Etude comparative des outils de manipulation XML pour Oracle 9i et DB2 v.8.2

Gabriel Gigliotti
Mémoire dirigé par Monsieur Esteban Zimanyi

16 août 2006

Mémoire réalisé en vue de l'obtention
du grade de licencié en informatique

UNIVERSITÉ LIBRE DE BRUXELLES

Année 2005-2006

Table des matières

Table des figures

Chapitre 1

Introduction

Aujourd'hui, les avis sont unanimes, XML est devenu le format de prédilection pour l'échange de données non seulement sur Internet mais aussi au niveau inter et intra entreprise [1]. Mais l'entreposage des informations contenues dans un document XML vers une base de données est loin d'être une opération triviale. Les bases de données relationnelles sont aujourd'hui le support le plus utilisé pour la persistance de données. Il est donc logique de vouloir sauvegarder des documents XML dans une base relationnelle. C'est pourquoi chaque fabriquant de SGBD a développé ses propres techniques et ses propres outils pour interagir avec XML.

Mais avant de réellement commencer à parler de XML et des bases de données, essayons de répondre à une question cruciale : « XML constitue t-il une base de données en soi ? ». Nous pouvons dire qu'un document XML peut être considéré simplement comme étant une collection de données. XML en cela n'est guère différent des autres fichiers, car, après tout, tous les fichiers contiennent d'une certaine manière des données. En tant que format de « base de données », XML présente à ce titre certains avantages. Il est, par exemple, auto-descriptif, car les balises permettent de décrire la structure et le type des noms des données et il permet aussi de décrire les données sous la forme d'un arbre hiérarchique ou d'un graphe. Toutefois, il présente

[1] La place que le XML s'est créée au sein de la technologie B2B " business to business " (il s'agit de l'échange de données et de services entre professionnels) en constitue un exemple significatif.

aussi quelques inconvénients. Il est verbeux et l'accès aux données est ralenti à cause de l'analyse de la structure du document et de la conversion de ce dernier en arbre en mémoire (il s'agit du parsing). Une question plus intéressante consiste à se demander si XML et les technologies qui lui sont associées constituent une « base de données » dans un sens plus large, c'est-à-dire au sens d'un SGBD. La réponse à cette question est complexe. Elle peut être positive, dans la mesure où XML fournit plusieurs des caractéristiques que l'on retrouve dans les bases de données : le stockage (les documents XML), les schémas qui définissent la structure des documents (DTD, XML Schemas, etc.), des langages de requête (XQuery, XPath, XQL, etc.), des interfaces de programmation (SAX, DOM, JDOM), et ainsi de suite. Mais elle peut être négative puisque de nombreuses caractéristiques présentes dans les bases de données lui font défaut : un stockage efficace, les index, la sécurité, les transactions et l'intégrité des données, l'accès multi-utilisateur, les déclencheurs [2], les requêtes sur plusieurs documents, etc.

Ainsi, bien qu'il soit possible d'utiliser un ou plusieurs documents XML comme une base de données dans les environnements rudimentaires qui présentent une faible quantité de données, qui comportent peu d'utilisateurs, et qui se satisfont de performances modestes, cela ne conviendra pas dans la plupart des environnements de production qui disposent de nombreux utilisateurs, requièrent une stricte intégrité des données et ont besoin de bonnes performances. Un fichier *.ini* [3] constitue un bon exemple de type de « base de données » pour lequel un document XML est approprié. Il est effectivement bien plus facile et intuitif d'inventer un petit langage XML et d'écrire une application SAX permettant d'interpréter ce langage plutôt que d'écrire un analyseur de fichier au format texte délimité par des virgules. De plus, XML permet d'imbriquer les entrées, ce qui est difficile à réaliser avec des fichiers au format texte délimité par des virgules. Il ne s'agit pourtant pas vraiment d'une base de données puisque le document XML est lu et écrit linéairement au début et à la fin de l'application. Il existe d'autres exemples plus sophistiqués de collections de données pour lesquelles un document XML pourrait être utilisé en tant que base de donnée : les listes de contacts personnels

[2] Plus communément appelé triggers.

[3] Fichier contenant les informations de configuration d'une application.

(noms, numéros de téléphone, adresses, etc.), les liens favoris dans un navigateur... Cependant, étant donné le coût réduit et la facilité d'utilisation de bases de données telle que Access, il semble, même pour les cas évoqués, qu'il existe peu de raisons d'utiliser un document XML en tant que base de données. Le seul véritable avantage d'un document XML est sa portabilité, et d'ailleurs, c'est un avantage moins crucial qu'il n'y paraît quand on considère le grand nombre d'outils disponibles pour exporter sous forme XML une base de données.

1.1 Pourquoi utiliser une base de données ?

La première question à laquelle nous devons répondre quand nous commençons à penser à XML et aux bases de données est « pourquoi utiliser une base de données ? ». Avons-nous des données déjà existantes [4] que nous souhaitons publier ? Recherchons-nous un endroit où stocker nos pages Web ? La base est-elle exploitée par une application de e-commerce dans laquelle XML est utilisé comme vecteur de données ? Les réponses à ces questions influenceront grandement notre choix de base de données et de logiciel intermédiaire [5] (s'il y a lieu) ainsi que la manière d'utiliser cette base. Supposons par exemple que la base de données soit exploitée par une application de e-commerce dans laquelle XML est utilisé comme vecteur de données. Il y a fort à parier que les données possèdent une structure hautement régulière et soient utilisées par des applications non XML. De plus, des notions telles que les entités et les techniques d'encodage utilisées dans les documents XML ne semblent pas importantes ; après tout, ce sont les données qui nous intéressent, pas la façon dont elles sont stockées dans un document XML. Dans ce cas de figure, nous aurons vraisemblablement besoin d'une base de données relationnelle et d'une couche logicielle pour transférer les données entre les documents XML et la base. Et si les applications sont orientées objet, nous pourrions même souhaiter un système capable de stocker ces objets dans la base ou de les sérialiser en XML. Supposons par contre que nous disposions

[4] Legacy data.
[5] Middleware.

d'un site Web constitué d'un grand nombre de documents orientés texte. Nous souhaitons alors non seulement gérer ce site, mais également fournir aux utilisateurs un moyen de recherche sur le contenu. Nos documents ont toutes les chances d'avoir une structure bien moins régulière, et des notions telles que les entités seront probablement importantes parce qu'il s'agit là d'un aspect fondamental de la manière dont les documents sont structurés. Dans ce cas, il serait préférable d'utiliser une base de données XML native ou un système de gestion de contenu (content management system). Cela permettrait de préserver la structure physique des documents, de supporter les transactions au niveau du document, et d'effectuer des interrogations dans un langage de requête XML.

1.2 XML et les SGBD

XML a suscité un intérêt particulier ces dernières années. Il a été reconnu à l'unanimité comme étant le format de prédilection pour l'échange de données sur Internet, au sein des entreprises et entre n'importe quel système d'information. Réputation qu'il a gagnée notamment grâce à certaines qualités irréfutables. En effet, XML est portable et indépendant d'une quelconque base de données existante, ou encore indépendant d'un quelconque langage de programmation. Les grands constructeurs de SGBD ont vite réalisé que l'utilisation croissante du langage XML dans de nombreux domaines a fait naître de nouveaux besoins en matière de stockage et de gestion de données. L'arrivée sur le marché des bases de données natives XML constituait pour eux une grande menace. Il leur fallait donc réagir au plus vite. Ils se sont effectivement tout de suite mis au travail et ont commencé à construire tout un ensemble d'outils permettant d'interagir au mieux avec le monde XML. Les éditeurs de SGBD se sont donc penchés sur le problème du stockage et de la manipulation des flux XML et ont tantôt choisi d'adapter leur moteur (IBM, Oracle, Microsoft...), tantôt préféré concevoir et développer un SGBD nativement XML. La première solution qui vient à l'esprit consiste à exploiter la puissance du modèle relationnel (ou mieux, relationnel-objet) en définissant un ensemble de tables permettant de modéliser la structure de

chaque document XML et d'en stocker les différents composants de manière parcellaire. C'est la technique du *" mapping "*. La complexité d'une telle opération dépend directement de la structure du document à *mapper*. Toutefois, le modèle relationnel n'est pas très adapté à la représentation arborescente et hiérarchique du langage XML. Plus particulièrement, les applications Internet produisent et manipulent de nombreux documents peu structurés, un véritable cauchemar pour les SGBD. Enfin, la notion de hiérarchie, qui existe en XML, s'avère difficile à représenter à l'aide du modèle relationnel. Un SGBD permet de définir, de stocker et de manipuler rapidement de grands volumes de données. Ces dernières peuvent être de différents types (nombres, chaînes de caractères, etc.).
L'essor des applications transactionnelles et le besoin croissant de disposer de structures d'informations complexes ont conduit les éditeurs de SGBD à étendre leurs fonctions à la gestion de données complexes : objets binaires de grande taille (Blob), objets binaires structurés (Clob), données multimédias... Pour stocker du contenu au format XML dans un SGBD, il est possible d'utiliser les Blob (Microsoft SQL Server n'offre d'ailleurs pas d'autre possibilité). Toutefois, le SGBD n'apporte dans ce cas aucune aide aux développeurs d'applications dans la manipulation des données XML ainsi stockées. Il faut donc faire mieux que le mapping ou l'utilisation des Blob. Certains SGBD [6] disposent d'un type de données *"natif XML"*. Naturellement, les systèmes qui intègrent le format natif permettent de définir des tables associant des données traditionnelles et du contenu XML. Mais, là aussi, une fois l'information stockée, il s'agit de permettre aux développeurs de mettre en œuvre des fonctions de recherche, de comparaison, etc.

Les bases de donnée natives (Tamino [9], eXist [1], Xindice [17] etc.) ont fait récemment et font encore l'objet de recherches intensives. Si elles arrivent à convaincre, elles peuvent connaître un bel avenir, dû notamment à un stockage et une récupération optimale de documents XML. De plus, ces bases proposent des facilités pour effectuer des requêtes sur ces documents. Cependant, ces produits sont encore jeunes, ils ne sont pas encore mature. Aussi, certaines technologies dont ils ont besoin telles que le langage de re-

[6] Oracle depuis sa version 9i et IBM a prévu d'offrir un type de données natif XML dans sa nouvelle version 9 nom de code "viper" pour juillet 2006.

quête XML standard XQuery sont encore en discussion au sein de W3C et ne sont pas encore arrivé au stade de recommandation.
Aujourd'hui, l'opinion générale s'accorde à dire que ces bases de données natives ne connaîtront un épanouissement complet que si elles arrivent à être combinées de façon efficaces avec SQL et les bases de données relationnelles dans une architecture globale.
Un critère important à tenir en considération est le fait que les bases de données relationnelles ont atteint aujourd'hui un niveau de maturité significatif, ce qui fait qu'elles sont implantées un peu partout et qu'elles fournissent un excellent degré d'efficacité dans les systèmes dans lesquels nous les retrouvons. Par conséquent, il est encore réellement impensable aujourd'hui de s'en passer complètement au détriment de systèmes orientés tout XML.

1.3 Un choix porté sur Oracle et DB2 UDB

Dans le chapitre 2, nous allons pénétrer plus en profondeur dans le monde XML en essayant de décrire les notions de base de son langage et des technologies qui s'articulent autour. Cela permettra de planter le décor et de parer le lecteur, en lui apportant un arrière plan culturel nécessaire pour comprendre la suite de ce travail.
Ensuite, dans les chapitre 3 et 4, nous nous focaliserons sur les moyens mis en œuvre par **IBM DB2** et **Oracle** pour manipuler les documents XML. Le choix s'est porté sur ces deux grands SGBD parce qu'ils sont incontestablement les plus en verve actuellement et parce qu'ils mènent la course en tête dans la rude compétition qui oppose les vendeurs de SGBD.
Oracle reste le leader prédominant du marché mondial de la base de données et il n'est absolument pas menacé à court terme. A noter aussi que le produit SQL Server de Microsoft grignote du terrain face à ses deux grands rivaux et que cette progression devrait s'accentuer dans un futur proche grâce à son nouveau SGBD, SQL Server 2005.

Le tableau suivant provient d'un article [32] de l'IDC [7] et il permet de visualiser cette tendance :

Worldwide RDBMS Software Revenue by Top 5 Vendor, 2003–2005 ($M)

Vendor	2003	2004	2005	2004 Share (%)	2005 Share (%)	2004–2005 Growth (%)
Oracle Corporation	5,362.7	5,982.4	6,494.7	45.0	44.6	8.6
IBM	2,825.0	2,923.0	3,113.0	22.0	21.4	6.5
Microsoft Corporation	1,650.0	2,013.0	2,441.5	15.1	16.8	21.3
Sybase Inc.	442.0	470.9	502.6	3.5	3.5	6.7
NCR Teradata	325.4	390.0	423.0	2.9	2.9	8.5
Other	1,441.3	1,528.8	1,590.8	11.5	10.9	4.1
Total	12,046.4	13,308.1	14,565.6	100.0	100.0	9.4

Note: 2005 values are preliminary estimates.

Source: IDC, 2006

FIG. 1.1 – les différentes parts de marché des grands vendeurs de SGBD de 2003 à 2005 ; chiffres d'IDC.

[7] IDC (*International Data Corporation*) est le premier groupe mondial de conseil et d'étude sur les marchés des technologies.

Chapitre 2

XML

2.1 Avant-propos

Aujourd'hui, XML s'infiltre dans pratiquement tous les systèmes d'informations. Mais que fait réellement cette technologie ? et à quoi peut-elle servir au sein d'un système d'information ? Avec l'envol de l'ère de la communication et de l'e-business, les besoins d'échanges d'informations connaissent un essor considérable. Jusqu'alors, aucune autre technologie n'avait fait l'unanimité pour répondre aux besoins liés à la communication. En apportant un standard ouvert, portable, facile à mettre en œuvre, à la fois souple et structurant, (et de surcroît gratuit), XML s'est imposé comme le support privilégié pour l'échange de l'information. On en parle beaucoup mais il n'est pas toujours évident d'acquérir rapidement une vision claire de cette technologie et de ce qu'elle peut réellement apporter au sein d'un système d'information.

2.2 Introduction

Très vite, l'*HyperText Markup Language* a fait ses preuves et est devenu une technologie incontournable dans le contexte Internet/Intranet. Toutefois, force est de constater que HTML a des carences et des limites auxquelles il faudrait remédier. Dans le contexte d'intégration des systèmes d'informations existants et de développements effectués à l'aide de technologies dites « nou-

velles » , l'enjeu reste toujours le même : diminuer les temps de développement, de test et de maintenance afin de maximiser les profits.
Sans avoir la prétention de vouloir remplacer le HTML, le XML (*eXtensible Markup Language*) émerge au moment propice du développement des nouvelles technologies et propose des aspects complémentaires à HTML, permettant de pallier un certain nombre de ses carences. Séparation des données et de la présentation, format de données orienté objet, nombreuses technologies associées telles que la mise en forme et la validation syntaxique, portabilité sur tous les systèmes d'exploitation et utilisation possible dans tous les langages de programmation sont autant d'atouts qui font d'XML non pas une simple mode mais bien une technologie qui a sa place dans les applications d'aujourd'hui et de demain. Qu'il s'agisse de développer un site de commerce électronique performant, de bâtir un intranet cohérent, de développer une application extensible et portable ou encore de construire un réseau d'échange de données évolutif entre clients et fournisseurs, XML s'avère un outil dont il devient difficile de se passer. Autour du langage XML gravitent plusieurs technologies. Nous pouvons citer, par exemple, la possibilité d'imposer la structure des documents XML (XML-Schema et DTD), de mettre en forme les informations (XSL), de naviguer dans un document XML (XPath), de générer des liens entre fichiers XML (XLink et XPointer) ou encore d'accéder aux informations de documents XML par la programmation (API DOM et SAX).

