

DB2 7.1 : SQL/XML

Le lot de nouveautés 7.1 relatif à DB2 est une fois de plus très riche avec notamment (liste non exhaustive) :

- L'introduction de variables globales
- Les nouveaux tableaux SQL pouvant être définis dans un PSM (Persistent Stored Module) à savoir les procédures, fonctions et triggers
- De nouvelles fonctions scalaires pour communiquer avec MQ Series ainsi que les fonctions BITAND, BITANDNOT, BITOR, BITXOR et BITNOT (identiques aux fonctions RPG)
- De nouvelles fonctionnalités concernant le comportement du Commitment Control (WAIT FOR OUTCOME, USE CURRENTLY COMMITTED) en prolongement des nouveautés V6R1 (SKIP LOCKED DATA)
- La possibilité de lire un RESULT SET depuis un Embedded SQL, c'est-à-dire depuis un langage hôte tel RPG, COBOL ou autre
- La possibilité d'utiliser des expressions en tant que paramètre dans un CALL de procédure
- L'ordre CREATE OR REPLACE comme sur d'autres SGBD
- La possibilité d'ajouter une colonne dans une table, ailleurs qu'en bout de table
- Les « Fields Procédures », c'est-à-dire la possibilité de déclencher un trigger externe (un *PGM en l'occurrence) sur une zone, pouvant être très utile pour du cryptage
- Des améliorations pour la création des EVI (Encoded Vector Index) pouvant inclure des agrégats afin d'améliorer sensiblement les performances
- Le nouvel ordre MERGE (issu de la norme ISO SQL :2003) combinant un ordre INSERT et un UPDATE
- Un nouveau type de donnée XML
- De nouvelles fonctions de manipulation des données XML.

C'est autour de ces deux dernières nouveautés que s'articule cet article.

Le XML a été développé il y a plus de 12 ans et s'est imposé comme une norme ouverte pour décrire et échanger des données entre systèmes d'information. Ce succès foudroyant s'explique en partie par l'explosion de l'interconnexion de machines hétérogènes sur Internet. Il s'impose donc aujourd'hui comme le format universel d'échange entre applications, libre de droits et indépendant des plates-formes.

XML est largement utilisé pour

- les échanges de données informatisées (EDI),
- Comme format universel entre applications (les suites bureautiques comme Office, StarOffice.. , les browsers...)
- c'est aussi la base de l'interopérabilité et des services Web,
- également comme principe de structuration de nouvelles normes

Mais si vous êtes un développeur Java sur UNIX ou Windows, vous savez probablement déjà tout sur le XML. Cependant, l'utilisation de XML est relativement nouvelle sur les IBM i malgré que le support de XML soit partie intégrante de la norme ANSI SQL:2003.

Petit Historique de XML sur DB2

En Juillet 2006, IBM sort sa version DB2 « Pure XML » sous le sobriquet DB2 « Viper » pour désigner DB2 9 sur les plates-formes LUW (Linux/Unix/ Windows). Dans la foulée, en 2007, les systèmes zOS bénéficient de cette nouveauté. L'IBM i attendra 2010 et la 7.1 avant de posséder cette fonction native auparavant prise en charge (partiellement) par le logiciel sous licence facturable DB2 XML Extender.

PureXML permet aux documents XML d'être stockés sans perdre les informations hiérarchiques qui leurs sont inhérentes. Ainsi DB2 devient non plus juste une base de données relationnelle qui peut stocker des informations dans le format traditionnel, mais peut aussi stocker des informations dans un format XML natif.

PureXML™ est le nom qu'utilise IBM pour parler de la possibilité de stocker du XML nativement.

Les apports de la version i 7.1

Si vous aviez investi sur le produit XML Extender tous ces concepts ne vous sont pas étrangers et XML Extender est toujours disponible et fonctionne en 7.1. Toutefois, l'intention d'IBM est de remplacer ce produit par les fonctions natives XML de DB2.

Dorénavant DB2 i 7.1 permet nativement :

- 1) le support de XML en tant que nouveau type de colonne
- 2) la possibilité de valider les données avec un schéma XML
- 3) l'aptitude à lire des données XML
- 4) la possibilité de décomposer un document XML en données relationnelles
- 5) la possibilité de générer un flux XML à partir de données relationnelles
- 6) La possibilité de transformer des documents XML avec XSLT.

1) Nouveau type de colonne XML

De nos jours les développeurs doivent faire cohabiter des données relationnelles avec des données XML dans leurs applications. Mais ces derniers choisissent le plus de souvent de stocker leurs données XML directement dans des colonnes de type VARCHAR (32 Ko) ou CLOB (2 Go) voire dans des documents sous forme de texte stockés dans l'IFS (Integrated File System) et de le traiter avec des opérations classiques comme sur n'importe quelle chaîne.

