PROJET MLBDA SQL3/XML

YOUVA ADDAD

Résumé :

Ce projet consiste donc à générer des documents XML à partir des données stockées dans une base de données relationnelle en utilisant SQL3 et le type abstrait XMLType, définir des DTD ,créations de méthodes.
Dans la partie SQL3 ,des méthodes toXML sont définies pour la transformation de données en instance de XML , on aura aussi à mettre à jour des tables a insérer des données dans des tables.
En ce qui concerne la partie XPATH on aura a utilisé les fonctions EXTRACTVALUE et EXTRACT pour extraire des nœuds ou toute autre spécifie dans un chemin.

Fonction Utile :

-Replace(char,search_string,[replacement_string]) qui remplace tous les caractères qu'on recherche et on le remplace

-Coalesce(expr,[expr*]) prend renvois la première expression non nulle

EXERCICE 01 :

Dans ce première exercice j'ai définie 2 fichiers « sql » un pour chaque DTD.

DTD1 :

La création des types Airport ,Continent,Coordinates,Island,Desert,Mountain chacun des types contient une méthode qui transforme la donnée en XML , pour un Airport donnée il appartient forcement a un seul pays donc j'ai rajouté un attribut code du Pays ,le Pays auquel appartient l'aéroport ,de même pour Island ,Désert et Mountain , qui peuvent appartenir à plusieurs province , mais ici j'ai rajouté un attribut province à chacun d'eux ce qui ne pose pas problème on va juste créer plus instance de la même montagne avec des provinces différentes j'ai préféré cette idée parce que ça évite de créer des tables imbriquées a plusieurs niveau , l'autre idée aura consister à enlever l'attribut « province » dans Montagne , Desert ,Island et rajouté un ensemble de ces types dans province sa nous aura fait éviter la redondance des Montagne/Desert/Island qui ce répète mais on aurais eu 3 nested table dans provinces et ça va devenir lourd sur le type country qui aura déjà un ensemble continent et un ensemble de province.

Donc l'autre idée a pour principe d'ajouter un ensemble de type montagne ,desert et island dans province et lors de la création de la table country faudra rajouter un nested table pour ces tables.

J'ai privilégié de ne pas ajouter de table a province pour éviter justement les nested table mais cela a un cout ,de la redondance d'information.

Le type continent n'a pas besoin de création de table juste l'utilisation de la table synonyme encompasses.

Le type country qui prend un ensemble de continent et de province pour être conforme à la dtd parce que un pays a une au plusieurs province et de même ,il peut avoir plus d'un continent mais a un nécessairement un.

Le type T_mondial sers juste de main c'est lui qui lance le root principale.

Donc en résume un aéroport a le nom de son pays en attribut vu qu'un aéroport appartient à un seul pays de même province ,le type continent a son nom et le pourcentage et on a pas besoin de créer de table continent on a juste besoin du type, le country ayant un ensemble de continent on le remplira depuis encompasses ,et chacun des types a une méthode toXML , et mondial est le programme principale si je peux dire.

Le fichier générer s'appelle « mondialExercice1DTD1.xml » qui est bien conforme à la DTD1
mondialExercice1DTD1.xml: 167 ms (4390 elems, 8322 attrs, 31002 spaces, 0 chars)

ici j'ai enlevé la capitale d'un pays et j'ai eu cette erreur
[Error] mondialExercice1DTD1.xml:5:35: Attribute "capital" is required and must be specified for element type "province".
mondialExercice1DTD1.xml: 210 ms (4390 elems, 8321 attrs, 31002 spaces, 0 chars)
cannot insert NULL into ("E3872388"."PROVINCES"."CAPITAL") [SQL State=23000, DB Errorcode=1400]
1 statement failed.
Mais par contre on peut avoir des capital sans nom comme le nom de capital est requis et comme y a des provinces ayant des capitales nulles ,si on décidais d'omettre les provinces sans capital il y aurai des pays qui n'aurai pas de province et donc ne serai pas conforme à la dtd ,du coup il faudrait mettre la capital en #IMPIELED .

