

**CC7220-1**

**LA WEB DE DATOS**

**PRIMAVERA 2021**

## **LECTURE 8: SPARQL [1.1]**

Aidan Hogan

aidhog@gmail.com

PREVIOUSLY ...



SPARQL (1.1)

First SPARQL (1.0)  
Then SPARQL 1.1

# COVERED SPARQL 1.0



<http://www.w3.org/TR/rdf-sparql-query/>

## SPARQL Query Language for RDF

W3C Recommendation 15 January 2008

**New Version Available: SPARQL 1.1 (Document Status Update, 26 March 2013)**

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:**

<http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/>

**Editors:**

Eric Prud'hommeaux, W3C <[eric@w3.org](mailto:eric@w3.org)>

Andy Seaborne, Hewlett-Packard Laboratories, Bristol <[andy.seaborne@hp.com](mailto:andy.seaborne@hp.com)>

TODAY:

SPARQL 1.1

# A 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:**

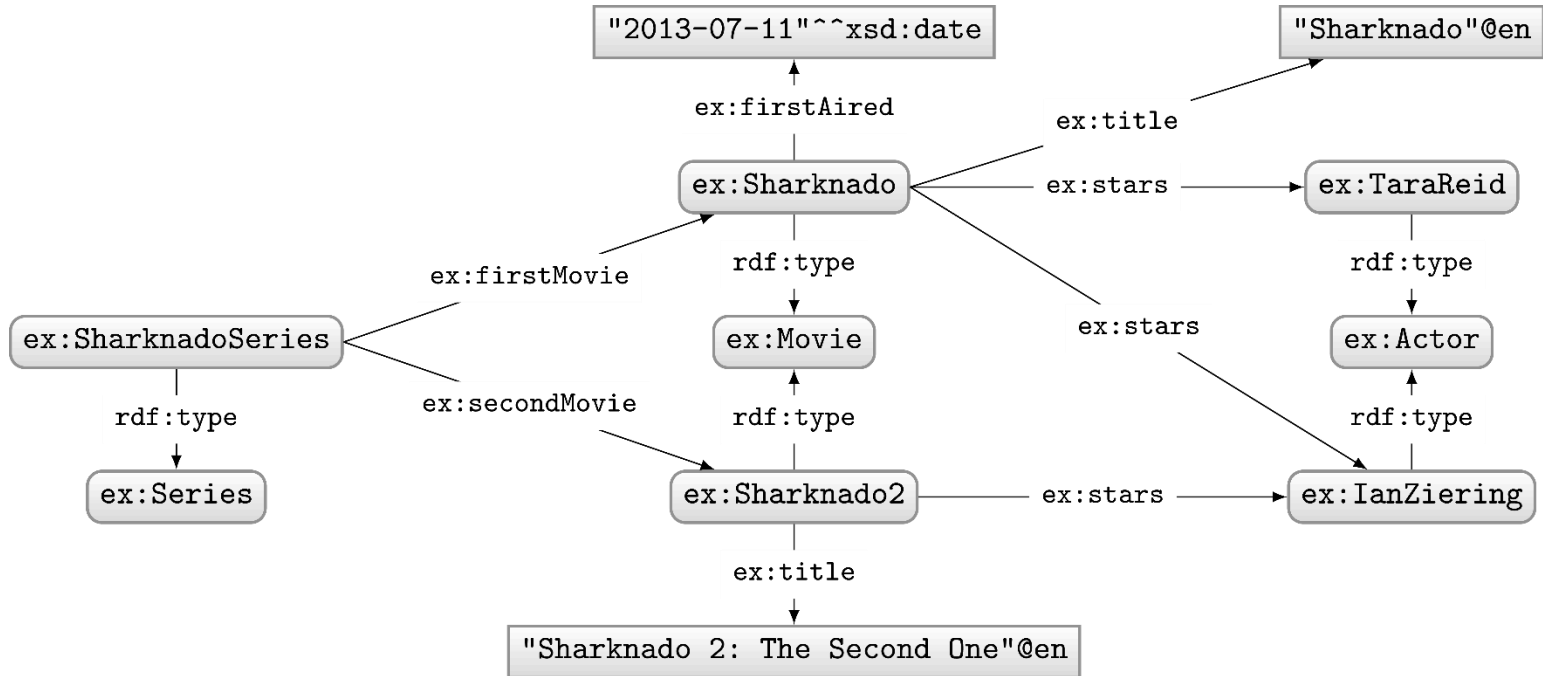
Eric Prud'hommeaux, W3C

Please refer to the [errata](#) for this document, which may include some normative corrections.

See also [translations](#).

QUERY FEATURE:  
NEGATION

# SPARQL 1.0: NEGATION POSSIBLE W/ A TRICK!



Query:

```
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?

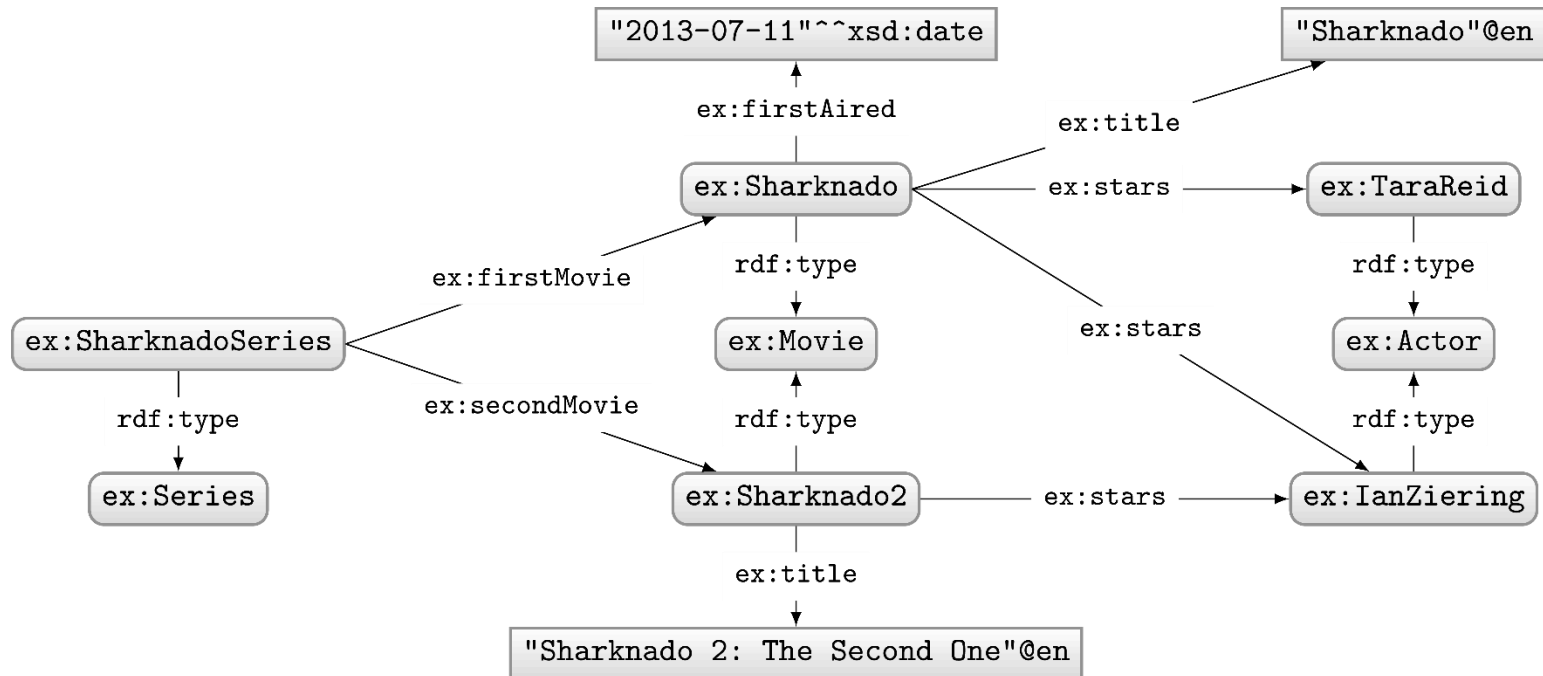
## Solutions:

?movie	?date
ex:Sharknado2	

Can do a closed-world style of negation!



# SPARQL 1.1: (NOT) EXISTS



Query:

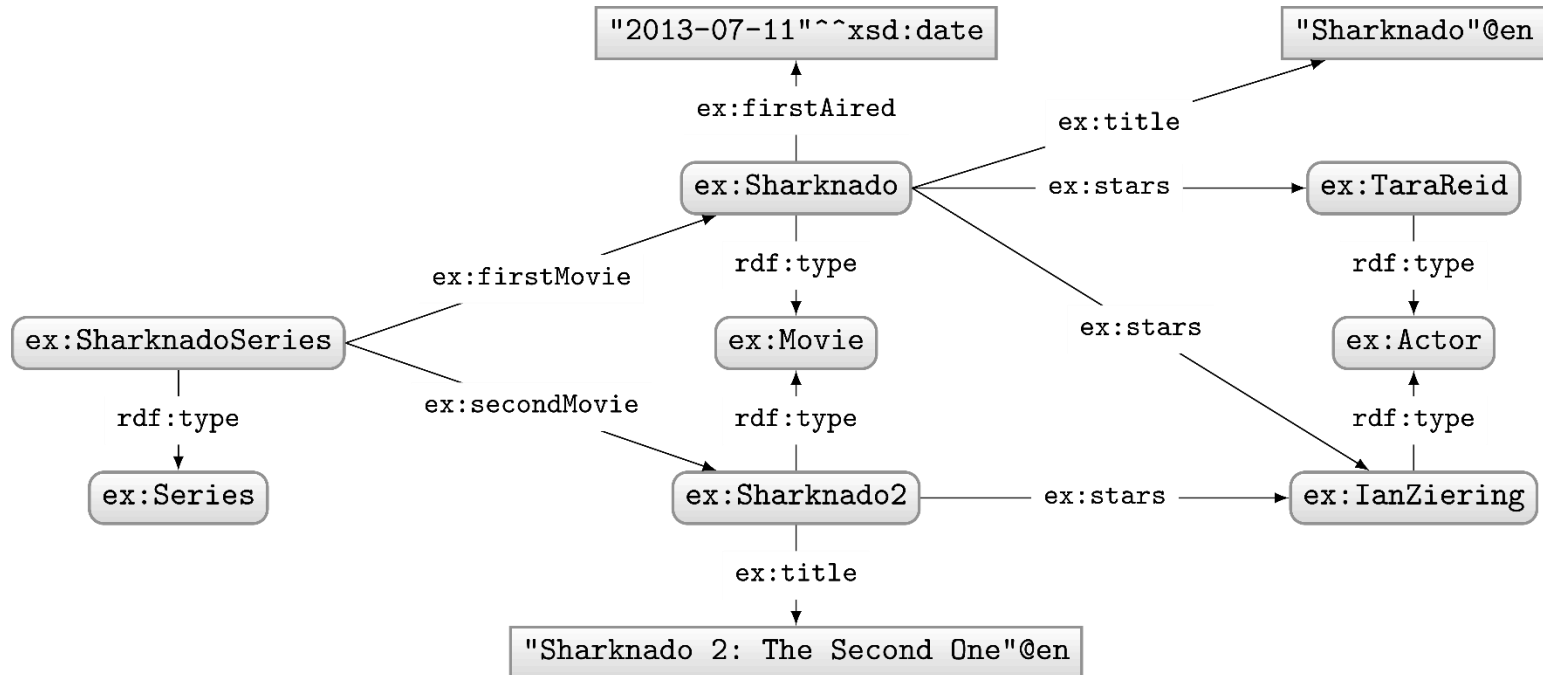
```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
    ?movie a ex:Movie .
    FILTER NOT EXISTS
        { ?movie ex:firstAired ?date }
}
```

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?



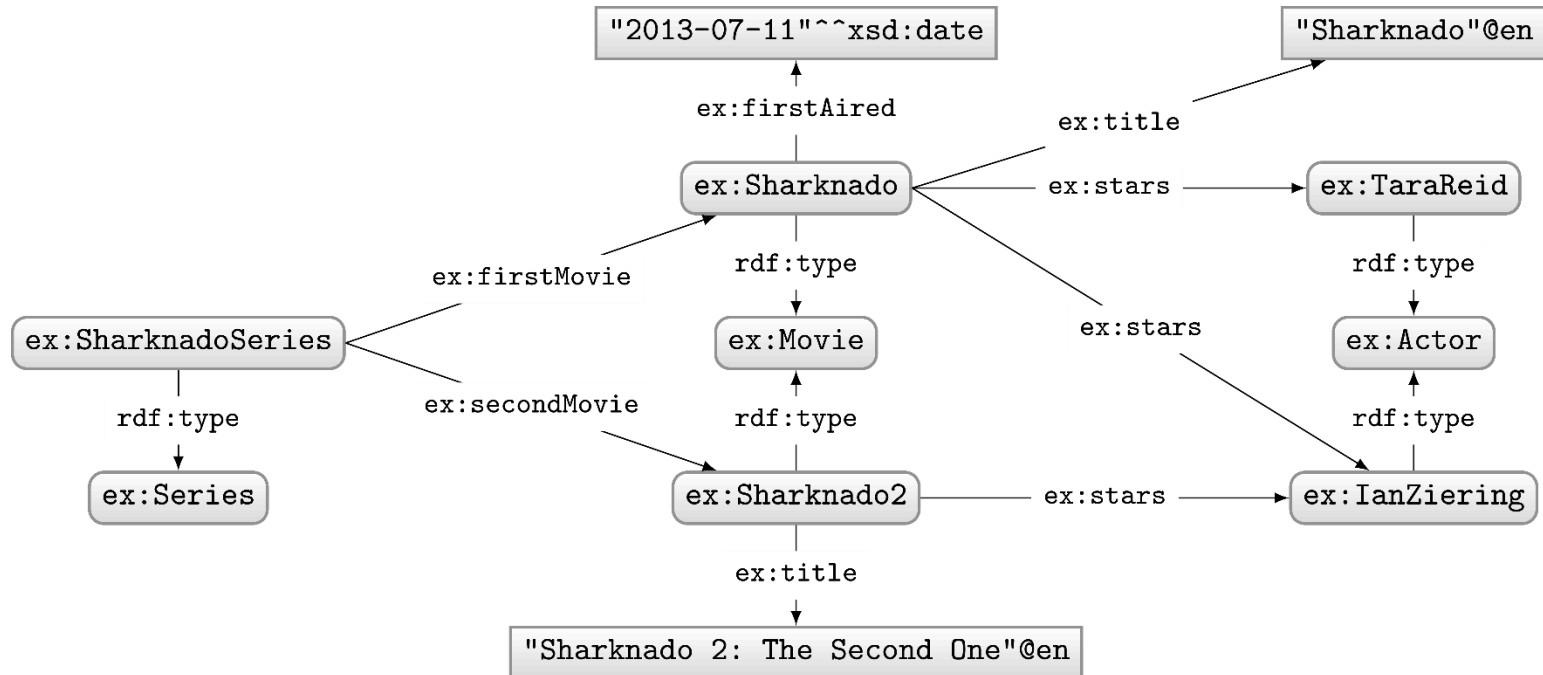
# DIFFERENCE BETWEEN MINUS AND NOT EXISTS?

- **NOT EXISTS**: Returns results if the pattern on the right has no matches when replacing variables from the left (actually not well-defined)
- **MINUS**: Removes solutions from the left that join on the right (with at least one variable)





# DIFFERENCE BETWEEN MINUS AND NOT EXISTS?



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

?movie

There is a match!  
Therefore no results!

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

?movie

ex:Sharknado  
ex:Sharknado2

There is no join  
variable between both!  
Therefore nothing removed!

NEW QUERY FEATURE:  
PROPERTY PATHS

# PROPERTY PATHS: REGULAR EXPRESSIONS

Only these features cannot be rewritten to something else.  
These features are “new”, offering arbitrary length paths!

---

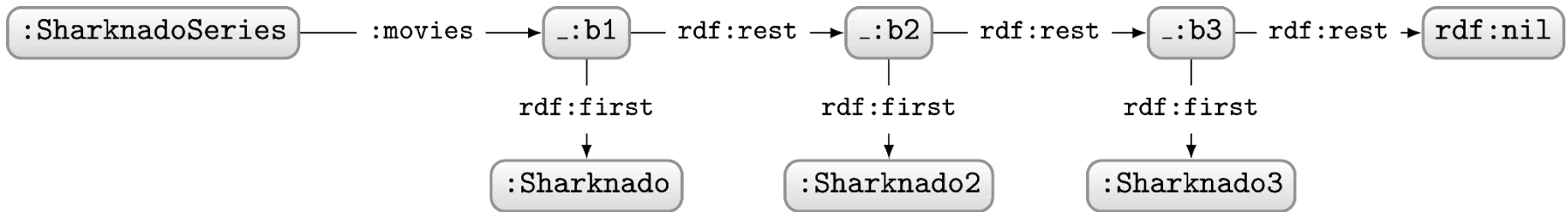
$e$  defined recursively as

---

$p$	a predicate
$\hat{e}$	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 $p$
$!(p_1 \dots p_k \hat{p}_{k+1} \dots \hat{p}_n)$	any (inverse) predicate not listed
$(e)$	brackets used for grouping

---

# PROPERTY PATHS EXAMPLE: RDF LIST



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

Query:

```
PREFIX : <http://ex.org/voc#>
SELECT ?movie
WHERE {
    :SharknadoSeries :movies/rdf:rest*/rdf:first ?movie .
}
```

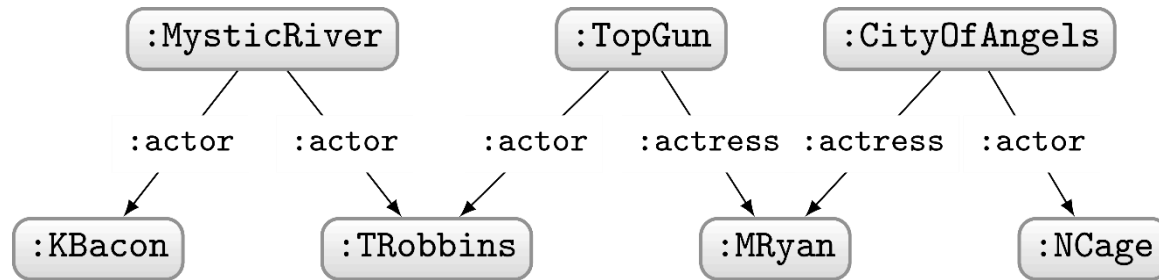
Solutions:

?movie

:Sharknado  
:Sharknado2  
:Sharknado3



# PROPERTY PATHS EXAMPLE: FINITE BACON NUMBER



How to ask: “Who has a finite Bacon number?”

Query:

```
PREFIX : <http://ex.org/voc#>
SELECT ?star
WHERE {
    :KBacon ((^:actor|^:actress)/(:actor|:actress))* ?star .
}
```

Solutions:

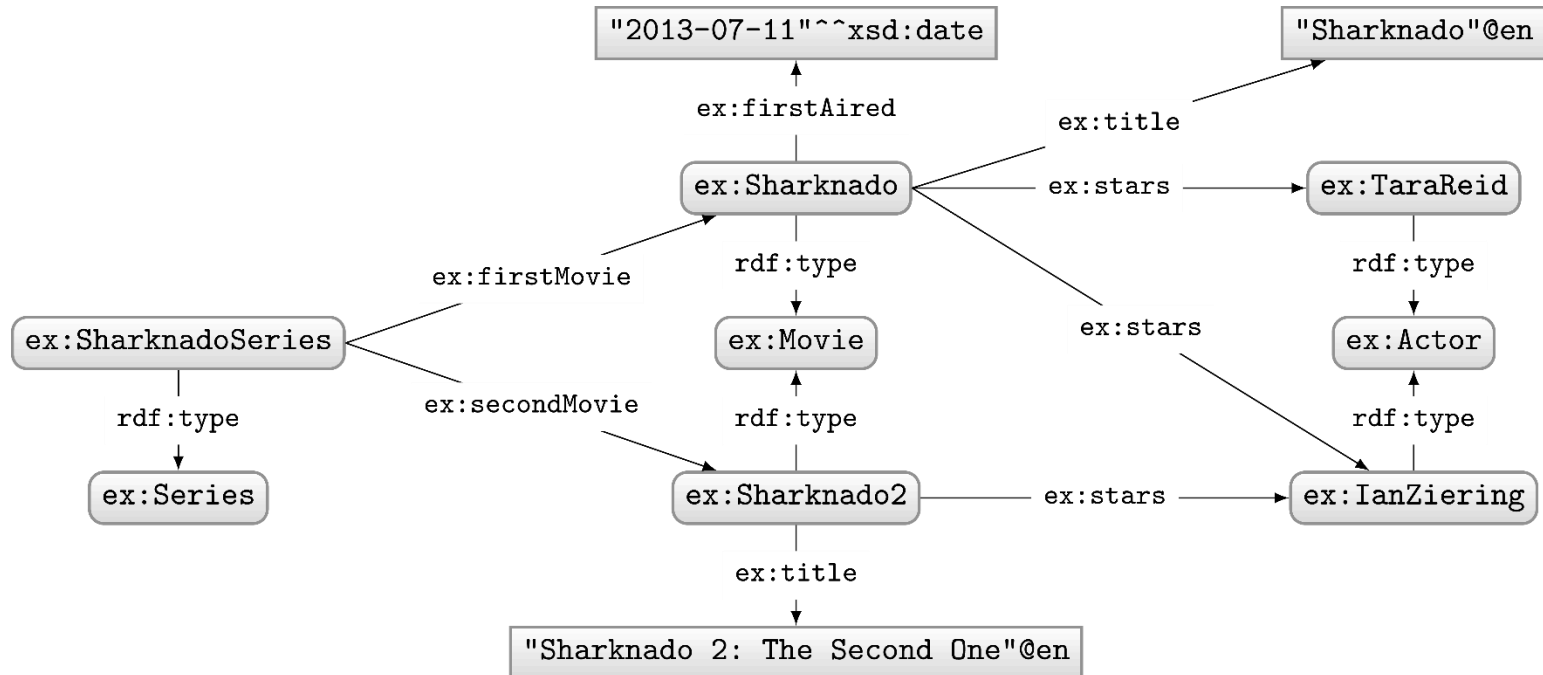
?star

:KBacon  
:TRobbins  
:MRyan  
:NCage

We cannot get the actual Bacon number  
(path length) for arbitrary length paths

NEW QUERY FEATURE:  
ASSIGNMENT

# ASSIGNMENT WITH BIND



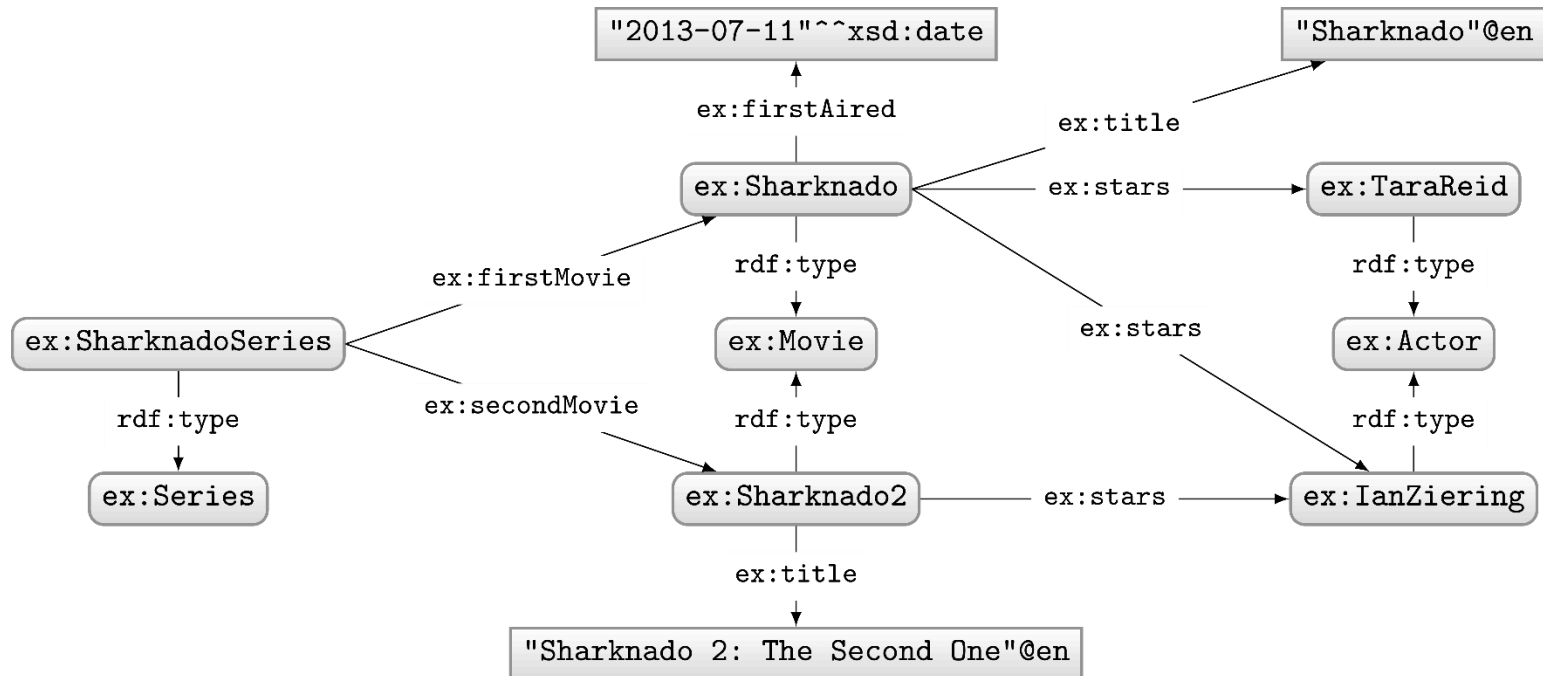
Query:

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

Solutions:

?movie	?year
ex:Sharknado	2013

# ASSIGNMENT WITH VALUES



Query:

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

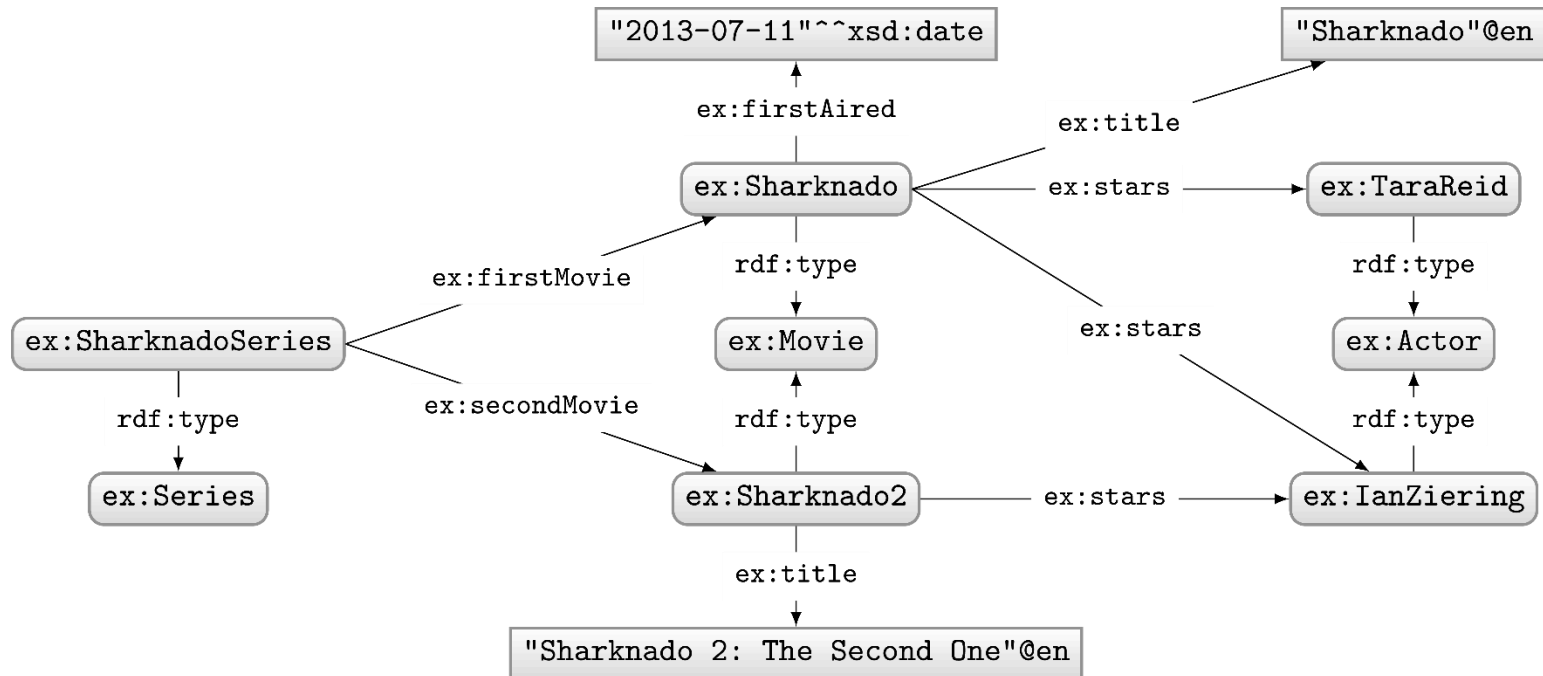
Solutions:

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

No result for  
ex:Sharknado ex:IanZiering!

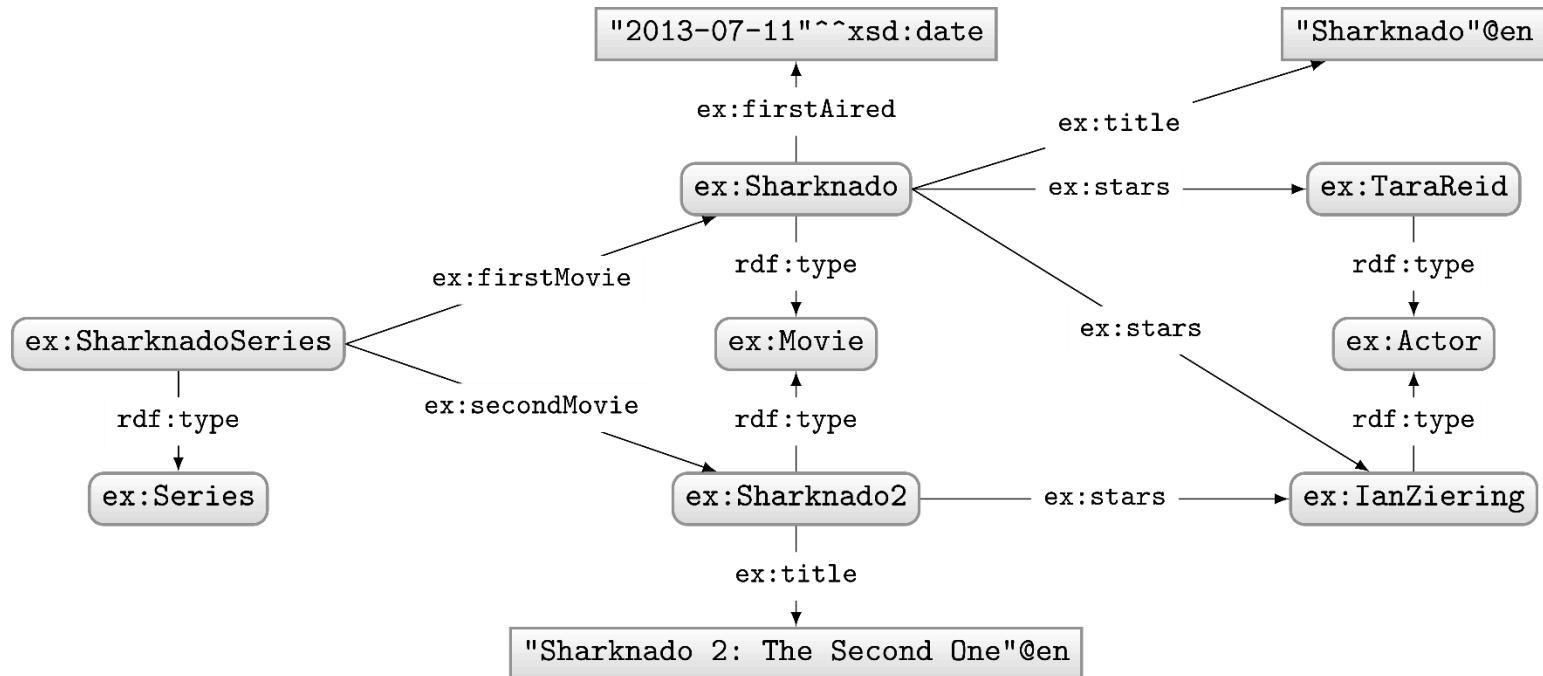
NEW QUERY FEATURE:  
AGGREGATES

# AGGREGATES



How to ask: "How many movie stars are in the data?"

# AGGREGATES: COUNT



Query:

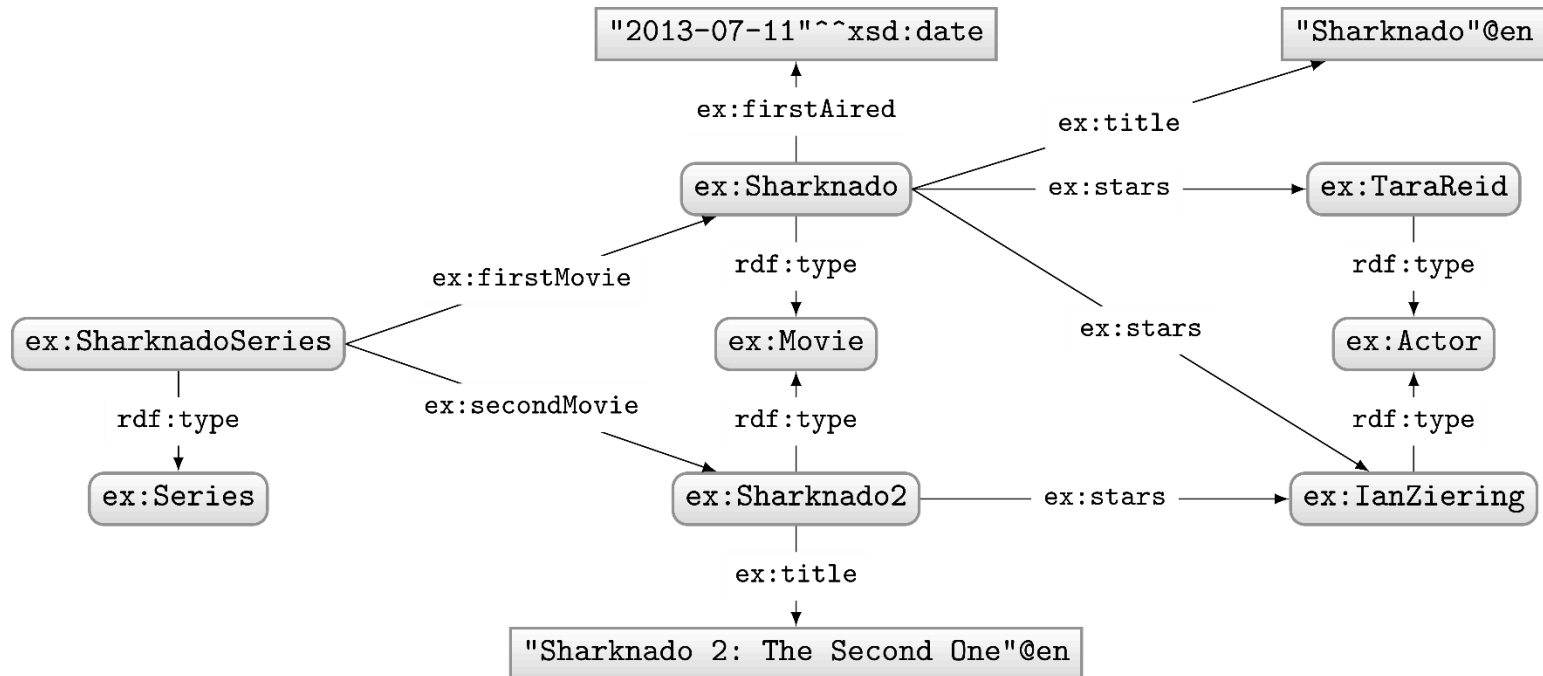
```
PREFIX ex: <http://ex.org/voc#>
SELECT (COUNT(?star) as ?count)
WHERE {
    ?movie ex:stars ?star .
}
```

Solutions:

?count
--------

3
---

# AGGREGATES: COUNT



Query:

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

Solutions:

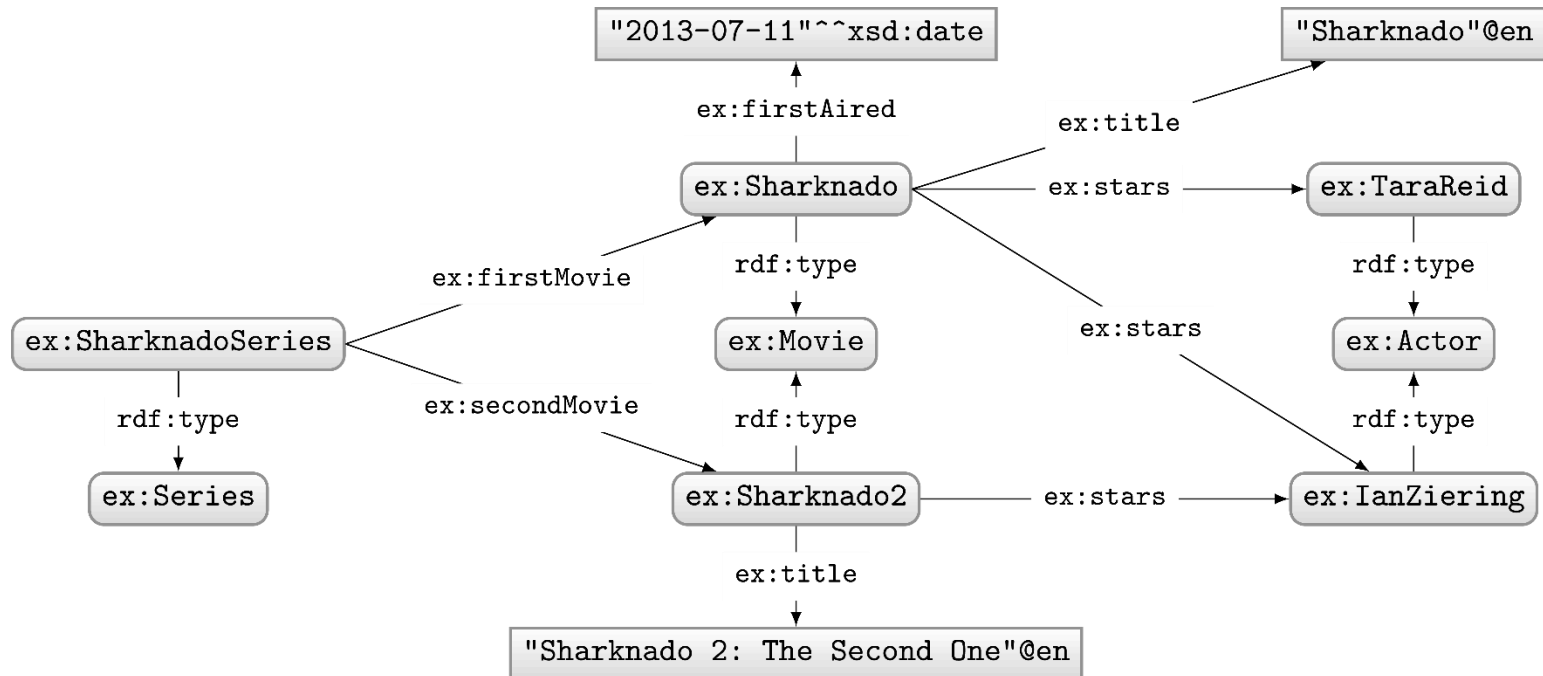
?count

3

**DISTINCT applied after COUNT!**



# AGGREGATES: COUNT



Query:

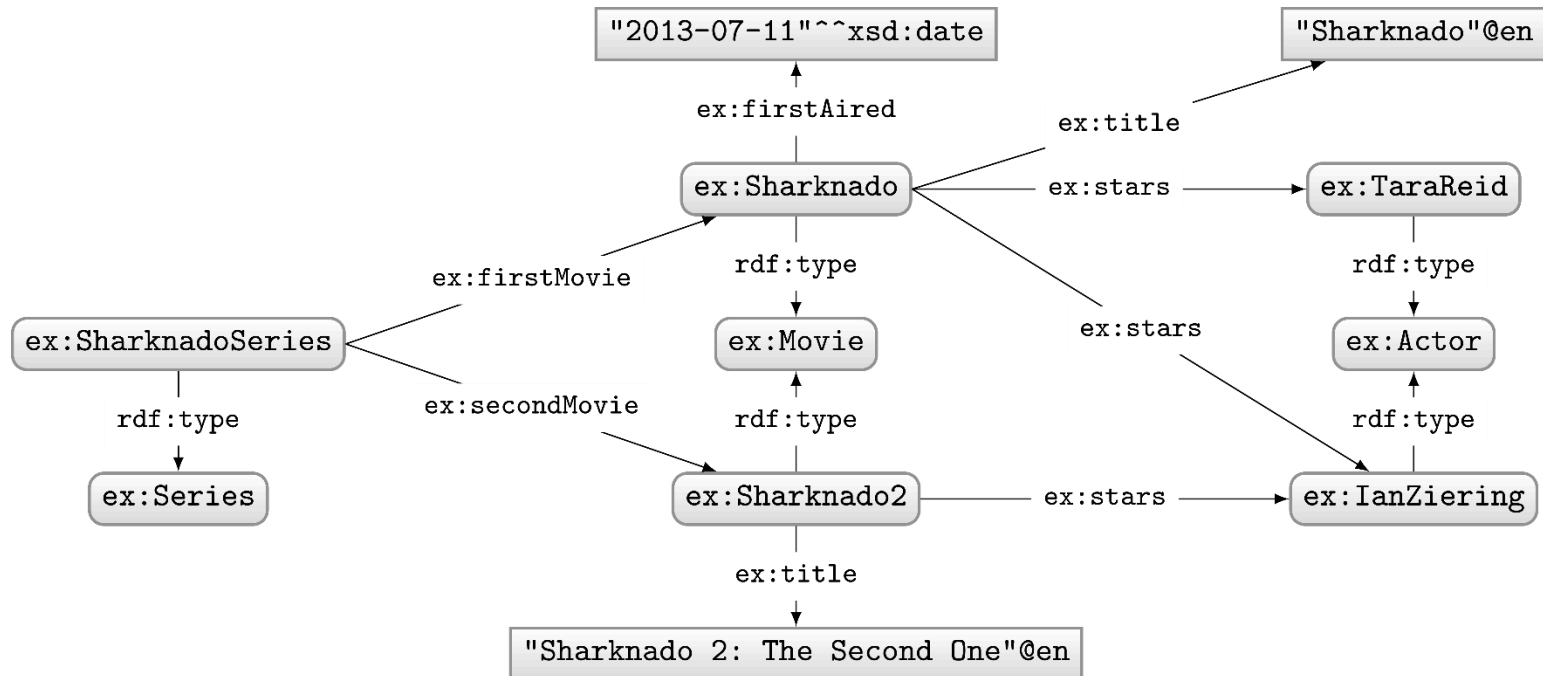
```
PREFIX ex: <http://ex.org/voc#>
SELECT (COUNT(DISTINCT ?star) as ?count)
WHERE {
  ?movie ex:stars ?star .
}
```

Solutions:

?count

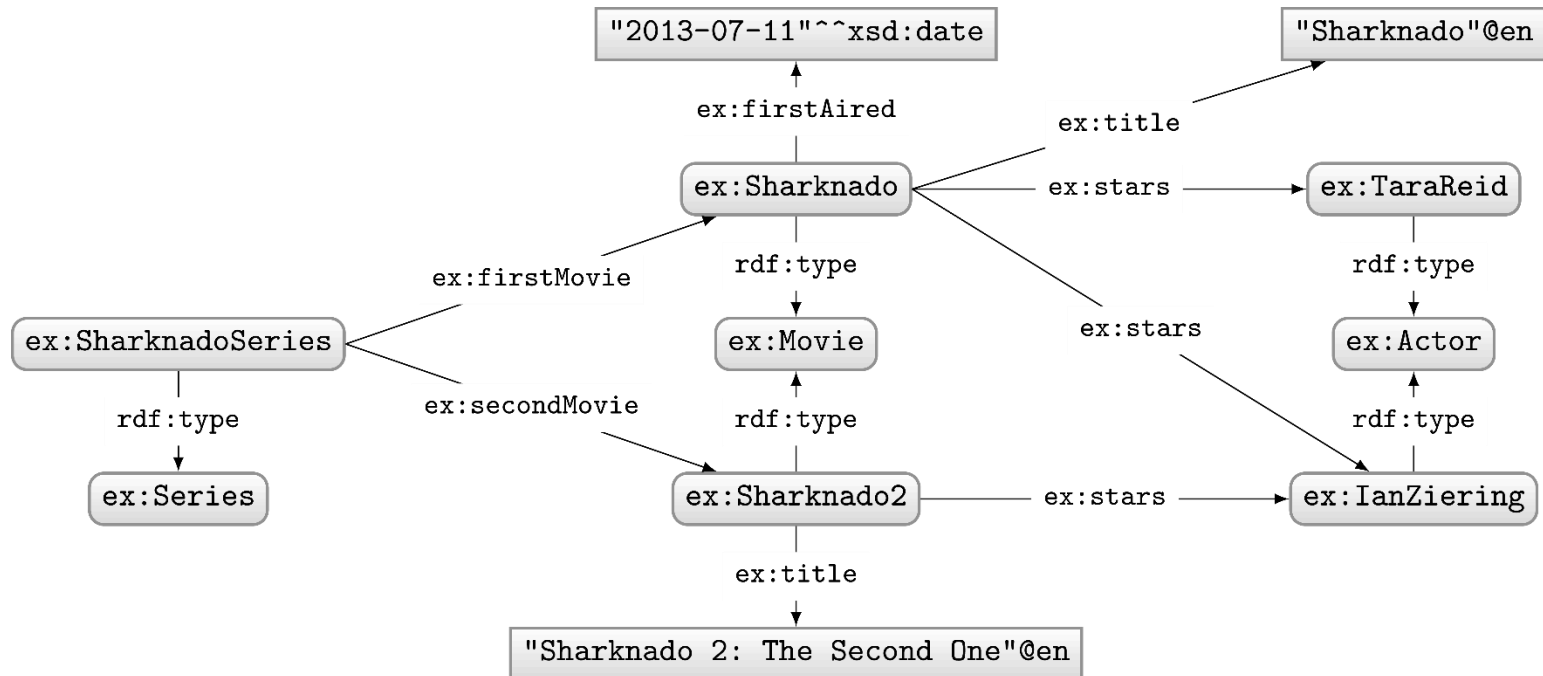
2

# AGGREGATES



How to ask: "How many stars does each movie have?"

# AGGREGATES: COUNT WITH GROUP BY



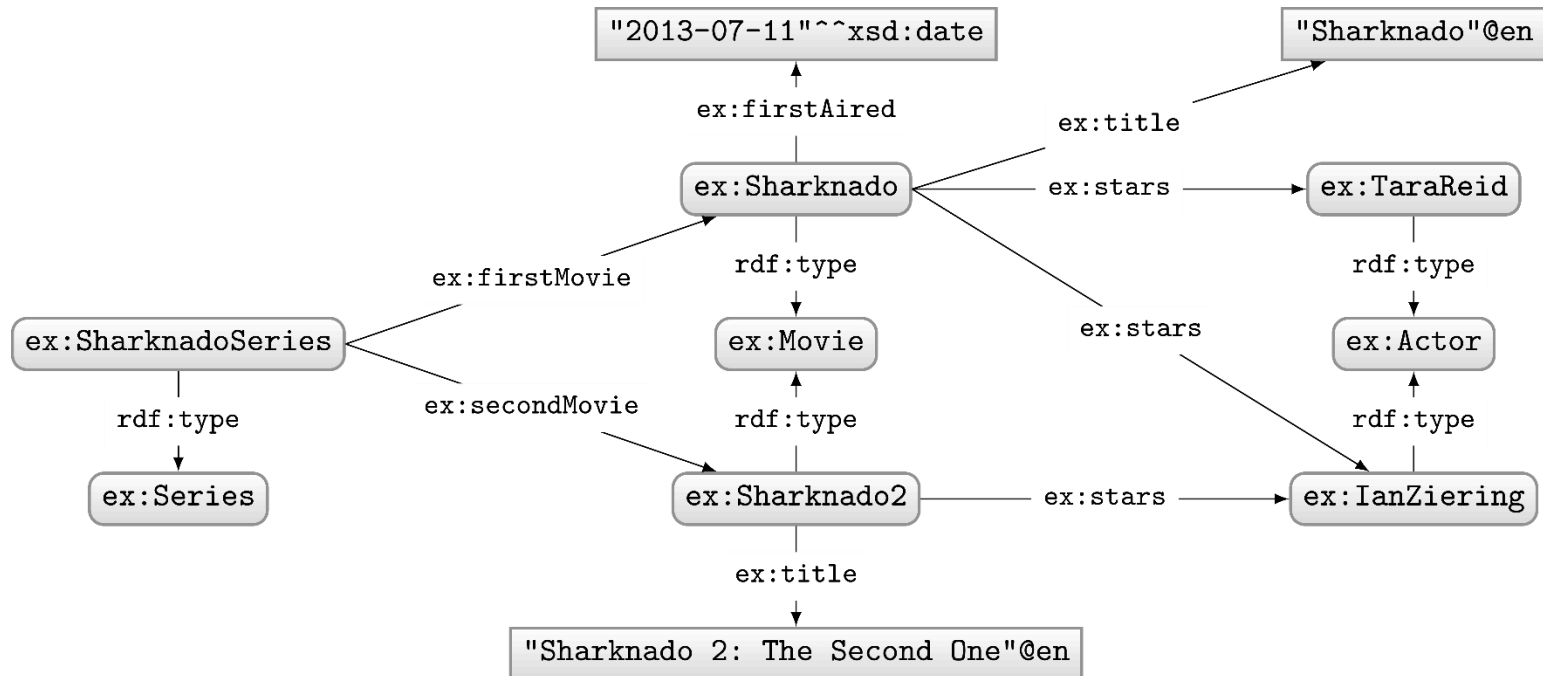
Query:

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

Solutions:

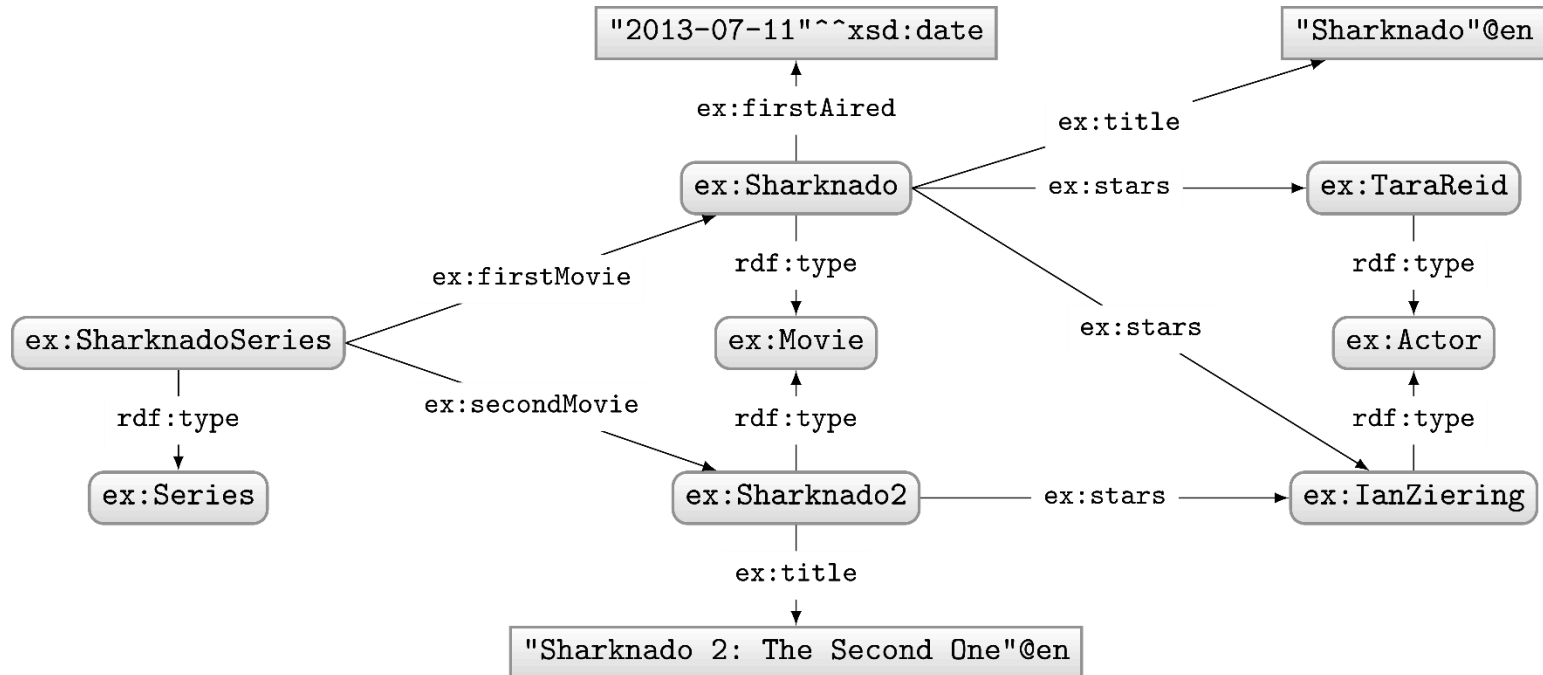
?movie	?count
ex:Sharknado	2
ex:Sharknado2	1

# 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 {
  ?movie ex:stars ?star .
}
GROUP BY ?movie
HAVING(COUNT(DISTINCT ?star) > 1)
```

