Bases de données et XML

Anne-Cécile Caron

Master MIAGE - BDA

1er trimestre 2010-2011

XML

- XML = eXtensible markup language
- Langage de description de documents (semi)-structurés, utilisant des balises

```
<lettre>
  <en-tete>
    <nom-ent>A.C. Caron</nom-ent>
    <adresse>... </adresse>
  </en-tete>
  <date>2 décembre 2006</date>
  <destinataire>...</destinataire>
  <objet>...</objet>
  <salut>Monsieur,</salut>
  <corps>
    <para>...</para>    ...
  </corps>
  <signature>Anne-Cécile Caron</signature>
</lettre>
```


Histoire

- Langages de balises :
  - SGML, description de documents techniques, normalisé en 1986 (début en 1979).
  - HTML, inventé pour le web 1991
- 1996 : création d'un groupe de travail du W3C dont les objectifs sont de définir un langage
  1. plus facile que SGML
  2. plus général que HTML
- 1998 : XML 1.0. Version simplifiée de SGML et plus adaptée au Web (support natif des différents codages internationaux).

Documents bien formés / valides

- *Bien formé* = suit les règles syntaxiques de XML.
  - Bon parenthèsage des balises ouvrantes et fermantes.
  - Un élément racine contient tous les autres (on parle d'arbre d'éléments)
- *Valide* = bien formé + conforme à un schéma, défini par une DTD ou un XML-schema.
- La validité n'est pas requise : il n'est jamais obligatoire de définir un schéma pour un document.

Contenu orienté document / orienté données

- Contenu orienté données :
  - très régulier, contenu structuré.
  - Souvent, fichier généré à partir d'un SGBD (relationnel) par un programme.
- Contenu orienté document :
  - contenu semi-structuré. Exploitation de la "souplesse" d'XML.
  - Fichier écrit manuellement, ou provenant d'une conversion à partir de PDF, RTF, ...

Pourquoi "XML et Bases de données" ?

- Technologies qui rapprochent XML des SGBD :
  - définition de schéma : DTD, XML-schema, Relax-NG
  - Langages de requêtes : XPath, XQuery
  - Interfaces de programmation : SAX, DOM, JDOM, ...
- Stocker du XML si
  - On possède déjà beaucoup de données sous ce format
  - Nécessité de faire de la recherche d'informations parmi ces données
- Quand on veut stocker des documents XML dans un SGBD, on peut
  1. Stocker dans un base de données XML native
  2. Stocker dans un SGBD relationnel sous la forme de CLOB
  3. Stocker dans un SGBD relationnel sous la forme de tables.

BdD XML native ou relationnelle ?

- SGBD XML natif :
  - respect des normes XML du W3C,
  - stockage adapté donc bonnes performances,
  - langage de requête XQuery, XUpdate, ...
  - MAIS peut-être moins performant sur les points propres aux SGBD (transaction, sécurité)
  - Pas de lien avec des données relationnelles
- SGBD relationnel
  - Le langage de requête/modification basé sur SQL est un peu lourd, difficulté de manipuler des modèles différents.
  - Des passerelles entre les données relationnelles et XML.
  - stockage sous forme de CLOB
    - juste la persistance
    - fidélité au texte : le document est conservé tel quel
    - peu performant (interrogation, modification).
  - stockage dans des tables :
    - Utilisation d'un stockage un peu mieux adapté.

SQL/XML

- Depuis 2003 : langage SQL/XML dans la norme SQL3.
  Aussi appelé SQLX
- Permet de fabriquer du XML à partir de données relationnelles
- A l'inverse, permet de poser des requêtes SQL sur des données XML.
- *attention, la norme SQL/XML est différente de la technologie Microsoft SQLXML de SQLServer*

Un type pour les données XML

- Il existe un type pour les données XML
- Ce type peut être utilisé pour typer une colonne d'une table, ou pour typer une ligne (cf modèle objet-relationnel)

```
create table T1 (
  clef NUMBER(3) constraint t1_pkey primary key,
  xml_col XMLType
);

create table T2 of XMLType ;
```

- Une valeur de ce type est une forêt XML.
- Sous Oracle, `XMLType` :
  Constructeur qui fabrique une valeur `XMLType` à partir de `VARCHAR`, `CLOB`, `BLOB`, `BFILE`.

