Indexing XML Data Stored in a Relational Database

Shankar Pal, Istvan Cseri, Oliver Seeliger, Gideon Schaller, Leo Giakoumakis, Vasili Zolotov
VLDB, 2004

Presented by: Meeta Mistry
Discussion: Haoran

XML: Extensible Mark-up Language

- A way of describing data, but can contain the data too
- The extra information, about the text's structure or presentation, is expressed using **markup**, which is intermingled with the primary text.

- XML is increasingly being used in enterprise applications and has motivated the need for native XML support within relational databases

```
<BOOK ISBN="1-55860-438-3">
    <SECTION>
        <TITLE>Bad Bugs</TITLE>
        Nobody loves bad bugs.
        <FIGURE CAPTION="Sample bug"/>
    </SECTION>
    <SECTION>
        <TITLE>Tree Frogs</TITLE>
        All right-thinking people
        <BOLD> love </BOLD> tree frogs.
    </SECTION>
</BOOK>
```

Figure 1. Sample XML data

XML Support in Relational Databases

- Existing solutions convert XML into a relational format: 'shredding approach'
  - Based on an XML schema definition
  - decompose XML instances; discards the XML tags, and stores the element and attribute values in regular relational tables
  - relative order of elements in the document are lost
- A single XML insert can result in a substantial number of relational inserts into a potentially large number of tables
  - During tuple oriented query processing this would require a large number of joins – very very expensive!

XML as a Native Datatype

- XML documents can be stored in the XML column as large binary objects (BLOB)
- XML documents are stored and manipulated in a parsed format, such as the XML Infoset or the XQuery/XPath Data Model
  - requires XML parsing but no mapping from the XML data model to a different data model.
- Parsed format serves as an indexing mechanism which can speed up query execution on XML BLOBs

Node Labeling using ORDPATH

```
<BOOK ISBN="1-55860-438-3">
    <SECTION>
        <TITLE>Bad Bugs</TITLE>
        Nobody loves bad bugs.
        <FIGURE CAPTION="Sample bug"/>
    </SECTION>
    <SECTION>
        <TITLE>Tree Frogs</TITLE>
        All right-thinking people
        <BOLD> love </BOLD> tree frogs.
    </SECTION>
</BOOK>
```

Figure 1. Sample XML data

- An internal representation is used for processing and storage on disk, which reflects the hierarchical structure of the XML data
- A mechanism for labeling nodes in an XML tree, which preserves structural fidelity
- Encodes a parent-child relationship

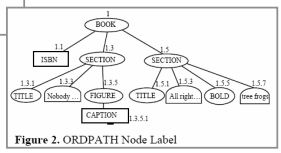


Figure 2. ORDPATH Node Label

Discussion

- The authors leave all negative and even integers out from their numbering on the ORDPATH. Does this seem like enough? Too much?

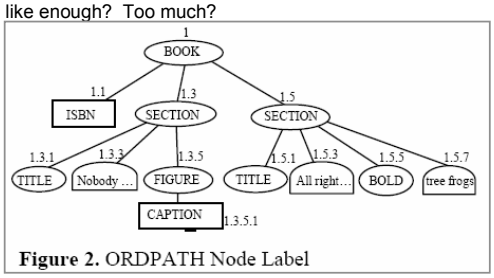
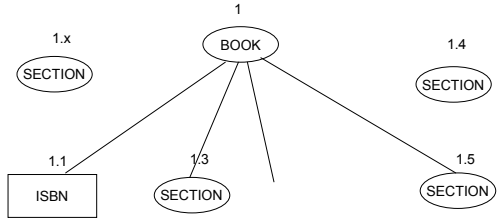


Figure 2. ORDPATH Node Label

Discussion (cont…)

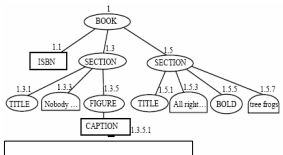
- Read the second paragraph after Figure 2.
- How does insertion work?



Primary XML Index

- For each XML instance in base table, the index creates several rows of data
  - The number of rows in the index is approximately equal to the number of nodes in the XML binary large object.
- Generate a subset of fields
- Primary key = (primary key ID, ORDPATH)

Primary XML Index



Infoset Table

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

Query Compilation and Execution

- An XQuery expression is translated into relational operations on the Infoset table
- Consider the evaluation of the path expression
  'Retrieve section titles in the book with the specified ISBN':

```
/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)
```

•Note that the primary XML index is not used when retrieving a full XML instance.

• increased I/O cost + serialization cost of converting back to XML makes it cheaper to retrieve XML BLOB from base table

Secondary XML Indexes

- Primary index may not provide the best performance for queries based on path expressions
- Performance slows down for large XML values.
  - all rows in the primary XML index corresponding to an XML BLOB are searched *sequentially* for large XML instances – slow!
- Having a secondary index built on the path values and node values in the primary index can significantly speed up the index search
  - PATH(PATH_VALUE), PROPERTY, VALUE, Content indexing

Secondary XML Indexes

- Secondary XML indexes help with bottom-up evaluation
  - After the qualifying XML nodes have been found in the secondary XML indexes, a back join with the primary XML index enables continuation of query execution with those nodes.
  - This yields significant performance gains.

XMark: An XML Benchmark Project

- An XML query benchmark that models an auction scenario
- Used to measure performance improvements found with different XML indexes compared with the BLOB case
  - Note disk space consumption: 345MB for primary index tables 101MB for secondary indexes – cost efficient?
- Results:
  - Table displays the factor by which the choice of an XML index speeds up queries relative to the BLOB case
  - Overall performance gains; thus XML indexes benefit the workload significantly

Conclusions

- Introducing an approach that supports interoperability between relational and XML data within the same database
- Primary XML index
  - Encodes Infoset items of XML nodes
- Avoided the approach of decomposition
- Secondary XML indexes yields significant performance gains
- Performance measurements show that indexing is highly effective for a wide class of queries

Discussion

- Assume you had a OO-Database, what about mapping XML to OO-Database?