Module: XML et les bases de données

SQL Serveur 2008

http://msdn2.microsoft.com/en-us/library/ms186918.aspx

Houda Chabbi Drissi

houda.chabbi@hefr.ch

Source: livre Implémentation et maintenance de SQL serveur 2005. ISBN 2 10 050137 2
Présentation de Michael Rys
http://download.microsoft.com/
http://msdn.microsoft.com/library/

Plan

- Fiche technique
- Possibilité de stockage du XML
- Les extensions SQL pour gérer le XML ⇔relationnel
- L'indexation du typeXML

Fiche technique

- SQL Server 2008 is available on Microsoft Windows® operating systems.
- SQL Server 2008 is a **relational database**.
- SQL Server 2008 does **not** support the ANSI **SQL 2003 functions**.
- Subset of XQuery implemented
- One of the three most used databases on the market

Plan

- Fiche technique
- Possibilité de stockage du XML
  - ✓Stockage colonne texte
  - ✓Annexe: Stockage en tables relationnelles
  - ✓Stockage dans colonne de type XML:
  - ✓Typé (XSD) ou non Typé
- Les extensions SQL pour gérer le XML ⇔ relationnel
- L'indexation du typeXML

Evolution du support de XML

Relationnel → XML

- **SQL Server 2000:** the FOR XML (extension to the SELECT) and OPENXML XML → Relationnel
- **SQL Server 2005:** xml.nodes
  - ✓ the XML type (typed / untyped), XML → Relationnel(XMLType)
  - ✓Xquery (partial)
  - ✓ and FOR XML (Xpath…) Relationnel/XMLType → XML
- **SQL Server 2008:**
  - ✓Improved schema validation capabilities
  - ✓Enhancements to XQuery support
  - ✓Enhanced functionality for performing XML data manipulation language (DML) insertions

Vue synthétique du support de XML dans sqlserver 2008

Besoins	Moyens à disposition
XML → relationnel	OPENXML (sqlserver 2000) ~ XMLTable
XML → Type XML	xml.nodes (sqlserver 2005)
Relationnel → XML	FOR XML (extension to the SELECT) → Pas de standard sql2003 ~ SQL/XML
XMLType → XML	Xquery

Les différentes possibilités de stockage

- Stockage colonne texte
- Annexe: Stockage en tables relationnelles
- Stockage dans colonne de type XML:
  - Typé (XSD) ou non Typé

Stockage colonne type texte

- Stocker un XML dans une colonne teste à l'aide des types

(n)char, (n)varchar ou varbinary – avec stockage max de 2Go

```
DECLARE @myxml AS nvarchar(max)
Set @myxml ='<personne>H. Chabbi</personne>'
```

! Stockage monolithique sans aucune validation

nvarchar: (national varying character) le sont en Unicode (pour stocker du grecque fr etc.)
varchar: (varying character) le sont en texte simple

Annexe

Les différentes possibilités de stockage

- Stockage colonne texte
- Annexe: Stockage en tables relationnelles
- Stockage dans colonne de type XML:
  - Typé (XSD) ou non Typé

XML en relationnel

Contrôler le "shredding" du XML via: XML → Relationnel

- OpenXML depuis SQL Server 2000
- xml.nodes depuis SQL Server 2005

OpenXML

XML → Relationnel

Pointeur vers le DOM en mémoire du XML analysé

XPATH pour filter les nœuds à traités

```
OPENXML( idoc int [ in] , rowpattern nvarchar [ in ] ,
  [ flags byte [ in ] ] )
  [ WITH ( SchemaDeclaration | TableName ) ]
```

Le type de map à utiliser

Le nom et type de colonne SQL
(+ correspondance avec elt/attributs xml)

Le nom d'une table qui existe déjà
Avec le bon schéma pour le mapping

~ XMLTable

OpenXML: type de mapping

Déterminer par la valeur du flag:

Valeur flag	Description
0	Defaults to attribute-centric mapping: Chaque attribut est converti en une colonne
1	Use the attribute-centric mapping. Can be combined with XML_ELEMENTS. In this case, attribute-centric mapping is applied first, and then element-centric mapping is applied.
2	Use the element-centric mapping: Chaque élément est converti en une colonne. Can be combined with XML_ATTRIBUTES. In this case, attribute-centric mapping is applied first, and then element-centric mapping is applied

Manque le flag 8: Can be combined (logical OR) with XML_ATTRIBUTES or XML_ELEMENTS.

OpenXML : exemples-mapping implicite

- Entrée XML: tous les éléments ont des attributs pas de texte

```
<Customer CustomerID="…" ContactName="…">
 <Order CustomerID="…" EmployeeID="…" OrderDate="…">
   <OrderDetail OrderID="…" ProductID="…" Quantity="…"/>
   <OrderDetail …/>
 </Order>
</Customer>
```

- Sortie tabulaire: correspond aux attributs de l'élément Customer

```
CustomerID   ContactName
----------   -------------------
```

```
<Customer CustomerID="…" ContactName="…">
```

Variable utile pour plus tard

```
DECLARE @idoc int
DECLARE @doc varchar(1000)
SET @doc ='
<ROOT>
<Customer CustomerID="VINET" ContactName="Paul Henriot">
   <Order CustomerID="VINET" EmployeeID="5" OrderDate="1996-07-
    04T00:00:00">
      <OrderDetail OrderID="10248" ProductID="11" Quantity="12"/>
      <OrderDetail OrderID="10248" ProductID="42" Quantity="10"/>
   </Order>
</Customer>
<Customer CustomerID="LILAS" ContactName="Carlos Gonzlez">
   <Order CustomerID="LILAS" EmployeeID="3" OrderDate="1996-08-
    16T00:00:00">
      <OrderDetail OrderID="10283" ProductID="72" Quantity="3"/>
   </Order>
</Customer>
</ROOT>'
```

2 éléments Customer

Démarche

1. XML → DOM:
   - created by using sp_xml_preparedocument.
2. A SELECT + OPENXML is then executed against the internal representation of the XML document. The OPENXML specifies:
   - The *flag* value (here 1). This indicates attribute-centric mapping: the XML attributes map to the columns in the rowset (here column name matches the XML attribute names otherwise use the optional *ColPattern* (column pattern) parameter)
   - The *rowpattern* specified as /ROOT/Customer identifies the <Customers> nodes to be processed.

