Querying RDF data using SPARQL

F. Toumani

Institut d'Informatique, LIMOS, UCA

February 6, 2018

What is SPARQL?

SPARQL 1.1 is a set of specifications that provide languages and protocols to query and manipulate RDF graph content on the Web or in an RDF store [W3C]

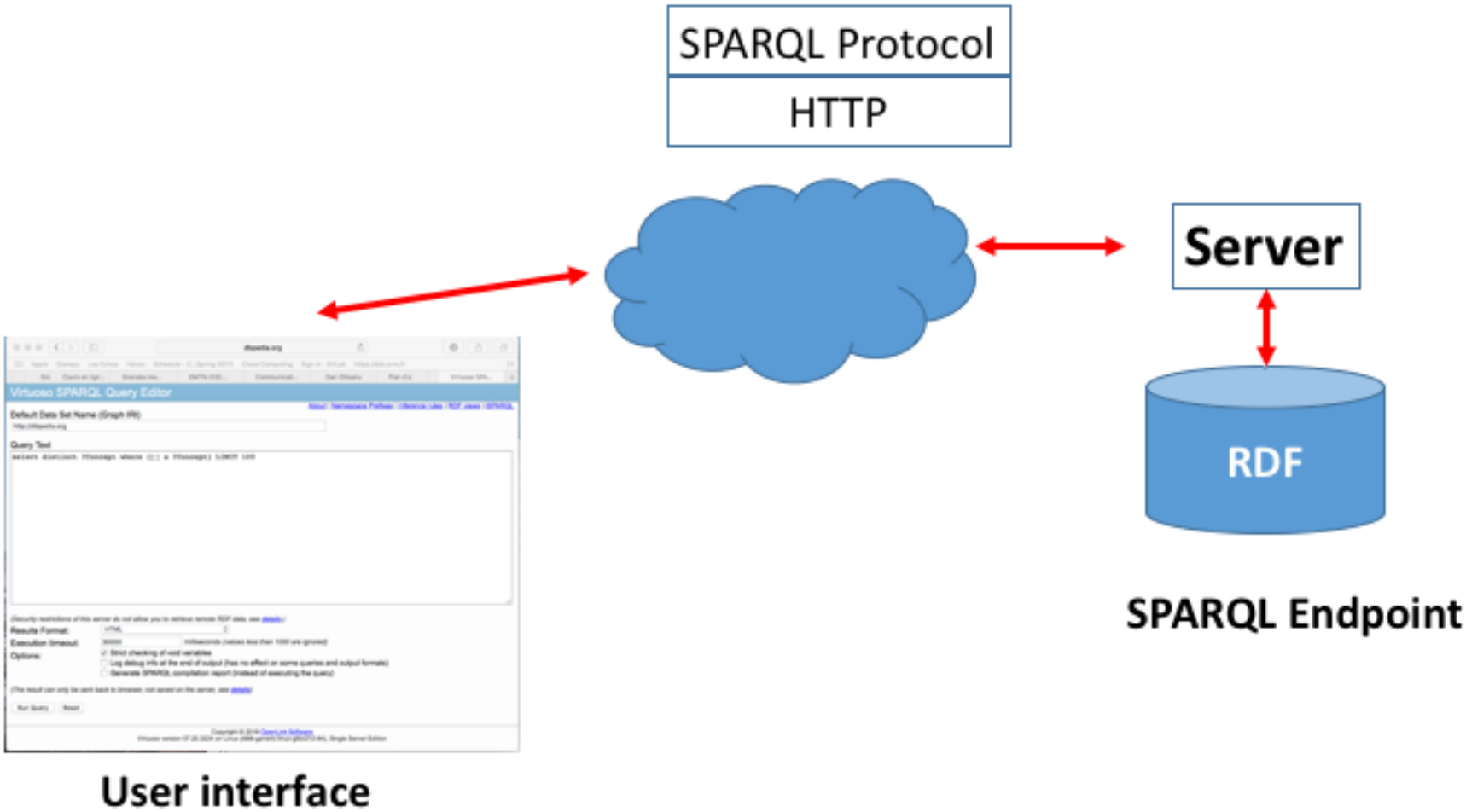
SPARQL 1.1 specifications

- A query language for RDF.
- Different query results formats: XML, JSON, CSV (comma separated values) and TSV (tab separated values)
