



SQL SERVER DAY 2009

Using XML in SQL Server

Dr. Nico Jacobs
SQL Server Architect & Trainer

u2u

whoami



Born @ June,17 1974 (Gemini)



Grew up around the castle of Horst, @
SPR



Graduated as lic. Informatics @
KULeuven



PhD in machine learning @ KULeuven



Daddy @ home



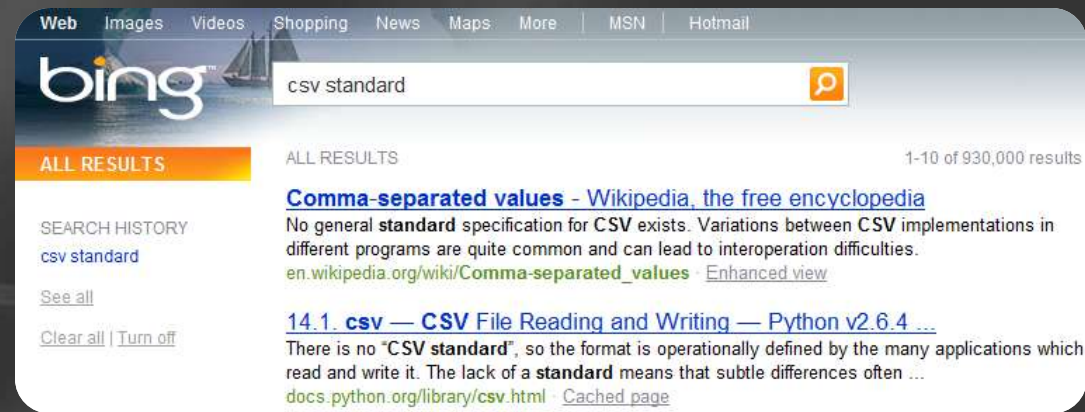
Since 2004 Nico@U2U.be

Agenda

- Why XML?
- Three XML Scenarios
- Getting XML out of tables
- Getting XML into tables
- Xml data type
- Querying and manipulating XML inside the database
- Four XML indexes
- The 30000 column table

Why XML?

- Data often transported, serialized, copied into data islands, ...
- CSV is not really well suited as platform independant data representation
 - No types
 - No constraints
 - No relationships
 - No standard
- XML = Lingua franca

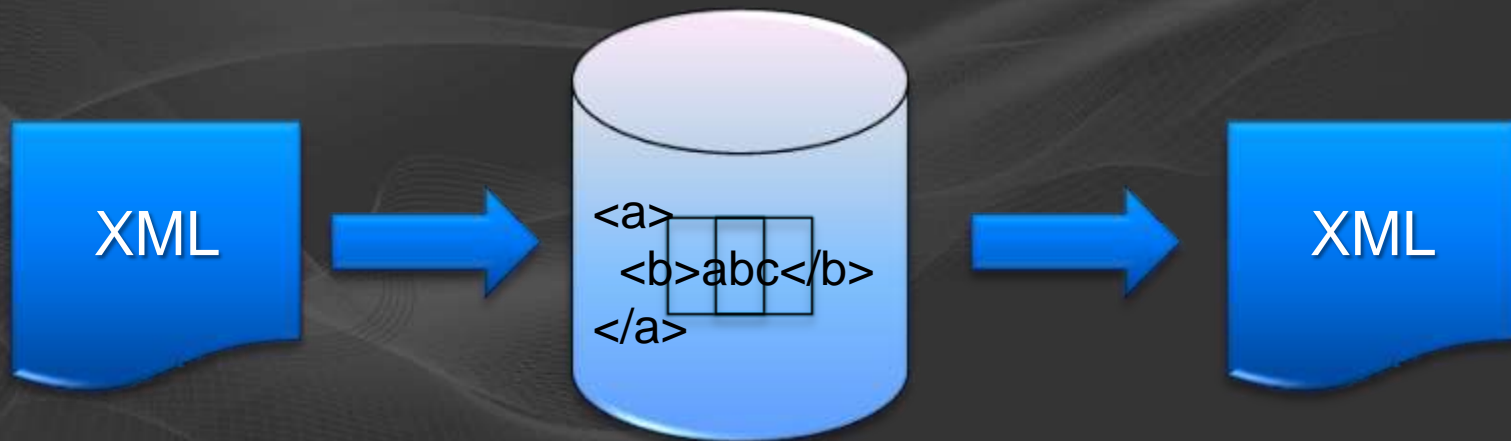


One slide intro into XML

- It is a `<emp>Markup</emp>` language
- It is extensible: just invent your own markers
- Markers are called `<name>elements</name>`
- Element can have uniquely named parameters: `<name fontsize="32">` attribute `</name>`
- **Xsd** schema = Type information and constraints
 - In separate file

