


CORSO I.F.T.S.
"TECNICHE PER LA PROGETTAZIONE E LA GESTIONE DI DATABASE"

Matricola 2014LA0033

DISPENSE DIDATTICHE

MODULO di "DATABASE SEMANTICI"

Ing. Simone Menabeni

Lezione del 08/10/2014

SPARQL

Simone Menabeni
Dipartimento di Ingegneria dell'informazione
Università di Firenze
simone.menabeni@unifi.it

DISIT Lab
http://www.disit.dinfo.unifi.it/

SPARQL

- SPARQL (acronimo di: Simple Protocol and RDF Query Language) è un linguaggio di interrogazione per RDF reso standard dal Data Access Working Group, gruppo di lavoro del consorzio W3C, che lo ha reso raccomandazione ufficiale il 15 gennaio 2008.
- SPARQL consente di estrarre informazioni dalle basi di conoscenza distribuite sul web. RDF descrive i concetti e le relazioni su di essi attraverso l'introduzione di triple (soggetto-predicato-oggetto); se tali triple hanno degli elementi in comune emerge un grafo di conoscenza. SPARQL non fa altro che ricercare dei sotto-grafi corrispondenti alla richiesta dell'utente che effettua la query.

SPARQL

- L'elemento chiave di RDF sono le URI, che identificano le risorse in maniera univoca e consentendo a chi usa SPARQL di scrivere query ben definite e non ambigue. L'elaborazione in SPARQL avviene introducendo due informazioni: il grafo dei dati e il grafo di query (descritto attraverso triple dall'utente).
- L'output può essere di più tipi, ma principalmente si utilizzano interrogazioni di tipo esistenziale (esiste o meno il sotto-grafo ricercato?) o tabellare (elencami i risultati possibili).

SPARQL

- SPARQL può essere usato per esprimere query attraverso varie fonte di dati: i dati possono essere memorizzati in modo nativo come RDF o visualizzati come RDF attraverso middleware.
- SPARQL contiene funzionalità per l'esecuzione di query su pattern graph obbligatori e facoltativi con le loro congiunzioni e disgiunzioni. SPARQL supporta anche l'esecuzione di query attraverso un grafico sorgente RDF. I risultati delle query SPARQL possono essere visualizzati come gruppi di risultati o grafici RDF.

Sintassi

- RDF/XML è la serializzazione raccomandata dal W3C ma non è una buona scelta perché permette molti modi per la rappresentazione di uno stesso grafo
- SPARQL adotta la sintassi Turtle, un'estensione di N-Triple, alternativa estremamente sintetica e intuitiva al tradizionale RDF/XML. Si considerino le seguenti triple RDF:

```
@prefix cd: <http://example.org/cd/>
@prefix: <http://example.org/esempio/>
:Permutation cd:autore "Amon Tobin".
:Bricolage cd:autore "Amon Tobin".
:Amber cd:autore "Autechre".
:Amber cd:anno 1994.
```

Sintassi

- Le asserzioni sono espresse in concise sequenze soggetto-predicato-oggetto e delimitate da un punto fermo.
- *@prefix* introduce prefissi e namespace.
- i due punti senza prefisso (seconda riga) definiscono il namespace di default.
- Gli URI sono inclusi tra parentesi angolari.
- I letterali di tipo stringa sono contrassegnati da virgolette.

Differenze con SQL

- Un database relazionale è caratterizzato da record organizzati in tabelle e il processo di identificazione degli oggetti informativi memorizzati tramite record avviene tramite le primary e foreign key.
- In RDF, ogni risorsa è identificata da un URI. Più grafi RDF possono essere connessi in un unico grafo e l'insieme delle informazioni circa una risorsa può essere recuperato tramite un meccanismo di unificazione sulle URI.

Triple patterns matching

- Le query SPARQL si basano sul meccanismo del "pattern matching" e in particolare su un costrutto, il "triple pattern". I triple patterns sono come triple RDF, salvo che ciascun soggetto del predicato, e oggetto può essere una variabile. In questo modo, i triple pattern forniscono un modello flessibile per la ricerca di corrispondenze.

Triple patterns matching

- Ad esempio nella tripla

  ?titolo cd:autore ?autore.
- in luogo del soggetto e dell'oggetto sono previste due variabili, contrassegnate con un punto interrogativo.
  - Le variabili fungono in un certo senso da incognite dell'interrogazione;
  - *cd:autore* funge invece da costante:
- le triple RDF che trovano riscontro nel modello assoceranno i propri termini alle variabili corrispondenti.

