www.mas-ict.ch

Hes·so
Haute Ecole Spécialisée de Suisse occidentale

Module: XML et les bases de données

SQL2006: SQL/XML
2ieme édition
(révision 2008-2011)

Houda Chabbi Drissi

houda.chabbi@hefr.ch

SQL/XML: 2ieme edition - 2006

Evolution du SQL2003 avec:

- Evolution du XMLType: (cf. Cours partie1)

- Introduction de nouvelles fonctions:
  - XMLQuery
  - XMLTable
  - XMLCast

SQL/XML: 2ieme edition

Evolution du SQL2003 avec:

- Evolution du XMLType

- Introduction de nouvelles fonctions:
  - XMLQuery
  - XMLTable
  - XMLCast

XML → XML

XMLQuery

XML →XML

It is an SQL scalar function enabling the execution of an XQuery expression from within an SQL context:

- variables could be passed to the XQuery expression.
- the returned XML value is an XML sequence. This sequence can be empty or contain one or more items.

XMLQuery: syntax

Pass variables to the XQuery expression

```
XMLQuery(
  Xquery-expression
  [PASSING { BY REF | BY VALUE}
        (value-expression AS identifier
                                  [BY REF | BY VALUE])*]
  RETURNING { CONTENT|SEQUENCE } { BY REF|BY VALUE}
```

XML Query: syntax

- The **XQuery-expression** is, a character string containing an XQuery expression with possible variables.

- The **XQuery global variable** "identifier" are binded via the:

value-expression AS identifier [BY REF | BY VALUE]

where value-expression is an XML-generating expressions or a SELECT statements.

```
XMLQuery(
    Xquery-expression
    [PASSING { BY REF | BY VALUE}
            (value-expression AS identifier [BY REF | BY VALUE])*]
    RETURNING { CONTENT|SEQUENCE } { BY REF|BY VALUE}
```

XMLQuery sous oracle

XMLQuery(
    Xquery-expression
    [PASSING { ~~BY REF |~~BY VALUE}
        (value-expression AS identifier ~~[BY REF | BY VALUE]~~)*]
    RETURNING { CONTENT~~|SEQUENCE~~ } ~~{ BY REF|BY VALUE})~~

Implémentation Oracle

XMLQuery(
    Xquery-expression
    [PASSING { BY VALUE}
        (value-expression AS identifier)*]
    RETURNING CONTENT **[NULL ON EMPTY]**)

XMLQuery expression: exemple1

```
XMLQuery(
    Xquery-expression
    [PASSING { BY REF | BY VALUE}
        (value-expression AS identifier [BY REF | BY VALUE])*]
    RETURNING { CONTENT|SEQUENCE } { BY REF|BY VALUE}
```

```
SELECT
  XMLQuery(
    '(1, 2 + 3, "a", 100 to 102, <A>33</A>)'
    RETURNING CONTENT) AS output
FROM DUAL;
```

```
OUTPUT
--------------------------
1 5 a 100 101 102<A>33</A>

1 row selected.
```

XML Query: syntax

```
XMLQuery(
    Xquery-expression
    [PASSING { BY REF | BY VALUE}
            (value-expression AS identifier  [BY REF | BY VALUE])*]
    RETURNING { CONTENT|SEQUENCE } { BY REF|BY VALUE}
```

- If **BY REF** is specified, then a **reference to the value** is bound to the variable; -preserves Id (of an XML type). When XML values are passed by reference, the XPath evaluation uses the input node trees which preserves all properties, including the original node identities and document order.
- if **BY VALUE** is specified, then a **copy of the value** is bound directly to the variable.

Pour ce faire, le concept de l'identité du nœud du modèle de données est utilisé. On utilise BY VALUE, pour obtenir des copies avec une nouvelle identité de nœud.

XML Query: syntax

```
XMLQuery(
    Xquery-expression
    [PASSING { BY REF | BY VALUE}
            (value-expression AS identifier [BY REF | BY VALUE])*]
    RETURNING { CONTENT|SEQUENCE } { BY REF|BY VALUE}
```

The first passing mechanism specified (before the argument-list) is applied to each argument for which neither BY REF nor BY VALUE is specified.

If the value-expression's type is not an XML type, then the passing mechanism cannot be specified (and the value is bound directly to the variable).

XMLQuery expression: exemple2-variable

```
create table orders (ordid integer, orddoc XML);
```

Un exemple de orddoc:

```
<doc>
  <order @id="1001">
    <date>January 1, 2001</date>
   <lineitem @id="LI100101" @quantity="200" @price=19.90>
  </order>
</doc>
```

```
SELECT XMLQuery(
XPATH → '$ord//lineitem[@price > 100]'
        passing orddoc as "ord")
FROM orders
```

Requête qui retourne autant de lignes qu'il ya de lignes dans la table orders.