Three XML scenarios

- We store relational data, but output XML
 - FOR XML
- We input XML into a relational table
 - OPENXML
- We store XML natively in SQL Server
 - XML type



Creating XML

- Why create XML from relational data in SQL Server?
 - Easy to write, easy to maintain
 - Automatically adopts to schema/query changes
 - Can generate XSD schema
 - Useful for all possible clients
 - SSIS, SQLCMD
- Why NOT in SQL Server?
 - Increasing server workload (scalability)
 - Power and familiarity client tools/languages

FOR XML

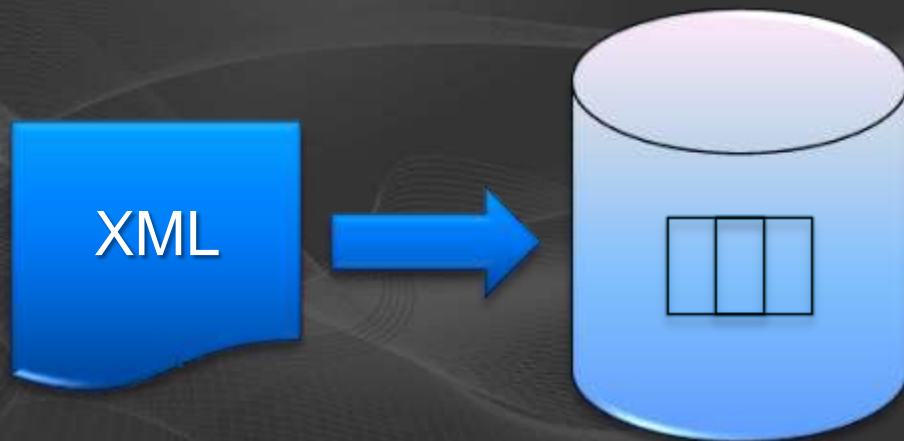
- Clause written after SELECT statement
- 4 modes available:
 - RAW: straight-forward mapping, every row is an element, every field is an attribute
 - AUTO: allows for hierarchical representation of joins
 - EXPLICIT: allows for more advanced hierarchical representation
 - PATH: offers a readable equivalent for explicit mode

FOR XML

demo

Parsing XML into table

- OPENXML is a table-valued function which queries an in-memory DOM
- sp_xml_preparedocument builds this in-memory DOM
- DOMs are memory-hungry, so use careful
- Clean-up the DOM as soon as possible



OPENXML

demo

Native XML type

- When?
 - Since SQL Server 2005
 - All editions except SQL Server CE
- Why?
 - XML as lingua franca for computer data
 - XML popular for semi-structured data (SSD)
 - SSD requires different treatment than BLOB
 - SQL has some SSD as well
 - EVENTDATA()
 - Service Broker

Native XML Type

- Native type: can be used for column, variable and parameter
- Can hold up to 2GB of data per entry
- Non-comparable type
 - No index key
 - No GROUP BY, ORDER BY, <,...
- Stores Infoset representation of XML
 - input \neq output
 - Semantics(input) = Semantics(output)

How to use

- Use XML native type only when
 - You need to query or modify XML in the database (instead of just storing it), or
 - You need to verify the XML against an XSD schema in the database
- Use `nvarchar(max)` in all other scenarios

Native XML type

demo

Query XML data

- 4 functions for querying XML:
 - XML → **exist** → bit
 - XML → **value** → scalar sql type
 - XML → **query** → XML
 - XML → **nodes** → table with 1 XML column

