Master 2 Info & Maths-Info

Bases de données spécialisées

Cristina Sirangelo
IRIF, Université Paris Cité
cristina@irif.fr

Interrogation de données RDF

Credits (materiel, transparents et exemples empruntés)
- Transparents du cours LODAS , Bernd Amman, Univ. Pierre et Marie Curie
- MOOC FUN INRIA Web Sémantique - Corby, Gandon, Faron
- Cours RDF and SPARQL, Universidad de Chile, M. Arenas et al
- Cours RDF(S) - SPARQL - Stockage, E.Coquery

Langages de requêtes RDF

- Remarque : SQL pourrait exprimer des requêtes simples sur les triplets (les triplets sont des relations binaires!)
- Cependant sur les données de type graphe comme RDF on a souvent besoin d'exprimer des requêtes de "navigation"
- Besoin d'un langage avec un pouvoir d'expression au delà du SQL de base
- Même si SQL peut exprimer des requêtes plus complexes (i.e. recursion), il n'est pas optimisé pour ce genre de requêtes
- Cela justifie des langages de requêtes ad hoc pour RDF

Langages de requêtes RDF

Trois familles de langages de requêtes RDF :

- langages ensemblistes (objet/relationnel) : RQL (chemins / objets), Squish (triplets), RDQL (triplets)
- langages "matching de graphes" : DQL, QEL, N3, **SPARQL**
- langages XML étendus pour RDF : RQuery (XQuery+fonctions), RDF Query

SPARQL

- SPARQL (prononcé “sparkle”) : le langage d’interrogation standard pour les données RDF
- Acronyme récursif qui veut dire : *SPARQL Protocol And RDF Query Language*.
- Une recommandation W3C depuis 2008.
- Dernière version du standard : *SPARQL 1.1 Query Language - W3C REC 21 Mar. 2013*
  - *Definition de SPARQL 1.2* ***en cours*** *(dernier working draft 25 Sept 2025)*
- Fondé sur le modèle de graphe sous-jacent RDF
- Deux composantes :
  - Recommandation *SPARQL 1.1* ***Query Language*** *- W3C REC 21 Mar. 2013*
  - Recommandation *SPARQL 1.1 Update - W3C REC 21 Mar. 2013*
- Documentation : https://www.w3.org/TR/sparql11-query/

Pas d'inférence en SPARQL!

- Une requête SPARQL s'évalue sur une base de triplets RDF/S
- Aucun nouveau triplet n'est inféré par défaut, même si les triplets interrogés contiennent des contraintes RDFS
- Une requête SPARQL s'évalue uniquement sur les triplets explicitement représentés
- Cela sera le role du système de raisonnement de faire les inférences et ensuite évaluer SPARQL en utilisant également les faits inférés
  - En général pas automatique (coûteux) dans les systèmes existants
    - Il faut lancer le système de raisonnement explicitement si on veut tenir compte des inférences
  - Nous en parlerons plus loin

- **Ce cours** :
  Syntaxe et sémantique SPARQL sur une base explicite de triplets RDF/S

Forme d'une requête de base SPARQL

SELECT *valeurs de retour* — Tête de la requête

FROM *graphe RDF* — Source (optionnelle si source par défaut)

WHERE {*motif de graphe à matcher*} — Corps de la requête

- Exemple :

```
SELECT ?x

WHERE

{ ?x rdf:type ex:Person .
  ?x ex:name ?name . }
```

- Retourne les ressources (noeuds du graphe RDF/S) qui sont de type Personne et ont un nom

Variables

```
SELECT ?x

WHERE

{ ?x rdf:type ex:Person .
  ?x ex:name ?name . }
```

- Une requête SPARQL utilise des variables pour matcher les noeuds ou arrêtes du graphe RDF (URI, littéraux ou blank nodes)
  - Le nom des variables doit commencer par **‘?’**
- SELECT est suivi d’une liste de variables, pas nécessairement présentes dans le corps

Motifs de triplets (triple pattern)

```
?x ex:name ?name .
```

- Le corps de la requête contient des motifs de triplets (syntaxe Turtle)
- **Motif de triplet** (triple pattern) : triplet RDF (à la syntaxe Turtle) où les URIs (du sujet, prédicat ou objet) peuvent être remplacés par des variables
  - Attention : les ressources anonymes peuvent être présentes dans un motif de triplet, mais dans ce cas il sont considérés comme une nouvelle variable (variable sans nom)

Motifs basique de graphe

- **Motif basique de graphe** (Basic graph pattern) : ensemble de motifs de triplets

```
{?x rdf:type ex:Person .
 ?x ex:name ?name .}
```

- Le corps d'une requête peut être un motif basique de graphe : intuitivement cela représente le motif à trouver dans le graphe RDF interrogé
  - **Conjonction** implicite entre les triplets : tous doivent être presents

Mêmes abréviations qu'en Turtle

- Triplets ayant un sujet commun :

```
SELECT ?name ?fname
WHERE {
    ?x a ex:Person;
         ex:name ?name ;
         ex:firstname ?fname ;
         ex:author ?y .
}
```

Mêmes abréviations qu'en Turtle

- Triplets ayant un sujet et prédicat commun :

```
SELECT ?name ?fname
WHERE {
    ?x a ex:Person;
         ex:name ?name ;
         ex:firstname ?fname ;
         ex:author “Semantic Web”, “Ontologies" .
}
```

- Ressource anonyme

```
SELECT ?x
WHERE {
    ?x foaf:knows [ ex:firstname "Fabien" ]
       ex:marriedTo [ ]
}
```

Ressources anonymes dans les triple patterns

- Les blank nodes dans un motif de triplets représentent une **variable anonyme** et non pas une donnée anonyme :

```
SELECT ?x
WHERE {
    ?x foaf:knows [ ex:firstname "Fabien" ]
         ex:marriedTo [] }
```

- Chaque ressource anonyme dans le motif est assimilée à une nouvelle variable (sans nom) distincte de toutes les autres. Donc le motif ci-dessus est equivalent à

```
SELECT ?x
WHERE {
    ?x foaf:knows ?y ;
       ex:marriedTo ?z  .
    ?y ex:firstname "Fabien" }
```

Ressources anonymes dans les triple patterns

- La syntaxe Turtle explicite pour les ressources anonymes est également possible dans le corps d'une requête :

```
SELECT ?name
WHERE { _:f foaf:name ?name;
              foaf:mbox _:c . }
```

- Une ressource anonyme dans un triple pattern se comporte comme toutes les autres variables : peut matcher n'importe quelle ressource (anonyme ou pas)
- Toutefois au contraire des variables nommées, les ressources anonymes ne peuvent pas apparaître dans la partie SELECT

Prefixes pour les namespaces

- Déclaration de prefixes. Syntaxe légèrement différente qu'en Turtle

```
PREFIX uparis: <http://www.u-paris.fr#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?student
WHERE {
 ?student uparis:inscritA ?x .
 ?x foaf:homepage <http://u-paris.fr/home> .
}
```

(pur rappel en Turtle :

```
    @prefix uparis: <http://www.u-paris.fr#> .
    @prefix foaf: <http://xmlns.com/foaf/0.1/> .
)
```

Base

- Declarer un base pour les URI relatifs. Syntaxe légèrement différente qu'en Turtle

```
BASE <http://www.u-paris.fr#>
SELECT ?x
WHERE {
   ?student <#inscritA> ?x .
   ?x foaf:homepage <http://u-paris.fr/home> .
}
```

