Building & Mining Knowledge Graphs

(KEN4256)

Lab 5: Interlinking and Advanced Querying

Amrapali Zaveri, Vincent Emonet

Maastricht University

Institute of Data Science

Querying Knowledge Bases

Querying Data

Using SPARQL query language https://www.w3.org/TR/rdf-sparql-query/

- SPARQL (pronounced sparkle) stands for: SPARQL Protocol And RDF Query Language
- SPARQL 1.0 W3C-Recommendation since January 15th 2008
- SPARQL 1.1 W3C-Recommendation since March 21st 2013 Query language to query instances in RDF documents
- Great practical importance (almost all applications need it) to query data stored in a graph

Note: w3.org material are standards and recommendations accepted by the World Wide Web Consortium (W3C, the organism defining the Internet standards)

Query the DBpedia SPARQL endpoint

https://dbpedia.org/sparql

Or use a nicer query editor:

https://yasgui.triply.cc

SPARQL Query - Example

```
SELECT *
WHERE {
  ?subject ?predicate ?object .
}
LIMIT 10
```

SPARQL Components

```
# prefix declarations: for abbreviating URIs
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
# dataset definition (optional): which RDF graph(s) are being queried
FROM
# result clause: what information to return from the query
SELECT *
# query pattern: specifying what to query for in the underlying dataset
WHERE {
    ?s ?p ?o .
}
# query modifiers: slicing, ordering, and rearranging query results
ORDER BY ?s
LIMIT 10
```

SPARQL Example - Get instances of a class

```
SELECT *
WHERE {
  ?book rdf:type <http://dbpedia.org/ontology/Book> .
}
```

The rdf prefix is defined by default
If run on those 2 statements it will return only the 1 one in ?book
<http://book1> rdf:type <http://dbpedia.org/ontology/Book> .
<http://country1> rdf:type <http://dbpedia.org/ontology/Country> .

SPARQL Example - Get property of a class

```
PREFIX dbo:<http://dbpedia.org/ontology/>
SELECT *
WHERE {
  ?book a dbo:Book .
  ?book dbo:author ?author .
}
```

“a” is a standard shorthand for rdf:type

Returns only http://book1 infos here:

```
<http://book1> rdf:type <http://dbpedia.org/ontology/Book> .
<http://book1> dbo:author <http://author1> .
<http://book2> rdf:type <http://dbpedia.org/ontology/Book> .
<http://book2> dbo:contributor <http://author2> .
```

SPARQL Example

```
PREFIX dbo:<http://dbpedia.org/ontology/>
SELECT *
WHERE {
  ?book a dbo:Book
  ?book dbo:author ?author
} LIMIT 10
```

What's wrong with this query?

SPARQL Example - Retrieving specific entities

```
PREFIX dbo:<http://dbpedia.org/ontology/>
SELECT ?author
WHERE {
  ?book a dbo:Book ;
    dbo:author ?author .
} LIMIT 10
```

SPARQL Example - Graph (or Context)

```
PREFIX dbo:<http://dbpedia.org/ontology/>
SELECT ?author ?graph
WHERE {
    GRAPH ?graph {
        ?book a dbo:Book .
        ?book dbo:author ?author .
    }
} LIMIT 10
```

N-QUADS

<http://dbp.org/Between_Planets> <rdf:type> <dbo:Book> <http://dbpedia.org> .
<http://dbp.org/Between_Planets> <dbo:author> <http://dbp.org/Robert_A._Heinlein> <http://dbpedia.org> .

