

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[2.1] Introduction to SPARQL

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.

- **Your are free:**



to Share — to copy, distribute and transmit the work



to Remix — to adapt the work

- **Under the following conditions**



Attribution — You must attribute the work by inserting

- “© streamreasoning.org” at the end of each reused slide
- a credits slide stating
 - These slides are partially based on
“Streaming Reasoning for Linked Data 2011”
by Emanuele Della Valle and Jeff Z. Pan
<http://streamreasoning.org/sr4ld2011>

- To view a copy of this license, visit
<http://creativecommons.org/licenses/by/3.0/>

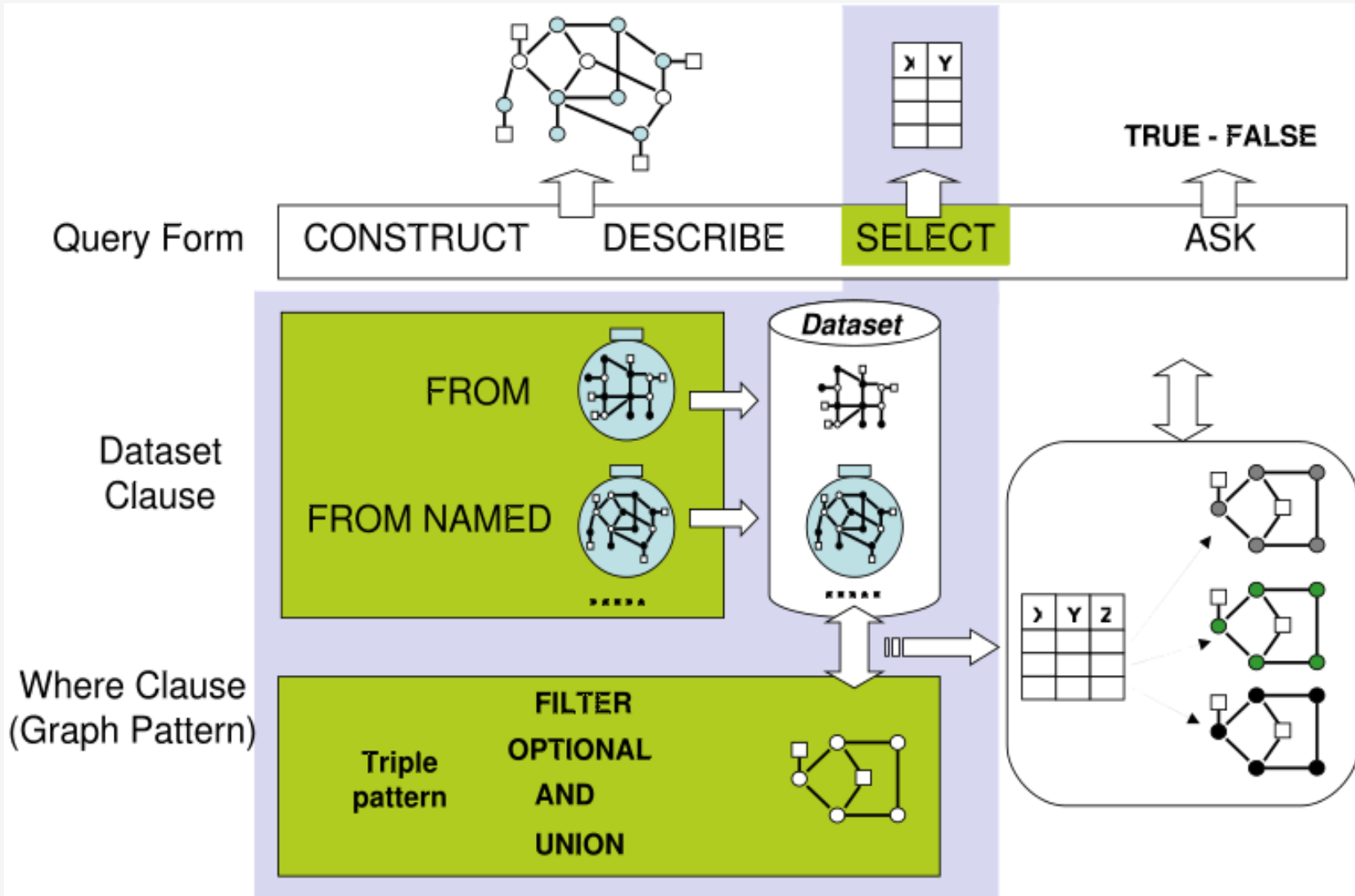
- Introduction
 - What is SPARQL?
 - Why SPARQL
 - Anatomy of a SPARQL query
- Writing simple queries
 - Graph Pattern Syntax
 - Matching RDF literals
 - Value Tests
- More sophisticated Graphs Patterns
 - OPTIONAL
 - UNION
- Results Forms
- Advance Features
 - Aggregates
 - Sub-queries
- Reasoning and Query Answering

