SPARQL

An RDF Query Language

SPARQL

- SPARQL is a recursive acronym for

  SPARQL **P**rotocol **A**nd **R**df **Q**uery **L**anguage
- SPARQL is the SQL for RDF triple stores
- Example query suitable for DBpedia

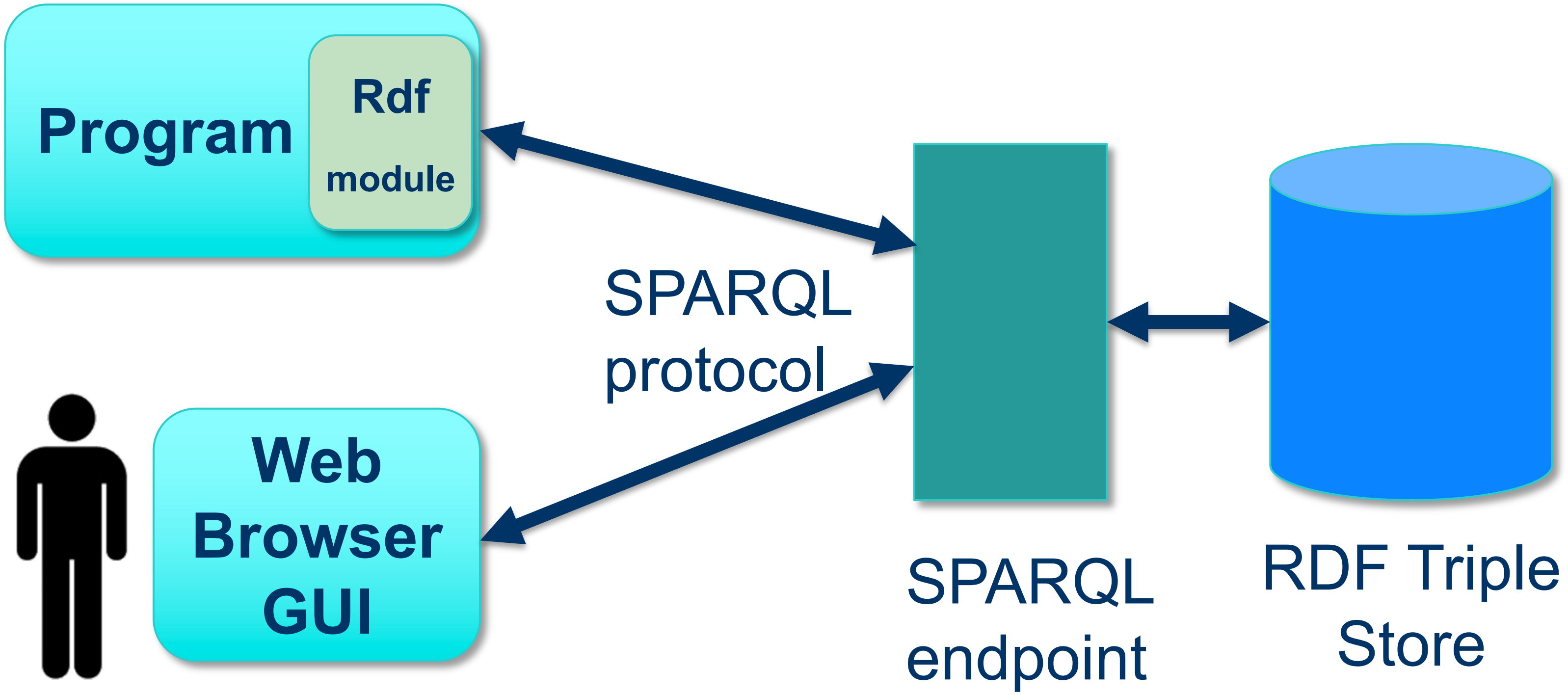
```
# find countries and their languages
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT * WHERE {
  ?country a dbo:Country;
           dbo:officialLanguage ?lang .
}
LIMIT 10
```

SPARQL History

- Several RDF query languages were developed prior to SPARQL
- W3C RDF Data Access Working Group (DAWG) worked out SPARQL 2005-2008
- Became a W3C recommendation in Jan 2008
- SPARQL 1.1 (2013) is the current standard
- Support for many prog. languages available
- W3 SPARQL 1.2 Community Group established in 2019 to explore extensions, not much activity now

Typical Architecture

SPARQL endpoint receives queries and requests via HTTP from programs or GUIs, accesses associated RDF triple store and returns result, e.g., data



SPARQL Over HTTP (the SPARQL Protocol)

`http://host.domain.com/sparql/endpoint?`*<parameters>*

where *<parameters>* **can include:**

`query=`*<encoded query string>*

e.g. `SELECT+*%0DWHERE+{`…

`default-graph-uri=`*<encoded graph URI>*

e.g. `http%3A%2F%2Fexmaple.com%2Ffoo`…

n.b. zero of more occurrences of `default-graph-uri`

`named-graph-uri=`*<encoded graph URI>*

e.g. `http%3A%2F%2Fexmaple.com%2Fbar`…

n.b. zero of more occurrences of `named-graph-uri`

HTTP `GET` or `POST`. Graphs given in the protocol override graphs given in the query.

Some SPARQL endpoints

There are many public endpoints, e.g.

- DBpedia: https://dbpedia.org/sparql/
- Wikidata: https://query.wikidata.org/sparql
- DBLP: https://dblp.l3s.de/d2r/sparql
- See W3C’s list of currently alive SPARQL endpoints (not up to date, tho)

It’s not hard to set up your own, e.g.

- UMBC cybersecurity knowledge graph: http://eb4.cs.umbc.edu:9090/ckg/query/

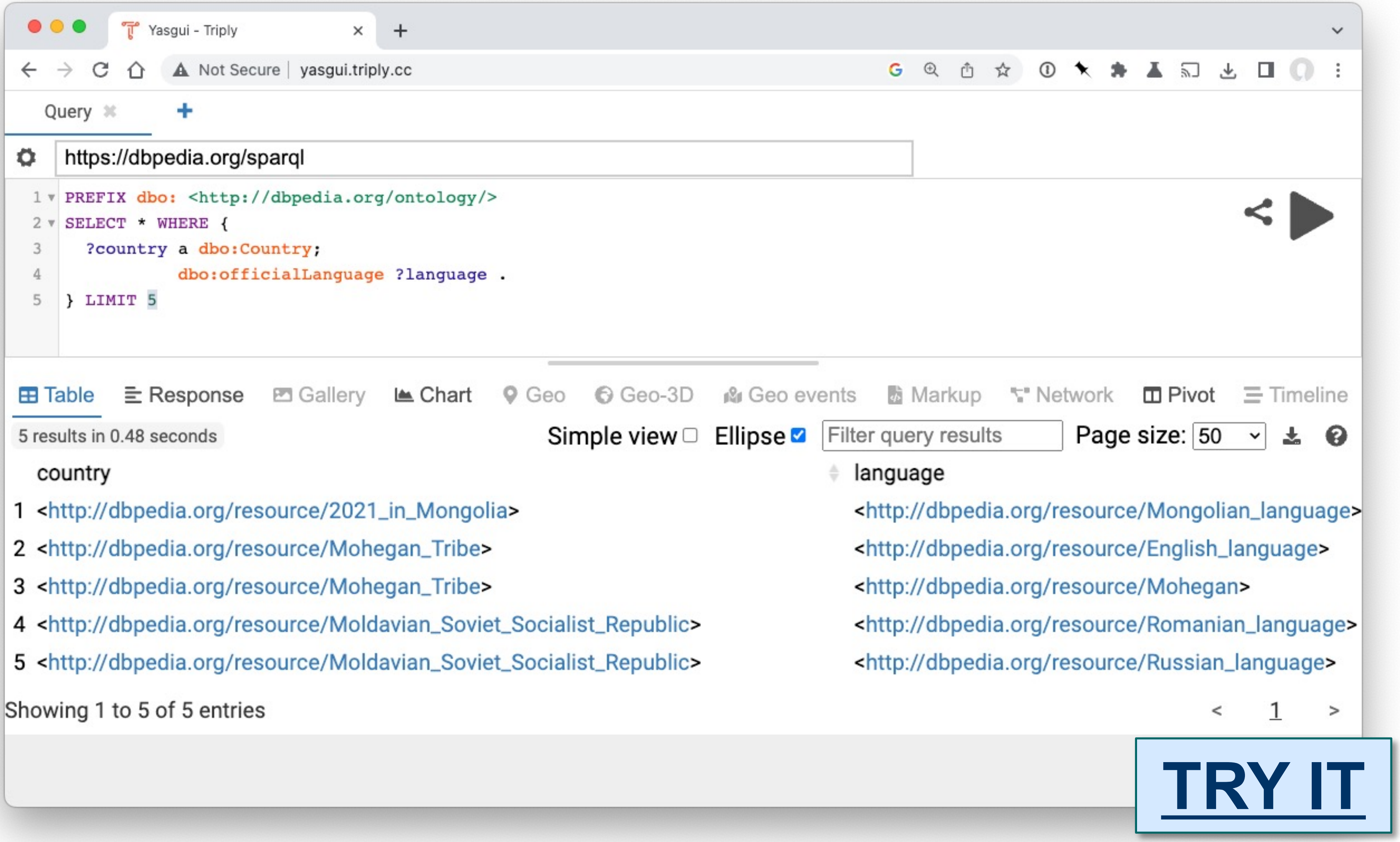
Endpoint GUIs

- Some endpoints offer their own SPARQL GUI you can use to enter ad hoc queries
- They may use the same URL as the REST interface and rely on the protocol to know when it’s a person and when a query
  - Dbpedia: http://dbpedia.org/sparql/
  - Wikidata: https://query.wikidata.org/
  - DBLP: https://dblp.l3s.de/d2r/snorql/

