More on SPARQL

Acknowledgements

- This presentation is based on the W3C Candidate Recommendation “SPARQL Query Language for RDF” (http://www.w3.org/TR/rdf-sparql-query/) and the W3C working draft for SPARQL 1.1 (http://www.w3.org/TR/sparql11-query/) .

- Much of the material in this presentation is verbatim from the above Web sites.

Presentation Outline

- **Negation in SPARQL**
- Other Useful Features of SPARQL
- Named Graphs
- Querying RDFS information using SPARQL
- SPARQL 1.1 features

Negation in SPARQL

- SPARQL offers two forms of negation:
  - The Boolean "**not**" (`!`) operator in FILTER conditions.
  - A limited form of **negation as failure** which can be simulated using OPTIONAL, FILTER and `!bound`.

- SPARQL does not offer an explicit algebraic **difference** operator (but this operator can be simulated in SPARQL as we will see below).

- See later what SPARQL 1.1. offers.

Negation in FILTER conditions

- **Data:**

```
@prefix ns: <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
_:a ns:p "42"^^xsd:integer .
```

- **Query:**

```
PREFIX ns: <http://example.org/ns#>
SELECT ?v
WHERE {
        ?v ns:p ?y . FILTER (?y != 42)
      }
```

- **Result:**

v

The Operator `bound`

- The expression `bound(var)` is one of the expresssions allowed in FILTER conditions.

- Given a mapping to which FILTER is applied, `bound(var)` evaluates to true if `var` is bound to a value in that mapping and false otherwise.

Example

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/schema/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:a foaf:givenName "Alice“ .

_:b foaf:givenName "Bob" .
_:b ex:age “30"^^xsd:integer .

_:m foaf:givenName “Mike" .
_:m ex:age “65"^^xsd:integer .
```

Example (cont'd)

- **Query:** Find the names of people with name and age.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.org/schema/>
SELECT ?name
WHERE { ?x foaf:givenName ?name . ?x ex:age ?age }
```

- **Result:**

name
"Bob"
"Mike"

Examples with Negation

- **Query**: Find people with a name but **no** expressed age:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.org/schema/>
SELECT ?name
WHERE { ?x foaf:givenName ?name .
        OPTIONAL { ?x ex:age ?age }
        FILTER (!bound(?age))
      }
```

- **Result**:

name
"Alice"

Examples with Negation (cont'd)

- **Query:** Find the names of people with name but **no** expressed age **or** age less than 60 years.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.org/schema/>
SELECT ?name
WHERE { ?x foaf:givenName ?name .
        OPTIONAL { ?x ex:age ?age .
                   FILTER(?age >= 60) } .
        FILTER (!bound(?age))
      }
```

- **Result:**

name
"Alice"
"Bob"

Examples with Negation (cont'd)

- Note that the OPTIONAL pattern in the previous query does not generate bindings in the following two cases:
  - There is no `ex:age` property for `?x` (e.g., when `?x=_a`).
  - There is an `ex:age` property for `?x` but its value is less than 60 (e.g., when `?x=_b`).

- These two cases are then selected for output by the FILTER condition that uses `!bound`.

Negation in SPARQL (cont'd)

- In the previous examples where we used

```
{ P1 OPTIONAL P2 } FILTER(!bound(?x) }
```

  to express negation as failure, the variable `?x` appeared in graph pattern `P2` but not in graph pattern `P1` otherwise we cannot have the desired effect.

- The paper

  Renzo Angles, Claudio Gutierrez. *The Expressive Power of SPARQL.* Proc. of ISWC 2008.

  shows that this simple idea might not work in more complicated cases, and shows a general way to express **difference** of graph patterns in SPARQL (using again `OPTIONAL, FILTER` and `!bound)`.

Negation in SPARQL (cont'd)

- We saw that it is possible to simulate a **non-monotonic** construct (negation as failure) through SPARQL language constructs.

- However, SPARQL makes **no** assumption to interpret statements in an RDF graph using negation as failure or some other non-monotonic assumption (e.g., **closed world assumption**).

- SPARQL (but also RDF and RDFS) make the **Open World Assumption.**

Monotonicity of FOL

- **Theorem.** Let *KB* be a set of FOL formulas and φ and θ two arbitrary FOL formulas. If *KB* entails φ then *KB* union $\{ \theta \}$ entails φ as well.

- The above theorem captures the **monotonicity** property of FOL.