- An update language for RDF graphs
- Federated Query defines an extension of the query language for executing queries distributed over different SPARQL endpoints
- SPARQL 1.1 Entailment Regimes: defines the semantics of SPARQL queries under entailment regimes such as RDF Schema, OWL, or RIF

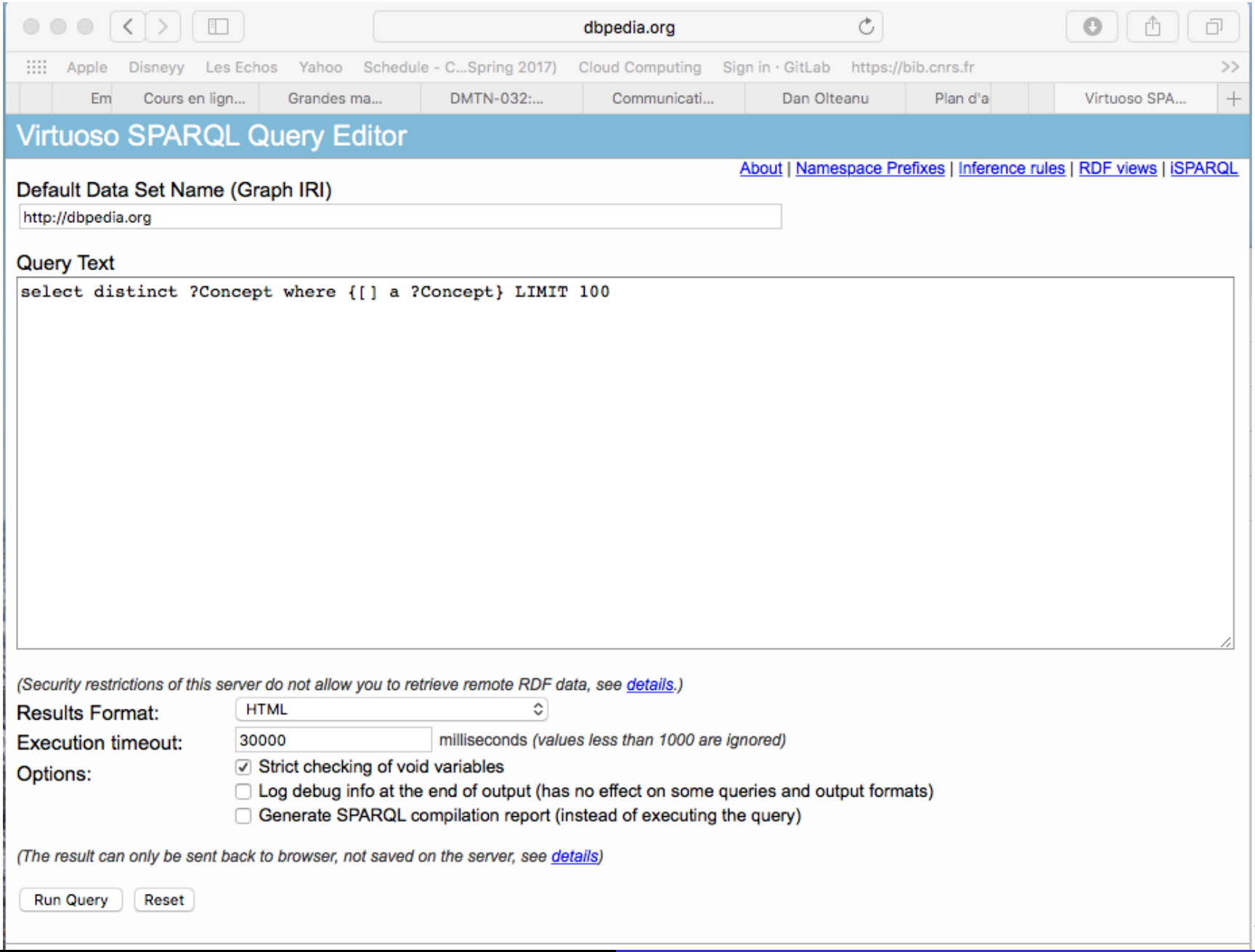
SPARQL 1.1 specifications (cont.)

