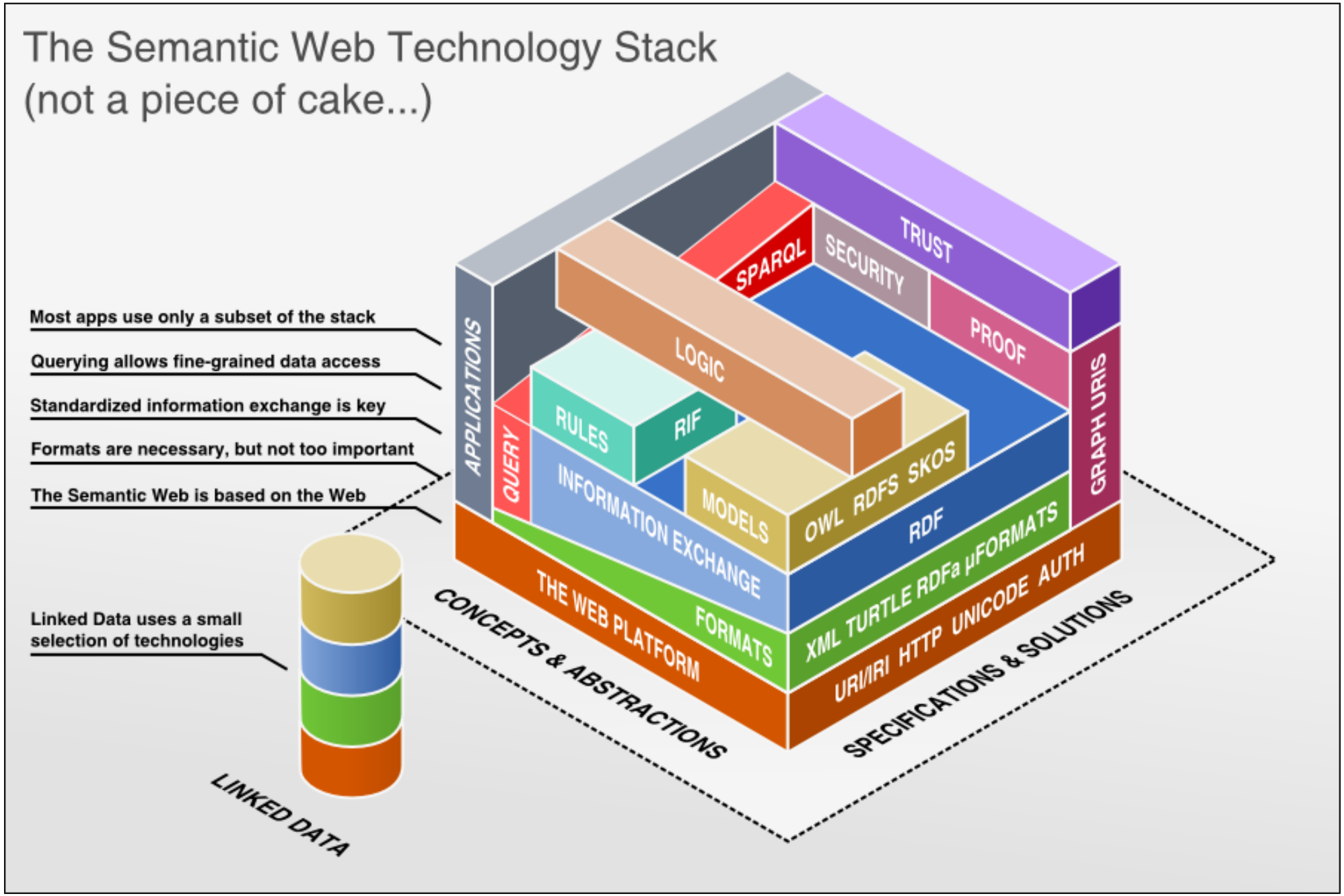
Clase 8.

Sparql

Basado en:
Web of Data, Aidan Hogan (lectures)
Curso Web Semántica, Olga Mariño.
Lectures Prof. Dr. Harald Sack, FIZ Karlsruhe (videos)

The Semantic Web Technology Stack
(not a piece of cake...)



¿Qué es SPARQL?

Sparql ***P***rotocol ***a***nd ***R***DF ***Q***uery ***L***anguaje

- ***R***DF ***Q***uery ***L***anguaje: Lenguaje declarativo de consultas (Queries) sobre bases de datos cuyos datos están almacenados en formato RDF

- ***P***rotocol : operaciones HTTP para que un cliente envíe una consulta a un servicio de SPARQL y este le retorne el resultado de la consulta

SELECT en SPARQL

- Consulta una base de datos
- Selecciona los valores que cumplen una determinada condición (posiblemente compuesta)
- Retorna esa selección (posiblemente en algún formato particular)

Similar a SQL

SELECT en SPARQL

- Consulta una base de datos ***de tripletas RDF***
- Selecciona los valores que cumplen una determinada condición (posiblemente compuesta)… ***unificación de patrones de tripletas***
- Retorna esa selección (posiblemente en algún formato particular)

SELECT estructura

```
PREFIX ...
```
Esquema utilizado en la consulta

```
SELECT ...
```
Lo que se va a retornar

```
WHERE { ... }
```
Patrón de tripletas que se va a buscar para el match

SELECT ejemplo - sintaxis

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

Ubicación de la ontología

Lo que se va a retornar:
3 variables
?nombre

Patrón (lista) de tripletas (4) que se va a buscar
Sintaxis de Turtle
la última tripleta no necesita punto

SELECT ejemplo - significado

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

SELECT ejemplo – unificación

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

SELECT ejemplo - unificación

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

SELECT ejemplo – consulta aplicada a una ontología (base de conocimiento)

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

cancion <= ej:TakeMeHome

autor <= ej:JohnDenver

titulo <= "Take me home, Country Road"

anio <= "12/31/1943"

Autor	*Título*	*Año*
ej:JohnDenver	"Take me home, country roads"	**"1943-12-31"**
ej:JohnDenver	"Annie´s song"	**"1943-12-31"**
ej:TracyChapman	"Subcity"	**"1964-03-30"**

```
…
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie ej:estilo ej:Folk.
ej:Rhapsody      ej:estilo ej:Rock.
ej:Subcity   ej:estilo ej:Folk

ej:TakeMeHome ej:titulo "Take me home, country roads".
ej:TakeMeHome ej:autor ej:JohnDenver.

ej:Annie ej:titulo "Annies's song".
ej:Annie ej:autor ej:JohnDenver.

ej:Rhapsody ej:titulo "Bohemian Rhapsody".
ej:Rhapsody ej:autor ej:Queen.

ej:Subcity ej:titulo "Subcity".
ej:Subcity ej:autor ej:TracyChapman.

ej:JohnDenver ej:fechaNacimiento "1943-12-31" ^^xsd:date.
ej:TracyChapman ej:fechaNacimiento "1964-03-30" ^^xsd:date.
```