Closed World Assumption (CWA) and Negation as Failure (NF)

- If *A* is a ground atomic formula in FOL, then the **closed world assumption** says:
  - If *KB* does not entail *A*, then assume *not A* to be entailed.

- If *A* is a ground atomic formula in FOL, then **negation as failure** says:
  - If you cannot prove *A* from the KB, then assume *not A* has been proven.

- CWA and NF result in **non-monotonicity**.

Example (relational databases or Prolog)

- DB: `tall(John)`
- Query: `?-tall(John).`
  Answer: `yes`
- Query: `?-tall(Mike)`
  Answer: `no` (using the CWA or negation as failure).
- Update DB with `tall(Mike).`
- Query: `?-tall(Mike)`
  Answer: `yes`

The Open World Assumption

- Things that are not known to be true or false are assumed to be **possible**.

Example (revisited)

- DB: `tall(John)`
- Query: `?-tall(John).`
  Answer: `yes`
- Query: `?-tall(Mike)`
  Answer: `I don't know` (using the OWA).

OWA vs. CWA in RDF

- In general, the OWA is the most natural assumption to make in RDF since we are writing **incomplete Web resource descriptions** and we expect that these resource descriptions will be **extended** and **reused** by us or others later on.

- But even in the world of Web resources, there are many examples where the CWA is more appropriate (e.g., when we describe the schedule of a course we give the times the course takes place; the course does **not** take place at any other time).

- It would be nice to have facilities to say what assumption to make in each case.

Presentation Outline

- Negation in SPARQL
- **Other Useful Features of SPARQL**
- Named Graphs
- Querying RDFS information using SPARQL
- SPARQL 1.1 features

Solution Sequences and Modifiers

- Graph patterns in a WHERE clause generate an **unordered collection** of solutions, each solution being a mapping i.e., a **partial function** from variables to RDF terms.

- These solutions are then treated as a sequence (a **solution sequence**), initially in no specific order; any **sequence modifiers** are then applied to create another sequence.

- Finally, this latter sequence is used to **generate the results** of a SPARQL query form.

Solution Sequences and Modifiers (cont'd)

- A **solution sequence modifier** is one of:
  - **Order modifier**: put the solutions in some given order.
  - **Projection modifier**: choose certain variables. This is done using the SELECT clause.
  - **Distinct modifier**: ensure solutions in the sequence are unique.
  - **Reduced modifier**: permit elimination of some non-unique solutions.
  - **Offset modifier**: control where the solutions start from, in the overall sequence of solutions.
  - **Limit modifier**: restrict the number of solutions.

- Solution sequence modifiers are introduced by certain clauses or keywords to be defined below.

The ORDER BY clause

- The ORDER BY clause and the optional order modifier ASC() or DESC() establish the order of a solution sequence.

- Example:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name }
ORDER BY ?name
```

- When ASC() and DESC() are missing, ASC() is assumed.

- The SPARQL specification defines the exact order among various values that can appear in the mappings that form a solution.

Examples

```
PREFIX : <http://example.org/ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?name ?emp
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY DESC(?emp)
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?emp
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC(?emp)
```

Removing Duplicates

- By default, SPARQL query results may contain **duplicates** (so the result of a SPARQL query is a **bag** not a set).

- The modifier DISTINCT enforces that no duplicates are included in the query results.

- The modifier REDUCED permits the elimination of duplicates (the implementation decides what to do e.g., based on optimization issues).

Example

- Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:x foaf:name "Alice" .
_:x foaf:mbox <mailto:alice@example.com> .

_:y foaf:name "Alice" .
_:y foaf:mbox <mailto:asmith@example.com> .

_:z foaf:name "Alice" .
_:z foaf:mbox <mailto:alice.smith@example.com> .
```

Example (cont'd)

- Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name }
```

- Answer:

name
"Alice"
"Alice"
"Alice"

Example (cont'd)

- Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name
WHERE { ?x foaf:name ?name }
```

- Answer:

name
"Alice"

OFFSET and LIMIT clauses

- The OFFSET clause causes the solutions generated to start **after the specified number of solutions**. An OFFSET of zero has no effect.

- The LIMIT clause puts **an upper bound on the number of solutions returned.** If the number of actual solutions is greater than the limit, then at most the limit number of solutions will be returned.

- Using LIMIT and OFFSET to select different subsets of the query solutions is not useful unless the order is made predictable by using ORDER BY.

