# Prise en charge du langage XML en mode natif

# Avec IBM DB2 9

Matthias Nicola

IBM Silicon Valley Lab

555 Bailey Avenue
San Jose, CA
USA
mnicola@us.ibm.com

Bert van der Linden

IBM Silicon Valley Lab

555 Bailey Avenue
San Jose, CA
USA
robbert@us.ibm.com

## Résumé

Les principaux systèmes de bases de données relationnelles assurent depuis plusieurs années la prise en charge du langage XML, en mappant pour l'essentiel ce langage XML à des concepts existants, tels que les objets LOB ou les tables relationnelles objet. Les limites de ces systèmes sont bien connues dans la recherche et l'industrie. Pour y remédier, des améliorations sont apportées à une prochaine version de DB2 Universal Database® de manière à prendre intégralement en charge le langage XML *natif*. « natif » signifie que les documents XML sont stockés sur des pages de disque dans des arborescences qui correspondent au modèle de données XML. Cela permet d'éviter le mappage entre les structures XML et relationnelles, ainsi que les limites qui en découlent. Des indexes XML, la prise en charge intégrale de XQuery, SQL/XML et XML Schema et des utilitaires (chargeur XML haute capacité et haut débit parallèle, par exemple) viennent compléter le stockage en langage XML natif. DB2 devient ainsi un véritable système de bases de données hybride qui traite à parts égales la gestion des données XML et relationnelles.


**Actes de la 31e conférence de VLDB,
Trondheim, Norvège, 2005**

## 1 Introduction

Le langage XML constitue la norme de facto d'échange de données entre différents systèmes, plates-formes, applications et organisations. Le langage XML présente les principaux avantages suivants : indépendance entre fournisseur et plate-forme et flexibilité élevée. Le langage XML est un modèle de données adapté à toute combinaison de données structurées, non structurées et semi-structurées. Les données XML sont facilement extensibles, car de nouvelles balises peuvent être définies en fonction des besoins. Les documents XML sont également facilement convertibles en langage XML « d'apparence différente » et même dans d'autres formats (HTML, par exemple).

La conformité des documents XML par rapport à un schéma est, par ailleurs, facilement vérifiable. Tout cela est devenu possible grâce à des outils et à des normes largement accessibles (analyseur XML, technologie XSLT, norme XML Schema, par exemple). Ils soulagent considérablement les applications qui n'ont plus la lourde tâche de traiter les spécificités des formats de données propriétaires. À une époque où les formats de message, les formulaires commerciaux et les services changent très souvent, le langage XML diminue en conséquence le coût et la durée de gestion de la logique applicative.

Outre l'échange de données au format XML, les entreprises conservent en permanence d'énormes quantités de données de gestion critiques au format XML. Cela s'explique de plusieurs manières. Certaines entreprises doivent conserver les documents XML dans leur format d'origine à des fins de contrôle et de conformité à des règlements. Les documents juridiques et financiers et les formulaires électroniques, notamment dans le secteur

public, en sont des exemples types.

Le recours au langage XML en tant que format de stockage permanent s'explique également par le fait que le langage XML peut s'avérer être un modèle de données mieux adapté qu'un schéma relationnel. Cela est non seulement vrai pour les applications orientées contenu, mais également pour certaines applications orientées données. Par exemple, dans les applications biologiques, les données sont par essence très complexes et hiérarchisées, mais elles peuvent contenir de grandes quantités d'informations non structurées. La plupart des données génomiques actuelles sont encore conservées dans des formats de fichiers plats propriétaires, mais de nombreuses initiatives vont aujourd'hui dans le sens du passage au langage XML [14].

Les bases de données relationnelles assurent la prise en charge du stockage, de la manipulation, de la recherche et de la récupération de données XML. En règle générale, cela repose sur le stockage de documents XML dans des objets LOB ou sur le mappage avec décomposition entre un schéma XML et un schéma relationnel. Ces solutions présentent des contraintes qui leur sont propres au niveau du fonctionnement et des résultats. Le stockage d'objets LOB accélère généralement l'insertion et la récupération de documents complets, mais ses résultats en matière de recherche et d'extraction sont mauvais du fait de l'analyse XML au moment de l'exécution de la requête. Ces résultats peuvent être légèrement améliorés lorsque les indexes sont générés au moment de l'insertion. Même si cela implique une surcharge d'analyse XML, cela peut accélérer les requêtes qui recherchent les documents correspondant à certaines conditions de recherche. L'extraction partielle de documents et les mises à jour au niveau des sous-documents requièrent néanmoins une analyse XML coûteuse. La décomposition du langage XML en tables relationnelles reste onéreuse au moment de l'insertion, du fait de l'analyse XML et des insertions multitables qui reviennent cher. Mais, une fois le langage XML décomposé en valeurs scalaires relationnelles, les requêtes et les mises à jour en langage SQL brut offrent de meilleures performances. Ce système présente toutefois des inconvénients : les schémas XML peuvent comporter de nombreux éléments imbriqués et récurrents, si bien que le schéma relationnel correspondant peut comprendre des dizaines voire même des centaines de tables.

La définition d'un tel mappage entre un schéma XML et un schéma relationnel reste une tâche complexe. Une fois les données insérées et suite aux modifications apportées au schéma XML, il est quasiment systématiquement impossible d'apporter des modifications au schéma relationnel. Cela réduit fortement la flexibilité pour laquelle le langage XML est souvent utilisé à la base. Les jointures multidirectionnelles nécessaires à la reconstitution des documents XML peuvent également représenter une dépense importante dans le cadre du traitement de grandes quantités de données [12].
De surcroît, des requêtes XQuery complexes peuvent même s'avérer intraduisibles dans le langage SQL [5].

Tous ces éléments sont à la base de la technologie de base de données en langage XML natif. Des améliorations ont été apportées à DB2 Universal Database® de manière à prendre intégralement en charge le langage XML natif. Cet article présente les caractéristiques XML de la prochaine version de DB2 et décrit les principaux concepts de mise en œuvre. Des exemples illustrent les capacités XML, ainsi que l'intégration du langage XML à la gestion des données relationnelles et SQL.

Les travaux connexes sont ensuite exposés dans la section 2. Une vue d'ensemble de la solution en langage XML natif de DB2 et son niveau élevé d'architecture sont proposés dans la section 3. Le stockage XML natif et le mécanisme d'indexage XML sont respectivement abordés dans les sections 4 et 5. La prise en charge de XQuery et de SQL/XML est expliquée dans la section 6. La prise en charge de XML Schema et le référentiel de schémas de DB2 sont ensuite décrits dans la section 7. L'importance de la fonction de décomposition XML et la nouvelle solution de décomposition de DB2 sont présentées dans la section 8. Il est question de la prise en charge des applications et des améliorations apportées aux interfaces API par rapport au langage XML dans la section 9. Divers outils XML (importation, exportation et chargement XML, par exemple) et le générateur XQuery graphique sont présentés dans la section 10. Cet article se termine par une synthèse.

## 2 Travaux connexes

Diverses bases de données (Tamino, XHive, Ipedo, NeoCore et Xyleme, par exemple) existent en langage XML natif depuis plusieurs années [4]. La méthode de stockage XML décrite en [7] s'apparente à celle d'IBM, en ce sens que les documents sont divisés en sous-arbres de nœuds.

Dans Oracle 10g, les documents XML peuvent être stockés sous forme d'objets CLOB grâce à la prise en charge de l'indexage, être décomposés en tables relationnelles objet ou les deux à la fois [11]. Dans Microsoft SQL Server 2005, la prise en charge XML permet de stocker les documents XML sous forme de séquences d'octets dans des colonnes d'objets BLOB [12]. Un index XML primaire peut être défini pour éviter l'analyse des objets BLOB XML au moment de l'interrogation [12]. Des indexes XML secondaires peuvent également être définis pour optimiser les performances des requêtes. Cela diffère sensiblement de la méthode de stockage et d'indexage XML de DB2 décrite dans les sections 4 et 5. Dans DB2, l'analyse XML n'est jamais obligatoire au moment de l'interrogation et les indexes peuvent être définis sur des chemins spécifiques. La prise en charge XML à venir dans DB2 repose sur un prototype décrit dans [2]. Les concepts de modélisation et d'architecture les plus généraux sont traités en [2] et ne sont pas abordés dans cet article. D'autres travaux connexes sont étudiés en [9] et en [2].

