SPARQL Protocol and RDF Query Language (SPARQL)

Semantic Web (CSC751)

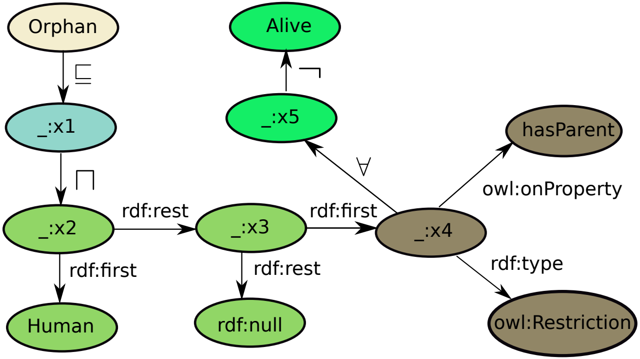
Ubbo Visser

Department of Computer Science
University of Miami

October 31, 2023

Outline

1. Announcements
2. In retrospect
3. Query types
4. Basics

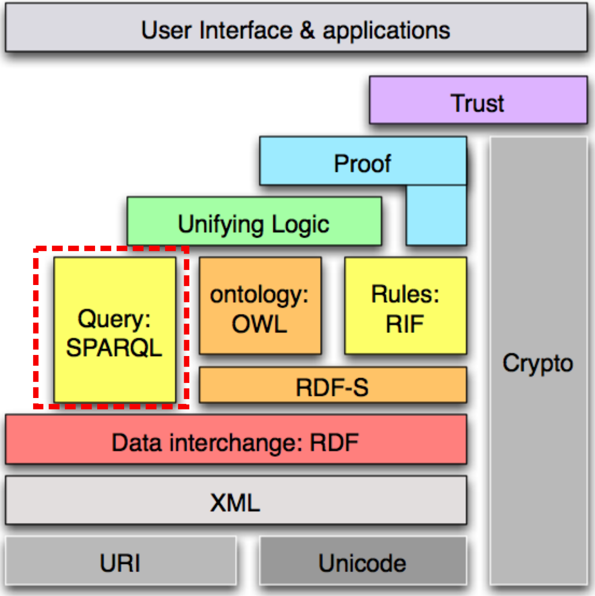


Reading

- 7.1.1-7.1.8 [HKR09]

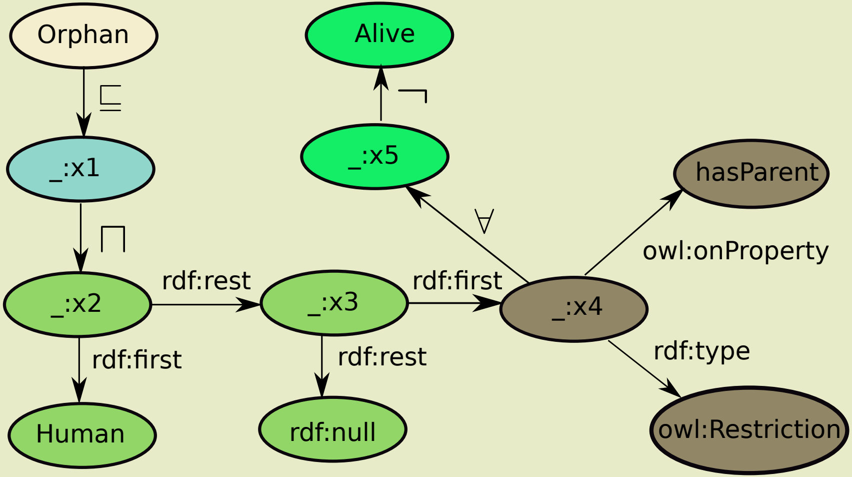
Acknowledgement

- Most of the examples in this lecture slides are borrowed from SPARQL Query Language for RDF


