Adila Krisnadhi

Data Semantics Lab
Wright State University, Dayton, OH

October 4, 2016
Outline

1. Motivation
2. Basic and Complex Graph Patterns
3. Filters
4. Solution Modifiers
5. SPARQL Output Forms
6. Assignment of New Values
7. Aggregates
8. Subqueries
9. Property Path
10. Basic SPARQL Semantics
11. Other SPARQL Features
Materials in this presentation are adapted from:

Outline

1 Motivation
2 Basic and Complex Graph Patterns
3 Filters
4 Solution Modifiers
5 SPARQL Output Forms
6 Assignment of New Values
7 Aggregates
8 Subqueries
9 Property Path
10 Basic SPARQL Semantics
11 Other SPARQL Features
RDF(S): “Can one RDF graph be inferred from another one?”
- Simple entailment
- RDF entailment
- RDFS entailment

OWL Ontologies: “Is a class a subclass of another class?”, “What are instances of an (OWL) class?”
- OWL (logical) entailment

How about:
- “Which properties relate two given individuals?”
- “Who co-authors a paper together with Adila Krisnadhi?”
- “What is the title of the national anthem of the USA?”

These are difficult to express (or even not all) via RDF/OWL entailments.
• Stands for: “SPARQL Protocol and RDF Query Language” (recursive acronym 😊)
• Query language for RDF graphs
• SPARQL 1.0: W3C Specification in 2008
• SPARQL 1.1: W3C Specification in 2013
SPARQL Specs Summary

- SPARQL 1.0
  - Syntax and semantics of query language
  - Query results in XML format: how to present results in XML
  - Protocol for transmitting queries to a query processing service and returning the results

- SPARQL 1.1
  - Query: extending SPARQL 1.0 query language
  - RDF graph update through SPARQL
  - Graph Store HTTP Protocol: HTTP operations for managing a graph collection
  - Entailment regimes: query results with (additional) inferences
  - Service description: methods for discovering and describing (using standard vocabulary) SPARQL services
  - Query federation: querying over distributed sources (multiple endpoints)
  - Additional query result format in JSON, CSV, TSV.

Note: SPARQL queries can, in practice, be run on SPARQL endpoints, behind which RDF data is stored. They cannot be run on plain RDF files.
Outline

1 Motivation
2 Basic and Complex Graph Patterns
3 Filters
4 Solution Modifiers
5 SPARQL Output Forms
6 Assignment of New Values
7 Aggregates
8 Subqueries
9 Property Path
10 Basic SPARQL Semantics
11 Other SPARQL Features
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix swrc: <http://swrc.ontoware.org/ontology#> .
@prefix dblpa: <http://dblp.l3s.de/d2r/resource/authors/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>
  rdf:type swrc:InProceedings ;
  dc:identifier "DBLP conf/rweb/KrisnadhiMH11"^^xsd:string ;
  foaf:maker dblpa:Pascal_Hitzler ;
  foaf:maker dblpa:Adila_Krisnadhi ;
  dcterms:issued "2011"^^xsd:gYear ;
  foaf:maker dblpa:Frederick_Maier ;
  rdfs:label "OWL and Rules."^^xsd:string .

<http://dblp.l3s.de/d2r/resource/publications/journals/semweb/BlomqvistHJKNS16>
  foaf:maker dblpa:Adila_Krisnadhi ;
  rdf:type swrc:Article .

dblpa:Adila_Krisnadhi rdf:type foaf:Agent ;
  foaf:name "Adila Krisnadhi" .

...
“List the title of the publications authored by Adila Krisnadhi?”

PREFIX dblpa: <http://dblp.l3s.de/d2r/resource/authors/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?publication ?title
WHERE {
}
The main part of a SPARQL query is located within the WHERE clause, consisting of graph patterns.

- The SELECT clause is one of the output forms – other output forms include: CONSTRUCT, ASK, DESCRIBE.

A basic graph pattern (BGP) is a set of triple patterns.

A triple pattern is an RDF triple, optionally containing variables (prefixed with '?') in subject/predicate/object positions.

BGP uses Turtle syntax.

Abbreviated IRIs possible (via PREFIX) – an alternative from the @prefix syntax from Turtle.

Intuition behind SPARQL

Match the given patterns against a given RDF graph and returns a multiset of solution mappings. Each solution mapping itself is a set of particular bindings for the variables in the pattern.
{ ?Pub foaf:maker dblpa:Adila_Krisnadhi ;
    foaf:maker ?CoAuthor .
    ?CoAuthor foaf:name ?Name . }

<http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>
    rdf:type swrc:InProceedings ;
    dcterms:issued "2011"^^xsd:gYear ;
    foaf:maker dblpa:Pascal_Hitzler ;
    foaf:maker dblpa:Adila_Krisnadhi ;
    foaf:maker dblpa:Frederick_Maier .
    dblpa:Adila_Krisnadhi foaf:name "Adila Krisnadhi" .
    dblpa:Pascal_Hitzler foaf:name "Pascal Hitzler" .
    dblpa:Frederick_Maier foaf:name "Frederick Maier" .
{ ?Pub foaf:maker dblpa:Adila_Krisnadhi ;
  foaf:maker ?CoAuthor .
?CoAuthor foaf:name ?Name . }

dcterms:issued "2011"^^xsd:gYear ;
foaf:maker dblpa:Pascal_Hitzler ;
foaf:maker dblpa:Adila_Krisnadhi ;
foaf:maker dblpa:Frederick_Maier .
dblpa:Adila_Krisnadhi foaf:name "Adila Krisnadhi" .
dblpa:Pascal_Hitzler foaf:name "Pascal Hitzler" .
dblpa:Frederick_Maier foaf:name "Frederick Maier" .
BGP Example

{ ?Pub foaf:maker dblpa:Adila_Krisnadhi ;
  foaf:maker  ?CoAuthor .
  ?CoAuthor foaf:name ?Name . }

Solution mappings (could be presented as a table):

Solution mappings (could be presented as a table):

1. ?Pub $\mapsto$ 
   <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor $\mapsto$ dblpa:Adila_Krisnadhi,
   ?Name $\mapsto$ "Adila Krisnadhi"
BGP Example

Solution mappings (could be presented as a table):

1. ?Pub ➔
   <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor ➔ dblpa:Adila_Krisnadhi,
   ?Name ➔ "Adila Krisnadhi"

2. ?Pub ➔
   <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor ➔ dblpa:Pascal_Hitzler,
   ?Name ➔ "Adila Krisnadhi"
BGP Example

{ ?Pub foaf:maker dblpa:Adila_Krisnadhi ;
  foaf:maker ?CoAuthor .
  ?CoAuthor foaf:name ?Name . }

Solution mappings (could be presented as a table):

1. ?Pub $\mapsto$<http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor $\mapsto$ dblpa:Adila_Krisnadhi,
   ?Name $\mapsto$ "Adila Krisnadhi"

2. ?Pub $\mapsto$
   <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor $\mapsto$ dblpa:Pascal_Hitzler,
   ?Name $\mapsto$ "Adila Krisnadhi"

3. ?Pub $\mapsto$
   <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>,
   ?CoAuthor $\mapsto$ dblpa:Frederick_Maier,
   ?Name $\mapsto$ "Frederick Maier"
Evaluating a BGP results in a **multiset of solution mappings**.
- Each solution mapping is a mapping from variables in the BGP into graph nodes or undefined value.
- The evaluation of BGP in the previous slide consists of 3 solution mappings.
- The result of a BGP evaluation is empty if no solution mapping is possible.
- It is a multiset, i.e., it’s unordered and the same binding may occur multiple times in a solution.
- If BGP has no variable but the pattern matches any part of the graph, then the result of the evaluation contains an empty solution mapping.
  - Watch out: an evaluation result consisting of an empty solution mapping is NOT the same as the empty result.
Empty Solution Mapping vs. Empty Solution

Krisnadhi (DaSeLab - Wright State)
Empty Solution Mapping vs. Empty Solution

Krisnadhi (DaSeLab - Wright State) 05 - SPARQL October 4, 2016 16 / 97

<http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>
    rdf:type swrc:InProceedings ;
    dcterms:issued "2011"^^xsd:gYear ;
    foaf:maker dblpa:Pascal_Hitzler ;
    foaf:maker dblpa:Adila_Krisnadhi ;
    foaf:maker dblpa:Frederick_Maier .