Les fonction collect et cast on était super utile pour le remplissage des tables ainsi que des tables imbriqué , de création d'ensemble directement depuis une requête.

DTD2 :

Dans la DTD2 j'ai créé un type le headquarter ,un type borders qui contiendra un ensemble de type border ,ici borders est un conteneur pour les frontières d'un pays, un type language qui contient le nom de la langue et le percentage pas besoin de créer la table language on utilisera le synonyme language pour le remplissage de country .

Ainsi la table country contient un attribut de type borders le conteneur des frontières, et un ensemble de langues, ici comme un pays a forcément un code j'ai pas fait de contrainte pour impieled ,tous les pays ont forcément un code d'ailleurs vaut mieux qu'il soit en #REQUIERED .

Pour le type organisation contient un ensemble de type pays qui contient lui-même des tables imbriqué.

Pour le headquarter j'ai décidé que c'était la city ou se trouve l'organisation ce qui me semble logique et donc pour être conforme à la DTD je ne rajoute pas les organizations qui ont des localisation nulle au sens de city(city is not null) sinon on violera les contraintes que j'ai rajouté pour les organisations.

Ici j'ai décidé d'éviter la redondance des langues pour donner une autre façon que celle de la DTD1 mais ceci entraine beaucoup de nested table comme je l'avais précisé précédemment.

un pays peut apparaitre plusieurs fois dans différentes organisations. Les avantages de cette redondance pour chercher les pays d'une organisation on a juste à regarder les nœuds fils, une requête XPath pour avoir les pays appartenant à une organisation consistera juste à faire « //organization[« nomOrg »]/country », l'inconvénient justement c'est cette redondance qui prendrai énormément de place parce que un pays pourrais exister plusieurs fois ,deux organisations différente pourrais avoir un pays qui appartient au deux et comme un pays a des borders donc sa deviens un peu lourd. Une autre idée consisterai à mettre en attribut de country les IDREFS des organizations auquel il appartient cella éviterai les problèmes de surcharges et on retrouve les informations facilement.

La validation de cette DTD après génération du fichier mondialExercice1DTD2.xml XML nous donne mondialExercice1DTD2.xml: 563 ms (44099 elems, 80561 attrs, 421787 spaces, 0 chars)
Donc cette DTD est bien conforme à ce qui a étais souhaité.

EXERCICE 02 :

Dans ce deuxième exercice on doit implémenter des méthodes .Ici j'ai décidé de créer la première et la deuxième méthode dans un même fichier , j'ai donc créer deux méthode toXML dans country ,une pour les geo et l'autre pour le calcul du peak.

Ici dans la première question j'ai créé un type geo qui stocke les montagnes/deserts/islands, la première difficulté pour cette question et de trié les objects donc ici pour chaque type Montagne ,Desert, Island j'ai rajouté une méthode « Order Member function match(*name*,*Type*) » qui compare entre deux objects plus précisément sur le nom de même type(je l'ai définie ainsi) afin de fournir un ordre.

A noter que la méthode peak utilise la méthode compute l'avantage de cette utilisation si on a pas rempli l'attribut geo d'un pays on pourra quand même utilisé la méthode peak . Mais comme j'ai utilisé le update ,pour mettre à jour les geos d'un pays j'aurais pu éviter d'utilisé la méthode .

Pour les méthodes de Q3 et Q4 elles sont définies de même dans le type T_Country.
Pour la première méthode elle renvoie le nom du continent ou l'appartenance d'un pays est plus grande ,donc j'ai directement depuis la table encompasse fait une jointure et fais une sous requête pour prendre le max.

Pour la fonction BorderPays j'ai utilisé la fonction qui calcul le pays principale ,et de là je prends les pays voisin a ce pays qui ont le même continent.

Dans la méthode blength j'ai décidé que c'était les pays voisins dans un même continent ou pas ,donc l'utilisation de la table synonyme borders était requise , de la en somme les longueurs des pays voisins et on renvois le résultat.

EXERCICE 03 :

Dans l'exercice 3 on doit définir une nouvelle DTD qui sera conforme et de telle manière qu'il soit possible de répondre aux requêtes données par des expressions XPATH.

