Examen MIF04 - Gestion de données pour le Web - session 1 - 8 janvier 2019
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) :

```
1  @prefix eq: <http://example.com/rdf/eq#> .
2  @prefix p: <http://example.com/rdf/p#> .
3  @prefix b: <http://example.com/rdf/b#> .
4  @prefix et: <http://example.com/rdf/et#> .
5  @prefix bat: <http://example.com/rdf/bat#> .
6  @prefix dpt: <http://example.com/rdf/dpt#> .
7  @prefix rel: <http://example.com/rdf/rel#> .
8  @prefix t: <http://example.com/rdf/t#> .
9  @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
10 @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
11
12 p:alice rel:bureau b:a224;
13         rel:equipe eq:bd.
14 p:bertrand rel:bureau b:a224;
15            rel:equipe eq:graphes.
16 p:chloe rel:bureau b:b321;
17         rel:empl bat:betelgeuse;
18         rel:equipe eq:bd.
19 b:a224 rel:dans et:a2.
20 b:b321 rel:dans et:b3.
21 et:a2 rel:dans bat:aldebaran.
22 et:b3 rel:dans bat:betelgeuse.
23 rel:bureau rdfs:subPropertyOf rel:empl;
24            rdfs:domain t:personne.
25 rel:empl rdfs:range t:place.
```

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

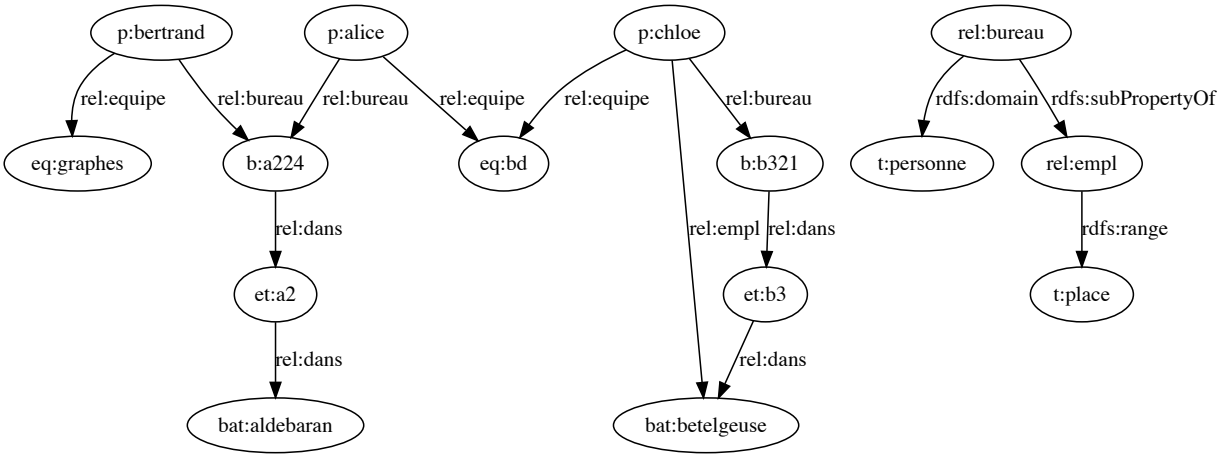


FIGURE 1 – représentation graphique

On considère les règles de déduction suivantes :

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

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

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

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

$$\frac{S\ \texttt{rel:empl}\ E \quad \text{et} \quad E\ \texttt{rel:dans}\ F}{S\ \texttt{rel:empl}\ F}\ (EmplPrpg)$$

$$\frac{S\ \texttt{rel:empl}\ E \quad \text{et} \quad S\ \texttt{rel:equipe}\ F}{F\ \texttt{rel:empl}\ E}\ (EmplPrpg2)$$

Par la suite, on nommera TD l'ensemble des triplets du graphe saturé qui ne sont pas dans le graphe de départ.

1. Donner l'ensemble des **prédicats** des triplets de TD.

2. Donner la taille de TD.

3. Soit la requête SPARQL suivante :

```
SELECT ?f WHERE {
  ?p rel:equipe eq:graphes.
  ?p r:empl ?f.
}
```

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

4. Une équipe est l'objet d'au moins un triplet dont le prédicat est `rel:equipe`. Une personne a le type `t:personne`. Donner deux triplets dont le prédicat est préfixé par `rdfs:` et tels que si on les ajoute au graphe précédent alors on pourra déduire les triplets de la forme X `rdf:type t:ressource` pour les nœuds X qui sont des personnes ou des équipes du graphe.