PS qui crée le DOM en mémoire du XML analysé

```
--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc

-- Execute a SELECT statement that uses the OPENXML rowset provider.
SELECT    *
FROM      OPENXML (@idoc, '/ROOT/Customer',1)
          WITH (CustomerID  varchar(10),
                ContactName varchar(20))
-- free the memory
EXEC sp_xml_removedocument @idoc
```

This indicates attribute-centric mapping

```
CustomerID ContactName
---------- --------------------
VINET      Paul Henriot
LILAS      Carlos Gonzlez
```

The OPENXML rowset provider creates a two-column rowset (CustomerID and ContactName) from which the SELECT statement retrieves the necessary columns (in this case, all the columns).

```
--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- Execute a SELECT statement that uses the OPENXML rowset provider.
SELECT    *
FROM      OPENXML (@idoc, '/ROOT/Customer',2)
          WITH (CustomerID  varchar(10),
                ContactName varchar(20))
```

```
CustomerID ContactName
---------- --------------------
NULL       NULL
NULL       NULL
```

This indicates element-centric mapping, the values of CustomerID and ContactName for both of the customers in the XML document are returned as NULL, because the <Customers> elements do not have any subelements CustomerID and Contact Name

OpenXML : exemples-mapping explicite

- Entrée XML: tous les éléments ont des attributs pas de texte

```
<Customer CustomerID="…" ContactName="…">
 <Order CustomerID="…" EmployeeID="…" OrderDate="…">
   <OrderDetail OrderID="…" ProductID="…" Quantity="…"/>
   <OrderDetail …/>
 </Order>
</Customer>
```

- Sortie tabulaire voulue:

```
OrderID CustomerID    OrderDate       ProdID    Qty
------- ------------- --------------- --------- ------
```

```
--Create an internal representation of the XML document.
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
-- SELECT stmt using OPENXML rowset provider
SELECT *
FROM   OPENXML (@idoc, '/ROOT/Customer/Order/OrderDetail',2)
       WITH (OrderID int                    '../@OrderID',
          CustomerID  varchar(10)     '../@CustomerID',
          OrderDate   datetime         '../@OrderDate',
          ProdID      int                   '@ProductID',
          Qty         int                   '@Quantity')
```

This indicates element-centric mapping,

This specifies the *ColPattern*

```
OrderID CustomerID  OrderDate        ProdID   Qty
---------- -------------- ---------------- --------- -------
10248     VINET        1996-07-04    11        12
10248     VINET        1996-07-04    42        10
10283     LILAS        1996-08-16    72         3
```

XML en relationnel

Contrôler le “shreeding” du XML via:

XML→Relationnel

- OpenXML depuis SQL Server 2000
- xml.nodes depuis SQL Server 2005

Méthode nodes

Décompose un type XML en données tabulaire sans passer par (une chaine + OpenXML)

✓meilleure performance

Input: un type Xml

Output: Une table à 1 colonne de type XML

nodes (XQuery) as Table(Column)

Doit retourner des nœuds et non des valeurs atomiques

Nom de la table de retour avec le nom de sa colonne

```
DECLARE @x xml
SET @x='<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>'
SELECT T.c.query('.') AS result
FROM   @x.nodes('/Root/row') T(c)
GO
```

Nom de la table de retour avec le nom de sa colonne

```
<row id="1"><name>Larry</name><oflw>some text</oflw></row>
<row id="2"><name>moe</name></row>
<row id="3"/>
```

Principe SQL: « 1 ligne en entrée 1 ligne en sortie ». Ici « 1 ligne en entrée plusieurs lignes en sortie » ☹ Va poser problème si utilisation sur une table donc +sieurs lignes où chacune contient un XML qui peut retourner plusieurs lignes!

☺ Utilisation du CROSS APPLY ou OUTER APPLY

```
DECLARE @x xml
SET @x='<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>'
SELECT T.c.query('..') AS result
FROM   @x.nodes('/Root/row') T(c)
```

```
<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>
<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>
<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>
```

```
DECLARE @x xml
SET @x='<Root>
   <row id="1"><name>Larry</name><oflw>some text</oflw></row>
   <row id="2"><name>moe</name></row>
   <row id="3" />
</Root>'
SELECT c.value('@id','int') AS ID,
       c.query('./name') AS Nom
FROM   @x.nodes('/Root/row') T(c)
GO
```

```
ID    Nom
-----  --------------------------------
1     <name>Larry</name>
2     <name>moe</name>
3
```

Ici on peut faire un insert dans une table ☺

Appliquer nodes() aux colonnes d’une tables: problème

```
CREATE TABLE T(C1 XML);

Select C1.nodes('Xquery') FROM T

Select * from T.C1.nodes('Xquery')
```

☹ Posent problème car +sieurs lignes par ligne de T

Solution: L’opérateur apply permet d’invoquer une fonction pour chaque ligne renvoyée de la requête.

Annexe: opérateur Apply

Nouveau sql2005

Syntaxe:

```
SELECT col1, col2…
FROM table_externe Te
CROSS APPLY|OUTER APPLY Fonction_qui_retourne_1_table
        (Te.col_jointure)
```

Comme nodes()

Fonctionnement:

L'opérateur APPLY permet d'appeler une fonction table pour chaque ligne retournée par l'expression de table_externe d'une requête. La fonction table agit en tant qu'entrée droite et l'expression de table externe en tant qu'entrée gauche. L'entrée droite est évaluée pour chaque ligne de l'entrée gauche, les lignes produites étant combinées dans l'entrée finale.

