Sparql

point Important

- motif de triplet ◦ U : ensemble des URI
  ◦ http://www.w3.org/TR/rdf-syntax-grammar
  ◦ dc:title
  ◦ L : litéraux RDF (valeurs) : "valeur"@motcle^^type
  ◦ @motcle : langue, monnaie, encodage, ... (optionnel)
  ◦ ^^type : type XML Schema (optionnel)
  ◦ "RDF1.1 XML Syntax"@en, "Dave Beckett", "false"^^xsd:boolean
  ◦ B : nœuds blancs
  ◦ V : ensembles de noms de variables (?nom ou $nom) :
  ◦ ?x, ?nom, ?y, $nom, $a
  Un motif de triplet RDF est un élément de l'ensemble :
  $(U \cup B \cup V) \times (U \cup V) \times (U \cup L \cup B \cup V)$

- `.` pour faire la jointure des triplet

- `;` pour faire les operations sur une meme variable

- `Filter()` permet de restreindre les solutions sur tout le groupe où le filtre apparaît. Sa position n'a pas d'importance

  - cf . TD ex3 q4

- `Optional` : permet d'obtenir des solutions même si des parties du motif d'interrogation ne correspondent pas.

- `UNION` : permet d'indiquer des alternatives de motifs (un motif OU un autre peut correspondre).

```
    ?x foaf:name ?n . }
    UNION
    { ?x foaf:mbox ?m .
    ?x foaf:age ?a . }
    }
```

- Négation : peut etre exprimet de 2 facon

  - `FILTER NOT EXIST { ...}`teste la non-existence d'un motif
  - `MINUS` permet de retirer des solutions provenant d'un autre motif de graphe
    - cf TD ex3 q6

- Operateurs : ◦ Séquence : `/`
  ◦ Alternative : `|`
  ◦ Répétition :
  ◦ `*` (0 ou plusieurs occurrences)

    - `+` (1 ou plusieurs occurrences)
    - Option : `?`
    - Arc inverse : `^` ◦ Négation : `!`

- La clause `FROM` est utilisée pour indiquer l'URI d'un graphe sur lequel effectuer la requête. En l'absence de clause FROM, l'interrogation s'effectue sur le graphe par défaut.
- `order by` : fait l'ordre en fonction du variable
- `limit` : limite le nb d'affichage
- `OFFSET` n : commencer à la solution n+1. OFFSET 0 n'a pas d'effet
- Tests (SPARQL) : `isURI, isBlank, isLiteral, bound`
- Logique : `!, &&, ||`
- `OPTIONAL + bound()` permet d'exprimer la quantification existentielle
    - Bound() renvoie true si la variable est liée, false sinon.

```
PREFIX : <http://mlbda.fr/>
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    SELECT ?x
    WHERE {
    { ?x foaf:mbox ?m . }
    OPTIONAL { ?x foaf:age ?a . }
    FILTER (!BOUND(?a))
}```
```

- `Ask` : sur un triplet, renvoie true si un truc demander existe

```
ASK { ?x foaf:name "Peter Goodguy" } // revoie true existe dans le RDF
ASK { ?x foaf:name "Bill Badguy"} // renvoie false  n'existe pas
```

- `Construct` : construit un graph rdf

```
CONSTRUCT { ?x a :Adult }
WHERE {
{ ?x foaf:age ?a . }
FILTER (?a > 18)
```

TD

2 triplets peuvent partager :

- un sujet
- une propriété permet de factoriser propriétés :
- enemy of
- a
- name on développe le format factoriser si : $p_1 ; o_1$ $p_2 ;o_2$ $\iff$ si $p_1 ; o_1$ si $p_2 ; o_2$

si $p_1 ; o_1 \rightarrow$ si $p_1 ; o1$ si $p_1 ; o_1$

Ex2

1 . a)

? p email ?e . ?p web ?w
? p :

- mail ?e
- web ?w

?p	?e
p2	rick@
p4	liz@

?p	?w
p3	lars
p4	liz.pers

Resultat :

?p	?e	?w
p4	liz@	liz.pers

b) requete avec union

(?p , ?e) union (?p, ?w) ?

(?p, ?e , null) union (?p , null. ?w)

Resultat:

?p	?e	?w
p2	rick@	null
p4	liz@	null
p3	null	lars.com
p4	null	liz.pers

c) **OPTIONNEL : la partie gauche est toujours incluse dans le resultat**

?p	?e	?w
p2	rick@	null
p4	liz@	liz.pers

2 . OPTIONNEL imbriqués {?pnme ?n optionnal ? p.email ?e } optionnal ?p web ?w

?p	?n	?e	?w
p1	john	null	null
p2	rick	rick@	null
p3	lars	null	lars.com
p4	liz	liz@	liz.pers

3 . les requetes 2) et 3) sont differentes

?p	?n	?e	?w
p1	john	null	null
p2	rick	rick@	null
p3	lars	null	null
p4	liz	liz@	liz.pers

la ligne 3 differente

4 a) filter(bound(...)) ?p web ?w optionnal ?p phone ?l filter (!bound(?l))

les personnes qui ont une page web et pas de telephone

resultat : (p3 , lars.com , null) p4 n'est pas dans le resultat

Ex 3

1. extraire les villes citees dans des triplets de cette base ville ? une entite `vit` dans une ville une entite `locatedAt` une ville une ville `a City`

```
select distinct ?v
where {
    {?x : livesIn ?v}
    UNION
    {?y : locatedAT ?v}
    UNION
    {?v  aCity : City }
}
```

