SPARQL (SPARQL Protocol and RDF Query Language)

1. Introduction

RDF est un format de données en graphe dirigé et étiqueté utilisé pour représenter des informations sur le Web. Cette spécification définit la syntaxe et la sémantique du langage de requête SPARQL pour RDF. SPARQL permet d'exprimer des requêtes à travers des sources de données diverses, que les données soient stockées nativement en tant que RDF ou visualisées comme RDF via une couche intermédiaire. SPARQL offre des capacités pour interroger des motifs de graphe obligatoires et facultatifs ainsi que leurs conjonctions et disjonctions. Les résultats des requêtes SPARQL peuvent être des ensembles de résultats ou des graphes RDF. Cette technologie joue un rôle essentiel dans la récupération et la manipulation d'informations à partir de sources de données liées sur le Web sémantique.

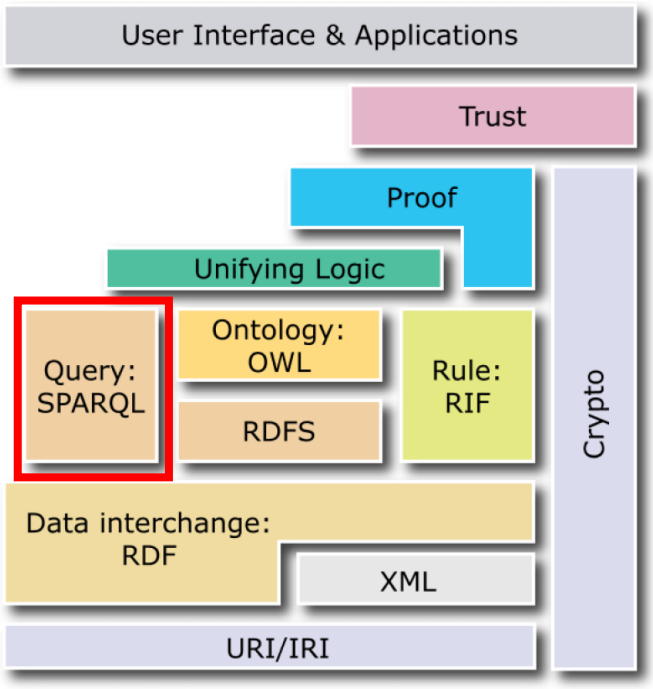

Figure 1 Pile des standards W3C [1]

2. SPARQL

SPARQL, acronyme de "SPARQL Protocol and RDF Query Language," est à la fois un langage de requête et un protocole standardisé utilisés dans le domaine du Web sémantique pour interroger et manipuler des données stockées au format Resource Description Framework (RDF). Le RDF est une structure de données en forme de graphe dirigé, où les informations sont représentées sous la forme de triplets comprenant un sujet, un prédicat et un objet. SPARQL permet d'exprimer des requêtes complexes sur des données RDF, facilitant ainsi la récupération et l'analyse d'informations dans des ensembles de données liées sur le Web.

- Un standard du W3C
  - Recommandation SPARQL 1.0 - Janvier 2008,
  - Recommandation SPARQL 1.1 - Mars 2013

http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/ –

- Un langage de requête pour accéder à un graphe RDF (Spécification du langage de requête SPARQL), inspiré de SQL
- Un protocole pour soumettre des requêtes via HTTP GET, HTTP POST ou SOAP (Protocole SPARQL pour RDF)
- Différents formats pour les résultats XML, JSON, etc.

