

XQuery dans SQL Server 2005

par Rudi Bruchez ([contact](#))

Date de publication : 01/02/2008

Une introduction au langage d'extraction de documents XML, XQuery, tel qu'il est implémenté dans Microsoft SQL Server 2005. Article paru dans l'édition de décembre 2007 de SQL Server Magazine, édition française.

I - Qu'est XQuery ?.....	3
II - XML d'exemple.....	4
III - Concepts de base.....	5
III-A - XPath.....	5
III-B - Le Prologue.....	5
III-C - Expressions.....	6
III-D - FLWOR.....	6
III-E - Séquence.....	7
III-F - Spécificités SQL Server.....	7
IV - Passer des requêtes XQuery dans SQL Server.....	8
IV-A - query().....	8
IV-B - value().....	9
IV-C - exist().....	9
IV-D - nodes().....	10
IV-E - modify().....	10
V - Conclusion.....	11
VI - Références.....	12

I - Qu'est XQuery ?

XQuery est un langage d'extraction de données opérant sur du contenu XML, développé par un groupe de travail du W3C (World Wide Web Consortium, <http://www.w3.org/>). Sa sémantique et ses capacités sont similaires à l'expression SELECT du langage SQL. La recommandation du W3C pour XQuery 1.0, en travail depuis plusieurs années, a finalement paru en janvier 2007. SQL Server 2005 implémente une portion déjà stable à l'époque de sa sortie, du document de travail de cette recommandation. Nous avons donc un XQuery très proche du standard, et toute personne familière avec XQuery ou XPath devrait se sentir très rapidement à l'aise. Cet article, destiné à la communauté SQL Server, s'adresse plutôt aux familiers du langage SQL, à qui ce mode de requêtage peut sembler troublant au premier abord. Notre souhait est de lui faciliter l'entrée dans la pratique, en nous basant beaucoup sur des exemples tout en introduisant la terminologie nécessaire.

XQuery est basé sur XPath 2.0, un langage permettant d'exprimer un chemin de navigation à l'intérieur d'une structure XML. XPath permettant de filtrer éléments et attributs, XQuery y ajoute les autres fonctionnalités communes du SELECT : transformation et restructuration du résultat, tri, manipulation de chaînes, calculs, agrégation. Nous le verrons en pratique.

Certaines fonctionnalités de XQuery 1.0, plus orientées vers l'établissement d'un langage modulaire complet, ne sont pas implémentées par SQL Server, comme la capacité de créer des fonctions ou des bibliothèques. Pour le reste, nous avons à disposition un langage d'expression de chemin et de requête souple et riche, agrémenté d'additions propres à SQL Server.

II - XML d'exemple

Nous allons utiliser deux structures d'exemple, représentant une partie d'échecs. L'une définit l'échiquier et l'historique de la partie, l'autre les joueurs en présence. La seconde structure est un exemple volontairement très simple d'utilisation d'espaces de noms.

Nous pouvons soit directement travailler avec une variable de type XML contenant ce document, soit avec une colonne. Nous allons travailler avec une table temporaire et des colonnes.

XML d'exemple

```

DECLARE @x as xml, @xn as xml

SET @x = '
<echiquier>
  <blancs>
    <pieces>
      < pion numero= "2" position= "B3">
        <historique>
          < coup sequence= "1">
            <source>B2</source>
            <destination>B3</destination>
          </coup>
        </historique>
      </ pion>
      < pion numero= "3" position= "C5">
        <historique>
          < coup sequence= "1">
            <source>C2</source>
            <destination>C4</destination>
          </coup>
          < coup sequence= "2">
            <source>C4</source>
            <destination>C5</destination>
          </coup>
        </historique>
      </ pion>
      < roi position= "C2">
        <historique>
          < coup sequence= "1">
            <source>D1</source>
            <destination>C2</destination>
          </coup>
        </historique>
      </ roi>
    </pieces>
  </blancs>
  <noirs>
</noirs>
</echiquier>'

SET @xn = '
<joueurs xmlns:ecj="http://www.babaluga.com/xmlnamespaces/echecs/joueur">
  <ecj:joueur position="blancs">
    <ecj:nom>Fischer</ecj:nom>
    <ecj:prenom>Bobby</ecj:prenom>
  </ecj:joueur>
</joueurs>
'

CREATE TABLE #partie (PartieId int not null identity(1,1) primary key, joueurs xml not null, echiquier
xml not null)
INSERT INTO #partie (joueurs, echiquier) VALUES (@xn, @x)