SPARQL Example - Graph

Some common queries are optimized on most triplestores

```
SELECT ?g
WHERE {
    GRAPH ?g {
        ?s ?p ?o .
    }
}
```

Using FILTER

Comparison operators: <, =, >, <=, >=, !=

- Comparison of data literals according to natural order
- Support for numerical data types, xsd:dateTime, xsd:string (alphabetic ordering), xsd:Boolean (1>0)
- For other types and other RDF-elements, only = and != are available
  Comparison of literals of incompatible types (e.g. xsd:string and xsd:integer) is not allowed

Arithmetic operators: +, -, *, /

- Support for numerical data types
- Used to combine values in filter conditions
  - E.g. FILTER(?weight/ (?size*?size)>=25)

Using FILTER

```
PREFIX dbo:<http://dbpedia.org/ontology/>
SELECT DISTINCT ?author
WHERE {
   ?book a dbo:Book .
   ?book dbo:author ?author .
   ?book dbo:numberOfPages ?pages .
   FILTER (?pages > 500)
} LIMIT 10
```

Special FILTER Functions

sameTERM(A,B)	true, if A and B are the same RDF-terms.
langMATCHES(A,B)	true, if the language specification A fits the pattern B
REGEX(A,B)	true, if the character string A contains the regular expression B

Special FILTER Functions

```
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbp:<http://dbpedia.org/property/>
SELECT *
WHERE {
  ?book a dbo:Book .
  ?book dbo:author ?author .
  ?book dbo:numberOfPages ?pages .
  ?book dbp:name ?name .
  FILTER (?pages > 500)
  FILTER regex(lcase(?name),"thug$")
} LIMIT 10
```

FILTER Functions: Boolean Operators

Filter conditions can be linked with boolean operators: **&&, ||, !**

Partially also expressible through graph pattern:

- Conjunction corresponds to specifications of several filters
- Disjunction corresponds to application of filters in alternative patterns

SORTING Results

- How can one retrieve defined parts of the output set?
- How are the results ordered?
- Can duplicate result rows be removed instantaneously?

SORTING Results

```
SELECT *
WHERE {
  ?book a dbo:Book .
  ?book dbo:author ?author .
  ?book dbo:numberOfPages ?pages .
  ?book dbp:name ?name.
  FILTER (?pages > 500)
  FILTER regex(?name,"en")
} ORDER BY ?pages
LIMIT 10
```

ORDERING Results

Other possible specifications:

- ORDER BY DESC(?page): descending
- ORDER BY ASC(?page): ascending
- ORDER BY DESC(?page), ?chapter: hierarchical classification criteria

LIMIT, OFFSET, DISTINCT

Restriction of output set:

- LIMIT: maximal number of results (table rows)
- OFFSET: position of the first delivered result SELECT
- DISTINCT: removal of duplicate table rows

LIMIT and OFFSET usually only make sense with ORDER BY!

DESCRIBE

DESCRIBE <http://dbpedia.org/resource/Maastricht>

The DESCRIBE query result clause allows the server to return whatever RDF it wants that describes the given resource(s).

Test - What is this query doing?

```
SELECT ?director_name ?movie_name ?actor_name
  WHERE {
   ?movie dbpedia-owl:starring dbpedia:Julia_Roberts  .
   ?movie dbpedia-owl:starring ?actor  .
   ?movie rdfs:label ?movie_name  .
   ?actor rdfs:label ?actor_name .
   ?movie dbpedia-owl:director ?director .
   ?director rdfs:label ?director_name  .
   FILTER (langMatches(lang(?movie_name), "EN") .
  }
ORDER BY ?director ?movie
```

Summary: SPARQL query breakdown

```
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbp:<http://dbpedia.org/property/>
SELECT ?name ?author
WHERE {
  ?book a dbo:Book .
  ?book dbo:author ?author .
  ?book dbo:numberOfPages ?pages .
  ?book dbp:name ?name .
  FILTER (?pages > 500)
  FILTER (langMATCHES(LANG(?name),"en"))
}
ORDER BY ?pages
LIMIT 10
```

Prefix declarations

Variables to display in the results

Where clause to define the triples to select

Patterns of triples to select

Filter triples

Order by, group by, limit clauses

SPARQL DIY

- Countries with population greater than 10,000,000 inhabitants
- Musicians who were born in a country with more than 10,000,000 inhabitants
- Films starring Richard Gere and starring Julia Roberts

TIPS:

- View the resource as NTriples to construct your query
- Use http://prefix.cc to look up prefixes
- Always use LIMIT
- Start by getting results one triple at a time and build up

Functions: cast to float

Compute countries density: error due to wrong type

```
SELECT ?country ?area ?population
  (?population/?area AS ?density)
WHERE {
  ?country a dbo:Country ;
    dbo:populationTotal ?population ;
    <http://dbpedia.org/ontology/PopulatedPlace/areaTotal> ?area .
    FILTER(?area != 0)
}
```

Functions: cast to float

Convert a variable to a specific type

Here compute countries density

```
SELECT ?country ?area ?population
  (xsd:float(?population)/xsd:float(?area) AS ?density)
WHERE {
  ?country a dbo:Country ;
    dbo:populationTotal ?population ;
    <http://dbpedia.org/ontology/PopulatedPlace/areaTotal> ?area .
    FILTER(?area != 0)
}
```

Functions: langMatches

Filter retrieved variable on its lang

Here get the Dutch name of Oceanian countries

```
SELECT ?country ?countryName
WHERE {
  ?country a dbo:Country ;
      dbp:continent ?continent ;
      rdfs:label ?countryName .
  FILTER(str(?continent) = "Oceania")
  FILTER langMatches( lang(?countryName), "NL" )
}
```

Many more functions

isLiteral, STRSTARTS, CONTAINS, ENCODE_FOR_URI, REPLACE, MD5 hashing...

A comprehensive specification of SPARQL can be found here:

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

Count

Counts the number of times a given expression has a value

Count the number of bands in each music genre

```
SELECT ?genre count(?band) as ?count
WHERE {
    ?band a dbo:Band .
    ?band dbo:genre ?genre .
} order by desc(?count)
```

Be careful count can be really expensive to run!

Optional

The query do not filter on this pattern, it returns the value if it exists.

Get countries from Oceania and display the dissolution date of this country if they have one.

```
SELECT ?country ?dissolutionDate
WHERE {
  ?country a dbo:Country .
  ?country dbp:continent ?continent .
  OPTIONAL { ?country dbo:dissolutionDate ?dissolutionDate . }
  FILTER(str(?continent) = "Oceania")
}
```

Subqueries

Queryception: a query **inside** a query

Order the **first** 10 countries to have been dissolved by date of creation.

- **Select** all countries that have been dissolved
- **Order** them by dissolution date (oldest to newest)
- **Limit** to 10
- Finally, **order** the results (countries) from the most recently created to the oldest created

Query optimization: do first the limit, then the order by! 🚀

Subqueries

```
SELECT *
WHERE {
  {
    SELECT ?country ?dissolutionDate
    WHERE {
      ?country a dbo:Country .
      ?country dbo:dissolutionDate ?dissolutionDate .
    } order by ?dissolutionDate limit 10
  }
  ?country dbo:foundingYear ?foundingYear .
} order by desc(?foundingYear)
```

Order countries by dissolution date and keep the 10 first

Order them from the most recently created to the oldest created

Bind and concat

Go to http://yasgui.org/

and select the following endpoint: http://dbpedia.org/sparql

- Bind define a new variable
- concat concatenate strings

Concatenate example: Generate URI out of countries ISO code

```
SELECT *
WHERE {
  ?country a dbo:Country .
  ?country dbp:iso31661Alpha ?isoCode
  BIND(uri(concat("http://country.com/", ?isoCode)) AS ?isoUri)
}
```

Aggregate and group by

Group solutions by variable value

Here get average GDP for all countries grouped by the currency they use

```
SELECT ?currency (AVG(xsd:integer(?gdp)) AS ?avgGdp)
WHERE {
  ?country dbo:currency ?currency ;
    dbp:gdpPppPerCapita ?gdp .
}
GROUP BY ?currency order by desc(?avgGdp)
```

SPARQL endpoint and prefixes to use

Go to https://query.wikidata.org/

And keep those prefixes:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX bd: <http://www.bigdata.com/rdf#>
PREFIX bl: <http://w3id.org/biolink/vocab/>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
```

Construct

Return a graph specified by a template (build triples)

Here generates type and bl:name triples for genes from Wikidata (only get genes which encode protein with a UniProt ID)