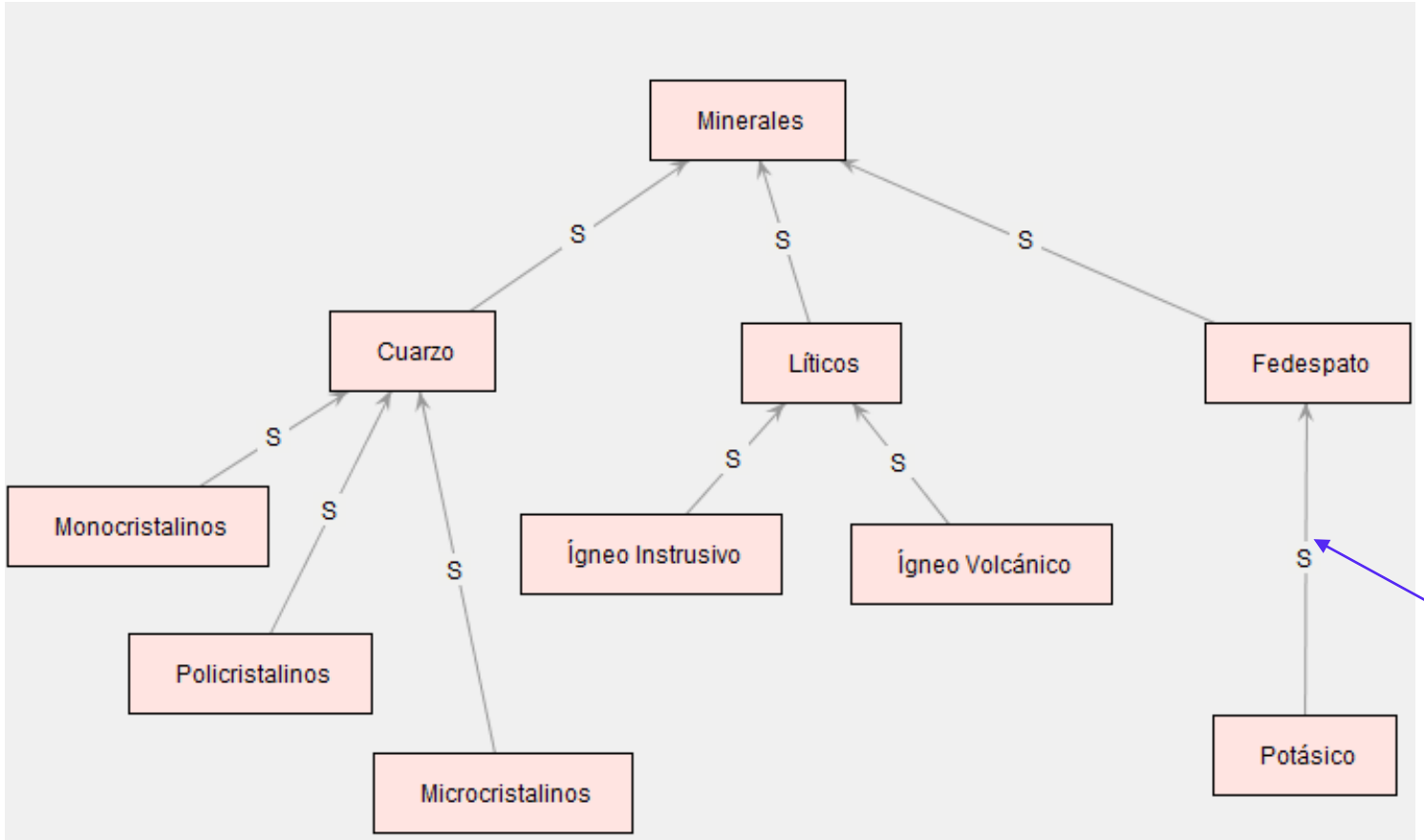
SELECT ejemplo 2: relaciones

```
PREFIX dbr:<http://dbpedia.org/resource/>
PREFIX dbp: <http://dbpedia.org/property/>

SELECT ?relacion

WHERE { dbr:Facebook ?relación dbr:Mark_Zuckerberg}
```

Relación
dbp:founder
dbp:CEO
dbp:author

SELECT ejemplo 3: taxonomía

¿Cómo encontrar las relaciones taxonómicas en una ontología que deseemos consultar?



Subclase	*Superclase*
Cuarzo	Minerales
Monocristalinos	Cuarzo
Policristalonos	Cuarzo
Microcristalinos	Cuarzo
Líticos	Minerales
…..	

rdfs:SubClassOf

SELECT ejemplo 3: taxonomía

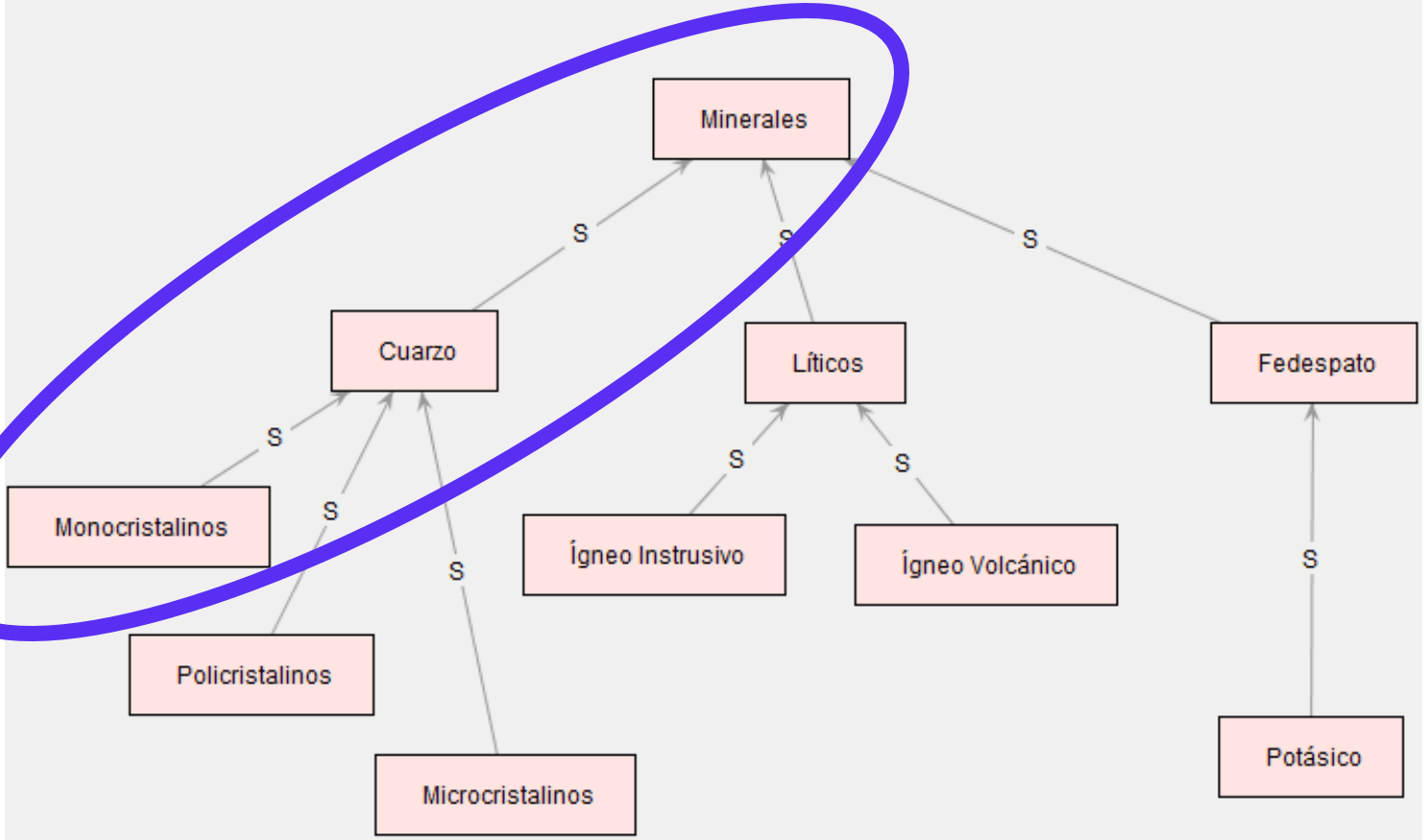
```
PREFIX ej3:<http://ejemplo3.org>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?sujeto ?objeto

WHERE { ?sujeto rdfs:subClassOf ?objeto}
```

SELECT ejemplo 3: taxonomía

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?sujeto ?objeto

FROM  <http://ejemplo3.org>

WHERE { ?sujeto rdfs:subClassOf ?objeto}
```

SELECT ejemplo 3: taxonomía con todas las superclases

¿Cómo encontrar todas las superclases de una clase?



Subclase	*Superclase*
Cuarzo	Minerales
Monocristalinos	Cuarzo
Monocristalino	Minerales

SELECT ejemplo 3: taxonomía encontrando todas las superclases

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?sujeto ?objeto

FROM  <http://ejemplo3.org>

WHERE { ?sujeto rdfs:subClassOf* ?objeto}
```

*

Después del nombre de una propiedad
en la cláusula WHERE,
hace que se encuentren todas las tripletas
encadenadas por la ocurrencia de esa propiedad

FILTROS de SELECT

```
PREFIX ...
```
Esquema utilizado en la consulta

```
SELECT ...
```
Lo que se va a retornar

```
WHERE { ...
        ...
        FILTER ( )
      }
```
Patrón de tripletas que se va a buscar para el match