La plupart de ces technologies sont standardisées par un organisme indépendant : le *World Wide Web Consortium* (W3C) [12]. Cette organisation internationale regroupe plus de 500 membres originaires de 34 pays. Parmi eux on compte des instituts de recherche (CERN, NASA...), des organisations gouvernementales (NATO Consultation, Command and Control Agency, National Security Agency...), de nombreux groupes industriels (IBM, BEA, Nokia...), etc.

Le W3C a été fondé en 1994 aux Etats-Unis par Tim Berners-Lee. Il a pour rôle l'émission de spécifications concernant les technologies gravitant autour de l'Internet. Le W3C est notamment à l'origine du langage HTML, du XML bien sûr, mais aussi de XHTML ou du protocole HTTP. Du fait de

son indépendance, le W3C a fortement contribué à l'interopérabilité du Web par la création et la promotion de langages et protocoles non propriétaires.

2.3 XML, ses origines

Pour bien comprendre comment est né le XML et comment il a connu un tel essor, nous allons nous attarder quelque peu sur son « soi-disant » ancêtre, le HTML.

2.3.1 HTML, ses avantages et ses limites

Pourquoi parler du HTML maintenant ?

Tout d'abord, HTML et XML sont des langages de description tous deux issus de la technologie SGML (*Standard Generalized Markup Language*), ce qui leur confère de nombreux points communs.

D'autre part, nous ne pouvons que constater que XML n'aurait pas le succès qu'il a aujourd'hui si HTML n'avait pas œuvré en faveur de sa naissance, ces dernières années. En effet, le Web a bénéficié de la force et de la facilité de mise en place de cette technologie dans les entreprises. HTML a prouvé qu'un standard d'échange d'information était devenu indispensable. Très vite le besoin de transférer et de partager non plus des documents formatés mais des données s'autodécrivant, exploitables par tous et faciles à manipuler s'est fait sentir : c'est de ce besoin d'échange de données standardisées qu'est né XML.

Les langages XML et HTML ont de nombreuses similitudes, de par leur origine commune, notamment en ce qui concerne leur syntaxe. Le lecteur ayant déjà manipulé HTML s'en rendra très vite compte.

Pourquoi HTML ne suffit plus ?

Le HTML a été conçu pour permettre de mettre en forme des informations, il n'était pas prévu à l'origine pour être intelligent et autoriser des dia-

logues compliqués en s'éloignant du standard. Ainsi, l'utilisateur qui consulte la page n'a aucun moyen de récupérer les données affichées afin de faire des traitements. Prenons, par exemple, un site de banque à domicile : ce site va vous permettre de consulter vos comptes, émettre des virements, etc., mais, en aucun cas, vous n'allez pouvoir directement, à partir de la page de vos dernières opérations, changer les données brutes pour les sauvegarder ou faire des manipulations, par exemple des tris, sur ces données.

Il faut donc voir HTML comme un moyen de présenter de l'information lisible par tous les navigateurs. Si on reprend des sites de banque à domicile, on s'aperçoit qu'il est impossible de savoir de quelle nature sont les données. Si on essayait d'écrire un programme capable de séparer les données de la présentation, afin d'extraire les lignes d'opérations bancaires de présentation, celui-ci serait très fastidieux. Si vous regardez le code d'une page HTML, par exemple la page récapitulant vos dix dernières opérations bancaires, vous vous apercevez très vite que cette page est composée d'un ensemble de balises, lesquelles fournissent le type de contenu. Est-ce un montant ? Est-ce un intitulé ou un numéro d'opération ? Sans l'intelligence et la connaissance du domaine par l'utilisateur, il n'est pas possible de se prononcer sur la nature d'une donnée à la consultation du code. Alors, pourquoi ne pas créer ses propres balises afin qu'elles présentent de manière compréhensible les données. Le problème est que le langage HTML ne permet pas de créer de nouvelle balise car, ce n'est pas un langage *extensible*.

La technologie HTML n'est donc applicable que lorsque les informations contenues dans les balises n'ont pas à faire l'objet de traitement spécifique.

2.4 XML, le langage

2.4.1 Un standard

Un groupe de sociétés, d'organisations et d'individus se rencontre régulièrement afin de discuter de ce qui sera ou ne sera pas dans le standard. La technologie XML est régie par la recommandation officielle du Consortium W3C (World Wide Web Consortium). Ce consortium regroupe des sociétés

prestigieuses. Globalement, ce sont des sociétés impliquées dans les technologies web et des éditeurs de logiciel, intéressés par l'adoption et le respect du standard pour leurs logiciels. On peut compter entre autres, au sein de ce groupe de travail des sociétés telles que Microsoft, Netscape, IBM, Sun Microsystems ou encore Oracle.

La technologie XML a été définie et spécifiée par le XML Working Group formé en 1996 par le W3C. C'est en 1998 que le projet XML est devenu le standard que l'on connaît aujourd'hui.

Un certain nombre de sociétés annonce le respect du standard, c'est le cas de :

- Sun Microsystem.
- IBM.
- Oracle.
- Netscape Communications.
- Microsoft Corporation.
- Adobe System Inc.
- Hewlett Packard Co.
- Corel Corporation.

2.5 Les apports de XML

XML et HTML ont la même origine, SGML, mais ce sont toutefois des technologies à ne pas confondre. Le langage XML est un métalangage, c'est-à-dire qu'il est extensible à souhait : il permet de créer de nouveaux langages. L'objectif principal du XML Working Group au moment de sa formation était de créer une technologie universelle pour structurer l'information ; cette technologie se devait d'être particulièrement adaptée à la diffusion et à l'échange d'information, notamment à travers Internet. Il ne faut pas percevoir XML comme une application mais bien comme une forme d'expression standardisée caractérisée par des règles de grammaire. Le langage XML est un standard ouvert, gratuit, libre de droits, adapté au web et indépendant des plates-formes, des langages et des systèmes d'exploitation. Le langage XML, avec

sa volonté de fournir une information structurée et s'autodécrivant, favor l'organisation des éléments et leur sémantique. Néanmoins, il conserve le même esprit que HTML : utilisation de balises, simplicité et de surcroît adapté au web. Le langage XML met à disposition un moyen de séparer le fond de la forme. Sa capacité à décrire les données et les relations entre elles lui promet un bel avenir, aussi, les documents XML sont simplement des documents enregistrés au format texte, format qui est bien évidemment exploitable par tous les systèmes d'exploitation. L'utilisation du langage XML dans un projet informatique fournit une garantie d'interopérabilité des applications. Le format texte est compréhensible par tous les systèmes d'exploitation, ce qui rend la communication possible entre les applications fonctionnant sur des plates-formes potentiellement différentes et écrites dans des langages hétérogènes. L'un des arguments clés des promoteurs du langage est de mettre en surbrillance la lisibilité des documents décrits avec le langage XML. Si le concepteur de la grammaire d'un document XML définit des balises possédant des noms explicites, le document sera lisible par n'importe quel être humain et exploitable par la machine ; d'autre part, le langage en lui-même n'est pas compliqué. Cependant, des grammaires trop longues ou des noms trop courts et non explicites peuvent rendre la compréhension du document plus ardue. Aussi, un autre avantage à signaler est le fait que les documents XML étant de simples fichiers au format texte implique qu'un simple éditeur de fichier texte suffise à créer, modifier ou consulter du code XML.

2.6 XML vu de l'intérieur

Le but premier de ce chapitre est de présenter le langage XML et ses dérivés via une série d'exemples. Cela nous permettra de nous familiariser progressivement à ce langage.

Le langage XML est basé sur l'utilisation et la création de balises. Toutes les informations contenues dans un document XML sont encadrées par les balises XML. Ces balises divisent le document en des containers d'informations que l'on appelle les *éléments* [33]. Ces éléments constituent l'emballage de l'information, ils permettent de donner un label à une information offrant

par la même occasion le moyen de « sémantiser » le contenu, ils permettent aussi de structurer un document . Cette structure étant définie par la façon d'imbriquer les éléments via un mécanisme de balises ouvrantes et fermantes. Voici un exemple simple de document XML reprenant ces concepts :

```
<?xml version=  "1.0" encoding="UTF-8">
<joueur id= "1" >
  <nom> Accou </nom>
  <prenom> Frédéric </prenom>
  <date_de_naissance> 30/10/1974</date_de_naissance>
  <adresse>
    <numero>68</numero>
    <rue>Haute</rue>
    <code_postal>5140</code_postal>
    <ville>Ligny</ville>
    <pays/>
    <boite_postale/>
  </adresse>
</joueur>
```

FIG. 2.1 – Exemple simple de document XML

Dans cet exemple, nous remarquons que les données sont toujours encapsulées dans une balise ouvrante et un balise fermante portant le même nom et constituant un élément (la différence entre minuscules et majuscules pour le label des éléments étant significative pour l'interpréteur). A signaler aussi que certains éléments peuvent ne contenir aucune information, ce sont des éléments vides, cela signifie qu'il s'agit d'une donnée optionnelle, non obligatoire. Un élément vide peut être représenté par la présence d'une seule balise fermante comme ceci, $<element\_vide/>$. Dans notre exemple, les éléments *pays* et *boite_postale* constituent les deux uniques éléments vides du document.
Dans ce même exemple, nous pouvons voir que l'élément joueur possède un

attribut id, dont la valeur et à 1. Les attributs constituent de petites informations qui viennent compléter ou décrire un élément. La valeur d'un attribut doit impérativement être contenue entre guillemets. XML ne s'occupe pas de la mise en forme mais uniquement de la description des données, il délègue la mise en forme aux feuilles de styles XSL, que nous aborderons un peu plus tard.

2.6.1 XML et ses dérivés

Un grand nombre de technologies gravitent autour du XML. Nous allons introduire brièvement les quelques technologies XML qui seront abordées dans la suite de ce travail pour en faciliter la compréhension au lecteur.

Le parser XML et DOM

Tout document XML peut être décrit sous la forme d'une structure arborescente. Cette transformation d'un document XML en un arbre d'objet est réalisée par le *parser*. Après s'être assuré que le document soit *bien formé* [1] le parser construit un arbre d'objet en mémoire représentant le document XML ainsi parsé. Il s'agit du modèle de représentation DOM [33] (Document Object Model) définissant des objets destinés à représenter les différents nœuds constitutifs d'un document XML. Le parser est en quelque sorte à XML ce que le compilateur est aux langages de programmation. Ensuite, là où le compilateur génère un arbre puis un exécutable, la parser s'arrête à la construction de l'arbre [22]. Comme dans la conception d'un arbre quelconque, ce dernier est constitué de nœuds et de feuilles [30]. Les nœuds correspondent au métadonnées, il en existe sept exactement :

- les éléments.
- les attributs.
- les commentaires.
- les références à des entités.
- les instructions de traitements.

[1] Un document est bien formé s'il est syntaxiquement correct.

- les sections de CDATA.
- la déclaration d'un type de document.

Les feuilles correspondent à l'information en tant que telle.

Voici un exemple de code XML et sa représentation arborescente après parsing :

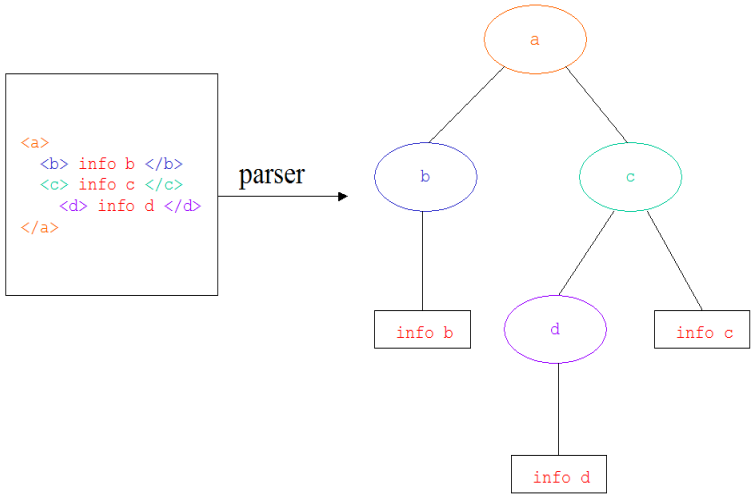


FIG. 2.2 – document XML et sa représentation arborescente

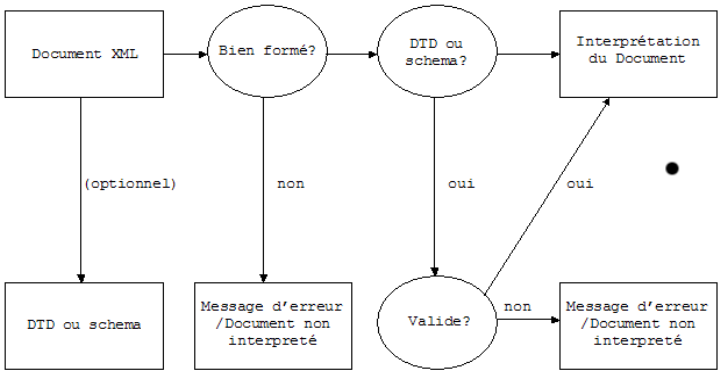
Un document XML est dit *bien formé* si le parser considère que celui-ci est syntaxiquement correct. Par exemple, il est hors de question d'interpréter un document XML si une balise est ouverte et que la balise fermante manque à l'appel. Aussi, il est interdit d'avoir recours à des balises imbriquées ou croisées, c'est-à-dire que le fils d'une balise doit impérativement se refermer avant la fermeture de son père. En conclusion, si le parser indique que le document est mal formé, il se peut que ce soit parce qu'il existe :

- une faute d'orthographe dans une ou plusieurs balises
- un oubli de fermeture de balise
- un croisement entre des balises

Une fois l'arbre d'objet créé en mémoire, il est manipulable par une application grâce à des méthodes d'accès en consultation et en modification fournies par l'API DOM.

Vérification de la validité d'un document : DTD et Schemas

Un document XML doit dans tout les cas être bien formé mais on peut aussi décider de le faire *valider*. Un document est dit valide s'il répond aux règles définies dans la grammaire associée au document XML. Le processus qui permet de s'assurer du respect de la structure définie est nommé la validation. Cette grammaire est définie sous la forme d'une *DTD* ou d'un *XML-Schema*. Une *DTD* possède une syntaxe particulière, tandis que le *XML-Schema* correspond à un document XML en tant que tel constitué de balises et de données comme n'importe quel autre document XML.