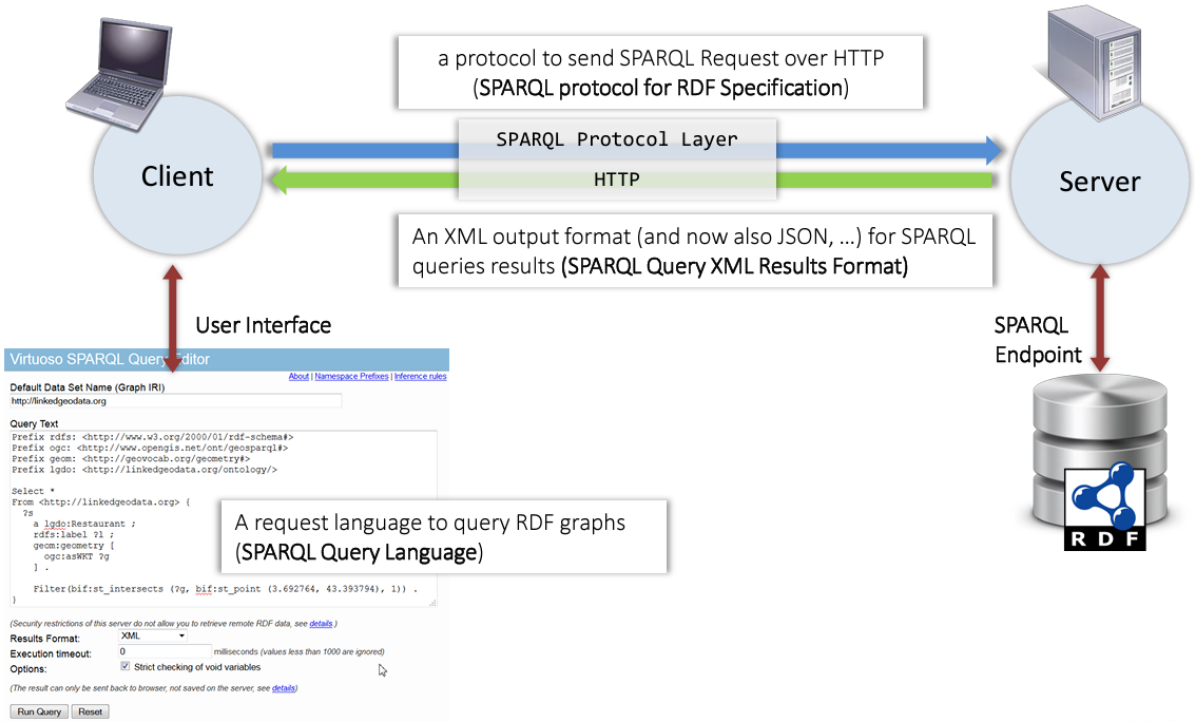

Figure 2 Aperçu de l'utilisation de SPARQL [1]

3. Types de requêtes SPARQL

SPARQL possède quatre formes de requête. Ces formes de requête utilisent les solutions issues de la correspondance de motifs pour former des ensembles de résultats ou des graphes RDF. Les formes de requête sont les suivantes :

- **SELECT**

Renvoie toutes, ou une partie, des variables liées dans une correspondance de motif de requête.

- **CONSTRUCT**

Renvoie un graphe RDF construit en substituant des variables dans un ensemble de modèles de triplets.

- **ASK**

Renvoie un booléen indiquant si un motif de requête correspond ou non.

- **DESCRIBE**

Renvoie un graphe RDF décrivant les ressources trouvées.

4. Forme générale de requête SPARQL

La forme générale d'une requête SPARQL (SELECT) est la suivante :

SELECT [DISTINCT] ?var_1 ?var_2 ... ?var_n

WHERE {

pattern$_1$.

pattern$_2$.

...

pattern$_n$ }

- Les « patterns » sont des triplets en format **TURTLE** (et non pas en RDF/XML)
- Les variables apparaissent dans les patterns et qui décrivent les conditions que les données doivent satisfaire pour être incluses dans les résultats
- Les variables (bindings) correspondant a (?var_1 ?var_2 ... ?var_n) qui apparaissent dans les résultats.
- Le mot-clé DISTINCT est optionnel et est utilisé pour s'assurer que les résultats ne contiennent pas de doublons.

La forme générale d'une requête SPARQL est généralement composée de trois parties : prologue, head et body.