Variable pour lier

ligne 1: <lineitem id = " 2" > ... </lineitem>
<lineitem id = " 7" > ... </lineitem >
ligne 2: ()
ligne 3: <lineitem id = " 9" > ... </lineitem>
….

XMLQuery expression: exemple3

```
create table Emps (empno integer, description XML);
-- With description has an address element
```

```
SELECT empno,
        XmlQuery( './/address'
                  PASSING description )
FROM Emps
```

description maps to default item

Requête qui retourne autant de lignes qu'il ya de lignes dans la table Emps avec 2 colonnes.

XMLQuery expression: exemple4

create table purchaseorder (pocol XML);

```
select XMLQuery(
    'for $i in ./PurchaseOrder
     where $i/PoNo = $j/val
     return $i//Item '
    passing p.pocol,
            xmlelement("val",2100) as "j"
  returning content)
from purchaseorder p
```

FLWR

Requête qui retourne autant de lignes qu'il ya de lignes dans la table purchaseorder toute vide sauf 1

Retourne les items impliqués dans la commande num 2100

```
<Item itemno="21"><Quantity>200</Quantity>..</Item>
<Item itemno="22"><Quantity>22</Quantity>..</Item>
```

XMLQuery: advantages

By executing XQuery expressions from within the SQL context, you can:

- operate on parts of stored XML documents, instead of entire XML documents (only XQuery can query within an XML document; SQL alone queries at the whole document level)
- enable XML data to participate in SQL queries
- operate on both relational and XML data

SQL/XML: 2ieme edition

Evolution du SQL2003 avec:

- Evolution du XMLType:

- Introduction de nouvelles fonctions:
  - XMLQuery
  - XMLTable: une fonction de table
  - XMLCast

XML → Relationnel

XMLTable

XML → Relationnel

XMLTABLE is an SQL table function that returns a table (pseudo) from the evaluation of XQuery expressions:

– XQuery expressions normally return values as a sequence,

– XMLTABLE allows to execute an XQuery expression and return values as a table instead.

– The table that is returned can contain columns of any SQL data type, including XML.

– XMLTABLE fits in FROM in SQL statement.

– Persistent table instead of the pseudo-table, by using CREATE TABLE mytable…or CREATE VIEW…

XMLTable: syntax

The operation of XMLTABLE is very much analogous to shredding XML for storage in relational tables.

Pass variables to the XQuery expression

```
XMLTABLE (
  [ namespace-declaration , ]
 XQUERY-expression
 [ PASSING argument-list ]
 COLUMNS XMLtbl-column-definitions )
```

Define the structure of the resulting table

XMLTable

The XQuery-expression is used to identify XML values that will be used to construct SQL rows for the virtual table generated by XMLTABLE.

Example

```
XMLTABLE (
  [ namespace-declaration , ]
  XQUERY-expression
  [ PASSING argument-list ]
  COLUMNS XMLtbl-column-definitions )
```

```
XMLTable(
  'doc("mybooks.xml")//Authors'
  COLUMNS Name VARCHAR(20))
```

Element Authors (Name,...)

Une "read-only view" où chaque ligne est associée à un nœud Authors avec la colonne Name qui contient la valeur de l'élément Name.

Column definition (1)

The XMLtbl-column-definitions is a comma-separated list of column definitions:

- one derived from the ordinary SQL column definitions allowing the provision of another XQuery-expression that specifies the data to be stored in the column being defined;

```
column-name   data-type   [ BY REF | BY VALUE ]
                          [ default-clause ]
                          [ PATH XQuery-expression ]
```

Column definition (2)

column-name data-type [BY REF | BY VALUE]
[default-clause]
[PATH XQuery-expression]

- If the PATH clause is not specified, then the column's data comes from an element whose name is the same as the column-name and that is an immediate child of the XML value that forms the row as a whole.

- If PATH is specified, then the XQuery-expression is evaluated in the context of the XML value that forms the row as a whole and the result is stored into the column being defined.

