MODÈLES ET LANGAGES BASES DE DONNÉES AVANCÉES

Célina Khalfat, M1 DAC

Table des matières

I – Les fichiers

J’ai préféré séparer mon travail en plusieurs fichiers pour mieux m’y retrouver. De plus, ce sera sûrement plus clair pour vous, pour vous y retrouver. Pour pouvoir générer les fichiers XML demandés pour le projet, il faut dans un premier temps, exécuter le fichier "*creation-insertion.sql*" qui créera les objets, les tables et les ensembles. De plus, ce fichier insère les données dans les tables. Dans un deuxième temps il faudra exécuter le fichier "*methode.sql*" qui contient toutes les fonctions nécessaires à la réalisation des documents XML. Le fichier "*ex.sql*" contient les lignes de codes nécessaires pour générer les documents XML, les commentaires précisent quel fichier sera généré. Vous pouvez également supprimer tous les objets, tables et ensembles de votre base de données grâce aux fichiers "*suppression.sql*". Je me suis également permis de joindre les DTD correspondant aux fichiers XML, ça rend la validation plus simple. Les fichiers générés contiennent dans leur nom, le numéro de leur DTD.

Tous les fichiers XML sont conformes à leur DTD, il suffit de copier/coller les DTD fournies au début du document généré.

J'ai également ajouté le fichier Xschema, comme exercice d'ouvertrure, qui valide le document de l'exercice 1 question 1.

II – Les objets, leur tables et leurs méthodes

Pour pouvoir générer les différents documents XML j’ai dû créer un certain nombre d'objets, de tables et d’ensembles. Pour chaque objet j’ai récupéré les attributs dont j’avais besoin dans les tables fournies de la base mondiale. En créant les tables, j’ai ajouté certaines contraintes sur les attributs. La plupart des objets ont des fonctions que détaillées ci-après.

A) Le monde

L’objet `T_Mondial` permet d’ajouter la balise *<mondial>...</mondial>* autour de chaque document. L’attribut n’est pas utile, il est seulement là pour pouvoir créer la table `Mondial`. La table contient un seul élément, cela permet que le fichier XML ne soit pas répété plusieurs fois. C’est à partir de cet objet, grâce à la fonction `member function toXML(i number) return XMLType`, que la génération du XML commence. L’argument *i* permet de spécifier, au sein de la fonction, grâce à un switch case, quelle autre fonction doit être appelée et ainsi pouvoir poursuivre la génération du document, autrement dit, le i permet de demander un document en particulier. Par exemple,

```
WbExport -type=text
         -file='mondialDTD1.xml'
         -createDir=true
         -encoding=UTF-8
         -header=false
         -delimiter=','
         -decimal=','
         -dateFormat='yyyy-MM-dd'
/
select m.toXML(1).getClobVal()
from Mondial m;
```

permet de générer le fichier XML qui correspond à la DTD1 (la question 1 de l'exercice 1).

B) Les continents

L'objet `T_Continent` a comme attribut `name`, qui correspond au nom d'un continent. La table `TheContinents` contient ainsi les noms de tous les continents. Le nom d'un continent ne peut pas être null et cela a été spécifié lors de la création de la table. La fonction `member function toXML return XMLType` associé à cet objet, permet de générer la sous-partie *<continent>...</continent>* du document de l'exercice 3. Je reviendrai plus tard sur la DTD choisie pour cet exercice.