Donc ici mon idée c'est que un continent a des nœuds fils country ,dans cette modélisation ,chaque continent a tous les pays qui sont englober dans ce continent ,et chaque pays a les provinces appartenant à ce pays ,et pour chaque province a les montagnes qui sont geo localiser dans cette province.
Les organizations et les river ont un ID pour l'attribut Source et isMember de country, ce sont des idrefs qui référencies respectivement les rivières qui prennent source dans ce pays et les organisations ou ce pays est membre .

Pour ce choix de modélisation on aura une redondance d'information , parce que un pays peut appartenir à plusieurs continent ,de même pour les montagnes ,elles peuvent appartenir à plusieurs province, mais malheureusement ici c'est la seul solution que j'ai trouvé qui fonctionne sur XPATH 1.0 pour résoudre les requêtes donner .Si y avait la possibilité d'utilisé XQuery on aurait directement donner pour chaque pays les idrefs des continents auxquels ils appartiennent, et donc de là on aurai la possibilité avec l'utilisation de variable en prenant tous les noms de continent différents et donc une jointure ensuite .

Dans cette exercice la fonction id() qui renvoie le nœud ou les nœud avec l'id en argument était utile ,qui est compatible avec XPATH 1.0 ainsi que la fonction sum qui fait la somme si c'est des number ou renvois 0 sinon aussi .

Pour la requête sur les organisations d'un pays ,ordonnée par date de création la table organizations a été remplit de sorte à avoir les organisations triées avec order by o.created . On avait juste besoin que les dates sois bien ordonnées dans le fichier XML ce qui est fait avec le order by .

Pour la DTD que j'avais créée pour cette exercice ,elle est bien conforme au XML générer

```
(base) MBPdeFalleADDAD:Exercice03 TristanLefebvre$ ./valide.sh MondialExercice3.xml
MondialExercice3.xml: 245 ms (3556 elems, 6524 attrs, 30329 spaces, 0 chars)
```

Les requêtes XPath sont ci-dessous :

1) `//continent/country[not(number(@population) < parent::continent/country/number(@population))]/@name`

Not(<) remplace le fonction max , en langage naturel on aurai quelque chose comme pour chaque continent le pays pour lequel il n'existe pas un pays dans le même continent(parent ::continent) qui a une population supérieur au pays courant (peu renvoyer plusieurs pays).

Ici on doit caster l'attribut population parce que la comparaison n'est pas supporter avec de gros chiffres en XPath ,la transformation en numbre est requise ici pour avoir des résultats cohérant.

2) id(//country/data(@isMember))
/data(@name)

Renvoie la séquence des organisations pour chaque pays, on pourra filtrer en de la sorte id(//country[@code="DZ"] /@isMember) pour avoir les organisations auquel le pays avec le code DZ appartient.

La fonction id(expr) prend une expression et renvois le nœud qui a comme identifiant « expr » ou tous les nœud si expr est une chaine séparer par des espaces.
Comme chaque pays a en attributs la référence aux organisations (idrefs) on accède à chaque organisation avec la fonction id

3) //province[@name="Nepal"]/mountain[not(@height<parent::province/mountain/@height)]

Ici on prend toutes les montagnes d'un province donnée ,et on prend celle qui a l'altitude supérieur a toute les autres ou sinon rien ,si il n y a pas de montagne dans cette province (c'est la même explication que la requête 1)

4) id(//country[@code="F"]/@Source)

Le type country ayant en attributs les river dont il est le pays source (idrefs) ,un pays peut avoir plusieurs rivières dont il est le source.
On accède a ces rivières par la fonction id comme l'attribut Source est de type IDREFS.

5) //country
[not(sum(border/(@length))<//country/sum(border/(@length)))]/data(@name)

Ici on prend le pays pour lequel il n'existe pas un pays qui a la somme de ces frontières supérieurs à la somme des frontières de ce pays.

La fonction sum(..) est une fonction de XPath 1.0 et donc comme précédemment on prend le max des pays avec le not(<).