Fait partie de la famille des jointures

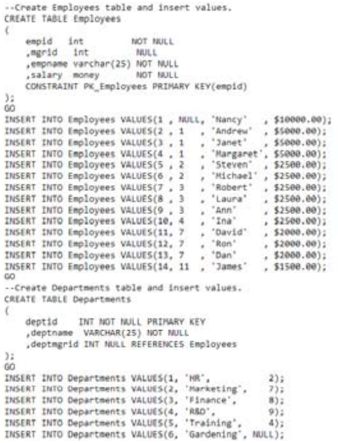
Annexe: opérateur Apply

```
CREATE TABLE Employees (
  empid int NOT NULL ,
  mgrid int NULL ,
  empname varchar(25) NOT NULL ,
  salary money NOT NULL PRIMARY KEY(empid) );

CREATE TABLE Departments (
  deptid INT NOT NULL PRIMARY KEY ,
  deptname VARCHAR(25) NOT NULL ,
  deptmgrid INT NULL REFERENCES Employees );

CREATE FUNCTION dbo.fn_getsubtree(@empid AS INT) RETURNS
  @TREE TABLE
```

A un num de manager donne tous les employés qu’il manage (transitivité comprise)



```
--Create Employees table and insert values.
CREATE TABLE Employees
(
    empid   int          NOT NULL
    ,mgrid   int          NULL
    ,empname varchar(25) NOT NULL
    ,salary  money        NOT NULL
    CONSTRAINT PK_Employees PRIMARY KEY(empid)
);
GO
INSERT INTO Employees VALUES(1 , NULL, 'Nancy'    , $10000.00);
INSERT INTO Employees VALUES(2 , 1    , 'Andrew'   , $5000.00);
INSERT INTO Employees VALUES(3 , 1    , 'Janet'    , $5000.00);
INSERT INTO Employees VALUES(4 , 1    , 'Margaret', $5000.00);
INSERT INTO Employees VALUES(5 , 2    , 'Steven'   , $2500.00);
INSERT INTO Employees VALUES(6 , 2    , 'Michael'  , $2500.00);
INSERT INTO Employees VALUES(7 , 3    , 'Robert'   , $2500.00);
INSERT INTO Employees VALUES(8 , 3    , 'Laura'    , $2500.00);
INSERT INTO Employees VALUES(9 , 3    , 'Ann'      , $2500.00);
INSERT INTO Employees VALUES(10, 4    , 'Ina'      , $2500.00);
INSERT INTO Employees VALUES(11, 7    , 'David'    , $2000.00);
INSERT INTO Employees VALUES(12, 7    , 'Ron'      , $2000.00);
INSERT INTO Employees VALUES(13, 7    , 'Dan'      , $2000.00);
INSERT INTO Employees VALUES(14, 11   , 'James'    , $1500.00);
GO
--Create Departments table and insert values.
CREATE TABLE Departments
(
    deptid    INT NOT NULL PRIMARY KEY
    ,deptname  VARCHAR(25) NOT NULL
    ,deptmgrid INT NULL REFERENCES Employees
);
GO
INSERT INTO Departments VALUES(1, 'HR',          2);
INSERT INTO Departments VALUES(2, 'Marketing',   7);
INSERT INTO Departments VALUES(3, 'Finance',     8);
INSERT INTO Departments VALUES(4, 'R&D',         9);
INSERT INTO Departments VALUES(5, 'Training',    4);
INSERT INTO Departments VALUES(6, 'Gardening', NULL);
```

Annexe

Annexe: opérateur Apply

```
CREATE FUNCTION dbo.fn_getsubtree(@empid AS INT)
  RETURNS @TREE TABLE
```

Retourne 1 table pour 1 id

```
empid      empname   mgrid   lvl
----------- ---------- --------- ---
```

On veut retourner la liste des hiérarchies pour chacun des responsables des départements:

```
deptid      deptname  deptmgrid  empid       empname mgrid       lvl
----------- ---------- ----------- ----------- ---------- ----------- ---
```

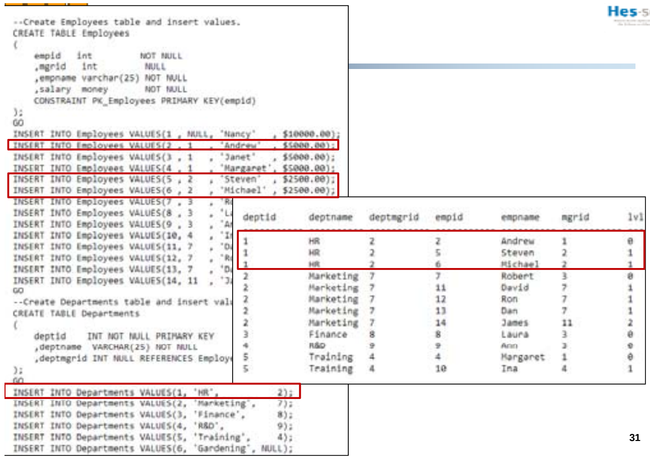
Appliquer la fonction fn_getsubtree(@empid AS INT) à chaque entrée deptmgrid de la table Departments.

Annexe

Annexe: opérateur Apply: solution

```
SELECT D.deptid, D.deptname, D.deptmgrid,
          ST.empid, ST.empname, ST.mgrid
FROM Departments AS D
          CROSS APPLY
          fn_getsubtree(D.deptmgrid) AS ST;
```

A chaque entrée de Departments applique la fonction sur l'deptmgrid retourne les lignes et les met dans la table de sortie.



```
--Create Employees table and insert values.
CREATE TABLE Employees
(
    empid   int          NOT NULL
    ,mgrid   int          NULL
    ,empname varchar(25) NOT NULL
    ,salary  money        NOT NULL
    CONSTRAINT PK_Employees PRIMARY KEY(empid)
);
GO
INSERT INTO Employees VALUES(1 , NULL, 'Nancy'    , $10000.00);
INSERT INTO Employees VALUES(2 , 1    , 'Andrew'   , $5000.00);
INSERT INTO Employees VALUES(3 , 1    , 'Janet'    , $5000.00);
INSERT INTO Employees VALUES(4 , 1    , 'Margaret', $5000.00);
INSERT INTO Employees VALUES(5 , 2    , 'Steven'   , $2500.00);
INSERT INTO Employees VALUES(6 , 2    , 'Michael'  , $2500.00);
INSERT INTO Employees VALUES(7 , 3    , 'Ro
INSERT INTO Employees VALUES(8 , 3    , 'La
INSERT INTO Employees VALUES(9 , 3    , 'An
INSERT INTO Employees VALUES(10, 4    , 'In
INSERT INTO Employees VALUES(11, 7    , 'Da
INSERT INTO Employees VALUES(12, 7    , 'Ro
INSERT INTO Employees VALUES(13, 7    , 'Da
INSERT INTO Employees VALUES(14, 11   , 'Ja
GO
--Create Departments table and insert val
CREATE TABLE Departments
(
    deptid    INT NOT NULL PRIMARY KEY
    ,deptname  VARCHAR(25) NOT NULL
    ,deptmgrid INT NULL REFERENCES Employe
);
GO
INSERT INTO Departments VALUES(1, 'HR',          2);
INSERT INTO Departments VALUES(2, 'Marketing',   7);
INSERT INTO Departments VALUES(3, 'Finance',     8);
INSERT INTO Departments VALUES(4, 'R&D',         9);
INSERT INTO Departments VALUES(5, 'Training',    4);
INSERT INTO Departments VALUES(6, 'Gardening', NULL);
```

```
deptid      deptname   deptmgrid   empid       empname    mgrid       lvl
----------- ---------- ----------- ----------- ---------- ----------- ---
1           HR         2           2           Andrew     1           0
1           HR         2           5           Steven     2           1
1           HR         2           6           Michael    2           1
2           Marketing  7           7           Robert     3           0
2           Marketing  7           11          David      7           1
2           Marketing  7           12          Ron        7           1
2           Marketing  7           13          Dan        7           1
2           Marketing  7           14          James      11          2
3           Finance    8           8           Laura      3           0
4           R&D        9           9           Ann        3           0
5           Training   4           4           Margaret   1           0
5           Training   4           10          Ina        4           1
```

