SPARQL
Langage d'interrogation du web sémantique

Anne-Cécile Caron

Master MIAGE parcours IPI-NT

2016-2017

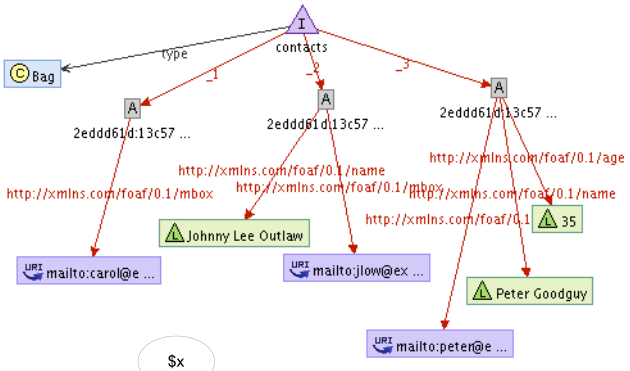
SPARQL

- SPARQL : SPARQL Protocol and RDF Query Langage ;
- Langage de requêtes du W3C pour RDF/RDFS.
  - SPARQL 1.0 - recommendation 15-01-2008
  - SPARQL 1.1 - recommendation 21-03-2013

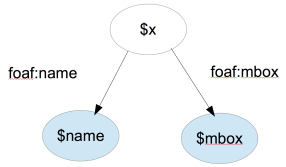
  SPARQL *1.1 est aussi un langage de modification, mais nous ne verrons pas cet aspect.*
- Utilise du *pattern matching* sur la donnée graphe :
  - la requête est un graphe avec variables
  - on recherche les valuations des variables qui soient des sous-graphes de la donnée.
- Syntaxe inspirée de SQL (SELECT, WHERE, GROUP BY).

Exemple de graphe RDF et d'un motif

- Graphe



- Requête



Syntaxe des requêtes usuelles

```
PREFIX ...
PREFIX ...
SELECT ...
WHERE ...
```

- Les préfixes permettent de définir des espaces de noms
- La clause SELECT permet de choisir les variables du résultat, parmi les variables de la clause WHERE
- La clause WHERE contient des triplets qui définissent le motif (pattern) recherché.

Exemple (suite)

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

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Motifs de triplets

- Comme en RDF, on utilise des triplets pour définir un motif pour la clause WHERE (i.e. graphe cible).

  On parle de *motif de triplet* car le triplet peut contenir des variables.
- Les termes utilisés dans les motifs de triplets sont
  - des IRIs (Internationalized Resource Identifiers), généralisation des URIs
  - des littéraux. En général, ce sont des chaînes de caractères entre simples quotes ou guillemets. Pour simplifier,
    - `1` est synonyme de `"1"^^xsd:integer`
    - `1.3` est synonyme de `"1.3"^^xsd:decimal`
    - `true` est synonyme de `"true"^^xsd:boolean`
  - des variables, préfixées par `?` ou par `$`
  - des noeuds blancs qui jouent le rôle de variables non distinguées.
- La syntaxe utilisée pour les requêtes est basée sur la **sérialisation "turtle"**.

IRI et espaces de noms

Comme pour les données RDF, la requête peut contenir une (ou plusieurs) définition de préfixe associé à un espace de noms, et un espace de noms "base" de la requête.
Les requêtes suivantes sont équivalentes :

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <http://example.org/book/book1> dc:title ?title }
```

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
PREFIX  : <http://example.org/book/>
SELECT  $title
WHERE   { :book1  dc:title  $title }
```

```
BASE    <http://example.org/book/>
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  $title
WHERE   { <book1>  dc:title  ?title }
```

SELECT : structure du n-uplet résultat

La clause SELECT permet de définir la forme du résultat. Ce résultat est un ensemble de n-uplets

- `select ?name ?mbox` par exemple permettra de conserver les couples *(name, mbox)* des valuations des variables `?name` et `?mbox` calculées par la requêtes
- `select *` permet de conserver toutes les variables qui apparaissent dans la requête.
- `select distinct ?name ?mbox` enlevera les doublons parmi les couples *(name, mbox)*

SELECT : création de valeurs à partir d'expressions

Depuis SPARQL 1.1, on peut utiliser des expressions dans le SELECT, et renommer les attributs :

