Projet MLBDA SQL3/XML

Nom : NAIT SLIMANI
Prenom : Kamel
Groupe :3

Introduction

Ce projet SQL3/XML Consiste à éxtraire de diefferente bases de donées XML correspendantes a des differentes DTD et ceci en utilisant une base de donnée relationnel SQL éxistante, ça nous a pérmit de bien decouvrir comment utiliser SQL Object et comment générer des fichiers XML conforme a des DTD specifiques avec de differentes variantes, par éxemple de differents attributs calculé au lieu de les stocker ou générer des fichiers XML qui puissent rependre a des requêtes XPATH.

«Pour tout les fichiers sql fournis il vous suffit de les lancer avec CTRL-E pour tout éxécuter et ça va vous générer :
- un ou plusieurs fichiers xml qui repondent au differentes questions de l'éxercice.
-la DTD correspedantes à chauque des question.
Vous pouvez directement utiliser un outil de validation xml vu que chaque fichiers xml est lié au nom de dtd généré.

*** toutes les Types et Tables crées seront supprimés et les requêtes de suppressions sont situées a la fin de chaque fichier»**

I. Exercice 1:

- L'éxercice 1 consiste a crée des fichiers qui sont conforme au dtd donnée la modelisation est faite d'une maniére a utiliser la puissance de sql3 et d'una maniére a eviter de creer de les jointures donc en on peut remarquer dans les **DTD1 et DTD2** que les elemetns en commun entre les deux deux DTD se reduise a country et mondial pour cela on a choisis de créer une table country que sera la bases pricipales des données puis on utilises des références a partir de organization et modial vers les pays correspendants alors

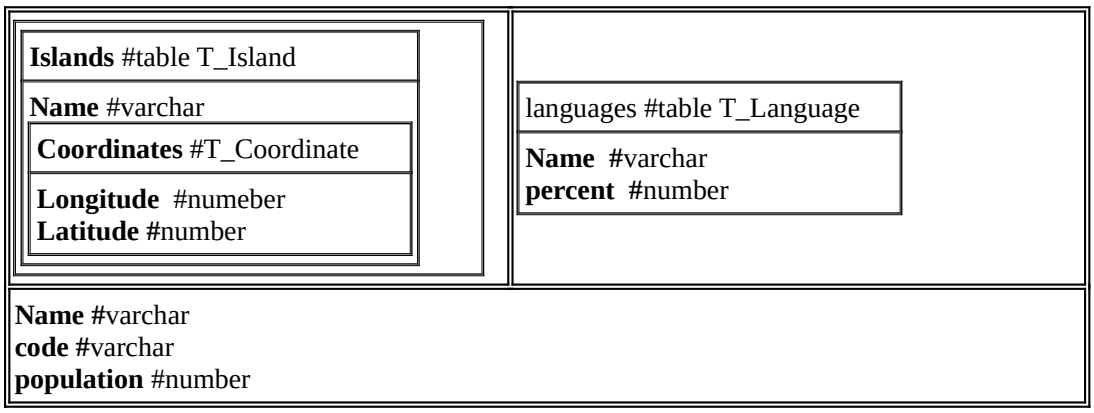
  "Dans ces deux tables vous avez la modelisation entiére du premier exercice a noter les element aprés les # signifient les types des attiribts qui se trouvent a ça droite."

La table mondial: elle contient une seule ligne

MONDIAL #T_Mondial	
ORGANIZATIONS #table T_Organization **headquarter** #T_Headquarter **Name** #varchar **Countrys** #table ref T_country @oracle121qaae…...	**COUNTRYS** #table ref T_country @oracle121fdqg…..

