

SPARQL

Query Language for the Web of Data

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

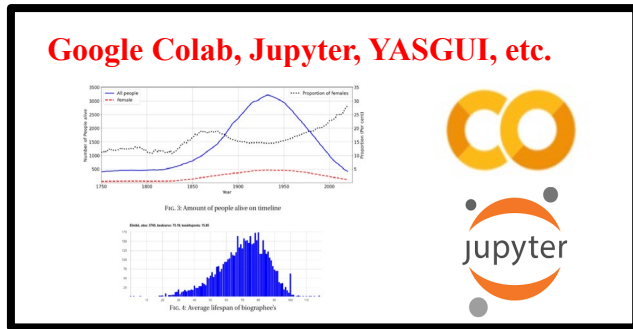
SPARQL Protocol and RDF Query Language

“sparkle”

SPARQL is used for querying Linked Data in SPARQL endpoints

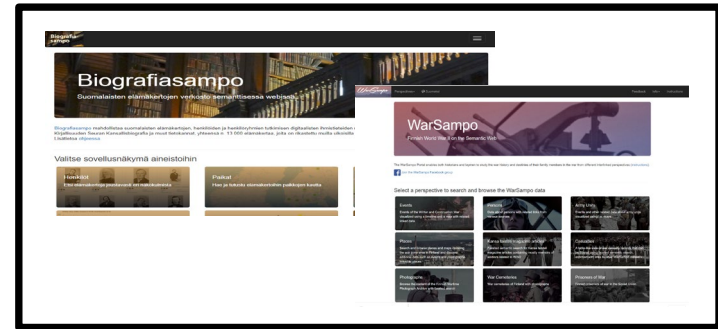
Data analysis tools for research

Google Colab, Jupyter, YASGUI, etc.



The image shows two data analysis tool interfaces. The top one is Google Colab, displaying a line graph with two series: 'All people' (solid blue line) and 'Population of Helsinki' (dashed blue line). The x-axis is 'Year' from 1800 to 2000, and the y-axis is 'Number of People' from 0 to 3000. Below it is a histogram showing the distribution of 'Average lifespan of biographies' from 1800 to 1900. To the right are the logos for CO (Google Colab) and Jupyter.

Applications, such as semantic portals



The image shows a screenshot of a semantic portal interface. The main header is 'Biografiasampo' with the subtitle 'Suomalaisen sivistystieteen tutkimuskeskus'. Below the header is a search bar and a grid of search results. One result is titled 'WarSampo' with a sub-header 'Pöytäkirjat 1800-1900'. The interface is clean and modern, with a dark theme.

queries/results



queries/results



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

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

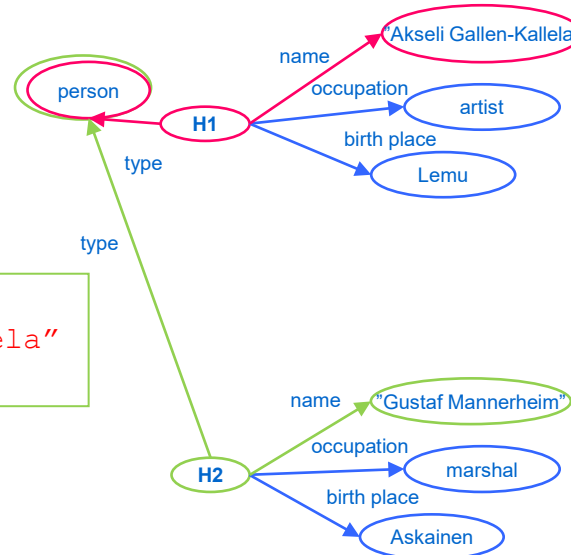
```
PREFIX ns:
  <http://www.domain.com/namespace/>
SELECT ?person ?name
WHERE {
  ?person ns:type ns:person .
  ?person ns:name ?name
}
```

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 }
```

**New variable ?name value is concatenated from matched variable values**

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)
}
```