General SPARQL GUIs

- You can also access or run a general SPARQL GUI that can talk to any SPARQL endpoint
- A nice example is YASGUI, which is available on GitHub
  - **Y**et **A**nother **S**parql **GUI**
- The source is available on GitHub for custimization for a site or project

YASGUI: Yet Another SPARQL GUI



https://dbpedia.org/sparql

```
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT * WHERE {
  ?country a dbo:Country;
           dbo:officialLanguage ?language .
} LIMIT 5
```

5 results in 0.48 seconds

Simple view ☐ Ellipse ☑ Filter query results Page size: 50

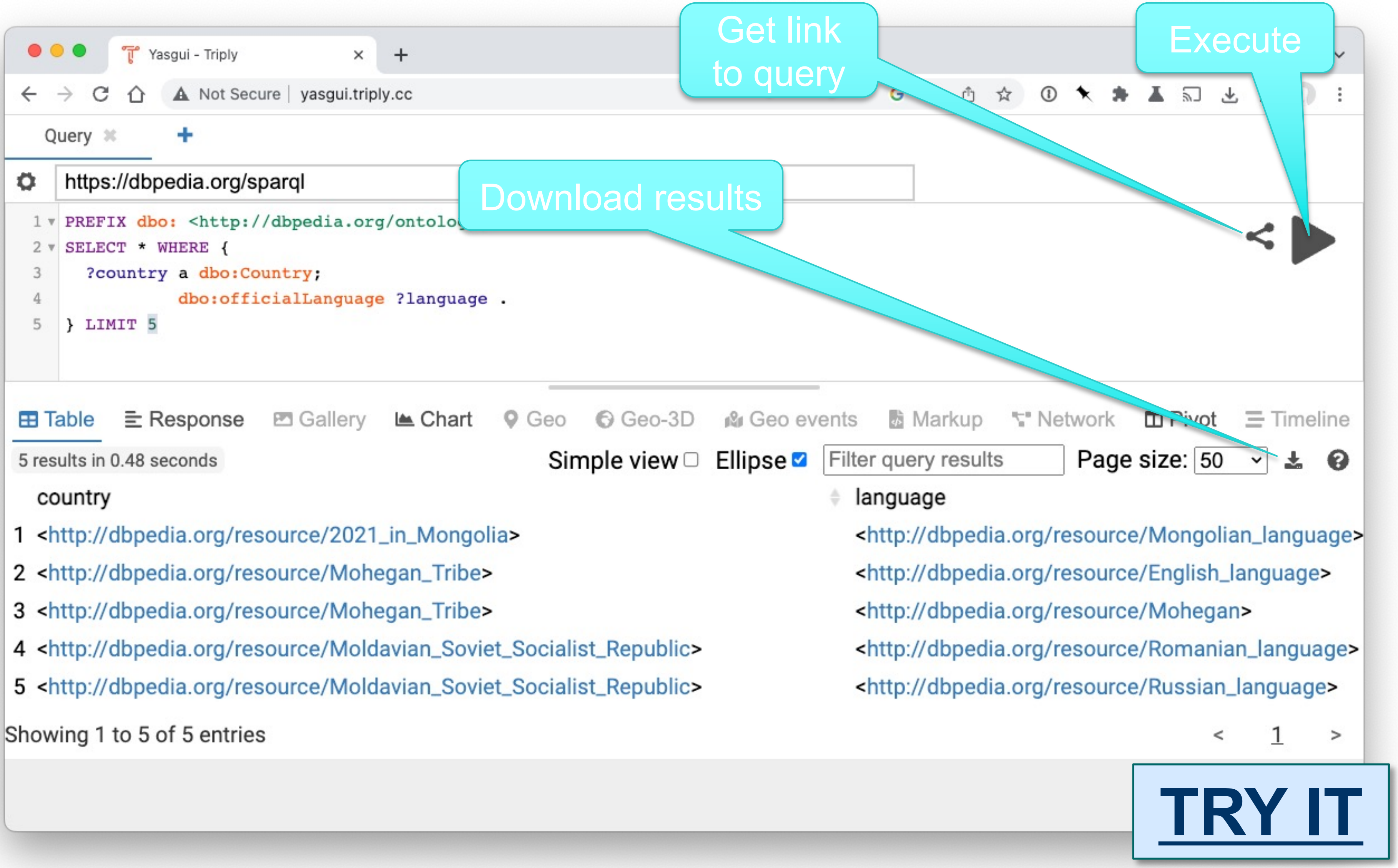
	country	language
1	<http://dbpedia.org/resource/2021_in_Mongolia>	<http://dbpedia.org/resource/Mongolian_language>
2	<http://dbpedia.org/resource/Mohegan_Tribe>	<http://dbpedia.org/resource/English_language>
3	<http://dbpedia.org/resource/Mohegan_Tribe>	<http://dbpedia.org/resource/Mohegan>
4	<http://dbpedia.org/resource/Moldavian_Soviet_Socialist_Republic>	<http://dbpedia.org/resource/Romanian_language>
5	<http://dbpedia.org/resource/Moldavian_Soviet_Socialist_Republic>	<http://dbpedia.org/resource/Russian_language>

Showing 1 to 5 of 5 entries

< 1 >

TRY IT

YASGUI: Yet Another SPARQL GUI



https://dbpedia.org/sparql

```
PREFIX dbo: <http://dbpedia.org/ontolog
SELECT * WHERE {
  ?country a dbo:Country;
            dbo:officialLanguage ?language .
} LIMIT 5
```

5 results in 0.48 seconds

Simple view ☐ Ellipse ☑ Page size: 50

	country	language
1	<http://dbpedia.org/resource/2021_in_Mongolia>	<http://dbpedia.org/resource/Mongolian_language>
2	<http://dbpedia.org/resource/Mohegan_Tribe>	<http://dbpedia.org/resource/English_language>
3	<http://dbpedia.org/resource/Mohegan_Tribe>	<http://dbpedia.org/resource/Mohegan>
4	<http://dbpedia.org/resource/Moldavian_Soviet_Socialist_Republic>	<http://dbpedia.org/resource/Romanian_language>
5	<http://dbpedia.org/resource/Moldavian_Soviet_Socialist_Republic>	<http://dbpedia.org/resource/Russian_language>

Showing 1 to 5 of 5 entries

< 1 >

Get link to query

Execute

Download results

TRY IT

Basic SPARQL Query Forms

- SELECT

  Returns all, or a subset of, the variables bound in a query pattern match

- ASK

  Returns boolean indicating whether a query pattern matches or not

- DESCRIBE

  Returns an RDF graph describing resources found

- CONSTRUCT

  Returns an RDF graph constructed by substituting variable bindings in a set of triple templates

The 4 Types of SPARQL Queries

SELECT queries

Project out specific variables and expressions:

```
SELECT ?c ?cap (1000 * ?people AS ?pop)
```

Project out all variables:

```
SELECT *
```

Project out distinct combinations only:

```
SELECT DISTINCT ?country
```

Results in a table of values (in XML or JSON):

?c	?cap	?pop
ex:France	ex:Paris	63,500,000
ex:Canada	ex:Ottawa	32,900,000
ex:Italy	ex:Rome	58,900,000

CONSTRUCT queries

Construct RDF triples/graphs:

```
CONSTRUCT {
   ?country a ex:HolidayDestination ;
       ex:arrive_at ?capital ;
       ex:population ?population .
}
```

Results in RDF triples (in any RDF serialization):

```
ex:France a ex:HolidayDestination ;
    ex:arrive_at ex:Paris ;
    ex:population 635000000 .
ex:Canada a ex:HolidayDestination ;
    ex:arrive_at ex:Ottawa ;
    ex:population 329000000 .
```

ASK queries

Ask whether or not there are any matches:

```
ASK
```

Result is either “true” or “false” (in XML or JSON):

```
true, false
```

DESCRIBE queries

Describe the resources matched by the given variables:

```
DESCRIBE ?country
```

Result is RDF triples (in any RDF serialization) :

```
ex:France a geo:Country ;
  ex:continent geo:Europe ;
  ex:flag <http://…/flag-france.png> ;
  …
```

