Προγραμματισμός Σημασιολογικού Ιστού

Ενότητα 9: Εισαγωγή στη SPARQL – Μέρος Β': Ερωτήματα σε SPARQL endpoints

Μ.Στεφανιδάκης

29-5-2018

Ερωτήματα σε SPARQL endpoints

- Η πραγματική αξία της SPARQL φαίνεται όταν απευθύνουμε ερωτήματα σε endpoints
  - Μέχρι τώρα η επεξεργασία του ερωτήματος γινόταν στον υπολογιστή μας
  - Σε σετ δεδομένων αποθηκευμένο τοπικά
- Ένα SPARQL endpoint είναι μια υπηρεσία web
  - Ένας server που δέχεται το ερώτημα SPARQL από την εφαρμογή μας (client)
  - εκτελεί την κατάλληλη λειτουργία σε ένα σετ δεδομένων RDF (και όχι μόνο RDF)
  - και επιστρέφει την απάντηση μέσω του πρωτοκόλλου HTTP
- Ένα καλό site για αναζήτηση SPARQL endpoints είναι το Datahub

Μην ξεχνάτε όμως

- Παρά τη συναρπαστική ιδέα να χρησιμοποιείτε σημασιολογικά δεδομένα από διάφορες πηγές
  - Με εφαρμογές που δεν φτιάχτηκαν ειδικά για τα δεδομένα αυτά
  - Αλλά μπορούν να προσαρμόσουν τη συμπεριφορά τους δυναμικά
- Υπάρχουν αρκετά εμπόδια:
  - Η διαθεσιμότητα των endpoints δεν είναι δεδομένη ανά πάρα στιγμή
    - Τα περισσότερα endpoints συντηρούνται εθελοντικά-μη κερδοσκοπικά!
  - Η απόδοση δεν είναι εγγυημένη ούτε σταθερή
    - Μη θεωρείτε δεδομένο ότι το endpoint θα απαντήσει σε πολύπλοκα ερωτήματα (timeout)
  - Το σχήμα των παρεχόμενων δεδομένων μπορεί να αλλάξει με τον χρόνο
    - Να αλλάξει το χρησιμοποιούμενο λεξιλόγιο RDF

SPARQL και HTTP

- Το πρότυπο της SPARQL ορίζει τον τρόπο αποστολής της ερώτησης και τη μορφή των αποκρίσεων
- Χρησιμοποιώντας τη μέθοδο GET ή POST
- Για την αποστολή παραμέτρων, η σπουδαιότερη εκ των οποίων είναι η query που ισούται με το κείμενο της ερώτησης SPARQL
  - Ακριβώς όπως ο browser στέλνει δεδομένα από φόρμες
  - Το κείμενο της ερώτησης SPARQL πρέπει να κωδικοποιηθεί με % (url-encoded)
- Στα παραδείγματά μας θα χρησιμοποιήσουμε το εργαλείο `curl` (αποστολή μέσω POST) ή προγράμματα σε Python (αποστολή μέσω GET)

Παράδειγμα ερωτήματος

- Ετοιμάστε αρχείο test.rq με την εξής ερώτηση SPARQL:
  - “ταινίες με τον Τζακ Νίκολσον ως ηθοποιό και η διάρκειά τους”

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>

SELECT ?fname ?ftime WHERE {
?film movie:actor
    <http://data.linkedmdb.org/resource/actor/29704> .
?film movie:runtime ?ftime .
?film rdfs:label ?fname .
}
```

- Στείλτε (μέσω του jupyterhub) στο http://data.linkedmdb.org/sparql:

```
!curl --data-urlencode query@your_file.rq
        http://data.linkedmdb.org/sparql
```

Απάντηση σε SELECT (XML)

```
<?xml version="1.0"?>
<sparql
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xs="http://www.w3.org/2001/XMLSchema#"
    xmlns="http://www.w3.org/2005/sparql-results#" >
  <head>
    <variable name="fname"/>
    <variable name="ftime"/>
  </head>
  <results>
    <result>
      <binding name="fname">
        <literal>Batman</literal>
      </binding>
      <binding name="ftime">
        <literal>126</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Ζητώντας εναλλακτικό περιεχόμενο

- Η απάντηση σε μορφή XML είναι αρκετά περίπλοκη...
  - Κληρονομιά από την εποχή της SPARQL 1.0
  - Σημαντική όμως γιατί πολλά endpoints τη χρησιμοποιούν ακόμα και σήμερα!
- Υπάρχουν και πιο μοντέρνες μορφές απάντησης σε ερωτήματα SELECT
  - JSON (και πιο σπάνια, CSV και TSV)
  - Πώς θα τα ζητήσετε;
- Μέσω του μηχανισμού διαπραγμάτευσης επιστρεφόμενου περιεχομένου (content negotiation) του HTTP

