Examen MIF04 - Gestion de données pour le Web - session 1 - 21 janvier 2021
Durée : 2h
Documents autorisés

Exercice 1: XML et RDF **(5 points)**

On considère une base de données contenant un ensemble de documents XML conformes à la DTD suivante :

```
1  <!DOCTYPE notices [
2  <!ELEMENT notices (notice+)>
3  <!ELEMENT notice (titre,auteur+,edition+)>
4  <!ATTLIST notice id ID #REQUIRED>
5  <!ELEMENT titre (#PCDATA)>
6  <!ELEMENT auteur (nom,prenom)>
7  <!ATTLIST auteur id ID #REQUIRED>
8  <!ELEMENT nom (#PCDATA)>
9  <!ELEMENT prenom (#PCDATA)>
10 <!ELEMENT edition (annee, editeur)>
11 <!ATTLIST editeur id ID #REQUIRED>
12 <!ELEMENT annee (#PCDATA)>
13 <!ELEMENT editeur (#PCDATA)>
14 ]>
```

À chaque document de cette base de données, on associe un graphe RDF comme suit :

— Une notice correspond à un noeud ayant pour URI `http://biblio.org/notice/`*`xyz`* où *`xyz`* est l'`id` de la notice.

— Une édition correspond à un noeud ayant pour URI `http://biblio.org/edition/`*`xyz`* où *`xyz`* est l'`id` de l'édition.

— Un auteur correspond à un noeud ayant pour URI `http://biblio.org/auteur/`*`xyz`* où *`xyz`* est l'`id` de l'auteur.

— Le titre d'une notice est indiqué par la propriété `http://www.w3.org/2000/01/rdf-schema#label` .

— Une notice est associée à chacun de ses auteurs par la propriété `http://biblio.org/prop/auteur` .

— Une notice est associée à chacune de ses éditions par la propriété `http://biblio.org/prop/edition` .

— Le nom d'un auteur est indiqué par la propriété `http://biblio.org/prop/nom`, son prénom par la propriété `http://biblio.org/prop/prenom` .

— L'année d'une édition est indiquée par la propriété `http://biblio.org/prop/annee` et son éditeur par la propriété `http://biblio.org/prop/editeur` .

Question 1.1: On considère la requête XQuery suivante :

```
<resultats>
    { for $no in //notice, $annee in $no/edition/annee
      return
        <ouvrage>
            {$no/titre}
            {$annee}
        </ouvrage>
    }
</resultats>
```

Donner une requête SPARQL permettant d'obtenir les mêmes informations mais en interrogeant le graphe RDF associé au document.

Question 1.2:

On rappelle la syntaxe XML pour représenter les graphes RDF. Le préfixe `rdf` représente l'espace de nommage `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. Le graphe est décrit dans une balise `rdf:RDF`, qui contient un élément `rdf:Description` par nœud sujet du graphe. L'attribut `rdf:about` de `rdf:Description` contient l'URI du nœud sujet représenté. Pour chaque triplet ayant pour sujet ce nœud, l'élément `rdf:Description` contient un élément enfant dont le nom est tel que la concaténation de son espace de nommage et de son nom local donne l'URI de la propriété du triplet. Enfin l'objet du triplet est soit du texte à l'intérieur de l'élément enfant (pour les objet qui sont des littéraux), soit une URI donnée par la valeur de l'attribut `rdf:resource`.

Par exemple, le graphe suivant :

```
<http://biblio.org/notice/n5> <http://biblio.org/prop/auteur>
                                        <http://biblio.org/auteur/a2>.
<http://biblio.org/auteur/a2> <http://biblio.org/prop/nom> "Alice".
```

peut être représenté en XML par le document suivant :

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:bibrel="http://biblio.org/prop/">
    <rdf:Description rdf:about="http://biblio.org/notice/n5">
        <bibrel:auteur rdf:resource="http://biblio.org/auteur/a2"/>
    </rdf:Description>
    <rdf:Description rdf:about="http://biblio.org/auteur/a2">
        <bibrel:nom>Alice</bibrel:nom>
    </rdf:Description>
</rdf:RDF>
```

Écrire une requête XQuery qui permet de transformer un document de la base de donnée en un document qui représente le graphe RDF qui lui est associé en syntaxe XML. On supposera qu'il existe une fonction XQuery `concat` qui concatène des concatène des chaînes de caractères. Par exemple, `concat("ab","cd","ef")` donnera `"abcdef"`. On supposera ques les variables suivantes sont prédéfinies :