- SPARQL
 - is the query language of the Semantic Web
 - stays for **SPARQL Protocol** and **RDF Query Language**

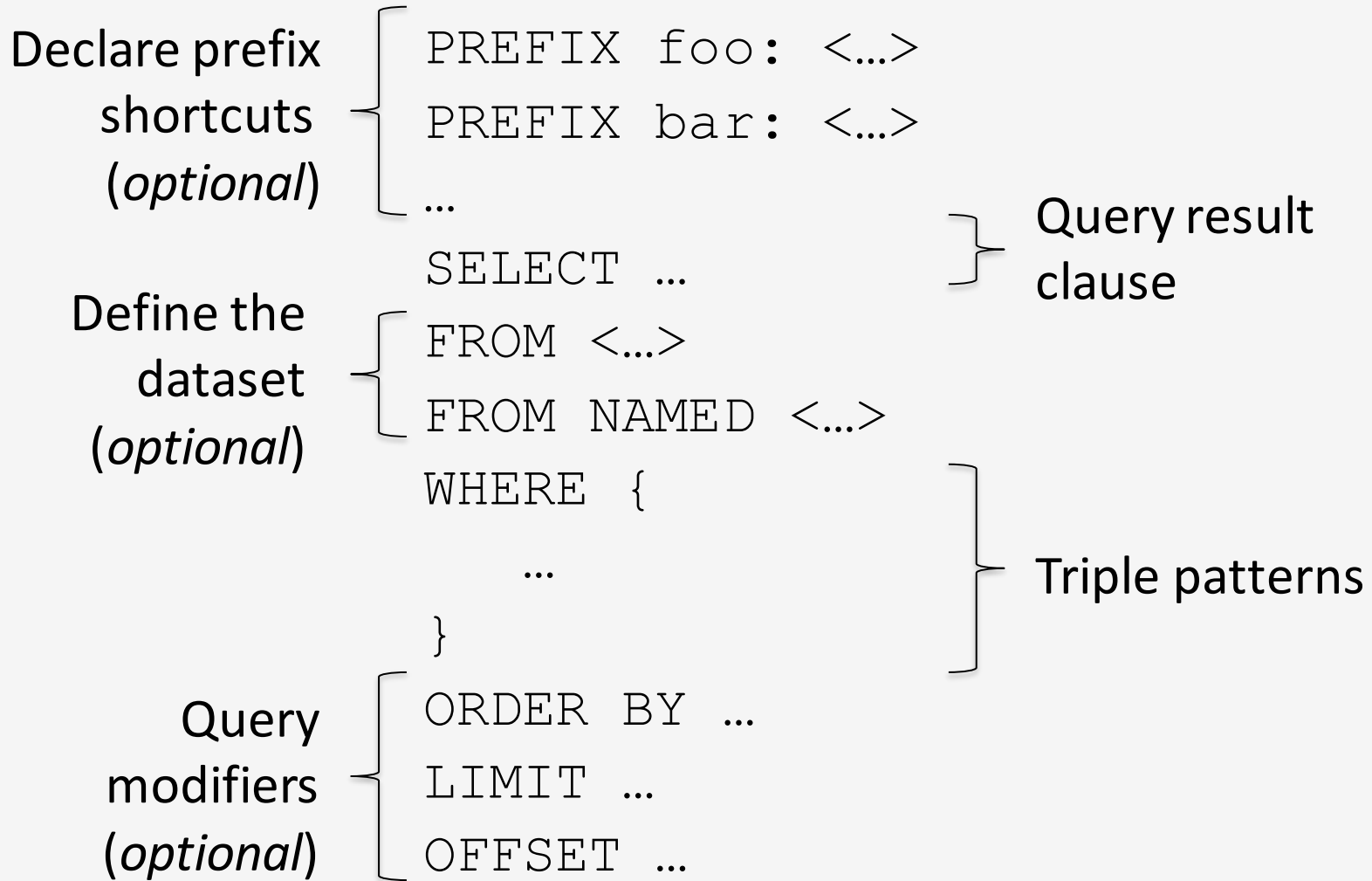


- A Query Language ...:
Find drugs in DBpedia:
- PREFIX sr: <<http://www.streamreasoning.org/sr4ld2011/onto#>>
SELECT ?poi
WHERE { ?poi a sr:NamedPlace . }
- ... and a Protocol.
`http://lod.openlinksw.com/sparql?&query=PREFIX+sr%3A+%3Chttp%3A%2F%2Fwww.streamreasoning.org%2Fsr4ld2011%2Fonto%2F%3E%0D%0ASELECT+%3Fpoi+WHERE+%3Fpoi+a+sr%3ANamedPlace+.+`

- SPARQL let us
 - Pull values from structured and semi-structured data represented in RDF
 - Explore RDF data by querying unknown relationships
 - Perform complex joins of disparate RDF repositories in a single query
 - Transform RDF data from one vocabulary to another
 - Develop higher-level cross-platform application

