# xml indexes and xquery performance

Bob Beauchemin

# Overview

- **XQuery methods and processing**
  - Querying non-indexed XML
- **XML Indexes**
  - Kinds of XML Indexes
  - Creating and Maintaining
  - The XML Node Table
  - Querying indexed XML
- **Usage of XML Indexes**
  - Query plan changes
  - Which index for which use case
- **XQuery Performance Hints**

# XQuery and SQL

- **XQuery uses the relational query engine**
  - XQuery methods are SQL "system defined functions"
  - Integrated query plan is produced
  - XQuery operations added to (relational) query plan
    - Shows up in query plans as UDX
  - Types of UDX operators
    - Check UDX - validates XML being inserted
    - XQuery  Serializer UDX - serializes the query result as XML
    - XQuery  Data UDX - evaluates the XQuery data() function
    - XQuery  String  UDX - evaluates the XQuery string() function
    - XQuery  Contains UDX - evaluates the XQuery contains() function
    - FOR XML UDX – used in SELECT… FOR XML construction

# XQuery Methods and Processing

- **XQuery methods are part of a specific SQL statement**
  - Both SQL and XQuery go through a static phase
  - SQL static phase includes
    - SQL Parser
    - Static Typing - using SQL metadata
    - Algebrization
  - XQuery static phase includes
    - XQuery Parser
    - Static Typing - using (optional) XML Schema Collection
    - Algebrization
  - Trees are combined optimized
    - Static optimization of the combined logical and physical tree
    - Runtime optimization and execution of the tree - using relational & XML indexes

# The Node Table

- **To be useable by SQL, XML is decomposed into a “node table”**
  - Node table is a relational table
  - Visible in sys.internal_tables if materialized in primary XML index
  - Almost always used in XQuery
  - One row per node - possible overflow rows
- **Appears as "XmlReader with XPath Filter"**
  - Similar but not the same as System.Xml.XmlReader
- **Non-indexed XML – no node cardinality estimates**
  - Decomposition at execution time

# XML Query Plans

- **Query plans against table with XML column can be catagorized**
- **This query can result in either top-down plan**
  - Select rows in base table that qualify
  - Do XQuery against XML instance in qualifying
- **Or bottom-up plan**
  - Do XQuery first
  - Then join to base table

```
-- get qualifying (non-NULL) rows first?
-- or get InvoiceID 1003 first?
SELECT xml_col.exist('/Invoice/@InvoiceID[.= "1003"])
  FROM xml_tab
GO
```