Διαπραγμάτευση Επιστρεφόμενου Περιεχομένου

- Το πρωτόκολλο HTTP ορίζει ότι ένας client μπορεί να ζητήσει τα επιστρεφόμενα δεδομένα σε διαφορετική μορφή από την εξ'ορισμού (default)
  - Δίνοντας την κατάλληλη επικεφαλίδα Accept: της αίτησης
    - Θυμηθείτε όμως ότι ο server έχει δικαίωμα να στείλει την εξ'ορισμού μορφή
    - Όταν δεν μπορεί να ικανοποιήσει το αίτημά σας
  - Θεωρητικά ο server υποδεικνύει αν ένα έγγραφο είναι διαθέσιμο σε πολλές μορφές (επικεφαλίδα Vary: της απόκρισης)
    - Πρακτικά, δεν τηρείται με συνέπεια
- Παράδειγμα:

```
curl --data-urlencode query@test.rq
        http://data.linkedmdb.org/sparql
                -H 'Accept: application/sparql-results+json'
```

Απάντηση σε SELECT (JSON)


```
{
  "head": {
    "vars": [ "fname" , "ftime" ]
  } ,
  "results": {
    "bindings": [
      {
        "fname": { "type": "literal" , "value": "Batman" },
        "ftime": { "type": "literal" , "value": "126" }
      } ,
      {
        "fname": { "type": "literal" , "value": "Chinatown" },
        "ftime": { "type": "literal" , "value": "131" }
      } ,
      {
        "fname": { "type": "literal" , "value": "Easy Rider" }
        "ftime": { "type": "literal" , "value": "94" }
      }
    ]
  }
}
```

Επεξεργασία απάντησης SELECT με Python

- Στα παραδείγματα που ακολουθούν
  - Χρησιμοποιούνται μόνο modules από τη standard library
    - Βολικό για γρήγορες δοκιμές και όταν δεν μπορείτε να έχετε 3rd-party modules!
- Σε πιο “σοβαρή” εφαρμογή θα μπορούσατε
  - Να χρησιμοποιήσετε το module Requests αντί της urllib
  - Να ψάξετε για SPARQL Clients στο pypi.python.org
  - Συμβουλή: δεν χρειάζονται 3rd-party modules για τις πολύ απλές περιπτώσεις!

Παράδειγμα Python με XML: 1.Λήψη

```
from urllib.request import urlopen
from urllib.parse import urlencode
import xml.etree.ElementTree as ET # to analyze xml results re


endpoint = "http://data.linkedmdb.org/sparql?"

sparqlq = """
 ... κείμενο ερωτήματος SPARQL ...
"""


# params sent to server
params = { 'query': sparqlq }
# create appropriate param string
paramstr = urlencode(params)

# send GET http request object with params appended
page = urlopen(endpoint+paramstr)
# get results and close
text = page.read().decode('utf-8')
page.close()
```

Παράδειγμα Python με XML: 2.Χειρισμός

```
# the default sparql results namespace - used by ET
NS = '{http://www.w3.org/2005/sparql-results#}' # note the {}!

# create a tree element from response
tree = ET.fromstring(text)^^I# this is root element, 'sparql'

# iterate over result sets
for result in tree.findall(NS+'results/'+NS+'result'):
        # iterate over all bindings of each result
        for binding in result.findall(NS+'binding'):
            bname = binding.get("name")^^I# what column?
            bcontent = binding[0]^^I# 1st child of <binding>

            print(bname,bcontent.text)
```

(κατεβάστε τον κώδικα του παραδείγματος)

Παράδειγμα Python με JSON: 1.Λήψη

```
from urllib.request import urlopen,Request
from urllib.parse import urlencode
import json

endpoint = "http://data.linkedmdb.org/sparql?"
sparqlq = """
 ... κείμενο ερωτήματος SPARQL ...
"""
# params sent to server
params = { 'query': sparqlq }
# create appropriate param string
paramstr = urlencode(params)

# create GET http request object with params appended
req = Request(endpoint+paramstr)
# request specific content type
req.add_header('Accept','application/sparql-results+json')
# dispatch request
page = urlopen(req)
# get results and close
text = page.read().decode('utf-8')
page.close()
```

Παράδειγμα Python με JSON: 2.Χειρισμός

```
# convert to json object
jso = json.loads(text)

# iterate over results
for binding in jso['results']['bindings']:
    # for every column in binding
    for bname,bcontent in binding.items():
        print(bname,bcontent['value'])
```

(κατεβάστε τον κώδικα του παραδείγματος)

Η σειρά σας!

