

**CC6202-1**

**LA WEB DE DATOS**

**PRIMAVERA 2015**

**Lecture 8: SPARQL (1.1)**

Aidan Hogan

aidhog@gmail.com

**PREVIOUSLY ...**



(1.1)  
QL

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/>

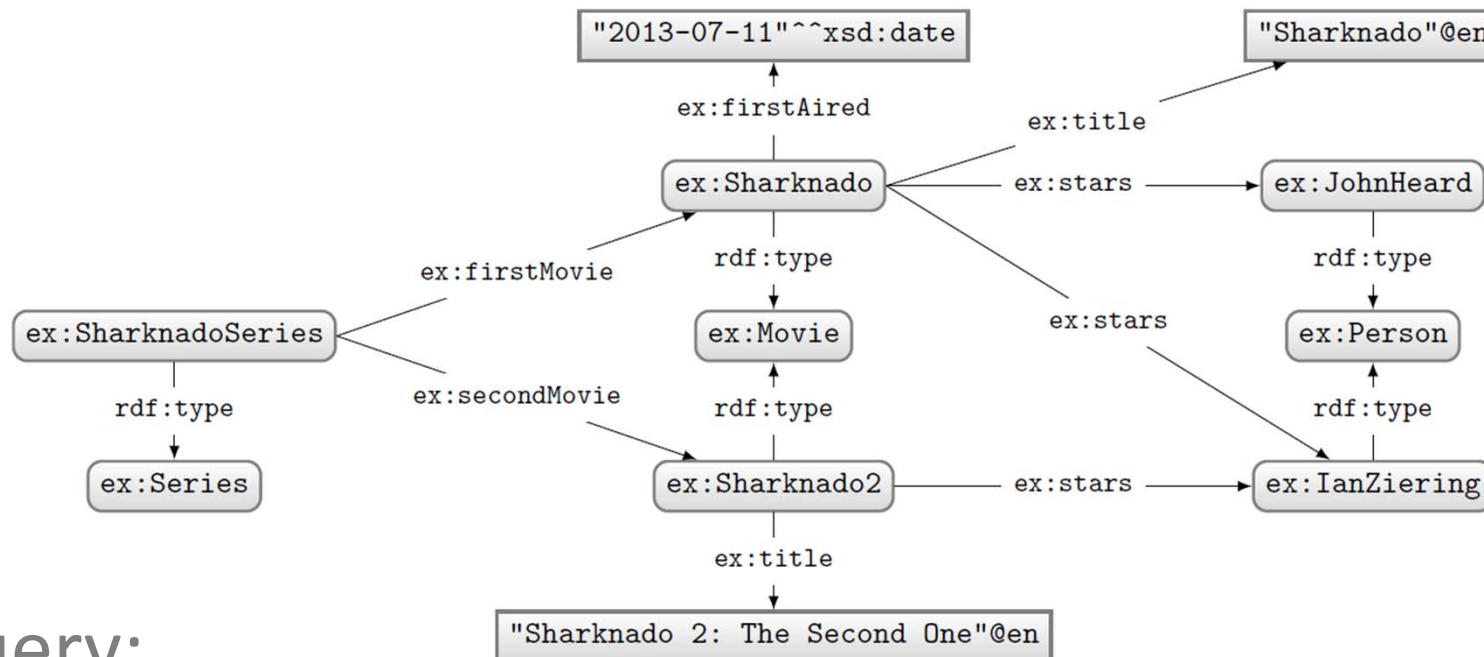
**Editors:**

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)>



# SPARQL: WHERE clause example



Query:

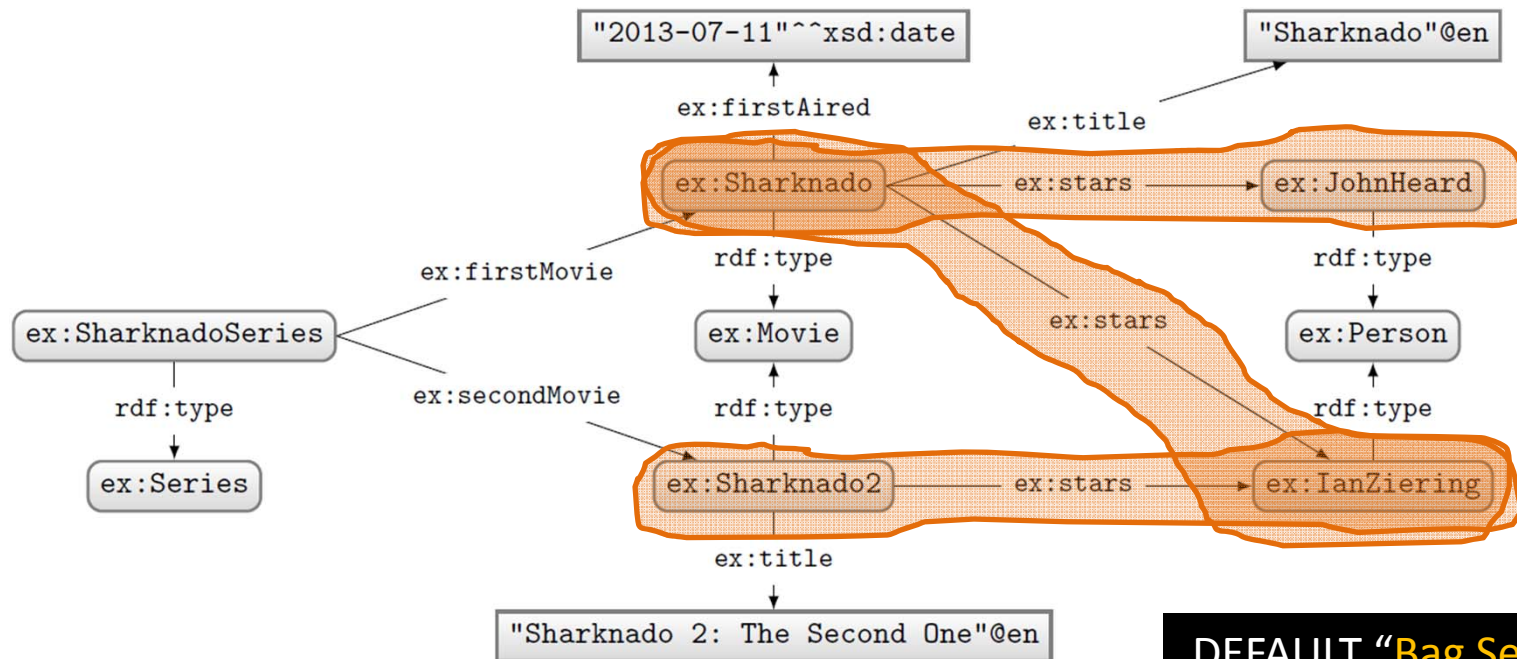
```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  { ex:SharknadoSeries ex:firstMovie ?movie . }
  UNION
  { ex:SharknadoSeries ex:secondMovie ?movie . }
  OPTIONAL
  { ?movie ex:firstAired ?date . }
  ?movie ex:title ?title .
  FILTER(REGEX(STR(?title),"*[0-9]*"))
}
```

What solutions would this query return?

Solutions:

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

# SPARQL: SELECT with projection



Query:

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

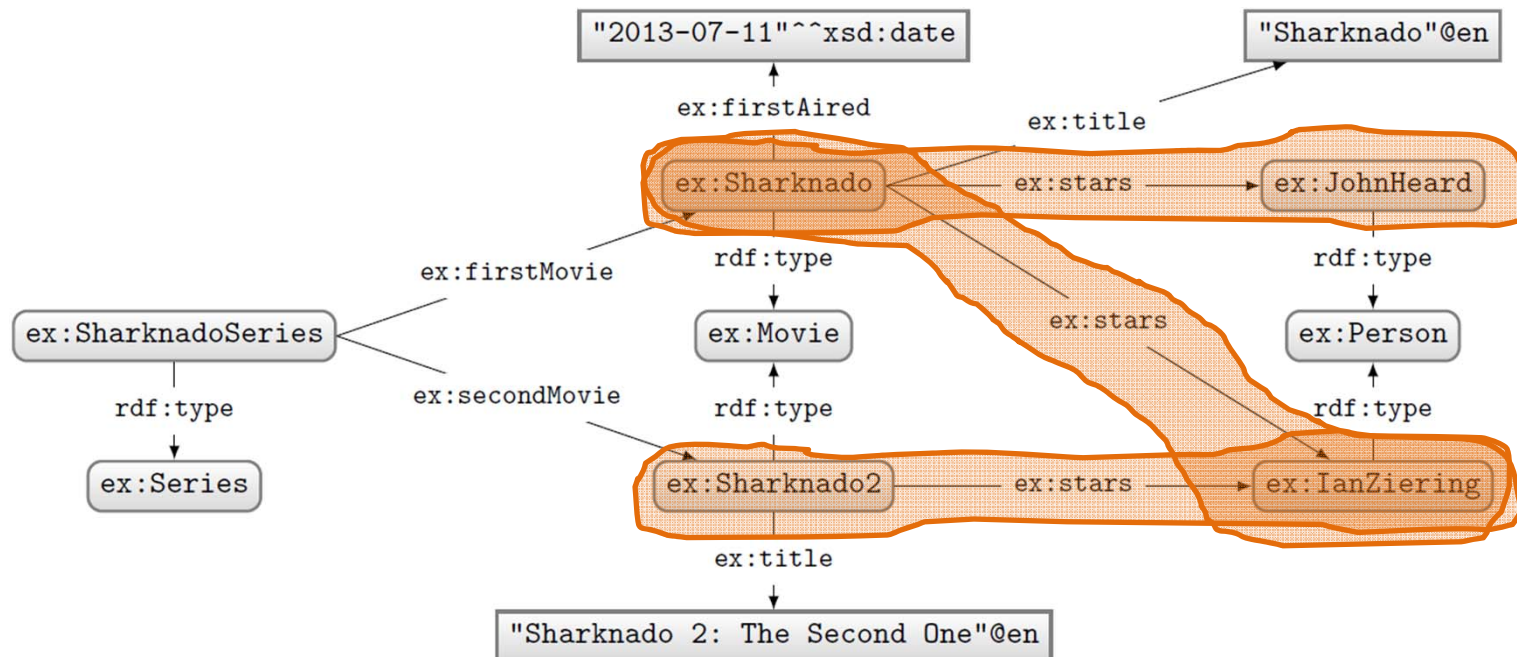
Solutions:

| ?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:

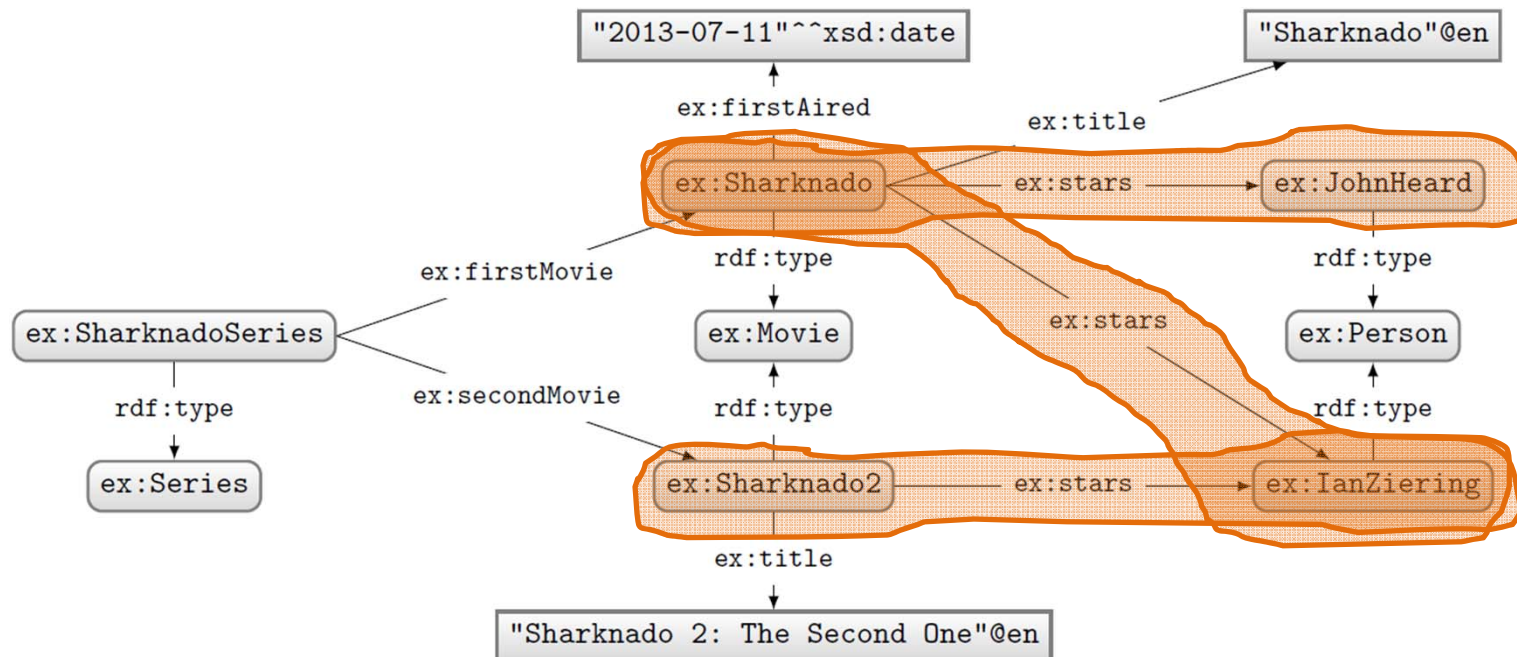
```
PREFIX ex: <http://ex.org/voc#>
ASK
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

Solutions:

true

Returns true if  
there is a match,  
false otherwise.

# SPARQL: CONSTRUCT



Query:

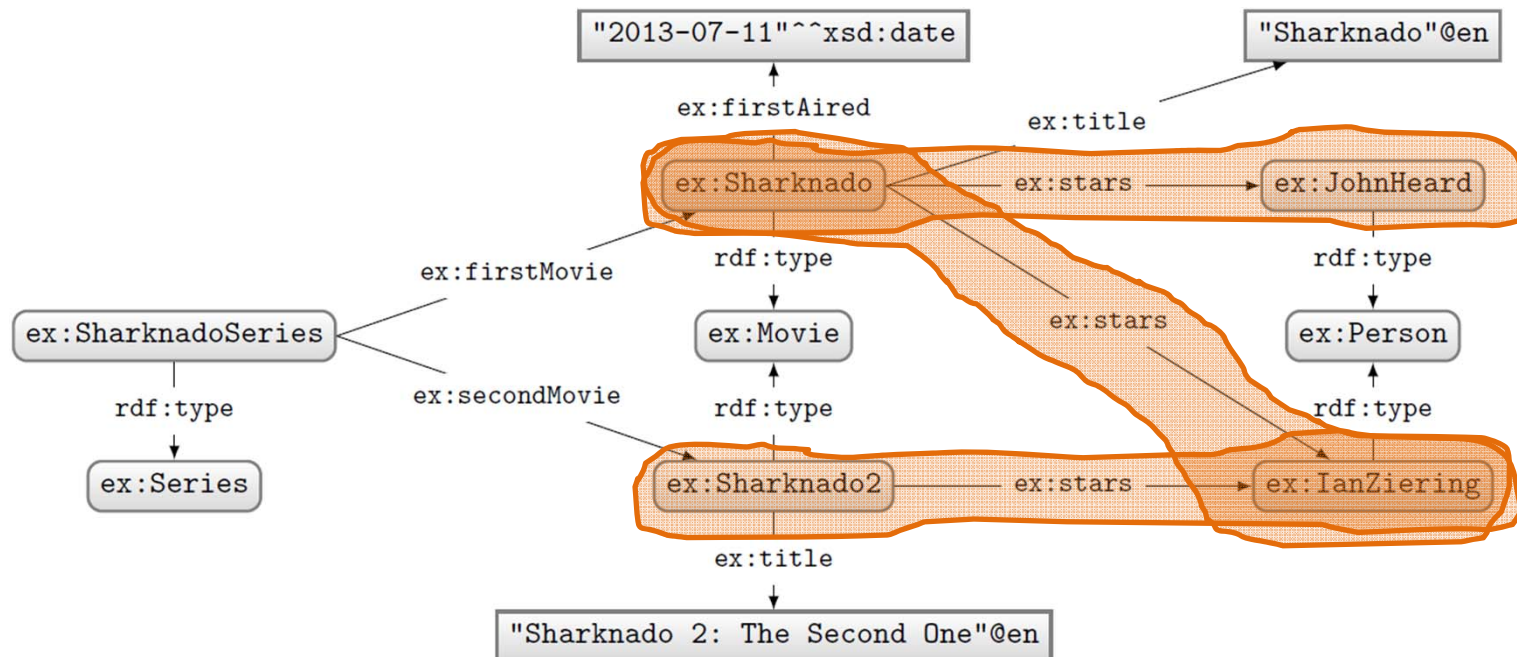
```
PREFIX ex: <http://ex.org/voc#>
CONSTRUCT { ?star ex:job ex:Actor }
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

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)



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.



# 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

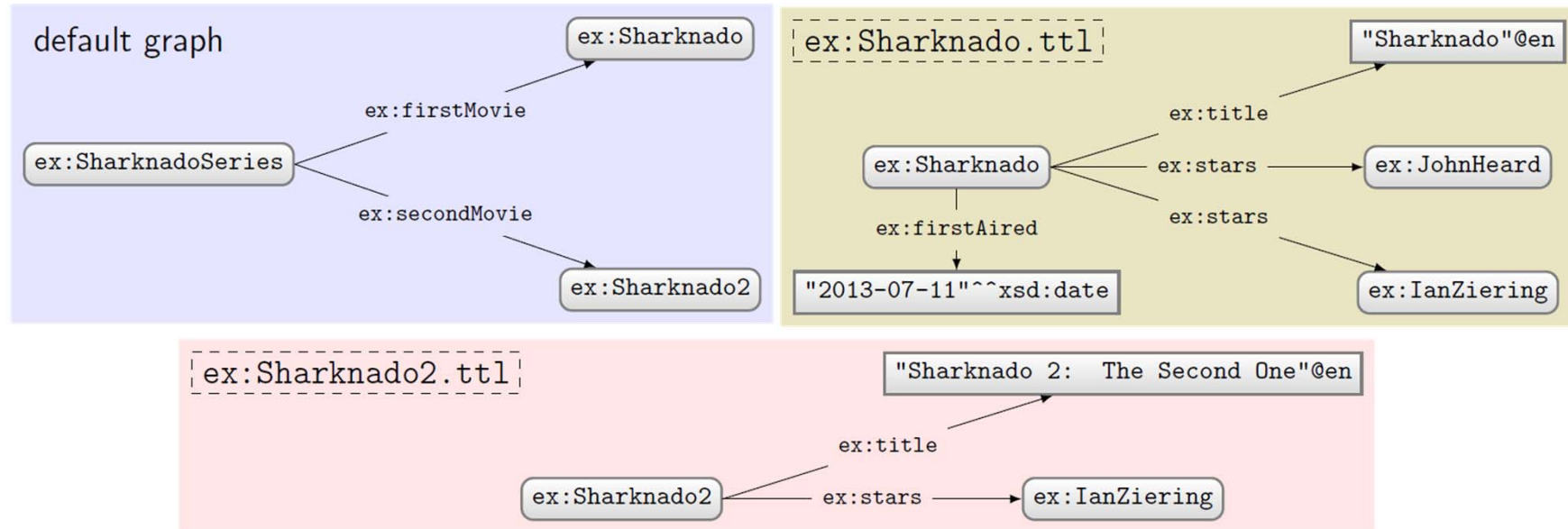
Strictly speaking, by default, no ordering is applied. Hence OFFSET means nothing without ORDER BY. However, some engines support a default ordering (e.g., the order of computation of results).

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
```



# 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:

| ?x            | ?q       |
|---------------|----------|
| ex:Sharknado2 | ex:stars |

**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.

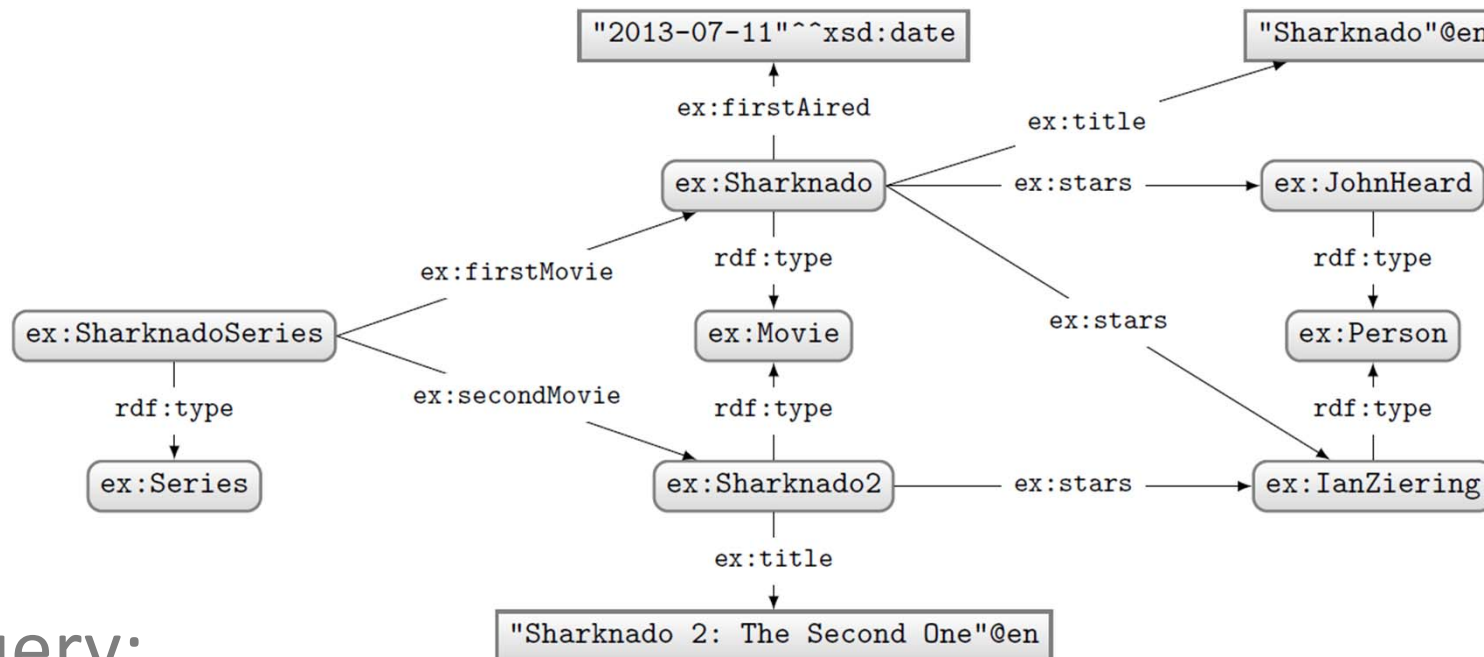
See also [translations](#).

# 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
  { ?movie ex:firstAired ?date . }
  FILTER(!BOUND(?date))
}
```

What solutions would this query return?

