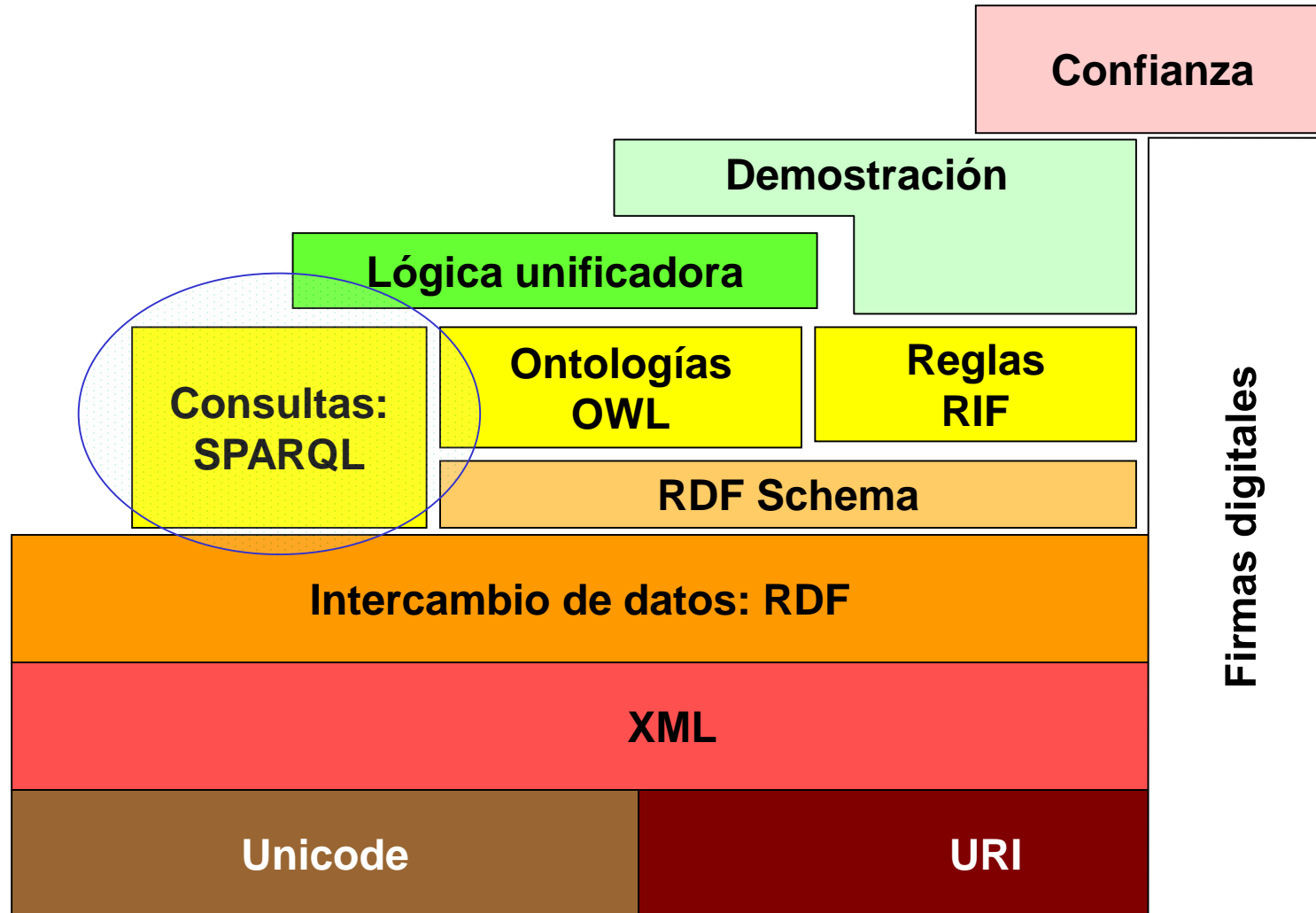


SPARQL

Jose Emilio Labra Gayo

Departamento de Informática
Universidad de Oviedo

SPARQL



SPARQL

Los ficheros RDF pueden considerarse bases de datos de tripletas

SPARQL (Abril 2006) es un lenguaje de consulta para datos RDF

- Similar a SQL para RDF

- Lenguaje de consultas

 - Basado en RDQL

 - Modelo = patrones sobre grafos

- También describe un protocolo de transporte

SPARQL 1.1 (2013, recomendación)

- Actualizaciones, consultas federadas, etc.

SPARQL

Sintaxis Turtle

Sintaxis similar a N3

URLs entre < >

`<http://www.uniovi.es>`

Prefijos para espacios de nombres

`PREFIX x: <http://www.alumnos.org/>`
`x:profesor`

Nodos anónimos

`_:nombre` ó `[]`

Literales entre " "

`"Jose", "234"^^xsd:integer`

Variables empiezan por ?

