Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg
University of Applied Sciences of Fribourg

Travail pratique
Oracle & XML

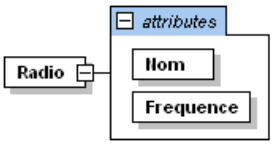
Houda CHABBI DRISSI

1 Solutions

1.1 Du relationnel vers XML

Le but de cet exercice était de transformer directement des données relationnelles en données XML grâce aux fonctions « XMLElement() », « XMLAttributes() », « XMLAgg() » et « XMLForest() » de Oracle. Voici les requêtes qui étaient demandées pour cet exercice :

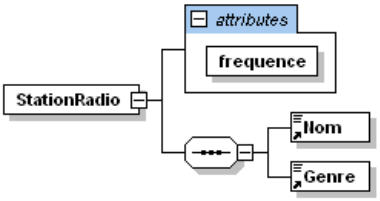
1. Afficher toutes les radios disponibles à Fribourg sous la forme d'une liste d'éléments « Radio » contenant les informations suivantes sous la forme d'attributs :




```
SELECT
  XMLElement("Radio",
    XMLAttributes(r.Nom as "nom", f.Frequence as "freq")
  )
FROM Radios r, Frequences f, Villes v
WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille AND v.Nom='Fribourg';
```

Pour cette requête, nous utilisons la fonction « XMLElement() » afin de créer un élément nommé « Radio » pour chaque ligne retournée par la requête. Puis, nous utilisons également la fonction « XMLAttributes() » directement à l'intérieur de la fonction « XMLElement() » afin d'attacher à chaque élément « Radio », une liste d'attributs. Dans notre cas, ces attributs se nomment « nom » et « freq » et leurs valeurs proviennent des champs « Nom » de la table « Radios » et « Frequence » de la table « Frequences ».

2. Afficher toutes les stations de radio disponible à Genève sous la forme suivante :



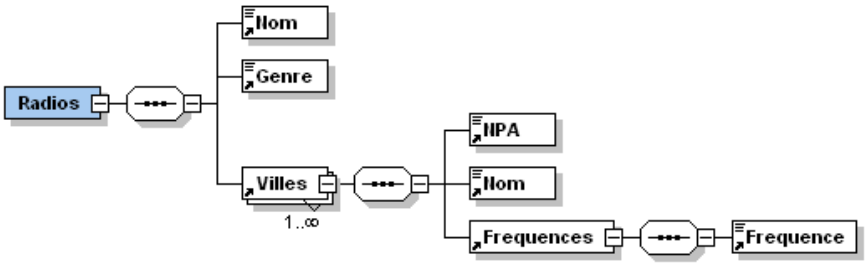

```
SELECT
  XMLElement("StationRadio",
    XMLAttributes(f.Frequence as "frequence"),
    XMLElement("Nom", r.Nom),
    XMLElement("Genre", r.Genre)
  )
```

```
FROM Radios r, Frequences f, Villes v
WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille AND v.Nom='Genève';
```

Dans cette requête, la fonction « XMLElement() » tout d'abord utilisée afin de créer un élément « StationRadio » pour chaque ligne du résultat. Puis, la fonction « XMLAttributes() » est utilisée pour ajouter une attribut « frequence » à cet élément et enfin, deux appels à la fonction « XMLElement() » permettent de lui ajouter encore deux éléments fils.
Dans la deuxième variante possible pour cet exercice, ces deux appels à la fonction « XMLElement() » sont remplacés par un seul appel à la fonction « XMLForest() ».

```
SELECT
  XMLElement("StationRadio",
    XMLAttributes(f.Frequence as "frequence"),
    XMLForest(r.Nom as "Nom", r.Genre as "Genre")
  )
FROM Radios r, Frequences f, Villes v
WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille AND v.Nom='Genève';
```

3. Afficher les fréquences des radios en les regroupant par radios, puis par ville. Le résultat ne doit contenir que des éléments (pas d'attributs). Sa structure doit être semblable à celle décrite ci-dessous :



Etape 1 : Affichage du nom et du genre de toutes les radios

```
SELECT
  XMLElement("Radios",
    XMLElement("Nom", r.Nom),
    XMLElement("Genre", r.Genre)
  )
FROM Radios r
```

La fonction « XMLElement() » permet de créer un élément « Radios » contenant deux sous-élément « Nom » et « Genre » ayant comme contenu les valeurs des champs « Nom » et « Genre » de la table « Radios ».