SPARQL query structure

- *Prefix declarations* for abbreviating URIs
- *Dataset definition:* what RDF graph(s) are being queried
- *Result clause:* what information to return from the query
- *Query pattern:* what to query for in dataset
- *Query modifiers*, slicing, ordering, rearranging query results

```
# prefix declarations
PREFIX ex: <http://example.com/rdf/>  …
# optional named graph source
FROM …
# result clause (select,ask,update...)
SELECT …
# query pattern
WHERE { … }
# query modifiers
ORDER BY …
GROUP BY ….
LIMIT 100
```

SPARQL protocol parameters

- To use this query, we need to know]
  - What endpoint (URL) to send it to
  - How we want the results encoded (JSON, XML, …)
  - … other parameters …
- These are set in GUI or your program
  - Except for the endpoint, all have defaults
- Can even query with the unix curl command:

```
curl https://dbpedia.org/sparql/ --data-urlencode query='PREFIX yago:
 <http://dbpedia.org/class/yago/> SELECT * WHERE {?city rdf:type
 yago:WikicatCitiesInMaryland.}'
```

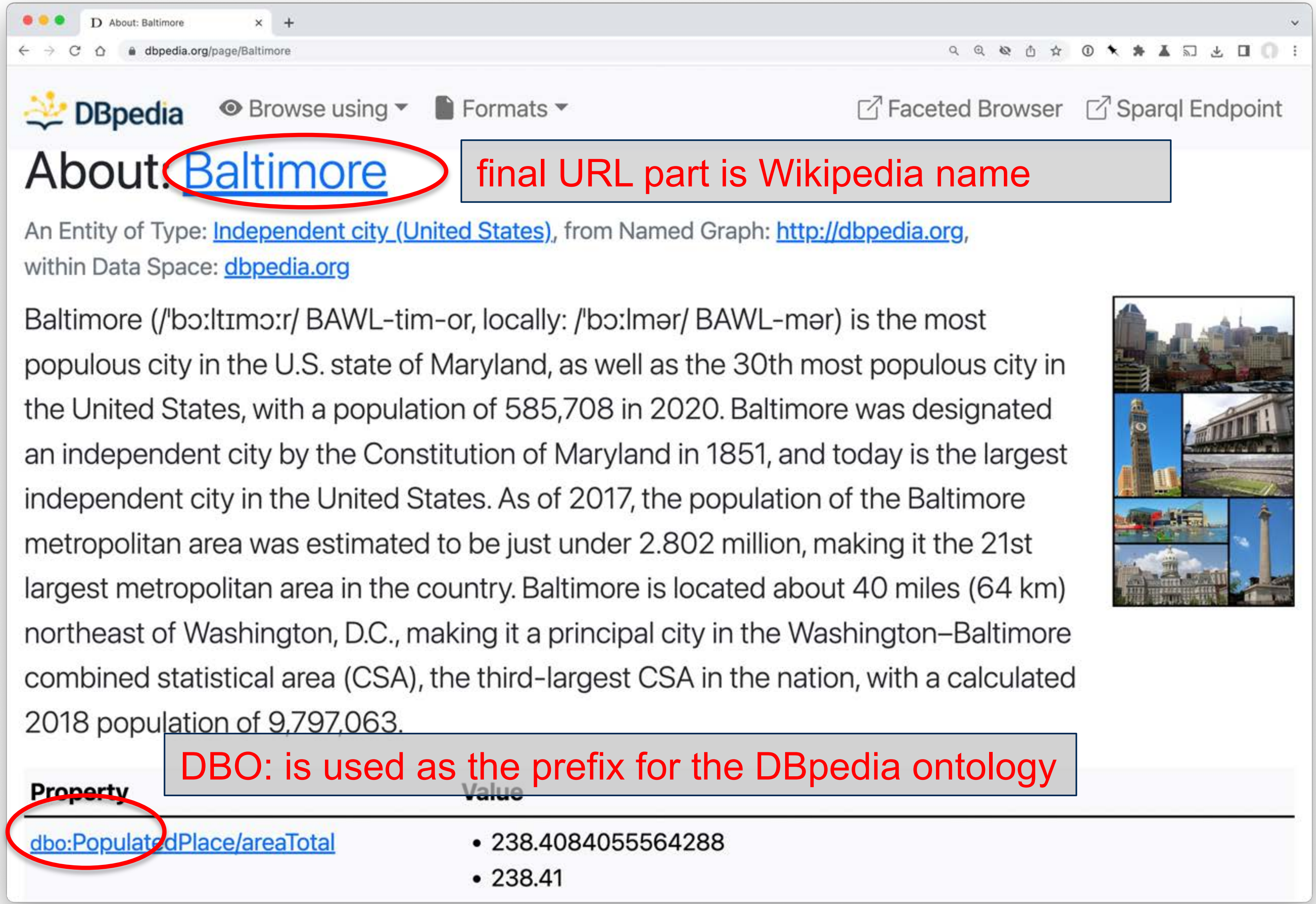
Exploring SPARQL with DBpedia

- DBpedia is a knowledge graph extracted from different Wikipedia sites
- Started in 2007, it continued to develop and offer services based on it
- Explore it in your browser in a human-readable form
- Query it via a public SPARQL endpoint to collect data
- Use services like DBpedia Spotlight to extract entities and concepts from text
- Download its data as JSON objects for your own use

Let’s find data about cities in MD

- Need to understand how DBpedia represents data about cites
- We can view the ontology with its ~700 classes and ~2,800 properties
- And/or examine an entity, like Baltimore by
  - Doing a web search on *dbpedia Baltimore*
  - Clicking on links in the resulting page
- Retrieves the RDF data and formats it for human viewing

Baltimore in DBpedia (1)



About: Baltimore

final URL part is Wikipedia name

An Entity of Type: Independent city (United States), from Named Graph: http://dbpedia.org, within Data Space: dbpedia.org

Baltimore (/ˈbɔːltɪmɔːr/ BAWL-tim-or, locally: /ˈbɔːlmər/ BAWL-mər) is the most populous city in the U.S. state of Maryland, as well as the 30th most populous city in the United States, with a population of 585,708 in 2020. Baltimore was designated an independent city by the Constitution of Maryland in 1851, and today is the largest independent city in the United States. As of 2017, the population of the Baltimore metropolitan area was estimated to be just under 2.802 million, making it the 21st largest metropolitan area in the country. Baltimore is located about 40 miles (64 km) northeast of Washington, D.C., making it a principal city in the Washington–Baltimore combined statistical area (CSA), the third-largest CSA in the nation, with a calculated 2018 population of 9,797,063.

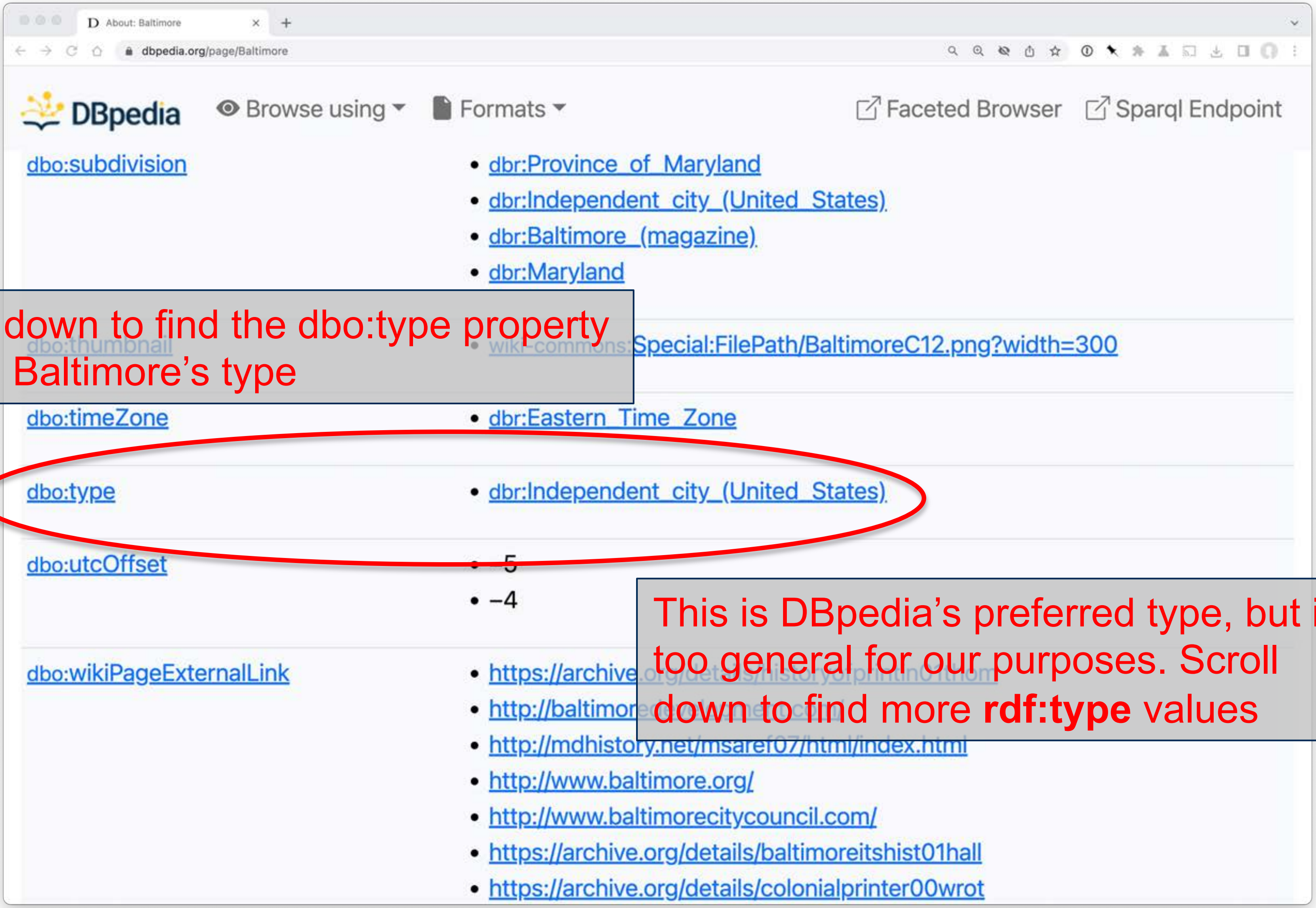
DBO: is used as the prefix for the DBpedia ontology