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

a. Prologue d'une requête SPARQL

- BASE est une déclaration qui est utilisée pour spécifier une URI de base pour la résolution des URI relatives dans la requête. Cela signifie que si vous utilisez des URI relatives dans votre requête, elles seront résolues par rapport à cette URI de base
  - Une seule base par requête.
  - Exemple : BASE <http://example.org/data/>
- PREFIX suivi d'une déclaration d'un raccourci pour des espaces de nom.
  - Il est courant d'avoir plusieurs préfixes.
  - Exemple : PREFIX foaf: <http://xmlns.com/foaf/0.1/>

b. Head d'une requête SPARQL

Contient l'une des formes possibles de requêtes SELECT, CONSTRUCT, ASK et DESCRIBE suivi d'une liste de variables dont on souhaite connaître les valeurs répondant à la requête.

c. Body d'une requête SPARQL

Peut contenir les clauses :

- **FROM** (optionnel) permet de spécifier différents graphes de triplets dans lesquels il faut chercher. Si la clause FROM n'est pas spécifiée dans la requête, le processeur SPARQL suppose qu'un graphe a été spécifié à un niveau supérieur ou par défaut. Cela signifie que la recherche s'effectuera dans le graphe par défaut ou dans tous les graphes disponibles, selon le comportement du processeur SPARQL spécifique.
- **WHERE** suivi d'une déclaration de pattern (motif) pouvant contenir : plusieurs patterns, plusieurs patterns de base, des filtres, des unions, des présences optionnelles, etc.
- **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

5. Requêtes SELECT

Une requête SELECT à la forme suivante :

```
SELECT what you want

FROM where you want

WHERE {conditions (as you want)}
```

Les variables dans SPARQL sont représentées par « ? » :

```
?x rdf:type ex:Person
```

La spécification du pattern se situe dans la clause WHERE. Une requête SELECT récupérant tous les triplets.

```
SELECT ?subject ?predicate ?object WHERE {?subject ?predicate ?object}
```

Un pattern est une **conjonction** de triplets

```
SELECT ?x ?name
WHERE {
        ?x rdf :type foaf:Person.
        ?x foaf:name ?name.
}
```

Les mêmes abréviations utilisées en Turtle peuvent être utilisés en SPARQL.

Abréviations	Origine

SELECT ?x
WHERE {
?x **a** foaf:Person **;**
foaf:name ?name**;**
ex:author ?y.
}
	SELECT ?x
	WHERE {
	?x rdf :type foaf:Person.
	?x foaf:name ?name.
	?x ex:author ?y.
	}

Nous pouvons commencer à construire une requête en recherchant des valeurs littérales marquées par la langue qui sont les objets des propriétés rdfs:label :

```
SELECT *
WHERE
   {
     ?x rdfs:label "John"@en
   }
```

Nous pouvons également spécifier le type de données des littéraux

```
SELECT *
WHERE
   {
     ?x foaf:name "John"@en;
        foaf:age “21”^^xsd:interger.
   }
```

Exercice :

Ecrire une requête SPARQL pour sélectionner la ressource dont la propriété rdfs:label est « Algeria » en anglais, ainsi que sa capitale dbo:capital .

Réponse :

```
SELECT *
WHERE
{
        ?cityName rdfs:label "Algeria"@en;
        dbo:capital ?capitale .
}
```

