Indexing XML Data Stored in a Relational Database

Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis, Vasili Zolotov

VLDB 2004

Presentation: Alex Bradley
Discussion: Cody Brown
CPSC 504 / November 2, 2009

Slides adapted from http://dblab.hangkong.ac.kr
Problem: how to add XML support to relational databases?

One solution (last paper): “shred” XML document into rows in a set of relational tables based on an XML schema definition

- Difficulty: XML is hierarchical, has recursive structure
- Also, document order must be preserved
- Reassembling result may require many joins (can be prohibitively expensive)
Introduction

- MS SQL Server 2005 takes different approach
- XML is a native data type for columns (stored as BLOBs)
- **XQuery** expressions embedded within SQL statements are used to query XML data
  - Query execution processes each XML instance at runtime. This becomes **expensive** if
    - The instance is **large in size**
    - The query is evaluated on a **large number of rows** in the table
  - Consequently, an indexing mechanism is required to speed up queries on XML blobs
XML Support in Microsoft SQL Server 2005

- XML as a native data type

Create TABLE DOCS (ID int primary key, XDOC xml)

- Node labeling using ORDPATH
  - Mechanism for labeling nodes in an XML tree
  - Preserves structural fidelity
  - Allows insertion of nodes anywhere without re-labeling
  - Independent of XML schemas
  - Permits efficient checking of ancestor-descendant relationships between nodes
<BOOK ISBN="1-55860-438-3">
    <SECTION>
        <TITLE>Bad Bugs</TITLE>
        Nobody loves bad bugs.
    </SECTION>
    <SECTION>
        <TITLE>Tree Frogs</TITLE>
        All right-thinking people love tree frogs.
    </SECTION>
</BOOK>

**Figure 1.** Sample XML data

- Hierarchical dot-separated labels assigned to nodes
  - Compressed binary form used internally
- Positive odd integers assigned initially
- Negative, even integers used for insertions

**Figure 2.** ORDPATH Node Label
XML Support in Microsoft SQL Server 2005

● Query Processing
  ▪ XML data type has methods that take XQuery expressions as arguments
    • `query()`: returns XML data type
    • `value()`: extracts scalar values
    • `exist()`: checks conditions on XML nodes
    • `nodes()`: returns a rowset of XML nodes that the XQuery expression evaluates to
  ▪ Query compilation produces single plan for both relational and XML parts of query
Running this on all table rows is expensive

- XDOC column value in each row must be shredded at runtime to evaluate the query
- We cannot determine which of the XML instances satisfies @ISBN="1-55860-438-3" without processing the XDOC values in all rows

→ Idea: speed up query processing by saving on parsing cost at runtime.

- Approach: Add primary and secondary indexes

Example of XQuery query

```xquery
SELECT ID, XDOC.query('for $s in /BOOK[@ISBN='1-55860-438-3']//SECTION return <topic>{data($s/TITLE)} </topic>')
FROM DOCS
```
Strategy: Materialize shredded XML in B+ tree (the primary index)

Two options available to query processor:

- **Top-down execution**: process rows of base table before those in XML index
- **Bottom-up execution**: targeted seeks/scans on XML index first, then back join with base table
Store “infoset” items of XML nodes in B+ tree

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDSPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1(BOOK)</td>
<td>1(Element)</td>
<td>Null</td>
<td></td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2(ISBN)</td>
<td>2(Attribute)</td>
<td>'1-55860-438-3'</td>
<td></td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td></td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4(TITLE)</td>
<td>1</td>
<td>‘Bad Bugs’</td>
<td></td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10(TEXT)</td>
<td>4(Value)</td>
<td>‘Nobody loves Bad bugs.’</td>
<td></td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5(Figure)</td>
<td>1</td>
<td>Null</td>
<td></td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6(CAPTION)</td>
<td>2</td>
<td>‘Sample bug’</td>
<td></td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td></td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4(TITLE)</td>
<td>1</td>
<td>‘Tree frogs’</td>
<td></td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>10(TEXT)</td>
<td>4</td>
<td>‘All right-thinking people’</td>
<td></td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>7(BOLD)</td>
<td>1</td>
<td>‘love’</td>
<td></td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>10(TEXT)</td>
<td>4</td>
<td>‘tree frogs’</td>
<td></td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>
## Primary XML Indexes

- Store “infoset” items of XML nodes in B+ tree

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>BOOK</td>
<td>Element</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
<td>ISBN</td>
<td>Attribute</td>
<td>‘1-55860-438-3’</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3</td>
<td>SECTION</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4</td>
<td>TITLE</td>
<td>1</td>
<td>‘Bad Bugs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10</td>
<td>TEXT</td>
<td>Value</td>
<td>‘Nobody loves Bad bugs.’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5</td>
<td>FIGURE</td>
<td>1</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6</td>
<td>CAPTION</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>3</td>
<td>SECTION</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.1</td>
<td>4</td>
<td>TITLE</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.3</td>
<td>10</td>
<td>TEXT</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.5</td>
<td>7</td>
<td>BOLD</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.5.7</td>
<td>10</td>
<td>TEXT</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2. ORDPATH Node Label**