## 3 Vue d'ensemble

Le niveau supérieur de DB2 avec prise en charge du langage XML natif est illustré dans la Figure 1. le composant de stockage de DB2 gère à la fois le stockage des données relationnelles et le nouveau stockage XML natif. Le moteur DB2 accède à ces deux types de stockage et passe au traitement SQL brut , SQL/XML [6] et XQuery [3] de manière intégrée. Différents analyseurs permettent de lire les instructions SQL et XQuery, mais un seul compilateur est utilisé pour les deux langages. Aucune traduction n'est effectuée de XQuery en SQL. Des améliorations ont été apportées au compilateur et à l'optimiseur de DB2 de manière à assurer le traitement SQL et XQuery dans une seule structure de modélisation [2]. La prise en charge du traitement correspondant est intégrée au gestionnaire d'indexes, au système d'exécution, à la gestion de mémoire, au dictionnaire de données, au contrôle des accès concurrents, à la couche de stockage et aux utilitaires de bases de données.

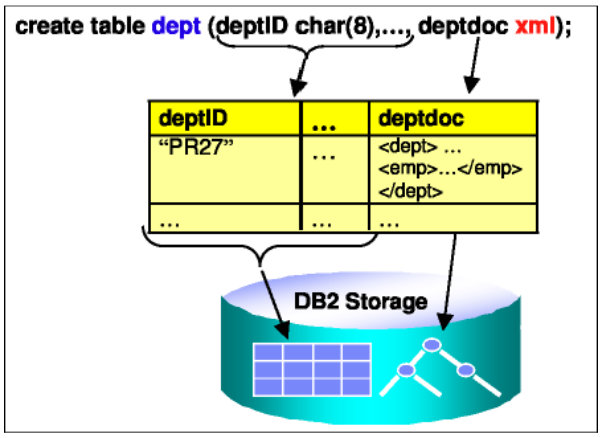


**Figure 1 :** Intégration de données relationnelles et XML dans DB2

Il n'y a aucune répercussion sur les applications SQL existantes. Une application cliente peut continuer à utiliser le langage SQL pour communiquer avec le serveur DB2 via les interfaces API relationnelles et accéder aux données qui figurent dans le magasin de données relationnelles et les manipuler. Les améliorations apportées au langage SQL/XML permettent également de publier les données relationnelles au format XML. Le langage SQL assure par ailleurs la récupération de documents complets à partir du stockage XML natif. Les applications SQL comportent de nouvelles fonctions SQL/XML, avec également des fonctionnalités de recherche et d'extraction au niveau des sous-documents (en intégrant XPath ou XQuery aux instructions SQL, par exemple).

Une application XML peut interagir avec le serveur DB2 via l'interface XML à l'aide du langage XQuery. En règle générale, les requêtes XQuery accèdent au magasin XML natif. XQuery est pris en charge en tant que langage d'interrogation autonome, indépendant du langage SQL. Les requêtes XQuery peuvent cependant comporter des instructions SQL permettant de combiner des données XML avec des données relationnelles et de les mettre en corrélation. Comme le langage XUpdate n'est pas encore normalisé, le serveur DB2 prend pour l'instant en charge les mises à jour de documents complets. Grâce à une procédure stockée de mise à jour XML, les applications disposent d'une interface flexible pour les mises à jour au niveau des sous-documents. Il n'est ainsi plus nécessaire d'envoyer les documents à mettre à jour du serveur DB2 au client et inversement.

### 3.1 Le type de données XML

La notion de 'data type' XML est à la base de la prise en charge du langage XML en mode natif dans DB2. Le langage XML est désormais un type de données au même titre que tout autre type de donnée SQL [6]. Le type de données XML peut être utilisé dans une instruction « create table » pour définir plusieurs colonnes de type XML (voir Figure 2). Comme le statut du langage XML ne diffère pas de celui des autres types, les tables peuvent comporter toute combinaison de colonnes de données XML et de colonnes de données relationnelles. Une application reposant uniquement sur le langage XML ne peut définir que des tables contenant des colonnes de données XML. Une colonne de type XML peut comporter un document XML syntaxiquement correct par ligne dans la table. La valeur NULL permet d'indiquer l'absence de document XML. Même si tout document XML est logiquement associé à une ligne dans une table, les colonnes de données XML et les colonnes de données relationnelles sont stockées différemment. Les données relationnelles et les données XML sont en effet stockées dans des formats différents, qui correspondent à leur modèle de données respectif. Les colonnes de données relationnelles sont stockées dans des structures linéaires traditionnelles, tandis que les données XML sont stockées dans des structures hiérarchisées. Les deux sont étroitement liées en vue d'optimiser leur accès croisé.


**Figure 2 :** Table avec une colonne de type « XML »

[DB2 Storage = Stockage DB2]

Aucun schéma XML n'est nécessaire pour définir une colonne XML ou insérer ou interroger des données XML. Une colonne XML peut contenir des documents sans schéma, ainsi que des documents avec plusieurs schémas XML différents ou évolutifs. La validation des schémas est facultative et s'effectue document par document. L'association entre les schémas et les documents s'effectue par conséquent document par document et non colonne par colonne, ce qui optimise la flexibilité.

À la différence du type Varchar ou CLOB, aucune longueur n'est associée au type XML. L'architecture de traitement et de stockage XML n'impose aucune limite quant à la taille d'un document XML. Seul le protocole de communication client-serveur limite actuellement les liaisons entrantes et sortantes XML à 2 Go par document. À de rares exceptions près, toutes les applications XML sont supportées.

Les valeurs de type XML sont traitées dans une représentation interne qui n'est pas une chaîne et qui n'est pas directement comparable à des chaînes. La fonction XMLSERIALIZE permet de convertir une valeur XML en valeur de chaîne représentant le même document XML.

De la même manière, la fonction MLPARSE permet de convertir une valeur de chaîne représentant un document XML en valeur XML correspondante.

Le type XML peut être utilisé non seulement comme type de colonne, mais aussi comme type de données pour les variables hôtes dans des langages tels que C, Java et COBOL. Cette amélioration apportée aux interfaces API de DB2 est détaillée dans la section 9. Le type XML est également admis avec les paramètres et les variables dans les procédures stockées SQL, les fonctions définies par l'utilisateur et les procédures stockées externes écrites en C et en Java. Cela revêt de l'importance dans le cadre du développement d'applications flexibles.

```
<dept>
 <employee id=901>
   <name>John Doe</name> <phone>408 555
   1212</phone> <office>344</office>
 </employee>
 <employee id=902>
   <name>Peter Pan</name> <phone>408 555
   9918</phone> <office>21 6</office>
 </employee>
</dept>
```

**Figure 3**: exemple de document XML:

## 4 Stockage XML natif

Pour insérer des données XML dans la base de données, les applications clientes envoient les documents XML dans leur représentation textuelle au serveur DB2. Le serveur utilise un analyseur SAX pour vérifier que la syntaxe des documents entrants est correcte et procéder à la validation facultative. Les événements SAX sont convertis en représentation hiérarchisée du document XML.

Dans l'exemple de document de la Figure 3, cette hiérarchie s'apparente à l'arbre de document qui figure dans la partie supérieure de la Figure 4.