Triple patterns matching

- La maggior parte delle forme di query SPARQL contengono una serie di triple patterns chiamato basic graph pattern. Un basic graph pattern corrisponde a un sottografo dei dati RDF quando i termini RDF di tale sottografo possono essere sostituiti con delle variabili. Per chiarire, ecco una semplice query di selezione SPARQL:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {
    ?titolo cd:autore ?autore.
    ?titolo cd:anno ?anno .
}
```

Triple patterns matching

- L'analogia con SQL è lampante:
  - PREFIX dichiara prefissi e namespace.
  - SELECT definisce le variabili di ricerca da prendere in considerazione nel risultato (nell'esempio: titolo, autore e anno).
  - FROM specifica il set di dati su cui dovrà operare la query (si suppone che le triple siano immagazzinate presso l'indirizzo fittizio "http://cd.com/listacd.ttl").

Triple patterns matching

- È inoltre possibile ricorrere a clausole FROM NAMED e alla parola chiave GRAPH per specificare più insiemi di dati.
- La clausola WHERE, infine, definisce il criterio di selezione specificando tra parentesi graffe uno o più "triple patterns" separati da punto fermo.
- Applicando la query all'insieme di triple del precedente esempio, si ottiene il risultato:

Titolo	**Autore**	**Anno**
"Amber"	"Autechre"	1994

Triple patterns matching

- Il collegamento (binding) tra variabili e termini reperiti corrispondenti (in questo caso, un termine per ciascuna variabile) è reso in forma di tabella come un rapporto campo-valore:
  - le righe rappresentano i singoli risultati,
  - le intestazioni di cella rappresentano le variabili definite nella clausola SELECT,
  - le celle i termini associati alle variabili.

Triple patterns matching

- La parola chiave PREFIX associa un'etichetta (prefisso) con un IRI. Un nome prefisso è composto da un'etichetta e da una parte locale, separati da due punti ":". Un nome prefisso è mappato a un IRI concatenando l'IRI associato al prefisso e la parte locale.
- L'etichetta o la parte locale può essere vuota. Si noti che i nomi locali SPARQL consentono di iniziare con delle cifre e caratteri non alfanumerici (mediante escape backslash), a differenza dei nomi locali XML, che invece non lo permettono.

Sintassi di base

- La sintassi generale per letterali è una stringa (racchiusa tra virgolette doppie, "...", o singoli apici, '...'), con o un tag di lingua opzionale (introdotto da @) o un tipo di dati IRI opzionale o un nome prefisso (introdotto dal ^ ^).

```
"John"
"Hello"@en-GB
"1.4"^^xsd:decimal
```

- Per comodità, i numeri interi possono essere scritti direttamente (cioè senza le virgolette e senza un esplicito tipo di dati IRI) e vengono interpretati come valori letterali di tipo *xsd:integer*;
- I numeri decimali con il '.' nel numero, ma con nessun esponente, sono interpretati come *xsd:decimal*;
- I numeri con esponenti sono interpretati come *xsd:double*.
- I valori di tipo *xsd:boolean* sono scritti come true o false.

Sintassi di base

- Per facilitare la scrittura di valori letterali che contengono virgolette o che sono lunghe e contengono caratteri di nuova riga, SPARQL permette di inserire tali caratteri racchiudendoli in tre singoli apici.
- Una variabile di query viene contrassegnata con l'uso di una "?" o "$", il "?" o "$" non fa parte del nome della variabile. In una query, $abc e ?abc identificano la stessa variabile.

```
?x
?title
?name
```

Sintassi di base

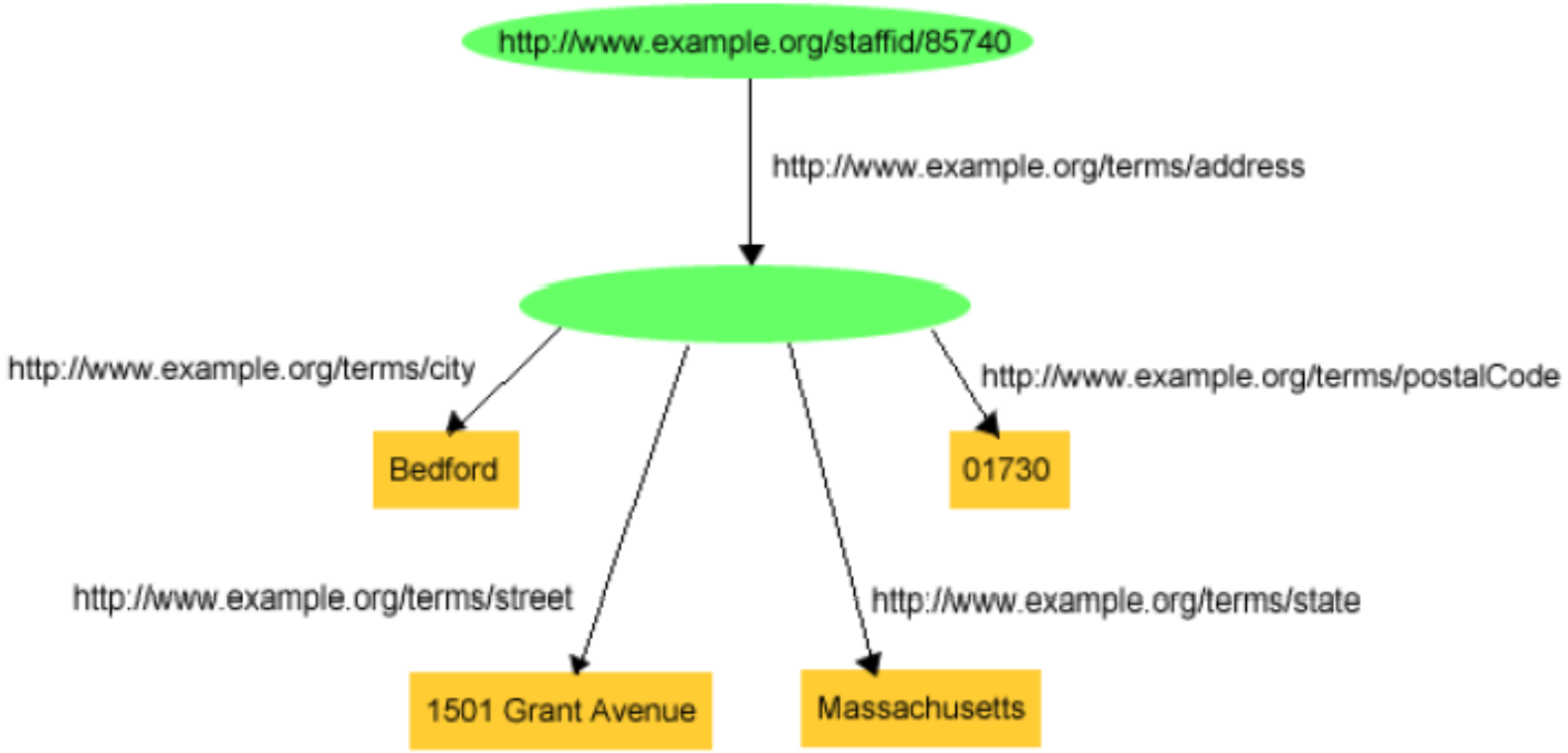
- I nodi vuoti nei pattern grafici agiscono da variabili. I nodi vuoti sono indicati come "_:abc", o nella forma abbreviata "[]".