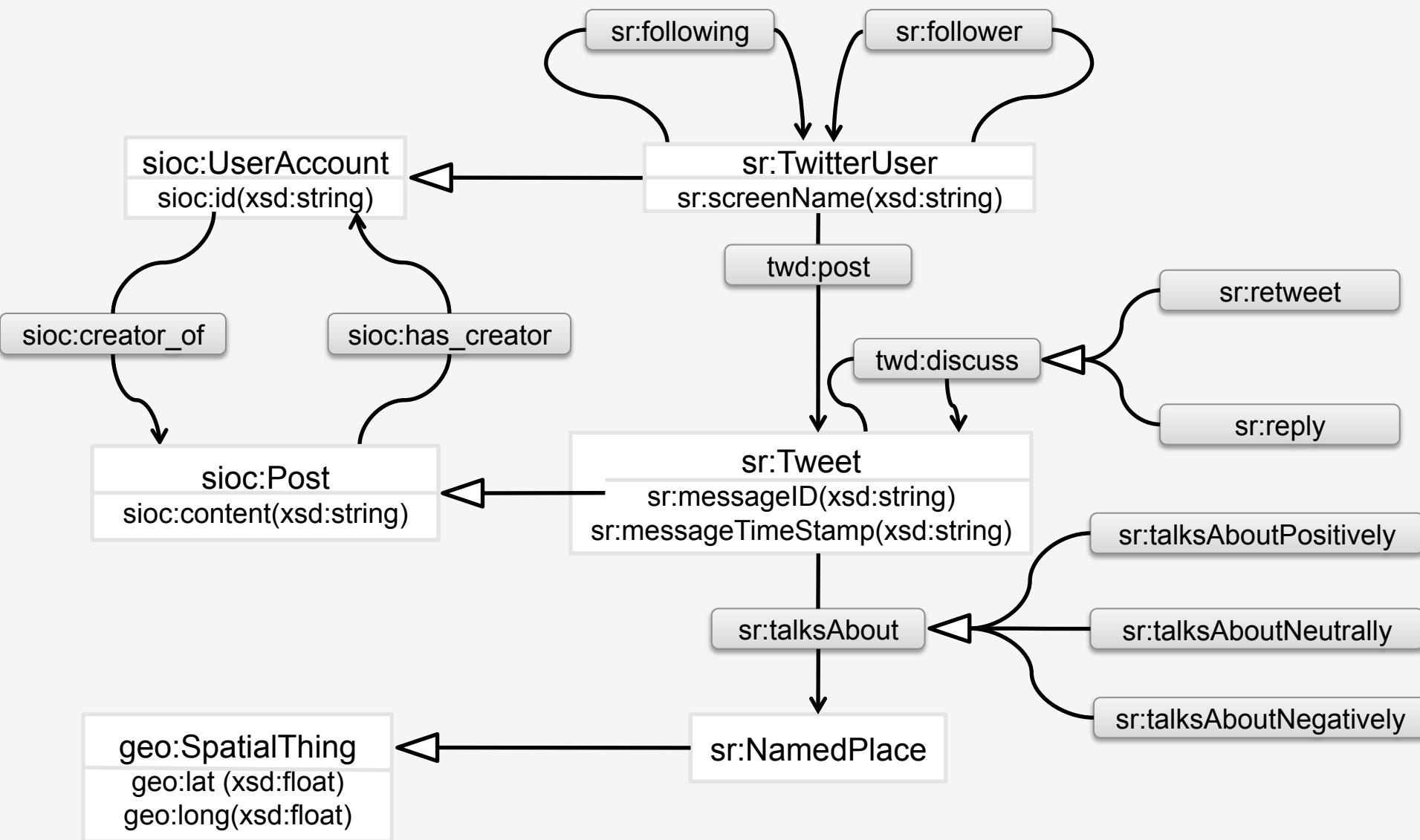


Source: Pérez, Arenas and Gutierrez, Chapter 1: On the Semantics of SPARQL, Semantic Web Information Management: A Model Based Perspective, Springer 2010



- Turtle-like: URIs, QNames, literals, convenience syntax.
- Adds variables to get basic patterns
 - ?var
 - Variable names are a subset of NCNames (no "-" or ".")
- E.g.,
 - simple
 - ?poi a sr:NamedPlace .
 - complex
 - ?poi a geo:NamedPlace .
 - ?poi skos:subject ?category .
- Adds
 - OPTIONAL to cope with semi-structured nature of RDF
 - FILTER to select solution according to some criteria
 - UNION operator to get complex patterns

Test data: Data Model



- Data

```
@prefix sr:<http://www.streamreasoning.org/sr4ld2011/onto#>.
sr:LaScala a sr:NamedPlace .
sr:GalleriaVittorioEmanueleII a sr:NamedPlace .
sr:Duomo a sr:NamedPlace .
```

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
SELECT ?poi
WHERE { ?poi a sr:NamedPlace . }
```

- Results

?poi

<http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII>

<http://www.streamreasoning.org/sr4ld2011/data#LaScala>

<http://www.streamreasoning.org/sr4ld2011/data#Duomo>

- *Matches the graph* means find a set of bindings such that the substitution of variables for values **creates a triple** that is in the set of triples making up the graph.
- Solution 1:
 - variable *poi* has value `sr:GalleriaVittorioEmanueleII`
 - Triple `sr:GalleriaVittorioEmanueleII a sr:NamedPlace .` is in the graph.
- Solution 2:
 - variable *poi* has value `sr:LaScala`
 - Triple `sr:LaScala a sr:NamedPlace .` is in the graph.
- Solution 3:
 - variable *poi* has value `sr:Duomo`
 - Triple `sr:Duomo a sr:NamedPlace .` is in the graph.
- No order of solutions in this query.

■ Query

```

PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi ?category
WHERE { ?poi a geo:NamedPlace ;
         skos:subject ?category . }

```

■ Results

?drug	?category
http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleI	http://dbpedia.org/resource/Category:Pedestrian_streets_in_Italy
http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII	http://dbpedia.org/resource/Category:Buildings_and_structures_in_Milan
http://www.streamreasoning.org/sr4ld2011/data#LaScala	http://dbpedia.org/resource/Category:Opera_houses_in_Italy
http://www.streamreasoning.org/sr4ld2011/data#Duomo	http://dbpedia.org/class/yago/ChurchesInMilan
...	...

- A basic Pattern is a set of triple patterns, all of which **must be matched**.
- In this case *matches the graph* means find a set of bindings such that the substitution of variables for values **creates a subgraph** that is in the set of triples making up the graph.

■ Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi sr:name "Duomo". }
```

■ Results

?poi

<http://www.streamreasoning.org/sr4ld2011/data#Duomo>

■ Alert!

- It may return 0 results if the literal have a language tag
 - E.g., if data contains only the triple
`sr:Duomo sr:name "Duomo"@it .`
- To obtain results also add the language tag to the triple pattern
 - E.g., `?poi sr:name "Duomo"@it.`

- As in the case of language tags, if the literals are typed (i.e., "3.14"^^xsd:float), they do not match if they are not given explicitly.

