Relational Databases for Querying XML Documents: Limitations and Opportunities

Manage XML Data using Relational DBMS
- Trend: XML represents most data on WWW
- Need: Answer a lot of XML Queries
  - Easy/Auto
  - Effective
  - Efficient
- How: Use Relational model to solve XML
- Why: Relational is Powerful, Let’s Reuse it

Background: XML Review
- XML
  - eXtensible Markup Language
    - Extended from SGML
  - Hierarchical / Semi-structured Data (sets)
  - Self-describing
    - Program can interpret data
  - Emerging as Standard in Web Applications

DTD
- Schema for XML
- Helps applications program interpret meaning of XML Data
- Pattern matching
  - * means zero or more
  - + means 1 or more
  - ? means zero or 1
- >= 1 element can be root

Background: XML Review
- XML
  - eXtensible Markup Language
  - Hierarchical / Semi-structured Data (sets)
  - Extended from SGML
  - Self-describing
  - Emerging as Standard in Web Applications

DTD
- Schema for XML
- Helps applications program interpret meaning of XML Data
- Pattern matching
  - * means zero or more
  - + means 1 or more
  - ? means zero or 1
- Root is not always the same element
xml databases

- form pairs
- talk / discuss about an interesting idea connected to topic
- share your ideas with the class

Focus of Presentation

Will talk about only 1 and 4

<table>
<thead>
<tr>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Write DTD</td>
<td>3</td>
</tr>
<tr>
<td>2. Instance each XML doc</td>
<td>3</td>
</tr>
<tr>
<td>3. Query XML-QL</td>
<td>4</td>
</tr>
</tbody>
</table>

How: Use Relational for XML : 4 things

- Convert from XML to Relational
  - Method: In-lining
  - Section: 3
  - 1. Schema DTD >= 1 Columns / Tables
  - 2. Instance each XML doc >= 1 Rows / Tuples
  - 3. Query 1 XML-QL/Lorel 1 SQL Translation

- Convert Query results from Relational back to XML
  - Method: Various
  - Section: 5
  - 4. Result > 1 tuples 1 XML Doc

How : 1. Schema

<table>
<thead>
<tr>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Write DTD</td>
<td>3</td>
</tr>
<tr>
<td>2. Instance each XML doc</td>
<td>3</td>
</tr>
<tr>
<td>3. Query XML-QL</td>
<td>4</td>
</tr>
</tbody>
</table>

Schema : Start with a DTD

XML Schema, e.g. DTD

- &lt;ELEMENT book (booktitle, author)
- &lt;ELEMENT article (title, author, copyright, abstract)
- &lt;ELEMENT section (sectiontitle, EMRY)
- &lt;ELEMENT author (authorID implicit)
- &lt;ELEMENT section (sectiontitle, EMRY implicit)
- &lt;ELEMENT booktitle (title)
- &lt;ELEMENT author (authorID implicit)
- &lt;ELEMENT copyright (copyright)
- &lt;ELEMENT abstract (abstract)
- &lt;ELEMENT sectiontitle (sectiontitle)
- &lt;ELEMENT booktitle (title)
- &lt;ELEMENT author (authorID implicit)

Schema: Turn DTD to tables

XML Schema, e.g. DTD

- &lt;ELEMENT book (booktitle, author)
- &lt;ELEMENT article (title, author, copyright, abstract)
- &lt;ELEMENT section (sectiontitle, EMRY)
- &lt;ELEMENT author (authorID implicit)
- &lt;ELEMENT section (sectiontitle, EMRY implicit)
- &lt;ELEMENT booktitle (title)
- &lt;ELEMENT author (authorID implicit)
- &lt;ELEMENT copyright (copyright)
- &lt;ELEMENT abstract (abstract)
- &lt;ELEMENT sectiontitle (sectiontitle)
- &lt;ELEMENT booktitle (title)
- &lt;ELEMENT author (authorID implicit)