EXERCICE 04 :

I)

Dans cette exercice j'ai décidé d'exécuter quelque requête XQuery . La DTD est définie de la sorte, chaque pays a en attribut la référence des continents auquel il appartient , c'est presque la même que celle de l'exercice 3 , à ceci près qu'ici on a plus la redondance de pays .

1) Le pays ayant la population maximal par rapport aux pays du même continent

```
for $con in //continent/@id
return
<country>
{//country[contains(@continent,$con)][not(number(@population)<
//country[contains(@continent,$con)]/number(@population))
]/data(@name)}
</country>
```

Ce qui donne comme résultat :

```
<country>Nigeria</country>   <country>United States</country>
<country>China</country>     <country>Indonesia</country>
<country>Russia</country>
```

2) les organisation ou aucun pays d'Europe n'appartiennent

```
let $orgEurope := distinct-
values(//country[@continent="europe"]/id(@isMember)/@id)
for $organ in //organization/@id
where not($orgEurope = $organ)
return <Organization>{//organization[@id=$organ]/@name}</Organization>
```

3) les pays frontalier qui appartiennent a deux pays donner

```
let $u := distinct-values(//country[@code="DZ"]/border/@country)
```

```
let $v :=distinct-values (//country[@code="LAR"]/border/@country)
for $w in $u
where $w=$v
return
<country>{//country[@id=$w]/@name}</country>
```

4) la longueurs de toute la frontière pour chaque pays

```
for $c in //country order by $c/@name
return<PopulationFrontalier>
    {$c/@name}<voisin_pop {sum($c/border/@length)}</voisin_pop>
      </PopulationFrontalier>
```

II)

Dans cette deuxième partie de l'exercice 4 j'ai exécuté quelque requête sur le fichier xml de l'exercice 3 ,a vrai dire les requêtes demandé transformer en sql ,l'utilisation des fonctions 'XMLQuery' et 'extractvalue' ,ainsi que 'extract' sont utilisées ,la premiere prend en argument un XQuery_string et l'autre un XPath_string.
XMLQuery peut renvoyer des nœuds comme résultat, Extractvalue ne renvoie qu'une valeur et pas de nœud sinon lève une exception.

1)
```
select XMLQuery('//continent/country[not(number(@population) <
parent::continent/country/number(@population))]/@name' passing
m.toXML() returning content).getClobVal()
from mondial m;
```

Renvoi Nigeria United State China Indonesia Russia

2)
```
select extractvalue(m.toXML(),'//province[@name="Nepal"]
/mountain[not(@height<parent::province/mountain/@height)]/@name')
from mondial m;
```

Renvoi Mount Everest

3)
```
select XMLQuery('//country [not(sum(border/(@length)) <
//country/sum(border/(@length)))]/data(@name)' passing m.toXML()
returning content).getClobVal()
from mondial m;
```

le Pays Chine est renvoyer ce qui est bien la réponse à la requête 5

4) La requête renvoie le nom et la longueurs de la frontière d'un pays ordonné par nom de pays

```
select XMLQuery('for $c in //country order by $c/@name
return<PopulationFrontalier>
    {$c/@name}<voisin_pop> {sum($c/border/@length)} </voisin_pop>
      </PopulationFrontalier>'
passing m.toXML() returning content).getClobVal()
       from mondial m;
```

5) Les rivières dont la France est source avec extract

```
select
m.toXML().extract('id(//country[@code="F"]/@Source)').getStringVal()
from mondial m;
```

III)

Dans cette troisième partie je me suis intéresser à la validation du fichier XML de l'exercice 3 .

Tout d'abord j'ai transformé le fichier en XSchema , le fichier en question est dans le le bloc pl/sql du fichier avec le nom ValidationSchema.Sql

Dans le bloc pl/sql j'enregistre ce fichier avec dbms_xmlschema.registerSchema qui prend en argument un nom de fichier xsd et prend le schéma ensuite il enregistre le fichier .

Après exécution du bloc pl/sql on lance la requête suivante :

```
select XMLISVALID(m.toXML(), 'SchemaEX3.xsd') from mondial m;
```

qui renvois 1 si le schéma est valide ou 0 sinon.