#### CC6202-1 LA WEB DE DATOS PRIMAVERA 2016

#### Lecture 7: SPARQL (1.0)

Aidan Hogan aidhog@gmail.com

# (1) Data, (2) Rules/Ontologies, (3) Query

INPUT: "(x, partOf, y)"





### SPARQL: Query Language for RDF



#### How to ask: "Who stars in 'Sharknado'?"

# SPARQL: Query Language for RDF



#### Query:

PREFIX ex: <http://ex.org/voc#>
SELECT \*
WHERE {
 ex:Sharknado ex:stars ?star .
}

#### Solutions:

?star

ex:JohnHeard

ex:IanZiering

# **SPARQL: PREFIX DECLARATIONS**

# SPARQL: prefix declarations

• Shortcuts for IRIs (exactly like in Turtle)

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
    ex:Sharknado ex:stars ?star .
}
```

# SPARQL: WHERE CLAUSE

## SPARQL: WHERE clause

- Where the magic happens
- Specifies what to match in the data

PREFIX ex: <http://ex.org/voc#>
SELECT \*

WHERE {

}

ex:Sharknado ex:stars ?star .



(a triple with variables)

## SPARQL: WHERE clause



#### Query:

PREFIX ex: <http://ex.org/voc#>
SELECT \*
WHERE {
 ex:Sharknado ex:stars ?star .
}

#### Solutions:

?star

ex:JohnHeard

ex:IanZiering

#### SPARQL: WHERE clause



How to ask: "What movies did the stars of 'Sharknado' also star in?"

#### SPARQL: Basic Graph Patterns



#### Query:

PREFIX ex: <http: ex.org="" voc#=""></http:>	•
SELECT *	
WHERE {	
ex:Sharknado ex:stars ?star .	
?movie ex:stars ?star .	
}	

?star	?movie
ex:IanZiering	ex:Sharknado2
ex:IanZiering	ex:Sharknado
ex:JohnHeard	ex:Sharknado

#### SPARQL: Basic Graph Patterns



### **SPARQL: Join Variables**



#### Query:



#### "Join Variable"

(a variable appearing multiple times)

#### **SPARQL:** Disjunction



How to ask: "What are the titles of the (first two) movies in the Sharknado series?"

# SPARQL: Disjunction (UNION)



#### Query:

# PREFIX ex: <http://ex.org/voc#> SELECT \* WHERE { { ex:SharknadoSeries ex:firstMovie ?movie . } UNION { ex:SharknadoSeries ex:secondMovie ?movie . } ?movie ex:title ?title . }

?movie	?title
ex:Sharknado	"Sharknado"@en
ex:Sharknado2	"Sharknado 2: The Second One"@en

#### SPARQL: Left-join



How to ask: "Give me the titles of all movies and, if available, their first-aired date?"

# SPARQL: Left-join (OPTIONAL)



#### Query:

# PREFIX ex: <http://ex.org/voc#> SELECT \* WHERE { ?movie a ex:Movie ; ex:title ?title . OPTIONAL { ?movie ex:firstAired ?date } }

?movie	?title	?date
ex:Sharknado	"Sharknado"@en	"2013-07-11"^^xsd:date
ex:Sharknado2	"Sharknado 2: The Second One"@en	
<b>"</b> U"	NBOUND Variable"	
(a variable without a binding in a solution)		

#### SPARQL: Filtering results



#### How to ask: "What movies were first aired in 2014?"

# SPARQL: FILTER



# SPARQL: FILTER



#### Query:

FILTERS (and other functions we see later) expect certain types. If not given, a type error is given.

What happens in this case where ?date bound in data to a string?

# SPARQL: Boolean FILTER operators

- FILTERs evaluate as true, false or error
- Only results evaluating as true are returned
- Can apply AND (&&) or OR (||)
- Can also apply NOT (!)

 $-!E \rightarrow E$ 

![](_page_21_Figure_6.jpeg)

## SPARQL: RDF term FILTER operators

- ISIRI(A), ISURI(A), ISBLANK(A), ISLITERAL(A)
  - checks the type of RDF term
  - ISIRI and ISURI are synonymous
- BOUND(A)
  - checks if the variable is bound

# SPARQL: (In)equality FILTER operators

- =, !=, SAMETERM(A,B)
  - = and != test value (in)equality
  - SAMETERM tests term equality
    - e.g., "2.0"^^decimal = "2"^^xsd:int gives true SAMETERM("2.0"^^decimal,"2"^^xsd:int) gives false
- >, <, >=, <=
  - can only compare "compatible" types
    - e.g., "2.0"^^decimal > "2"^^xsd:int okay, "2.0"^^decimal > "2" an error

### SPARQL: Numeric FILTER operators

- +A, -A, A+B, A-B, A\*B, A/B (numeric)
  - input numeric, output numeric

# SPARQL: Literal/string FILTER operators

- STR(A), LANG(A), DATATYPE(A)
  - STR returns string of RDF term (literal or IRI)
  - LANG returns language tag of literal
  - DATATYPE returns datatype of literal
  - All return xsd:string
- LANGMATCHES(A,B)tests (sub-)language
  - e.g.:
    - LANGMATCHES("en", "en") gives true
    - LANGMATCHES("en-US", "en") gives true
    - LANGMATCHES("en", "en-US") gives false
- **REGEX(A,B,C)** tests a regular expression
  - C sets some optional tags like case insensitivity
  - e.g.:
    - REGEX("blah","^B") gives false
    - REGEX("blah", "^B", "i") gives true

## SPARQL: Casting between types

- Y: always allowed
- N: never allowed
- M: depends on value
  - e.g., "2"^^xsd:string can be mapped to xsd:int but "P"^^xsd:string cannot

From \ To	str	flt	dbl	dec	int	dT	bool
str	Y	М	М	М	М	М	М
flt	Y	Y	Y	М	М	Ν	Y
dbl	Y	Y	Y	М	М	Ν	Y
dec	Y	Y	Y	Y	Y	Ν	Y
int	Y	Y	Y	Y	Y	Ν	Y
dT	Y	Ν	Ν	Ν	Ν	Y	Ν
bool	Y	Y	Y	Y	Y	Ν	Y
IRI	Y	N	Ν	Ν	Ν	N	Ν
ltri	Y	М	М	М	М	М	М

bool = <u>xsd:boolean</u> dbl = <u>xsd:double</u> flt = <u>xsd:float</u> dec = <u>xsd:decimal</u> int = <u>xsd:integer</u> dT = <u>xsd:integer</u> dT = <u>xsd:dateTime</u> str = <u>xsd:string</u> IRI = IRI Itrl = simple literal

## SPARQL: Extensible/User-defined Functions

- A SPARQL implementation can choose to implement custom functions
  - e.g., ex:isOddNumber(A)
- A common example in practice is for free-text search
- The SPARQL syntax allows it but the engine must interpret the function (or throw an error if not supported)

# SPARQL: WHERE clause example (i)

![](_page_28_Figure_1.jpeg)

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
    { ex:SharknadoSeries ex:firstMovie ?movie . }
    UNION
    { ex:SharknadoSeries ex:secondMovie ?movie . }
    OPTIONAL
    { ?movie ex:firstAired ?date . }
    ?movie ex:title ?title .
    FILTER(REGEX(STR(?title),"*[0-9]*"))
}
```