Solutions:

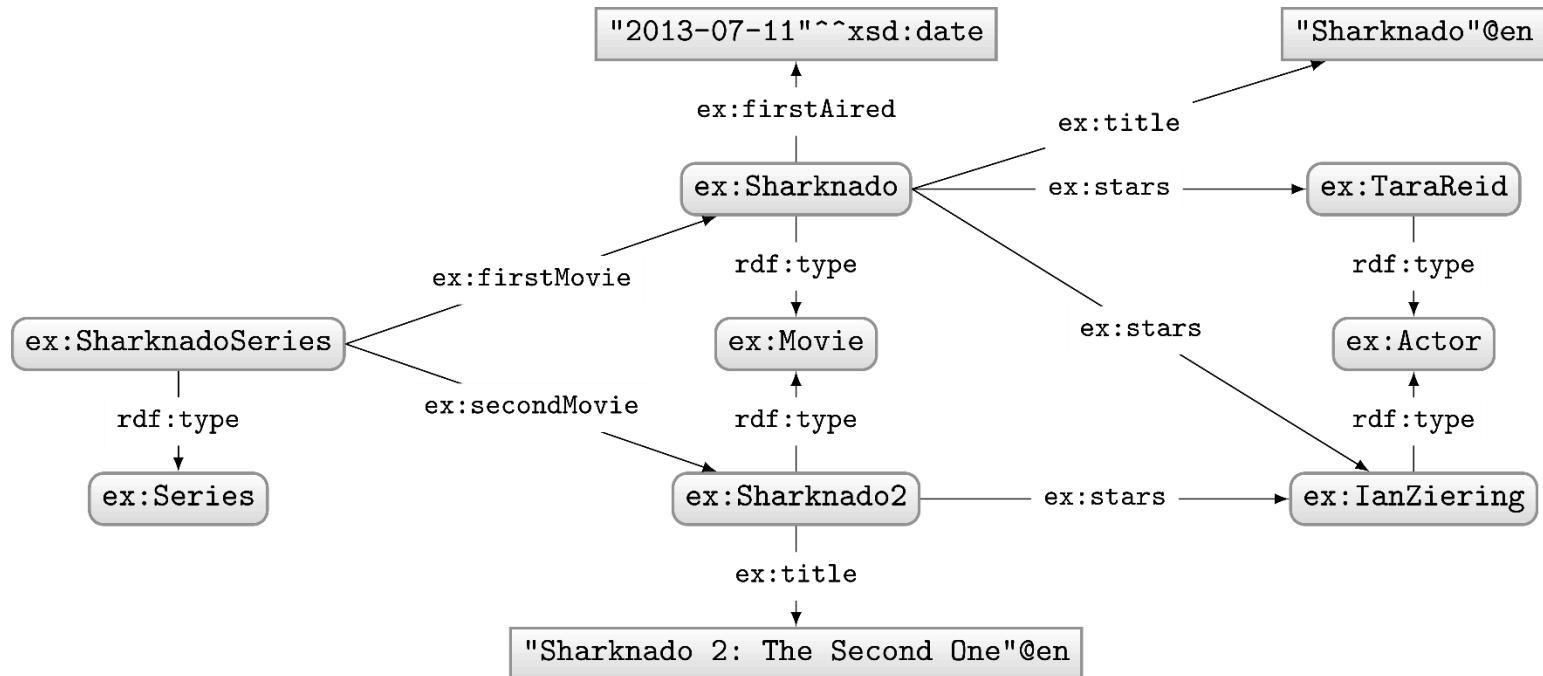
?movie	?count
ex:Sharknado	2

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 {
  ?movie ex:stars ?star .
}
GROUP BY ?movie
HAVING(COUNT(DISTINCT ?star) > 1)
```

Solutions:

?movie	?aStar
ex:Sharknado	ex:TaraReid

OR

?movie	?aStar
ex:Sharknado	ex:IanZiering

## 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 graph?



ex:Vito

- ~~Vito has 3 children?~~
- ~~Vito has at least 3 children?~~

:hasChild



ex:Connie



ex:Sonny



ex:Fredo



ex:Michael

```
ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael .  
ex:Vito :hasChild ex:Fredo .  
... ?
```

## NO UNIQUE NAME ASSUMPTION (No UNA)

How many children does Vito have according to this RDF graph?



ex:Vito

- ~~Vito has 3 children?~~
- ~~Vito has at least 3 children?~~
- Vito has at least one child!

:hasChild



ex:Connie



ex:Sonny



ex:Fredo



ex:Michael

```
ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael .  
ex:Vito :hasChild ex:Fredo .  
... ?
```

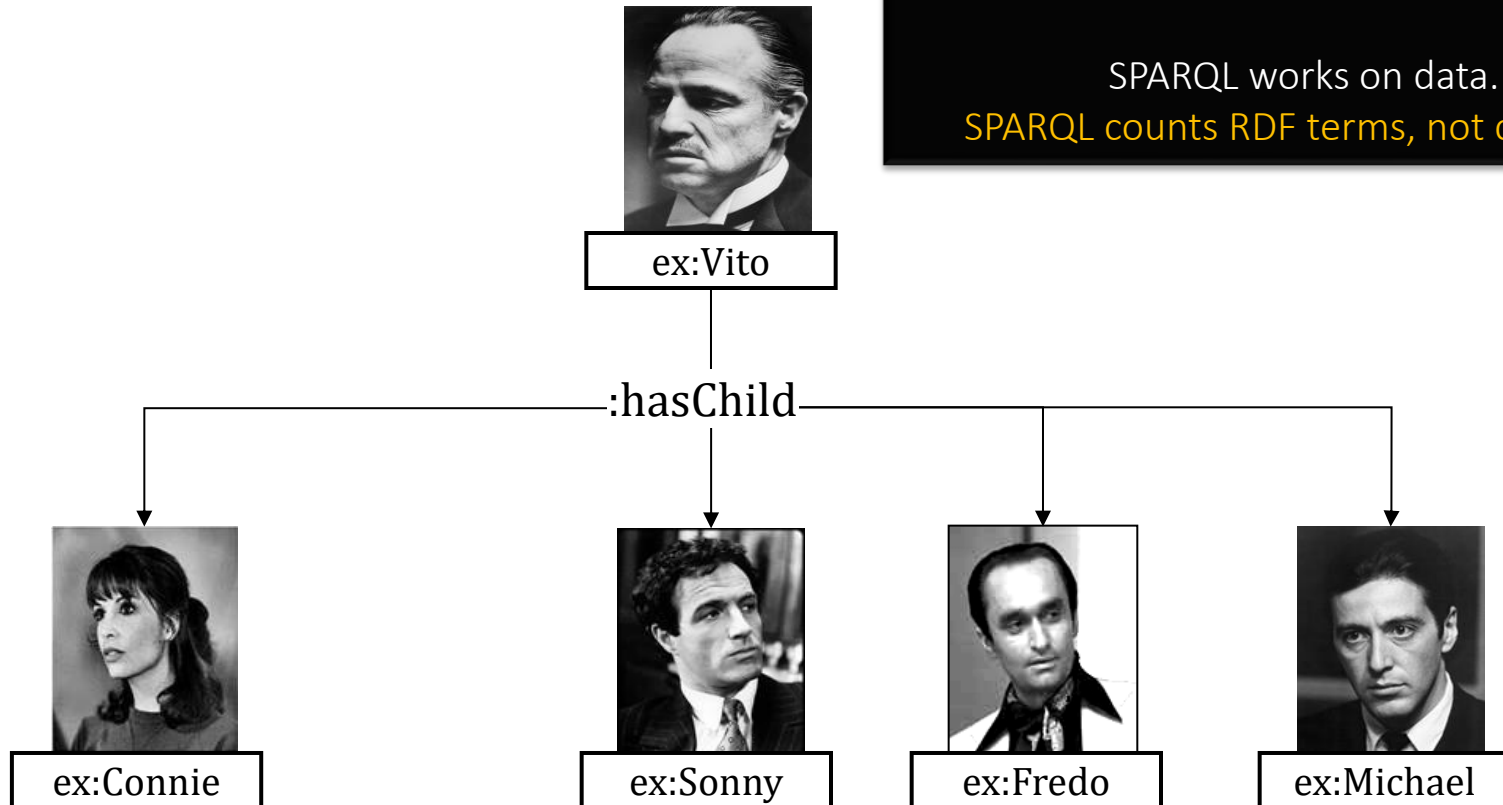
# BUT IN SPARQL ...

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.



Query:

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

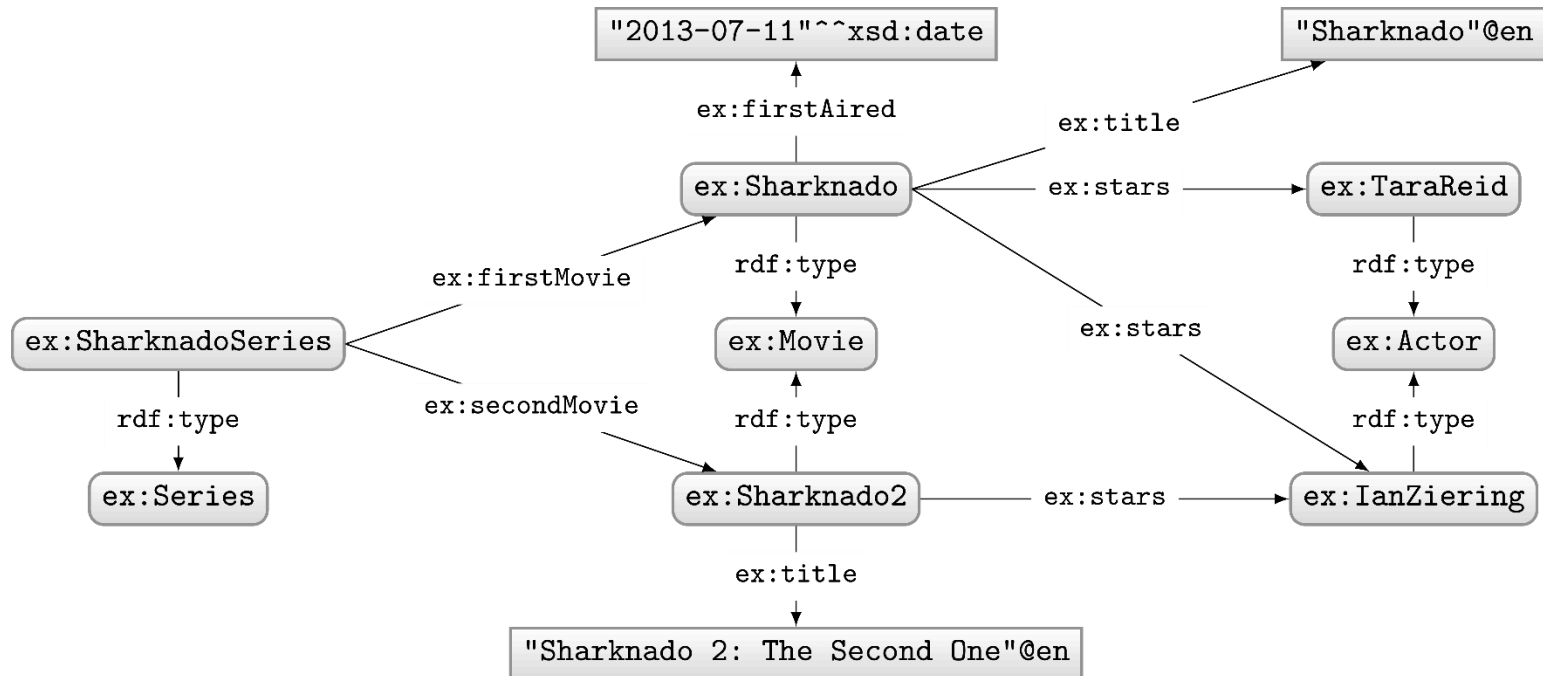
Solutions:

?count

4

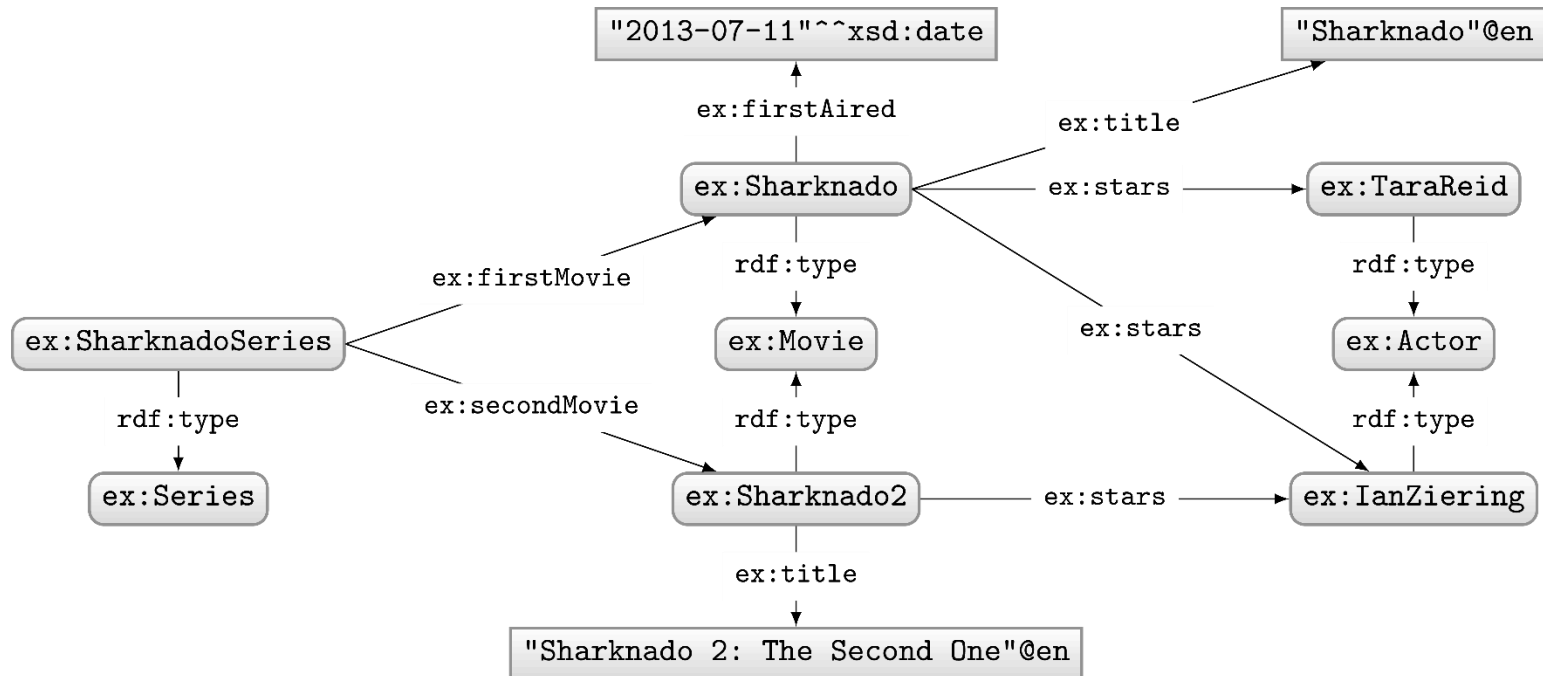
NEW QUERY FEATURE:  
SUBQUERIES

# 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 { movie ex:stars ?star . }
    GROUP BY ?movie
  }
}
```