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ( CONCAT(?G, " ", ?S) AS ?name )
WHERE  { ?P foaf:givenName ?G ; foaf:surname ?S }
```

Autre exemple :

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

Ici, on définit une variable `fullprice` puis une variable `customerPrice` à partir de `fullprice`.

WHERE : Groupe de motifs de graphe

La clause WHERE est formée d'un groupe de motifs de graphe :
Un groupe peut être

1. un motif de graphe basique :
   = ensemble de motifs de triplets entre { }
2. ou un groupe de groupes de motifs entre { }

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

1 motif basique, composé de deux motifs triplets

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

décomposition du groupe en 2 sous-groupes motifs basiques contenant chacun 1 motif triplet

Ces deux exemples sont équivalents.

Chemins de propriétés

- Ils permettent de chercher des ressources qui sont reliées par un chemin de longueur variable, et pas seulement par 1 propriété (i.e. chemin de longueur 1).
- Un chemin de propriétés est une expression formée de noms de propriétés et d'opérateurs :
  - répétition : `*`, `+`
  - répétition bornée : `{n, m}`
  - option : `?`
  - négation : `!`
  - séquence : `/`
  - alternative : `|`
  - arc inverse : `^`

Exemples

- itération :

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

  -- autre exemple :
{ ?x rdf:type/rdfs:subClassOf* ?type }

-- parcourir les elements d'une collection :
{ ex:mylist rdf:rest*/rdf:first ?element }
```

- négation :

```
{ ?x !(rdf:type|^rdf:type) ?y }
```

Exemples

- séquence :

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

- alternative :

```
{ :book1 dc:title|rdfs:label ?displayString }

-- combin{\'e} avec une it{\'e}ration :
{ ?ancestor (ex:motherOf|ex:fatherOf)+ <#me> }
```

- arc inverse :

```
{
    ?x foaf:knows/^foaf:knows ?y .
    FILTER(?x != ?y)
  }
```

Filtres

On peut ajouter des conditions à la requête, sous la forme de *filtres*. Un filtre s'applique à un motif de graphe.

- Les titres qui commencent par "SPARQL"

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { ?x dc:title ?title
          FILTER regex(?title, "^SPARQL")
        }
```

- Les titres et prix des livres qui coûtent moins de 30.5.

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          FILTER (?price < 30.5)
          ?x dc:title ?title . }
```

Filtres et groupes de motifs

Le filtre permet de restreindre les solutions du groupe où le filtre apparaît. La position du filtre dans le groupe n'a donc pas d'importance. Les trois motifs suivants sont donc équivalents.

```
{  ?x foaf:name ?name .
   ?x foaf:mbox ?mbox .
   FILTER regex(?name, "Smith")
}

{  FILTER regex(?name, "Smith")
   ?x foaf:name ?name .
   ?x foaf:mbox ?mbox .
}

{  ?x foaf:name ?name .
   FILTER regex(?name, "Smith")
   ?x foaf:mbox ?mbox .
}
```

Clause BIND

BIND sert à définir de nouvelles variables et à leur affecter une valeur. Comme on l'a vu pour un select, on peut définir la valeur de cette nouvelle variable à l'aide d'une expression.

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>

SELECT  ?title ?price
WHERE {  ?x ns:price ?p .
   ?x ns:discount ?discount .
   BIND (?p*(1-?discount) AS ?price)
   FILTER(?price < 20)
   ?x dc:title ?title . }
```

- La clause BIND marque la fin du motif de graphe (donc la requête précédente contient 2 motifs de graphe).
- La variable définie par le bind ne peut pas être utilisée avant sa définition.
- Le filtre s'applique au groupe de motifs de graphes.

Valeurs optionnelles

- Un motif de graphe basique permet de rechercher les solutions qui correspondent entièrement à ce motif.
- On peut définir des parties optionnelles dans le motif : clause OPTIONAL
- Lorsque la solution sélectionne des variables présentes dans une partie optionnelle, alors elles n'auront pas nécessairement de valuation.

Exemple

Donnée

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
_:a  rdf:type        foaf:Person .
_:a  foaf:name       "Alice" .
_:a  foaf:mbox       <mailto:alice@example.com> .
_:a  foaf:mbox       <mailto:alice@work.example> .
_:b  rdf:type        foaf:Person .
_:b  foaf:name       "Bob" .
```

Requête

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

name	*mbox*
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Exemple avec filtre

Donnée

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .
:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

Requête

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

title	*price*
"SPARQL Tutorial"	
"The Semantic Web"	*23*

Exemple avec plusieurs motifs optionnels

Donnée

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .
_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
_:c  foaf:name       "Charly" .
_:c  foaf:mbox       <mailto:charly@work.example> .
_:c  foaf:homepage   <http://work.example.org/charly/> .
```

Requête

```
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 }
       }
```

name	*mbox*	*hpage*
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	
"Charly"	<mailto:charly@work.example>	<http://work.example.org/charly/>

Exemple-suite

La requête précédente n'est pas équivalente à :

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

name	mbox	hpage
"Alice"		
"Bob"		
"Charly"	<mailto:charly@work.example>	<http://work.example.org/charly/>

Factorisation "Turtle" des motifs de triplets

- Triplets de même sujet

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

est équivalent à :
?x  foaf:name  ?name .
?x  foaf:mbox  ?mbox .
```