Les DTD

Voici un exemple de DTD :

```
<!ELEMENT client (nom, prenom, adresse)>
<!ATTLIST client id CDATA #REQUIRED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT adresse (rue, numero, cp, bte)>
<!ELEMENT rue (#PCDATA)>
```

```
<!ELEMENT numero (#PCDATA)>
<!ELEMENT cp (#PCDATA)>
<!ELEMENT bte (#PCDATA)>
```

La déclaration d'un élément fournit au parser les informations suivantes : son nom, le modèle du contenu et, éventuellement, les attributs associés à cet élément.
Par exemple, la déclaration suivante :

```
<!ELEMENT nom (#PCDATA)>
```

signifie que l'élément qui a pour nom `nom` a pour modèle du contenu `#PCDATA` ce qui indique qu'il s'agit d'une simple chaîne de caractères. Dans la définition de l'adresse :

```
<!ELEMENT adresse (rue, numero, cp, bte)>
```

nous comprenons que l'élément `<adresse>` devra contenir obligatoirement les quatre éléments`<rue>`, `<numero>`, `<cp>` et `<bte>`. Le problème rencontré avec cette définition concerne le caractère obligatoire de tous les champs. Si nous voulons rendre l'élément `<bte>` optionnel, il suffira de rajouter un point d'interrogation derrière le champ bte dans la définition de l'élément adresse. La définition de l'adresse devient alors :

```
<!ELEMENT adresse (rue, numero, cp, bte? )>
```

Si l'on veut qu'un élément puisse apparaître zéro ou plusieurs fois alors nous utiliserons le caractère * , et si l'on veut qu'un élément puisse apparaître une ou plusieurs fois alors nous utiliserons le caractère +. Si aucune contrainte d'occurrence n'est spécifiée, l'élément doit apparaître une et une seule fois.

Voici un petit tableau reprenant les trois contraintes d'occurrence disponible :

Contrainte d'occurrence	**Signification**
?	0 ou 1
*	0 ou plusieurs
+	1 ou plusieurs

Notez que si la DTD définie plus haut s'appelle "adresse.dtd", alors pour associer cette DTD à un document XML il suffit d'insérer dans ce dernier la ligne suivante :

```
<! DOCTYPE adresse SYSTEM "adresse.dtd">
```

La déclaration d'un attribut fournit au parser les informations suivantes : le nom de l'élément auquel il se rapporte, le nom de l'attribut, le type de l'attribut et la valeur par défaut ou une contrainte d'occurrence. Les types d'attributs seront le plus souvent soit une chaîne, soit une énumération`(valeur 1 | valeur2 | valeur3)`. En terme de contraintes d'occurrence et de valeurs par défaut, les quatre qualificatifs suivants sont disponibles :

- `#REQUIRED` : l'attribut doit être obligatoirement présent.
- `#IMPLIED` : l'attribut peut manquer à l'appel.
- `#FIXED` : le domaine de valeurs possible est inclus dans la déclaration et prend la première valeur de la liste comme valeur par défaut.
- `( "valeur" )` : indique la valeur prise par défaut dans la liste si aucun élément n'est spécifié dans le document.

Par exemple, la définition de l'attribut suivant :

```
<! ATTLIST club id CDATA #REQUIRED existe (oui|non) "oui" #IMPLIED>
```

signifie que l'élément `club` doit obligatoirement posséder un attribut `id` de type chaîne de caractères. L'élément `club` peut aussi avoir un autre attribut `existe` non obligatoire ayant deux valeurs possibles (`oui` ou `non`) avec `"oui"` comme valeur par défaut.

Xml-Schemas

« *En utilisant un schéma XML, un document et ses données peuvent être modélisés et validés avec beaucoup plus de précision que ce que permet aujourd'hui une DTD* » [22]. XML Schema [16] est un langage formel de description orienté objet de documents XML qui sert à documenter, valider et automatiser leur traitement [36]. Leur contenu est au format XML, contrairement aux DTD. Ceci a pour avantage qu'il peuvent être édités par un éditeur XML classique. Un schéma XML à le même objectif qu'une DTD : valider un document XML. Aussi, deux parsers différents doivent absolument avoir le même résultat de validation pour un même document se basant sur un schéma XML.

Avantages des schémas XML par rapport aux DTD

Les schémas XML permettent de décrire l'imbrication et l'ordre d'apparition des éléments et des attributs. Les schémas XML sont des documents XML, ainsi, ils pourront être manipulés par des API comme n'importe quel autre document XML. Les schémas XML permettent de typer les données, type simple et type complexe sont disponible. Avec les DTD, seul le type chaîne de caractère est à disposition.

Nous n'allons pas nous attarder à expliquer comment définir un schéma XML puisque nous n'utilisons, dans les exemples de ce travail, que des DTD pour valider un document. Cependant, tout laisse supposer que les schémas XML vont, dans un futur proche, remplacer les DTD.

XPath

XPath [18] est un standard du W3C qui permet d'écrire des expressions ou filtres de sélection[2] de nœuds dans des modèles DOM. En d'autres termes, cette technologie permet de naviguer dans un document XML et d'effectuer une sélection d'éléments plus ou moins précise dans ce dernier. La syntaxe de ce langage a évolué et a intégré un certain nombre de notions déjà dévelop-

[2]Pattern matching

pées dans XQL[3] pour aboutir à la version 1.0 finale en novembre 1999. Les expressions XPath sont toujours évaluées sur base d'un chemin ou d'un axe qui sert à présélectionner un certain nombre de nœuds. Ensuite, nous pouvons affiner notre sélection sur base de « conditions de sélections » qui seront appliquées sur l'ensemble des nœuds présélectionnés. Nous pouvons encore affiner notre sélection en ajoutant un prédicat facultatif entre crochets après les conditions de sélections.
L'axe de sélection et les conditions sont séparés par deux fois deux points, ce qui donne la syntaxe :

```
NOM_DE_L_AXE : :CONDITIONS_DE_SELECTIONS[PREDICATS]
```

[3]XML Query Language

Il existe treize axes de sélections différents, en voici la liste exhaustive :

Nom de l'axe	**Description**
Child	Les nœuds fils du nœud courant mais pas les descendants
Parent	Le nœud parent du nœud courant
Self	Le nœud courant
Attribute	Les attributs du nœud courant
Descendant	Les nœuds fils du nœud courant ainsi que les descendants
Descendent-or-self	Les nœuds de l'axe descendant plus le nœud courant
Ancestor	Les ancêtres du nœud courant
Ancestor-or-self	Les ancêtres du nœud courant et le nœud courant
Preciding-sibling	Les nœuds frères du nœud courant le précédant dans l'ordre de déclaration
Following-sibling	Les nœuds frères du nœud courant le suivant dans l'ordre de déclaration
Preciding	Tous les nœuds précédant le nœud courant dans l'ordre de déclaration
Following	Tous les nœuds suivant le nœud courant dans l'ordre de déclaration
namespace	Tous les nœuds de type namespace du nœud courant

Après le choix de l'axe, la deuxième partie de l'expression XPath consiste dans la plupart des cas :

- en un symbole * , désignant tous les nœuds de l'axe.
- en un nom , désignant tous les éléments de même nom.

Voici quelques exemples :

Expression XPath	Explication
Child : : *	Parmi tous les nœuds fils, tout sélectionner
Child : : nom	Parmi tous les nœuds fils, sélectionner tous les nœuds nom
Descendant : : *	Parmi tous les nœuds fils et descendants des nœuds fils, tout sélectionner
Child : : nom[position()=1]	Parmi tous les nœuds fils, sélectionner uniquement le premier nœud nom

XSLT

XSL [14] (*eXtensible Stylesheet Language*) est une spécification du W3C. Elle est composée de deux technologies complémentaires prévues pour la présentation (XSL-FO, *XSL Formatting Objects*) et XSLT [15](*XSLTransformations*). Bien que XSLT ait été conçu pour supporter XSL-FO, il est apparu comme la tehnologie préférée pour toutes les sortes de transformations [28]. La transformation XML vers HTML est la plus courante mais la transformation peut se faire au sein même d'un document XML. Il s'agit d'un langage déclaratif, cela signifie qu'il ne spécifie **pas comment** effectuer les transformations (algorithme), mais plutôt **quoi** transformer.

Ces transformations seront opérées par un processeur XSLT. Un processeur XSLT est un moteur prenant en entrée un document XSLT (décrivant la transformation à effectuer) et un document XML à transformer, et produisant en sortie un flux, pouvant être utilisé en tant que nouveau document XML ou comme code source d'autres langages (HTML, texte, etc.) [7]. Ce processeur crée une structure logique arborescente du document XML (il s'agit de **l'arbre source**) et lui fait subir des transformations selon des règles de transformations (*template rules*) qui sont définies dans la feuille de style XSL.

Au terme de ce processus un **arbre résultat** est construit.

XSLT utilise la technologie XPath que nous avons décrite (supra) pour exprimer une requête de sélection de nœuds de l'arbre source. Cette sélection se fait via une expression XPath, c'est-à-dire une chaîne de caractères [4] qui localise un ou plusieurs nœuds de l'arbre source.

Il faut noter que l'arbre résultat peut être totalement différent, au niveau de sa structure, de l'arbre source.

Le langage permet offre beaucoup de fonctionnalités au développeur : variables, paramètres, tests, boucles, fonctions, etc. Voici un bref aperçu des principales fonctions :

<xsl :template>
Cette fonction est très importante dans la mesure où elle permet de définir une règle de transformation pour un ensemble de nœuds spécifique.
Syntaxe :

```
<xsl :template match= "pattern">
...
</xsl :template>
```

Le paramètre pattern correspond à un pattern de sélection XPath. Pour chaque nœud correspondant au(x) critère(s) de sélection de l'attribut match, le processeur XSLT évalue le contenu de la balise `<xsl :template>` et l'ajoute dans l'arbre résultat.

<xsl :apply-template>
Cette fonction permet de réitérer sur les descendants directs d'un nœud et donc de continuer le processus en traitant les nœuds fils. Cette fonction per-

[4] A la façon des chemins d'accès au système de fichiers Unix.

met donc d'effectuer des traitements récursifs sur les nœuds de son choix. Un attribut select optionnel permet de ne réitérer que sur certains sous-nœuds que l'on précise à l'aide d'une expression XPath.

<xsl :value-of>
Cette fonction permet d'ajouter dans l'arbre résultat une chaîne de caractères dont la valeur est déterminée à partir de l'attribut select par une expression XPath.
Syntaxe :
`<xsl :value-of select= "pattern">`

<xsl :if>
Cette fonction permet de conditionner les traitements XSLT de son choix par une expresion XPath présente dans l'attribut test.
Syntaxe :
`<xsl :if test= "condition">`

<xsl :choose>, <xsl :when> et <xsl :otherwise>
Ces functions offrent une structure conditionnelle identique au switch / case / default du C++. Les conditions étant définies par une expresion XPath présente dans l'attribut test.
Syntaxe :

```
<xsl :choose>
  <xsl :when test="condition">
  ...
  </xsl :when>
  <xsl :otherwise>
  ...
  </xsl :otherwise>
</xsl :choose>
```

<xsl :for-each>

Cette fonction permet d'effectuer des itérations non récursives. Elle inique au moyen d'un attribut select exprimé en XPath quel est l'ensemble des nœuds sur lesquels le processeur devra faire les itérations.
Syntaxe :
`<xsl :for-each select= "pattern">`

Chapitre 3

Manipuler des documents XML avec DB2

Maintenant que nous avons vu les notions élémentaires du langage XML, nous sommes parés à aborder les différentes façons de manipuler des documents XML au sein des systèmes de gestion de bases de données. Dans ce chapitre, nous portons notre attention sur le SGBDR **DB2** d'IBM. Nous allons voir quels outils sont nécessaires à l'extraction et au stockage de documents XML dans une base de données DB2 et comment manipuler ces outils dans un contexte particulier.

3.1 DB2 et XML

La famille IBM DB2 UDB [3] (Universal Database) fournit un ensemble de produits pour la gestion de données objets-relationnelles au sein d'applications XML. XML Extender [4], est un module d'extension qui vient se greffer à DB2 UDB pour permettre l'intégration de documents XML. En quelques mots, XML Extender permet de stocker et de récupérer des documents XML en entier dans une colonne, mais aussi de décomposer des documents XML en vue de stocker les fragments dans plusieurs tables. DB2 UDB pour Linux et Windows est fourni avec un support SQL qui inclut la création de fragments XML à partir de données relationnelles grâce à l'utilisation de

fonctions SQL/XML.

3.2 XML Extender

3.2.1 Introduction à XML Extender

DB2 XML Extender permet de stocker des documents XML, d'y accéder, mais aussi de composer des documents XML à partir des données relationnelles existantes dans les tables d'une base DB2 ou d'en décomposer le contenu et de le stocker dans une base DB2 . En somme, XML Extender offre un ensemble d'outils qui permet de manipuler des documents XML dans une base DB2.

Cette manipulation de document XML est facilitée grâce notamment à un ensemble de nouveaux types de données, de fonctions, de procédures stockées (*stored procedures*) [24]. Il existe deux modes de stockage pour intégrer des documents XML dans DB2.

Le premier permet à des documents XML d'être stockés simplement, à l'état brut dans une colonne prévue à cet usage. Ce type de stockage est appelé stockage *XML Column*. Certains éléments et attributs peuvent être mis en correspondance via des tables appelées les *side tables* [1]. Le but visé étant d'accélérer le temps de réponse des recherches dans le document sur des éléments spécifiques, une fois que ce dernier est stocké dans la base.
Le deuxième mode de stockage prévoit d'exploser le document XML et de l'insérer dans plusieurs tables relationnelles-objet pour un stockage plus affiné, ce type de stockage plus ventilé est appelé stockage *XML Collections*.

La génération de documents à partir d'un contenu nativement XML ou non XML est facilitée par l'existence de fonctions spécifiques. Nous préférerons l'un ou l'autre mode de stockage en fonction de ce que l'on veut faire

[1] Voir infra.

avec le document XML [2].

XML Extender utilise des DAD [3] (*Document Access Definition*), pour *mapper* les éléments et les attributs XML avec les données relationnelles. Cela permettra de considérer les données XML comme des tables standards dans les requêtes SQL.

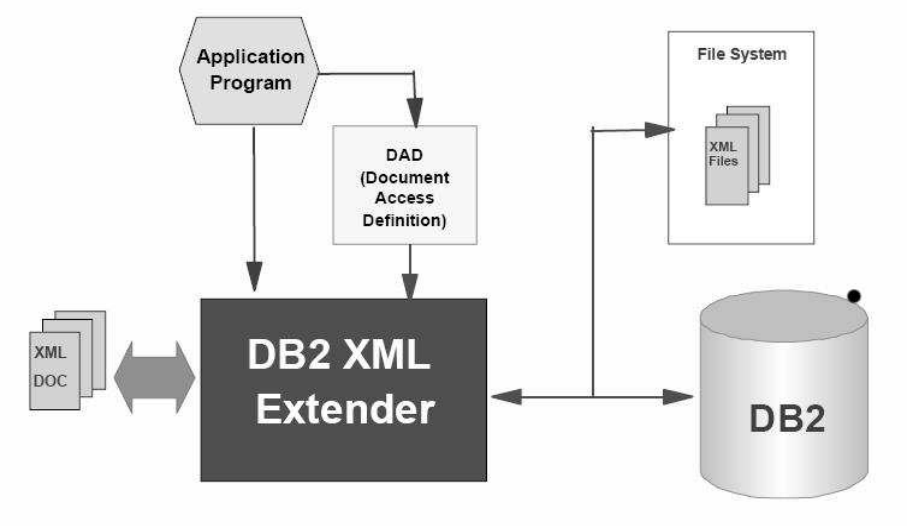


FIG. 3.1 – architecture de XML Extender [5]

3.2.2 Pourquoi XML Extender ?

L'avantage significatif de XML Extender est l'utilisation du langage SQL pour accéder aux documents XML et y effectuer des requêtes. Cela simplifie grandement la tâche d'accès et de requêtes sur ces documents. Il se trouve que XML Extender est le pur produit XML d'IBM et a été réalisé dans le but d'interagir au mieux avec les bases de données DB2 UDB. Avec XML Extender, nous pouvons donc tirer le meilleur parti de la puissance de DB2 dans les applications XML.