```
CONSTRUCT {
  ?gene a bl:Gene ;
    bl:name ?geneLabel .
}
WHERE {
  SERVICE wikibase:label { bd:serviceParam wikibase:language "en". }
  ?gene wdt:P688 ?encodedProtein .
  ?encodedProtein wdt:P352 ?uniprotId .
} LIMIT 20
```

Triples to generate

Update: Insert data

Simply use SPARQL to insert data into your triplestore!

```
PREFIX bl: <http://w3id.org/biolink/vocab/>
INSERT DATA {
  GRAPH <http://graph> {
      <http://whatever> bl:name "whatever" .
  }
}
```

Insert this exact triple

Update: Delete data

To delete particular statements

Here we delete the **bl:name** statements for the genes we just created

```
PREFIX bl: <http://w3id.org/biolink/vocab/>
DELETE DATA {
  GRAPH <http://graph> {
      <http://whatever> bl:name "whatever" .
  }
}
```

Delete this exact triple

Update: Insert

Same as construct but directly insert triples into your triplestore

```
INSERT {
  GRAPH <http://graph> {
    ?geneUri a bl:Gene ;
      bl:name ?geneLabel.
  } }
```

Triples to insert in a graph

```
WHERE {
  SERVICE <https://query.wikidata.org/sparql> {
    SELECT * WHERE {                                    Subquery to limit to 20
      ?geneUri wdt:P688 ?encodedProtein .
      ?encodedProtein wdt:P352 ?uniprotId .
      SERVICE wikibase:label { bd:serviceParam wikibase:language "en".
        ?encodedProtein rdfs:label ?encodesProtein .
        ?geneUri rdfs:label ?geneLabel . } } LIMIT 20
} }
```

Call to a remote SPARQL endpoint to get the data to insert in our triplestore

Update: Delete

To delete particular statements

Here we delete the bl:name statements for the genes we just created

```
DELETE {
  GRAPH <http://graph> {
    ?geneUri bl:name ?geneLabel.
  } }
WHERE {
  ?geneUri a bl:Gene .
  ?geneUri bl:name ?geneLabel .
}
```

Triple pattern to delete

Based on the data retrieved in this where

Inference

Infer statements from a shared vocabulary

Get chemical substances (including drugs) from https://graphdb.dumontierlab.com/sparql in the *ncats-red-kg* repository

