Examen MIF04 - Gestion de données pour le Web - session 1 - 16 janvier 2018
Durée : 2h
Documents autorisés

Numéro de copie :

Il faut rendre ces feuilles en les glissant dans votre copie anonyme. Ne pas l'utiliser comme brouillon. Les réponses sont à donner sur ces feuilles, pas dans la copie. Remplir le champ ci-dessus avec le *numéro de la copie* dans laquelle vous allez la glisser. Remplir la partie d'anonymat de la copie, puis coller le coin. Le barème est donné à titre indicatif, l'examen sera noté sur 20.

Exercice 1: (6 points)

On considère le graphe RDF suivant (représenté graphiquement dans la figure 1) :

```
@prefix f: <http://films.example.com/rdf/f/> .
@prefix p: <http://films.example.com/rdf/p/> .
@prefix r: <http://films.example.com/rdf/r/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

p:ecoen rdfs:label "Ethan Coen";
        r:realise f:bareading .
p:jcoen rdfs:label "Joel Coen".
p:gclooney rdfs:label " George Clooney";
           r:jouedans f:obrother, f:bareading, f:mmen, f:lmpouvoir.

f:obrother rdfs:label "O' Brother";
           r:aPourRealisateur p:ecoen, p:jcoen.
f:bareading rdfs:label "Burn After Reading" ;
            r:aPourRealisateur p:jcoen.
f:mmen rdfs:label "Monuments Men";
       r:aPourRealisateur p:gclooney .
f:lmpouvoir rdfs:label "Les marches du pouvoir";
            r:aPourRealisateur p:gclooney.

r:realise rdfs:subPropertyOf r:participe .
r:jouedans rdfs:subPropertyOf r:participe .
r:jouedans rdfs:domain r:acteur;
           rdfs:range f:film .
r:acteur rdfs:subClassOf r:personne.
r:realisateur rdfs:subClassOf r:personne.
```

Dans la suite de l'exercice, on supposera que les préfixes déclarés ci-dessus sont prédéfinis pour toutes les requêtes.

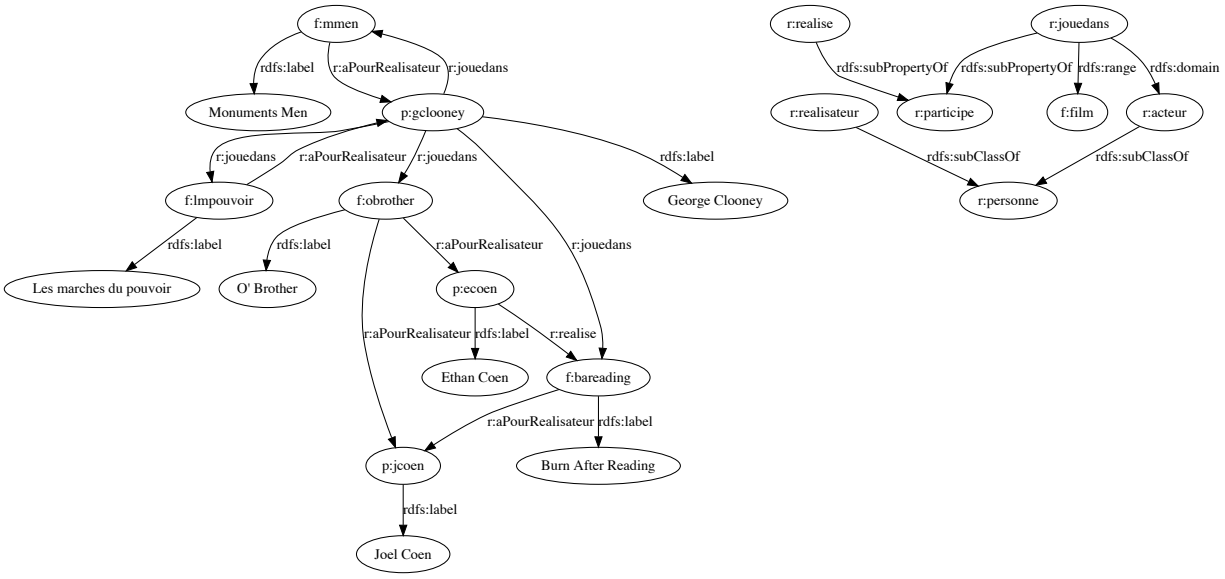


FIGURE 1 – représentation graphique

On considère les règles de déduction suivantes :

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

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

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

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

$$\frac{S\ \texttt{r:aPourRealisateur}\ O}{O\ \texttt{r:realise}\ S}\ (RealInv)$$

1. Soit la requête SPARQL suivante :

```
1 SELECT ?p WHERE {
2   ?p rdf:type r:personne.
3   ?p r:realise ?f.
4 }
```

Donner le résultat de l'exécution de cette requête sur le graphe saturé :

2. Donner le nombre de triplets du graphe saturé dont le prédicat est `rdf:type`

3. Donner un unique triplet à ajouter dans le graphe de façon à ce qu'on puisse déduire le type `r:realisateur` à la fois pour les sujets des triplets dont le prédicat est `r:realise` et les objets des triplets dont le prédicat est `r:aPourRealisateur`.