Exercice 2: **(9 points)**

On considère une collection `meteo` dans MongoDB. Dans cette collection, chaque document représente une série de mesures prises par capteur durant une journée. Un exemple de tel document est donné ci-dessous :

```
{
  "_id": 123456,
  "capteur": "nautibus_temp_3",
  "type": "temp",
  "unite": "C",
  "jour": "2019-01-08",
  "mesures": [
    { "valeur": 2.0,
      "heure": 15.0 },
    { "valeur": 3.0,
      "heure": 14.0 }
    /* ... */
  ]
}
```

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

```
var map1 = function() {
  var s = 0;
  for(var i in this.mesures) {
    if (this.mesures[i].heure > "12:00") {
      s = s + 1;
    }
  }
  emit(this.capteur, 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 la commande : `db.meteo.mapReduce(map1, reduce1, {out : {inline : 1}})` :

2. Écrire les fonctions `map2`, `reduce2` et `finalize2` correspondant à un *job* qui donne, pour chaque jour, la moyenne des variations de température sur la journée (on ne s'intéresse donc pas aux autres capteurs). 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`. Ce *job* sera lancé avec la commande

```
db.meteo.mapReduce(map2, reduce2,
                   {out : {inline : 1}, finalize : finalize2})
```

Code de la fonction `map2`

Code de la fonction `reduce2`

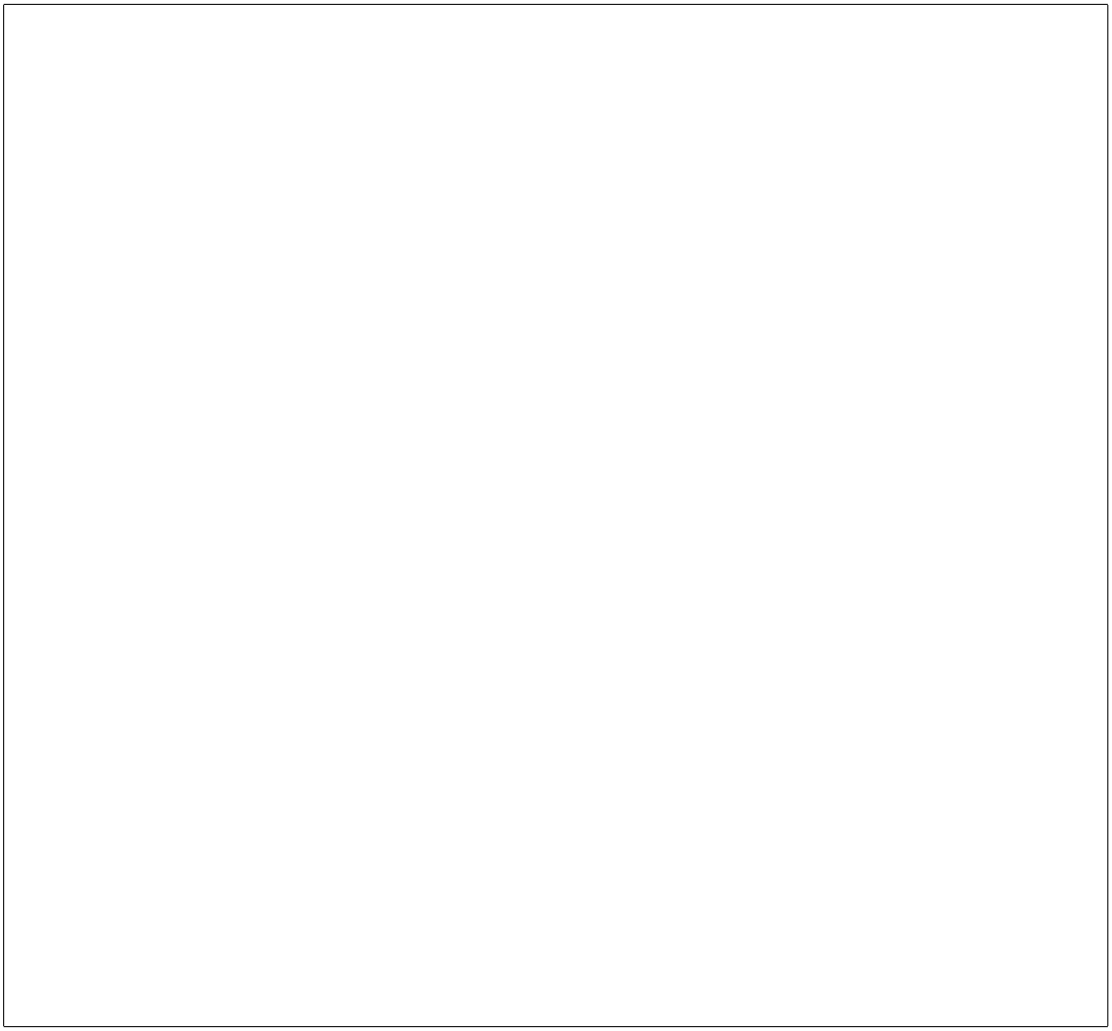
Code de la fonction `finalize2`

3. Certains capteurs de température ont été reconfigurés durant la période où les données ont été accumulées. Écrire les fonctions `map3`, `reduce3` et `finalize3` correspondant à un *job* permettant de savoir quels sont les capteurs dont l'unité de mesure a changé des Celsius (`"unite": "C"`) en Fahrenheit (`"unite": "F"`) ou vice-versa. La fonction `finalize3` devra renvoyer la valeur `undefined` si un capteur n'envoie que des valeurs en Celsius ou que des valeurs en Fahrenheit. Sinon, elle renverra la première date où un changement de configuration a été constaté. Ce *job* sera lancé avec la commande

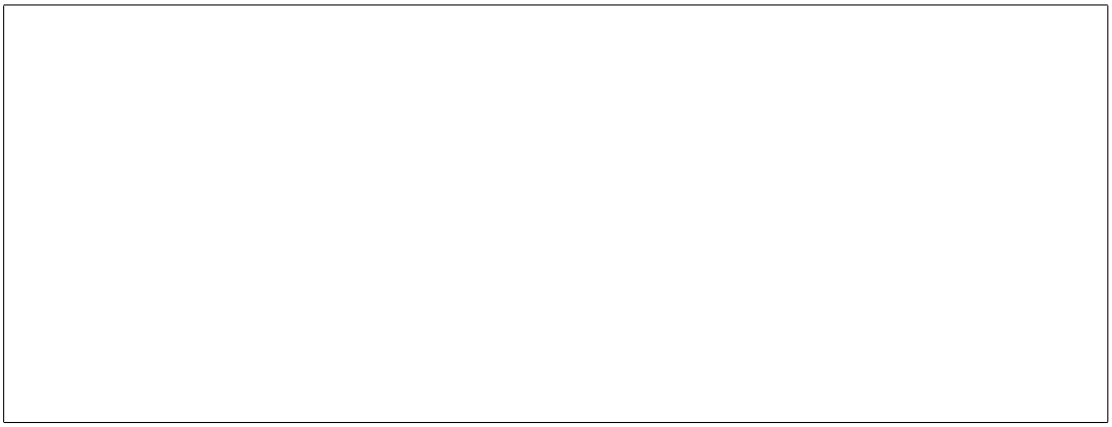
```
db.meteo.mapReduce(map3, reduce3,
                   {out : {inline : 1}, finalize : finalize3})
```

Code de la fonction `map3`

reduce3



finalize3



Exercice 3: **(5 points)**

Le cadastre est un registre contenant les informations sur les différentes parcelles de terrains, leur emplacement, leur propriétaire, etc. Une partie des informations du cadastre a été extraite dans la collection `cadastre` d'une base MongoDB. Dans cette collection, chaque parcelle est représentée par un document. Un exemple de tel document est donné ci-dessous :

