Introduction à SPARQL
(Simple Protocol & RDF Query Language)

Bernard ESPINASSE

Aix-Marseille Université
LIS UMR CNRS 7020

Septembre 2019

- Introduction à SPARQL
- Les requêtes SELECT
- Mise en œuvre de SPARQL
- Exemple d'interrogation d'une base RDF
- Aide mémoire SPARQL 1.1

Références

- **Livres, articles et rapports :**
  - O. Corby and F. Gandon and C. Faron-Zucker, Le Web sémantique : comment lier les données et les schémas sur le web ? Dunod, 2012.
  - G. Antoniou, Van Harmelen F., A Semantic Web Primer, The MIT Press Cambridge, Massachusetts London, England, 1999.
  - John Hebeler and Matthew Fisher and Ryan Blace and Andrew Perez-Lopez and Mike Dean, Semantic Web Programming, Wiley, 2009.
- **Web W3C :**
  - Page du W3C : https://www.w3.org/TR/rdf-sparql-query/
  - Page du W3C : https://www.w3.org/TR/sparql11-query/
  - Sparql par l'exemple : https://www.w3.org/2009/Talks/0615-qbe/
- **Cours/tutoriaux :**
  - Cours de M. Gagnon, Ecole Polytechnique de Montréal, 2007.
  - Cours de S. Staab, ISWeb – « Semantic Web », Univ. de Koblenz-Landau.
  - Cours de I. Mougenot, LIRMM, Univ. Montpellier
  - Cours de R. Gilleron, LIFL, Iniv. Lille 3
  - Cours de O. Corby, INRIA Nice
  - Cours de G. Lapalme, Université de Montréal.

Plan

1. Introduction à SPARQL

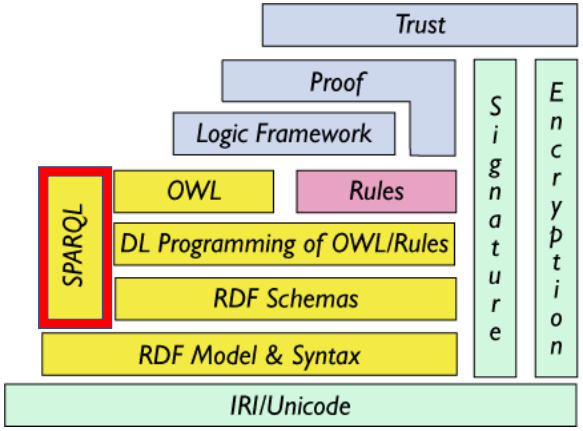
- **Qu'est ce que SPARQL**
- **SPARQL langage de requête**
- **Forme générale d'une requête SPARQL**
- **Types de requêtage en SPARQL**

Introduction à SPARQL

- **SPARQL** pour Simple Protocol and RDF Query Language
- Standard W3C 2008 pour faciliter l'interrogation de sources de données RDF distribuées
- SPARQL c'est :
  - Un langage de requête pour RDF
  - Un protocole : spécification pour émettre et envoyer des requêtes SPARQL (services Web) vers des serveurs dédiés et en recevoir les résultats
  - Un format XML pour l'affichage des résultats obtenus (requêtes de type SELECT et ASK)

On mettra ici l'accent essentiellement sur le **langage de requête**

Place de SPARQL dans le gâteau du WS …



Source : W3C, T Berners-Lee, Ivan Herman

SPARQL : langage de requête

- SPARQL = langage de requêtes standardisé d'interrogation de graphes RDF
- Comme SQL est le langage d'interrogation de bases de données relationnelles :
  - **SQL : interrogation de BD relationnelles :**
    - basé sur le calcul relationnel
    - les requêtes sont exprimées dans un langage logique de description du résultat : les champs sélectionnés, les jointures, les filtrages, les groupes et les opérations de groupe,
  - **SPARQL : interrogation de BD RDF :**
    - basé sur la notion de graphes de triplets
    - les requêtes sont décrites par des motifs (patterns) et des variables.