4. Soit Q1 la requête SPARQL suivante :

```
1 SELECT ?a ?b WHERE {
2   ?a r:participe ?b.
3 }
```

Réécrire cette requête en une requête Q2 de façon à ce que le résultat de Q2 sur le graphe non saturé soit identique au résultat de Q1 sur le graphe saturé.

Exercice 2: (9 points)

On considère une collection `notes` dans MongoDB. Dans cette collection, chaque document représente un étudiant, avec ses notes dans chaque matière. Un exemple de tel document est donné ci-dessous :

```
{
  "_id": "23456789",
  "nom": "Toto",
  "formation": "M1IF",
  "matieres": [
    { "code": "INF1001M",
      "notes": [ 10, 12, 9 ] },
    { "code": "INF1002M",
      "notes": [ 18, 12 ] }
    /* ... */
  ]
}
```

1. Soit les deux fonctions `map1` et `reduce1` suivantes :

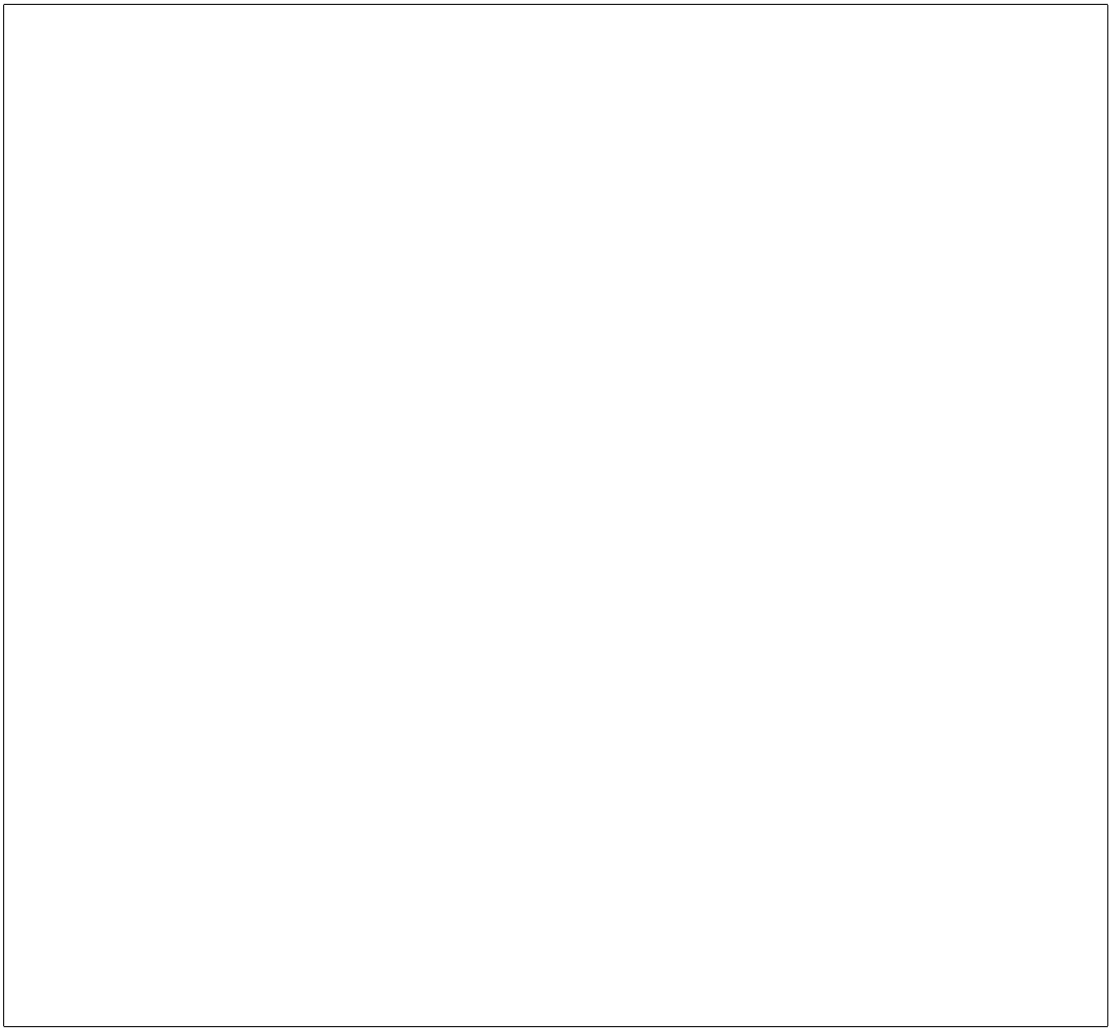
```
var map1 = function() {
  var s = 0;
  for(var i in this.matieres) {
    s = s + this.matieres[i].notes.length;
  }
  emit(1, s);
}

var reduce1 = function(k, values) {
  var s = 0;
  for(var i = 0; i < values.length; i++) {
    s = s + values[i];
  }
  return s;
}
```