Property	Value
dbo:PopulatedPlace/areaTotal	• 238.4084055564288 • 238.41

Dbpedia’s prefixes

- DBpedia uses several prefixes for its ontology and data
  - **DBO:** is used for its ontology, e.g., classes and properties (http://dbpedia.org/ontology/)
  - **DBR:** is used for its resources, i.e., instances (http://dbpedia.org/resource/
- DBpedia also uses RDF and RDFS terms with their standard prefixes as well as other common ontologies

Baltimore in DBpedia (2)

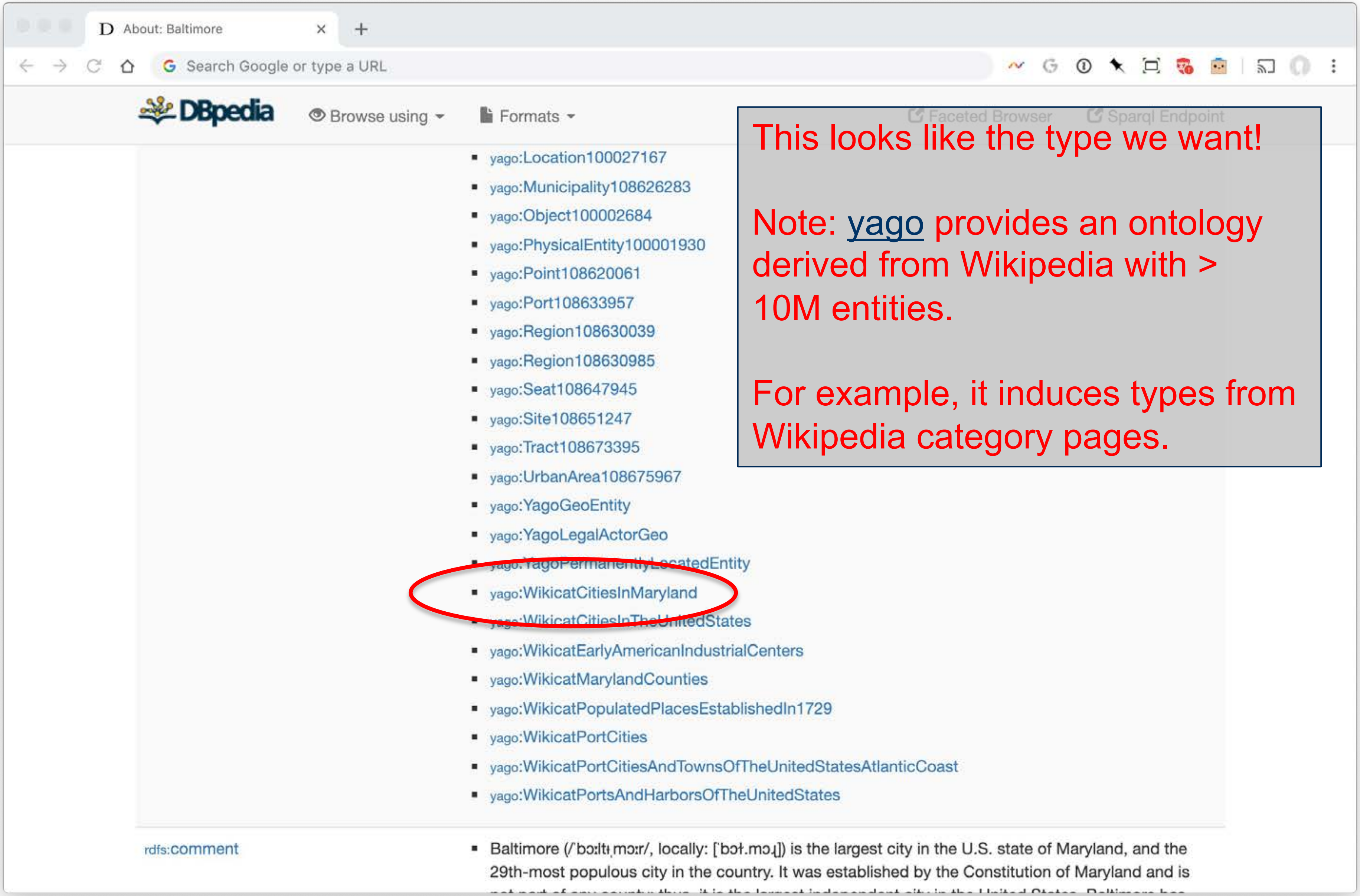


dbo:subdivision	- dbr:Province_of_Maryland - dbr:Independent_city_(United_States) - dbr:Baltimore_(magazine) - dbr:Maryland
dbo:thumbnail	- wiki-commons:Special:FilePath/BaltimoreC12.png?width=300
dbo:timeZone	- dbr:Eastern_Time_Zone
dbo:type	- dbr:Independent_city_(United_States)
dbo:utcOffset	- −5 - −4
dbo:wikiPageExternalLink	- https://archive.org/details/historyofprintin01thom - http://baltimoredevelopment.com/ - http://mdhistory.net/msaref07/html/index.html - http://www.baltimore.org/ - http://www.baltimorecitycouncil.com/ - https://archive.org/details/baltimoreitshist01hall - https://archive.org/details/colonialprinter00wrot

Scroll down to find the dbo:type property to see Baltimore's type

This is DBpedia's preferred type, but it's too general for our purposes. Scroll down to find more **rdf:type** values

Baltimore in DBpedia (3)



- yago:Location100027167
- yago:Municipality108626283
- yago:Object100002684
- yago:PhysicalEntity100001930
- yago:Point108620061
- yago:Port108633957
- yago:Region108630039
- yago:Region108630985
- yago:Seat108647945
- yago:Site108651247
- yago:Tract108673395
- yago:UrbanArea108675967
- yago:YagoGeoEntity
- yago:YagoLegalActorGeo
- yago:YagoPermanentlyLocatedEntity
- yago:WikicatCitiesInMaryland
- yago:WikicatCitiesInTheUnitedStates
- yago:WikicatEarlyAmericanIndustrialCenters
- yago:WikicatMarylandCounties
- yago:WikicatPopulatedPlacesEstablishedIn1729
- yago:WikicatPortCities
- yago:WikicatPortCitiesAndTownsOfTheUnitedStatesAtlanticCoast
- yago:WikicatPortsAndHarborsOfTheUnitedStates

rdfs:comment

- Baltimore (/ˈbɔːltɪˌmɔːr/, locally: [ˈbɔɫ.mɔɹ]) is the largest city in the U.S. state of Maryland, and the 29th-most populous city in the country. It was established by the Constitution of Maryland and is not part of any county; thus, it is the largest independent city in the United States. Baltimore has

This looks like the type we want!

Note: yago provides an ontology derived from Wikipedia with > 10M entities.

