SPARQL

BASICS

Breaking Down a SPARQL Query

'Prefix' shortcuts (optional)

Prefixes in SPARQL:
@PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#> .
@PREFIX dbpedia: <http://dbpedia.org/resource/> .
@PREFIX dbpedia-owl: <http://dbpedia.org/ontology/> .

Qualified name; Namespace prefix — URI

Keyword — Query

Result clause: **SELECT** ?location
"Select/return ?locations which..."

Dataset (optional): **FROM** example:graph1

Triple pattern: A triple with zero or more variables (i.e., instead of values)

WHERE{
swp:BaronWayBuilding dbpedia-owl:location dbpedia:Amsterdam@en .
"...include a 'BaronWayBuilding located in Amsterdam' ..."

Subject — Property/Predicate — Object

swp:BaronWayBuilding dbpedia-owl:location ?location
"and for all triples that satisfy the above condition, return their ?location"

Query pattern

Graph pattern: An RDF graph that has zero nodes or edges as variables. It consists of multiple triple patterns.

Filters: **FILTER**(filter_query_here)
}

Query modifiers (optional):
LIMIT 10
"Limit number of returned patterns to 10." (Useful for large triple stores [i.e., databases])
ORDER BY ASC(?location)
"Order results alphabetically."

Tag: Provides further information about literals (in this example, language that the string is written in)

Returns

?location
dbpedia:Amsterdam dbpedia:Netherlands

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

A Simpler Query

SELECT(?city)
"Select/return ?cities where..."

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

Returns

?city
Amsterdam

Shorthand Syntax

ex:ourHouse rdf:type ex:House .
ex:ourHouse rdf:type ex:Building .
ex:ourHouse ex:locatedIn ex:Amsterdam .

=

Comma: Add **new object** while reusing the statement up to this point (i.e., reuse the subject and predicate).

ex:ourHouse rdf:type ex:House ,
ex:Building .
ex:ourHouse ex:locatedIn ex:Amsterdam .

=

Semicolon: Add **new predicate and object** while reusing the statement up to this point (i.e., reuse the subject).

ex:ourHouse rdf:type ex:House, ex:Building ;
ex:locatedIn ex:Amsterdam .

QUERY CONSTRUCTION

Simple Query with Plain Notation (Only 'SELECT' and 'WHERE' Keywords)

SELECT(?property ?object)
WHERE{
swp:BaronWayBuilding ?property ?object .
"Select all triples that has 'BaronWayBuilding' as subject."
"Return all information attached to "BaronWayBuilding".
}

Returns

?property	?object
swp:hasNumberOfBedrooms swp:isPartOf	3 swp:BaronWayBuilding

Queries with plain notation, and with Keywords 'UNION' and 'OPTIONAL'

Example #1: UNION and OPTIONAL

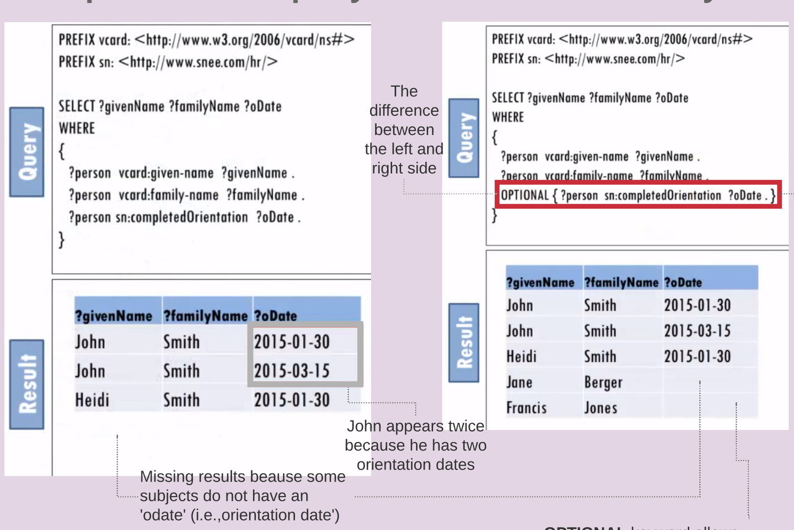
SELECT(?apartments ?labels)
WHERE{
?apartments dbpedia-owl:location dbpedia:Amsterdam .
"Select all apartments that are located in Amsterdam (which is specified by "location" property)..."
UNION — Think of this like a '+' sign between the lines above and below it.
"...and add to this selection..."
?apartments dbpedia-owl:locationCity dbpedia:Amsterdam .
"...all apartments that are located in Amsterdam (which is specified by "locationCity" property)..."
OPTIONAL
"...and if possible, also..."
?apartments rdfs:label ?labels .
"...return the human readable labels of all these apartments that are now selected. (And if there are no labels, just don't return it for that binding.)"
}