```

III - Concepts de base

XQuery est strictement sensible à la casse, ses mots-clés sont en minuscule. Il peut travailler avec du contenu XML typé (validé par un schéma) ou non typé. Sachez toutefois que le travail sur le contenu typé est plus rapide, car il évite une conversion implicite de type lors du traitement. XQuery est, comme SQL, un langage fortement typé. Si un paramètre est passé à une fonction XQuery, celui-ci doit être du type attendu. Si le XML est non typé, une conversion implicite est tentée. Si par contre le contenu XML est typé, vous devez explicitement transtyper votre paramètre si nécessaire.

Précisons également la nomenclature : on appelle **noeud** un élément ou un attribut, son identifiant est un **QName** (*Qualified Name*), c'est-à-dire un nom, possiblement qualifié par un espace de noms. La donnée scalaire que contient ce noeud est nommée **valeur atomique**. Un **item** est soit un noeud, soit une valeur atomique. Une **séquence** est une liste comportant de zéro à plusieurs items.

III-A - XPath

XQuery est basé sur XPath 2.0, un langage permettant de parcourir le document pour retrouver un noeud ou une valeur atomique. Vous pouvez très simplement dessiner un chemin d'élément à sous-élément et filtrer le noeud recherché. Le filtrage se fait à l'aide d'un prédicat contenu entre crochets [], qui applique son contenu (une expression évaluée à vrai ou faux) au contexte du chemin.

Par exemple, dans notre XML, le chemin pour trouver le premier coup du troisième pion blanc peut être exprimé en XPath ainsi :

```
/echiquier/blancs/pièces/pion[@numero = "3"]/historique/coup[@sequence = "1"]
```

Un chemin est composé d'une suite de noeuds (des étapes) séparés par des /. Chaque étape comporte un axe, qui spécifie la direction du mouvement, un test de noeud (le QName de l'élément ou de l'attribut) et peut introduire des expressions de filtre, nommées qualificateurs d'étapes (step qualifiers). L'axe peut être explicite, ou abrégé. Par exemple, l'exemple ci-dessus avec axes explicites donne :

```
self::node()/child::echiquier/child::blancs/child::pièces/child::pion[attribute::numero = "3"]  
/child::historique/child::coup[attribute::sequence = "1"]
```

:: étant le séparateur d'axe. Vous avez compris que child est l'axe par défaut, et que attribute:: s'abrège « @ ». Le noeud parent s'abrège « .. », et le noeud actuel (le noeud de contexte), « . ».

Vous pouvez également sauter des étapes et chercher des sous-noeuds dans tous les enfants, en utilisant la syntaxe // (déconseillée pour des raisons de performance, et les risques encourus en cas de changement de structure). Pour retourner des noeuds au même niveau mais portant des QNames différents, vous pouvez utiliser le caractère générique *.

III-B - Le Prologue

Le type XML supporte les espaces de noms (*namespaces*, recommandation W3C <http://www.w3.org/TR/REC-xml-names/>), qui sont des identificateurs de vocabulaire, permettant d'attribuer un noeud à un vocabulaire. Si le contenu de la colonne ou de la variable les utilise, la requête XQuery doit déclarer les namespaces avec lesquels elle va travailler. Cette déclaration peut se faire en ligne # à l'intérieur de la requête Xquery # à l'aide d'un prologue, ou en préfixe à la requête SQL (au SELECT) dans laquelle on va écrire des requêtes XQuery. Cette deuxième solution est à préférer pour améliorer la lisibilité du code, et pour éviter une redéclaration dans les diverses requêtes. La recommandation W3C indique que le prologue contient une suite de déclarations diverses séparées par des points-virgule. En SQL Server, il ne peut contenir que des déclarations d'espaces de noms.