La table Countrys: elle contient toutes les information sur contry spécifiées dans les deux DTD

Countrys #Table T_Country	
Provinces #table T_province **name** #varchar **capital** #varchar **Moutains** #table T_mountain **Name** #varchar **height** #number **Derserts** #table T_Dersert **Name** #varchar **area** #number	Airports #table T_Airport **Name** #varchar **nearcity** #varchar **Continents** #table T_Continent **Name** #varchar **percent #**number **borders** #T_Borders **borders #**table T_Border **countryCode** #varchar **length #**number



Creation : Avec une telle modelisation on voit bien qu'on evité toutes jointure entre les types crée mais ce qui a crée une certaine difficulté dans de remplissage sur plusieur niveaux car on a que chaque pays contient la table de ses provinces , airports , languages , borders , continents et a leur tour les attributs qui sont composé comme provinces elle contient la table de ses montagnes , deserts et iles.Donc lors de la création on utilise une imbrication de nesteed table pour pouvoir instantier toutes les sous tables correspedantes et puis pour l'insertion on a utilisé
la fonction **cast collect** pour les convertir directement le resultat d'un select en un type ensembliste et ceci pour éviter de remplir chaque colonne séparément donc on a la table COUTRYS est remplise avec une seule reuqête et ainsi la table modial.

1. **Genération du XML :**

Les deux dtd du prmiers exercice sont générées par deux fonctions distinctives a partir de l'uniqe élément T_Mondial qui se trouve dans la table Mondial:

- La fonction **toXML1** nous génére le XMLType corresependant a la **DTD1** et ceci on parcourant les pays '**countrys**' qui est une table de référence et elle recupére le XMLType généré par chaque elle l'englobe dans un element **<Mondial>,** chaque pays a son tour le type pays a aussi deux fonction vu que c'est un élément en coummun entre les deux DTDs elle a une fonction **toXML1** qui recupére les XMLType a partir des élements spécifiques à la **DTD1** de chaque country (**continents** , **provinces** , **airports**) puis elle construit l'element **<country>** contenant touts les éléments XMLType correspendant à ces derniers en appellant la fonction **toXML** sur chaqu'un de ces attributs.

- La fonction **toXML2** nous génére le XMLType correspendant à la **DTD2** en parcourant les élements de **organizations** on appelle la fonction **toXML** des éléments T_Organization et a son tour lui il appélle tous ses elemets country qui sont dans sa table de reférence **countrys** de cet element en appelant la fonction **toXML2** de T_country qui recupére les (**borders, languages)** et on appelles leur fonction **toXML** pour recupérer les XMLType correspendants.

II. Exercice2:

2. Dans cette éxecice on réalisé les DTD demnadé en ayant des attributs XML qui sont calculés et non pas stockés donc la modélisation reste en pricipe la meme pour country de l'exercice 1.

Countrys #Table T_Country

Name #varchar
code #varchar

Provinces #table T_province

name #varchar
capital #varchar

Moutains #table T_mountain

Name #varchar
height #number
map function nameM return #varchar

Deserts #table T_Dersert

Name #varchar
area #number
map function nameD return #varchar

Islands #table T_Island

Name #varchar
map function nameI return #varchar

Coordinates #T_Coordinates

Longitude #numeber
Latitude #number

function geos return #T_geo
function peak return #number
function continent return #varchar2
function contCountries return #T_ContCountries
function Blength return #number
+les function XML

T_geo

ListM # table T_Mountain
ListD #table T_Dersert
ListI #table T_Island

T_ContCountries

borders #table T_Border

countryCode #varchar
length #number

les spécification que rapporte cette modelisation qu'on utilise directement les tables de la base mondial pour calculer des attributs pour le XML souhaité, donc pour la creation on a le même pricipe que le premier Execrcice on a pas d'attributs de jointure dans les types des sous tables des elements spécifique à cet élément.

Pour réaliser le cacul des attribut on a du créer deux table en plus qui sont my_encompasses et my_borders qui sont des copies des synonymes encompasses et borders respectivement et ce car on ne peut pas utiliser des synonumes privé qui nous appartienne pas dans les fonctions a l'intérieur des types :

- la fonction **geos()** recupére touts les éléments disitincs de toutes les provinces donc les montagnes les deserts et les iles et crée un ensemble pour chacun d'eux puis renvoie un types **T_geo** en lui passant les ensembles resultants en parametre la selection des object distinct n'est pas directement utilisable donc on a rajoute des fonctions map pour chaqu'un des types T_province , T_mountain,T_Desert qui renvoie le nom des elements pour qu'il puisse les comparer.