Solutions:

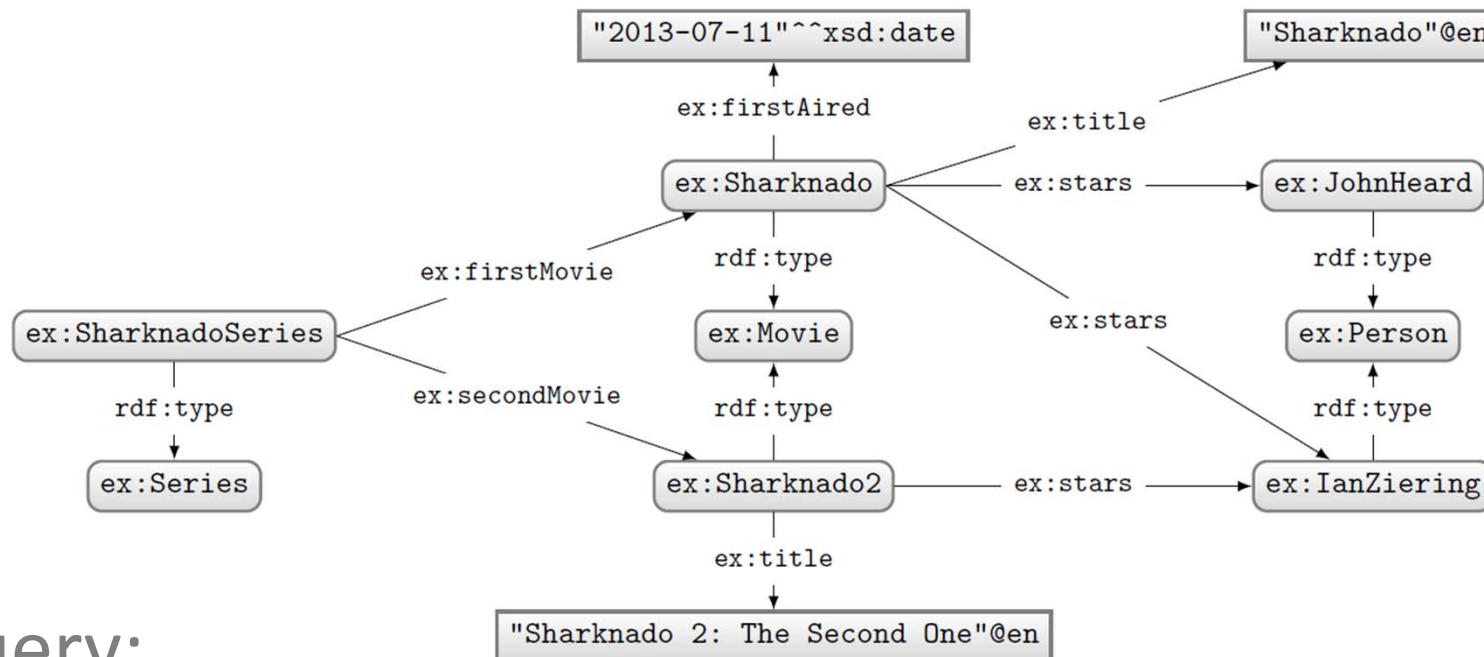
?movie

ex:Sharknado2

Can do a closed-world style of negation!



# SPARQL 1.1: (NOT) EXISTS



Query:

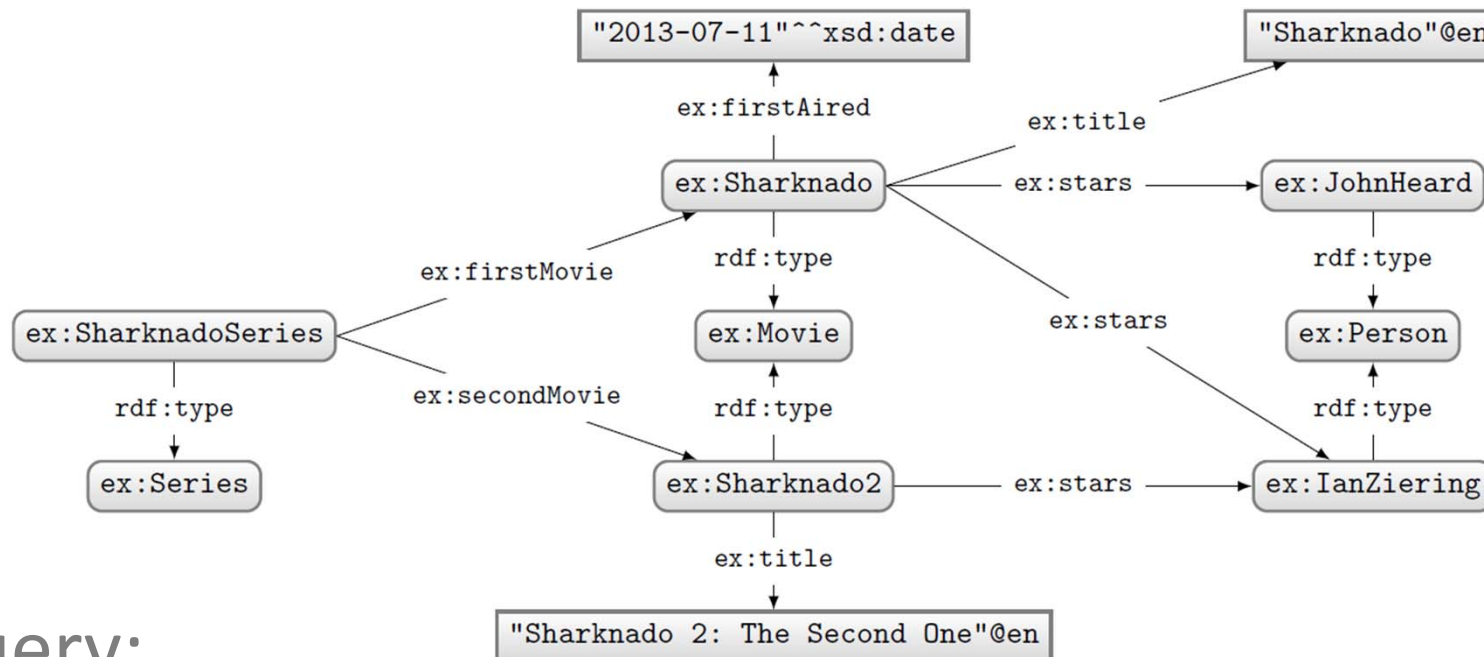
```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
  ?movie a ex:Movie .
  FILTER NOT EXISTS
    { ?movie ex:firstAired ?date }
}
```

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

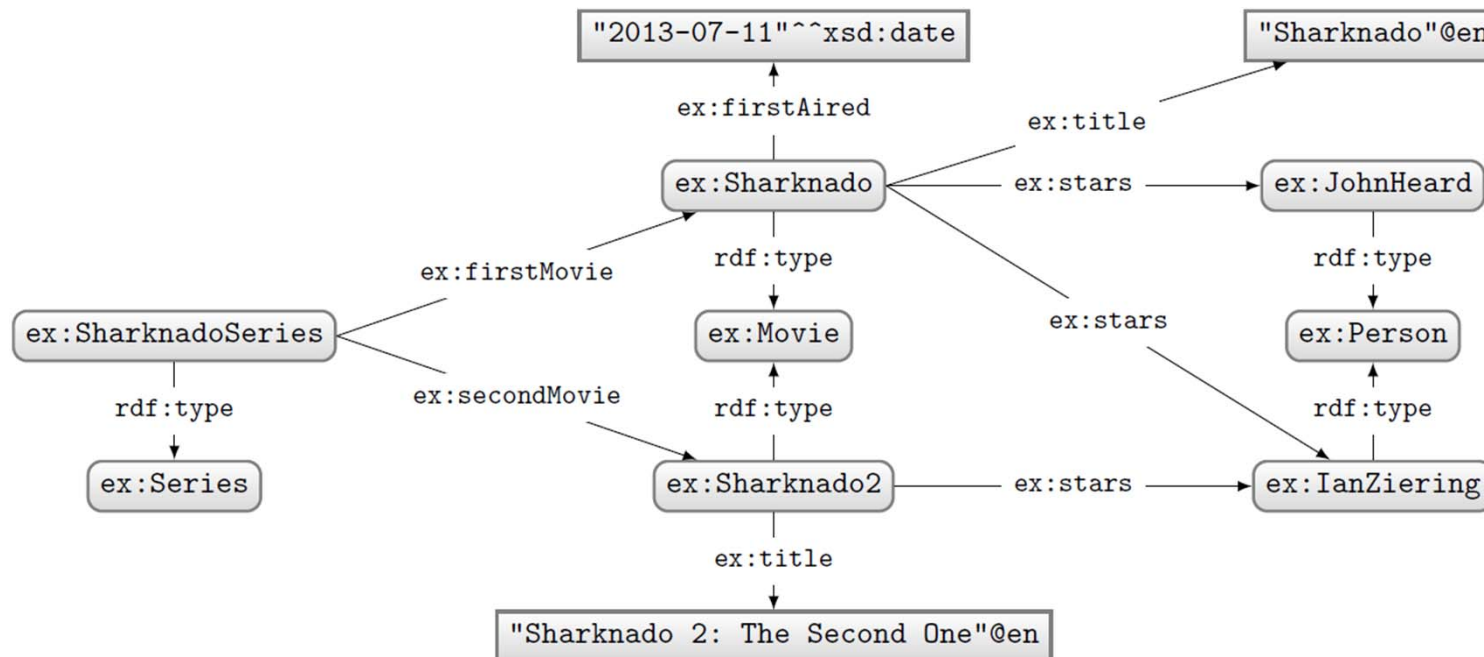
## Difference between MINUS and NOT EXISTS?

- **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?



```
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  
between the results!**  
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

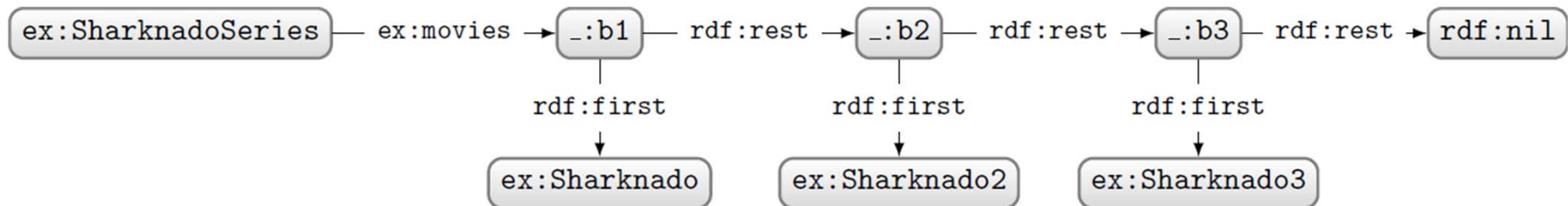
---

|                                                  |                                    |
|--------------------------------------------------|------------------------------------|
| $p$                                              | 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 (over RDF list)



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

Query:

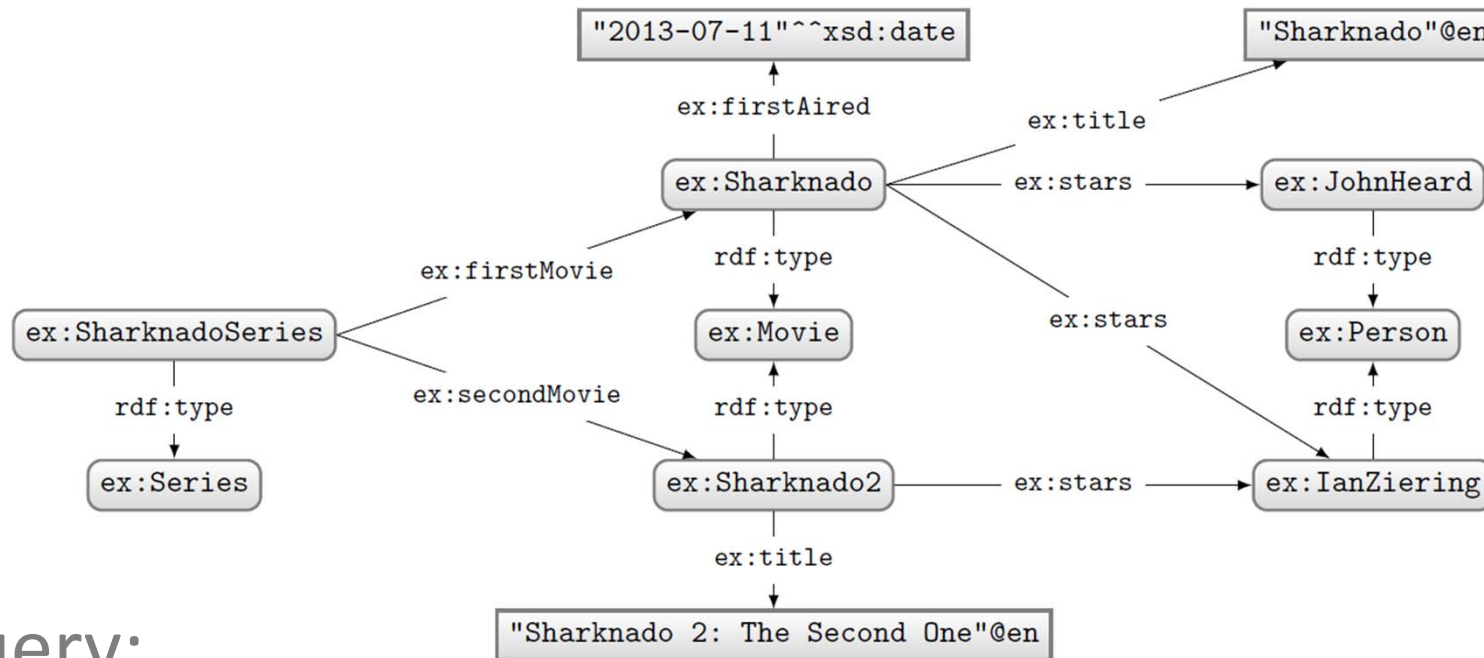
```
PREFIX ex: <http://ex.org/voc#>
SELECT ?movie
WHERE {
  ex:SharknadoSeries ex:movies/rdf:rest*/rdf:first ?movie .
}
```

Solutions:

| ?movie        |
|---------------|
| ex:Sharknado  |
| ex:Sharknado2 |
| ex:Sharknado3 |

# **NEW QUERY FEATURE: ASSIGNMENT**

# Assignment with BIND



Query:

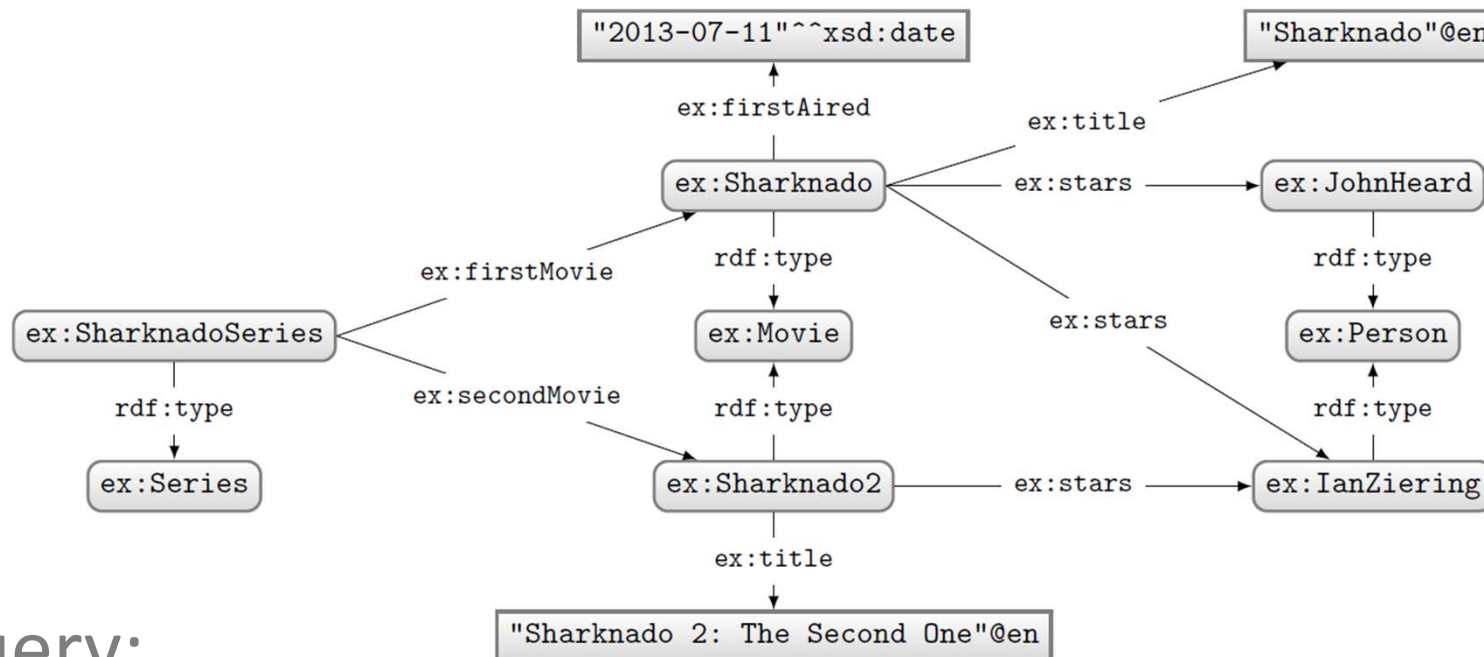
```