a. Optional

OPTIONAL nous permet de spécifier une contrainte de la requête qui n'est pas obligatoire et qui est donc optionnelle.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?name
WHERE {
 ?person foaf:homepage <http://example.org/asma> .
 OPTIONAL { ?person foaf:name ?name . }
}
```

La déclaration OPTIONAL est le motif facultatif. Il permet de spécifier une partie d'un motif de graphe qui n'est pas obligatoire dans la réponse de la requête. Dans l'exemple ci-dessous, il y a une partie obligatoire qui est la variable de personne avec la page d'accueil Asma, puis il y a une partie facultative de la requête, où nous aimerions avoir le nom de la personne ressource, mais si la requête ne trouve pas de nom dans le graphe cible, elle ne va pas échouer, nous aurons toujours un résultat. Ainsi, parmi les résultats de cette requête, nous pouvons avoir des valeurs, ou pas, pour la variable de nom et donc dans certains résultats, la variable de nom peut être non liée, cela signifie que cette variable peut ne pas avoir de valeur dans les résultats.

b. Union

L'UNION nous permet d'avoir une union des résultats de patterns. Ces motifs alternatifs nous permettent de calculer l'union des résultats de motifs de graphe. Dans cet exemple, nous recherchons des ressources ayant soit la page d'accueil Asma, soit la page d'accueil Asma_bn. C'est l'équivalent d'un OR. Ainsi, la première partie peut réussir, ou la deuxième partie, ou les deux parties de l'union peuvent réussir, et dans le résultat de la requête, nous aurons l'union des résultats.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?name
WHERE {
 ?person foaf:name ?name .
  {
   ?person foaf:homepage <http://example.org/asma>.
  }
  UNION
  {
   ?person foaf:homepage <http://example.org/asma_bn>.
  }
}
```

c. Négation

La négation supprime le/les résultat(s) qui correspond à un pattern. La négation permet de récupérer des ressources qui ne correspondent pas à un motif de graphe spécifique. C'est l'instruction "minus", donc dans cet exemple, nous recherchons des ressources X de type Personne qui ne sont pas de type Homme. Ainsi, dans la clause "minus", nous écrivons X de type Homme et cela sera exclu des résultats de la requête, mettant en œuvre la négation.

```
PREFIX ex: <http://www.example.>
SELECT ?x
WHERE {
   ?x a ex:Person
   MINUS { ?x a ex:Man }
}
```

d. Variables à valeurs prédéfinis

En utilisant VALUES nous pouvons définir des résultats qui font partie des liaisons. Il est possible d'avoir des valeurs de variables prédéfinies de manière à ce qu'une requête SPARQL se concentre sur des valeurs spécifiques. Dans cet exemple, la variable "name" prendra comme valeurs possibles Asma, Anis et Amira, et aucune autre valeur, nous permettant ainsi de concentrer l'exécution de la requête SPARQL sur des valeurs prédéfinies spécifiques.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?name
WHERE {
 ?person foaf:name ?name .
}
VALUES ?name  { "Asma" "Anis" "Amira" }
```

e. Binding de valeurs

Nous pouvons utilisés BIND pour avoir des résultats où les liaisons sont calculées. Il est possible de spécifier des liaisons de variables où les variables sont liées au résultat d'une expression. Ainsi, dans cet exemple, la variable "name" est liée aux résultats de la concaténation du prénom et du nom. Cela crée une nouvelle valeur et une nouvelle variable dans la partie WHERE, et

cette variable peut être un résultat de la requête. Ici, dans la clause SELECT, nous voyons la variable "name", et donc la variable "name" qui est calculée par une expression, fera partie du résultat de la requête.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://www.example.abc#>
SELECT ?person ?name
WHERE {
 ?person ex:fname ?fname ;
   ex:lname ?lname .
 BIND (concat(?fname, ?lname) AS ?name)
}
```

f. Distinct Result

DISTINCT permet de conserver uniquement des résultats distincts, c'est-à-dire ne conserver qu'une seule occurrence de résultats similaires avec les mêmes valeurs pour les mêmes variables, et donc nous utilisons un mot clé distinct dans la clause SELECT. Dans cet exemple, nous écrivons "SELECT DISTINCT ?name", ce qui signifie que nous voulons avoir une seule occurrence de chaque valeur de name dans les résultats de la requête.