Returns

?apartments	?labels
swp:BaronWayApartment swp:FloridaAveStudio	swp:Baron Way Apartment for Rent

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

Example #2: Plain query versus OPTIONAL Keyword



Property Paths (Chain Queries)

A property path uses a 'pipeline' between statements instead of using UNION keyword, in order to retrieve multiple elements related to subject. Essentially, it is bridging multiple triples to each other by using the same variable in all of them; it is to write statements that transition into each other. Using property paths is optional; the same things can be achieved by using keywords such as UNION, yet when they are used, property paths make code more concise and clear.

Example #1

Using "?person" in both statements creates a property path between them

SELECT(?person ?givenName)
WHERE{
?person vcard:familyName "Smith" .
"Select all persons with family name "Smith"...
After the first line is ran, the results are kept in memory, and the second line is ran on this limited set that is in memory.
?person vcard:givenName ?givenName .
"...And then, from this selection, select a subgroup of all persons with a 'givenName' "
}
"...i.e., return all persons with the family name smith, and also return their given names."

And (Conjunction): family name — Return — given name

Returns

?person	?givenName
sn:employee1 sn:employee2	Heidi John

Example #2

SELECT(?apartments)
WHERE{
?apartments swp:isPartOf ?buildings .
"Return all apartments that are part of ?building..."
?buildings dbpedia-owl:location dbpedia:Amsterdam .
"...?buildings being all buildings located in Amsterdam."
}
"...i.e., return all apartments (of buildings) located in Amsterdam."

The same statement in concise notation

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

List of Property Path Operators

Syntax Form	Property Path Expression Name	Matches
iri	PredicatePath	An IRI. A path of length one.
^elt	InversePath	Inverse path (object to subject).
elt1 / elt2	SequencePath	A sequence path of elt1 followed by elt2.
elt1 | elt2	AlternativePath	A alternative path of elt1 or elt2 (all possibilities are tried).
elt*	ZeroOrMorePath	A path that connects the subject and object of the path by zero or more matches of elt.
elt+	OneOrMorePath	A path that connects the subject and object of the path by one or more matches of elt.
elt?	ZeroOrOnePath	A path that connects the subject and object of the path by zero or one matches of elt.
!iri or !(iri1| ...|irin)	NegatedPropertySet	Negated property set. An IRI which is not one of iri1...irin. !iri is short for !(iri).
!^iri or !(^iri1| ...|^irin)	NegatedPropertySet	Negated property set where the excluded matches are based on reversed path. That is, not one of iri1...irin as reverse paths. !^iri is short for !(^iri).
!(iri1| ...|irij|^irij+1| ...|^irin)	NegatedPropertySet	A combination of forward and reverse properties in a negated property set.
(elt)		A group path elt, brackets control precedence.

Example #3: Three Options in More Detail: Conjunction, Disjunction, and Join

Search scope narrows with each query
1. Run this query and store results
2. Run this query **on the results of the** previous query
3. Return the result

Plain Notation: And (Conjunction)

This is what is retrieved by default if no special keywords such as UNION or OPTIONAL is used

@PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?x
WHERE {
?x dbo:capital ?y .
?y dbo:areaCode "020" .
}
LIMIT 10

"Something that has a capital whose capital has the area code "020"

Returns Amsterdam but not Westminster (which has the same area code with Amsterdam), because Westminster is not a capital:

UNION: Aggregate (Disjunction)

Search scope does not narrow with each query
1. Run this query and store results
2. Run this query **independently** from the first one and store results
3. Return results from both queries in one combined output

@PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?y
WHERE {
{?y dbo:areaCode "020" .}
UNION
{?y dbo:areaCode "010" .}
}
LIMIT 10