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       | ?star |
|--------------|-------|
| ex:Sharknado | 2013  |

# Assignment with VALUES



Query:

```

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

```

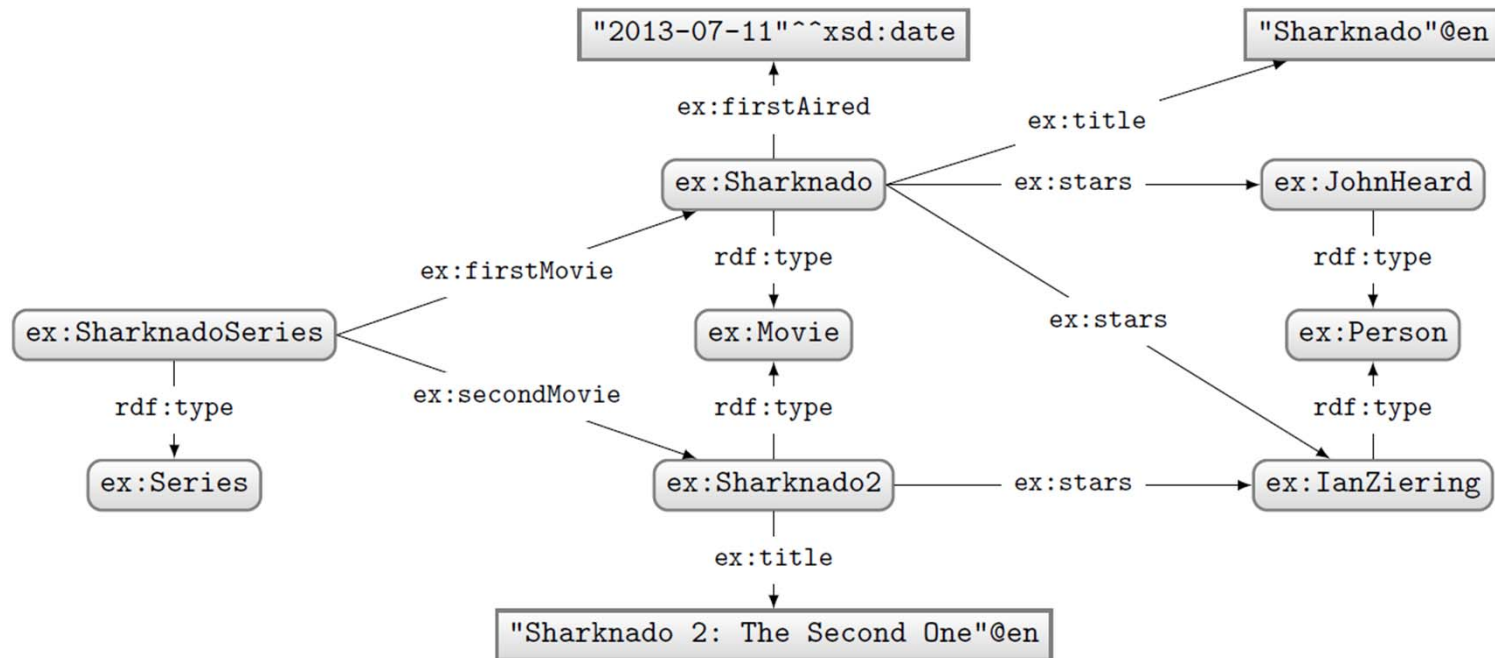
Solutions:

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

No result for  
ex:Sharknado ex:IanZiering!