`?nombre`

Comentarios empiezan por #

`# esto es un comentario`

Nota: En N3 se ponía @prefix
Declaraciones de prefijos no terminan en punto

RDF

RDF = Modelo de grafo

Diferentes sintaxis: N3, Turtle, RDF/XML

Ejemplo en Turtle

```
@prefix dc: <http://purl.org/dc/terms/> .
@prefix uni: <http://uniovi.es/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

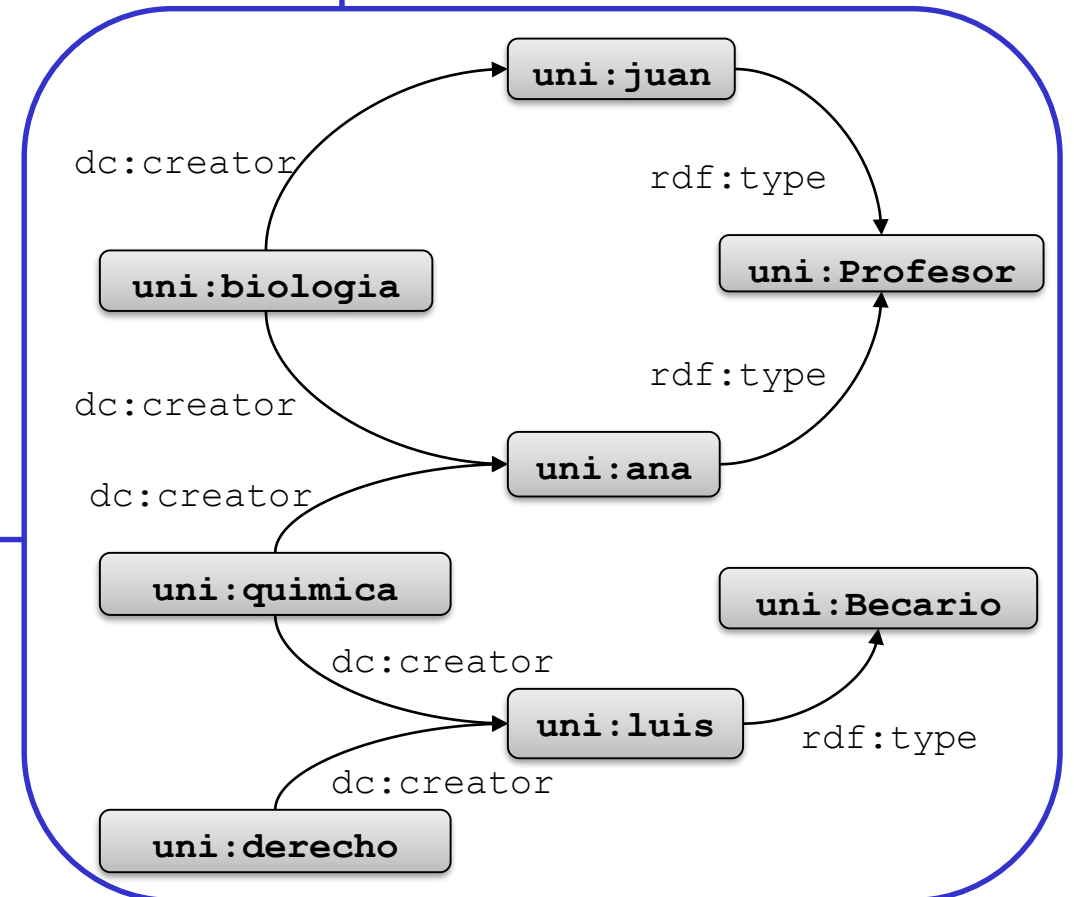
uni:biologia dc:creator uni:juan .
uni:biologia dc:creator uni:ana .
uni:quimica dc:creator uni:ana .
uni:quimica dc:creator uni:luis .
uni:derecho dc:creator uni:luis .
uni:ana rdf:type uni:Profesor .
uni:juan rdf:type uni:Profesor .
uni:luis rdf:type uni:Becario .
```

Grafo RDF

Ejemplo en Turtle

```
@prefix dc: <http://purl.org/dc/terms/> .  
@prefix uni: <http://uniovi.es/> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
```

```
uni:biologia dc:creator uni:juan .  
uni:biologia dc:creator uni:ana .  
uni:quimica dc:creator uni:ana .  
uni:quimica dc:creator uni:luis .  
uni:derecho dc:creator uni:luis .  
uni:ana rdf:type uni:Profesor .  
uni:juan rdf:type uni:Profesor .  
uni:luis rdf:type uni:Becario .
```



Consulta RDF

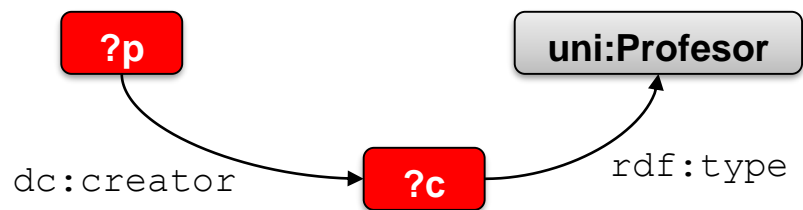
Buscar páginas creadas por un profesor

```
PREFIX dc: <http://purl.org/dc/terms/>
PREFIX uni: <http://uniovi.es/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?p ?c WHERE {
  ?p dc:creator ?c .
  ?c rdf:type uni:Profesor .
}
```

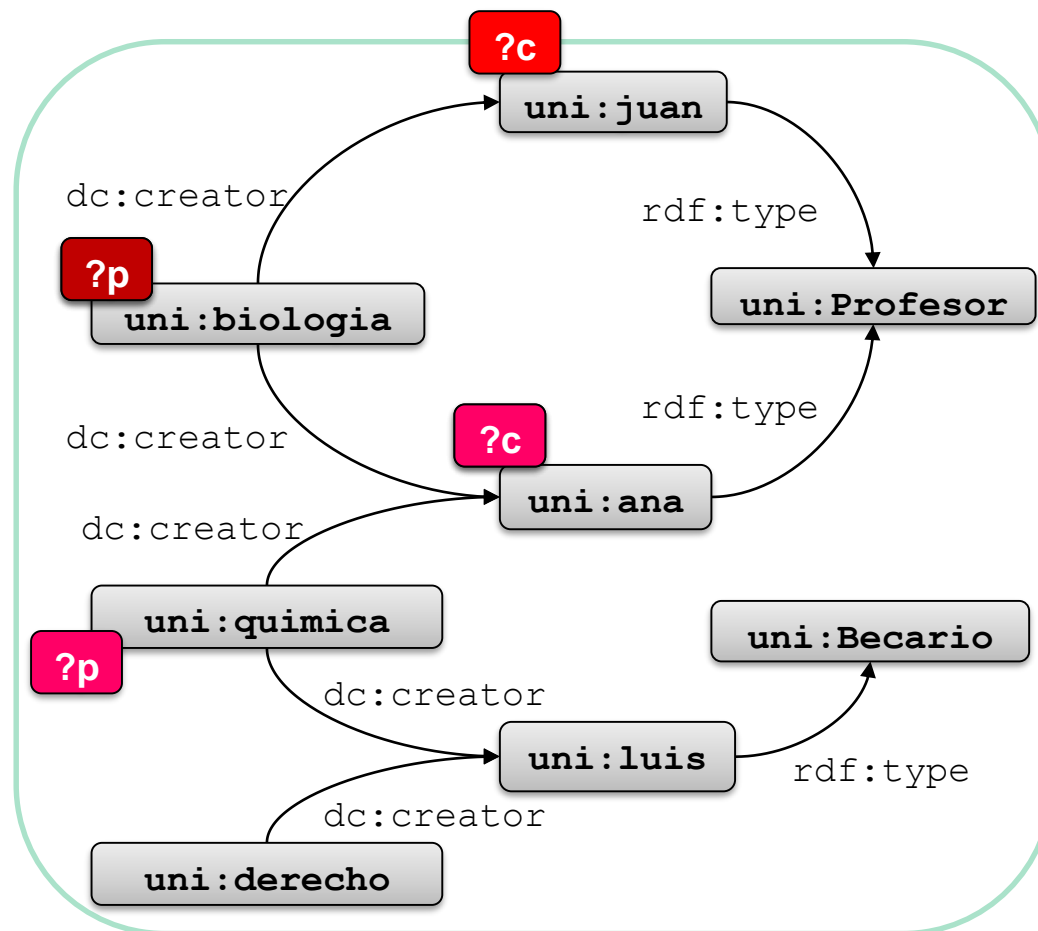
Encaje de grafos

```
SELECT ?p ?c WHERE {  
  ?p dc:creator ?c .  
  ?c rdf:type uni:Profesor .  
}
```



