



University of Rome “Tor Vergata”

SPARQL

Armando Stellato <stellato@uniroma2.it>

Linguaggi di interrogazione per il Web Semantico

- RDF è una recommendation W3C sin dal 1998.
- Negli anni a seguire, vari linguaggi hanno fornito soluzioni diversificate per l'interrogazione di repository RDF. Possiamo identificare la varietà di approcci secondo le seguenti categorie:
 - SQL-like: RDQL/Squish, SeRQL, RDFDB QL, RQL, ...
 - Rule-based: N3QL, Triple, DQL, OWL-QL, ...
 - XML based: XSLT, XPath, XQuery
 - XPath-like: Versa, RDFPath
- Tra questi, i linguaggi simili a SQL hanno incontrato maggior riscontro nella comunità scientifica e, in particolare, nel Data Access Working Group istituito dalla W3C con lo scopo di definire le basi per un linguaggio di interrogazione per RDF. Dalle ceneri di diverse proposte, è emerso SPARQL, divenuto W3C Recommendation dal 15 Gennaio 2008

SPARQL: Introduzione

- SPARQL (Simple Protocol and RDF Query Language) è un *linguaggio di interrogazione* per RDF.
- Da un punto di vista sintattico, può ricordare SQL (Structured Query Language), il linguaggio per interrogare basi di dati, anche se i due modelli di rappresentazione sottostanti presentano notevoli differenze:
 - Un DB relazionale è caratterizzato da record organizzati in tabelle. 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

- SPARQL è anche un protocollo, *on top* di protocolli esistenti per il trasporto di informazione, come http, attraverso il quale è possibile spedire query su RDF a dei triple store e recuperare i risultati da questi forniti.

- Es:

`http://.../qps`

`?query-lang=http://www.w3.org/TR/rdf-sparql-query/`

`&graph-id=http://planetrdf.com/bloggers.rdf`

`&query=PREFIX foaf: <http://xmlns.com/foaf/0.1/...`

Graph Pattern

Le query SPARQL definiscono un template, chiamato *Graph Pattern*, che deve *unificare* con le triple presenti nel grafo interrogato.

L'esempio seguente mostra una query ad un grafo dove si chiedono tutti gli oggetti del dominio che sono legati alla risorsa `tor_vergata` tramite la proprietà `works_in` (ovvero tutti i dipendenti di Tor Vergata).

```
PREFIX : http://art.uniroma2.it/ontologies/st\_example#
```

```
SELECT ?x
```

```
WHERE { ?x :worksIn :tor_vergata . }
```

Unificazione (Matching)

- Dati due termini, questi unificano se:
 - sono identici (sono cioè formalmente lo stesso oggetto)
 - i termini sono interamente rappresentati o sintatticamente costituiti da variabili e può essere effettuata una *sostituzione di (tali) variabili* tale che i due termini unifichino
- Una unificazione (*matching*) di un graph pattern GP su un grafo G è fornita da una sostituzione di variabili S tale che $S(GP)$ sia un sottografo di G
 - ...ovvero, le triple presenti nel pattern, una volta istanziate attraverso la sostituzione di variabili S, devono essere presenti nel grafo interrogato

- Basata su serializzazione *Turtle* di RDF
- Oggetti:
 - URI, rappresentabili anche tramite QNames (qualified names), ovvero con sintassi <prefisso>:<resourcenam>
 - Literals
 - Variabili (precedute, indifferentemente, da un ? o un \$)
 - Es: ?var o \$var
 - Operatori su grafi
- Sintassi Triple Pattern
 - Le singole triple sono separate da punti (.)
 - *Predicate-Object Lists*: In un pattern è possibile specificare più triple aventi uno stesso soggetto indicando questo una volta sola e facendolo seguire da una lista di coppie predicato-oggetto seguite dal simbolo “;”
es:
`?person foaf:knows :mario_rossi ;
foaf:name ?name .`
 - *Object Lists*: Se dei triple-pattern condividono sia il predicato che il soggetto, allora è possibile esprimere tali triple indicandone il soggetto e il predicato e separando i vari oggetti con il simbolo “,”
es:
`?person foaf:knows :mario_rossi , :gino_bianchi .`
 - *rdf:type*: la prop *rdf:type* è esprimibile attraverso lo shortcut “a”. Es: `?x a :Person .`

