

SPARQL on Wikidata

MPRI 2.26.2: Web Data Management

Antoine Amarilli

Friday, January 11th

General presentation

- **SPARQL**: query language for RDF graphs
- Many KBs have a **SPARQL endpoint** with a Web interface
 - **Wikidata** <https://query.wikidata.org/>:
 - serves around **5M** requests per day,
 - of which around **100k** are by humans
 - **DBpedia** <https://dbpedia.org/sparql>
 - **YAGO** <https://linkeddata1.calcul.u-psud.fr/sparql>
 - **DBLP** <http://dblp.rkbexplorer.com/sparql/>
 - **INSEE** <http://rdf.insee.fr/sparql>
 - **BNF** <http://data.bnf.fr/sparql/>

Basic shape of a query

Prefixes are declared with PREFIX (like in Turtle)

```
SELECT ?x ?y ?z WHERE {  
  # ... facts about ?x ?y ?z ...  
  # syntax is similar to Turtle  
}
```

- SELECT ... WHERE {...}, most queries
- ASK WHERE {...}, for Boolean queries
- CONSTRUCT {...} WHERE {...}, to build a graph (see later)
- DESCRIBE <entity>: return an implementation-defined description of an entity (set of facts about the entity)

Facts in the query body

- You can put **facts** in the body of the query, using
 - **variables** (selected in the query or not)
 - **constants** (from any namespace, or raw URIs)
 - **literals** (in any format)
 - **blank nodes** (e.g., with brackets; anonymous variables)
- For every **way** to assign the variables such that the pattern holds, produce a **result**

```
SELECT ?item WHERE  
{  
  ?item wdt:P31 wd:Q146.  
}
```

Property paths

Property paths allow us to use **regular expressions** in facts:

```
SELECT ?descendant
```

```
WHERE
```

```
{
```

```
  ?descendant (wdt:P22|wdt:P25)+ wd:Q1339.
```

```
}
```

Wikidata-specific techniques

- To avoid **typing** entity and property numbers in the **query**, you can use **Ctrl+Space** to search by name
- To avoid them in the **results**, you can use the **label service**:
SERVICE wikibase:label
`{ bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }`
- You can get this incantation with **Ctrl-Space**
- Automatically creates a variable `?xLabel` for every variable `?x`
- This is done **late** in the evaluation (e.g., cannot filter on them)

LIMIT and ORDER

- To avoid returning **too many results**, you can use **LIMIT**
- Also useful to **speed up the query**

```
SELECT * WHERE {  
  ?s ?p ?o .  
} LIMIT 1000
```

- We can also **sort** (using the implicit order on types)

```
SELECT ?country ?population WHERE  
{  
  ?country wdt:P31/wdt:P279* wd:Q3624078;  
           wdt:P1082 ?population.  
}  
ORDER BY DESC(?population)  
LIMIT 10
```

OPTIONAL

- SPARQL has an **open-world semantics** on missing data
- Extend results **if possible** and **keep them as-is** otherwise

```
SELECT ?book ?title
WHERE {
  ?book wdt:P50 wd:Q35610.
  ?book wdt:P1476 ?title.
  OPTIONAL { ?book wdt:P123 ?publisher. }
}
```

- **Semantics:**
 - Consider **every solution** of what precedes
 - For each solution, **run the optional query**
 - If it produces outputs, **combine them** with the solution
 - If it does not, leave the solution **as-is**

OPTIONAL subtleties

- The **order** of patterns matter!

```
# selects people, which may have an image
```

```
SELECT * WHERE {  
  ?person rdf:type ex:Person  
  OPTIONAL { ?person ex:image ?image }  
}
```

```
# selects only people with an image
```

```
SELECT * WHERE {  
  OPTIONAL { ?person ex:image ?image }  
  ?person rdf:type ex:Person  
}
```

- Makes the **computational complexity** much worse (query evaluation is **PSPACE-complete** in combined complexity)

<https://arxiv.org/abs/0812.3788>

FILTER

We can also **filter** query results based on conditions that cannot easily be expressed as a pattern of triples:

- Order **comparisons** on a value (e.g., “after 2015”); beware of **types!**
- String comparison and **regexps**
- Testing if a string is **in some set** (also: **VALUES**)
- Booleans, arithmetic operations, etc.

```
SELECT ?item ?bblid
```

```
WHERE {
```

```
  ?item wdt:P2580 ?bblid .
```

```
  FILTER(!REGEX(STR(?bblid),
```

```
    "^[A-Za-z][-0-9A-Za-z]{1,}$"))
```

```
}
```

Aggregates

- Like in SQL:
 - Group the results according to the value of some variables
 - Compute some **aggregate** within each group

```
SELECT ?country (MAX(?population) AS ?maxPop)
WHERE
{
  ?city wdt:P31/wdt:P279* wd:Q515;
        wdt:P17 ?country;
        wdt:P1082 ?population.
}
GROUP BY ?country
```

- Can use **HAVING** to filter out some groups
- Can be useful with **nested queries**

DISTINCT

- We can get **duplicate results**, e.g.,
 - When **projecting away variables**
 - When **multiple paths** exist

```
SELECT ?class WHERE {  
    wd:Q6602 wdt:P31/wdt:P279* ?class .  
}
```

