SPARQL Protocol and RDF Query Language

Pasquale Minervini
p.minervini@ed.ac.uk

Main References

SPARQL stands for **SPARQL Protocol and RDF Query Language**

1. SPARQL 1.1 W3C Recommendation from March 2013: https://www.w3.org/TR/sparql11-overview/
2. Apache Jena: SPARQL Tutorial: https://jena.apache.org/tutorials/sparql.html
3. Feigenbaum et al. SPARQL By Example: https://cambridgesemantics.com/blog/semantic-university/learn-sparql/sparql-by-example/
4. Hitzler et al. Foundations of Semantic Web Technologies, Ch. 7 (mail me if you need to consult this one!)

Structure of SPARQL SELECT Queries

A SPARQL query is composed by:

- **Prefix declarations**, for abbreviating URIs;
- **Result clause**, identifying what information to return from the query;
- **Dataset definition**, stating what RDF graphs are being queried;
- **Query pattern**, specifying what to query for in the underlying dataset;
- **Query modifiers**: slicing, ordering, and otherwise rearranging query results.

```
# prefix declarations
PREFIX foaf: <http://xlns.com/
foaf/0.1/>
…
# result clause
SELECT …
# dataset definition
FROM …
# query pattern
WHERE {
   …
}
# query modifiers
ORDER BY …
```

SPARQL Architecture and Endpoints

SPARQL queries are executed against **RDF datasets**

- RDF datasets are composed by one or more RDF graphs

A **SPARQL endpoint** accepts queries and returns results via HTTP

The results are returned/rendered in a variety of formats:

- SPARQL specifies an **XML** vocabulary for result sets
- A JSON port of the XML vocabulary
- Certain SPARQL result clauses trigger **RDF** responses
- **HTML** often as an XSLT transformation of the XML
- **CSV**

Example — RDF Graph (Turtle)

```
@base <http://foo.bar/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<#Cat> a rdfs:Class ; rdfs:label "Cat" .

<#owns> a rdf:Property ; rdfs:label "owns" ;
rdfs:domain foaf:Person ; rdfs:range <#Cat>

<#victor> a <#Cat> ; foaf:name "Victor"
<#gaston> a <#Cat> ; foaf:name "Gaston"
<#bettina> a <#Cat> ; foaf:name "Bettina"

<#chrdebru> a foaf:Person ; foaf:name "Christophe Debruyne” ;
   <#owns> <#victor> ; <#owns> <#gaston> ; <#owns> <#bettina> .
```

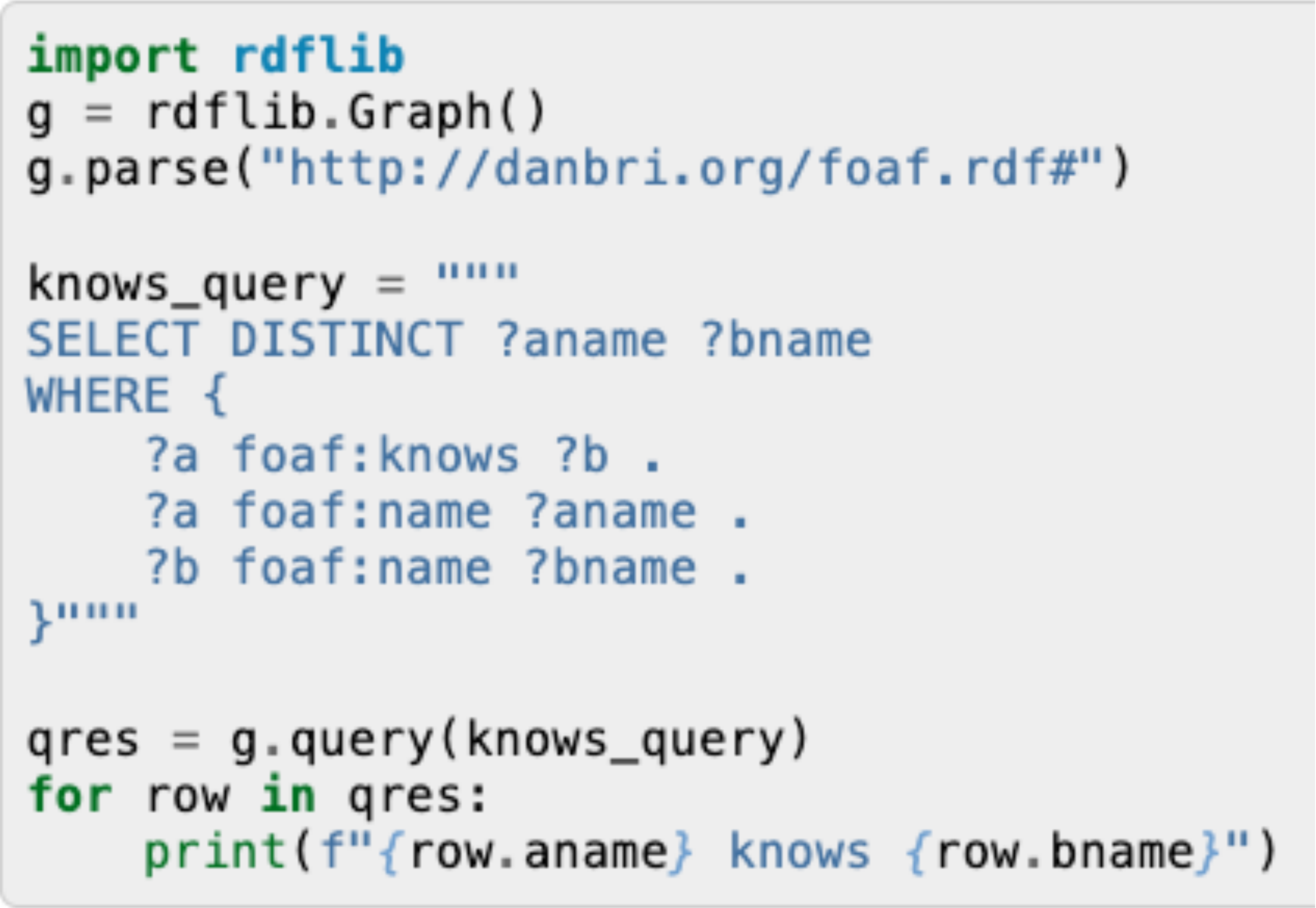
Simple SPARQL Queries

Finding all cats and their names:

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {
   ?cat a ex:Cat .
   ?cat foaf:name ?name .
}
```

- Variables start with a `?` and can match any term, resource or literal;
- Triple patterns are just like triples, except that any of the parts of a triple can be replaced with a variable;
- The `SELECT` result clause returns a table of variables and values that satisfy the query;
- `a` is just syntactic sugar for `rdf:type`.

Simple SPARQL Queries

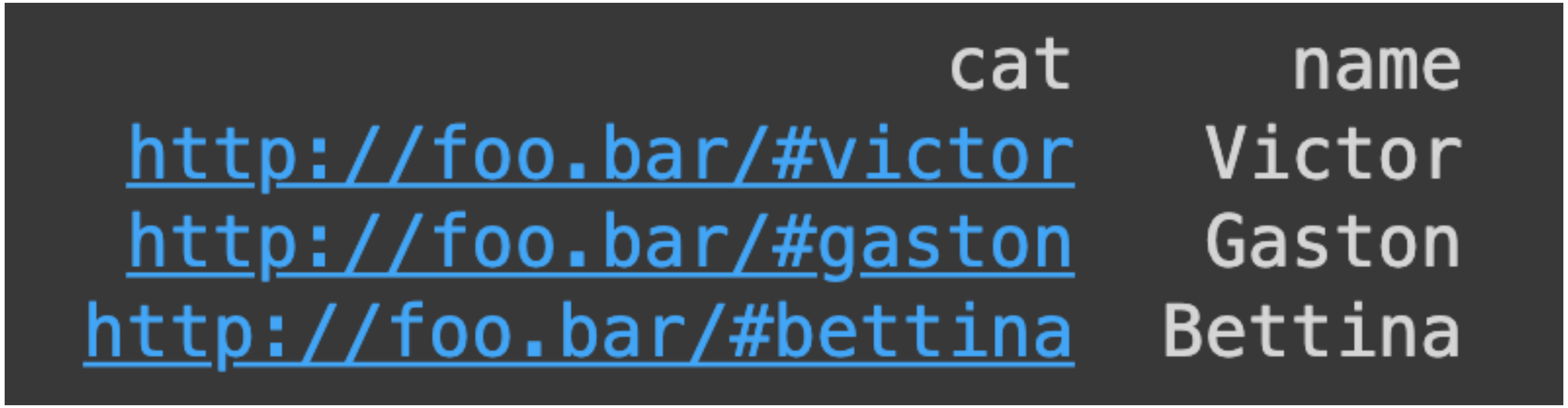


```
import rdflib
g = rdflib.Graph()
g.parse("http://danbri.org/foaf.rdf#")

knows_query = """
SELECT DISTINCT ?aname ?bname
WHERE {
    ?a foaf:knows ?b .
    ?a foaf:name ?aname .
    ?b foaf:name ?bname .
}"""

qres = g.query(knows_query)
for row in qres:
    print(f"{row.aname} knows {row.bname}")
```

Simple SPARQL Queries

Finding all cats and their names:

```
PREFIX ex: <http://foo.bar/#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?cat ?name
WHERE {
   ?cat a ex:Cat .
   ?cat foaf:name ?name .
}
```



```
                       cat     name
  http://foo.bar/#victor     Victor
  http://foo.bar/#gaston     Gaston
http://foo.bar/#bettina    Bettina
```

Querying SPARQL Endpoints

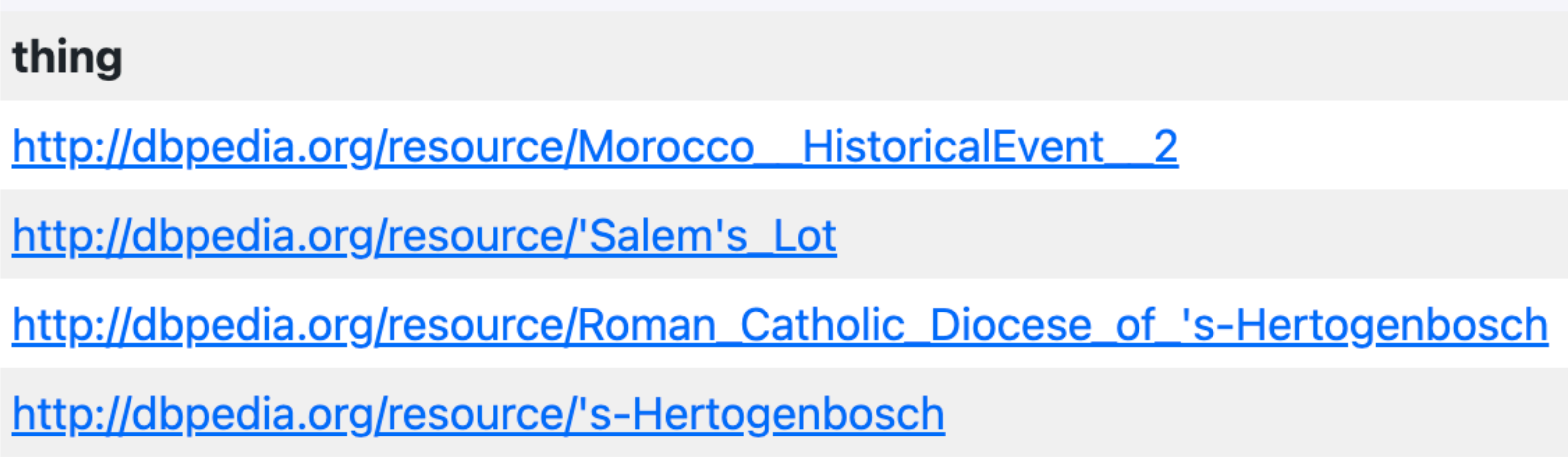
DBpedia is an effort to expose the knowledge in Wikipedia as a RDF graph

- SPARQL endpoint: https://dbpedia.org/sparql
- “Find me 50 things with names in DBpedia”

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?thing
WHERE {
  ?thing foaf:name ?name .
} LIMIT 50
```