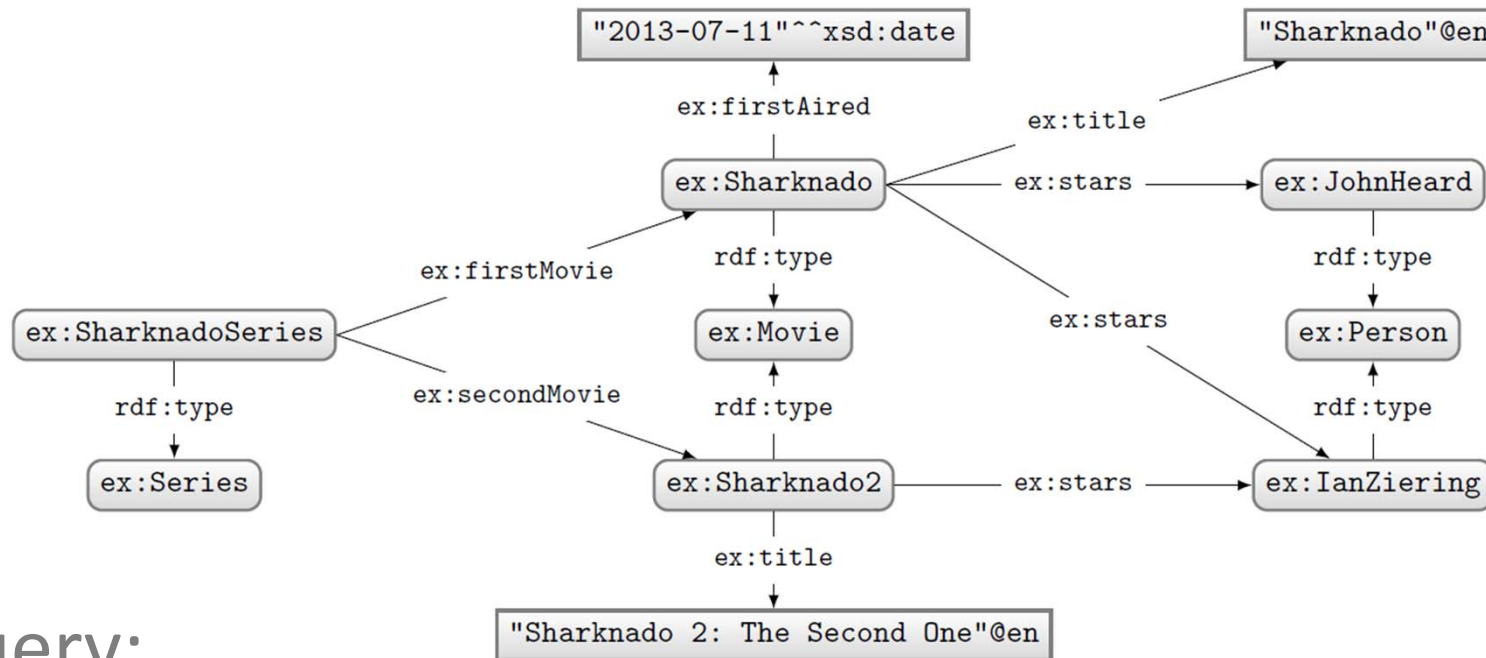
# **NEW QUERY FEATURE: AGGREGATES**

# 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 .
}
```

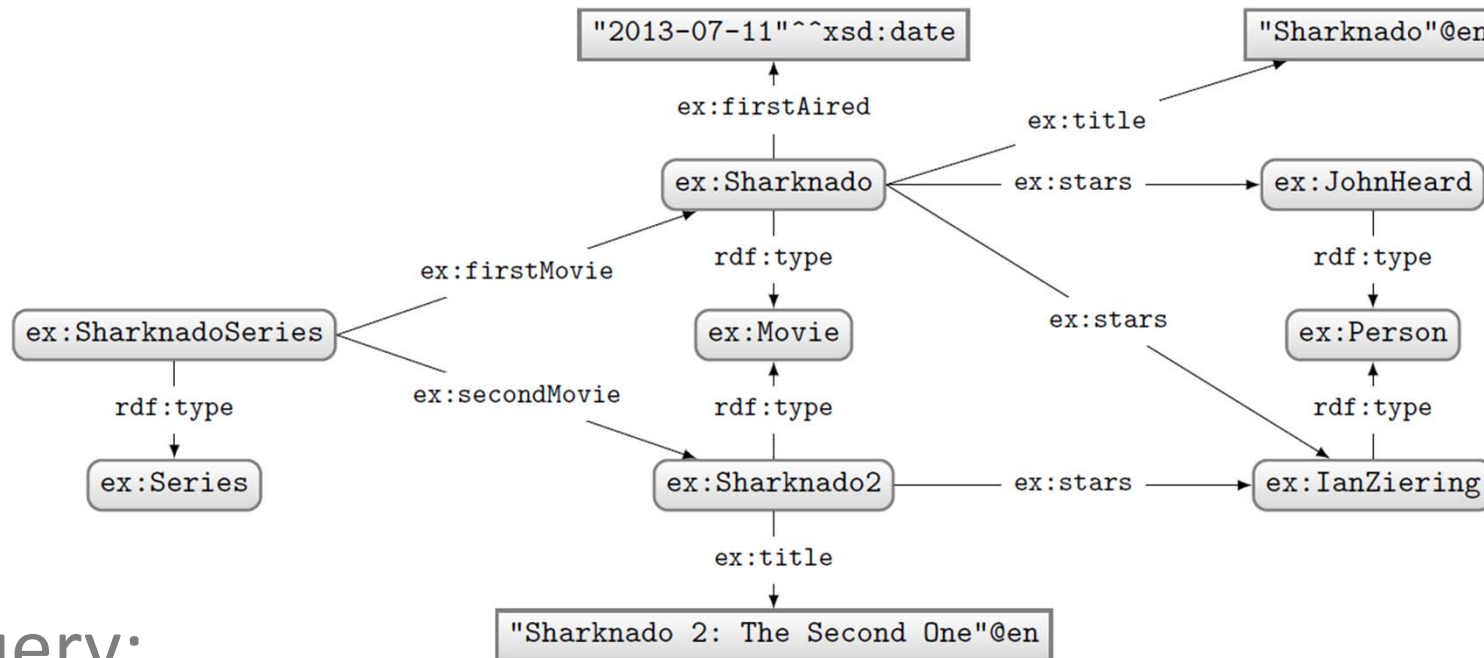
Solutions:

|        |
|--------|
| ?count |
|--------|

|   |
|---|
| 3 |
|---|



# Aggregates: COUNT



Query:

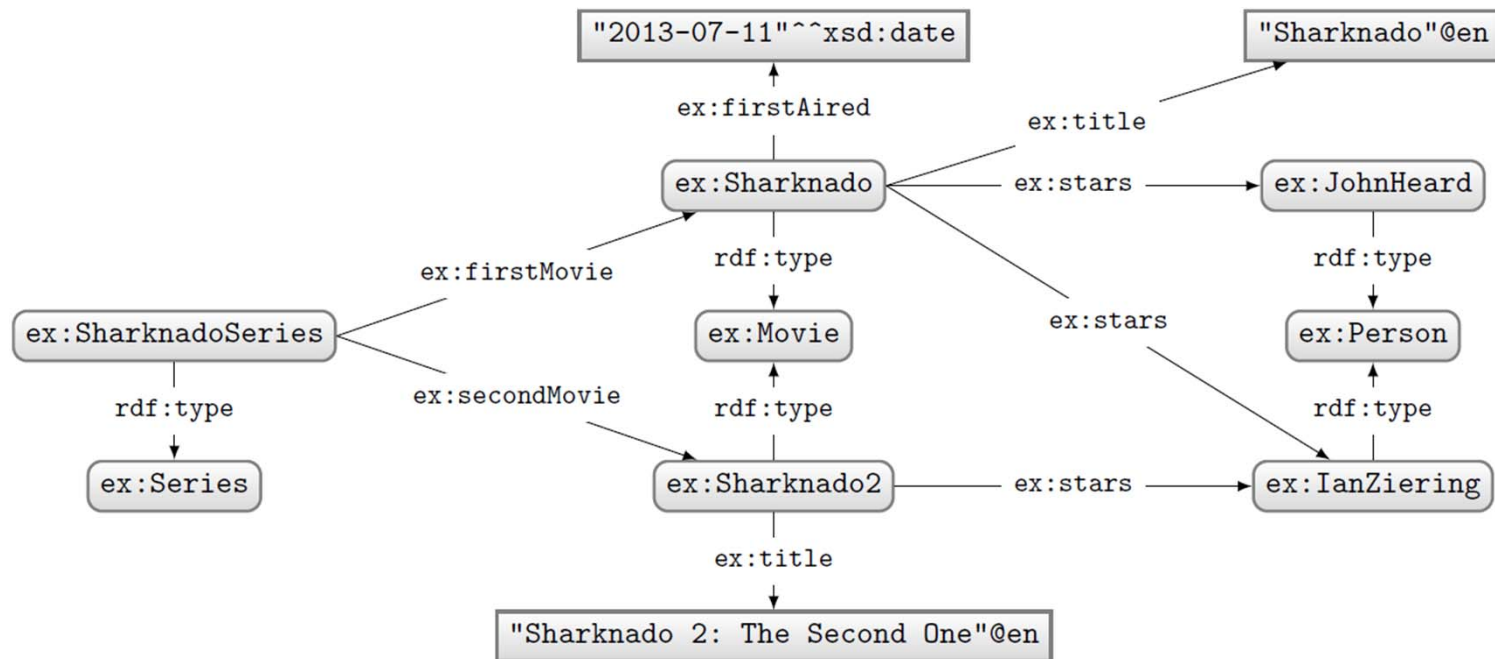
```
PREFIX ex: <http://ex.org/voc#>
SELECT
  (COUNT(DISTINCT ?star) as ?count)
WHERE {
  ?movie ex:stars ?star .
}
```

Solutions:

?count

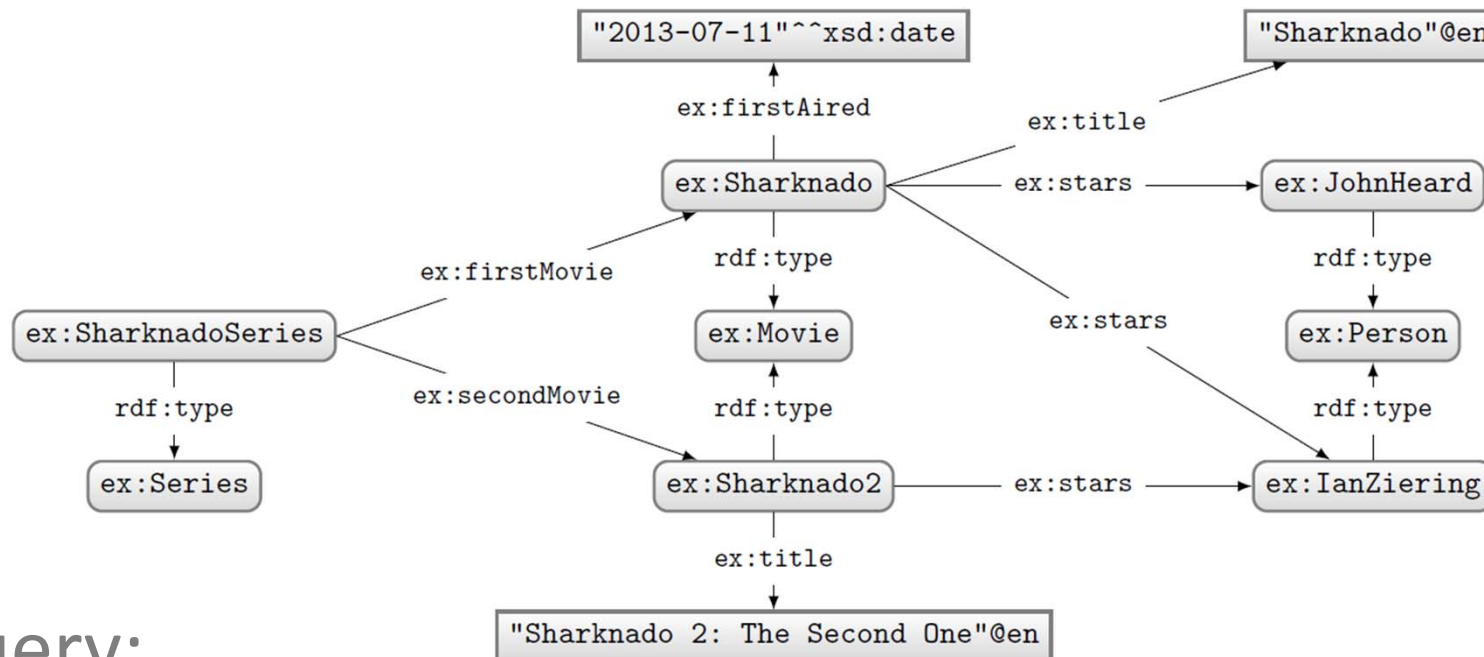
2

# Aggregates



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

# Aggregates: COUNT with GROUP BY



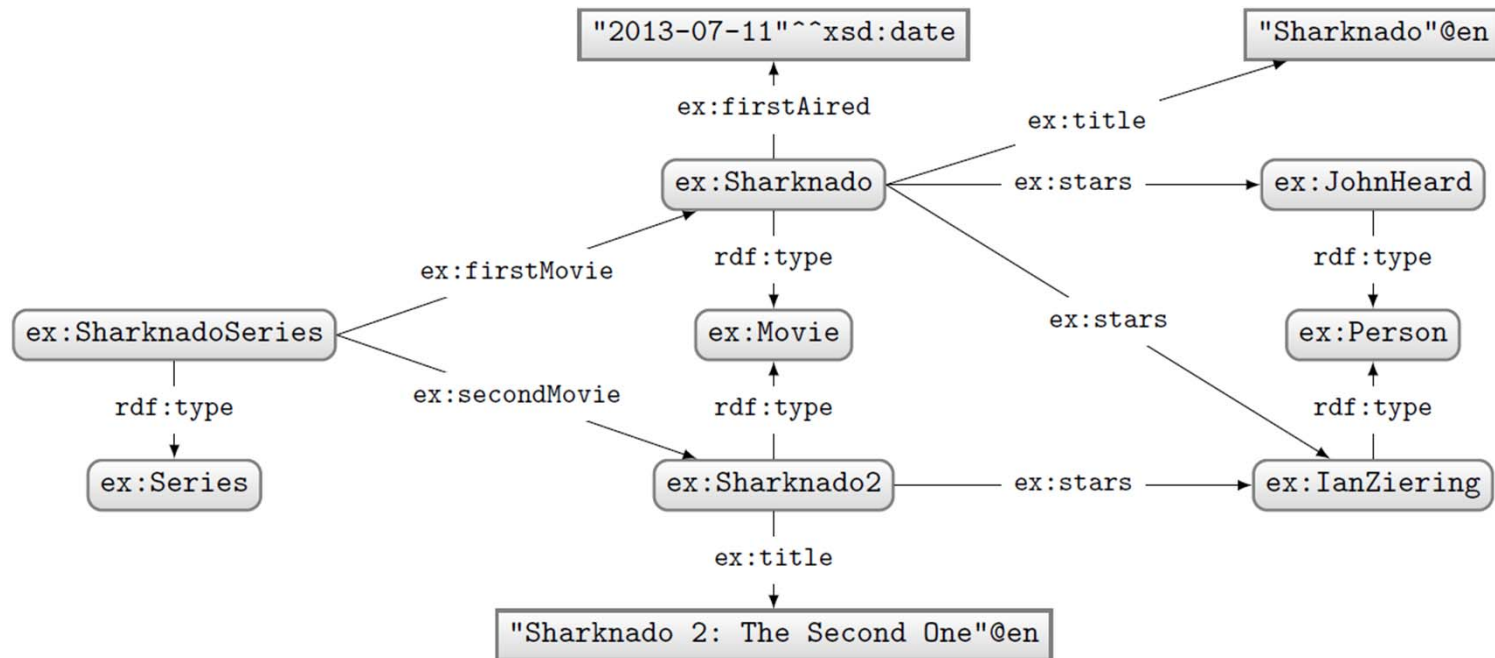
Query:

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

Solutions:

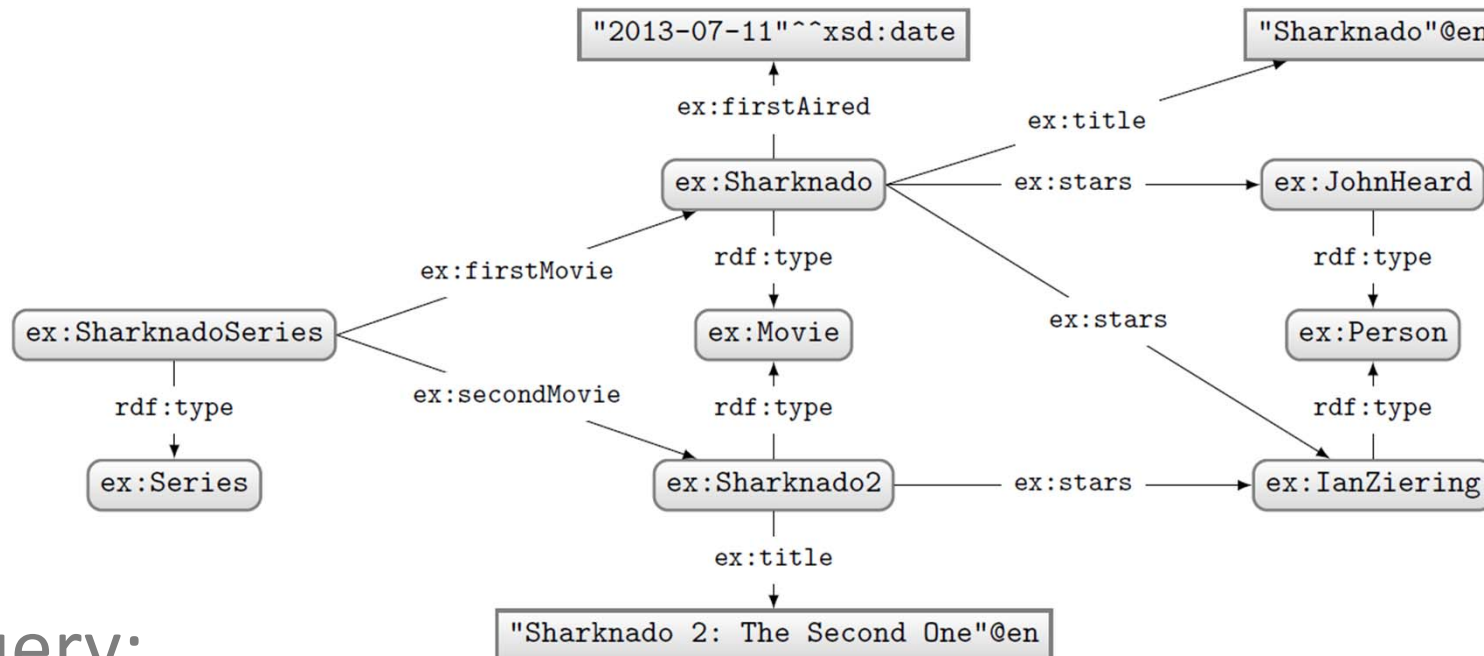
| ?movie        | ?count |
|---------------|--------|
| ex:Sharknado  | 2      |
| ex:Sharknado2 | 1      |

# 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:

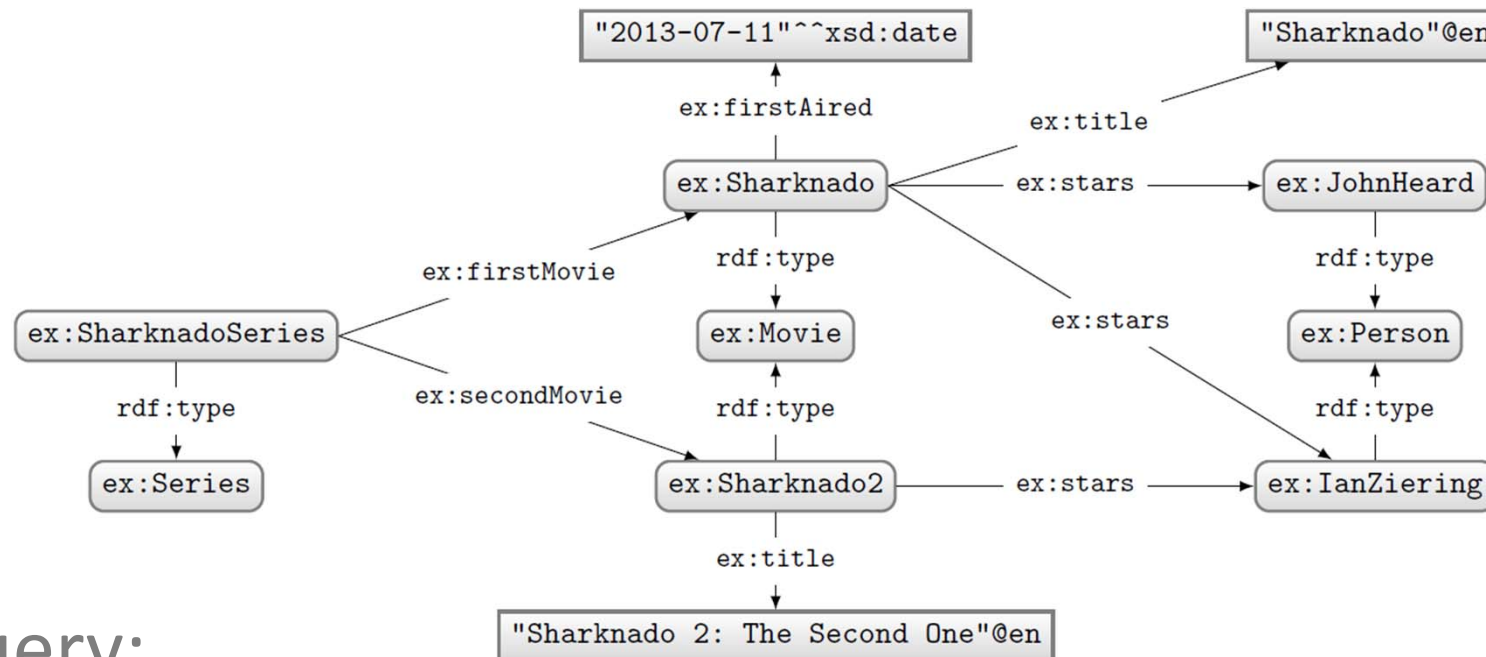
| ?movie       | ?count |
|--------------|--------|
| ex:Sharknado | 2      |

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:JohnHeard |

OR

| ?movie       | ?aStar        |
|--------------|---------------|
| ex:Sharknado | ex:IanZiering |

## **QUICK NOTE ON SEMANTICS**



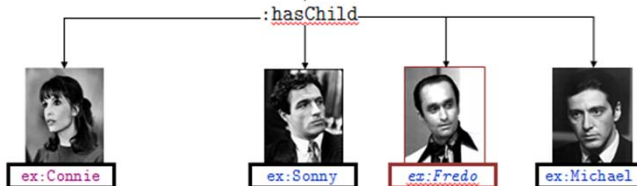
# Recall from OWL: OWA and lack of UNA

## Open World Assumption (OWA)

How many children does Vito have according to this RDF?



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



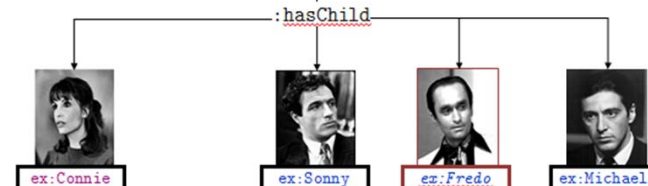
ex:Vito :hasChild ex:Connie, ex:Sonny, ex:Michael .  
ex:Vito :hasChild ex:Fredo .  
... ?

## No Unique Name Assumption (No UNA)

How many children does Vito have according to this RDF?

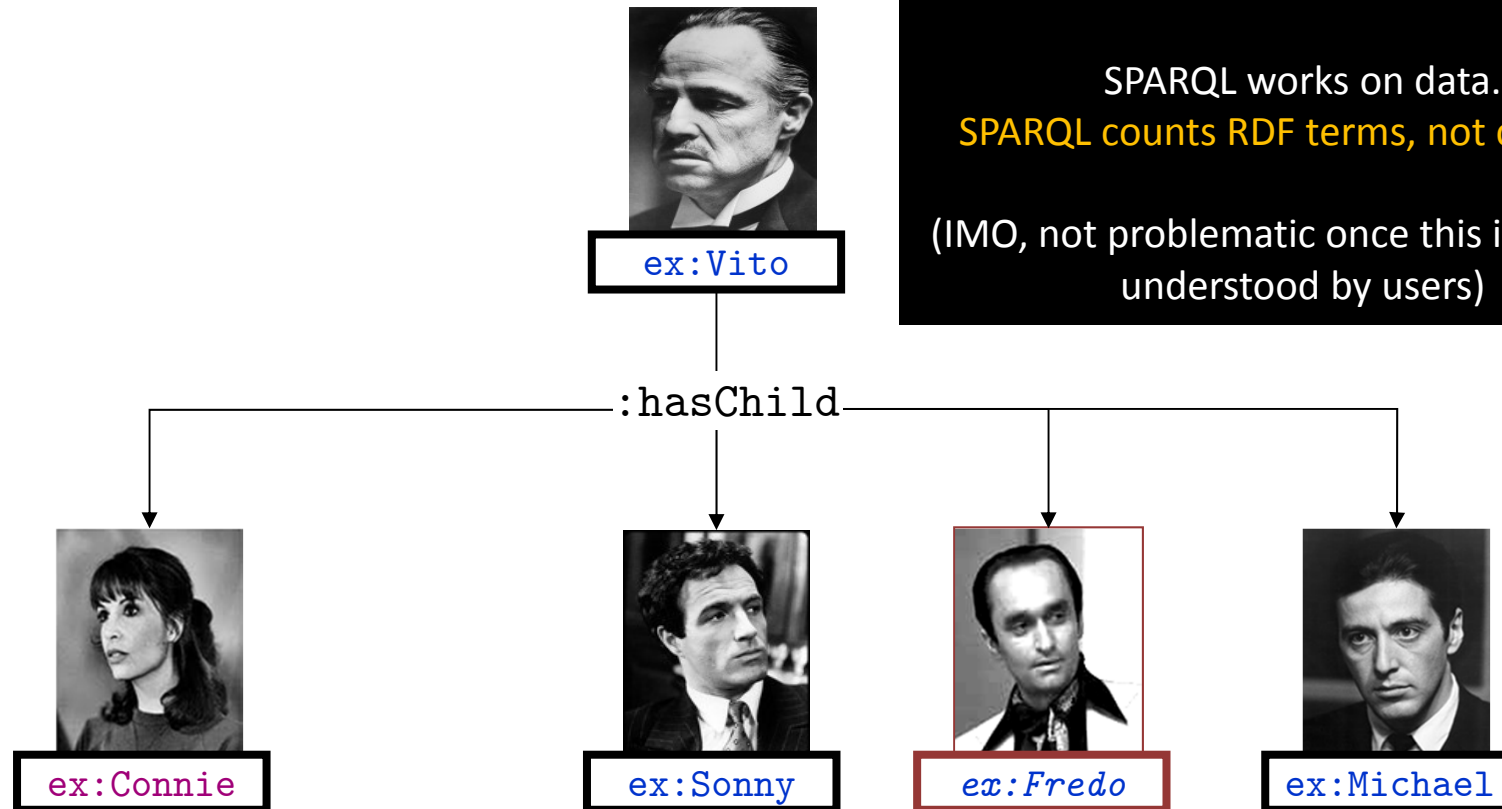


- Vito has 3 children?
- Vito has at least 3 children?
- Vito has at least one child!



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.**

(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 . }
```