[2] Voir infra.

[3] Le DAD est un fichier XML dans lequel on retrouve les règles de mapping entre le document XML et les tables relationnelles.

3.2.3 Comment DB2 et XML interagissent-ils ?

XML Extender offre les dispositifs suivants pour permettre de traiter et exploiter les documents XML avec DB2 [25] :

- Des outils d'administration pour nous aider à gérer l'intégration des données XML dans les tables relationnelles.
- Des méthodes de stockage et d'accès pour les données XML au sein de la base.
- Un *DTD repository* pour stocker les DTDs qui serviront à valider les documents XML.
- Un fichier de mapping appelé DAD, utilisé pour faire la correspondance entre le document XML et les données relationnelles.

Les outils d'administration :
XML Extender fournit divers outils d'administration.
Par exemple :

- La commande **dxxadm** permet d'effectuer certaines tâches d'administration via le CLP [4], interpréteur de commande fournit avec IBM DB2 UDB.
- Les procédures stockées (*stored procedures*) d'administration de XML Extender permettent d'invoquer les commandes d'administration à partir du CLP ou à partir d'un programme.

Les méthodes de stockage et d'accès :
Il s'agit des 2 modes de stockages XML Columns et XML Collections déjà évoqués. Chacun possède ses méthodes pour pouvoir, d'une part, insérer de façon intacte le document XML dans une colonne prévue à cet effet et, d'autre part, permettre de composer un document XML à partir de données éclatées sur plusieurs tables ou décomposer un document XML et caser chaque fragment dans plusieurs tables relationnelles.

[4] Command Line Processor.

Les DTDs :
XML Extender permet aussi de stocker des DTDs , grammaire à laquelle le document doit se coller pour être « valide ». Quand une base de donnée est prête à interagir avec XML Extender[5] (via la commande **dxxadm enable_db < database_name >**), DB2 crée des tables qui lui permettront de travailler avec XML, une de ces tables se nomme *db2xml.DTD_REF* et chaque ligne de cette table contient une DTD avec des métadonnées complémentaires. Les utilisateurs peuvent y insérer leur propre DTD.

Le fichier DAD :
Le fichier DAD est un document XML qui mappe la structure du document XML avec une table relationnelle. L'utilisation d'un fichier DAD se fait à la fois dans le mode XML Column et dans le mode XML Collections. Nous y reviendrons pus en détails dans la suite de ce chapitre.

3.2.4 La technologie XML Extender

L'architecture XML Extender fournit un support fonctionnel pour répondre a des besoins typique dans le cadre de manipulations de données XML, comme le stockage, l'accès et la récupération de données XML. Comme mentionné précédemment, XML Extender possède deux modes de stockage : XML Columns et XML Collections. Dans les deux cas de figure un fichier DAD sera présent pour établir la correspondance entre le document XML et les tables relationnelles.

XML Column

Dans DB2, l'utilisation de XML Column permet de stocker un document XML entier dans son format natif et dans une colonne prévue à cet effet. Evidemment, seule la récupération entière du document est possible. Cette méthode de stockage et d'accès permet de garder le document intact, tout en offrant la possibilité d'indexer et d'effectuer des recherches sur des éléments

[5]Dans la suite de ce travail nous utiliserons le terme permise XML pour dire qu'un base est prête à interagir avec XML Extender.

que nous aurons préalablement choisis comme étant les éléments qui seront le plus souvent sollicités. Après avoir spécifié qu'une base est permise XML (via la fonction *enable_db*) , les types de données UDT (*user-defined types*) suivants sont alors disponibles pour cette base :

XMLVARCHAR

Cet UDT est utilisé pour les documents XML de petite taille, allant jusqu'à 3KB.

XMLCLOB

Cet UDT est utilisé pour les documents XML d'une taille maximale de 2GB.

XMLFILE

Cet UDT spécial est utilisé pour les documents XML stockés en dehors de la base de données ou stockés dans un fichier sur un file system local.

Les deux premiers types étant de loin les plus utilisés.

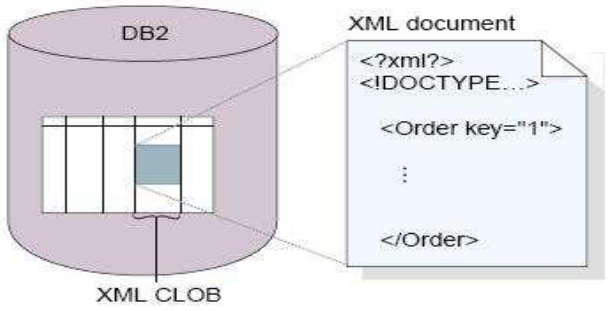


FIG. 3.2 – un document XML dans une colonne de type XMLCLOB [25]

L'utilisation de cette méthode, à savoir stocker l'entièreté du document dans une colonne, est avantageuse si un ou plusieurs critères suivants sont présents :

- Le document XML existe déjà ; par exemple, si l'on veut archiver des documents XML existants.
- Le document XML doit être souvent lu et rarement mis à jour.
- La performance de la mise à jour n'est pas critique.
- Nous voulons stocker le document de façon intacte.
- Nous voulons garder le document XML en dehors de la base sur un système de fichier local.
- Nous savons quels éléments ou attributs seront fréquemment recherchés. Dans ce cas, il existe la possibilité de créer des index dans des tables externes sur les éléments ou attributs pour lesquels un accès régulier sera effectué.

Fichiers DAD pour XML Column

Ici, le fichier DAD contiendra la balise `<Xcolumn>` qui dénote le fait qu'on travail selon le mode de stockage XML Column.

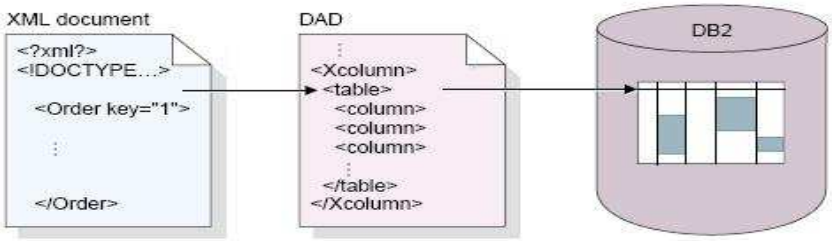


FIG. 3.3 – le fichier DAD permet le mapping entre le document XML et les tables relationnelles [25]

Le fichier DAD va donc mapper les éléments et les attributs qui apparaîtront dans les *side tables* en vue d'accélérer les performances. La définition

de ces side tables se fait dans le fichier DAD et elles sont créées automatiquement lors de l'exécution de la commande suivante :

dxxadm enable_column <database_name> <table_name> <column_name> <path_to_dad_file>

Dans ces side tables apparaîtront les éléments qui auront été préalablement choisis comme étant les éléments susceptibles d'être invoqués régulièrement par l'application.

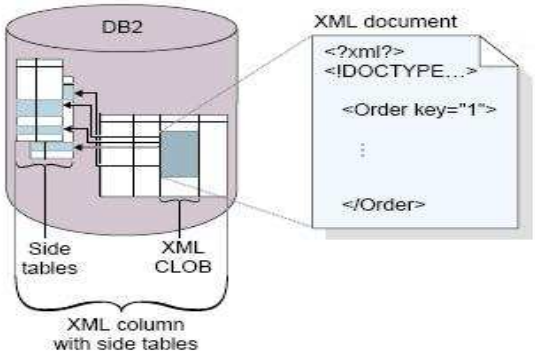


FIG. 3.4 – les side tables dans le cas XML Column [25]

Voici un exemple éclairant :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE DAD SYSTEM "c :\dxx\dtd\dad.dtd">
<DAD>
  <dtdid>Order.dtd</dtdid>
  <validation>YES</validation>
  <Xcolumn>
    <table name = "order">
```

```
    <column name= "customer_num"
            type= "Integer"
            path= "/Order/Customer"
            multi_occurence= "NO">
    </column>
    </table>
    <table name="..">
    ...
    </tabe>
  </Xcolumn>
</DAD>
```

Explications :

1. La ligne `<!DOCTYPE DAD SYSTEM "c :\dxx\dtd\dad.dtd">` permet de spécifier la DTD qui sera utilisée pour valider le fichier DAD.
2. L'ensemble des éléments constituant le DAD est encapsulé dans le tag `<DAD></DAD>`.
3. Si nous désirons valider notre document XML avant que ce dernier soit inséré dans la base de données, il faut alors :
   (a) Insérer la balise suivante pour valider le document : `<dtdid>Order.dtd</dtdid>`

   (b) Ensuite spécifier que la validation doit être effectuée via la ligne : `<validation>YES</validation>`

4. La balise `<Xcolumn></Xcolumn>` signifie simplement que l'on travaille en mode XML column.
5. Pour chaque side table à créer :
   (a) Insérer la balise `<table name="..."></table>`
   (b) Pour chaque colonne que nous voulons faire apparaître dans la side table, préciser le nom de la colonne, le type de la colonne, le chemin dans le document XML qui correspond à cette colonne et

si oui ou non il peut y avoir plusieurs occurrences de cet élément dans le document XML.

Exemple :

```
<table name = "order">
  <column name= "customer_num"
          type= "Integer"
          path= "/Order/Customer"
          multi_occurence= "NO">
  </column>
</table>
```

Dans cet exemple, la première balise signifie que notre side table se nommera `order`, pour la balise `<column>` l'attribut `name` spécifie que le nom de la colonne dans notre side table sera `customer_num`, l'attribut `type` indique que le type de donnée SQL au sein de la table sera un `Integer`, l'attribut `path` spécifie que le chemin d'accès au sein du document XML qui correspond à cette colonne est `/Order/Customer` et l'attribut `Multi_occurrence` indique, dans ce cas-ci, que l'élément `Customer` n'a pas plusieurs occurrences, donc il ne peut apparaître qu'une seule fois.

XML Collection

Dans le cas d'un partage de données spécifiques avec une application ou autre, en XML, il faudra généralement récupérer des données de-ci de-là dans diverses tables relationnelles pour composer le document XML approprié. C'est ici que XML Collection intervient. En effet, l'approche XML Collection sera préférée si l'on veut splitter le document XML dans plusieurs tables relationnelles ou si l'on veut former un document XML à partir de données éparpillées dans plusieurs tables relationnelles. XML Collection est aussi préférable dans le cas où des *update* sont nécessaires sur des petites portions d'un document XML ou sur de petits documents XML dans leur entièreté. Il est aussi préférable lorsque le temps de réponse des *update* est important.

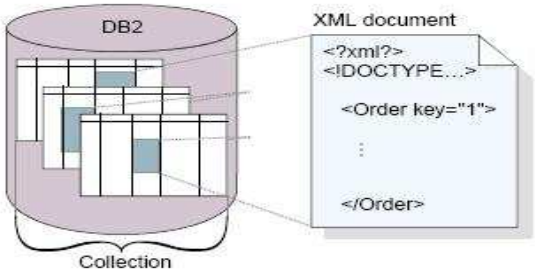


FIG. 3.5 – un document XML stocké dans la base en mode XML Collection [25]

Fichiers DAD pour XML Collection :

Le fichier DAD est d'une importance primordiale. Il *mappe* la structure du fichier XML en tables DB2, et c'est à partir de ce fichier DAD que la composition ou la décomposition va s'effectuer. Dans ce cas-ci, le fichier DAD contiendra la balise `<Xcollection>` qui dénote le fait que l'on travaille selon le mode de stockage XML Collection. Cela signifie que l'on s'apprête, soit à composer un document XML à partir d'un ensemble de tables, soit à décomposer un document XML pour insérer des données intégrées à celui-ci vers un ensemble de tables relationnelles. Le fichier DAD définit la structure arborescente du document XML en utilisant les types de nœuds suivants :

- root_node : identifie l'élément root du document
- element_node : identifie un élément, pouvant être le root élément ou un élément fils
- text_node : représente un élément de type CDATA (Character DATA)
- attribute_node : représente l'attribut d'un élément

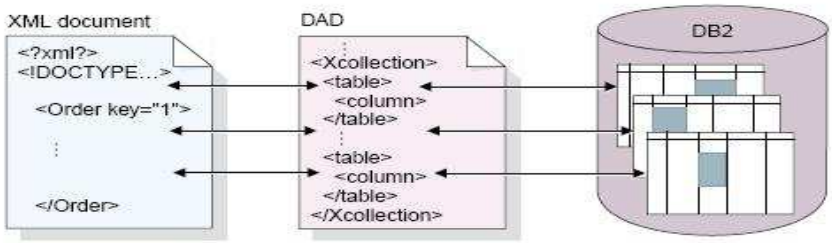


FIG. 3.6 – un fichier DAD dans le cas XML Collection [25]

Le mapping schemas avec XML Collection :

Si XML Collection est utilisé comme mode de stockage il faut mettre en place un *mapping schema*, qui permettra de définir comment les données XML seront représentées dans la base de données relationnelle. Il s'agit donc de réaliser une comparaison entre la structure arborescente du document XML et la structure relationnelle de la base.

Le *mapping schema* est spécifié au sein du fichier DAD dans la balise `<Xcollection>`. Il existe 2 types de *mapping schema*, la reconnaissance de l'un ou l'autre type de mapping s'effectue selon la présence de la balise `<SQL_stmt>` ou de la balise `<RBD_node>`.

SQL mapping

La commande suivante permet, via la procédure stockée **dxxgenx**, de se connecter à la base `db_name`, de génèrer un document XML en suivant les règles de mapping qui se trouvent dans le fichier DAD spécifié par le chemin `path_to_DAD_file` et finalement insère le résultat dans la table

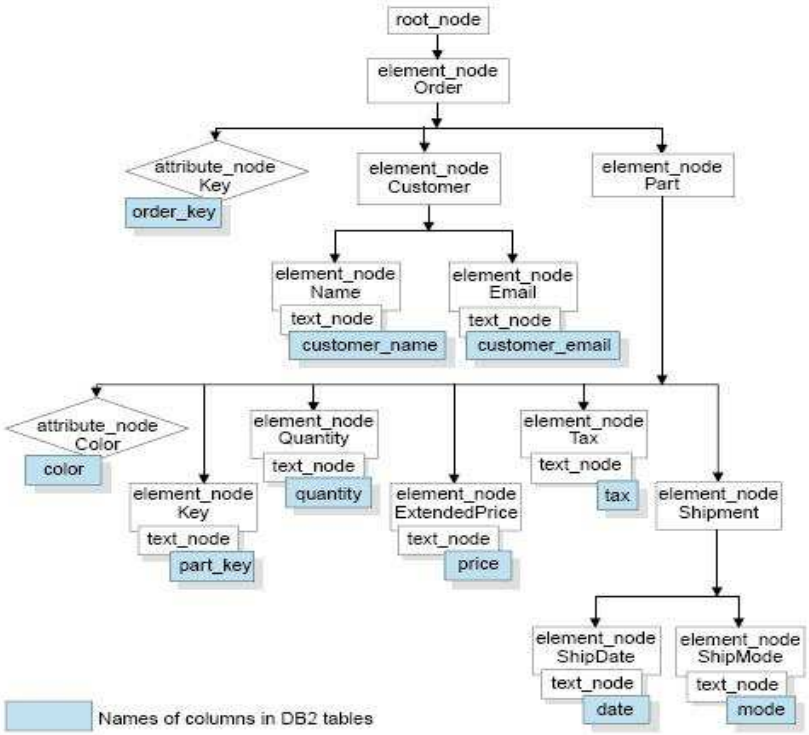


FIG. 3.7 – un exemple de correspondance entre un document XML et sa représentation relationnelle [25]

`result_table` :

dxxgenx <db_name> <path_to_DAD_file> <result_table>

Voici un exemple de fichier DAD, dans le cas du SQL mapping :

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM
"C :\Progra~1\IBM\SQLLIB\samples\db2xml\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
```

```
<SQL_stmt> SELECT *
           FROM club c, member m
           WHERE c.id = m.id_club AND m.victory > 10
           ORDER BY club_name
</SQL_stmt>
<prolog> ?xml version="1.0" ?</prolog>
<doctype>
!DOCTYPE club SYSTEM
"C :\Progra~1\IBM\SQLLIB\samples\db2xml\dtd\club.dtd"
</doctype>
<root_node>
  <element_node name="club">
    <attribute_node name="cle">
      <column name="id"/>
    </attribute_node>
    <attribute_node name="nom">
      <column name="club_name"/>
    </attribute_node>
    <element_node name="membre" multi_occurrence="YES">
      <attribute_node name="id">
      <column name="id_member"/>
      </attribute_node>
      <element_node name="nom">
      <text_node>
      <column name="name"/>
      </text_node>
      </element_node>
      <element_node name="prenom">
      <text_node>
      <column name="surname">
      </text_node>
      </element_node>
      <element_node name="classement">
```

```
      <text_node>
      <column name="classification"/>
      </text_node>
      </element_node>
      <element_node name="victoire">
      <text_node>
      <column name="victory"/>
      </text_node>
      </element_node>
    </element_node>
  </element_node>
</root_node>
</Xcollection>
</DAD>
```

