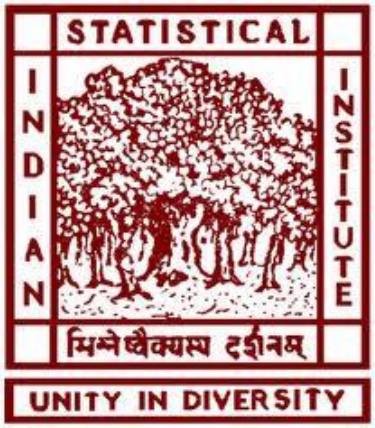

SPARQL: **SPARQL** **P**rotocol **a**nd **RDF** **Q**uery **L**anguage

Biswanath Dutta

bisu@drtc.isibang.ac.in

Documentation Research and Training Centre
Indian Statistical Institute
Bangalore

Introduction

- SPARQL is a query language for RDF graph traversal
  - SPARQL query language specification
- A protocol* for using SPARQL via HTTP
  - SPARQL Protocol for RDF Specification
- SPARQL queries and manipulates RDF graph content on the web or in the RDF store
- SPARQL is independent of any particular serialization format (e.g., RDF/XML, N3, Turtle)
- Inspired by SQL

- SPARQL 1.0: W3C Recommendation (15th January 2008) (https://www.w3.org/TR/rdf-sparql-query/)
- SPARQL 1.1: W3C Recommendation (21st March 2013) (https://www.w3.org/TR/sparql11-query/)

*a means of conveying SPARQL queries from query clients to query processors.

Introduction (contd…2)

- The SPARQL query results is usually displayed in Tabular form.

- SPARQL query results can be exchanged using any of the following formats:
  - Extensible Markup Language (XML)
  - JavaScript Object Notation (JSON)
  - Comma Separated Value (CSV)
  - Tab Separated Value (TSV).

SPARQL Four Query Forms

- These query forms use the solutions from pattern matching to form result sets or RDF graphs.
- SELECT query
  - Returns all, or a subset of, the variables bound in a query pattern match.
  - Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.
- CONSTRUCT query
  - Returns an RDF graph constructed by substituting variables in a set of triple templates.
  - Used to extract information from the SPARQL endpoint and transform the results into valid RDF.
- ASK query
  - Returns a boolean indicating whether a query pattern matches or not.
  - Used to provide a simple True/False result for a query on a SPARQL endpoint.
- DESCRIBE query
  - Returns an RDF graph that describes the resources found.
  - Used to extract an RDF graph from the SPARQL endpoint, the content of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Query Form: SELECT

Prefix mechanism to abbreviate URI

```
PREFIX a: <http://localhost/Institute#>

SELECT ?x ?y
    WHERE { ?x a:researchInterest ?y }
```

Variables to be returned

Query patterns (list of triple patterns)

Query Form: SELECT

- Variables: ?string
  - E.g., ?x, ?name, ?email, ?title, ?experts
- Syntax: SELECT var_1, var_2, … var_n
  - E.g.: SELECT ?name ?email

```
PREFIX a: <http://localhost/Institute#>

SELECT ?x ?y
    WHERE { ?x a:researchInterest ?y }
```

Alternative query:

```
SELECT ?x ?y
      WHERE { ?x <http://localhost/Institute#researchInterest> ?y .}
```

WHERE

- Graph patterns to match a set of triples
- Syntax: WHERE {

  subject predicate object .

  subject predicate object .

