Lecture 3: SPARQL (1.1)

Aidan Hogan
aidhog@gmail.com
PREVIOUSLY ...
First SPARQL (1.0)
Then SPARQL 1.1
SPARQL Query Language for RDF

W3C Recommendation 15 January 2008


The SPARQL Working Group has produced a W3C Recommendation for a new version of SPARQL which adds features to this 2008 version. Please see SPARQL 1.1 Overview for an introduction to SPARQL 1.1 and a guide to the SPARQL 1.1 document set.

This version:
http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/

Latest version:
http://www.w3.org/TR/rdf-sparql-query/

Previous version:

Editors:
Eric Prud'hommeaux, W3C <eric@w3.org>
Andy Seaborne, Hewlett-Packard Laboratories, Bristol <andy.seaborne@hp.com>
SPARQL: WHERE clause example

Query:

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

What solutions would this query return?

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?title</th>
<th>?date</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado2</td>
<td>&quot;Sharknado 2: The Second One&quot;@en</td>
<td></td>
</tr>
</tbody>
</table>
SPARQL: SELECT with projection

Query:

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

Solutions:

```
?star
  ex:JohnHeard
  ex:IanZiering
  ex:IanZiering
```

DEFAULT "Bag Semantics"

(number of results returned must correspond to number of matches in data)
SPARQL: ASK

Query:

```sparql
PREFIX ex: <http://ex.org/voc#>
ASK
WHERE {
  ?movie a ex:Movie.
}
```

Solutions:

Returns true if there is a match, false otherwise.
SPARQL: CONSTRUCT

Query:

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

Solutions:

```
@prefix ex: <http://ex.org/voc#> . 
```

Returns an RDF graph based on the matching CONSTRUCT clause.
SPARQL: DESCRIBE (optional feature)

Query:

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

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.
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

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
```
Using GRAPH with FROM and FROM NAMED

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?

Solutions:

<table>
<thead>
<tr>
<th>?x</th>
<th>?q</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado2</td>
<td>ex:stars</td>
</tr>
</tbody>
</table>
TODAY: SPARQL 1.1
A recent Web standard

http://www.w3.org/TR/sparql11-query/

SPARQL 1.1 Query Language

W3C Recommendation 21 March 2013

This version:
http://www.w3.org/TR/2013/REC-sparql11-query-20130321/

Latest version:
http://www.w3.org/TR/sparql11-query/

Previous version:
http://www.w3.org/TR/2012/PR-sparql11-query-20121108/

Editors:
Steve Harris, Garlik, a part of Experian
Andy Seaborne, The Apache Software Foundation

Previous Editor:
Eric Prud'hommeaux, W3C

Please refer to the errata for this document, which may include some normative corrections.

See also translations.
What’s new in SPARQL 1.1?

• New query features
• An update language
• Support for RDFS/OWL entailment
• New output formats
NEW QUERY FEATURE:
NEGATION
SPARQL 1.0: Negation possible w/ a trick!

Query:

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

What solutions would this query return?

Solutions:

- ?movie
- ex:Sharknado2

Can do a closed-world style of negation!
SPARQL 1.1: (NOT) EXISTS

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
  ?movie a ex:Movie .
  FILTER NOT EXISTS 
  { ?movie ex:firstAired ?date }
}

Solutions:

?movie
ex:Sharknado2
SPARQL 1.1: MINUS

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
  ?movie a ex:Movie .
  MINUS
  { ?movie ex:firstAired ?date }
}

Solutions:

?movie
ex:Sharknado2
Difference between MINUS and NOT EXISTS?

- **NOT EXISTS**: Returns results if right hand side has no matches
- **MINUS**: Removes solutions from the left hand side that would join with the right hand side
- Very subtle!
Difference between MINUS and NOT EXISTS?
Difference between MINUS and NOT EXISTS?

There is a match!
Therefore no results!

There is no join between the results!
Therefore nothing removed!

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie WHERE {
  ?movie a ex:Movie .
  FILTER NOT EXISTS { ?s a ex:Series }
}

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie WHERE {
  ?movie a ex:Movie .
  MINUS {?s a ex:Series}
}
NEW QUERY FEATURE:
PROPERTY PATHS
Only these features cannot be rewritten to something else. These features are “new”, offering arbitrary length paths!

\[ e \] defined recursively as

- \( p \): a predicate
- \( ^e \): inverse path
- \( e_1/e_2 \): a path of \( e_1 \) followed by \( e_2 \)
- \( e_1|e_2 \): a path of \( e_1 \) or \( e_2 \)
- \( e^* \): a path of zero or more \( e \)
- \( e^+ \): a path of one or more \( e \)
- \( e? \): a path of zero or one \( e \)
- \( !p \): any predicate not \( p \)
- \( !(p_1|\ldots|p_k|^p_{k+1}|\ldots|^p_n) \): any (inverse) predicate not listed
- \( (e) \): brackets used for grouping
Property paths example (over RDF list)

How to ask: “Which movies are in the Sharknado series?”

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
}
```