- SPARQL 1.1 Protocol for RDF: a protocol defining means for conveying arbitrary SPARQL queries and update requests to a SPARQL service
- SPARQL 1.1 Service Description: a specification defining a method for discovering and a vocabulary for describing SPARQL services
- SPARQL 1.1 Graph Store HTTP Protocol: as opposed to the full SPARQL protocol, this specification defines minimal means for managing RDF graph content directly via common HTTP operations.

Using SPARQL to query RDF data



Example of SPARQL query Editor



Example of an RDF dataset
from DBtune (http://dbtune.org/bbc/peel/#dump)

```
@prefix mo: <http://purl.org/ontology/mo/> .
...
<http://dbtune.org/bbc/peel/work/1216>
        a               mo:MusicalWork ;
        rdfs:label      "Wolfcub" ;
        dc:title        "Wolfcub" .

<http://dbtune.org/bbc/peel/artist/acc07..29d>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/acc07..29d> ;
        foaf:name           "James Valentin" .

<http://dbtune.org/bbc/peel/perf_ins/a48..b7>
        a                   mo:Performance ;
        mo:instrument       "Keyboards, Guitar, Trumpet, Backing Vocals" ;
        mo:performer        <http://dbtune.org/bbc/peel/artist/a48..b7>.
```

- **Variables** are prefixed by either **?** or **$**

  ?x, ?toto, $x, $name

- **Triple patterns** are RDF triples where each of the subject, predicate and object may be a **variable**

  ?person foaf:knows ?friend .

  ?x foaf:knows ?y .

- A Basic Graph Pattern (**BGP**) is a set of triple patterns

SPARQL: simple queries
The Select, Where clauses

- **SELECT** clause identifies the variables to appear in the query results
- **WHERE** clause provides the basic graph pattern to **match** against the data graph

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

SELECT      ?name
  WHERE {
            dbtune:acc0729d    foaf:name    ?name .
            }
```

- **SELECT** clause identifies the variables to appear in the query results
- **WHERE** clause provides the basic graph pattern to **match** against the data graph

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

SELECT        ?name
  WHERE {
                dbtune:acc0729d    foaf:name    ?name .
                }
```

name
James Valentin

SPARQL Syntax

- Literals
  "chat"
  'chat'@fr with language tag "fr"
  "xyz"^^<http://example.org/ns/userDatatype>
  "abc"^^appNS:appDataType
  1, which is the same as "1"^^xsd:integer
  1.3, which is the same as "1.3"^^xsd:decimal
  true, which is the same as "true"^^xsd:boolean

SPARQL Syntax

- Triple patterns
  - Predicate-Object Lists
    ?x foaf:name ?name ;
    foaf:mbox ?mbox .
  - Object Lists
    ?x foaf:mbox "alice@uca.fr" , "alice@uca.fr" .
  - RDF Collections
    (1 ?x 3 4) :p "w" .
  - rdf:type
    ?x a my:person .