Examples

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name }
ORDER BY ?name
LIMIT 5
OFFSET 10
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

Presentation Outline

- Negation in SPARQL
- Other Useful Features of SPARQL
- **Named Graphs**
- Querying RDFS information using SPARQL
- SPARQL 1.1 features

RDF Datasets

- An **RDF dataset** is a collection of graphs against which we can execute a SPARQL query.

- An RDF dataset consists of:
  - one graph, the **default graph**, which does not have a name.
  - zero or more **named graphs**, where each named graph is identified by an IRI.

- An RDF dataset may contain **zero named graphs**. An RDF dataset always contains **one default graph**.

- A query does not need to involve matching the default graph; the query **can just involve matching named graphs**.

RDF Datasets (cont'd)

- The definition of RDF Dataset **does not restrict** the relationships of named and default graphs.

- Examples of relationships:
  - There is no information in the default graph. We just have named graphs and queries relate information from the two graphs.
  - The information in the default graph includes **provenance information** about the named graphs. Queries may use this provenance information.

Example: the default graph contains provenance

- **Default graph**

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
<http://example.org/bob> dc:publisher "Bob" .
<http://example.org/alice> dc:publisher "Alice" .
```

- **Named graph: http://example.org/bob**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

- **Named graph: http://example.org/alice**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example.org> .
```

Specifying RDF Datasets

- The specification of the RDF dataset for a query is done using the **FROM** and **FROM NAMED** clauses of the query.
- A SPARQL query may have **zero or more** FROM clauses and **zero or more** FROM NAMED clauses.
- The **RDF dataset** resulting from a number of FROM and FROM NAMED clauses consists of:
  - a **default graph** consisting of the **RDF merge** of the graphs referred to in the FROM clauses.
  - a **set of (IRI, graph) pairs**, one from each FROM NAMED clause.
- If there is **no FROM clause** then the dataset is assumed to have the **empty graph** as the default graph.

Specifying RDF Datasets (cont'd)

- A **merge** of a set of RDF graphs is defined as follows:
  - If the graphs in the set have no blank nodes in common, then the union of the graphs is a merge.
  - If the graphs share blank nodes, then it is the union of a set of graphs that is obtained by replacing the graphs in the set by equivalent graphs that share no blank nodes (blank nodes are standardized apart).

- See the **RDF semantics** for more related concepts http://www.w3.org/TR/rdf-mt/

Specifying RDF Datasets (cont'd)

- The RDF dataset may also be specified in a **SPARQL protocol request**, in which case the protocol description overrides any description in the query itself.

Simple Example

- **Default graph, stored at** `http://example.org/foaf/aliceFoaf`**:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
FROM http://example.org/foaf/aliceFoaf
WHERE { ?x foaf:name ?name }
```

- **Answer:**

name
"Alice"

Queries with GRAPH

- When querying a collection of named graphs, the **GRAPH keyword** is used to match patterns against named graphs.
- GRAPH can **provide an IRI** to select one graph or **use a variable** which will range over the IRIs of all the named graphs in the query's RDF dataset and can further be constrained by the query.
- The use of GRAPH with a variable changes dynamically the **active graph** for matching basic graph patterns within part of the query.
- **Outside the use of GRAPH**, the default graph is used to match graph patterns.

Example: Two FOAF Files

- **Named graph** `http://example.org/foaf/aliceFoaf`

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .

_:a foaf:knows _:b .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
_:b foaf:nick "Bobby" .
_:b rdfs:seeAlso <http://example.org/foaf/bobFoaf> .

<http://example.org/foaf/bobFoaf> rdf:type
   foaf:PersonalProfileDocument .
```

Example (cont'd)

- **Named graph** `http://example.org/foaf/bobFoaf`

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .


_:z foaf:mbox <mailto:bob@work.example> .
_:z rdfs:seeAlso <http://example.org/foaf/bobFoaf> .
_:z foaf:nick "Robert" .

<http://example.org/foaf/bobFoaf> rdf:type
   foaf:PersonalProfileDocument .
```

Queries

Query 1: Give me the IRIs of all the graphs where Bob has a nickname and the value of that nickname.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?src ?bobNick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE {
        GRAPH ?src
              { ?x foaf:mbox <mailto:bob@work.example> .
                ?x foaf:nick ?bobNick
              }
      }
```

Queries (cont'd)

- **Answer:**

src	**bobNick**
<http://example.org/foaf/aliceFoaf>	"Bobby"
<http://example.org/foaf/bobFoaf>	"Robert"

Queries (cont'd)

Query 2: Use Alice's FOAF file to find the personal profile document of everybody Alice knows. Use that document to find this person's e-mail and nickname.