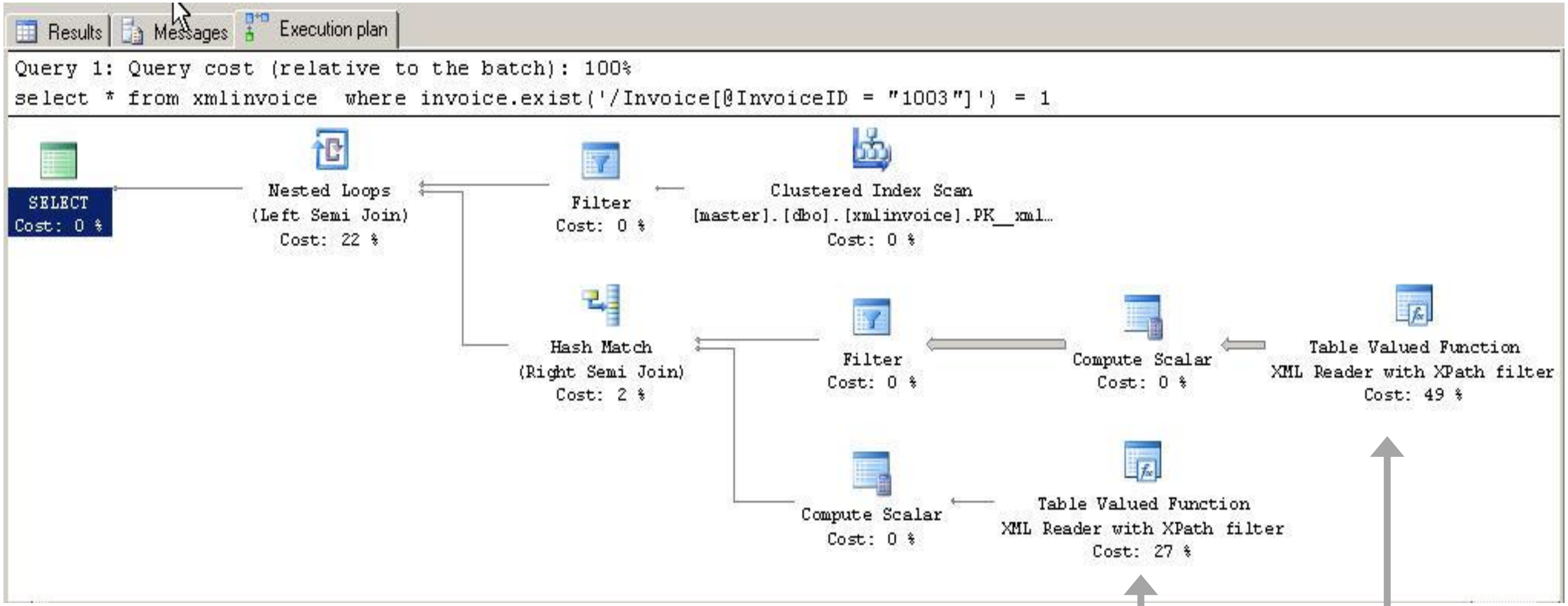
# Non-indexed XML and Evaluation Steps

- **Equivalent evaluations are converted to relational plans differently**
  - prefer predicates that use current node over last location step of path
  - This only works when the node returned is irrelevant (when using xml.exist)

```
-- this syntax preferred
SELECT xml_col.exist('/Invoice/@InvoiceID[.= "1003"])
  FROM xml_tab
GO

-- over this query
-- note: this does not produce the same result
-- except when using xml.exist
SELECT xml_col.exist('/Invoice[@InvoiceID = "1003"])
  FROM xml_tab
GO
```

# Query Plan Without Index



```
Query 1: Query cost (relative to the batch): 100%
select * from xmlinvoice  where invoice.exist('/Invoice[@InvoiceID = "1003"]') = 1
```

Multiple Evaluation Steps
Using XML Reader

# XML Indexes

- **XML INDEXes are special indexes on an XML column**
  - Optimizes XQuery operations on the column
  - Primary XML index must be created first
    - Before any additional XML indexes can be created
  - Table must have clustered index
    - XML column cannot be  the clustered index
  - Primary XML index provides cardinality estimates for the query optimizer

```
CREATE TABLE xml_tab (
  id integer primary key,
  doc xml)
GO
CREATE PRIMARY XML INDEX xml_idx on xml_tab (doc)
GO
```