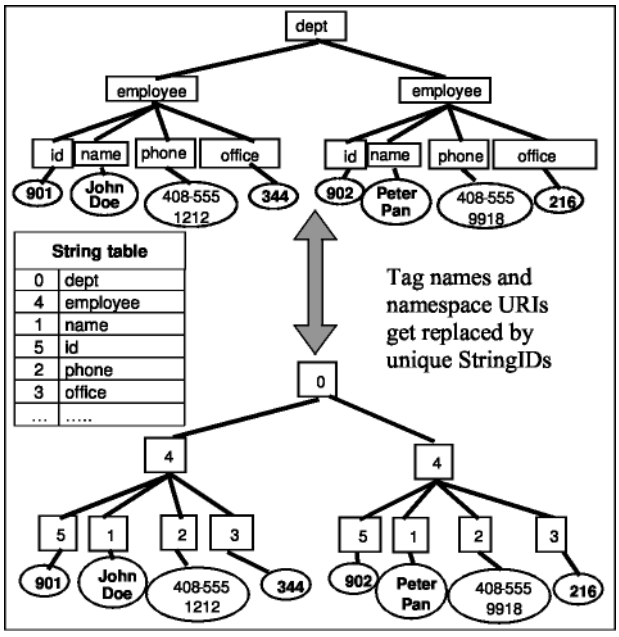


**Figure 4 :** ID de chaîne dans le stockage XML

[Sting table = Table des chaînes]

[Tag names and namespace URIs get replaced by unique StringIDs = Les noms de balise et les identificateurs URI de l'espace de noms sont remplacés par des ID de chaîne uniques.]

Au cours de l'insertion, tous les noms de balise et les identificateurs URI de l'espace de noms figurant dans l'arbre de document sont remplacés par des valeurs entières (ID de chaîne). La nouvelle table de catalogue SYSXMLSTRINGS indique le mappage des balises en ID de chaîne de toutes les colonnes XML de la base de données.

Il n'existe qu'une seule entrée par chaîne. Dans l'exemple donné dans les Figures 3 et 4, deux occurrences de la balise « id » apparaissent dans l'exemple de document et probablement bien davantage dans d'autres documents de la base de données active, mais chaque occurrence de cette balise est remplacée par le même ID de chaîne « 5 ».
Lors de la première insertion d'une balise à l'échelle de la base de données, un ID de chaîne est affecté à la table des chaînes et y est enregistré. Dans toutes les occurrences suivantes, cette balise sera remplacée par le même ID de chaîne. En règle générale, la table de mappage est de très petite taille car elle correspond au nombre de balises uniques figurant dans la base de données (le plus souvent elles se comptent en centaines ou en milliers). Un cache *ad hoc* optimise l'accès à cette table.

L'arbre de document représenté dans la partie inférieure de la Figure 4 s'apparente au format de stockage des documents insérés sur les pages de disque. D'autres informations sont stockées avec chaque nœud (annotation sur le type en cas de validation du document, par exemple).

Le remplacement des balises par des ID de chaîne permet non seulement de réduire l'espace occupé, mais également d'optimiser les performances des requêtes navigationnelles. Certaines opérations, telles que les comparaisons de nœuds, fonctionnent maintenant avec des entiers plutôt que des chaînes. Des informations détaillées sur la navigation dans les documents sont disponibles dans [2]. À chaque retour de documents ou de nœuds XML sous forme de résultats de requête, les nœuds sont resérialisés dans leur forme textuelle. Dans ce processus, le mappage des ID de chaîne en balises proprement dites est inversé.

Lorsque l'arbre d'un document est trop grand pour tenir sur une seule page, il est divisé en *régions* (voir Figure 5). Quel que soit le niveau du document, un sous-arbre de nœuds peut faire l'objet d'un découpage et devenir une région. Les régions d'un document peuvent être stockées sur des pages distinctes qui ne doivent pas nécessairement se suivre physiquement. Plusieurs régions peuvent être stockées sur une seule page, notamment lorsque les documents sont de plus petite taille que celle de la page et que chaque document ne représente qu'une seule région.

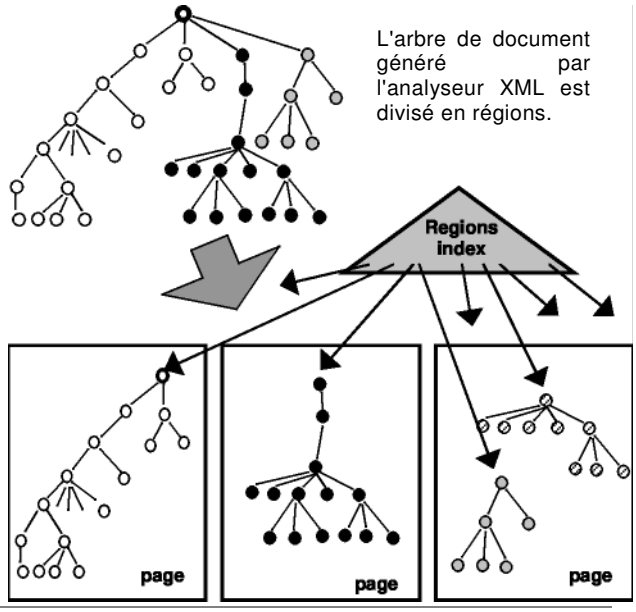


**Figure 5 :** Régions interconnectées d'un document

[Regions index = Index des régions]

Lorsqu'un document s'étend sur plusieurs pages, ses régions sont interconnectées par l'index des régions. Un index de régions est un index système qui est automatiquement créé pour chaque table contenant une ou plusieurs colonnes XML. Dans la Figure 5, l'arbre de document est divisé en trois régions représentées en blanc, en noir et en gris. La région en blanc et la région en noir occupent chacune une page complète. La région en gris est plus petite et figure sur une page qui comporte déjà une autre petite région.

Lors du balayage d'un document, un déplacement vers un nœud parent, frère ou enfant peut ne pas mener à un nœud de la même page, mais à une région différente. Dans ce cas, un index de régions permet de rechercher la page qui contient la région correspondante. Dans la Figure 5, il est possible de naviguer de la région en blanc vers la région en gris du document sans passer par la région en noir. Pour les documents de grande taille, cela signifie que seules les pages qui sont réellement indispensables à l'évaluation d'une requête donnée ont besoin d'être extraites du disque. Pour accéder à un document partiel, cela permet d'économiser de coûteuses entrées/sorties.

L'établissement de liens directs entre les régions, qui s'apparente à la mise en œuvre décrite en [7], constitue une alternative à l'index de régions. Dans notre système, c'est l'index de régions qui est utilisé pour accéder de manière efficace au niveau des sous-documents et préextraire de façon intelligente les régions.

Le stockage paginé des documents XML utilise les composants existants de DB2 (gestionnaire du pool de mémoire tampon, couche de l'espace table et fonction de consignation).

# 5 Indexes XML

Les applications XML qui gèrent des millions de documents XML ne sont pas rares. La prise en charge de l'indexage des données XML s'avère par conséquent nécessaire pour optimiser les performances des requêtes. DB2 prend en charge les indexes de valeurs de chemin dans les colonnes XML, de manière à indexer les éléments et les attributs fréquemment utilisés dans les prédicats et les jointures interdocuments. DB2 prend également en charge l'indexage en texte intégral XML.

## 5.1 Indexes de valeurs XML

Dans le cadre des exemples de table et de document donnés dans les Figures 2 et 3, un index de valeurs XML est défini dans l'instruction suivante pour tous les noms des employés dans tous les documents de la colonne XML « deptdoc » :

```
create index idx1 on dept(deptdoc) generate key
using xmlpattern '/dept/employee/name' as sql varchar(35)
```

Le xmlpattern correspond à un chemin qui identifie les nœuds XML à indexer. Il est désigné par *xmlpattern* et non par *xpath*, car seul un sous-ensemble du langage XPath est autorisé dans les définitions d'index. Par exemple, les caractères génériques (//,*) et les espaces de noms sont autorisés, mais les prédicats XPath tels que /a/b[c=5] ne sont pas pris en charge. Comme un schéma XML unique n'est pas exigé pour tous les documents d'une colonne XML, DB2 risque de ne pas connaître le type de données à utiliser dans l'index pour un xmlpattern donné. L'utilisateur doit par conséquent spécifier explicitement le type de données dans la clause *as sql* <*type*>. Les types suivants peuvent être définis :

- VARCHAR(*n*) - Pour les nœuds dont la longueur maximale des valeurs est connue.
- VARCHAR HASHED - Pour les nœuds dont la longueur des valeurs est arbitraire. Dans ce cas, l'index contient les valeurs hachées des chaînes réelles. Ce type d'index peut être utilisé pour les prédicats d'égalité, mais pas pour les prédicats de type BETWEEN.
- DOUBLE - Pour les nœuds de type numérique.
- DATE et TIMESTAMP - Pour les nœuds et leurs valeurs XML correspondantes.