Exemple avec une déclaration de namespaces en prologue :

```
SELECT joueurs.query('
  declare namespace ecj="http://www.babaluga.com/xmlnamespaces/echechs/joueur";
  /joueurs/ecj:joueur[@position = "blancs"]')
FROM #partie
```

La déclaration en préfixe de la requête utilise la syntaxe SQL suivante :

```
WITH XMLNAMESPACES ('urn' as alias, 'urn2' as alias2) ...
```

Exemple :

```
WITH XMLNAMESPACES ('http://www.babaluga.com/xmlnamespaces/echechs/joueur' as e)
SELECT joueurs.query('/joueurs/e:joueur[@position = "blancs"]')
FROM #partie
```

Le mot-clé WITH étant déjà utilisé dans la syntaxe T-SQL pour spécifier des options de requête par exemple, l'utilisation de WITH XMLNAMESPACES dans un lot de requêtes doit être précédée de la fermeture de l'instruction précédente par un ;.

III-C - Expressions

XQuery permet plusieurs types d'expressions. Outre les expressions primaires, comme les noeuds ou les séquences, et les chemins XPath, vous pouvez évidemment utiliser des opérateurs de comparaison, des opérateurs logiques (or et and) et arithmétiques, des branchements conditionnels (if then else), des quantificateurs (some, every), qui permettent de tester les séquences, ainsi que la structure FLWOR.

III-D - FLWOR

La vraie richesse de XQuery fleurit grâce à la syntaxe FLWOR. FLWOR, qu'on prononce flower, permet d'exprimer une requête d'extraction complexe des noeuds XML, un peu à l'image du langage SQL. Il s'agit d'une abréviation mnémotechnique qui indique quel mot-clé peut être utilisé à quelle position dans la requête.

Détaillons l'abréviation :

F	for	Crée une séquence de noeuds.
L	let	Affecte des noeuds à des variables. Le mot-clé LET n'est pas implémenté dans le XQuery de SQL Server, l'unique affectation se fait dans le FOR.
W	where	Applique une clause de filtrage.
O	order by	Trie la séquence.
R	return	Retourne le résultat.

Les clauses WHERE et ORDER BY sont optionnelles. A minima, une instruction FLWOR est une instruction FR.

Exemple de FLWOR complet, qui retrouve toutes les pièces blanches actuellement dans la colonne B de l'échiquier (dans notre exemple, le Roi) :

```
SELECT echiquier.query('
  for $piece in (/echiquier/blancs/pièces/*)
  where contains($piece/@position,"B")
  order by $piece/@position
  return $piece')
FROM #partie;
```

Vous noterez que le signe \$ est utilisé pour indiquer une variable.

III-E - Séquence

En XQuery, une séquence est une liste de chemins. Une expression XPath qui retourne plusieurs noeuds, génère une séquence. Vous pouvez aussi indiquer explicitement une séquence en utilisant un constructeur, c'est-à-dire une liste entre parenthèses contenant des chemins séparés par des virgules. Exemple :

```
SELECT echiquier.query('
  for $piece in (/echiquier/blancs/pièces/pion, /echiquier/blancs/pièces/roi)
  return $piece')
FROM #partie;
```

Ce qui est équivalent à :

```
SELECT echiquier.query('(
  /echiquier/blancs/pièces/pion, /echiquier/blancs/pièces/roi)')
FROM #partie;
```

Une séquence contenant un seul item, est nommée un singleton. Parfois, une séquence peut être passée, par exemple dans le for, comme nous venons de le voir, ou comme paramètre de fonction d'agrégation. À d'autres endroits, un singleton est attendu, par exemple en paramètre d'une fonction de chaîne, ou dans la méthode value().