```
:john ex:hasFather [] .
:john ex:hasFather _:x .
```

- Un nodo vuoto viene utilizzato in maniera univoca per formare il triple pattern. Un nodo vuoto con etichetta viene utilizzato come oggetto di tutte le coppie predicato-oggetto contenute.
- Il nodo vuoto creato può essere utilizzato anche in altri triple pattern nelle posizioni soggetto e oggetto.
- Ad esempio, le forme [:p "v"] e [] :p "v" allocano un nodo vuoto univoco e nell'espressione [:p "v"] :q "w" viene utilizzato come soggetto in un triple pattern, mentre nell'espressione :x :q [:p "v"] viene utilizzato come oggetto del triple pattern.

```
[ ex:hasName "John"] .
[ ex:authorOf :lotr ;
  ex:hasName "Tolkien"] .
```

Sintassi di base



Sintassi di base

- I triple pattern con un soggetto comune possono essere scritti in modo che il soggetto sia scritto solo una volta e viene utilizzato al posto di più triple pattern utilizzando la notazione ";", come ad esempio:

  ?x foaf:name ?name ; foaf:mbox ?mbox.

- Se invece i triple pattern condividono sia il soggetto che il predicato, allora gli oggetti possono essere separati da una “,”, come ad esempio:

  ?x foaf:nick "Alice" , "Alice_“.

Graph patterns e filtraggio

- I basic graph pattern sono insiemi di triple pattern.
- Lo SPARQL graph pattern matching è definito in termini di combinazione dei risultati della corrispondenza dei basic graph patterns.
- Una sequenza di triple pattern, con filtri opzionali, comprende un unico modello grafico di base. Qualsiasi altro graph pattern termina con un basic graph pattern.

Graph patterns e filtraggio

- In una stringa query SPARQL, un gruppo di graph pattern è delimitato dalle parentesi graffe {} .
- Ad esempio:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE {
?x foaf:name ?name .
?x foaf:mbox ?mbox .
}
```

Graph patterns e filtraggio

- Il pattern di gruppo {} corrisponde a qualsiasi grafo (compreso il grafo vuoto) con una soluzione che non vincola le variabili.
- Per esempio