- Triplets de même sujet et même prédicat

```
?x foaf:nick  "Alice" , "Alice_" .

est équivalent  à :
 ?x  foaf:nick  "Alice" .
 ?x  foaf:nick  "Alice_" .
```

- Combinaison

```
 ?x  foaf:name ?name ; foaf:nick  "Alice" , "Alice_" .

est équivalent  à :
 ?x  foaf:name  ?name .
 ?x  foaf:nick  "Alice" .
 ?x  foaf:nick  "Alice_" .
```

Collections RDF

Dans les motifs de triplets, on peut utiliser la notation `(element1 element2 ...)` pour représenter une collection.

Par exemple :

```
(1 ?x 3 4) :p "w" .
```

est équivalent à :

```
_:b0  rdf:first  1 ;
      rdf:rest   _:b1 .
_:b1  rdf:first  ?x ;
      rdf:rest   _:b2 .
_:b2  rdf:first  3 ;
      rdf:rest   _:b3 .
_:b3  rdf:first  4 ;
      rdf:rest   rdf:nil .
_:b0  :p         "w" .
```

Négation

Il y a 2 styles de négation :

1. tester la non-existence d'un motif, avec un FILTER NOT EXISTS (...)
2. enlever les solutions provenant d'un autre motif de graphe avec un MINUS

Non existence d'un motif

Donnée

```
_:a  foaf:givenName  "Alice".
_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

Requête

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
PREFIX  dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name
WHERE {  ?x foaf:givenName  ?name .
    FILTER NOT EXISTS { ?x dc:date ?date } }
```

Résultat

name
"Alice"

Négation : MINUS

Donnée

```
_:a  foaf:givenName  "Alice".
_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

Requête

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

Résultat

o
"Alice"

Remarque : on retire de l'ensemble des résultats de la première expression les résultats *compatibles* de la seconde expression. Un résultat est compatible s'il contient au moins une variable et si toutes ses variables communes avec un résultat de la première expression ont la même valeur dans les 2 résultats.

FILTER NOT EXISTS vs MINUS : partage de variables

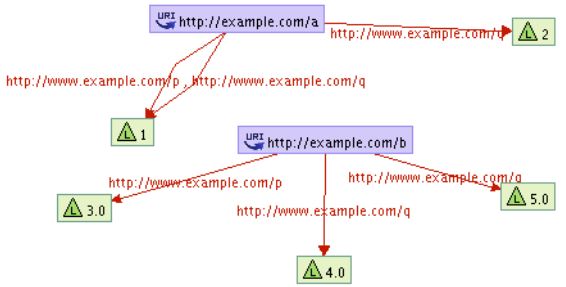
1. La requête suivante ne renvoie jamais aucune solution :

```
SELECT *
{ ?s ?p ?o
  FILTER NOT EXISTS { ?x ?y ?z }
}
```

2. alors que dans cette requête, la seconde expression ne retire aucun résultat à la première, car il n'y a aucune variable en commun :

```
SELECT *
{ ?s ?p ?o
   MINUS  { ?x ?y ?z }
}
```

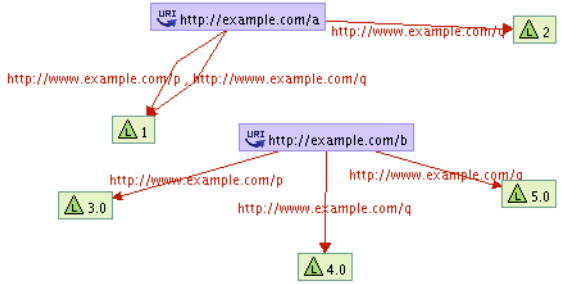
FILTER NOT EXISTS vs MINUS : portée des filtres




```
PREFIX : <http://example.com/>
SELECT * WHERE {
        ?x :p ?n
        FILTER NOT EXISTS {
                ?x :q ?m .
                FILTER(?n = ?m)
        }
}
```

Ici, le FILTER a accès à la valeur de ?n dans ?x :p ?n.

x	n
<http://example.com/b>	3.0