---

Primary key of XML instance's row in base table (used for back join)
Query Compilation and Execution

- Consider the evaluation of the path expression

/BOOK[@ISBN='1-55860-438-3']/SECTION

SELECT SerializeXML (N2.ID, N2.ORDPATH)
FROM infosettab N1 JOIN infosettab N2 ON (N1.ID = N2.ID)
WHERE N1.PATH_ID = PATH_ID(/BOOK/@ISBN)
  AND N1.VALUE = '1-55860-438-3'
  AND N2.PATH_ID = PATH_ID(BOOK/SECTION)
  AND Parent (N1.ORDPATH) = Parent (N2.ORDPATH)

- SerializeXML (ID, ORDPATH)
  - Assembles the XML subtree rooted at the node(ID, ORDPATH) from the Infoset table

- Parent (Child-ORDPATH)
  - Returns the parent’s ORDPATH as the prefix of C-ORDPATH without the last component for the child
Query Compilation and Execution

- Cost of reassembly (SerializeXML) may be high
- In some circumstances, may be cheaper to operate on XML blob than on index
- Query optimizer must make decision based on cost estimation

```
SELECT SerializeXML (N2.ID, N2.ORDPATH)
FROM infosettab N1 JOIN infosettab N2 ON (N1.ID = N2.ID)
WHERE N1.PATH_ID = PATH_ID(/BOOK/@ISBN)
    AND N1.VALUE = '1-55860-438-3'
    AND N2.PATH_ID = PATH_ID(BOOK/SECTION)
    AND Parent (N1.ORDPATH) = Parent (N2.ORDPATH)
```
Secondary XML Indexes

- The primary XML index is clustered in document order
- Each path expression is evaluated by **scanning all rows** in the primary XML index for a given XML instance
- Performance **slows down** for **large** XML values

- **Secondary indexes** on the primary XML index optimize for certain classes of queries
  - Four introduced in this paper:
    - PATH/PATH_VALUE, PROPERTY, VALUE, Content indexing

- Secondary XML indexes help with **bottom-up evaluation**
  - Nodes found in secondary XML indexes can be **back joined** with primary XML index, enabling continuation of query execution with those nodes
  - Leads to significant performance gains
# Secondary XML Indexes

## Table: PATH

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1(BOOK)</td>
<td>1(Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
<td>2(ISBN)</td>
<td>2(Attribute)</td>
<td>‘1-55860-438-3’</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4</td>
<td>4(TITLE)</td>
<td>1</td>
<td>‘Bad Bugs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10</td>
<td>10(TEXT)</td>
<td>4(Value)</td>
<td>‘Nobody loves Bad bugs.’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5</td>
<td>5(Figure)</td>
<td>1</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6</td>
<td>6(Caption)</td>
<td>2</td>
<td>‘Sample bug’</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4</td>
<td>4(TITLE)</td>
<td>1</td>
<td>‘Tree frogs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>10</td>
<td>10(TEXT)</td>
<td>4</td>
<td>‘All right-thinking people’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>7</td>
<td>7(BOLD)</td>
<td>1</td>
<td>‘love’</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>10</td>
<td>10(TEXT)</td>
<td>4</td>
<td>‘tree frogs’</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>

- Primary key of XML instance's row in base table (used for back join)
- Helps with evaluation of path expressions over entire XML column (e.g. /BOOK/SECTION/TITLE)
## Secondary XML Indexes

### PATH_VALUE

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>1 (BOOK)</td>
<td>1 (Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td>2 (ISBN)</td>
<td>2 (Attribute)</td>
<td>‘1-55860-438-3’</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td></td>
<td>3 (SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td></td>
<td>4 (TITLE)</td>
<td>1</td>
<td>‘Bad Bugs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td></td>
<td>10 (TEXT)</td>
<td>4 (Value)</td>
<td>‘Nobody loves Bad bugs.’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td></td>
<td>5 (FIGURE)</td>
<td>1</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td></td>
<td>6 (CAPTION)</td>
<td>2</td>
<td>‘Sample bug’</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td></td>
<td>3 (SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td></td>
<td>4 (TITLE)</td>
<td>1</td>
<td>‘Tree frogs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td></td>
<td>10 (TEXT)</td>
<td>4</td>
<td>‘All right-thinking people’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td></td>
<td>7 (BOLD)</td>
<td>1</td>
<td>‘love’</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td></td>
<td>10 (TEXT)</td>
<td>4</td>
<td>‘tree frogs’</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>

- Helps with searches for a path + value match (e.g. /BOOK/SECTION[TITLE=”Tree frogs”])
### Secondary XML Indexes