#### What solutions would this query return?

?movie	?title	?date
ex:Sharknado2	"Sharknado 2: The Second One"@e	1

# SPARQL: WHERE clause example (ii)

![](_page_29_Figure_1.jpeg)

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
    ?movie a ex:Movie .
    OPTIONAL
    { ?movie ex:firstAired ?date . }
    FILTER(!BOUND(?date))
}
```

#### What solutions would this query return?

#### Solutions:

?movie	?date
ex:Sharknado2	

#### Can do a closed-world style of negation!

# **SPARQL: QUERY TYPES**

# SPARQL: SELECT with \*

![](_page_31_Figure_1.jpeg)

#### Query:

# PREFIX ex: <http://ex.org/voc#> SELECT \* WHERE { ?movie a ex:Movie. ?movie ex:stars ?star . }

?movie	?star
ex:Sharknado	ex:JohnHeard
ex:Sharknado	ex:IanZiering
ex:Sharknado2	ex:IanZiering

# SPARQL: SELECT with projection

![](_page_32_Figure_1.jpeg)

# SPARQL: SELECT with DISTINCT

![](_page_33_Figure_1.jpeg)

# SPARQL: SELECT with REDUCED

![](_page_34_Figure_1.jpeg)

#### SPARQL: ASK

![](_page_35_Figure_1.jpeg)

#### Query:

![](_page_35_Figure_4.jpeg)

![](_page_35_Picture_5.jpeg)

# SPARQL: CONSTRUCT

![](_page_36_Figure_1.jpeg)

#### Query:

# PREFIX ex: <http://ex.org/voc#> CONSTRUCT { ?star ex:job ex:Actor } WHERE { ?movie a ex:Movie. ?movie ex:stars ?star . }

#### Solutions:

@prefix ex: <http://ex.org/voc#> .
ex:JohnHeard ex:job ex:Actor .
ex:IanZiering ex:job ex:Actor .

# Returns an RDF graph based on the matching CONSTRUCT clause.

# SPARQL: DESCRIBE (optional feature)

![](_page_37_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
DESCRIBE ?star
WHERE {
    ?movie a ex:Movie.
    ?movie ex:stars ?star .
}
```

#### Solutions:

@prefix ex: <http://ex.org/voc#> .
ex:JohnHeard a ex:Person .
ex:IanZiering a ex:Person .

Returns an RDF graph "describing" the returned results. This is an optional feature. What should be returned is left open.