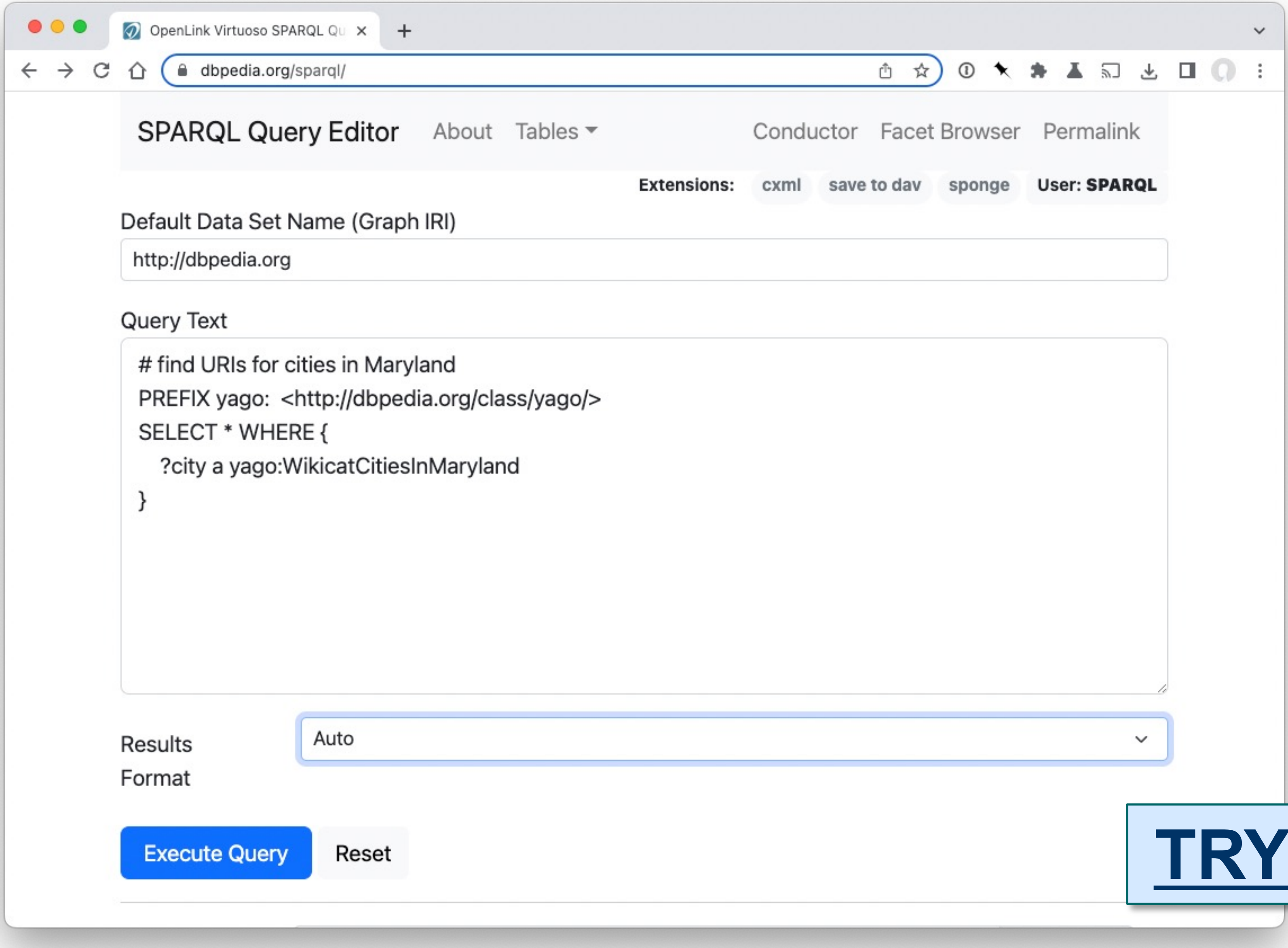
For example, it induces types from Wikipedia category pages.

A Query: Maryland Cities

- This is the the SPARQL query that we want
- We can put it into DBpedia's SPARQL interface at https://dbpedia.org/sparql
- And choose how we want to see the results

```
# find URIs for cities in Maryland
PREFIX yago:  <http://dbpedia.org/class/yago/>
SELECT * WHERE {
    ?city a yago:WikicatCitiesInMaryland
}
```

Dbpedia's SPARQL Interface



Default Data Set Name (Graph IRI)

http://dbpedia.org

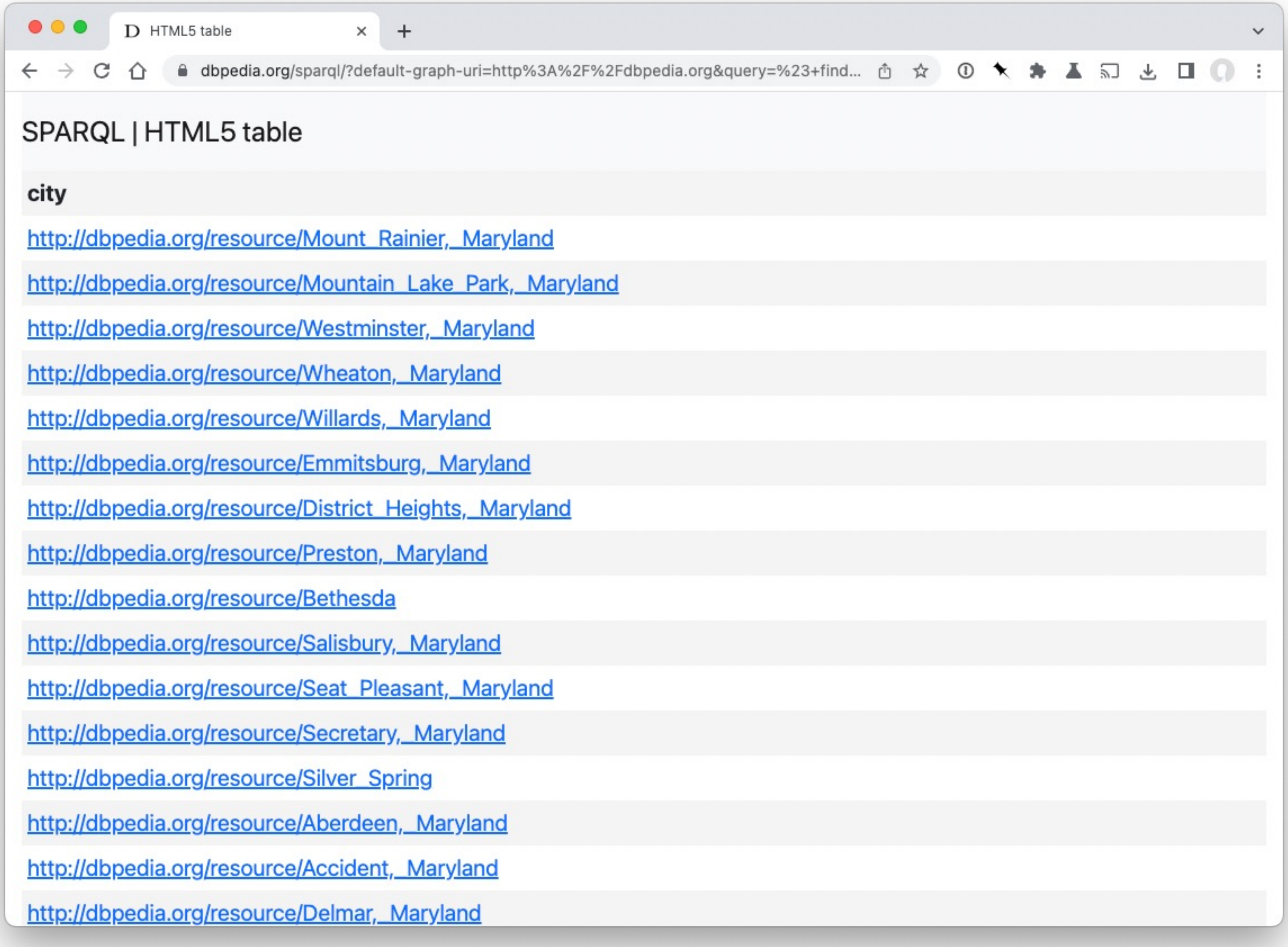
Query Text

```
# find URIs for cities in Maryland
PREFIX yago:  <http://dbpedia.org/class/yago/>
SELECT * WHERE {
   ?city a yago:WikicatCitiesInMaryland
}
```

Results Format: Auto

TRY IT

Dbpedia's SPARQL Interface



HTML5 table
dbpedia.org/sparql/?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=%23+find...