Condición para ser incluida

SELECT ejemplo

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

Queremos recuperar sólo los nacidos en la década de los 60s

SELECT ejemplo con FILTRO

```
PREFIX ej:http://ejemplo.org …

SELECT ?autor ?titulo ?anio
WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
        FILTER (
         ?anio >= “1960-01-01”^^xsd:date &&
         ?anio < “1970-01-01”^^xsd:date
        )
      }
```

A	Op	B	Return type and value	
	`!`	BOOL b	BOOL	`true` if $I_L(b)$ is false; `false` otherwise
BOOL b_1	`||`	BOOL b_2	BOOL	`true` if $I_L(b_1)$ or $I_L(b_2)$; `false` otherwise
BOOL b_1	`&&`	BOOL b_2	BOOL	`true` if $I_L(b_1)$ and $I_L(b_2)$; `false` otherwise
TERM* t_1	`=`	TERM* t_2	BOOL	`true` if t_1 same term as t_2; `false` otherwise
TERM* t_1	`!=`	TERM* t_2	BOOL	`true` if t_1 not same term as t_2; `false` otherwise
COM v_1	`=`	COM v_2	BOOL	`true` if $I_L(v_1) = I_L(v_2)$; `false` otherwise
COM v_1	`!=`	COM v_2	BOOL	`true` if $I_L(v_1) \neq I_L(v_2)$; `false` otherwise
COM v_1	`<`	COM v_2	BOOL	`true` if $I_L(v_1) < I_L(v_2)$; `false` otherwise
COM v_1	`>`	COM v_2	BOOL	`true` if $I_L(v_1) > I_L(v_2)$; `false` otherwise
COM v_1	`<=`	COM v_2	BOOL	`true` if $I_L(v_1) \leq I_L(v_2)$; `false` otherwise
COM v_1	`>=`	COM v_2	BOOL	`true` if $I_L(v_1) \geq I_L(v_2)$; `false` otherwise
	`+`	NUM n	NUM	n
	`-`	NUM n	NUM	$-n$
NUM n_1	`+`	NUM n_2	NUM	$I_L(v_1) + I_L(v_2)$
NUM n_1	`-`	NUM n_2	NUM	$I_L(v_1) + I_L(v_2)$
NUM n_1	`*`	NUM n_2	NUM	$I_L(v_1) \times I_L(v_2)$
NUM n_1	`/`	NUM n_2	NUM	$\frac{I_L(v_1)}{I_L(v_2)}$

Function	Return type and value	
`strlen`(STR s)	INT	length of string s
`substr`(STR s,INT b,[INT l])	STR	substring of s from index b [of length l]
`ucase`(STR s)	STR	uppercase s
`lcase`(STR s)	STR	lowercase s
`strstarts`(STR s,STR p)	BOOL	`true` if s starts with p; `false` otherwise
`strends`(STR s,STR p)	BOOL	`true` if s ends with p; `false` otherwise
`strbefore`(STR s,STR p)	STR	string before first match for p in s
`strafter`(STR s,STR p)	STR	string after first match for p in s
`encode_for_iri`(STR s)	STR	s percent-encoded
`concat`(STR s_1,...,s_n)	STR	$s_1, ..., s_n$ concatenated
`langMatches`(STR s,STR l)	BOOL	`true` if s a language tag matching l; `false` otherwise
`regex`(STR s,STR p[,STR f])	BOOL	`true` if s matches regex p [with flags f]; `false` otherwise
`replace`(STR s,STR p,STR r[,STR f])	STR	s with matches for regex p [with flags f] replaced by r

Function	Return type and value	
`abs`(NUM n)	NUM	absolute value of n
`round`(NUM n)	NUM	round to nearest whole number (towards $+\infty$ for *.5)
`ceil`(NUM n)	NUM	round up (towards $+\infty$) to nearest whole number
`floor`(NUM n)	NUM	round down (towards $-\infty$) to nearest whole number
`rand`(NUM n)	NUM	random double between 0 (inclusive) and 1 (exclusive)

Function	Return type and value	
`bound`(TERM t)	BOOL	`true` if t is bound; `false` if unbound
`if`(BOOL b,TERM t_1,TERM t_2)	TERM	t_1 if b is `true`; t_2 otherwise
`coalesce`(TERM t_1,...,t_n)	TERM	first t_i $(1 \leq i \leq n)$ that is not an error or unbound
`not exists`(SUB Q)	BOOL	`true` if Q has any solution; `false` otherwise
`exists`(SUB Q)	BOOL	`true` if Q has no solution; `false` otherwise
`sameTerm`(TERM t_1, TERM t_2)	BOOL	`true` if t_1 same term as t_2; `false` otherwise
TERM t `in`(TERM t_1,...,t_n)	BOOL	`true` if $t = t_i$ for any $t_i \in \{t_1, ..., t_n\}$; `false` otherwise
TERM t `not in`(TERM t_1,...,t_n)	BOOL	`true` if $t \neq t_i$ for all $t_i \in \{t_1, ..., t_n\}$; `false` otherwise

Function	Return type and value	
`isIRI`(TERM t)	BOOL	`true` if t is an IRI; `false` otherwise
`isBlank`(TERM t)	BOOL	`true` if t is a blank node; `false` otherwise
`isLiteral`(TERM t)	BOOL	`true` if t is a literal; `false` otherwise
`isNumeric`(TERM t)	BOOL	`true` if t is a numeric value; `false` otherwise
`str`(LIT l|IRI i)	STR	lexical value of l | string of i
`lang`(LIT l)	STR	language tag string of l
`datatype`(LIT l)	IRI	datatype IRI of l
`iri`(STR s|IRI i)	IRI	s resolved against the in-scope base IRI | i
`bnode`([STR s])	BNODE	fresh blank node [unique to s]
`strdt`(STR s,IRI i)	LIT	"s"^^<i>
`strlang`(STR s,STR l)	LIT	"s"@l
`uuid`()	IRI	fresh IRI (from UUID URN scheme)
`struuid`()	STR	fresh string (from UUID URN scheme)

SELECT ejemplo con FILTRO

```
PREFIX ej:<http://ejemplo.org>
SELECT ?autor ?titulo ?anio
WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
        FILTER (
            ?anio >= "1960-01-01"^^xsd:date &&
            ?anio < "1970-01-01"^^xsd:date
        )
      }
```

Autor	*Título*	*Año*
ej:TracyChapman	"Subcity"	**"1964-03-30"**

...
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie ej:estilo ej:Folk.
ej:Rhapsody ej:estilo ej:Rock.
ej:Subcity ej:estilo ej:Folk

ej:TakeMeHome ej:titulo "Take me home, country roads".
ej:TakeMeHome ej:autor ej:JohnDenver.

ej:Annie ej:titulo "Annies's song".
ej:Annie ej:autor ej:JohnDenver.

ej:Rhapsody ej:titulo "Bohemian Rhapsody".
ej:Rhapsody ej:autor ej:Queen.

ej:Subcity ej:titulo "Subcity".
ej:Subcity ej:autor ej:TracyChapman.

ej:JohnDenver ej:fechaNacimiento "1943-12-31" ^^xsd:date.
ej:TracyChapman ej:fechaNacimiento "1964-03-30" ^^xsd:date.