2. Extraire les personnes qui ont etudie dans la meme universtie que l'un de leur parents

```
select distinct ?p
where { les points pour faire les jointures afin de relier les variables
IMPORTANT
    {
        {
            ?p :hasFather ?f .
            ?f :studiedAt ?u .
            ?p :studiedAt ?u
        }
        UNION
        {
            ?p :hasMother ?m .
            ?m :studiedAt ?u.
            ?p :studiedAt ?u
        }
    }.
    ?p a :Person   //ou bien de mettre ca comme jointure dans les 2
    Rmq : toutes les "personnes " de la base ne sont pas types en tant que
:Personne donc on evite de preciser cette condition
    }
```

utilisation d'un **.** pour la jointure Autre solution **mieux factoriser**

```
select distinct ?p
where { les points pour faire les jointures afin de relier les variables
IMPORTANT
    {
        ?p :studiedAt ?u.
        ?parent :studiedAt ?u
    }.
    {
        {
            ?p :hasFather ?parent
        }
        UNION
        {
        ?p :hasMother ?parent.
        }
    }.
    ?p a :Person  //ou bien de mettre ca comme jointure dans les 2
}
```

3. extraire les personnes qui ont etudee dans une universite ou leurs deux parents ont etudie

```
select distinct ?p
where {
    ?p :studiedAt ?u.
    ?p  :hasMother ?m.
```

```
    ?m  :studiedAt ?u.
    ?p :hasFather ?f.
    ?f :studiedAt ?u
}
```

4. extraire les personnes qui ont etudie dans une universite differente de celle de leur pere et leur mere

```
select distinct ?p
where {
    ?p :studiedAt ?u1.
    ?p  :hasMother ?m.
    ?m  :studiedAt ?u2.
    ?p :hasFather ?f.
    ?f :studiedAt ?u3.
    filter(?u1 != u2 && ?u1 !=u3)
}
```

autre solution avec **MINUS** (Minus c'est une operation ensembliste, importance de renommage pour faire la soustraction que sur la variable qui nous interesse) Rmq: les Etudiants sans parent sont inclus dans le resultat

```
select distinct ?p
where {
    {?p :studiedAt ?u.}
    MINUS
    {
        ?p :hasMother ?m.
        ?p :hasFather ?f.
        ?m :studiedAt ?u.
        ?f :studiedAt ?u.
    }
}
```

6. extraire les personnes qui etudie dans une ville differente de celle ou ils habitent

```
select distinct ?p
where{
    ?p :livesIn ?v1.
    ?p :studiedAt ?u.
    ?u :locatedAt ?v2.
    filter(?v1!=?v2)
}
```

modelisation de filter

?p	?v1	?v2	?u
p1	v1	v2	u1
p1	v3	v2	u1

autre solution avec **MINUS**

```
select distinct ?p
where{
    {
        ?p :livesIn ?v1.
        ?p :studiedAt ?u.
        ?u :locatedAt ?v2.
    }
    MINUS
    {
        ?p :studiedAt ?v3
        ?p :studiedAt ?u2.
        ?u2 :locatedAt ?v3

    }
}
```

OU

```
select distinct ?p
where{
    {
        ?p :livesIn ?v.
        ?p :studiedAt ?u.
    }
    MINUS
    {
        ?u :locatedAt ?v

    }
}
```

TME

1. 1-a les affluents (:River) directs et indirects de la mer du nord

```
Select distinct ?r
where {
    ?r a :River.
    ?r :flowsInto <seas/North+Sea/>
}
```

3. 1-b- le nombre d'affluents (:River) directs et indirects de la mer du nord

```
Select distinct ?r
where {
    {
        ?r a :River.
        ?r :flowsInto+ <seas/North+Sea/>.  // le + tres important en
sparql
    }
}
```

5. 2-a Le affluents (:River) directs ou indirects de la mer du nord qui passent par le Rhin

```
Select distinct ?r

where {
    {
    ?r a :River.
    ?r :flowsInto+ <rivers/Rhein/>.
    ?r :flowsInto+ <seas/North+Sea/>.
    }
}
```

7. 2-b Le affluents (:River) directs et indirects de la mer du nord qui ne passent pas par le Rhin

```
Select distinct ?r
where {
{
?r a :River.
?r :flowsInto+ <seas/North+Sea/>.
}
MINUS
{
?r a :River.
?r :flowsInto+ <rivers/Rhein/>.
?r :flowsInto+ <seas/North+Sea/>.
}
}
```

9. 3- Les rivières avec au moins vingt affluents.

```
SELECT ?r
WHERE {
?r a :River.
?r1 :flowsInto ?r.
```

```
}
GROUP BY ?r
HAVING ( count(?r1) >= 20)
```

11. 4- Les lacs traversés par le Rhone

```
SELECT distinct ?l
WHERE {
?l a :Lake.
?r a :River.
?r :name "Rhone".
?r :flowsThrough ?l.
}
```

13. 5- Les pays avec plus que 10000000 habitants traversés par le Danube

```
SELECT distinct ?p ?pop
WHERE {
    ?r a :River.
    ?r :name "Donau".
    ?p a :Country.
    ?r :locatedIn ?p.
    ?p :population ?pop.
    FILTER (?pop > 10000000)
}
```

15. 6 .Les Pays qui appartiennent à toutes les organisations auxquelles appartient aussi le Liechtenstein

```
SELECT ?name
WHERE {
?c a :Country.
?c :name "Liechtenstein".
?c :isMember ?o.
?res :isMember ?o.
?res :name ?name.
}
```