{ <http://dblp.l3s.de/d2r/resource/publications/conf/rweb/KrisnadhiMH11>
    foaf:maker <http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi> . }
Empty Solution Mapping vs. Empty Solution

Solution = \{\emptyset\}.
Empty Solution Mapping vs. Empty Solution

Solution = \{\emptyset\}.

Solution = \{\emptyset\}.
Empty Solution Mapping vs. Empty Solution

Solution = \{\emptyset\}.

Solution = \emptyset.
SELECT `variableList` – returns a sequence of solution mappings restricted to the given variables.

- If a variable in the SELECT clause does not occur in the BGP, it will be returned unbound.
- If all variables in SELECT clause are unbound, the corresponding solution mapping is empty.
- SELECT DISTINCT `variableList` – removes duplicate solution mappings.
- SELECT (DISTINCT) * – returns the whole multiset (set) of solution mappings over all variables in the BGP.

- ASK, CONSTRUCT, DESCRIBE – see later discussion.
Result may contain duplicates.

```sparql
SELECT ?CoAuthor
WHERE {
  ?Pub foaf:maker <http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi> ;
  foaf:maker ?CoAuthor .
  ?CoAuthor foaf:name ?Name .
}
```
Result may contain duplicates.

```
SELECT ?CoAuthor
WHERE {
  ?Pub foaf:maker <http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi> ;
  foaf:maker ?CoAuthor .
  ?CoAuthor foaf:name ?Name .
}
```

<table>
<thead>
<tr>
<th>?CoAuthor</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi">http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Eva_Blomqvist">http://dblp.l3s.de/d2r/resource/authors/Eva_Blomqvist</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Krzysztof_Janowicz">http://dblp.l3s.de/d2r/resource/authors/Krzysztof_Janowicz</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Monika_Solanki">http://dblp.l3s.de/d2r/resource/authors/Monika_Solanki</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Pascal_Hitzler">http://dblp.l3s.de/d2r/resource/authors/Pascal_Hitzler</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Tom_Narock">http://dblp.l3s.de/d2r/resource/authors/Tom_Narock</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi">http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Pascal_Hitzler">http://dblp.l3s.de/d2r/resource/authors/Pascal_Hitzler</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi">http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/d2r/resource/authors/Carsten_Lutz">http://dblp.l3s.de/d2r/resource/authors/Carsten_Lutz</a></td>
</tr>
</tbody>
</table>
```
Result contains no duplicate.

SELECT DISTINCT ?CoAuthor
WHERE {
    ?Pub foaf:maker <http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi> ;
        foaf:maker ?CoAuthor .
    ?CoAuthor foaf:name ?Name .
}
SELECT DISTINCT ?CoAuthor
WHERE {
  ?Pub foaf:maker <http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi> ;
  foaf:maker    ?CoAuthor .
?CoAuthor foaf:name    ?Name .
}

?CoAuthor

<http://dblp.l3s.de/d2r/resource/authors/Adila_Krisnadhi>
<http://dblp.l3s.de/d2r/resource/authors/Eva_Blomqvist>
<http://dblp.l3s.de/d2r/resource/authors/Krzysztof_Janowicz>
<http://dblp.l3s.de/d2r/resource/authors/Monika_Solanki>
<http://dblp.l3s.de/d2r/resource/authors/Pascal_Hitzler>
<http://dblp.l3s.de/d2r/resource/authors/Tom_Narock>
<http://dblp.l3s.de/d2r/resource/authors/Carsten_Lutz>
...
Blank node in query pattern:

- Permitted as subject or object (as in RDF)
- Given an arbitrary ID; not permitted to be reused in different BGPs within one query
- Act like variables, but cannot be selected.

Blank node in result/solution:

- Represent an unknown element
- Given arbitrary ID (possibly different from its ID in the input RDF graph); repeated occurrences in result denote the same element
Queries involving Datatypes

Given RDF data:

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
ex:ex1 ex:p "test" .
ex:ex2 ex:p "test"^^xsd:string .
ex:ex3 ex:p "test"@en .
ex:ex4 ex:p "42"^^xsd:integer .

What are the solution mappings for the following BGP?

{ ?subject <http://example.org/p> "test" . }
Queries involving Datatypes

Given RDF data:

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/> .
ex:ex1 ex:p "test" .
ex:ex2 ex:p "test"^^xsd:string .
ex:ex3 ex:p "test"@en .
ex:ex4 ex:p "42"^^xsd:integer .

What are the solution mappings for the following BGP?

{ ?subject <http://example.org/p> "test" . }

?subject matches only ex:ex1. Exact match for datatypes/language tags are required.
Queries involving Datatypes

Given RDF data:

```PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> .
PREFIX ex: <http://example.org/> .
ex:ex1 ex:p "test" .
ex:ex2 ex:p "test"^^xsd:string .
ex:ex3 ex:p "test"@en .
ex:ex4 ex:p "42"^^xsd:integer .
```

What are the solution mappings for the following BGP?

```{ ?subject <http://example.org/p> "test" . }
```

?subject matches only ex:ex1. Exact match for datatypes/language tags are required. Special for numeric values, some syntactic sugar is allowed:

```{ ?subject <http://example.org/p> 42 . }
```

has ex:ex4 as a match for ?subject.

- Datatype is determined solely from the syntactic form.
"List all publication titles authored by Adila Krisnadhi or Adila Alfa Krisnadhi or Adam Shepherd."

PREFIX dblpa: <http://dblp.l3s.de/d2r/resource/authors/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?Title
WHERE {
  { [ foaf:maker dblpa:Adila_Krisnadhi ; rdfs:label ?Title ] . }
  UNION
  { [ foaf:maker dblpa:Adila_Alfa_Krisnadhi ; rdfs:label ?Title ] . }
  UNION
  { [ foaf:maker dblpa:Adam_Shepherd ; rdfs:label ?Title ] . }
}

- Use keyword UNION. BGPs are grouped by { ... }
- Result is the multi-set union of the results for each of the alternative BGPs.
- Identical variables within different UNION patterns do not influence each other.
- Some variables may be unbound, e.g., when a BGP in the UNION pattern has a variable that does not occur in the other BGPs.
“List the collections edited by Adila Krisnadhi and publications authored by him.”