SPARQL query: struttura

Osserviamo la seguente query:

```
PREFIX stx: < http://art.uniroma2.it/ontologies/st\_example# >
```

```
SELECT ?person
```

```
FROM < http://art.uniroma2.it/ontologies/st\_example >
```

```
WHERE { ?person stx:name ?name. }
```



PREFIX assegna un prefisso ad un namespace. Tale prefisso può essere utilizzato per riferire tramite un qname compatto (rappresentato da <prefisso>:<resourcename>) le risorse presenti nel grafo da interrogare.

SELECT fornisce l'elenco delle variabili da riportare nel risultato

FROM indica la sorgente dei dati da interrogare (ad esempio, come in questo caso, un file owl presente su web)

WHERE definisce una serie di vincoli sulle triple da estrarre attraverso un GRAPH Pattern da unificare sul grafo interrogato. Il termine WHERE può essere rimosso inserendo direttamente il pattern tra parentesi graffe, senza alterare il risultato della query

SPARQL query: esecuzione

- Un Graph Pattern è *unificato* rispetto a uno sorgente dati RDF
- *Per ogni* unificazione del pattern, viene generata una soluzione
- La sequenza di soluzioni è successivamente filtrata attraverso le primitive:
 - Project, distinct, order, limit/offset
- A seconda dell'operatore scelto tra:
 - SELECT, CONSTRUCT, DESCRIBE o ASKviene applicato un *result form* differente

SPARQL query: risultati

- I risultati sono espressi in forma tabulare

| x | y | z |
|-------------------------|--------------------------|-----------|
| “Maria Teresa Pazienza” | <http://this_example/mt> | Professor |
| “Armando Stellato” | <http://this_example/as> | |

- In questo esempio ci sono due soluzioni
- Ciascuna offre istanziazione differente delle variabili della query.
- Come si può vedere dal secondo risultato, la variabile di una query non deve necessariamente essere istanziata con un valore

Matching su datatype

Language Tags: Se un literal è espresso da un language tag, è possibile cercare delle triple che lo contengono solo riferendo anche il linguaggio con il simbolo @:

```
SELECT ?v WHERE { ?v ?p "cat"@en }
```

Tipi di Datatype: devo specificarli nella espressione, es:

```
SELECT ?v WHERE { ?v ?p "abc"^^<http://example.org/datatype#specialDatatype> }
```

Numeri: possono essere espressi direttamente, e vengono automaticamente *boxed* nel loro tipo più immediato, ad es:

```
SELECT ?v WHERE { ?v ?p 42 }
```

viene automaticamente sostituito con:

```
SELECT ?v WHERE { ?v ?p "42"^^<http://www.w3.org/2001/XMLSchema#integer> }
```

Blank nodes

Il risultato di una query può contenere dei “blank node”. I *blank node* dell’esempio sono scritti con un prefisso “_:” seguito da una label anonima.

Le label anonime hanno il loro ambito di risoluzione limitato al result set nel quale appaiono (o, nel caso di un query form di una CONSTRUCT, al grafo risultato).

Label che unificano perciò in uno stesso result set individuano uno stesso nodo, ma la loro ricorrenza in diverse sessioni/risultati/etc... non determina alcun tipo di legame

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

```
_:a foaf:name "Alice" .
```

```
_:b foaf:name "Bob" .
```

Sintassi alternativa:

Le due forme alternative: [:p “v”] e [] :p “v” individuano entrambe un BNode e sono equivalenti a:

```
_:b57 :p "v" .
```

Queste possono essere ulteriormente combinate con altre triple, es:

```
:x :q [ :p "v" ] .
```

è equivalente a

```
:x :q _:b57 .  
_:b57 :p "v"
```

Vincoli

È possibile vincolare il matching del Graph Pattern utilizzando diversi operatori, funzioni (e.g. espressioni regolari) o anche delle estensioni dedicate.