Resultados

?p	?c
uni:biologia	uni:juan
uni:biologia	uni:ana
uni:quimica	uni:ana





Ejercicio Test

¿Cuál sería la respuesta de la consulta SPARQL ante el fichero N3 siguiente?

```
@prefix : <http://www.pp.org#>.
```

```
:a :p :b.  
:a :p :c.  
:b :q "M".  
:b :q "N".
```

```
PREFIX e: <http://www.pp.org#>
```

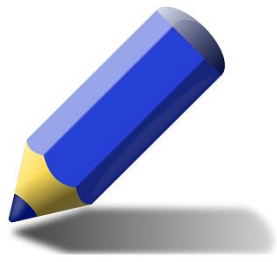
```
SELECT ?z WHERE {  
  ?x e:p ?y.  
  ?y e:q ?z.  
}
```

M
N

b
c

M

:a
:b
:c



Ejercicio Test

¿Cuál sería la respuesta de la consulta SPARQL ante el fichero N3 siguiente?

```
@prefix : <http://www.pp.org#>.
```

```
:a :p :b.
```

```
:a :p :c.
```

```
:b :q "M".
```

```
:b :q "N".
```

```
PREFIX e: <http://www.pp.org#>
```

```
SELECT ?x WHERE {  
  e:a ?x e:c.  
}
```

?x

e:p

e:q

e:b

Filtros

FILTER añade restricciones a los valores encajados

```
@prefix e: <http://ejemplo.org#>.
```

```
e:Pepe e:nombre "Jose" .
```

```
e:Pepe e:edad 31 .
```

```
e:Juan e:nombre "Juan" .
```

```
e:Juan e:edad 12 .
```

```
e:Ana e:nombre "Ana" .
```

```
e:Ana e:edad 25.
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?n ?e WHERE {
```

```
  ?x e:nombre ?n .
```

```
  ?x e:edad ?e
```

```
  FILTER (?e > 18)
```

```
}
```

n	e	
=====		
"Ana"	25	
"Jose"	31	

Operadores en los Filtros

FILTER utiliza funciones y operadores de XPath 2.0

Tipos de datos: Boolean, Integer, Float, dateTime, etc.

Operadores habituales: >, <, >=, <=, =, !=, ||, &&

```
PREFIX e: <http://ejemplo.org#>

SELECT ?n ?e WHERE {
  ?x e:nombre ?n .
  ?x e:edad ?e
  FILTER (?e > 30 || ?e < 18)
}
```

Conversión/creación de tipos de datos

str(arg) : convierte el argumento a una cadena

NOTA: Las URIs deben convertirse a cadenas si se quieren tratar como tales

lang(arg): devuelve el idioma del literal

Si `?x = "University"@en`

Entonces: `lang(?x) = "en"`

datatype(arg): devuelve el tipo de datos del literal

Si `?x = "123"^^xsd:integer`

Entonces `datatype(?x) = xsd:integer`

Creación tipos de datos

`uri(arg)`, `iri(arg)`: convierten el argumento a una URI/IRI

`bnode(arg)`: genera un nodo anónimo

`strdt(literal, tipo)`: genera un literal con un tipo de datos

`STRDT("123", "xsd:integer") = "123"^^<xsd:integer>`

`strlang(literal, tipo)`: genera un literal con un idioma dado

`strlang("University", "en") = "University"@en`

Funciones de comprobación de tipos

isNumeric(arg) = true si el argumento es un número

isBlank(arg) = true si el argumento es un nodo anónimo

isLiteral(arg) = true si el argumento es un literal

isIRI(arg) = true si el argumento es una IRI

Funciones condicionales

bound(arg) = true si el argumento tiene un valor

exists(patrón) = true si se cumple el patrón

not exists(patrón) = true si no se cumple el patrón

if(cond,expr1,expr2) = si se cumple cond, devuelve expr1, si no, devuelve expr2

coalesce(e1,e2,...)= devuelve la primer expresión que se evalúa sin error

Ejemplo

Filtrar las notas numéricas

```
@prefix : <http://ejemplo.org#> .
```

```
_:1 :nombre "Juan" .
```

```
_:1 :nota 8.5 .
```

```
_:2 :nombre "Luis" .
```

```
_:2 :nota "No presentado" .
```

```
_:3 :nombre "Ana" .
```

```
_:3 :nota 6.0 .
```

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

```
SELECT ?n WHERE {  
  ?x :nota ?n .  
  FILTER (isNumeric(?n))  
}
```

| n |

=====

| 6.0 |

| 8.5 |

Funciones con cadenas

`strlen(str)` = longitud de str

`ucase(str)` convierte a mayúsculas

`lcase(str)` convierte a minúsculas

`substr(str,inicio,tam?)`= subcadena a partir de inicio de tamaño tam

`substr('camino',3,2)='mi'`

`strstarts(str1,str2)` = true si str1 comienza con str2

`strends(str1,str2)` = true si str1 finaliza con str2

`contains(str1,str2)` = true si str1 contiene str2

`encode_for_uri (str)` = resultado de codificar str

`concat (str1,...strN)` = concatenación de cadenas

`langMatches(str,lang)` = true si encaja el idioma

`regex(str,patrón,flags)` = true si encaja la expresión regular

Ejemplo

```
@prefix : <http://ejemplo.org#>.
```

```
_:1 :nombre "Juan" .  
_:1 :apellidos "Gallardo" .
```

```
_:2 :nombre "Julio" .  
_:2 :apellidos "Zamora" .
```

```
_:3 :nombre "Luis" .  
_:3 :apellidos "Castro" .
```

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

```
SELECT  
    (concat(?nombre, ' ', ?apells) AS ?persona)  
WHERE  
{  
    ?x :nombre ?nombre .  
    ?x :apellidos ?apells .  
    FILTER (contains(ucase(?nombre), 'L'))  
}
```

```
-----  
| persona                |  
=====
```