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
</tr>
<tr>
<td>ex:Sharknado2</td>
</tr>
<tr>
<td>ex:Sharknado3</td>
</tr>
</tbody>
</table>
NEW QUERY FEATURE: ASSIGNMENT
Assignment with BIND

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie ?year
WHERE {
    BIND(xsd:int(SUBSTR(STR(?date),1,4)) AS ?year)
}

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?star</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
<td>2013</td>
</tr>
</tbody>
</table>
Assignment with VALUES

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  VALUES (?movie ?star) {
    (UNDEF ex:JohnHeard)
    (ex:Sharknado2 UNDEF) }
}

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?star</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
<td>ex:JohnHeard</td>
</tr>
<tr>
<td>ex:Sharknado2</td>
<td>ex:IanZiering</td>
</tr>
</tbody>
</table>

No result for ex:Sharknado ex:IanZiering!
NEW QUERY FEATURE: AGGREGATES
Aggregates

How to ask: “How many movie stars are in the data?”
Aggregates: COUNT

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT (COUNT(?star) as ?count)
WHERE {
}
```

Solutions:

```
?count
3
```
Aggregates: COUNT

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT (COUNT(DISTINCT ?star) as ?count)
WHERE {
}
```

Solutions:

<table>
<thead>
<tr>
<th>?count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
How to ask: “How many stars does each movie have?”
Aggregates: COUNT with GROUP BY

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
  (COUNT DISTINCT(?star) as ?count)
WHERE {
}
GROUP BY ?movie

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
<td>2</td>
</tr>
<tr>
<td>ex:Sharknado2</td>
<td>1</td>
</tr>
</tbody>
</table>
Aggregates

How to ask: “Give me movies with more than 1 star?”
Aggregates: COUNT, GROUP BY, HAVING

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
  (COUNT DISTINCT(?star) as ?count)
WHERE {
}
GROUP BY ?movie
HAVING(COUNT DISTINCT(?star) > 1)

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
<td>2</td>
</tr>
</tbody>
</table>

HAVING is like a FILTER for aggregates
Aggregates in SPARQL 1.1

- **COUNT**: Count values
- **SUM**: Sum a set of values
- **MIN**: Find the lowest value
- **MAX**: Find the highest value
- **AVG**: Get the average of values
- **GROUP_CONCAT**: String-concat values
- **SAMPLE**: Select a value (pseudo-randomly)
One more aggregates example: SAMPLE

Query:

```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
    (SAMPLE(?star) as ?aStar)
WHERE {
}
GROUP BY ?movie
HAVING(COUNT DISTINCT(?star) > 1)
```

Solutions:

<table>
<thead>
<tr>
<th>?movie</th>
<th>?aStar</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Sharknado</td>
<td>ex:JohnHeard</td>
</tr>
<tr>
<td>ex:Sharknado</td>
<td>ex:IanZiering</td>
</tr>
</tbody>
</table>
QUICK NOTE ON SEMANTICS
Recall from OWL: OWA and lack of UNA

Open World Assumption (OWA)

How many children does Vito have according to this RDF?

ex:Vito :hasChild ex:Fredo .
... ?

No Unique Name Assumption (No UNA)

How many children does Vito have according to this RDF?

ex:Vito :hasChild ex:Fredo .
... ?
But in SPARQL ...

Looks like SPARQL has a UNA and a CWA ...
But SPARQL does not have “worlds”. It does not interpret “real people”.

SPARQL works on data. SPARQL counts RDF terms, not children.

