Examen MIF18 - Gestion de données pour le Web - session 1 - 14 décembre 2015
Durée : 1h30
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: Relationnel et XML (4 points)

On considère la DTD suivante :

```
<!ELEMENT recette (nom,ingredient+,description)>
<!ELEMENT ingredient (nom,quantite)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT quantite (#PCDATA)>
<!ATTLIST quantite unite CDATA #IMPLIED>
<!ELEMENT description (#PCDATA)>
```

On considère également le schéma relationnel mis en place via le code SQL suivant :

```
CREATE TABLE recette(id INTEGER PRIMARY KEY,
                     nom VARCHAR(255) NOT NULL,
                     DESCRIPTION TEXT NOT NULL);
CREATE TABLE ingredient(id INTEGER PRIMARY KEY,
                        nom VARCHAR(255) NOT NULL);
CREATE TABLE quantite(id_recette INTEGER REFERENCES recette(id),
                      id_ingredient INTEGER REFERENCES ingredient(id),
                      valeur FLOAT NOT NULL,
                      unite VARCHAR(255),
                      PRIMARY KEY (id_recette,id_ingredient) );
```

1. Donner une requête SQL permettant pour chaque recette (*i.e.* chaque valeur d'`id` de la table `recette`) de calculer sa représentation XML conforme à la DTD. On utilisera les fonction de génération XML (`XMLElement`, `XMLAttributes`, `XMLAgg`, etc) pour créer les arbres XML. On rappelle que par exemple :

   ```
   XMLElement(name "a", XMLAttributes(ATT1 as "b"), ATT2)
   ```

   génère le XML suivant, en supposant que la valeur de l'attribut relationnel `ATT1` est 'toto' et que celle de `ATT2` est 'titi' :

   ```
   <a b="toto">titi</a>
   ```

2. En quoi le fait d'avoir utilisé + et non ∗ dans la première ligne de la DTD peut-il poser un problème vis-à-vis d'une problématique d'export de ces données relationnelles vers le XML ?

Exercice 2: Relationnel et RDF (4 points)

On considère l'instance suivante d'une base de donnée relationnelle :

Ville			
Nom	**Code**	Superficie	Population
Lyon	69123	47,87	496343
Bron	69029	10,3	39232
Caluire-et-Cuire	69034	10,45	42038
Vénissieux	69259	15,33	61183
Villeurbanne	69266	14,52	146282

Voisines	
Ville1	**Ville2**
69123	69029
69123	69034
69123	69259
69123	69266
69034	69266
69266	69029
69029	69259

On considère également les triplets RDF suivants[1], générés à partir d'une partie de cette instance :

```
@prefix v: <http://www.grand-lyon.com/ville#>
@prefix p: <http://www.grand-lyon.com/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>

v:69123 rdfs:label "Lyon";
        p:superficie "47.87"^^xsd:double;
        p:population "496343"^^xsd:integer;
        p:voisine v:69029, v:69034, v:69259, v:69266.

v:69029 rdfs:label "Bron";
        p:superficie "10,3"^^xsd:double;
        p:population "39232"^^xsd:integer;
        p:voisine v:69123, v:69259, v:69256.
```

1. Donner les triplets qui seraient générés suivant le même principe pour la ville de Villeurbanne. On réutilisera les préfixes définis ci-dessus (inutile de les recopier). On fera particulièrement attention aux triplets ayant pour prédicat `p:voisine`.

1. présentés ici en utilisant la notation Turtle

2. Traduire la requête SPARQL ci-dessous en une requête SQL sur la base de données de départ :

```
PREFIX v: <http://www.grand-lyon.com/ville#>
PREFIX p: <http://www.grand-lyon.com/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?n WHERE {
  ?v rdfs:label ?n.
  ?v p:voisine ?v2.
  ?v p:population ?p.
  ?v2 p:population ?p2.
  FILTER(?p > 3 * ?p2).
}
```

Exercice 3: MongoDB (10 points)

Dans cet exercice, on souhaite calculer les cooccurrences de mots dans les lignes d'un texte en anglais. Le résultat du calcul est une matrice carrée M, où les lignes et les colonnes sont des mots du texte et où $M(w_i, w_j)$ indique le nombre de fois où le mot w_i apparaît à proximité du mot w_j.