Example: PATH XQuery-expr.

column-name **data-type** [BY REF | BY VALUE]
[default-clause]
[PATH XQuery-expression]

```
XMLTable(
  'doc("mybooks.xml")//Authors'
  COLUMNS Name VARCHAR(20),
                DateNaiss DATE Path '@dateN')
```

Element Author (Name,...)
Attribut Author dateN Cdata

Une "read-only view" où chaque ligne est associée à un nœud *Authors* avec les 2 colonnes:

- Name qui contient la valeur de l'élément Name
- DateNaiss qui contient la valeur de l'attribut dateN

Example (sybase)

```
select *
from xmltable('/doc/item'
        passing
          '<doc><item><id>1</id><name>Box</name></item>'
         +'<item><id>2</id><name>Jar</name></item></doc>'
        columns id      int            path 'id',
                name    varchar(20) path 'name')
```

id	name
---	---
1	Box
2	Jar

Column definition (3)

column-name data-type [BY REF | BY VALUE]

[default-clause]

[PATH XQuery-expression]

- If the data-type is XML(SEQUENCE), then either BY REF or BY VALUE must be specified, and the XQuery-expression will return a value whose type is XML(SEQUENCE) by reference or by value, respectively.

- If the data-type is anything else, then neither BY REF nor BY VALUE can be specified, and the XQuery-expression returns XML(ANY CONTENT) or XML(UNTYPED CONTENT).

Column definition (4)

The XMLtbl-column-definitions is a comma-separated list of column definitions:

- one derived …;
- the other creates a special column, an ordinality column, that can be used to capture the ordinal position of an item in an XQuery sequence:

```
column-name FOR ORDINALITY
```

At most one ordinality column can be defined in a given XMLTABLE invocation

XMLTable: advantages

Once this shredding has taken place (returning a table, instead of a sequence), the virtual table can be:

- Iterated over within an SQL fullselect

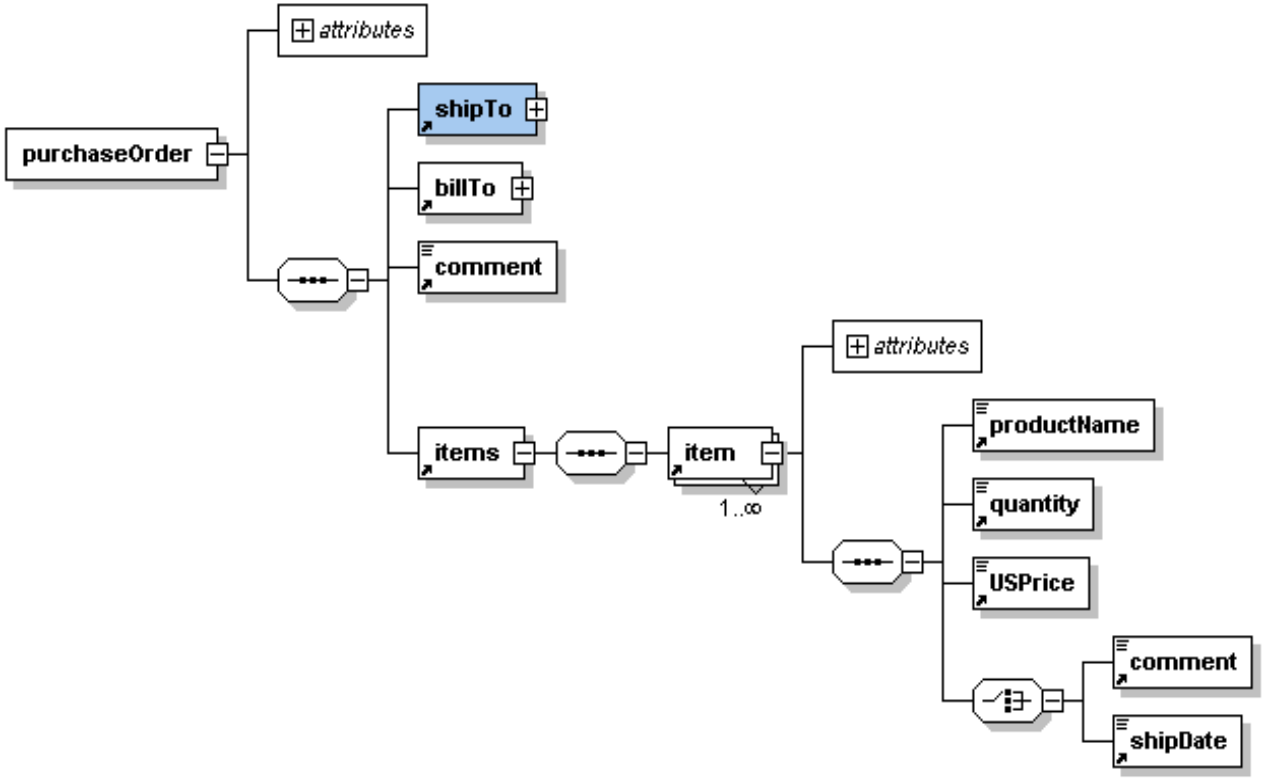
```
SELECT X.* FROM XMLTABLE ( ) as X
```

- Inserted into pre-existing tables
- Used in another SQL statement as a virtual table, even possibly in a join expression
- Sorted on values from an XML doc

```
SELECT X.* FROM XMLTABLE ( ) as X
ORDER BY 1
```

Exemple

On dispose d'une table "purchaseorder" qui a une colonne XMLpo qui contient du XML.



