Breaking Down a SPARQL Query

Prefixes (optional)
@PREFIX soap: <http://www.semantcweb/ner.org/ontology/apartments.ttl#>,
@PREFIX dbo: <http://dbpedia.org/resource/>,
@PREFIX dbpedia-owl: <http://dbpedia ia.org/ontology/>.

Qualified name: Namespace prefix

Keyword

SELECT ?location
"Selections: ?location which..."

From example graph:

WHERE {
soap: BaroniWayBuilding dbpedia-owl:location dbpedia:Amsterdam.
"... which is Baroni Wayburg located in Amsterdam"

soap: BaroniWayBuilding dbpedia-owl:location ?location .
"and for all rows that satisfy the above condition, return their location"

FILTER (filter query, here)
}

LIMIT 10
"Limit the output of results to 10" (Useful for large triple stores (i.e., databases))

ORDER BY ASC(?location)
"Order results alphabetically"

A Simpler Example

SELECT(?city)
"Select the city name"

Returns

WHERE{
?city dbpedia-owl:areaCode "020".
"...the city's area code is 020"
}

?city — rdf:type — dbpedia-owl:City

?city — dbpedia-owl:areaCode — "020"

Top
Triple pattern information about Results (in this example, language that the string is written in)

Bindings
(i.e., a table is returned, and not triples)