- Επισκεφτείτε το site Linked Movie Database και δείτε το παράδειγμα αναπαράστασης της πληροφορίας ενός φίλμ σε τριάδες RDF
- Στη συνέχεια, τροποποιήστε το προηγούμενο ερώτημα έτσι ώστε να λαμβάνετε τις ταινίες του Τζακ Νίκολσον και την ημερομηνία κυκλοφορίας τους, ταξινομημένες με φθίνουσα ημερομηνία
  - Το URI του ηθοποιού είναι `<http://data.linkedmdb.org/resource/actor/29704>`
- Δοκιμάστε πρώτα με το curl και μετά μέσω Python!

Προαιρετικά ταιριάσματα (OPTIONAL)

- Δοκιμάστε το εξής ερώτημα :

```
PREFIX lmdbm: <http://data.linkedmdb.org/resource/movie/>
PREFIX lmdba: <http://data.linkedmdb.org/resource/actor/>
PREFIX lmdbf: <http://data.linkedmdb.org/resource/film/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?film ?place WHERE {
  ?film lmdbm:actor lmdba:29704 .
  ?film foaf:based_near ?place .
}
```

- Γιατί δεν επιστρέφονται όλα τα φιλμ όπως πριν;
  - Πολλά δεν συνδέονται με `foaf:based_near`, άρα το ταίριασμα αποτυγχάνει
- Κι αν θέλαμε το ?film ακόμα κι όταν δεν υπάρχει ?place;

Η χρήση του OPTIONAL

- Δοκιμάστε τώρα το ερώτημα:

```
PREFIX lmdbm: <http://data.linkedmdb.org/resource/movie/>
PREFIX lmdba: <http://data.linkedmdb.org/resource/actor/>
PREFIX lmdbf: <http://data.linkedmdb.org/resource/film/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?film ?place WHERE {
  ?film lmdbm:actor lmdba:29704 .
  OPTIONAL {
    ?film foaf:based_near ?place .
  }
}
```

- Σχετικά με το OPTIONAL:
  - Μπορείτε να έχετε πολλαπλά OPTIONAL, εξετάζονται με τη σειρά που τα δίνετε
  - Σε περίπτωση μη ταιριάσματος, κάποιες μεταβλητές μένουν χωρίς τιμή (unbound)

Η άρνηση (negation) στη SPARQL 1.0

- Ψάξτε για ό,τι δεν ταιριάζει:

```
PREFIX lmdbm: <http://data.linkedmdb.org/resource/movie/>
PREFIX lmdba: <http://data.linkedmdb.org/resource/actor/>
PREFIX lmdbf: <http://data.linkedmdb.org/resource/film/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?film WHERE {
  ?film lmdbm:actor lmdba:29704 .
  OPTIONAL {
    ?film foaf:based_near ?place .
  }
  FILTER (!bound(?place))
}
```

- Συνδυασμός OPTIONAL και FILTER(!bound()):
  - Φίλμ χωρίς σύνδεση με τοποθεσία

Αναζήτηση με κανονικές εκφράσεις

- Θα έχετε ήδη καταλάβει ένα σημαντικό πρόβλημα:
  - Πώς θα μάθετε τα URIs που αντιστοιχούν σε συγκεκριμένα ονόματα;
  - Αν ξέρατε ακριβώς πώς αναφέρεται ένα string σε μια τριάδα θα μπορούσατε να γράψετε:

```
SELECT ?uri WHERE {
  ?uri rdfs:label "Agent Smith" .
}
```

- Πώς θα ψάξετε όμως επιμέρους λέξεις; Οπουδήποτε μέσα σε ένα string;
  - Μπορείτε να χρησιμοποιήσετε κανονικές εκφράσεις σε συνδυασμό με το FILTER
  - Η αναζήτηση όμως μπορεί να είναι αργή σε μεγάλα σετ δεδομένων...
    - Με εξαίρεση τις αναζητήσεις προθέματος (prefix) που μπορούν να εκμεταλλευτούν συνήθως κάποιο ευρετήριο

FILTER και regex()