Il y a également l'objet `T_Encompasses` qui permet de faire le lien entre les pays (sous-partie suivante) et leur proportion dans un continent, grâce aux attributs `country`, `continent` et `percentage`. On peut également trouver la fonction `member function toXML return XMLType` qui permet d'écrire dans le document toutes les lignes de ce type : *<continent name="'||continent||'" percent="'||percentage||'"/>.*

C) Les pays, leur langages et leur frontières

L'objet qui représente les pays est T_Country. Il contient principalement le nom, le code, la population. Il possède également de nombreuses fonctions, les objets pays étant souvent manipulés. Nous pouvons y trouver d'une part les fonctions :

- `member function peak return VARCHAR2`
- `member function continent return VARCHAR2`
- `member function lengthborders return number`

qui ont été utilisées dans l'exercice 2, ce sont les fonctions qui permettent de calculer la valeur d'un attribut et d'autres part un certain nombre de fonctions qui retournent des XMLType pour la génération des documents XML.
Pour savoir quels pays sont frontaliers, j'ai créé l'objet `T_Borders`. Ce dernier possède également une fonction qui prend en argument le code d'un pays et qui renvoie un XMLType avec toutes ses frontières. De plus T_Language permet de dire que dans tel pays, ce langage est parlé par tant de pourcent de la population. Country et name sont not null et le pourcentage est supérieur ou égale à 0.

D) Les provinces

Les informations sur les provinces sont dans un objet nommé `T_Province`. Ce dernier contient des informations sur le nom de la province, son pays, sa population et sa capitale. Tous sont dans la table `TheProvinces`. Le nom de la province et du pays ne doivent pas être null. De plus, la valeur de l'attribut population, qui est un nombre, ne doit pas être négative. Cet objet possède deux fonctions. La première liste les montagnes, déserts et îles associés à cette province. La deuxième quant à elle, liste les montagnes et les rivières associées à une province. Bien que ces deux fonctions auraient pu être factorisées pour n'en former qu'une, et à l'aide qu'un boolean, savoir quelles informations seront affichées, j'ai fait le choix de ne pas le faire pour bien différencier leur utilisation dans deux exercices distincts.

E) Les montagnes

Toutes les données concernant les montagnes se trouvent dans les objets `T_Mountain` et `T_GeoMountain`. T_Mountain a comme attribut `name`, `height`, `latitude`

et `longitude`. Les contraintes sur ces objets sont que le name ne doit pas être null, la valeur de la latitude doit être comprise entre -90 et 90, et celle de la longitude entre -180 et 180. La fonction `member function toEx3XML return XMLType` permet d'afficher les lignes correspondant à : *<mountain altitude="'||height||'" latitude="'|| latitude||'" longitude="'||longitude||'">' ||name||'</mountain>* pour le document de l'exercice 3. De même pour `member function toXML return XMLType` sauf qu'elle n'affiche pas les attributs de latitude et longitude. Le deuxième objet, `T_GeoMountain` permet de relier un nom de montagne à son pays et sa province. Les attributs `mountain`, `country` et `province` ne peuvent pas être null.

F) Les déserts et les îles

De la même façon que pour les montagnes, pour représenter les déserts je me suis appuyée sur deux objets T_Desert et T_GeoDesert, et pour représenter les îles T_Island et T_GeoIsland. T_Desert et T_Island ont tous les attributs nécessaires pour décrire ses éléments, en plus d'une fonction `member function toXML return XMLType` qui permet de mettre les valeurs de chaque attribut dans un fichier XML. J'ai fait le choix de créer des fonctions et pas seulement ajouter des *XMLType(...)* lorsque j'en avais besoin, pour que le code soit modifiable plus facilement : si j'avais besoin de modifier la manière dont j'affiche mes attributs, j'avais seulement à modifier cette fonction, et pas toutes les occurrences de *XMLType('<desert>...</desert')* par exemple. Quant aux objets et T_GeoDesert et T_GeoIsland, ils permettent de faire le lien avec les provinces et les pays.

