



Aalto University
School of Science



SPARQL

ME-E4300 Semantic Web, 23.1.2019

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

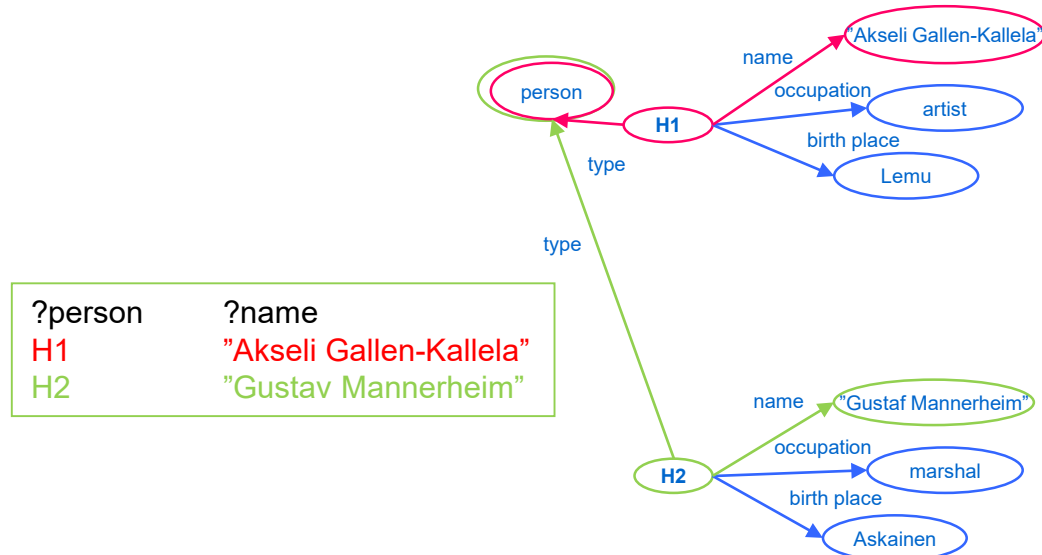
“sparkle”

SPARQL basics

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

Example query

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



Basic query form: SELECT

SELECT

- List of the wanted result variables

FROM

- The graph the query is restricted to
- Optional

WHERE

- Condition expressions as a graph pattern with variables; most restricting conditions first

Variables

- Marked by the use of either “?” or “\$” character

Prefixes

- Eases the writing of queries
 - *http://www.domain.com/namespace/property* → *ns:property*

Writing queries

- Turtle notation with variables
- 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 Literals

- `@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#> .`
`:x ns:p "cat"@en .`
`:y ns:p "42"^^xsd:integer .`
`:z ns:p "abc"^^dt:specialDatatype .`
- Language Tags
 - `SELECT ?v WHERE { ?v ?p "cat" } → Empty result`
 - `SELECT ?v WHERE { ?v ?p "cat"@en } → :x`
- Numbers
 - `SELECT ?v WHERE { ?v ?p 42 } → :y`
- Datatypes
 - `SELECT ?v WHERE { ?v ?p "abc"^^dt:specialDatatype } -> :z`

(Example from
SPARQL 1.1. W3C Specification)

Creating new values with expressions: 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"

(Example from
SPARQL 1.1. W3C Specification)

Restricting Solutions: FILTER

Type conversions (cast) for strings and numbers

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

Literals can be filtered using regular expressions

- `Regex()`

Numeric values can be compared

- “<”, “>”, “!=”, “=”

Other datatypes: `xsd:boolean`, `xsd:dateTime`

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

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

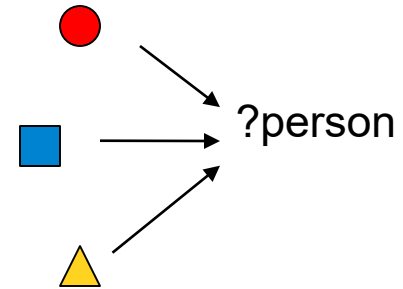
```
?book ns:title ?title . FILTER regex(str(?title), "work", "i")
```

"i" = case-insensitive

Combining alternatives: UNION

Alternative results (into same variable)

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



OPTIONAL

Optional information, which is included in the result set, if the information exists

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

name		age
=====		
"Matti"		
"Teppo"		34
"Liisa"		52

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

Solution sequence (result set) modifiers

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"	

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)

Many other SPARQL 1.1 features available...

- Path expressions
- Aggregating values over solution sets
- Federated queries
 - *Using multiple SPARQL endpoints*
- Subqueries
- Dealing with multiple graphs in datasets
- ...