```
let $rdf := "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    $rdfs := "http://www.w3.org/2000/01/rdf-schema#",
    $biblio := "http://biblio.org/"
```

Exercice 2: Map/Reduce (5 points)

On considère une plateforme de diffusion de videos. Une collection MongoDB représente pour chaque video le nombre de vues de cette video chaque jour.

Un exemple de document de cette collection est présenté ci-dessous :

```
{
    "_id": 2345,
    "titre": "Des chats trop mignons",
    "vues": [
        { "jour": "2021-01-19", "nombre": 3451 },
        { "jour": "2021-01-20", "nombre": 5678 },
        { "jour": "2021-01-21", "nombre": 6789 }
    ]
}
```

Écrire des fonctions `maptk`, `reducetk` et au besoin `finalizetk` à passer à la commande map/reduce de MongoDB afin de calculer pour chaque jour quelles sont les 10 vidéos (`_id` et `titre`) ayant eu le plus de vues.

Remarques :

- On suppose que la plateforme possède plusieurs millions de vidéos, le calcul devra donc se faire principalement lors de l'étape de *reduce* et le phénomène de *re-reduce* se produira nécessairement.
- Les tableaux Javascript possèdent une méthode `sort` qui prend en argument une fonction de comparaison. Une fonction de comparaison prend deux arguments et indique leur position respective dans le tableau trié. Elle renvoie un nombre négatif si son premier argument doit être placé avant son deuxième argument, un nombre positif si c'est le deuxième qui doit être placé avant le premier et enfin 0 si les deux arguments sont interchangeables.

Exercice 3: Déductions en RDF-S (5 points)

On considère le graphe RDF G suivant, représenté graphiquement en figure 1 page 4 :

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xs: <http://www.w3.org/2001/XMLSchema> .
@prefix jp: <http://jouets.example.com/p/> .
@prefix jt: <http://jouets.example.com/t/> .
@prefix jj: <http://jouets.example.com/j/> .
@prefix js: <http://jouets.example.com/s/> .
```

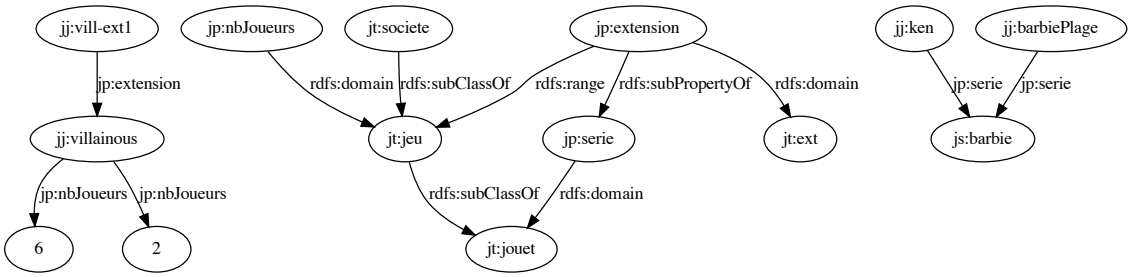

FIGURE 1 – Représentation du graphe RDF de l'exercice 3

```
8
9  jp:nbJoueurs rdfs:domain jt:jeu .
10 jt:jeu rdfs:subClassOf jt:jouet .
11 jt:societe rdfs:subClassOf jt:jeu .
12 jp:extension rdfs:domain jt:ext;
13              rdfs:range jt:jeu;
14              rdfs:subPropertyOf jp:serie.
15 jp:serie rdfs:domain jt:jouet.
16
17 jj:villainous jp:nbJoueurs "2".
18 jj:villainous jp:nbJoueurs "6".
19 jj:vill-ext1 jp:extension jj:villainous.
20
21 jj:barbiePlage jp:serie js:barbie.
22 jj:ken jp:serie js:barbie.
```

La figure 2 page 5 regroupe des règles de déduction. Dans la suite de l'exercice, on considérera uniquement ces règles de déduction (et on ignorera donc les règles RDFS standard qui ne feraient pas partie de la figure 2).