Forme générale des requêtes SPARQL

- **Forme la plus courante d'une requête SPARQL (SELECT) :**

SELECT [DISTINCT] $?var_1$ $?var_2$ ... $?var_m$
WHERE { $pattern_1$.
$pattern_2$.
...
$pattern_n$ }

- Les « patterns » sont des triplets en format **TURTLE (pas RDF/XML)**
- Les variables apparaissent dans les patterns
- Requêtes conjonctives (et)
- Résultat: table de valeurs (bindings) correspondant à ($?var_1$ $?var_2$ ... $?var_m$)

Types de requêtage dans SPARQL

4 Types de requêtage mais uniquement du **requêtage** à la différence de SQL (LDD + LMD) :

- **SELECT** : **rechercher** de ressources du modèle, résultats restitués sous un format tabulaire
- **ASK** : **indiquer** si la requête retourne un **résultat non vide** (test de vacuité)
- **DESCRIBE** : **obtenir des informations** à propos de ressources présentes dans le modèle (le moins exploité des quatre)
- **CONSTRUCT** : **construire de nouveaux graphes RDF** à partir des résultats de la requête servant de ”template”

Forme générale d'une requête SPARQL

ici requête SELECT :

Prologue

BASE prefix :<namespace-uri >

PREFIX prefix :<namespace-uri >

Head

SELECT [DISTINCT] variable-list

Body

FROM source-list

WHERE pattern

ORDER BY expression

LIMIT integer > 0

OFFSET integer > 0

Prologue d'une requête SPARQL

- BASE suivi d'un raccourci pour une URI de base auxquelles les URIs seront concaténées.
  - Une seule base par requête.
  - Exemple : BASE <http ://www.lis.univ-amu/data/>
- PREFIX suivi d'une déclaration d'un raccourci pour des espaces de nom.
  - Il est courant d'avoir plusieurs préfixes.
  - Exemple : PREFIX fb :<http ://rdf.freebase.com/ns/>

Head d'une requête SPARQL

- SELECT suivi d'une liste de variables dont on souhaite connaître les valeurs répondant à la requête.
  - Note : la requête peut utiliser d'autres variables.
- CONSTRUCT suivi d'un template de triplets pour construire un entrepôt des triplets répondant à la requête.
- ASK pour tester s'il existe des solutions vérifiant les conditions de la requête.

Body d'une requête SPARQL

Peut contenir les clauses :

- **FROM** (optionnel) permet de spécifier différents graphes de triplets dans lesquels il faut chercher. Si pas mentionné, on suppose qu'un graphe a été spécifié au processeur SPARQL
- **WHERE** suivi d'une déclaration de pattern (motif) pouvant contenir : plusieurs patterns, plusieurs patterns de base, des filtres, des unions, des présences optionnelles
- **ORDER BY** trier par …
- **LIMIT** pour donner un nombre maximal de réponses
- **OFFSET** pour commencer à partir d'un numéro de réponse
- Les contraintes **OPTIONAL** et **FILTER**

Remarques :

- pattern = ensemble de triplets contenant des noms de variables
- les triplets sont séparés par le symbole « . »
- un tuple de variables satisfait le pattern si tous les triplets sont satisfaits par le tuple : on parle de requête conjonctive

Simplification d'écriture

- Sur les types RDF :

  ?x **rdf:type** ?c ➔ **?x a ?c**

- Pour les sujets :

  ?x voc:b ?y ET **?x** voc:c ?z ➔ **?x voc:b ?y ; voc:c ?z**

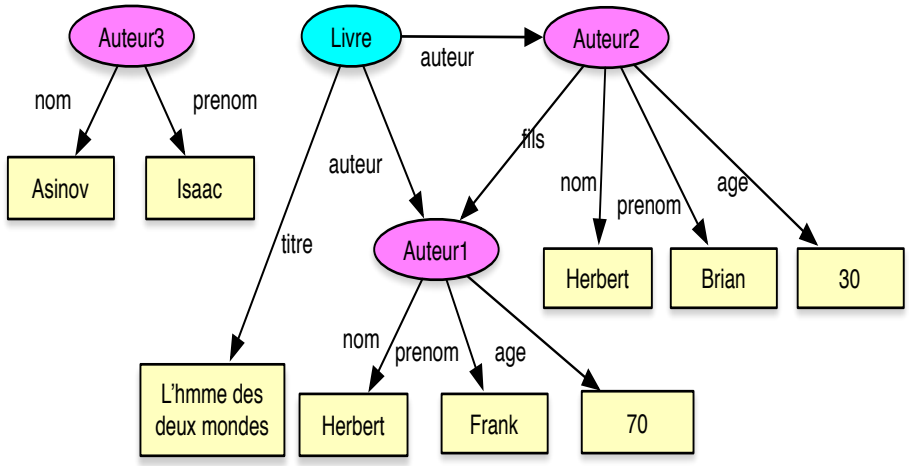
- Pour les prédicats :