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

OWL DL

$$Orphan \sqsubseteq Human \sqcap \forall hasParent.\neg Alive$$



Query types

1. Retrieve instances.

```
SELECT ?x WHERE {
  ?x rdf:type family:Person .
}
```

2. Retrieve subclasses.

```
SELECT ?x WHERE {
  ?x rdfs:subClassOf family:Person .
}
```

3. Retrieve subclasses, and their instances.

```
SELECT ?x ?y WHERE {
  ?x rdf:type ?y .
  ?y rdfs:subClassOf family:Person .
}
```

- We have written a simple framework to query the knowledge base using Jena and Pellet API. It is available in the class web site.

```
public interface OwlHelper {
  InfModel loadInfModel(File kb);
  InfModel loadInfModel(File tBox, File aBox);
  void startReasoner(InfModel model);
  void execQuery(String query, Model model, ResultSetCallback callback);
}

public interface ResultSetCallback {
    void run(ResultSet resultSet, Query query);
}
```

- You can use this framework in your code as follows:

```
  OwlHelper helper = OwlHelperFactory.createDefaultOwlHelper();
  InfModel model = helper.loadInfModel(kbFile);
  helper.startReasoner(model);
  ...
  String query = ...;
  helper.execQuery(query, model, new ResultSetFormatterCallback());
```

SPARQL

- W3C recommendation for querying RDF and RDFS.
- We can use SPARQL to a certain extent to query OWL 2 DL knowledge bases. But the preferred way is *conjunctive queries*.

Graph patterns

- SPARQL is based on matching *graph patterns* w.r.t. RDF, RDFS (supported features), or OWL (supported features) graphs.
- A *graph pattern* is similar to *triple pattern*, but with the option of **variables** in subject, predicate or object. e.g.,

```
<http://family.org/family.owl#daughter>
<http://family.org/family.owl#hasParent> ?parent .
```

- `?parent` is a **variable**. This variable could also be written as `$parent`.

Basic graph patterns

- A *basic graph pattern* (BGP) is a set of *triple patterns* written as a sequence of triple patterns separated by a period if necessary.
- Therefore, BGP is a *conjunction of triple patterns*. e.g.,

```
?x <http://family.org/family.owl#hasParent> ?parent .
?x <http://family.org/family.owl#hasUncle> ?uncle
```

- There is no keyword for conjunction in SPARQL.

Group graph patterns

- A group graph pattern is a set of graph patterns delimited with braces. e.g.,

```
{
  { ?x  <http://family.org/family.owl#hasParent> ?y . }
  { ?x  <http://family.org/family.owl#hasUncle>  ?z . }
  {   }
}
```

- `{   }` is the empty group graph pattern.
- Group graph patterns are used with other constructors, which we will see in few slides.

Major query parts

- `PREFIX` : declares the namespace prefix,
- `SELECT` : determines the general result format, and
- `WHERE` : actual query is initiated with group graph patterns.
- The result of a query is a set of *bindings* for the variables appearing in the `SELECT` clause. These binding are shown in tabular format.
- `SELECT` and `WHERE` clauses are like in SQL. But keep in mind that SPARQL and SQL are very different languages.

```
PREFIX xsd:    <http://www.w3.org/2001/xmlschema#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
PREFIX family: <http://family.org/family.owl#>
SELECT ?x
WHERE
{
  family:daughter  family:hasParent ?x .
}
```

```
 -----------------
| x             |
=================
| family:mother |
| family:father |
-----------------
```

```
PREFIX xsd:    <http://www.w3.org/2001/xmlschema#>
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:    <http://www.w3.org/2002/07/owl#>
PREFIX family: <http://family.org/family.owl#>
SELECT ?x ?y ?z
{
  ?x  family:hasParent ?y .
  ?x  family:hasUncle ?z
}
```

```
--------------------------------------------------
| x               | y             | z            |
==================================================
| family:daughter | family:mother | family:uncle |
| family:daughter | family:father | family:uncle |
| family:son      | family:mother | family:uncle |
| family:son      | family:father | family:uncle |
--------------------------------------------------
```

Queries with literals

- We have careful when matching literals. e.g.,

```
  SELECT ?x WHERE { ?x ?p "Ubbo" .}
```

  and

```
  SELECT ?x WHERE { ?x ?p "Ubbo"@en .}
```

  have different results.