- Pour rappel en Turtle :

```
@base <http://www.u-paris.fr#> .
```

Spécifier la langue et le type des littéraux

Pour obliger à respecter un attribut : `@fr , ^^xsd:integer`

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x WHERE {
 ?x foaf:name "Fabien"@fr ;
    foaf:age "21"^^xsd:integer .
}
```

BGP

- Le fragment de SPARQL vu jusqu'à maintenant est appelé Basic Graph Patterns (BGP)
- Une requête SPARQL est un BGP quand le corps est un motif basique de graphe

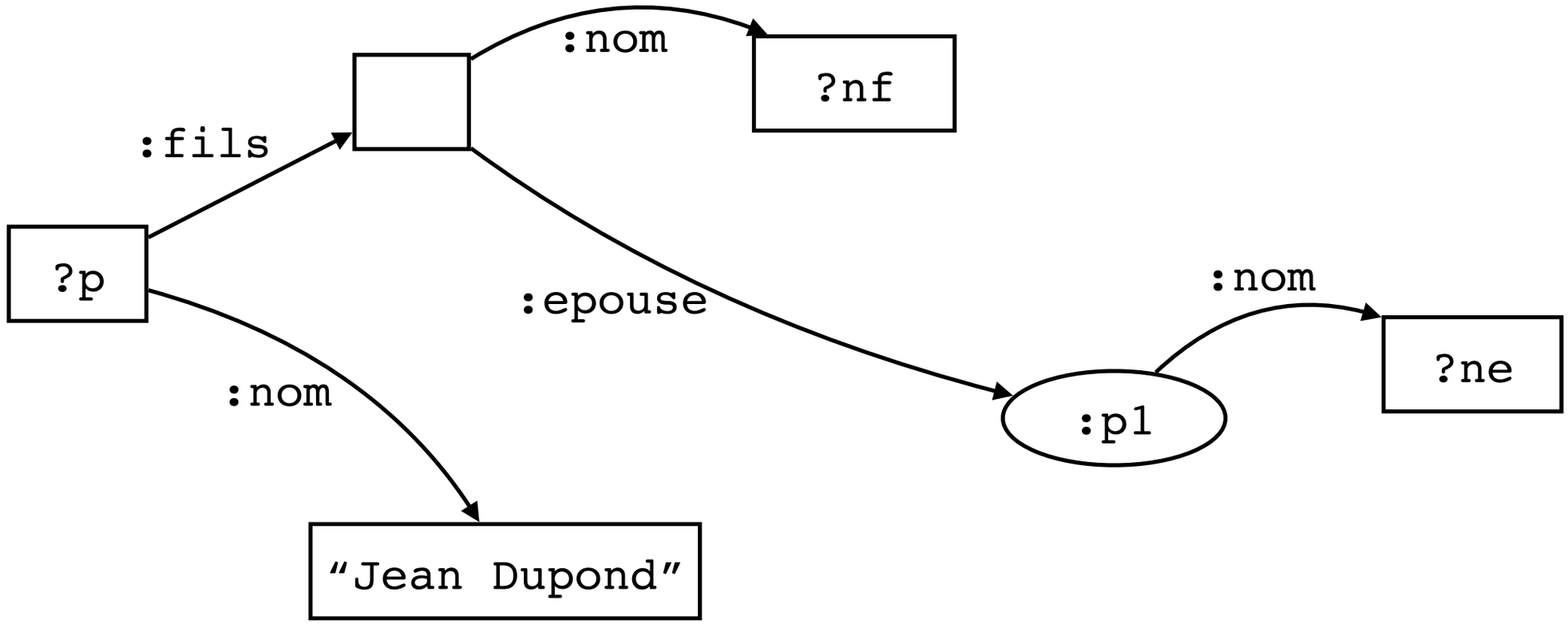
```
SELECT ?x

WHERE

{ ?x rdf:type ex:Person .
  ?x ex:name ?name . }
```

N.B. Puisqu'un motif de graphe mentionnant des blank nodes est toujours equivalent à un motif sans blank nodes, on supposera, pour définir la sémantique, que le corps de la requête ne contient pas de blank nodes

BGP comme graphe

- Un BGP peut être vu comme un graphe (tout comme n'importe quel ensemble de triplets)

```
select ?nf ?ne
from <ex.ttl>
where {
  ?p :nom “Jean Dupond" ;
     :fils [:nom ?nf; :epouse  :p1].
     :p1 :nom ?ne
}
```



Sémantique des BGP

- Soit `{ P(?x1, ?x2, …) }`
  un motif basique de graphe mentionnant les variables `?x1, ?x2, …`

Evalué sur un graphe RDF G, `{ P(?x1, ?x2, …) }` **retourne**

toutes les assignations de variables **`?x1 -> v1, ?x2 -> v2, …`**

telles que `P(?x1->v1, ?x2->v2, …)` ⊆ G

- Les valeurs `v1, v2, …` des variables peuvent être aussi bien des URI, que des littéraux, que des ressources anonymes de G

Sémantique des BGP

Exemple

```
{?x   ex:name ?name ;
      ex:author ? z .
}
```

Evalué sur le graphe :

```
:jd   ex:name “John Doe”;
      ex:author :SemWeb .
:ab   ex:name “Alain Black”;
      ex:author :Onto .
:wd ex:author :Cendrillon
```

Retourne

?x	?name	?z
:jd	“John Doe”	:Sem_Web
:ab	“Alain Black”	:Onto

Remarque : assignations vues comme n-uplets

Sémantique des BGP

La requête

```
SELECT xi, xj,..
{ P(?x1, ?x2, …) }
```

retourne à son tour les assignations retournées par le motif `{P}`, restreintes aux variables `xi, xj,..`

Exemple

```
SELECT ?x, ?name
{?x  ex:name ?name ;
     ex:author ? z .}
```

Evalué sur le graphe :

```
:jd  ex:name “John Doe”;
     ex:author :SemWeb .
:ab  ex:name “Alain Black”;
     ex:author :Onto .
:wd ex:author :Cendrillon
```

Retourne

?x	?name
:jd	“John Doe”
:ab	“Alain Black”

Sémantique des BGP

- SELECT peut mentionner des variables non présentes dans le motif de graph

```
SELECT ?x, ?name, ?w
{?x  ex:name ?name ;
     ex:author ? z .}
```

- Dans ce cas elles n'auront pas de valeur dans les assignations retournées par la requête

?x	?name	?w
:jd	"John Doe"	
:ab	"Alain Black"	

- De telles variables d'une assignation sont appelées **unbound**

Résultats distincts

- Pour retourner un seul exemplaire des chaque réponse

```
SELECT DISTINCT ?name
WHERE { ?person foaf:name ?name . }
```

Au delà des BGP : motifs de graphe

- SPARQL possède plusieurs caractéristiques qui vont au delà des BGP
- Le corps de la requête peut être un motif de graphe plus général que les motifs basiques
- Les motifs de graphes autorisés peuvent être définis récursivement comme suit :

Motif de graphe

- Tout motif de graphe basique est est un motif de graphe
- Si P est un motif de graphe, alors {P} est un motif de graphe (dit groupé)
- Si P1, P2 sont des motifs de graphe, alors
  - {P1} UNION {P2}, P1 P2 (ou P1 . P2), P1 OPTIONAL {P2}, P1 MINUS {P2} sont des motifs de graphe
- Si P est un motif de graphe et R est une condition booléenne alors P FILTER (R) est un motif de graphe

Au delà des BGP : motifs de graphe

- N'importe quel motif de graphe est autorisé :

Requête SPARQL :

Si P est un motif de graphe et W un ensemble fini de variables (pas nécessairement présentes en P), alors

SELECT W [FROM <source>] WHERE {P}

est une requête SPARQL

Sémantique des requêtes SPARQL

- Un motif de graphe P retourne toujours un ensemble d'assignations
- (Comme pour les BGP)

> SELECT W WHERE {P}
>
> retourne les assignations retournée pas P restreintes aux variables W.

(Si t est une assignation retournée par P, et W contient une variable w non présente en t, alors w sera *unbound* dans la restriction de t à W)

- Reste à définir la sémantique de chaque type de motif de graphe P

UNION

UNION

{P1} U {P2} retourne l'union des assignations retournées par P1 et de celles retournées par P2

- Exemple. **Pattern** :

```
{?x  ex:genre "roman";
     ex:author ?z .} UNION {?x ex:editeur :gallimard}
```

- **Données**

```
:la_romana   ex:genre "roman";
             ex:author :moravia ;
             ex:editeur :gallimard .
:miserables  ex:genre "roman";
             ex:author :hugo .
:la_noia   ex:genre "roman";
           ex:editeur :gallimard .
```

- **Resultat**

?x	?z
:la_romana	:moravia
:miserables	:hugo

?x
:la_romana
:la_noia

UNION

- Remarque : les assignations retournées par UNION ne sont pas toutes définies sur les mêmes variables
- SELECT retourne comme d'habitude *unbound* pour les variables absentes d'une assignation

```
SELECT ?x, ?z WHERE
{?x  ex:genre "roman";
     ex:author ?z .}
UNION
{?x ex:editeur :gallimard}
```

- Resultat

?x	?z
:la_romana	:moravia
:miserables	:hugo
:la_romana	
:la_noia	

Conjonction

P1 P2 (ou P1 . P2) retourne chaque assignation t1 retournée par P1, complétée par chaque assignation t2 retournée par P2 qui est compatible avec t1

- (t1 et t2 sont **compatibles** si elles sont d'accord sur les variables en commun)

- Exemple. **Pattern** :

```
SELECT ?x, ?z, ?y WHERE
?x  ex:genre “roman” ;
      ex:author ?z .
{?x ex:editeur ?y.
 ?y ex:name “Gallimard”.}
```

- **Données**

```
:la_romana  ex:genre “roman”;
            ex:author :moravia ;
            ex:editeur :gallimard .
:miserables ex:genre “roman”;
            ex:author :hugo .
:la_noia    ex:genre “roman”;
            ex:editeur :gallimard .
:gallimard ex:name “Gallimard”
```

- **Resultat**

?x	?z	?y
:la_romana	:moravia	:gallimard

Conjonction - exemple expliqué

Résultat expliqué en passant par les résultats des deux operands :

- P1 = `?x  ex:genre “roman” ; ex:author ?z .`
  retourne

	?x	?z
t1	:la romana	:moravia
t3	:miserables	:hugo

- P2 = `?x ex:editeur ?y. ?y ex:name “Gallimard”.`

  retourne

	?x	?y
t2	:la_romana	:gallimard
t4	:la_noia	:gallimard

- t1 compatible avec t2 mais pas avec t4; t3 pas compatible.
- Remarque : il s’agit de la **jointure naturelle** des deux résultats!

OPTIONAL

- P1 OPTIONAL {P2}
- Retourne :
  - Les résultats de P1 complétés avec les results compatibles de P2 (comme P1 P2)
  - plus les résultats de P1 qui ne peuvent pas être complétés par les résultats de P2

- Remarque : il s'agit du "**Left outer join**" des résultats de P1 et P2

OPTIONAL : Example

Exemple:

- Base RDF :

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
[ foaf:name    "Alice"   ;
  foaf:knows   [foaf:name    "Bob" ]  ;
  foaf:knows   [foaf:name    "Clare" ;
                foaf:nickname "CT"   ] ]
```