Comment générer du XML

- `xmlelement()` crée un élément XML, dont le nom peut être spécifié
- `xmlattributes()` crée des attributs XML à partir de colonnes (le nom de la colonne est le nom de l'attribut)
- `xmlforest()` crée des éléments XML à partir de colonnes, le nom de chaque élément étant le nom de chaque colonne
- `xmlconcat()` concatène des valeurs XML en 1 seule valeur qui représente une forêt XML
- `xmlagg()` combine une collection de lignes, chacune contenant une valeur XML, pour créer une forêt XML. `xmlagg` est une fonction d'agrégation, comme `sum`. On a donc un résultat par groupe de lignes.
- `xmlparse`, `xmlcomment`, ...

Exemple

On reprend le schéma qui sert à la simulation d'un supermarché.

```
CAISSE(identifiant,ouverte) ;
CLIENT(numero, duree_prevue, idcaisse, rang)

select
xmlelement(name "client",
  xmlforest(numero as numClient, duree_prevue as duree, rang, idCaisse))
from client ;
```

```
<client>
        <NUMCLIENT>1</NUMCLIENT>
        <DUREE>4</DUREE>
        <RANG>4</RANG>
        <IDCAISSE>1</IDCAISSE>
</client>
<client>
        <NUMCLIENT>2</NUMCLIENT>
        <DUREE>8</DUREE>
        <RANG>1</RANG>
        <IDCAISSE>2</IDCAISSE>
</client>
...
```


Exemple

- Dans la requête précédente, chaque ligne de la table `CLIENT` engendre une ligne du résultat.
- On peut fabriquer un résultat avec une seule ligne qui soit un vrai document XML enraciné. Pour cela,
  1. utiliser `xmlagg` pour fabriquer 1 seule ligne contenant une forêt avec toutes les lignes du résultat précédent,
  2. puis mettre cette forêt dans un élément père `lesClients`.

```
select xmlelement(name "lesClients",
  xmlagg(
    xmlelement(name "client",
      xmlforest(numero as numClient, duree_prevue as duree, rang, idCaisse)
    )
  )
)
from client ;
```

Exemple avec jointure

```
select xmlelement(name "caisse",
  xmlattributes(identifiant as id),
  xmlagg( xmlelement("client",numero) )
  )
from caisse
join client on identifiant = idCaisse
group by identifiant ;

<caisse ID="1">
  <client>1</client>
  <client>3</client>
  <client>12</client>
  <client>16</client>
</caisse>
...
```

utiliser `left join` *si on veut les caisses qui n'ont pas de client*

Exemple avec sous-requête

Requête qui extrait toutes les données sous la forme d'un seul document :

```
select xmlelement(name "lesCaisses",
  xmlagg(
    xmlelement(name "caisse",
      xmlattributes(identifiant as id),
      xmlelement(name "ouverte", ouverte),
      xmlelement(name "clients",
        (select xmlagg(
           xmlelement(name "client",
             xmlattributes(numero as id),
             xmlforest(duree_prevue, rang)
           ) order by rang
         )
        from client
        where client.idCaisse = caisse.identifiant
        )
      )
    )
  ) )
from caisse ;
```


Exemple

Résultat de la requête précédente :

```
<lesCaisses>
  <caisse ID="1">
    <ouverte>1</ouverte>
    <clients>
      <client ID="12"><DUREE_PREVUE>9</DUREE_PREVUE><RANG>1</RANG></client>
      <client ID="16"><DUREE_PREVUE>1</DUREE_ PREVUE><RANG>2</RANG></client>
      <client ID="3"><DUREE_PREV UE>3</DUREE_PREVUE><RANG>3</RANG></client>
      <client ID="1"><DUREE_PREVUE>4</DUREE_PREVUE><RANG>4</RANG></client>
    </clients>
  </caisse>
...
  <caisse ID="10">
    <ouverte>0</ouverte>
    <clients></clients>
  </caisse>
</lesCaisses>
```


Manipulation de données XML

SQL/XML définit des opérateurs applicables sur des données de type XML.

- `existsNode()` qui teste si un noeud particulier existe, noeud désigné par requête XPath
- `schemaValidate()` qui valide selon un XML-schema
- `transform()` qui applique une transformation XSLT
- `extract()` prend une expression XPath *e* et retourne les noeuds résultat de *e*
- `extractValue()` prend une expression XPath *e* et retourne les valeurs (feuilles) des noeuds résultats de *e*
- `updateXML()` Permet de modifier une données XML en utilisant des expression XPath
- ...