"Luis Castro"
"Julio Zamora"

```
-----
```

Regex

REGEX invoca el encaje de expresiones regulares

Utiliza la función de XPath 2.0

`regex(?Expresión, ?Patrón [, ?Flags])`

?Expresión = expresión a encajar

?Patrón = expresión regular con la que se encaja

?Flags = opciones para el encaje

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?n ?e WHERE {  
  ?x e:nombre ?n .  
  ?x e:edad ?e  
  FILTER regex(?n, "A", "i")  
}
```

Selecciona los nombres que contengan la "A" ó la "a"

Regex

Expresiones regulares

^ = Inicio de cadena
\$ = Fin de la cadena
. = Cualquier carácter
\d = dígito
? = opcional, * = 0 ó más, + = 1 ó más
X{n} = encaja X n veces
X{m,n} = encaja X de m a n veces

Flags:

i = insensible mayúsculas/minúsculas
m = múltiples líneas
s = línea simple
x = elimina espacios en blanco



Ejercicio

El siguiente documento contiene una lista de países

<http://www.di.uniovi.es/~labra/cursos/XML/europa.ttl>

1. Mostrar países cuyo nombre empieza por 'A'
2. Mostrar países cuyo nombre termina por 'a'
3. Mostrar países cuyo nombre empieza por 'A' y termina por 'a'
4. Mostrar países cuyo pib es mayor que 20000
5. Mostrar países cuyo pib es mayor que 20000 y su población menor de 40 millones

Funciones numéricas

`abs(n)` = valor absoluto

`floor(n)` = redondear n^0 hacia bajo

`round(n)` = redondear n^0

`ceil(n)` = redondear n^0 hacia arriba

`rand()` = n^0 aleatorio entre 0 y 1

Funciones con fechas (1)

`now()` = devuelve el instante actual

`year(i)` = devuelve el año de un instante de tiempo `i`

`year("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 2011`

`month(i)` = devuelve el mes de `i`

`month("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 1`

`day(i)` = devuelve el día de `i`

`day("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 10`

`hours(i)` = devuelve la hora de `i`

`hours("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 14`

Funciones con fechas (2)

minutes(i) = devuelve los minutos de i

`minutes("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 45`

seconds(i) = devuelve los segundos de i

`seconds("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 13.815`

timezone(i) = devuelve la zona temporal de i

`timezone("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = -PT5H`

tz(i) = devuelve la zona temporal de i

`tz("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = -5`

Funciones HASH

md5(str) = aplica el algoritmo MD5 a str

sha1(str), sha224(str), sha256(str), sha384(str),

sha512(str) = calculan el valor HASH de str utilizando las variaciones correspondientes del algoritmo SHA

```
@prefix : <http://ejemplo.org#>.
```

```
@prefix : <http://ejemplo.org#> .
```

```
_:1 :nombre "Juan" .
```

```
_:1 :email "juan@uni.com" .
```

```
_:2 :nombre "Luis" .
```

```
_:2 :email "luis@uni.com" .
```

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

```
SELECT ?nombre (SHA1(?email) AS ?sha1Email)  
WHERE {  
  ?x :nombre ?nombre .  
  ?x :email ?email .  
}
```

nombre	sha1Email
"Luis"	"c3fbbedeb973ffbf0b319f152562b31552f03f157"
"Juan"	"bb402e5fe4d9ccbc8895caf0bae12122f1b0d2fa"

Unión de grafos

UNION combina resultados de varios grafos

```
@prefix e: <http://ejemplo.org#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
```

```
e:Pepe e:nombre "Jose" .
e:Pepe e:edad 31 .
e:Pepe e:conoceA e:Juan .
```

```
e:Juan foaf:name "Juan" .
e:Juan e:edad 25 .
e:Juan e:conoceA e:Ana .
```

```
e:Ana foaf:name "Ana" .
e:Ana e:nombre "Ana Mary".
```

```
PREFIX e: <http://ejemplo.org#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?n
WHERE {
  { ?x foaf:name ?n }
  UNION
  { ?y e:nombre ?n }
}
```

n
"Ana"
"Juan"
"Ana Mary"
"Jose"

Encajes opcionales

OPTIONAL permite obtener valores en tripletas sin fallar cuando éstas no existan

```
@prefix e: <http://ejemplo.org#>.
```

```
@prefix foaf: <http://xmlns.com/foaf/01./>.
```

```
e:Pepe e:nombre "Jose" .
```

```
e:Pepe e:edad 31 .
```

```
e:Juan e:nombre "Juan" .
```

```
e:Ana e:nombre "Ana" .
```

```
e:Ana e:edad 13.
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?n ?e WHERE {  
  ?x e:nombre ?n .  
  OPTIONAL { ?x e:edad ?e }  
}
```

n	e	
=====		
"Ana"	13	
"Juan"		
"Jose"	31	

Especificar grafos de entrada

FROM indica la URL de la que proceden los datos

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n
FROM <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
WHERE { ?x foaf:name ?n }
```

Si se incluyen varios conjuntos de entrada se realiza la mezcla de los grafos resultantes

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n
FROM <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
    ?x foaf:name ?n
}
```



Ejercicio

Modelizar las siguientes tablas en 2 ficheros Turtle diferentes

DNI	Nombre	Apellidos
9999	Juan	Gallardo
8888	Jose	Torre
7777	Ana	Cascos

DNI	Nota
9999	3
7777	5

Construir una consulta que permita visualizar la nota de cada alumno junto con su nombre y apellidos

Grafos con nombre

FROM NAMED asigna un nombre al grafo de entrada

GRAPH encaja con el nombre del grafo que corresponda

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n ?g
FROM NAMED <http://www.w3.org/People/Berners-Lee/card>
FROM NAMED <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
WHERE {
    GRAPH ?g { ?x foaf:name ?n }
}
```

Control de los resultados

DISTINCT elimina valores duplicados

ORDER BY permite especificar el orden de los resultados (puede especificarse ASC, DESC...)

LIMIT n indica el número de resultados

OFFSET m indica a partir de qué resultado empezar a contar

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?n
WHERE { ?x foaf:knows ?y .
        ?y foaf:name ?n . }
ORDER BY ASC(?n)
LIMIT 5
OFFSET 10
```


CONSTRUCT

Permite crear un grafo de salida

```
@prefix : <http://ejemplo.org#> .

_:1 :nombre "Juan" .
_:1 :amigoDe _:2, _:3 .

_:2 :nombre "Luis" .
_:2 :amigoDe _:1 .

_:3 :nombre "Ana" .
_:3 :amigoDe _:1 .
```

```
PREFIX : <http://ejemplo.org#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
CONSTRUCT {
  ?x foaf:name ?n .
  ?x foaf:knows ?y .
} WHERE {
  ?x :nombre ?n .
  ?x :amigoDe ?y .
}
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:b1 foaf:knows [ foaf:knows _:b1 ;
                  foaf:name "Luis"
                ] ;
      foaf:knows [ foaf:knows _:b1 ;
                  foaf:name "Ana"
                ] ;
      foaf:name "Juan" .
```

ASK

ASK devuelve sí o no

Puede ser útil para chequeo de errores

```
@prefix : <http://ejemplo.org#>
```

```
.
```

```
_:1 :nombre "Juan" .
```

```
_:1 :nota 8.5 .
```

```
_:2 :nombre "Luis" .
```

```
_:2 :nota "No presentado" .
```

```
_:3 :nombre "Ana" .
```

```
_:3 :nota 6.0 .
```

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

```
ASK WHERE {  
  ?x :nota ?n .  
  FILTER ( ! isNumeric(?n))  
}
```

Yes

DESCRIBE

Devuelve una descripción RDF de uno o varios nodos