- xsd data types:

```
SELECT ?x WHERE { ?x ?hasAge "38"^^xsd:nonNegativeInteger .}
-----------------
| x             |
=================
| family:father |
-----------------
or
SELECT ?x WHERE { ?x ?hasAge 38.}
```

Queries with literals - continued

```
"chat"
’chat’@fr with language tag "fr"
"xyz"^^<http://example.org/ns/userDatatype>
"abc"^^appNS:appDataType
’’’The librarian said, "Perhaps you would enjoy ’War and Peace’."’’’
1, which is the same as "1"^^xsd:integer
1.3, which is the same as "1.3"^^xsd:decimal
1.300, which is the same as "1.300"^^xsd:decimal
1.0e6, which is the same as "1.0e6"^^xsd:double
true, which is the same as "true"^^xsd:boolean
false, which is the same as "false"^^xsd:boolean
```

Blank nodes in query results

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:b foaf:name "Bob" .

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

---------------------------------
| x             | name          |
=================================
| _:c           | "Alice"       |
---------------------------------
| _:d           | "Bob"         |
---------------------------------
```

Blank nodes in graph patterns

- Blank nodes assert the existence of a corresponding element in the input graph, but they do not provide any information about the identity of this element.
- Blank nodes cannot appear in a `SELECT` clause.
- The scope of blank node is the BGP in which it appears. A blank node which appears more than once in the same BGP stands for the same term.

Constraints on variables

- FILTER restricts variable bindings to those for which the filter expression evaluates to *true*.

```
@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 50 .
:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE {
 ?x ns:price ?price .
 FILTER (?price < 30.5)
 ?x dc:title ?title .
}

=> "The Semantic Web"   23
```

Constraints on variables

- Regular expression filter:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE
{
  ?x dc:title ?title
  FILTER regex(?title, "^SPARQL")
}

=> SPARQL Tutorial
```

SPARQL Tutorial

- Group graph patters are used to restrict the scope of the FILTER.
- FILTER is a restriction on solutions over the whole group in which it appears.
- One can have multiple FILTER conditions in a group graph pattern. The result is equivalent to a single filter with conjuncted filter conditions.
- FILTER can have very complex boolean conditions.

These graph patterns have same set of solutions

```
{
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox .
  FILTER regex(?name, "Smith")
}

{
  FILTER regex(?name, "Smith")
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox .
}

{
  ?x foaf:name ?name .
  FILTER regex(?name, "Smith")
  ?x foaf:mbox ?mbox .
}
```

OPTIONAL graph patterns

- With OWA, the complete structures cannot be assumed in all RDF graphs (this is of the ABox).
- Therefore, we need a way to extract the available information, even though some part of the query pattern does not match.
- OPTIONAL provides this facility. If the graph pattern does not match, it does not create bindings, but does not eliminate the solution as well.

KB

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .
_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
```

OPTIONAL example

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

------------------------------------------------
| name            | mbox                       |
================================================
| "Alice"         | <mailto:alice@example.com> |
------------------------------------------------
| "Alice"         | <mailto:alice@work.example> |
------------------------------------------------
| "Bob"           |                            |
------------------------------------------------
```

OPTIONAL properties

- Normally, we start with a graph pattern P1 and then apply OPTIONAL to another graph pattern P2 that follows it.

```
  P1 OPTIONAL { P2 }
```

- OPTIONAL is a binary operator.
- OPTIONAL is left-associative.

```
P1 OPTIONAL { P2 } OPTIONAL { P3 }
<=>
{ P1 OPTIONAL { P2 } } OPTIONAL { P3 }
```

```
{ OPTIONAL { P } }
<=>
{ { } OPTIONAL { P } }
```

- OPTIONAL has higher precedence that conjunction.

FILTER in OPTIONAL

- The group graph pattern following the OPTIONAL can be as complex as possible.