Dans ce qui suit on donne l'instance XML du 1er tuple de la table KeyField = 1

Instance XMLpo pour KeyField = 1

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
      <shipTo country="US">
            <name>Alice Smith</name>
            <street>…</street><city>…</city><state>…</state><zip>…</zip>
      </shipTo>
      <billTo country="US">
            <name>Robert Smith</name>
            <street>…</street><city>…</city><state>…</state><zip>…</zip>
      </billTo>
      <comment>bla bla…</comment>
      <items>
            <item partNum="872-AA">
                  <productName>Lawnmower</productName>
                  <quantity>1</quantity><USPrice>148.95</USPrice>
                  <comment>Confirm this is electric</comment>
            </item>
            <item partNum="926-AA">
                  <productName>Baby Monitor</productName>
                  <quantity>1</quantity><USPrice>39.98</USPrice>
                  <shipDate>1999-05-21</shipDate>
            </item>
      </items>
</purchaseOrder>
```

Requête 1

On veut obtenir la table suivante:

Seqno	Part #	ProductName	Quantity	USPrice	Ship Date	Comment
1	872-AA	Lawnmower	1	148.95	*null*	Confirm is electric
2	926-AA	Baby monitor	1	39.98	1999-05-21	*null*

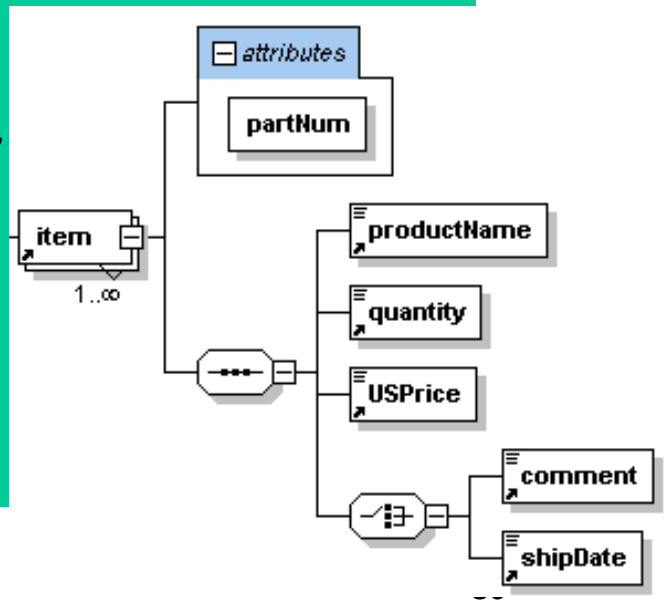
```
<?xml version="1.0"?>
                                                    ›…</state><zip>…</zip>
    </shipTo>
    <billTo country="US">
        <name>Robert Smith</name>
        <street>…</street><city>…</city><state>…</state><zip>…</zip>
    </billTo>
    <comment>bla bla…</comment>
    <items>
        <item partNum="872-AA">
            <productName>Lawnmower</productName>
            <quantity>1</quantity><USPrice>148.95</USPrice>
            <comment>Confirm this is electric</comment>
        </item>
        <item partNum="926-AA">
            <productName>Baby Monitor</productName>
            <quantity>1</quantity><USPrice>39.98</USPrice>
```

Où Seqno: est le numéro de séquence des items dans le fichiers XML

Utilisation de **xmltable**, avec une référence à une colonne existante dans la table purchaseorder : cette table doit précéder la clause de xmltable dans la clause from.

Seqno	Part #	ProductName	Quantity	USPrice	Ship Date	Comment
1	872-AA	Lawnmower	1	148.95	*null*	Confirm is electric
2	926-AA	Baby monitor	1	39.98	1999-05-21	*null*