Ici, les cooccurrences seront recherchées dans une fenêtre glissante de taille fixée à 3 sur la ligne de texte. Autrement dit, on va ajouter 1 à la case $M(w_i, w_j)$ à chaque fois que w_i apparaît à moins de 3 mots de distances de w_j. Par exemple pour la ligne "Applies function fn to each of the elements in the range [first,last)", la matrice de cooccurrences limitée aux mots de longueur supérieure ou égale à 3 et dont on ne garde que le triangle supérieur est la suivante :

	applies	each	elements	first	function	last	range	the
applies	-	0	0	0	1	0	0	0
each	-	-	1	0	1	0	0	1
elements	-	-	-	0	0	0	1	2
first	-	-	-	-	0	1	1	1
function	-	-	-	-	-	0	0	0
last	-	-	-	-	-	-	1	1
range	-	-	-	-	-	-	-	1
the	-	-	-	-	-	-	-	-

1. Expliquez pourquoi on ne garde que le triangle supérieur de la matrice M.

2. Proposez un job map/reduce en javascript pour MongoDB qui calcule M limitée aux mots de longueur supérieure ou égale à 3. Les clefs des objets retournés par le reduce seront les paires (w_i, w_j) du triangle supérieur de M pour lesquelles $M(w_i, w_j) > 0$, la valeur retournée par le reduce pour cette clef sera $M(w_i, w_j)$.

   On suppose qu'on dispose d'une fonction `tokenize` qui transforme une chaîne en un tableau des mots qu'elle contient et que le texte de chaque document est accessible dans la fonction `map` via `this.text`.

Fonction `map()` :

Fonction `reduce(key,values)` :

3. On désire maintenant normaliser la matrice, c'est-à-dire diviser chaque nombre de cooccurrences $M(w_i, w_j)$ par le nombre total de cooccurrences concernant w_i. En reprenant l'exemple, on obtiendrait $M'(applies, function) = 1$, $M'(elements, each) = 1/4$, $M'(elements, range) = 1/4$ et $M'(elements, the) = 2/4 = 1/2$. On peut remarquer que pour obtenir le nombre total de cooccurrences dans l'exemple, le calcul doit prendre en compte la ligne et la colonne du mot w_i dans M.

   La solution proposée à la question précédente rend ce nouveau calcul difficile.

   Expliquez comment résoudre ce problème en proposant une représentation alternative de M et donnez le pseudo code des fonctions map, reduce et finalize (le code complet javascript des fonctions n'est pas demandé).

Clé et valeurs choisies :

Fonction `map()` :

Fonction `reduce(key,values)` :

Fonction `finalize(key,value)` :

Exercice 4: Correspondance Objet-Relationnel (4 points)

On considère le code source Java pour les 3 classes suivantes annotées via l'API JPA :

```
package exercice4;

import javax.persistence.*;
import java.util.*;

@Entity
@Table(name = "jeu")
public class Jeu {
```

```
    @Id
    public String nom;

    @OneToOne(mappedBy = "jeu")
    @OneToMany(mappedBy = "jeu")
    @ManyToOne(mappedBy = "jeu")
    @ManyToMany(mappedBy = "jeu")
    public Collection<Equipe> equipes;
}
```

```
package exercice4;

import javax.persistence.*;
import java.util.*;

@Entity
@Table(name = "joueur")
public class Joueur {
    @Id
    public String mail;

    @OneToOne(mappedBy = "joueurs")
    @OneToMany(mappedBy = "joueurs")
    @ManyToOne(mappedBy = "joueurs")
    @ManyToMany(mappedBy = "joueurs")
    public Collection<Equipe> equipes;
}
```

```
package exercice4;

import javax.persistence.*;
import java.util.*;

@Entity
@Table(name = "equipe")
public class Equipe {
    @Id
    public String nom;

    @OneToOne
    @OneToMany
    @ManyToOne
    @ManyToMany
    public Jeu jeu;

    @OneToOne
    @OneToMany
    @ManyToOne
    @ManyToMany
    public Collection<Joueur> joueurs;
```

```
    @OneToOne
    @OneToMany
    @ManyToMany
    public Joueur capitaine;
}
```

1. Rayer dans le code des classes précédentes les *annotations* qui vont nécessairement provoquer une erreur
2. Proposer un schéma relationnel compatible avec les annotations restantes. On listera les tables avec leurs attributs en soulignant le ou les attributs qui font partie de la clé primaire. Pour chaque clé étrangère, on l'écrira sous la forme $table_1(att_1, att_2) \rightarrow table_2(att_3, att_4)$ pour un clé étrangère dans $table_1$ qui référence $table_2$, l'attribut att_1 (resp. att_2) devant correspondre à att_3 (resp. att_4).