Es: vincoli su interi...

...o su stringhe

```
FILTER ?x < 3 .
```

```
FILTER regex(?name ,  
"Smith") .
```

Vincoli (Test sui valori)

- Possono essere usati tutti gli operatori e funzioni di XQuery 1.0 e XPath 2.0
- I tipi sono quelli dell'XML Schema Definition (XSD): boolean, string, integer, decimal, float, double, dateTime
- Essendo definito un ordinamento parziale per i suddetti tipi di dato, è possibile operare confronti tramite gli operatori <, >, <=, >= e per ognuno di essi.
- Uguaglianza: = (vedi specifiche *rdf-term equality* su documento: [RDF Concepts and Abstract Syntax](#)), !=, sameTerm. Per tipi di dato *custom*, è necessario fornire una estensione che sia in grado di trattarli ridefinendo “=”. sameTerm non necessita di tali estensioni.
- Funzioni booleane di interrogazione sullo stato/natura di una variabile:
 - BOUND (true se la variabile è istanziata con un valore), ISURI(ISIRI), ISBLANK, ISLITERAL
- Funzioni di analisi/estrazione:
 - REGEX: Invoca la funzione XPath [fn:matches](#) per unificare il testo rispetto ad un pattern di una espressione regolare
 - LANGMATCHES: verifica che il language tag di un letterale unifichi con il pattern proposto (vedere specifica [RFC4647](#))
 - LANG: restituisce il language tag di un letterale, se questo è presente (“” altrimenti)
 - DATATYPE: restituisce il tipo di dato di una risorsa
 - STR: restituisce la lexical form di un lexical o cmq una rappresentazione a stringa di un URI
- Operatori logici booleani: AND (&&) e OR (||)
- Funzioni definite e implementate da terzi (chiaramente utilizzabili solo su implementazioni di engine SPARQL che le supportano)

Utilissima per implementare la: **negation-as-failure !**

Casting

- SPARQL ammette casting impliciti tra tipi di dati primitivi, attraverso il riutilizzo dei costruttori definiti in [XQuery 1.0 and XPath 2.0 Functions and Operators](#)
- La tabella qui a destra riporta i casting ammessi (Y), non permessi (N) e quelli che dipendono dallo specifico valore considerato (M). Un esempio di quest'ultima categoria è dato dalla conversione da un xsd:string verso un qualsiasi dato numerico: ovviamente il casting è effettuabile solo a condizione che il valore di tipo stringa sia interpretabile come un valore numerico del tipo considerato.

| From \ To | str | flt | dbl | dec | int | dT | bool |
|-----------|-----|-----|-----|-----|-----|----|------|
| str | Y | M | M | M | M | M | M |
| flt | Y | Y | Y | M | M | N | Y |
| dbl | Y | Y | Y | M | M | N | Y |
| dec | Y | Y | Y | Y | Y | N | Y |
| int | Y | Y | Y | Y | Y | N | Y |
| dT | Y | N | N | N | N | Y | N |
| bool | Y | Y | Y | Y | Y | N | Y |
| IRI | Y | N | N | N | N | N | N |
| ltrl | Y | M | M | M | M | M | M |

OPTIONAL

Un Graph Pattern può avere dei componenti opzionali, espressi tramite la keyword OPTIONAL

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

Restituirà tutti quelli che hanno un foaf:name nel grafo e – se *la hanno* – ne riporterà la foaf:mbox.