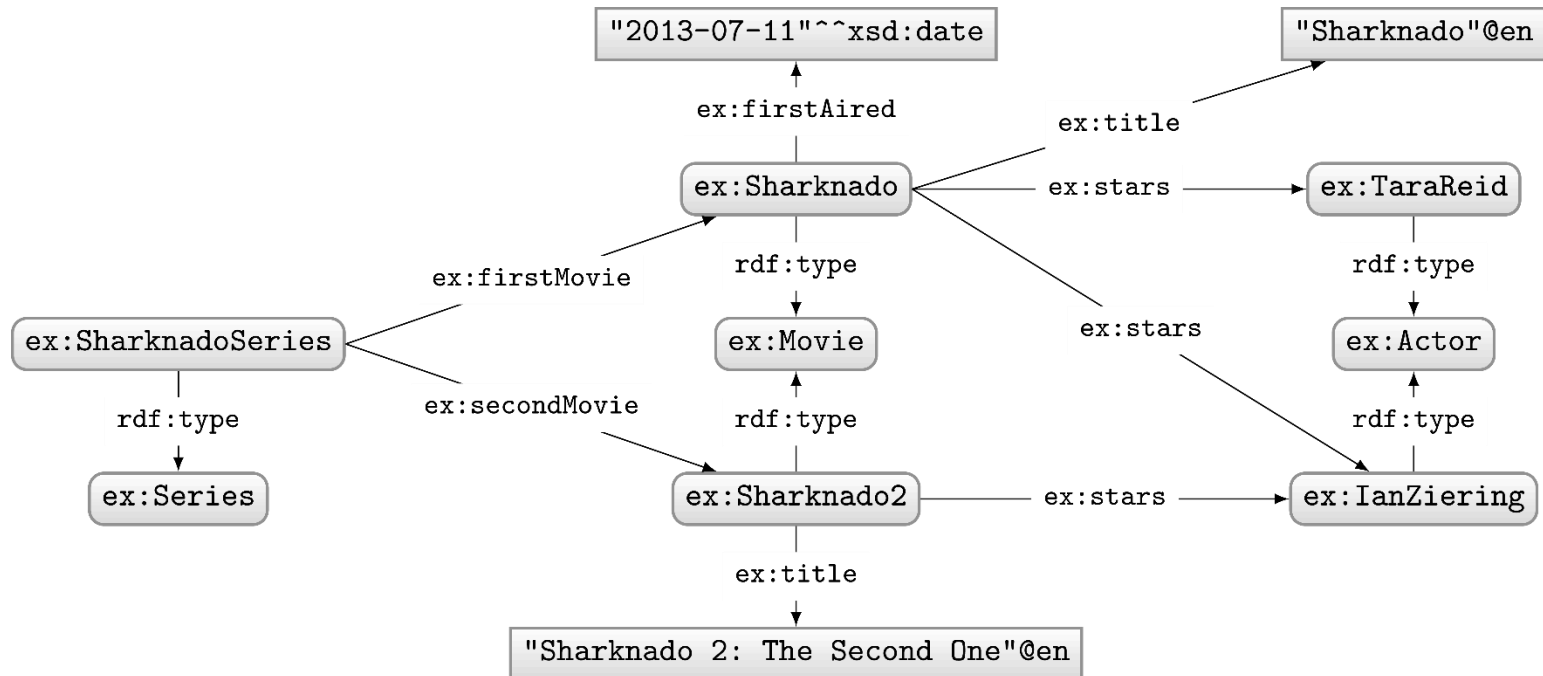
Solutions:

?avg

1.5

Sub-queries useful when you need solution modifiers or aggregates in the middle of a more complex query.

# SUBQUERIES

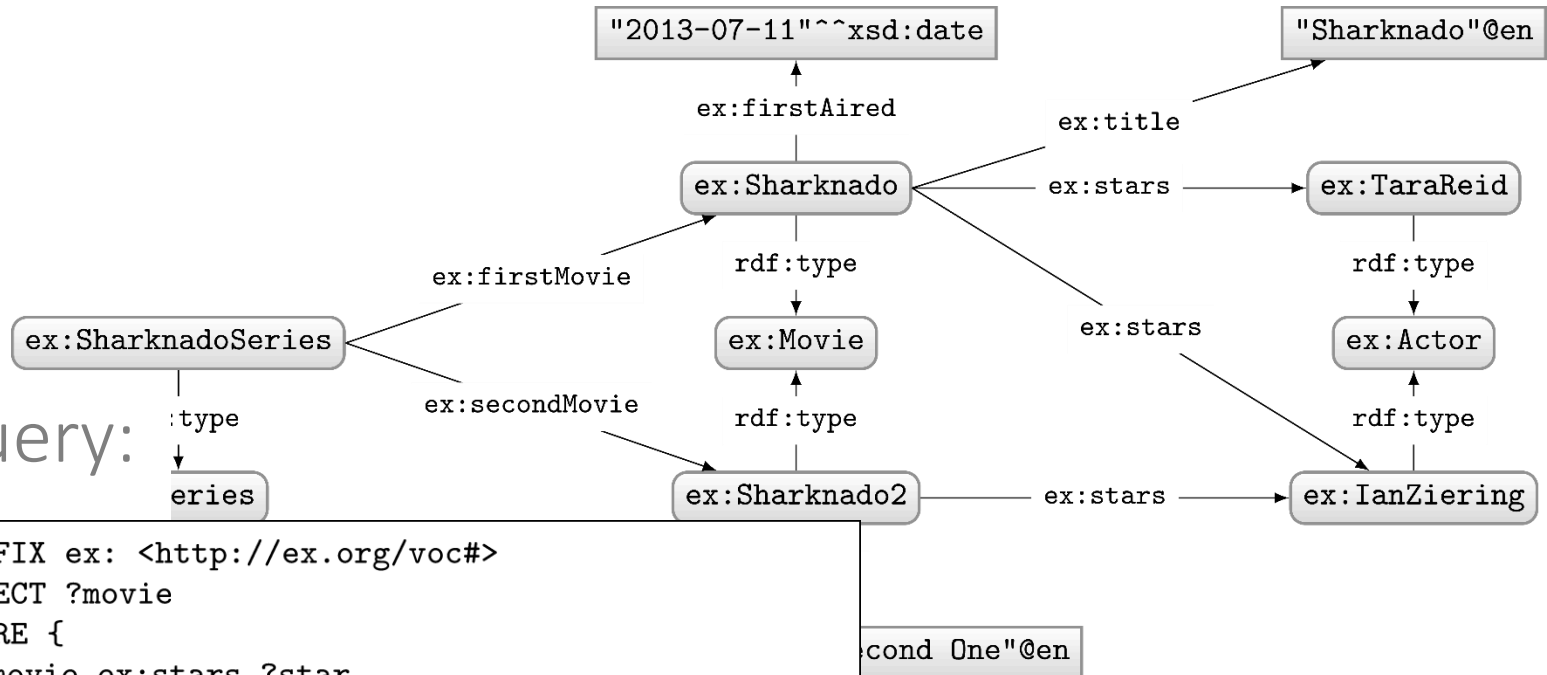


How to ask: “Which movies have more stars than average?”

# SUBQUERIES

Query:

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



Solutions:

?movie


:Sharknado

?movie in the sub-query is not correlated with ?movie in the outmost query










NEW QUERY FEATURE:  
FEDERATION




# ENDPOINTS OFTEN MADE PUBLIC/ONLINE

 Wikidata Query Service

ExamplesHelpMore toolsEnglish



1 |(Input a SPARQL query or choose a query example)



# FEDERATION: EXECUTE SUB-QUERY REMOTELY

```
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX lgdo: <http://linkedgeodata.org/ontology/>
PREFIX geom: <http://geovocab.org/geometry#>
PREFIX bif: <bif:>

SELECT ?country ?geometry ?label WHERE {
  SERVICE <http://linkedgeodata.org/sparql> {
    ?s geom:geometry [ geo:asWKT ?geometry ] ;
    a lgdo:Embassy ;
    lgdo:country ?code ;
    rdfs:label ?label .
    FILTER(bif:st_intersects(?geometry, bif:st_point(-70.6693,-33.4489), 10))
  }
  ?country wdt:P297 ?code ;
  wdt:P30 wd:Q48 . # continent: Asia
}
```

Finds Asian embassies within 10 km of Santiago centre.