Queries (cont'd)

```
PREFIX data: <http://example.org/foaf/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?mbox ?nick ?ppd
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE {
        GRAPH data:aliceFoaf
        { ?alice foaf:mbox <mailto:alice@work.example> ;
                 foaf:knows ?whom .
          ?whom foaf:mbox ?mbox ;
                 rdfs:seeAlso ?ppd .
          ?ppd a foaf:PersonalProfileDocument .
        } .
        GRAPH ?ppd
        { ?w foaf:mbox ?mbox ;
             foaf:nick ?nick
        }
      }
```

Queries (cont'd)

- **Answer:**

mbox	**nick**	**ppd**
<mailto:bob@work.example>	"Robert"	<http://example.org/foaf/bobFoaf>

RDF Datasets in Other Query Forms

- RDF datasets can also be used in other SPARQL query forms e.g., CONSTRUCT.

- **Example:** The following query extracts a graph from the target dataset based on provenance information in the default graph.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
CONSTRUCT { ?s ?p ?o }
WHERE {
       GRAPH ?g { ?s ?p ?o } .
       { ?g dc:publisher <http://www.w3.org/> } .
       { ?g dc:date ?date } .
       FILTER ( ?date > "2005-02-28T00:00:00Z"^^xsd:dateTime )
}
```

Semantics of Queries on RDF Datasets

- See the W3C formal semantics of SPARQL (http://www.w3.org/TR/rdf-sparql-query/#sparqlDefinition).
- Alternatively, see the papers
  - Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. *Semantics and Complexity of SPARQL*. Proc. of ISWC 2006.
  - Renzo Angles, Claudio Gutierrez. *The Expressive Power of SPARQL.* Proc. of ISWC 2008.

Presentation Outline

- Negation in SPARQL
- Other Useful Features of SPARQL
- Named Graphs
- **Querying RDFS information using SPARQL**
- SPARQL 1.1. features

Querying RDFS information with SPARQL

- SPARQL can be used to query RDFS information as well (we have the same query language for querying data and schema).

- You can do this by using the **RDFS reasoners** offered by various RDF stores to compute RDFS entailments. For example:
  - The ForwardChainingRDFSInferencer of Sesame (http://www.openrdf.org/doc/sesame2/api/org/openrdf/sail/inferencer/fc/ForwardChainingRDFSInferencer.html)
  - The RDFS reasoner of Jena2 (http://jena.sourceforge.net/inference/index.html).

RDFS Reasoners

- The available RDFS reasoners implement the RDFS entailments as given by the W3C Recommendation “RDF Semantics” (http://www.w3.org/TR/rdf-mt/) which we will discuss in detail in a future lecture.

- Most reasoners work in a **forward chaining** fashion: when they are used, they infer all triples that are entailed by the RDFS entailments rules and the given RDF/RDFS graph. The resulting set of triples is the one queried by SPARQL in our examples.

- Some reasoners can be **configured** regarding what RDFS entailment rules to apply (see the manual of the reasoner for details).

Example: A Class Hierarchy

Example

```
@prefix  ex: <http://example.org/schemas/vehicles#> .
@prefix  rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .

ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .
```

Queries (cont'd)

Query: Find the subclasses of MotorVehicle

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x  rdfs:subClassOf  ns:MotorVehicle }
```

Answer:

x
<http://example.org/schemas/vehicles#Truck>
<http://example.org/schemas/vehicles#Van>
<http://example.org/schemas/vehicles#PassengerVehicle>
<http://example.org/schemas/vehicles#MotorVehicle>
<http://example.org/schemas/vehicles#MiniVan>

Queries (cont'd)

Query: Find the subclasses of MiniVan

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x  rdfs:subClassOf  ex:MiniVan }
```

Answer:

x
<http://example.org/schemas/vehicles#MiniVan>

Note: Remember that `rdfs:subClassOf` is reflexive.

Queries (cont'd)

Query: Find the superclasses of MiniVan

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?y
WHERE { ex:MiniVan rdfs:subClassOf ?y }
```

Answer:

y
<http://example.org/schemas/vehicles#PassengerVehicle>
<http://example.org/schemas/vehicles#Van>
<http://example.org/schemas/vehicles#MiniVan>
<http://example.org/schemas/vehicles#MotorVehicle>
<http://www.w3.org/2000/01/rdf-schema#Resource>

Note: the answer contains the predefined class `rdf:Resource`.