KB

```
@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" .
:book2 dc:title "A New SPARQL Tutorial" .
:book2 ns:price 42 .
:book3 dc:title "The Semantic Web" .
:book3 ns:price 23 .
```

FILTER in OPTIONAL example

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE {
  ?x dc:title ?title .
  OPTIONAL {
    ?x ns:price ?price .
    FILTER (?price < 30)
  }
}

------------------------------------------------
| title                    | price              |
================================================
| "SPARQL Tutorial"        |                    |
------------------------------------------------
| "A New SPARQL Tutorial" |                    |
------------------------------------------------
| "The Semantic Web"       | 23                 |
------------------------------------------------
```

Multiple OPTIONAL

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

PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox . }
        OPTIONAL { ?x foaf:homepage ?hpage . }
      }
------------------------------------------------------------------------------
| name        | mbox               | hpage                                   |
==============================================================================
| "Alice"     |                           | <http://work.example.org/alice/> |
------------------------------------------------------------------------------
| "Bob"       | <mailto:bob@work.example> |                                  |
------------------------------------------------------------------------------
```

Example

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

ex:book1 dc:creator ex:smith .
ex:book1 dc:title "Semantic Web" .
ex:book1 ns:price 30 .

ex:book2 dc:creator ex:jones .
ex:book2 dc:title "SPARQL" .

ex:book3 dc:creator ex:doyle.
ex:book3 ns:price 34 .

ex:book4 dc:title "RDF" .
ex:book4 ns:price 50 .
```

Queries

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
***********************************************
SELECT ?book ?title
WHERE { ?book dc:creator ?author .
        OPTIONAL { ?book dc:title ?title .}
        { ?book ns:price ?price .}
      }
***********************************************
SELECT ?book ?title
WHERE { { ?book dc:creator ?author .
        OPTIONAL { ?book dc:title ?title .} }
        { ?book ns:price ?price .}
      }
***********************************************
SELECT ?book ?title
WHERE { ?book dc:creator ?author .
        OPTIONAL { { ?book dc:title ?title .}
                   { ?book ns:price ?price .} }
      }
***********************************************
```

UNION

- UNION provides the facility to form *disjunction of graph patterns*, such that one of several graph patterns may match. All the alternative matching patterns are returned.

KB

```
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .

_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .

_:b dc11:title "SPARQL Protocol Tutorial" .
_:b dc11:creator "Bob" .

_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .
```

UNION example

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title }
        UNION
        { ?book dc11:title ?title }
      }

------------------------------------
| title                            |
====================================
| "SPARQL Protocol Tutorial"       |
------------------------------------
| "SPARQL"                         |
------------------------------------
| "SPARQL (updated)"               |
------------------------------------
| "SPARQL Query Language Tutorial" |
------------------------------------
```

UNION example

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?author ?title
WHERE { { ?book dc10:title ?title .
          ?book dc10:creator ?author . }
        UNION
        { ?book dc11:title ?title .
          ?book dc11:creator ?author . }
      }

----------------------------------------------------
| author          | title                          |
====================================================
| "Alice"         | "SPARQL Query Language Tutorial" |
----------------------------------------------------
| "Bob"           | "SPARQL Protocol Tutorial"     |
----------------------------------------------------
```

Semantic of UNION

- UNION is a binary operator.
- Group graph patterns are evaluated independently and combine the results using *set theoretic union*.
- We have to decide whether to use same variable in each alternative, as this decision provides different results.

UNION example

```
SELECT ?x ?y
WHERE { {?book dc10:title ?x} UNION {?book dc11:title ?y} }
----------------------------------------------
| x                  | y                     |
==============================================
|                    | "SPARQL (Updated)"    |
----------------------------------------------
|                    | "SPARQL Protocol ..."|
----------------------------------------------
| "SPARQL"           |                       |
----------------------------------------------
| "SPARQL Query ..." |                       |
----------------------------------------------
```

Properties of UNION

- UNION is left-associative.
- UNION and OPTIONAL have same precedence.
- UNION has higher precedence than conjunction.
- Commutative

```
  P UNION Q <=> Q UNION P
```