Comme le système de type SQL diffère sensiblement du système de type XML, des mécanismes ont spécialement été mis en place pour compenser les principales différences entre les deux. Par exemple, des améliorations ont été apportées au gestionnaire d'indexes de DB2 de manière à gérer explicitement des valeurs spécifiques à partir du système de type XML, à savoir +0, -0, +INF, -INF et NaN.

Lorsqu'un nœud correspondant au xmlpattern ne parvient pas à procéder à un transtypage vers le type d'index spécifié, aucune entrée d'index n'est alors créée pour ce nœud sans signaler une erreur [2]. Un seul document peut comporter zéro, un ou plusieurs nœuds correspondant au xmlpattern. Il peut par conséquent exister zéro, une ou plusieurs entrées d'index pour une seule ligne de la table. Il s'agit d'une différence significative par rapport aux indexes des colonnes relationnelles.

Dans l'exemple d'instruction suivante, un index unique est défini pour tous les attributs d'ID employé. L'unicité est appliquée à l'intérieur d'un document et entre tous les documents de la colonne XML.

```
create unique index idx2 on dept(deptdoc) generate key
using xmlpattern '/dept/employee/@id' as sql double
```

Dans certaines applications, les éléments ou les attributs qui seront recherchés sont difficilement prévisibles. Dans ce cas, les définitions d'index suivantes peuvent être utilisées pour indexer respectivement tous les nœuds de texte et tous les attributs, si nécessaire. L'exemple ci-dessous illustre des éléments dont la longueur des valeurs est arbitraire, ainsi que des attributs numériques :

```
create index idx3 on dept(deptdoc) generate key using
xmlpattern '//text()‘ as sql varchar(hashed)

create index idx4 on dept(deptdoc) generate key
using xmlpattern '//@*' as sql double
```

Pour établir une correspondance entre les nœuds et les indexer, le xmlpattern peut contenir des déclarations de noms d'espace et des préfixes de noms d'espace :

```
create index idx5 on dept(deptdoc) generate key using
xmlpattern 'declare namespace m="http://www.me.com/";
/m:dept/m :employee/m : name‘ as sql varchar(45)
```

Pour diminuer la taille des entrées d'index, chaque chemin unique figurant dans les documents d'une colonne XML est mappé à un entier PathID. Ce système s'apparente au concept d'ID de chaîne décrit pour les balises dans la section 4. Là encore, les informations de mappage sont placées dans la mémoire cache pour optimiser les performances et elles occupent généralement peu d'espace car seuls les chemins uniques sont enregistrés.

Chaque entrée d'index indique le PathID qui identifie le chemin du nœud indexé, la valeur du nœud transtypé vers le type d'index, un RowID et un NodeID.

Les RowID identifient les lignes qui comportent les documents correspondants, à l'image des indexes relationnels classiques. Les NodeID identifient quant à eux les régions et les nœuds correspondants dans les documents.

En règle générale, les indexes sont définis avec des xmlpatterns qui identifient les nœuds atomiques. Un nœud est réputé « atomique » lorsqu'il s'agit d'un attribut, d'un nœud de texte ou d'un élément dépourvu d'éléments enfants, mais possédant précisément un nœud de texte enfant. Tous les exemples d'indexes présentés ci-dessus indexent les nœuds atomiques dans le document illustré dans la Figure 3. Il est toutefois également possible de définir des indexes sur des nœuds non atomiques. Dans notre exemple, le schéma XML « /dept/employee » serait considéré comme « non atomique », car chaque élément « employee » possède trois éléments enfants avec chacun un nœud de texte. Il en résulte une seule entrée d'index pour chaque élément « employee ». La valeur d'une entrée de ce type correspond à la concaténation de tous les nœuds de texte du sous-arbre figurant sous « employee ». Cela est conforme au modèle de données XML. Pour indexer tous les noms des employés, les bureaux et les numéros de téléphone sous forme de valeurs séparées, il convient d'utiliser le xmlpattern « /dept/employee/*/text() » ou trois instructions create index distinctes. Les indexes non atomiques sont rarement utiles pour le langage XML orienté données, mais ils peuvent s'avérer utiles dans le cadre d'un contenu mixte en langage XML orienté texte. Par exemple, l'élément « title » comporte du contenu mixte indiquant une suggestion de mise en forme. Dans ce cas, un index non atomique « …/title » s'avère utile, car la valeur complète de « title » est indexée sans prise en compte de la suggestion de mise en forme :

```
<title>The benefits of<bold>XML</bold></title>
```

Un index donné peut être utilisé pour déterminer la valeur d'un prédicat XPath uniquement lorsque le type de données utilisé dans le prédicat correspond à celui qui figure dans l'index et que le XPath qualifie un sous-ensemble de nœuds indexés.

Par exemple, l'index **idx3** ci-dessus pourrait être utilisé pour déterminer la valeur du prédicat /dept//name[text()=”Joe”]. L'index **idx2** ne pourrait par contre pas être utilisé pour déterminer la valeur du prédicat //@id=”A167” pour les deux raisons suivantes : (a) **idx2** est un index numérique, mais le prédicat demande une comparaison de chaînes, (b) le prédicat recherche les attributs @id dans l'ensemble du document, mais **idx2** ne couvre que ceux figurant sous /dept/employee. Des informations détaillées sur l'admissibilité des indexes sont disponibles dans [2] et [1].

## 5.2 Indexes en texte intégral XML

La recherche en texte intégral est une opération courante dans les applications XML orientées documents et contenu. Des améliorations ont été apportées aux fonctions de recherche textuelle actuelles de DB2 de manière à utiliser le nouveau type de colonne XML. Les indexes en texte intégral avec reconnaissance des structures de documents XML peuvent être définis sur toute colonne en langage XML natif. Les documents figurant dans une colonne XML peuvent être indexés complètement ou partiellement (par exemple, lorsque l'on sait à l'avance qu'une partie spécifique de chaque document fera l'objet d'une recherche en texte intégral, telle qu'un élément « description » ou « comment »). Les expressions de recherche textuelle sont de ce fait applicables à des chemins spécifiques dans un document.

L'instruction suivante définit un index de texte qui indexe complètement les documents de la colonne XML **deptdoc** figurant dans la table **dept** de la base de données **personnelldb :**

```
create index mylndex for text on dept (deptdoc) format xml
connect to personnelldb
```

La requête suivante exploite cet index, mais limite la recherche à un élément spécifique. La requête récupère tous les documents dans lesquels l'élément « /dept/comment » comporte le terme « Brésil » :

```
select deptdoc from dept where
contains (deptdoc,‘sections(”/dept/comment”) “Brésil” ') = 1
```

La recherche textuelle dans des parties spécifiques de documents est une fonction essentielle dans de nombreuses applications. Des fonctions de recherche textuelle standard sont également disponibles, telles que le pointage et le classement des résultats de la recherche et la recherche par synonymie et analogie.

Pour optimiser les performances des opérations d'insertion, de mise à jour et de suppression XML, l'index de texte est géré de manière asynchrone, à savoir en dehors du contexte d'une transaction DML. Une commande « update index » est toutefois disponible pour forcer la synchronisation de l'index de texte.