Expliquer en une phrase ce qui est calculé par le *job* lancé par `db.notes.mapReduce(map1, reduce1, {out : {inline : 1}})` :

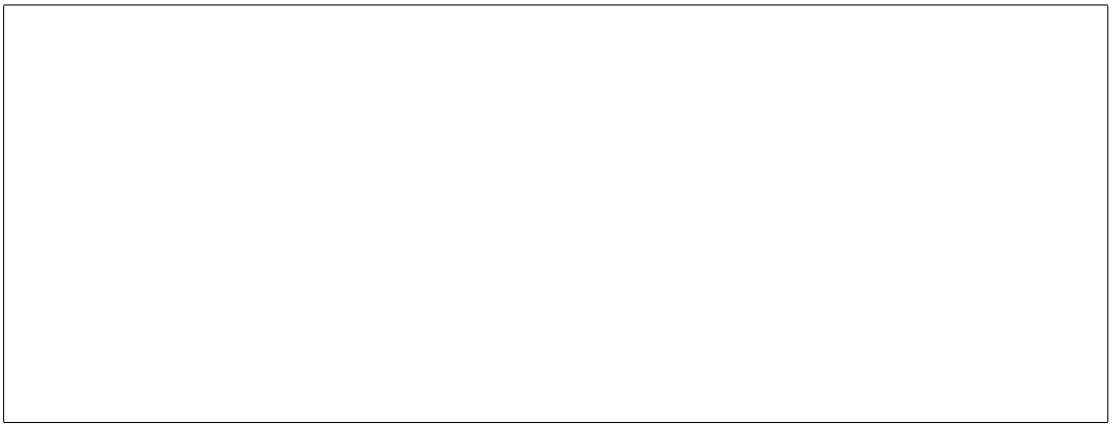
2. Sachant que la note *finale* d'un étudiant dans une matière est la moyenne de ses notes dans cette matière (toutes avec un coefficient égal à 1), écrire les fonctions `map2`, `reduce2` et `finalize2` correspondant à un *job* qui calcule la moyenne des notes finales des étudiants pour chaque matière. Le *job* sera lancé par `db.notes.mapReduce(map2, reduce2, {out : {inline : 1}, finalize : finalize2})`. On rappelle l'existence du phénomène de *re-reduce* et le fait que la fonction `finalize2` sera appelée une fois par valeur de clé, après exécution des `reduce2`.

   `map2`

reduce2



finalize2



3. On souhaite, pour chaque formation, trouver **tous** les étudiants (leur nom) ayant la meilleure moyenne (*i.e.* la moyenne des notes finales de chacune de leurs matières). Donner les fonctions `map3` et `reduce3` permettant d'effectuer ce calcul lorsqu'on lance le *job* via la commande `db.notes.mapReduce(map3, reduce3, {out : {inline : 1}})`. Enfin ce calcul doit se faire sans utiliser de fonction `finalize` et en prenant en compte le phénomène de *re-reduce*. Il est possible, si besoin, de renvoyer plus d'informations sur un étudiant que son nom. Il est possible d'avoir plusieurs étudiants ayant la même moyenne.

   `map3`

`reduce3`

Exercice 3: (5 points)

On considère un graphe RDF représenté par une collection `graphe` dans une base MongoDB. Chaque document de la collection correspond à un triplet. On suppose que pour un triplet t, son sujet (respectivement son prédicat et son objet) est accessible via $t.sujet$ (respectivement $t.predicat$ et $t.objet$).

On considère la requête SPARQL suivante :

```
1 PREFIX u: <http://www.univ-lyon1.fr/>
2
3 SELECT ?ue  (count(*) as ?c) WHERE {
4   ?e u:enseigne ?ue.
5   ?ue u:formation u:m1if.
6 }
7 GROUP BY ?ue
```

On souhaite implémenter calculer cette requête via un *job* Map/Reduce sur la collection `graphe`. Donner les fonctions `map`, `reduce` et `finalize` permettant de faire ce calcul si on le lance via la commande `db.graphe.mapReduce(map, reduce, {out : {inline : 1}, finalize : finalize})`. On supposera que cette commande sera suivie de l'application d'un filtre supprimant toutes les UEs pour lesquelles la valeur correspondant à `?c` est `undefined`.

`map`

`reduce`

`finalize`