  ?x voc:**b** ?y ET ?x voc:**b** ?z ➔ **?x voc:b ?y, ?z**

2. Requêtes SELECT

- **Requêtes simples**
- **DISTINCT, ORDERBY et LIMIT**
- **Contraintes OPTIONAL et FILTER**
- **Jointure classique et à gauche**
- **ADDITION, UNION et NEGATION**

Base RDF : livre de Frank et Brian HERBERT



Base RDF : livre de Frank et Brian HERBERT

Triplets RDF	Formulation Turtle (.ttl)
(#livre1, auteur, #auteur1) (#livre2, auteur, #auteur1) (#livre2, auteur, #auteur2) (#livre1, titre, Dune) (#livre2, titre, L'homme de deux mondes) (#auteur1, nom, Herbert) (#auteur1, prenom, Frank) (#auteur2, nom, Herbert) (#auteur2, prenom, Brian) (#auteur2, fils, #auteur1) (#auteur1, age, 70) (#auteur2, age, 30)	@prefix voc: <http://www.monvoc.fr> . @prefix xsd: <http://www.w3.org/2001/XMLSchema#> . <http://#livre1> voc:auteur <http://auteur1> ; voc:titre "Dune" . <http://#livre2> voc:auteur <http://auteur1>, <http://auteur2>; voc:titre "L'homme de deux mondes" . <http://#auteur1> voc:prenom "Frank" ; voc:nom "Herbert"; voc:age "70"^^xsd:integer . <http://#auteur2> voc:prenom "Brian" ; voc:nom "Herbert" ; voc:age "30"^^xsd:integer; voc:fils <http://auteur1> .

Requête SELECT très simple

Utilisation de Twinkle
(http://www.iro.umontreal.ca/~lapalme/ift6282/Twinkle.html)

Requête récupérant tous les triplets :

```
PREFIX voc:http://www.monvoc.fr
SELECT * WHERE {?subject ?predicate ?object}
```

Résultats :

subject	predicate	object
<http://#auteur1>	voc:age	70
<http://#auteur1>	voc:nom	"Herbert"
<http://#auteur1>	voc:prenom	"Frank"
<http://#livre1>	voc:titre	"Dune"
<http://#livre1>	voc:auteur	<http://auteur1>
<http://#auteur2>	voc:fils	<http://auteur1>
<http://#auteur2>	voc:age	40
<http://#auteur2>	voc:nom	"Herbert"
<http://#auteur2>	voc:prenom	"Brian"
<http://#livre2>	voc:titre	"L'homme de deux mondes"
<http://#livre2>	voc:auteur	<http://auteur2>
<http://#livre2>	voc:auteur	<http://auteur1>

Requête SELECT simple

Requête

```
PREFIX voc:http://www.monvoc.fr
SELECT ?prenom
WHERE { ?auteur voc:prénom ?prenom }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom
Frank
Brian

Requête SELECT : DISTINCT

DISTINCT : affiche seulement les éléments qui sont différents

Requête

```
PREFIX voc:http://www.monvoc.fr
SELECT DISTINCT ?nom
WHERE { ?auteur voc:nom ?nom. }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom
Herbert

Requête SELECT : ORDERBY et LIMIT

- ORDERBY permet d'ordonner les résultats de la requête suivant un ou plusieurs des éléments à afficher :

```
PREFIX voc:URIVOC
SELECT élément à afficher
WHERE { sujet voc:propriété objet. }
ORDERBY élément à afficher
```

- LIMIT permet de limiter le nombre d'éléments à afficher :

```
PREFIX voc:URIVOC
SELECT élément à afficher
WHERE { sujet voc:propriété objet. }
LIMIT Nb d'élément à afficher
```

- OFFSET permet de décaler les éléments à afficher du nombre indiqué :

```
PREFIX voc:URIVOC
SELECT élément à afficher
WHERE { sujet voc:propriété objet . }
OFFSET Valeur du décalage
```