- La fonction **peak()** elle appelle la function geos () puis elle recupére le max de lattribut height et elle le renvoie si il est different de null si non elle renvoie 0 avec la function **Coalesce**.

- La fonction **continent()** renvoie un nom de continent a partir de la table my_enompasses elle recupére le continent avec le pourcentage maximum en filtrant avec le code du pays dans cette table.

- La fonction **contCountries()** qui renvoie un **T_contCountrie()** les voisins de ce pays qui sont dans le même continent et cela en faisant une jointure entre la table **countrys** des T_country et la table my_borders en filtrant sur le code du pays dans les deux cas soit il est egale a country1 ou contry2 puis on filtre que continent() du pays est egale a self.contient().
- La fonction **Blength()** on recupére la somme des longeur des longeures a partir de la table **my_borders** ou le code pays est le country1 ou le country 2

- **Generation du XML :**Dans le type T_Country il y a quatre fonction de génération de XMLType correspendante chacune a une DTD demandé :

    - **toXML1()** Renvoie le XMLType correspendant a la DTD1 et pour recupérer l'element **geo** elle appélle **geos().toXML()** et elle l'ajoute a l'element **<country>** et ce pour chaque country.

    - **toXML2()** Renvoie le XMLType correspendant a la **DTD2** et pour recupérer l'element **geo** elle appélle geos().toXML() et elle l'ajoute l'element <country> et ce pour chaque country puis on récupére le peak par la fonction **peak()** on verifie si

il n’est pas égale a zéro on rajoute un élement peak avec l’attribut height de ça valeur.
«**Dans la DTD2 l’element country on aura country aura l’attribut name**»

- **toXML3()** Renvoie le XMLType correspondant a la **DTD3** et pour recupérer l’element **contCountry** on appellant la fonction qui nous retourne contCountry puis on ajoute l’élement retourné a l’élement **<country>** qui aura deux attributs **name** et **continent** qui sera calculé par la fonction **contienent**()

- **toXML4**() Renvoie le **XMLType** correspondant a la **DTD4** et pour recupérer l’element **contCountry** on appellant la fonction qui nous retourne contCountry puis on ajoute l’élement retourné a l’élement <**country**> qui aura deux attributs name et **blength** qui sera calculé par la fonction **Blength**().

Le fichier XML compred aussi une fonction PL/SQL **toXML**(dtd **number**) en l’appelant sur avec un numéro de dtd (1,2,3,4) nous génre le xml type corespendant touts les pays on et on les met dans un element englobant **<ex2>** et nous renvoie une un XMLType qui contient le resultat pour la dtd specifiée.

III. Exercice3:

Dans cette éxercice pour rependre aux questions demandé on vous sugére la **DTD** suivante :

```
<!ELEMENT mondial ((continent|organization|river)*) >
 <!ELEMENT continent (country*) >
<!ATTLIST continent name CDATA #REQUIRED>
<!ELEMENT country (province*, border*) >
<!ATTLIST country name CDATA  #REQUIRED
                code ID #REQUIRED
                population CDATA #IMPLIED
                memberin IDREFS #IMPLIED
                sourceof IDREFS #IMPLIED>
<!ELEMENT province (mountain*) >
<!ATTLIST province name CDATA #REQUIRED
                 capital CDATA #REQUIRED>
<!ELEMENT mountain EMPTY>
<!ATTLIST mountain name CDATA #REQUIRED
                 height CDATA #REQUIRED
                 latitude CDATA #REQUIRED
                 longitude CDATA #REQUIRED>
<!ELEMENT organization EMPTY>
<!ATTLIST organization name CDATA #REQUIRED
                   id ID #REQUIRED
                   creationdate CDATA #REQUIRED>
<!ELEMENT border EMPTY>
<!ATTLIST border countryCode IDREF #REQUIRED
              length CDATA #REQUIRED>
<!ELEMENT river EMPTY>
<!ATTLIST river id ID #REQUIRED
      name CDATA #REQUIRED
      length  CDATA #IMPLIED>
```

