown_Soldier_(1955_film)
xmlns:n12	http://rdf.freebase.com/ns/m.
xmlns:n23	http://www.w3.org/2006/03/wn/wn20/instances/
xmlns:rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns:n24	http://wikidata.org/entity/
xmlns:owl	http://www.w3.org/2002/07/owl#
xmlns:n16	http://dbpedia.org/resource/Tuntematon_Sotilas_(1955_film)
xmlns:dbp	http://dbpedia.org/property/
xmlns:dbc	http://dbpedia.org/resource/Category:
xmlns:n13	http://www.w3.org/ns/prov#
xmlns:xsdh	http://www.w3.org/2001/XMLSchema#
xmlns:n11	http://nl.dbpedia.org/resource/
xmlns:dbr	http://dbpedia.org/resource/

Subject	Predicate	Object

Example using Firefox / Firebug



http://dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&**query=DESCRIBE+%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2FThe_Unknown_Soldier_%281955_film%29%3E**&format=text%2Fx-html-nice-turtle&CXML_redir_for_subjs=121&CXML_redir_for_hrefs=&timeout=30000&debug=on

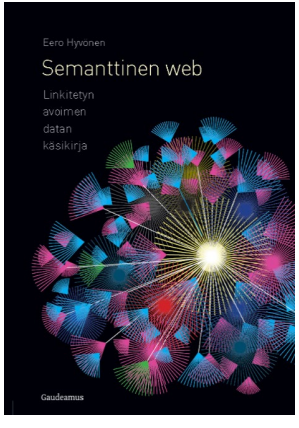


More Information – Questions?

SPARQL online tutorials:

https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial
https://www.stardog.com/tutorials/sparql/
https://jena.apache.org/tutorials/sparql.html

In Finnish



2018

https://www.gaudeamus.fi/semanttinen-web/