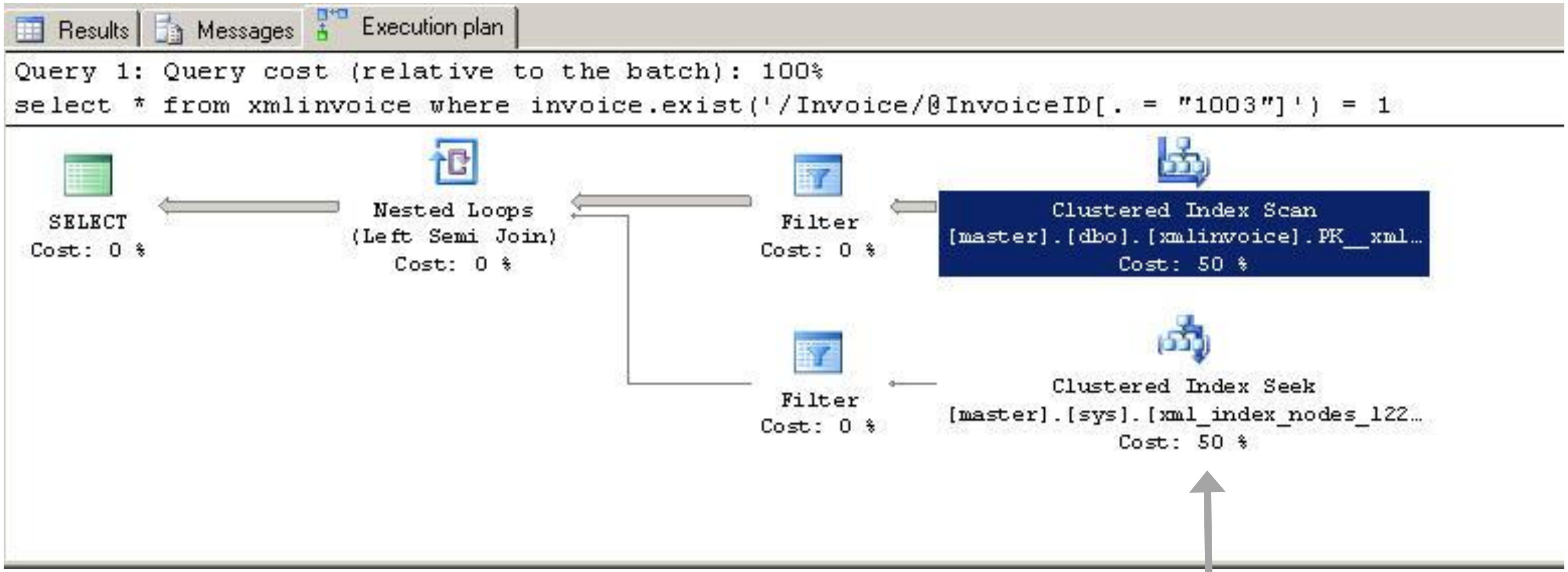
# XML Primary Index Details

- **Node table is materialized**
  - Base table clustered index, plus 12 columns of info
  - Column metadata visible with join of sys.indexes and sys.columns
  - Clustered index on
    - primary key (of base table)
    - id (node_id of the node table)
- **Most of the same create/alter options as SQL index**
- **Only available on columns (not variables)**
- **Space utilization is 2-5 times original data**
  - Can slow down inserts

# Columns in the Node Table

| Column Name | Column Description | Data Type |
|---|---|---|
| id | node identifier in ordpath format | varbinary(900) |
| nid | node name (tokenized) | int |
| tagname | tag name | nvarchar(4000) |
| taguri | tag uri | nvarchar(4000) |
| tid | node data type (tokenized) | int |
| value | first portion of the node value | sql_variant |
| lvalue | long node value (pointer) | nvarchar(max) |
| lvaluebin | long node value in binary (pointer) | varbinary(max) |
| hid | path (tokenized) | varchar(900) |
| xsinil | is it NULL (xsi:nil) | bit |
| xsitype | does it use xsi:type | bit |
| pk1 | primary key of the base table | int |

# Using Primary XML Index



```
Query 1: Query cost (relative to the batch): 100%
select * from xmlinvoice where invoice.exist('/Invoice/@InvoiceID[. = "1003"]') = 1
```

Clustered Index Seek
On Primary XML Index

# Secondary XML Indexes

- **Three specialized index types**
  - VALUE – optimizes content queries
  - PATH – optimizes when path selective
  - PROPERTY – top-down queries, i.e. rows selected first

```
CREATE TABLE xml_tab (
  id integer primary key,
  doc xml)
GO
CREATE PRIMARY XML INDEX xml_idx on xml_tab (doc)
GO
-- secondary index
CREATE XML INDEX path_idx on xml_tab (doc)
 USING XML INDEX xml_idx FOR PATH
GO
```