# 6 XQuery et SQL/XML

SQL et XQuery sont tous deux pris en charge par DB2 en tant que langages d'interrogation de base. Ils fonctionnent avec leur propre modèle de données et peuvent être utilisés indépendamment l'un de l'autre. L'intégration de ces deux langages pris en charge par DB2 peut toutefois s'avérer extrêmement utile pour les applications de base de données. Comme de nombreuses applications traitent simultanément les données relationnelles et les données XML, les requêtes doivent combiner ces deux types de données et les mettre en corrélation. Des plus amples informations sont disponibles dans les sous-sections ci-dessous. Il est fait référence aux deux tables suivantes dans les exemples donnés dans cet exposé :

```
create table dept(deptID char(8) primary key, deptdoc xml)
create table unit(ID char(8), name char(20), manager char(20))
```

## 6.1 Interrogation de données XML avec XQuery

Dans DB2, les requêtes XQuery peuvent fonctionner avec des documents XML figurant dans une ou plusieurs colonnes XML. Chaque colonne XML est interprétée comme étant une séquence de nœuds de documents XML. Cette interprétation s'effectue grâce à l'une des deux fonctions de DB2 « db2-fn:xmlcolumn » et « db2-fn:sqlquery ». Comme indiqué dans l'exemple ci-dessous, la fonction db2-fn:xmlcolumn contient un littéral de chaîne qui identifie une colonne XML. La fonction db2-fn:xmlcolumn retourne une séquence XML qui comporte tous les documents de la colonne spécifiée. La clause **for** qui figure dans cet exemple itère par conséquent sur tous les documents de la colonne XML. Si une valeur de colonne est null, la séquence XML qui en résulte est alors vide pour la ligne en question.

```
for $e in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept/employee
where $e/office = 344
return $e/name
```

La fonction db2-fn:xmlcolumn peut être utilisée à plusieurs reprises dans une même requête XQuery pour faire référence à différentes colonnes XML figurant dans une même table ou dans des tables distinctes ou pour faire plusieurs fois référence à une même colonne XML. La fonction db2-fn:xmlcolumn génère systématiquement tous les documents d'une colonne XML servant de base à la requête XQuery. Ce cas d'emploi est très fréquent. Il peut toutefois s'avérer parfois souhaitable de limiter la base d'une requête XQuery en émettant des conditions sur des colonnes relationnelles dans la même table ou des tables connexes. Cette limitation peut s'effectuer à l'aide de la fonction db2-fn:sqlquery qui accepte toute instruction select retournant une seule colonne XML.

L'exemple de requête donné dans la Figure 6A équivaut à la requête comportant la fonction db2-fn:xmlcolumn ci-dessus, car l'instruction SQL imbriqué retourne simplement tous les documents XML de la colonne XML. Dans la Figure 6B, la base de la requête XQuery est toutefois judicieusement limitée à un seul document, car le prédicat relationnel exploite l'index des clés primaires sur deptID.

Dans l'exemple donné dans la Figure 6C, l'ensemble des documents servant de base à la requête XQuery est filtré à l'aide d'une jointure et d'un prédicat sur une autre table relationnelle. La puissance de l'intégration des langages XQuery et SQL prend dans ce cas toute sa valeur. Les utilisateurs peuvent exploiter toutes les données relationnelles existantes pour qualifier des documents XML en vue de leur traitement XQuery. La fonction db2-fn:sqlquery permet non seulement de limiter la base d'une requête XQuery, mais aussi de la développer. Tel est le cas dans la Figure 6D où une requête UNION est utilisée pour rechercher l'employée Jane Doe dans tous les services américains et tous les services britanniques (à l'aide des tables deptUS et deptUK, légère variante par rapport à l'exemple en cours).

```
for $e in db2-fn:sqlquery('select deptdoc
                           from dept')/dept/employee
where $e/office = 344
return $e/name
```
A

```
for $e in db2-fn:sqlquery('select deptdoc
                           fromdept
                           where deptID = PR27"')/dept/employee
where $e/office = 344
return $e/name
```
B

```
for $e in db2-fn:sqlquery('select deptdoc
                           fromdept, unit
                           where dept.deptID=unit.ID and
                           unit.manager = "Jim
                           Qu"')/dept/employee
where $e/office = 344
return $e/name
```
C

```
for $e in db2-fn:sqlquery('select deptdoc
                           from deptUS UNION
                           select deptdoc
                           from deptUK')/dept/employee
where $e/name = "Jane Doe"
return $e
```
D

**Figure 6 :** Requêtes XQuery avec SQL imbriqué

Grâce à la fonction db2-fn:sqlquery, les applications peuvent également utiliser XQuery pour accéder à des données relationnelles et les récupérer. Les fonctions de construction SQL/XML, qui convertissent les données relationnelles au format XML et génèrent une seule colonne de type XML pouvant servir de base à une requête XQuery, facilitent ce travail. L'adoption du modèle de données XQuery par le langage SQL/XML a rendu possible cette intégration [6].

Dans l'exemple donné ci-dessous, une requête XQuery construit un document de résultats qui comporte des informations sur l'unité et le service. Les informations sur le service figurent dans un document XML récupéré à partir de la colonne XML deptdoc. Les informations sur l'unité proviennent d'une table purement relationnelle. L'instruction SQL/XML construit un élément XML

« Unit » avec trois éléments enfants dont les valeurs sont extraites des colonnes relationnelles de la table des unités, à savoir le responsable, le nom et l'ID des colonnes.

```
let $d := db2-fn:sqlquery('select deptdoc from dept
                           where deptID = "PR27"')
let $u := db2-fn:sqlquery('select XMLELEMENT(NAME "Unit",
                           XMLFOREST(ID, name, manager)) from
                           unit where ID = "PR27"')
return <report>
          <units>{$u}</units>
          <department>{$d}</department>
       </report>
```

Une requête XQuery et une ou plusieurs requêtes SQL imbriqué sont compilées dans un seul plan d'exécution et comporte une seule instruction. Les niveaux d'isolement SQL et les privilèges de sécurité s'appliquent à l'ensemble de l'instruction sous forme d'unité simple, comme pour toute instruction SQL classique.

Le résultat retourné par une instruction XQuery est traité sous forme de table comportant une seule colonne de type XML. Chaque ligne retournée représente un élément de la séquence XML résultant de la requête XQuery. Les mécanismes existants dans DB2 peuvent par conséquent être utilisés pour déclarer et ouvrir des curseurs, extraire des éléments de la séquence XML retournée par la requête XQuery et fermer des curseurs. Notez que ces éléments peuvent tout aussi bien être des documents XML que des valeurs atomiques telles que des entiers ou des chaînes.

### 6.2 Interrogation de données XML avec SQL

Il est souvent recommandé d'utiliser et/ou d'améliorer des instructions SQL pour récupérer des données XML. Cela s'explique par le fait que les utilisateurs des bases de données connaissent bien le langage SQL, ce qui constitue un bon point de départ pour la gestion du langage XML. Les applications relationnelles existantes sont fréquemment complétées avec des données XML. Il est par conséquent normal d'améliorer les applications SQL existantes et même les instructions SQL existantes avec des capacités XML.

Comme le langage XML est devenu un type de données SQL classique [6], des documents complets peuvent être extraits d'une colonne XML à l'aide d'une simple instruction select :

```
select deptdoc from dept where deptID LIKE "PR%";
```

DB2 prend également en charge la plupart des nouveaux prédicats et fonctions SQL/XML, parmi lesquels XMLQUERY, XMLEXISTS, XMLTABLE, XMLVALIDATE, XMLPARSE et XMLCAST. Comme ils font l'objet d'une description détaillée en [6], seuls les modes de déploiement les plus utiles de ces fonctions sont ici abordés.

XMLEXISTS est un prédicat booléen qui vérifie si un document XML correspond aux critères donnés. Il retourne soit la valeur true soit la valeur false pour chaque ligne. Le prédicat XMLEXISTS détermine la valeur d'une expression XPath ou XQuery pour chaque valeur d'une colonne XML. Si l'expression XQuery a pour résultat une séquence vide, XMLEXISTS retourne alors la valeur false, sinon il retourne la valeur true.

La requête donnée en exemple ci-dessous retourne les documents complets du service, comme dans l'exemple précédent mais avec le prédicat XMLEXISTS cette fois-ci pour augmenter le filtrage. Des lignes ne sont retournées que lorsque le document du service mentionne un employé du bureau 344. La clause **passing by** établit le lien entre le contexte SQL et le contexte XQuery [6].

```
select deptID, deptdoc
from dept d
where deptID LIKE "PR%" and
            xmlexists('$deptdoc/dept/employee[office = 344]'
            passing by ref d.deptdoc as "deptdoc")
```

Outre le filtrage du document, il est recommandé d'extraire et de retourner des documents XML partiels (sous-arbres ou valeurs d'attribut et d'élément atomiques, par exemple). Cela s'effectue grâce à la fonction XMLQUERY. Elle détermine la valeur des expressions XPath ou XQuery et retourne le résultat réel sous forme de séquence XML à l'application XML. Dans l'exemple ci-dessous, la requête sélectionne le deptID de tous les services PR et la fonction XMLQUERY extrait le nom de tous les employés PR du bureau 344.

```
select deptID, xmlquery('for $e in $deptdoc/dept/employee
                         where $e/office = 344
                         return $e/name
        passing by ref d.deptdoc as "deptdoc"
        returning sequence)
from dept d
where deptID LIKE "PR%";
```

Dans cette instruction, la fonction XMLQUERY retourne une séquence vide pour chaque document du service dans lequel aucun employé correspondant au bureau 344 n'a été trouvé. Pour que ces lignes n'apparaissent pas dans le résultat de la requête, il convient d'ajouter le prédicat XMLEXISTS correspondant à la clause where.