```
PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dblpa: <http://dblp.l3s.de/d2r/resource/authors/>
SELECT ?EditedPub ?AuthoredPub
WHERE {
  { ?EditedPub swrc:editor dblpa:Adila_Krisnadhi . } 
  UNION 
  { ?AuthoredPub foaf:maker dblpa:Adila_Krisnadhi . } }
```
“List the collections edited by Adila Krisnadhi and publications authored by him.”

PREFIX swrc: <http://swrc.ontoware.org/ontology#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dblpa: <http://dblp.l3s.de/d2r/resource/authors/>
SELECT ?EditedPub ?AuthoredPub
WHERE {
    { ?EditedPub swrc:editor dblpa:Adila_Krisnadhi . }
    UNION
    { ?AuthoredPub foaf:maker dblpa:Adila_Krisnadhi . } }

<table>
<thead>
<tr>
<th>EditedPub</th>
<th>AuthoredPub</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dblp.l3s.de/.../conf/semweb/2015wop">http://dblp.l3s.de/.../conf/semweb/2015wop</a></td>
<td><a href="http://dblp.l3s.de/.../journals/semweb/BlomqvistHJKNS16">http://dblp.l3s.de/.../journals/semweb/BlomqvistHJKNS16</a></td>
</tr>
<tr>
<td><a href="http://dblp.l3s.de/.../conf/ijcai/2015jowo">http://dblp.l3s.de/.../conf/ijcai/2015jowo</a></td>
<td><a href="http://dblp.l3s.de/.../reference/snam/KrisnadhiH14">http://dblp.l3s.de/.../reference/snam/KrisnadhiH14</a></td>
</tr>
<tr>
<td></td>
<td><a href="http://dblp.l3s.de/.../conf/lpar/KrisnadhiL07">http://dblp.l3s.de/.../conf/lpar/KrisnadhiL07</a></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

?AuthoredPub is unbound in the first and second solution mappings, while ?EditedPub is unbound in the remaining solution mappings.
From DBPedia repository (http://dbpedia.org/sparql/):
“Give all European countries, and optionally, their capital if the country is a member of European Union,”

PREFIX dbo: <http://dbpedia.org/ontology/capital>
PREFIX ygc: <http://dbpedia.org/class/yago/>
SELECT ?country ?capitalname
WHERE { ?country a ygc:EuropeanCountries .
    OPTIONAL { ?country a ygc:MemberStatesOfTheEuropeanUnion ;
        dbo:capital ?capital .
        ?capital rdfs:label ?capitalname . } } }

Note: the predicate 'a' is an abbreviation for rdf:type according to the Turtle syntax spec.

<table>
<thead>
<tr>
<th>?country</th>
<th>?capitalname</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Netherlands">http://dbpedia.org/resource/Netherlands</a></td>
<td>&quot;Amsterdam&quot;@en</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Germany">http://dbpedia.org/resource/Germany</a></td>
<td>&quot;Berlin&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Portugal">http://dbpedia.org/resource/Portugal</a></td>
<td>&quot;Lisbon&quot;@en</td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Albania">http://dbpedia.org/resource/Albania</a></td>
<td></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Denmark">http://dbpedia.org/resource/Denmark</a></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
- OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.
- OPTIONAL and UNION has equal precedence. Grouping is left-associative.
OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.

OPTIONAL and UNION has equal precedence. Grouping is left-associative.

This BGP

```
  { ?book ex:author ?author . } UNION
  { ?book ex:editor ?author . } OPTIONAL
  { ?author ex:surname ?name . } }
```

is equivalent to
• OPTIONAL always applies to one pattern group, specified to the right of the OPTIONAL keyword.

• OPTIONAL and UNION has equal precedence. Grouping is left-associative.

This BGP

```
  { ?book ex:author ?author . } UNION
  { ?book ex:editor ?author . } OPTIONAL
  { ?author ex:surname ?name . } }
```

is equivalent to

```
  { { ?book ex:author ?author . } UNION
    { ?book ex:editor ?author . }
  } OPTIONAL { ?author ex:surname ?name . } }
```
Another Example

Given the data below:

@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://ex.org/> .
_:a dc10:title "SPARQL Query Tutorial" .
_:a dc10:creator "Alice" .
_:b dc11:creator "Bob" .
_:b ex:level "beginners" .

Write a query that returns all titles (dc10:title or dc11:title) and, if given, the level (ex:level).
Recall: an **RDF Dataset** is a set of RDF graphs containing one **default graph** and zero or more **named graphs**. Every named graph has an IRI as its name, while default graph has no name and may be empty.

RDF dataset can be given by the triple store (by default) or specified by the user. The latter by specifying one or more `FROM` or `FROM NAMED` clauses. The user-specified RDF dataset is as follows:

- a default graph consisting of the RDF merge of the graphs referred to in the `FROM` clauses, and
- a set of (IRI, graph) pairs, one from each `FROM NAMED` clause.

BGPs are matched to an **active graph** in the RDF dataset. Use `GRAPH` keyword to specify which named graph is active when matching a BGP. Without enclosing `GRAPH` keyword, BGP is matched to the default graph.
"List people that occurs in the default graph and two different named graphs in a RDF dataset":

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

- The first BGP is matched against the default graph.
- The variables ?g1, ?g2 can appear in any BGP and can also instead be IRIs.
(On http://data.semanticweb.org/snorql): “List people appearing in either ESWC 2007 or 2008 graphs, who have the same affiliation to the people appearing in ISWC 2012 or ISWC 2013 graphs.”

```
SELECT ?person ?p
FROM <http://data.semanticweb.org/conference/eswc/2008/complete>
FROM NAMED <http://data.semanticweb.org/conference/iswc/2012/complete>
FROM NAMED <http://data.semanticweb.org/conference/iswc/2013/complete>
WHERE {
    ?person a foaf:Person .
    GRAPH ?g { ?p a foaf:Person .
        ?org foaf:member ?p .}
    FILTER (?person != ?p )}
```

The user-specified RDF dataset consists of

- A default graph obtained by merging the ESWC 2007 and ESWC 2008 graphs.
- Two named graphs as specified in FROM NAMED clauses.
- BGPs inside GRAPH clause are matched only to the specified named graphs.
- For FILTER keyword, see next.
Outline

1. Motivation
2. Basic and Complex Graph Patterns
3. Filters
4. Solution Modifiers
5. SPARQL Output Forms
6. Assignment of New Values
7. Aggregates
8. Subqueries
9. Property Path
10. Basic SPARQL Semantics
11. Other SPARQL Features
Why Filters?

Even with complex query patterns, some queries are not expressible:

- “Which persons are between 18 and 23 years old?”
- “Which person has a name that contains a hyphen character?”
- “List the English name of the capital city of European countries”.

We use filter as a general mechanism for such expressions.
Filter

- Syntax: `FILTER( filterExpression )`
- By substituting variables in it, a filter expression returns an effective boolean value `true` or `false`, or produces an error. See https://www.w3.org/TR/sparql11-query/#ebv for details about EBV of a numeric value, a string, etc.
- A solution mapping is eliminated from the result if substituting it to the filter results in the EBV `false` or produces an error.
- Many SPARQL filters come from outside RDF, e.g., XQuery/XPath
- Filter can be used to express some form of negation.
- See https://www.w3.org/TR/sparql11-query/#expressions.
Syntax: `FILTER(filterExpression)`

By substituting variables in it, a filter expression returns an effective boolean value `true` or `false`, or produces an error. See https://www.w3.org/TR/sparql11-query/#ebv for details about EBV of a numeric value, a string, etc.

A solution mapping is eliminated from the result if substituting it to the filter results in the EBV `false` or produces an error.

Many SPARQL filters come from outside RDF, e.g., XQuery/XPath

Filter can be used to express some form of negation.

See https://www.w3.org/TR/sparql11-query/#expressions.