L'operatore OPTIONAL è left-associative, quindi il pattern:

```
pattern OPTIONAL { pattern } OPTIONAL { pattern }
```

è equivalente a:

```
{ pattern OPTIONAL { pattern } } OPTIONAL { pattern }
```

UNION

È possibile combinare diversi Graph Pattern di modo che *almeno* uno di essi unifichi con il grafo interrogato.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX stx: <http://art.uniroma2.it/ontologies/st_example#>
```

```
SELECT ?guy
```

```
WHERE { { ?guy a foaf:Person } UNION { ?guy a stx:Person } }
```

In questo caso verranno restituite tutte le risorse definite come appartenenti alla classe `stx:Person` o *anche* alla classe `foaf:Person`

Lavorare su più Dataset RDF: FROM NAMED

Le sorgenti informative in un dataset RDF sono principalmente due: un grafo di *default* (o *background graph*) e un set di *named graphs*. Possono essere invocate nel modo seguente:

```
FROM <http://art.uniroma2.it/ontologies/st_example>  
FROM NAMED <http://xmlns.com/foaf/0.1/>  
FROM NAMED <http://purl.org/dc/elements/1.1/>
```

Come la clausola FROM, anche FROM NAMED specifica una sorgente dati RDF, indicandola però come un NAMED GRAPH

FROM, FROM NAMED e il DEFAULT GRAPH

Vi è notevole confusione sull'uso di FROM e FROM NAMED, anche dovuta a una non eccessiva chiarezza delle specifiche. Per chiarire, introduciamo un'altra nozione: quella di *default graph*.

Il concetto di default graph è in realtà duplice, a seconda che si parli di lettura (query) o di scrittura (update) e di ciò che si trova concretamente dentro il triple store.

1. SPARQL QUERY definisce il "default graph" di un dataset come un grafo definito come
 1. l'*RDF Merge* di tutti i named graph, se non viene fornita alcuna clausola FROM/FROM NAMED
 2. l'*RDF Merge* dinamico dei singoli grafi specificati in uno o più punti FROM
2. SPARQL UPDATE definisce un default graph di un "Graph Store" come l'unico "UNNAMED GRAPH" nel dataset. Se in un aggiornamento non viene specificato alcun grafo, le informazioni vengono memorizzate nel grafo senza nome.

Alcune note interessanti:

1. non è possibile esplicitare l'unnamed graph. Alcuni triple store o middleware (e.g. RDF4J) forniscono una estensione di SPARQL in cui il grafo senza nome può essere indicato tramite una parola speciale
2. Ognuna delle clausole FROM e FROM NAMED annulla le impostazioni di entrambe. Ossia:
 1. l'inserimento di almeno una clausola FROM NAMED implica un grafo di default vuoto (occorre quindi specificare le FROM), e questo è scritto proprio alla fine dell'introduzione del capitolo 13.2 delle SPARQL QUERY specifications
 2. l'inserimento di almeno una clausola FROM implica avere zero named graphs riferibili (devo specificarli tramite FROM NAMED uno ad uno). Verificato personalmente su alcune implementazioni, non so se e dove si trova nelle specs!

Lavorare su più Dataset RDF: La keyword GRAPH (1)

La keyword GRAPH mi permette di fare un riferimento esplicito ai grafi acceduti

```
SELECT ?mail
WHERE GRAPH as:myfoaf.rdf {
  ?x      foaf:mbox <mailto:stellato@info.uniroma2.it> ;
         foaf:knows ?friend .
  ?friend foaf:mbox ?mail
}
```

mi resituirà esclusivamente gli indirizzi mail delle persone presenti nel grafo del mio file foaf personale myfoaf.rdf

Lavorare su più Dataset RDF: La keyword GRAPH (2)

Anche con la keyword GRAPH posso usare delle variabili

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?graph, ?creator
WHERE GRAPH ?graph {
  ?graph dc:creator ?creator .
  ?creator foaf:name "Armando".
}
```

Mi restituirà tutti gli URI *dei grafi* che contengono delle triple indicanti il foaf:name “Armando”, assieme con l’informazione su chi li ha creati

Lavorare su più Dataset RDF: La keyword GRAPH (3)