- Associative

```
{P UNION Q} UNION R <=> P UNION {Q UNION R}
```

OPTIONAL, UNION examples

```
{ {s1 p1 o1} UNION {s2 p2 o2}
             OPTIONAL {s3 p3 o3}
}
<=>
{ { {s1 p1 o1} UNION {s2 p2 o2}
  } OPTIONAL {s3 p3 o3}
}
```

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o1} UNION {s3 p3 o3}
             OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
<=>
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o1}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```

UNION and conjunction

```
{ {s1 p1 o1} UNION {s2 p2 o1}
  {s3 p3 o3}
}
<=>
{ { {s1 p1 o1} UNION {s2 p2 o1}
  }
  {s3 p3 o3}
}
```

KB Queries with data values

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

ex:book1 dc:creator ex:smith .
ex:book1 dc:title "Semantic Web" .

ex:book2 dc:creator ex:jones .
ex:book2 dc:title "SPARQL" .
ex:book2 ns:price 30 .

ex:book3 dc:creator ex:jones.
ex:book3 dc:title "RDF" .
ex:book3 ns:price 35 .
```

Example

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title ?price
WHERE
{
  { ?book dc:creator ex:smith . ?book dc:title ?title . }
  UNION
  { ?book dc:creator ex:jones .}
  { ?book ns:price ?price . }
}

-------------------------------------------------
| book                        | title   | price    |
=================================================
| <http://example.org/book3> |         | 35       |
-------------------------------------------------
| <http://example.org/book2> |         | 30       |
-------------------------------------------------
```

Example

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title ?price
WHERE
{
  { ?book dc:creator ex:smith . ?book dc:title ?title . }
  UNION
  { ?book dc:creator ex:jones . ?book ns:price ?price . }
}

---------------------------------------------------------
| book                        | title           | price    |
=========================================================
| <http://example.org/book1> | "Semantic Web"|          |
---------------------------------------------------------
| <http://example.org/book3> |                 | 35       |
---------------------------------------------------------
| <http://example.org/book2> |                 | 30       |
---------------------------------------------------------
```

More about FILTER and special operators

- FILTER supports $=, >, <, \geq, \leq$, and $!=$ operators.
- Each operator is defined for all datatype that SPARQL supports. e.g., `xsd:dataTime`
- All literals that have different datatypes are not compatible with prior operators, but $=$ and $!=$.
- But, they produce an error if unknown datatypes are given.
- Multiple filter conditions are combined with && (logical *and*), || (logical *or*) and ! (logical *not*).
- Conjunction: can be expressed with multiple FILTER within one graph pattern.
- Disjunction: a graph pattern could be split into multiple alternative patterns that use equal conditions with one of filter part.
- Supports numerical operators, +, -, *, and /, only if the variable are bounded in a meaningful way.

Unary operators

`BOUND(A)`	true if A is a bounded variable
`isURI(A)`	true if A is a URI
`isBLANK(A)`	true if A is a blank node
`isLITERAL(A)`	true if A is a RDF literal
`STR(A)`	maps RDF literals or URIs to the corresponding lexical representation of type `xsd:string`
`LANG(A)`	returns language code of an RDF literal as `xsd:string`, or an empty string if no such setting is specified
`DATATYPE(A)`	returns the URI of an RDF literal datatype of the value "`xsd:string`" for untyped literals without language setting; not applicable to literals with language setting
`sameTERM(A,B)`	true if A and B are the same RDF terms (direct term comparison)
`langMATCHES(A,B)`	true if the literal A is a language tag that matches the pattern B
`REGEX(A,B)`	true if the regular expression B can be matched to the string A

Example

```
PREFIX ex: <http://example.org/>
SELECT ?book
WHERE
{
  { ?book ex:isPublishedBy <http://crc-press.com/uri> . }
  OPTIONAL { ?book ex:author ?author .}
  FILTER( DATATYPE(?author) = <http://www.w3.org/2001/XMLSchema#string>)
}