OPCIONAL de SELECT

```
PREFIX ...
SELECT ...
WHERE { ...
        ...
        OPTIONAL ( )
      }
```

Esquema utilizado en la consulta

Lo que se va a retornar

Patrón de tripletas que tiene que unificarse

Si no se cumple igual se retorna la respuesta parcial de la unificación de las otras tripletas

SELECT ejemplo

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento ?anio
      }
```

SELECT ejemplo

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        ?autor ej:fechaNacimiento
?anio
      }
```

Queremos recuperar también los autores y títulos de canciones Folk así no se tenga el año del cantante

```
…
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie      ej:estilo ej:Folk.
ej:Rhapsody   ej:estilo ej:Rock.
ej:Subcity    ej:estilo ej:Folk

ej:TakeMeHome ej:titulo "Take me home, country roads".
ej:TakeMeHome ej:autor ej:JohnDenver.

ej:Annie ej:titulo "Annies's song".
ej:Annie ej:autor ej:JohnDenver.

ej:Rhapsody ej:titulo "Bohemian Rhapsody".
ej:Rhapsody ej:autor ej:Queen.

ej:Subcity ej:titulo "Subcity".
ej:Subcity ej:autor ej:TracyChapman.

ej:JohnDenver ej:fechaNacimiento "1943-12-31" ^^xsd:date.
ej:TracyChapman ej:fechaNacimiento "1964-03-30" ^^xsd:date.
```

SELECT ejemplo con OPCIONAL

```
PREFIX ej:<http://ejemplo.org>

SELECT ?autor ?titulo ?anio

WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        OPTIONAL (?autor ej:fechaNacimiento ?anio)
      }
```

SELECT ejemplo con OPCIONAL

```
PREFIX ej:<http://ejemplo.org>
SELECT ?autor ?titulo ?anio
WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
        OPTIONAL (
            ?autor ej:fechaNacimiento ?anio)
       )
      }
```

Autor	*Título*	*Año*
ej:JohnDenver	"Take me home, country roads"	**"1943-12-31"**
ej:JohnDenver	"Annie´s song"	**"1943-12-31"**
ej:TracyChapman	"Subcity"	

…
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie ej:estilo ej:Folk.
ej:Rhapsody ej:estilo ej:Rock.
ej:Subcity ej:estilo ej:Folk

ej:TakeMeHome ej:titulo "Take me home, country roads".
ej:TakeMeHome ej:autor ej:JohnDenver.

ej:Annie ej:titulo "Annies's song".
ej:Annie ej:autor ej:JohnDenver.

ej:Rhapsody ej:titulo "Bohemian Rhapsody".
ej:Rhapsody ej:autor ej:Queen.

ej:Subcity ej:titulo "Subcity".
ej:Subcity ej:autor ej:TracyChapman.

ej:JohnDenver ej:fechaNacimiento "1943-12-31" ^^xsd:date.
~~ej:TracyChapman ej:fechaNacimiento "1964-03-30" ^^xsd:date.~~

UNIÓN de SELECT

```
PREFIX  ...
SELECT  ...
WHERE  {  ..A..
          {..B..  }
        UNION
          {..C..  }
  }
```

Esquema utilizado en la consulta

Lo que se va a retornar

Puede haber tripletas antes de la unión

La consulta puede ser unificando A y B o unificando A y C. Retorna todas AB y AC

SELECT Ejemplo

```
PREFIX ej:<http://ejemplo.org>

SELECT ?nombre ?id

WHERE { ?contribuyente ej:nombre
ej:?nombre.
        ?contribuyente ej:cedula  ?id
      }
```

ej:nit

Queremos recuperar también los contribuyentes (empresas) que no tienen cédula sino nit

SELECT Ejemplo con UNION

```
PREFIX ej:<http://ejemplo.org>

SELECT ?nombre ?id

WHERE { ?contribuyente ej:nombre
ej:?nombre.
        { ?contribuyente ej:cedula  ?id}
        UNION
        { ?contribuyente ej:nit  ?id}
      }
```

MODIFICADORES DE SECUENCIAS

- La solución de una consulta SPARQL es una secuencia de unificaciones de las variables de la consulta

Autor	*Título*	*Año*
ej:JohnDenver	"Take me home, country roads"	`"1943-12-31"`
ej:JohnDenver	"Annie´s song"	`"1943-12-31"`
ej:TracyChapman	"Subcity"	

- Los MODIFICADORES manipulan esa secuencia para generar otra
  - Ordenada
  - Sin valores repetidos
  - Con menos registros…

MODIFICADORES de secuencias de SELECT

```
PREFIX ...
```
Esquema utilizado en la consulta

```
SELECT ...
```
Lo que se va a retornar

```
WHERE { ...
      }
```
Patrón de tripletas que se va a buscar para el match

```
modificador1..
modificador2..
```
Modifican el resultado de la consulta

SELECT Ejemplo con MODIFICADORES DE SECUENCIA

Retorne nombre e identificación de los contribuyentes 1001 a 1300, según orden de identificación

```
PREFIX ej:<http://ejemplo.org>
SELECT ?nombre ?id
WHERE { ?contribuyente ej:nombre ej:?nombre.
         { ?contribuyente ej:cedula  ?id}
        UNION
         { ?contribuyente ej:nit  ?id}
      }
ORDER BY ?id
LIMIT 300
OFFSET 1000
```

ASK

```
PREFIX ...
```

Esquema utilizado en la consulta

```
ASK
```

Retorna True o False / yes o no

```
WHERE { ...
        ...
      }
```

Consulta: si se logra instanciar, la respuesta es verdadera, si no es falsa

Ejemplo

Se desea saber si en la base de conocimientos hay alguna canción Folk para la que se conozca su autor y su título

…
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie ej:estilo ej:Folk.
ej:Rhapsody ej:estilo ej:Rock.
ej:Subcity ej:estilo ej:Folk

ej:TakeMeHome ej:titulo "Take me home, country roads".
ej:TakeMeHome ej:autor ej:JohnDenver.

ej:Annie ej:titulo "Annies's song".
ej:Annie ej:autor ej:JohnDenver.

ej:Rhapsody ej:titulo "Bohemian Rhapsody".
ej:Rhapsody ej:autor ej:Queen.

ej:Subcity ej:titulo "Subcity".
ej:Subcity ej:autor ej:TracyChapman.

ej:JohnDenver ej:fechaNacimiento "1943-12-31" ^^xsd:date.
ej:TracyChapman ej:fechaNacimiento "1964-03-30" ^^xsd:date.

ASK

```
PREFIX ej:<http://ejemplo.org>
ASK
WHERE { ?cancion ej:estilo ej:Folk.
        ?cancion ej:autor  ?autor.
        ?cancion ej:titulo ?titulo.
      }
```

```
…
ej:TakeMeHome ej:estilo ej:Folk.
ej:Annie      ej:estilo ej:Folk.
ej:Rhapsody               ej:estilo ej:Rock.
ej:Subcity    ej:estilo ej:Folk

ej:TakeMeHome ej:titulo “Take me home, country roads”.
ej:TakeMeHome ej:autor ej:JohnDenver.

…
```

CONSTRUCT

```
PREFIX ...

CONSTRUCT
          {...
           ...
        }
WHERE  {  ...
          ...
        }
```

Esquema utilizado en la consulta

Retorna un grafo RDF para cada resultado encontrado

Tripletas con variables

Consulta

CONSTRUCT

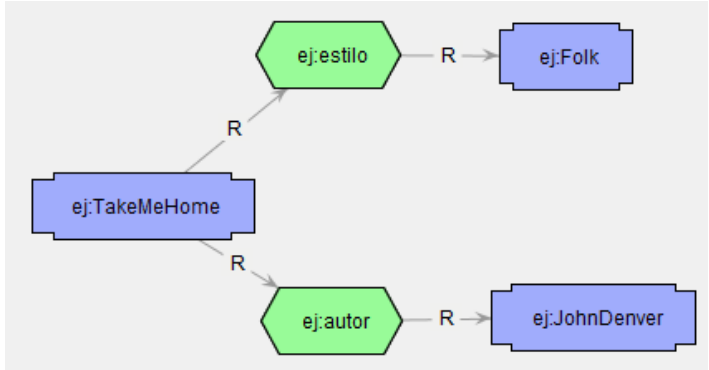
Para generar nuevas relaciones / tripletas

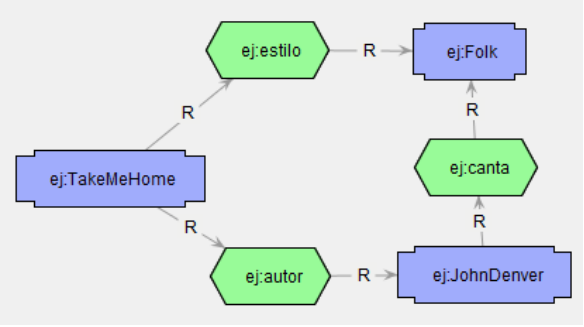
```
PREFIX ej:<http://ejemplo.org>
CONSTRUCT
        {
         ?autor ej:creacion ?cancion
         }
WHERE {
           ?cancion ej:autor  ?autor.
         }
```