L'utilisateur peut créer une table composée d'une ou plusieurs colonnes de type XML en plus des colonnes relationnelles :

```
CREATE TABLE ARTICLES ( ARTID          ROWID NOT NULL,  
                        INFOS          XML  
                        )
```

Figure 1

Désormais, il est possible de stocker des documents XML nativement. Son avantage sur un champ de type « texte » est qu'il vérifie que le document soit « bien formé » c'est-à-dire n'ayant pas d'erreurs de syntaxe et respectant toutes les règles qui régissent l'aspect d'un fichier XML. Ainsi, un crochet manquant fera qu'un fichier XML n'est pas bien formé et ne pourra être inséré dans la base de données sous peine d'une erreur SQ20398.

On entend par « document bien formé » le respect des contraintes ci-dessous :

- ✓ Le document doit posséder une racine
- ✓ Chaque balise ouvrante possède une balise fermante, la casse devant être respectée
- ✓ Chaque nœud XML est correctement emboîté.
- ✓ Chaque attribut XML est encadré de guillemets

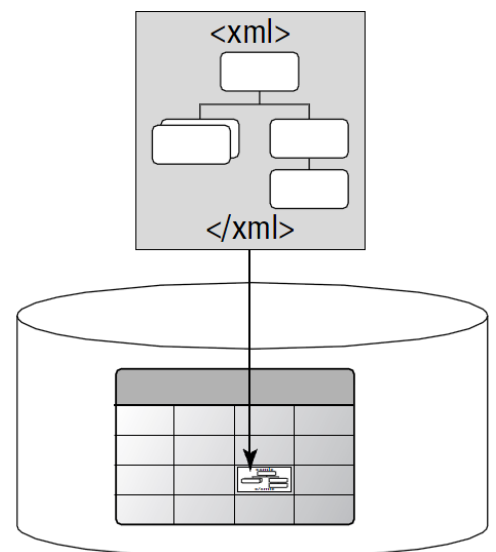
Il faut ajouter en tant qu'avantage d'une colonne XML par rapport à un document stocké dans l'IFS :

- une sauvegarde/restauration simplifiée
- une sécurité identique à la table elle même

A la différence des champs de type « texte », on ne définit pas de longueur pour un champ de type XML qui est limité à 2 Go comme n'importe quel autre LOB (Large Object), voir figure 1.

```
INSERT INTO ARTICLES (INFOS) VALUES ('  
<Informations>  
  <Libelle>Nokia N40 </Libelle>  
  <Prix>32,71</Prix>  
</Informations>')
```

Figure 2



La conversion d'une constante (considérée comme un VARCHAR) en format XML se fait automatiquement (voir Figure 2) mais il est aussi possible d'utiliser la fonction scalaire XMLPARSE pour réaliser ce « casting ». Cette dernière permet en option, d'ajouter le paramètre « STRIP WHITESPACE » (valeur par défaut) ou « PRESERVE WHITESPACE » qui permet de définir le comportement à adopter vis-à-vis des espaces blancs dans une valeur de chaîne de caractères. Le flux XML sera stocké en UTF-8 (1208) valeur par défaut indiquée dans le registre SQL_XML_DATA_CCSID de QAQQINI.

Bien évidemment on peut importer du contenu dans une colonne de type XML provenant d'un fichier de l'IFS (ou d'un fichier source) grâce à la fonction scalaire GET_XML_FILE ou GET_BLOB_FROM_FILE comme montré en figure 3

```
INSERT INTO ARTICLES (INFOS) VALUES (  
  GET_XML_FILE('/home/doc.xml')  )
```

Figure 3

Pourquoi une BD avec du XML ?

Si vos données sont très structurées, le modèle relationnel est sans doute le mieux adapté au stockage des données. En revanche, s'il s'agit de données semi-structurées voire non structurées, ou si la structure est inconnue voire variable dans le temps, mieux vaut envisager l'utilisation de XML.

Bien évidemment comme toutes les nouveautés depuis de nombreuses années, ce nouveau type de champ n'est pas supporté dans les DDS.

Limites des types de données XML

- les types de données XML ne peuvent pas être utilisés dans une instruction GROUP BY ou ORDER BY
- Ils ne sont pas permis avec les prédicats BETWEEN, DISTINCT, IN, LIKE
- ils ne peuvent faire l'objet de contraintes de clé (primaire, unique ou étrangère) ou de vérification.
- On ne peut pas créer un index sur une colonne XML, il faudra utiliser les « Index Text » du produit Omnifind Text Search for IBM i

2) Validation des données avec un schéma XML

Il est aussi possible de vérifier que le document soit « valide » à partir d'un XML Schéma (XSD). On comprendra qu'un XSD permet de délocaliser une partie de l'intelligence de l'application : quoi de plus simple que de valider des règles métier avec un Schéma XML !