La présence d'une fonction XMLQUERY dans la clause where peut également permettre d'utiliser une valeur extraite dans une condition de jointure avec une colonne relationnelle dans une autre table. La valeur XML extraite doit être transtypée dans le type SQL de la colonne de jointure relationnelle. Dans l'exemple ci-dessous, les tables **unit** et **dept** sont jointes de manière à obtenir le nom de l'unité dont le responsable s'avère être l'employé numéro 901. La fonction XMLQUERY effectue une recherche dans les documents du service de la table **dept** et extrait le nom de l'employé dont l'ID est 901. La fonction xmlserialize transtype la valeur XML extraite en char(20) pour pouvoir la comparer à la colonne manager de la table **unit**.

```
select u.name, u.manager, d.deptID
from dept d, unit u
where xmlserial ize(content
   xmlquery('$deptdoc/dept/employee[@id=901]/name/text()'
                passing by ref d.deptdoc as "deptdoc"
                returning sequence) as char(20)
                ) = u.manager
```

### 6.3 Plans d'exécution de requêtes et opérateurs

DB2 possède des analyseurs distincts pour les instructions SQL et XQuery, mais il utilise un seul compilateur de requêtes intégré pour ces deux langages. Les plans d'exécution de requêtes peuvent comporter de nouveaux opérateurs XML pour la navigation XML (XSCAN), l'accès aux indexes XML (XISCAN) et les nouvelles jointures sur les indexes XML (XANDOR). Pour obtenir plus d'informations à ce sujet, reportez-vous à [2] et [8]. DB2 collecte également des statistiques sur les données XML que l'optimiseur de requêtes utilise pour créer des plans d'exécution de requêtes efficaces. Ces statistiques sur les données XML sont plus complexes que celles sur les données relationnelles, car elles doivent tenir compte non seulement de la répartition des valeurs mais également des statistiques structurelles. Des histogrammes représentant les occurrences des éléments, les occurrences des attributs et les occurrences des valeurs correspondantes contribuent à optimiser les requêtes.

## 7 Prise en charge de la norme XML Schema

DB2 prend en charge la validation XML Schema facultative des documents au cours des opérations d'insertion, de mise à jour et d'interrogation. Les définitions de type de document et les entités externes sont par ailleurs prises en charge de façon limitée. L'annotation sur le type générée par la validation est conservée avec le document en vue d'être utilisée au moment de l'exécution de la requête. DB2 est conforme aux normes XML Query, XML Schema et XML pour les opérations susmentionnées.

### 7.1 Enregistrement et validation XML Schema

Pour pouvoir utiliser les schémas XML et les définitions de type de document pour valider des documents, ils doivent préalablement être enregistrés dans la base de données. En cas de validation, la base de données repose sur les schémas XML, stocke des documents de type annoté sur le disque et compile des plans d'exécution avec des références à d'autres schémas XML. Un accès à la fois stable et très performant aux schémas s'avère par ailleurs nécessaire pour optimiser la validation lors d'opérations d'insertion, de mise à jour ou d'interrogation XML. Pour répondre à ces exigences en matière de stabilité et de performances, il n'existe pas d'autre choix que celui de stocker les schémas dans la base de données elle-même. DB2 propose par voie de conséquence un référentiel XML Schema (XSR).

Le référentiel de schémas comporte en son sein plusieurs nouvelles tables de catalogues de bases de données. Ces tables stockent les documents des schémas XML d'origine qui comprennent un schéma XML, ainsi qu'une « représentation binaire » du schéma pour accélérer les références lors de la validation d'un document.

L'enregistrement des schémas XML s'effectue à l'aide de commandes DB2, de procédures stockées ou d'interfaces API propres à un langage. L'enregistrement d'un schéma simple est donné en exemple ci-dessous. L'identificateur URI du schéma est « http://my.dept.com », le fichier qui contient le document du schéma est « dept.xsd », l'identificateur du schéma dans la base de données est « deptschema » et il appartient au schéma de base de données relationnelle « departments ». Notez que l'identificateur URI de l'espace de noms est déduit du document du schéma proprement dit.

```
register xmlschema http://my.dept.com
        from dept.xsd
        as departments.deptschema complete
```

Les documents peuvent être validés dans des instructions SQL avec la fonction XMLVALIDATE. Le schéma à utiliser pour la validation peut être soit explicitement spécifié soit déduit des paramètres schemaLocation dans les documents de l'instance. Un schéma peut être explicitement désigné par son identificateur URI de schéma ou par son identificateur de schéma. Dans l'exemple ci-dessous, deux instructions insert valident le document de base par rapport

au « deptschema » précédemment enregistré. Ces deux instructions spécifient explicitement le schéma, respectivement par l'identificateur URI du schéma et l'identificateur du schéma.

```
insert into dept(deptdoc) values xmlvalidate(? according to
xmlschema uri 'http://my.dept.com')
```

```
insert into dept(deptdoc) values xmlvalidate(? according to
xmlschema id departments.deptschema)
```

Ces instructions indiquent clairement que la validation XML Schema dans DB2 s'effectue document par document et non colonne par colonne. Chaque document inséré est susceptible d'être validé par rapport à un schéma XML différent, ce qui prouve la flexibilité du magasin XML de DB2. Cette flexibilité s'avère nécessaire pour les applications « orientées documents » où l'organisation et la classification des documents revêtent plus d'importance que leur homogénéité.

Dans l'exemple d'insertion ci-dessous, aucun schéma n'est explicitement désigné. Dans ce cas, DB2 tente de déduire le schéma du document de base et de le trouver dans le référentiel.

```
insert into dept(deptdoc) values xmlvalidate(?)
```

Les documents qui comportent des définitions de type de document ou des entités externes et/ou s'y réfèrent peuvent également faire l'objet d'une insertion, mais la définition de type de document permet uniquement de résoudre les références aux entités et d'ajouter des éléments et des attributs par défaut.

### 7.2 Évolution et flexibilité de XML Schema

Le référentiel de schémas de DB2 est essentiellement conçu selon deux principes. Le premier principe est le suivant : le référentiel ne doit pas obliger et n'obligera pas les utilisateurs à modifier un schéma préalablement à son enregistrement ni à modifier des documents XML préalablement à leur insertion et à leur validation. Une fois les documents insérés et validés, ils ne doivent par ailleurs jamais être invalidés ni requérir de mises à jour pour rester valides. Comme les applications XML traitent souvent un grand nombre de documents, les mises à jour en bloc destinées à les rendre conformes à un schéma non compatible sont quasiment systématiquement irréalisables.

Le second principe reposant à la base du référentiel de schémas XML de DB2 consiste à autoriser l'évolution des schémas. L'évolution des schémas désigne une séquence de modifications apportées à un schéma XML au cours de sa durée de vie. En règle générale, ces modifications répondent à l'évolution des besoins des entreprises ou aux nouveaux besoins des entreprises. Par exemple, la modification ou le lancement de services, produits ou processus commerciaux peut se traduire par de nouvelles exigences en matière de gestion des informations. Le schéma XML peut de ce fait nécessiter des modifications.