Queries (cont'd)

Query: Find the superclasses of MiniVan that are not predefined RDFS classes.

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?y
WHERE { ex:MiniVan rdfs:subClassOf ?y .
        FILTER(?y != rdf:Resource)
      }
```

Answer:

y
<http://example.org/schemas/vehicles#PassengerVehicle>
<http://example.org/schemas/vehicles#Van>
<http://example.org/schemas/vehicles#MiniVan>
<http://example.org/schemas/vehicles#MotorVehicle>

Note: use FILTER to remove `rdf:Resource` from the answer set.

Queries (cont'd)

Query: Find all the classes (i.e., the instances of rdfs:Class)

```
PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x rdf:type rdfs:Class }
```

Answer

x
<http://www.w3.org/2000/01/rdf-schema#Resource>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://www.w3.org/2000/01/rdf-schema#Literal>
<http://www.w3.org/2000/01/rdf-schema#Class>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#List>
<http://example.org/schemas/vehicles#MiniVan>
<http://example.org/schemas/vehicles#Truck>
<http://example.org/schemas/vehicles#Van>
<http://example.org/schemas/vehicles#PassengerVehicle>
<http://example.org/schemas/vehicles#MotorVehicle>
<http://www.w3.org/2000/01/rdf-schema#Container>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt>
<http://www.w3.org/2000/01/rdf-schema#ContainerMembershipProperty>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag>
<http://www.w3.org/2000/01/rdf-schema#Datatype>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral>

Queries (cont'd)

Query: Find all the subclasses of rdfs:Class.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x rdfs:subClassOf rdfs:Class }
```

Answer:

x
<http://www.w3.org/2000/01/rdf-schema#Class>
<http://www.w3.org/2000/01/rdf-schema#Datatype>

Queries (cont'd)

Query: Find all the subclasses of rdfs:Resource.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x rdfs:subClassOf rdfs:Resource }
```

Answer

x
<http://example.org/schemas/vehicles#MotorVehicle>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#List>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
<http://www.w3.org/2000/01/rdf-schema#Literal>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement>
<http://www.w3.org/2000/01/rdf-schema#Class>
<http://www.w3.org/2000/01/rdf-schema#Resource>
<http://example.org/schemas/vehicles#Van>
<http://example.org/schemas/vehicles#MiniVan>
<http://example.org/schemas/vehicles#PassengerVehicle>
<http://example.org/schemas/vehicles#Truck>

Queries (cont'd)

Query: Find all the resources (i.e., instances of rdfs:Resource).

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?x
WHERE { ?x rdf:type rdfs:Resource }
```

Answer: ?

Try it!

Queries (cont'd)

Query: Find all the properties (i.e., instances of rdf:Property).

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
SELECT ?x
WHERE { ?x rdf:type rdf:Property }
```

Answer

x
<http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#object>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#subject>
<http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
<http://www.w3.org/2000/01/rdf-schema#seeAlso>
<http://www.w3.org/2000/01/rdf-schema#comment>
<http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2000/01/rdf-schema#domain>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.w3.org/2000/01/rdf-schema#label>

Example – Defining Instances

```
@prefix  exthings: <http://example.org/instances#> .
@prefix  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#> .

exthings:mv1 rdf:type ex:MiniVan .

exthings:tr1 rdf:type ex:Truck .

exthings:pv1 rdf:type ex:PassengerVehicle .

exthings:v1 rdf:type ex:Van .
```

Queries (cont'd)

Query: Find all the motor vehicles

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
SELECT ?x
WHERE { ?x rdf:type ex:MotorVehicle }
```

Answer:

x
<http://example.org/instances#mv1>
<http://example.org/instances#pv1>
<http://example.org/instances#v1>
<http://example.org/instances#tr1>

Queries (cont'd)

Query: Find all the vans and passenger vehicles

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x
WHERE { { ?x rdf:type ex:Van }
        UNION
        { ?x rdf:type ex:PassengerVehicle }
      }
```

Answer:

x
<http://example.org/instances#v1>
<http://example.org/instances#mv1>
<http://example.org/instances#pv1>
<http://example.org/instances#mv1>

Note: the duplicate `mv1` is due to the two arguments of UNION.

Example – Defining Properties