```
SELECT X.*  FROM PurchaseOrders PO,
    XMLTable ( '//item'
    PASSING PO.XMLpo
   COLUMNS
      "Seqno" FOR ORDINALITY,
      "Part #" CHAR(6) PATH '@partnum',
      "ProductName" CHAR(20) PATH 'productName',
      "Quantity" INTEGER PATH 'quantity',
      "US Price" DECIMAL(9,2) PATH 'USPrice',
      "Ship Date" DATE PATH 'shipDate',
      "Comment" CHAR(80) PATH 'comment'
) AS X
WHERE PO.KeyField = 1
```



Requête 2

Seqno	Part #	ProductName	Quantity	USPrice	Ship Date	Comment
1	872-AA	Lawnmower	1	148.95	*null*	Confirm this is electric
2	926-AA	Baby monitor	1	39.98	1999-05-21	***RAS***

Le NULL est remplacé par RAS

XMLtable expression:default

```
SELECT X.*  FROM PurchaseOrders PO,
    XMLTable ( '//item'
    PASSING PO.XMLpo
   COLUMNS
      "Seqno" FOR ORDINALITY,
      "Part #" CHAR(6) PATH '@partnum',
      "Product Name" CHAR(20) PATH 'productName',
      "Quantity" INTEGER PATH 'quantity',
      "US Price" DECIMAL(9,2) PATH 'USPrice',
      "Ship Date" DATE PATH 'shipDate',
      "Comment" CHAR(80) DEFAULT 'RAS' PATH 'comment'
) AS X
WHERE PO.KeyField = 1
```

Requête 3

Même littéral pour la colonne que celui désignant le XPATH

Seqno	@partnum	productName	quantity	USPrice	shipDate	comment
1	872-AA	Lawnmower	1	148.95	*null*	Confirm this is electric
2	926-AA	Baby monitor	1	39.98	1999-05-21	*RAS*

```
SELECT X.*  FROM PurchaseOrders PO,
    XMLTable ( '//item'
    PASSING PO.XMLpo
   COLUMNS
      "Seqno" FOR ORDINALITY,
      "@partum" CHAR(6) ,
      "productName" CHAR(20) ,
      "quantity" INTEGER ,
      "USPrice" DECIMAL(9,2),
      "shipDate" DATE,
      "comment" CHAR(80) DEFAULT 'RAS'
) AS X
WHERE PO.KeyField = 1
```

Requête 4

Même littéral pour la colonne que celui désignant le XPATH sauf lui

Seqno	Part #	productName	quantity	USPrice	shipDate	comment
1	872-AA	Lawnmower	1	148.95	*null*	Confirm this is electric
2	926-AA	Baby monitor	1	39.98	1999-05-21	*RAS*

```
SELECT X.*  FROM PurchaseOrders PO,
    XMLTable ( '//item'
    PASSING PO.XMLpo
   COLUMNS
      "Seqno" FOR ORDINALITY,
      "Part #" CHAR(6) PATH '@partnum',
      "productName" CHAR(20) ,
      "quantity" INTEGER ,
      "USPrice" DECIMAL(9,2),
      "shipDate" DATE,
      "comment" CHAR(80) DEFAULT 'RAS'
) AS X
WHERE PO.KeyField = 1
```

Autre manière de renommer les colonnes

```
SELECT X.*  FROM PurchaseOrders PO,
    XMLTable ( '//item'
    PASSING PO.XMLpo
   COLUMNS
      "Seqno" FOR ORDINALITY,
      “@partum" CHAR(6) ,
      “productName" CHAR(20) ,
      “quantity" INTEGER ,
      "USPrice" DECIMAL(9,2),
      “shipDate" DATE,
      “comment" CHAR(80) DEFAULT ‘RAS’
) AS X ("Seqno", “Part #”, "Product Name", "Quantity“,
   "US Price", "Ship Date", "Comment" )
WHERE PO.KeyField = 1
```

Pas forcément lisible!

SQL/XML: 2ieme edition

Evolution du SQL2003 avec:

- Evolution du XMLType

- Introduction de nouvelles fonctions:
  - XMLQuery
  - XMLTable
  - XMLCast

XMLCast

XMLCAST (value-expression AS type)

- casts an XML value into an SQL type.
- casts an SQL value into XML.

XMLCast: exemples

XMLCAST (value-expression AS type)

```
XMLCAST(NULL AS XML)
```

Create a null XML value.

```
XMLCAST(XMLQUERY('/PRODUCT/QUANTITY'
  PASSING xmlcol) AS INTEGER)
```

Convert a value extracted from an XMLQUERY expression into an INTEGER.

```
XMLCAST((SELECT quantity FROM product AS p
  WHERE p.id = 1077) AS XML)
```

Convert a value extracted from an SQL scalar subquery into an XML value