III-F - Spécificités SQL Server

XQuery 1.0 n'offre aucune construction pour modifier le contenu du document XML. Cette possibilité est en travail et sera ajoutée dans une version future. SQL Server implémente sa propre syntaxe additionnelle, nommée XML-DML. De plus, un certain nombre de constructions XQuery ne sont pas supportées (par exemple les opérateurs ensemblistes comme union). Ces limitations sont indiquées dans les *Books Online*.

IV - Passer des requêtes XQuery dans SQL Server

Le type de données XML, introduit par SQL Server 2005, agit comme une classe, comportant des méthodes qu'on peut appliquer au contenu de la variable ou de la colonne définie sur ce type.

Il supporte cinq méthodes, permettant de manipuler le contenu avec beaucoup de flexibilité.

- 1 `query()`, applique au document toute la panoplie du langage XQuery, et retourne un fragment XML ;
- 2 `value()`, extrait un item (un singleton) du document XML, et le convertit dans un type SQL défini en paramètre ;
- 3 `exist()`, recherche l'existence d'un critère ;
- 4 `node()`, retourne chaque noeud correspondant au chemin XPath exprimé en une nouvelle ligne. Vous utilisez `node()` dans la clause FROM pour retourner un jeu de résultats composé des noeuds extraits ;
- 5 `modify()`, permet d'insérer, modifier et supprimer un noeud.

IV-A - `query()`

`query()` permet d'appliquer toute la richesse de la syntaxe XQuery pour extraire un nouveau fragment XML du document original. Nous en avons déjà vu des exemples. Voici quelques applications pratiques, montrant les possibilités du langage.

Vous pouvez générer un nouveau fragment XML à la volée, simplement en écrivant des chevrons en littéraux dans votre return, ou dans n'importe quelle expression de retour XQuery. Si vous voulez interpoler le résultat d'une expression, encadrez-la par `{}`.

```
SELECT echiquier.query('{/echiquier/blancs/pièces/roi/@position}')
FROM #partie;
```

Vous noterez en essayant ce code, que le résultat retourne la position en attribut. Ici, `/echiquier/blancs/pièces/roi/@position` retourne un noeud, qui est un attribut. Or, si vous voulez le « mapper » en élément, vous avez besoin d'extraire la valeur atomique du noeud position, par un processus qu'on appelle atomisation. La fonction d'atomisation est `data()` :

```
SELECT echiquier.query('{data(/echiquier/blancs/pièces/roi/@position)}')
FROM #partie;
```

Dans certains cas, XQuery atomise le résultat par défaut (par exemple en paramètre de fonctions). Parfois, comme ici, nous devons le faire explicitement.

Voyons également l'utilisation des agrégations (`sum`, `min`, `max`, `count`, `avg`). Ici un `count()` :

```
SELECT echiquier.query('{ count(/echiquier/blancs//historique/coup) }')
FROM #partie;
```

Vous pouvez au besoin transtyper vos valeurs atomiques en types XML (du namespace `xs`), en utilisant la syntaxe « `cast as type?` ». Attention, vous ne pouvez transtyper que des singletons :

```
SELECT echiquier.query('
(/echiquier/blancs/pièces/pion[@numero = "3"]/historique/coup[@sequence = "1"]/source)[1] cast as
xs:string?')
FROM #partie;
```

Vous pouvez également insérer des valeurs de colonne ou de variable du jeu de résultat relationnel « externe » au type xml, en utilisant les fonctions `sql:column()` et `sql:variable()`

```
SELECT echiquier.query('{sql:column("PartieId")}')
FROM #partie;
```

Attention, certains type comme `datetime` et `smalldatetime` ne sont pas supportés, vous devez les convertir avant de les utiliser dans XQuery.