Figure 2

Figure 12
Schema: DTD to ER

Given an XML Schema, intuitive to map
- Each node element -> an ER Entity / Tables
- XML attributes -> ER attributes / Columns

But,
- Child node elements can be elements themselves
- Can’t map directly them into ER Attributes

Schema: DTD to Relational: In-lining

1. Simplify DTD
2. Make DTD Graph from simplified DTD
3. Use DTD graph and create Tables: 3 ways
   1. Basic
   2. Shared
   3. Hybrid

Schema: Simplify DTDs

- Simplify DTDs
  1. Fig 5. Flattening
  2. Fig 6. Simplification
  3. Fig 7. Grouping

- Nothing gets deleted/modified/added
- Easier to make DTD graph

Schema: DTD to Relational

1. Simplify DTD
2. Make DTD graph from simplified DTD
   - First step to map XML node elements/attributes to relational tables/columns
   - Elements appear exactly once
   - Attributes and operators appear as many times as they appear in the DTD
3. Create Relations from DTD graph

DTD Schema to DTD Graph

Node: elements/attributes/operators
Edge: arrow from Parent to Child
DTD Graph: more than 1 XML Doc

- DTD describes > 1 type of XML Doc
- Book can be root in 1 XML Doc
- Article can be root in another

Schema: DTD to Relational

1. Simplify DTD
2. Make DTD graph from simplified DTD
3. Create Relations from DTD graph, 3 ways
   1. Basic
   2. Shared = Basic + ...
   3. Hybrid = Shared + ...
- Like buying a car
- Shared XOR Hybrid option packages

Basic In-lining: 4 steps

- Input: an Element graph, element is root
- Output: a set of Relations

4 steps algorithm, apply for each element graph
- e.g. Editor

DTD graph | Editor Element graph

Element graph is sub-graph of DTD graph: e.g. Editor element graph, Editor is root
**Basic1**: Create relation for each root

**Basic2**: relations for sets/recursions

**Basic2**: relations for sets/recursions

**Basic3**: inline rest of attributes

**Basic4**: connect using foreign keys

**Basic In-lining**: Output : 14 tables
**Basic In-lining: Output : 14 tables**

- monograph.author.name.surname
- Attribute name equals to path name

**Basic in-lining: pros & cons**

**Pros**
- Easy to do certain queries, such as "list all authors of a book"
  - ...in-lined as attributes

**Cons**
- Large number of relations
- ...in-lined as attributes

**Shared in-lining : 3 more steps**

**Input:**
- Relations from Basic

1st additional step
- Identify nodes spread across multiple relations in Basic
- Separate these shared nodes into new relations
- e.g. book.author and article.author into author
- Reduce replication

2nd and 3rd additional steps
- Output: every node in exactly one relation, either
  - separate relation or
  - in-lined into parent
- Merge mutually recursive
  - e.g. editor & monograph
- In-line a child node if it has no children
  - e.g. first and last name into author
  - fewer join operations

**Shared1: Extract shared columns**

e.g. Author 5 to 1
- Extract book, editor.monograph, article.author, author into single Author

**Shared2: Merge mutually recursive**

e.g. monograph 2 to 1, merge editor & editor.monograph into monograph

**Shared3: in-line when no child node**

e.g. merge first and last name, name, address into Author.name.first_name, Author.name. etc.
**Shared3: in-line when no child node**

- e.g. merge `firstname`, `lastname`, `name`, `address` into `Author.name.firstname`, `Author.name` etc.