Annexe

Appliquer nodes() aux colonnes d'une tables: solution

```
CREATE TABLE T(C1 XML);


SELECT T2.C2.query('.')
FROM   T
CROSS APPLY T.C1.nodes('blabla_xquery') as T2(C2)
```

☺ Parfait

Les différentes possibilités de stockage

- Stockage colonne texte
- Annexe: Stockage en tables relationnelles
- Stockage dans colonne de type XML:
  - Typé (XSD) ou non Typé

Microsoft SQL Server 2008 - XML

- Native column type for XML storage
  - Binary XML – to represent XML documents and fragments
- Flexible schema support
  - XML documents are either typed (associated with a schema) or un-typed (not associated with any schema)
- Single query system:
  - Support augmented SQL with XQuery
  - Support most (but not all) of the XQuery functions
  - Tries to map as many operations from XPath and XQuery to the relational model for faster execution

XML type

- XMLType:
  - Untyped
  - Typed needs an XSD
- XML indexing:
  - Primary XML indexes
    - ✓Contains the data model content of the XML nodes
  - Secondary XML indexes
    - ✓PATH indexes: for path-based queries
    - ✓PROPERTY indexes: for property bag scenarios
    - ✓VALUE indexes: for value-based queries

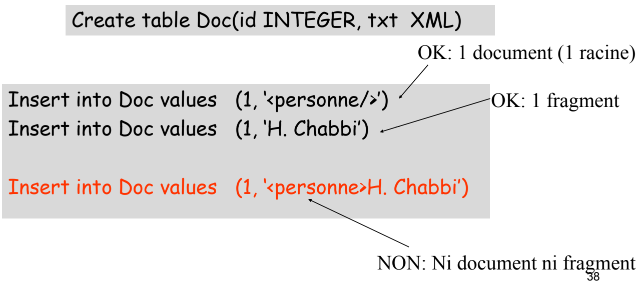
Type XML: limites

- L'ordre et la structure des documents sont préservés
- La fidélité textuelle n'est pas préservée.
  - Les espaces, la déclaration XML, les commentaires dans le XML, l'ordre des attributs sont supprimées.
- La profondeur de noeuds maximales est de 128
- La taille maximale est de 2GO

Un-typed = no XSD

- No schema collection name is associated to the xml column
- The XML is stored as a simple character string:
  - This is less efficient,
  - but does maintain the complete original content of the XML document (such as comments, etc.).
  - the inserted XML must be well-formed.

- Accepte des fragments ou des documents XML



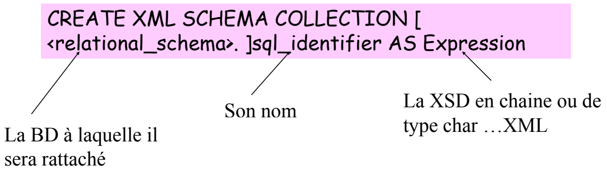
```
Create table Doc(id INTEGER, txt  XML)
```

```
Insert into Doc values   (1, '<personne/>')
Insert into Doc values   (1, 'H. Chabbi')

Insert into Doc values   (1, '<personne>H. Chabbi')
```

Type XML = XSD

- Permet la validation des données stockées
- Les XSD doivent être enregistrés dans la base



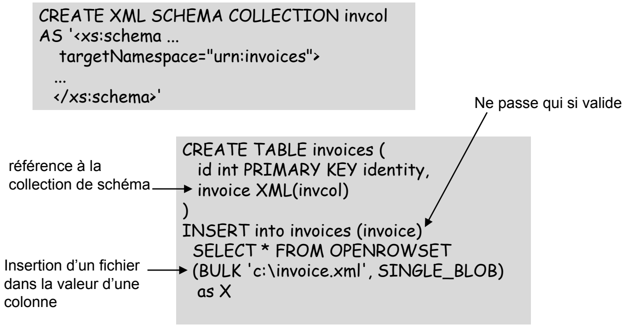
```
CREATE XML SCHEMA COLLECTION [
<relational_schema>. ]sql_identifier AS Expression
```

La BD à laquelle il sera rattaché
Son nom
La XSD en chaine ou de type char …XML

```
CREATE TABLE MyTable(MyKey int,
                     MyXml xml(MyNewSchemaCol))
```

- The inserted XML document into that column will be shredded automatically into its individual data items
- When queried the column, SQL Server automatically reconstructs the XML document into its original form. Note, however, that this will not include things like comments that are not part of the original data content of the document.