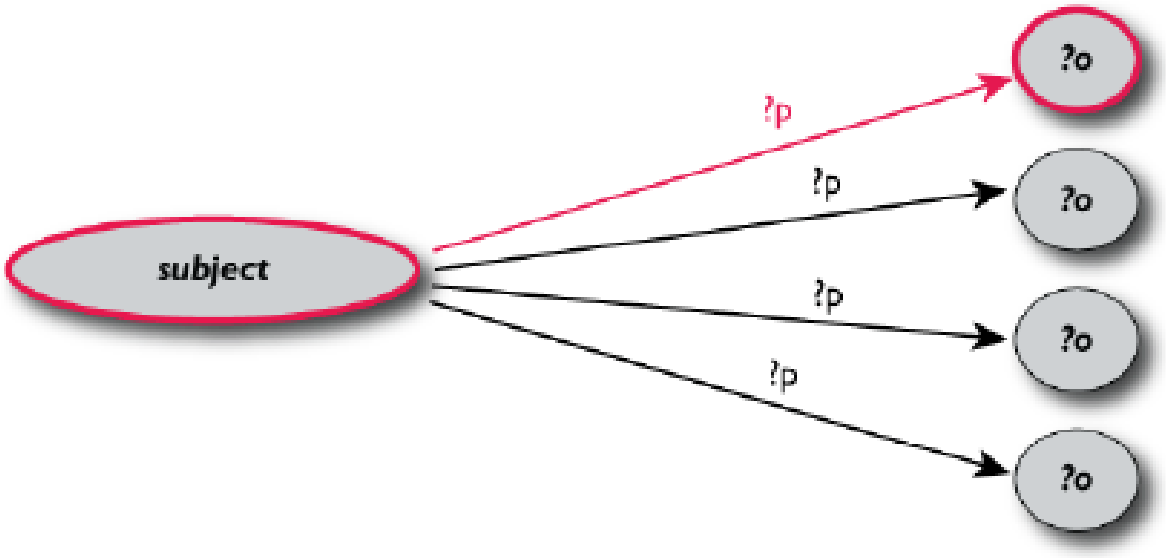
  SELECT ?x

  WHERE {}

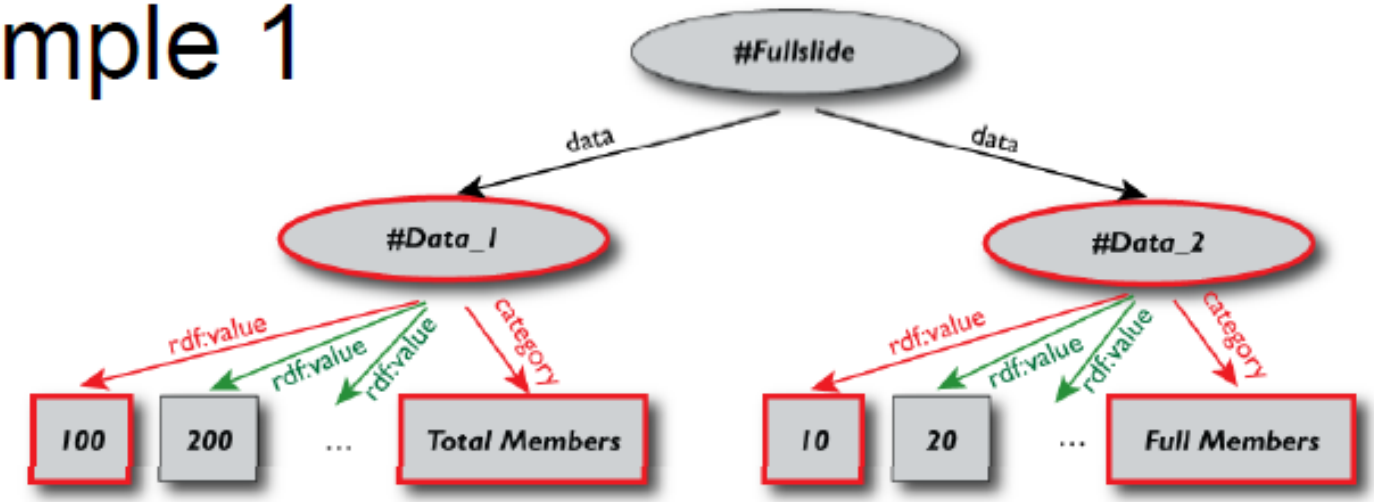
- corrisponde a una soluzione in cui la variabile *x* non è vincolata.

Graph patterns e filtraggio

```
SELECT ?p ?o
WHERE {subject ?p ?o}
```



Example 1




```
SELECT ?cat ?val
WHERE { ?x rdf:value ?val.
        ?x category ?cat }
```

- Returns:

```
[["Total Members",100],["Total Members",200],…,
["Full Members",10],…]
```

Graph patterns e filtraggio

- Un vincolo, espresso dalla parola chiave FILTER, è una restrizione delle soluzioni su tutto il gruppo in cui il filtro appare. È dunque possibile porre restrizioni sui valori da associare alle variabili. Un esempio di applicazione del filtraggio con FILTER è il codice:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?anno
FROM <http://cd.com/listacd.ttl>
WHERE {?titolo cd:anno ?anno.
FILTER (?anno > 2000).
}
```

Graph patterns e filtraggio

- Sono messi a disposizione diversi operatori specifici del linguaggio: tra questi, in particolare abbiamo *regexp*, valido corrispettivo dei criteri di ricerca LIKE dell'SQL che permette di adoperare espressioni regolari per il matching dei letterali. Si consideri:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM <http://cd.com/listacd.ttl>
WHERE
{
    ?titolo cd:autore ?autore .
    FILTER regex(?autore, “^au”, “i”)
}
```

Graph patterns e filtraggio

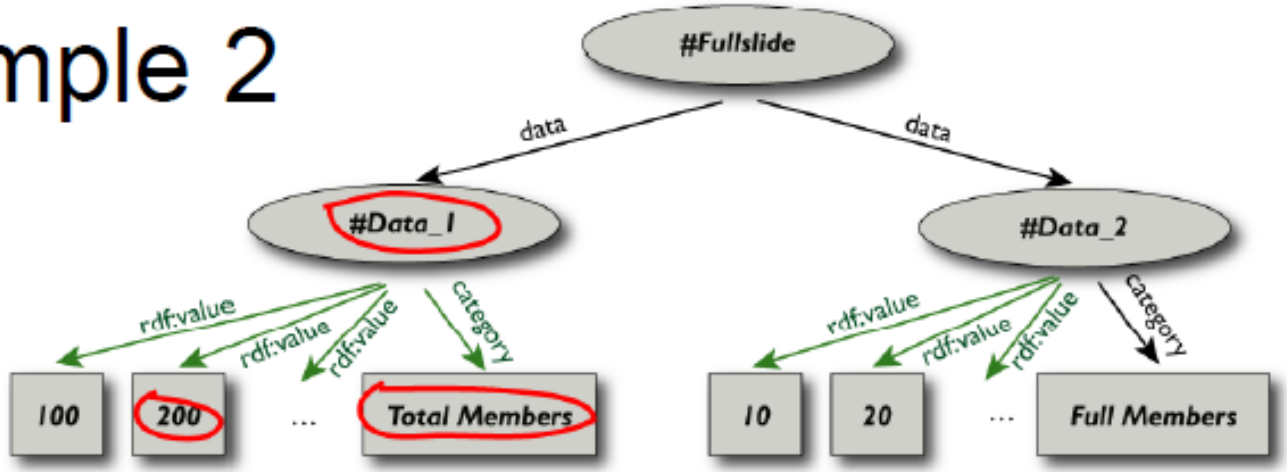
- Sono messi a disposizione diversi operatori specifici del linguaggio: tra questi, in particolare abbiamo *regexp*, valido corrispettivo dei criteri di ricerca LIKE dell'SQL che permette di adoperare espressioni regolari per il matching dei letterali. Si consideri:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore
FROM <http://cd.com/listacd.ttl>
WHERE
{
    ?titolo cd:autore ?autore .
    FILTER regex(?autore, “^au”, “i”)
}
```

- Il filtro seleziona, senza riguardo per maiuscole o minuscole, solo gli autori che iniziano per "au". Otteniamo il risultato:

Titolo	**Autore**
"Amber"	"Autechre"

Example 2




```
SELECT ?cat ?val
WHERE { ?x rdf:value ?val.
        ?x category ?cat.
        FILTER(?val>=200). }
```

- Returns:

```
[["Total Members",200],…]
```

Ricerca per opzioni e alternative

- Tutte le query viste finora hanno catturato esclusivamente le triple dotate di tutti i termini richiesti, escludendo le triple che possedevano soltanto alcuni termini.
- È utile poter disporre di query che permettono l'aggiunta di informazioni alla soluzione nel caso in cui tale informazione è disponibile, ma senza rifiutare una soluzione perché una parte del query pattern non corrisponde.
- Questi servizi sono offerti da associazioni opzionali: se la parte opzionale non corrisponde, anche se non crea un binding, quantomeno non elimina la soluzione.
- Le parti opzionali del graph pattern possono essere specificate sintatticamente con la parola chiave OPTIONAL applicato ad un graph pattern. È possibile formulare le query in modo più elastico, prevedendo l'eventuale assenza di alcuni termini.

Ricerca per opzioni e alternative

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM
<http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
OPTIONAL {?titolo cd:anno ?anno}
}
```

- Nell'esempio, il secondo pattern è dichiarato opzionale: l'informazione è aggiunta al risultato solo se disponibile, altrimenti le variabili compariranno prive di valore (unbound).

Ricerca per opzioni e alternative

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM
<http://cd.com/listacd.ttl>
WHERE {?titolo cd:autore ?autore.
OPTIONAL {?titolo cd:anno ?anno}
}
```

- Nell'esempio, il secondo pattern è dichiarato opzionale: l'informazione è aggiunta al risultato solo se disponibile, altrimenti le variabili compariranno prive di valore (unbound).
- Il risultato della query:

Titolo	Autore	Anno
"Permutation"	"Amon Tobin"	
"Bricolage"	"Amon Tobin"	
"Amber"	"Autechre"	1994

Ricerca per opzioni e alternative

- I graph pattern vengono definiti in modo ricorsivo. Un graph pattern può avere zero o più graph pattern opzionali, e qualsiasi parte di un query pattern può avere un componente opzionale. Nell'esempio seguente ci sono due graph pattern opzionali.

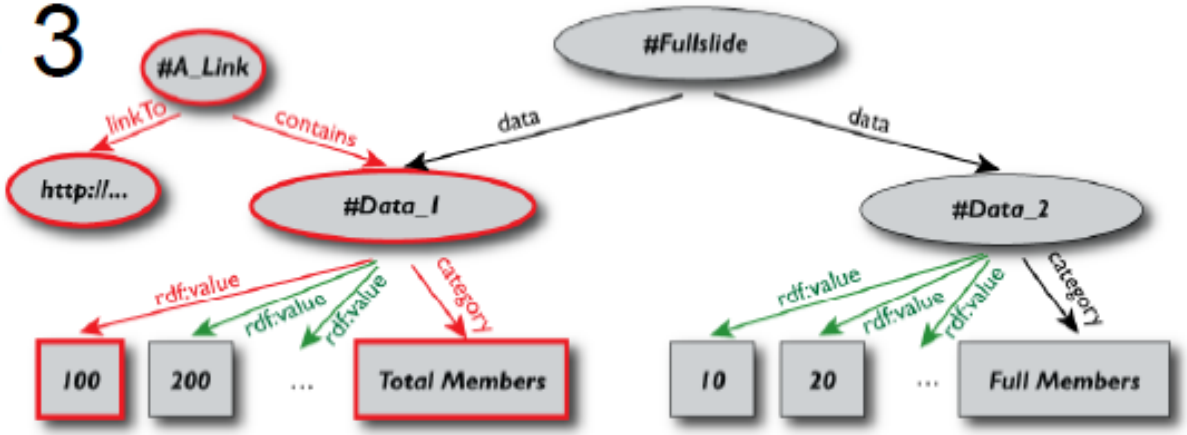
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE
{ ?x foaf:name
?name .
OPTIONAL { ?x foaf:mbox ?mbox } .
OPTIONAL { ?x foaf:homepage ?hpage }
}
```

Ricerca per opzioni e alternative

- Con un output del tipo

Name	Mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

Example 3

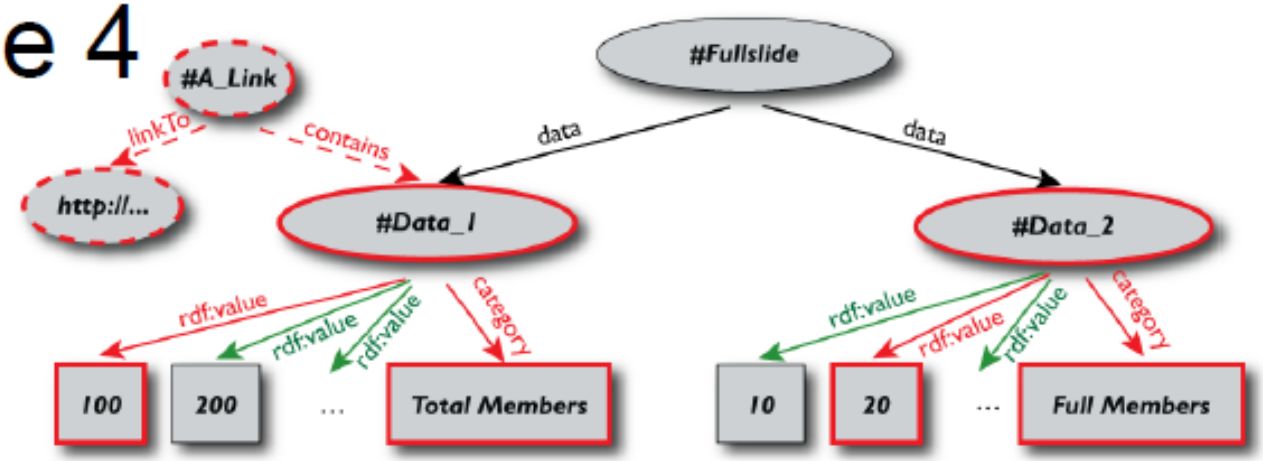



```
SELECT ?cat ?val ?uri
WHERE { ?x rdf:value ?val.
        ?x category ?cat.
        ?al contains ?x.
        ?al linkTo ?uri }
```

