Examen - M2 Big Data - Session 1
Notes personnelles manuscrites autorisées

17 janvier 2020

Le barème est donné à titre indicatif et n'est en aucun cas définitif. Merci de remettre une copie lisible et propre (1pt).

1 XML (9 pts)

Soit une DTD `movies.dtd`

```
<!ELEMENT movies (movie*,artist*)>
<!ELEMENT movie (title , year, country*, genre, summary?, role*)>
<!ATTLIST movie director IDREF >
<!ELEMENT artist (first_name, last_name, birth_date)>
<!ATTLIST artist id ID >
<!ELEMENT role #PCDATA >
<!ATTLIST role actor #IDREF >
<!ELEMENT title (#PCDATA)>
<!ELEMENT country (#PCDATA)>
<!ELEMENT year (#PCDATA)>
<!ELEMENT genre (#PCDATA)>
<!ELEMENT summary (#PCDATA)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT birth_date (#PCDATA)>
<!ELEMENT role (#PCDATA)>
```

Les attributs `director` (réalisateur d'un film) et `actor` qui joue un rôle (acteur) sont des références vers des éléments `artist`.

1.1 XQuery

Exprimez les requêtes sur un document `mymovies.xml` validé par la DTD `movies.dtd`. On indique pour chaque requête la DTD du résultat.

1. Un élément `comedy_titles` qui contient les titres de comédies (`GENRE='COMEDY'`) avec leur pays triés par titre.

```
<!ELEMENT comedy_titles (comedy*)>
<!ELEMENT comedy (title , year)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT country (#PCDATA)>
```

2. Un élément `allen_titles` qui contient tous les titres de films tournés par Woody Allen triés par l'année.

```
<!ELEMENT allen_titles (title*)>
<!ELEMENT title (#PCDATA)>
```

3. Les films avec leur titre et l'année (`title` and `year`) comme attribut et pour chaque film les acteurs dont le rôle apparaît dans le résumé du film. Les films sont triés par leur

titre.

```
<!ELEMENT movies (movie*)>
<!ELEMENT movie (actor*)>
<!ATTLIST movie title CDATA
               year CDATA >
<!ELEMENT actor (first_name, last_name, birth_date, role)>
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT first_name (#PCDATA)>
<!ELEMENT birth_date (#PCDATA)>
<!ELEMENT role (#PCDATA)>
```

1.2 Restructuration relationnelle

On suppose que les documents sont stockés dans une base de données relationnelle.

1. Définissez les différents schémas relationnels (tables et attributs) qui permettent de stocker tous les documents validés par la DTD en fonction des différentes méthodes de transformation existantes.

2. Traduisez la requête suivante en requête SQL sur ces schémas : *tous les titres de films tournés par Woody Allen.*

1.3 Stockage relationnel

Soit 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)>
```

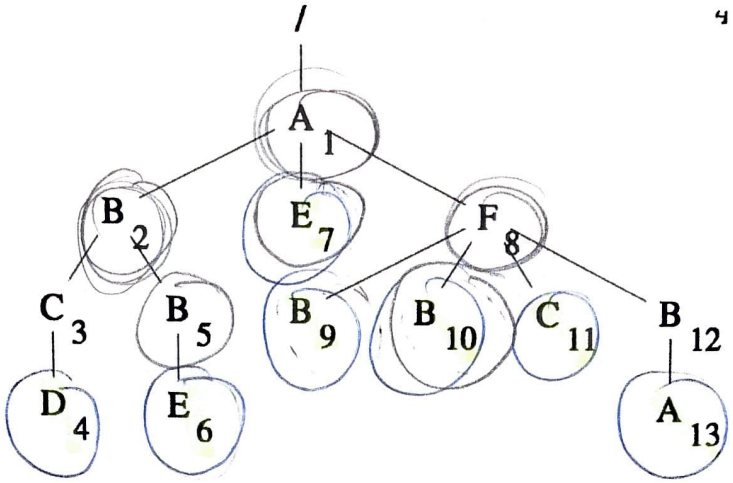
Soit le schéma relationnel suivant :

```
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) );
```

Donner une requête SQL permettant de calculer pour chaque recette sa représentation XML conforme à la DTD. Vous utiliserez les fonctions de génération XML (XMLElement, XMLAttributes, XMLAgg, etc) pour créer les arbres XML. Pour rappel : `XMLElement(name "a",XMLAttributes(ATT1 as "b"),ATT2)` génère l'élément a suivant en supposant que la valeur de l'attribut relationnel ATT1 est 'val1' et que celle de ATT2 est 'val2' : `<a b="val1">val2</a>`

2 XPath (4pts)

Soit donné l'arbre XML suivant où chaque noeud est identifié par un attribut `@id` l'identifiant de la racine est égal à 0 :



1. Donnez 3 expressions `XPath` qui retournent la liste des noeuds 4, 6, 7, 9, 10, 11, 13 (feuilles de l'arbre).
2. Donnez pour chaque expression `XPath` ci-dessous la liste des identifiants des nœuds retournés :
   (a) /descendant : : B/descendant : : ∗/@id
   (b) /descendant : : B[child : : ∗]/@id tout les noeud enfant de b.
   (c) /descendant : : ∗/child : : ∗[1]/@id le premier enfant du noeud 1
   (d) /descendant : : ∗/following − sibling : : B[1]/@id
   (e) /descendant : : E[not(following − sibling : : ∗)]/@id
   (f) /descendant : : ∗[child : : E[not(following − sibling : : ∗)]]/@id
3. Donnez pour chaque séquence de nœuds ci-dessous une expression `XPath` (syntaxe étendue ou abrégée) qui la calcule :
   (a) 6, 7. //E/@id.
   (b) 2, 9, 10. //B[@id<]/@id
   (c) 5, 7, 10. //child::[2]/@id
   (d) 1, 2, 8. /A[@id%2 =0]/@id.

3 MapReduce (6 pts)

On dispose d'un fichier de résultats sportifs en accrosport (gymnastique acrobatique qui se pratique en équipe) contenant pour chaque médaille, une liste de noms de gymnastes. On souhaite calculer, pour chaque trio de gymnastes (x, y, z), le nombre de médailles que ces trois gymnastes ont gagnées. Si la médaille a été obtenue par un sur-ensemble de (x, y, z) (par ex. (x, u, v, y, w, z)) ça compte pour 1. On ne considère que les ensembles de gymnastes, leur ordre/rôle dans l'équipe ne compte pas (ex. voltigeu.r.se ou porteu.r.se) (z, y, x) est considéré comme une occurrence.

1. Expliquer comment faire ce calcul en utilisant le modèle de programmation parallèle MapReduce. Vous pouvez utiliser un schéma avec légende sur un exemple.
2. Spécifiez la(les) fonction(s) qu'il est nécessaire de développer.

Map.
reduce.

// descendant::*