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX lmdba: <http://data.linkedmdb.org/resource/actor/>
PREFIX lmdbf: <http://data.linkedmdb.org/resource/film/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?actor ?aname WHERE {
  ?actor a movie:actor .
  ?actor movie:actor_name ?aname.
  FILTER (regex(?aname,"^kevin","i"))
}
```

- Το πρώτο όρισμα της regex() πρέπει να είναι string
- Το δεύτερο όρισμα είναι η κανονική έκφραση με τη συνήθη σύνταξη
- Το τρίτο όρισμα είναι προαιρετικά flags, όπως το ”i” (case insensitive)

Επιλογή γλώσσας

- Δοκιμάστε το εξής ερώτημα στο endpoint http://fr.dbpedia.org/sparql:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia-fr: <http://fr.dbpedia.org/resource/>
SELECT ?label WHERE {
  dbpedia-fr:Corfou rdfs:label ?label .
}
```

- Πώς θα επιλέξετε τη γλώσσα ενός literal:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia-fr: <http://fr.dbpedia.org/resource/>
SELECT ?label WHERE {
  dbpedia-fr:Corfou rdfs:label ?label .
  FILTER (langMatches(lang(?label),"ja"))
}
```

Επιλογή γλώσσας: σημείωση

- Πιθανόν να δείτε παραδείγματα με `FILTER(lang(?label)="en")`, τα οποία όμως πρέπει να αποφύγετε!
- Η `langMatches()` ξέρει να ταιριάζει καλύτερα τις πιθανές παραλλαγές της γλώσσας, όπως π.χ. τα ”en-US”,”el-GR” κλπ

Η σειρά σας!

- Αρχικά, επισκεφτείτε το http://fr.dbpedia.org/resource/Catégorie:Île_de_Grèce και παρατηρήστε πώς περιγράφονται τα νησιά της Ελλάδας
- Στη συνέχεια χρησιμοποιήστε το endpoint http://fr.dbpedia.org/sparql
  - Ρωτήστε για όλες τις οντότητες (νησιά) που ανήκουν (prop-fr:catégorie) στην κατηγορία αυτή
    - `PREFIX prop-fr: <http://fr.dbpedia.org/property/>`
  - Προσθέστε στα αποτελέσματα τον πληθυσμό κάθε νησιού (ιδιότητα prop-fr:population)
  - Ταξινομήστε τα αποτελέσματα κατά φθίνουσα σειρά πληθυσμού

Εύρεση μεγίστου (SPARQL 1.1)

- Τροποποιήστε το προηγούμενο ερώτημά σας ως εξής:

```
SELECT (max(?pop) AS ?maxpop)  WHERE{
  # το προηγούμενο WHERE σας,
  # έστω ?pop η μεταβλητή πληθυσμού
}
```

- Το ίδιο μπορείτε να κάνετε, στη SPARQL 1.1 πάντα, με τα MIN(), SUM(), AVG(), COUNT()
  - Στη SPARQL 1.0 μπορούσατε μόνο να βρείτε μέγιστα και ελάχιστα με συνδυασμό των ORDER BY και LIMIT 1

Και ποιο νησί έχει τον μέγιστο πληθυσμό;

- Εδώ χρειάζεται ένα υποερώτημα (subquery)!

```
SELECT ?maxisl ?maxpop WHERE {
 {# υποερώτημα
  SELECT (max(?pop) AS ?maxpop) WHERE{
    # το αρχικό WHERE σας,
    # έστω ?pop η μεταβλητή πληθυσμού
  }
 }
 {# υπόλοιπο κύριου ερωτήματος
    # το αρχικό WHERE σας,
    # με ?maxisl μεταβλητή νησιού
    # και ?maxpop μεταβλητή πληθυσμού
 }
}
```

- Η υποερώτηση θα εκτελεστεί πρώτα
  - και στη συνέχεια το δεύτερο μέρος της κύριας ερώτησης

Bonus Υλικό: GROUP BY και HAVING

- Όπως ακριβώς στην SQL (aggregation)

```
PREFIX prop-fr: <http://fr.dbpedia.org/property/>

SELECT ?arc  (sum(?pop) AS ?sumpop) WHERE{
  ?isl prop-fr:catégorie
    <http://fr.dbpedia.org/resource/Catégorie:Île_de_Grèce> .
  ?isl prop-fr:population ?pop.
  ?isl <http://dbpedia.org/ontology/archipelago> ?arc .
} GROUP BY ?arc HAVING (sum(?pop)>0) ORDER BY desc(?sumpop)
```

- Σειρά σύνταξης: GROUP BY $\rightarrow$ HAVING $\rightarrow$ ORDER BY $\rightarrow$ LIMIT/OFFSET