```
@prefix  ex: <http://example.org/schemas/vehicles#> .
@prefix  rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:registeredTo a rdf:Property;
    rdfs:domain ex:MotorVehicle;
    rdfs:range ex:Person.

ex:Person a rdfs:Class.

ex:rearSeatLegRoom a rdf:Property;
    rdfs:domain ex:PassengerVehicle;
    rdfs:range xsd:integer.

ex:driver rdf:type rdf:Property .
    rdfs:domain ex:MotorVehicle;
    rdfs:range ex:Person.

ex:primaryDriver rdf:type rdf:Property .
ex:primaryDriver rdfs:subPropertyOf ex:driver .
```

Example – Inferences

```
@prefix  ex: <http://example.org/schemas/vehicles#> .
@prefix  exthings: <http://example.org/instances#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

exthings:johnSmithsCar a ex:PassengerVehicle;
     ex:registeredTo <http://www.example.org/staffid/85740>;
     ex:rearSeatLegRoom "127"^^xsd:integer;
     ex:primaryDriver <http://www.example.org/staffid/85740>.
```

Inferences:

- Resource `http://www.example.org/staffid/85740` has not been defined to be an instance of class `ex:Person;` this fact is inferred by RDFS due to the **range definition** of property `ex:registeredTo` (and `ex:primaryDriver`).

- The same resource has not been defined as a driver of resource `exthings:johnSmithsCar;` this is also inferred by RDFS because `ex:driver` is a **superproperty** of `ex:primaryDriver`.

Queries (cont'd)

Query: Find all persons

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
SELECT ?x
WHERE { ?x rdf:type ex:Person}
```

Answer:

x
<http://www.example.org/staffid/85740>

Queries (cont'd)

Query: Find all drivers and the vehicles they drive.

```
PREFIX ex:   <http://example.org/schemas/vehicles#>
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
SELECT ?d ?v
WHERE { ?v ex:driver ?d }
```

Answer:

d	**v**
<http://www.example.org/staffid/85740>	<http://example.org/instances#johnSmithsCar>

Queries (cont'd)

Query: Find all properties (i.e., instances of rdf:Property).

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-
   ns#>
SELECT ?x
WHERE { ?x rdf:type rdf:Property }
```

Answer

x
<http://www.w3.org/1999/02/22-rdf-syntax-ns#first>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#rest>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#object>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#subject>
<http://example.org/schemas/vehicles#primaryDriver>
<http://example.org/schemas/vehicles#driver>
<http://example.org/schemas/vehicles#rearSeatLegRoom>
<http://example.org/schemas/vehicles#registeredTo>
<http://www.w3.org/2000/01/rdf-schema#isDefinedBy>
<http://www.w3.org/2000/01/rdf-schema#seeAlso>
<http://www.w3.org/2000/01/rdf-schema#comment>
<http://www.w3.org/2000/01/rdf-schema#range>
<http://www.w3.org/2000/01/rdf-schema#domain>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/2000/01/rdf-schema#subPropertyOf>
<http://www.w3.org/2000/01/rdf-schema#subClassOf>
<http://www.w3.org/2000/01/rdf-schema#label>

The Expressive Power of SPARQL

- The ISWC2008 paper by Angles and Gutierrez proves that the languages:
  - SPARQL as defined by the W3C - **$\mathbf{SPARQL_{WG}}$**
  - SPARQL with compositional semantics as studied in the ISWC 2006 paper by Perez, Arenas and Gutierrez - **$\mathbf{SPARQL_{C}}$**

  are **equivalent** and they are also equivalent with:
  - Relational algebra
  - Non-recursive datalog with negation

- This is an excellent result since it tell us that lots of techniques from relational algebra query evaluation are essentially available to use for SPARQL (how?).

Presentation Outline

- Negation in SPARQL
- Other Useful Features of SPARQL
- Named Graphs
- Querying RDFS information using SPARQL
- **SPARQL 1.1. features**

SPARQL 1.1

- The newest version of SPARQL is SPARQL 1.1 with support for:
  - **New query features**:
    - Aggregate functions
    - Subqueries
    - Negation
    - Expressions in the SELECT clause
    - Property Paths
    - Assignment
    - A short form for CONSTRUCT
    - An expanded set of functions and operators
  - **Updates**
  - **Federated queries**
  - …

- See the web page of the SPARQL Working Group for more information: http://www.w3.org/2009/sparql/wiki/Main_Page

Aggregates

- **Aggregate functions** can be used to do computations over groups of solutions that satisfy certain graph patterns. By default a solution set consists of a single group, containing all solutions.

- Grouping is specified using the `GROUP BY` clause.

- The `HAVING` clause can also be used to constrain grouped solutions in the same way `FILTER` constrains ungrouped ones.

- The following aggregate functions are allowed: `COUNT`, `SUM`, `MIN`, `MAX`, `AVG`, `GROUP_CONCAT`, and `SAMPLE`.

Example: Aggregates

- **Data:**

```
@prefix : <http://books.example/> .