SPARQL | HTML5 table

city
http://dbpedia.org/resource/Mount_Rainier,_Maryland
http://dbpedia.org/resource/Mountain_Lake_Park,_Maryland
http://dbpedia.org/resource/Westminster,_Maryland
http://dbpedia.org/resource/Wheaton,_Maryland
http://dbpedia.org/resource/Willards,_Maryland
http://dbpedia.org/resource/Emmitsburg,_Maryland
http://dbpedia.org/resource/District_Heights,_Maryland
http://dbpedia.org/resource/Preston,_Maryland
http://dbpedia.org/resource/Bethesda
http://dbpedia.org/resource/Salisbury,_Maryland
http://dbpedia.org/resource/Seat_Pleasant,_Maryland
http://dbpedia.org/resource/Secretary,_Maryland
http://dbpedia.org/resource/Silver_Spring
http://dbpedia.org/resource/Aberdeen,_Maryland
http://dbpedia.org/resource/Accident,_Maryland
http://dbpedia.org/resource/Delmar,_Maryland

Maryland Cities and population

- We can build on this to get more information

```
# get cities in MD and their populations
PREFIX yago: <http://dbpedia.org/class/yago/>t
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT * WHERE {
    ?city a yago:WikicatCitiesInMaryland;
          dbo:populationTotal ?population .
}
```

Maryland cities, population, names

```
# this returns names in multiple languages ☹
PREFIX yago:  <http://dbpedia.org/class/yago/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?city ?name ?population WHERE {
   ?city a yago:WikicatCitiesInMaryland;
         dbo:populationTotal ?population ;
          rdfs:label ?name .
}
```

TRY IT

Just the @en names, w/o lang tag

While we’re at it we can drop the prefix declarations because the DBpedia endpoint has them pre-defined.

```
# FILTER gives conditions that must be true
# LANG(x) returns string’s language tag or ””
# STR(x) returns a string or numbervalue, i.e. w/o lang or type tag
select (str(?name) as ?name) (str(?pop) as ?pop)
where {
 ?city a yago:WikicatCitiesInMaryland;
      dbo:populationTotal ?pop;
      rdfs:label ?name .
      FILTER (LANG(?name) = "en") }
```

TRY IT

Order results by population (descending)

```
# sort results by population
# we must leave ?pop as a number for this to work
select (str(?name) as ?name) ?pop where {
 ?city a yago:WikicatCitiesInMaryland;
      dbo:populationTotal ?pop;
      rdfs:label ?name .
      FILTER (LANG(?name) = "en") }
ORDER BY DESC(?pop)
```

TRY IT

Wait, where's Catonsville? ☹

- MD's government focused on counties
- Catonsville not considered a city – it has no government
- We need another category of place
  - Census designated place? Populated Place?
- Populated places include counties & regions; let's use census designated place
- But some 'real' cities in Maryland are not listed as census designated places and some are

UNION operator means OR

```
# better model for a MD city, town or village
SELECT str(?name) ?population where {

 {?city dbo:type dbr:Census-designated_place;
          dbo:isPartOf dbr:Maryland .}
 UNION
 {?city a yago:WikicatCitiesInMaryland . }

 ?city dbo:populationTotal ?population; rdfs:label ?name .
 FILTER (LANG(?name) = "en")
}
ORDER BY DESC(?population)
```

TRY IT

Some cities are missing ☹

- Experimentation with query showed there are some cities missing, e.g., Bethesda & Hagerstown
- Some have no population and one has neither a population nor a label Aberdeen (Maryland) that’s an unintended duplicate
  - Typical of a large and somewhat noisy knowledge graph created from crowdsourced data and extraction from NLP text
- SPARQL’s OPIONAL directive to the rescue

OPTIONAL handles missing data

```
select DISTINCT ?city str(?name) ?population where {
  {?city dbo:type dbr:Census-designated_place;
          dbo:isPartOf dbr:Maryland .}
  UNION
  {?city a yago:WikicatCitiesInMaryland . }
  OPTIONAL {?city dbo:populationTotal ?population.}
  OPTIONAL {?city rdfs:label ?name . FILTER (LANG(?name) =
"en") }
}
ORDER BY DESC(?population)
```

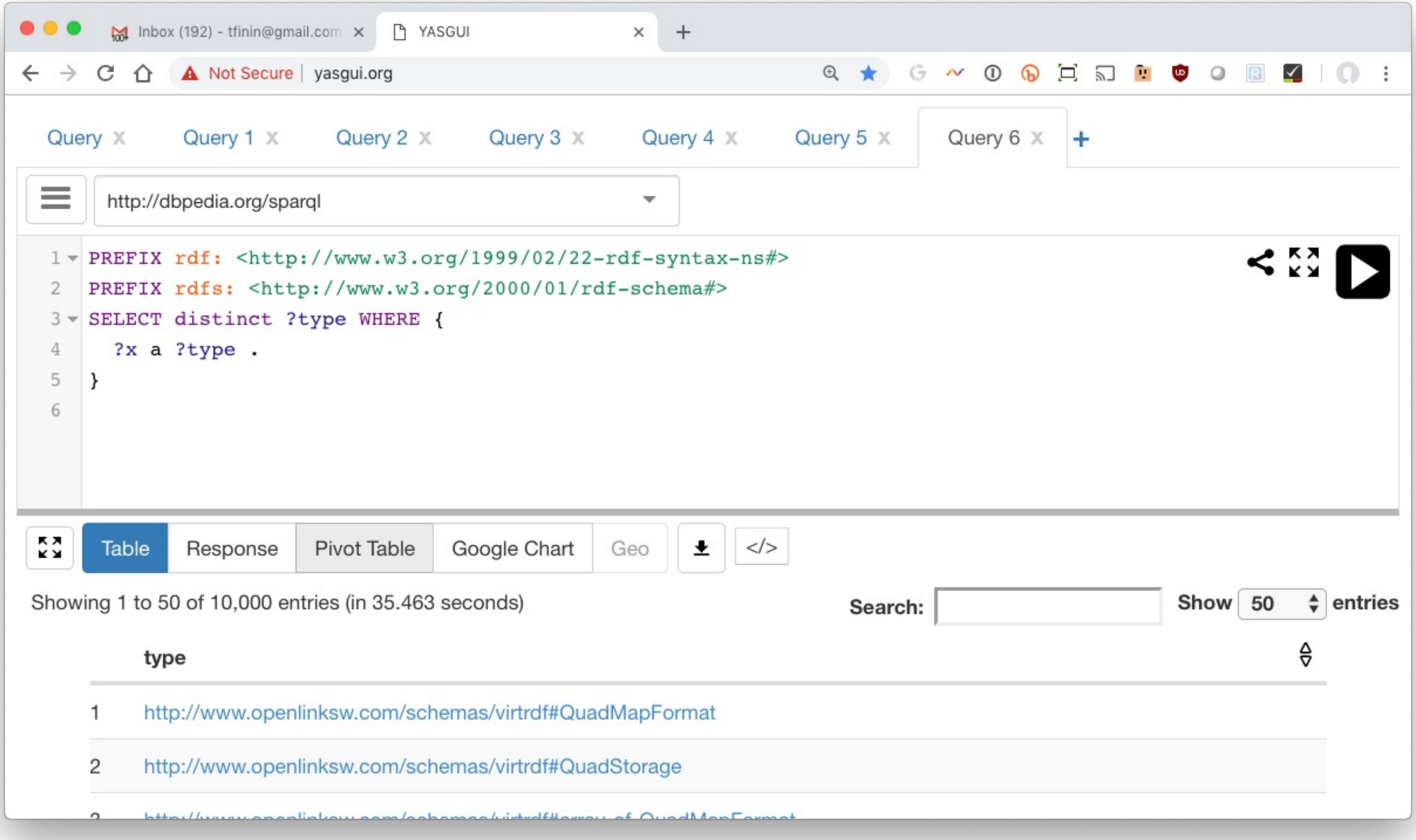
TRY IT

Handling queries with many results

- Public endpoints usually limit a query’s runtime or the number of results
  - DBpedia limits queries to 10K results
  - Wikidata sets a limit of 120 seconds
- There may also be limits on how frequently you can query
- You can use the LIMIT and OFFSET query modifiers to manage large queries
- Suppose we want to find all types DBpedia uses

```
SELECT distinct ?type WHERE {?x a ?type.}
```