- We can **remove duplicates** with `SELECT DISTINCT`
- For **projection** it can be faster to use `FILTER EXISTS`
- `SELECT REDUCED` (in theory) to tell the optimizer you **do not care about multiplicity**
- Also `COUNT(DISTINCT ?x)` to count **distinct values**

MINUS

- We can **subtract** a set of results from another with **MINUS**
- Often, what we really want is **FILTER NOT EXISTS**, i.e., testing that a match **cannot** be extended

Reified triples: quantifiers, sources, etc. (Wikidata-specific)

- The default way to talk about Wikidata facts does **not** talk about ranks, qualifiers, sources
- By default, only shows the facts with the **highest available rank**
- To talk about ranks, qualifiers, sources, you need to **reify** and use a different namespace

```
wd:Q12418 p:P186 ?stmt1. # Mona Lisa: material used: ?stmt1
?stmt1 ps:P186 wd:Q296955. # value: oil paint
```

```
wd:Q12418 p:P186 ?stmt2. # Mona Lisa: material used: ?stmt2
?stmt2 ps:P186 wd:Q291034. # value: poplar wood
?stmt2 pq:P518 wd:Q861259. # qualifier: applies to part: painting surface
```

```
wd:Q12418 p:P186 ?stmt3. # Mona Lisa: material used: ?stmt3
?stmt3 ps:P186 wd:Q287. # value: wood
?stmt3 pq:P518 wd:Q1737943. # qualifier: applies to part: stretcher bar
?stmt3 pq:P580 1951. # qualifier: start time: 1951 (pseudo-syntax)
```

- This is **not** the same as usual RDF reification!

Some value, and no value

- **Some value** (known to be unknown) are represented by **blank nodes**

[https://www.wikidata.org/wiki/Wikidata:](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know)

[SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know)

- **No value** (known not to exist) represented with **rdf:type**

[https://www.wikidata.org/wiki/Wikidata:](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know)

[SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know](https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples#Humans_whose_gender_we_know_we_don't_know)

Other features

- **CONSTRUCT** keyword to **build RDF facts** to be returned instead of rows
- **UNION**, to union the results of two queries (may have different variables)
- **BIND**, to introduce new variables for arithmetic expressions, etc. (but may harm query optimization)
- **BOUND**, to test if a variable has been bound (e.g., with **OPTIONAL**)
- Renaming variables in the **SELECT** clause with **AS**
- **VALUES** to define tables of values in results
- Querying from **multiple files** (not useful for SPARQL endpoints)
- Many other keywords...

Vizualisations

- `BlazeGraph` allows different `visualizations` of the resulting data
- Can change the default visualization with a `#defaultView` comment
- Some possible views (cf links):
 - Map
 - Timeline
 - Dimensions
 - Graph (and way to retrieve deprecated values)
 - Tree

Federation

- You can query **external SPARQL endpoints** with **SERVICE**
- **Semantics**: the service runs the query for each answer
- Tricky in terms of **performance** and **uptime**
- Wikidata allows federation with a **specific list of other sources**

www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual/SPARQL_Federation_endpoints

- Example: OSM has a SPARQL endpoint <https://sophox.org/>
- Example (link): Places near me with known opening hours
- **Some special services**:
 - Get **labels** of elements **SERVICE wikibase:label**
 - Search **around point** **SERVICE wikibase:around**
 - Search **within box** **SERVICE wikibase:box**
 - Query Wikipedia from Wikibase

https://www.mediawiki.org/wiki/Wikidata_Query_Service/User_Manual/MWAPI

Useful for page links, categories...

- Another experimental way <https://www.mediawiki.org/wiki/MW2SPARQL>

Performance

- Running SPARQL queries is **often slow**
 - Main reasons:
 - SPARQL engines are not as **mature** as, e.g., relational databases
 - No **fixed schema** makes it hard to leverage properties of the data
 - No fixed choice of **indexes**
 - Cardinality estimation is complicated
- SPARQL endpoints such as the Wikidata Query Service impose a **timeout** (60 seconds)

Optimizing SPARQL queries

- **Complicated issue**, and depends on the underlying engine!
- Blazegraph has support for some optimization, e.g., reordering the query (sometimes needs to be disabled)
- **Explain mode** to see the execution plan of a query (ad hoc)
https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/query_optimization
- Blazegraph supports **query hints** to the optimizer (very ad hoc)
- Paths of properties and **stars** are especially **problematic**

Wikidata Query Lag

- The data on `query.wikidata.org` is a **mirror** of Wikidata
- Synchronisation is automated but can have **delays**, e.g., 1 hour
- There can be **inconsistencies** across mirrors of `query.wikidata.org`, hence **inconsistent results!**

- `https:`

`//grafana.wikimedia.org/d/000000489/wikidata-query-service?orgId=1&panelId=8&fullscreen`

Alternatives to SPARQL

- **SPARUL**: support for modifying the dataset (INSERT/DELETE)
- In the **property graph** model: **Cypher** (Neo4j) and **Gremlin**
- **GraphQL**

Slide acknowledgements

- Slides 7, 8, 10, 11, 14, 5:
https://www.wikidata.org/wiki/Wikidata:SPARQL_tutorial#Qualifiers
- Slide 9: https://wiki.blazegraph.com/wiki/index.php/SPARQL_Order_Matters
- Slide 17: https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples
- Many thanks to Thomas Pellissier-Tanon for his helpful feedback