Etape 2 : Affichage des radios avec les villes dans lesquelles elles peuvent être captées

```
SELECT
  XMLElement("Radios",
    XMLElement("Nom", r.Nom),
    XMLElement("Genre", r.Genre),
    (SELECT
```

```
      XMLAgg(
        XMLElement("Ville",
            XMLElement("NPA", v.NPA),
            XMLElement("Nom", v.Nom)
        )
      )
      FROM Villes v
      WHERE exists(SELECT * FROM Frequences f WHERE f.IdVille=v.IdVille AND
f.IdRadio=r.IdRadio)
    )
  )
FROM Radios r
```

La partie complétée est affichée en gras. Dans cette partie, un nouveau SELECT est exécuté afin de rechercher toutes les villes dans lesquelles la radio courante peut être captée.
Les différentes fonctions « XMLElement() » sont utilisées afin de formater les données retournées et la fonction « XMLAgg() » permet de regrouper chaque élément « Ville » retourné dans l'élément « Radio » courant.

Etape 3 : Affichage des fréquences des radios (requête complète)

```
SELECT
  XMLElement("Radios",
    XMLElement("Nom", r.Nom),
    XMLElement("Genre", r.Genre),
    (SELECT
      XMLAgg(
        XMLElement("Ville",
          XMLElement("NPA", v.NPA),
          XMLElement("Nom", v.Nom),
          (SELECT
            XMLAgg(
              XMLElement("Frequence",
               XMLElement("Valeur",f.Frequence)
              )
            )
            FROM Frequences f WHERE f.IdVille=v.IdVille AND f.IdRadio=r.IdRadio
          )
        )
      )
      FROM Villes v
      WHERE exists(SELECT * FROM Frequences fr WHERE fr.IdVille=v.IdVille AND
fr.IdRadio=r.IdRadio)
    )
  )
FROM Radios r
```

La partie complétée est en gras. Dans cette partie, un troisième SELECT est effectué afin de trouver la fréquence de la radio courante dans la ville courante.
Le résultat est formaté et inséré dans l'élément « Ville » courant grâce aux fonction « XMLElement() » et « XMLAgg() ».

Il est encore possible de simplifier quelque peu la requête ci-dessus en remplaçant les suites d'appels à la fonction « XMLElement() » utilisés pour créer les fils d'un élément XML par un seul appel à la fonction « XMLForest() ». En effet, cette fonction

permet de générer une forêt (une suite) d'élément XML d'après une liste de valeurs provenant d'une table.

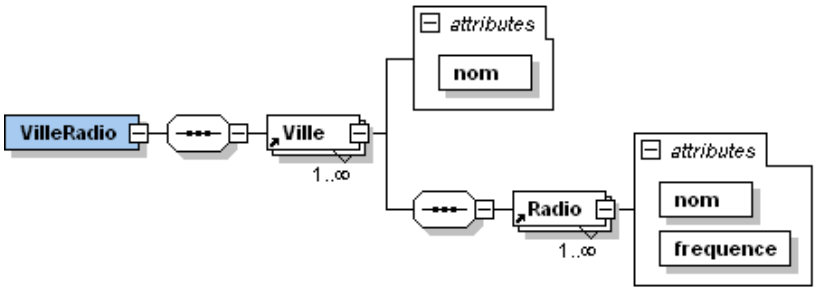
Voici la requête utilisant les fonctions « XMLForest() » :

```
SELECT
  XMLElement("Radios",
    XMLForest(r.Nom as "Nom", r.Genre as "Genre"),
    (SELECT
      XMLAgg(
        XMLElement("Ville",
          XMLForest(v.NPA as "NPA", v.Nom as "Nom"),
          (SELECT
            XMLAgg(
              XMLElement("Frequence",
               XMLElement("Valeur",f.Frequence)
              )
            )
            FROM Frequences f WHERE f.IdVille=v.IdVille AND f.IdRadio=r.IdRadio
          )
        )
      )
      FROM Villes v
      WHERE exists(SELECT * FROM Frequences fr WHERE fr.IdVille=v.IdVille AND
fr.IdRadio=r.IdRadio)
    )
  )
FROM Radios r
```