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

g. Filter

Il est possible de filtrer les résultats d'une requête à l'aide d'une expression évaluable qui définit des contraintes sur les valeurs des variables. La clause FILTER qui fait partie du WHERE. La clause FILTER permet d'exprimer une condition supplémentaire qui sera évaluée [2].

```
PREFIX ex: <http://inria.fr/schema#>
SELECT ?person ?name
WHERE {
 ?person rdf:type ex:Person ;
    ex:name  ?name ;
    ex:age   ?age .
 FILTER (xsd:integer(?age) >= 18)
}
```

Dans cet exemple, nous recherchons des ressources de type personne qui ont un âge, et nous introduisons une contrainte supplémentaire selon laquelle l'âge doit être supérieur ou égal à dix-huit ans. Ainsi, nous voulons trouver des personnes d'au moins dix-huit ans. Nous utilisons le mot-clé FILTER dans la partie WHERE, et à l'intérieur du filtre, nous avons une expression qui sera évaluée par l'interpréteur SPARQL. Cette expression renverra soit vrai, soit faux, c'est-à-dire

une valeur booléenne. Si le résultat est vrai, il sera conservé dans le résultat de la requête ; s'il est faux, il sera exclu du résultat de la requête. Nous pouvons voir l'expression où il est dit que la conversion en entier de l'âge doit être supérieure ou égale à dix-huit. Ainsi, nous utilisons la fonction xsd:integer pour convertir la valeur de l'âge en un entier, car elle pourrait être une chaîne de caractères et nous voulons obtenir sa valeur entière.

- **Test values**

Il est possible de tester des valeurs et de comparer des constantes, des variables et des expressions en utilisant les comparateurs habituels.

- Comparateurs : <, >, =, <=, >=, !=
- Opérateurs booléens : &&, ||, !
- Opérateur arithmétique : + * - /
- Tests de liaison de variables : isURI(?x) Est-ce une URI ? , isBlank(?x) Est-ce un nœud vide ?, isLiteral(?x) Est-ce un littéral ?, bound(?x) La variable a-t-elle une valeur ?
- Expressions régulières : regex(?x, "A.*") cette exemple correspond à toute chaîne qui commence par "A" et est suivie par n'importe quel nombre de caractères, y compris une chaîne vide (car .* peut correspondre à zéro caractère).
- Accès aux attributs/valeurs lang(), datatype(), str()
- Conversion de types ((re-)typage) xsd:integer(?x)

- **Autres fonctions**

Il existe un ensemble de fonctions pour les chaînes de caractères et les valeurs littérales.

- CONTAINS($literal_1$, $linteral_2$) permet de tester si une chaîne de caractères contient une autre.
- STRSTARTS($literal_1$, $literal_2$) si une chaîne de caractères commence par une autre,
- STRSENDS($literal_1$, $literal_2$) ou se termine par une autre chaîne de caractères.
- STRDT(value, type) permet de construire une valeur littérale avec un type de données.
- STRLANG(value, lang) Il est possible de créer une littérale avec une langue (tag).
- CONCAT(($literal_1$, $linteral_2$, … $literal_n$) permet de concaténer des littéraux.
- SUBSTR(literal, start [,lenght]) permet d'extraire une partie d'une chaine de caractère.

- **Expressions contidionnelles**

Nous pouvons utiliser les expressions conditionnelles habituelles, **IF… THEN… ELSE…** dans le langage de filtre, ce qui signifie que nous pouvons tester une expression de manière conditionnelle.

Ainsi, dans l'exemple ci-dessous, nous testons d'abord si le tag de langue du nom est le français. Si c'est le cas alors nous évaluons l'expression d'âge supérieur à dix-huit. Et si la langue ne correspond pas au français, alors nous évaluons l'expression d'âge supérieur à vingt et un.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT * where {
 ?x foaf:name ?name ; foaf:age ?age .
 FILTER ( if (langMatches(lang(?name), "FR"),
        ?age >= 18, ?age >= 21) )
}
```

h. ORDER, LIMIT et OFFSET

Il est possible d'ordonner les résultats et de limiter leur nombre. Par exemple, nous pouvons avoir les résultats dans l'ordre alphabétique des noms. Il est également possible de limiter le nombre de résultats à vingt, peut-être s'il y a trop de résultats. De plus, il est possible de spécifier si nous voulons commencer les résultats à partir du onzième en utilisant OFFSET.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:name ?name . }
ORDER BY ?name
LIMIT 20
OFFSET 10
```

i. Résultats agrégés

- Nous pouvons agréger les résultats en utilisant les fonctions comme count, sum, min, max, avg.
- Nous pouvons regrouper des résultats en utilisant GROUP BY
- Nous pouvons filtrer les résultats agrégés en utilisant HAVING

```
PREFIX ex: <http://example.org>
SELECT ?student
WHERE { ?student ex:score ?score . }
GROUP BY ?student
HAVING(AVG(?score) >= 10)
```