(IMO, not problematic once this is properly understood by users)

Query:

PREFIX ex: <http://ex.org/voc#>
SELECT (COUNT(?child) as ?count)
WHERE { ex:Vito :hasChild ?child . }

Solutions:

?count
4
NEW QUERY FEATURE: SUBQUERIES
How to ask: “How many stars does a movie have on average?”
Subqueries

**Query:**

```
PREFIX ex: <http://ex.org/voc#>
SELECT (AVG(?count) as ?avg) WHERE {
  {
    SELECT (COUNT DISTINCT(?star) as ?count)
    WHERE {
    }
    GROUP BY ?movie
  }
}
```

**Solutions:**

<table>
<thead>
<tr>
<th>?avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5</td>
</tr>
</tbody>
</table>

Sub-queries useful when you need solution modifiers or aggregates in the middle of a more complex query.
EXTENDED QUERY FEATURE: FUNCTIONS
Lots more functions added

• Includes SPARQL 1.0 features
• Will skim them quickly just to give an idea
  – No need to remember the list but good to know at least what each does and which are included
• More details available at:
  http://www.w3.org/TR/sparql11-query/#SparqlOps
Recall: boolean functions in SPARQL 1.0

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 (!)

\[ !E \rightarrow E \]
SPARQL 1.1 functions (Branching)

- **IF**: If first argument true, return second argument, else return third argument

<table>
<thead>
<tr>
<th>IF expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>?x = 2, &quot;yes&quot;, &quot;no&quot;</td>
<td>returns &quot;yes&quot;</td>
</tr>
<tr>
<td>bound(?y), &quot;yes&quot;, &quot;no&quot;</td>
<td>returns &quot;no&quot;</td>
</tr>
<tr>
<td>?x=2, &quot;yes&quot;, 1/?z</td>
<td>returns &quot;yes&quot;, the expression 1/?z is not evaluated</td>
</tr>
<tr>
<td>?x=1, &quot;yes&quot;, 1/?z</td>
<td>raises an error</td>
</tr>
<tr>
<td>&quot;2&quot; &gt; 1, &quot;yes&quot;, &quot;no&quot;</td>
<td>raises an error</td>
</tr>
</tbody>
</table>

- **COALESCE**: Return first non-error argument

<table>
<thead>
<tr>
<th>COALESCE expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>?x, 1/0</td>
<td>returns 2, the value of x</td>
</tr>
<tr>
<td>1/0, ?x</td>
<td>returns 2</td>
</tr>
<tr>
<td>5, ?x</td>
<td>returns 5</td>
</tr>
<tr>
<td>?y, 3</td>
<td>returns 3</td>
</tr>
<tr>
<td>?y</td>
<td>raises an error because y is not bound.</td>
</tr>
</tbody>
</table>
SPARQL: (In)equality FILTER operators

- \( =, \neq, \text{SAMETERM}(A,B) \)
  - \( = \) and \( \neq \) test value (in)equality
  - \text{SAMETERM}\) tests term equality
    - e.g., \(2.0\)~\text{xsd:decimal} = \(2\)~\text{xsd:int}\) gives true
    - \text{SAMETERM}(\(2.0\)~\text{xsd:decimal}, \(2\)~\text{xsd:int})\) gives false

- \( >, <, \geq, \leq \)
  - can only compare “compatible” types
    - e.g., \(2.0\)~\text{xsd:decimal} > \(2\)~\text{xsd:int}\) okay, \(2.0\)~\text{xsd:decimal} > \(2\) an error
Lots more functions (Checking values)

- **IN**: Returns true if left-hand term is a member of right-hand list

<table>
<thead>
<tr>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 IN (1, 2, 3)</td>
<td>true</td>
</tr>
<tr>
<td>2 IN ()</td>
<td>false</td>
</tr>
<tr>
<td>2 IN (<a href="http://example/iri">http://example/iri</a>, &quot;str&quot;, 2.0)</td>
<td>true</td>
</tr>
<tr>
<td>2 IN (1/0, 2)</td>
<td>true</td>
</tr>
<tr>
<td>2 IN (2, 1/0)</td>
<td>true</td>
</tr>
<tr>
<td>2 IN (3, 1/0)</td>
<td>raises an error</td>
</tr>
</tbody>
</table>

- **NOT IN**: Same as above but NOT in 😊
Recall: RDF term functions in SPARQL 1.0

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 1.1 functions (RDF Terms)

- **ISNUMERIC**: Is a term a valid numeric term?
  