Voici le fichier club.dtd qui y est associé :

```
<?xml version="1.0" encoding="utf-8" ?>
<!ELEMENT club (membre*)>
<!ATTLIST club cle CDATA #REQUIRED nom CDATA #IMPLIED>
<!ELEMENT membre (nom, prenom, classement, victoire)>
<!ATTLIST membre id CDATA #REQUIRED>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT classement (#PCDATA)>
<!ELEMENT victoire (#PCDATA)>
```

Le SQL mapping est utilisé pour la composition uniquement. Il permet d'effectuer un mapping direct à partir de données relationnelles vers un document XML, grâce à une requête SQL incluse dans le fichier DAD. Cette requête SQL étant plus précisément encapsulée dans la balise `<SQL_stmt>` présente à cet effet. Une table temporaire est créée via cette requête. Cette table contient l'ensemble des données qui sert à composer le document XML. Les règles de mapping présentent dans le fichier DAD permettent de faire

la correspondance entre les colonnes de cette table et les éléments du document XML en cours de construction. Le SQL mapping se sert du langage de requête standard SQL pour récupérer les données, ce qui constitue bien entendu un avantage. Aussi, il faut pouvoir effectuer la jointure avec toutes les tables contenant les informations désirées en une seule requête atomique si l'on veut pouvoir utiliser le SQL mapping, dans le cas échéant, il sera alors nécessaire d'utiliser le *RDB_ node mapping*.

RDB_node mapping

Le RDB_node mapping permet d'effectuer une composition ou une décomposition de document XML.

La composition :

La commande permettant de procéder à la composition du document est exactement la même que celle dans le cas du SQL mapping[6] :

dxxgenx <db_name> <path_to_DAD_file> <result_table>

Il faut spécifier que l'on travaille en mode XML Collection au sein du fichier DAD par l'ajout de la balise `<Xcollection>`, mais cette fois, il faut remplacer la balise `<SQL_stmt>` par des balises `<RBD_node>`. Dans une balise `<RBD_node>`, XML Extender trouvera toutes les informations qui lui permettront, dans le cas d'une composition, de savoir où aller chercher les données dans les tables relationnelles et, de cette manière, pouvoir les insérer dans le document XML en cours de construction.

La balise `<RBD_node>` est de la forme :

[6] Cette commande a déjà été expliquée, voir supra.

```
<RBD_node>
  <table name = "table_name"/>
  <column name = "column_name">
    <condition>...</condition>
</RDB_node>
```

Et sa définition DTD est la suivante :

```
<!ELEMENT RDB_node (table+, column?, condition?)>
```

A noter que l'élément `table` doit apparaître au moins une fois et peut avoir plusieurs occurrences, les deux autres éléments étant optionnels (0 ou 1 occurrence).

table
: Détermine le nom de la table dans laquelle XML Extender trouvera la valeur recherchée.

column
: Détermine le nom de la colonne qui contient cette même valeur.

condition
: Permet d'effectuer une sélection sur l'ensemble des valeurs possibles.

Le premier `element_node` dans le fichier DAD représente l'élément racine du document XML. Le `RDB_node` attaché à ce premier élément est « particulier » dans la mesure où les éléments encapsulés dans celui-ci fournissent des informations dont la sémantique est propre à ce premier `RDB_node`. C'est dans ce dernier que nous devons déclarer toutes les tables concernées par le `XML Collection`. En d'autres mots, il s'agit de toutes les tables dans lesquelles nous allons recueillir des informations pour les insérer dans notre document XML. L'élément `column` est ici superflu. L'élément `condition` doit apparaître uniquement si plusieurs tables sont concernées. Dans ce cas, la jointure entre toutes ces tables sera établie dans cet élément `condition`.

Ensuite, pour chaque `attribute_node` et chaque `text_node`, un `RDB_node` est nécessaire. Il reprend alors son sens conventionnel, à savoir, déterminer où se trouve la bonne information dans les tables relationnelles.
Par exemple :

```
<element_node name="victoire">
  <text_node>
    <RDB_node>
    <table name="member"/>
    <column name="name"/>
      <condition> victory > 10
      </condition>
    </RDB_node>
    </text_node>
  </element_node>
```

La portion de document XML suivante signifie que le contenu de l'élément `victoire` sera du texte et la valeur à insérer dans cet élément se trouve dans la table `member` (le nom de la table doit avoir été déclaré dans le `RDB_node` du premier élément), à la colonne `name` et, de surcroît, n'apparaîtront que les noms des joueurs ayant un nombre de victoire strictement supérieur à 10.

La décomposition :

Il n'existe pas beaucoup de différence entre un fichier DAD de type RDB_node mapping pour la composition, d'un fichier DAD pour la décomposition. La correspondance entre un élément du document XML et sa représentation dans la base relationnelle étant effectuée de la même manière.
Parmi les quelques différences, nous pouvons signaler :

Premièrement, l'appel à la procédure stockée se fait différemment, il faut ici préciser le fichier XML qui sera éclaté dans les tables. La commande sui-

vante se connecte à la base de données `<db_name>`, prend le fichier XML `<path_to_XML_file>` comme fichier d'input et grâce aux règles de mapping du fichier DAD spécifié par `<path_to_DAD_file>` récupère les données et les insère dans les tables correspondantes :

```
dxxshrd <db_name> <path_to_DAD_file> <path_to_XML_file>
```

Deuxièmement, dans le `RDB_node` « particulier » du premier `element_node`, il faut spécifier une clé primaire pour chaque table. Pour ce faire, il suffit d'ajouter un attribut à chaque élément table du `RDB_node`.
L'exemple suivant définit une clé primaire pour chaque table composant la Collection :

```
<RDB_node>
  <table name="order_tab" key="order_key "/>
  <table name="part_tab" key="part_key "/>
  <table name="ship_tab" key="date"/>
  <condition>
    order_tab.order_key=part_tab.part_key
    AND part_tab.part_key=ship_tab.date
  </condition>
</RDB_node> ...
```

Troisièmement, l'ajout d'un attribut `type` dans l'élément column d'un `RDB_node`. Ce nouvel attribut permet tout simplement de spécifier le type de la colonne dans laquelle la donnée doit attérrir. Cela permet de s'assurer que le type de donnée de destination est bien identique à celui spécifié dans cet attribut.

3.3 Le standard SQL/XML dans DB2

SQL/XML est un standard ANSI et ISO qui, grosso modo, permet de combiner le XML et le SQL pour manipuler les données stockées dans la

base relationnelle. Plus précisément, il s'agit de fonctions supplémentaires et spécifiques à la manipulation de document XML qui viennent se greffer au langage de requête standard SQL.

SQL/XML permet :

- de stocker un document XML dans une base relationnelle
- d'interroger ces documents via XPath et XQuery
- de publier les données relationnelles en un document XML

DB2 supporte cette technologie en incluant, dans IBM DB2 UDB, des fonctions SQL/XML « built-in ». SQL/XML fournit une certaine flexibilité au développeur lui offrant une palette de fonctions qui lui permettent de construire des fragments XML [7].
Voici les fonctions SQL/XML les plus significatives :

XMLELEMENT :
Construit un nœud XML de type élément, le premier paramètre étant le nom de l'élément. Imbrications possible.
XMLATTRIBUTES :
Construit un ou plusieurs nœuds de types attributs, cette fonction est toujours incluse dans la fonction XMLELEMENT à laquelle elle se rapporte.
XMLAGG :
Agrège des valeurs XML comme une suite d'éléments XML.
XML2CLOB :
effectue la conversion d'un résultat XML en un CLOB. C'est par ce biais que l'application effectue le rendu du résultat.

[7] A noter que les fonctions incluses dans DB2 UDB permettent uniquement la composition de documents XML à partir de données relationnelles.

L'exemple suivant [2] utilise les quatre fonctions que nous venons de décrire :

```
SELECT XML2CLOB (XMLELEMENT(
  NAME "Department",
  XMLATTRIBUTES ( e.dept AS "name" ),
  XMLAGG ( XMLELEMENT ( NAME "emp", e.lname)
  ORDER BY e.lname)
  ) )
AS "dept_list"
FROM employees e
GROUP BY dept ;
```

Le résultat de cette requête est placé dans une colonne de type CLOB :

```
dept_list
  <Department name="Accounting">
     <emp>SMITH</emp>
     <emp>Yates</emp>
  </Department>
  <Department name="Shipping">
   <emp>Martin</emp>
   <emp>Oppenheimer</emp>
  </Department>
```

Nous pouvons remarquer qu'il est possible de faire beaucoup de choses intéressantes via cette technologie, elle offre une certaines souplesse en terme de composition de fragments XML à partir de données relationnelles grâce aux fonctions SQL/XML disponibles. Cependant, nous pouvons noter l'inconvénient qui consiste à imbriquer les fonctions dans la requête SQL, ce qui conduit très vite à des requêtes illisibles, dans lesquelles on se perd facilement.

Chapitre 4

Manipuler des documents XML avec Oracle 9i

4.1 Introduction

Nous allons nous intéresser dans ce chapitre aux différents outils offerts par Oracle pour stocker et récupérer des données XML, nous allons étudier les quelques outils qui sont offerts et comment les utiliser. Du point de vue du stockage, le SGBD se veut pouvoir stocker aussi bien les données relationnelles que les données XML en natif. Du point de vue des requêtes, autant les requêtes SQL que les requêtes XML (Xquery, XPath...) doivent être supportées, et cela indépendamment de la façon dont sont stockées les données. En d'autres termes, la structure utilisée pour stocker les données doit rester transparente par rapport aux différents langages de requêtes utilisés pour accéder à ces données. C'est ce que l'on appelle la dualité XML-SQL. Ce qui veut dire que le SQL doit pouvoir être utilisé pour effectuer des requêtes sur une vue relationnelle d'un document XML, et inversement, des requêtes XML doivent pouvoir être effectuées aussi bien sur des documents XML natifs que sur des vues XML de données relationnelles. Oracle tente de répondre à ces besoins. Les solutions adoptées par Oracle permettent de s'assurer que des requêtes formulées en XML seront efficaces à la fois sur des documents XML purs ainsi que sur les représentations XML des données relationnelles.

Il existe deux principales approches pour le stockage de documents XML sous Oracle :

1. Stocker des documents XML bruts dans un LOB (*Large Objects*).
2. Décomposer les documents XML et stocker les fragments dans plusieurs tables.

La première approche a pour avantage de ne tenir compte d'aucune restriction sur le type de document (simple texte, XML, DTD, XSD, etc.) que l'on insère dans le champ de type LOB puisque le document y est stocké tel quel. L'inconvénient, non négligeable, est qu'il n'existe pas actuellement de moyen efficace pour effectuer des requêtes SQL sur ce type d'objet. Un autre inconvénient est le fait que l'on ne puisse pas effectuer de modification sur une partie spécifique du document. L'entièreté du document doit être remplacée si l'on veut y apporter une quelconque modification. La deuxième approche quant à elle permet d'utiliser la puissance du langage SQL et de ce fait offrir une meilleure performance au niveau des requêtes. De surcroît, cette approche permet d'effectuer facilement des modifications puisque les données sont stockées dans des colonnes de tables relationnelles. Ce dernier point est très important puisque si les données XML sont stockées dans les tables relationnelles, alors, nous pourrons utiliser les outils et les technologies relationnelles avec un rendement maximal. Le type de données CLOB [1] est utilisé par Oracle pour stocker des documents XML en entier. Dans ce cas spécifique, la récupération de données via des recherches associatives a fait et fait d'ailleurs encore l'objet de nombreuses recherches. Oracle tente de résoudre ce problème grâce à Oracle Text, un outil de recherche textuelle. Ce dernier contient plusieurs fonctionnalités avancées pour la gestion de textes, différents services pour la gestion de documents et des outils spécifiques pour le format XML. En outre, Oracle 9i introduit un nouveau type de données, *XMLType* destiné à recevoir un document XML natif [34]. Les méthodes attachées à XMLType facilitent la manipulation des documents XML bruts. XSU [2] de Oracle permet de stocker des documents XML de façon ventilée sur plusieurs tables. En effet, XSU permet de réaliser les opérations suivantes :

[1] Character Large Object
[2] XML SQL Utility

- XSU permet de créer des documents XML à partir de données récupérées dans la base de données.
- XSU permet d'extraire des données à partir d'un document XML et, se basant sur certaines règles de mapping, éclate le document et insère les données dans les colonnes des tables ou vues spécifiées.
- XSU permet d'extraire des données à partir d'un document XML et, à partir de ces données, permet d'effectuer des mises à jour ou des suppressions dans certaines tables ou vues.
- Il existe des APIs XSU pour java et pour le langage propriétaire PL/SQL.

4.2 Stocker un document XML dans un CLOB

Qu'est ce qu'un CLOB ? Le type de données CLOB est un type de données qui permet de stocker d'importants volumes de données de type "chaîne de caractères". La taille maximale d'un CLOB est de 4Go. Il faut savoir que les CLOB conservent les mêmes règles transactionnelles que les types de données standard tels que VARCHAR, NUMBER, etc.