Get the first 10K



http://dbpedia.org/sparql

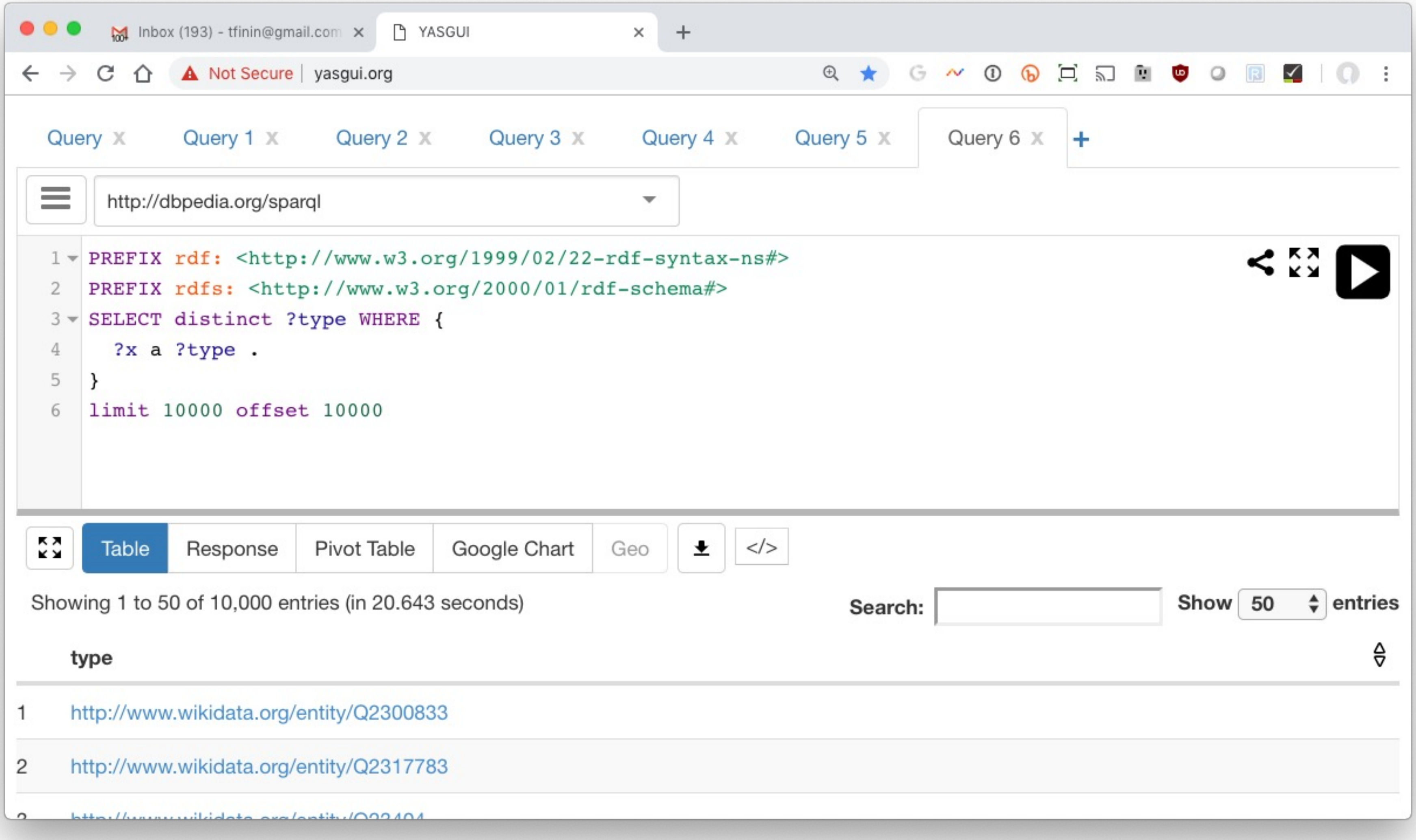
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?type WHERE {
  ?x a ?type .
}
```

Showing 1 to 50 of 10,000 entries (in 35.463 seconds)

Search: Show 50 entries

	type
1	http://www.openlinksw.com/schemas/virtrdf#QuadMapFormat
2	http://www.openlinksw.com/schemas/virtrdf#QuadStorage
3	http://www.openlinksw.com/schemas/virtrdf#array-of-QuadMapFormat

Get the second 10K with OFFSET



http://dbpedia.org/sparql

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT distinct ?type WHERE {
  ?x a ?type .
}
limit 10000 offset 10000
```

Showing 1 to 50 of 10,000 entries (in 20.643 seconds)

Search:

Show 50 entries

	type
1	http://www.wikidata.org/entity/Q2300833
2	http://www.wikidata.org/entity/Q2317783
3	http://www.wikidata.org/entity/Q23404

```python
from SPARQLWrapper import SPARQLWrapper, JSON
default_endpoint = "http://dbpedia.org/sparql"
type_query = """SELECT DISTINCT ?class WHERE {{?x a ?class}} LIMIT {LIM} OFFSET {OFF}"""
def getall(query, endpoint=default_endpoint):
    limit = 10000
    offset = total = 0
    found = limit
    tuples = []
    sparql = SPARQLWrapper(endpoint)
    sparql.setReturnFormat('json')
    while found == limit:  # keep going until we don't get limit results
        q = query.format(LIM=limit, OFF=offset)
        sparql.setQuery(q)
        results = sparql.query().convert()
        found = 0
        for result in results["results"]["bindings"]:
            found += 1
            tuples.append(tuple([str(v['value']) for v in result.values()]))
        print('Found', found, 'results')
        total = total + found
        offset = offset + limit
    return tuples
```

A simple program gets them all

ASK query

- An ASK query returns True if it can be satisfied and False if not
- Note: WHERE token is completely optional, capitalization of keywords not significant
- Was Barack Obama born in the US?

```
ask {
  {dbr:Barack_Obama dbo:birthPlace dbr:United_States}
  UNION
  {dbr:Barack_Obama
      dbo:birthPlace/dbo:subdivision*/dbo:country
      dbr:United_States}
}
```

TRY IT

Note property path in query

- SPARQL 1.1 supports property paths with a / between properties
- Properties can be followed by a * (any number) or + (one or more) indicate repetition
- This looks for a birthplace that's (a subdivision* of) in the country Unted_states

```
ask {
 {dbr:Barack_Obama dbo:birthPlace dbr:United_States}
 UNION
 {dbr:Barack_Obama
     dbo:birthPlace/dbo:subdivision*/dbo:country
     dbr:United_States}  }
```

TRY IT

More Property Paths Syntax

- Property paths allow triple patterns to match arbitrary-length paths through a graph
- Predicates are combined with regular-expression-like operators:

Construct	Meaning
path1/path2	Forwards path (path1 followed by path2)
^path1	Backwards path (object to subject)
path1|path2	Either path1 or path2
path1*	path1, repeated zero or more times
path1+	path1, repeated one or more times
path1?	path1, optionally
path1{m,n}	At least m and no more than n occurrences of path1
path1{n}	Exactly n occurrences of path1
path1{m,}	At least m occurrences of path1
path1{,n}	At most n occurrences of path1

DESCRIBE Query

- “Describe ?x” means “tell me everything you know about ?x
- Example: Describe Alan Turing …

```
DESCRIBE <http://dbpedia.org/resource/Alan_Turing>
```

-- or –

```
PREFIX dbr: <http://dbpedia.org/resource/>
DESCRIBE dbr:Alan_Turing
```

- Returns a collection of ~1500 triples in which dbr:Alan_Turing is either the subject or object

Describes's results?

- The DAWG did not reach a consensus on what describe should return
- Possibilities include
  - All triples where the variable bindings are mentioned
  - All triples where the bindings are the subject
  - Something else
- What is useful might depend on the application or the amount of data involved
- So it was left to the implementation

Construct query (1)

- Having a result form that produces an RDF graph is a good idea
- It enables on to construct systems by using the output of one SPARQL query as the data over which another query works
- This kind of capability was a powerful one for relational databases

Construct query (2)

- Actors & directors or producers they’ve worked for

```
PREFIX ex: <http://example.org/>
CONSTRUCT {?actor ex:workedFor ?directorOrProducer}
WHERE {
 ?film a dbo:Film;
       dbo:director|dbo:producer ?directorOrProducer;
       dbo:starring ?actor}
```