<table>
<thead>
<tr>
<th>Term</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNumeric(12)</td>
<td>true</td>
</tr>
<tr>
<td>isNumeric(&quot;12&quot;)</td>
<td>false</td>
</tr>
<tr>
<td>isNumeric(&quot;12&quot;^^xsd:nonNegativeInteger)</td>
<td>true</td>
</tr>
<tr>
<td>isNumeric(&quot;1200&quot;^^xsd:byte)</td>
<td>false</td>
</tr>
<tr>
<td>isNumeric(<a href="http://example/">http://example/</a>)</td>
<td>false</td>
</tr>
</tbody>
</table>

- **IRI**: create an IRI from a string
- **BNODE**: create a new blank node
- **STRDT**: create a new datatype literal
- **STRLANG**: create a new language-typed literal
- **UUID**: create a fresh IRI (in uuid scheme)
- **STRUUID**: create a fresh UUID string
Recall: String functions in SPARQL 1.0

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
• `STRLLEN"abc") = 3
• `STRSUB"abc",3,1) = "c"
• `UCASE"shout") = "SHOUT"
• `LCASE"WHISPER") = "whisper"
• `STRSTARTS"asd","as") = true
• `STRENDS"asd","sd") = true
• `CONTAINS("WHISPER","HIS") = true
• `STRBEFORE"abc","b") = "a"
• `STRAFTER"abc","b") = "c"
• `ENCODE_FOR_URI"a c") = "a%20c"
• `CONCAT"shi","p") = "ship"
• `REPLACE"ship","p","n") = "shin"
Recall: RDF term functions in SPARQL 1.0

SPARQL: Numeric FILTER operators

- $+A$, $-A$, $A+B$, $A-B$, $A\times B$, $A/B$ (numeric)
  - input numeric, output numeric
SPARQL 1.1 functions (Numerics)

- **ABS**\((-3.2) = 3.2\)
- **ROUND**\((2.5) = 3.0\)
- **CEIL**\((-2.5) = -2.0\)
- **FLOOR**\((-2.5) = -3.0\)
- **RAND() = 0.5612381239123 \((0 \geq n > 1)\)
SPARQL 1.1 functions (Datetimes)

- **NOW()** = "2015-10-21T02:12:14-04:00"^^xsd:dateTime
- **YEAR("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 2015**
- **MONTH("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 10**
- **DAY("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 21**
- **HOURS("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 02**
- **MINUTES("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 12**
- **SECONDS("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = 14**
- **TIMEZONE("2015-10-21T02:12:14-04:00"^^xsd:dateTime)**
  = "-PT4H"^^xsd:dayTimeDuration
- **TZ("2015-10-21T02:12:14-04:00"^^xsd:dateTime) = "-04:00"**
SPARQL 1.1 functions (Datetimes)

• `NOW()` = "2015-10-21T02:12:14-04:00"^^xsd:dateTime
SPARQL 1.1 functions (Hashes)

• Creates a hash of the input string
  – MD5
  – SHA1
  – SHA256
  – SHA384
  – SHA512
NEW QUERY FEATURE:
FEDERATION
Endpoints often made public/online

```
select distinct ?Concept where {{} a ?Concept} LIMIT 100
```

(Virtuoso SPARQL Query Editor, Default Data Set Name: http://dbpedia.org, Results Format: HTML, Execution timeout: 30000 milliseconds, Strict checking of void variables option checked.)
Federation: execute sub-query remotely

PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?actor_name ?birth_date
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf> # placeholder graph
WHERE {
  {
    SERVICE <http://data.linkedmdb.org/sparql> {
      ?actor movie:actor_name ?actor_name
    }
    BIND(STRLANG(?actor_name, "en") AS ?actor_name_en)
  }
  SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a foaf:Person ; foaf:name ?actor_name_en ;
    dbpedia:birthDate ?birth_date .
  }
}

Get actors for Star Trek movie from LinkedMDB. Use DBpedia to get the birthdate of the actor

Can be run at http://sparql.org/sparql

Example borrowed from: http://www.cambridgesemantics.com/semantic-university/sparql-by-example
NEW FEATURE:
SPARQL 1.1 UPDATE
What’s new in SPARQL 1.1?