```
@prefix : <http://ejemplo.org#> .
```

```
_:1 :nombre "Juan" .
```

```
_:1 :nota 8.5 .
```

```
_:2 :nombre "Luis" .
```

```
_:2 :nota "No presentado" .
```

```
_:3 :nombre "Ana" .
```

```
_:3 :nota 6.0 .
```

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

```
DESCRIBE ?x WHERE
```

```
{
```

```
  ?x :nota ?n .
```

```
  FILTER(?n = 6.0)
```

```
}
```

```
@prefix : <http://ejemplo.org#> .
```

```
[] :nombre "Ana" ;
```

```
:nota 6.0 .
```

Asignaciones

BIND *expr* AS *v* = Asigna el valor de *expr* a la variable *v*

```
@prefix e: <http://ejemplo.org#>.
```

```
_:1 e:nombre "Manzanas" .
```

```
_:1 e:cantidad 3 .
```

```
_:1 e:precio 3 .
```

```
_:2 e:nombre "Peras" .
```

```
_:2 e:cantidad 2 .
```

```
_:2 e:precio 2 .
```

```
_:3 e:nombre "Naranjas" .
```

```
_:3 e:cantidad 4 .
```

```
_:3 e:precio 1 .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?n    ?precioTotal
```

```
WHERE {
```

```
    ?x e:nombre ?n .
```

```
    ?x e:cantidad ?cantidad .
```

```
    ?x e:precio ?precio .
```

```
    BIND ((?cantidad * ?precio) AS ?precioTotal)
```

```
}
```

```
-----  
| n          | precioTotal |  
=====
```

"Naranjas"	4	
"Peras"	4	
"Manzanas"	9	

```
-----
```

Asignaciones en SELECT

Es posible realizar la asignación directamente

```
@prefix e: <http://ejemplo.org#>.
```

```
_:1 e:nombre "Manzanas" .
```

```
_:1 e:cantidad 3 .
```

```
_:1 e:precio 3 .
```

```
_:2 e:nombre "Peras" .
```

```
_:2 e:cantidad 2 .
```

```
_:2 e:precio 2 .
```

```
_:3 e:nombre "Naranjas" .
```

```
_:3 e:cantidad 4 .
```

```
_:3 e:precio 1 .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?n
```

```
((?cantidad * ?precio) AS ?precioTotal)
```

```
WHERE {
```

```
?x e:nombre ?n .
```

```
?x e:cantidad ?cantidad .
```

```
?x e:precio ?precio .
```

```
}
```

n	precioTotal	
=====		
"Naranjas"	4	
"Peras"	4	
"Manzanas"	9	

Funciones de agregación: AVG, SUM, COUNT, SAMPLE

```
@prefix e: <http://ejemplo.org#>.
```

```
e:Pepe e:nombre "Jose" .  
e:Pepe e:edad 31 .
```

```
e:Juan e:nombre "Juan" .  
e:Juan e:edad 12 .
```

```
e:Ana e:nombre "Ana" .  
e:Ana e:edad 25.
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT (AVG(?edad)      AS ?media)  
       (SUM(?edad)      AS ?suma)  
       (COUNT(?edad)   AS ?cuenta)  
       (SAMPLE(?edad)   AS ?muestra)  
WHERE {  
    ?n e:edad ?edad .  
}
```

media	suma	cuenta	muestra	
=====	=====	=====	=====	=====
22.66	68	3	25	

Funciones de agregación: MAX, MIN

```
@prefix e: <http://ejemplo.org#>.
```

```
e:Pepe e:nombre "Jose" .  
e:Pepe e:edad 31 .
```

```
e:Juan e:nombre "Juan" .  
e:Juan e:edad 12 .
```

```
e:Ana e:nombre "Ana" .  
e:Ana e:edad 25.
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT (MAX(?edad) as ?mayor)  
       (MIN(?edad) as ?menor) WHERE {  
    ?n e:edad ?edad .  
}
```

mayor	menor	
=====		
31	12	

Funciones de agregación

GROUP_CONCAT

```
@prefix e: <http://ejemplo.org#>.
```

```
e:Pepe e:nombre "Jose" .  
e:Pepe e:edad 31 .
```

```
e:Juan e:nombre "Juan" .  
e:Juan e:edad 12 .
```

```
e:Ana e:nombre "Ana" .  
e:Ana e:edad 25.
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT (GROUP_CONCAT(?edad; SEPARATOR=',')  
        as ?edades) where {  
    ?n e:edad ?edad .  
}
```

```
| edades |  
=====
```

25, 12, 31

Agrupaciones: GROUP_BY

GROUP BY permite agrupar conjuntos de resultados

```
@prefix e: <http://ejemplo.org#>.
```

```
_:1 e:nombre "Ana".  
_:1 e:edad 18 .  
_:1 e:nota 8 .
```

```
_:2 e:nombre "Juan".  
_:2 e:edad 20 .  
_:2 e:nota 7 .
```

```
_:3 e:nombre "Luis".  
_:3 e:edad 18 .  
_:3 e:nota 5 .
```

```
_:4 e:nombre "Mario".  
_:4 e:edad 19 .  
_:4 e:nota 6 .
```

```
_:5 e:nombre "Carlos".  
_:5 e:edad 20 .  
_:5 e:nota 9 .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT (AVG(?nota) AS ?mediaNota) ?edad  
WHERE {  
  ?x e:nombre ?n .  
  ?x e:edad ?edad .  
  ?x e:nota ?nota .  
}  
GROUP BY ?edad
```

```
-----  
| mediaNota | edad |  
=====
```

6.5	18	
8.0	20	
6.0	19	

```
-----
```

Agrupaciones: HAVING

HAVING permite filtrar los grupos que cumplan una condición

```
@prefix e: <http://ejemplo.org#>.
```

```
_:1 e:nombre "Ana".
```

```
_:1 e:edad 18 .
```

```
_:1 e:nota 8 .
```

```
_:2 e:nombre "Juan".
```

```
_:2 e:edad 20 .
```

```
_:2 e:nota 7 .
```

```
_:3 e:nombre "Luis".
```

```
_:3 e:edad 18 .
```

```
_:3 e:nota 5 .
```

```
_:4 e:nombre "Mario".
```

```
_:4 e:edad 19 .
```

```
_:4 e:nota 6 .
```

```
_:5 e:nombre "Carlos".
```

```
_:5 e:edad 20 .
```

```
_:5 e:nota 9 .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT (AVG(?nota) AS ?mediaNota) ?edad
```

```
WHERE {
```

```
?x e:nombre ?n .
```

```
?x e:edad ?edad .
```

```
?x e:nota ?nota .
```

```
}
```

```
GROUP BY ?edad
```

```
HAVING (?mediaNota < 8)
```

```
-----  
| mediaNota | edad |  
=====
```

6.5	18	
6.0	19	

```
-----
```

Subconsultas

Es posible realizar consultas dentro de consultas

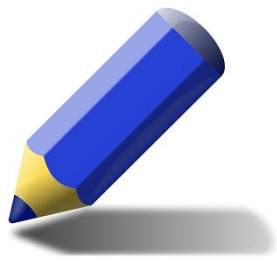
```
@prefix e: <http://ejemplo.org#>.
```

```
_:1 e:nombre "Ana".  
_:1 e:edad 18 .  
_:1 e:nota 8 .  
  