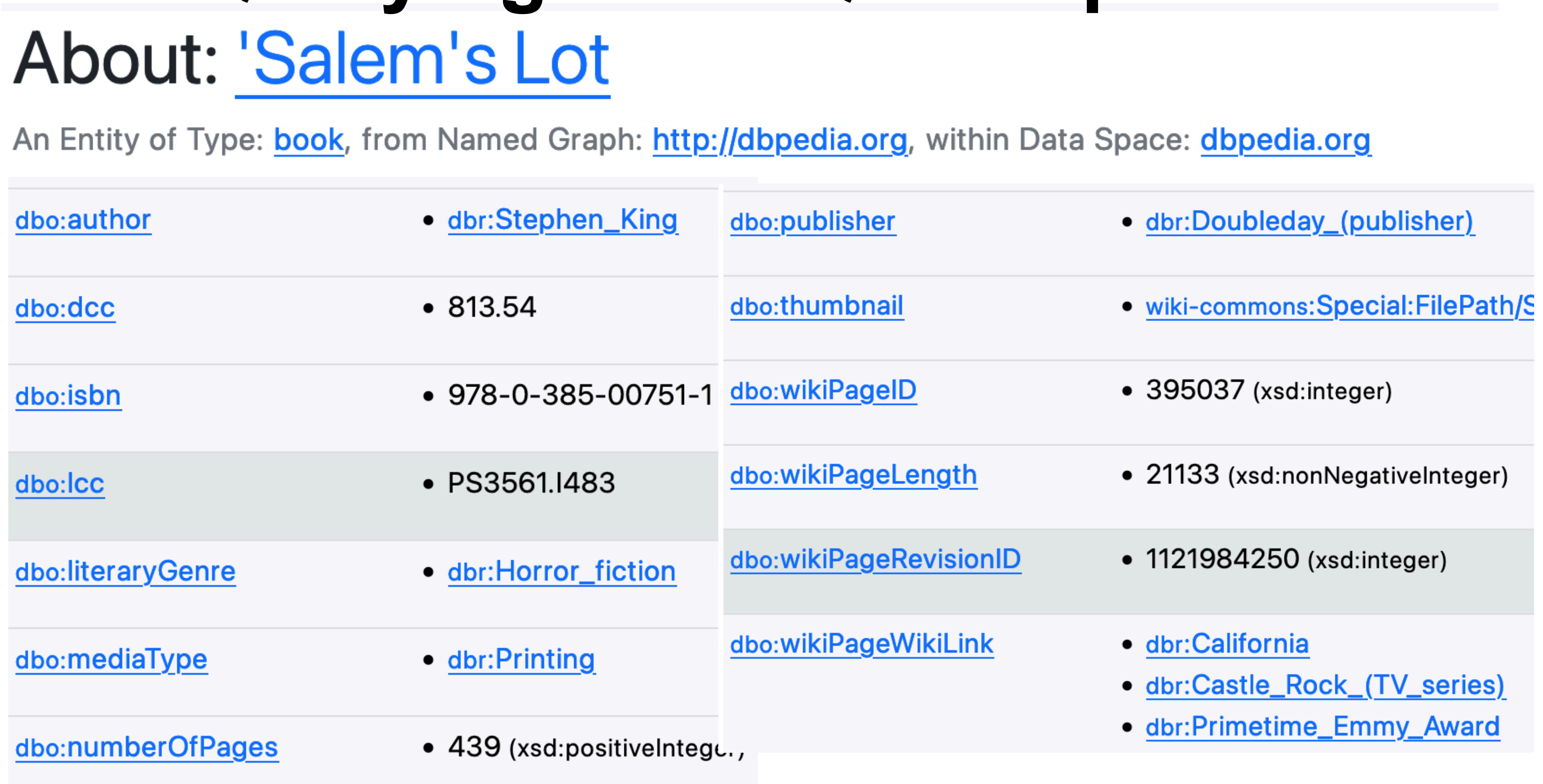
Querying SPARQL Endpoints

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?thing
WHERE {
   ?thing foaf:name ?name .
} LIMIT 50
```



thing
http://dbpedia.org/resource/Morocco__HistoricalEvent__2
http://dbpedia.org/resource/'Salem's_Lot
http://dbpedia.org/resource/Roman_Catholic_Diocese_of_'s-Hertogenbosch
http://dbpedia.org/resource/'s-Hertogenbosch

Querying SPARQL Endpoints



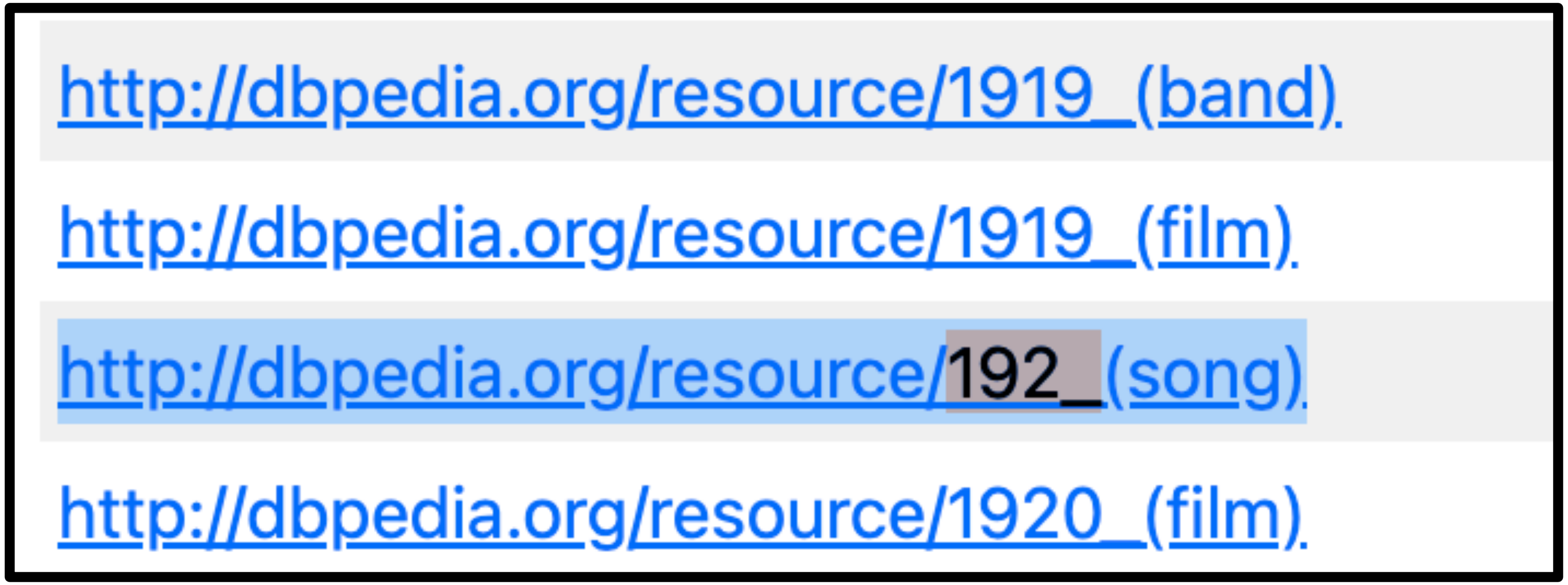
About: 'Salem's Lot

An Entity of Type: book, from Named Graph: http://dbpedia.org, within Data Space: dbpedia.org

dbo:author	• dbr:Stephen_King
dbo:dcc	• 813.54
dbo:isbn	• 978-0-385-00751-1
dbo:lcc	• PS3561.I483
dbo:literaryGenre	• dbr:Horror_fiction
dbo:mediaType	• dbr:Printing
dbo:numberOfPages	• 439 (xsd:positiveIntege

dbo:publisher	• dbr:Doubleday_(publisher)