```sparql
PREFIX ex: <http://ex.org/>
SELECT ?book
WHERE {
  FILTER (?price < 35)
}
```

Above, any solution mapping where the value of `?price` is less than 35 is eliminated from the result.
Unary Boolean operators: \( \neg \) \( \sim \) \( \neg A \) is \textbf{true} if \( A \) is \textbf{false}, vice versa.

Logical connectives: \( \lor \), \( \land \)

Comparison operators: \( <, =, >, \leq, \geq, \neq \)

- Comparison for literals according to the natural ordering
- Support for numerical datatypes (\texttt{xsd:integer}, \texttt{xsd:decimal}, etc.), \texttt{xsd:dateTime}, \texttt{xsd:string} (alphabetical order), \texttt{xsd:Boolean} (1 > 0)
- For non-literals, only \( = \) and \( \neq \) are available.
- Comparison cannot be done between incompatible types, e.g., between an \texttt{xsd:string} literal and an \texttt{xsd:integer} literal.
Unary functions: +, −

Binary functions: +, −, *, /

- Support for numerical datatypes.
- Not Boolean; used to obtain a value from other values in filter expression. For example:

  FILTER( ?weight / (?size * ?size) >= 25 )
var is a variable, expr1, expr2, expr3 are expressions interpreted as an EBV, term, term1, term2 are RDF terms (IRIs, literals, blank nodes), pattern is a graph pattern, lit is a literal, res is an IRI

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOUND( var )</td>
<td>true if var is a bound variable</td>
</tr>
<tr>
<td>IF( expr1, expr2, expr3 )</td>
<td>returns EBV of expr2 if expr1 is true, otherwise returns EBV of expr3</td>
</tr>
<tr>
<td>EXISTS { pattern }</td>
<td>true if pattern matches; false otherwise</td>
</tr>
<tr>
<td>NOT EXISTS { pattern }</td>
<td>false if pattern matches; true otherwise</td>
</tr>
<tr>
<td>sameTerm( term1, term2 )</td>
<td>true if term1 and term2 are the same; false otherwise. more general than = operator</td>
</tr>
<tr>
<td>term IN ( expr1, ... )</td>
<td>true if term can be found in the list on the right hand side</td>
</tr>
<tr>
<td>term NOT IN ( expr1, ... )</td>
<td>true if term cannot be found in the list</td>
</tr>
<tr>
<td>isIRI( term ), isURI( term )</td>
<td>true if term is an IRI</td>
</tr>
<tr>
<td>isBlank( term )</td>
<td>true if term is a blank node</td>
</tr>
<tr>
<td>isLiteral( term )</td>
<td>true if term is a literal</td>
</tr>
<tr>
<td>isNumeric( term )</td>
<td>true if term is a numeric value</td>
</tr>
</tbody>
</table>

17 and "17"\"\~\~xsd:integer" are numeric, while "17" is not
## Other Notable Functions and Function Forms

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>STR(lit)</code></td>
<td>returns the lexical form of the literal <code>lit</code>.</td>
</tr>
<tr>
<td><code>STR(res)</code></td>
<td>returns the codepoint/string representation of the IRI <code>res</code></td>
</tr>
<tr>
<td><code>LANG(lit)</code></td>
<td>returns the language tag of the literal <code>lit</code>, if any; returns &quot;&quot; otherwise</td>
</tr>
<tr>
<td><code>DATATYPE(lit)</code></td>
<td>returns the datatype of <code>lit</code></td>
</tr>
<tr>
<td><code>IRI(lit), IRI(res)</code></td>
<td>returns an IRI from the literal <code>lit</code> or an IRI <code>res</code></td>
</tr>
<tr>
<td><code>BNODE(), BNode(lit)</code></td>
<td>creates a blank node; if given a simple literal argument, the same literal within an expression for the same solution mapping yields the same blank node</td>
</tr>
<tr>
<td><code>STRDT(lit, res)</code></td>
<td>creates a typed literal with lexical form <code>lit</code> and datatype <code>res</code></td>
</tr>
<tr>
<td><code>STRLANG(lit, ltag)</code></td>
<td>creates a language-tagged literal with lexical form <code>lit</code> and language tag <code>ltag</code></td>
</tr>
<tr>
<td><code>UUID()</code></td>
<td>returns a fresh IRI using URN scheme (Note: not a HTTP IRI!)</td>
</tr>
<tr>
<td><code>STRUUUID()</code></td>
<td>returns a string that is a scheme specific part of a UUID</td>
</tr>
</tbody>
</table>
Other Notable Functions and Function Forms

- **String functions:** `langMatches`, `REGEX`, `REPLACE`, `CONCAT`, `STRLEN`, `SUBSTR`, `UCASE`, `LCASE`, `STRSTARTS`, `STRENDS`, `CONTAINS`, `STRBEFORE`, `STRAFTER`, `ENCODE_FOR_URI`
- **Numeric functions:** `ABS`, `ROUND`, `CEIL`, `floor`, `RAND`
- **Data/Time functions:** `NOW`, `YEAR`, `MONTH`, `DAY`, `HOURS`, `MINUTES`, `SECONDS`, `TIMEZONE`, `TZ`
- **Hash functions:** `MD5`, `SHA1`, `SHA256`, `SHA384`, `SHA512`
- **Casting operations:** `STR`, `BOOL`, `DBL`, `FLT`, `DEC`, `INT`, `dT`, `ltrl`
Examples

Try in DBPedia

- Find countries that were founded after December 31, 1799, but were then dissolved before the year of 1900.
- Find the English name of all landlocked countries whose population exceeds 15 million.
Given data:

[] foaf:name "Alice".
[ foaf:name "Bob" ;

the following query:

SELECT ?name
WHERE { ?x foaf:name ?name .
  OPTIONAL { ?x foaf:age ?age } .
  FILTER (!bound(?age)) }

returns "Alice" as the only value for ?name
Outline

1. Motivation
2. Basic and Complex Graph Patterns
3. Filters
4. Solution Modifiers
5. SPARQL Output Forms
6. Assignment of New Values
7. Aggregates
8. Subqueries
9. Property Path
10. Basic SPARQL Semantics
11. Other SPARQL Features
Sorting Results

Use keyword ORDER BY

```sparql
SELECT ?book, ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price
```

- Sorting as with comparison operators in filters.
- IRIs are sorted alphabetically.
- Ordering of elements of different types:
  unbound variables < blank nodes < IRIs < RDF literals
- Spec does not define all possible orderings.
- Descending order: use ORDER BY DESC (?price)
- Ascending order (default): ORDER BY ASC (?price)
- Hierarchical ordering criteria: ORDER BY ASC(?price), title
LIMIT, OFFSET, and DISTINCT

- SELECT DISTINCT: removal of duplicates
- LIMIT: maximal number of results
- OFFSET: position of the first returned result (within the whole result).
- LIMIT and OFFSET only meaningful with ORDER BY.

SELECT DISTINCT ?book, ?price
ORDER BY ?price LIMIT 5 OFFSET 25
Output Form SELECT

- SELECT returns sequence of solution mappings.
- Syntax: SELECT variableList or SELECT *
- Advantage: simple sequential processing of results.
- Disadvantage: structure and relationships between the objects are lost.
CONSTRUCT returns an RDF graph (i.e., a set of triples) created using results from the graph patterns.

- Can be used to transform a graph to another.
- **Advantage**: structured results data between the objects
- **Disadvantage**: harder to process sequentially
- **Disadvantage**: if a solution mapping contains unbound variable, triples corresponding to that solution mapping will be omitted.

```
PREFIX ex: <http://example.org/>
CONSTRUCT {
 ?person ex:mailbox ?email .
 ?person ex:telephone ?tel . }
WHERE {
 ?person ex:tel ?tel . }
```
Given data:

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:firstname "Alice" ;
    foaf:surname "Hacker" .
_:b foaf:firstname "Bob" ;
    foaf:surname "Hacker" .

and query:

PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT {
    ?x vcard:N _:v .
    _:v vcard:givenName ?gname ;
       vcard:familyName ?fname 
} WHERE {
    ?x foaf:surname ?fname 
}
we would obtain an RDF graph:

```sparql
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
_:v1 vcard:N _:x .
_:x vcard:givenName "Alice" ;
vcard:familyName "Hacker" .
_:v2 vcard:N _:z .
_:z vcard:givenName "Bob" ;
vcard:familyName "Hacker" .
```

Notice that the blank nodes in the output may have completely different IDs than what was provided by the solution mappings and the template.
ASK and DESCRIBE Output Forms

- **ASK**: checks if the query has at least one answer, i.e., non-empty solution – returns true/false.
- **DESCRIBE**: returns an RDF description for each resulting IRI – the actual description returned is application-dependent.