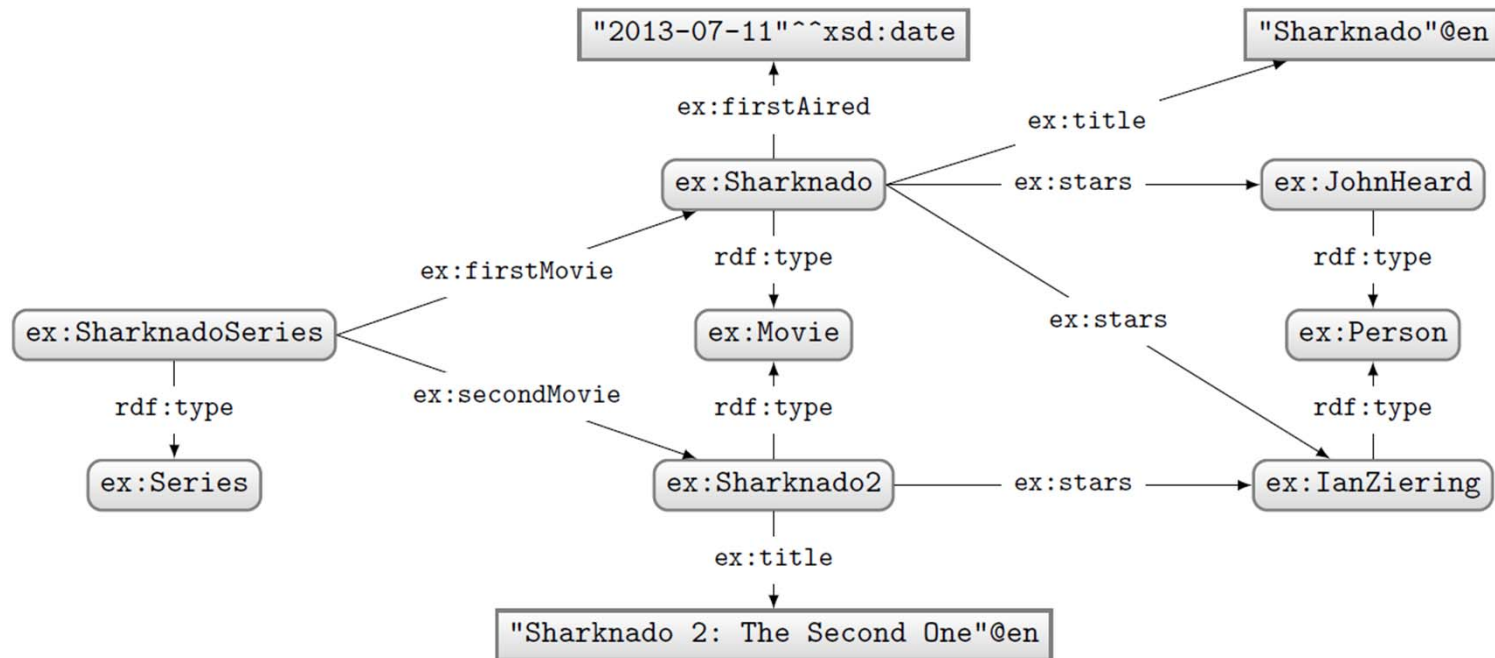
## Solutions:

|        |
|--------|
| ?count |
|--------|

|   |
|---|
| 4 |
|---|

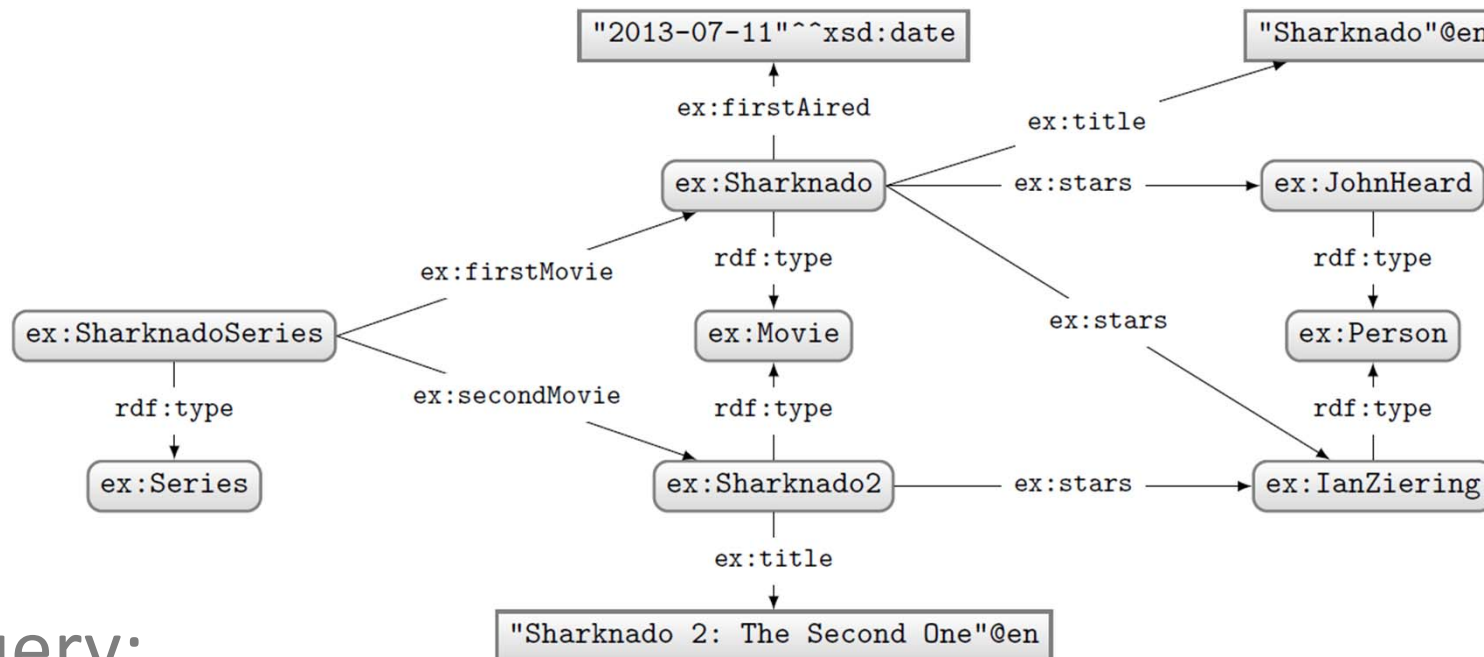
# **NEW QUERY FEATURE: SUBQUERIES**

# 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.

# **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 → E

| A | B | A    B | A && B |
|---|---|--------|--------|
| T | T | T      | T      |
| T | F | T      | F      |
| F | T | T      | F      |
| F | F | F      | F      |
| T | E | T      | E      |
| E | T | T      | E      |
| F | E | E      | F      |
| E | F | E      | F      |
| E | E | E      | E      |



## SPARQL 1.1 functions (Branching)

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

|                            |                                                     |
|----------------------------|-----------------------------------------------------|
| IF(?x = 2, "yes", "no")    | returns "yes"                                       |
| IF(bound(?y), "yes", "no") | returns "no"                                        |
| IF(?x=2, "yes", 1/?z)      | returns "yes", the expression 1/?z is not evaluated |
| IF(?x=1, "yes", 1/?z)      | raises an error                                     |
| IF("2" > 1, "yes", "no")   | raises an error                                     |

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

|                   |                                         |
|-------------------|-----------------------------------------|
| COALESCE(?x, 1/0) | returns 2, the value of x               |
| COALESCE(1/0, ?x) | returns 2                               |
| COALESCE(5, ?x)   | returns 5                               |
| COALESCE(?y, 3)   | returns 3                               |
| COALESCE(?y)      | raises an error because y is not bound. |

# Recall: RDF term functions in SPARQL 1.0

## 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

## Lots more functions (Checking values)

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

|                                         |                 |
|-----------------------------------------|-----------------|
| 2 IN (1, 2, 3)                          | true            |
| 2 IN ()                                 | false           |
| 2 IN (<http://example/iri>, "str", 2.0) | true            |
| 2 IN (1/0, 2)                           | true            |
| 2 IN (2, 1/0)                           | true            |
| 2 IN (3, 1/0)                           | raises an error |

- **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?

|                                                      |                    |
|------------------------------------------------------|--------------------|
| <code>isNumeric(12)</code>                           | <code>true</code>  |
| <code>isNumeric("12")</code>                         | <code>false</code> |
| <code>isNumeric("12"^^xsd:nonNegativeInteger)</code> | <code>true</code>  |
| <code>isNumeric("1200"^^xsd:byte)</code>             | <code>false</code> |
| <code>isNumeric(&lt;http://example/&gt;)</code>      | <code>false</code> |

- **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

## SPARQL 1.1 functions (Strings)

- `STRLEN("abc") = 3`
- `STRSUB("abc",3,1) = "c"`
- `UCASE("shout") = "SHOUT"`
- `LCASE("WHISPER") = "whisper"`
- `STRSTARTS("asd","as") = true`
- `STREND("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

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



## SPARQL 1.1 functions (Numerics)

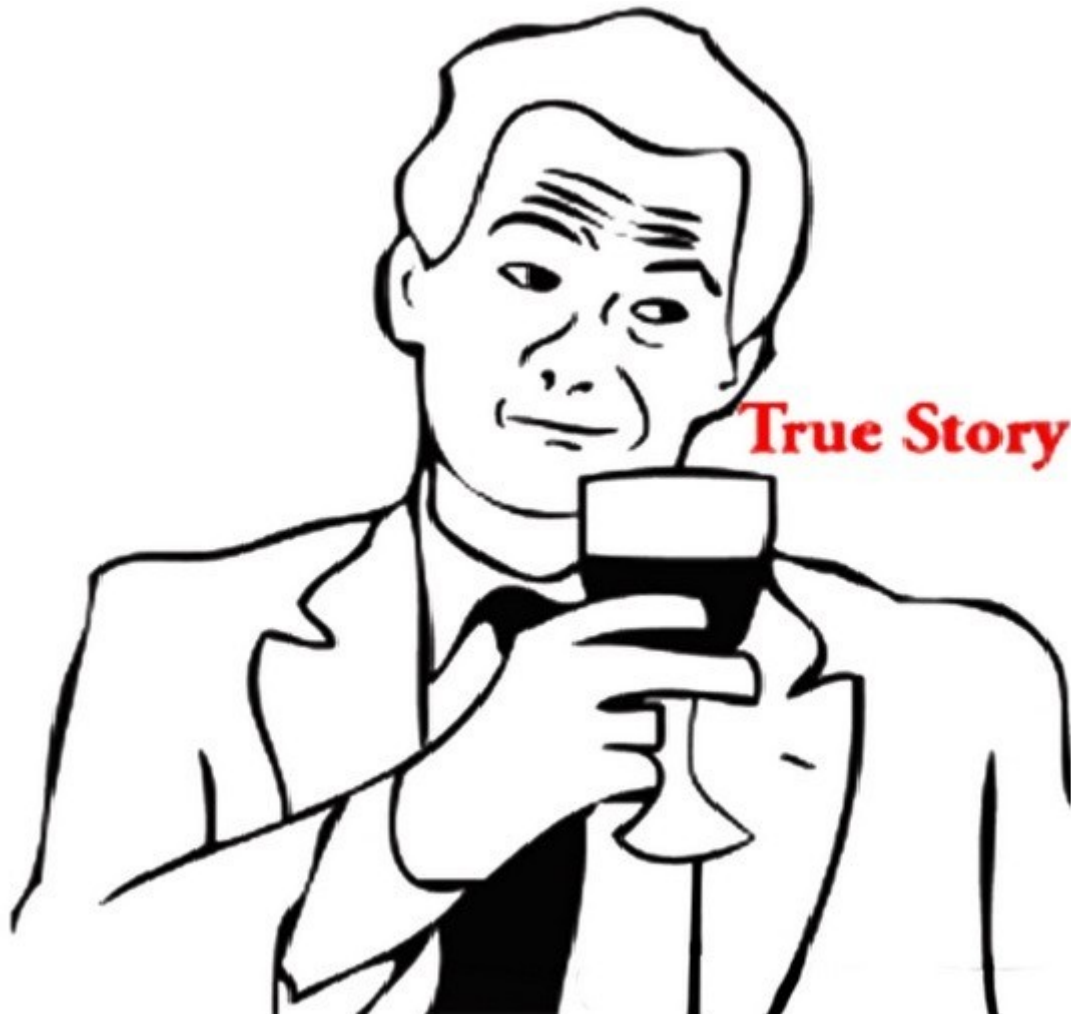
- `ABS(-3.2)` = 3.2
- `ROUND(2.5)` = 3.0
- `CEIL(-2.5)` = -2.0
- `FLOOR(-2.5)` = -3.0
- `RAND()` = 0.5612381239123 ( $0 \leq n < 1$ )