dbo:thumbnail	• wiki-commons:Special:FilePath/S
dbo:wikiPageID	• 395037 (xsd:integer)
dbo:wikiPageLength	• 21133 (xsd:nonNegativeInteger)
dbo:wikiPageRevisionID	• 1121984250 (xsd:integer)
dbo:wikiPageWikiLink	• dbr:California • dbr:Castle_Rock_(TV_series) • dbr:Primetime_Emmy_Award

Querying SPARQL Endpoints

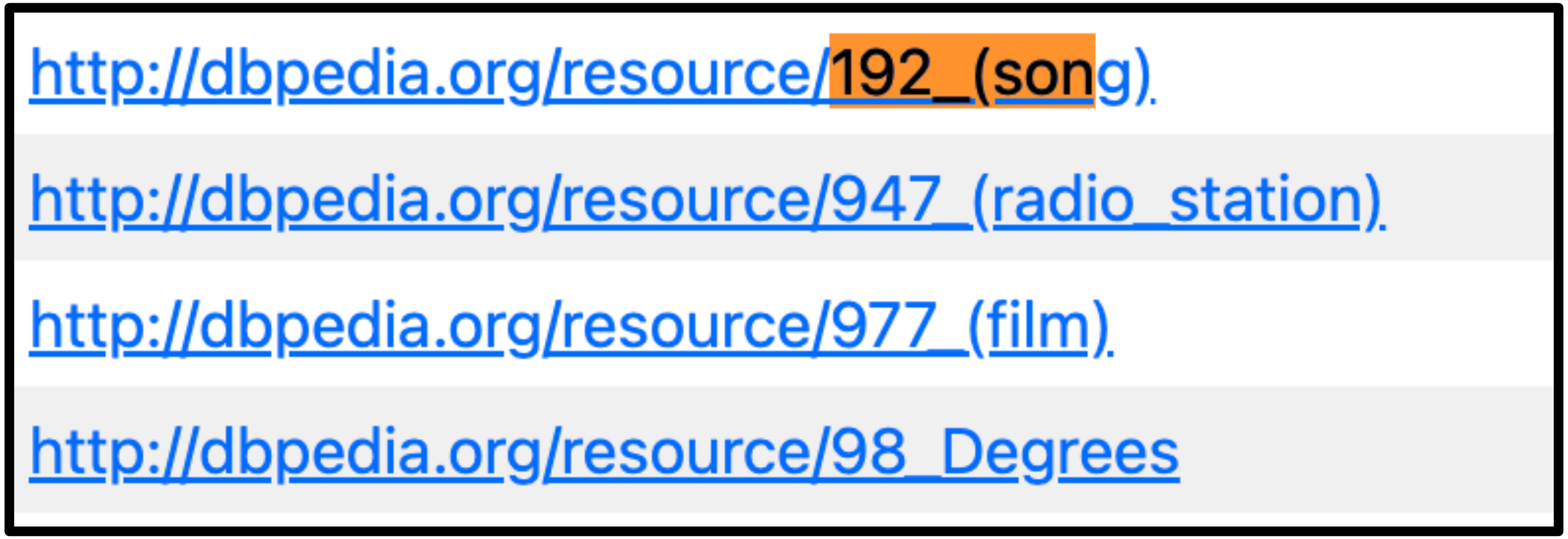


http://dbpedia.org/resource/1919_(band)

http://dbpedia.org/resource/1919_(film)

http://dbpedia.org/resource/192_(song)

http://dbpedia.org/resource/1920_(film)



http://dbpedia.org/resource/192_(song)

http://dbpedia.org/resource/947_(radio_station)

http://dbpedia.org/resource/977_(film)

http://dbpedia.org/resource/98_Degrees

Querying SPARQL Endpoints

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?thing
WHERE {
  ?thing foaf:name ?name .
} LIMIT 50
```

Solution modifiers:

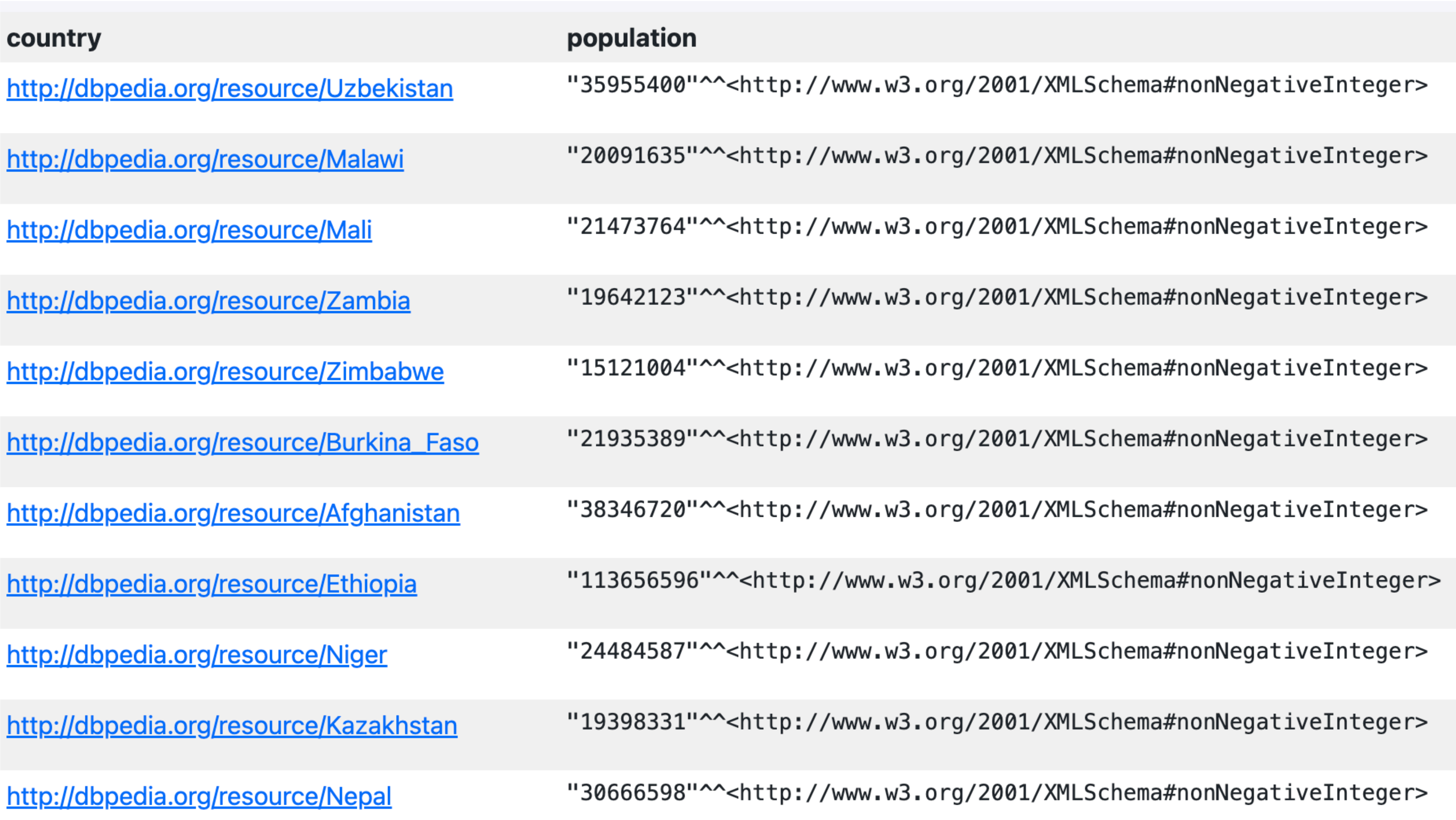
- `LIMIT`, limits the number of returned rows
- `ORDER BY`, sorting
- `OFFSET`, used together with `LIMIT` and `ORDER BY` for slicing, e.g., paging

Filtering SPARQL Query Results

```
PREFIX [..]
SELECT ?country_name ?population
WHERE {
  ?country a yago:WikicatLandlockedCountries;
    rdfs:label ?country_name;
    dbo:populationTotal ?population .
  FILTER (?population > 15000000)
}
```

- `FILTER` constraints use boolean conditions to filter out unwanted results
- We use `;` (semicolon) for abbreviating some triples.

Filtering SPARQL Query Results



country	population
http://dbpedia.org/resource/Uzbekistan	"35955400"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Malawi	"20091635"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Mali	"21473764"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Zambia	"19642123"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Zimbabwe	"15121004"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Burkina_Faso	"21935389"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Afghanistan	"38346720"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Ethiopia	"113656596"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Niger	"24484587"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Kazakhstan	"19398331"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
http://dbpedia.org/resource/Nepal	"30666598"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Filtering SPARQL Query Results

- Logical: !, &&, ||
- Math: +, -, *, /
- Comparison: =, !=, >, <, IN, NOT IN...
- SPARQL tests: isIRI, isURI, isBlank, isLiteral, isNumeric, bound
- SPARQL accessors: str, lang, datatype
- Other: sameTerm, langMatches, regex, REPLACE
- Conditionals: IF, COALESCE, EXISTS, NOT EXISTS
- Constructors: URI, BNODE, STRDT, STRLANG, UUID, STRUUID
- Strings: STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDS, CONTAINS, STRBEFORE, STRAFTER, CONCAT, ENCODE_FOR_URI
- More math: abs, round, ceil, floor, RAND
- [..]