_:2 e:nombre "Juan".  
_:2 e:edad 20 .  
_:2 e:nota 7 .  
  
_:3 e:nombre "Luis".  
_:3 e:edad 18 .  
_:3 e:nota 5 .  
  
_:4 e:nombre "Mario".  
_:4 e:edad 19 .  
_:4 e:nota 6 .  
  
_:5 e:nombre "Carlos".  
_:5 e:edad 20 .  
_:5 e:nota 9 .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?nombre ?nota (?nota - ?notaMedia AS ?desv)  
WHERE {  
  ?x e:nombre ?nombre .  
  ?x e:nota ?nota .  
  {  
    SELECT (AVG(?nota) AS ?notaMedia) WHERE {  
      ?x e:nota ?nota .  
    }  
  }  
}
```

nombre	nota	desv
"Carlos"	9	2
"Mario"	6	-1
"Luis"	5	-2
"Juan"	7	0
"Ana"	8	1



Ejercicio

El siguiente documento contiene una lista de países

<http://www.di.uniovi.es/~labra/cursos/XML/europa.ttl>

1. Mostrar el país con mayor PIB
2. Mostrar el PIB medio
3. Mostrar países cuyo PIB es mayor que el PIB medio
4. Mostrar países de una población similar a la de España cuyo PIB esté por encima

Caminos de propiedades

La URI que identifica la propiedad puede contener una expresión regular

p	Encaja con la propiedad p
(e)	Camino agrupado entre paréntesis
$\wedge e$	Camino inverso de e
!p	No encaja con la propiedad p
e1 / e2	Camino e1 seguido de e2
e1 e2	Camino e1 ó e2
e*	0 ó más apariciones de e
e+	1 ó más apariciones de e
e?	0 ó 1 aparición de e
e{n}	n apariciones de e
e{m,n}	Entre m y n apariciones de e
e{n,}	n ó más apariciones de e
e{,n}	Entre 0 y n apariciones de e

Caminos de propiedades

```
@prefix e: <http://ejemplo.org#>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.

e:Pepe e:nombre "Jose" .
e:Juan foaf:name "Juan" .
e:Ana foaf:name "Ana" .
e:Ana e:nombre "Ana Mary".
```

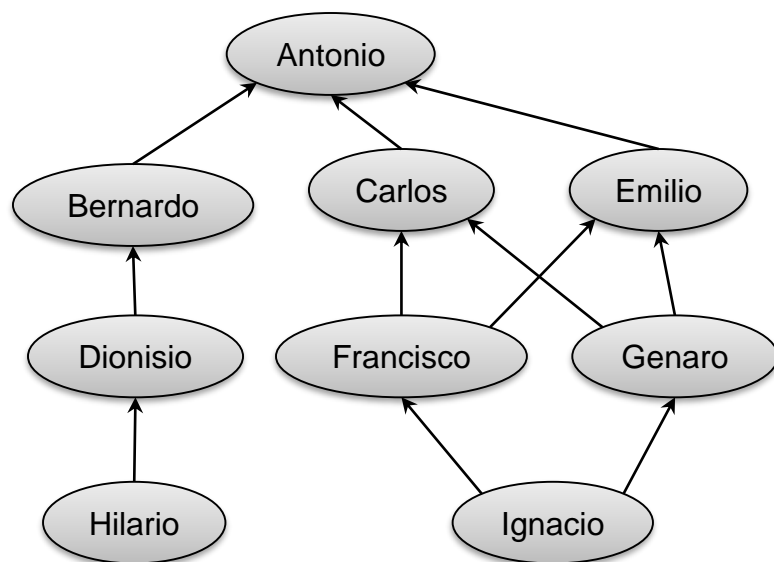
```
PREFIX e: <http://ejemplo.org#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?n
WHERE {
    ?x (foaf:name | e:nombre) ?n
}
```

n
"Ana"
"Ana Mary"
"Jose"
"Juan"

Caminos de propiedades

```
e:Ignacio e:conoceA e:Francisco, e:Genaro.  
e:Francisco e:conoceA e:Carlos, e:Emilio .  
e:Genaro e:conoceA e:Carlos, e:Emilio .  
e:Carlos e:conoceA e:Antonio .  
e:Emilio e:conoceA e:Antonio .  
e:Hilario e:conoceA e:Dionisio .  
e:Dionisio e:conoceA e:Bernardo .  
e:Bernardo e:conoceA e:Antonio .
```



```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?p  
{  
  ?p e:conoceA+ e:Antonio.  
}
```

```
-----  
| p |  
=====
```

e:Bernardo
e:Dionisio
e:Hilario
e:Emilio
e:Genaro
e:Ignacio
e:Francisco
e:Ignacio
e:Carlos
e:Genaro
e:Ignacio
e:Francisco
e:Ignacio

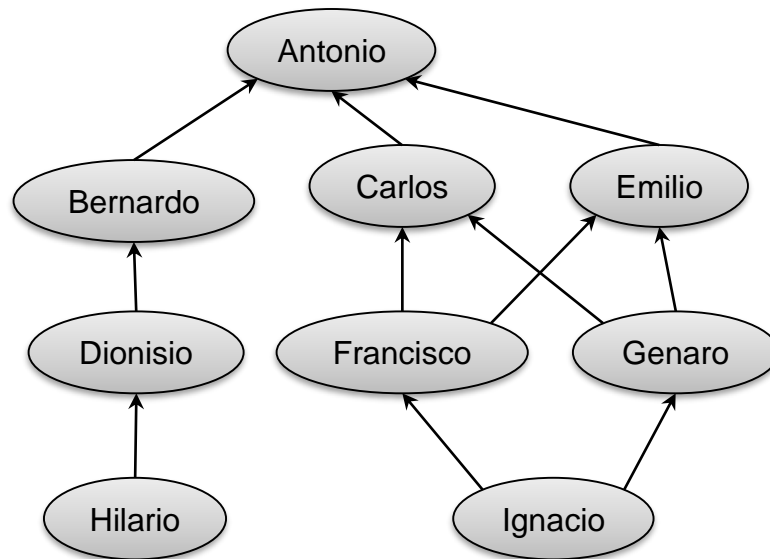
```
-----
```

Caminos de propiedades