CONSTRUCT

Para generar nuevas relaciones / tripletas

```
PREFIX ej:<http://ejemplo.org>
CONSTRUCT
        {
            ?autor ej:canta ?estilo
         }
WHERE { ?cancion ej:estilo ?estilo.
            ?cancion ej:autor  ?autor.
        }
```





CONSTRUCT

Para expresar reglas de inferencia del dominio

- `Si un artista A nació en un país B y creó una obra C => C es patrimonio cultural de B`

CONSTRUCT

Para expresar reglas de inferencia del dominio

```
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX ej:<http://ejemplo.org>

CONSTRUCT { ?obra  ej:patrimonio ?pais }

WHERE {
                ?obra dbp:author ?artista
                ?artista dbp:citizenship ?pais
        }
```

CONSTRUCT

Para unificar vocabulario de una ontología

- Usar el mismo predicado, por ejemplo: id para el identificador de un contribuyente independientemente de si este identificador es la cédula o el nit

CONSTRUCT

Para unificar vocabulario de una ontología

```
PREFIX ej:<http://ejemplo.org>

CONSTRUCT { ?contribuyente ej:id ?id }

WHERE {
          { ?contribuyente ej:cedula  ?id}
        UNION
          { ?contribuyente ej:nit  ?id}
      }
```

CONSTRUCT

Para hacer explícitas relaciones de inferencia

- Si X es instancia de C1, y C1 es subclase de C2, entonces x es instancia de C2

CONSTRUCT

Para hacer explícitas relaciones de inferencia

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

CONSTRUCT { ?instancia rdf:type ?superClase }

FROM  <http://ejemplo3.org>

WHERE {
                ?instancia rdf:type ?clase
                ?clase rdfs:subClassOf* ?superClase
        }
```

CONSTRUCT

Para establecer conexiones entre ontologías

- Por ejemplo para conectar las personas de mi ontología local con su equivalente en DBPedia
  => supongamos que si el nombre y fecha de nacimiento coinciden en ambas ontologías, es la misma persona

Inferencia de relaciones implícitas de instanciación

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX dbp: <http://dbpedia.org/property/>
PREFIX dbo: < http://dbpedia.org/ontology/>
PREFIX ej:<http://ejemplo.org>

```
CONSTRUCT { ?x owl:sameAs ?y }

WHERE {
        ?x rdf:type dbo:Person
        ?y rdf:type ej:Persona
        ?x dbp:birthDate ?fecha
        ?x dbp:name ?nombre
       ?y ej:fechaNacimiento ?fecha
        ?y ej:nombre ?nombre
    }
```

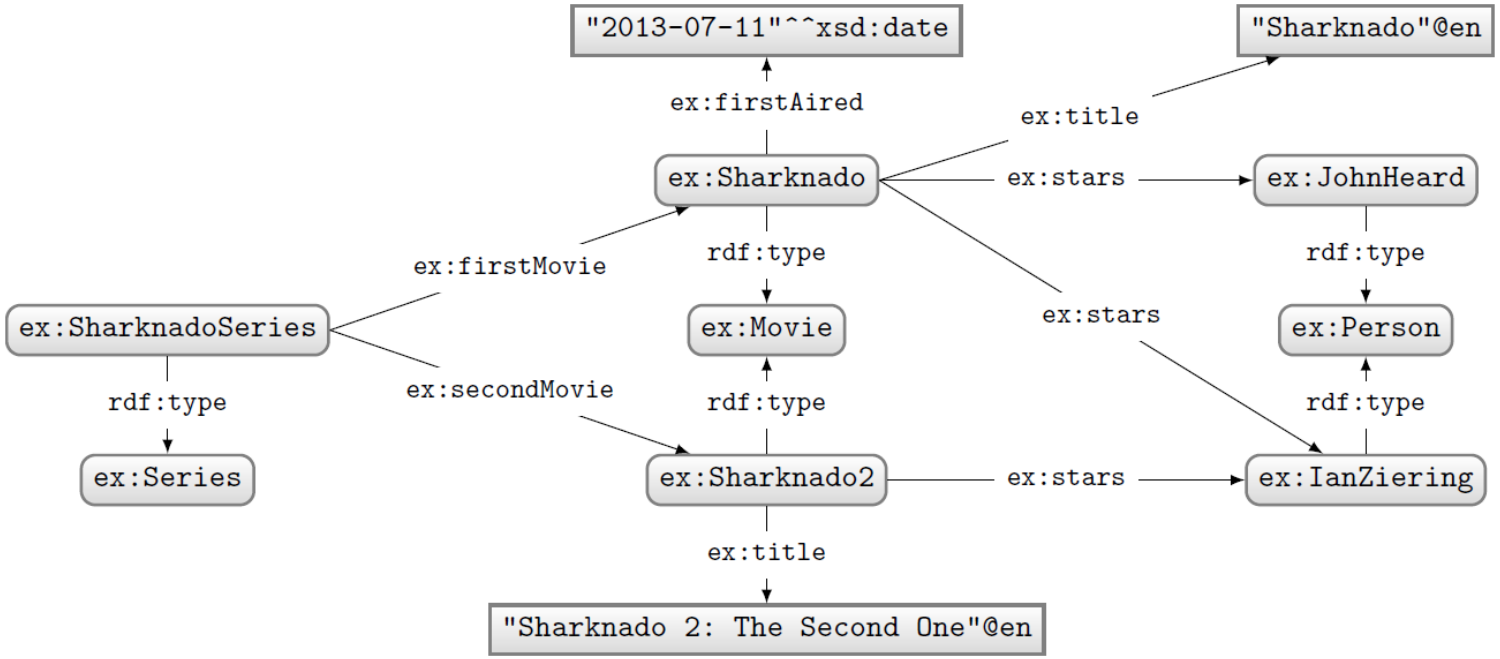
Breve reflexión sobre eficiencia

```
PREFIX …

CONSTRUCT { ?x owl:sameAs ?y }
WHERE {
            ?x rdf:type dbo:Person
            ?y rdf:type ex:Persona
            ?x dbp:birthDate ?fecha
            ?x dbp:name ?nombre
         ?y ej:fechaNacimiento ?fecha
            ?y ej:nombre ?nombre
       }
```

```
PREFIX …

CONSTRUCT { ?x owl:sameAs ?y }
WHERE {
            ?y rdf:type ex:Persona
            ?y ej:fechaNacimiento ?fecha
            ?y ej:nombre ?nombre
            ?x dbp:birthDate ?fecha
            ?x dbp:name ?nombre
            ?x rdf:type dbo:Person
       }
```

Ejercicio 1

Cómo preguntar: "¿Cuáles son los títulos de las (dos primeras) películas de la serie Sharknado?"



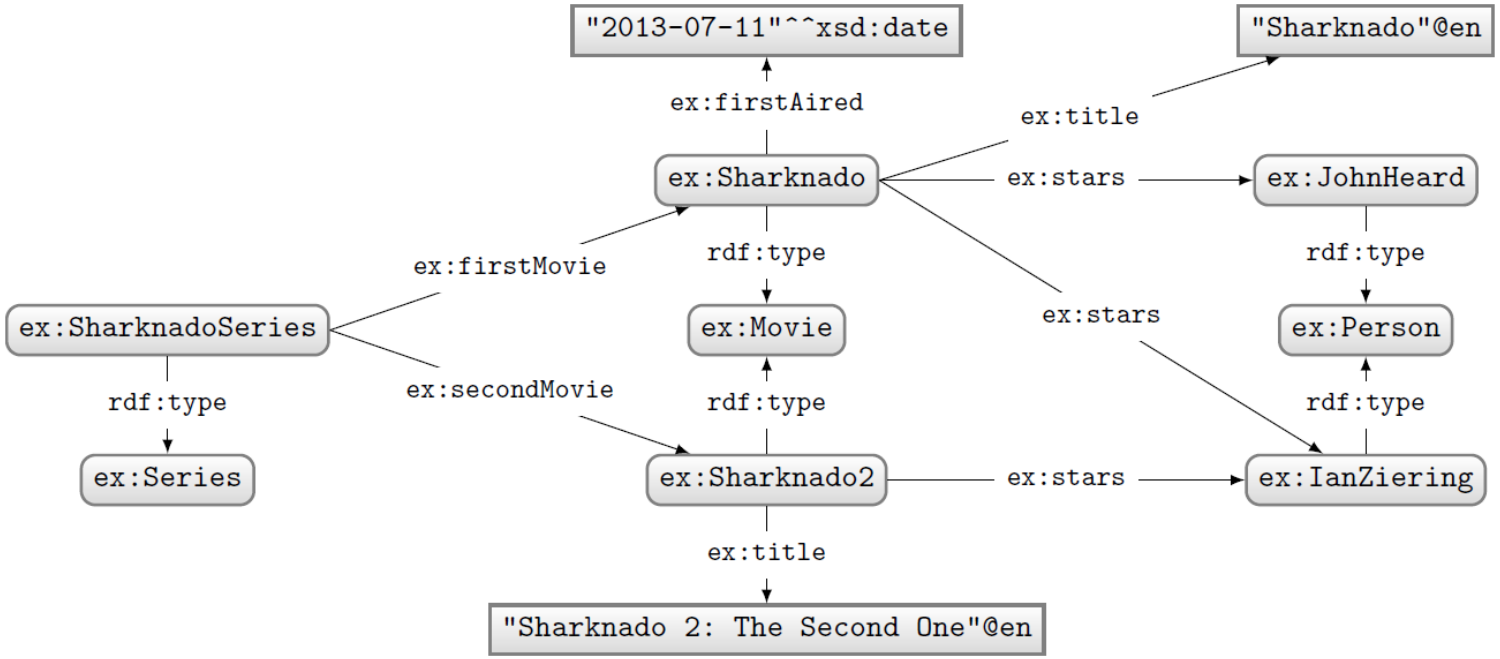