```
DESCRIBE ?x WHERE { ?x <http://ex.org/emplID> "123" }
```

may return something like:

```
_:a ex0rg:emplID "123" ;
    foaf:mbox_sha1sum "ABCD1234" ;
    vcard:N
        [ vcard:Family "Smith" ;
          vcard:Given "John" ] .
foaf:mbox_sha1sum a owl:InverseFunctionalProperty .
```
Outline

1 Motivation
2 Basic and Complex Graph Patterns
3 Filters
4 Solution Modifiers
5 SPARQL Output Forms
6 Assignment of New Values
7 Aggregates
8 Subqueries
9 Property Path
10 Basic SPARQL Semantics
11 Other SPARQL Features
Assignment of New Values I

Inside SELECT clause:

SELECT ?Item (?Pr * 1.1 AS ?NewP )
WHERE { ?Item ex:price ?Pr . }

Note: cannot assign values to variables inside the expression.

Data:

ex:lemonade1 ex:price 3 .
ex:icetea1 ex:price 3.
ex:coke1 ex:price 3.50 .
ex:coffee1 ex:price "n/a" .

Result (leaves errors unbound):

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:lemonade1</td>
<td>3.3</td>
</tr>
<tr>
<td>ex:icetea1</td>
<td>3.3</td>
</tr>
<tr>
<td>ex:coke1</td>
<td>3.85</td>
</tr>
<tr>
<td>ex:coffee1</td>
<td></td>
</tr>
</tbody>
</table>
Alternatively, using BIND:

```sparql
SELECT ?Item ?NewP
    BIND (?Pr * 1.1 AS ?NewP) }
```

Data:

- ex:lemonade1 ex:price 3.
- ex:icetea1 ex:price 3.
- ex:coke1 ex:price 3.50.
- ex:coffee1 ex:price "n/a".

Result (leaves errors unbound):

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:lemonade1</td>
<td>3.3</td>
</tr>
<tr>
<td>ex:icetea1</td>
<td>3.3</td>
</tr>
<tr>
<td>ex:coke1</td>
<td>3.85</td>
</tr>
<tr>
<td>ex:coffee1</td>
<td></td>
</tr>
</tbody>
</table>
Note: BIND is evaluated in-place!

SELECT ?Item ?NewP
WHERE { BIND (?Pr * 1.1 AS ?NewP)
    ?Item ex:price ?Pr . }

Data:

ex:lemonade1 ex:price 3 .
ex:icetea1 ex:price 3.
ex:coke1 ex:price 3.50 .
ex:coffee1 ex:price "n/a" .

Result is empty:

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?Item</th>
<th>?NewP</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
</tbody>
</table>

Krisnadhi (DaSeLab - Wright State)
Providing Inline Data with VALUES

:drink1 rdfs:label "Latte" ; ex:price 4 .
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .
Providing Inline Data with VALUES

:drink1 rdfs:label "Latte" ; ex:price 4 .
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .

Use VALUES keyword to enumerate tuples of values to be assigned to variables.

SELECT ?drink ?name ?price
WHERE {
  ?drink rdfs:label ?name ;
    ex:price ?price .
VALUES (?drink ?name)
  (UNDEF "Latte")
  (:drink2 UNDEF)
  (:drink5 "Espresso")
}
Providing Inline Data with VALUES

`:drink1 rdfs:label "Latte" ; ex:price 4 .
:drink2 rdfs:label "Capuccino" ; ex:price 3.5 .
:drink3 rdfs:label "Dark Roast" ; ex:price 2 .
:drink4 rdfs:label "Espresso" ; ex:price 3.5 .

Use VALUES keyword to enumerate tuples of values to be assigned to variables.

```
SELECT ?drink ?name ?price
WHERE {
  ?drink rdfs:label ?name ;
  ex:price ?price .
VALUES (?drink ?name)
{ (UNDEF "Latte")
  (:drink2 UNDEF)
  (:drink5 "Espresso")
}

?drink | ?name      | ?price |
--------|-------------|--------
:drink1 | "Latte"    | 4      |
:drink2 | "Capuccino"| 3.5    |
```

Krisnadhi (DaSeLab - Wright State)
Outline

1. Motivation
2. Basic and Complex Graph Patterns
3. Filters
4. Solution Modifiers
5. SPARQL Output Forms
6. Assignment of New Values
7. Aggregates
8. Subqueries
9. Property Path
10. Basic SPARQL Semantics
11. Other SPARQL Features
Count items:

SELECT (COUNT(?Item) AS ?C)
WHERE { ?Item ex:price ?Pr . }

Data:

ex:smoothie1 ex:price 4 ;
    a ex:Colddrink .

ex:icetea1 ex:price 3 ;
    a ex:Colddrink .

ex:coke1 ex:price 3.50 ;
    a ex:Colddrink .

ex:tea1 ex:price 3 ;
    a ex:Hotdrink .

ex:coffee1 ex:price "n/a" ;
    a ex:Hotdrink .

Results:

<table>
<thead>
<tr>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
Count categories:

```
SELECT (COUNT(?Ty) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
```

Data:

```
ex:smoothie1 ex:price 4 ;
a ex:Colddrink .
ex:icetea1 ex:price 3 ;
a ex:Colddrink .
ex:coke1 ex:price 3.50 ;
a ex:Colddrink .
ex:tea1 ex:price 3 ;
a ex:Hotdrink .
ex:coffee1 ex:price "n/a" ;
a ex:Hotdrink .
```

Results:

```
<table>
<thead>
<tr>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>
```
Count distinct categories:

SELECT (COUNT(DISTINCT ?Ty) AS ?C)  
WHERE { ?Item rdf:type ?Ty . }

Data:

ex:smoothie1 ex:price 4 ;  
   a ex:Colddrink .

ex:icetea1 ex:price 3 ;  
   a ex:Colddrink .

ex:coke1 ex:price 3.50 ;  
   a ex:Colddrink .

ex:tea1 ex:price 3 ;  
   a ex:Hotdrink .

ex:coffee1 ex:price "n/a" ;  
   a ex:Hotdrink .

Results:

<table>
<thead>
<tr>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Aggregates with Grouping

Count item per categories:

```
SELECT ?Ty (COUNT(?Item) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
GROUP BY ?Ty
```

Data:

```
ex:smoothie1 ex:price 4 ;
   a ex:Colddrink .
ex:icetea1 ex:price 3 ;
   a ex:Colddrink .
ex:coke1   ex:price 3.50 ;
   a ex:Colddrink .
ex:tea1     ex:price 3 ;
   a ex:Hotdrink .
ex:coffee1  ex:price "n/a" ;
   a ex:Hotdrink .
```

Results:

```
<table>
<thead>
<tr>
<th>?Ty</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Colddrink</td>
<td>3</td>
</tr>
<tr>
<td>ex:Hotdrink</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Count item per categories, for those categories with more than two items:

```
SELECT ?Ty (COUNT(?Item) AS ?C)
WHERE { ?Item rdf:type ?Ty . }
GROUP BY ?Ty
HAVING COUNT(?Item) > 2
```

Data:

- `ex:smoothie1 ex:price 4 ;
  a ex:Colddrink .`
- `ex:icetea1 ex:price 3 ;
  a ex:Colddrink .`
- `ex:coke1 ex:price 3.50 ;
  a ex:Colddrink .`
- `ex:tea1 ex:price 3 ;
  a ex:Hotdrink .`
- `ex:coffee1 ex:price "n/a" ;
  a ex:Hotdrink .`

Results:

```
<table>
<thead>
<tr>
<th>?Ty</th>
<th>?C</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Colddrink</td>
<td>3</td>
</tr>
</tbody>
</table>
```

Krisnadhi (DaSeLab - Wright State)
Other Aggregates

- SUM
- AVG
- MIN
- MAX
- GROUP_CONCAT – concatenate values with a given separator string
- SAMPLE – 'pick' one non-deterministically
Subqueries: SELECT query inside a graph pattern.

“List all distinct titles of papers authored by at most 6 co-authors of Pascal Hitzler”

#Try at http://data.semanticweb.org/snorql
PREFIX swp: <http://data.semanticweb.org/person/>
SELECT DISTINCT ?title
WHERE {
  { SELECT DISTINCT ?person
    WHERE {
      FILTER (?person != swp:pascal-hitzler)
    } LIMIT 6
  }
}
Property Path Expressions

Allows one to query using arbitrary length of paths in the graphs.
“List all names of people who transitively co-authors with Pascal Hitzler”

PREFIX swp: <http://data.semanticweb.org/person/>
SELECT DISTINCT ?name
WHERE {
  swp:pascal-hitzler (~foaf:maker/foaf:maker)+/foaf:name ?name
}

That is, we find the name of:

1. people who co-authors with Pascal Hitzler;
2. people who co-authors with the people from (1)
3. people who-co-authors with the people from (2)
4. etc.
The forms are somewhat similar to regular expression.