L'évolution des schémas et son meilleur mode de mise en œuvre font largement l'objet d'un débat. Aucune norme n'est jusqu'ici prévue pour l'évolution des schémas. La pression exercée dans les entreprises force toutefois l'évolution des schémas et obligent les utilisateurs XML à trouver des modes de mise en œuvre. Heureusement, la plupart des applications ne nécessitent pas de solution au problème général de l'évolution des schémas ; elles maîtrisent au contraire suffisamment le problème pour faire en sorte que des solutions relativement simples restent possibles. La flexibilité du référentiel de schémas revêt par conséquent une importance suprême. Cela signifie concrètement que le référentiel de schémas de DB2 n'a pas besoin de l'espace de noms ni de l'identificateur URI de schéma de chacun des schémas enregistrés pour être unique, car l'utilisateur risque de ne pas en être maître. Or l'utilisateur est bel et bien le maître de l'identificateur de schéma propre à la base de données, qui doit être unique. Le référentiel de schémas n'impose par ailleurs aucun mode d'évolution de schémas spécifique.

DB2 intègre la prise en charge d'un type très simple mais très important d'évolution de schémas. Si le nouveau schéma est rétrocompatible par rapport à l'ancien schéma, l'ancien schéma peut alors être remplacé par le nouveau schéma dans le référentiel de schémas. Pour effectuer cette opération, DB2 vérifie que tous les éléments et les attributs pouvant exister dans l'ancien schéma possèdent les mêmes types nommés dans le nouveau schéma. Ce type d'évolution de schémas limite le type de modifications pouvant être apportées aux ajouts d'éléments et d'attributs facultatifs, mais il est simple et utile.

Concernant le problème général de l'évolution des schémas, il est possible d'autoriser la coexistence des anciens et des nouveaux schémas sous différents noms. Il est tout à fait possible de mélanger des documents conformes à l'ancien schéma avec des documents conformes au nouveau schéma dans la même colonne d'une table. Il est également possible d'écrire des requêtes par rapport à cette table, de manière à ne traiter que les documents conformes à l'ancien schéma, que les documents conformes au nouveau schéma ou encore les deux. Pour permettre à l'application d'effectuer des opérations plus compliquées en fonction de la version, DB2 propose une option qui identifie le schéma qui était utilisé pour valider un document particulier :

```
select deptid, xmlxsrobjectid(deptdoc) from dept
where deptid = “PR27”
```

Cette instruction retourne l'identificateur de schéma du schéma qui était utilisé pour la validation du document XML du service PR27.

## 8 Décomposition de schémas annotés

Même si le magasin XML natif de DB2 peut insérer et interroger tout document XML, la décomposition de documents XML en lignes et colonnes relationnelles reste dans certains cas logique. Dans certains cas d'emploi, le langage XML n'est utilisé que pour transporter des données vers la base de données, mais la structure XML ne présente plus aucun intérêt une fois que les données sont intégrées aux données relationnelles existantes. Si, par exemple, une application extrait toutes les données pertinentes d'un message de services Web et décompose ces données en tables, le message XML d'origine risque alors de devenir inutile.

La décomposition peut aussi s'avérer nécessaire, car de nombreux outils d'exploitation de données et outils décisionnels ne fonctionnent que dans le format relationnel des données. Les performances des requêtes sur des données relationnelles peuvent également être supérieures à celles des requêtes sur des données XML si le schéma est suffisamment simple.

DB2 propose un produit de décomposition évolué qui mappe les données XML à des tables relationnelles. Le processus de décomposition repose sur des annotations au sein du schéma XML, qui s'apparentent aux mappages de schémas annotés dans MS-SQL Server [11] et Oracle [13]. Ces annotations, qui sont ajoutées au schéma par l'utilisateur, décrivent les attributs et les éléments XML qui sont mappés à telles ou telles tables et colonnes.

DB2 automatise ce processus de décomposition en se servant à la base du schéma annoté. Un exemple d'annotation est donné ci-dessous. Lorsqu'un document est inséré et décomposé sur la base de cette partie de schéma annoté, la valeur de l'élément salary qui figure sous l'élément payroll est insérée dans la colonne salary de la table T. Dans DB2, les annotations de décomposition se trouvent dans leur propre espace de noms et elles utilisent le préfixe d'espace de noms db2-xdb.

```
<xsd:element name=”payroll" >
  <xsd :complexType>
      <xsd:sequence>
         <xsd :element name="salary" type="xsd:string"
                    db2-xdb:rowSet=”T” db2-
                    xdb:column="salary"/>
         <xsd :element name="bonus" type="xsd:i nteger"
                    db2-xdb:rowSet=”T” db2-
                    xdb:column="bonus" />
      </xsd :sequence>
  </xsd:complexType>
</xsd :element>
```

Les annotations permettent à l'utilisateur de contrôler dans le moindre détail le processus de décomposition :

- Les données peuvent être normalisées, leur blanc manipulé et les données peuvent être manipulées dans une expression ou tronquées préalablement à l'insertion.
- Les données peuvent faire l'objet d'une insertion sous condition : par exemple, seules les valeurs qui correspondent à certains critères doivent être décomposées dans des paires tables-colonnes.
- Les relations de clés externes peuvent faire l'objet d'une description.
- Un même élément ou attribut peut être inséré dans plusieurs paires tables-colonnes.
- Plusieurs éléments ou attributs peuvent être insérés dans la même paire table-colonne.

Comme le langage XML est un type de données de première classe dans DB2, la décomposition d'un document XML peut passer par l'insertion totale ou partielle du document sous la forme d'une valeur XML dans une colonne XML. Une application est ainsi en mesure de décomposer efficacement un document XML en plusieurs parties et de stocker uniquement les parties requises dans une ou plusieurs colonnes XML.

# 9 Prise en charge d'applications et d'interfaces API XML

DB2 a mis en place un nouveau type de colonne SQL dans la base de données : le type de données XML. Les applications peuvent lier divers types de données propres à un langage pour l'entrée et la sortie de paramètres ou de colonnes XML. Ces types de données propres à un langage permettent uniquement d'utiliser le langage XML en tant que type de caractères ou type binaire.

Pour utiliser le langage XML avec efficacité et transparence, de nouveaux types XML propres à un langage peuvent être ajoutés aux interfaces clientes existantes. Ces nouveaux types XML propres à un langage optimisent l'efficacité de la base de données qui est en mesure de proposer une interface API plus complète pour les applications. En rendant le langage XML explicite dans l'application, la base de données évite les conversions de pages de codes inutiles et/ou superflues. Les documents XML intègrent une déclaration de codage qui requiert uniquement la conversion de l'analyseur XML. Le fait d'éviter des conversions de pages de codes inutiles permet souvent d'optimiser les performances. La conversion d'un document XML sans adaptation minutieuse de la déclaration de codage XML présente par ailleurs un risque d'invalidation de ce document XML.

Toutes les principales interfaces de base de données assurent la prise en charge native du type XML, à savoir le traitement des données XML en tant que type de données XML et non en tant que type de caractères. Les interfaces de base de données JDBC, ODBC, ADO.NET et SQL imbriqué sont abordées ci-dessous.

## 9.1 JDBC

Des améliorations ont été apportées à l'interface de base de données JDBC de manière à rendre les données XML compatibles avec des chaînes, des tableaux d'octets et des flux de données. En d'autres termes, les colonnes XML et les paramètres XML peuvent être liés à des chaînes, à des tableaux d'octets et à des flux de données. IBM travaille à la normalisation d'un type XML JDBC. Un type XML propriétaire (com.ibm.db2.DB2Xml) est dans le même temps disponible, de sorte que l'application puisse migrer avec transparence vers le future type JDBC standard.