È possibile combinare più gruppi definiti su grafi diversi:

```
PREFIX data: <http://example.org/foaf/>  
PREFIX foaf: <http://xmlns.com/foaf/0.1/>  
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
SELECT ?mbox ?nick ?ppd  
FROM NAMED <http://example.org/foaf/aliceFoaf>  
FROM NAMED <http://example.org/foaf/bobFoaf>  
WHERE  
{  
  GRAPH data:aliceFoaf  
  {  
    ?alice    foaf:mbox    <mailto:alice@work.example> ;  
             foaf:knows  ?whom .  
    ?whom    foaf:mbox    ?mbox ;  
             rdfs:seeAlso ?ppd .  
    ?ppd     a            foaf:PersonalProfileDocument .  
  }.  
  GRAPH ?ppd  
  {  
    ?w       foaf:mbox    ?mbox ;  
             foaf:nick    ?nick  
  }  
}
```

Questa interrogazione restituisce
tutte le mailbox e i nick delle
persone conosciute da Alice

...ma attenzione!
non utilizza
nessuna delle triple nel file
FOAF di Alice che forniscono il
nick delle persone da lei
conosciute.
Invece, attraverso la proprietà
seeAlso che lega i conoscenti
di Alice al loro
PersonalProfileDocument, va
a prendere i nick direttamente
dai documenti redatti questi
ultimi!

Modificatori di Sequenza per le Soluzioni

- I Query Pattern generano un insieme non ordinato di soluzioni
 - ognuna di esse rappresenta una funzione parziale da variabili verso termini RDF
- Le soluzioni sono gestite come sequenze:
 - Inizialmente non ordinate
 - Vengono successivamente applicati dei modificatori di sequenza per creare una ulteriore sequenza
 - Dall'ultima sequenza generata viene generato il risultato
- Qui di seguito l'elenco dei modificatori di sequenza disponibili in SPARQL
 - Order: definisce un ordine sulle soluzioni generate
 - Projection: effettua una proiezione su un sottoinsieme delle variabili (SELECT)
 - Distinct: garantisce la unicità delle soluzioni nella sequenza
 - Reduced: permette di eliminare alcune soluzioni non uniche
 - Offset: data un ordinamento su una sequenza di soluzioni, definisce da quale soluzione fornire il risultato
 - Limit: restringe il numero di soluzioni restituite

I modificatori sono applicati alla sequenza di soluzioni originaria secondo questo esatto ordine con cui li presentiamo

ORDER BY

Definisce l'ordinamento di una sequenza di soluzioni.

- Alla clausola ORDER BY segue un elenco di variabili della query che impongono il criterio di ordinamento
- Su ognuna della variabili può essere applicato un criterio di ordinamento ascendente tramite il modificatore ASC() (cmq considerato di default) o discendente tramite DESC()

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?name
```

```
WHERE { ?x foaf:name ?name ; :empId ?emp }
```

```
ORDER BY ?name DESC(?emp)
```

Gli URI sono comparati castandoli come simple literals.

Per i termini RDF per i quali non è possibile stabilire un ordine, si applica il seguente ordinamento:

1. Il più basso quando non è possibile assegnare un valore
2. Blank nodes
3. IRIs
4. RDF literals

Soluzioni duplicate

Le clausole DISTINCT e REDUCED influiscono sulla presentazione dei duplicati nelle soluzioni

DISTINCT: rimuove tutte le soluzioni alternative

REDUCED: possono essere rimosse delle soluzioni alternative

- la cardinalità è cmq compresa tra #1 e la cardinalità senza REDUCE o DISTINCT

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

Result:

```
| Name |
-----
| "Alice" |
```

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

Result:

```
| Name |
-----
| "Alice" |
-----
| "Alice" |
```

LIMIT & OFFSET

Pongono delle restrizioni sul numero di soluzioni visualizzate (che non dipendono direttamente dal *contenuto* delle stesse, ma solo della loro *posizione* nella *sequenza*). Entrambe hanno ovviamente senso se applicate a delle soluzioni ordinate tramite ORDER BY

LIMIT pone, per l'appunto, un limite al numero di soluzioni visualizzate. Un limite non può essere negativo (e 0 non ritornerà mai alcuna soluzione).

OFFSET definisce, rispetto all'insieme di soluzioni trattato nella sua fase, la prima che verrà fornita, escludendo le precedenti.

PREFIX foaf:

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