La validation à partir d'un DTD ne sera pas possible, mais il existe de nombreux outils gratuits qui permettent la conversion d'un DTD en XSD (ex TRANG).

Par exemple, si un XML Schéma dit qu'une valeur de tag devrait être un nombre et si le fichier XML possède une lettre, le fichier XML, même bien formé, sera non valide. Il est donc très facile de contrôler le bon état d'un document avant son intégration dans DB2, mais ce n'est pas une obligation.

Si vous voulez créer un fichier XSD à partir d'un document XML existant, il existe de nombreux utilitaires pour ce faire. Il existe notamment TRANG, un programme Java qui peut être installé sur votre IBM i :

Télécharger TRANG.JAR sur <http://code.google.com/p/jing-trang/downloads/list>. Il suffit de FTP l'utilitaire dans un répertoire de l'IFS puis sous QSH :

cd /monrepertoire; java -jar trang.jar source.xml cible.xsd



ASTUCE

Avant qu'un schéma XML puisse être utilisé pour valider un document XML, ce dernier doit être enregistré dans le « XML Schema Repository » appelé XSR. Cette étape préalable sera accomplie grâce à de nouvelles procédures stockées intégrés à DB2 : XSR_REGISTER et XSR_COMPLETE comme montré en figure 4.

```
CALL SYSPROC.XSR_REGISTER ('MABIB', 'MONXSR', NULL,  
    GET_XML_FILE('/xml/articles.xsd'), NULL);  
  
CALL SYSPROC.XSR_COMPLETE('MABIB', 'MONXSR', null, 0);
```

*Le XSD sera stocké dans le nouvel objet MABIB/MONXSR de type *SQLXSR*

Figure 4

Un appel à la procédure XSR_COMPLETE est nécessaire afin d'achever l'enregistrement du XSD, car il est possible de combiner de multiples schémas XML (les fusionner) en un unique

XSR en utilisant la procédure XSR_ADDSCHEMADOC. Ceci offre la possibilité de valider un flux XML à partir de plusieurs schémas en une seule passe.

Mais dans quel cas ? Tout simplement si votre flux XML évolue avec l'ajout de nouvelles informations, vous pourrez valider ces nouvelles informations avec un nouveau schéma partiel relatif à ces nouvelles données sans tout recommencer.

A l'inverse, la procédure XSR_REMOVE permet la suppression d'un objet *SQLXSR.

Le contenu du référentiel de schémas XML (XSR) peut être interrogé en lançant une requête SQL sur QSYS2.XSROBJECTS faisant dorénavant partie des références croisées de DB2 ou directement via iSeries Navigator.

Pour insérer dans une colonne XML un document XML en demandant sa validation, il faudra utiliser l'ordre XMLVALIDATE comme montré en figure5.

```
INSERT INTO Articles(Info)
VALUES (XMLVALIDATE (XMLPARSE (DOCUMENT
GET_XML_FILE('/dir1/art6.xml'))
ACCORDING TO XMLSCHEMA ID mabib.monxsr) )
```

La validité du document XML en regard du XSD stocké dans le XSR sera contrôlé.

Figure 5

Pourquoi ne pas pouvoir valider directement un XML dans une colonne à partir d'un XSD stocké dans l'IFS sans passer par un XSR ? Simplement pour des questions de performances qui seraient impactées si DB2 devait à chaque fois accéder à un fichier externe stocké dans l'IFS.

Si vous voulez vous créer un jeu d'essai sur votre système, sachez qu'il existe une nouvelle procédure stockée CREATE_XML_SAMPLE à l'identique de la fameuse procédure CREATE_SQL_SAMPLE.

*Un CALL de cette procédure stockée en lui passant en paramètre un nom d'une bibliothèque à créer et vous obtiendrez un jeu d'essai complet de tables contenant du XML. C'est sur cette base que nous continuerons cet article : **CALL CREATE_XML_SAMPLE('XMLSAMPLE')***

ASTUCE

3) Lecture des données XML

Comme vous l'aviez compris, un document XML contient des données semi-structurées. C'est une base de données au sens conceptuel du terme mais avec une structure pouvant être très versatile. Les données représentent une hiérarchie d'imbrications, et non des références entre entités, et peuvent être récursives. Qu'un document XML soit stocké dans une colonne XML ou de type texte à travers des VARCHAR ou des CLOB, il faut pouvoir en extraire sélectivement le contenu. Généralement, une application n'utilise qu'un sous-ensemble de données réparti dans tout le document XML; elle agrège et traite une partie de ces informations. Il s'agit alors d'interroger un ou plusieurs documents XML pour extraire les parties significatives à l'application. Pour interpréter ce flux, les langages C/C++ et Java possèdent leur propre « parser » (SAX & DOM, JDOM) à travers les logiciels sous licence XML Toolkit et le JDK Toolkit, ou encore les projets open source Xerces & ANT d'« Apache Foundation ».