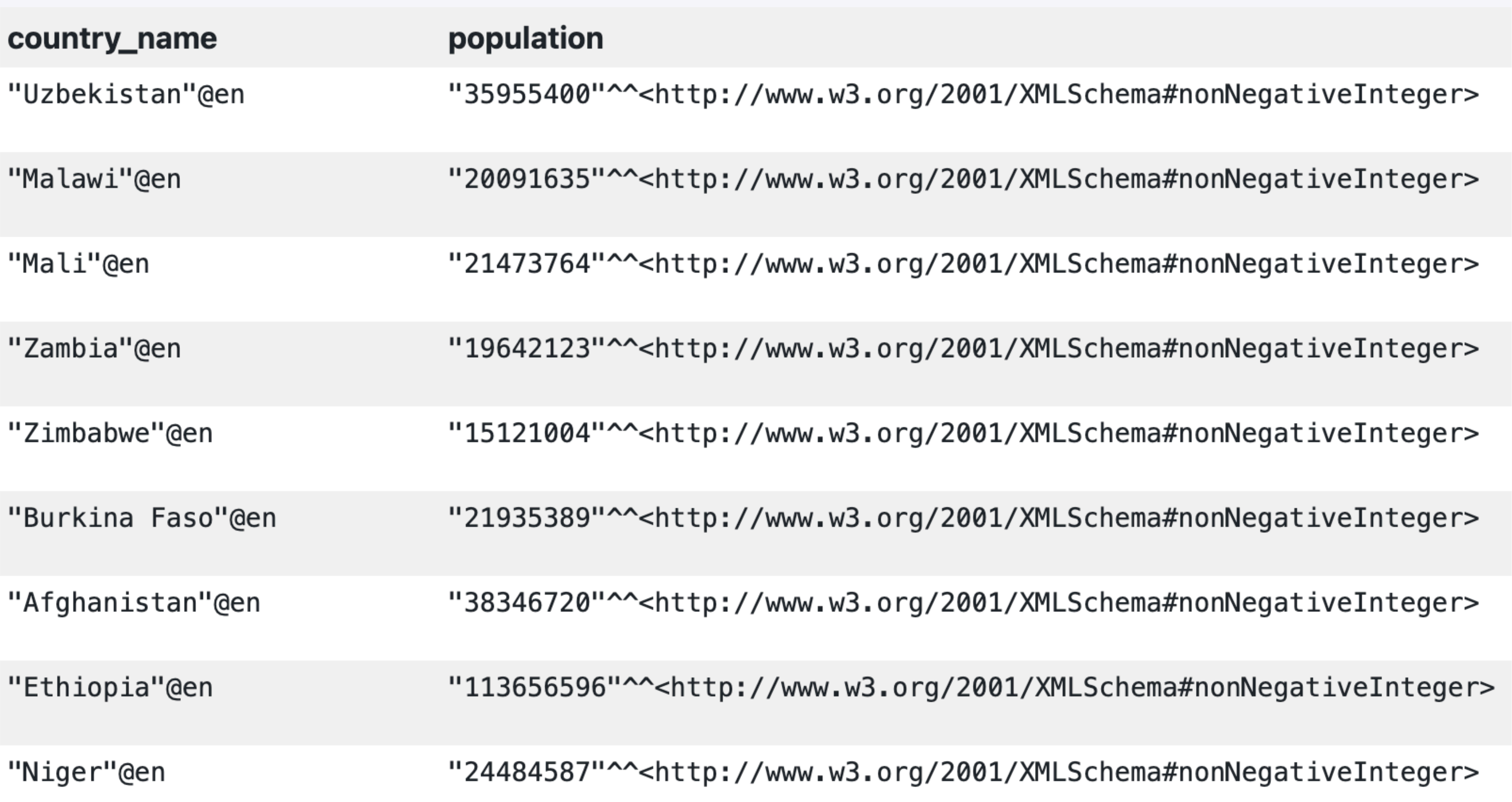
Filtering SPARQL Query Results

```
PREFIX [..]
SELECT ?country_name ?population
WHERE {
  ?country a yago:WikicatLandlockedCountries;
    rdfs:label ?country_name;
    dbo:populationTotal ?population .
  FILTER (?population > 15000000)
  FILTER (langMatches(lang(?country_name), “en”))
}
```

Filtering SPARQL Query Results

```
PREFIX [..]
SELECT ?country_name ?population
WHERE {
  ?country a yago:WikicatLandlockedCountries;
    rdfs:label ?country_name;
    dbo:populationTotal ?population .
  FILTER (?population > 15000000) &&
    (langMatches(lang(?country_name), “en”))
}
```

Filtering SPARQL Query Results



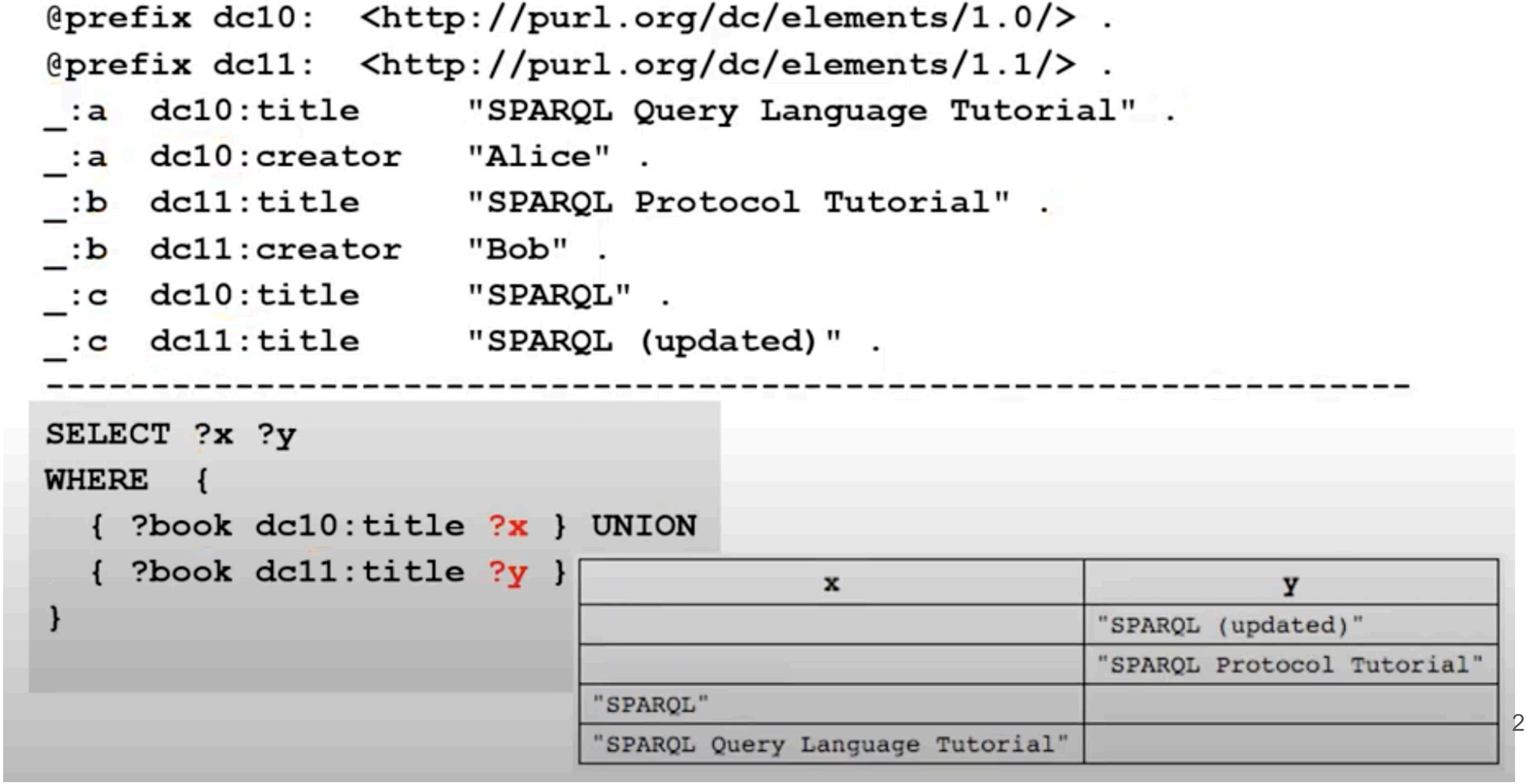
country_name	population
"Uzbekistan"@en	"35955400"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Malawi"@en	"20091635"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Mali"@en	"21473764"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Zambia"@en	"19642123"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Zimbabwe"@en	"15121004"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Burkina Faso"@en	"21935389"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Afghanistan"@en	"38346720"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Ethiopia"@en	"113656596"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>
"Niger"@en	"24484587"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

UNION and OPTIONAL

```
PREFIX [..]
SELECT ?book ?author
WHERE {
   { ?book ex:author ?author . }
   UNION
   { ?book ex:writer ?author . }
}
```

- `UNION` is useful to match alternatives, by combining two or more *graph patterns*. It is not restricted to triples patterns.