- Returns a graph with ~31,000 triples

TRY IT

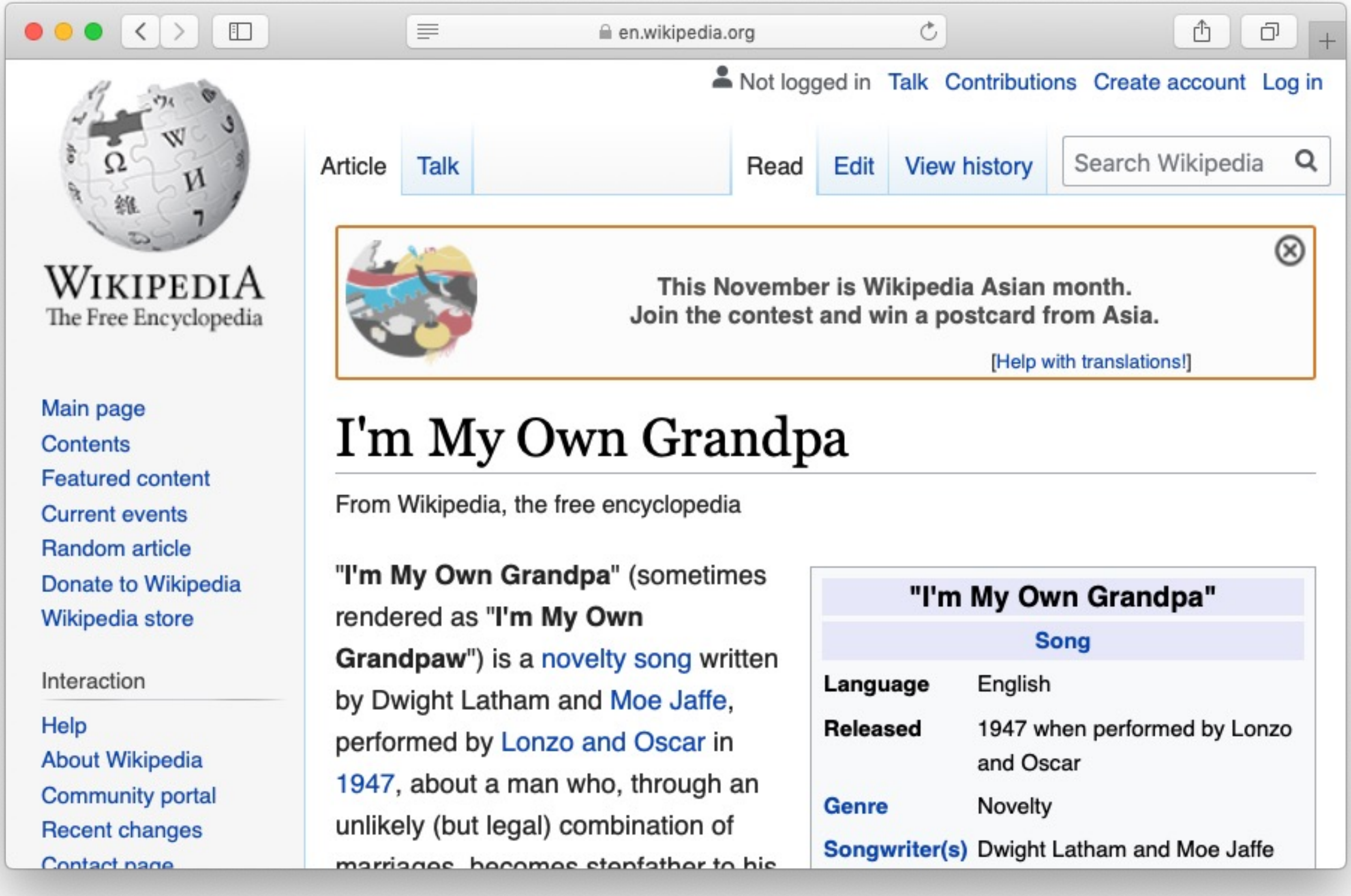
Example: finding missing inverses

- DBpedia is missing many inverse relations, including more than 10k missing spouse relations
- This creates a graph of all the missing ones, which can be added back to the KG via UPDATE ADD

```
PREFIX dbo: <http://dbpedia.org/ontology/>
CONSTRUCT { ?p2 dbo:spouse ?p1. }
WHERE {?p1 dbo:spouse ?p2.
      FILTER NOT EXISTS {?p2 dbo:spouse ?p1}}
```

- Not the **NOT EXISTS** operator that succeeds iff its graph pattern is not satisfiable

I’m my own grandpa



This November is Wikipedia Asian month.
Join the contest and win a postcard from Asia.

[Help with translations!]

I'm My Own Grandpa

From Wikipedia, the free encyclopedia

"**I'm My Own Grandpa**" (sometimes rendered as "**I'm My Own Grandpaw**") is a novelty song written by Dwight Latham and Moe Jaffe, performed by Lonzo and Oscar in 1947, about a man who, through an unlikely (but legal) combination of marriages, becomes stepfather to his

"I'm My Own Grandpa"	
Song	
Language	English
Released	1947 when performed by Lonzo and Oscar
Genre	Novelty
Songwriter(s)	Dwight Latham and Moe Jaffe

HEAR IT

I’m my own grandparent

```
# find people who are their own grandparent
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT (count(distinct ?P) as ?N)
WHERE {?P dbo:child/dbo:child ?P}
```

TRY IT

See the 297 results

I’m my own ancestor

```
# find people who are their own ancestor
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT (count(distinct ?P) as ?N)
WHERE {?P dbo:child/dbo:child* ?P}
```

TRY IT

See 318 results

UPDATE Query

- **Simple insert**

  ```
  INSERT DATA { :book1 :title "A new book" ; :creator "A.N.Other" . }
  ```

- **Simple delete**

  ```
  DELETE DATA { :book1 dc:title "A new book" . }
  ```

- Combine the two for a modification, optionally guided by the results of a graph pattern

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  DELETE { ?person foaf:givenName 'Bill’ }
  INSERT { ?person foaf:givenName 'William’ }
  WHERE { ?person foaf:givenName 'Bill' }
  ```

Aggregation Operators

- SPARQL 1.1 added many aggregation operators, like count, min, max, sum, avg, sample, group_concat…
- Generally used in the results specification, where the final values are known

  ```
  SELECT (COUNT(?film) AS ?numberOfFilms)
  WHERE {?film a dbo:Film .}
  ```

- This finds 129,980 films

Group by

- GROUP BY breaks the query's result set into groups before applying the aggregate functions
- Find Barack Obama’s properties, group them by property and show number of values for each

```
SELECT ?p (COUNT(?p) as ?number) WHERE {
  dbr:Barack_Obama ?p ?o .
}
GROUP BY ?p
ORDER BY DESC(count(?p))
```

TRY IT

COUNT aggregation operator (1)

```
# How many instances of a dbo:film in DBpedia?
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT (COUNT(?film) AS ?numberOfFilms)
WHERE {?film a dbo:Film .}
```

- Returns 171730

TRY IT

COUNT aggregation operator (2)

- How may films has each director in DBpedia made?
- We need to know how to identify a director
- We can use the strategy of looking at a known director, e.g., Billy Wilder and seeing if there’s an appropriate type
- Another option is to collect objects in a relation dbo:director relation to a film

COUNT aggregation operator (3)

```
# How many films has each director made
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?dir (COUNT (DISTINCT ?film) as ?num)
WHERE {?film a dbo:Film;
          dbo:director ?dir. }
GROUP BY ?dir
ORDER BY DESC(?num)
```

TRY IT

COUNT aggregation operator (4)

```
# How many films has each director made, with example
SELECT ?director
              (COUNT (DISTINCT ?film) as ?num)
              (SAMPLE (?film) as ?example)
WHERE {?film a dbo:Film;
             dbo:director ?director. }
GROUP BY ?director
ORDER BY DESC(?num)
```

TRY IT

COUNT aggregation operator (5)

```
# How many films has each director made, with example
SELECT ?director
              (COUNT (DISTINCT ?film) as ?num)
              (SAMPLE (?film) as ?example)
WHERE {?film a dbo:Film;
             dbo:director ?director. }
GROUP BY ?director
ORDER BY DESC(?num)
LIMIT 10000 OFFSET 10000
```

# Get More!

TRY IT

SPARQL 1.1 Query Forms

- New query forms were added in SPARQL 1.1
- They support modifying a graph, easing the use of programs to update existing graphs

SPARQL Update Language Statements
INSERT DATA { *triples* }
DELETE DATA {*triples*}
[DELETE { *template* }] [INSERT { *template* }] WHERE { *pattern* }
LOAD <*uri*> [INTO GRAPH <*uri*>]
CLEAR GRAPH <*uri*>
CREATE GRAPH <*uri*>
DROP GRAPH <*uri*>

Inference via SPARQL

- We can test if triples exisit and use this for inference
- E.g., add inverse spouse relations that are missing:

```
INSERT { ?p2 dbo:spouse ?p1. }
WHERE {?p1 dbo:spouse ?p2.
         FILTER NOT EXISTS {?p2 dbo:spouse ?p1}}
```

- SPIN and SHACL are newer systems to represent simple constraint & inference rules that are done by SPARQL
  - A big feature is that the rules are represented in the graph

SPARQL 1.1 Additions

- SPARQ 1.1 added many more features …
  - Subqueries
  - Negation: MINUS
  - Federated queries that access multiple endpoints
- Data you want to extract from an RDF graph can probably be returned by one query
  - Might be a complicated one, though …
- Search web for SPARQL tricks or this book

Some Public SPARQL Endpoints

Name	URL	What's there?
SPARQLer	http://sparql.org/sparql.html	General-purpose query endpoint for Web-accessible data
DBPedia	http://dbpedia.org/sparql	Extensive RDF data from Wikipedia
DBLP	http://www4.wiwiss.fu-berlin.de/dblp/snorql/	Bibliographic data from computer science journals and conferences
LinkedMDB	http://data.linkedmdb.org/sparql	Films, actors, directors, writers, producers, etc.
World Factbook	http://www4.wiwiss.fu-berlin.de/factbook/snorql/	Country statistics from the CIA World Factbook
bio2rdf	http://bio2rdf.org/sparql	Bioinformatics data from around 40 public databases

Sparql Resources

- The SPARQL Specification
  - http://www.w3.org/TR/rdf-sparql-query/
- SPARQL implementations
  - http://esw.w3.org/topic/SparqlImplementations
- SPARQL endpoints
  - http://esw.w3.org/topic/SparqlEndpoints
- SPARQL Frequently Asked Questions
  - http://www.thefigtrees.net/lee/sw/sparql-faq
- SPARQL Working Group
  - http://www.w3.org/2009/sparql/wiki/
- Common SPARQL extensions
  - http://esw.w3.org/topic/SPARQL/Extensions