H) Les aéroports

L'objet `T_Airport` contient les trois attributs suivants : `name`, `country` et `city`. Il n'y a pas de contrainte not null sur la ville, ainsi il existe des aéroports dont l'attribut city ne contient rien. Pour traiter ce cas, dans la fonction `member function toXML return XMLType` associé à l'objet, il y a un test qui vérifie que l'attribut city d'un objet T_Aeroport ne soit pas null, si c'est le cas au lieu de trouver la ligne *<airport name="'||name||'" nearCity="'||city||'"/>* dans le document, *'<airport name="'|| name||'"/>'* sera trouvée. Je trouvais que cela rendait le document plus joli et moins lourd, plutôt que simplement mettre une chaîne vide comme valeur de nearCity. La table qui contient les objets T_Airport est `TheAirports`.

I) Les organisations

Concernant les organisations, on peut trouver dans l'objet `T_Organization`, les attributs `abbreviation` (not null), `name` (not null), `city`, `country` et `established`. Ainsi qu'une fonction `member function toXML return XMLType`, qui pour une instance de T_Objet, listera les pays membres et le siège social. Les pays membres sont retrouvés grâce à l'objet `T_IsMember` qui permet de dire quels sont les pays membres d'une certaine organisation.

J) Les rivières

Toutes les informations nécessaires pour les documents demandés relatives aux rivières sont dans l'objet `T_Source`, on y retrouve le nom de chaque rivière ainsi que la province et pays d'où est sa source. Ces informations sont stockées dans une table

`TheSources`. Ces données m'ont permit d'ajouter la ligne suivante : *<river>*'||*name*||'*</river>*' lors de la génération du document de l'exercice 3.

III – Les ensembles

Dans les fonctions, il était courant d'avoir besoin d'ensemble pour stocker le résultat des requêtes. Par exemple, il a fallu que je crée un ensemble T_ensAirports pour stocker les objets T_Airport qui correspondent à tous les aéroports d'un même pays.

```
select value(a)
       bulk collect into ensAirports
       from TheAirports a
       where code = a.country;
```

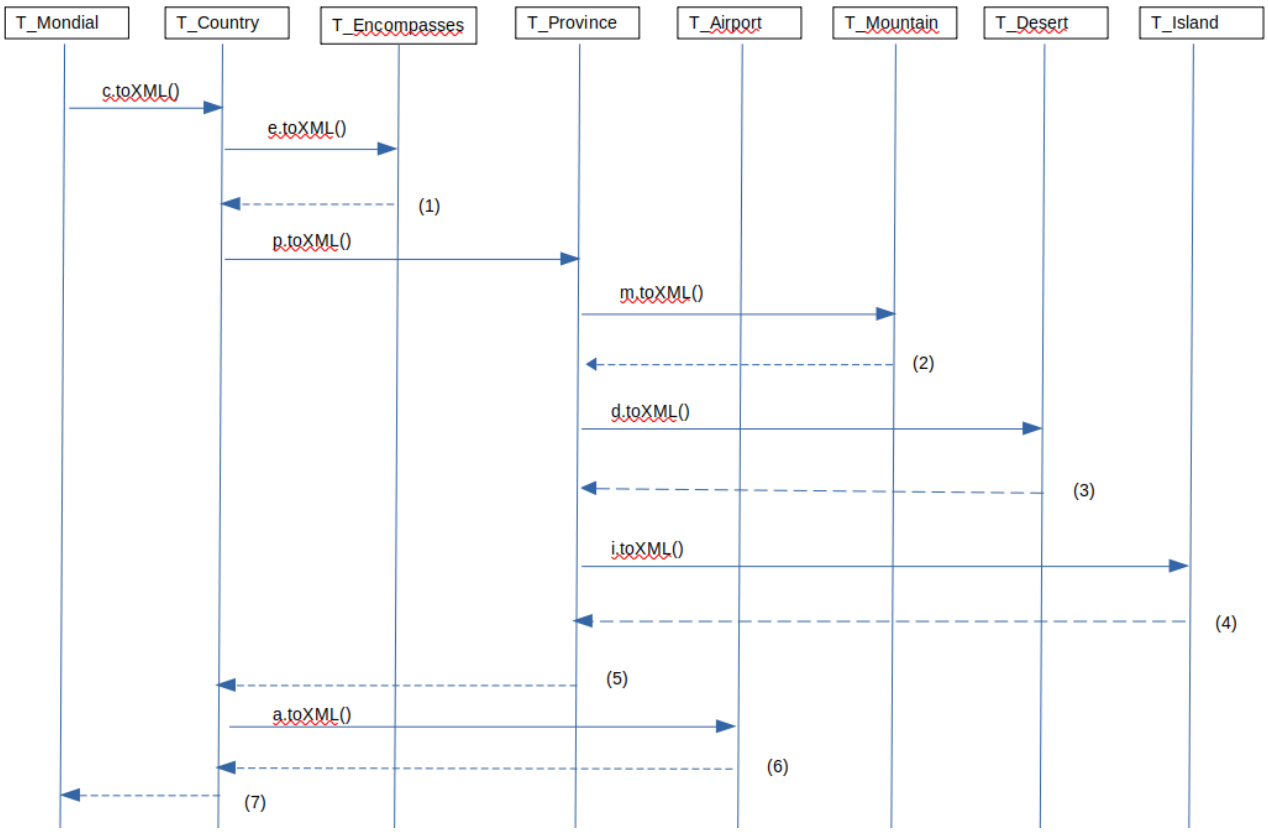
(avec ensAirports de type T_ensAirports)