Préparation de l'exemple

On suppose que T2 contient des lignes de documents XML, une ligne représentant une caisse du supermarché. Pour remplir une table contenant du XML on peut :

- Remplir à partir d'un fichier XML externe (donnée BFILE)

```
insert into T2 values(xmltype(  bfilename(...),  nls_charset_id(...)  )) ;
```

- Remplir à partir d'une chaîne de caractères

```
insert into T2 values(
'<caisse ID="1"><client>3</client><client>16</client>...</caisse>');
```

- Remplir à partir d'une requête

```
insert into T2
select xmlelement(name "caisse",
  xmlattributes(identifiant as id),
  xmlagg( xmlelement("client",numero) )
  )
from caisse
join client on identifiant = idCaisse
group by identifiant ;
```


Contenu de la table exemple

```
select * from T2;

<caisse ID="1">  <client>1</client>  ...  <client>16</client> </caisse>
<caisse ID="2"><client>2</client> <client>5</client> </caisse>
<caisse ID="3"> <client>7</client> ... </caisse>
<caisse ID="4"> <client>4</client>... </caisse>
<caisse ID="5"> <client>17</client> ... </caisse>
<caisse ID="6"/>
<caisse ID="10"/>
<caisse ID="9"/>
<caisse ID="7"/>
<caisse ID="8"/>
```


Requête XPath et extract

```
select extract(t.object_value, '/caisse/client')
from T2 t ;

<client>3</client><client>16</client><client>12</client><client>1</client>
<client>2</client><client>5</client>
<client>7</client><client>9</client><client>13</client>
<client>11</client><client>18</client><client>14</client><client>4</client>
<client>17</client><client>10</client><client>6</client><client>15</client>



10 ligne(s) sélectionnée(s).
-- dont 5 contiennent NULL
```


update

On peut remplacer un sous-arbre par un autre, du texte par un autre, ...

```
update T2
set object_value = updateXML(
  object_value,
  '/caisse[@ID="2"]',
  xmltype(
   '<caisse ID="2"><caissiere>Annie</caissiere><client>2</client>...</caisse>'
  )
);
  -- on fait pareil pour la caisse 5

<caisse ID="1">  <client>3</client>  ... <client>1</client> </caisse>
<caisse ID="10"> <client/> </caisse>
<caisse ID="2"> <caissiere>Annie</caissiere> <client>2</client> ... </caisse>
<caisse ID="3"> <client>7</client>...</caisse>
<caisse ID="4"> ... <client>4</client> </caisse>
<caisse ID="5"> <caissiere>Eve</caissiere> <client>15</client>... </caisse>
<caisse ID="6"> <client/> </caisse>
<caisse ID="7"> <client/> </caisse>
<caisse ID="8"> <client/> </caisse>
<caisse ID="9"> <client/> </caisse>
```

existsNode

```
select extract(t.object_value, '/caisse/client')
from T2 t
where existsNode(t.object_value,'/caisse/caissiere')=1;

<client>2</client><client>5</client>
<client>17</client><client>10</client><client>6</client><client>15</client>
```


Vues

- On peut définir une vue relationnelle sur des données XML

```
create or replace view ma_vue(idcaisse,numclient)
as
select
  extract(object_value,'/caisse/@ID'),
  extractValue(value(c),'.')
from T2 t,
    table(xmlsequence(extract(value(t),'/caisse/client'))) c
```

- A l'inverse, on peut définir une vue XML sur des données relationnelles.

```
create or replace view caisses_xml of xmltype
with object id (substr(extractValue(object_value,
                                    '/lesCaisses/caisse/@id'),
                       1,128))
as
select xmlelement(
...
)
from caisse ;
```

- Les requêtes sur les vues sont réécrites dans le modèle sous-jacent. Efficacité ? ?

JDBC

- JDBC 4.0, (java 6) : interface `java.sql.SQLXML`.
- méthodes `getSQLXML`, `updateSQLXML` de `ResultSet`, méthode `setSQLXML` de `PreparedStatement`, idem pour `CallableStatement`.
- On peut profiter de toutes les fonctionnalités des sous-paquetages de `javax.xml` : validation (SAX, StAX, DOM), requête XPath, transformation XSLT, ...

Conclusion

- SQL/XML permet la manipulation de données XML et la publication vers ce format.
- Intégration de données XML et relationnelles dans le même SGBD.
- Vues pour passer d'un modèle à l'autre.
- Exploitation de SQL/XML avec JDBC.
- Langage de modification peu pratique, langage de sélection moins lisible que XQuery.
- Traitement des requêtes très variable selon le mode de stockage (CLOB ou non)