OPTIONAL : définir des mappings « optionnels » :

Requête ex4sansopt

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
FROM <ex4.ttl>
WHERE  { ?x foaf:knows ?y ;
            foaf:name ?nameX .
         ?y foaf:name ?nameY ;
            foaf:nickname ?nickY }
```

Résultat de ex4sansopt

```
---------------------------
| nameX   | nameY   | nickY |
===========================
| "Alice" | "Clare" | "CT"  |
---------------------------
```

Requête ex4

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
FROM <ex4.ttl>
WHERE  { ?x foaf:knows ?y ;
            foaf:name ?nameX .
         ?y foaf:name ?nameY .
OPTIONAL { ?y foaf:nickname ?nickY } }
```

Résultat de ex4

```
---------------------------
| nameX   | nameY   | nickY |
===========================
| "Alice" | "Clare" | "CT"  |
| "Alice" | "Bob"   |       |
---------------------------
```

MINUS

P1 MINUS {P2} retourne les résultats t1 de P1 pour lesquels il n'existe aucun résultat de P2 qui **étend** t1

t1 et t2 **s'tendent** s'ils sont **compatibles et** ont **au moins une variable en commun**

- Seulement présent depuis SPARQL 1.1 (2013)

MINUS : Exemple

ex14.ttl :

```
:book1 dc:title "SPARQL Tutorial" ; ns:price 42 ;  ns:editor :jena .
:book2 dc:title "The Semantic Web" ; ns:price 23 ; ns:editor :w3c .
:book3 dc:title "RDF Framework" ; ns:price 15 ; ns:editor :w3c .
:book4 dc:title "SPARQL 1.1" ; ns:editor :jena .
```

```
?y ns:editor ?editor
MINUS { ?y ns:price ?price . }
```

```
P1 = ?y ns:editor ?editor
P2 = ?y ns:price ?price
```

Résultat de P1

?y	?editor
:book1	:jena
:book2	:w3c
:book3	:w3c
:book4	:jena

Résultat de P2

?y	?price
:book1	42
:book2	23
:book3	15

Résultat de P1 MINUS {P2}

?y	?editor
:book4	:jena

MINUS : Exemple suite

- Les éditeurs dont au moins un livre n'a pas de prix

```
SELECT ?editor FROM <ex14. ttl >
WHERE {
?y ns:editor ?editor MINUS { ?y ns:price ?price . }
```

?editor
:jena

MINUS : Attention à la notion d'extension

ex14.ttl :

```
:book1 dc:title "SPARQL Tutorial" ; ns:price 42 ;  ns:editor :jena .
:book2 dc:title "The Semantic Web" ; ns:price 23 ; ns:editor :w3c .
:book3 dc:title "RDF Framework" ; ns:price 15 ; ns:editor :w3c .
:book4 dc:title "SPARQL 1.1" ; ns:editor :jena .
```

```
?y ns:editor ?editor
MINUS { ?z ns:price ?price . }
```

```
P1 = ?y ns:editor ?editor
P2 = ?z ns:price ?price
```

Résultat de P1

?y	?editor
:book1	:jena
:book2	:w3c
:book3	:w3c
:book4	:jena

Résultat de P2

?z	?price
:book1	42
:book2	23
:book3	15

Résultat de P1 MINUS {P2}

?y	?editor
:book1	:jena
:book2	:w3c
:book3	:w3c
:book4	:jena

Aucun résultat de P1 ne peut être étendu par un résultat de P2 (bien que tous les résultats soient compatibles entre eux!) - pas de variable en commun !!

FILTER

P FILTER(R) retourne tous les résultats de P qui satisfont R

ex5.ttl

```
ns:book1 dc:title "SPARQL" ; ns:price 42 .
ns:book2 dc:title "Sem Web"; ns:price 23 .
ns:book3 dc:title "RDF Framework" .
```

```
SELECT ?title ?price FROM <ex5. ttl >
WHERE
{ ?x ns:price ?price ;
     dc: title ?title
   FILTER ( ?price < 30) .
}
```

P= `?x ns:price ?price ;`
`dc: title ?title`

R= `?price < 30`

Resultat de P

?x	?title	?price
ns:book1	“SPARQL”	42
ns:book2	"Sem Web"	23

Resultat de P FILTER (R)

?x	?title	?price
ns:book2	"Sem Web"	23

Résultat de la requête

?title	?price
"Sem Web"	23

FILTER - condition de test

- P FILTER(R), R peut être :
  - Comparaison de variables entre eux ou avec des termes RDF
    - `?x` *op* `?y` , `?x` *op* `c`
      avec *op* : `<, >, =, <=, >=, !=`
  - Le résultat d'une fonction booléenne (fonctions XPath/XQuery autorisées). Exemples :
    - Test sur la nature des valeurs des variables :
      - `isURI(?x), isBlank(?x), isLiteral(?x), bound(?x),…`
    - Et d'autre tests boolean : `regex(?x, expr), contains(?s,?sub), in,…`
    - Tester l'existence d'autres motifs (possiblement corrélés au motif P)
    - `NOT EXISTS { pattern }`
    - `EXISTS { pattern }`
  - Combinaison booléenne de tests `&&, ||, !, ()`

FILTER : test de EXISTS / NOT EXISTS

- `EXISTS { pattern }`
  - vrai si le motif `pattern` retourne au moins un résultat, faux autrement
- `NOT EXISTS { pattern }`
  - vrai si le motif `pattern` ne retourne aucun résultat, faux autrement

Données

```
:alice   rdf:type    foaf:Person .
:alice   foaf:name   "Alice" .
:bob     rdf:type    foaf:Person .
```