Requête SELECT : ORDERBY

Requête

```
PREFIX voc:http://www.monvoc.fr
SELECT ?prenom
WHERE { ?auteur voc:prénom ?prenom.}
ORDERBY ?prenom
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom
Brian
Frank

Requête SELECT : LIMIT

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom
WHERE { ?auteur voc:prénom ?prenom. }
LIMIT 1
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom
Frank

Requête SELECT : OFFSET

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom
WHERE { ?auteur voc:prénom ?prenom. }
OFFSET 1
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom
Brian

Requête SELECT : contraintes OPTIONAL et FILTER

- OPTIONAL :
  - permet d'utiliser l'information requise si elle est présente et de ne pas éliminer les solutions ou elle est absente (très important pour le WEB sémantique)
  - sera utilisé pour des jointures à gauche (cf plus loin)

- FILTER permet de filtrer les données selon des critères prédéfinis :

```
PREFIX voc:URIVOC
SELECT élément à afficher
WHERE { sujet voc:propriété objet .
FILTER élément de filtre sur les données }
```

Lié à FILTER :

- BOUND() teste si une variable est liée à une valeur
- REGEX() pour test d'expression régulière

Requête SELECT : FILTER

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom ?age
WHERE { ?auteur voc:prenom ?prenom.
        ?auteur voc:age ?age.
FILTER (?age> 40)
      }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)

Résultat :

Prénom	Age
Frank	70

Requête SELECT : Expression régulière

- Permet de filtrer les données selon une expression régulière :

```
PREFIX …
SELECT élément à afficher
WHERE { sujet voc:propriété objet .
      FILTER regex (variable, donnée)
      élément de filtre sur les données
}
```

- Exemple :

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?titre
WHERE {?livre voc:titre ?titre.
       FILTER regex (?titre, "^L'homme" )
}
```

Résultat :

Titre
L'homme de deux mondes

Requête SELECT : Jointures classique et à gauche

- **Jointure classiques :**

```
PREFIX …
SELECT élément à afficher
WHERE { sujet voc:propriété objet .
        sujet2 voc:propriété2 objet2
      }
```

  - le point correspond à une jointure entre 2 graphes :

- **Jointure à gauche ou optionnal :**

```
PREFIX …
SELECT élément à afficher
WHERE { sujet voc:propriété objet .
        OPTIONAL {sujet2 voc:propriété2 objet2 }
      }
```

  - L'utilisation d'**OPTIONAL** correspond à une **jointure à gauche**.
  - Les **éléments du graphe de gauche sont conservés** même s'ils ne répondent pas à la clause présente dans la partie optionnelle (à droite)

Requête SELECT : jointure classique

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom ?age
WHERE { ?auteur voc:prenom ?prenom.
        ?auteur voc:age ?age .
      }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)
(#auteur3, nom, Asinov)
(#auteur3, prenom, Isaac)

Résultat :

Prénom	Age
Frank	70
Brian	30

Requête SELECT : jointure à gauche

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom ?age
WHERE { ?auteur voc:prenom ?prenom.
        OPTIONAL {?auteur voc:age ?age} .
      }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur3, nom, Asinov)
(#auteur3, prenom, Isaac)

Résultat :

Prénom	Age
Isaac	
Brian	30
Frank	70

Requête SELECT : ADDITION, UNION et NEGATION

- ADDITION / UNION permet de faire l'addition / l'union de 2 graphes RDF :

```
PREFIX …
SELECT élément à afficher
WHERE { {sujet voc:propriété objet }
   UNION {sujet2 voc:propriété2 objet2 } .
        }
```

- FILTER !bound (variable) permet de ne garder les éléments qui ne possèdent pas de valeur pour la variable testée (négation) :

```
PREFIX …
SELECT élément à afficher
WHERE { sujet voc:propriété objet .
        FILTER !bound (variable)
        }
```


Requête SELECT : UNION

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom
WHERE { ?auteur voc:prenom ?prenom .
    UNION {?auteur voc:firstname ?prenom}.
      }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, **firstname**, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)
(#auteur3, nom, Asinov)
(#auteur3, prenom, Isaac)

Résultat :

Prénom
Frank
Brian

Requête SELECT : Négation

Requête

```
PREFIX voc:http://www.monvoc.fr
PREFIX xsd:http://www.w3.org/2001/XMLSchema#

SELECT ?prenom
WHERE { ?auteur voc:prenom ?prenom.
        ?auteur voc:age ?age.
        FILTER bound(?age)
      }
```

Base RDF

(#livre1, auteur, #auteur1)
(#livre2, auteur, #auteur1)
(#livre2, auteur, #auteur2)
(#livre1, titre, Dune)
(#livre2, titre, L'homme de deux mondes)
(#auteur1, nom, Herbert)
(#auteur1, prenom, Frank)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur2, fils, #auteur1)
(#auteur1, age, 70)
(#auteur2, age, 30)
(#auteur2, nom, Herbert)
(#auteur2, prenom, Brian)
(#auteur3, nom, Asinov)
(#auteur3, prenom, Isaac)

Résultat :

Prénom
Brian
Frank

!bound : l'inverse.