Vous trouverez tous les ensembles créés dans le fichier "*creation-insertion.sql*" à partir de la ligne 370.

IV – Les exercices

A) Exemple de procédure mise en place pour la génération de document

Je vais ici vous expliquer comment j'ai procédé pour générer le document XML correspondant à la première DTD autrement dit le lien entre chaque fonction. J'ai appliqué la même méthode pour les autres documents.



Ces fonctions sont appelées seulement si les requêtes associées contiennent des résultats, par exemple d.toXML() ne sera pas appelé si la province ne contient pas de désert.

Pour notre exemple supposons que nous avons qu'un seul pays de nom 'P' et qui a pour code '01'. Ce pays se trouve dans le continent 'C' à 90 % et possède un aéroport nommé 'A' dans une ville nommée 'city'. Il a une province 'Pro' dont la capitale est 'Cap' qui a une montagne 'M' d'altitude '1777 et une île 'I' qui a pour coordonnées (17,17). Cette province n'a pas de désert.

Avec cette simulation, nous allons exécuter la requête en II A) avec le même paramètre 1, qui ouvrira la balise mondial, `c.toXML` va permettre l'ouverture de la balise <country>. Dans country nous voulons des informations concernant le continent, les provinces et les aéroports, dans cet ordre. Donc `e.toXML()` est appelé en premier, en (1) nous aurons comme valeur un XMLType qui contiendra : `<continent name="C" percent="90"/>`. Pour les provinces, après avoir appelé m.toXML qui renverra comme XMLType `<mountain name="M" height="1777"/>`, `d.toXML()` qui renverra `<desert name="D"/>` et `i.toXML()` qui renverra

```
 <island name="I">
        <coordinates latitude="17" longitude="17"/>
 </island>
```

p.toXML() pourra enfin renvoyer sa valeur de retour qui est donc un enchaînement du tout entre des balises <province> comme suit :

```
<province name="Pro" capital="Cap">
      <mountain name="M" height="1777"/>
      <island name="I">
        <coordinates latitude="17" longitude="17"/>
      </island>
    </province>
```

Puis dans la fonction de T_Country, on appelle a.toXML qui renvoie à son tour un XMLType comme suit : `<airport name="Al" nearCity="city"/>`
La fonction toXML() de T_Country se finit en retournant :

```
<country idcountry="P1" nom="P">
    <continent name="C" percent="90"/>
    <province name="Pro" capital="Cap">
      <mountain name="M" height="1777"/>
      <island name="I">
        <coordinates latitude="17" longitude="17"/>
      </island>
    </province>
    <airport name="Al" nearCity="city"/>
  </country>
```