#### PROPERTY

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>BOOK</td>
<td>1(Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
<td>ISBN</td>
<td>2( Attribute)</td>
<td>‘1-55860-438-3’</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3</td>
<td>SECTION</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4</td>
<td>TITLE</td>
<td>1</td>
<td>‘Bad Bugs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>10</td>
<td>TEXT</td>
<td>4( Value)</td>
<td>‘Nobody loves Bad bugs.’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>5</td>
<td>FIGURE</td>
<td>1</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>6</td>
<td>CAPTION</td>
<td>2</td>
<td>‘Sample bug’</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3</td>
<td>SECTION</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4</td>
<td>TITLE</td>
<td>1</td>
<td>‘Tree frogs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>10</td>
<td>TEXT</td>
<td>4</td>
<td>‘All right-thinking people’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>7</td>
<td>BOLD</td>
<td>1</td>
<td>‘love’</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>10</td>
<td>TEXT</td>
<td>4</td>
<td>‘tree frogs’</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>

- Helps find (possibly multi-valued) properties of object with known ID and PATH_ID

Primary key of XML instance's row in base table (used for back join)
### Secondary XML Indexes

- **Primary key of XML instance's row in base table** (used for back join)

<table>
<thead>
<tr>
<th>ID</th>
<th>ORDPATH</th>
<th>TAG</th>
<th>NODE_TYPE</th>
<th>VALUE</th>
<th>PATH_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1(BOOK)</td>
<td>1(Element)</td>
<td>Null</td>
<td>#1</td>
</tr>
<tr>
<td>1.1</td>
<td>2</td>
<td>2(ISBN)</td>
<td>2(Attribute)</td>
<td>’1-55860-438-3’</td>
<td>#2#1</td>
</tr>
<tr>
<td>1.3</td>
<td>3</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.3.1</td>
<td>4</td>
<td>4(TITLE)</td>
<td>1</td>
<td>’Bad Bugs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.3.3</td>
<td>5</td>
<td>10(TEXT)</td>
<td>4(Value)</td>
<td>’Nobody loves Bad bugs.’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.3.5</td>
<td>6</td>
<td>5(Figure)</td>
<td>1</td>
<td>Null</td>
<td>#5#3#1</td>
</tr>
<tr>
<td>1.3.5.1</td>
<td>7</td>
<td>6(CAPTION)</td>
<td>2</td>
<td>’Sample bug’</td>
<td>#6#3#1</td>
</tr>
<tr>
<td>1.5</td>
<td>3</td>
<td>3(SECTION)</td>
<td>1</td>
<td>Null</td>
<td>#3#1</td>
</tr>
<tr>
<td>1.5.1</td>
<td>4</td>
<td>4(TITLE)</td>
<td>1</td>
<td>’Tree frogs’</td>
<td>#4#3#1</td>
</tr>
<tr>
<td>1.5.3</td>
<td>5</td>
<td>10(TEXT)</td>
<td>4</td>
<td>’All right-thinking people’</td>
<td>#10#3#1</td>
</tr>
<tr>
<td>1.5.5</td>
<td>6</td>
<td>7(BOLD)</td>
<td>1</td>
<td>’love’</td>
<td>#7#3#1</td>
</tr>
<tr>
<td>1.5.7</td>
<td>7</td>
<td>10(TEXT)</td>
<td>4</td>
<td>’tree frogs’</td>
<td>#10#3#1</td>
</tr>
</tbody>
</table>

- Helps when we're looking for a data value but have a wildcard in the path (e.g. /BOOK/SECTION[FIGURE/@*="Sample Bug"]))
Optimize text search (IR) over XML data

- **Full text index** over XML column works well for traditional IR
  - Not optimal if we want to search for a certain word in the context of a specific XML element

We want...

- to exploit indexes over XML infoset
- to have finer granularity than text nodes
  - VALUE index does not help us locate individual words efficiently

Solution: **Word break index** has same structure as infoset table, but text nodes are broken into words according to XML whitespace
Considering the amount of indexes introduced in this section (primary, PATH, PROPERTY, VALUE, content), does this emphasis surprise or worry you?

What's the significance? Are so many needed?

With respect to the space and maintenance, are these resources justified? How about with today’s databases?
XMark

- XML query benchmark that models an auction scenario (slightly modified from original)
- Provides tool to generate XML data and 20 queries over the data

Experimental setup

- Query workload run on two sets of generated XML data at scale factors 0.5 (60 MB raw XML) and 30 (3.35 GB raw XML)
- Measure speedup given by indexing techniques compared to unindexed blobs
Results (scale factor 30)

Query Improvement Ratio

<table>
<thead>
<tr>
<th>Query</th>
<th>PRIMARY</th>
<th>PATH_VALUE</th>
<th>PROPERTY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td></td>
<td></td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>Q5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q15</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Q16</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Conclusion

- Introduced **primary XML index**
  - B+ tree containing Infoset items of XML nodes
- **Secondary indexes** on the primary index
  - Improve the performance of common classes of queries
- Experimental results
  - Primary and secondary indexes give significant improvements for a wide range of queries (though not all)
Now that we have seen two separate approaches to dealing with XML data:

- **First paper:** decomposing XML into tables, and using the power of the relational engine, or
- **Second paper:** storing XML as BLOBs and using index intensive techniques to speed retrieval

What advantages (especially with implementation and performance) do you think each method has?

Which do you think you would personally prefer? Why?