Avec l'arrivée de la V5R4, RPG bénéficie de « Parsers » natifs avec les nouveaux codes opération XML-INTO et XML-SAX. Le COBOL quant à lui possède cette possibilité de « parser » depuis la V5R3. Mais il s'agit là de mécanismes de « bas niveau » et l'approche est plutôt programmatique, or la plupart des interrogations de données se font par un langage spécifique comme SQL qui lui, par contre, est conçu pour l'interrogation de données structurées.

Comment donc interroger un flux XML avec efficacité et rapidité ?

Il existe plusieurs méthodes (hors parseurs traditionnels cités plus haut) :

- Le « Shredding » qui consiste à décomposer un document XML en une base de données relationnelle et ainsi permettre SQL d'en extraire le contenu (nous aborderons ce sujet dans le chapitre suivant) avec de simples requêtes;
- L'utilisation des extensions SQL tel xQuery de la norme SQL/XML ISO SQL :2003 soutenue par le W3C. Malheureusement cette extension n'a pas encore été implémentée dans DB2 en 7.1 ! mais une évolution logique des choses devrait nous permettre d'espérer une future livraison de cette extension dans les toutes prochaines versions ;
- La recherche grâce à d'autres outils standards, tel XSLT (eXtensible Stylesheet Language Transformation) proche de xQuery, qui sera lui aussi abordé dans le dernier chapitre de cet article;
- La recherche « plain text » qui est différente d'une recherche d'occurrence de chaîne de caractère tel un LIKE. L'opération de base d'une recherche « plain text » est le découpage d'une chaîne afin d'en détecter les différents mots et d'indexer ces derniers. Cette recherche et indexation est possible avec l'extension gratuite « Omnifind Text Search for IBM i » livrée à partir de la V6R1 et de base en 7.1. Il s'agit d'une extension relative à la norme SQL :2003 « Full Text ». Elle permet l'utilisation de nouvelles expressions SQL

comme CONTAINS ou encore SCORE pour retrouver des données contenant certains mots ou formes fléchies de mots (grâce à de nombreux dictionnaires intégrés en fonction des langues utilisées), et éventuellement des synonymes dans les lignes des tables, y compris pour des colonnes de type LOB et XML. Mais « **Omnifind Text Search for IBM i** » ne s'arrête pas là, il **supporte la syntaxe XPath** afin de pouvoir faire des recherches sur un élément donné et/ou un attribut :

Exemple :

```
SELECT storeid, storezip FROM books
WHERE CONTAINS(storebooks, '@xpath:'
'/bookstore/book[@price<15.00 or @genre="autobiography"]'' ')=1
```

Concernant l'approche programmatique, le nouveau type de donnée XML pourra être lu dans un programme HLL comme ses homologues LOBs à travers un LOCATOR. Il est aussi possible de régénérer un fichier dans l'IFS à partir d'une colonne XML comme montré en Figure 6.

```
D MON_XML      S          SQLTYPE (XML_CLOB_FILE)
/free
MON_XML_NAME= '/out1.xml';
MON_XML_NL = 9;
MON_XML_FO = SQFCRT;
exec sql
      SELECT Infos INTO :MON_XML FROM Articles
      WHERE ARTID=1;
/end-free.
```

Figure 6

Un LOCATOR étant un pointeur, un ordre SELECT sur une colonne XML en 5250 renverra le résultat : *POINTER. Pour visualiser le contenu d'un champ XML il faudra utiliser d'autres interfaces telles ODBC/OLE DB ou encore JDBC, c'est le cas de iSeries Navigator (Figure 7). Sinon, il faudra utiliser la nouvelle fonction XMLSERIALIZE permettant de faire un casting d'un champ XML vers un CHAR/VARCHAR/LOB comme montré en Figure 8 (Le CCSID du Job ne doit pas être 65535).

Select * from XMLSample.Customer

CID	INFO	HISTORY
1000	<customerinfo Cid="1000"><name>Kathy Smith</name><addr cou...	-
1001	<customerinfo Cid="1001"><name>Kathy Smith</name><addr cou...	-
1002	<customerinfo Cid="1002"><name>Jim Noodle</name><addr count...	-
1003	<customerinfo Cid="1003"><name>Robert Shoemaker</name><ad...	-
1004	<customerinfo Cid="1004"><name>Matt Foreman</name><addr co...	-
1005	<customerinfo Cid="1005"><name>Larry Menard</name><addr cou...	-

Figure 7

```

.....+..... ! +.....4.....+.....5.....+.....6.....+.....7.....+.....8.....+.....9.....+...10
INFO          ! XMLSERIALIZE
*POINTER     ! <customerinfo Cid="1000"><name>Kathy Smith</name><addr country="Ca
*POINTER     ! <customerinfo Cid="1001"><name>Kathy Smith</name><addr country="Ca
*POINTER     ! <customerinfo Cid="1002"><name>Jim Noodle</name><addr country="Can
*POINTER     ! <customerinfo Cid="1003"><name>Robert Shoemaker</name><addr countr
*POINTER     ! <customerinfo Cid="1004"><name>Matt Foreman</name><addr country="C
*POINTER     ! <customerinfo Cid="1005"><name>Larry Menard</name><addr country="C
*****      Fin de données      *****