- Query

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi a sr:NamedPlace ;
          geo:lat "45.46416854858398"^^xsd:float ;
          geo:long "9.191389083862305"^^xsd:float . }
```

- Results

?poi

<http://www.streamreasoning.org/sr4ld2011/data#Duomo>

- SPARQL allows restricting solutions by applying the FILTER clause.
- An RDF term bound to a variable appears in the results if the FILTER expression, applied to the term, evaluates to TRUE.

- Query

```

PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?poi ?lat ?log
WHERE {
    ?poi geo:lat ?lat ; geo:long ?long .
    FILTER (
        ?lat > "45.46"^^xsd:float && ?lat < "45.47"^^xsd:float &&
        ?long > "9.18"^^xsd:float && ?long < "9.20"^^xsd:float )
}

```

- Results

?poi
http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII
http://www.streamreasoning.org/sr4ld2011/data#LaScala
http://www.streamreasoning.org/sr4ld2011/data#Duomo

- SPARQL FILTERs allows also restricting values of strings using the `regex()`

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?poi ?c
WHERE { ?poi rdfs:comment ?c .
        FILTER(regex(?c, "glass-vaulted arcades", "i" )) }
```

- Results

?poi	?c
http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII	The Galleria Vittorio Emanuele II is a covered double arcade formed of two glass-vaulted arcades at right angles intersecting in an octagon, prominently sited on the northern side of the Piazza del Duomo in Milan, and connects to the Piazza della Scala.

- Notation for value comparison: $<$, $>$, $=$, \leq , \geq and \neq
- Test functions
 - Check if a variable is bound: BOUND
 - Check the type of resource bound: isIRI, isBLANK, isLITERAL
- Accessing accessories: LANG, DATATYPE
- Logic operators: $||$ and $\&\&$
- Comparing strings: REGEX, langMatches
- Constructor functions: bool, dbl, flt, dec, int, dT, str, IRI
- Extensible Value Testing
 - E.g., FILTER (aGeo:distance(?axLoc, ?ayLoc, ?bxLoc, ?byLoc) < 10) .
 - (see <http://www.w3.org/TR/rdf-sparql-query/#extensionFunctions>)

- Find all schools within a 5km radius around a specific location, and for each school find coffeeshops that are closer than 1km.
- ```

PREFIX lgdo: <http://linkedgeodata.org/ontology/>
SELECT ?schoolname ?schoolgeo ?coffeeshopname ?coffeeshopgeo
WHERE {
 ?school a lgdo:School .
 ?school geo:geometry ?schoolgeo .
 ?school rdfs:label ?schoolname .
 ?coffeeshop a lgdo:CoffeeShop .
 ?coffeeshop geo:geometry ?coffeeshopgeo .
 ?coffeeshop rdfs:label ?coffeeshopname .
 FILTER (
 bif:st_intersects (
 ?schoolgeo, bif:st_point (4.892222, 52.373056), 5) &&
 bif:st_intersects (?coffeeshopgeo, ?schoolgeo, 1)
) .
}

```
- [Click here for query results](#) on a Virtuoso endpoint used by LinkedGeoData project.

- Signature
  - `st_intersects(g1, g2, prec)`
- Parameters
  - `g1` – The first geometry.
  - `g2` – The second geometry.
  - `prec` – A tolerance for the matching in units of linear distance appropriate to the srid. Default is 0.
- Description
  - Returns intersects between two geometries. If `prec` is supplied, this is a tolerance for the matching in units of linear distance appropriate to the srid. Both geometries should have the same srid. `st_intersects` is true if there is at least one point in common.

- RDF is "semi structured" and has no integrity constraints
- SPARQL addresses this issue with
  - **Group patterns** match if all subpatterns match and all constraints are satisfied
    - In SPARQL syntax, groups are { ... }
  - **OPTIONAL graph patterns** accommodate the need to add information to a result but without the query failing just because some information is missing.
    - In SPARQL syntax, OPTIONAL { ... }
  - **UNION graph patterns** allows to match alternatives
    - In SPARQL syntax, { ... } UNION { ... }

- **OPTIONAL graph patterns** accommodate the need to add information to a result but without the query failing just because some information is missing.