UNION and OPTIONAL



```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .
_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .
_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .
_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
---------------------------------------------------------------
```

```
SELECT ?x ?y
WHERE  {
  { ?book dc10:title ?x } UNION
  { ?book dc11:title ?y }
}
```

x	y
	"SPARQL (updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

UNION and OPTIONAL

```
PREFIX [..]
SELECT ?name ?mbox
WHERE {
   ?x foaf:name ?name .
   OPTIONAL {
      ?x foaf:mbox ?mbox .
   }
}
```

- `OPTIONAL` tries to match a graph pattern, but doesn’t fail the whole query if the optional match fails. If an `OPTIONAL` pattern fails to match a particular solution, any variables in that pattern remain unbound for that solution.

UNION and OPTIONAL

- `UNION` is **associative** and **commutative**

```
{ Pattern1 } UNION { PATTER2 } UNION { Pattern3 }

({ Pattern1 } UNION { PATTER2 }) UNION { Pattern3 }

{ Pattern1 } UNION ({ PATTER2 } UNION { Pattern3 })
```

- `OPTIONAL` is **left-associative**

```
OPTIONAL { ?s ?p1 ?o1 . }
OPTIONAL { ?s ?p2 ?o2 . }
OPTIONAL { ?s ?p3 ?o3 . }
```

```
(
   (
      OPTIONAL { ?s ?p1 ?o1 . }
   ) OPTIONAL { ?s ?p2 ?o2 . }
) OPTIONAL { ?s ?p3 ?o3 . }
```

Unbound Variables

```
SELECT ?subject ?predicate ?object
WHERE {
   ?subject ?predicate ?object .
   FILTER (?some_unbound_variable > 5)
}
```

- When a variable is unbound in a result set, functions that expect a bound variable may produce errors.

- In the above query, `?some_unbound_variable` is unbound because it is not linked to any triple pattern in the `WHERE` clause. This will result in an error because the `FILTER` expression is trying to evaluate a variable that has no value.

Named Graphs

```
SELECT DISTINCT ?name
WHERE { ?person foaf:name ?name .
  GRAPH ?g1 { ?person a foaf:Person }
  GRAPH ?g2 { ?person a foaf:Person }
  GRAPH ?g3 { ?person a foaf:Person }
  FILTER (?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) . }
```

- http://data.semanticweb.org/snorql — dataset of conference events where each graph represents a particular *SWC conference.
- The query is “Find persons that occur in at least three different conferences.”

Named Graphs

- Instead of triples, we now have *quadruples*
- The graph can serve as “context”
- Queries may specify the datasets to be used for matching:
  - `FROM` clauses to refer to default graphs;
  - `FROM NAMED` clauses to refer to named graphs;
  - If `FROM NAMED` clauses are provided without a `FROM` clause, the default empty graph is assumed to be used.

SPARQL CONSTRUCT

```
PREFIX [..]
CONSTRUCT { ?agent a foaf:Agent . }
WHERE {
   { ?agent a foaf:Agent . }
     UNION
   { ?agent a foaf:Person . }
}
```

```
@prefix [..]
ex:bettina a foaf:Agent .
ex:victor a foaf:Agent .
ex:gaston a foaf:Agent .
ex:chrdebru a foaf:Agent .
[..]
```

SPARQL ASK

```
PREFIX [..]
ASK WHERE {
   ?person ex:owns ?cat .
}
```

- Returns True or False
- Do we have cat owners?

SPARQL DESCRIBE

```
PREFIX [..]
DESCRIBE ?cat
WHERE {
   ?cat foaf:name “Victor” .
}
```

```
[..]
@prefix ex: <http://foo.bar/#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
ex:victor a ex:Cat ;
   foaf:name “Victor” .
```

Projected Expressions

```
PREFIX [..]
SELECT ?element ?protons
   (ROUND(?weight) - ?protons AS ?neutrons)
FROM <http://www.daml.org/2003/[..]/PeriodicTable.owl>
WHERE {
   [] a :Element ;
      :atomicNumber ?protons ;
      :atomicWeight ?weight ;
      :name ?element .
} ORDER BY ?protons
```

- `[]` is used for unnamed variables

Assignments

```
PREFIX [..]
SELECT ?element ?protons ?neutrons
FROM <http://www.daml.org/2003/[..]/PeriodicTable.owl>
WHERE {
   [] a :Element ;
     :atomicNumber ?protons ;
     :atomicWeight ?weight ;
     :name ?element .
   BIND(ROUND(?weight) - ?protons AS ?neutrons)
} ORDER BY ?protons
```

- `[]` is used for unnamed variables

Aggregates

```
PREFIX [..]
SELECT ?cat (COUNT(DISTINCT ?thing) AS ?roads)
WHERE {
   ?thing a roads:Road .
   ?thing roads:countPointRoadCategory ?cat .
} GROUP BY ?cat
```

- Aggregation in SPARQL is similar to aggregation in SQL
- `COUNT`, `MIN`, `MAX`, `SUM`, `AVG`, `GROUP_CONCAT`, `SAMPLE`
- The `HAVING` clause can be used to filter the results of the query after applying aggregates, e.g., `HAVING MAX(?price) > 500`

Subqueries

```
PREFIX [..]
SELECT ?name ?email
FROM <http://../webdav/timbl/foaf.rdf>
WHERE {
   {
     SELECT DISTINCT ?person ?name WHERE {
        ?person foaf:name ?name
     } ORDER BY ?name LIMIT 10 OFFSET 10
   }
   OPTIONAL { ?person foaf:mbox ?email }
}
```

Negation in SPARQL 1.0

```
PREFIX [..]
SELECT ?name
WHERE {
   ?person a foaf:Person;
      foaf:name ?name .
   OPTIONAL { ?person rdfs:seeAlso ?url }
   FILTER(!bound(?url))
}
```

- The `OPTIONAL` statement causes the `?url` variable to be unbound for some results, and the `FILTER` statements filters for results where `?url` is unbound.