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

| name       |
|------------|
| "John Doe" |

(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

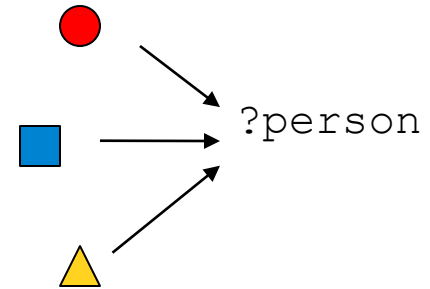
- "<", ">", "!=", "="

### 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 optional 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 .
```

?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

Query:

```
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:

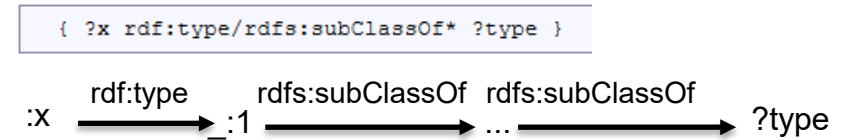
| totalPrice |
|------------|
| 21         |

(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:



(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



# 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/>



## 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](#).

Copyright © 2013 W3C® ([MIT](#), [ERCIM](#), [Keio](#), [Beihang](#)), All Rights Reserved. W3C [liability](#), [trademark](#) and [document use](#) rules apply.

## 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. [SPARQL 1.1 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](mailto: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](mailto: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. [SPARQL 1.1 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.triply.cc> Syntax highlighting; prefix, property, and class autocompletion

**Many endpoints have SPARQL query interfaces of their own**

# There are SPARQL endpoints around

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



Page Discussion

Read View source View history Search

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

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

# 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:

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.