Matching

The result of a query is a set of solutions corresponding to the ways in which the query's graph pattern **matches** the data

- Each solution gives one way in which the selected variables can be bound to RDF terms so that the query pattern matches the data
- The result set gives all the possible solutions

$\Rightarrow$ Substitution of variables with RDF terms

SPARQL: simple queries

Examples

```
<http://dbtune.org/bbc/peel/artist/acc07..29d>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/acc07..29d> ;
        foaf:name           "James Valentin" .

<http://dbtune.org/bbc/peel/artist/00f46>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/00f46> ;
        foaf:name           "Pete Baron" .

<http://dbtune.org/bbc/peel/artist/00daec>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/00daec> ;
        foaf:name           "David Gamble" .
```

```
SELECT ?name
  WHERE {
                ?artist mo:performed ?ref .
                ?artist foaf:name ?name .
            }
```

SPARQL: simple queries
Examples

```
<http://dbtune.org/bbc/peel/artist/acc07..29d>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/acc07..29d> ;
        foaf:name           "James Valentin" .

<http://dbtune.org/bbc/peel/artist/00f46>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/00f46> ;
        foaf:name           "Pete Baron" .

<http://dbtune.org/bbc/peel/artist/00daec>
        a                   foaf:Person ;
        mo:performed        <http://dbtune.org/bbc/peel/perf_ins/00daec> ;
        foaf:name           "David Gamble" .
```

```
SELECT ?name
  WHERE {
                ?artist mo:performed ?ref .
                ?artist foaf:name ?name .
              }
```

name	ref
James Valentin	<http://dbtune.org/bbc/peel/perf_ins/acc07..29d>
Pete Baron	<http://dbtune.org/bbc/peel/perf_ins/00f46>
David Gamble	<http://dbtune.org/bbc/peel/perf_ins/00daec>

Matching RDF Literals

```
@prefix my: <http://ex.org/myexample> .

my:doc1 my:title "Alice"@en .
my:doc1 my:price "42"^^xsd:integer .
my:doc1 my:period "abc"^^my:specialDatatype .
```

- Matching Literals with Numeric Types
  `select ?v where { ?v ?p 42 }`
- Matching Literals with Language Tags
  `select ?v where { ?v ?p "Alice"@en }`
- Matching Literals with Arbitrary Datatypes
  `select ?v where  ?v ?p "abc"^^<http://ex.org/datatype#specialDatatype>`

Dealing with blank nodes

@prefix my: <http://ex.org/myexample> .

_:a my:title "Alice" .

_:b my:title "Automata Theory" .

select ?a ?y where { ?x my:title ?title }

x	title
_:e	"Alice"
_:f	"Automata Theory"

Dealing with blank nodes

@prefix my: <http://ex.org/myexample> .

_:a my:title "Alice" .

_:b my:title "Automata Theory" .

select ?a ?y where { ?x my:title ?title }

x	title
_:e	"Alice"
_:f	"Automata Theory"

x	title
_:g	"Alice"
_:h	"Automata Theory"

Blank nodes are scoped to a result set or to the result graph (in case of the **construct** *query form)*

SPARQL query forms

- Select
  returns variable bindings
- Construct
  returns an RDF graph specified by a graph template
- ASK
  returns a boolean indicating whether a query pattern matches or not
- Describe
  returns an RDF graph that describes the resources found

Examples

```
@prefix my: <http://ex.org/myexample> .

my:id01 my:name "Alice" .
my:id01 my:pcode 63 .
my:id02 my:pcode 6" .
my:pcode my:department "Puy-de-Dome" .
my:pcode my:department "Rhone" .
```

```
@prefix my: <http://ex.org/myexample> .

CONSTRUCT   { ?name my:livingDepartment ?dep }
WHERE        { ?x      my:name      ?name .
               ?x      my:code      63 .
               63      my:department      ?dep   }
```

Examples

```
@prefix my: <http://ex.org/myexample> .

my:id01 my:name "Alice" .
my:id01 my:pcode 63 .
my:id02 my:pcode 6" .
my:pcode my:department "Puy-de-Dome" .
my:pcode my:department "Rhone" .
```

```
@prefix my: <http://ex.org/myexample> .

CONSTRUCT   { ?name my:livingDepartment ?dep }
WHERE       { ?x    my:name             ?name .
              ?x    my:code             63 .
              63    my:department       ?dep   }

ASK   {       my:id01   my:name   "Alice" ;
              my:pcode  63 }
```