La Modelisation est la même que la précédente donc on aura le schéma suivant:

Table Mondial

Mondial #table T_Mondial	
Contients #table T_Continent **Name** #varchar2 **Countrys** #table ref T_country @ASD1214……..	**Organizations** #table T_Organization **Name** #varchar **abreviation** #varchar **creation** #date **Rivers #table T_river** **Name** #varchar **length** #number

Table Countrys :

Countrys #Table T_Country	
Provinces #table T_province **name** #varchar **capital** #varchar **Moutains** #table T_mountain **Name** #varchar **height** #number **Coordinates** #T_Coordinate **Longitude** #numeber **Latitude #**number	**memberin** #table VARCHAR **Name #**varchar **nearcity #**varchar **sourceof** #table VARCHAR **Name #**varchar **percent #**number **borders** #T_Borders **countryCode #**varchar **length #**number
Name #varchar **code #**varchar **population** #number function continent return #varchar2	

1. **Creation :** Le schéma contient que deux tables explicites comme spécifié dans le shéme et elle sont les tables Mondial et countrys donc lors de la creation on a utilisé plusieur nesteed table en profondeur pour instantier les tables qui sont contenu dans des attributs et pour l'insertion on a inseré a chaque table avec plusieurs requêtes impbiriqué et en utilisant toujours **cast Collect.**

2. Genération XML et reuquêtes XPath:

1. La reqête correspedante à la premiere question est :
   //continent/country[
   not (number(@population)
   <
   ../country/number(@population))]/@name
   Donc on a pas de spécification il nous suffit d'avoir le conntry dans continent et qu'il contienne l'attribut population.

2. La reqête correspedante à la dexiéme question est :
   id(//country[@code='F'] /data(@memberin))

   On a utilisé le fait qu'on peut accéder a des élements dans xpath par les idrefs dans organization l'attribut **id** retourné par le xml est unique et ainsi on remplie la table country on utilisant ce pricipe donc l'attribut memberin contient les même id qui existe dans organization et vu qu'il est de mandé que les organization soient trié par la date de creation expérimentalement on a pu voir que l'evulation des requête en Xpath depends de l'ordre des élements dans le document et non pas de l'ordre des idrefs donc lors de notre insertion dans oraganization on a utilisé un order by et vu que ces derniéres elles sont traité dans le même ordre donc on a les bons résultats triés.
3. La reqête correspedante à la troisiéme question est:
   //province[@name='Rhone Alpes']/
   mountain[not(@height<../mountain/@height)]

   Pour celle ci il nous suffit d'avoir les les montagnes qui soient dans province et qu'elles aient un attribut height correspendant a leur altitude.

4. La reqête correspedante à la Quatriéme question est:
   id(//country[@code='F']/data(@sourceof))

   Pour celle ci on a toujours utilisé le prinipe des idrefs donc on eu besoin d'unifier les identifiants donc on a utilisé est la fonction de replace pour pouvoir et on a ajouté 'riv-' dériere pour que ça soit plus distinctife et dans l'attribut source of de country on a tous les identifiant dont le pays est source.

5. La reqête correspedante à la cinquiéme question est:
   //country[not(sum(data(border/@length)) <
   //country/sum(data(border/@length)))]/@name

Pour celle ci on utilise seleument le fait d'avoir toutes les bordures des pays donc on les somman avec la fonction sum() bien existante dans xpath1.0 si non on aurait pu ajouter un attribut blength comme l'execice précédent pour pouvoir le tester directement sur cette attribut.