• **New query features**
• An update language
• Support for RDFS/OWL entailment
• New output formats
PREFIX ex: <http://ex.org/voc#>
INSERT DATA {
}
PREFIX ex: <http://ex.org/voc#>
INSERT DATA {
  GRAPH ex:Sharknado2.ttl
  { ex:Sharknado2 ex:firstAired "2014-07-30"^^xsd:date . }}

DELETE DATA

PREFIX ex: <http://ex.org/voc#>
DELETE DATA {
}

PREFIX ex: <http://ex.org/voc#>
DELETE DATA {
  GRAPH ex:Sharknado2.ttl
    { ex:Sharknado2 ex:firstAired "2014-07-30"^^xsd:date . }
}

DELETE DATA
INSERT/DELETE with WHERE

PREFIX ex: <http://ex.org/voc#>
INSERT {
    GRAPH ?g { ?movie ex:description "2nd Sharknado Movie" . }
}
WHERE {
    GRAPH ?g { ?movie ?p ?o }
}

PREFIX ex: <http://ex.org/voc#>
DELETE {
    GRAPH ?g { ?movie ex:title ?title . }
}
WHERE {
    GRAPH ?g { ?movie ex:title ?title . }
}
Combining INSERT/DELETE

PREFIX ex: <http://ex.org/voc#>
DELETE {
    GRAPH ?g { ?movie ex:description ?olddescription . } }
INSERT {
    GRAPH ?g { ?movie ex:description "Best of the series" . }
}
WHERE {
    GRAPH ?g { ?movie ex:description ?olddescription . }
}
Set default update graph: WITH

PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE {
}
INSERT {
    GRAPH ex:Sharknado { ex:Sharknado ex:sequel ?movie }
}
WHERE {
}
Simple DELETE WHERE

PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE WHERE {
}

Equivalent to ...

PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE {
}
WHERE {
}
Managing named graphs: LOAD

• LOAD a graph from the Web

```
LOAD ( SILENT )? IRIref_from ( INTO GRAPH IRIref_to )?
```

– **SILENT**: If load fails, suppress error
– **IRIref_from**: location of graph online
– **IRIref_to**: local named graph to load into
  • (If INTO GRAPH IRIref_to not given, default graph will be used)

If destination graph exists, data will be appended. Will fail if RDF cannot be extracted from source graph (unless silent is specified).
Managing named graphs: CLEAR

- **CLEAR** all triples from some graph(s)

```
CLEAR ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )
```

- **SILENT**: If clear fails, suppress error
- **GRAPH IRIref**: clear specific named graph
- **DEFAULT**: clear default graph
- **NAMED**: clear all named graphs
- **ALL**: clear all graphs

Will fail if graph does not exist (unless silent is specified, in which case nothing happens).
Managing named graphs: CREATE

• CREATE a new blank named graph

  CREATE ( SILENT )? GRAPH IRIref

  – SILENT: If create fails, suppress error
  – GRAPH IRIref: name of graph to create

Will fail if graph already exists (unless silent is specified). Existing graphs cannot be affected.
Managing named graphs: DROP

- **DROP** (remove) some graph(s)

  
  ```
  DROP ( SILENT )? (GRAPH IRIref | DEFAULT | NAMED | ALL )
  ```

  - **SILENT**: If drop fails, suppress error
  - **GRAPH IRIref**: name of graph to drop
  - **DEFAULT**: drop default graph
  - **NAMED**: drop all named graphs
  - **ALL**: drop all graphs

  Fails if graph does not exist (unless silent is specified). An engine must have a default graph so actually DROP DEFAULT same as CLEAR DEFAULT.
Managing named graphs: COPY

• COPY one graph to another

COPY ( SILENT )? ( ( GRAPH )? IRIref_from | DEFAULT ) TO ( ( GRAPH )? IRIref_to | DEFAULT )

– **SILENT**: If copy fails, suppress error
– **IRIref_from**: name of graph to copy from
– **IRIref_to**: name of graph to copy to
– **DEFAULT**: copy from/to default graph

May fail if source graph does not exist (unless silent is specified). Destination graph will be created or cleared before the copy is done.
Managing named graphs: MOVE