  }

```
PREFIX a: <http://localhost/Institute#>
SELECT ?student ?interest
    WHERE { ?x a:researchInterest ?y }
```

```
:John  rdf:type              :Senior_Research_Student .
:John    :hasQualification    :MLISc .
:John    :researchInterest :Ontology .
:John    :researchInterest :SWS .
:Mary  rdf:type              :Senior_Research_Student .
:Mary    :hasQualification    :MLISc .
:Mary    :researchInterest :Research_methodology .
```

Class inf. of Institutional Ontology (excerpt) (**Turtle** form)

```
<http://localhost/1#Academic_staff> rdf:type owl:Class ;
                                    rdfs:subClassOf <http://localhost/1#Staff> .
<http://localhost/1#AdminStaff> rdf:type owl:Class ;
                                rdfs:subClassOf <http://localhost/1#Staff> .
<http://localhost/1#AssistantProfessor> rdf:type owl:Class ;
                                        rdfs:subClassOf <http://localhost/1#TeachingStaff> .
<http://localhost/1#AssociateProfessor> rdf:type owl:Class ;
                                        rdfs:subClassOf <http://localhost/1#TeachingStaff> .
<http://localhost/1#BachelorStudent> rdf:type owl:Class ;
                                     rdfs:subClassOf <http://localhost/1#Student> .
<http://localhost/1#DoctoralStudent> rdf:type owl:Class ;
                                     rdfs:subClassOf <http://localhost/1#Student> .
<http://localhost/1#BachelorsProgramme> rdf:type owl:Class ;
                                        rdfs:subClassOf <http://localhost/1#Programme> .
<http://localhost/1#BookPublication> rdf:type owl:Class ;
                                     rdfs:subClassOf <http://localhost/1#Publication> .
```

A Simple Query and Result

```
SELECT ?class ?sub_class
    WHERE { ?class rdfs:subClassOf ?sub_class}
```

class	sub_class
Staff	Person
AssociateProfessor	TeachingStaff
AssistantProfessor	TeachingStaff
ProceedingsPublication	Publication
ResearchAssistant	Academic_staff
JournalPublication	Publication
ResearchProgramme	Programme
MastersProgramme	Programme
BachelorsProgramme	Programme
AdminStaff	Staff
BachelorStudent	Student
PostDoctoralStudent	Student

Data in Institutional Ontology

Data in Turtle:

<http://localhost/1#Inamdar> <http://localhost/1#researchInterest>
<http://localhost/1#Probability> .

<http://localhost/1#Manish> <http://localhost/1#researchInterest>
<http://localhost/1#KnowledgeOrganization> ,
<http://localhost/1#KnowledgeRepresentation> ,
<http://localhost/1#Ontology> ,
<http://localhost/1#SemanticDigitalLibrary> ,
<http://localhost/1#SemanticWeb> .

<http://localhost/1#Sasthi> <http://localhost/1#researchInterest>
<http://localhost/1#NLP> ,
<http://localhost/1#TextMining> .

.....

A Simple Query and Result

Alternative query:

```
PREFIX a: <https://w3id.org/into#>
SELECT ?name ?researchInterest
    WHERE { ?name a:hasResearchInterest ?researchInterest .}
        ORDER BY ?name
```

name	researchInterest
A.R.D._Prasad	Ontology
A.R.D._Prasad	Digital_library
A.R.D._Prasad	NLP
A.R.D._Prasad	AI
Anand_Kumar_Pandey	Digital_library
Biswanath_Dutta	SWS
Biswanath_Dutta	Ontology
I.K._Ravichandra_Rao	Bibliometrics
I.K._Ravichandra_Rao	Data_mining
JohnSmith	Automatic_classification
JohnSmith	Ontology

Multiple Match

- Graph patterns to match a set of triples
- Syntax: WHERE {

  subject predicate object .

  subject predicate object .