3. Conclusion

- **Mise en œuvre de SPARQL**
- **Exécution d'une requête SPARQL**
- **Force et limites et évolution de SPARQL**

Mise en œuvre de SPARQL

- **Editeur leger de requêtes**
  - **Twinkle (http://www.iro.umontreal.ca/~lapalme/ift6282/Twinkle.html)**
  et (http://www.ldodds.com/projects/twinkle/)
  - …
- **Moteurs RDF/SPARQL :**
  - **Jena** (HP Lab.)
  - **Corese** (INRIA Sophia - O. Corby)
  - **Redland** RDF framework (C)
  - **Snorql**
  - …
- **Point SPARQL :** Un point SPARQL est un service web permettant d'effectuer des requêtes sur un graphe RDF. Exemple :
  - http://data.bnf.fr/sparql
  - http://dbpedia.org/sparql
  - http://data.linkedmdb.org/sparql

- …

Exécution d'une requête SPARQL

Source : Remi Gilleron

- **Calcul d'une requête SPARQL :**
  1. le processeur construit le graphe correspondant aux sources du FROM
  2. il construit les tuples de variables qui satisfont dans le graphe les conditions décrites dans le pattern donné dans le WHERE
  3. si le pattern est complexe on effectue une intersection des solutions ou une union selon le cas
  4. enfin, il restreint les résultats aux variables précisées dans le SELECT, ou plus généralement dans l'entête de la requête
- **Remarques :**
  - attention, cet ordre est important à retenir pour écrire vos requêtes SPARQL, n particulier si on utilise des négations
  - l'ordre des réponses ne peut pas être connu à l'avance
  - les résultats peuvent être de très grande taille : utiliser LIMIT et OFFSET dans les tests
  - on utilise des requêtes SPARQL pour explorer un entrepôt de triplets : quelles propriétés utilisées ? quels sujets possibles pour un prédicat ? quels objets possibles pour un prédicat ?

Conclusion sur SPARQL

- SPARQL est intégré dans diverses plates-formes de WEB sémantique (comme ARQ dans JENA d'APACHE)
- SPARQL est utilisé dans diverses implantations existantes de Triples-stores (Virtuoso, …)
- SPARQL est aussi utilisé pour exploiter des entrepôts RDF gérés dans une base de données relationnelle avec un interfaçage entre le SGBDR et SPARQL
- SPARQL est une norme qui **évolue** :
  - introduction de EXISTS (remplace BOUND),
  - introduction de DELETE et UPDATE
  - introduction de patterns élaborés avec des chemins (utilisation de + et *) pour enrichir la description par triplets
  - …

4. Aide mémoire SPARQL

-

SPARQL 1.1 - Aide-mémoire (1)

Source : Guy Lapalme inspiré de SPARQL By Example: The Cheat Sheet de Lee Feigenbaum

Bases

URI

Complet	`<http://nom.de.domaine/URI/complet#id>`

Abrégé	`PREFIX ex: <http://nom.de.domaine/prefixe#>`
	`ex:toto → <http://nom.de.domaine/prefixe#toto`
	`a → rdf:type`

Raccourcis	`?coll rdfs:member ?el →` éléments d'une collection

Variable

URI ou Littéral	`?var ?une_autre_var $v`
Noeuds vides	`_:id []`

Patron de triplets

Match complet	`ex:machin ex:numero "45692"`
Match avec une variable	`?truc ?ex:numero "?valeur`
Match complet	`ex:machin ?prop ?valeur`

Commentaire

```
# tout ce qui suit un dièse sur une ligne
```

Littéral

Simple	`"bonjour"` ou `'bonjour'`
Simple avec étiquette de langue	`"hello"@en`
Typé	`"25"^^xsd:integer`

Raccourcis	`true → "true"^^xsd:boolean`
	`7 → "7"^^xsd:integer`
	`3.1416 → "3.1416"^^xsd:decimal`