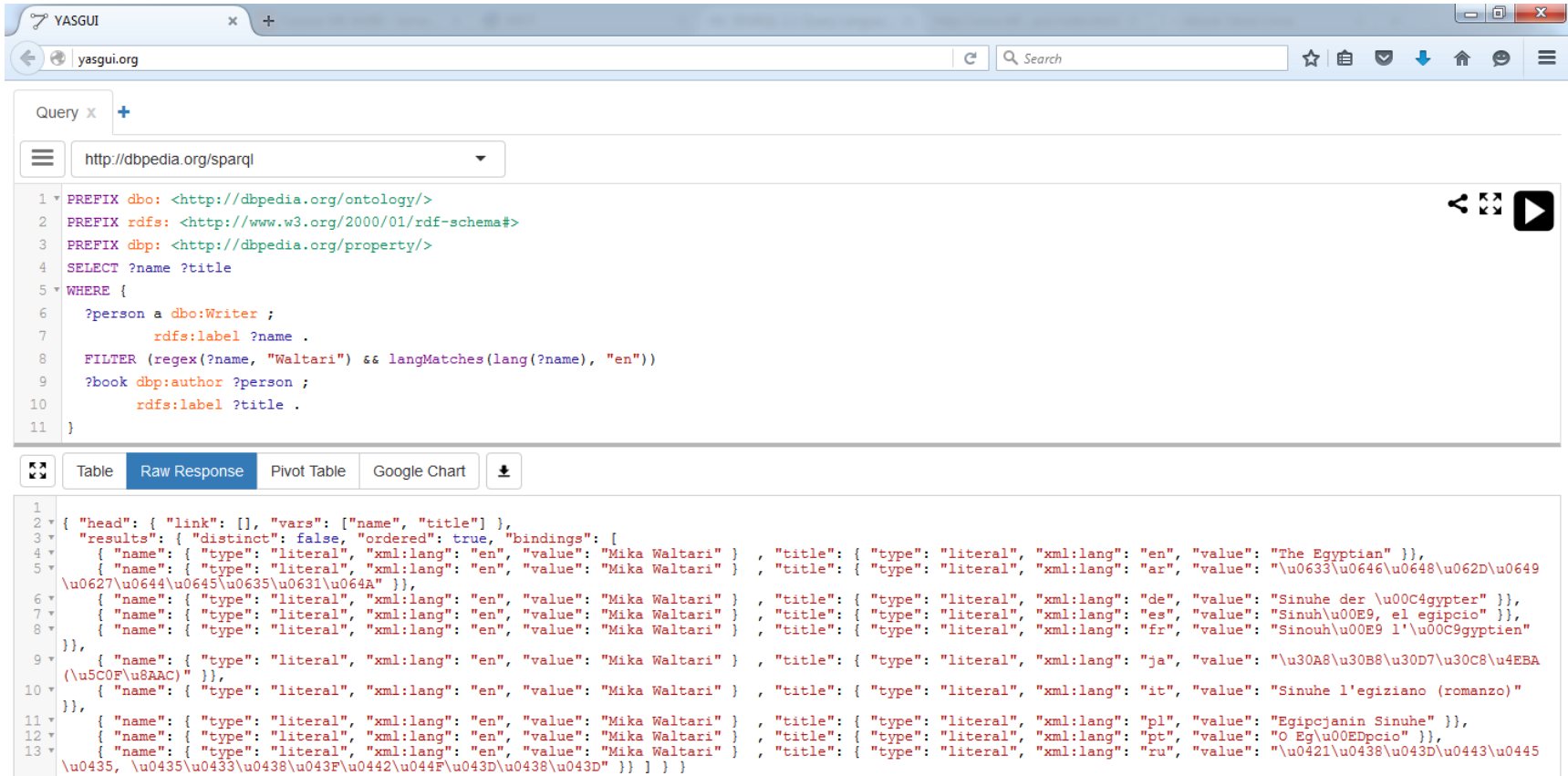
Query

# Examples of Using SPARQL with YASGUI Tool: <https://yasgui.triply.cc/>





# SPARQL result in JSON



The screenshot shows the YASGUI web interface. The browser address bar displays `yasgui.org`. The query input field contains `http://dbpedia.org/sparql`. The query editor shows the following SPARQL query:

```
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 }
```

Below the query editor, there are tabs for "Table", "Raw Response", "Pivot Table", and "Google Chart". The "Raw Response" tab is selected, showing the JSON output of the query:

```
1 {
2 "head": { "link": [], "vars": ["name", "title"] },
3 "results": { "distinct": false, "ordered": true, "bindings": [
4 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "en", "value": "The Egyptian" },
5 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "ar", "value": "\u0633\u0646\u062D\u0649
6 \u0627\u0644\u0645\u0635\u0631\u064A" },
7 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "de", "value": "Sinuhe der \u00C4gypter" },
8 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "es", "value": "Sinuh\u00E9, el egipcio" },
9 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "fr", "value": "Sinuh\u00E9 1'\u00C9gyptien"
10 }},
11 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "ja", "value": "\u30A8\u30B8\u30D7\u30C8\u4EBA
12 (\u5C0F\u8AAC)" },
13 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "it", "value": "Sinuhe l'egiziano (romanzo)"
14 }},
15 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "pl", "value": "Egipcjanin Sinuhe" },
16 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "pt", "value": "O Eg\u00EDpcio" },
17 { "name": { "type": "literal", "xml:lang": "en", "value": "Mika Waltari" }, "title": { "type": "literal", "xml:lang": "ru", "value": "\u0421\u0438\u0443\u0445
18 \u0435\u0433\u0438\u0442\u0438\u044F" }]] }
```

# Same result output as a table

The screenshot shows the YASGUI web interface. At the top, the browser address bar shows 'yasgui.org'. Below the address bar, there is a 'Query' section with a dropdown menu set to 'http://dbpedia.org/sparql'. The main area contains a SPARQL query:

```
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 }
```