1. iri – an IRI, a path of length one.
2. ^path – inverse of path
3. path1 / path2 – concatenation of path1 followed by path2
4. path1 | path2 – alternative between path1 and path2 (try all possibilities)
5. path* – zero or more concatenation of path
6. path+ – one or more concatenation of path
7. path? – zero or one of path
8. !(iri1|...|irin) – an IRI not one of iri1,..., irin.
9. !(^iri1|...|^irin) – an IRI not one of reverse of iri1, ..., irin. Can be combined with the negated path expression in (8)
10. (path) – grouping of path with brackets to control precedence

Precedence from highest to lowest: IRI, negated property sets, groups, unary operators, unary inverse links, concatenation binary operator, binary operator for alternatives
Outline

1 Motivation
2 Basic and Complex Graph Patterns
3 Filters
4 Solution Modifiers
5 SPARQL Output Forms
6 Assignment of New Values
7 Aggregates
8 Subqueries
9 Property Path
10 Basic SPARQL Semantics
11 Other SPARQL Features
Need to clarify:

- User: “Which answers can I expect from my query?”
- Developer: “Which behavior is expected from my SPARQL implementation?”
- Marketing: “Is our product already conformant with SPARQL standard?”
Recall:

- **Logic-based semantics:**
  - Model-theoretic semantics: Which interpretations satisfy my knowledge base?
  - Proof-theoretic semantics: Which derivations can be obtained from my knowledge base?

- **Programming languages:**
  - Axiomatic semantics: Which logical statements hold for my program?
  - Operational semantics: What happens during the execution of my program?
  - Denotational semantics: How can we describe the input/output function of the program in an abstract way?

- **What about query languages?**
Query Entailment
- Query as a description of allowed results
- Data as a set of logical statements (set of axioms/theory)
- Results as logical entailment

  e.g., RDF(S), OWL DL as query languages; conjunctive queries; etc.

Query Algebra
- Query as instruction for computing results
- Queried data as input
- Results as output

  e.g., Relational algebra for SQL, SPARQL algebra
  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
}

The semantics of a SPARQL query is given by:

1. Transformation into a SPARQL algebra expression.
2. Evaluation of the algebra expression.
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
  FILTER (?price < 15)
}

Note: filters applies to the whole group in which they occur.
  FILTER (?price < 15)
}

1. Expand abbreviated IRIs.
  OPTIONAL { Bgp(?book <http://ex.org/title> ?title) }
}

2 Replace triple patterns with Bgp(·)
Introduce `LeftJoin(·)` for the optional parts. Indicate with “true” if the optional part has no filter.
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
    true)
  Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
  FILTER (?price < 15)
}

Combine alternatives with Union(·). Note: has higher precedence than conjunction of graph patterns and it’s left-associative.
{ Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
        true),
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
) FILTER (?price < 15)
}

5 Apply Join(·) to combine non-filter elements.
Filter(?price < 15,
  Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
             true),
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)

6 Translate a group with filters into expression with Filter(·) operator.
- **Bgp(\(P\))** – match/evaluate pattern \(P\)
- **Join(\(M_1, M_2\))** – conjunctive join of solutions of \(M_1\) and \(M_2\)
- **Union(\(M_1, M_2\))** – union of solutions of \(M_1\) and \(M_2\)
- **LeftJoin(\(M_1, M_2, F\))** – optional join (left join) of \(M_1\) and \(M_2\) with filter constraint \(F\) (**true** if no filter given)
- **Filter(\(F, M\))** – filter solutions with constraint \(F\)
- **Z** – empty pattern (identity for join operation)
A solution mapping of a BGP $P$, denoted $\text{Bgp}(P)$ over the active graph $G$ is a partial function $\mu$ such that:

1. the domain of $\mu$, denoted $\text{dom}(\mu)$, is exactly the set of variables in $P$
2. there exists an assignment $\sigma$ from blank nodes in $P$ to IRIs, blank nodes, or RDF literals such that the RDF graph $\mu(\sigma(P))$ is a subgraph of $G$.

Note:
- A solution mapping corresponds to one row in the result table.
- Compare this with the condition for RDF simple entailment!
- $\mu$ is a partial function: unbound variables are those that have no assigned value.
- for a partial function $\mu$ to be a solution, the existence of $\sigma$ as above is needed.

The result of evaluating a BGP $P$ over $G$, denoted $\mathbb{Bgp}(P)_G$ is a multiset of solution mappings.
A multiset $A$ over an underlying set $S$ can be defined as a set $A = \{\langle s, A(s) \rangle \mid s \in S\}$. Here, we slightly abuse the notation for the multiset $A$ by using it as a function where $A(s)$ is the multiplicity of $s$ in the multiset of $A$.

A multiplicity $A(s)$ of an element $s$ is the number of occurrences of $s$ in the multiset $A$.

For every multiset $A$ and any element $s$ in $A$, $A(s) \in \mathbb{Z}^+ \cup \{\omega\}$ where $\omega > n$ for every positive integer $n \in \mathbb{Z}^+$.

For any other $s$ not occurring in $A$, $A(s) = 0$.

Alternative notation: the multiset $A = \{\langle a, 2 \rangle, \langle b, 1 \rangle\}$ can be written as $\{a, a, b\}$.
Multiset operations, etc.

- **Equality** $\langle a, a, b \rangle = \langle b, a, a \rangle = \langle a, b, a \rangle = \{\langle a, 2 \rangle, \langle b, 1 \rangle\}$

- **Membership:**
  - $\langle a, 2 \rangle \in \langle a, a, b \rangle$, $\langle b, 1 \rangle \in \langle a, a, b \rangle$, $\langle a, 1 \rangle \notin \langle a, a, b \rangle$, $\langle c, 0 \rangle \notin \langle a, a, b \rangle$, $\langle c, 2 \rangle \notin \langle a, a, b \rangle$.
  - We write $a \in A$ if $A(a) > 0$, and $a \notin A$ otherwise.

- **Union:** $\langle a, a, a, b \rangle \cup \langle a, b \rangle = \langle a, a, a, b, c \rangle$
  - For multisets $A, B$, the union $A \cup B = \{\langle s, n \rangle \mid n = \max(A(s), B(s)) > 0\}$

- **Intersection:** $\langle a, a, a, b \rangle \cap \langle a, b, b, c \rangle = \langle a, b \rangle$
  - $A \cap B = \{\langle s, n \rangle \mid n = \min(A(s), B(s)) > 0\}$

- **Sum:** $\langle a, a, a, b \rangle \uplus \langle a, b, b, c \rangle = \langle a, a, a, a, b, b, b, c \rangle$
  - $A \uplus B = \{\langle s, n \rangle \mid n = A(s) + B(s) > 0\}$
Solution Mapping Example

ex:pascal foaf:maker [ 
  a ex:ConferencePaper ; 
  dc:title "GeoLink Ontology" ] .
ex:pascal foaf:maker [ r 
  a ex:JournalPaper ; 

Bgp(?who foaf:maker _:x . _:x dc:title ?what .)
Solution Mapping Example

```
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .
```

\[
\text{Bgp(?who foaf:maker _:x . _:x dc:title ?what .)}
\]
Solution Mapping Example

ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title ?what .)

\[ \mu_1: \ \ ?who \mapsto \text{ex:pascal}, \ ?what \mapsto \text{"GeoLink Ontology"} \]

\[ \sigma_1: \ \ _:x \mapsto \_:a \]
Solution Mapping Example

ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title ?what .)