One slide intro into XPath

- XML = Tree of nodes: elements & attributes
- Xpath expression selects nodes with a folder-alike naming
- Special axes:
 - / child
 - // descendant
 - .. parent
- Many functions available as well
 - contains, floor, abs, count, ...
- W3c standard

exist and value demo

demo

One slide intro into XQuery

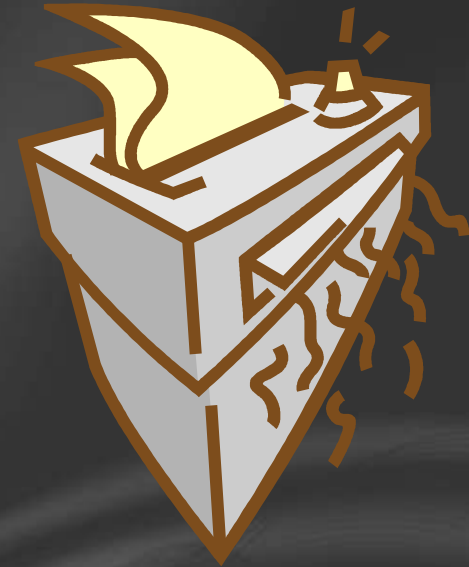
- **query** method uses XQuery
- XQuery = XPath + FLOWR (w3c)
- FLOWR is more expressive language:
 - **F**or: iterate over the nodes
 - **L**et: calculate derived values
 - **O**rders by: resort the nodes
 - **W**here: filter the nodes
 - **R**eturn: build the returned xml fragement

query demo

demo

Nodes and cross apply

- exist, value and query return one output row for every input row
- **nodes** is a table-valued function:
xml → nodes → table(xml)
- Every returned row meets Xpath expression
- What if we want to run **nodes** for every row of a table?
- **cross apply** is sort of TVF-loop:
for every row call TVF and append resulting rows



nodes and cross apply demo

demo

Modifying XML

- W3c only has DOM to modify XML
- We want declarative mechanism to modify XML
- Microsoft invented own method: **modify**
 - **insert**, **replace** and **delete**

modify demo

demo

One slide intro into namespaces

- Different applications, departments, ..., can mix information in a single XML file
- Every element can be tagged with a namespace such that queries can filter out unwanted nodes
- Namespace names are usually url-alike to avoid conflicts
- SQL Server can work with XML namespaces in prolog or explicit **WITH XMLNAMESPACES** statement

Working with XML namespaces

demo

Typed versus untyped XML

- XML allows any element, any attribute
- XSD schemas put constraints on:
 - Names of elements and attributes
 - Type of element and attribute data
 - Order in which elements occur
 - Frequency of element occurrence
- Limited support for inline DTD schemas

Typed versus untyped XML

- XSD schema is stored in SQL Server
XML SCHEMA COLLECTION
- Inserts run slower because of extra checks
- Can avoid data conversion at query time
- More compact storage possible
- Can force input to be XML document instead of XML fragment

Typed XML

demo

Indexing XML

- Without dedicated XML index SQL Server must shred XML fragment with every query
- **Primary XML index** shreds XML into table
- Every XML node becomes row in index
- Table must already have **primary key** to link XML with relational

PK	XML
1	<el at="val"> Hello</el>



PK	Path	Type	Value	Name
1	/el	E	Hello	el
1	@at/el	A	val	at

Secondary XML indexes

- We could make good indexes on the nodes table... but it is an internal table!
- Three predefined indexes on the primary XML index:
 - **FOR PATH**: (Path, Value)
 - **FOR PROPERTY**: (PK, Path, Value)
 - **FOR VALUE**: (Value, Path)

PK	Path	Type	Value	Name
1	/el	E	Hello	el
1	@at/el	A	val	at

XML Indexes

demo

XML in SQL Server

- Many features of SQL Server are using XML behind the scene:
 - eventdata function in DDL Trigger
 - Event notifications message types
 - Policies & policy evaluation outcome
 - Service broker message types
 - ...
- A handy new feature in SQL Server 2008: column sets

The 30000 column table

- Property bags can be large:
 - Size limitation (1024 columns, 8060 bytes) becomes realistic problem
 - Challenging to query and update
- In SQL Server 2008, a table can have up to 30000 **sparse columns**...
- ... which can be queried and updated with one **column set**

Column set

demo

Conclusions

- SQL Server remains relational database:
 - Parse XML into tables (**OPENXML**)
 - Export relational tables as XML (**FOR XML**)
- Semi-structured data can be queried when stored in the native XML type
 - Dedicated methods for querying
 - Non-w3c declarative method for modifying XML
 - Typed XML (XSD schemas)
 - Primary and three secondary XML indexes



join the conversation

SQLUG.BE