Autre version : Voici la requête utilisant le « group by » et le « order by » :

```
-- On peut ajouter encore une entrée pour voir l'effet
-- insert into frequences values (400, 1,1, -32);
SELECT
  XMLElement("Radios",
    XMLElement("Nom", r.Nom),
    XMLElement("Genre", r.Genre),
    XMLElement("Villes",
                XMLAgg(
                  XMLElement("Ville",
                     XMLElement("NPA", v.NPA),
                     XMLElement("Nom", v.Nom),
                     XMLElement("Frequences",
                        (SELECT
                           XMLAgg(XMLElement("Valeur",f1.Frequence))
                         FROM Frequences f1
                         WHERE f1.IdVille=f.IdVille AND f1.IdRadio=f.IdRadio)))
                  order by v.Nom asc)))
FROM Radios r, Frequences f, Villes v
WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille
Group by r.nom,r.genre;
```

4. Afficher la liste des radios disponibles dans chaque ville avec leurs fréquences respectives sous la forme suivante :



Etape 1 : Affichage de toutes les villes

```
SELECT
  XMLElement("Ville",
    XMLAttributes(v.Nom as "Nom")
  )
FROM Villes v;
```

Ici, les fonctions « XMLElement() » et « XMLAttributes() » sont utilisées afin de créer une série d'éléments « Ville » ayant leur nom en tant qu'attribut.

Etape 2 : Affichage des radios disponibles dans chaque ville

```
SELECT
  XMLElement("Ville",
    XMLAttributes(v.Nom as "Nom"),
    (SELECT
      XMLAgg(
        XMLElement("Radio",
          XMLAttributes(r.nom as "nom", f.Frequence as "frequence")
        )
      )
      FROM Radios r, Frequences f
      WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille
    )
  )
FROM Villes v;
```

La partie mise à jour est affichée en gras. Dans cette partie, un deuxième SELECT nous permet de rechercher toutes les radios disponibles dans la ville courante. Les fonctions « XMLElement() » et « XMLAttributes() » sont utilisées pour le formatage des données et la fonction « XMLAgg() » permet de lier le résultat de cette sous requête aux résultats du premier SELECT (liste des villes).

Etape 3 : Encapsulation des résultats dans une racine unique nommée « VilleRadio »

```
SELECT
      XMLElement("VilleRadio",
            XMLAgg(
              XMLElement("Ville",
                XMLAttributes(v.Nom as "Nom"),
                (SELECT
                  XMLAgg(
                    XMLElement("Radio",
```

```
                    XMLAttributes(r.nom as "nom", f.Frequence as "frequence")
                  )
                )
                FROM Radios r, Frequences f
                WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille
              )
            )
          )
      )
FROM Villes v;

Ou encore avec les « order by »
SELECT
      XMLElement("VilleRadio",
            XMLAgg(
              XMLElement("Ville",
                XMLAttributes(v.Nom as "Nom"),
                (SELECT
                  XMLAgg(
                    XMLElement("Radio",
                      XMLAttributes(r.nom as "nom", f.Frequence as "frequence")
                    )
                   order by r.nom
                  )
                  FROM Radios r, Frequences f
                  WHERE r.IdRadio=f.IdRadio AND v.IdVille=f.IdVille))
             order by v.nom
            )
      )
FROM Villes v;
```

Dans cette dernière étape, nous utilisons la fonction « XMLElement() » afin de créer l'élément racine « VilleRadio » ainsi que la fonction « XMLAgg() » afin d'encapsuler le résultat de la requête du point 2 dans cet élément racine.