- Query

```

PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
SELECT ?t ?p ?poi
WHERE {
 ?t a sr:Tweet .
 ?poi a sr:NamedPlace .
 OPTIONAL { ?t ?p ?poi }
}

```

- Results

| ?t      | ?p                      | ?poi                           |
|---------|-------------------------|--------------------------------|
| _:post1 | sr:talksAbout           | srd:GalleriaVittorioEmanueleII |
| _:post1 | sr:talksAboutPositively | srd:LaScala                    |
| _:post1 | null                    | srd:Duomo                      |

- **UNION graph patterns** allows to match alternatives

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
SELECT ?t ?poi
WHERE {
 ?t a sr:Tweet .
 ?poi a sr:NamedPlace .
 { ?t sr:talksAbout ?poi }
 UNION
 { ?t sr:talksAboutPositively ?poi }
}
```

- Results

| ?t      | ?poi                           |
|---------|--------------------------------|
| _:post1 | srd:GalleriaVittorioEmanueleII |
| _:post1 | srd:LaScala                    |

- Besides selecting tables of values, SPARQL allows three other types of queries:
  - ASK - returns a boolean answering, does the query have any results?
  - CONSTRUCT - uses variable bindings to return new RDF triples
  - DESCRIBE - returns server-determined RDF about the queried resources
- SELECT and ASK results can be returned as XML or JSON.
- CONSTRUCT and DESCRIBE results can be returned via any RDF serialization (e.g. RDF/XML or Turtle).

### ■ Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
CONSTRUCT { ?u sr:talksAboutPositively ?poi }
WHERE {
 ?u a sr:TwitterUser .
 ?u sr:posts ?t .
 ?t sr:talksAboutPositively ?poi .
 ?poi a sr:NamedPlace .
}
```

### ■ Meaning

- it requires to compute a property chain



### ■ Results

```
@prefix sr:<http://www.streamreasoning.org/sr4ld2011/onto#> .
srd:Alice sr:talksAboutPositively srd:LaScala .
```

- How many points of interest are in the dataset?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT (count(?poi) AS ?numberOfPOI)
WHERE {
 ?poi a sr:NamedPlace .}
```

- How many points of interest are in each category?

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
 ?poi a sr:NamedPlace .
 ?poi skos:subject ?category .
}
GROUP BY ?category
```

- NOTES

- The available built-ins are SUM, AVG, COUNT, MIN, MAX
- A DISTINCT clause can be use to avoid processing duplicates

- Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category
WHERE {
 ?poi a sr:NamedPlace .
 ?poi skos:subject ?category .
 ?tweet sr:talksAboutPositively ?poi .
}
GROUP BY ?category
HAVING (count(?tweet) > 100)
```

- Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets? How many points of interest does each category contain?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
 ?poi a sr:NamedPlace .
 ?poi skos:subject ?category .
 ?tweet sr:talksAboutPositively ?poi .
}
GROUP BY ?category
HAVING (count(?tweet) > 100)
```

- Any group pattern can be a sub-query
- E.g., which are the categories of points of interest in which **each point of interest is positively discussed** in more than 10 tweets? How many points of interest does each category contain?
- ```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
  ?poi a sr:NamedPlace .
  ?poi skos:subject ?category .
  {
    SELECT ?poi
    WHERE {
      ?poi a sr:NamedPlace .
      ?tweet sr:talksAboutPositively ?poi .
    }
    GROUP BY ?poi
    HAVING (count(?tweet) > 10)
  }
} GROUP BY ?category
```