```
SELECT ?name
```

```
WHERE { ?x foaf:name ?name }
```

```
ORDER BY ?name
```

```
LIMIT 20
```

PREFIX foaf:

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

```
SELECT ?name
```

```
WHERE { ?x foaf:name ?name }
```

```
ORDER BY ?name
```

```
LIMIT 5
```

```
OFFSET 10
```

Query Forms

- SPARQL fornisce quattro query form. Tutti partono da delle soluzioni ottenute tramite pattern matching, per dare vista a dei result set o dei grafi RDF a seconda della loro natura.
- I query form sono:
 - SELECT (quello visto finora) Restituisce in un *Result Set* i binding per una proiezione delle variabili contenute nel pattern.
 SELECT * non effettua alcuna proiezione e restituisce i binding per tutte le variabili della query
 - CONSTRUCT Restituisce un *grafo RDF* graph ottenuto per sostituzione di variabili in un insieme di template di triple
 - ASK Restituisce un *risultato booleano* che indica se vi è stata almeno una unificazione con successo
 - DESCRIBE Restituisce un *grafo RDF* che descrive le risorse trovate

CONSTRUCT

CONSTRUCT “costruisce” letteralmente dei nuovi grafi RDF usando il template definito al suo interno. Il template di CONSTRUCT genera un nuovo set di triple basato su di esso per ognuna delle soluzioni ottenute tramite la normale unificazione sul template presente all’interno della WHERE, sostituendo le variabili all’interno del template CONSTRUCT con le istanziazioni delle variabili della WHERE.

Qualora tali istanziazioni producano una tripla contenente un valore non istanziato o un costruito RDF illegale (ad es. un literal come soggetto o predicato di una tripla), allora tale tripla non verrà inclusa nel grafo RDF del risultato.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX stx:    <http://art.uniroma2.it/ontologies/st_example#>
CONSTRUCT    { ?p a foaf:Person }
WHERE        { ?p a stx:Person }
```

Questa query estrae tutti gli individui dalla ontologia caricata che sono definiti come stx:Person, e li converte in istanze della classe Person secondo l’ontologia foaf

DESCRIBE

Come la CONSTRUCT, fornisce per risultato un grafo RDF sulla base delle soluzioni ottenute tramite pattern matching su un grafo esistente.

A differenza di CONSTRUCT però, invece di specificare un template di costruzione del nuovo grafo, DESCRIBE recupererà tutte le informazioni connesse alle risorse unificate sulle variabili specificate nella sua dichiarazione

Tali informazioni sono rappresentate da tutte le triple che contengono una delle risorse estratte¹

QUERY:

```
PREFIX ent: <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg: <http://org.example.com/employees#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl: <http://www.w3.org/2002/07/owl#>
```

RISULTATO:

```
_:a exOrg:employeeId "1234";

foaf:mbox_sha1sum "ABCD1234";
vcard:N
[ vcard:Family "Smith";
  vcard:Given "John" ].

foaf:mbox_sha1sum rdf:type owl:InverseFunctionalProperty .
```

¹ depending on the triple store implementation, these can be all the outgoing triples (triples containing the resource as the subject), or all the triples containing the resource in any position

Nell'esempio qui sopra, è stata estratta la risorsa avente come codice impiegato "1234", aggiungendo al grafo risultato la sua *descrizione*, ovvero tutte le triple che contengono informazioni su di essa

- [SPARQL Query Language for RDF - W3C Recommendation 15 January 2008](#)
- [Introduction to RDF Query with SPARQL Tutorial](#)

Document History

| Data | Contributor | Note |
|------------|------------------|--|
| 05/12/2022 | Armando Stellato | Aggiunta slide "FROM, FROM NAMED e il DEFAULT GRAPH" dopo quella su "FROM NAMED" |