1.2 XML → XML

1.2.1 Requêtes avec les outils clients

1.2.1.1 Requêtes de lecture

Le but de cet exercice était d'écrire les requêtes correspondantes aux questions ci-dessous à l'aide des outils mis à disposition par Oracle :

1. Tous les titres des DB triés par ordre alphabétique (utiliser les méthodes « XMLQuery() » et « XMLCast()) »)

```
SELECT XMLQuery('
for $bd in //bd
order by $bd/titre
return $bd/titre' passing BD_LIST returning content).getStringVal()
FROM BD
```

La fonction « XMLQuery() » de Oracle permet d'exécuter des requêtes XQuery sur des données stockées dans un champ du type XMLType. Les résultats de cette fonction sont retournés également au format XML.

Cette fonction reçoit en paramètre la requête a exécuter ainsi que quelques information pour son exécution. La clause « PASSING » permet de spécifier le champ XMLType sur lequel la requête devra être exécutée. La clause « RETURNING CONTENT » permet de spécifier que le résultat retourné sera un document ou un fragment de document conforme au format XML 1.0.
Au niveau XQuery, la clause « for » permet de récupérer tous les éléments « BD » puis, la clause « order by » permet de trier ces élément par ordre alphabétique d'après la valeur de leur élément « titre ». La clause « return » permet enfin de retourner les titres des BD, préalablement triés.
Le « getStringVal » permet de convertir le XMLType pour l'affichage

A la place d'utiliser « getStringVal », il est possible d'utiliser la fonction « XMLCast() ». Dans ce cas, les balises « <titre> » n'apparaissent plus dans le résultat.

```
SELECT
  XMLCast(
    XMLQuery('
    for $bd in //bd
    order by $bd/titre
    return $bd/titre' passing BD_LIST returning content)
    AS CLOB )
FROM BD;
```

Cette fonction permet de modifier le type XMLType par un autre type. La clause « AS XXX » permet de de définir le type souhaité, dans ce cas « CLOB » pour une chaine de caractères large.

2. Les titres des DB sorties après 2000 avec leur date de sortie, sous la forme de données relationnelles. (utiliser la méthode « XMLTable() »)

```
SELECT bd2."Titre", bd2."Sortie"
FROM BD, XMLTable(
'for $bd in //bd
where $bd/sortie >= 2000
return $bd' passing BD_LIST
   columns
      "Titre" varchar2(50) path '//bd/titre',
      "Sortie" varchar2(6) path '//bd/sortie') bd2
```

La fonction « XMLTable() » permet d'exécuter des requêtes XQuery et d'en présenter les résultats sous forme relationnelle.
Ici, notre requête affiche deux champs « Titre » et « Sortie » provenant d'une table « bd2 » créée par la fonction « XMLTable() ». Pour créer cette table, la fonction « XMLTable() » exécute un requête XQuery sur le champ « BD_LIST » (clause « PASSING ») permettant de sélectionner uniquement les BD ayant un sous élément « sortie » dont la valeur est supérieur ou égale à 2000. Ensuite, la clause « COLUMNS » permet de définir les colonnes qui seront créées. La définition d'une colonne se fait en spécifiant tout d'abord son nom ainsi que son type puis, à l'aide de la clause « PATH », le sous-élément du résultat de la requête XQuery à utiliser pour remplir cette colonne peut également être spécifié.

3. Combien de DB compte la série « Le retour a la terre » (utiliser la méthode « XMLQuery() »)

```
SELECT XMLQuery('
count(//serie[titreserie="Le retour a la terre"]/bd)'
passing BD_LIST returning content)
FROM BD
```

Dans le cas de cette requête, nous avons simplement utilisé la fonction « XMLQuery() » afin d'évaluer la requête passé en paramètre sur notre champ XMLType.

1.2.1.2 Requête d'écriture

Pour la deuxième partie de cet exercice, le but était d'ajouter, modifier et supprimer une BD dans la liste actuelle.

