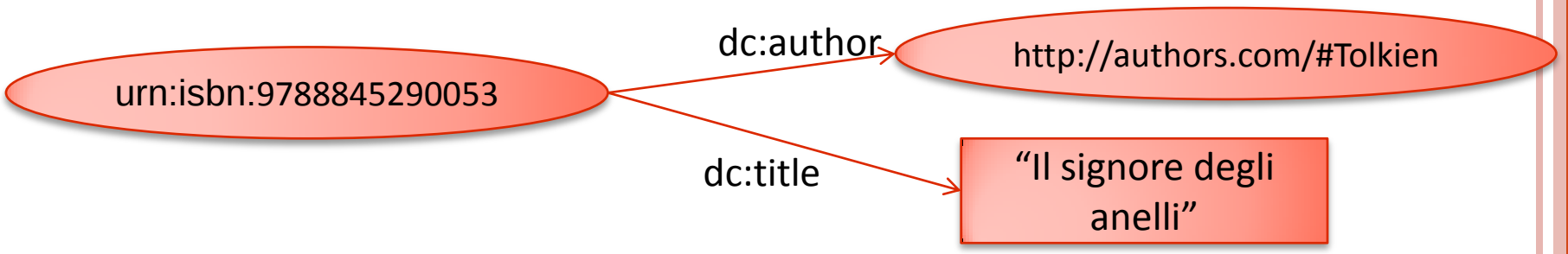
SPARQL

Dott.sa Vincenza Anna Leano
vincenzaanna.leano@unina.it

Basi di Dati II mod. B
Prof. F. Cutugno
A.A. 2010/2011

Esempio

- Concetto:



- Query: Trovare il titolo del libro “urn:isbn:9788845290053”

Serializzazione RDF (I)

- Concetto

```
<rdf:Description rdf:about="urn:isbn:9788845290053">
        <dc:title> Il signore degli Anelli </dc:title>
        <dc:auhor> http://authors.com/#Tolkien</dc:author>
</rdf:Description>
```

- XPATH query:

```
//rdf:Description[@rdf:about="urn:isbn:9788845290053"]/
dc:title/text()
```

Serializzazione RDF (II)

- Concetto

```
<rdf:Description rdf:about="unr:isbn:9788845290053"
dc:title="Il signore degli Anelli">
        <dc:auhor> http://authors.com/#Tolkien </dc:author>
</rdf:Description>
```

- XPATH query

```
//rdf:Description[@rdf:about="urn:isbn:9788845290053"]/
@dc:title
```

Serializzazione RDF (III)

- Concetto

```
<rdf:Description rdf:about="unr:isbn:9788845290053">
    <rdf:Description rdf:about="myuri:LOTR">
       <dc:title>Il Signore degli Anelli</dc:title>
    </rdf:Description>
 <dc:author> Tolkien</dc:author>
</rdf:Description>
```

- XPATH Query:

```
//rdf:Description[@rdf:about="urn:isbn:9788845290053
"]/rdf:Description/dc:title
```

SPARQL

- Linguaggio di interrogazione per RDF
- **S**PARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- Sintassi SQL-LIKE
- Utilizza i concetti di triple e grafo
- Una query è la ricerca del sottografo rdf corrispondente alle triple richieste dall'utente
- Esempio:

```
PREFIX: dc= "http://dublincore.org/documents/dcmi-namespaces"

SELECT ?title
WHERE <urn:isbn:978845290053> dc:title ?title
```

Elementi di SPARQL

- Graph Pattern
  - Grouping
  - Optional
  - Union
  - Filter
- Tipi di query
  - Select
  - Construct
  - Describe
  - Ask

Anticipo: Tipica Query Sparql

`prefix`

- Prefisso per gli uri

`SELECT`

- Variabili che si vogliono visualizzare

`FROM`

- File rdf o grafo

`WHERE`

- Condizioni: graph pattern, filtri, ordinamento

Graph Pattern

Basic Graph Pattern

- Insieme di Triple Patterns
  - **Triple Pattern -** si basa sulle triple RDF (soggetto, predicato, Oggetto) ma :
    - ogni componente può essere una variabile.
    - Sono consentiti soggetti di tipo letterale.

```
<urn:isbn:978845290053> dc:title ?title
```

- Matching di una tripla su un grafo:
  - Una **Pattern Solution** di una tripla GP su un grafo G è una qualsiasi sostituzione S tale per cui S(GP) è un sottografo di G.