```

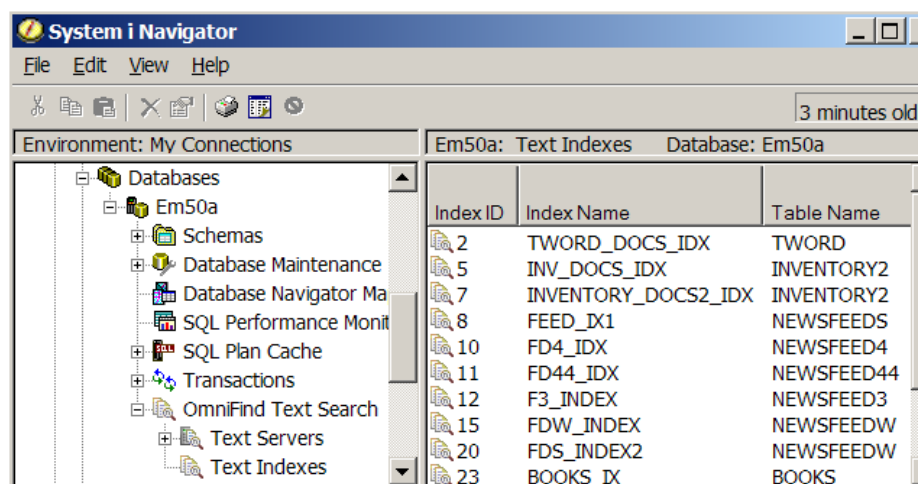
```

SELECT Info, XMLSERIALIZE(Info as VARCHAR(1000) )
FROM XMLSample/Customer

```

Figure 8

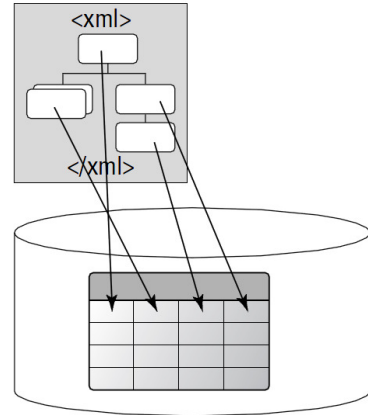
Il est important de rappeler que sans la présence de xQuery (en 7.1), il faudra utiliser « Omnifind Text Search for IBM i » afin de pouvoir faire des recherches pertinentes et rapides dans un flux XML en utilisant ses extensions XPath. C'est aussi le seul moyen d'indexer un document XML. Omnifind possède bien d'autres avantages, il permet notamment de faire des recherches à l'intérieurs de documents PDF, XLS, DOC etc.. stockés dans une colonne BLOB. De plus, la gestion des index Omnifind est dorénavant intégrée à iSeries Navigator.



4) Décomposition d'un document XML en données relationnelles

Vous aviez compris que quand on veut stocker des documents XML dans DB2, on peut les stocker :

- dans une arborescence sous forme de fichier (dans l'IFS) en les référençant dans DB2 avec des DataLink
- dans une base de données relationnelle sous la forme de LOB
- dans une base de données XML native
- dans une base de données relationnelle sous la forme de tables : Cette technique est appelée « le Schredding ».



C'est de ce dernier point que nous allons traiter, c'est-à-dire la possibilité de décomposer un document XML en colonnes dans une ou plusieurs tables de façon à préserver la fidélité des données au niveau relationnel (bien que l'ordre des éléments soit ignoré).

Dans quel but ? Par exemple, le désir d'accéder aux données avec des ordres de lecture natifs (STELL/READ, CHAIN etc..) permettant ainsi à vos applications existantes d'accéder à des données XML.

Schredding Intéressant si

L'arrivée de nouvelles données XML alimente juste une base de données relationnelle existante.

Votre objectif principal est de permettre aux applications existantes d'avoir accès aux données XML.

Vous êtes à l'aise avec votre base relationnelle et voudriez l'utiliser autant que possible.

Schredding NON Intéressant si

Vos données XML sont complexes et imbriquées, difficile des le décomposer dans un schéma relationnel.

Votre Schéma XML est fortement variable ou a tendance à changer rapidement.

Votre objectif principal est de gérer des documents XML comme des objets à part entière.

Le concept de « Shredding » est illustré en Figure 9. Dans cet exemple, le document XML possède un nom de client, son adresse et des numéros de téléphone. Etant donné la multiplicité des numéros de téléphone, il faudra décomposer ce flux dans deux tables distinctes de type Entête/Détail, avec une table pour les clients et une table pour les numéros de téléphone.

