Knowledge management: Querying with SPARQL

Querying RDF & RDFS

- Several query languages exist to retrieve resulting triples from RDF
  - RDQL
  - SERQL
  - SPARQL
- These languages use triple patterns as input and return matching triples as results
- SPARQL is the current W3C recommendation for querying RDF data

SPARQL

- SPARQL is an RDF query language
- SPARQL is based on **matching graph patterns against RDF graphs**
  - Graph patterns are based on triple patterns
  - RDF graphs represent the ontologies

SPARQL Triple Patterns

- A triple pattern is like an RDF triple, but with the option of a variable in place of RDF terms (i.e., IRIs, literals or blank nodes) in the subject, predicate or object positions.
- Example:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> ?title .
```

  - ?title is the variable

SPARQL Graph Patterns

- Basic graph patterns
- Group graph patterns
- Optional graph patterns
- Alternative graph patterns
- Union graph patterns

Group graph patterns are the more general case of graph pattern.

Basic Graph Pattern

- A Basic Graph Pattern is a set (sequence) of Triple Patterns
- Example:

```
?x foaf:name ?name . ?x foaf:mbox ?mbox
```

Group Graph Pattern (1/2)

- A group graph pattern is a set of graph patterns delimited with braces { }.
  - Examples:

```
{ ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```

```
{ ?x foaf:name ?name . ?x foaf:mbox ?mbox . }
```

```
{ { ?x foaf:name ?name . } { ?x foaf:mbox ?mbox . } }
```

  - When a group graph pattern consists only of BGPs, these patterns are interpreted conjunctively, and the group graph pattern is equivalent to the corresponding set of BGPs.

Group Graph Pattern (2/2)

- Group graph patterns can involve **other constructs** to define Optional Graph Patterns, Alternative Graph Patterns, etc. These constructs are introduced by certain **keywords.**

- There is **no keyword for conjunction** (e.g., AND) in SPARQL. Conjunctive triple patterns or BGPs are simply juxtaposed and then enclosed in { and } to form a group graph pattern.

Simple SPARQL Query

Data:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> "SPARQL Tutorial" .
```

Query:

```
SELECT ?title WHERE {
      <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title>
?title .
}
```

Result:

title
''SPARQL Tutorial''

- Comments:
  - Variables can also be written as $x instead of ?x.
  - We can write SELECT * **like in SQL.**

Simple SPARQL Query – Multiple Matches

Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .
_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .
_:c foaf:mbox <mailto:carol@example.org> .
```

Query:

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

Result:

Name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Exercises

```
@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 .
```

- Write SPARQL queries to:
  - Find the subclasses of MotorVehicle
  - Find the subclasses of MiniVan
  - Find the superclasses of MiniVan
- Specify the result of each query

Matching RDF Literals

- **Matching Literals with Language Tags**

Data:

```
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:x ns:p "cat"@en .
:y ns:p "42"^^xsd:integer .
:z ns:p "abc"^^dt:specialDatatype .
```

Query:

```
SELECT ?v WHERE { ?v ?p "cat" }
```

```
SELECT ?v WHERE { ?v ?p "cat“@en }
```

Result:

v

v
<http://example.org/ns#x>

Matching RDF Literals

- **Matching Literals with Numeric Types**

Data:

```
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:x ns:p "cat"@en .
:y ns:p "42"^^xsd:integer .
:z ns:p "abc"^^dt:specialDatatype .
```

Query:

```
SELECT ?v WHERE { ?v ?p 42 }
```

Result:

v
<http://example.org/ns#y>

Blank Node in Query Results (1/2)

Data:

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

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
```

Query:

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

Result:

x	name
_:c	"Alice"
_:d	"Bob"

OR

x	name
_:r	"Alice"
_:s	"Bob"

Blank Node in Query Results (2/2)

Data:

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

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name1 ?y ?name2 WHERE {
        ?x foaf:name ?name1 .
        ?y foaf:name ?name2 .
        ?x foaf:knows ?y }
```

Result:

x	name1	y	name2
_:c	"Alice"	_:d	"Bob"
_:d	"Bob"	_:c	"Alice"

Blank Node in Graph Patterns (1/4)

- Blank nodes in graph patterns act as **variables, not as references to** specific blank nodes in the data being queried.
- Blank nodes **cannot appear in a SELECT clause.**
- The **scope of blank node is the Basic Graph Pattern (BGP) in which it appears. A blank** node which **appears more than once in the same BGP stands for** the same RDF term.
- There is no reason to use blank nodes in a query; you can get the same functionality using variables.