4.2.1 Utiliser CLOB et OracleText

Sous Oracle, la façon la plus simple pour manipuler un document XML est d'utiliser des tables relationnelles composées d'une ou plusieurs colonnes CLOB. Cependant, dans ce cas, le stockage et la récupération de documents ne peuvent se faire que pour l'entièreté du document. En effet, la mise à jour d'une partie de documents stockés dans un CLOB fait l'objet de recherches intensives et aucune solution concrète n'a encore réellement vu le jour. Sous Oracle, il n'existe pas de différences fondamentales entre un STRING et un CLOB mis à part le fait que le type de données CLOB est un type de données qui va permettre de stocker d'importants volumes de données. La taille maximale d'un CLOB étant de 4Go. A noter que, autant le type STRING que le type CLOB offrent des fonctionnalités de recherche, comme, par exemple, des fonctions de comparaison ou de recherche de sous-chaînes. Ces fonctionnalités constituent un bagage bien trop léger que pour pouvoir prétendre effectuer

des recherches avancées. L'outil Oracle Text répond à cette lacune en offrant des fonctionnalités avancées pour la recherche textuelle dans un document XML (stocké dans un CLOB, par exemple). Voici quelques exemples de fonctions de recherches accessibles avec OracleText pour les textes :

- Recherche exacte : contient un ou plusieurs mots
- Position de mots : phrases complètes, positionnement dans les paragraphes, proximité de certains mots etc.
- Recherche inexacte : ressemblance phonétique, etc.
- Recherche intelligente : thésaurus, sujet, etc.
- Combinaison de ces recherches *(AND, OR, NOT, etc.)*

4.2.2 OracleText pour XML

La fonction la plus importante de OracleText est sans aucun doute la fonction `CONTAINS()`. Elle peut être utilisée directement dans une requête SQL. Prenons pour exemple la table suivante :
`TABLE (id INTEGER, text_xml CLOB);`
Alors, la requête SQL suivante recherche la chaîne de caractère `"XML_BOOK"` dans la colonne `text_xml` de type CLOB et sélectionne l'id correspondant si `"XML_BOOK"` a été trouvé au moins une fois dans l'entièreté du document.

```
SELECT id
FROM TABLE
WHERE CONTAINS (text_xml, "XML_BOOK") > 0;
```

En fait, la valeur de retour de la fonction `CONTAINS()` est un entier compris entre 0 et 100, il reflète la pertinence de la recherche dans la mesure où cet entier est calculé par une formule complexe basée sur le nombre d'occurrences effectivement trouvées. Si 0 est la valeur de retour alors cela signifie que le mot n'apparaît pas une seule fois dans le document, tandis que si la valeur de retour est 100 cela signifie que seul ce mot a été trouvé dans le document. C'est pourquoi il suffit d'écrire `CONTAINS(col,"x")> 0` pour savoir si la chaîne `"x"` apparaît au moins une fois dans une colonne CLOB nommée col (cf. l'exemple ci-dessus). Il existe un certain nombre de fonctionnalités

spécifiques aux documents XML. Par exemple, les clauses `WITHIN, INPATH` et `HASPATH` permettent d'effectuer une recherche dans un document à l'aide des expressions XPath.

- L'opérateur `WITHIN` permet de rechercher un terme dans une partie[3] du document plutôt que dans l'entièreté de celui-ci. Par exemple : `CONTAINS (text_xml, 'term WITHIN tag ')` et `CONTAINS (text_xml, 'term WITHIN att@tag ')` correspondent respectivement à la recherche du terme `term` dans le contenu de la balise spécifique `<tag>...</tag>` et la recherche du terme `term` dans la valeur de l'attribut `att` de la balise `tag <tag att= "..."/>`.
- L'opérateur `HASPATH` permet d'effectuer deux types de recherches différentes. La première, `'HASPATH(xpath)'`, permet de vérifier si le chemin déterminé par l'expression Xpath existe dans le document. La seconde, `'HASPATH(xpath= "valeur")'`, permet de vérifier si au moins un élément sélectionné via l'expression Xpath a pour contenu la valeur `valeur`.
- L'opérateur `INPATH` est de la forme `'term INPATH (xpath)'`. Le document doit posséder un chemin déterminé par l'expression XPath et le terme `term` doit y apparaître. Si le terme `term` constitue une sous-chaîne d'une valeur trouvée dans un élément matchant avec l'expression Xpath, alors la fonction renvoi `true`. Imaginons que nous avons dans le document XML :

```
<Name>
<Lastname> blablatermblabla </Lastname>
</Name>
```

L'expression `'term INPATH(/Name/Lastname)'` sera acceptée, car term constitue une sous-chaîne dans la balise `<Lastname>`.

L'expression `'HASPATH(/Name/Lastname = "term" )'` quant à elle ne sera pas acceptée car on ne retrouve pas **exactement** la chaîne `"term"` dans le contenu de la balise `<Lastname>`.

[3] Ces parties peuvent être le contenu d'éléments XML ou la valeur d'attributs

Inutile de s'attarder d'avantage sur les possibilités offertes pas OracleText, le but étant ici d'exposer au lecteur les principes de bases de cet outil. Pour de plus amples informations, se reporter à la page consacrée à OracleText sur le site d'Oracle [8].

4.3 XMLType : le type de données XML

`XMLType` est un nouveau type de données introduit dans la version 9i d'Oracle et qui offre l'avantage d'être idéalement adapté aux contraintes apportées par les documents XML.

4.3.1 Qu'est ce que XMLType ?

Il s'agit d'un type de données qui facilite la manipulation des documents XML natifs stockés dans la base.

- XMLType permet de représenter un document XML comme une instance (de XMLType) en SQL.
- XMLType possède des méthodes destinées à faciliter le traitement des documents XML [4].
- XMLType met à la disposition du développeur un ensemble d'APIs pour les langages PL/SQL et Java.
- XMLType peut être utilisé comme paramètre, valeur de retour et variables des procédures stockées du langage propriétaire PL/SQL.

Voici quelques fonctions membres à `XMLType` permettant de manipuler des documents XML :

- `STATIC createXML (xml VARCHAR | CLOB)` prend en paramètre un document XML sous forme de chaîne de caractères ou en tant que CLOB et le transforme en un objet `XMLType`.
- `MEMBER getClobVal () RETURN CLOB` retourne le contenu de l'objet `XMLType` en objet CLOB.

[4] L'utilisation de ces méthodes permet d'extraire, de créer et d'indexer des données XML qui sont stockées dans la base de données.

- `MEMBER getStringVal () RETURN NUMBER` retourne le contenu de l'objet `XMLType` en String.
- `MEMBER getNumberVal () RETURN NUMBER` retourne la valeur numérique d'un élément.
- `MEMBER existsNode(VARCHAR xpath) RETURN NUMBER` cette fonction sert à vérifier l'existence d'un élément via le chemin XPath passé en paramètre.
- `MEMBER extract(VARCHAR xpath) RETRURN XMLType` permet d'extraire une partie d'un document stocké dans un objet `XMLType` via le chemin XPath passé en paramètre.

Grâce à `XMLType` le développeur peut effectuer des opérations SQL sur un support XML ainsi que des opérations XML sur un support relationnel (respect du standard SQL/XML). Il peut effectivement utiliser les méthodes associées à `XMLType` directement dans ses requêtes SQL. Par exemple, définissons la table `XMLTypeTab` utilisant un type de donnée `XMLType` plutôt que CLOB :

```
XMLTypeTab(id INTEGER, txt XMLTYPE)
```

L'utilisation de la fonction membre `createXML` dans une requête SQL se fait alors de la manière suivante :

```
INSERT INTO XMLTypeTab
VALUES (1, SYS.XMLType.createXML
('<?xml version=''1.0''?> <person id=''1"> <firstname> John </firstname>
...
</person')
```

Notons que lors de l'utilisation de la méthode `createXML(...)` le parser vérifie si le document est bien formé ou non. Donc, si nous avons par exemple : ... , `sys.XMLType.createXML('<Oracle> <version>9i </Oracle>')` ...

Nous aurons le message d'erreur suivant :

```
ORA-31011 : XML parsing failed
Cause : XML parser returned an error while trying to parse the document.
Action : Check if the document to be parsed is valid.
```

Nous remarquons effectivement qu'il manque la balise fermante `</version>`. Si maintenant nous avons dans l'objet `xmltype` de la colonne `txt` le fichier XML bien formé suivant :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE person SYSTEM "C :  Program Files  IBM  SQLLIB  samples
montest  DTD  person.dtd">
<person id="1">
  <firstname>John</firstname>
  <lastname>Smith</lastname>
  <country>US</country>
  <phone>
    <type>office</type>
    <number>12345678</number>
  </phone>
  <phone>
    <type>home</type>
    <number>3456789</number>
  </phone>
</person>
```

Alors, la requête suivante renverra `"Smith"` comme résultat parce que dans le document, l'élément `person/phone` existe et le string `"US"` a été trouvé dans l'élément `person/country` :

```
SELECT t.txt.extract('person/lastname')
FROM XMLTypeTab t
WHERE t.txt.existsNode('person/phone')=1
AND t.txt.extract('person/country/text()').getStringVal() = "US"
```

Nous pouvons remarquer dans cet exemple le caractère dual du SQL/XML, c'est-à-dire l'intégration de requêtes XML (ici via des expressions XPath) dans une requête SQL.

4.3.2 Qu'est ce que XMLType apporte de plus ?

`XMLType` apporte de nombreux avantages, parmi ceux-ci, on peut citer :

- `XMLType` possède une API très souple pour le développement d'application, permettant au développeur de manipuler facilement le contenu d'un document XML grâce notamment à ses fonctions membres, l'indexation, la navigation dans le document via Xpath, etc.
- `XMLType` peut être utilisé avec des requêtes SQL en combinaison avec d'autres colonnes et types de données. Nous pouvons, par exemple, faire une requête sur un objet `XMLType` et utiliser le résultat pour effectuer une jointure avec d'autres colonnes.
- `XMLType` est optimisé pour ne pas effectuer le rendu du document dans sa représentation arborescente si cela ne s'avère pas nécessaire. A noter que, la transformation du document en arbre s'effectue toujours lors de l'utilisation des fonctions membres existsNode et extract puisque ces dernières utilisent XPath (celui-ci ne travaillant qu'avec une représentation arborescente). La structure interne est optimisée en une représentation arborescente quasi équivalente à DOM [5].

[5] Dans un souci d'amélioration des performances d'utilisation mémoire.

4.3.3 XMLType par la programmation

`XMLType` possède des APIs pour java et PL/SQL. Nous allons introduire la bibliothèque permettant d'utiliser les fonctions d'`XMLType` uniquement pour le langage java. Le lecteur intéressé pourra trouver des informations sur la façon d'utiliser `XMLType` via l'API PL/SQL dans la documentation d'Oracle. Les fonctionnalités restent cependant fort similaires entre java et PL/SQL.

XMLType et java

L'archive `xdb_g.jar` contient la classe java `oracle.xdb.XMLType` qui correspond à l'objet SQL `XMLType`. En effet, cette classe possède les même fonctionnalités et toutes les méthodes associées sont aussi accessibles (par exemple `java.lang.String getStringVal(), oracle.sql.CLOB getCLOB())` . Cependant, les fonctions `createXML( Connection, oracle.sql.clob)` et `createXML (connection, String)` nécessitent toutes deux une connexion JDBC comme premier paramètre.

Exemple [10] :

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver()) ;
 Connection conn =
(DriverManager.getConnection"jdbc :oracle :oci8 :@","scott", "tiger") ;
 OraclePreparedStatement stmt =
 (OraclePreparedStatement) conn.prepareStatement(
"select e.poDoc from po_xml_tab e") ;
 ResultSet rset = stmt.executeQuery() ;
 OracleResultSet orset = (OracleResultSet) rset ;
 // get the XMLType
 XMLType poxml = XMLType(orset.getOPAQUE(1)) ;
 // get the XML as a string...
 String poString = poxml.getStringVal() ;
```

Dans cet exemple, une fois que la connexion JDBC est créée, l'exécution de la requête peut s'effectuer via la fonction `prepareStatement(String)` dont

la valeur de retour est un objet de type `OraclePreparedStatement`. La classe `OracleResultSet` est une sous-classe de `ResultSet`, elle bénéfice donc des méthodes de celle-ci. La fonction `getOpaque(int)` de la classe `OracleResultSet` permet de lire un objet `XMLType` venant d'une base de données. Nous voyons qu'une fois que l'objet `XMLType` est créé, il est facile d'utiliser les méthodes qui lui sont associées.

4.4 XML SQL Utility (XSU)

Nous allons découvrir dans ce chapitre l'outil de *shredding*[6] XSU spécifique à Oracle. Cet outil est fourni en standard à partir d'Oracle 8i (version 8.1.7 et plus) ainsi que sous Oracle 9i.

4.4.1 Tour d'horizon de XSU

D'une façon générale XSU est un outil dont le but est de faciliter les échanges entre le monde XML et les bases de données. Il permet de composer un document XML à partir de données stockées dans des tables ou des vues de bases de données objets-relationnelles, ainsi que d'éclater un document XML et d'insérer les données dans plusieurs tables ou vues de bases de données objets-relationnelles. Pour se connecter aux données, XSU utilise des connexions JDBC. Bien que cet utilitaire puisse fonctionner avec n'importe quel driver JDBC, ceux fournis par Oracle sont optimisés. Les fonctionnalités de XML SQL Utility sont accessibles de trois manières différentes :

- Via l'API Java.
- Via l'API PL/SQL.
- En lignes de commandes.

[6]shredding : "découper" le document et mettre les différents morceaux dans des tables relationnelles, ou inversement, recomposer un document à partir de données éparpillées dans des tables.

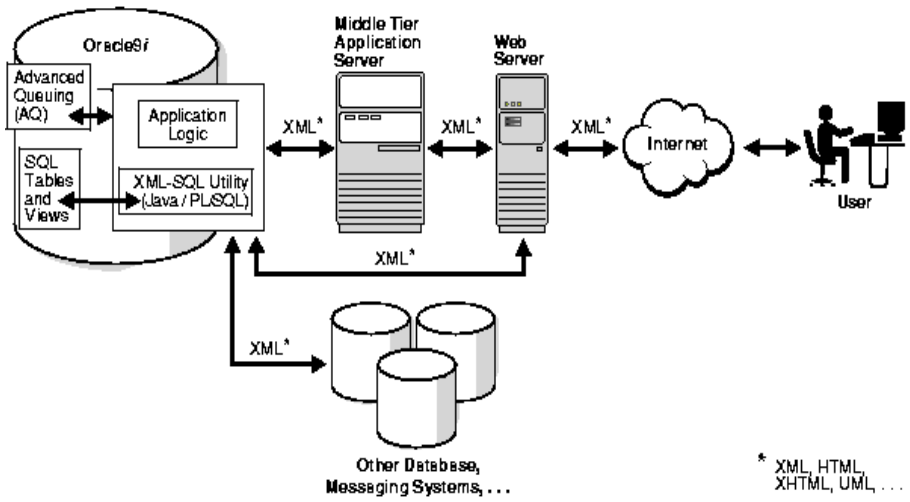


FIG. 4.1 – XSU au sein d'un système d'information utilisant une base de données Oracle [8]

4.4.2 Qu'est ce que XSU permet de réaliser ?

XSU permet de réaliser les tâches suivantes :

- La génération d'un document XML à partir d'une requête SQL. XSU reconnaît tous les types de données supportés pas Oracle 9i.
- La génération dynamique de DTDs et de Schemas.
- Durant la génération du document XML, XSU permet de réaliser quelques transformations simples, comme par exemple la modification du nom d'un élément. Pour des transformations plus complexes, l'utilisation d'une feuille de transformations XSLT sera alors préférée.
- Permet de créer des attributs durant la phase de génération. Cela offre la possibilité de spécifier qu'une colonne ou un groupe de colonnes doivent être mappé en un attribut XML plutôt qu'en un élément.
- La génération d'un document XML dans sa représentation DOM ou sous forme de String.
- L'insertion de données XML dans la base de données objet-relationnelle. XSU permet également de mettre à jour, ou de supprimer des enregistrements d'une base objet-relationnelle à partir d'un document XML.