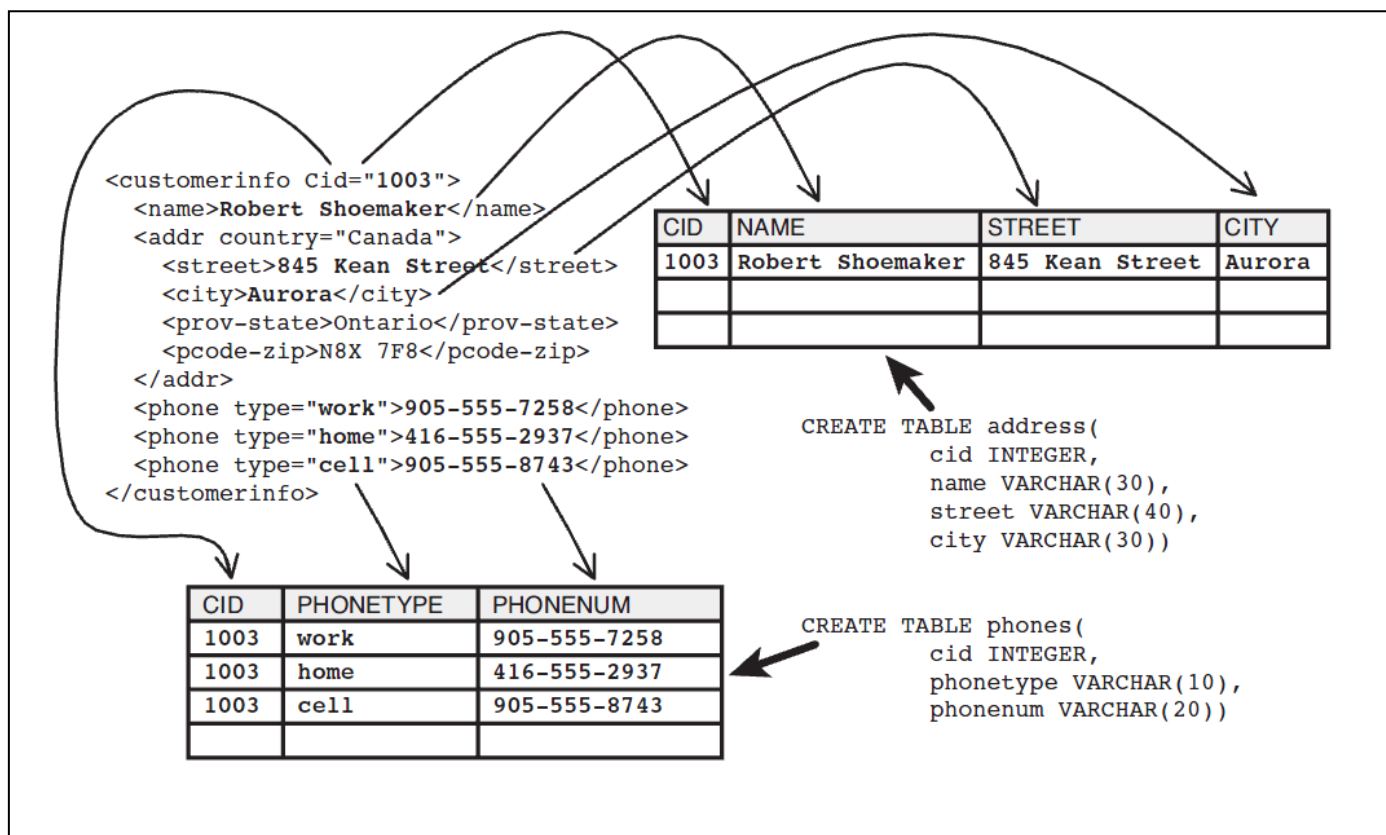


Figure 9

On comprend tout de suite la complexité et la limite du « Shredding » dans ce genre d'exercice surtout si le client possède, tels les numéros de téléphone, de multiples e-mails, plusieurs commandes, plusieurs articles par commande, chaque article possédant plusieurs libellés etc... Le nombre de tables cibles dans la base de donnée relationnelle pourrait croître exponentiellement et malheureusement ce genre de structure, telle que je viens de vous citer, n'est pas rare en XML.

Pour réaliser ce « mapping », le XML Schéma, qui sera encapsulé dans un XSR, doit être annoté d'informations supplémentaires. Puis **il faudra appeler la procédure XDBDECOMPXML** afin de procéder à la décomposition du document XML dans les différentes tables cibles.