Negation in SPARQL 1.1

```
PREFIX [..]
SELECT ?name
WHERE {
   ?person a foaf:Person;
      foaf:name ?name .
   MINUS { ?person refs:seeAlso ?url }
}
```

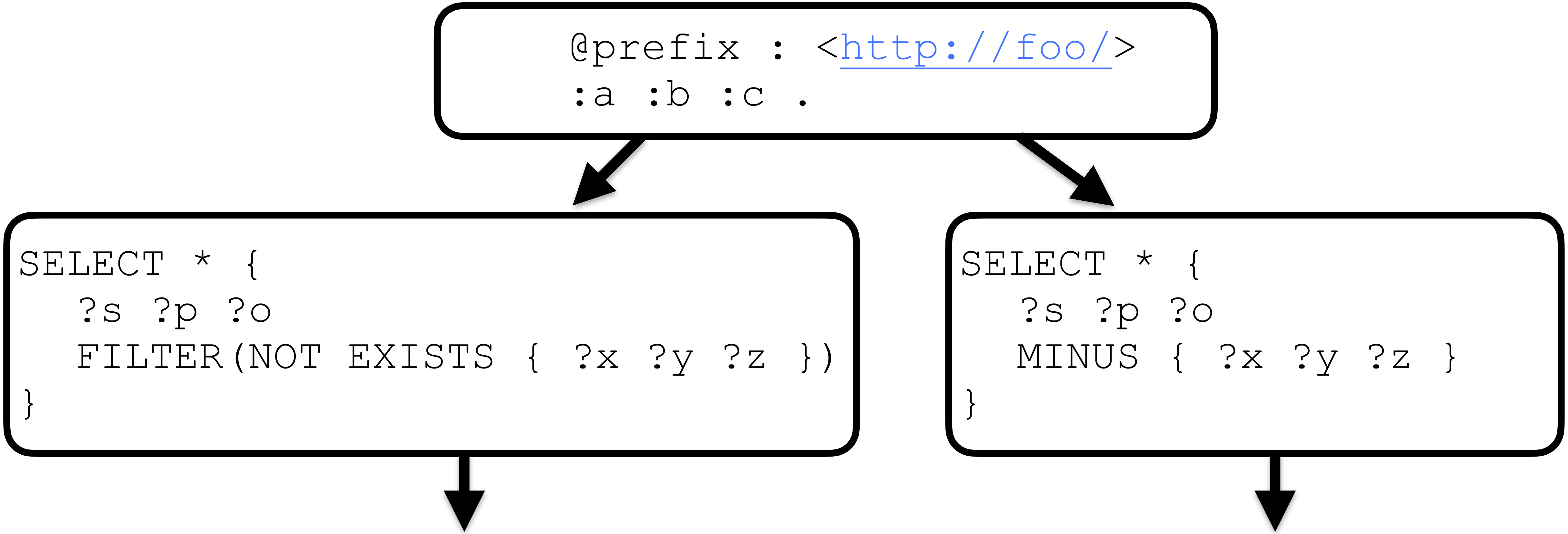
- The SPARQL 1.1 `MINUS` graph pattern clause is a binary operator that removes bindings that match the right-hand side.
- Requires shared variables between the graph pattern before and after the statement — it's based on removing matches based on the evaluation of two patterns.

Negation in SPARQL 1.1

```
PREFIX [..]
SELECT ?name
WHERE {
   ?person a foaf:Person;
      foaf:name ?name .
   FILTER(NOT EXISTS { ?person rdfs:seeAlso ?url })
}
```

- The SPARQL 1.1 `NOT EXISTS` filter uses the bindings from a solution to test whether a given pattern exists.
- Based on testing whether a pattern exists in the data, given the bindings already determined by the query pattern.
- `NOT EXISTS` and `MINUS` can produce different answers.

MINUS vs FILTER NOT EXISTS



```
@prefix : <http://foo/>
:a :b :c .
```

```
SELECT * {
   ?s ?p ?o
   FILTER(NOT EXISTS { ?x ?y ?z })
}
```

```
SELECT * {
   ?s ?p ?o
   MINUS { ?x ?y ?z }
}
```

s	p	o

`{ ?x ?y ?z }` matches any `?s ?p ?o` so `NOT EXISTS` eliminates any solution

s	p	o
:a	:b	:c

No shared variables, so no bindings are eliminated

Property Paths

Property paths can be used for querying arbitrary-length paths through the RDF graphs. For example:

- Find all instances of type beer:

```
?beer rdf:type beer:Beer
```

- Find all instances of beer, or instances of subclasses of beer:

```
?beer rdf:type/rdfs:subClassOf* beer:Beer .
```

They allow to perform some form of “syntactic reasoning” over the RDF graph.

Property Paths

Match one or both possibilities (logical OR):

```
SELECT * { :book1 dc:title|rdfs:label ?name }
```

Find the name of any people that Alice knows (sequence)

```
SELECT ?x ?name {
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows/foaf:name ?name .
}
```

Property Paths

Find the name of any people known by someone that Alice knows (2 foaf:knows links away):

```
SELECT ?x ?name {
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

Equivalent to:

```
SELECT ?x ?name {
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows ?a1 . ?a1 foaf:knows ?a2 . ?a2 foaf:name ?name .
}
```

Property Paths

Find the name of any people known by someone that Alice knows (2 foaf:knows links away):

```
SELECT ?x ?name {
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows/foaf:knows/foaf:name ?name .
}
```

Equivalent to (without explicit variables):

```
SELECT ?x ?name {
  ?x foaf:mbox <mailto:alice@example> .
  ?x foaf:knows [ foaf:knows [ foaf:name ?name ]] .
}
```

Summary

Different types of SPARQL queries:

- `SELECT`, `CONSTRUCT`, `DESCRIBE`, `ASK`

Named graphs, negation, property paths, `UNION`, `OPTIONAL`, etc.

Not covered here:

- SPARQL Update
- Graph Store HTTP Protocol
- Service descriptions
- Entailment regimes