```
e:Ignacio    e:conoceA e:Francisco, e:Genaro.  
e:Francisco e:conoceA e:Carlos, e:Emilio .  
e:Genaro     e:conoceA e:Carlos, e:Emilio .  
e:Carlos     e:conoceA e:Antonio .  
e:Emilio     e:conoceA e:Antonio .  
e:Hilario    e:conoceA e:Dionisio .  
e:Dionisio   e:conoceA e:Bernardo .  
e:Bernardo   e:conoceA e:Antonio .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT ?p WHERE {  
    ?p e:conoceA{2} e:Antonio.  
}
```



```
-----  
| p                |  
=====
```

e:Dionisio
e:Genaro
e:Francisco
e:Genaro
e:Francisco

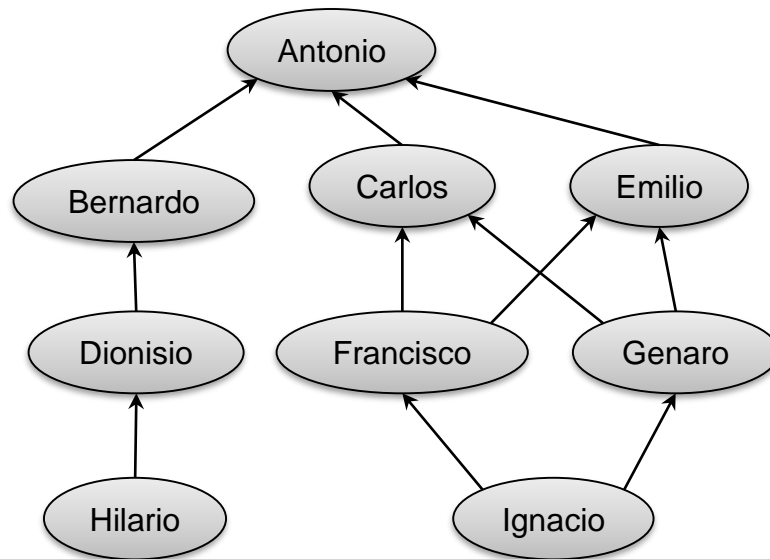
```
-----
```


Caminos de propiedades

```
e:Ignacio    e:conoceA e:Francisco, e:Genaro.  
e:Francisco  e:conoceA e:Carlos, e:Emilio .  
e:Genaro     e:conoceA e:Carlos, e:Emilio .  
e:Carlos     e:conoceA e:Antonio .  
e:Emilio     e:conoceA e:Antonio .  
e:Hilario    e:conoceA e:Dionisio .  
e:Dionisio   e:conoceA e:Bernardo .  
e:Bernardo   e:conoceA e:Antonio .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT  DISTINCT ?p  
{  
    ?p e:conoceA/e:conoceA e:Antonio.  
}
```



```
-----  
| p                |  
=====
```

e:Dionisio	
e:Genaro	
e:Francisco	
e:Genaro	
e:Francisco	

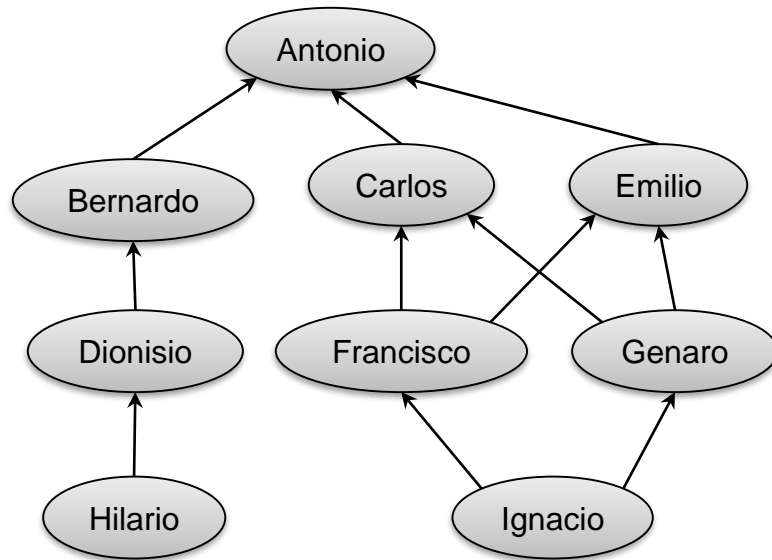
```
-----
```

Caminos de propiedades

```
e:Ignacio      e:conoceA e:Francisco, e:Genaro.  
e:Francisco e:conoceA e:Carlos, e:Emilio .  
e:Genaro      e:conoceA e:Carlos, e:Emilio .  
e:Carlos      e:conoceA e:Antonio .  
e:Emilio      e:conoceA e:Antonio .  
e:Hilario     e:conoceA e:Dionisio .  
e:Dionisio    e:conoceA e:Bernardo .  
e:Bernardo    e:conoceA e:Antonio .
```

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT  ?p  
{  
  ?p e:conoceA/^e:conoceA e:Francisco.  
  FILTER (?p != e:Francisco)  
}
```



```
-----  
| p                |  
=====
```

e:Genaro
e:Genaro

```
-----
```

Actualizaciones SPARQL Update

Tratamiento de grafos

Actualización

INSERT DATA	= insertar tripletas
DELETE/INSERT...	= borrar/insertar tripletas condicionalmente
DELETE DATA	= borrar tripletas
LOAD	= cargar tripletas de un documento
CLEAR	= borrar todas las tripletas de un grafo

Gestión de grafos

CREATE	= crear grafo
DROP	= eliminar grafo
COPY...TO...	= copiar grafo
MOVE...TO...	= mover grafo
ADD	= insertar todos los datos de un grafo en otro

Inserción

INSERT DATA permite insertar tripletas

```
PREFIX e: <http://ejemplo.org#>
```

```
INSERT DATA {
```

```
e:ana e:nombre "Ana".
```

```
e:ana e:edad 18 .
```

```
e:ana e:nota 8 .
```

```
e:juan e:nombre "Juan Manuel".
```

```
e:juan e:edad 20 .
```

```
e:juan e:nota 7 .
```

```
}
```

Inserción en un grafo concreto

INSERT DATA puede especificar el grafo

```
PREFIX e: <http://ejemplo.org#>
PREFIX g: <http://grafos.org#>

INSERT DATA {

  GRAPH g:g1 {
    e:ana e:nombre "Ana".
    e:ana e:edad   18 .
    e:ana e:nota   8 .
  }
}
```

Inserción

INSERT permite insertar tripletas en un grafo.