  }

E.g.,

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX a :<http://localhost/1#>
SELECT ?name ?topic
    WHERE {?person foaf:name ?name .
           ?person a:researchInterest ?topic . }
```

- The above query involves two triple patterns, each triple ends with a '.'
  - the dot ('.') after the last triple can be omitted.

Optional Clause

- SPARQL also allows to define OPTIONAL blocks.
- They offer the ability to query for data but not to fail query when that data does not exist.
- Optional blocks define additional graph patterns that do bind to the graph when they can be matched, but do not cause solutions to be rejected if they are not matched.

Optional Pattern Matching

```
SELECT ?name ?email
Where { ?person foaf:name ?name .
        OPTIONAL { ?person :mbox ?email .}
      }
```

name	email
"Saroj Meher"^^<http://www.w3.org/2001/XMLSchema#string>	"saroj@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"Manish Kumar"^^<http://www.w3.org/2001/XMLSchema#string>	"manish@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"Sasthi"^^<http://www.w3.org/2001/XMLSchema#string>	
"Inamdar Murthy"^^<http://www.w3.org/2001/XMLSchema#string>	"inamdar@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"Ranaki Jha"^^<http://www.w3.org/2001/XMLSchema#string>	
"Bhaskar Bose"^^<http://www.w3.org/2001/XMLSchema#string>	"bhaskar@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"TSS Rao"^^<http://www.w3.org/2001/XMLSchema#string>	"rao@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"John Smith"^^<http://www.w3.org/2001/XMLSchema#string>	"john@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"Marry Smith"^^<http://www.w3.org/2001/XMLSchema#string>	"marry@example.com"^^<http://www.w3.org/2001/XMLSchema#string>
"Smitha Pal"^^<http://www.w3.org/2001/XMLSchema#string>	"smitha@example.com"^^<http://www.w3.org/2001/XMLSchema#string>

Filter Clause

- SPARQL allows to pose restrictions on the values in query solutions.
- These restrictions are defined in FILTER clauses.
- These clauses are, to some extent, similar to WHERE clause of an SQL query.
- SPARQL filters can be used to restrict:
  - String values (using REGEX function),
  - Numeric values (using <, >, =, <=, >= and != operators).
- SPARQL also provides test functions:
  - BOUND, isURI, isBLANK, isLITERAL

Filter – String matching

SPARQL provides an operation to test strings, based on regular expressions

This allows to ask SQL "LIKE" style tests, although the syntax of the regular expression is different from SQL.

Syntax:

FILTER regex(?x, "pattern" [, "flags"])

*Regex invokes the XPath *fn:matches* function to match text against a regular expression pattern. The regular expression language is defined in XQuery 1.0 and XPath 2.0

**The flags argument is optional. The flag "i" means a case-insensitive pattern match is done.

Filter – String matching (contd…2)

Data:

<http://localhost/1#Sasthi> <http://xmlns.com/foaf/0.1/name> "Sasthi"^^xsd:string .
<http://localhost/1#Saroj> <http://xmlns.com/foaf/0.1/name> "Saroj Meher"^^xsd:string .
<http://localhost/1#Smitha> <http://xmlns.com/foaf/0.1/name> "Smitha Pal"^^xsd:string .

```
SELECT ?name
        WHERE { ?person foaf:name ?name .
         FILTER regex(?name, "^^s", "i")
         }
```

name
"Saroj Meher"^^<http://www.w3.org/2001/XMLSchema#string>
"Sasthi"^^<http://www.w3.org/2001/XMLSchema#string>
"Smitha Pal"^^<http://www.w3.org/2001/XMLSchema#string>

Filter - Arithmetic Expression

```
<http://localhost/1#John> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                    <http://xmlns.com/foaf/0.1/age> 26 ;
<http://localhost/1#Marry> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                      <http://xmlns.com/foaf/0.1/age> 25 ;
<http://localhost/1#Mithun> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual
;
                    <http://xmlns.com/foaf/0.1/age> 29 ;
<http://localhost/1#Vijay> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                    <http://xmlns.com/foaf/0.1/age> 26 ;
```

```
SELECT ?student ?age
WHERE {?student rdf:type :DoctoralStudent;
            foaf:age ?age .
            FILTER (?age <= 26) }
```

student	age
John	"26"^^<http://www.w3.org/2001/XMLSchema#integer>
Marry	"25"^^<http://www.w3.org/2001/XMLSchema#integer>
Vijay	"26"^^<http://www.w3.org/2001/XMLSchema#integer>

Optional + Filter