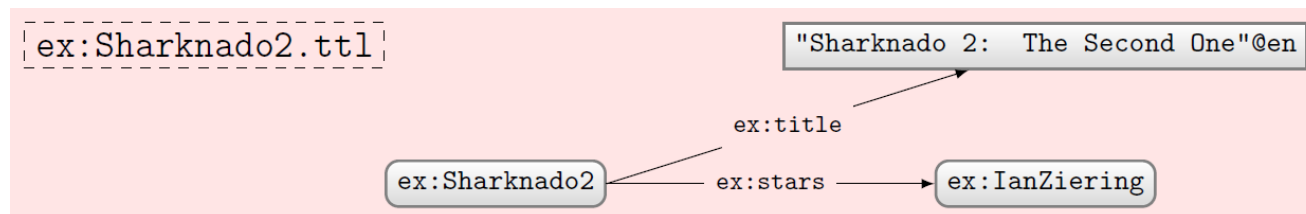
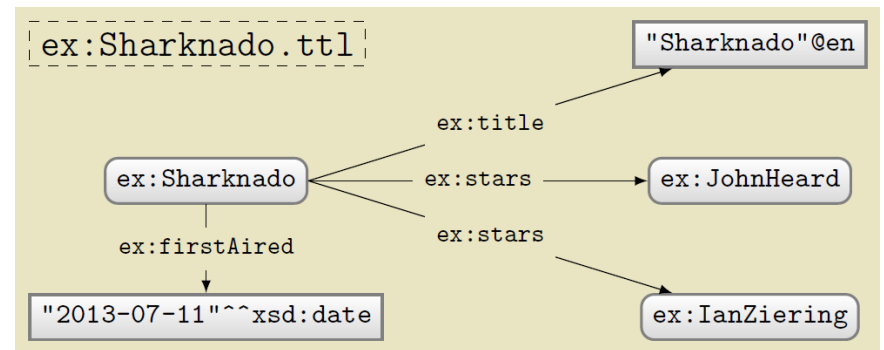
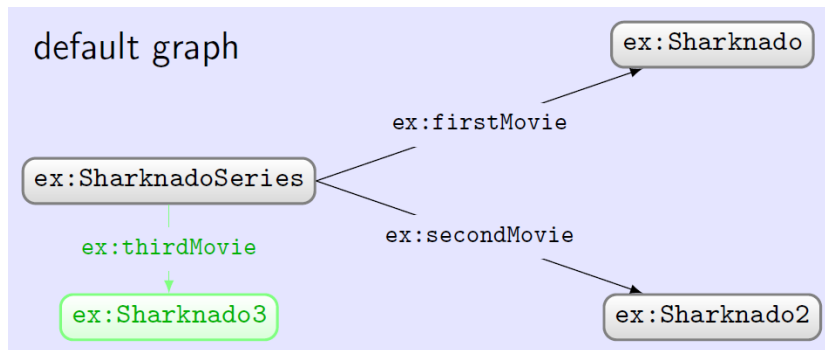
Embassies from [LinkedGeoData](http://linkedgeodata.org).

Continents from Wikidata.

NEW LANGUAGE:

SPARQL 1.1 UPDATE

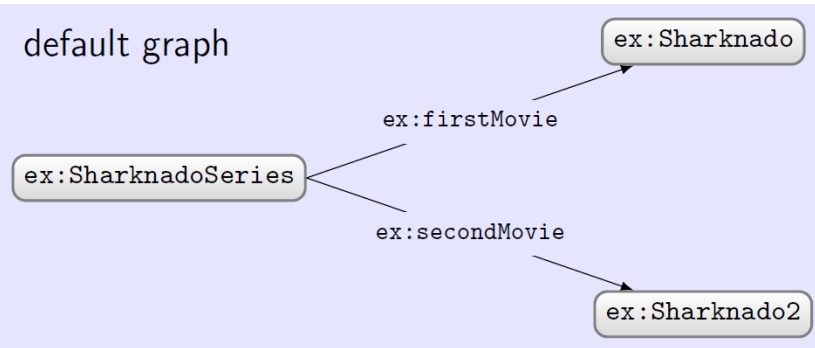
# INSERT DATA default graph



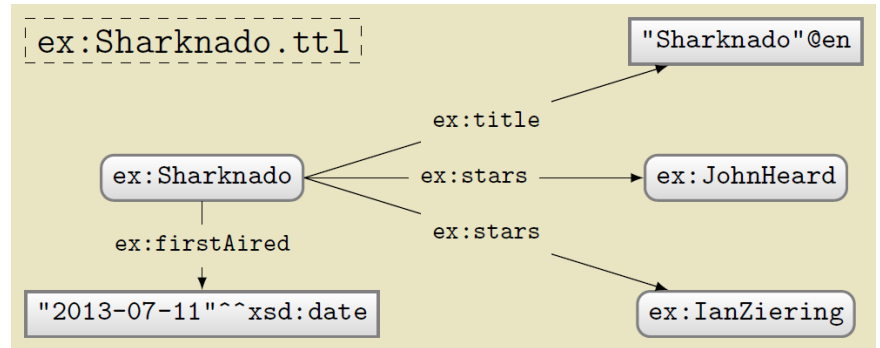
```
PREFIX ex: <http://ex.org/voc#>
INSERT DATA {
  ex:SharknadoSeries ex:thirdMovie ex:Sharknado3 .
}
```

# INSERT DATA named graph

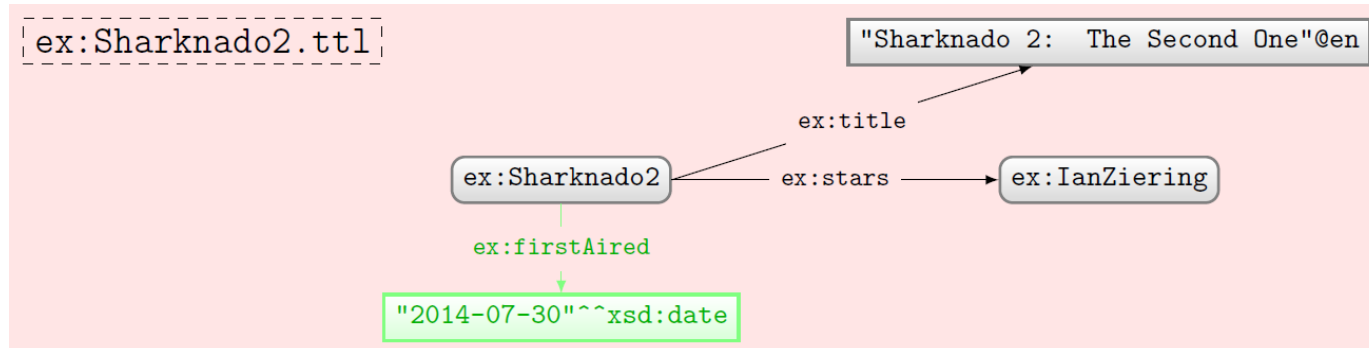
default graph



`ex:Sharknado.ttl`



`ex:Sharknado2.ttl`



```
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 {
  ex:SharknadoSeries ex:thirdMovie ex:Sharknado3 .
}
```

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

# INSERT/DELETE WITH WHERE

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

```
PREFIX ex: <http://ex.org/voc#>
DELETE {
  GRAPH ?g { ?movie ex:title ?title . }
}
WHERE {
  ex:SharknadoSeries ex:firstMovie ?movie .
  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 {
  ex:SharknadoSeries ex:secondMovie ?movie .
  GRAPH ?g { ?movie ex:description ?olddescription . }
}
```

Solutions for WHERE generated before  
insertions/deletions

Deletions performed before insertions.



# SET DEFAULT UPDATE GRAPH: WITH

```
PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE {
    ?movie ex:description ?olddescription .
}
INSERT {
    GRAPH ex:Sharknado.ttl { ex:Sharknado ex:sequel ?movie }
}
WHERE {
    ?movie ex:title "Sharknado 2: The Second One"@en .
}
```

# SIMPLE DELETE WHERE

```
PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE WHERE {
    ?movie ex:description ?olddescription .
}
```

Equivalent to ...

```
PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE {
    ?movie ex:description ?olddescription .
}
WHERE {
    ?movie ex:description ?olddescription .
}
```

# MANAGING NAMED GRAPHS: LOAD

- LOAD a graph from the Web

```
LOAD ( SILENT )? IRI-from ( INTO GRAPH IRI-to )?
```

- **SILENT**: If load fails, suppress error
- **IRI-from**: location of graph online
- **IRI-to**: local named graph to load into
  - If not given, default graph will be appended

- Destination graph created if it does not exist (otherwise data are 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 IRI | DEFAULT | NAMED | ALL )
```

- SILENT: If clear fails, suppress error
- GRAPH IRI: 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)

# MANAGING NAMED GRAPHS: CREATE

- CREATE a new blank named graph

```
CREATE ( SILENT )? GRAPH IRI
```

- SILENT: If create fails, suppress error
- GRAPH IRI: 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 IRI | DEFAULT | NAMED | ALL )
```

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

- Will fail if graph does not exist (unless silent is specified)
- An engine must have a default graph: DROP DEFAULT same as CLEAR DEFAULT

# MANAGING NAMED GRAPHS: COPY

- COPY one graph to another

```
COPY ( SILENT )? ( ( GRAPH )? IRI-from | DEFAULT ) TO  
      ( ( GRAPH )? IRI-to | DEFAULT )
```

- SILENT: If copy fails, suppress error
- IRI-from: name of graph to copy from
- IRI-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
- Source graph unaffected

# MANAGING NAMED GRAPHS: MOVE

- MOVE one graph to another

```
MOVE ( SILENT )? ( ( GRAPH )? IRI-from | DEFAULT ) TO  
      ( ( GRAPH )? IRI-to | DEFAULT )
```

- SILENT: If move fails, suppress error
- IRI-from: name of graph to move
- IRI-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 )? IRI-from | DEFAULT ) TO  
      ( ( GRAPH )? IRI-to | DEFAULT )
```

- SILENT: If add fails, suppress error
- IRI-from: name of graph to add
- IRI-to: name of graph to add to
- DEFAULT: add from/to default graph

- May fail if source graph does not exist (unless silent is specified)
- Destination graph created if it does not exist (otherwise data are appended)
- 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 the book chapter or
  - <http://www.w3.org/TR/sparql11-entailment/>



NEW FEATURE:

SPARQL 1.1 OUTPUT FORMATS

# WHAT'S NEW IN SPARQL 1.1?

- New query features
- An update language
- Support for RDFS/OWL entailment
- New 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
  - 4XX if SPARQL query is invalid
  - 5XX if query was okay but server failed to answer
    - ... etc. See more details:
      - <http://www.w3.org/TR/sparql11-protocol/>



# SPARQL ENDPOINTS ON THE WEB!



Wikidata Query Service

<https://query.wikidata.org/#PREFIX%20wd%3A%20%3Chttp%3A%2F%2Fwww.wikidata.org%2Fentity%2F%3>

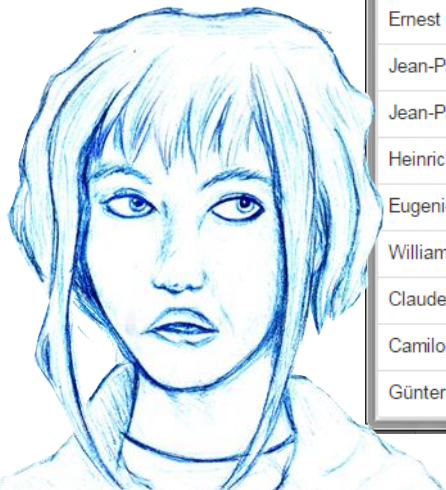
Aplicaciones Linguee SGICM Apache Any23: Anyth

```
16 ?war wdt:P580 ?warStart . # Get date the war started
17 BIND(YEAR(?warStart) as ?warYear) # Get year from date
18 ?laureate rdfs:label ?laureateName . # Get name of laureate
19 FILTER(lang(?warName)="en" # Filter for English
20 && lang(?laureateName)="en") # ... names only
21 } ORDER BY ?awardYear # Order by year
22
```

Press [CTRL-SPACE] to activate auto completion. Data updated a few seconds ago

Run Clear 12 Results in 1589 ms Display Download Link

laureateName	awardYear	warName	warYear
Carl Spitteler	1919	World War I	1914
Winston Churchill	1953	World War I	1914
Ernest Hemingway	1954	World War I	1914
Ernest Hemingway	1954	World War II	1939
Jean-Paul Sartre	1964	Algerian War	1954
Jean-Paul Sartre	1964	World War II	1939
Heinrich Böll	1972	World War II	1939
Eugenio Montale	1975	World War I	1914
William Golding	1983	World War II	1939
Claude Simon	1985	Spanish Civil War	1936
Camilo José Cela	1989	Spanish Civil War	1936
Günter Grass	1999	World War II	1939



QUESTIONS?