Requête :

```
SELECT ?person
WHERE {
     ?person rdf:type  foaf:Person .
     FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

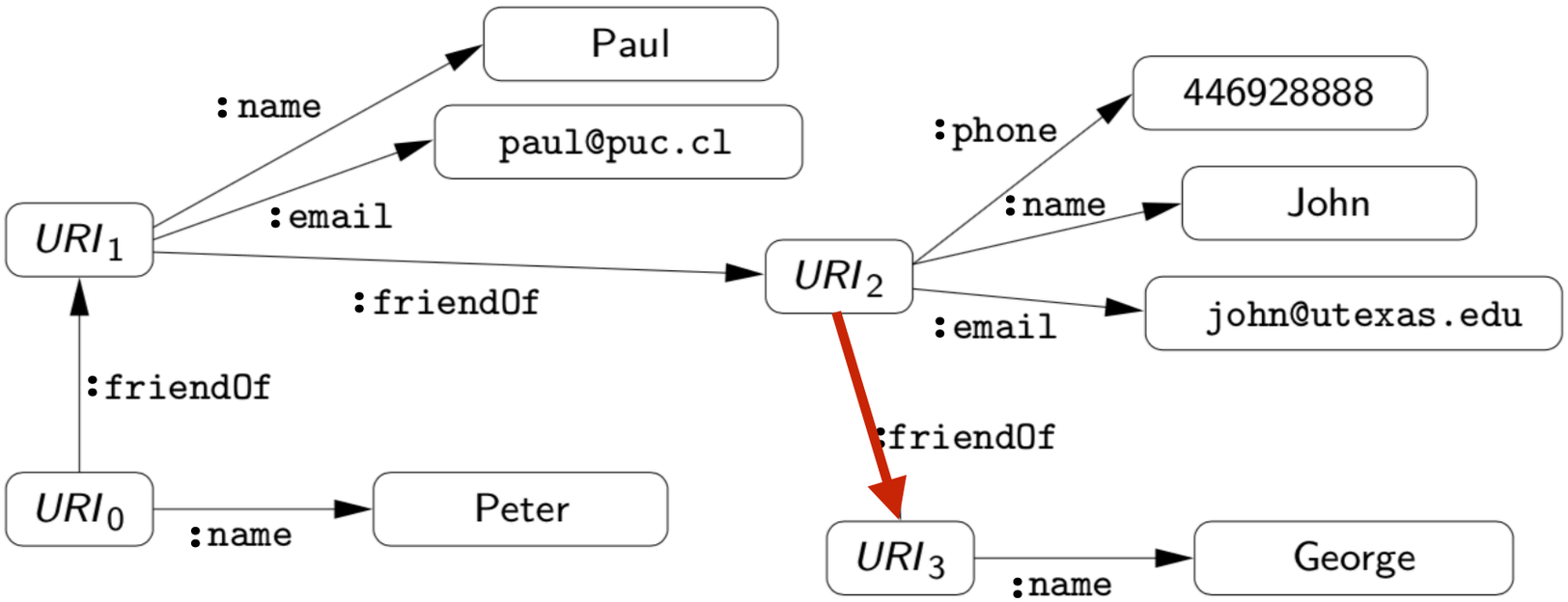
Resultat

?person
:bob

- Remarquer la **correlation** :
  FILTER peut faire reference à l'assignation courante qu'il faut filtrer

Limites des motifs de graphes

- Les motifs de graphes vus jusqu'à maintenant permettent d'interroger de façon très limitée l'existence de chemins dans un graphe RDF
- Le chemin doit être complètement spécifié dans la requête
- Exemple



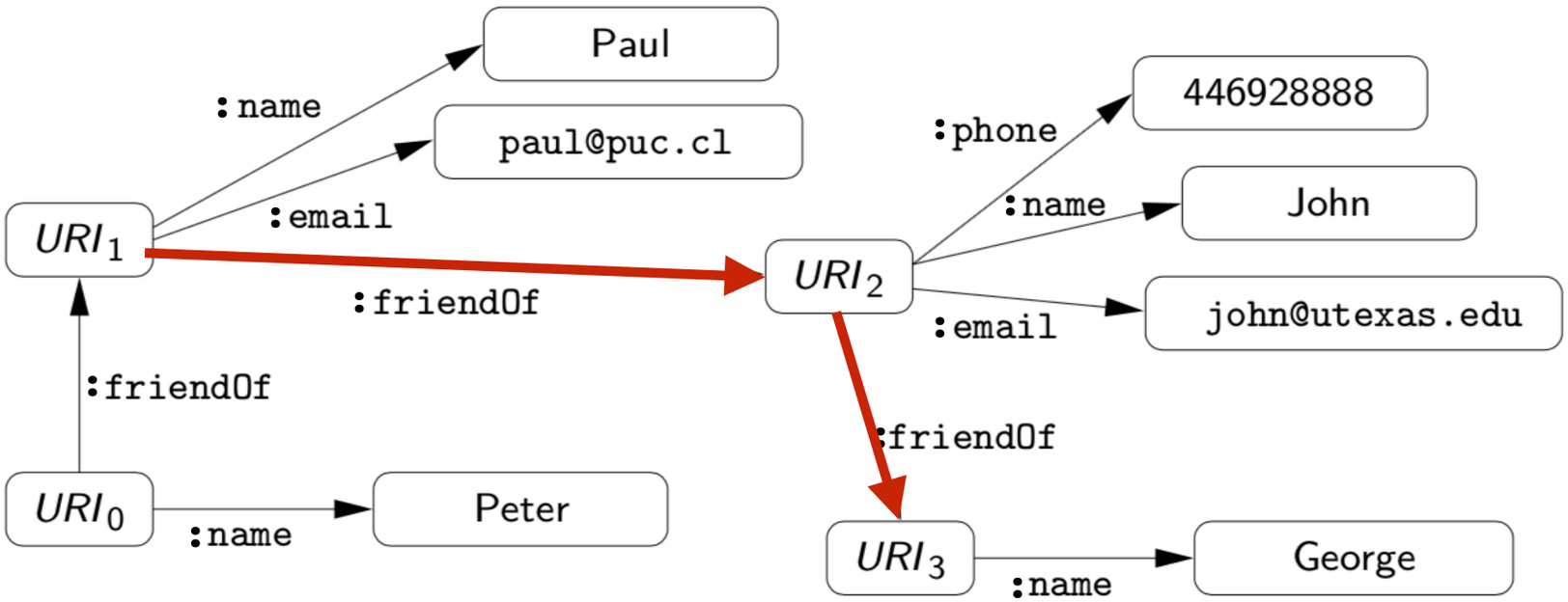

```
SELECT ?x WHERE{
  ?x :friendOf ?y .
  ?y  :name “George” }
```

Résultat de la requête

?x
URI2

Limites des motifs de graphes

- Les motifs de graphes vus jusqu'à maintenant permettent d'interroger de façon très limitée l'existence de chemins dans un graphe RDF
- Le chemin doit être complètement spécifié dans la requête
- Exemple



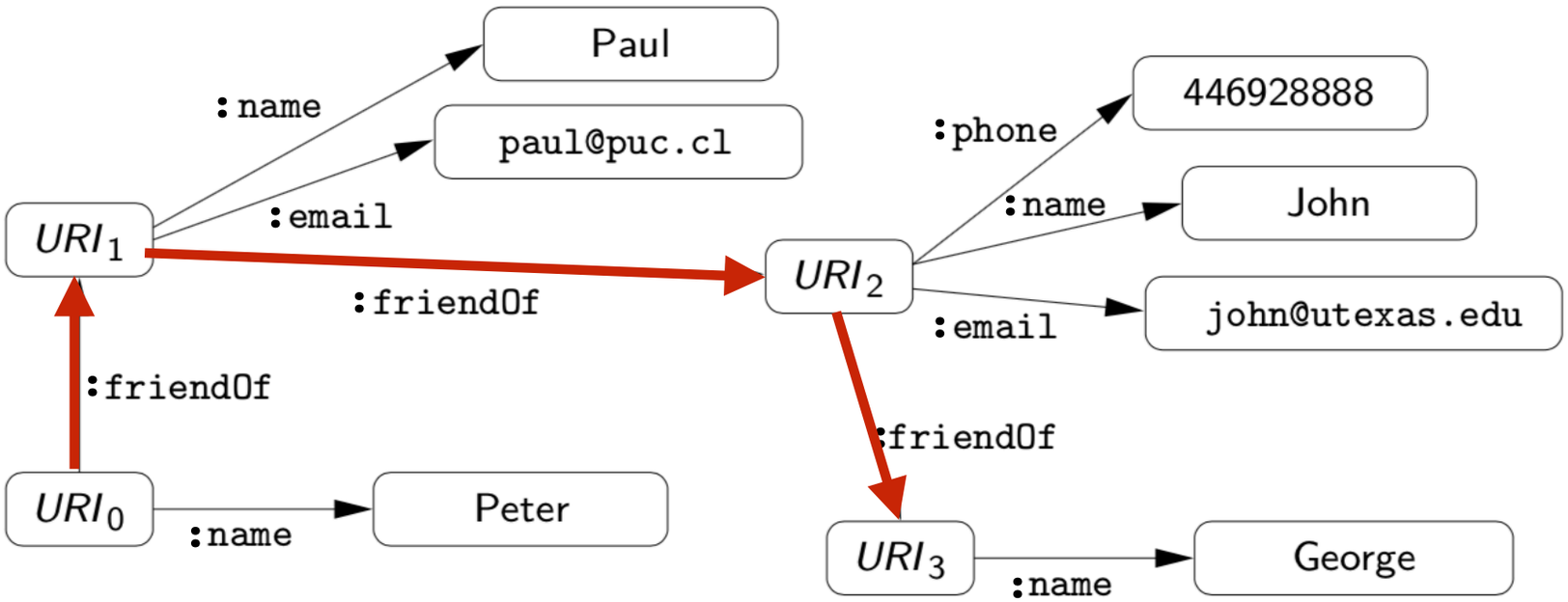

```
SELECT ?x WHERE{
  ?x :friendOf ?y .
  ?y :friendOf ?z .
  ?z  :name "George" }
```

Résultat de la requête

?x
URI1

Limites des motifs de graphes

- Les motifs de graphes vus jusqu'à maintenant permettent d'interroger de façon très limitée l'existence de chemins dans un graphe RDF
- Le chemin doit être complètement spécifié dans la requête
- Exemple