Examples

```
@prefix my: <http://ex.org/myexample> .

my:id01 my:name "Alice" .
my:id01 my:pcode 63 .
my:id02 my:pcode 6" .
my:pcode my:department "Puy-de-Dome" .
my:pcode my:department "Rhone" .
```

```
@prefix my: <http://ex.org/myexample> .

CONSTRUCT   { ?name my:livingDepartment ?dep }
WHERE       { ?x      my:name      ?name .
              ?x      my:code      63 .
              63      my:department      ?dep   }

ASK   {       my:id01      my:name      "Alice" ;
              my:pcode      63 }

ASK   {       my:id01      my:name      "Alice" ;
              my:pcode      75 }
```

RDF Term Constraints

FILTERs restrict solutions to those for which the filter expression evaluates to TRUE

```
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        FILTER (?price < 30.5) .
        ?x dc:title ?title . }
```

```
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        FILTER (LANG(?title)="en") . }
```

```
SELECT ?title
WHERE { ?x dc:title ?title
        FILTER regex(?title, "^SPARQL") .
        }
```

Example of filter functions

- Arithmetic operators
- SPARQL Operators
  isIRI(A), isURI(A), isBLANK(A), isLITERAL(A), LANG(A), DATATYPE(A) ...
- Testing regular expressions (regex)
  Regular expression language of XQuery 1.0 and XPath 2.0
  - Matching the Start and End of the String ^ and $
  - Quantifiers ?, *, ...

Graph Patterns

- Basic Graph Patterns
  - A set of triple patterns
  - All the patterns must match

```
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        ?x dc:title ?title . }
```

- Group Graph Pattern
  - A set of triple patterns delimited by braces
  - All the sets must match

```
SELECT ?title ?price
WHERE { { ?x ns:price ?price . }
        { ?x dc:title ?title . } }
```

Graph Patterns (cont.)

- Filters are scoped to the basic graph pattern

```
SELECT ?title ?price
WHERE { ?x ns:price ?price .
FILTER (?price < 30.5) .
?x dc:title ?title . }
```

```
SELECT ?title ?price
WHERE { FILTER (?price < 30.5) .
?x ns:price ?price .
?x dc:title ?title . }
```

- Labels for blank nodes are scoped to the basic graph pattern

```
SELECT ?title ?price
WHERE { _:xb ns:price ?price .
_:xb dc:title ?title . }
```

```
SELECT ?title ?price
WHERE { { _:xb ns:price ?price . }
{ _:xb dc:title ?title . } }
```

Graph Patterns (cont.)

- Optional Graph patterns

```
SELECT ?title ?price
WHERE {
        ?x dc:title ?title .
        OPTIONAL { ?x ns:price ?price .
                   FILTER (?price < 30.5) . } }
```

- Union (alternative matching)

```
SELECT ?title ?price
WHERE {
        { ?x dc:title ?title .
          ?x ns:price ?price .
          FILTER (LANG(?title)="en") . }
        UNION
        { ?x dc:title ?title
          ?x ns:price ?price .
          FILTER (?price < 30.5) . }
}
```

Solutions sequences modifiers

- Order modifier (Order By)
  - Order between RDF terms that would not otherwise be ordered: No value $<$ Blank nodes $<$ IRIs $<$ RDF literals
- Projection modifier
- Distinct modifier
- Reduced modifier: 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

Examples

```
Select ?title ?year
Where { ?x  my:title  ?title .
?x  my:year  ?year . }
ORDER BY DESC (?year)
```

```
Select ?title
Where { ?x  my:title  ?title . }
```

```
Select DISTINCT ?title
Where { ?x  my:title  ?title . }
```

```
Select ?title ?year
Where { ?x  my:title  ?title .
?x  my:year  ?year . }
Order by desc (?year)
LIMIT 3
OFFSET 2
```