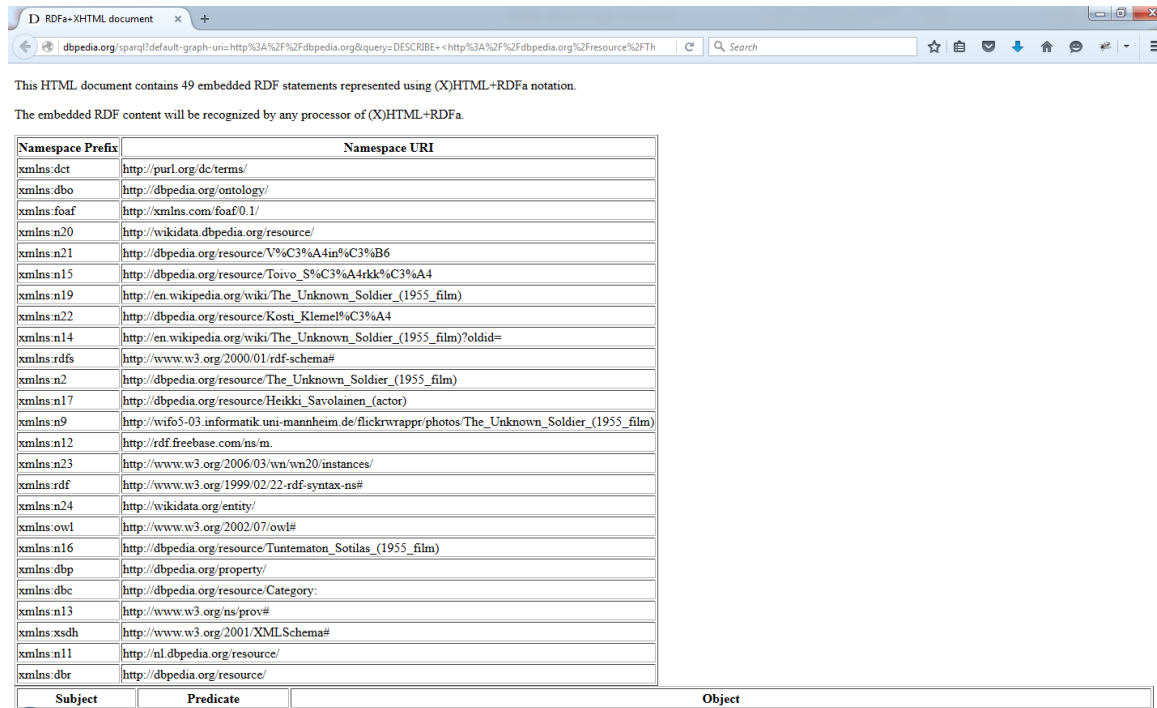
Below the query, there are tabs for 'Table', 'Raw Response', 'Pivot Table', and 'Google Chart'. The 'Table' tab is selected. Below the tabs, it says 'Showing 1 to 10 of 10 entries (in 0.93 seconds)'. On the right, there is a search box and a 'Show 50 entries' dropdown. The results are displayed in a table with two columns: 'name' and 'title'.

	name	title
1	"Mika Waltari"@en	"The Egyptian"@en
2	"Mika Waltari"@en	"السنوحى 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*