Blank Node in Graph Patterns (2/4)

Data:

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

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

Query:

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

Result:

name
"Alice"
"Bob"

Blank Node in Graph Patterns (3/4)

Data:

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

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE {
        _:z foaf:name ?name1 .
        _:v foaf:name ?name2 .
        _:z foaf:knows _:v }
```

Result:

name1	name2
"Alice"	"Bob"
"Bob"	"Alice"

Blank Node in Graph Patterns (4/4)

Data:

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

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE {
        {_:z foaf:name ?name1}
        {_:z foaf:name ?name2}
}
```

Result:

ERROR

Exercises

```
@prefix ex: <http://example.org/data#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

ex:john     vcard:FN        “John Smith”
ex:john     vcard:N         _:X1
_:X1        vcard:Given     “John”
_:X1        vcard:Family    “Smith”
ex:john     ex:hasAge       “32”
ex:john     ex:marriedTo>   ex:mary
ex:mary     vcard:FN        “Mary Smith”
ex:mary     vcard:N         _:X2
_:X2        vcard:Given     “Mary”
_:X2        vcard:Family    “Smith”
ex:mary     ex:hasAge       “29”
```

- Write SPARQL queries to:
  - Return the full names of all people in the graph
  - Return the relation between John and Mary
  - Return the spouse of a person by the name of John Smith
  - Return the last name and the first name of all people in the KB
- Specify the result of each query

Query Forms

- SPARQL has four query forms
  - **SELECT**: Returns all, or a subset of, the variables bound in a query pattern match.
  - **CONSTRUCT**: Returns an RDF graph specified by a graph template.
  - **ASK**: Used to test whether or not a graph pattern has a solution. Returns a boolean indicating whether a query pattern matches or not.
  - **DESCRIBE**: Returns an RDF graph that describes the resources found.

Query Forms - CONSTRUCT

Data:

```
@prefix org: <http://example.com/ns#> .

_:a     org:employeeName     "Alice" .
_:a     org:employeeId       12345 .
_:b     org:employeeName     "Bob" .
_:b     org:employeeId       67890 .
```

Query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX org: http://example.com/ns#

CONSTRUCT { ?x foaf:name ?name }
WHERE {
        ?x org:employeeName ?name }
```

Result (a graph):

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

Query Forms – ASK (1/2)

Data:

```
@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> .
```

Query:

```
PREFIX foaf: http://xmlns.com/foaf/0.1/

ASK { ?x foaf:name "Alice" }
```

Result (a Boolean):

Yes

Query Forms – ASK (2/2)

Data:

```
@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> .
```

Query:

```
PREFIX foaf: http://xmlns.com/foaf/0.1/

ASK {  ?x       foaf:name       "Alice" ;
                foaf:mbox       <mailto:alice@work.example>
}
```

Result (a Boolean):

No

Testing Values

- The FILTER construct **restricts variable bindings to those for which the filter** expression evaluates to TRUE.

Testing Values - Arithmetic Filters

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

Query:

```
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 . }
```

Result:

Title	price
"The Semantic Web"	23

Testing Values - String Filters

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

Query:

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

Starts with SPARQL

Result:

Title
"SPARQL Tutorial"

Testing Values - Scope of Filters

- A FILTER condition is **a restriction on solutions over the whole Group Graph Pattern in which** the filter appears.
- The following graph patterns all have the 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 .}
```

Testing Values

- We can have **multiple FILTERs in a group** graph pattern. They are equivalent to a single filter with conjoined filter conditions.
- FILTERs can be very complex Boolean conditions. Details can be found in the SPARQL specification
  - http://www.w3.org/TR/rdf-sparql-query/
- The regular expression language used by regex is defined in XQuery 1.0 and Xpath 2.0.
  - http://www.w3.org/TR/xquery/
  - http://www.w3.org/TR/xpath20/

Including Optional Values

- Regular, complete structures cannot be assumed in all RDF graphs.
- It is useful to have queries that allow **information to be added** to the answer **when** the **information is available**, **but** do **not reject** the answer because some part of the query pattern does not match.
- The **OPTIONAL** construct provides this facility: if the optional part does not match, it creates no bindings but does not eliminate the solution.