Noeud vide

Notation simplifiée	Notation complète
`[ex:p "o"] .`	`_:b1 ex:p "o" .`
`[ex:p "o"] ex:q ex:o2 .`	`_:b2 ex:p "o".` `_:b2 ex:q ex:o2 .`
`ex:s ex:p [ex:p1 ex:o].`	`ex:s ex:p _:b3 .` `_:b3 ex:p1 ex:o .`
`[ex:p ex:o;ex:p1 "o1"].`	`_:b4 ex:p ex:o .` `_:b4 ex:p1 "o1" .`

SPARQL 1.1 - Aide-mémoire (2)

Source : Guy Lapalme inspiré de SPARQL By Example: The Cheat Sheet de Lee Feigenbaum

Forme d'une requête

```
PREFIX ex: <http://exemple.com/ressource> # Déclarations de préfixe
...
SELECT vars                               # type de requête (ou CONSTRUCT, ASK, DESCRIBE)
FROM <...>                                # ensemble de données
WHERE {                                   # patron de graphe cherché dans la source
    ...
}
GROUP BY ...                              # modificateurs optionnels
HAVING ...
ORDER BY ...
LIMIT ...
OFFSET ...
```

Type de requêtes

SELECT

Variables et expressions	`SELECT ?s ?prop (1000*?v as ?total)`
Toutes les variables	`SELECT *`
Ne garder que les valeurs différentes	`SELECT DISTINCT ?v`

CONSTRUCT

Créer des triplets `CONSTRUCT { ?s ex:prop ?v }`

ASK

Vérifier s'il y a un triplet qui concorde retourne `true` ou `false` — `ASK {?s ex:prop ?v}`

DESCRIBE

Donne des triplets à propos d'une ressource — `DESCRIBE <...uri...>`

SPARQL 1.1 - Aide-mémoire (3)

Source : Guy Lapalme inspiré de SPARQL By Example: The Cheat Sheet de Lee Feigenbaum

Combinaisons de graphes

Soient A et B deux graphes:

`A . B`	Réunir les solutions de A et de B en combinant les valeurs de variables communes.
`A OPTIONAL { B }`	Valeurs optionnelles: réunir les résultats de A et B par leur valeurs en commun. Garder toutes les solutions de A qu'il y ait une solution dans B ou non.
`{ A } UNION { B }`	Union: réunir toutes les solutions de A et celles de B.
`A MINUS { B }`	Différence: trouver les solutions de A et celles B et ne garder que les solutions de A qui ne sont pas compatibles avec celles de B
`A. {SELECT ?cvars WHERE { C } } B.`	Sous-requête: les résultats (`?cvars`) sont combinés avec ceux de A et de B.

SPARQL 1.1 - Aide-mémoire (4)

Source : Guy Lapalme inspiré de SPARQL By Example: The Cheat Sheet de Lee Feigenbaum

Filtres

Un filtre est de la forme `FILTER` *`expression`* qui apparaît dans un modèle de graphe où *expression* est composée d'une combinaison des fonctions et opérateurs suivants:

Conversion	`xsd:integer, xsd:string`
Arithmétique	`+, -, *, /`
Logique	`!, &&, ||`
Comparaison	`=, !=, <, <=, >, >=`
test	`isURI, isBlank, isLiteral, bound, sameTerm, langMatches, regex`
mathématique	`ABS, ROUND, CEIL, FLOOR, RAND`
conversion	`STR, LANG, DATATYPE, URI, STRDT, STRLANG`
conditionnel	`IF, COALESCE`
chaînes	`REGEX, STRLEN, SUBSTR, UCASE, LCASE, STRSTARTS, STRENDS, CONTAINS, CONCAT, ENCODE_FOR_URI, REPLACE`
constructeurs	`URI, BNODE, STRDT, STRLANG`
date	`NOW, YEAR, MONTH, DAY, HOURS, MINUTES, SECONDS, TIMEZONE`

Aggrégation

Appliquer les opérations suivantes:

- Séparer les résultats en groupes selon l'expression donné par `GROUP BY`.
- Évaluer les projections et les fonctions d'agrégation utilisées dans le `SELECT` pour n'avoir qu'un seul résultat par groupe.
- Filtrer les valeurs agrégées par l'expression `HAVING`
- Fonctions d'agrégation: `COUNT, SUM, AVG, MIN, MAX, SAMPLE, GROUP_CONCAT`