- Simplifie la génération de documents XML comportant une imbrication complexe.
- A noter aussi que XSU supporte le type de données `XMLType`. L'utilisation du combo XSU, `XMLType` peut s'avérer utile si, par exemple, nous disposons d'une colonne de type `XMLType` [19].

4.4.3 Le mapping

XSU se base sur un mapping des structures relationnelles ou objets en XML.

Le mapping SQL-XML par défaut

Dans le cas d'un schéma relationnel classique, si l'on exécute une requête SQL, le document XML qui sera produit aura la structure suivante :

- L'entièreté du résultat est encapsulé dans un élément `<ROWSET>...</ROWSET>`.
- Chaque tuple du résultat de la requête est encapsulé dans un élément `<ROW>...</ROW>`.
- Chaque valeurs d'un tuple, résultant de la requête SQL est encapsulée dans un élément `<Attribut>...</Attribut>` où `Attribut` correspond au nom de la colonne contenant la valeur courante.

En considérant la table suivante :

```
CREATE TABLE member
(
  id_member INTEGER PRIMARY KEY NOT NULL,
  id_club INTEGER,
  name VARCHAR(20),
  surname VARCHAR(20),
  classification VARCHAR(2),
  victory INTEGER
);
```

Un exemple de document XML généré grâce à XSU et résultant de la requête SQL suivante, `"select * from member"`, sera alors :

```
<?xml version="1.0"?>
<ROWSET>
  <ROW num="1">
    <id_member> 15 </id_member>
    <id_club> 1 </id_club>
    <name> Accou </name>
    <surname> Frederic </surname>
    <classification> D0 </classification>
    <victory> 12 </victory>
  </ROW>
  <ROW num="2">
  ...
  </ROW>
</ROWSET>
```

Si des alias sont utilisés dans la requête SQL, ces derniers seront alors utilisés pour dénommer les éléments plutôt que le nom de la colonne. Ceci peut être utile pour donner un nom aux éléments correspondant à des expressions telles que `COUNT(), MAX(), MIN(), etc.` , qui sans cette technique ne seraient pas nommable. Nous allons voir dans le prochain chapitre qu'il est possible de modifier les noms des balises ROW et ROWSET et de l'attribut num de façon à personnaliser un minimum notre document XML de sortie. Il est important de signaler que ce type de mapping ne permet pas de modifier l'arbre XML, en d'autres termes, si nous travaillons avec des tables relationnelles, cette technique ne permet pas de créer de nouveaux niveaux dans la structure arborescente. Cela ne constitue pas un problème lorsque nous effectuons une requête sur une seule table relationnelle puisque une simple table relationnelle ne contient qu'un seul niveau hiérarchique. Si maintenant nous effectuons une requête sur deux tables ayant une relation 1..N les choses sont différentes. En effet, l'idéal serait de regrouper dans une collection, tous les tuples du côté N dans une collection du côté 1. Chose impossible avec le

mapping simple que nous venons de décrire. Pour effectuer des changements plus complexes dans la structure du document XML il nous faudra utiliser les feuilles de style XSL et son langage de transformation associé XSLT. (voir exemple en annexe)

Le mapping SQL-XML dans le cas d'un schema objet-relationnel

Intéressons-nous maintenant au mapping dans le cas d'un shema objet-relationnel. En considérant le type scalaire `AddressType`, et la table `Member`, tout deux définis ci-dessous :

```
CREATE TYPE AddressType AS OBJECT
(
  STREET VARCHAR(20),
  NUM NUMBER,
  CITY VARCHAR(20),
  ZIP VARCHAR(4),
  STATE CHAR(20)
);


CREATE TABLE member
(
  id_member INTEGER PRIMARY KEY NOT NULL,
  id_club INTEGER,
  name VARCHAR(20),
  surname VARCHAR(20),
  classification VARCHAR(2),
  victory INTEGER,
  address AddressType
);
```

Alors, un exemple de document XML résultant de la requête SQL suivante "select * from member" sera :

```
<? xml version="1.0" ?>
<ROWSET>
  <ROW num="1">
    <id_member> 15 </id_member>
    <id_club> 1 </id_club>
    <name> Accou </name>
    <surname> Frederic </surname>
    <classification> D0 </classification>
    <victory> 12 </victory>
    <address>
      <STREET> Rue haute </STREET>
      <NUM> 78 </NUM>
      <CITY> Ligny </CITY>
      <ZIP> 5140 </ZIP>
      <STATE> Belgique </STATE>
    </address>
  </ROW>
  <ROW num="2">
  ...
  </ROW>
</ROWSET>
```

Nous pouvons remarquer que l'élément `<address>` comporte cinq sous-éléments, à savoir autant de sous-éléments qu'il y a d'attributs dans le type complexe `AddressType`. Il correspond effectivement à la colonne address de la table `member` qui est du type complexe `AdressType`. Si un de ces sous-éléments correspond à un type complexe, il aura à son tour des sous-éléments. Nous venons de voir que le mapping dans le cas d'un schema objet-relationnel permet d'obtenir simplement des données imbriquées, sans passer pas une transformation XSLT, bien que celle-ci reste possible pour une transformation plus spécialisée.

4.4.4 XSU par la programmation

XSU est accessible par la programmation via l'API Java et l'API PL/SQL. L'API Java propose la classe `OracleXMLQuery` pour la composition d'un document XML à partir de données récupérées dans la base relationnelle. Elle propose la classe `OracleXMLSave` pour le stockage de données XML dans la base. Quant à l'API PL/SQL, elle met à la disposition du développeur les deux packages suivants, `DBMS_XMLQuery` et `DBMS_XMLSave`, possédant les mêmes caractéristiques que `OracleXMLQuery` et `OracleXMLSave` respectivement. A nouveau, nous allons nous intéresser de plus prêt à l'API Java. Les fonctionnalités en PL/SQL sont intimement liées à celles proposées par Java.

OracleXMLQuery pour la composition de documents XML

Pour pouvoir créer un document XML à partir de données recueilles via une requête SQL, il faut d'abord définir la connexion avec la base, créer une instance `OracleXMLQuery`[7] et transformer le résultat de la requête en quelque chose de lisible. C'est à dire, transformer le résultat, soit en String, soit en un objet DOM.

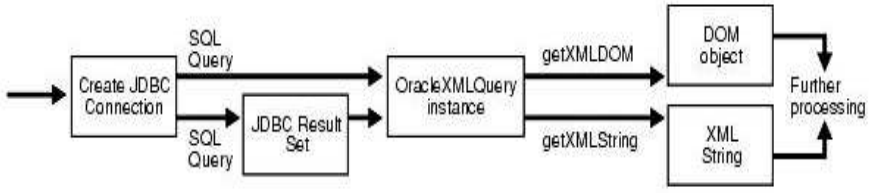


FIG. 4.2 – Les différentes étapes possibles lors de la génération d'un document XML par `OracleXMLQuery` [8].

[7] `OracleXMLQuery` possède deux constructeurs prenant en paramètre la requête SQL soit en tant que String, soit en tant que JDBC ResultSet.

Voici un exemple de code Java illustrant ces étapes :

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver()) ;
//création de la connexion
Connection conn = DriverManager.getConnection
("jdbc :oracle :oci8 :@'',"gaby","gaby") ;
//création de l'instance OracleXMLQuery, avec requête de type String
OracleXMLQuery qry = new OracleXMLQuery ( conn, "select * from EMPLOYEES") ;
//récupération d'un String XML
String str = qry.getXMLString() ;

System.out.println (str) ;

qry.close() ;
```

Dans cet exemple, le résultat du document XML généré a été mis sous forme de String. Une autre possibilité aurait été de convertir le résultat en un objet DOM. Pour ce faire, il aurait fallu remplacer la ligne de code

```
String str = qry.getXMLString() ;
```

par la suivante, `XMLDocument domDoc = (XMLDocument)qry.getXMLDOM() ;`

Comment paginer le résultat de la requête

Il existe des méthodes qui permettent de décomposer l'ensemble des résultats de la requête SQL en γ sous-ensembles de π tuples et de répartir ceux-ci dans plusieurs documents XML. Prenons l'exemple où nous avons n tuples en résultats de notre requête SQL et que nous voulons créer des documents XML ne contenant que π tuples maximum (donc $\frac{n}{\pi} = \gamma$ documents de π tuples chacun). Les méthodes `setMaxRows(int max)` et `setSkipRows(int y)` sont prévues à cet effet. La première permet de ne générer maximum que `max` tuples, tandis que la deuxième permet de passer `y` tuples avant de générer le document.

Un algorithme en pseudo code serait :

```
i=0 ;
While (i < n/π)
{
  setSkipRows(i* π) ;
  setMaxRows(π) ;
  generateXML( ) ;
  ++i ;
}
```

Modifications des balises par défaut durant la phase de génération ou mapping personnalisé

Les fonctionnalités suivantes permettent de modifier le nom des balises qui sont attribuées par défaut par XSU :

`void setRowTag(String tagname)` : renomme la balise ROW.
`void setRowsetTag(String tagname)` : renomme la balise ROWSET.
`void setRowIdAttrName(String tagname)` : renomme l'attribut NUM de la balise ROW.
`void useUpperCaseTagNames()` : met les balise en majuscules.
`void useLowerCaseTagNames()` : met les balise en minuscules.

Il est important de signaler que ce personnalisation du mapping SQL vers XML ne permet pas la modification de l'arbre XML en tant que tel, c'est à dire qu'on ne peut pas créer de nouveau niveaux avec cette technique de mapping. Ce problème n'est pas vraiment gênant lorsque l'on effectue une requête sur une seule table relationnelle car nous avons forcément un seul niveau hiérarchique puisque les tables relationnelles sont plates. Par contre, lorsque nous avons une relation de type 1..n entre deux tables, nous préférerons regrouper dans une collection tous les tuples du côté n dans une

collection du côté 1. Or, cela n'est pas possible dans le mapping que nous venons de décrire [8]. Il faut alors avoir recours à une feuille de style xsl. Elle constitue un moyen efficace et souple pour effectuer des transformations personnalisées sur la structure du document XML à générer. Ces transformations étant effectuées par un processeur XSLT. Dans le code, l'association d'une feuille de style à un objet OracleXMLQuery se fait via les fonctions :

```
void setXSLT(String stylesheet)
void setXSLT(Reader stylesheet)
```

L'objet passé en paramètre peut être passé à convenance sous forme de String ou de Reader.

OracleXMLSave

La classe OracleXMLSave permet d'effectuer des insertions, des mises à jour et des suppressions d'enregistrements dans la base de données à partir de données récupérées d'un document XML. Cependant, sa seule utilisation ne permet pas de pouvoir insérer dans la base de données certains documents XML un peu trop complexes. Il faut en effet créer une structure Oracle (objet, vue) correspondant sur mesure à la structure de ces documents. Même si le chemin à parcourir pour aller des données XML vers la base de données n'est que "l'inverse" de celui qu'on emprunte pour engendrer du XML à partir de la base, il existe tout de même une différence de taille entre les deux : il n'existe pas de "mapping" entre les balises XML et les champs de la base de données. D'où la nécessité de créer une "structure d'accueil" sur mesure pour les documents XML.

[8] Nous ne parlons pas ici du mapping objet-relationnel

Prenons le document XML suivant et nommons le "club.xml" :

```
<CLUB>
  <CLUBID>N153</CLUBID>
  <CLUBNAME>Palette Club Ligny</CLUBNAME>
  <PLAYERLIST>
    <PLAYERLIST_ITEM>
      <NAME>Accou</NAME>
      <SURNAME>FREDERIC</SURNAME>
      <CLASSIFICATION>D0</CLASSIFICATION>
      <VICTORY>12</VICTORY>
    </PLAYERLIST_ITEM>
    <!- OTHER PLAYERS... ->
  </PLAYERLIST>
</CLUB>
```

Si nous désirons stocker les données contenues dans ce document vers la base, il faut prévoir la ou les tables Oracle adaptées pour les réceptionner. Pour faciliter cette opération nous allons faire correspondre certaines parties de la structure XML à des types que nous allons définir :

```
CREATE OR REPLACE TYPE PLAYER_TYPE AS OBJECT (
  name     VARCHAR2(25),
  surname     VARCHAR2(25),
  classification VARCHAR2(2),
  victory NUMBER(3)
);

CREATE TYPE PLAYER_COLL_TYPE AS TABLE OF PLAYER_TYPE;

CREATE TYPE CLUB_TYPE AS OBJECT (
```

```
        clubid      VARCHAR2(4),
        clubname    VARCHAR2(25),
        players     PLAYER_COLL_TYPE
);
```

`PLAYER_TYPE` recueillera les informations liées à un `<PLAYERLIST_ITEM>`.
`PLAYER_COLL_TYPE` décrira une collection de joueurs, correspond à `<PLAYERLIST>`.
`CLUB_TYPE` définit un club et correspond à `<CLUB>`.

Maintenant que les types sont définis, la table peut être créée :

```
CREATE TABLE club OF club_type
NESTED TABLE players STORE as nested_players ;
```

Nous savons donc que `OracleXMLSave` permet d'effectuer, une insertion, une mise à jour ou une suppression d'enregistrements dans une base à partir d'un document XML, nous allons maintenant décrire brièvement le code à utiliser dans les trois cas. Voici quelques explications sur les méthodes qui seront utilisées dans les exemples suivants. Tout d'abord, le constructeur :

```
OracleXMLSave(java.sql.Connection conn, java.lang.String tab)
```

Permet de créer l'objet `OracleXMLSave`. Le paramètre conn correspond à la connexion JDBC qui aura été préalablement établie et tab est le nom de la table que nous voulons modifier. Ensuite, les trois fonctions principales :

```
int insertXML(org.w3c.dom.Document)
int updateXML(org.w3c.dom.Document)
int deleteXML(org.w3c.dom.Document)
```

Chacune d'elles renvoie un entier qui reflète le nombre de tuples ayant été modifiés dans la table.

Insertion :
Une fois que l'on est sur que la structure d'accueil est correctement définie,

l'utilisation des méthodes est assez simple. En effet, les trois lignes de codes suivantes sont suffisantes pour que la table club accueille le document spécifié par l'URL.