1. Ajouter la DB suivante à la série nommée « Le retour à la terre » :

```
<bd id="123" numero="4">
      <titre>Le Deluge</titre>
      <auteur ref="JYFerri"/>
      <illustrateur ref="Larcenet"/>
      <sortie>2005</sortie>
</bd>
```

```
UPDATE BD SET BD_LIST =
   APPENDCHILDXML(BD_LIST,
   '//serie[titreserie = "Le retour a la terre"]',
   XMLType('<bd id="123" numero="4">
   <titre>Le Deluge</titre>
   <auteur ref="JYFerri"/>
   <illustrateur ref="Larcenet"/>
   <sortie>2005</sortie>
   </bd>'))
```

La fonction « AppendChildXML() » de Oracle permet d'ajouter un fragment XML à l'intérieur d'un nœud défini par une expression XPath. Comme cette fonction effectue une modification des données, elle doit être utilisé à l'intérieur d'une requête « UPDATE » standard.
Le premier paramètre de cette fonction permet de définir le champ dans lequel se trouvent les données à modifier. Le deuxième paramètre permet de spécifier l'expression XPath définissant le nœud dans lequel le fragment XML donné en troisième paramètre devra être inséré.

2. Modifier la date de sortie de cette BD en « 2006 »

```
UPDATE BD SET BD_LIST =
UPDATEXML(BD_LIST,
'//bd[titre="Le Deluge"]/sortie/text()',2006)
```

Pour modifier une partie d'un document XML existant, il faut utiliser la fonction « UpdateXML() ». Cette fonction prend comme premier paramètre le champ

contenant les données à modifier. Le deuxième paramètre permet de définir le nœud à modifier (expression XPath) et le troisième paramètre la valeur que devra avoir ce nœud après la modification.

3. Supprimer cette bande dessinée de la base de données

```
UPDATE BD SET BD_LIST =
DELETEXML(BD_LIST,
'//bd[@id="123"]')
```

La suppression d'un élément dans nos données XML peut se faire grâce à la fonction « DeleteXML() ». Cette fonction reçoit comme premier paramètre le champ contenant les données dans lesquelles le nœud devra être supprimé. Le deuxième paramètre est une simple expression XPath permettant de définir l'élément a supprimé.

1.3 Annexe : XQuery dans Oracle avec Java et JDBC

Pour cet exemple, nous avons créé une classe « OracleTest.java » effectuant les requêtes vers la base de données. Voici le détail du contenu de cette classe :

```
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.xdb.*;
import oracle.jdbc.OracleResultSet;
```

Les librairies à importer pour cette classe sont les suivantes :

- java.sql : classe de bases pour les interactions avec le BD
- oracle.jdbc.driver : classes pour le driver Oracle
- oracle.jdbc.OracleResultSet : classe de résultat pour les requêtes Oracle
- oracle.xdb : classes pour l'utilisation du type XMLType de Oracle

```
public static void main(String[] args){
        Connection conn = null;
        //définition de l'URL de connexion
        String urlStr = "jdbc:oracle:thin:@160.98.2.105:9203:dbinf";

        // définition du nom d'utilisateur et du mot de passe
    String userName = "m-1"; //choisir votre numero de compte
    String password = "M-1"; //et le password qui va avec

    try {
        // Enregistrement du driver utilisé pour l'accès à la base.
        DriverManager.registerDriver(new OracleDriver());
        // Initialisation de l'objet "Connection"
        conn = DriverManager.getConnection(urlStr,userName,password);
    }
    catch (SQLException sqle) {
        System.out.println(sqle);
        System.exit(0);
    }

    // Contenu de la requête
    String query = "select b.BD_LIST.EXTRACT('//serie[titreserie=\"Titeuf\"]/bd/titre') from BD b";

    try{
    // Création d'un objet "Statement" qui permettra de
    // représenter la requête.
    Statement stmt = conn.createStatement();

    // Exécution de la requête et récupération du résultat
    // dans un objet "OracleResultSet"
    OracleResultSet ors = (OracleResultSet) stmt.executeQuery(query);

    // Affichage du résultat en utilisant un objet "XMLType"
    while(ors.next()){
        if(ors.getOPAQUE(1) != null)
        {
                XMLType xml = XMLType.createXML(ors.getOPAQUE(1));
                System.out.println(xml.getStringVal());
        }
    }

    // Fermeture de la connexion.
    stmt.close();
    ors.close();
    }
    catch(SQLException sqle){
        System.out.println(sqle);
        System.exit(0);
    }
}
```

Cette classe ne contient qu'une méthode « main » qui s'occupe d'exécuter les requêtes et d'afficher les résultats (voir commentaires pour plus de détail).