```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  { ex:SharknadoSeries ex:firstMovie ?movie . }
  UNION
  { ex:SharknadoSeries ex:secondMovie ?movie . }
  ?movie ex:title ?title .
}
```

Solutions:

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

Ejercicio 2

Cómo preguntar: "¿Dame los títulos de todas las películas y, si está disponible, su fecha de estreno?"




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

Solutions:

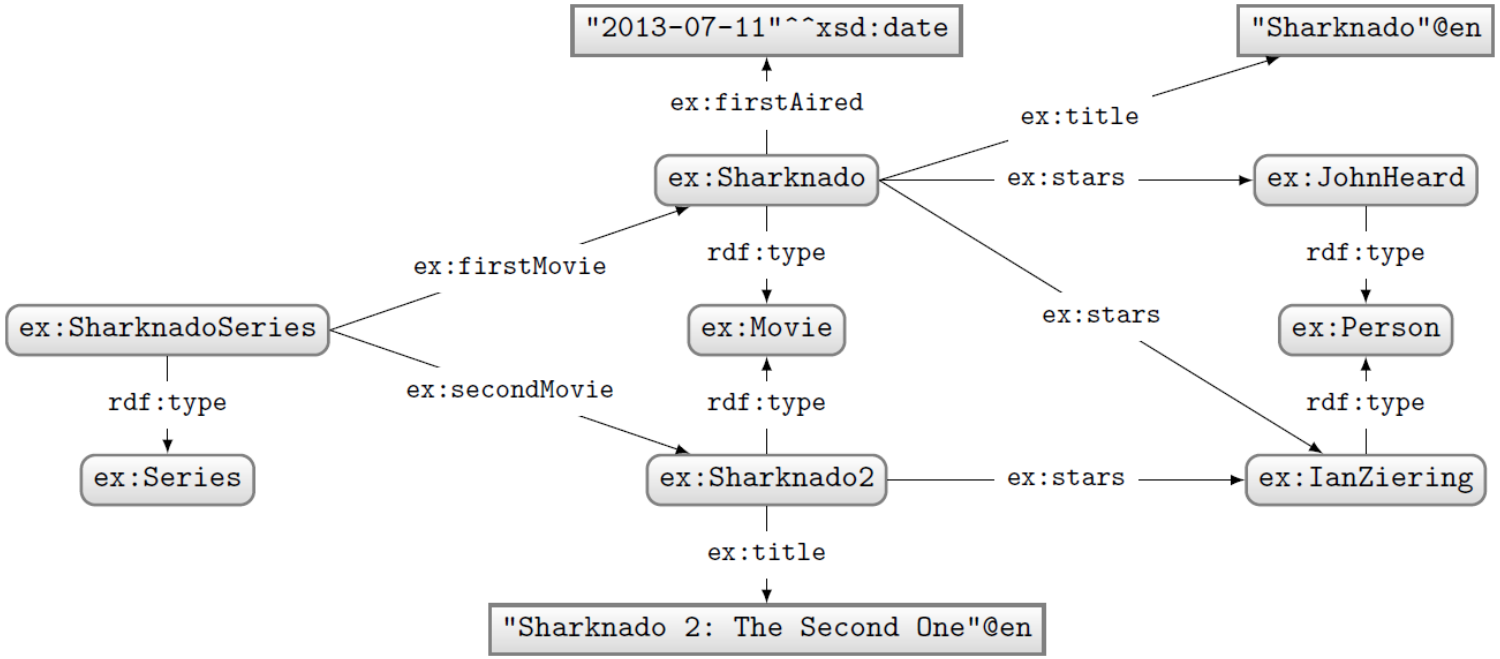
?movie	?title	?date
ex:Sharknado	"Sharknado"@en	"2013-07-11"^^xsd:date
ex:Sharknado2	"Sharknado 2: The Second One"@en	

"UNBOUND Variable"

(a variable without a binding in a solution)

Ejercicio 3

Cómo preguntar: "¿Qué películas se emitieron por primera vez en 2014?"

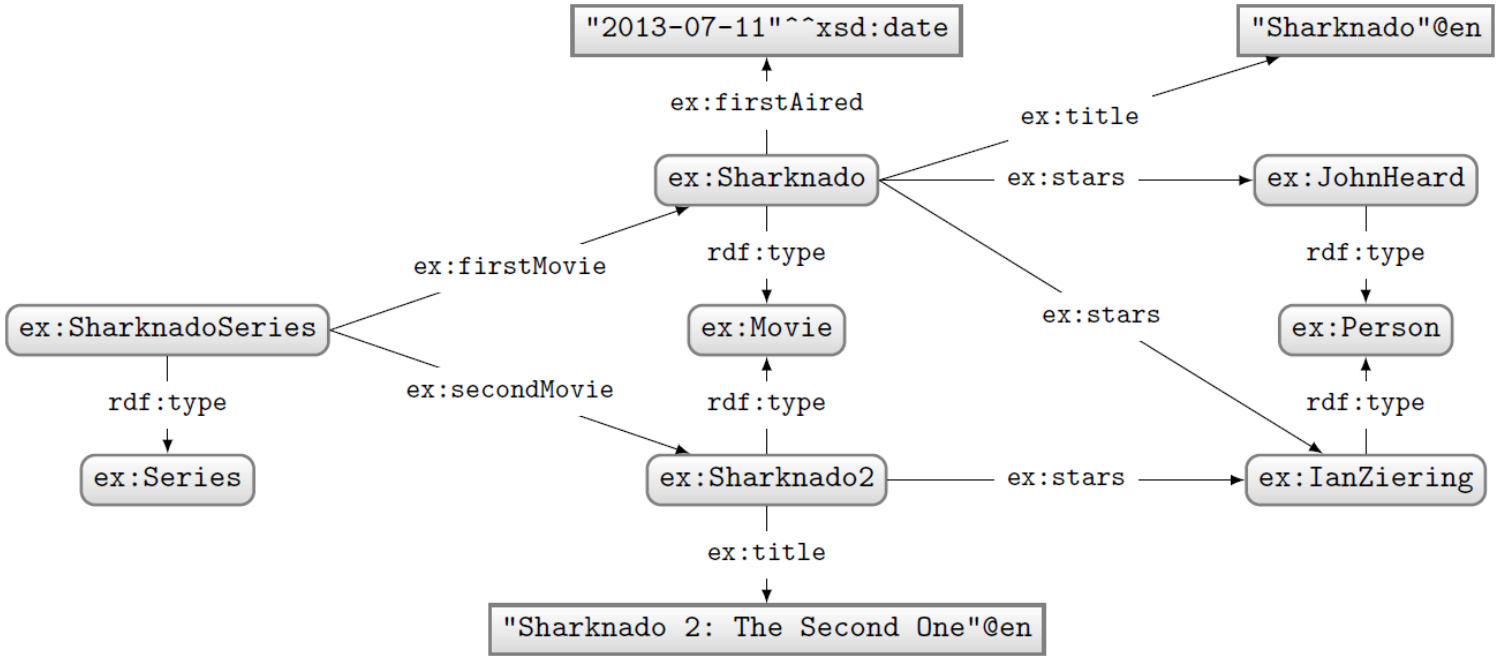