```
<http://localhost/1#John> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                <http://xmlns.com/foaf/0.1/age> 26 ;
<http://localhost/1#Marry> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual
;
                <http://xmlns.com/foaf/0.1/age> 25 ;
<http://localhost/1#Mithun> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual
;
                <http://xmlns.com/foaf/0.1/age> 29 ;
<http://localhost/1#Vijay> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                <http://xmlns.com/foaf/0.1/age> 26 ;
```

```
SELECT ?student ?age
WHERE { ?student rdf:type :DoctoralStudent .
        OPTIONAL { ?student foaf:age ?age .
                FILTER (?age <= 26)
                }
        }
```

student	age
John	"26"^^<http://www.w3.org/2001/XMLSchema#integer>
Vijay	"26"^^<http://www.w3.org/2001/XMLSchema#integer>
Mithun	
Marry	"25"^^<http://www.w3.org/2001/XMLSchema#integer>

Optional + Filter

```
<http://localhost/1#John> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                    <http://xmlns.com/foaf/0.1/age> 26 ;
<http://localhost/1#Marry> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual
;
                        <http://xmlns.com/foaf/0.1/age> 25 ;
<http://localhost/1#Mithun> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual
;
                    <http://xmlns.com/foaf/0.1/age> 29 ;
<http://localhost/1#Vijay> rdf:type <http://localhost/1#DoctoralStudent> , owl:NamedIndividual ;
                    <http://xmlns.com/foaf/0.1/age> 26 ;
```

```
SELECT ?student ?age
WHERE { ?student rdf:type :DoctoralStudent .
        OPTIONAL { ?student foaf:age ?age .
                FILTER (?age != 26)
                }
        }
```

Answer: Will display all the names and their ages, expect the person who's age is 26. Here, John and Vijay will not be shown in the result as both of them are 26 years old.
Here, "!=" refers to "not equal".

Matching Alternatives: Union

- SPARQL supports combining graph patterns so that one of several alternative graph patterns may match.
- If there is a match of more than one alternatives, all the possible pattern solutions are found.
- Pattern alternatives are syntactically specified with the UNION keyword.

Data in Institutional Ontology

<http://localhost/1#Bhaskar> <http://xmlns.com/foaf/0.1/name> "Bhaskar Bose"^^xsd:string .

<http://localhost/1#Ranaki> <http://xmlns.com/foaf/0.1/name> "Ranaki Jha"^^xsd:string .

<http://localhost/1#Saroj> <http://xmlns.com/foaf/0.1/name> "Saroj Meher"^^xsd:string .

<http://localhost/1#TSSRAO> <http://xmlns.com/foaf/0.1/name> "TSS Rao"^^xsd:string .

<http://localhost/1#Inamdar> <http://xmlns.com/foaf/0.1/name> "Inamdar Murthy"^^xsd:string .

…….

<http://localhost/1#Shiv> vcard:FN "Shiv Pal"^^xsd:string .

<http://localhost/1#Smitha> vcard:FN Smitha Jha .

<http://localhost/1#Tanmay> vcard:FN Tanmay Bose .

Matching Alternatives: Union

```
SELECT ?foafName ?vcardName
WHERE {
      {?x foaf:name ?foafName .}
    UNION
      {?y vcard:FN ?vcardName .}
    }
```

foafName	vcardName
"Bhaskar Bose"^^<http://www.w3.org/2001/XMLSchema#string>	
"Ranaki Jha"^^<http://www.w3.org/2001/XMLSchema#string>	
"Saroj Meher"^^<http://www.w3.org/2001/XMLSchema#string>	
"TSS Rao"^^<http://www.w3.org/2001/XMLSchema#string>	
"Inamdar Murthy"^^<http://www.w3.org/2001/XMLSchema#string>	
"Smitha Pal"^^<http://www.w3.org/2001/XMLSchema#string>	
"Manish Kumar"^^<http://www.w3.org/2001/XMLSchema#string>	
"John Smith"^^<http://www.w3.org/2001/XMLSchema#string>	
"Sasthi"^^<http://www.w3.org/2001/XMLSchema#string>	
	"Shiv Pal"^^<http://www.w3.org/2001/XMLSchema#string>
	"Smitha Jha"^^<http://www.w3.org/2001/XMLSchema#integer>
	"Tanmay Bose"^^<http://www.w3.org/2001/XMLSchema#integer>

Algebra – Count

- Counting the number of places a person lives in.
- The following query displays the person names and the total number of places they live. Here, the grouping is by "person."