```
SELECT ?x WHERE{
  ?x :friendOf ?y .
  ?y :friendOf ?z .
  ?z :friendOf ?w .
  ?w  :name “George” }
```

Résultat de la requête

?x
URI0

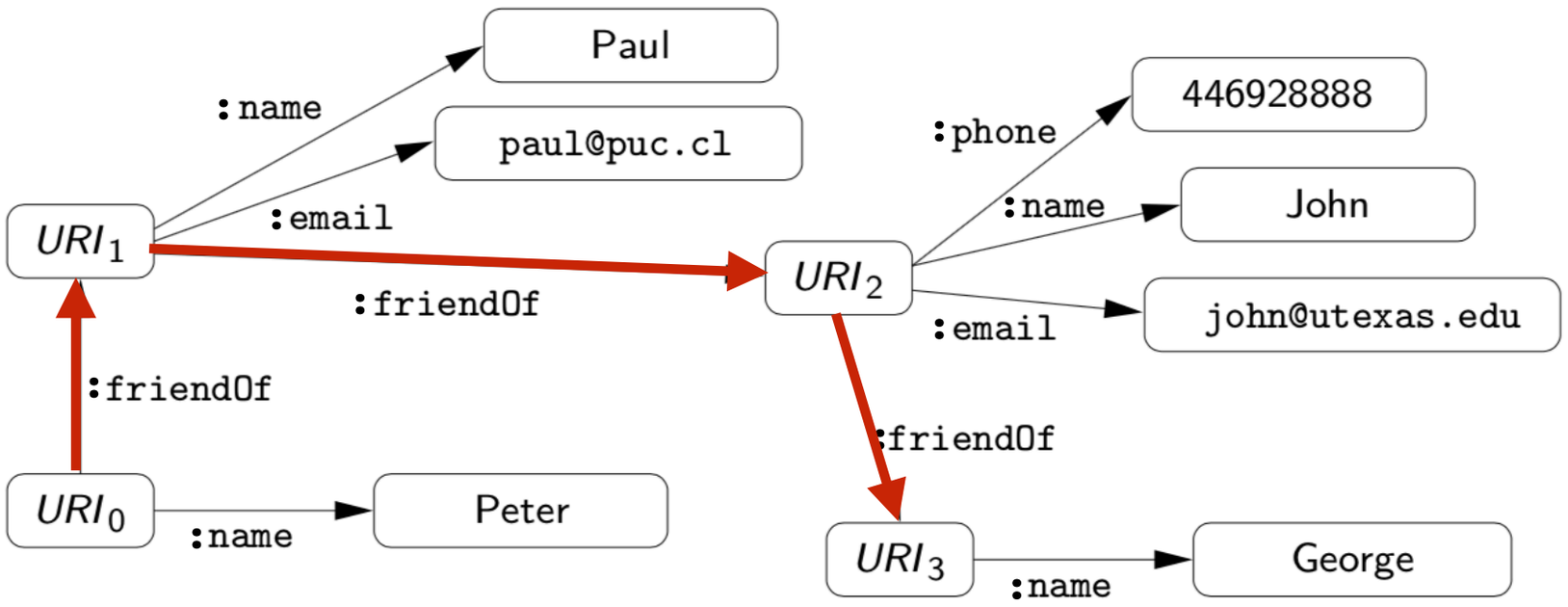
Limites des motifs de graphes

- Et si on voulait une seule requête pour calculer les amis des amis des amis des amis…sans limite sur la longueur du chemin?
- Pas possible dans le fragment de SPARQL vu jusqu'à maintenant
  - La taille du chemin est limitée par la taille de la requête
  - On ne peut pas interroger pour l'existence d'un chemin de n'importe quelle taille (taille qui dépende des données)
  - Limite similaire à SQL '92
  - En SQL pour surmonter cette limite la recursion (WITH RECURSIVE) a été introduite (SQL3)
    - Syntaxe complexe, difficilement optimisable par les SGBD relationnel

Expressions de chemin (property paths)

- En SPARQL : property paths
  - Depuis SPARQL 1.1 (2013)
  - Introduisent une forme de recursion
  - Étendent les motifs de graphes pour tester l'existence de chemins
  - Utilisent les expressions régulières pour spécifier le motif du chemin qu'on cherche
  - Extension beaucoup plus naturelle qu'en SQL
    - En SPARQL on parle déjà de motifs de graphes à la base!
    - Implémentés efficacement

Expressions de chemin (property paths)




```
SELECT ?x WHERE{
  ?x :friendOf+ ?y .
  ?y  :name “George” }
```

Résultat de la requête

?x
URI0
URI1
URI2

Motifs et expressions de chemin

- **Motif de chemin :**
  Un motif de triplet **`s p o`** où **`p`** est une expression de chemin
- **Expression de chemin :** une expression régulière sur l'alphabet des propriétés. Plus précisément :

Syntaxe des expressions de chemin :

- Un URI est une expression de chemin
- Si `e, e1, e2` sont expressions de chemins alors les suivantes le sont aussi :

```
e1 / e2
e1 | e2
e?
e*
^e
(e)
```

Sémantique des motifs de chemin

- Un chemin dans un graphe RDF peut "matcher" une expression de chemin.
- Avec reference à un graphe RDF G on écrira

- Si n et m dans G sont connectés par un chemin qui est conforme à e (défini formellement dans le slide suivant)

Un motif de chemin `?x e ?y` evalué sur G retourne `(?x->n, ?y->m)` ssi

dans G

Sémantique des expressions de chemin

e	$n \xrightarrow{e} m$ dans G ssi n,m sont des URI de G et G contient :
`URI`	$n \xrightarrow{URI} m$
`e1 / e2`	$\exists\ n_1 \quad n \xrightarrow{e_1} n_1 \xrightarrow{e_2} m$
`e1 | e2`	$n \xrightarrow{e_1} m$ ou $n \xrightarrow{e_2} m$
`e ?`	$n \xrightarrow{e} m$ ou $n = m$
`e*`	$\exists\ n_1 \dots n_k \mid n_0 (=n) \xrightarrow{e} n_1 \xrightarrow{e} n_2 \cdots \xrightarrow{e} n_k (=m) \quad k \geq 0$
`^e`	$n \xleftarrow{e} m$
`(e)`	$n \xrightarrow{e} m$

Expressions de chemin : raccourcis

`e+` ⟷ `e / e*`

`e1 ^ e2` ⟷ `e1 / ^e2`

`e{k}` ⟷ un chemin de k occurrences de `e`

`e{k,j}` ⟷ un chemin d'au moins k occurrences et au plus j occurrences de `e`

`e{k,}` ⟷ un chemin d'au moins k occurrences de e

`e{,k}` ⟷ un chemin d'au plus k occurrences de `e`

Expressions de chemin : Exemple 1

Turtle ex11

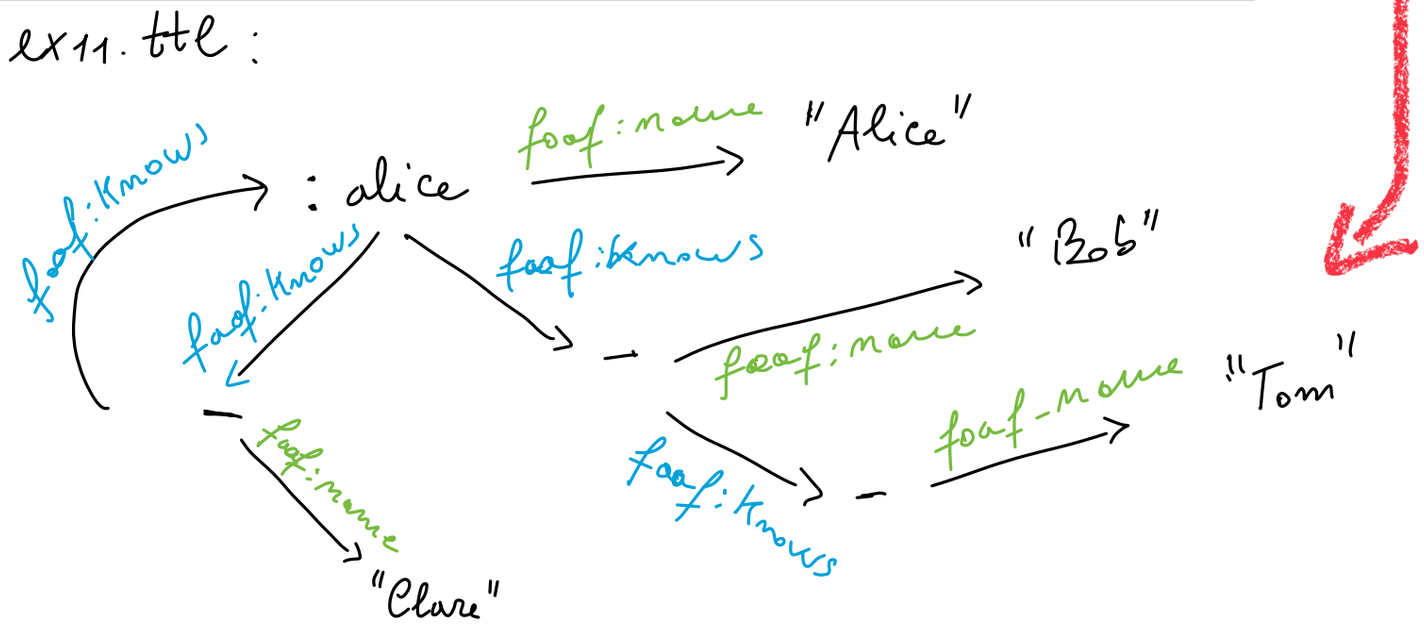
```
:alice  foaf:name   "Alice"   ;
        foaf:knows [foaf:name "Bob";   foaf:knows [foaf:name "Tom"]];
        foaf:knows [foaf:name "Clare"; foaf:knows :alice ] .
```

Requête ex11