Returns Amsterdam and Rotterdam, because they are the cities that match the queried area codes (since Westminster has the same area code with Amsterdam, it is also returned.):

OPTIONAL: Aggregate if Exists (Left Join)

@PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?x ?y
WHERE {
?y dbo:areaCode "020" .
OPTIONAL
{?x dbo:capital ?y .}
}
LIMIT 10

1. Run this query and store results
2. Run this query **on the results of the** previous query and store the results **if there are any**
3. Return results from both queries in one combined output

Search scope narrows **for the optional query** with each line, but remains the same for main query

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

QUERY TYPES

Query Type: SELECT DISTINCT

Prevents results rom returning more than once.

@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 ?ams .
?ams dbo:country dbpedia:Netherlands ;
rdfs:label ?name .
FILTER (lang(?name) = "en")
}

Query Type: CONSTRUCT

CONSTRUCT (?apartments swp:hasNumberOfBedrooms ?bedroomNumbers .
?apartments swp:isBigApartment true .)
WHERE{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers .
}
FILTER(?bedroomNumbers > 2) .

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

Returns

swp:BaronWayApartment swp:hasNumberOfBedrooms 3.
swp:BaronWayApartment swp:isBigApartment true.

Query Type: ASK

ASK (?apartments)
WHERE{
?apartments swp:hasNumberOfBedrooms 3 .
}

Returns

'True' or 'False'

Query Type: DESCRIBE

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

DESCRIBE dbpedia:Amsterdam

FILTERS

Filters: Literals

Example #1

SELECT(?apartments)
WHERE{
?apartments swp:hasNumberofBedrooms ?bedroomNumbers .
"Return the number of bedrooms of all apartments and keep this in memory..."
FILTER (?bedroomNumbers > 2) .
"...and then filter the results to return only those have more than 2 bedrooms."
}
"...i.e., return all apartments that has more than two bedrooms."

Returns

?apartments
swp:BaronWayApartment

Example #2

@PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?y
WHERE {
?y dbo:areaCode "020" .
?y dbo:populationTotal ?x .
FILTER (?x > 500000)
}

Filters out

Returns Amsterdam but not Borehamwood, because Borehamwood's population is less than 500000:

Filters: Regular Expressions

SELECT(?apartments)
WHERE{
?apartments swp:address ?addresses .
"Return all apartments that has ?addresses..."
FILTER **regex**(?addresses, "^4 Baron Way") .
"...?addresses being strings starting with "4 Baron Way".
}
"...i.e., return all apartments whose address starts with '4 Baron Way.' "

Returns

?apartments
swp:BaronWayApartment

Filters: NOT EXISTS

@PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
@PREFIX sn: <http://www.snee.com/hr/>

SELECT ?givenName ?familyName
WHERE {
?person vcard:given-name ?givenName .
?person vcard:family-name ?familyName .
NOT EXISTS {?person sn:completedOrientation ?oDate .}
}

?givenName	?familyName
Jane	Berger
Francis	Jones

Returns subjects that do not have an oDate (i.e., Orientation date)

Filters: Advanced Uses and Literal to String Conversion

str()

SELECT(?apartments)
WHERE{
?apartments swp:address ?addresses .
"Return all apartments that has ?addresses..."
FILTER regex(**str**(?apartments), "Baron") .
"...?addresses being all triples whose full address contains the string "Baron".
}
"...i.e., return all apartments whose URL contains the string 'Baron' "

Returns

?apartments
swp:BaronWayApartment

<http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#FloridaAveStudio>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#SomeOtherBuilding>

FORMATTING

Formatting Results: Sorting

ORDER BY DESC() | ORDER BY ASC()

SELECT (?apartments ?bedroomNumbers)
"Return apartments and number of bedrooms."
WHERE{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers .
}
ORDER BY ASC(*?bedroomNumbers*) — Also exists — **GROUP BY**
'ASC' for ascending, and '**DESC**' for descending

?apartments	?bedroomNumbers
swp:BaronWayApartment swp:FloridaAveStudio	3 1

Formatting Results: Summarizing

COUNT
SUM
MIN
MAX

Other aggregation functions

This name has to be changed into this when displaying results (or column label would be inaccurate)

"Change variable name"

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

Returns

?avgNumBedrooms
2

UPDATE

Update: INSERT and INSERT DATA

Example #1: INSERT DATA

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

Example #2: INSERT

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

INSERT {
?y ex:has-area-code "020" .
}
WHERE {
?y dbo:areaCode "020" .
}

Update: LOAD

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

A graph (i.e., set of triples; triple dataset), instead of a (few) triple(s) that may be added with INSERT.