μ₁:   ?who → ex:pascal,   ?what → "GeoLink Ontology"
σ₁:   _:x → _:a
σ₂:   _:x → _:b
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title ?what .)

\[\mu_1: \quad ?who \mapsto \text{ex:pascal}, \quad ?what \mapsto \text{"GeoLink Ontology"}\]
\[\sigma_1: \quad _:x \mapsto _:a\]
\[\mu_2: \quad ?who \mapsto \text{ex:pascal}, \quad ?what \mapsto \text{"Paraconsistent DL"}\]
\[\sigma_2: \quad _:x \mapsto _:b\]

Two solution mappings (\(\mu_1\) and \(\mu_2\)), each with multiplicity 1
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title _:y .)
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title _:y .)

$\mu_1: \quad ?\text{who} \mapsto \text{ex:pascal}$

$\sigma_1: \quad _:x \mapsto _:a , \quad _:y \mapsto "\text{GeoLink Ontology}"$
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .
ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title _:y .)

\[ \mu_1 : \begin{align*} 
?who & \mapsto ex:pascal \\
\sigma_1 : \begin{align*} 
_:x & \mapsto _:a, \\
_:y & \mapsto "GeoLink Ontology" 
\end{align*} \\
\mu_2 : \begin{align*} 
?who & \mapsto ex:pascal \\
\sigma_2 : \begin{align*} 
_:x & \mapsto _:b, \\
_:y & \mapsto "Paraconsistent DL" 
\end{align*} 
\end{align*} \]
ex:pascal foaf:maker _:a .
_:a rdf:type ex:ConferencePaper .
_:a dc:title "GeoLink Ontology" .

ex:pascal foaf:maker _:b .
_:b rdf:type ex:JournalPaper .
_:b dc:title "Paraconsistent DL" .

Bgp(?who foaf:maker _:x . _:x dc:title _:y .)

\[
\begin{align*}
\mu_1 : & \quad ?who \mapsto \text{ex:pascal} \\
\sigma_1 : & \quad _:x \mapsto _:a, \\
& \quad _:y \mapsto \text{"GeoLink Ontology"} \\
\mu_2 : & \quad ?who \mapsto \text{ex:pascal} \\
\sigma_2 : & \quad _:x \mapsto _:b, \\
& \quad _:y \mapsto \text{"Paraconsistent DL"}
\end{align*}
\]

One solution mapping with multiplicity 2, since \( \mu_1 = \mu_2 \).
Evaluation of more complex patterns (Union, Join, LeftJoin, Filter, etc.) is done based on the solution mapping of its BGP components. The involved solution mappings often need to be compatible to each other.

Compatibility of Solutions

Two solutions $\mu_1$ and $\mu_2$ are compatible iff for every $x$ for which both $\mu_1(x)$ and $\mu_2(x)$ are defined, $\mu_1(x) = \mu_2(x)$ holds.

That is, $\mu_1$ and $\mu_2$ are compatible iff they agree on their shared variables.

$\mu_1 = \{?x \mapsto \text{ex:a}, ?y \mapsto \text{ex:b}\}$ and $\mu_2 = \{?y \mapsto \text{ex:b}, ?z \mapsto \text{ex:c}\}$ are compatible.

$\mu_1 = \{?x \mapsto \text{ex:a}\}$ and $\mu_2 = \{?y \mapsto \text{ex:b}\}$ are compatible.

$\mu_1 = \{?x \mapsto \text{ex:a}, ?y \mapsto \text{ex:b}\}$ and $\mu_2 = \{?y \mapsto \text{ex:c}, ?z \mapsto \text{ex:d}\}$ are NOT compatible.
Two compatible solution mappings may need to be merged/combined when defining operations such as Join, etc.

Union of two compatible solution mappings corresponds to union of two matching table rows.

### Union of Compatible Solutions

Let $\mu_1$ and $\mu_2$ be two compatible solution mappings. Their union is denoted by $\mu_1 \cup \mu_2$ and defined as

$$
(\mu_1 \cup \mu_2)(x) = \begin{cases} 
\mu_1(x) & \text{if } x \in \text{dom}(\mu_1) \\
\mu_2(x) & \text{otherwise}
\end{cases}
$$

If we view $\mu_1$ and $\mu_2$ as sets of bindings, then $\mu_1 \cup \mu_2$ is really just the union of both sets and it is still a well-defined mapping since for every variable $x$ occurring in $\text{dom}(\mu_1) \cap \text{dom}(\mu_2)$, we have that $\mu_1(x) = \mu_2(x)$.
Evaluation of Union($P_1$, $P_2$) over a graph $G$ is denoted by $\bigcup \text{Union}(P_1, P_2) G$ and defined as:

$$\bigcup \text{Union}(P_1, P_2) G = \bigcup P_1 G \uplus \bigcup P_2 G$$

Note that $\bigcup P_1 G$ and $\bigcup P_2 G$ are both multisets of solution mappings.
Evaluation of $\text{Join}(P_1, P_2)$ over a graph $G$ is denoted by $[[\text{Join}(P_1, P_2)]]_G$ and defined as:

$$[[\text{Join}(P_1, P_2)]]_G = \bigcup_{\langle \mu_1, m_1 \rangle \in [P_1]_G} \{ \langle \mu_1 \cup \mu_2, m_1 m_2 \rangle | \mu_1 \text{ and } \mu_2 \text{ are compatible} \}$$
Example Evaluation of Join(·)

Suppose

\[ [P_1]_G = \{ \langle \mu_1 : ?x \mapsto ex:a, ?y \mapsto ex:b, 2 \rangle, \langle \mu_2 : ?x \mapsto ex:c, 1 \rangle \} \]
\[ [P_2]_G = \{ \langle \mu_3 : ?y \mapsto ex:b, ?z \mapsto ex:c, 2 \rangle, \langle \mu_4 : ?y \mapsto ex:d, 1 \rangle, \langle \mu_5 : ?z \mapsto ex:c, 1 \rangle \} \]

Then,

\[ [\text{Join}(P_1, P_2)]_G \]
\[ = \{ \langle \mu_1 \cup \mu_3, 2 * 2 \rangle \} \uplus \{ \langle \mu_1 \cup \mu_5, 2 * 1 \rangle \} \uplus \{ \langle \mu_2 \cup \mu_3, 1 * 2 \rangle \} \uplus \{ \langle \mu_2 \cup \mu_4, 1 * 1 \rangle \} \]
\[ \uplus \{ \langle \mu_2 \cup \mu_5, 1 * 1 \rangle \} \]
\[ = \{ \langle ?x \mapsto ex:a, ?y \mapsto ex:b, ?z \mapsto ex:c, 4 \rangle \} \uplus \{ \langle ?x \mapsto ex:a, ?y \mapsto ex:b, ?z \mapsto ex:c, 2 \rangle \} \]
\[ \uplus \{ \langle ?x \mapsto ex:c, ?y \mapsto ex:b, ?z \mapsto ex:c, 2 \rangle \} \uplus \{ \langle ?x \mapsto ex:c, ?y \mapsto ex:d, 1 \rangle \} \]
\[ \uplus \{ \langle ?x \mapsto ex:c, ?z \mapsto ex:c, 1 \rangle \} \]
\[ = \{ \langle ?x \mapsto ex:a, ?y \mapsto ex:b, ?z \mapsto ex:c, 6 \rangle, \langle ?x \mapsto ex:c, ?y \mapsto ex:b, ?z \mapsto ex:c, 2 \rangle, \langle ?x \mapsto ex:c, ?y \mapsto ex:d, 1 \rangle, \langle ?x \mapsto ex:c, ?z \mapsto ex:c, 1 \rangle \} \]
Evaluating Filter(·)

Given a filter condition $F$ and a solution mapping $\mu$ whose domain includes all variables in $F$, $\mu(F)$ is a Boolean expression obtained by replacing variables in $F$ with IRIs/literals/bnodes according to $\mu$. Furthermore, $\llbracket \mu(F) \rrbracket$ is the Boolean result of evaluating $\mu(F)$.