(Examples from
SPARQL 1.1. W3C Specification)

All resources and all their inferred types:

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

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:

```
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)
```

Results:

totalPrice
21

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](#)

Try SPARQL in various SPARQL endpoints: you can type in queries in forms

For a list see <http://www.w3.org/wiki/SparqlEndpoints>

Linked Data Finland <http://ldf.fi>

- Each dataset has a form for SPARQLing

Linked Open Aalto <http://data.aalto.fi>

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

- Syntax highlighting; prefix, property, and class autocompletion

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

SPARQL result in JSON

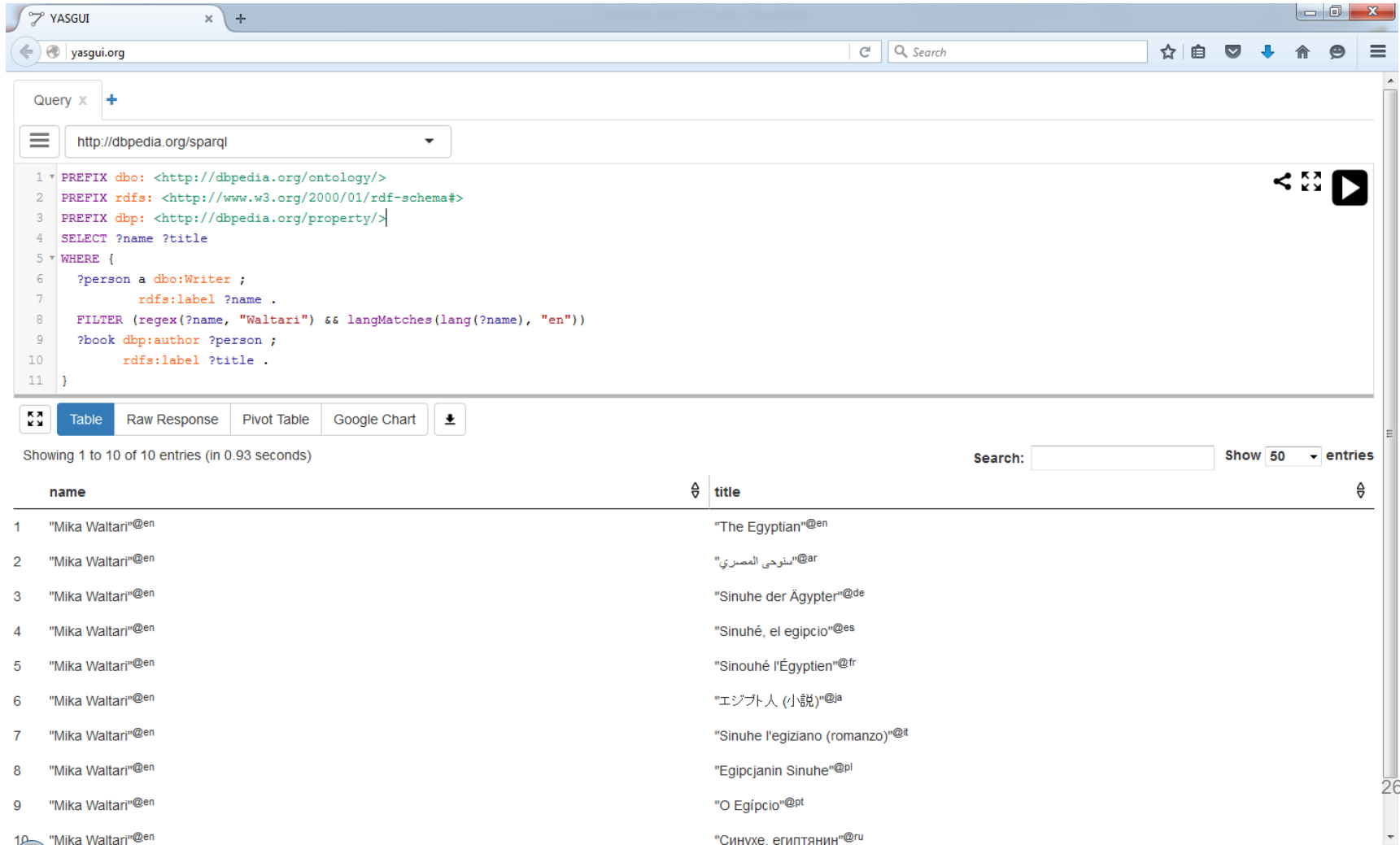
The screenshot shows the YASGUI web interface. The browser address bar displays 'yasgui.org'. The query editor contains the following SPARQL query:

```
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, displaying 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": "Sinouh\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\u043F\u0442\u0430\u0440\u0438" } ] ] }
```