Question 3.1: Combien de types différents possède le nœud `jj:vill-ext1` ?

Question 3.2: Combien de triplets sont dans le graphe saturé de G, mais pas dans G lui-même (i.e. combien la saturation a-t-elle ajouté de triplets) ?

Question 3.3: Dans le cas général (i.e. en considérants d'autres graphes que G), la relation `jp:compatible` est-elle **nécessairement** symétrique et/ou transitive dans le graphe saturé associé ? Justifiez.

Question 3.4: Si on considère des graphes **sans triplets** ayant pour propriété `jp:compatible`, la relation `jp:compatible` est-elle **nécessairement** symétrique et/ou transitive dans le graphe saturé associé ? Justifiez.

$$\frac{S\ P\ O \text{ et } P\ \texttt{rdfs:domain}\ T}{S\ \texttt{rdf:type}\ T}\ (Domain)$$

$$\frac{S\ P\ O \text{ et } P\ \texttt{rdfs:range}\ T}{O\ \texttt{rdf:type}\ T}\ (Range)$$

$$\frac{X\ \texttt{rdf:type}\ C \text{ et } C\ \texttt{rdfs:subClassOf}\ D}{X\ \texttt{rdf:type}\ D}\ (SC)$$

$$\frac{S\ P\ O \text{ et } P\ \texttt{rdfs:subPropertyOf}\ Q}{S\ Q\ O}\ (SP)$$

$$\frac{S\ \texttt{jp:extension}\ O \text{ et } O\ \texttt{jp:nbJoueurs}\ N}{S\ \texttt{jp:nbJoueurs}\ N}\ (NJ)$$

$$\frac{X\ \texttt{jp:serie}\ S \text{ et } Y\ \texttt{jp:serie}\ S}{X\ \texttt{jp:compatible}\ Y}\ (Se)$$

FIGURE 2 – Règles de déduction de l'exercice 3

Exercice 4: XML et relationnel **(5 points)**

Soit le script SQL suivant :

```
1  CREATE TABLE R(A INTEGER PRIMARY KEY, B TEXT);
2  CREATE TABLE S(C INTEGER PRIMARY KEY, D TEXT, A INTEGER REFERENCES R(A));
3  CREATE TABLE T(E INTEGER PRIMARY KEY, F TEXT, C INTEGER REFERENCES S(C));
4
5  CREATE VIEW V1 AS
6  SELECT XMLELEMENT(name "x",
7                    XMLATTRIBUTES(S.C as "c"),
8                    XMLFOREST(S.D as "y"),
9                    XMLAGG(XMLFOREST(T.F as "z")))
10          AS xml_data,
11        S.A AS A,
12        S.C AS C
13 FROM S JOIN T ON S.C = T.C
14 GROUP BY S.A, S.C, S.D;
15
16 CREATE VIEW V2 AS
17 SELECT XMLELEMENT(name "u",
18                   XMLATTRIBUTES(R.A as "a"),
19                   XMLFOREST(R.B as "w"),
20                   XMLAGG(V1.xml_data))
21          AS xml_data,
22          R.A as A
23 FROM R JOIN V1 on R.A = V1.A
24 GROUP BY R.A, R.B;
```

On suppose que les tables `R`, `S`, `T` ne sont pas vides.

Question 4.1:

Soit la requête XQuery Q1 suivante :

```
<res1 h="{/x/@c}">
  {/x/y}
</res1>
```

Écrire une requête SQL qui donne, pour chaque document `xml_data` produit par la vue `V1`, le résultat de la requête XQuery Q1 appliquée à ce document.

Question 4.2:

Soit la requête XQuery Q2 suivante :

```
<res2>
  {
      for $x1 in /u/x, $x2 in /u/x[@c != $x1/@c]
      where $x1/z = $x2/z
      return <p>
                <p1>{$x1/y/text()}</p1>
                <p2>{$x2/y/text()}</p2>
             </p>
  }
</res2>
```

Écrire une requête SQL qui donne, pour chaque document `xml_data` produit par la vue `V2`, le résultat de la requête XQuery Q2 appliquée à ce document.