```
1 {
2   "_id": "B753",
3   "proprietaire": "Alphonse et Alphonsine",
4   "superficie": 312.5,
5   "ville": "Villeurbanne",
6   "quartier": "Saint Jean",
7     /* ... diverses informations variant selon les parcelles ... */
8   ]
9 }
```

Écrire les fonctions `mapc` et `reducec` correspondant à un *job* qui permet de connaître pour chaque ville les identifiants des 20 plus grandes parcelles, sans utiliser de fonction `finalize` et en prenant en compte le phénomène de *re-reduce*. On supposera que deux parcelles ont toujours une superficie différente.

On rappelle la restriction technique de MongoDB qui ne sait pas gérer les valeurs tableaux directement dans les `emit` : les tableaux doivent être encapsulés dans des objets (*i.e.* des dictionnaires). On rappelle également que les tableaux en Javascript disposent d'une méthode `sort` qui prend en argument une fonction de comparaison. Cette fonction prendra deux arguments et renverra un nombre négatif si le premier est inférieur au second, 0 s'ils sont égaux et un nombre positif si le second est inférieur au premier. La méthode `sort` trie le tableau en place, elle ne crée pas un nouveau tableau. Par ailleurs la méthode `slice(x,y)` renvoie les cases `x` à `y`-1 d'un tableau.

Ce *job* sera lancé avec la commande `db.cadastre.mapReduce(mapc,reducec, {out : {inline : 1}})`.

Code de la fonction `map2`

Code de la fonction `reduce2`

On sait que MongoDB exécute ses reduce par paquets 100 valeurs. On souhaite à présent connaître les 1000 plus grandes parcelles de France. Expliquer en français comment devrait procéder la fonction `reducec` pour éviter de construire un tableau à 100 000 entrées dans chaque reduce.

.