```
OracleXMLSave sav = new OracleXMLSave(conn, "club") ;
sav.insertXML(sav.getURL("url")) ;
sav.close() ;
```

La fonction `getURL (String)` permet d'importer un fichier situé à l'url spécifiée dans le paramètre. Cette fonction crée un objet URL, lequel est prêt à être utilisé par la fonction `insertXML`. Si il manque des éléments dans le fichier d'input XML, les enregistrements correspondant prendront la valeur NULL. Aussi, nous pouvons n'insérer que certains attributs dans la table. Ceci se fait grâce à la méthode `setUpdateColumnList(String)`. Plusieurs attributs peuvent être passés en paramètre sous forme d'un vecteur de String. Si par exemple nous voulons insérer uniquement le clubid et le clubname alors nous aurons :

```
OracleXMLSave sav = new OracleXMLSave(conn, "CLUB") ;
String[] list = new String[2] ;
list[0] = "clubid" ;
list[1] = "clubname" ;
sav.setUpdateColumnList(list) ;
sav.insertXML(sav.getURL("url")) ;
sav.close() ;
```

Si le paramètre url est le chemin correspondant à notre exemple "club.xml" alors cela revient à effectuer la requête SQL suivante :

```
INSERT INTO CLUB(clubid, clubname) VALUES ( "N153", "Palette Club Ligny")
```

Mise à jour :
La mise à jour n'est pas plus compliquée :

```
OracleXMLSave sav = new OracleXMLSave(conn, "CLUB") ;
sav.setUpdateColumnList("clubname")) ;
sav.setKeyColumnList("clubid") ;
sav.updateXML(sav.getURL("url")) ;
sav.close() ;
```

Nous utilisons ici une nouvelle méthode `setKeyColumnList(String)`, cette méthode est très importante dans la mesure où elle permet de spécifier une condition. Elle correspond à la clause SQL `WHERE`. Si le paramètre `url` est le chemin correspondant à notre exemple "club.xml" alors cela revient à effectuer la requête SQL suivante :

```
UPDATE CLUB SET clubname= "Palette Club Ligny" WHERE clubid= "N153"
```

Suppression :
Rien de bien étonnant pour la suppression, les lignes de code suivantes :

```
OracleXMLSave sav = new OracleXMLSave(conn, "CLUB") ;
sav.setKeyColumnList("clubid") ;
sav.deleteXML(sav.getURL("url")) ;
sav.close() ;
```

correspondent à la requête SQL :

```
DELETE FROM CLUB WHERE clubid= "N153"
```

Petites remarques sur la nouvelle version 10g. La version 10g est pour l'essentiel une mise à jour des possibilités offertes par la 9i. Notons aussi que si la version actuelle réalise ses interrogations XML en XPath, le langage XQuery devrait bientôt faire son apparition dans la version Oracle 10g release 2.

Chapitre 5

Conclusion

Nous avons pu remarquer lors de notre étude, que d'une façon générale, DB2 et Oracle fournissent les mêmes outils pour intéragir avec XML. Cependant, leur cuisine interne et leur approche pour y arriver est différente, ce qui a un impact sur la flexibilité des solutions. Nous allons essayer de mettre en surbrillance dans cette conclusion les points forts et les points faibles de ces deux SGBD. Tout d'abord, nous allons effectuer un bref récapitulatif des outils développés par chacun pour communiquer avec le monde XML, en vue d'offrir au lecteur une vision plus globale. Ensuite, nous essayerons de mettre en évidence les atouts et les lacunes de chacun d'entre eux.

5.1 Bref rappel sur DB2 v8.2

Chez IBM, tout passe par XML Extender, une extension qui vient se greffer à DB2 UDB et qui fournit une collection de procédures stockées permettant d'insérer des documents XML dans des colonnes (XML Column) de type VARCHAR, CLOB ou FILE via les UDTs (User Defined Types) XMLVARCHAR, XMLCLOB et XMLFILE. XML Extender permet aussi d'exploser des documents XML et de stocker les fragments dans des collections (XML Collection). XML Collection permet d'effectuer un mapping de données non-XML vers un document XML, les règles de mapping étant définies dans le fichier DAD. Les fichiers DAD (Data Access Definition) permettent de map-

per la structure du document XML avec une table relationnelle. IBM offre, par ailleurs, un petit outil graphique permettant d'administrer les DAD.
Il existe deux sortes de mapping, le SQL mapping et le RDB node mapping. Le SQL mapping permet d'effectuer une requête SQL et spécifie où les résultats doivent être placés. RDB node est, quant à lui, un mapping objet-relationnel et permet de transférer des données XML de et vers une base de données.
Pour arriver à ce résultat, IBM utilise toute une mécanique interne de « side tables », transparentes pour le développeur. Ces side tables permettent à un ou plusieurs éléments d'être indexés pour améliorer les temps de réponse lors des accès au document XML.
XML Extender permet aussi d'effectuer la validation d'un document XML basée sur une DTD et la transformation d'un document grâce à un processeur XSLT. DB2 offre la possibilité d'utiliser des fonctions SQL/XML. Ce standard, qui est en quelque sorte une extension du SQL, permet de créer simplement des fragments XML à partir de données relationnelles. Ces fonctions étant intégrées à la requête SQL.

5.2 Bref rappel sur Oracle 9i

Chez Oracle, l'outil XDB (XML Data Base) permet à la fois le stockage sur plusieurs tables et le stockage en natif d'un document XML. Il efface les frontières qui délimitent les données relationnelles et XML en offrant à l'utilisateur la possibilité de voir les données XML comme des données relationnelles et inversement. L'approche la plus simple pour stocker un document XML dans une base Oracle consiste à insérer le document entier dans une colonne de type CLOB (Character Large Object). Grâce à l'outil Oracle Text, des recherches textuelles peuvent être effectuées. Cependant les fonctionnalités de recherches restent rudimentaire et ne sont pas adaptées aux documents XML. Une approche plus intéressante consiste à utiliser le type de données natif XML, XMLType. Comme n'importe quel autre type d'objet, XMLType peut être utilisé comme type de données dans une table ou dans une vue. Cela signifie qu'une vue XML peut être créée sans se préoccuper

du fait de savoir s'il s'agit de données relationnelles ou XML. Fianlement, XSU (XML SQL Utility) est l'outil de shredding propre à Oracle. Il permet de créer des documents XML à partir de données récupérées dans la base de données. Il permet aussi d'insérer, de mettre à jour et de supprimer des enregistrements de la base à partir de documents XML. Les fonctionnalités XSU peuvent être accédées via une API Java, une API PL/SQL ou en ligne de commandes. XSU prévoit un mapping par défaut. Certaines fonctions permettent de changer les noms des éléments et des attributs qui sont donnés par défaut pour permettre de personnaliser un minimum le mapping. Pour une personnalisation plus complète et détaillée, le développeur passera par une transformation XSLT.

5.3 Avantage Oracle

5.3.1 Une architecture XML intégrée

Oracle, grâce à une intégration réussie de XML DB dans sa version 9i release 2 (toutes plates-formes confondues), permet l'utilisation de sa base à la fois comme une base de données relationnelles-objets et à la fois comme une base de données native XML. Pas besoin de middleware ou d'extension pour communiquer avec XML. Ce qui n'est pas le cas d'IBM. En effet, ce dernier à besoin d'être « étendu » par divers éléments pour supporter XML. D'où le nom, XML Extender. S'il on veut utiliser la recherche textuelle, il faudra aussi ajouter l'extension Text Extender. Ces deux composants logiciels étant développés séparément, il faut les installer explicitement et de surcroît, cela amène quelques fois à des inconsistances. De ce fait, le développeur qui veut construire une application XML avec DB2 devra effectuer l'ensemble des tâches propres à IBM suivantes :

Installer XML Extender. Opérer la jonction entre XML Extender et la base de données, mais aussi entre XML Extender et les tables spécifiques. Sans cela, l'utilisation de XML Extender n'est pas possible. Idem pour Text Extender. Créer un fichier de mapping DAD (propriétaire à IBM). Gérer les opérations de recherches textuelles (Text Extender) et de manipulation XML

(XML Extender) séparément.

5.3.2 Un type de données natif

Bien que DB2 permette le stockage de documents XML en natif (dans une colonne de type UDT XMLVARCHAR, XMLCLOB ou XMLFILE), il ne dispose pas à proprement parler d'un type de données natif XML tel que XMLType d'Oracle[1], disposant de ses propres méthodes pour accéder, extraire et requêter du XML via XPath. A noter que Oracle et DB2 sont en passe d'intégrer le langage de requête Xquery dans leur composant [2], ce dernier étant beaucoup plus souple que XPath. Cela permet de manipuler un document XML stocké dans la base relationnelle comme n'importe quel autre type de données via ses fonctions membres. Aussi, cette solution simplifie grandement la manipulation des documents XML.

Le seul souci avec le type natif d'Oracle est que un objet XMLType est stocké dans la base comme un CLOB et il nécessite la construction d'un arbre DOM en mémoire avant de pouvoir effectuer une quelconque manipulation sur un document [8]. Donc, que ce soit via les méthodes de XMLType ou via l'utilisation de l'outil Text Extender, Oracle et DB2 sont quasi équivalent en en terme de possibilités offertes pour la manipulation de données XML. Ils offrent tous deux de nombreuses possibilités de recherche sélective sur des données au format XML : extraction de valeurs textuelles ou numériques, recherche de balises, possibilité d'utiliser les fonctions de manipulation de textes : CONTAINS()... Oracle, grâce à son type natif possède l'avantage d'être plus intuitif et plus simple à utiliser.

[1] Attention, depuis sa nouvelle version DB2 v.9 nom de code « viper » (juillet 2006), IBM offre un type de données natif XML à la manière XMLType dont le nom est xml, plus besoin d'extension pour communiquer avec XML.

[2] Oracle dans sa version 10g release 2 et DB2 dans son projet Xperanto et dans sa nouvelle version 9 « viper ».

5.3.3 Au niveau des APIs

Oracle offre dans son XDK (XML Development Kit) de nombreuses possibilités en terme d'interface de programmation pour les langages suivants : Java, PL/SQL, C, C++. Plus particulièrement pour les languages Java et PL/SQL via l'API XSU. XDK fournit son propre parser pour Java, PL/SQL, C et C++, un processeur XML Schema, un générateur de classe XML, un processeur XSLT, etc. Une fois le document XML parsé en DOM, l'application peut y effectuer des manipulations à sa guise. Nous avons vu que les classes Java OracleXMLQuery et OracleXMLSave offrent de nombreuses solutions pour interagir facilement avec le monde XML dans une application cliente écrite en Java. DB2 propose quant à lui une fine gamme de possibilités en terme d'interaction entre les langages de programmation et XML. XML Extender repose essentiellement sur des procédures stockées, si nous voulons interagir avec du XML dans une application cela se fera dès lors par un appel à la procédure stockée correspondante au sein même de la requête SQL.

5.3.4 La gestion des XML Schemas

Nous l'avons vu, les deux systèmes permettent de valider un document XML par une DTD. Mais qu'en est-il des XML Schemas ? Jusqu'à sa version 8.2, DB2 ne permettait pas de gérer la validation d'un document XML par un schéma XML, une lacune qui n'existe plus depuis la toute nouvelle version 9, nom de code « viper ». Le module Oracle XML DB offre depuis sa version 9i release 2, la possibilité d'enregistrer un schéma conforme à la recommandation XML Schema du W3C dans la base de données. Une fois le schéma enregistré dans la base, un ensemble de types d'objets et de tables sont créés dans le but de recevoir une instance du document XML conforme au schéma spécifié. Cette instance du document XML sera ensuite éclatée et répartie dans les différents objets et tables préalablement construits.

D'une façon générale, nous constatons qu'il y a définitivement plus d'avantages à attribuer à Oracle qu'à IBM. Cette constatation est consolidée par une étude menée au sein de la société européenne Butler Group[3] [21], *« IBM DB2 comes out disappointingly... overall, it does not support quite so many features as Oracle's 9i ... Before commencing this report, given the expertise of the people involved, an expectation was formed as to how the scoring would come out ... The one possibility that was dreaded was that there would be a tie for first place between Oracle and IBM. Not only was this not the case, but the margin of difference was so great, which came as a considerable surprise... »*.

[3]Butler Group a pour rôle de donner une vision professionnelle et neutre sur un produit spécifique.

Bibliographie

[1] http ://exist-db.org/.

[2] http ://publib.boulder.ibm.com/infocenter/dzichelp/v2r2/
index.jsp ?topic=/com.ibm.db2.doc.sqlref/bjnrmstr198.htm.

[3] http ://www-306.ibm.com/software/data/db2/.

[4] http ://www-306.ibm.com/software/data/db2/extenders/xmlext/.

[5] http ://www2.iicm.edu/cguetl/education/projects/rjernej/seminarxml.htm.

[6] http ://www.butlergroup.com.

[7] http ://www.microsoft.com/france/msdn/support/colones/info/
info.asp ?mar=/france/msdn/support/colones/info/xsltprocessor.html.

[8] http ://www.oracle.com/technology/products/text/index.html.

[9] http ://www.softwareag.com/corporate/products/tamino/default.asp.

[10] http ://www.stanford.edu/dept/itss/docs/oracle/10g/appdev.101/b10790/xdb11jav.htm.

[11] http ://www.utexas.edu/its/unix/reference/oracledocs/v92/b1050101/
appdev.920/a96620/xdb09jav.htm.

[12] http ://www.w3.org.

[13] http ://www.w3.org/dom/.

[14] http ://www.w3.org/style/xsl/.

[15] http ://www.w3.org/tr/xslt.

[16] http ://www.w3.org/xml/schema.

[17] http ://xml.apache.org/xindice/.

[18] http ://xmlfr.org/w3/tr/xpath/.

[19] Oracle9i xml developer's kits guide - xdk release 2 (9.2).

[20] *XML and DB2.* IEEE Computer Society, Washington, DC, USA, 2000.

[21] *Database Management Systems - Managing Relationnal and XML Data Structures.* Butler Direct Ltd., 2002.

[22] Boukhors, Kaszycki, Laplace, Munerot, and Poublan. *XML La synthèse.* Dunod, 2002.

[23] Chang, Scardina, and Kiritzov. *Oracle 9i XML handbook.* Mc graw-hill professional edition, 2001.

[24] Medicke J. Cutlip R. *Integrated solutions with DB2.* Addison-wesley professional edition, 2003.

[25] IBM DB2 Universal Database. *XML Extender Administration and Programming Version 8.*

[26] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. A query language for xml. In *WWW '99 : Proceeding of the eighth international conference on World Wide Web*, New York, NY, USA, 1999. Elsevier North-Holland, Inc.

[27] Florescu and Kossmann. *Storing and querying XML data using a RDBMS.*

[28] Harry J. Foxwell. Java and xslt. *Markup Lang.*, 3(4) :446–447, 2001.

[29] Rick Greenwald and David C. Kreines. *Oracle in a Nutshell.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2002.

[30] Steven Holzner. *Inside XML.* New Riders Publishing, Thousand Oaks, CA, USA, 2000.

[31] Marais and Georger. Module bddo : Outils proposés par oracle pour la liaison xml - sgbd xsu et xsql servlet. Ecole Nationale Supérieure des Télécommunications Département Informatique et Réseaux.

[32] Carl W. Olofson. *Worldwide RDBMS 2005 Vendor Shares : Preliminary Results for the Top 5 Vendors Show Continued Growth.* IDC, 2006.

[33] Erik T. Ray and Christopher R. Maden. *Learning XML.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[34] Nicholas Rhodes. "xml data management : Native xml and xml-enabled database systems" by akmal b. chaudhri, awais rashid, and roberto zi-

cari, boston, ma : Addison-wesley 2003. *J. Am. Soc. Inf. Sci. Technol.*, 55(1), 2004.

[35] Sanyal, Martineau, Gashyna, and Kyprianou. *DB2 universal database V8 application development certification guide.* Prentice Hall Ptr., 2003.

[36] Eric van der Vlist. *XML Schema.* O'Reilly, Jun 2002.

[]