Prenons l'exemple d'une ligne de notre XSD :

```
<xs:element name="street" type="xs:string" minOccurs="1"/>
```

On peut lire ici que l'élément « street » est type type « xs:string » et cet élément doit être renseigné au moins une fois. On peut ajouter une annotation à la définition de cet élément afin qu'il soit mappé dans la colonne STREET de la table ADDRESS. Cette annotation consiste à rajouter deux attributs supplémentaires comme dans l'exemple ci-dessous :

```
<xs:element name="street" type="xs:string" minOccurs="1"
  db2-xdb:rowSet="ADDRESS" db2-xdb:column="STREET"/> />
```

```

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  xmlns:db2-xdb="http://www.ibm.com/xmlns/prod/db2/xdb1" >
  <xs:annotation>
    <xs:appinfo>
      <db2-xdb:defaultSQLSchema>db2admin</db2-xdb:defaultSQLSchema>
    </xs:appinfo>
  </xs:annotation>
  <xs:element name="customerinfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string" minOccurs="1"
          db2-xdb:rowSet="ADDRESS" db2-xdb:column="NAME"/>
        <xs:element name="addr" minOccurs="1"
          maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="street" type="xs:string"
                minOccurs="1" db2-xdb:rowSet="ADDRESS"
                db2-xdb:column="STREET"/>
              <xs:element name="city" type="xs:string"
                minOccurs="1" db2-xdb:rowSet="ADDRESS"
                db2-xdb:column="CITY"/>
              <xs:element name="prov-state" type="xs:string"
                minOccurs="1" />
              <xs:element name="pcode-zip" type="xs:string"
                minOccurs="1" />
            </xs:sequence>
            <xs:attribute name="country" type="xs:string" />
          </xs:complexType>
        </xs:element>
        <xs:element name="phone" minOccurs="0"
          maxOccurs="unbounded" db2-xdb:rowSet="PHONES"
          db2-xdb:column="PHONENUM">
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="type" form="unqualified"
                  type="xs:string" db2-xdb:rowSet="PHONES"
                  db2-xdb:column="PHONETYPE"/>
              </xs:extension>
            </xs:simpleContent>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="Cid" type="xs:integer">
        <xs:annotation>
          <xs:appinfo>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>ADDRESS</db2-xdb:rowSet>
              <db2-xdb:column>CID</db2-xdb:column>
            </db2-xdb:rowSetMapping>
            <db2-xdb:rowSetMapping>
              <db2-xdb:rowSet>PHONES</db2-xdb:rowSet>
              <db2-xdb:column>CID</db2-xdb:column>
            </db2-xdb:rowSetMapping>
          </xs:appinfo>
        </xs:annotation>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>

```

Figure 10

Un exemple plus complet de mapping entre notre document XML et nos deux tables en Figure 10.

Si vous décidez de faire du « Shredding » entre un document XML et une ou plusieurs tables, gardez bien à l'esprit que les modèles de données entre ces deux entités sont fondamentalement différentes, modèle hiérarchique pour le premier et relationnel pour le second. Tenter de faire le lien entre ces deux modèles relève d'une gymnastique complexe voire parfois impossible.

5) Générer un flux XML à partir de données relationnelles

Dans le chapitre précédent, on pouvait extraire des données d'un flux XML pour les stocker dans la base de données en modèle relationnel. Mais l'inverse est-il possible ?

Dans les versions antérieures à la 7.1, pour générer du XML à partir des données stockées dans les tables, l'on pouvait s'appuyer sur :

- Des outils externes (SQL via ODBC ou JDBC...) gratuits ou payants.
- Les ordres natifs COBOL ou Fichiers plats copiés dans l'IFS
- Les toolkits IBM (CGIDEV2, XML toolkits)
- Le middleware DB2 XML Extenders (logiciel sous licence payant)
- Divers frameworks : (projets HTTPAPI de Scott Klement, RPG-XML Suite, ...)

Dorénavant la 7.1 nous offre une collection de fonctions et d'expressions disponibles nous permettant de produire du contenu XML appelées « Publishing ».

Liste des fonctions de publication dans le tableau ci-dessous :

FONCTION	DESCRIPTIF
XMLATTRIBUTES	Cette expression produit un attribut XML, à utiliser à l'intérieur de l'expression XMLEMENT.
XMLCOMMENT	Cette fonction crée une valeur XML contenant un commentaire.
XMLCONCAT	Cette fonction concatène une liste de valeurs XML individuelles pour créer une valeur simple contenant un fragment de contenu XML
XMLDOCUMENT	Casting d'un CHAR/VARCHAR/LOB vers un type XML bien formé. Une colonne de type XML stockée dans une table DB2 doit être un DOCUMENT.
XMLEMENT	Cette expression produit un élément XML
XMLFOREST	Cette expression produit un arbre XML (autrement dit une séquence) d'éléments utilisant les noms et le contenu donnés.
XMLPI	Cette expression crée une instruction de traitement XML
XMLNAMESPACES	Cette expression produit un espace de nommage, elle ne peut être utilisée seulement en tant qu'argument de XMLEMENT ou XMLFOREST.
XMLROW	Cette expression produit une ligne XML à partir des colonnes d'une table
XMLTEXT	Cette fonction permet de produire un texte compatible XML
XMLAGG	Cette fonction est, à la différence des fonctions décrites ici, une fonction d'agrégat. Elle concatène les valeurs en entrée pour les passer en argument à la fonction d'agrégat.
XMLGROUP	Cette expression un flux XML par groupes de données