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 }
```

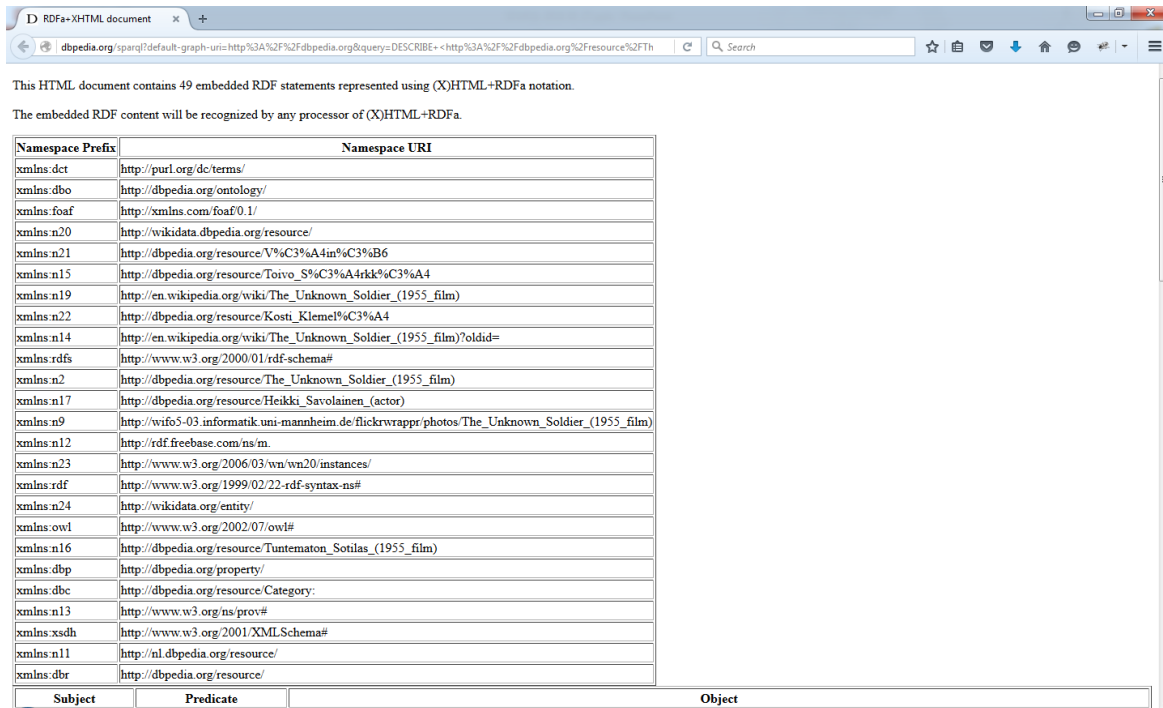
Below the query, there are tabs for 'Table', 'Raw Response', 'Pivot Table', and 'Google Chart'. The 'Table' tab is selected. Below the tabs, it says 'Showing 1 to 10 of 10 entries (in 0.93 seconds)'. On the right, there is a search box and a 'Show 50 entries' dropdown. The results are displayed in a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

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

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



The screenshot shows a web browser window with the address bar containing the URL: `dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=DESCRIBE+<http%3A%2F%2Fdbpedia.org%2Fresource%2FTH`. The page content includes a table of namespace prefixes and their corresponding URIs.

Namespace Prefix	Namespace URI
xmlns:dc	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%A4m%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/

Below the table, there is a table with three columns: Subject, Predicate, and 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)" is `http://dbpedia.org`. The "Query Text" is `DESCRIBE <http://dbpedia.org/resource/The_Unknown_Soldier_(1955_film)>`. The "Results Format" is set to "Turtle-style HTML (for browsing, not for export)". The "Execution timeout" is 30000 milliseconds. The "Options" section has "Strict checking of void variables" checked and "Log debug info at the end of output" unchecked. Below the query editor are "Run Query" and "Reset" buttons. The Firebug Net panel is open at the bottom, showing a table with columns: URL, Status, Domain, Size, Remote IP, and Timeline. The table is currently empty, and the status bar shows "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 response to a SPARQL query. The response is an XML document containing RDF data about the film "The Unknown Soldier (1955 film)". The browser's developer tools are open, showing the "Response" tab with the XML content and the "Params" tab with the query parameters.

Response (XML):

```
<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 Un
  rdfs:comment "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
  dbp:name "The Unknown Soldier"@en ;
  <http://www.w3.org/ns/prov#wasDerivedFrom> <http://en.wikipedia.org/wiki/The_Unknown_Soldier_(1955_film)?oldid=642345469> ;
  dbo:abstract "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
  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 fi
  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 ;
```

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
```

Request Summary:

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

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

Examples & practice in course assignments