- Returns:

```
[["Total Members",100,http://...)],…,]
```

Example 4




```
SELECT ?cat ?val ?uri
WHERE  { ?x rdf:value ?val.
         ?x category ?cat.
OPTIONAL ?al contains ?x.
         ?al linkTo ?uri }
```

- Returns:

```
[["Total Members",100,http://...], …,
["Full Members",20, ],…,]
```

Ricerca per opzioni e alternative

- SPARQL fornisce un mezzo di combinazione di graph pattern in modo che si possa trovare una corrispondenza tra diversi graph pattern alternativi. Questi pattern alternativi sono sintatticamente specificati con la parola chiave UNION. Questo è un altro modo per assicurare una certa elasticità nel reperimento dei dati. Consideriamo l'esempio:

```
PREFIX cd: <http://example.org/cd/>
SELECT ?titolo ?autore ?anno
FROM
<http://cd.com/listacd.ttl>
WHERE{
{?titolo cd:autore ?autore}
UNION
{?titolo cd:anno ?anno}
}
```

- La parola chiave UNION esprime un OR logico: la query non si limita pertanto alle triple che soddisfano entrambi i triple patterns, ma cattura sia quelli che soddisfano il primo, sia quelli che soddisfano il secondo.

Negazione

- Il linguaggio di query SPARQL incorpora due tipi di negazione, una basata sul filtraggio dei risultati a seconda che un graph pattern trova o non trova una corrispondenza nel contesto delle soluzioni filtrate, e l'altra basata sulla eliminazione delle soluzioni relative ad un altro pattern.
- La negazione del filtraggio delle soluzioni di una query viene fatto all'interno di un'espressione FILTER con NOT EXISTS e EXISTS.
- Si noti che le regole di filtraggio si applicano a tutto il gruppo in cui il filtro appare.
- L'espressione NOT EXISTS del filtro verifica se un graph pattern non corrisponde all'insieme dei dati dei valori delle variabili nel gruppo di graph pattern in cui opera il filtro. Esso non genera altre associazioni di binding.
- È prevista anche una espressione EXISTS del filtro. In questo caso si verifica se il pattern può essere trovato nei dati; esso non genera altre associazioni.

Negazione

- L'altro stile di negazione disponibile in SPARQL è MINUS che valuta entrambi gli argomenti e quindi calcola le soluzioni nel lato sinistro che non sono compatibili con le soluzioni sul lato destro. Un esempio di codice che utilizza MINUS è

```
PREFIX :
<http://example/>
PREFIX foaf:
<http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?s
WHERE {
?s ?p ?o . MINUS { ?s foaf:givenName "Bob" .}
}
```

Associazione

- Il valore di un'espressione può essere aggiunto ad una soluzione di mapping associando ad una nuova variabile il valore dell'espressione, che è un termine RDF.
- Questa variabile può quindi essere utilizzata nella query e può anche essere restituita nei risultati.
- La forma di assegnazione è *(espressione AS ?var)*.

Associazione

- Tre forme di sintassi permettono questo: la parola chiave BIND, le espressioni nella clausola SELECT e le espressioni nella clausola GROUP BY.
- La forma BIND consente l'assegnazione di un valore a una variabile in un gruppo di graph pattern. L'uso di BIND è un elemento separatore di gruppi di graph pattern e termina qualsiasi basic graph pattern. Un esempio di codice che usa BIND è

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX ns:<http://example.org/ns#>
SELECT ?title ?price
{
?x ns:price ?p .
?x ns:discount ?discount
BIND (?p*(1-?discount) AS ?price)
FILTER(?price < 20)
?x dc:title ?title .
}
```

Aggregazioni

- Gli aggregati applicano espressioni su gruppi di soluzioni.
- Per impostazione predefinita, un insieme di soluzioni è costituito da un singolo gruppo, che contiene tutte le soluzioni.
- Il raggruppamento può essere impostato con la sintassi GROUP BY.
- Gli aggregati definiti in SPARQL sono COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT e SAMPLE. Gli aggregati sono utilizzati quando si desidera visualizzare il risultato di una query calcolato su un gruppo di soluzioni, piuttosto che su una singola soluzione.

Aggregazioni

- Consideriamo i dati:

```
@prefix : <http://books.example/> .
:org1 :affiliates :auth1, :auth2 .
:auth1 :writesBook :book1, :book2 .
:book1 :price 9 .
:book2 :price 5 .
:auth2 :writesBook :book3 .
:book3 :price 7 .
:org2 :affiliates :auth3 .
:auth3 :writesBook :book4 .
:book4 :price 7 .
```

- e la query