Dati di esempio

```
<myuri:person1> <myuri:name> "Lina" .
<myuri:person1> <myuri:age> "27" .
<myuri:person1> <myuri:marriedTo> "myuri:person2" .
<myuri:person1> <myuri:workTo> "Tribunale" .

<myuri:person2> <myuri:name> "Gino" .
<myuri:person2> <myuri:age> "30" .
<myuri:person2> <myuri:marriedTo> "myuri:person1" .
<myuri:person2> <myuri:workTo> "Ministero" .

<myuri:person3> <myuri:name> "Paolo" .
<myuri:person3> <myuri:age> "18" .
<myuri:person3> <myuri:workTo> "Pizzeria" .

<myuri:person4> <myuri:name> "Gino" .
<myuri:person4> <myuri:age> "35" .
<myuri:person4> <myuri:marriedTo> "myuri:person5" .

<myuri:person5> <myuri:name> "Pina" .
<myuri:person5> <myuri:age> "25" .
<myuri:person5> <myuri:marriedTo> "myuri:person4" .
<myuri:person5> <myuri:workTo> "Tribunale" .
```

Graph Pattern multipli (AND)

- Vengono restituite le variabili per cui valgono tutti i vincoli di graph pattern richiesti
- I path sono racchiusi tra {} e vengono concatenati con il simbolo “.”
- Esempio: Nomi e lavori delle persone nel db

```
prefix myuri: <http://someurl.com/example/>
SELECT ?name ?work
WHERE {?x myuri:name ?name .
       ?x myuri:workTo ?work}
```

```
  name     | work
==========================
| "Pina"   | "Tribunale" |
| "Paolo"  | "Pizzeria"  |
| "Gino"   | "Ministero" |
| "Lina"   | "Tribunale" |
--------------------------
```

OPTIONAL

- Il costrutto optional viene utilizzato quando il graph pattern deve essere valutato opzionalmente
- Esempio: “Nomi di tutte le persone Del DB e se sono sposate il nome dei loro coniugi”

```
prefix myuri: <http://someurl.com/example/>
SELECT ?name ?sname
WHERE { ?x myuri:name ?name .
        OPTIONAL {?x myuri:marriedTo ?spouse .
                  ?spouse myuri:name ?sname}}
```

name	sname
"Pina"	"Gino"
"Gino"	"Pina"
"Paolo"	
"Alessio"	"Lina"
"Lina"	"Alessio"

Graph pattern Alternativi (OR): Union

- UNION effettua l'OR tra vari graph pattern. Se più di una alternativa ha il suo match nel grafo vengono restituite tutte le soluzioni valide.
- Esempio: Per ogni persona l'età oppure il lavoro

```
prefix myuri: <http://someurl.com/example/>
SELECT *
WHERE { {?x myuri:age ?age}  UNION {?x
myuri:workTo ?work}}
```

```
---------------------------------------
| x                | age   | work       |
=======================================
| <myuri:person5> | "25" |              |
| <myuri:person4> | "35" |              |
| <myuri:person3> | "18" |              |
| <myuri:person2> | "30" |              |
| <myuri:person1> | "27" |              |
| <myuri:person5> |      | "Tribunale" |
| <myuri:person3> |      | "Pizzeria"  |
| <myuri:person2> |      | "Ministero" |
| <myuri:person1> |      | "Tribunale" |
---------------------------------------
```

Filtri

- È possibile porre dei filtri sulle variabili
- Funzioni di filtro:
  - *Logical:* !, &&, ||
  - *Math:* +, -, *, /
  - *Comparison:* =, !=, >, <, ...
  - *SPARQL tests:* isURI, isBlank, isLiteral, bound
  - *SPARQL accessors:* str, lang, datatype
  - *Other:* sameTerm, langMatches, regex

Filtri numerici

- Nomi delle persone che hanno meno di 30 anni

```
prefix myuri: <http://someurl.com/example/>
SELECT ?name ?age
WHERE { ?x myuri:name ?name . ?x myuri:age ?age
. FILTER(?age < 30) }
```

```
------------------
| name    | age  |
==================
| "Pina"  | "25" |
| "Paolo" | "18" |
| "Lina"  | "27" |
------------------
```

Filtri su stringhe

- Nomi che contengono la parola “in”

```
prefix myuri: <http://someurl.com/example/>
SELECT ?name
WHERE { ?x myuri:name ?name .
FILTER(regex(?name, ”^in”, “i”) }
```

```
-----------
| name    |
===========
| "Pina"  |
| “Gino“  |
| "Lina"  |
-----------
```

Path Properties (1/2)

- Una property path expression è simile a un'espressione regolare sulle stringhe ma viene effettuata sulle proprietà (archi)

Syntax Form	Matches
uri	A URI or a prefixed name. A path of length one.
^elt	Reverse path (object to subject).
(elt)	A group path *elt*, brackets control precedence.
elt1 / elt2	A sequence path of *elt1*, followed by *elt2*
elt1 ^ elt2	Shorthand for *elt1 / ^elt2*, that is *elt1* followed by reverse *elt2*.
elt1 | elt2	A alternative path of *elt1*, or *elt2* (all possibilities are tried)
*elt**	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.