:org1 :affiliates :auth1, :auth2 .
:auth1 :writesBook :book1, :book2 .
:book1 :price 9 .
:book2 :price 5 .
:auth2 :writesBook :book3 .
:book3 :price 7 .

:org2 :affiliates :auth3 .
:auth3 :writesBook :book4 .
:book4 :price 7 .
```

Example (cont'd)

- **Query:** Find the total price of books written by authors affiliated with some organization. Output organization id and total price only if the total price is greater than 10.

```
PREFIX : <http://books.example/>
SELECT (?org SUM(?lprice) AS ?totalPrice)
WHERE { ?org :affiliates ?auth .
        ?auth :writesBook ?book .
        ?book :price ?lprice . }
GROUP BY ?org
HAVING (SUM(?lprice) > 10)
```

- **Result:**

org	totalPrice
<http://books.example/org1>	21

Subqueries

- Subqueries are a way to **embed SPARQL queries inside other queries** to allow the expression of requests that are not possible otherwise.
- Subqueries are useful when combining limits and aggregates with other constructs.
- Subqueries are evaluated first and then the outer query is applied to their results.
- Only variables projected out of the subquery (i.e., appearing in its SELECT clause) will be visible to the outer query.

Example: Subqueries

- **Data:**

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix : <http://sales.com/> .

:sale1 a :Sale ; :company :c1 ; :amount 7500^^xsd:integer  ; :year "2011" .

:sale2 a :Sale ; :company :c1 ; :amount 17000^^xsd:integer ; :year "2011" .

:sale3 a :Sale ; :company :c1 ; :amount 5500^^xsd:integer ; :year "2012" .

:sale4 a :Sale ; :company :c1 ; :amount 7000^^xsd:integer ; :year "2012" .

:sale5 a :Sale ; :company :c2 ; :amount 3000^^xsd:integer ; :year "2011" .

:sale6 a :Sale ; :company :c2 ; :amount 4000^^xsd:integer ; :year "2011" .

:sale7 a :Sale ; :company :c2 ; :amount 5000^^xsd:integer ; :year "2012" .

:sale8 a :Sale ; :company :c2 ; :amount 6000^^xsd:integer ; :year "2012" .
```

Example (cont'd)

- **Query:** Find companies that increased their sales from 2011 to 2012 and the amount of increase.

```
PREFIX : <http://sales.com/>
SELECT ?c ((?total2012 - ?total2011) AS ?increase)
WHERE {
    { SELECT ?c (SUM(?m) AS ?total2012)
      WHERE { ?s a :Sale ; :company ?c ;
                  :amount ?m ; :year: "2012" . }
      GROUP BY ?c
    } .
    { SELECT ?c (SUM(?m) AS ?total2011)
      WHERE { ?s a :Sale ; :company ?c ;
                  :amount ?m ; :year: "2011" . }
      GROUP BY ?c
    } .
      FILTER (?total2012 > ?total2011)
      }
```

Example (cont'd)

- **Result:**

c	**increase**
<http://sales.com/c2>	"4000"^^<http://www.w3.org/2001/XMLSchema#integer>

Negation

- In SPARQL 1.1 we have two ways to express negation:
  - The operator NOT EXISTS in FILTER expressions (testing for the non-existence of a pattern)
  - The algebraic operator MINUS

Example: NOT EXISTS

- **Data:**

```
@prefix : <http://example/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-
  syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:alice rdf:type foaf:Person .
:alice foaf:name "Alice" .
:bob rdf:type foaf:Person .
```

Example (cont'd)

- **Query:** Find persons for whom we have no name in the database.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE { ?person rdf:type foaf:Person .
        FILTER NOT EXISTS { ?person foaf:name ?name } }
```

- **Result:**

person
<http://example/bob>

- This is what we expressed earlier with OPTIONAL, FILTER and `!bound`.

Example: MINUS

- **Data:**

```
@prefix : <http://example/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

:alice foaf:givenName "Alice" ;
       foaf:familyName "Smith" .
:bob foaf:givenName "Bob" ;
     foaf:familyName "Jones" .
:carol foaf:givenName "Carol" ;
       foaf:familyName "Smith" .
```

Example (cont'd)

- **Query:** Find all persons that do not have given name "Bob".