• MOVE one graph to another

MOVE (SILENT)? ( ( GRAPH )? IRIref_from | DEFAULT) TO ( ( GRAPH )? IRIref_to | DEFAULT)

- SILENT: If move fails, suppress error
- IRIref_from: name of graph to move
- IRIref_to: name of graph to move to
- DEFAULT: move from/to default graph

May fail if source graph does not exist (unless silent is specified). Destination graph will be created or cleared before the copy is done. Source graph dropped after the move.
Managing named graphs: ADD

- **ADD data from one graph to another**

```
ADD ( SILENT )? ( ( GRAPH )? IRIref_from | DEFAULT ) TO ( ( GRAPH )? IRIref_to | DEFAULT )
```

- **SILENT**: If move fails, suppress error
- **IRIref_from**: name of graph to move
- **IRIref_to**: name of graph to move to
- **DEFAULT**: move from/to default graph

May fail if source graph does not exist (unless silent is specified). Destination graph will be created if it does not exist (it will not be cleared if it does). Source graph unaffected.
NEW FEATURE:
SPARQL 1.1 ENTAILMENT REGIMES
What’s new in SPARQL 1.1?

• **New query features**
• **An update language**
• Support for RDFS/OWL entailment
• **New output formats**
SPARQL 1.1 Entailment Regimes

• States how entailments can be included in SPARQL results
• Support for RDFS / sublanguages of OWL
• Not well supported (to best of my knowledge)
• Not going to cover it
• If interested, check out
  – http://www.w3.org/TR/sparql11-entailment/
NEW FEATURE:
SPARQL 1.1 OUTPUT FORMATS
SPARQL 1.1 Output Formats

• SELECT, ASK (non RDF):
  – XML (1.0), JSON (1.1), CSV/TSV (1.1)

• CONSTRUCT, DESCRIBE (RDF)
  – Standard RDF syntaxes: RDF/XML, Turtle, etc.
QUICK MENTION: SPARQL 1.1 PROTOCOL
Defines a HTTP protocol

• How to issue queries/update over HTTP
  – GET / POST

• How different output formats can be requested
  – Accept: text/turtle, application/rdf+xml

• What response codes should be returned; e.g.
  – 200 if successful
  – 400 if SPARQL query is invalid
  – 500 if query was okay but server failed to answer

• ... etc. See more details:
  – http://www.w3.org/TR/sparql11-protocol/
Can host SPARQL endpoints on the Web!

```
select distinct ?Concept where {{} a ?Concept} LIMIT 100
```
RECAP
SPARQL 1.1 New Query Features

• **Negation shortcuts**
  – Do negation checks or set difference

• **Property paths**
  – Query arbitrary length paths

• **Assignment**
  – Assign values to variables statically or from functions

• **Aggregates**
  – Compute one value from multiple (possibly grouped)

• **Subqueries**
  – Nest SELECT queries inside the WHERE clause

• **Lots of new functions**
  – For strings, numerics, dates, branching, etc.
SPARQL 1.1 Update

• **INSERT DATA/DELETE DATA**
  – For static data

• **INSERT/DELETE with WHERE**
  – For data generated from query results

• **WITH**
  – Specify the default update graph

• **LOAD/CLEAR/CREATE/DROP/COPY/MOVE/ADD**
  – Manage default/named graphs
La evaluación del curso

• Aidan
  – 10% Lab 1
    • Antes que el viernes, 18:00
  – 20% Lab 2 (la versión extendida)
    • Antes que el viernes, 18:00
  – 20% Tarea (con Wikidata; individual)
    • Antes que el lunes (25 de julio), 18:00

• Pablo
  – 50% Tarea (él va a darles más detalles)
Tarea: Crear consultas originales sobre Wikidata

- Aquí hay el servicio público de Wikidata usando SPARQL: [http://query.wikidata.org/](http://query.wikidata.org/)
- Se puede ver ejemplos allá
- Hay que crear cuatro consultas *originales* (no copiar y pegar ejemplos)
- Hay que someter tu tarea solo
- Las consultas deberían utilizar:
  - OPTIONAL, UNION, ORDER BY (SPARQL 1.0)
  - Property paths, GROUP BY (SPARQL 1.1)
  - (Es decir, hay que utilizar cada rasgo al menos una vez en las cuatro consultas, no todos en cada consulta)
- Puedes crear las consultas en cualquier dominio
Questions?