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:dc	<a href="http://purl.org/dc/terms/">http://purl.org/dc/terms/</a>
xmlns:dbo	<a href="http://dbpedia.org/ontology/">http://dbpedia.org/ontology/</a>
xmlns:foaf	<a href="http://xmlns.com/foaf/0.1/">http://xmlns.com/foaf/0.1/</a>
xmlns:n20	<a href="http://wikidata.dbpedia.org/resource/">http://wikidata.dbpedia.org/resource/</a>
xmlns:n21	<a href="http://dbpedia.org/resource/V%C3%A4m%C3%B6">http://dbpedia.org/resource/V%C3%A4m%C3%B6</a>
xmlns:n15	<a href="http://dbpedia.org/resource/Toivo_S%C3%A4kk%C3%A4">http://dbpedia.org/resource/Toivo_S%C3%A4kk%C3%A4</a>
xmlns:n19	<a href="http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)">http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)</a>
xmlns:n22	<a href="http://dbpedia.org/resource/Kosti_Klemel%C3%A4">http://dbpedia.org/resource/Kosti_Klemel%C3%A4</a>
xmlns:n14	<a href="http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)?oldid=">http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)?oldid=</a>
xmlns:rdfs	<a href="http://www.w3.org/2000/01/rdf-schema#">http://www.w3.org/2000/01/rdf-schema#</a>
xmlns:n2	<a href="http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)">http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)</a>
xmlns:n17	<a href="http://dbpedia.org/resource/Heikki_Savolainen_(actor)">http://dbpedia.org/resource/Heikki_Savolainen_(actor)</a>
xmlns:n9	<a href="http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/photos/The_Unknown_Soldier_(1955_film)">http://wifo5-03.informatik.uni-mannheim.de/flickrwrappr/photos/The_Unknown_Soldier_(1955_film)</a>
xmlns:n12	<a href="http://rdf.freebase.com/ns/m">http://rdf.freebase.com/ns/m</a>
xmlns:n23	<a href="http://www.w3.org/2006/03/wn/wn20/instances/">http://www.w3.org/2006/03/wn/wn20/instances/</a>
xmlns:rdf	<a href="http://www.w3.org/1999/02/22-rdf-syntax-ns#">http://www.w3.org/1999/02/22-rdf-syntax-ns#</a>
xmlns:n24	<a href="http://wikidata.org/entity/">http://wikidata.org/entity/</a>
xmlns:owl	<a href="http://www.w3.org/2002/07/owl#">http://www.w3.org/2002/07/owl#</a>
xmlns:n16	<a href="http://dbpedia.org/resource/Tuntematon_Sotilas_(1955_film)">http://dbpedia.org/resource/Tuntematon_Sotilas_(1955_film)</a>
xmlns:dbp	<a href="http://dbpedia.org/property/">http://dbpedia.org/property/</a>
xmlns:dbc	<a href="http://dbpedia.org/resource/Category:">http://dbpedia.org/resource/Category:</a>
xmlns:n13	<a href="http://www.w3.org/ns/prov#">http://www.w3.org/ns/prov#</a>
xmlns:xsdh	<a href="http://www.w3.org/2001/XMLSchema#">http://www.w3.org/2001/XMLSchema#</a>
xmlns:n11	<a href="http://nl.dbpedia.org/resource/">http://nl.dbpedia.org/resource/</a>
xmlns:db	<a href="http://dbpedia.org/resource/">http://dbpedia.org/resource/</a>

Subject      Predicate      Object

# Example using Firefox / Firebug

The screenshot shows a Firefox browser window with the Virtuoso SPARQL Query Editor open. The address bar shows `dbpedia.org/sparql`. The page title is "Virtuoso SPARQL Query Editor". The "Default Data Set Name (Graph IRI)" field contains `http://dbpedia.org`. The "Query Text" field contains `DESCRIBE <http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)>`. Below the query text, there are options for "Results Format" (set to "Turtle-style HTML (for browsing, not for export)"), "Execution timeout" (set to 30000 milliseconds), and "Options" (with "Strict checking of void variables" checked and "Log debug info at the end of output" unchecked). At the bottom of the browser window, the Firebug Net panel is visible, showing a table with columns for URL, Status, Domain, Size, Remote IP, and Timeline. The table is currently empty, and the status bar at the bottom indicates "0 B" and "0ms".

Virtuoso SPARQL Query Editor

Default Data Set Name (Graph IRI)  
`http://dbpedia.org`

Query Text  
`DESCRIBE <http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)>`

(Security restrictions of this server do not allow you to retrieve remote RDF data, see [details](#))

Results Format: `Turtle-style HTML (for browsing, not for export)`

Execution timeout: `30000` milliseconds (values less than 1000 are ignored)

Options:  Strict checking of void variables  Log debug info at the end of output (has no effect on some queries and output formats)

(The result can only be sent back to browser, not saved on the server, see [details](#))

Run Query Reset

Net

URL	Status	Domain	Size	Remote IP	Timeline
Net panel activated. Any requests while the net panel is inactive are not shown.					

0 B 0ms

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%2Fxml-nice-turtle&CXML\_redir\_for\_subjs=121&CXML\_redir\_for\_hrefs=&timeout=30000&debug=on

The screenshot shows a web browser displaying the result of an RDF query. The main content area contains a list of prefixes and a detailed RDF description of the film 'The Unknown Soldier (1955 film)'. The Net panel at the bottom shows the request details.