IV-B - value()

`value()` permet d'extraire une valeur de noeud, et de la retourner dans un type de données SQL, spécifié en paramètre. Le chemin doit permettre de retourner une seule valeur. Pour indiquer le singleton de votre chemin, utilisez cette syntaxe : (chemin)[n° de position] (ce qu'on appelle un prédicat de position). Exemple sur un attribut, avec recherche dans le chemin :

```
SELECT echiquier.value('/echiquier/blancs/pièces/pion[@numero = "3"]/@position[1]', 'char(2)')
FROM #partie;
```

Pour un élément :

```
SELECT echiquier.value('/echiquier/blancs/pièces/pion[@numero = "3"]/historique/coup[@sequence = "1"]/
source[1]', 'char(2)')
FROM #partie;
```

Puisqu'il vous retourne une valeur, `value()` sera utile dans un `SELECT`. Pour un test dans une clause `WHERE`, préférez-lui `exist()`, les performances sont meilleures.

IV-C - exist()

`exist()` vous permet, dans une clause `WHERE`, de tester une existence à l'aide d'une instruction XPath. Un bit vous est retourné.

Test de valeur atomique :

```
SELECT echiquier.query('/echiquier/blancs/pièces/piece[@position = "B3"]')
FROM #partie
WHERE echiquier.exist('/echiquier/blancs/pièces/piece[@position = "B3"]') = 1;
```

Exemple de recherche avec deux critères d'attribut :

```
SELECT echiquier.query('/echiquier/blancs/pièces/pion[@numero = "2"]')
FROM #partie
WHERE echiquier.exist('/echiquier/blancs/pièces/pion[@numero = "2" and @position = "B3"]') = 1;
```

Autre syntaxe possible :

```
SELECT echiquier.query('/echiquier/blancs/pièces/pion[@numero = "2"]')
FROM #partie
WHERE echiquier.exist('/echiquier/blancs/pièces/pion[@numero = "2"][@position = "B3"]') = 1;
```

Présence d'un élément :

```
SELECT echiquier.query('/echiquier/blancs/pièces/roi')
FROM #partie
WHERE echiquier.exist('/echiquier/blancs/pièces/roi') = 1;
```

IV-D - nodes()

nodes() extrait des noeuds, créant pour chacun une ligne de résultat. À utiliser dans la clause FROM. Pour spécifier une colonne, vous devez utiliser l'opérateur CROSS APPLY avec cette syntaxe :

```
FROM table
CROSS APPLY collexml.nodes('XQuery') as aliastable(aliascolonne)
```

Exemple avec une liste :

```
SELECT col.query('.')
FROM #partie p
CROSS APPLY echiquier.nodes('/echiquier/blancs/pièces/pion, /echiquier/blancs/pièces/pion/historique/
coup') tbl(col);
```

IV-E - modify()

modify() vous permet d'appliquer une modification au contenu du document, en utilisant une extension créée par Microsoft au langage XQuery, nommée XML-DML.

Exemple :

```
UPDATE #partie
SET echiquier.modify('
  replace value of (/echiquier/blancs/pièces/pion[@position = "C5"]/@position)[1]
  with "pris!";')
WHERE echiquier.exist('/echiquier/blancs/pièces/pion[@position = "C5"]') = 1

SELECT echiquier.query('/echiquier/blancs/pièces/pion[@position = "pris!"]')
FROM #partie
```

V - Conclusion

Nous vous avons présenté, de façon synthétique, comment appliquer des instructions XQuery au type XML dans SQL Server. Nous n'avons fait qu'effleurer la puissance de XQuery. Expérimentez, approfondissez, vous verrez que ce langage regorge de souplesse et de surprises !

VI - Références

- Melton, Jim; Buxton, Stephen. Querying XML. Morgan Kaufmann (2006)
- Klein, Scott. Professional SQL Server 2005 XML. Wrox (2006)