```
PREFIX : <http://example/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?s
WHERE { ?s ?p ?o .
        MINUS { ?s foaf:givenName "Bob" . } }
```

- **Result:**

s
<http://example/carol>
<http://example/alice>

MINUS

- The operator MINUS is the same as the **difference** operation we defined earlier in our algebraic semantics of SPARQL.

- M_1 MINUS M_2 returns all the mappings in M_1 that cannot be extended by any mapping in M_2 (i.e., are incompatible with all mappings in M_2).

- MINUS and NOT EXISTS do not always return the same result if they are not applied with care.

Example: Expressions in the SELECT clause

- **Data:**

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .
:book1 ns:discount 0.2 .

:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
:book2 ns:discount 0.25 .
```

Example (cont'd)

- **Query:** Find all book titles and their discounted price.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title (?p*(1-?discount) AS ?price)
WHERE { ?x ns:price ?p .
        ?x dc:title ?title .
        ?x ns:discount ?discount }
```

- **Result:**

title	**price**
"The Semantic Web"	17.25
"SPARQL Tutorial"	33.6

Example (cont'd)

- **Query:** Find all book titles, their full price and their discounted price.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title (?p AS ?fullPrice) (?fullPrice*(1-?discount) AS
?customerPrice)
WHERE { ?x ns:price ?p .
        ?x dc:title ?title .
        ?x ns:discount ?discount }
```

- **Result:**

title	**fullPrice**	**customerPrice**
"The Semantic Web"	23	17.25
"SPARQL Tutorial"	42	33.6

Property Paths

- SPARQL 1.1 allows us to specify **property paths** in the place of a predicate in a triple pattern.
- Property paths use **regular expressions** to enable us to write sophisticated queries that traverse an RDF graph.
- Property paths allow for the more concise expression of some queries plus the ability to refer to paths of arbitrary length.
- See http://www.w3.org/TR/2012/WD-sparql11-query-20120724/#propertypaths for the exact property path operators and syntax.

Example

- **Query:** Find the name of any people that Alice knows.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE { ?x foaf:mbox <mailto:alice@example> .
        ?x foaf:knows/foaf:name ?name . }
```

- The / path operator denotes **sequence**.

Example

- **Query:** Find the name of people that Alice knows that are 2 "`foaf:knows`" links away.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE { ?x foaf:mbox <mailto:alice@example> .
        ?x foaf:knows/foaf:knows/foaf:name ?name . }
```

Example (cont'd)

- The same query can be written equivalently **without property path expressions** as follows:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE { ?x foaf:mbox <mailto:alice@example> .
        ?x foaf:knows ?a1 .
        ?a1 foaf:knows ?a2 .
        ?a2 foaf:name ?name . }
```

Example

- **Query:** Find all the people `:x` connects to via the `foaf:knows` relationship (using a path with an arbitrary length).

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX : <http://example/>
SELECT ?person
WHERE { :x foaf:knows+ ?person }
```

- The `+` operator denotes one or more ocurrences of `foaf:knows`.

Example

- **Query:** Find all types, including supertypes ,of each resource in the dataset.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
SELECT ?x ?type
WHERE { ?x rdf:type/rdfs:subClassOf* ?type }
```

- The `*` operator denotes zero or more ocurrences of `rdfs:subClassOf`.

Readings

- Chapter 7 of the book “Foundations of Semantic Web Technologies”.
- The original 2008 SPARQL standard “SPARQL Query Language for RDF” from http://www.w3.org/TR/rdf-sparql-query/ .
- The SPARQL 1.1 standard (2013) available at http://www.w3.org/TR/sparql11-query/
- The SPARQL tutorial given at ESWC 2007 for the semantics of SPARQL: http://axel.deri.ie/%7Eaxepol/sparqltutorial/ .
- The more recent SPARQL tutorial by Cambridge Semantics: http://www.cambridgesemantics.com/el/semantic-university/sparql-by-example
- The Sesame documentation regarding RDFS inferencing available at http://www.openrdf.org/doc/sesame2/users/ .