La fonction T_Mondial a fini par recevoir toutes les informations dont elle avait besoin et peut renvoyer le XMLType complet, c'est à dire

La fonction toXML() de T_Country se finit en retournant :

```
<mondial>
  <country idcountry="P1" nom="P">
    <continent name="C" percent="90"/>
    <province name="Pro" capital="Cap">
      <mountain name="M" height="1777"/>
      <island name="I">
        <coordinates latitude="17" longitude="17"/>
      </island>
    </province>
    <airport name="Al" nearCity="city"/>
```

```
  </country>
</mondial>
```

Et grâce aux premières lignes (celles avant la requête), la requête sera retranscrite dans un fichier XML.
Ceci était un exemple simplifié, chaque fonction est appelée autant de fois qu'il y a d'éléments. Par exemple, si dans le monde il y a 20 pays, c.toXML sera appelée 20 fois à l'aide d'une boucle for.

B) Les calculs d'attributs

Dans l'exercice 2 il était demandé que la valeur d'un attribut était le résultat d'un appel de méthode. Le premier attribut était height qui correspondait à l'altitude de la plus haute montagne du pays. La fonction `member function peak return VARCHAR2` dans T_Country, grâce à une requête, trouve le nom de la montagne qui a comme altitude la plus grande des altitudes de toutes les montagnes du pays. Si cette requête ne renvoie aucun résultat, la chaîne de caractère '0' est retournée, sinon le nom de la montagne est retournée. Par la suite, on trouve cette ligne dans une fonction de T_Country : `XMLType.createxml('<peak name="'||self.peak()||'"/>');`

On peut également trouver la fonction `member function continent return VARCHAR2` dans T_Country, qui à l'aide d'une requête et de la table T_Encompasses renvoie pour un pays le continent dominant. Par exemple pour la Russie, la fonction renverra « Asia ». Cette fonction est appelée de cette manière :

```
XMLType.createxml('<country name="'||name||'" continent="'||self.continent()||'"/>');
```

Pour finir, le dernier attribut qu'il fallait calculer était blength, qui calculait la longueur totale de sa frontière. C'est la fonction `member function lengthborders return number` dans T_Country qui nous permet de calculer ce résultat, grâce à la fonction sum qui additionne toutes les longueurs de frontières pour un pays. Cette fonction est appelée comme cela : `XMLType.createxml('<country name="'||name||'" blength="'||self.lengthborders()||'"/>');`

C) Exercice 3

Cet exercice était celui qui permettait d'être le plus libre dans son élaboration. J'ai ainsi formé la DTD suivante pour pouvoir répondre aux 5 requêtes demandées :

```
<!DOCTYPE mondial [
<!ELEMENT mondial (continent+)>
<!ELEMENT continent (country+)>
<!ELEMENT country (province*,  organizations?)>
<!ELEMENT province (mountains?, rivers?)>
<!ELEMENT mountains (mountain*)>
<!ELEMENT rivers (river+)>
<!ELEMENT organizations (organization+)>
<!ELEMENT river (#PCDATA)>
<!ELEMENT mountain (#PCDATA)>
<!ELEMENT organization (#PCDATA)>

<!ATTLIST continent name CDATA #REQUIRED>
<!ATTLIST province name CDATA #REQUIRED>
<!ATTLIST country name CDATA #REQUIRED
                  population CDATA #REQUIRED
                  borderslength CDATA #REQUIRED>
<!ATTLIST mountain  altitude CDATA #REQUIRED
                    latitude CDATA #REQUIRED
                    longitude CDATA #REQUIRED>
<!ATTLIST organization date CDATA #IMPLIED>
]>
```

Dans un premier temps j'ai décidé d'organiser les pays selon leur continent pour pouvoir faire la requête 1 :

```
//continent/country[@population=max(../country/@population)]
```

Ici, la subtilité était de bien penser à remonter dans l'arborescence d'un cran pour calculer selon les continents et ne pas prendre un chemin relatif. J'ai ainsi afficher chaque pays avec toutes les caractéristiques. Pour afficher seulement les noms il faut ajouter /@name.
Ainsi l'élément mondial a au moins un continent et chaque continent a au moins un pays. L'élément country a comme attribut son nom, sa population et la taille totale de ses frontières.