Utilisation des collections de XSD



```
CREATE XML SCHEMA COLLECTION invcol
AS '<xs:schema ...
   targetNamespace="urn:invoices">
   ...
  </xs:schema>'
```

```
CREATE TABLE invoices (
  id int PRIMARY KEY identity,
  invoice XML(invcol)
)
INSERT into invoices (invoice)
  SELECT * FROM OPENROWSET
  (BULK 'c:\invoice.xml', SINGLE_BLOB)
   as X
```

Ne passe qui si valide
référence à la collection de schéma
Insertion d'un fichier dans la valeur d'une colonne

Les collections XSD

Peuvent contenir plus d'une XSD
- Inclure une nouvelle XSD:

```
ALTER XML SCHEMA COLLECTION MaCollection1 ADD '<?xml
version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"  >
  <xsd:element name="elt2">
    <xsd:complexType>
      …
    </xsd:complexType>
  </xsd:element>
</xsd:schema>'
```

La validation se fera avec les 2 XSD

Synthèse sur les différents types de stockage

- XML en text
  - Stockage/manipulation en bloc
- XML en tables relationnelles
  - Utilisation des techniques relationnelles
- XML en type natif XML
  - Utilisation des technologies XML

Plan

- Fiche technique
- Possibilité de stockage du XML
- Les extensions SQL pour gérer le XML ⇔ relationnel
- L'indexation du typeXML

XML ⇔ Relationnel

XML → Relationnel

- **OpenXML** et les procédures stockées
- Méthode **nodes()** du type XML et ses opérateurs *apply*

XML ← Relationnel

- **FOR XML (RAW, AUTO, PATH, EXPLICIT)**

XML ← Relationnel (XML)

- **Méthodes du XMLType**

FOR XML

Relationnel → XML

Extension du SQL

```
SELECT…
FROM…
WHERE…
ORDER BY…
FOR XML (raw [, ELEMENTS] |
         auto [, ELEMENTS] |
         path |
         explicit) [ROOT('blabla')]
    [, XMLData]
    [, BINARY base64])
```

~ SQL/XML

FOR XML – Raw Mode

Relationnel → XML

- One <row> element per row in the result set. A fragment no root.
- No nested elements
- Columns/values in result set are attributes/value on the <row>
- It is possible to
  - Rename the attributes and the row tags (alias)
  - To get Columns/values set to element/value

Récupération basique!

~ XSU(oracle)

```
SELECT C.CustomerID, OrderID
FROM Customers C, Orders O
WHERE C.CustomerID = O.CustomerID
FOR XML raw
```

```
<row CustomerID="ALFKI" OrderID="10643"/>
<row CustomerID="ALFKI" OrderID="10643"/>
<row CustomerID="ANATR" OrderID="10308"/>
…
<row CustomerID="MAISD" OrderID="11004"/>
```

1 fragment

To get Columns/values set to element/value

```
SELECT CustomerID, OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML raw, ELEMENTS
```

```
<row> <CustomerID>ALFKI</CustomerID>
  <OrderID>10643</OrderID> </row>
…
<row> <CustomerID>MAISD</CustomerID>
  <OrderID>11004</OrderID> </row>
```

1 fragment

Rename the attributes and the row tags (alias)

```
SELECT CustomerID as Cid, OrderID as Oid
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML raw('ligne'), ELEMENTS
```

```
<ligne> <Cid>ALFKI</Cid> <Oid>10643</Oid> </ligne>
…
<ligne> <Cid>MAISD</Cid> <Oid>11004</Oid> </ligne>
```

1 fragment

```
SELECT CustomerID as Cid, OrderID as Oid
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML raw('ligne'), ELEMENTS, ROOT('Resultat')
```

Crée un document: ajoute une racine

```
<resultat>
  <ligne> <Cid>ALFKI</Cid> <Oid>10643</Oid> </ligne>
  …
  <ligne> <Cid>MAISD</Cid> <Oid>11004</Oid> </ligne>
</resultat>
```

1 document

FOR XML – Auto Mode

- Columns/values in result set are attributes/value
  - ELEMENTS directive produces sub elements instead
- Supports nested XML output
  - Nesting determined by ordering of columns in SELECT clause
  - Sibling relationships not supported
- Change names using table and column aliases

```
SELECT Customers.CustomerID, OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto
```

```
<Customers CustomerID="ALFKI">
  <Orders OrderID="10643"/>
  <Orders OrderID="10642"/>
</Customers>
<Customers CustomerID="ANATR">
  <Orders OrderID="10308"/>
…
```

FOR XML – Auto Mode: fonctionnement

XML Auto génère un nouveau niveau de hiérarchie, pour chaque table de la requête select, construit dans l'ordre suivant:

1. Le 1er niveau correspond à la table à qui appartient la 1ere colonne déclarée dans le select. Le 2ieme niveau correspond à la table de la colonne suivante etc.
2. Si des colonnes sont mélangées dans la requête select, XML Auto modifie l'ordre des nœuds XML de façon à ce que tous les nœuds qui appartiennent au même niveau soient regroupés sous le même nœud parent.

```
SELECT Customers.CustomerID, OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto
```

```
<Customers CustomerID="ALFKI">
  <Orders OrderID="10643"/>
   <Orders OrderID="10642"/>
</Customers>
<Customers CustomerID="ANATR">
  <Orders OrderID="10308"/>
…
```

```
SELECT OrderID, Customers.CustomerID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto
```

```
<Orders OrderID="10643">
  <Customers CustomerID="ALFKI"/>
</Orders>
<Orders OrderID="10642">
  <Customers CustomerID="ALFKI"/>
</Orders>
<Orders OrderID="10308">
  <Customers CustomerID="ANATR"/>
</Orders>
…
```

```
SELECT Customers.CustomerID, OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto
```

```
<Customers CustomerID="ALFKI">
  <Orders OrderID="10643"/>
   <Orders OrderID="10642"/>
</Customers>
<Customers CustomerID="ANATR">
  <Orders OrderID="10308"/>
…
```

```
SELECT Customers.CustomerID, OrderID
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto, ELEMENTS
```

```
<Customers>
  <CustomerID>ALFKI</CustomerID>
  <Orders>
     <OrderID>10643</OrderID>
  </Orders>
   <Orders>
     <OrderID>10642</OrderID>
  </Orders>
</Customers>
<Customers>
…
```

```
SELECT C.CustomerID, OrderID as oid
FROM Customers C, Orders O
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML auto, ELEMENTS
```

```
<C>
  <CustomerID>ALFKI</CustomerID>
  <O>
    <oid>10643</oid>
  </O>
   <O>
    <oid>10642</oid>
  </O>
</C>
<C>
…
```