Update: DELETE (WHERE)

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

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

The FILTER keyword is inside the brackets of WHERE this time.

Update: CLEAR ALL

CLEAR ALL

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

MISC

Variable concatenation, renaming, and long statements

BIND(), concat(), ... AS ....

@PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
@PREFIX sn: <http://www.snee.com/hr/>

SELECT ?givenName ?familyName ?fullName
WHERE {
?person vcard:given-name ?givenName .
?person vcard:family-name ?familyName .
BIND {concat(?givenName, " ", ?familyName) AS ?fullName.}
}

Creates new values via combining old ones
Combines statements into longer ones.
Renames a variable

?givenName	?familyName	?fullName
John	Smith	John Smith
Heidi	Smith	Heidi Smith
Jane	Berger	Jane Berger
Francis	Jones	Francis Jones

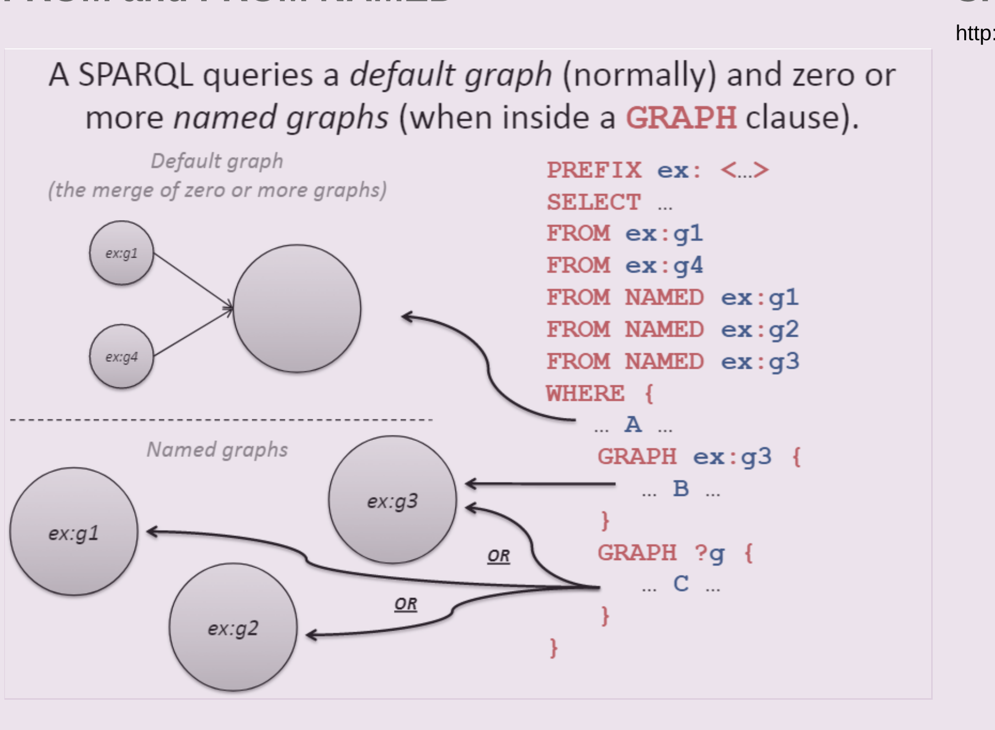
Concatenates '?givenName' and '?familyName' variables, and assigns the concatenated version to a new variable '?fullName'.

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
  &&, ||, =, <, >, =>, =<
  isIRI, isBlank, isLiteral, isNumeric, str, lang, datatype
  STRLEN, SUBSTR, CONTAINS, CONCAT, REGEX
  now, year, month, day, hours, minutes, seconds

  ... and more

FROM and FROM NAMED



SPARQL Cheatsheet

http://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf

Illustration: John C. Lokman
Produced in collaboration with Klaas Andries de Graaf

Image credits: Fresh by Heiko Maiwand from the Noun Project