```
PREFIX bl: <http://w3id.org/biolink/vocab/>
SELECT DISTINCT *
{
  ?chemUri a bl:ChemicalSubstance .
  OPTIONAL {?chemUri bl:name ?chemName . }
}
```

The data use the following vocabulary: https://biolink.github.io/biolink-model/

Inference

See inferred statements

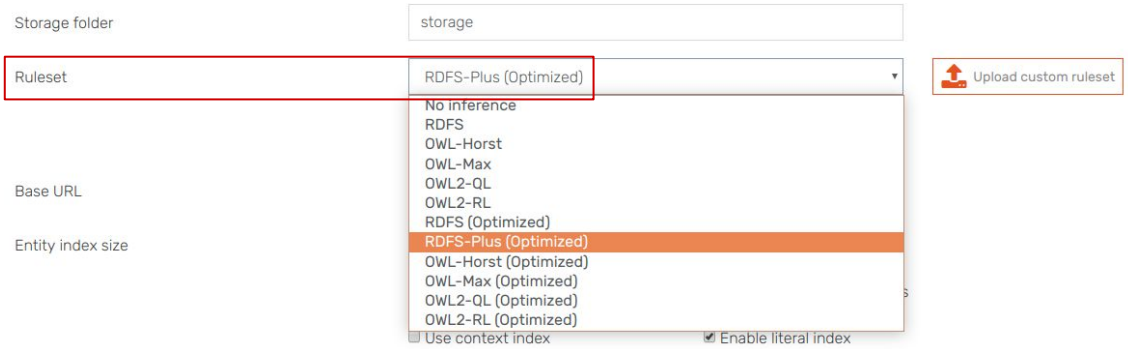
Local

ncats-red-kg • ncats-red-kg

total statements
395,303,401

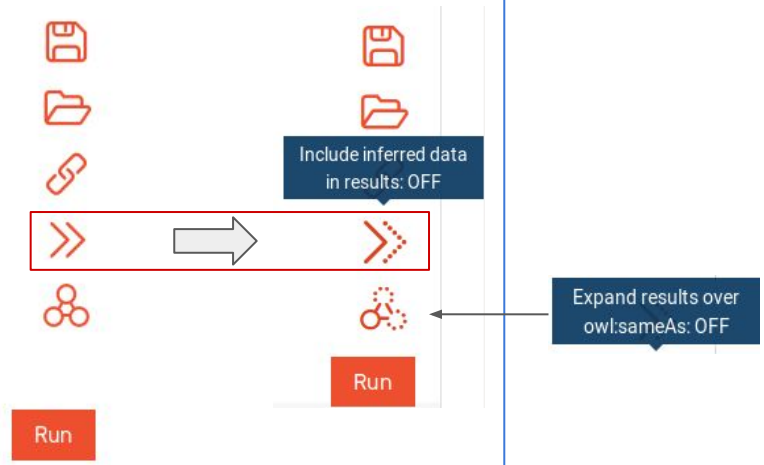
93,924,539 explicit
301,378,862 inferred
4.21 expansion ratio

Choose inference rules when creating repository



Activate/deactivate inference in results

```
1  PREFIX bl: <http://w3id.org/biolink/vocab/>
2  SELECT distinct *
3  {
4    ?drugUri a bl:ChemicalSubstance .
5    OPTIONAL {?drugUri bl:name ?drugName .}
6  }
7
```



Federated query

Go to **http://yasgui.org/**

and select the following endpoint: https://bio2rdf.org/sparql/

Find the **drugs** that interacts with **10 proteins** and the label of the Gene that encode them 

Federated query enables you to query and join multiple SPARQL endpoints

- Get “gene encodes protein” information on UniProt SPARQL endpoint
- Get “drug interacts with protein” on IDS GraphDB

⚠️ Federated queries are much slower, use a subquery in the service call to avoid multiple call between services

Federated query

To run in https://sparql.uniprot.org/

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?author
WHERE {
  SERVICE <https://dbpedia.org/sparql> {
    ?book a dbo:Book ;
      dbo:author ?author .
  }
} LIMIT 10
```

The query in SERVICE is executed in the defined endpoint

Federated query

```
SELECT ?protein ?geneName
WHERE {
  SERVICE <https://sparql.uniprot.org/> {
    SELECT * WHERE {
      ?protein a up:Protein .
      ?protein up:encodedBy ?gene .
      ?gene skos:prefLabel ?geneName .
    } LIMIT 10
  }
}
```

Subquery with limit

UniProt call to get 20 genes and the protein they encode

More complex federated query

```
SELECT ?protein ?geneName ?affectedByDrug
WHERE {
  SERVICE <https://sparql.uniprot.org/> {
    SELECT * WHERE {
      ?protein a up:Protein .
      ?protein up:encodedBy ?gene .
      ?gene skos:prefLabel ?geneName .
    } LIMIT 10
  } BIND(uri(replace(str(?protein), "http://purl.uniprot.org/", "http://identifiers.org/"))
as ?idUri)
  SERVICE <http://graphdb.dumontierlab.com/repositories/ncats-red-kg> {
    ?association a bl:ChemicalToGeneAssociation ;
      bl:object ?idUri ;
      bl:subject [ bl:name ?affectedByDrug ] .
  } }
```

UniProt call to get 20 genes and the protein they encode

Subquery with limit

Generate identifiers.org URI from UniProt URI

GraphDB call to get drugs that interact with the encoded protein