Path Properties (2/2)

Syntax Form	Matches
elt?	A path of zero or one *elt*.
elt{n,m}	A path between n and m occurrences of *elt*.
elt{n}	Exactly *n* occurrences of *elt*. A fixed length path.
elt{n,}	*n* or more occurrences of *elt*.
elt{,n}	Between 0 and *n* occurrences of *elt*.
!uri	A path matching a property which isn't *uri* (negated property set)
!(uri1|...|uriN)	A path matching a property which isn't *any of uri1 ... uriN* (negated property set)

Path Properties - Esempi

- Trovare il nome di tutte le persone che possono essere raggiunte a partire da Alice tramite la proprietà foaf:knows

```
{
 ?x foaf:mbox <mailto:alice@example> .
 ?x foaf:knows+/foaf:name ?name .
}
```

- Trovare tutte le classi e le sottoclassi di una risorsa

```
{ <http://example/> rdf:type/rdfs:subClassOf*
?type }
```

PATH - CORESE

- Corese: Conceptual Resource Search Engine.
- E' un framework RDF basato sui Conceptual Graphs (CG).
- Sviluppato dall' INRIA
- Tra le altre cose, estende SPARQL con alcuni operatori sui path:
- Una path expression consente di trovare un percorso di lunghezza arbitraria (>1) tra due risorse
- Notazione sintattica: usare una variabile ($) al posto della proprietà.

```
SELECT *
WHERE { ?x $path ?y .
        FILTER(?x = myuri:nodeStart && ?y =
myuri:nodeEnd) }
```

Tipi di query

SELECT

```
SELECT (DISTINCT | REDUCED) AGGREGATES(var)
FROM
WHERE
ORDER BY (OrderCondition) ASC | DESC
LIMIT
OFFSET
```

- **Distinct**: elimina sicuramente i duplicati
- **Reduced**: potrebbe eliminare i duplicati. Se ci sono N duplicati ritorna tra 1 e N-1
- **Aggregates** (sparql 1.1)
  - COUNT, MIN, MAX, SUM, AVG, GROUP_CONCAT, SAMPLE
- **Order by**: stabilisce l'ordine dei risultati
- **Limit**: uppere bound sul numero di risultati
- **Offset**: da quale risultato partire a mostrare le soluzioni

Esempio Select (1/2)

- I primi tre nomi in ordine alfabetico

```
prefix myuri: <http://someurl.com/example/>
SELECT ?name
WHERE { ?x myuri:name ?name }
ORDER BY ?name
LIMIT 3
```

```
-------------
| name      |
=============
| "Alessio" |
| "Gino"    |
| "Lina"    |
-------------
```

Esempio Select (2/2)

- Età media

```
prefix myuri: <http://someurl.com/example/>
SELECT AVG(?age)
WHERE { ?x myuri:age ?age }
```

```
-------------
| AVG(?age)   |
===============
|    27       |
-------------
```

CONSTRUCT

- Costruisce un grafo RDF sostituendo le variabili in un insieme di triple di input
- Se qualche istanziazione di variabile produce un costrutto RDF illegale, quella tripla non viene considerata

```
prefix myuri: <http://someurl.com/example/>
CONSTRUCT {?x myuri:name ?name}
WHERE  {?x myuri:name ?name}
```

```
<myuri:person4> myuri:name    "Alessio"
<myuri:person3> myuri:name    "Paolo" .
<myuri:person2> myuri:name    "Gino" .
<myuri:person5> myuri:name    "Pina" .
<myuri:person1> myuri:name    "Lina" .
```

Ask

- Ritorna un booleano che indica se la query ha un match nel grafo o meno

```
prefix myuri: <http://someurl.com/example/>
ASK {?x myuri:name “Alessio”}
```

```
Ask => yes
```

DESRCIBE

- Restituisce un grafo RDF che descrive le risorse trovate

```
prefix myuri: <http://someurl.com/example/>
DESCRIBE ?x
WHERE { ?x myuri:name "Alessio"}
```

```
@prefix myuri:   <http://someurl.com/example/> .
@prefix rdf:     <http://www.w3.org/1999/02/22-
rdf-syntax-ns#> .

<myuri:person4>
      myuri:age      "35" ;
      myuri:marriedTo  <myuri:person5> ;
      myuri:name     "Alessio" .
```

Riferimenti Bibliografici

- http://www.w3.org/TR/rdf-sparql-query
- http://jena.sourceforge.net/ARQ/Tutorial/
- http://www.w3.org/TR/sparql11-property-paths/
- http://www.cambridgesemantics.com/2008/09/sparql-by-example/
- http://www-sop.inria.fr/acacia/soft/corese/
- http://www-sop.inria.fr/edelweiss/software/corese/v2_4_1/manual/next.php