Shorthands

Prefixes can be added to the query to abbreviate URIs.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?name ?email
WHERE {
  ?person a foaf:Person .
  ?person foaf:name ?name .
  OPTIONAL { ?person foaf:mbox ?email }
  ?person rdfs:label "John Smith"@en .
}
ORDER BY ?name
LIMIT 10
```

Multiple Queries with SELECT

SELECT(?predicate ?object)
WHERE{
soap: BaroniWayBuilding ?predicate ?object .
"Select all triples that have BaroniWayBuilding as subject."
"Return all information attached to "BaroniWayBuilding"."
}

?predicate	?object
soap:hasNumberOfBedrooms	3
soap:isPartOf	soap:BaroniWayBuilding

Returns (+ Bindings)

Multiple Queries with UNION and OPTIONAL

Example #1

SELECT(?apartments ?labels)
WHERE{
?apartments dbpedia-owl:location dbpedia:Amsterdam.
"Apartments that are located in Amsterdam (which is specified by "location" property)..."
UNION
?apartments soap:isPartOf dbpedia:Amsterdam.
"...or apartments that are part of Amsterdam (which is specified by "location" property)."
OPTIONAL
?apartments rdfs:label ?labels.
"...and if available, return the labels of these apartments (for any rows selected). (Don't show rows w/ no labels, just return null in the binding.)"
}

Returns

?apartments	?labels
soap:BaroniWayApartment	"Baron Way Apartment for Rent
soap:FloridaAveStudio	

Can also be written as:
?apartments dbpedia-owl:location | | dbpedia-owl:locationCity dbpedia:Amsterdam.

Example #2

Graph

Subject	Predicate	Object
soap:John	soap:hasNumberOfBedrooms	"2"
soap:John	soap:isPartOf	soap:Smith
soap:Smith	soap:hasNumberOfBedrooms	"3"
soap:Smith	soap:isPartOf	soap:Jones

Bindings

?person	?hasNumberOfBedrooms	?isPartOf
John	Smith	2014-01-01
John	Smith	2014-01-11
Smith	Smith	2014-01-01

Missing results because subjects do not have an "isPart of a property-value" pair

Bindings

?person	?hasNumber	?isPart
John	Smith	2014-01-01
John	Smith	2014-01-11
Smith	Smith	2014-01-01
Jones	Berger	
Francis	Jones	

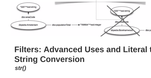
OPTIONAL keyword allows subjects without data to also be included in the results set.

Example #3: Junction and Disjunctions

And (Conjunction)
PREFIX dbo: <http://dbpedia.org/ontology>

SELECT ?x WHERE {
?x dbo:capital dbo:areaCode . "020".
} LIMIT 10

Returns Amsterdam (or no Westminster (which has the same area code with Amsterdam), because Westminster is not a capital)

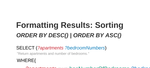


Or (Disjunction) with UNION

PREFIX dbo: <http://dbpedia.org/ontology>

SELECT ?x WHERE {
{ ?x dbo:areaCode "020" }
UNION
{ ?x dbo:areaCode "010" }
} LIMIT 10

Returns Amsterdam and Westminster, because they are the cities that match the queried area codes (since Westminster has the same area code with Amsterdam, it should have and does similarly)



Or (Disjunction) with UNION and OPTIONAL

PREFIX dbo: <http://dbpedia.org/ontology>

SELECT ?x ?y WHERE {
{ ?x dbo:areaCode "020" }
UNION
{ ?x dbo:capital ?y }
OPTIONAL { ?x dbo:capital ?y }
} LIMIT 10

Returns Amsterdam and Westminster (area code match), and also Netherlands

Property Paths (Chain Queries)

A property path uses a grammar between statement subject and object (SPO) keyword, in order to retrieve multiple elements related to subject. Example: *x is the act of walking over the same property relation more than once* using, determines that transitive into each other. Using that is optional (i.e. same thing can be achieved by using keywords such as UNION, etc. when they are used, property paths make code more concise and clear.)

Example #1

SELECT(?person ?givenName)
WHERE{
?person soap:familyName "Smith".
"Select all persons with family name "Smith"."
?person soap:givenName ?givenName .
"...and then, from the collection selected, categorize all of persons with "givenname"."
}

Returns "...return all persons with the family name smith, and also return their given names."

?person	?givenName
← employee1	Heidi
← employee2	John

Example #2

SELECT(?apartments)
WHERE{
?apartments soap:isPartOf ?buildings.