```
PREFIX ex: <http://ex.org/voc#>
SELECT *
WHERE {
  ?movie a ex:Movie ; ex:firstAired ?date .
  FILTER(?date > "2013-12-31"^^xsd:date
        && ?date <= "2014-12-31"^^xsd:date)
}
```

Solutions:

?movie	?date

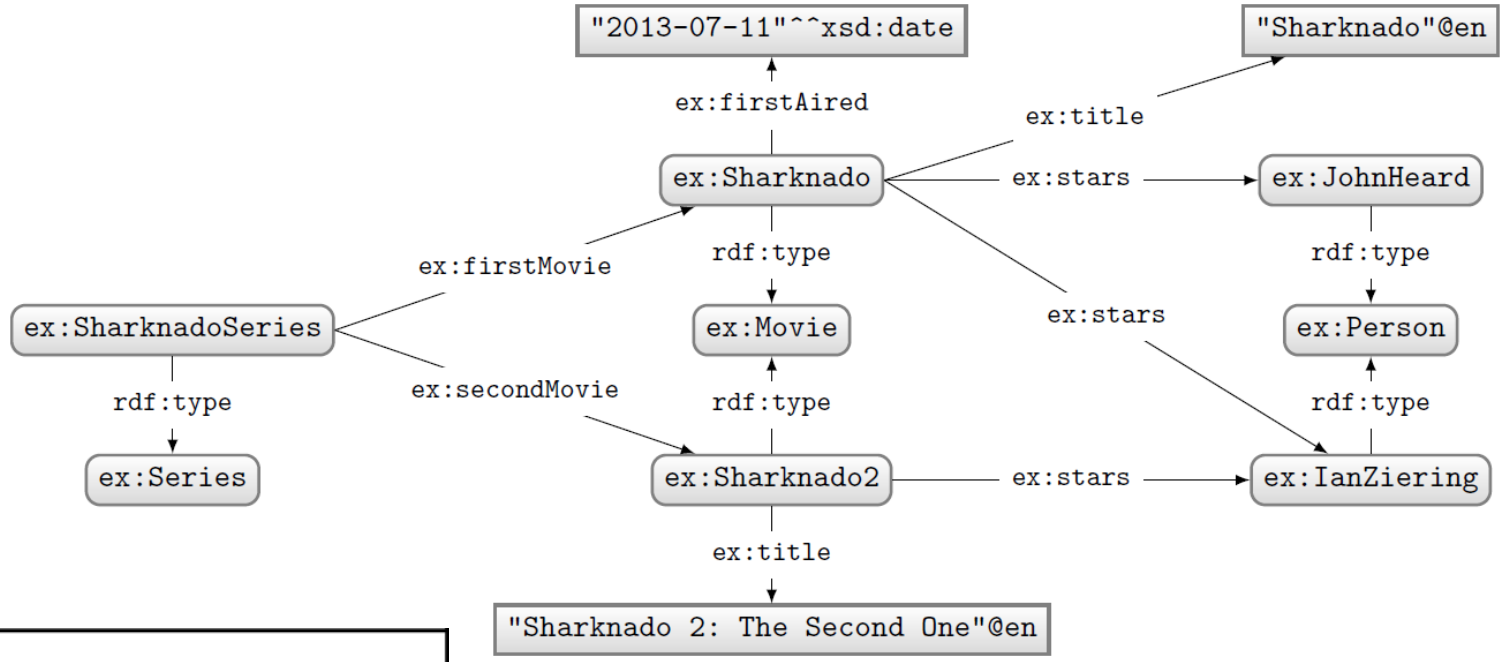
Ejercicio 4




```
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]*"))
}
```

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

Ejercicio 5




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

?movie	?date
ex:Sharknado2	

Can do negation!

NO! also persons who know something else !

```
PREFIX ex: <http://inria.fr/schema#>
SELECT ?name
WHERE {
  ?person ex:name ?name .
  ?person ex:knows ?x
  FILTER ( ?x != "Java" )
}
```

fabien *ex:knows* ***"Java"***

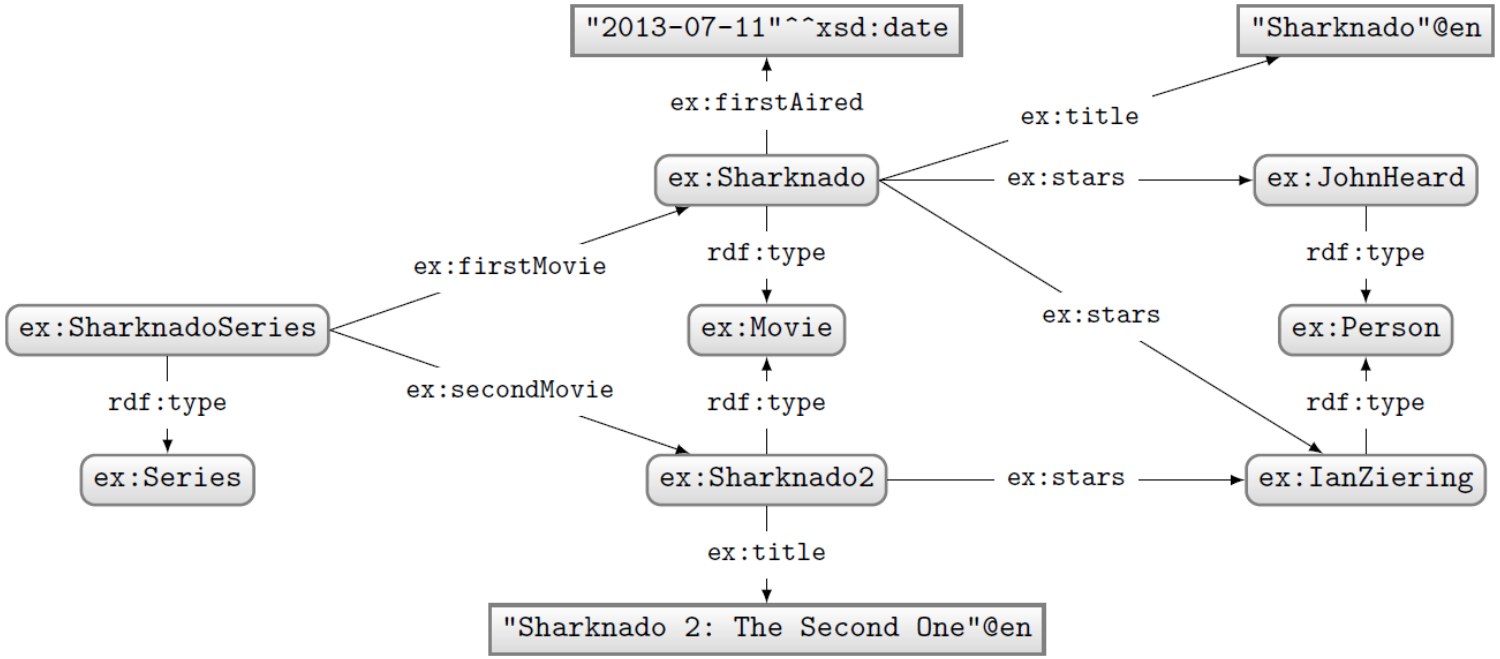
fabien *ex:knows* ***"C++"***

fabien is a answer...

YES! persons who are not known to know "java" ... negation of an option...