```
PREFIX : <http://example.com/>
SELECT * WHERE {
        ?x :p ?n
        MINUS {
                ?x :q ?m .
                FILTER(?n = ?m)
        }
}
```

Ici, le FILTER a une version locale de ?n. Ce ?n n'a pas de valeur donc le sous-graphe est vide.

x	n
<http://example.com/b>	3.0
<http://example.com/a>	1

UNION

- Permet de satisfaire un motif *ou* un autre.
- On peut différencier les résultats des deux réponses, en utilisant des variables différentes dans les motifs.
- L'opérateur d'union n'enlève pas les doublons

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

name	mbox	hpage
"Bob"	<mailto:bob@work.example>	
"Charly"	<mailto:charly@work.example>	
"Alice"		<http://work.example.org/alice/>
"Charly"		<http://work.example.org/charly/>

Agrégats

- Par défaut, un ensemble de solutions consiste en 1 groupe de solutions
- Le GROUP BY permet de partitionner la solution en plusieurs groupes, sur lesquels on peut appliquer une fonction d'agrégat.
- Les fonctions de SPARQL 1.1 sont : COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT, et SAMPLE.
- On retrouve les mêmes contraintes qu'en SQL, concernant les variables que l'on peut écrire dans le SELECT par rapport au GROUP BY.

Exemple

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

@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 .
```

org	totalPrice
<http://books.example/org1>	21

Sous-requête

On considère la requête :

```
PREFIX : <http://people.example/>
PREFIX : <http://people.example/>
SELECT ?y ?minName
WHERE {
:alice :knows ?y .
 {
  SELECT ?y
    (MIN(?name) AS ?minName)
 WHERE {
   ?y :name ?name .
  } GROUP BY ?y
 }
}
```

et la donnée :

```
@prefix : <http://people.example/> .

:alice :name "Alice", "Alice Foo", "A. Foo".
:alice :knows :bob, :carol .
:bob :name "Bob", "Bob Bar", "B. Bar" .
:carol :name "Carol", "Carol Baz", "C. Baz".
```

Sous-requête

L'évaluation de la requête imbriquée donne :

y	minName
:alice	"A. Foo"
:bob	"B. Bar"
:carol	"C. Baz"

La requête externe (motif `:alice :knows ?y .`) a pour résultat :

y
:bob
:carol

Donc le résultat de la requête entière est (par jointure) :

y	minName
:bob	"B. Bar"
:carol	"C. Baz"

Modifier la séquence des solutions

- les motifs génèrent une collection non-ordonnée de solutions, chaque solution étant une valuation des variables présentes dans les motifs.
- Ces solutions sont ensuite traitées comme une séquence, sur laquelle on peut appliquer un opérateur (*sequence modifier*).
- Il existe 6 opérateurs permettant de modifier la séquence des solutions :
  1. `order by` : trier les solutions. Il faut qu'une relation d'ordre existe pour ces solutions.
  2. projection pour choisir les variables. Comme en SQL, liste des variables après le SELECT, et $*$ pour toutes les variables.
  3. `distinct` : éliminer les doublons parmi les solutions

     ```
     PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
     SELECT DISTINCT ?name WHERE { ?x foaf:name ?name }
     ```
  4. `reduced` : autoriser l'élimination des doublons par le moteur de requête (pas d'obligation)
  5. `offset` : indiquer à partir de quelle position dans la séquence on démarre
  6. `limit` : borner la taille de la séquence des solutions

ORDER BY - exemples

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

```
PREFIX     :    <http://example.org/ns#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY DESC(?emp)
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ASC(?name) DESC(?emp)
```

Offset et Limit

- Offset n signifie qu'on "passe" les *n* premières solutions. Un offset de 0 n'a pas d'effet.
- Limit n signifie qu'on donne au maximum *n* solutions.

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

SELECT  ?name
WHERE   { ?x foaf:name ?name }
ORDER BY ?name
LIMIT   5
OFFSET  10
```

Cette requête donne au maximum 5 solutions, à partir de la 11ème dans la séquence des solutions.

Plusieurs formes de requêtes

- Les exemples jusqu'à présent :
  1. prologue : définition optionnelle d'une base, puis définitions de préfixes.
  2. puis la partie requête proprement dite avec un Select.
- Le mot clé Select signifie que les solutions construites sont des valuations des variables par des termes RDF
- Il existe d'autres formes de requêtes :
  - Ask : retourne un booléen indiquant l'existence d'une solution qui satisfait le motif
  - Construct : pour construire un graphe RDF solution, à partir des valuations des variables
  - Describe : retourne un graphe RDF *décrivant* les ressources trouvées. Le résultat dépend du service qui publie la donnée.

Ask

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

true

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" ;
         foaf:mbox  <mailto:alice@work.example> }
```

false

Construct

Donnée

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a    foaf:givenname   "Alice" .
_:a    foaf:family_name "Hacker" .
_:b    foaf:firstname   "Bob" .
_:b    foaf:surname     "Hacker" .
```

Requête

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?x  vcard:N _:v .
            _:v vcard:givenName ?gname .
            _:v vcard:familyName ?fname }
WHERE { { ?x foaf:firstname ?gname } UNION  { ?x foaf:givenname   ?gname } .
        { ?x foaf:surname   ?fname } UNION  { ?x foaf:family_name ?fname } .  }
```