"Select all apartments that are part of "buildings"."
?buildings dbpedia-owl:location dbpedia:Amsterdam.
"...buildings among all buildings located in Amsterdam."
}
"...return all apartments (of buildings) located in Amsterdam."

The same statement in positive notation

?apartments soap:isPartOf / dbpedia-owl:location dbpedia:Amsterdam.

List of Property Paths

Syntax Form	Property Path Expression Name	Matches
^elt	InverseLiteral	Both, if elt is IRI or a literal
!(iri1|...|irin)	NegatedPropertySet	Any IRI except iri1 ... irin
elt1 / elt2	SequencePath	p1 followed by p2
elt1 | elt2	AlternativePath	p1 or p2 (or both)
elt*	ZeroOrMorePath	Zero or more occurrences of p
elt+	OneOrMorePath	At least one occurrence of p
elt?	ZeroOrOnePath	Zero or one occurrence of p
elt1 ^elt2	SimplifiedPath	Path that connects the subject and object of the path to a and b, where elt1 is a and elt2 is b
^elt1 / elt2	SimplifiedPath	Path that connects the subject and object of the path to a and b, where elt1 is a and elt2 is b
!(iri1|...|irin|^irj1|...|^irjn)	SimplifiedPath	Negated property set, iri1 ... irin is a set of IRIs, irj1 ... irjn is a set of IRIs
!(^iri1|...|^irin|irj1|...|irjn)	SimplifiedPath	Negated property set, iri1 ... irin is a set of IRIs, irj1 ... irjn is a set of IRIs
(elt)	BracketedPath	Brackets are used to group subexpressions
elt{n,m}	BoundedLengthPath	Or combination of between and minimum of exactly n and maximum of m occurrences of p
elt{n}	FixedLengthPath	Or combination of exactly n occurrences of p
elt{n,}	BoundedLengthPath	Or combination of at least n and minimum of n occurrences of p
elt{,n}	BoundedLengthPath	Or combination of between zero and maximum of n occurrences of p

Filters: Literals

Example #1

SELECT(?apartments)
WHERE{
?apartments soap:hasNumberofBedrooms ?bedroomNumbers.
"Return the number of bedrooms of all apartments and name them."
FILTER (?bedroomNumbers > 2).
"...and then take the results to return only those have more than 2 bedrooms."
}
"...return all apartments that has more than two bedrooms."

Returns

?apartments
soap:BaroniWayApartment

Example #2

PREFIX dbo: <http://dbpedia.org/ontology>

SELECT ?x WHERE {
?y dbo:areaCode "020".
?y dbo:populationTotal ?x .
FILTER (?x > 500000) .
}

Returns Amsterdam but not Berlin(area code, because Berlin(area code) population is less than 500000)

Filters: Advanced Uses and Literal to String Conversion

str()

SELECT(?apartments)
WHERE{
?apartments soap:address ?addresses.
"Return all apartments that has "addresses"."
FILTER regex(str(?apartments), "Baron").
"Filter the results of "apartments" (converted to string) for the string "Baron"."
}
"...return all apartments whose URL contains the string "Baron" ."

Returns

?apartments
soap:BaroniWayApartment

<http://www.semantcweb/ner.org/ontology/apartments.ttl#BaroniWayBuilding>
<http://www.semantcweb/ner.org/ontology/apartments.ttl#FloridaAveStudio>
<http://www.semantcweb/ner.org/ontology/apartments.ttl#SomeOtherBuilding>

Filters: Regular Expressions

SELECT(?apartments)
WHERE{
?apartments soap:address ?addresses.
"Return all apartments that has "addresses"."
FILTER regex(?addresses, "^4 Baron Way").
"Filter the results to only those starting with "4 Baron Way"."
}
"...return all apartments whose address starts with "4 Baron Way" ."

Returns

?apartments
soap:BaroniWayApartment

Filters: NOT EXISTS

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://www.example.com/>

SELECT ?person ?hasNoJobs
WHERE
{
?person rdf:type ex:Person .
?person ex:lastEndyear "2014" .
NOT EXISTS { ?person ex:employedJobs ?jobs . }
}

?person	?hasNoJobs
Jane	Berger
Francis	Jones

Returns subjects that do not have an inverse (i.e., Checkbox data)

Advanced Uses: Variable concatenation, renaming, and long statements

BIND(), CONCAT() ... AS ...

SELECT ?person ?hasNoJobs ?fullName
WHERE
{
?person rdf:type ex:Person .
?person ex:lastEndyear "2014" .
BIND(concat(?person," ",?hasNoJobs) AS ?fullName).
}

?person	?hasNoJobs	?fullName
Jane	Jones	Jane Jones