# 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`

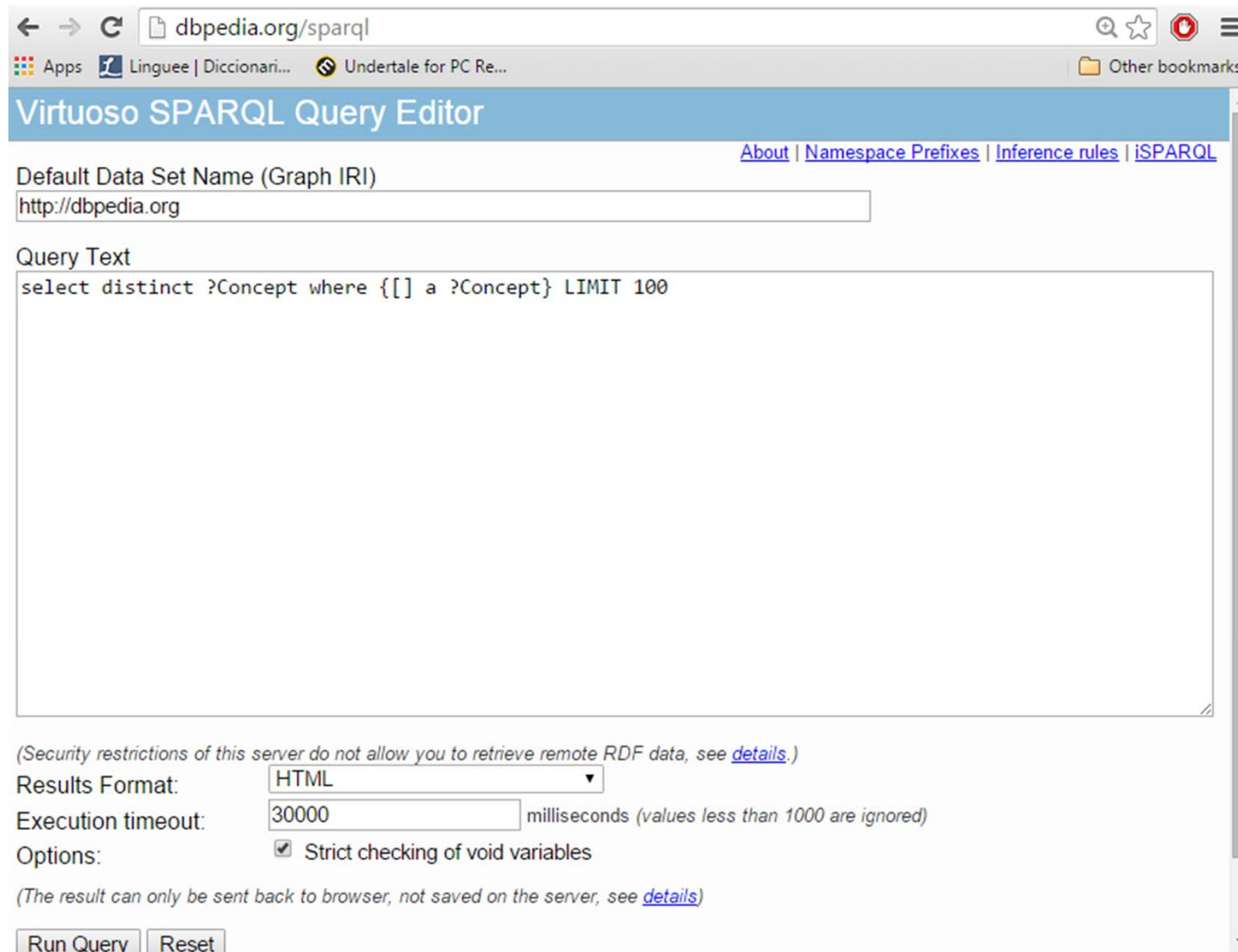


## SPARQL 1.1 functions (Hashes)

- Creates a hash of the input string
  - MD5
  - SHA1
  - SHA256
  - SHA384
  - SHA512

# **NEW QUERY FEATURE: FEDERATION**

# Endpoints often made public/online



The screenshot shows a web browser window with the address bar displaying `dbpedia.org/sparql`. The browser's tab bar shows several open tabs: "Apps", "Linguee | Dictionari...", and "Undertale for PC Re...". The browser's bookmark bar shows "Other bookmarks".

The main content area is titled "Virtuoso SPARQL Query Editor". It features a blue header bar with the title and a navigation menu with links: [About](#), [Namespace Prefixes](#), [Inference rules](#), and [iSPARQL](#).

Below the header, there is a section for "Default Data Set Name (Graph IRI)" with a text input field containing `http://dbpedia.org`.

The "Query Text" section contains a large text area with the following SPARQL query:

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

Below the query text area, there is a note: *(Security restrictions of this server do not allow you to retrieve remote RDF data, see [details](#).)*

The "Results Format" section has a dropdown menu set to "HTML".

The "Execution timeout" section has a text input field containing "30000" and the label "milliseconds (values less than 1000 are ignored)".

The "Options" section has a checkbox labeled "Strict checking of void variables" which is checked.

At the bottom, there is a note: *(The result can only be sent back to browser, not saved on the server, see [details](#).)*

At the very bottom, there are two buttons: "Run Query" and "Reset".

# 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> {
      <http://data.linkedmdb.org/resource/film/675> movie:actor ?actor .
      ?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>

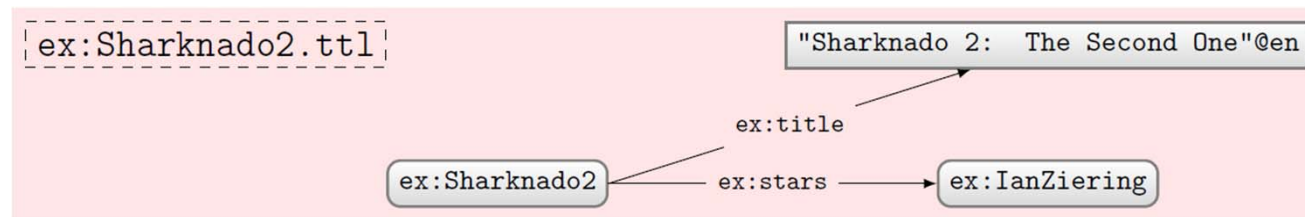
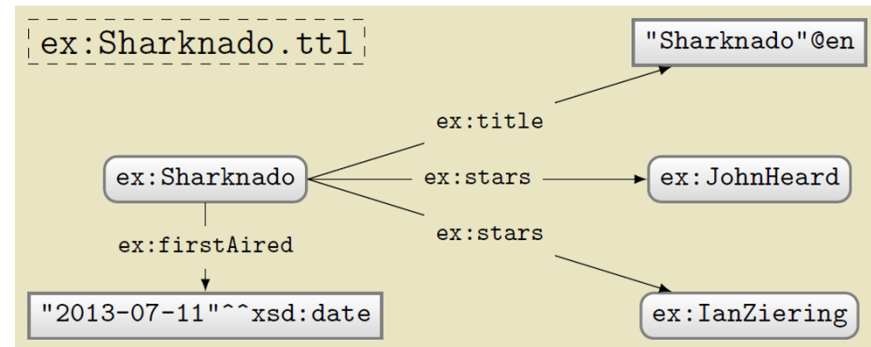
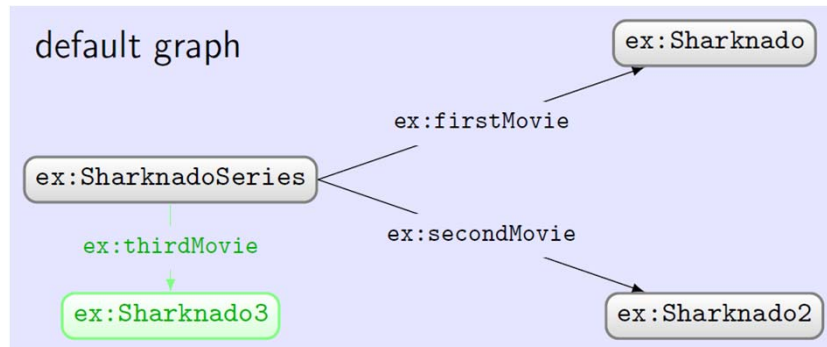
**NEW FEATURE:**  
**SPARQL 1.1 UPDATE**



# What's new in SPARQL 1.1?

- **New query features**
- An update language
- Support for RDFS/OWL entailment
- New output formats

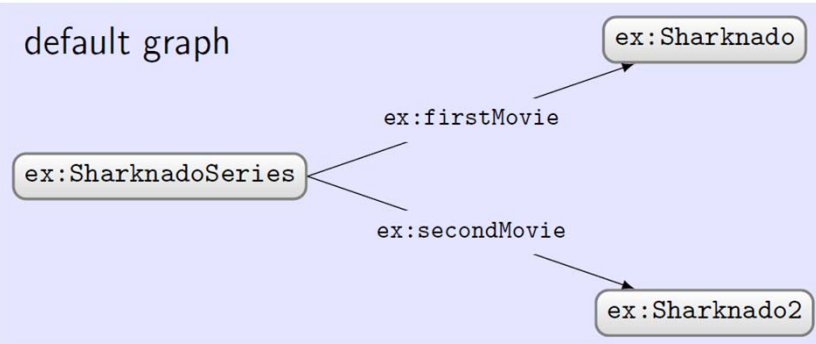
# INSERT DATA default graph



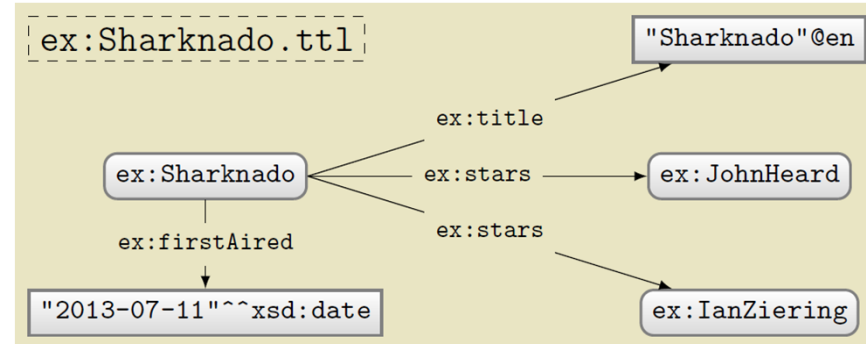
```
PREFIX ex: <http://ex.org/voc#>
INSERT DATA {
  ex:SharknadoSeries ex:thirdMovie ex:Sharknado3 .
}
```

# INSERT DATA named graph

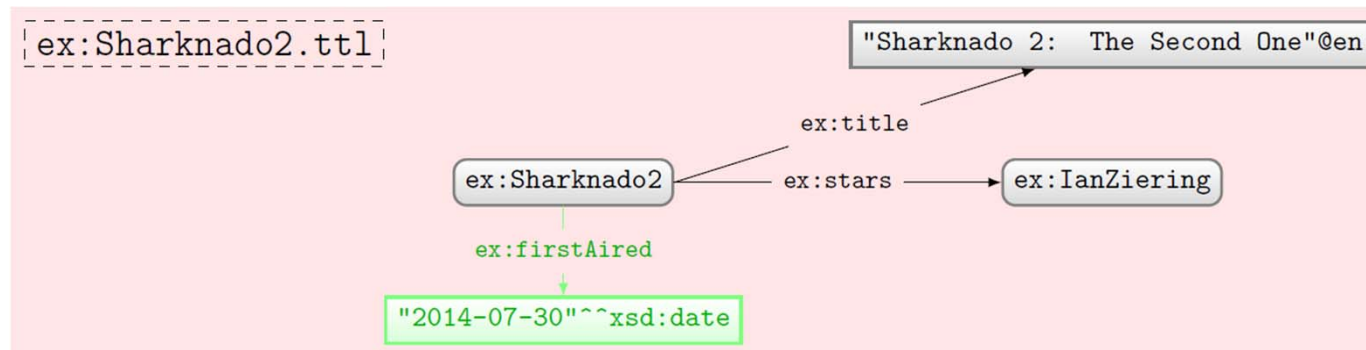
default graph



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 . }
}
```

# Set default update graph: WITH

```
PREFIX ex: <http://ex.org/voc#>
WITH ex:Sharknado2.ttl
DELETE {
    ?movie ex:description ?olddescription .
}
INSERT {
    GRAPH ex:Sharknado { 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 )? 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**

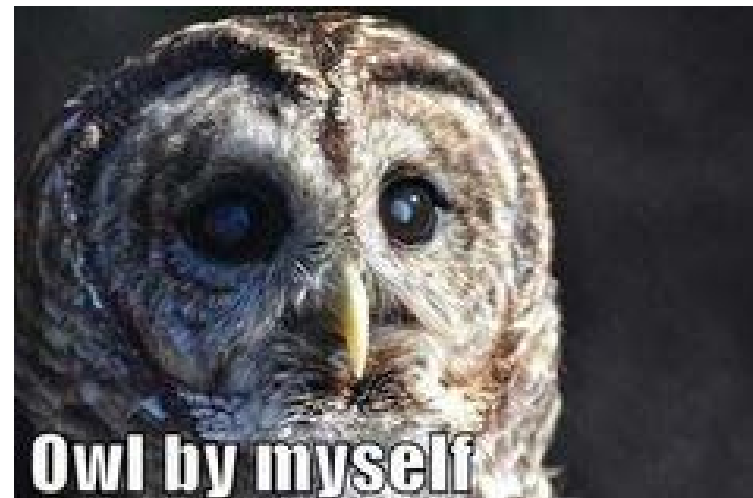


# What's new in SPARQL 1.1?

- **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!

← → ↻ dbpedia.org/sparql 🔍 ☆ 🔴 ☰

Apps Linguee | Diccioni... Undertale for PC Re... Other bookmarks

## Virtuoso SPARQL Query Editor

[About](#) | [Namespace Prefixes](#) | [Inference rules](#) | [iSPARQL](#)

Default Data Set Name (Graph IRI)

Query Text  

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

(Security restrictions of this server do not allow you to retrieve remote RDF data)

Results Format:

Execution timeout:  milliseconds (value)

Options: ☒ Strict checking of void variables

(The result can only be sent back to browser, not saved on the server, see [detail](#))



**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

Questions?