# Secondary XML Indexes

- **Secondaries are non-clustered indexes on the node table**
  - PATH is index on (HID, value)
    - HID is the path
  - VALUE is index on (value, HID)
    - Same columns as PATH, but reverse order
  - PROPERTY is index on (pk, HID, value)
    - Same as path but includes base primary key

# XML Index Maintenance

- **XML Indexes are efficiently maintained on update**
  - Key part - maintained like relational index
  - Path part - efficient because of ORDPATH representation
    - Only the changes part of the document is built
    - Semantics similar to XML modify
- **On insert, XML must be decomposed to build new index rows**

# Which Index?

- **If all indexes are present**
  - Primary index is "default" index
    - Used if no other index selective enough
    - Used for construction
      - with special value 'DESC'
  - VALUE is often used with wildcard queries
    - /SomeNode/@*[. = "special"]
    - (i.e. any attribute with the value special)
  - PATH used only if ragged documents
    - many different paths, high path selectivity
  - PROPERTY used with sparse attributes
    - when rows are selected first
    - when ‘//’ appears in the path
    - many different name-value pairs, different names

# Path and Axis Performance Hints

- **Queries that use parent axis generate extra query plan steps**
  - Avoid parent axis queries if possible
  - Use multiple CROSS APPLY steps rather than parent axis to get nodes at multiple nesting levels using xml.nodes
- **Move ordinals to the end of path expressions**
  - Ensure that the answer is equivalent
- **Avoid predicates in the middle of path expressions**
- **Specify a single root node in query**
  - Optimizer assumes XML can be a fragment

# More XQuery Performance Hints

- **Use xml.value rather than xml.query and a cast**
  - Almost always produce same answer
- **Casting in XQuery  is expensive**
  - Prohibits index use
- **Avoid functions that aggregate all text nodes if there is a single text node**
  - Prohibits  index use
- **Avoid multiple XQuery evaluations in a single SQL query**
  - Use SQL subqueries instead
- **Avoid applying inline functions to XQuery results**
- **When joining SQL and XML values,**
  - xml.exist and sql:column  - better
  - xml.value and comparison to SQL column - worse

# Constructing XML from SQL Values

- **XML Path can be quicker for construction**
  - □ Composing XML using XQuery most useful for transforming existing document, not construction from SQL values
  - □ Composing XML nodes from multiple relational columns better using FOR XML PATH
  - □ Composing  XML using XML DML insert node-at-a-time is slowest

# Optimizations in xml.modify

- **The xml.modify method uses**
  - sparse updates
  - sparse logging
- **Concurrency is still at row (document) level**
- **For best results**
  - insert new nodes as last sibling
  - insert/update nodes deeper in the hierarchy

# Review

- **XML Indexes speed up XQuery**
  - XQuery uses relational query engine
- **Primary XML Index creates node table**
  - Primary XML Index refers to clustered index on node table
  - Secondary are non-clustered index
- **Primary XML Index almost always useful**
  - 2-5 times size of document
- **Secondary XML Indexes use-case based**
  - Like SQL indexes
- **Phrasing of XQuery can effect performance**

# References

- XML Indexes in SQL Server 2005, Bob Beauchemin, MSDN Online
- XQuery Implementation in a Relational Database System, Shankar Pal, et al., Conference on VLDB
- Performance Optimizations for the XML Data Type in SQL Server, Shankar Pal, et al., MSDN Online
- SQL Server 2005 XQuery Performance Tips, Ward Pond
  - http://blogs.technet.com/b/wardpond/archive/2005/06/23/sql-server-2005-xquery-performance-tips.aspx
- MSDN Webcast: Making the Most of XQuery with SQL Server 2005, Michael Rys

For more in-depth online developer training visit

pluralsight

see what you can learn

on-demand content from authors you trust