Requiere una cláusula WHERE

```
PREFIX e: <http://ejemplo.org#>

INSERT {
    ?p e:nombreNota "Notable".
}
WHERE {
    ?p e:nota ?nota .
    FILTER (?nota >= 7 && ?nota < 9)
}
```

Carga de grafo

LOAD permite cargar todas las tripletas existentes en una URI

```
LOAD <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
```


Borrado

DELETE DATA permite eliminar tripletas de un grafo

```
PREFIX e: <http://ejemplo.org#>
```

```
DELETE DATA
```

```
{
```

```
  e:luis e:nota 5 .
```

```
}
```

NOTA: DELETE DATA No admite variables

Borrado

DELETE WHERE permite eliminar tripletas de un grafo especificando una condición

```
PREFIX e: <http://ejemplo.org#>

DELETE { ?x e:nota ?nota . } WHERE {
  ?x e:nota ?nota .
  FILTER (?nota >= 8)
}
```

Actualización

DELETE/INSERT permite actualizar tripletas de un grafo

Ejemplo: incrementar la edad

```
PREFIX e: <http://ejemplo.org#>

DELETE { ?x e:edad ?edad }
INSERT { ?x e:edad ?edad1 }
WHERE {
  ?x e:edad ?edad .
  BIND((?edad + 1) AS ?edad1)
}
```

Borrado total

CLEAR borra todas las tripletas

Puede indicarse el conjunto de datos

CLEAR g = Borra grafo g

CLEAR DEFAULT = Borra grafo actual

CLEAR ALL = Borra todos los grafos

Consulta universal

Para ver todas las tripletas de la base de datos

```
PREFIX e: <http://ejemplo.org#>
```

```
SELECT * WHERE {  
  { ?x ?p ?y . }  
  UNION
```

```
  { GRAPH ?g { ?x ?p ?y . } }  
}
```

Acceso a servicios remotos

SERVICE uri = indica un endpoint SPARQL

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

SELECT ?nombre WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    SELECT ?nombre WHERE {
      ?pais rdf:type dbo:Country .
      ?pais rdfs:label ?nombre .
      FILTER (lang(?nombre)='es')
    }
  }
}
```

Lista de terminales SPARQL
<http://esw.w3.org/topic/SparqlEndpoints>

Combinando resultados

DBPedia: <http://dbpedia.org>
IMDB: <http://data.linkedmdb.org>

Consultas federadas

```
PREFIX imdb: <http://data.linkedmdb.org/resource/movie/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbpo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT * {
  { SERVICE <http://dbpedia.org/sparql>
    { SELECT ?fechaNacim ?nombreMujer WHERE {
      ?actor rdfs:label "Javier Bardem"@en ;
      dbpo:birthDate ?fechaNacim ;
      dbpo:spouse ?mujerURI .
      ?mujerURI rdfs:label ?nombreMujer .
      FILTER ( lang(?nombreMujer) = "en" )
    }
  }
}
{ SERVICE <http://data.linkedmdb.org/sparql>
  { SELECT ?peli ?fechaPeli WHERE {
    ?actor imdb:actor_name "Javier Bardem".
    ?movie imdb:actor ?actor ;
    dcterms:title ?peli ;
    dcterms:date ?fechaPeli .
  }
}
}
```



Ejercicio

Buscar en la DBPedia películas españolas, mostrando el título, el nombre del director y el año en que se hicieron.

Combinar la información con otros terminales SPARQL

Patrón de negación por fallo en SPARQL

Combinando FILTER, OPTIONAL y !BOUND se puede simular la negación por fallo
Ejemplo: Buscar personas que no están casadas.

```
@prefix : <http://ej.org#>.
```

```
:Pepe :estaCasadoCon :Ana .  
:Pepe :nombre "Jose" .
```

```
:Luis :estaCasadoCon :Marta .  
:Luis :nombre "Luis" .
```

```
:Carlos :nombre "Carlos" .
```

```
PREFIX : <http://ej.org#>  
SELECT ?n WHERE {  
  ?x :nombre ?n  
  OPTIONAL {?x :estaCasadoCon ?y }  
  FILTER ( !BOUND(?y) )  
}
```

¿Realmente muestra los que no están casados?

Creación de Base de Datos RDF

Fuseki permite trabajar con datos RDF como una base de datos

Es posible realizar consultas SPARQL

Arrancar servidor

```
> mkdir dirDatos  
> fuseki-server --update --loc=dirDatos /datos
```

Acceso en puerto: <http://localhost:3030>

Validación de RDF mediante SPARQL

Ejemplo:

Una persona tiene una edad (entero) y uno ó más nombres (string)

Persona
foaf:age xsd:integer
foaf:name xsd:string+

Ejemplos de RDF

```
:john foaf:age 23;  
      foaf:name "John" .
```



```
:bob foaf:age 34;  
     foaf:name "Bob", "Robert" .
```

```
:mary foaf:age 50, 65 .
```



Ejemplo de consulta SPARQL

Persona

foaf:age xsd:integer

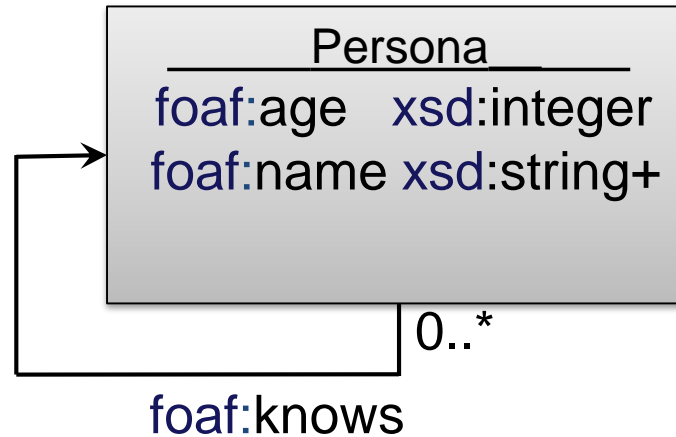
foaf:name xsd:string+

```
1  ASK {
2    { SELECT ?Person {
3      ?Person foaf:age ?o .
4    } GROUP BY ?Person HAVING (COUNT(*)=1)
5  }
6    { SELECT ?Person {
7      ?Person foaf:age ?o .
8      FILTER ( isLiteral(?o) &&
9                datatype(?o) = xsd:integer )
10     } GROUP BY ?Person HAVING (COUNT(*)=1)
11  }
12   { SELECT ?Person (COUNT(*) AS ?Person_c0) {
13     ?Person foaf:name ?o .
14   } GROUP BY ?Person HAVING (COUNT(*)>=1)
15  }
16   { SELECT ?Person (COUNT(*) AS ?Person_c1) {
17     ?Person foaf:name ?o .
18     FILTER (isLiteral(?o) &&
19              datatype(?o) = xsd:string)
20   } GROUP BY ?Person HAVING (COUNT(*)>=1)
21  } FILTER (?Person_c0 = ?Person_c1)
22 }
```

¿Es posible añadir recursividad al modelo?

Ejemplo:

Una persona tiene una edad (entero), uno o más nombres (string) y conoce a 0 ó más personas



Fin de la Presentación