La deuxième requête faisait référence aux organisations et en particulier il fallait lister les organisations dont chaque pays était membre dans l'ordre croissant de date de création. Ainsi dans l'élément country, il peut y avoir ou non un élément organisations qui lui-même est constitué d'une liste d'organisations. J'aime bien l'idée de regrouper toutes les organisations dans une balise supérieure. J'ai fait de même pour les montagnes et rivières.
Comme mountains, rivers et organizations ne sont pas obligatoires, grâce au symbole « ? », si cette balise existe, cela implique qu'au moins un élément sera présent. Dans mon code j'ai fait en sorte de ne pas afficher les balises englobantes si rien n'allait les remplir, pour ne pas encombrer le fichier.
Mon objectif était qu'il soit le plus lisible possible avec simplement un coup d'œil : qu'on remarque directement les différentes grosses parties. C'est également pour cette raison, que les noms des organisations, des rivières et montagnes ne sont pas des attributs mais mis entre les balises, pour une question de lisibilité.
Ainsi pour un pays donné, prenons par exemple la France, la requête 2 correspond :

```
//country [@name ='France']/organizations/organization/text()
```

Avec la requête //country [@name ='France']/organizations/organization, on peut voir que les dates sont effectivement bien triées dans l'ordre croissant. Certaines organisations n'avaient pas de dates de création, elles se trouvent en dernière position. Ces dernières ne possèdent pas d'attribut date, c'est pourquoi cet attribut est #IMPLIED.

Pour chaque pays, il est possible d'avoir une liste de provinces. On trouve ensuite les montagnes et rivières qui se trouvent dans ces provinces, et implicitement dans le pays de cette province. Par exemple pour afficher la description de la plus haute montagne de la province 'Albania', on effectue la requête suivante :

```
//province[@name='Albania']/mountains/mountain[@altitude=max(//
    province[@name='Albania']/mountains/mountain/@altitude)]
```

On affiche la montagne dont l'altitude est égale à la plus haute altitude trouvée en Albania. On affiche directement l'élément mountain pour avoir des informations sur l'altitude, la latitude, la longitude ainsi que son nom.

Le document XML a été créé de sorte que seules les rivières qui prennent source dans ce pays soient les éléments river, ainsi pour y accéder, la requête est la suivante :

```
//country[@name='Russia']//rivers/river/text()
```

Ici, le pays est la Russie, mais il peut bien évidemment être remplacé par un autre.
Contrairement à la première requête, maintenant on cherche le pays qui a la plus longue frontière parmi tous les continents confondus, on peut donc procéder de la sorte :

```
//country[@borderslength=max(//country/@borderslength)]/@name
```

J'ai pris les décisions de faire les « gros » calcul en SQL3 comme la somme des frontières dans un attribut, ou encore de trier en amont les dates de créations des organisations car je ne voyais pas l'intérêt de compliquer les requêtes Xpath sachant que j'avais la main sur la création du document. Je trouvais cela plus ingénieux de faire le travail en amont et d'avoir simplement à naviguer dans le document pour récupérer les informations dont j'avais besoin. Je me suis également permise d'utiliser la fonction « max » de Xpath, je pouvais le faire moi même mais je trouve ça dommage de pas utiliser ces petites fonctions déjà faites. J'aime bien utiliser les nouveaux outils que les langages mettent à notre disposition.

La génération de ce document se fait en exécutant le fichier ex3.sql. Les étapes sont les mêmes que celles expliquées pour les exercices précédents.