**Prefixes:**

- @prefix dbo: <http://dbpedia.org/ontology/> .
- @prefix dbr: <http://dbpedia.org/resource/> .
- @prefix owl: <http://www.w3.org/2002/07/owl#> .
- @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
- @prefix dbp: <http://dbpedia.org/property/> .
- @prefix foaf: <http://xmlns.com/foaf/0.1/> .
- @prefix dct: <http://purl.org/dc/terms/> .
- @prefix dbc: <http://dbpedia.org/resource/Category:> .

**RDF Description:**

- <http://dbpedia.org/resource/Tuntematon\_Sotilas\_(1955\_film)>
- dbo:wikiPageRedirects <http://dbpedia.org/resource/The\_Unknown\_Soldier\_(1955\_film)> .
- dbr:Unknown\_Soldier
- dbo:wikiPageDisambiguates <http://dbpedia.org/resource/The\_Unknown\_Soldier\_(1955\_film)> .
- <http://dbpedia.org/resource/The\_Unknown\_Soldier\_(1955\_film)>
- owl:sameAs <http://rdf.freebase.com/ns/m.0c3jw0c>, <http://wikidata.dbpedia.org/resource/Q1867862>, <http://wikidata.org/entity/Q1867862>, <http://nl.dbpedia.org/rdfs:label "Tuntematon sotilas"@nl, "\u0627\u0644\u062C\u0646\u0627\u0644 \u0627\u0644\u0627\u0644\u0645\u062C\u0647\u0648\u0644 (\u0641\u0642\u0644\u0645 1955)"@ar, "The Unknown Soldier (Finnish: Tuntematon sotilas) is a Finnish film directed by Edvin Laine and premiered in December 1955. It is based on The Unknown Soldier"@en ; <http://www.w3.org/ns/prov#wasDerivedFrom> <http://en.wikipedia.org/wiki/The\_Unknown\_Soldier\_(1955\_film)?oldid=642345469> ;
- foaf:isPrimaryTopicOf <http://en.wikipedia.org/wiki/The\_Unknown\_Soldier\_(1955\_film)> ;
- dct:subject dbc:Finnish\_war\_films , <http://dbpedia.org/resource/Category:1955\_films> , dbc:Films based on military novels , dbc:Finnish films , dbc:World War II films ;
- dbo:wikiPageID 4717461 ;
- dbo:wikiPageRevisionID 642345469 ;
- dbp:caption "A DVD cover for The Unknown Soldier."@en ;
- dbp:country "Finland"@en ;
- dbp:director dbr:Edvin\_Laine ;
- dbp:distributor dbr:Suomen\_Filmitoimisto ;
- dbp:id 48752 ;
- dbp:language "Finnish"@en ;
- dbp:producer <http://dbpedia.org/resource/Toivo\_S%C3%A4rkk%C3%A4> ;
- dbp:released "December 1955"@en ;

**Net Panel:**

URL	Status	Domain	Size	Remote IP	Timeline
GET sparql?default-graph-uri...&timeo	200 OK	dbpedia.org	2,6 KB	194.109.129.58:80	47ms

**Params:**

- CXML\_redir\_for\_hrefs
- CXML\_redir\_for\_subjs 121
- debug on
- default-graph-uri http://dbpedia.org
- format text/xml-nice-turtle
- query DESCRIBE <http://dbpedia.org/resource/The\_Unknown\_Soldier\_(1955\_film)>
- timeout 30000

1 request 2,6 KB 47ms (onload: 259ms)



## More Information – Questions?

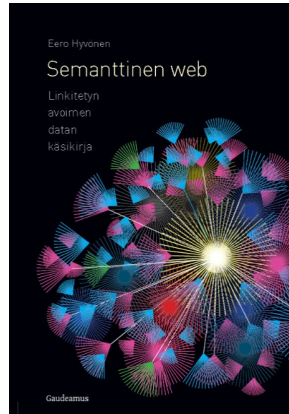
SPARQL online tutorials:

[https://www.wikidata.org/wiki/Wikidata:SPARQL\\_tutorial](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/>