Evaluation of Filter($F$, $P$) over $G$ where $F$ is a filter condition and $P$ is an algebra object is denoted with $\lbrack \text{Filter}(F, P) \rbrack_G$ and defined as

$$\lbrack \text{Filter}(F, P) \rbrack_G = \{ \langle \mu, n \rangle \mid \llbracket P \rrbracket_G(\mu) = n > 0 \text{ and } \llbracket \mu(F) \rrbracket = \text{true} \}$$
Evaluation of $\text{LeftJoin}(\cdot)$

Let $P_1, P_2$ be algebra objects, $F$ be a filter condition over a graph $G$, and $M_1 = \llbracket P_1 \rrbracket_G$ and $M_2 = \llbracket P_2 \rrbracket_G$.

Then, the evaluation of $\text{LeftJoin}(P_1, P_2, F)$ over $G$ is denoted with $\llbracket \text{LeftJoin}(P_1, P_2, F) \rrbracket_G$ and defined as:

\[
\llbracket \text{LeftJoin}(P_1, P_2, F) \rrbracket_G = \llbracket \text{Filter}(F, \text{Join}(P_1, P_2)) \rrbracket_G
\]
\[
\cup \left\{ \langle \mu_1, M_1(\mu_1) \rangle \mid \mu_1 \in M_1 \text{ and there is no } \mu_2 \in M_2 \text{ compatible with } \mu_1 \right\}
\]
\[
\cup \left\{ \langle \mu_1, M_1(\mu_1) \rangle \mid \mu_1 \in M_1 \text{ and for all } \mu_2 \in M_2 : \llbracket (\mu_1 \cup \mu_2)(F) \rrbracket = \text{false} \right\}
\]
Evaluation Example

@prefix ex: <http://ex.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet ex:author ex:Shakespeare ;
ex:price "10.50"^^xsd:decimal .
ex:Macbeth ex:author ex:Shakespeare .
ex:Tamburlaine ex:author ex:Marlowe ;
ex:price "17"^^xsd:integer .
ex:DoctorFaustus ex:author ex:Marlowe ;
ex:price "12"^^xsd:integer ;
ex:title "The Tragical History of Doctor Faustus" .
ex:RomeoJulia ex:author ex:Brooke ;
ex:price "9"^^xsd:integer .

  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare }
  UNION
  { ?book ex:author ex:Marlowe }
}
Evaluation Example

@prefix ex: <http://ex.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet ex:author ex:Shakespeare ;
ex:price "10.50"^^xsd:decimal .
ex:Macbeth ex:author ex:Shakespeare .
ex:Tamburlaine ex:author ex:Marlowe ;
ex:price "17"^^xsd:integer .
ex:DoctorFaustus ex:author ex:Marlowe ;
ex:price "12"^^xsd:integer ;
ex:title "The Tragical History of Doctor Faustus" .
ex:RomeoJulia ex:author ex:Brooke ;
ex:price "9"^^xsd:integer .

Filter(?price < 15,
   Join(
      LeftJoin(Bgp(?book <http://ex.org/price> ?price),
               true),
      Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)
Filter(?price < 15,
    Join(
        LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true),
        Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
</tbody>
</table>
Filter(?price < 15, 
  Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
      true),
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)

<table>
<thead>
<tr>
<th>book</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>ex:Hamlet</td>
</tr>
</tbody>
</table>
Filter(?price < 15,
  Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
      true),
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
<tr>
<td>ex:Hamlet</td>
</tr>
<tr>
<td>ex:Macbeth</td>
</tr>
</tbody>
</table>
Filter(?price < 15,  
   Join(  
       LeftJoin(Bgp(?book <http://ex.org/price> ?price),  
           true),  
       Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),  
   ))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>
Filter(?price < 15, 
Join( 
    LeftJoin(Bgp(?book <http://ex.org/price> ?price), 
    true), 
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>), 

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:DoctorFaustus</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Filter(?price < 15,
   Join(
       LeftJoin(Bgp(?book <http://ex.org/price> ?price),
               true),
       Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
   ))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Filter(?price < 15, 
  Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true),
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Filter(?price < 15, 
  Join( 
    LeftJoin(Bgp(?book <http://ex.org/price> ?price), 
      true), 
    Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>), 
))

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Filter(?price < 15,
    Join(
        LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            true),
        Union(Bgp(?book <http://ex.org/author> <http://ex.org/Shakespeare>),
)

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
All other SPARQL constructs and solution modifiers are associated with their own particular algebra operations.

These operations are recursively evaluated.

See formal definition at https://www.w3.org/TR/sparql11-query/#sparqlDefinition
Outline

1 Motivation
2 Basic and Complex Graph Patterns
3 Filters
4 Solution Modifiers
5 SPARQL Output Forms
6 Assignment of New Values
7 Aggregates
8 Subqueries
9 Property Path
10 Basic SPARQL Semantics
11 Other SPARQL Features
Federated Query

Use keyword SERVICE to specify a remote endpoint. More than one remote endpoints can be specified in a query.

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT (SUM(?pop) AS ?P )
{ SERVICE <http://dbpedia.org/sparql/>
   { SELECT DISTINCT ?C ?pop
     WHERE {
       ?C dbo:populationTotal ?pop ;
       dbo:country dbr:United_States .
     }
     ORDER BY DESC ( ?pop )
     LIMIT 3
   }
}
DELETE removes triples. INSERT inserts triples.
- Can be done via standard HTTP protocol.
- Security issues not handled by SPARQL standards!

PREFIX ex: <http://example.org/>
DELETE { ?Item ex:price ?Pr }
INSERT { ?Item ex:price ?NewPr }
WHERE { ?Item ex:price ?Pr
    BIND (?Pr * 1.1 AS ?NewPr ) }
SPARQL Entailment Regimes

- SPARQL on its own is only capable of simple entailment.
  - BGP evaluation corresponds to simple entailment between two RDF graphs.
- To allow, e.g., RDF, RDFS or OWL entailments, SPARQL engine needs to be aware of the corresponding semantics of those languages.
- SPARQL standards define several entailment regimes, which specify how an entailment relation can be used to redefine the evaluation of BGPs in a SPARQL query.
- Specification is at https://www.w3.org/TR/sparql11-entailment/, providing
  - RDF entailment regime
  - RDFS entailment regime
  - D-entailment regime – correspond to datatype entailment
  - OWL 2 RDF-based Semantics entailment regime
  - OWL 2 Direct Semantics entailment regime
  - RIF Core entailment regime
Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?prop WHERE { ?prop rdf:type rdf:Property }
Example

Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?prop WHERE { ?prop rdf:type rdf:Property }

- Normal SPARQL query (i.e., simple entailment): empty answer.
Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?prop WHERE { ?prop rdf:type rdf:Property } 
  
  Normal SPARQL query (i.e., simple entailment): empty answer.
  
  With RDF entailment regime, we have the following answer:

<table>
<thead>
<tr>
<th>prop</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdf:type</td>
</tr>
<tr>
<td>rdf:type</td>
</tr>
<tr>
<td>rdfs:subClassOf</td>
</tr>
<tr>
<td>rdfs:range</td>
</tr>
<tr>
<td>ex:publishes</td>
</tr>
</tbody>
</table>

- Krisnadhi (DaSeLab - Wright State)
Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?pub WHERE { ?pub rdf:type ex:Publication }

Krisnadhi (DaSeLab - Wright State)
Example

Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?pub WHERE { ?pub rdf:type ex:Publication }
Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?pub WHERE { ?pub rdf:type ex:Publication }

With RDF entailment:

<table>
<thead>
<tr>
<th>pub</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:book1</td>
</tr>
</tbody>
</table>
Example

Consider data:

(1) ex:book1 rdf:type ex:Publication .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .

Query: SELECT ?pub WHERE { ?pub rdf:type ex:Publication }  

With RDFS entailment:

<table>
<thead>
<tr>
<th>pub</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:book1</td>
</tr>
<tr>
<td>ex:book2</td>
</tr>
<tr>
<td>ex:book3</td>
</tr>
</tbody>
</table>