FOR XML – XPATH Mode

Nouveau sql2005

- Contrôle complet sur le mode de génération
- Plusieurs options pour créer des:
  - Attributs/éléments, des commentaires, du contenu de nœuds ou attributs etc.
- Chaque colonne peut être configurée individuellement.

```
SELECT Customers.CustomerID '@Cid',
             OrderID 'comment()'
FROM Customers, Orders
WHERE Customers.CustomerID = Orders.CustomerID
FOR XML path ('ligne')
```

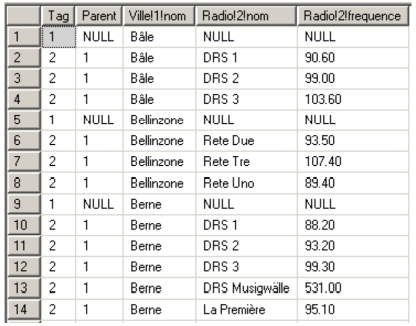
```
<ligne Cid="ALFKI">
  <!- 10643 -->
</ligne>
<ligne Cid="ALFKI">
  <!- 10642 -->
</ligne>
<ligne>
…
```

FOR XML – Explicit Mode

- Degré de contrôle le plus élevé pour la génération de structures XML complexes
- Le résultat du select doit respecter le modèle de table universelle

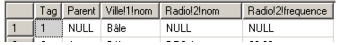
~ SQL/XML

La table universelle exemple



	Tag	Parent	Ville!1!nom	Radio!2!nom	Radio!2!frequence
1	1	NULL	Bâle	NULL	NULL
2	2	1	Bâle	DRS 1	90.60
3	2	1	Bâle	DRS 2	99.00
4	2	1	Bâle	DRS 3	103.60
5	1	NULL	Bellinzone	NULL	NULL
6	2	1	Bellinzone	Rete Due	93.50
7	2	1	Bellinzone	Rete Tre	107.40
8	2	1	Bellinzone	Rete Uno	89.40
9	1	NULL	Berne	NULL	NULL
10	2	1	Berne	DRS 1	88.20
11	2	1	Berne	DRS 2	93.20
12	2	1	Berne	DRS 3	99.30
13	2	1	Berne	DRS Musigwälle	531.00
14	2	1	Berne	La Première	95.10

Le modèle de table universelle

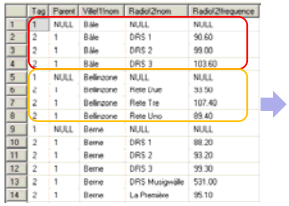


	Tag	Parent	Ville!1!nom	Radio!2!nom	Radio!2!frequence
1	1	NULL	Bâle	NULL	NULL

- **Colonne** Tag: 1ère colonne du jeu de résultats. Elle indique la profondeur de la structure XML, à partir de 1.
- **Colonne** **parent:** 2ième colonne. Elle indique le numéro de balise du nœud parent dans la structure XML.
- **Colonnes suivantes** doivent avoir un alias qui respecte le modèle suivant: **NomElement!NumBalise!NomAttribut!Directive**.
  - **NomElement** le nom de l'élément
  - **NumBalise** le niveau (d'après la colonne tag) auquel doit être placé ce noeud
  - **NomAttribut** facultatif
  - **Directive** facultatif. Plusieurs options. Hide par exemple signifie colonne qui est nécessaire uniquement au tri. Element passe les attributs en élément.

Attention les lignes résultats doivent être dans un ordre spécifique. Les lignes doivent être triées de sorte que chaque nœud parent soit suivi de ses nœuds enfants

Le format table universelle: signification

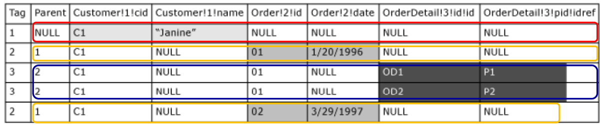


	Tag	Parent	Ville!1!nom	Radio!2!nom	Radio!2!frequence
1	1	NULL	Bâle	NULL	NULL
2	2	1	Bâle	DRS 1	90.60
3	2	1	Bâle	DRS 2	99.00
4	2	1	Bâle	DRS 3	103.60
5	1	NULL	Bellinzone	NULL	NULL
6	2	1	Bellinzone	Rete Due	93.50
7	2	1	Bellinzone	Rete Tre	107.40
8	2	1	Bellinzone	Rete Uno	89.40
9	1	NULL	Berne	NULL	NULL
10	2	1	Berne	DRS 1	88.20
11	2	1	Berne	DRS 2	93.20
12	2	1	Berne	DRS 3	99.30
13	2	1	Berne	DRS Musigwälle	531.00
14	2	1	Berne	La Première	95.10

```
<Ville nom="Bâle">
  <Radio nom="DRS 1" frequence="90.60" />
  <Radio nom="DRS 2" frequence="99.00" />
  <Radio nom="DRS 3" frequence="103.60" />
</Ville>
<Ville nom="Bellinzone">
  <Radio nom="Rete Due" frequence="93.50" />
  <Radio nom="Rete Tre" frequence="107.40" />
  <Radio nom="Rete Uno" frequence="89.40" />
</Ville>
...
```

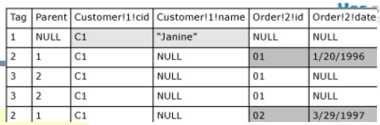
* Table universelle: exemple

(msdn)



Tag	Parent	Customer!1!cid	Customer!1!name	Order!2!id	Order!2!date	OrderDetail!3!id!id	OrderDetail!3!pid!idref
1	NULL	C1	"Janine"	NULL	NULL	NULL	NULL
2	1	C1	NULL	01	1/20/1996	NULL	NULL
3	2	C1	NULL	01	NULL	OD1	P1
3	2	C1	NULL	01	NULL	OD2	P2
2	1	C1	NULL	02	3/29/1997	NULL	NULL

```
<Customer cid="C1" name="Janine">
  <Order id="O1" date="1/20/1996">
    <OrderDetail id="OD1" pid="P1"/>
    <OrderDetail id="OD2" pid="P2"/>
  </Order>
  <Order id="O2" date="3/29/1997">
</Customer>
```