```
SELECT ?xn ?yn
FROM <ex11.ttl>
WHERE  { ?x foaf:name ?xn; foaf:knows+ [ foaf:name ?yn ] . }
```

Résultat de ex11

xn	yn
"Clare"	"Alice"
"Clare"	"Clare"
"Clare"	"Bob"
"Clare"	"Tom"
"Bob"	"Tom"
"Alice"	"Clare"
"Alice"	"Alice"
"Alice"	"Bob"
"Alice"	"Tom"



Expressions de chemin : Exemple 2

Turtle ex11

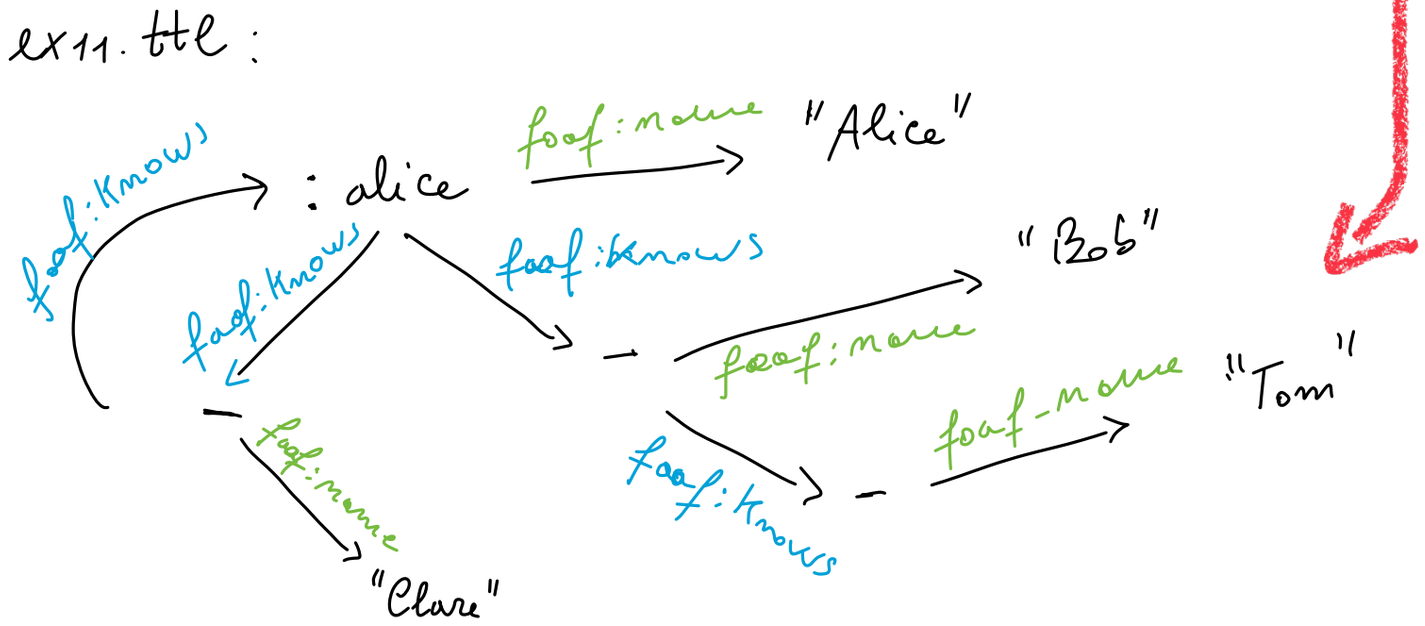
```
:alice  foaf:name   "Alice"   ;
        foaf:knows [foaf:name "Bob";   foaf:knows [foaf:name "Tom"]];
        foaf:knows [foaf:name "Clare"; foaf:knows :alice ] .
```

Requête ex13

```
SELECT DISTINCT ?xn ?yn
FROM <ex11.ttl>
WHERE  { ?x foaf:name ?xn; (foaf:knows | ^foaf:knows)  [ foaf:name ?yn ] . }
```

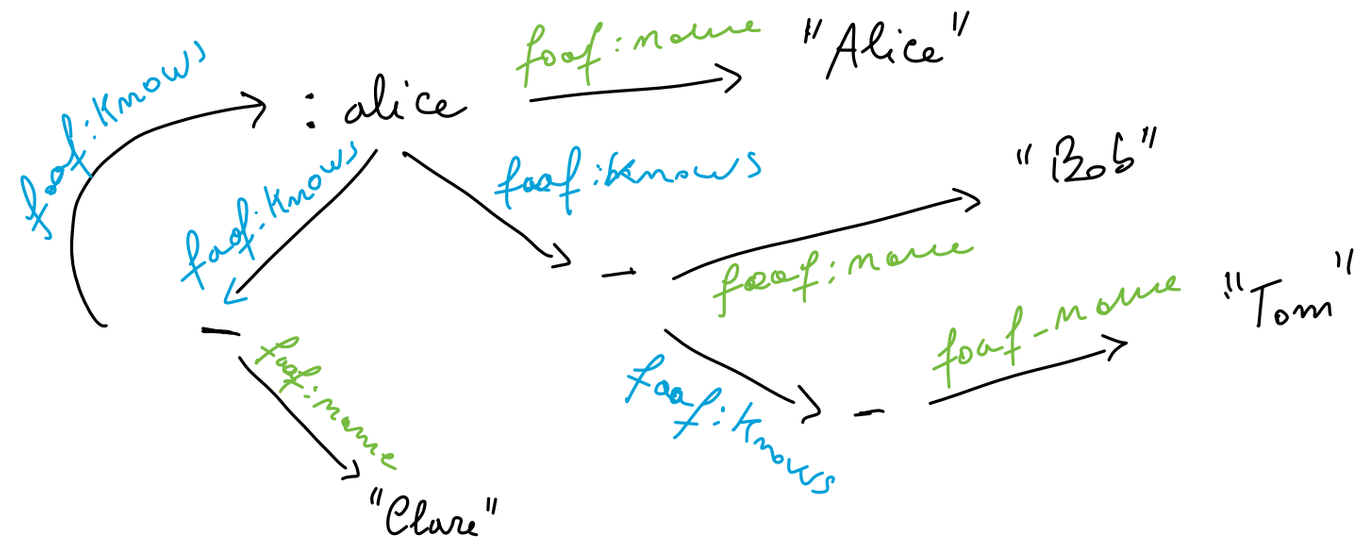
Résultat de ex13

xn	yn
"Clare"	"Alice"
"Tom"	"Bob"
"Bob"	"Tom"
"Bob"	"Alice"
"Alice"	"Clare"
"Alice"	"Bob"



Expressions de chemin : Exemple 3

ex11.ttl :



Requête ex12

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?xn ?yn
FROM <ex11.ttl>
WHERE  { ?x foaf:name ?xn;
         (foaf:knows+ | ^foaf:knows+)
                  [ foaf:name ?yn ] . }
```

Résultat de ex12

xn	yn
"Clare"	"Alice"
"Clare"	"Clare"
"Clare"	"Bob"
"Clare"	"Tom"
"Clare"	"Alice"
"Clare"	"Clare"
"Tom"	"Bob"
"Tom"	"Alice"
"Tom"	"Clare"
"Bob"	"Tom"
"Bob"	"Alice"
"Bob"	"Clare"
"Alice"	"Clare"
"Alice"	"Alice"
"Alice"	"Bob"
"Alice"	"Tom"
"Alice"	"Clare"
"Alice"	"Alice"

SPARQL : Enrichir les requêtes

ORDER BY /LIMIT /OFFSET

Requête orderby

```
SELECT ?title ?price
FROM <ex5.ttl>
WHERE { ?x ns:price ?price ;
          dc:title ?title . }
ORDER BY ?price
```

Résultat de orderby

```
-----------------------------
| title               | price |
===============================
| "The Semantic Web" | 23    |
| "SPARQL Tutorial"  | 42    |
-----------------------------
```

Requête orderbyoffset

```
SELECT ?title ?price
FROM <ex5.ttl>
WHERE { ?x ns:price ?price ;
          dc:title ?title . }
ORDER BY ?price
LIMIT 1
OFFSET 1
```

Résultat de orderbyoffset

```
-----------------------------
| title               | price |
===============================
| "SPARQL Tutorial" | 42    |
-----------------------------
```

BIND

- Enrichir les assignations retournées par un motif :

```
@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 .
:book1  ns:discount  0.2 .
:book2  dc:title     "The Semantic Web" .
:book2  ns:price     23 .
:book2  ns:discount  0.25 .
```

```
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)
   }
   {?x dc:title ?title . }
   FILTER(?price < 20)
}
```

BIND enrichit les assignations retournées par le motif de graphe avec une nouvelle variable `?price` calculée par une expression sur les variables du motif

```
-----------------------------
| title               | price |
=============================
| "The Semantic Web" | 17.25 |
-----------------------------
```

Clause FROM : graphes anonymes et graphes nommés

- RDF permet de nommer les graphes, c'est-à-dire les identifier par un URI

Si **`n` est un URI** et **`e`** est un ensemble de triplets (graphe) RDF , **`n{e}`**

désigne le graphe **`e`** nommé par l'URI **`n`**

- l'URI d'un graphe nommé peut être utilisé dans la partie FROM d'une requête SPARQL

Clause FROM : graphes anonymes et graphes nommés

- **pas de clause FROM** :
  - graphe par défaut choisi, par exemple, dans l'interface utilisateur.
- **une ou plusieurs clauses FROM <...>** : graphe(s) anonyme(s) ;
  - le graphe par défaut est défini par l'union des graphes anonymes donnés dans les clauses FROM
- **une ou plusieurs clauses FROM ou FROM NAMED <...>** :
  - Graphe par défaut défini par les clauses FROM
  - Graphe(s) nommé(s) définis par les clauses FROM NAMED
    - peuvent être référencés dans la clause WHERE avec la clause GRAPH (cf slide suivant)