Jane	Jones	Jane Jones
Jane	Jones	Jane Jones

Concatenation, Renaming, and Long Statements (Variable Binding)

Formatting Results: Sorting

ORDER BY DESC() | ORDER BY ASC()

SELECT (?apartments ?bedroomNumbers)
"Select apartments and number of bedrooms."
WHERE{
?apartments soap:hasNumberOfBedrooms ?bedroomNumbers.
"Return all apartments that has "bedrooms"."
}
ORDER BY ASC(?bedroomNumbers) **GROUP BY**
"ASC" for ascending, and "DESC" for descending

?apartments	?bedroomNumbers
soap:BaroniWayApartment	3
soap:FloridaAveStudio	2

Formatting Results: Summarizing

COUNT
SUM
MIN
MAX
AVG
Other aggregate functions

This name has to be changed, if the same function is used twice (or column data would be overwritten)

"Change variable name"

SELECT (**AVG**(?bedroomNumbers) **AS** ?averageBedroomNumber)
WHERE{
?apartments soap:hasNumberOfBedrooms ?bedroomNumbers.
}
LIMIT 10
Limit to show of returned patterns to 10" (Useful for large triple stores (i.e., databases))

Returns

?avgNumberBedrooms
2

Query Type: ASK

ASK {?apartments}

WHERE{
?apartments soap:hasNumberOfBedrooms 3.
}

Returns

'True' or 'False'

Query Type: CONSTRUCT

CONSTRUCT {?apartments soap:hasNumberOfBedrooms ?bedroomNumbers
?apartments soap:isBigApartment true}

WHERE{
?apartments soap:hasNumberOfBedrooms ?bedroomNumbers.
}
FILTER(?bedroomNumbers > 2).

Returns

A graph (instead of bindings)
"This is effectively a subgraph of the main graph."

soap:BaroniWayApartment soap:hasNumberOfBedrooms 3.
soap:BaroniWayApartment soap:isBigApartment true.

Query Type: SELECT DISTINCT

Return all unique results (i.e., remove duplicate results)

PREFIX dbo: <http://dbpedia.org/ontology>
PREFIX dbpedia: <http://dbpedia.org/resource>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?name WHERE {
?country rdf:type dbo:Country ;
dbo:capital ?capital .
?capital dbo:name ?name .
FILTER (lang(?name) = "en")
}

?name
Amsterdam
The Hague
Netherlands
Tirana

Query Type: DESCRIBE

Returns all **predicates** and **objects** about a the queried subject.

PREFIX dbo: <http://dbpedia.org/resource>

DESCRIBE dbo:Amsterdam

Update: INSERT and INSERT DATA

Example #1: INSERT DATA

INSERT DATA
{
soap: LuxuryApartment rdfs:subClassOf soap:Apartment.
}

Example #2: INSERT

PREFIX dbo: <http://dbpedia.org/ontology>
PREFIX ex: <http://example.com/ocean>

INSERT {
?y ex:hengstal "020".
} WHERE {
?y dbo:areaCode "020".

Update: LOAD

LOAD <http://www.example.com/apartment.rdf>.

A graph (i.e., set of triples; triple dataset), specified by a (file) location(s) that may be added with INSERT

Update: DELETE (WHERE)

DELETE DATA
{
soap: LuxuryApartment rdfs:subClassOf soap:Apartment.
}

Update: CLEAR ALL

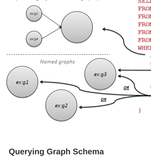
CLEAR ALL

There is no more to it. CLEAR ALL removes all content in a triple store, effectively emptying it.

DELETE WHERE
{
?apartments soap:hasNumberOfBedrooms ?bedroomNumbers.
FILTER (?bedroomNumbers > 2).
}
The FILTER keyword is inside the brackets of WHERE this time.

FROM and FROM NAMED

A SPARQL queries a *default graph* (normally) and zero or more *named graphs* (when inside a GRAPH clause).




```
PREFIX ex: <...>
SELECT ...
FROM ex:g1
FROM ex:g4
FROM NAMED ex:g1
FROM NAMED ex:g2
FROM NAMED ex:g3
WHERE {
  A ...
  GRAPH ex:g3 {
    B ...
  }
  GRAPH ?g {
    C ...
  }
}
```

Querying Graph Schema

More SPARQL Properties

- Named Graphs
  provide context for statements they contain.
  GRAPH <http://example.com/ ... > { }
- Aggregates
  COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT and SAMPLE
- Grouping and group aggregates
  GROUP BY and HAVING
- Provide data inline
  VALUES
- Negation
  MINUS
- Functions
  IF, BOUND, EXISTS, NOT EXISTS, IN, NOT IN
  &&, || , +, -, *, =, >=
  STR, LANG, LANGMATCH, isURI(or: isIRI(or: isLiteral, isBlank, str, lang, datatype
  STRLEN, SUBSTR, CONTAINS, CONCAT, REGEX
  now, year, month, day, hours, minutes, seconds
  ... and more

SPARQL Cheatsheet

http://www.cs.unicornbot.ca/~tepatmore/f6281/sparql_1_1-cheat-sheet.pdf