PREFIX ex: <http://example.org/>
SELECT ?book
WHERE
{
  ?book ex:title ?title .
  FILTER( REGEX(?title, "^Foundations of") )
}
```

Query forms

- Tabular representation is useful for processing results sequentially.
- If the structure and mutual relations of objects in the results set are more important, RDF representation of the results is more appropriate.
- `CONSTRUCT` returns an RDF graph specified by a graph template.
- `ASK` tests whether or not a graph pattern has a solution. This returns whether or not a solution exists.

Example CONSTRUCT

```
@prefix ex: <http://example.org/> .

ex:alice ex:email  "alice@example.org" .
ex:alice ex:email  "a.miller@example.org" .
ex:alice ex:phone  "123456789" .
ex:alice ex:phone  "987654321" .

PREFIX ex: <http://example.org/>
CONSTRUCT  {
 _id1 ex:email ?email .
 _id1 ex:phone ?phone .
 _id1 ex:person ?person . }
WHERE
{
  ?person ex:title ?email .  ?person ex:phone ?phone.
}

_y ex:email "alice@example.org";
   ex:phone "123456789"; ex:person ex:alice .
...
```

Example ASK

```
PREFIX ex: <http://example.org/>

ASK
{
  ?person ex:title ?email .
  ?person ex:phone ?phone.
}

=> TRUE
```

Modifiers

- To narrow down the result set.
- Modifiers controls the details regarding the form and size of result lists.
- Most constructs affects only results obtained with `SELECT`.

`ORDER BY`	sort in ascending order based on the meaningful bounded variable.
`DESC`	sort by descending order
`ASC`	sort by ascending order
`LIMIT`	maximum results
`OFFSET`	staring position for piecewise retrieval of results
`DISTINCT`	remove repetitions from result set

Order of application

- All the parameters are allowed to be combined. Therefore, SPARQL defines the following processing steps:
  - Sort results based on `ORDER BY`.
  - Remove non selected variables from the result set (*projection*).
  - Remove duplicate results.
  - Remove the number of initial results as specified by `OFFSET`.
  - Remove all results after the number specified by `LIMIT`.
- LARQ: combination of ARQ and Lucene. This is a specific example.

Examples

```
PREFIX ex: <http://example.org/>
SELECT ?book ?price
WHERE
{
  ?book ex:price ?price .
}
ORDER BY ?price
************************************
SELECT ?book ?price
WHERE
{
  ?book ex:price ?price .
}
ORDER BY ASC(?price)
************************************
SELECT *
WHERE
{
  ?s ?p ?o .
}
ORDER BY ?s LIMIT 5 OFFSET 25
```

The Manchester OWL

DL Query

- Searching in a classified ontology using Manchester OWL syntax.
- It is based on OWL abstract syntax and DL style syntax.
- Supports `some`, `only`, `value`, `min`, `exactly`, `max`, `and`, `or`, and `not`.
- Supports data values and datatypes with XSD facets.
- Lets see an example based on photography ontology (OWL 2).

Example[1]

- Which equipment can reduce blur?

  ```
  Equipment and reduces some Blur
  ```

- What types of lens is a 35-120mm?

  ```
  Lens and (hasMinEffectiveFocalLength value 35) and
  (hasMaxEffectiveFocalLength value 120)
  ```

- Which adjustments can I use to increase the exposure without affecting the depth of field?

  ```
  Adjustment and increases some ExposureLevel and not(affects
  some DepthOfField)
  ```

[1] http://protegewiki.stanford.edu/wiki/DLQueryTab

Example

- `Person and hasAge some nonNegativeInteger`
- `Person and hasAge some int[>40]`
- `Person and hasAge some int[>10,<40]`

Since Protégé 5.6.x: add datatype (e.g. xsd:int)

Acknowledgement

Acknowledgement

The slides for this course have been prepared by Saminda Abeyruwan.

Pascal Hitzler, Markus Krötzsch, and Sebastian Rudolph.
Foundations of Semantic Web Technologies.
Chapman & Hall/CRC, 2009.