Clause FROM : graphes anonymes et graphes nommés

- **ex5.ttl**

```
ns:book1 dc:title "SPARQL Tutorial" ; ns:price 42 .
ns:book2 dc:title "The Semantic Web"; ns:price 23 .
ns:book3 dc:title "RDF Framework" .
```

- **ex6.ttl**

```
ns:book3 ns:price 15 .
ns:book5 dc:title "SPARQL 1.1" ;
         ns:price 8 .
```

- **ex0.ttl**

```
ns:book4 ns:price 28;
      dc:title "Computer Science" .
```

Requête :

```
SELECT ?title ?price
FROM  <ex0.ttl>
FROM NAMED <ex5.ttl>
FROM NAMED <ex6.ttl>
WHERE {
       ?x ns:price ?price .
      FILTER (?price < 30) .
      ?x dc:title ?title .
}
```

Interroge uniquement le graphe par défaut (**<ex0.ttl>**)

Résultat

title	price
"Computer Science"	28

Clause FROM : graphes anonymes et graphes nommés

- **ex5.ttl**

```
ns:book1 dc:title "SPARQL Tutorial" ; ns:price 42 .
ns:book2 dc:title "The Semantic Web"; ns:price 23 .
ns:book3 dc:title "RDF Framework" .
```

- **ex6.ttl**

```
ns:book3 ns:price 15 .
ns:book5 dc:title "SPARQL 1.1" ;
         ns:price 8 .
```

- **ex0.ttl**

```
ns:book4 ns:price 28;
       dc:title "Computer Science" .
```

Requête :

```
SELECT ?title ?price
FROM  <ex0.ttl>
FROM NAMED <ex5.ttl>
FROM NAMED <ex6.ttl>
WHERE { GRAPH <ex5.ttl> {
        ?x ns:price ?price .
       FILTER (?price < 30) .
       ?x dc:title ?title .}
}
```

Interroge uniquement le graphe nommé **<ex5.ttl>**

Résultat

title	price
"The Semantic Web"	23

Clause FROM : graphes anonymes et graphes nommés

- **ex5.ttl**

```
ns:book1 dc:title "SPARQL Tutorial" ; ns:price 42 .
ns:book2 dc:title "The Semantic Web"; ns:price 23 .
ns:book3 dc:title "RDF Framework" .
```

- **ex6.ttl**

```
ns:book3 ns:price 15 .
ns:book5 dc:title "SPARQL 1.1" ;
         ns:price 8 .
```

- **ex0.ttl**

```
ns:book4 ns:price 28;
       dc:title "Computer Science" .
```

Requête :

```
SELECT ?title ?price
FROM  <ex0.ttl>
FROM NAMED <ex5.ttl>
FROM NAMED <ex6.ttl>
WHERE { GRAPH ?g {
        ?x ns:price ?price .
       FILTER ( ?price < 30) .
       ?x dc:title ?title .}
}
```

Le pattern est évalué séparément sur chaque graphe nommé (pas sur le graphe par défaut). Résultat

title	price
"The Semantic Web"	23
"SPARQL 1.1"	8

Clause FROM : graphes anonymes et graphes nommés

`GRAPH ?g {motif de graphe}`

Est un nouveau motif de graphe.

Résultat de `GRAPH ?g {motif de graphe}`
évalué sur un ou plusieurs graphes nommés :

- `{motif de graphe}` est évalué indépendamment sur chaque graphe nommé (et pas sur le graphe par défaut)
- Chaque assignation retournée par `{motif de graphe}` sur un graphe nommé d'identifiant `n` est complétée par `?g -> n`.

Clause FROM : graphes anonymes et graphes nommés

- **ex5.ttl**

```
ns:book1 dc:title "SPARQL Tutorial" ; ns:price 42 .
ns:book2 dc:title "The Semantic Web"; ns:price 23 .
ns:book3 dc:title "RDF Framework" .
```

- **ex6.ttl**

```
ns:book3 ns:price 15 .
ns:book5 dc:title "SPARQL 1.1" ;
         ns:price 8 .
```

- **ex0.ttl**

```
ns:book4 ns:price 28;
      dc:title "Computer Science" .
```

Requête :

```
SELECT ?g ?title ?price
FROM  <ex0.ttl>
FROM NAMED <ex5.ttl>
FROM NAMED <ex6.ttl>
WHERE { GRAPH ?g {
        ?x ns:price ?price .
       FILTER ( ?price < 30) .
       ?x dc:title ?title .}
}
```

=> On peut inclure dans le résultat le graphe :

g	title	price
<ex5.ttl>	"The Semantic Web"	23
<ex6.ttl>	"SPARQL 1.1"	8

Agrégation

- Rappel : le résultat d'un motif de graphe (partie WHERE) est un ensemble d'assignations
- Ensemble d'assignations ⇔ table

ex14.ttl

```
:book1 dc:title "SPARQL Tutorial" ; ns:price 42 ; ns:editor :jena .
:book2 dc:title "The Semantic Web" ; ns:price 23 ; ns:editor :w3c .
:book3 dc:title "RDF Framework" ; ns:price 15 ; ns:editor :w3c .
:book4 dc:title "SPARQL 1.1" ; ns:editor :jena .
```

```
SELECT *
FROM <ex14. ttl >
WHERE { ?x ns:editor ?editor }
```

Resultat

?x	?editor
:book1	:jena
:book2	:w3c
:book3	:w3c
:book4	:jena

- On peut regrouper et agréger la table résultat de la même façon qu'en SQL

Agrégation

- Rappel : le résultat d'un motif de graphe (partie WHERE) est un ensemble d'assignations
- Ensemble d'assignations ⇔ table

ex14.ttl

```
:book1 dc:title "SPARQL Tutorial" ; ns:price 42 ; ns:editor :jena .
:book2 dc:title "The Semantic Web" ; ns:price 23 ; ns:editor :w3c .
:book3 dc:title "RDF Framework" ; ns:price 15 ; ns:editor :w3c .
:book4 dc:title "SPARQL 1.1" ; ns:editor :jena .
```

```
SELECT (COUNT( ?editor ) AS ?c)
FROM <ex14. ttl>
WHERE { ?x ns:editor ?editor }
```

Resultat

?c
4

- Remarque : comme en SQL, `COUNT` ne compte pas les valeurs distinctes, `COUNT (DISTINCT ?editor)` pour les valeur distinctes

Agrégation avec GROUP BY

Turtle ex14

```
:book1  dc:title  "SPARQL Tutorial" ; ns:price  42 ; ns:editor :jena .
:book2  dc:title  "The Semantic Web" ; ns:price  23 ;  ns:editor :w3c .
:book3  dc:title  "RDF Framework" ; ns:price  15 ;  ns:editor :w3c .
:book4  dc:title  "SPARQL 1.1" ; ns:editor :jena .
```

Requête ex15

```
SELECT  ?editor (AVG(?price) AS ?c)
FROM <ex14.ttl>
WHERE   { ?x ns:editor ?editor ; ns:price ?price }
GROUP BY ?editor
```

Résultat de ex15

```
----------------
| editor | c    |
==================
| :w3c   | 19.0 |
| :jena  | 42.0 |
----------------
```

Agrégation - GROUP BY et HAVING

Turtle ex14

```
:book1  dc:title  "SPARQL Tutorial" ; ns:price  42 ; ns:editor :jena .
:book2  dc:title  "The Semantic Web" ; ns:price  23 ;  ns:editor :w3c .
:book3  dc:title  "RDF Framework" ; ns:price  15 ;  ns:editor :w3c .
:book4  dc:title  "SPARQL 1.1" ; ns:editor :jena .
```

Requête ex16

```
SELECT  ?editor (AVG(?price) as ?c)
FROM <ex14.ttl>
WHERE   { ?x ns:editor ?editor ;
          ns:price ?price }
GROUP BY ?editor
HAVING (AVG(?price) > 20)
```

Résultat de ex16

```
----------------
| editor | c    |
================
| :jena  | 42.0 |
----------------
```

Fonction d'agrégation

```
COUNT
COUNT (DISTINCT …)
SUM
AVG
MIN
MAX
SAMPLE
GROUP_CONCAT
```

- Voir http://www.w3.org/TR/2013/ REC-sparql11-query-20130321/#aggregates

Fonctions et expressions

- Dans une requête SPARQL toute variable ou terme (URI ou littéral) peut être manipulé avec des fonctions
  - fonctions XPath 2.0 ou externes
  - exemple, dans un filtrage: `FILTER (xsd:integer(?age) >= 18)`

- Quelques exemples de fonctions :
  - (re-)typage (casting XML Schema): `xsd:integer(val)`
  - Sur les chaînes de caractères : `CONCAT(lit1,…,litn), SUBSTR(lit, start [,length]), STRLEN(str), …`
  - Sur les dates : `NOW(), YEAR(date), MONTH(date),…`
  - Numériques : `ABS(val), CEIL(val), …`
  - Booléennes (peuvent être utilisées directement comme tests dans FILTER) : `regex (str, pat), in(expr1, expr2,…), contains (str, sub),…`