```
PREFIX : <http://books.example/>
SELECT (SUM(?lprice) AS ?totalPrice)
WHERE {
?org :affiliates ?auth .
?auth :writesBook ?book .
?book :price ?lprice .
}
GROUP BY ?org
HAVING (SUM(?lprice) > 10)
```

Aggregazioni

- Il risultato

totalPrice
21

- Questo esempio mostra due caratteristiche degli aggregati: GROUP BY, che raggruppa le soluzioni della query secondo una o più espressioni (in questo caso ?org), e HAVING, che è analogo all'espressione FILTER, ma opera su raggruppamenti di soluzioni, piuttosto che su singole soluzioni.
- L'esempio è prodotto da soluzioni di raggruppamento in base alla espressione GROUP BY (vale a dire tutte le soluzioni dove ?org assume un particolare valore all'interno dello stesso gruppo), e la valutazione della funzione SUM su tale gruppo. I gruppi vengono filtrati dall'espressione HAVING, che rimuove tutti i gruppi che hanno il valore SUM(?lprice) non maggiore di 10.

Sub-query e manipolazione del risultato

- Le sub-query sono un modo per inserire delle query SPARQL in altre query, al fine di raggiungere risultati che non possono essere conseguiti diversamente, come ad esempio la limitazione del numero dei risultati di alcuni sotto espressioni nella query.
- A causa della natura bottom-up di valutazione delle query SPARQL, le sub-query vengono valutate logicamente prima, ed i risultati sono proiettati fino alla query esterna.
- Si noti che solo le variabili proiettate fuori dalle sub-query fino alla query più esterna saranno visibili. Normalmente una sequenza di soluzioni fornite da una query SPARQL prevede la possibilità di mostrare soluzioni duplicate.

Sub-query e manipolazione del risultato

- Come in SQL, è possibile escludere dal risultato i valori duplicati mediante la parola chiave DISTINCT, ad esempio:

SELECT DISTINCT ?titolo ?autore

- Altri costrutti supportati da SPARQL per la manipolazione del risultato sono:
  - *ORDER BY DESC(?autore)*
  - *OFFSET 10*
  - *LIMIT 10*

Sub-query e manipolazione del risultato

- L'espressione ORDER BY imposta l'ordine dei risultati della query: i risultati verranno presentati in ordine crescente se è utilizzata la clausola ASC o nessuna clausola, o in ordine decrescente se è utilizzata la clausola DESC (come nell'esempio basato sulla variabile ?autore).

Sub-query e manipolazione del risultato

- OFFSET fa in modo che le soluzioni generate iniziano dopo il numero di soluzioni specificate; in pratica permette di saltare un certo numero di risultati, escludendo, stando all'esempio, i primi 10. Un OFFSET di 0 non ha alcun effetto.
- LIMIT pone un limite superiore al numero dei risultati che vengono restituiti, limitandoli, secondo quanto indicato nell'esempio, ai soli primi 10.

Query forms

- SPARQL ha quattro query forms. Queste query forms utilizzano le soluzioni di pattern matching per formare un insieme di risultati o grafi RDF. Le query forms sono:
  - SELECT: restituisce tutti, o un sottoinsieme, delle variabili vincolate a un query pattern match;
  - CONSTRUCT: restituisce un grafo RDF costruito sostituendo le variabili in un insieme di triple pattern;
  - ASK: restituisce un valore booleano che indica se è stata trovata una corrispondenza al query pattern oppure no;
  - DESCRIBE: restituisce un grafo RDF che descrive le risorse trovate.

Query forms

- La query form CONSTRUCT restituisce un singolo grafo RDF specificato da un graph template. Il risultato è un grafo RDF formato prendendo ogni soluzione della query nella sequenza di soluzioni, sostituendole alle variabili nel graph template, e combinando le triple in un unico grafo RDF.
- Se una qualsiasi istanza produce una tripla contenente una variabile unbound o un costrutto RDF illegale, come un letterale nella posizione soggetto o predicato, allora la tripla non è inclusa nel grafo RDF di output.
- Il graph template può contenere triple senza variabili (note come ground triples o triple esplicite), e questi appaiono anche nel grafo RDF restituito in uscita dalla forma query CONSTRUCT.

Query forms

- Ad esempio, con i dati di partenza

```
@prefix  foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.org> .
```

- la query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { <http://example.org/person#Alice> vcard:FN ?name }
WHERE { ?x foaf:name ?name }
```

Query forms

- produce in uscita le proprietà vcard

@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
<http://example.org/person#Alice> vcard:FN "Alice" .

Query forms

- Le applicazioni possono utilizzare la form ASK per testare se un query pattern ha una soluzione oppure no. Nessuna informazione è restituita dalle possibili soluzioni della query, dice solo se una soluzione esiste oppure no. Restituisce o false o true.

Query forms

- La form DESCRIBE restituisce un unico grafo RDF come risultato contenente i dati relativi alle risorse RDF.
- Il query pattern viene utilizzato per creare un insieme di risultati.
- La forma DESCRIBE prende ciascuna delle risorse individuate in una soluzione, insieme a tutte le risorse nominate direttamente dall'IRI, e assembla un unico grafo RDF prendendo una "descrizione", che può provenire da qualsiasi informazione disponibile, compreso il Dataset RDF.
- La sintassi DESCRBE * è una abbreviazione che descrive tutte le variabili in una query.

SPARQL endpoints

- Un endpoint SPARQL accetta in ingresso query e ritorna I risultati via HTTP
- Endpoint generici permettono di effettuare query su qualsiasi dato RDF accessibile via Web
- Endpoint specifici sono strettamenti colleggati a specifici dataset
- I risultati di query SPARQL possono essere presentati in vari formati:
  - XML, JSON, RDF, HTML
  - JSON (JavaScript Object Notation): formato di interscambio di dati leggero; formato text-based e human-readable per rappresentare semplici strutture dati e array associativi

Dataset: DBPedia

- DBPedia è una versione RDF delle informazioni contenute in Wikipedia
- Contiene dati derivati dalle infobox di Wikipedia, da abstract di articoli e da vari link esterni
- Contiene più di 100 milioni di triple
- Dataset: http://dbpedia.org/

Esempio – esplorare DBPedia

- Trovare 15 concetti di esempio nel dataset DBPedia

```
SELECT DISTINCT ?concept
WHERE {
    ?s a ?concept .
} LIMIT 15
```

concept
http://www.w3.org/2004/02/skos/core#Concept
http://dbpedia.org/ontology/MusicalWork
http://dbpedia.org/ontology/Resource
http://dbpedia.org/ontology/Work
http://dbpedia.org/ontology/Album
http://dbpedia.org/ontology/Musical
http://umbel.org/umbel/sc/Product
http://umbel.org/umbel/ac/Artifact
http://dbpedia.org/class/yago/1982Novels
http://dbpedia.org/ontology/Book
http://dbpedia.org/class/yago/EnglishAstronomers
http://dbpedia.org/class/yago/EnglishPoliticians
http://umbel.org/umbel/sc/Astronomer
http://dbpedia.org/class/yago/LivingPeople
http://dbpedia.org/class/yago/AmericanCompetitiveEaters