```
SELECT ?person (COUNT (?live) AS ?InNoOfPlacesLivesIn)
WHERE {?person a:livesIn ?live .}
        GROUP BY ?person
```

person	InNoOfPlacesLivesIn
Anne	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
John	"1"^^<http://www.w3.org/2001/XMLSchema#integer>

Algebra – Count (contd…2)

Counting the number of people living is a place.

The following query displays the place names and the total number of people living in that place. Here, the grouping is by "place name."

```
SELECT ?placeName (COUNT (?placeName) AS
    ?noOfPeopleLiveInAPlace)
WHERE {?x a:livesIn ?placeName .}
        GROUP BY ?placeName
```

placeName	noOfPeopleLiveInAPlace
Bangalore	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
France	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
USA	"1"^^<http://www.w3.org/2001/XMLSchema#integer>

Algebra – Count (contd…3)

```
PREFIX a: <https://w3id.org/into#>
SELECT ?person (COUNT (?y) AS ?researchInterest)
WHERE {?person a:hasResearchInterest ?y .}
        GROUP BY ?person
```

person	researchInterest
A.R.D._Prasad	"4"^^<http://www.w3.org/2001/XMLSchema#integer>
Anand_Kumar_Pandey	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
TSSK_Rao	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
KS_Raghavan	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
SS_Panza	"1"^^<http://www.w3.org/2001/XMLSchema#integer>
JohnSmith	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
Biswanath_Dutta	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
I.K._Ravichandra_Rao	"2"^^<http://www.w3.org/2001/XMLSchema#integer>
Nabonita_Guha	"2"^^<http://www.w3.org/2001/XMLSchema#integer>

Querying for Properties and schema

```
Select ?property ?value
Where {:Manish ?property ?value}
```

This query is very powerful. Because we may not know what properties are defined for a resource (here, say for :Manish), but this query will find them along with their values.

Alternatively, we can make the following query, which tells "what are the sorts of things (i.e., metadata) this dataset knows about :Manis?"

```
Select DISTINCT ?property
Where {:Manish ?property ?value}
```

*Here, DISTINCT to filter out the duplicate results.

Querying for Properties and schema (Contd…2)

This ability of querying the properties, make it possible to reverse-engineer schema information from the data itself. For instance, we change the query about properties used to describe :Manish to find all properties used to describe any :Faculty.

```
Select DISTINCT ?property
Where {?x a :Faculty .
       ?x ?property ?object .}
```

Note: If we don't know about the class :Faculty, we can ask about that as well:

```
Select DISTINCT ?class
Where {?class rdfs:subClassOf  :Faculty}
```

Querying for Properties and schema (Contd…3)

If we do not know anything about the data at all, we can find the classes used in the data.

```
Select DISTINCT ?class
Where {?x a ?class}
```

Also, alternatively, find all the properties used anywhere in the data.

```
Select DISTINCT ?property
Where {?x ?property ?y}
```

ASK Query

The query is:

ASK WHERE {a:Anne a:age ?any}

The answer/result is: True

References

1. SPARQL 1.1 Query Language (W3C Recommendation). http://www.w3.org/TR/sparql11-query/
2. Corno, Fulvio (2012). SPARQL and Linked Data. http://www.slideshare.net/
3. Ghaiwi, R. (2013). SPARQL.