Ces fonctions sont détaillées dans le Redbook « DB2 for i SQL Reference 7.1 » dont ci-dessous un aperçu partiel de leurs possibilités avec un exemple sur deux tables, EMPLOYEE et DEPARTMENT que vous pouvez créer avec la procédure stockée :

CALL CREATE_SQL_SAMPLE('SQLSAMPLE')

```
SELECT xmlelement(name "Personnel",
  xmlagg(
    xmlelement(name "Service",
      xmlattributes(Deptno as Code),
      xmlelement(name "Libelle", trim(Deptname)),
      xmlelement(name "Employes",
        (select xmlagg(
          xmlelement(name "Matricule",
            xmlattributes(empno as ID),
            xmlforest(sex, job, salary)
          ) order by salary
        )
      )
    )
  )
  from SQLSample.Employee
  Where Workdept = Deptno
) ) ) ) )
from SQLSample.Department
Where Deptno = 'A00'
```

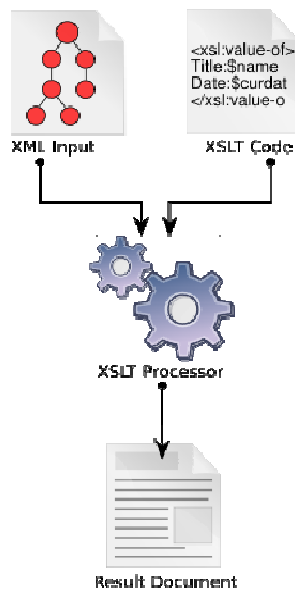
Figure 11

```
<Personnel>
  <Service CODE="A00">
    <Libelle>SPYFFY COMPUTER SERVICE DIV.</Libelle>
    <Employes>
      <Matricule ID="200010">
        <SEX>F</SEX>
        <JOB>SALESREP</JOB>
        <SALARY>1898.77</SALARY>
      </Matricule>
      <Matricule ID="200120">
        <SEX>M</SEX>
        <JOB>CLERK </JOB>
        <SALARY>15199.43</SALARY>
      </Matricule>
      <Matricule ID="000120">
        <SEX>M</SEX>
        <JOB>CLERK </JOB>
        <SALARY>23900.25</SALARY>
      </Matricule>
      <Matricule ID="000110">
        <SEX>M</SEX>
        <JOB>SALESREP</JOB>
        <SALARY>24929.52</SALARY>
      </Matricule>
    </Employes>
  </Service>
</Personnel>
```

6) Transformation de documents XML avec XSLT

La version 7.1 intègre dorénavant la nouvelle fonction **XSLTRANSFORM**, un processeur XSLT (eXtensible Stylesheet Language Transformations) prenant en entrée un document XSLT (décrivant la transformation à effectuer) et un document XML à transformer, afin de produire en sortie un nouveau flux.

Il permet ainsi de séparer la gestion du contenu de la présentation notamment pour produite des documents :



- HTML : pour la présentation Web standard;
- WML pour la présentation WAP ;
- SMIL pour une présentation multimédia ;
- XSL-FO pour la production de documents Papier (PDF) ;
- un autre document XML;
- tout autre format permettant des échanges entre applications (EDI, SOAP, etc..)

La fonction XSLTRANSFORM (conforme aux recommandations du W3C XSLT version 1.10) peut notamment accepter des paramètres XSLT lors de l'exécution et supporte le langage XPath. Cette fonctionnalité était auparavant possible en faisant appel à la classe Java « Transform ».

Conclusion

A partir de la version IBM i 7.1, DB2 est devenue une base de données hybride pouvant héberger conjointement des données relationnelles et des données XML natives, elle offre dorénavant la possibilité de décomposer un document XML en une base de données relationnelle, d'utiliser XSLT avec simplicité et enfin permet d'extraire les informations actuellement stockées dans une structure relationnelle sous la forme de document XML grâce aux nouvelles fonctions de « publishing ».

Des possibilités natives XML de DB2, cet article n'en dévoile qu'une partie de l'iceberg, sans parler de la puissance « d'Omnifind Text Search for IBM i » couplé à XPath. A quand xQuery ? Toutes ces nouveautés rendent désormais, le middleware « DB2 XML Extender » ainsi que les différents « Toolkits XML », définitivement obsolètes.

Patrick THOMAS

www.LesCahiersDeLIBMi.fr