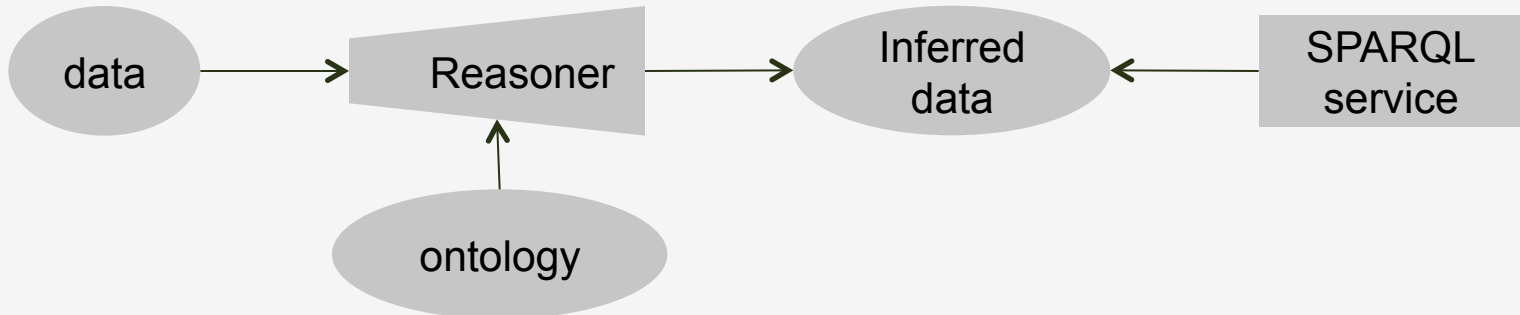
- When you need to output an RDF graph that contains the results of a query that uses aggregates, you need to use sub-queries.
- Output the points of interest in each category as an RDF graph

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
CONSTRUCT { ?category sr:countedPOIs ?numberOfPOIs }
WHERE {
  {
    SELECT ?category (count(?poi) AS ?numberOfPOIs)
    WHERE {
      ?poi a sr:NamedPlace .
      ?poi skos:subject ?category .
    }
    GROUP BY ?category
  }
}
```

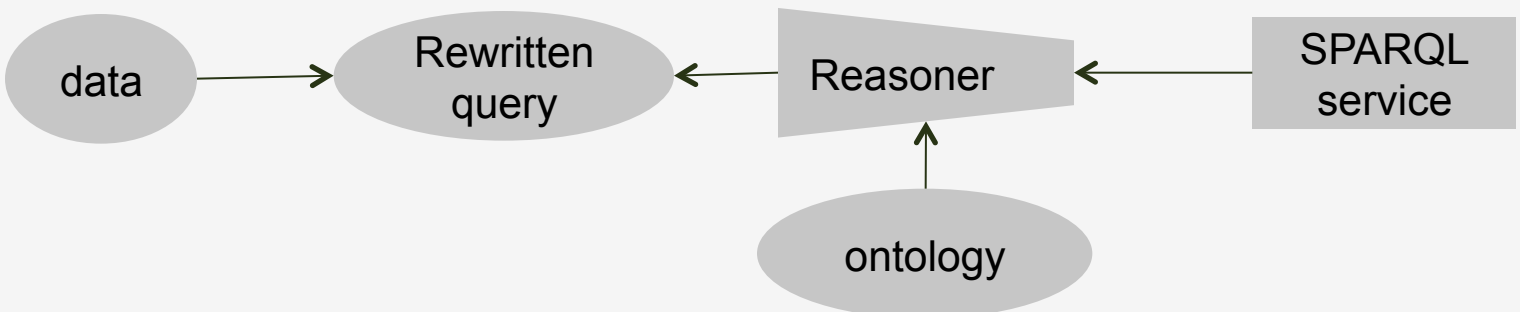
- SPARQL alone cannot answer queries that require reasoning



- but a reasoner can be exposed as a SPARQL service.



- Or a query can be rewritten in order to incorporate the ontology



- SPARQL Frequently Asked Questions
 - <http://thefigtrees.net/lee/sw/sparql-faq>
- SPARQL implementations - community maintained list of open-source and commercial SPARQL engines
 - <http://esw.w3.org/topic/SparqlImplementations>
- Public SPARQL endpoints - community maintained list
 - <http://esw.w3.org/topic/SparqlEndpoints>
- SPARQL extensions - collection of SPARQL extensions implemented in various SPARQL engines
 - <http://esw.w3.org/topic/SPARQL/Extensions>

- When cutting and pasting the SPARQL query from the slides into a query textbox you may get:

```
37000 Error SP030: SPARQL compiler, line 0:  
Invalid character in SPARQL expression at ' '
```

- A workaround is changing all (so called) *white spaces* in " "
 - e.g., in MS Word replace "^w" with " "

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[2.1] Introduction to SPARQL

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>