6. Requête ASK

La requête ASK vérifie l'existence d'une solution en revoyant TRUE ou FALSE, contrairement à la requête SELECT qui énumère toutes les solutions. Permet de vérifier qu'il y a au moins une réponse.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK { ?person foaf:age ?age .
FILTER (?age > 30)
```

```
}
```

7. Requête CONSTRUCT

La requête CONSTRUCT permet de construire un graphe de résultats. C'est-à-dire qu'il nouveau RDF graphe sera construit contenant le résultat d'une requête. Les clauses WHERE et FILTER fonctionnent de la même manière que dans la forme SELECT.

```
PREFIX ex: <http://example.org/>

CONSTRUCT {
  ?country ex:hasCapital ?capital .
}
WHERE {
  ?country ex:hasPopulation ?population .
  FILTER (?population > 10000000)
}
```

Cette requête sélectionne les pays ayant une population supérieure à 10 000 000 et construit des triplets avec le prédicat `ex:hasCapital` pour ces pays.

8. Requête DESCRIBE

La requête DESCRIBE renvoie un graphe RDF, en fonction de la configuration du processeur de requêtes pour le retour. La spécification SPARQL indique : "les informations utiles que le service possède sur une ressource". Cela aide à comprendre le contexte des ressources renvoyées [3].

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX ex: <http://example.com/>
DESCRIBE ex:karen ?friend {  ex:karen foaf:knows ?friend .  }
```

9. Résumé

- SPARQL est un langage de requête et un protocole pour interroger et manipuler des données RDF. C'est un standard du W3C qui joue un rôle essentiel pour accéder aux données liées du web sémantique.
- Il existe 4 types de requêtes SPARQL : SELECT pour sélectionner des variables, CONSTRUCT pour construire un nouveau graphe RDF, ASK pour avoir une réponse booléenne, et DESCRIBE pour obtenir une description RDF des ressources trouvées.
- La forme générale d'une requête SPARQL se compose d'un prologue avec des préfixes, d'un head qui définit le type de requête, et d'un body qui contient les patterns de triplets et les conditions.

- On peut utiliser des fonctions et filtres pour restreindre les résultats, faire des unions, des négations, ordonner, limiter, faire des agrégations et group by.
- Les résultats peuvent être formatés en XML, JSON, etc.
- SPARQL permet d'exprimer des requêtes complexes sur des sources de données RDF, ce qui facilite l'analyse du web sémantique.

References

[1] P. G. –. Z. -LIG-Steamer, «INTERROGERLESDONNÉESAVECSPARQL,» [En ligne]. Available: https://lig-membres.imag.fr/genoud/teaching/coursSW/cours/pdf/WS_Cours04_SPARQL.pdf.

[2] C. F.-Z. O. C. Fabien Gandon, Le web sémantique, Paris: Dunod, 2012.

[3] P. G. –. Z. -LIG-Steamer, «INTERROGER LES DONNÉES AVEC SPARQL,» [En ligne]. Available: https://lig-membres.imag.fr/genoud/teaching/coursSW/cours/pdf/WS_Cours04_SPARQL.pdf.

[4] «FOAF Vocabulary Specification,» W3C, [En ligne]. Available: http://xmlns.com/foaf/0.1/.

[5] F.-R. Chaumartin, «Antelope, a NLP platform for extracting meaning from text: theory and applications of the syntax-semantics interface,» Thèse de doctorat. Université Paris-Diderot-Paris VII, Paris, France, 2012.