Tag	Parent	Customer!1!cid	Customer!1!name	Order!2!id	Order!2!date
1	NULL	C1	"Janine"	NULL	NULL
2	1	C1	NULL	01	1/20/1996
3	2	C1	NULL	01	NULL
3	2	C1	NULL	01	NULL
2	1	C1	NULL	02	3/29/1997

```
<Cmdes>
  <Customer id="ALFKI" name="Toto1" />
  <Customer id="ANATR" name="Toto2" />
  ...
</Cmdes>
```

~ On prépare la structure
et on la rempli avec les union ALL

```
SELECT 1 as Tag,
          NULL as Parent,
          Customer.CustomerID  as 'Customer!1!id',
          Customer.Cname  as 'Customer!1!name',
          NULL  as 'Order!2!id',
          NULL  as 'Order!2!qte'
FROM Customer
ORDER BY Customer.CustomerID
FOR XML EXPLICIT, ROOT('Cmdes')
```

```
SELECT 1 as Tag,
          NULL as Parent,
          Customer.CustomerID  as 'Customer!1!id',
          Customer.Cname  as 'Customer!1!name',
          NULL  as 'Order!2!id',
          NULL  as 'Order!2!qte'
FROM Customer
UNION ALL
SELECT 2 as Tag,
          1 as Parent,
     Pb   NULL, -- pour Customer.CustomerID
          NULL,
          Orderid,
          qte
FROM Customer, Orders
WHERE Customer.CustomerID = Orders.CustomerID
ORDER BY [Customer!1!id], [Order!2!id]
FOR XML EXPLICIT, ROOT('Cmdes')
go
```

- Msg 6833, Level 16, State 1, Line 1
- L'ID de balise parente 1 ne fait pas partie des balises ouvertes. FOR XML EXPLICIT nécessite que les balises parentes soient d'abord ouvertes. Vérifiez l'ordre de l'ensemble de résultats.

```
SELECT 1 as Tag,
          NULL as Parent,
          Customer.CustomerID  as 'Customer!1!id',
          Customer.Cname  as 'Customer!1!name',
          NULL  as 'Order!2!id',
          NULL  as 'Order!2!qte'
FROM Customer
UNION ALL
SELECT 2 as Tag,
          1 as Parent,
          Customer.CustomerID,  -- Néces. pour le tri !
          NULL,
          Orderid,
          qte
FROM Customer, Orders
WHERE Customer.CustomerID = Orders.CustomerID
ORDER BY [Customer!1!id], [Order!2!id]
FOR XML EXPLICIT, ROOT('Cmdes')
go
```

```
<Cmdes>
  <Customer id="ALFKI"
    name="Toto1">
    <Order id="10642" qte="2" />
    <Order id="10643" qte="1" />
  </Customer>
  <Customer id="ANATR"
    name="Toto2">
    <Order id="10308" qte="3" />
  ...
</Cmdes>
```

*FOR XML – Explicit Mode: directive Element

```
SELECT 1 as Tag,
         NULL as Parent,
         EmployeeID as [Employee!1!EmpID],
        NULL as [Name!2!FName!ELEMENT],
        NULL as [Name!2!LName!ELEMENT]
FROM  HumanResources.Employee E, Person.Contact C
WHERE E.ContactID = C.ContactID
UNION ALL
SELECT 2 as Tag,
        1 as Parent,
        EmployeeID,
        FirstName,
        LastName
FROM HumanResources.Employee E, Person.Contact C
WHERE E.ContactID = C.ContactID
ORDER BY
   [Employee!1!EmpID],[Name!2!FName!ELEMENT]
FOR XML EXPLICIT
```

```
<Employee EmpID=...>
  <Name>
    <FName>...</FName>
    <LName>...</LName>
  </Name>
</Employee>
```

XML ⇔Relationnel

- **OpenXML** et les procédures stockées
- Méthode **nodes()** du type XML et ses opérateurs *apply*

XML →Relationnel

XML ← Relationnel

- FOR XML (RAW, AUTO, PATH, EXPLICIT)

XML ← Relationnel (XML)

– Méthodes du XMLType

XML Methods of XML type

The new xml data type has methods:

- The query method, which returns a fragment of un-typed XML
- The value method, which returns a single value from the XML and exposes it as a standard (non-xml) SQL data type
- The exist method, which can be used to test whether a specific node/value exists in the XML data. Returns 1 if the XQuery expression returns at least one item, 0 otherwise
- The modify method, which executes an XML Data Modification Language (XML-DML) statement
- The nodes function, which returns a single-column rowset of nodes from the XML

The query, value, exist and modify methods can do this without having to extract the whole document.

~ SQL/XML

Xquery dans SQL2008

- SQL Server 2008 supports a subset of the XQuery language, that can be used to extract data from XML documents stored in both typed and un-typed xml columns.
- SQL Server 2005 does not support the let statement. SQL Server 2008 does.
- XQuery also implements a large selection of built-in functions. SQL Server 2008 implements the most common functions.

Exemple: un-typed column

An **un-typed** column named MyXml in a table named MyTable:

```
SELECT MyXml.query('/root/product[@id="304"]/name') FROM
   MyTable
```

```
<name>Dell D800</name>
```

```
SELECT MyXml.query('data(/root/product[@id="304"]/name)') FROM
   MyTable
```

```
Dell D800
```

Exemple: untyped column

This assumes that each row in the ProductList table is an XML document containing a list of products in a specific category (ProductGroup is the column containing the category number). It will returns an XML document such as

```
SELECT ProductXML.query('
<myproductlist>
{
  for $p in //product
  where data($p/@id) > 10
  order by $p/name[1]
  return $p/description
}
</myproductlist>')
FROM ProductList
WHERE ProductGroup = 3
```

```
<myproductlist>
  <description>blabla1</description>
  <description>blabla2</description>
</myproductlist>
```

Exemple: typed column

The namespace must be specified for the schema in the query (assign the namespace to a prefix, and use this prefix with each element in the query).

```
SELECT MyXml.query('
  declare namespace s="http://myns/mydemoschema";
 /s:root/s:product[@s:id="304"]/s:name') FROM MyTable
```

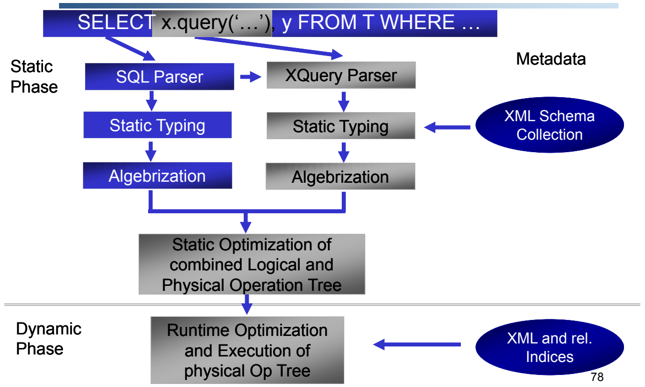
MyTable(MyXml XML)

Traitement du XQuery

XQuery operations are built into the SQL Server query engine itself to increase efficiency:

the SQL and XQuery parts of a SQL statement are folded into a single query plan, with optimization of the entire plan.