Esempio – esplorare DBPedia

- Trovare tutte le nazioni senza sbocco sul mare con una popolazione superiore ai 15 milioni.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .
    FILTER (?population > 15000000) .
}
```

Dataset: Jamendo

- Jamendo è una collezione di musica tutta con licenza libera (Creative Commons)
  - http://www.jamendo.com/it/
- DBTune.org ospita una versione interrogabile delle informazioni sulla collezione di musica di Jamendo
  - Dati su migliaia di artisti, decine di migliaia di album e circa 100,000 canzoni
  - http://dbtune.org/

Esempio – modo sbagliato

- Trovare tutti gli artisti presenti in Jamendo insieme con le loro immagini, home page, e la località di provenienza

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name ;
     foaf:img ?img ;
     foaf:homepage ?hp ;
     foaf:based_near ?loc .
}
```

Esempio – modo corretto

- Non tutti gli artisti hanno un'immagine, homepage o località!
- OPTIONAL prova a identificare un pattern, ma non fa fallire l'intera query se falliscono i pattern opzionali
- Se un pattern OPTIONAL fallisce per una particolare soluzione, le variabili in quel pattern rimangono vuote

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name .
  OPTIONAL { ?a foaf:img ?img }
  OPTIONAL { ?a foaf:homepage ?hp }
  OPTIONAL { ?a foaf:based_near ?loc }
}
```

Dataset: semanticweb.org

- data.semanticweb.org ospita dati RDF riguardanti workshop, calendari e relatori per la serie di eventi delle conferenze International Semantic Web (ISWC) e European Semantic Web (ESWC)
- Presenta i dati attraverso le ontologie FOAF, SWRC, e iCal
- I dati per ogni singolo evento delle due conferenze sono memorizzati in uno specifico named graph
- i.e., c'è un named graph per ogni evento della conferenza contenuto in questo dataset
- http://data.semanticweb.org/

Esempio – interrogare i named graphs

- Trovare le persone che sono state coinvolte in almeno 3 eventi delle conferenze ISWC o ESWC

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?person
WHERE {
    GRAPH ?g1 { ?person a foaf:Person }
    GRAPH ?g2 { ?person a foaf:Person }
    GRAPH ?g3 { ?person a foaf:Person }
    FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3) .
}
```

Esercizi- RDF

```
@prefix : <http://example.org/data#> .
@prefix ont: <http://example.org/myOntology#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

:john
  vcard:FN "John Smith" ;
  vcard:N [
    vcard:Given "John" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 32 ;
  ont:marriedTo :mary .
:mary
  vcard:FN "Mary Smith" ;
  vcard:N [
    vcard:Given "Mary" ;
    vcard:Family "Smith" ] ;
  ont:hasAge 29 .
```

SPARQL query – Esercizio 1

- Trovare i nomi completi di tutte le persone nel grafo

SPARQL query – Esercizio 1

- Trovare i nomi completi di tutte le persone nel grafo

```
PREFIX vCard:
<http://www.w3.org/2001/vcardrdf/3.0#>
SELECT ?fullName
WHERE {?x vCard:FN ?fullName}
```

- Risultati

```
fullName
==============
"John Smith"
"Mary Smith"
```

SPARQL query – Esercizio 2

- Trovare la relazione tra John e Mary

SPARQL query – Esercizio 2

- Trovare la relazione tra John e Mary

```
PREFIX : <http://example.org/data#>
SELECT ?p
WHERE { :john ?p :mary }
```

- Risultati

```
p
=================
<http://example.org/myOntology#marriedTo>
```

SPARQL query – Esercizio 3

- Trovare il coniuge di una persona che si chiama John Smith

SPARQL query – Esercizio 3

- Trovare il coniuge di una persona che si chiama John Smith

```
PREFIX vCard:
<http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
       ?x ont:marriedTo ?y}
```

- Risultati

```
y
================
<http://example.org/data#mary>
```

SPARQL query – Esercizio 4

- Trovare nome completo e il nome di tutte le persone nella base di conoscenza

SPARQL query – Esercizio 4

- Trovare il nome completo e il nome di tutte le persone nella base di conoscenza

```
PREFIX vCard:
<http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
       ?name vCard:Given ?firstName}
```

- Risultati

```
    name     firstName
=========================
"John Smith"   "John"
"Mary Smith"   "Mary"
```

SPARQL query – Esercizio 4 (corretto)

- Trovare il nome completo e il nome di tutte le persone nella base di conoscenza

```
PREFIX vCard: <http://www.w3.org/2006/vcard/ns#>
SELECT ?name ?firstName
WHERE {     ?x vCard:FN ?name .
            ?x vCard:N ?xname .
            ?xname vCard:Given ?firstName}
```

- Risultati

```
    name     firstName
=========================
"John Smith"   "John"
"Mary Smith"   "Mary"
```

SPARQL query – Esercizio 5

- Trovare tutte le persone over 30 nella base di conoscenza

SPARQL query – Esercizio 5

- Trovare tutte le persone over 30 nella base di conoscenza

```
PREFIX ont: <http://example.org/myOntology#>
SELECT ?x
WHERE {?x ont:hasAge ?age .
       FILTER(?age > 30)}
```

- Risultati

```
x
==================
<http://example.org/data#john>
```