**Pros**
- Fewer tables -> `Shared`
- 1 table vs. 5 tables for `Basic` for `Author`
- List all authors having first name `Jack`

**Cons**
- More joins than `Basic` if we start at a particular element node.

**Shared In-lining: Output : 5 tables**

**Hybrid In-lining**

- Reverse `Shared` a little ->
- In-line sub-element as long as it’s not a SET (*) or Recursive
  - e.g. `Title -> article.title & monogrph title`
- Even when in-degree > 1
  - e.g. `Author -> book.author + article.author`
Hybrid In-lining: Output: 4 tables

Pros:
- In-lining further reduces joins
- As good as Shared in most cases
- Better than Shared in some cases

Cons:
- Higher degree of in-lining could cause more SQL queries to be generated
  - Have to Balance between Basic and Shared
- Maintain the order of sets as in XML -> add column

Summary: 3 types of In-lining
1. Schema: XML -> Relational
   1. Basic In-lining
   2. Shared In-lining
   3. Hybrid In-lining

- Basic In-lining: DTD element graph
- Shared In-lining: extract/merge shared tables
- Hybrid In-lining: in-line for simplicity
Focus of Presentation

Finished on 1, now move onto to 4

<table>
<thead>
<tr>
<th>Focus of Presentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finished on 1, now move onto to 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From XML</th>
<th>To Relational</th>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Schema</td>
<td>DTD</td>
<td>Tables/Columns</td>
<td>In-lining</td>
</tr>
<tr>
<td>2. Instance</td>
<td>XML-QL, Lorel</td>
<td>1 XML</td>
<td>Translation</td>
</tr>
<tr>
<td>3. Query</td>
<td>XML-QL, Lorel</td>
<td>2 XML</td>
<td>Translation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From Relational</th>
<th>To XML</th>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Query Result</td>
<td>Projected Columns</td>
<td>Selected Rows</td>
<td>various</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>From Relational</th>
<th>To XML</th>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Query Results</td>
<td>Projected Columns</td>
<td>Selected Rows</td>
<td>Node Elements Attributes Sets Parent/Child</td>
</tr>
</tbody>
</table>

### How : 4. Query Results

<table>
<thead>
<tr>
<th>From Relational</th>
<th>To XML</th>
<th>Method</th>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>4. Query Results</td>
<td>Projected Columns</td>
<td>Selected Rows</td>
<td>Node Elements Attributes Sets Parent/Child</td>
</tr>
</tbody>
</table>

### Relational Results to XML

3 categories

1. Simple Structuring -> fill in tag values
2. Tag Variables -> tag names
3. Grouping -> group sets / hierarchies

### 4. Results : 1. Simple Structuring

- Fill in **Firstname** tag value

Query → Tuple

XML

### 4. Results : 2. Tag Variables

- **Book** tag vs. **Monograph** tag

Query → Tuple

XML
4. Results : 3. Grouping
• Titles by Darwin: 2 books + 1 monograph
• Group together under same author <name>
• Book titles -> Set elements of <book> tag

Query Tuple XML

4. Results : 3. Grouping
• Titles by Darwin: 2 books + 1 monograph
• Group together under same author <name>
• Book titles -> Set elements of <book> tag

Query Tuple XML

Side Benefit
Interoperability and Reuse
Mapping Relational/XML makes it easier to
• Integrate data across multiple DBMS in WWW
  – Relational as popular local persistent storage (Oracle)
  – XML as data transmission protocol (SOAP)
  – XML is OS-independent, self-describing
  – XML can get thru firewalls, relational data don’t
• Re-use existing relational data

Conclusion
Managing XML Data using the Relational Model
4 things to convert
• XML -> Relational: Schema, Instance, Query
• Relational -> XML: Query Results

Limitations
• Approach is not perfect
• Simple XML query requires Many SQL queries, or few SQL queries but Many joins
• Open question whether semi-structure querying works more efficiently

Opportunities
• Reusing a mature tech
• High performance
• Seamless querying
• Enable interoperability
• Data Reuse
Which solution might be more successful in the future? Justify!

- storing XML in relational databases
- having a dedicated XML semi structured system

Merci

James and Roland

DTD to Relational Schema

- XML is powerful when there is agreement among inter-operating applications
- Vast majority of the Internal files as XML docs conforming to DTDs
- Simplify DTDs
  - E.g. (e1, e2)* into e1*, e2*

Inlining

- Having “as many descendants of an element as possible into a single relation”
- No correspondence between elements and attributes of the ER-model
- Excessive fragmentation
- Basic / Shared / Hybrid Inlining