Combined SQL And XQuery/DML Processing


SELECT x.query('…'), y FROM T WHERE …
Static Phase
SQL Parser
XQuery Parser
Metadata
Static Typing
Static Typing
XML Schema Collection
Algebrization
Algebrization
Static Optimization of combined Logical and Physical Operation Tree
Dynamic Phase
Runtime Optimization and Execution of physical Op Tree
XML and rel. Indices

sql:column() / sql:variable()

Map SQL value and type into XQuery values and types in context of XQuery or XML-DML

- sql:variable(): accesses a SQL variable/parameter

```
declare @value int
set @value=42
select * from T
where T.x.exist('/a/b[@id=sql:variable("@value")]')=1
```

- sql:column(): accesses another column value

```
tables: T(key int, x xml), S(key int, val int)

select * from T join S on T.key=S.key
where T.x.exist('/a/b[@id=sql:column("S.val")]')=1
```

- Restrictions in SQL Server 2005:
  - No XML, CLR UDT, datetime, or deprecated text/ntext/image

Example: value Method

- The value method works much like the query method, except that it takes a second parameter that is the name of one of the SQL Server built-in data types.
- The value is returned as an instance of that type.
- The specified XPath must return a single node, so you should specify the node index

For example:

```
SELECT
  MyXml.value('(/root/product[@id="304"]/name)[1]',
  'nvarchar(30)')
FROM MyTable
```

MyTable(MyXml XML)

Example: exist Method

- The exist method takes an XPath expression that selects a single node within the XML document, and returns either True (bit value 1) if the node exists or False (bit value 0) if it does not. If the source column is a typed xml column and the element contains null, the method returns NULL instead.

```
SELECT MyXml.exist('(/root/product[@id="304"])[1]'
  FROM MyTable
```

will return

- True if there is a product element with the attribute id="304",
- or False if not.

- It can be used in the WHERE clause of a SQL statement:

```
SELECT column1, column2, column3 FROM MyTable
WHERE MyXml.exist('(/root/product[@id="304"])[1]') = 1
```

XML Data Modification

- XQuery extensions: Insert, update, and delete
- XML sub-tree modification:
  - Add or delete XML sub-trees
  - Update values
- Generate consistent state

Example: modify Method

General syntax

xml-column.modify('*insert-query*'|'*delete-query*'|'*replace-query*')

```
UPDATE MyTable
SET MyXml.modify('delete /root/product[@id="304"]')

UPDATE MyTable
SET MyXml.modify('delete /root/product/description')

UPDATE MyTable
MyXml.modify('insert <newelement>New element
   content</newelement>
          as first into (/root/product[@id="304"])[1]')
```

MyTable(MyXml XML)

XQuery: modify()

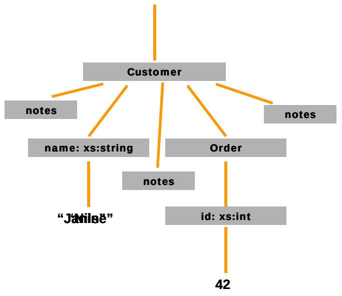
- Used with SET:

```
declare @xdoc xml
set @xdoc.modify('delete /a/b[@id="42"]')

update T
set T.xdoc.modify('insert <b/> into /a')
where T.id=1
```

- Relational row-level concurrency: whole XML instance is locked

XML-DML: insert delete update replace value of



```
delete /Customer/Order[id = 42]
replace value of (/Customer/name)[1]
with "Nils"
insert <notes/>
as last
into /Customer
insert <notes/>
as first
into /Customer
insert <notes/>
before
/Customer/name
insert <notes/>
after
/Customer/name
```

Target needs to be statically one node

Plan

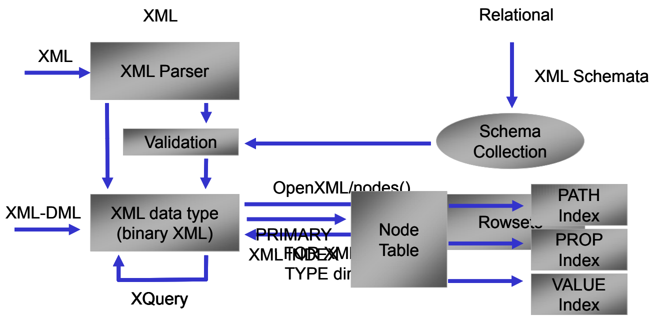
- Fiche technique
- Possibilité de stockage du XML
- Les extensions SQL pour gérer le XML ⇔ relationnel
- L'indexation XML

Indexation XML

Four different types of XML indexes can be built to speed certain types of XQuery operations.

- A primary XML index which creates a B+tree index on all tags, values, and paths of the XML instances in the column. It provides efficient evaluation of queries on XML data, and reassembly of the XML result from the B+tree while preserving document order and document structure.
- Secondary XML indexes: (needs the primary XML index)
  - PATH index for path-based queries,
  - PROPERTY index for property bag scenarios,
  - VALUE index for value-based queries.

SQL Server 2005 XML Architecture



XML
Relational
XML
XML Parser
XML Schemata
Validation
Schema Collection
OpenXML/nodes()
XML-DML
XML data type (binary XML)
PRIMARY XML INDEX
FOR XML TYPE dir
Node Table
Rowsets
PATH Index
PROP Index
VALUE Index
XQuery