Integrated Querying of XML Data in RDBMSs

Albrecht Schmidt  Stefan Manegold  Martin L. Kersten
al@cs.auc.dk  S.Manegold@cwi.nl  M.L.Kersten@cwi.nl

Aalborg University  CWI
Aalborg, Denmark  Amsterdam, The Netherlands
Contents

• Introduction: XML and databases

• System Architecture

• Internal XML Storage Model

• Query Processing

• Summary
XML and Databases: Issues

- Role of database: index, metadata, store, transactions, security
- Physical mapping: semantics, transaction and access granularity
- Impact of mapping on query processing and optimization?
- Cost estimation? Search space structure?
- What is the framework of XML applications?
Approaches to XML Document Storage

- Flat-files or Large Objects
- Relational mapping
- Object-oriented mapping
- Native storage
System Architecture: Ideas/Goals

- Rely on approved RDBMS technology to store and query large collections of XML documents

- Clear separation of concepts:

  SQL: - Store and query “native” relational data “as usually”
     - Publish relations as XML

  XML: - Store documents logically as “black-box” user-defined datatype
     - XPath/XQuery are implemented as user-defined functions
     - Query results must adhere to given XML Schema for automatic translation into SQL tables
     - Store documents physically in shredded schema
System Architecture: Overview

SQL
publish relations as XML

XQuery
extract structure with XQuery/XPath

SQL Schema
URI–document pairs

XML Schema

Stefan Manegold XML & RDBMS: System Architecture SAC-2003
### Translation between SQL tables and XML documents

**Relation1**

<table>
<thead>
<tr>
<th>att1</th>
<th>att2</th>
<th>att3</th>
<th>att4</th>
</tr>
</thead>
<tbody>
<tr>
<td>val1</td>
<td>val2</td>
<td>val3</td>
<td>val4</td>
</tr>
<tr>
<td>val5</td>
<td>val6</td>
<td>val7</td>
<td>val8</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mapping:

```xml
<?xml version='1.0'>
<relation name='Relation1'>
  <tuple>
    <att name='att1'> val1 </att>
    <att name='att2'> val2 </att>
    ...
  </tuple>
</relation>
```
Internal XML Storage Model: “Monet XML”*

- Three types of binary relations $R$: oid × oid, oid × string, and oid × int. We call a tuple $(h, t) \in R$ association.

- Elements are assigned unique OIDs.

- The syntax tree is decomposed into associations:
  - element relationships ($\subseteq$ oid × oid),
  - attribute relationships ($\subseteq$ oid × string),
  - topological information (i.e., ranks) ($\subseteq$ oid × int).

- Semantic clustering achieved by storing all associations with the same path in one relation $\implies$ one relation per “tag sequence”.

- Mapping is lossless.

*) MonetDB: monetdb.cwi.nl
Example Document

<image key="18934" source="/cdrom/images1/23493.jpeg">
  <date> 999010530 </date>
  <colors>
    <histogram> 0.399 0.277 0.344 </histogram>
    <saturation> 0.390 </saturation>
    <version> 0.8 </version>
  </colors>
</image>
Schema Tree and Mapping of Sample XML Document

All Documents

R1(oid,oid): /image
R2(oid,str): /image[key]
R3(oid,str): /image[source]
R4(oid,oid): /image/date/PCDATA
R5(oid,str): /image/date/PCDATA
R6(oid,str): /image/date/PCDATA
R7(oid,oid): /image/PCDATA
R8(oid,oid): /image/PCDATA
R9(oid,oid): /image/PCDATA
R10(oid,oid): /image/PCDATA
R11(oid,oid): /image/PCDATA
R12(oid,oid): /image/PCDATA

Stefan Manegold  XML & RDBMS: Storage Model  SAC-2003
Querying with a Relational Engine

• Problem:
  XML query algebras contain primitives that are hard to model in relational engines, especially wildcards in path expressions.

• Solution:
  Use Path Summary to remove wildcards from queries.

• Benefit:
  Pass the result on to query optimizer and execution engine.
Removing Wildcards

- Example: Retrieve all keys and the corresponding color features.

\[
\text{image}([\text{key}] \bowtie_{hd=hd} (\text{colors} // \text{cdata}[\text{string}])))
\]

Translates to:

\[
\text{R1} \bowtie_{tl=hd} (\text{R2} \bowtie_{hd=hd} (\text{R6} \bowtie_{tl=hd} \text{R7} \bowtie_{tl=hd} \text{R8} \\
\cup \text{R6} \bowtie_{tl=hd} \text{R9} \bowtie_{tl=hd} \text{R10} \\
\cup \text{R6} \bowtie_{tl=hd} \text{R11} \bowtie_{tl=hd} \text{R12}))
\]
Summary

- Method to integrate SQL tables and XML documents
- Can be implemented on existing RDBMS
- Re-uses existing SQL query optimizer and execution engine
- Key ideas:
  - Model XML documents as user-defined types
  - Shredded physical schema allows fast associative access
  - Dedicated XML Schema allows translation between SQL tables and XML documents
  - Elimination of wildcards in path expressions