Optional Pattern Matching

Data:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntaxns#>
.
_: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" .
```

Query:

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

Result:

Name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Constraints in Optional Pattern Matching (1/2)

- The group graph pattern following a keyword OPTIONAL can of course be as complex as possible e.g., it can contain a FILTER.

Constraints in Optional Pattern Matching (2/2)

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

Query:

```
SELECT ?title ?price
WHERE { ?x  dc:title  ?title .
        OPTIONAL { ?x  ns:price  ?price . FILTER (?price < 30) }
}
```

Result:

Title	Price
"SPARQL Tutorial"	
"A New SPARQL Tutorial"	
"The Semantic Web""	23

There is no ns:price property for ?x

There is an ns:price property for ?x but its value is >=30

Multiple Optional Graph Patterns (1/4)

- A graph pattern may have zero or more optional graph patterns, and any part of a query pattern may have an optional part.
- OPTIONAL is left-associative:

```
P1 OPTIONAL { P2 } OPTIONAL { P3 }
   is equivalent to
{ P1 OPTIONAL { P2 } } OPTIONAL { P3 }
```

```
{ P1 OPTIONAL  P2 } OPTIONAL  P3
   is not equivalent to
   P1 OPTIONAL { P2 OPTIONAL  P3 }
```

- The operator **OPTIONAL** has **higher precedence** than **conjunction** (remember: conjunction is encoded as juxtaposition of graph patterns).

Multiple Optional Graph Patterns(2/4)

Data:

```
@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> .
```

Query:

```
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 . }
}
```

Result:

name	mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

Multiple Optional Graph Patterns(3/4)

Data:

```
@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 .
```

Query 1:

```
SELECT ?book ?title
WHERE { ?book   dc:creator   ?author .
          OPTIONAL { ?book   dc:title   ?title .}
          { ?book   ns:price   ?price .}
}
```

Result:

book	title
"<http://example.org/book3>"	
"<http://example.org/book1>"	"Semantic Web"

Multiple Optional Graph Patterns(4/4)

Data:

```
@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 .
```

Query 2:

```
SELECT ?book ?title
WHERE { ?book   dc:creator   ?author .
          OPTIONAL { { ?book   dc:title    ?title .}
                     { ?book   ns:price    ?price .} }
}
```

Result:

book	title
"<http://example.org/book3>"	
"<http://example.org/book2>"	
"<http://example.org/book1>"	"Semantic Web"

Matching Alternatives

- SPARQL provides a means of combining graph patterns so that one of several alternative graph patterns may match. If more than one of the alternatives matches, all the possible pattern solutions are found.

- Pattern alternatives are syntactically specified with the UNION keyword.

Matching Alternatives – Example 1

Data:

```
@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)" .
```

Query 1:

```
SELECT  ?title
WHERE {
  { ?book  dc10:title  ?title }
   UNION
  { ?book  dc11:title  ?title }
}
```

Result:

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Matching Alternatives - Example 2

Data:

```
@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)" .
```

Query 2:

```
SELECT  ?author  ?title
WHERE {
  { ?book  dc10:title  ?title . ?book  dc10:creator  ?author . }
   UNION
  { ?book  dc11:title  ?title . ?book  dc11:creator  ?author . }
}
```

Result:

author	title
"Alice"	"SPARQL Query Language Tutorial"
"Bob"	"SPARQL Protocol Tutorial"

Matching Alternatives – use of variables

- We have to be careful whether or not to use the same variables in each alternative. This decision depends on what we want to compute.

Data:

```
_: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)" .
```

Query 3:

```
SELECT ?x ?y
WHERE { {?book  dc10:title  ?x} UNION {?book  dc11:title  ?y} }
```

Result:

x	y
	"SPARQL (updated) "
	"SPARQL Protocol Tutorial"
"SPARQL“	
"SPARQL Query Language Tutorial"	

Matching Alternatives
combining UNION and conjunctions

Data:

```
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 .
```

Query 1:

```
SELECT ?book ?title ?price
WHERE {
{ ?book  dc:creator  ex:Smith . ?book  dc:title  ?title . }
UNION
{ ?book  dc:creator  ex:Jones . ?book  ns:price  ?price . }
}
```

Result:

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

Matching Alternatives
combining UNION and conjunctions

Data:

```
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 .
```

Query 2:

```
SELECT ?book ?title ?price
WHERE {
{ ?book  dc:creator  ex:Smith . ?book  dc:title  ?title . }
UNION
{ ?book  dc:creator  ex:Jones .} { ?book  ns:price  ?price . }
}
```

Result:

book	title	price
"<http://example.org/book3>"		35
"<http://example.org/book2>"		30

Exercises

```
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 .
exthings:mv1             rdf:type          ex:MiniVan .
exthings:tr1             rdf:type          ex:Truck .
exthings:pv1             rdf:type          ex:PassengerVehicle .
exthings:v1              rdf:type          ex:Van .
```

- Write SPARQL queries to:
  - Vehicle Knowledge Base
    - Find the superclasses of MiniVan that are not rdf:Resource.
    - Find all the motor vehicles
    - Find all the vans and passenger vehicles
  - Person Knowledge Base
    - Return all people over 30 in the KB
- Specify the result of each query

Solution Sequence Modifiers

- Order modifier: put the solutions in order
- Projection modifier: choose certain variables
- 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 – ORDER BY

Data:

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

_:a  foaf:name   "Alice" .
_:a  foaf:age    30 .

_:b  foaf:name   "Bob" .
_:b  foaf:age    20 .
```

Query:

```
SELECT ?name
WHERE { ?x      foaf:name      ?name .
        ?x      foaf:age       ?age
}
ORDER BY  DESC (?age)
```

Result:

name
"Bob"
"Alice"

Solution Sequence Modifiers – DISTINCT

Data:

```
_: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> .
```

Query:

```
SELECT DISTINCT ?name
WHERE { ?x   foaf:name   ?name }
```

Result:

name
"Alice"

Solution Sequence Modifiers – LIMIT

Data:

```
_: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> .
```

Query:

```
SELECT  ?name
WHERE { ?x  foaf:name  ?name }
LIMIT 2
```

Result:

name
"Alice"
"Alice"

Negation in SPARQL (1/2)

- The Boolean “not” (!) operator 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:

y

Negation in SPARQL (2/2)

- A form of negation simulated using OPTIONAL, FILTER and !bound.

Data:

```
_:a      foaf:givenName  "Alice“ .

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

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

Query: Display names of peoples with no expressed age

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

Result:

Name
“Alice”

Exercise

Data:

```
_:a     foaf:givenName  "Alice“ .

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

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

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

References

- Slides based on:
  - SPARQL Query Language for RDF, W3C recommendation

    http://www.w3.org/TR/rdf-sparql-query/
  - Introduction to SPARQL

    http://cgi.di.uoa.gr/~pms509/lectures/SPARQL%20-%20lecture%201.pdf

Exercise

```
<foaf:Person rdf:about="#Anakin">
        <foaf:name>Darth Vader</foaf:name>
        <foaf:familyname>Skywalker</foaf:familyname>
        <foaf:givenname>Anakin</foaf:givenname>
        <foaf:gender>male</foaf:gender>
</foaf:Person>

<foaf:Person rdf:about="#Padme">
        <foaf:name>Amidala Padme</foaf:name>
        <foaf:familyname>Padme</foaf:familyname>
        <foaf:givenname>Amidala</foaf:givenname>
        <foaf:gender>female</foaf:gender>
        <foaf:knows><foaf:Person rdf:ID="Anakin" /></foaf:knows>
        <foaf:knows><foaf:Person rdf:ID="Bail" /></foaf:knows>
</foaf:Person>

<foaf:Person rdf:about="#Bail">
        <foaf:name>Bail Oranga</foaf:name>
        <foaf:knows><foaf:Person rdf:ID="Padme" /></foaf:knows>
        <foaf:knows>
                <foaf:Person><foaf:name>Yoda</foaf:name></foaf:Person>
        </foaf:knows>
</foaf:Person>
```