```
PREFIX ex: <http://inria.fr/schema#>
SELECT ?name
WHERE {
 ?person ex:name ?name .
 OPTIONAL { ?person ex:knows ?x
  FILTER ( ?x = "Java" ) }
 FILTER ( ! bound(?x) )
}
```

Ejercicio X

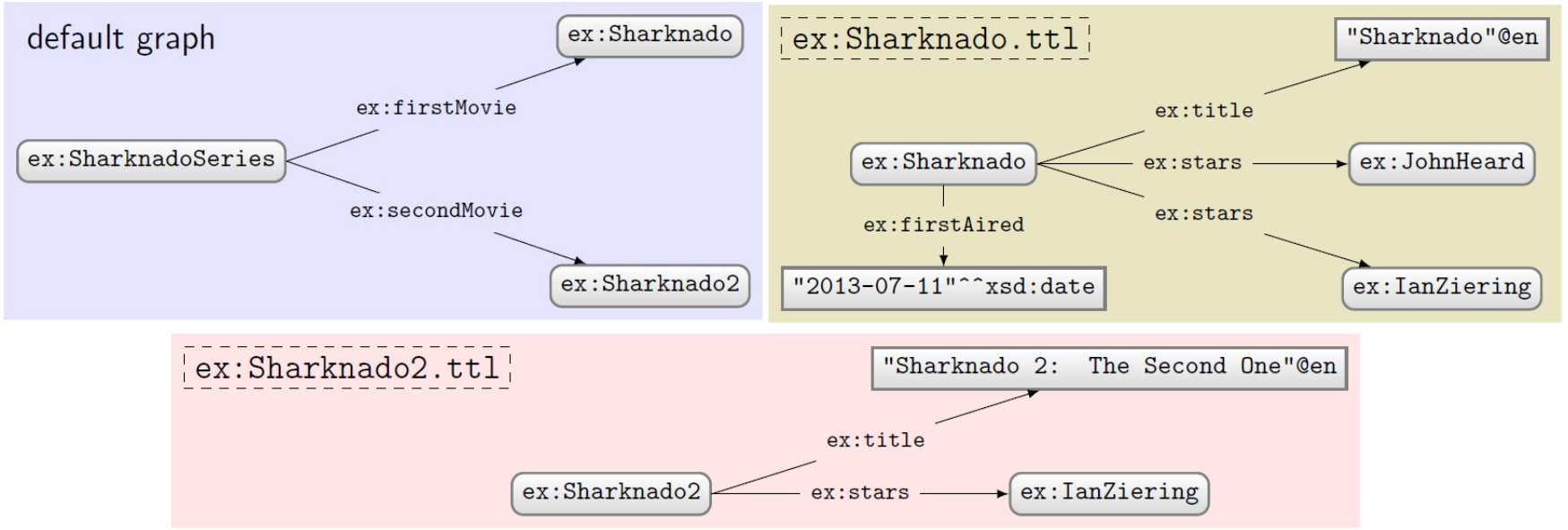



```
PREFIX ex: <http://ex.org/voc#>
SELECT DISTINCT ?star
WHERE {
  ?movie a ex:Movie.
  ?movie ex:stars ?star .
}
```

?star
ex:JohnHeard
ex:IanZiering

?star
ex:JohnHeard
ex:IanZiering

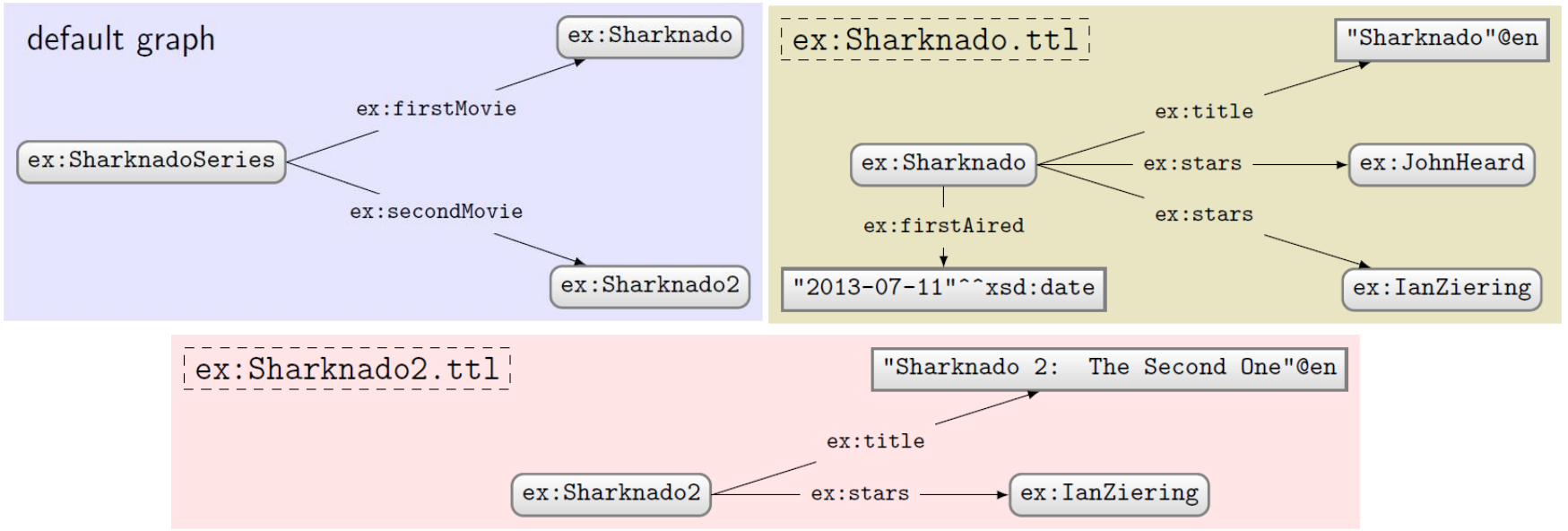
Ejercicio X




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

?s
ex:SharknadoSeries

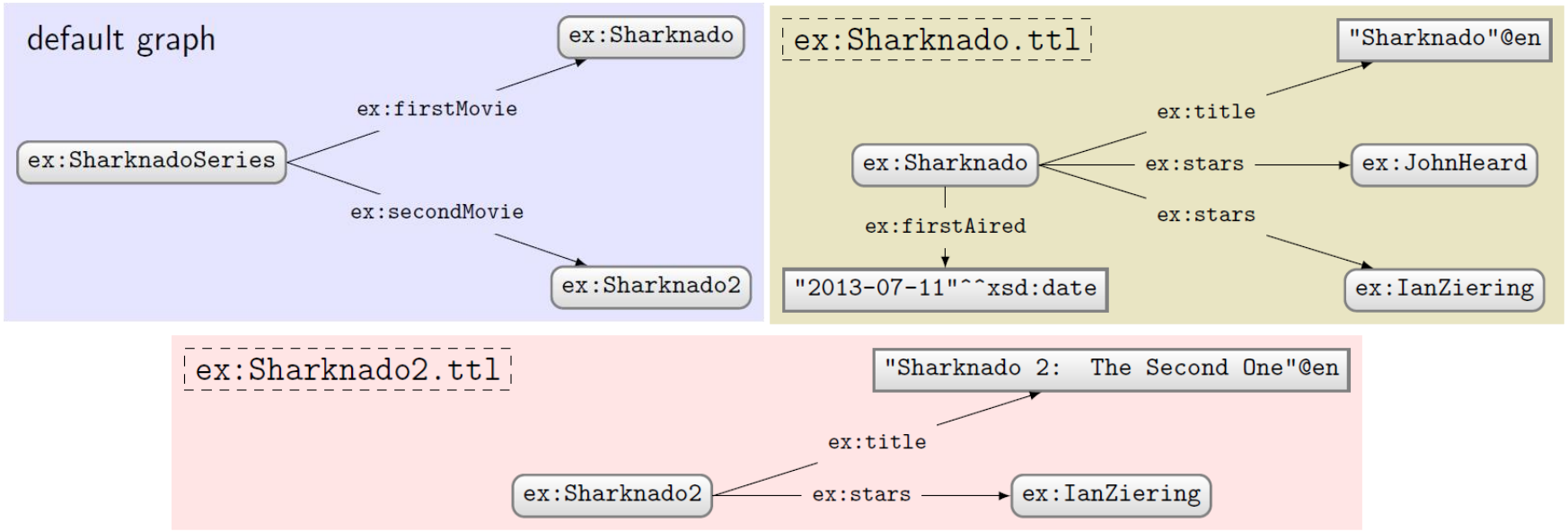
Ejercicio X




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

?s
ex:Sharknado
ex:Sharknado2

Ejercicio X




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

?s
ex:Sharknado