# **SPARQL: SOLUTION MODIFIERS**

# Solution modifiers

- ORDER BY (DESC)
  - Can be used to order results
  - By default ascending (ASC), can specify descending (DESC)
  - Can order lexicographically on multiple items
- LIMIT n
  - Return only *n* results
- OFFSET n
  - Skip the first *n* results

Strictly speaking, by default, no ordering is applied. Hence OFFSET means nothing without ORDER BY. However, some engines support a default ordering (e.g., the order of computation of results).

How might we ask for the second and third most recently released movies?

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE { ?movie ex:firstAired ?date . }
ORDER BY DESC(?date)
LIMIT 2
OFFSET 1
```

# **SPARQL: NAMED GRAPHS**

#### SPARQL defined over a Dataset

- A dataset  $D = \{G, (G_1, n_1), \dots, (G_k, n_k)\}$
- $G, G_1, \ldots, G_k$  are RDF graphs
- $n_1, \ldots, n_k$  are IRIs
- G is called the **default graph**
- each  $(G_i, n_i)$  is a named graph  $(1 \le i \le n)$

Core idea: SPARQL can support multiple RDF graphs, not just one.When using SPARQL, you can partition your data into multiple graphs. The default graph is chosen if you don't specify a graph.Otherwise you can explicitly select a named graph using it's IRI name.

### An example dataset

![](_page_42_Figure_1.jpeg)

## Creating a dataset for a query

- Say an index has dataset  $D = \{G, (G_1, n_1), \dots, (G_k, n_k)\}$
- A query can pick an active dataset from the named graphs
- FROM
  - Used to define a default graph for the query using graph names
  - If multiple graphs are specified, they are RDF-merged
- FROM NAMED
  - Used to select the active named graphs to be used for the query
- Using either feature clears the index dataset

# Querying the named graphs in a dataset

- We can query parts of the dataset using ...
- GRAPH: specifies the URI of a named graph from which results or a variable that ranges over all named graphs
  - Does not access the default graph!
  - If <u>not</u> specified, default graph is accessed

# An example query

![](_page_45_Figure_1.jpeg)

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }

**No** GRAPH clause so answers come from default graph only

What solutions would this query return?

#### Solutions:

?s

ex:SharknadoSeries

# Using FROM

![](_page_46_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado.ttl
FROM ex:Sharknado2.ttl
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }
```

**No** GRAPH clause so answers come from default graph defined by FROM (existing default graph cleared)

#### What solutions would this query return?

![](_page_46_Picture_7.jpeg)

# Using FROM NAMED

![](_page_47_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?s
WHERE { ?s ?p ?o }
```

#### What solutions would this query return?

```
Solutions:
```

?s

**No** GRAPH clause so answers come from default graph, which is empty (since existing default graph cleared)!

# Using GRAPH with variable

![](_page_48_Figure_1.jpeg)

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s ?g
WHERE { GRAPH ?g { ?s ?p ?o } }
```

GRAPH clause only ranges over the named graphs.

What solutions would this query return?

?s	?g
ex:Sharknado	ex:Sharnado.ttl
ex:Sharknado2	ex:Sharnado2.ttl

# Using GRAPH with a name

![](_page_49_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?s ?g
WHERE {
   GRAPH ex:Sharknado.ttl { ?s ?p ?o }
}
```

#### What solutions would this query return?

?s	?g
ex:Sharknado	ex:Sharnado.ttl

# Using GRAPH with FROM

![](_page_50_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado.ttl
SELECT DISTINCT ?s ?g
WHERE {
    GRAPH ?g { ?s ?p ?o }
}
```

#### What solutions would this query return?

# Using GRAPH with FROM NAMED

![](_page_51_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?s ?g
WHERE {
   GRAPH ?g { ?s ?p ?o }
}
```

#### What solutions would this query return?

#### Solutions:

?s	?g
ex:Sharknado	ex:Sharnado.ttl

GRAPH accesses the one and only named graph

# Using GRAPH with FROM and FROM NAMED

![](_page_52_Figure_1.jpeg)

#### Query:

```
PREFIX ex: <http://ex.org/voc#>
FROM ex:Sharknado2.ttl
FROM NAMED ex:Sharknado.ttl
SELECT DISTINCT ?x ?q
WHERE {
   GRAPH ?g { ?s ?p ?o }
   ?x ?q ?o .
}
```

#### What solutions would this query return?

?x	?q
ex:Sharknado2	ex:stars

# RECAP

# Parts of a SPARQL query (i)

- Prefix declarations
- WHERE clause
  - Joins / Basic Graph Patterns
  - UNION
  - OPTIONAL
  - FILTER
- Solution modifiers
  - ORDER BY
  - LIMIT
  - OFFSET

# Parts of a SPARQL query (ii)

- Types of queries:
  - SELECT (DISTINCT/REDUCED)
  - ASK
  - CONSTRUCT
  - DESCRIBE
- Dataset selection / querying
  - FROM
  - FROM NAMED
  - GRAPH

![](_page_56_Picture_0.jpeg)