Cette interface DB2Xml possède un certain nombre de méthodes qui facilitent l'utilisation des données XML. Dans l'exemple ci-dessous, une « colonne » est récupérée sous la forme d'un objet DB2Xml. La méthode getDB2 String retourne ensuite la représentation sérialisée de la valeur XML (sans déclaration XML) sous la forme d'un objet String. getDB2XMLBinaryStream(“UTF-16”) retourne ensuite un flux binaire avec la valeur XML codée en UTF-16 et la déclaration XML correspondante.

```
com.ibm.db2.jcc.DB2Xml xml1 =
        (com.ibm.db2.jcc.DB2Xml) rs.getObject ("xml_stuff");
String s = xml1.getDB2String();
InputStream is = xml2.getDB2XMLBinaryStream(“UTF-1 6”);
```

## 9.2 ODBC

Des améliorations ont été apportées à l'interface de base de données ODBC de manière à prendre en charge le langage XML via un nouveau type XML : SQL_C_XML. Comme il n'existe toutefois pas de type XML natif dans C, ce type ne peut être utilisé que dans les appels de l'interface API ODBC pour marquer les valeurs XML comme étant typées XML. L'avantage est le suivant : comme le client et le serveur DB2 savent qu'il s'agit de données XML, ils évitent les conversions de pages de codes inutiles ou superflues. Un exemple d'insertion de données XML dans une colonne typée XML est donné ci-dessous :

```
char xmlBuf[1 0240]; // SQL_C_XML
SQLExecDirect( hStmt, "Insert into T values (?)", SQL_NTS );
    SQLBindParameter( hStmt, 1, SQL_PARAM_INPUT,
    SQL_C_XML, SQL_XML, xmlBuf, &xmlBufLen);
```

## 9.3 ADO.NET

L'objectif de la prise en charge de DB2 .NET est l'intégration optimale aux interfaces API .NET. Dans l'exemple ci-dessous, un document XML est extrait de DB2 et l'application peut utiliser l'interface .NET standard (XmlReader) pour manipuler le résultat.

```
DB2Command cmd = DB2Connection.CreateCommand();
cmd.CommandText = "select deptdoc from dept";
cmd.CommandType = CommandType.Text;
DB2DataReader dr = cmd.Execute();
dr.Read( );
// retrieve the column as an XML reader
XmlReader xml1 = dr.GetXmlReader( 0 );
```

## 9.4 SQL imbriqué

De nouvelles déclarations de variables hôtes ont été définies pour les types XML dans le cadre de la norme SQL. DB2 s'en sert dans sa mise en œuvre.

```
EXEC SQL BEGIN DECLARE;
SQL TYPE IS XML AS CLOB( 10K ) xmlBuf;
EXEC SQL END DECLARE SECTION;
EXEC SQL SELECT deptdoc INTO :xmlBuf from dept
                    where deptID = '001';
```

## 10 Outils et utilitaires XML

Les utilitaires standard de DB2 sont mis à niveau pour fonctionner avec le nouveau type XML. Par exemple, les données de type de colonne XML sont prises en charge par les fonctions de DB2 suivantes : sauvegarde & restauration et réplication de données à haute disponibilité dans le cadre de la reprise et de la tolérance aux pannes.

Les commandes IMPORT/EXPORT permettent d'insérer des données dans des tables de bases de données ou d'en extraire avec flexibilité. Une seule commande IMPORT peut alimenter toute combinaison de colonnes relationnelles et XML d'une table. L'utilitaire IMPORT peut lire et importer des documents XML à partir d'un nombre illimité de fichiers XML distincts figurant dans le système de fichiers. DB2 peut par ailleurs importer des documents XML qui sont concaténés dans un seul grand fichier d'entrée. De la même manière, l'utilitaire EXPORT peut écrire des documents XML dans des fichiers distincts ou les concaténer dans un seul fichier.

Les utilitaires IMPORT et EXPORT permettent de maîtriser avec finesse les options d'analyse et de validation XML. Ces options s'apparentent aux fonctions SQL/XML XMLParse et XMLValidate. La validation des documents reste facultative pendant l'importation. Si la validation est choisie, tous les documents importés peuvent être validés par rapport à un seul schéma ou des schémas peuvent être spécifiés document par document. Il est également possible de valider uniquement certains documents pendant l'importation. Lors de l'exportation de données XML, un fichier plat est écrit en plus des données XML. Ce fichier plat peut comporter des données relationnelles qui ont pu faire partie de l'exportation. Il contient également des références aux documents XML exportés. Un identificateur de schéma peut par ailleurs être inclus pour chaque document exporté ayant fait l'objet d'une validation au moment de l'insertion. La relation entre les documents et les schémas peut par conséquent être exportée avec les données réelles et être utilisée en vue d'une validation dans le cadre d'une réimportation dans une base de données.

La commande LOAD permet d'insérer les données avec rapidité. Des modifications ont été apportées à la commande LOAD de manière à optimiser le traitement des données XML (mise en parallèle de l'analyse XML, contournement direct du flux d'insertion moyen et formatage des pages d'écriture). L'analyse simultanée de plusieurs documents d'entrée a été abordée de manière à augmenter de façon significative les temps de chargement en bloc XML [10]. Là encore, la validation XML Schema reste facultative pendant le chargement.

XQuery est un langage d'interrogation fonctionnel qui permet d'interroger des sources de données XML, y compris les colonnes XML. Les utilisateurs débutants risquent de trouver ce langage relativement complexe et non intuitif, même pour de simples requêtes. Pour résoudre ce problème, DB2 propose l'outil d'interface graphique XQuery Builder. XQuery Builder présente les fonctionnalités du langage XQuery sous forme d'ensembles de grilles. Par simple exploration en aval et glisser-déplacer, l'utilisateur peut concevoir des requêtes relativement complexes. L'outil interprète les actions de l'utilisateur sur l'interface graphique et génère les requêtes correspondantes en assistant considérablement l'utilisateur dans l'élaboration et la manipulation de la syntaxe XQuery.

## 11 Synthèse

Des améliorations ont été apportées à DB2 Universal Database® de manière à prendre intégralement en charge le langage XML *natif* et surmonter les difficultés inhérentes au mappage des tables ou des objets CLOB de données XML en données relationnelles. Les documents XML sont stockés sous forme d'arbres de type annoté sur des pages de disques, indexés avec des indexes propres à des chemins et interrogés avec XQuery, SQL/XML ou une combinaison des deux. La validation des schémas reste facultative et s'effectue document par document, assurant ainsi une certaine flexibilité et l'évolution des schémas. Les améliorations apportées aux principales interfaces API de bases de données proposent aux applications clientes les fonctionnalités nécessaires à l'exploitation des nouvelles capacités XML dans le serveur DB2. La prise en charge XML dans des utilitaires (import/export XML, par exemple) et dans un outil de conception visuelle XQuery vient renforcer la solution en langage XML natif proposée par DB2.

**Remerciements**

Nous tenons à saluer le travail effectué par de nombreux ingénieurs chez IBM Toronto Lab, IBM Silicon Valley Lab, IBM Almaden Research Center, IBM Portland Lab et IBM T.J. Watson Research Center et à les remercier pour leur participation à l'intégration de la prise en charge du langage XML natif dans DB2.

## Références

[1] Balmin, A. et al. : *A Framework for Using Materialized XPath Views in XML Query Processing,* VLDB, 2004, pages 60-71

[2] Beyer, K. et al. : *System RX: One Part Relational, One Part XML,* SIGMOD Conference, 2005

[3] Boag et al. : XQuery 1.0: An XML Query Language, February 2005, http://www.w3.org/TR/xquery

[4] Bourret, R. : *XML Database Products,* http://www.rpbourret.com/xml/XMLDatabaseProds.htm

[5] DeHaan et al. : *A Comprehensive XQuery to SQL Translation using Dynamic Interval Encoding,* Sigmod, 2003

[6] Eisenberg, Melton : *Advancements in SQL/XML,* ACM SIGMOD Record 33(3), pages 79-86, 2004

[7] Fiebig, T. et.al. : *Anatomy of a Native XML Base Management System,* VLDB Journal 11(4), décembre 2002

[8] Josifovski, V. et al. : *Querying XML Streams,* VLDB Journal, Vol. 14, N° 2, avril 2005

[9] Katz, H., (Editor) : *XQuery from the Experts,* Addison-Wesley, 2004

[10] Nicola, M. et al.: *XML Parsing, A Threat to Database Performance,* CIKM, 2003

[11] Oracle XML DB 10g www.oracle.com/technology/tech/xml/xmldb

[12] Pat et al. : *Indexing XML Data Stored in a Relational Database,* VLDB, 2004

[13] *SQLXML in MS SQL Server 2000*, http://msdn.microsoft.com/sqlxml

[14] *XML Efforts in Life Sciences and Bioinformatics*, http://www.xml.com/pub/rg/Bioinformatics