- Reference : http://www.w3.org/TR/sparql11-query/#expressions

Fonctions et expressions - exemples

- Des expressions sur les variables et les termes, utilisant éventuellement des fonctions, peuvent être utilisées à la place d'une variable à tout endroit de la requête

Turtle ex14

```
:book1  dc:title  "SPARQL Tutorial" ; ns:price  42 ; ns:editor :jena .
:book2  dc:title  "The Semantic Web" ; ns:price  23 ;  ns:editor :w3c .
:book3  dc:title  "RDF Framework" ; ns:price  15 ;  ns:editor :w3c .
:book4  dc:title  "SPARQL 1.1" ; ns:editor :jena .
```

Requête ex14regex

```
SELECT   ?x ?t
FROM <ex14.ttl>
WHERE    { ?x dc:title ?t .
           FILTER (regex(str(?t),"^S.*r.*l$", "i")) }
```

Résultat de ex14regex

```
-------------------------------------------------
| x                          | t                |
=================================================
| <http://example.org/book/book1> | "SPARQL Tutorial" |
-------------------------------------------------
```

Fonctions et expressions - exemples

COALESCE(exp1,exp2,...) retourne le résultat de la première expression expi sans erreur.
Si ?x = 2, !bound(?y) :

COALESCE(?x, 1/0) = 2
COALESCE(1/0 , ?x) = 2
COALESCE(5 , ?x) = 5
COALESCE(?y, 3) = 3
COALESCE(?y) = erreur

Fonctions et expressions - exemples

Turtle coalesce

```
:book1  :title  "SPARQL Tutorial" ; :price  42 ; :editor :jena .
:book2  :title  "The Semantic Web" ; :price  23 ; :reduction 10 .
:book3  :title  "RDF Framework"; :price  53 ; :reduction 20.
:book4  :title  "SPARQL pour les Nuls" ; :editor :pourlesnuls .
:pourlesnuls :address  "Paris" .
```

Requête coalesce

```
SELECT ?titre (COALESCE(?price*(1.0-?reduc/100), ?price) as ?prix)
FROM <coalesce.ttl>
WHERE { ?x ns:title ?titre ; ns:price ?price .
                    OPTIONAL { ?x ns:reduction ?reduc } }
```

Résultat de coalesce

```
---------------------------
| titre              | prix |
=============================
| "RDF Framework"    | 42.4 |
| "The Semantic Web" | 20.7 |
| "SPARQL Tutorial"  | 42   |
---------------------------
```

Sous-requêtes

- **Remarque** : une requête retourne un ensemble d'assignations tout comme un motif de graphe.
- ⇒ un requête peut être utilisée à la place d'un motif de graphe dans la partie WHERE (sous-requête)
- Le résultat de la sous-requête se combine avec les autres motifs de graphes par la même sémantique déjà vue pour les motifs de graphes

Sous-requêtes - exemples

ex-sub.ttl

```
:book2 dc:title "The Semantic Web" ; ns:price 24 ; ns:editor :w3c .
:book3 dc:title "RDF Framework" ; ns:price 16 ; ns:editor :w3c .
:book3 dc:title "RDF Framework 2" ; ns:price 20 ; ns:editor :w3c .
:book1 dc:title "SPARQL Tutorial" ; ns:price 42 ; ns:editor :jena .
:book4 dc:title "SPARQL 1.1" ; ns:editor :jena .
```

ex-sub.sparql

```
SELECT ?editor ?livre
FROM <test-sub.ttl>
WHERE { ?livre ns:editor ?editor; ns:price ?maxprice .
        { SELECT ?editor (max(?price) as ?maxprice)
          WHERE { ?y ns:editor ?editor ; ns:price ?price . }
          GROUP BY ?editor }
}
```

Pour chaque éditeur le(s) livre(s) ayant le prix le plus élevé

Resultat

```
-------------------------------------------------
| editor                  | livre                 |
=================================================
| <http://asws.org/jena> | <http://asws.org/book1> |
| <http://asws.org/w3c>  | <http://asws.org/book2> |
-------------------------------------------------
```

SPARQL Endpoints

- **SPARQL Endpoint (point de terminaison SPARQL)**: un service d'accès à un jeu de données RDF/S qui accepte des requêtes SPARQL et renvoie les résultats via HTTP
  - Identifié par l'URI auquel le service écoute
  - Peut être interrogé par n'importe quel client capable d'executer des requêtes HTTP de GET or POST
  - Par un programme en general via REST API
  - Possède en général également une interface Web qui permet de saisir les requêtes SPARQL via un navigateur
- Cf. liste des SPARQL endpoints actifs : https://www.w3.org/wiki/SparqlEndpoints (avec URI + adresse de la GUI Web)
- Les frameworks/bibliothèques (comme Jena pour Java, RDFLib pour python,…) permettent en general l'interrogation des endpoints (méthodes qui envoient des requêtes à un URI où un endpoint écoute)
- Toutefois SPARQL 1.1 a intégrée dans le langage de requêtes lui même l'interrogation de services endpoints (requêtes fédérées)

Requêtes fédérées (SERVICE)

- En plus du graphe par défaut et des graphes nommés, une requête SPARQL peut interroger un “remote SPARQL endpoint"
- La syntaxe des motifs de graphes est augmentée :
  - `SERVICE <uri> {graph pattern}`
    est un nouveau motif de graphe
  - On peut le retrouver à la place d’un motif de graphe de la requête
- Ce nouveau motif interroge le SPARQL endpoint à l’URI `<uri>` en lui envoyant la requête `{graph pattern}`
- Le résultat du motif est un ensemble d’assignations, celles retournées par le SPARQL endpoint suite l’evaluation de `{graph pattern}`
  - Cela se combine avec les résultats des autres motifs de la requête selon la sémantique habituelle
- Les requêtes avec SERVICE s’appellent fédérées

Requêtes fédérées (SERVICE)

test-federe.ttl

```
[ rdf:type dbo:Artist ] .
```

test-federe.sparql

```
SELECT ?a ?t
FROM <test-federe.ttl>
WHERE {
    ?x rdf:type ?art
    SERVICE <http://dbpedia.org/sparql> {
        ?a rdf:type ?art;
           dbp:partner ?t;
           dbo:occupation ?o
        filter( contains(str(?o), "designer"))
    }
}   LIMIT 10
```

Résultat

a	t
dbr:Gianni_Versace	<http://dbpedia.org/resource/Antonio_D'Amico>
<http://dbpedia.org/resource/Yves_Saint_Laurent_(designer)>	dbr:Pierre_Bergé
dbr:Alber_Elbaz	"Alex Koo"^^rdf:langString
dbr:Bob_Mackie	dbr:Ray_Aghayan

Alternatives à SELECT

Requête SPARQL

```
<format>
  FROM <source>
 WHERE { <motif> }
<transform>
```

- <format> définit le format du résultat
- FROM définit la source RDF (optionnel si source par défaut)
- <motif> est un *motif de graphe*
- <transform> est un transformateur : ORDER, LIMIT, OFFSET

format	**résultat**
SELECT, SELECT DISTINCT	table de données (bindings)
CONSTRUCT	graphe RDF
ASK	valeur Booléenne (résultat non-vide)
DESCRIBE	“description” des ressources trouvées

Formatter le résultat : CONSTRUCT

```
CONSTRUCT {GP1}
FROM <source>
WHERE {GP2}
```

GP1 et GP2 : motifs de graphes

- Pour chaque assignation v retournée par l'evaluation de GP2
  - L' ensemble de triplets v(GP1) est ajouté au résultat
    - (union ensembliste des triplets)
  - v(GP1) : triplets obtenues de GP1 après avoir remplacé chaque variable par sa valeur en v
    - Les triplets de v(GP1) ayant des valeurs de variables unbound en v ne sont pas incluses dans le résultat

Formatter le résultat : CONSTRUCT

ex5.ttl

```
Turtle ex5
ns:book1 dc: title "SPARQL Tutorial" ; ns:price 42 .
ns:book2 dc: title "The Semantic Web"; ns:price 23 .
ns:book3 dc: title "RDF Framework" .
```

Union.sparql

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://asws.org/ns#>
CONSTRUCT { ?x ns:prix ?price;
              ns:titre ?title . }
FROM <ex5.ttl>
WHERE { { ?x ns:price ?price . }
        UNION
        { ?x dc:title ?title . } }
```

Resultat :

```
ns:book2  ns:prix  23 ;
          ns:titre  "The Semantic Web" .
ns:book1  ns:prix  42 ;
          ns:titre  "SPARQL Tutorial" .
ns:book3  ns:titre  "RDF Framework” .
```

Formatter le résultat : ASK

- Au lieu de retourner le résultat, ASK retourne un booléen : true ssi la requête retourne au moins un résultat

ask.ttl

```
_:a   foaf:name        "Alice" .
_:a   foaf:homepage    <http://work.example.org/alice/> .

_:b   foaf:name        "Bob" .
_:b   foaf:mbox        <mailto:bob@work.example> .
```

ask.sparql

```
ASK
WHERE { ?x foaf:name  "Alice" }
```

Resultat :

```
 true
```