A Journey towards More Efficient Processing of XML Data in (O)RDBMS

Irena Mlynkova
irena.mlynkova@mff.cuni.cz

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Prague, Czech Republic
Introduction

• XML = a standard for data representation and manipulation
• Growing demand for efficient managing and processing of XML data

⇒ A natural alternative: To exploit tools and functions of (object-)relational database management systems ((O)RDBMS)

(−) XML trees vs. flat relations ⇒ inefficiency
(+ ) Long theoretical and practical history, mature technology

⇒ The techniques should be further enhanced
Goals of This Presentation

Proposal of improvement of XML processing based on (O)RDBMS

- Overview and classification of existing approaches
- Motivation for improvements
- Proposal of improvement of user-driven methods
- Experiments
- Conclusion
1. Overview of existing approaches
2. Motivation for improvements
3. Proposed improvement
4. Experiments
5. Conclusion
Database-Based XML Processing Methods (1)

Key concern: Choice of the most efficient XML-to-relational mapping strategy

- **Fixed** – predefined set of mapping rules and heuristics
  - Generic vs. schema-driven
- **Adaptive** – adapt the target schema to intended usage
  - Cost-driven
- **User-involving** – storage decisions in hands of users
  - User-defined vs. user-driven
Database-Based XML Processing Methods (2)

- **Generic vs. schema-driven** – omitting / exploiting XML schema
  - Straightforward mapping
- **Cost-driven** – search a space of possible mappings and choose the one which conforms the target application most = the least "expensive"
  - Application: sample XML data, XML queries
- **User-defined vs. user-driven** – the amount of user involvement
  - User-driven = a type of adaptivity
    - Schema is adapted to the annotations
User-Driven Methods: Shortcomings and Improvements

- Default mapping strategy is always fixed
- Systems are able to store schema fragments in various ways ⇒ adaptive enhancing is natural
- Weak exploitation of user-given information
  - Annotations are just directly applied
  - Idea: Annotations = "hints" how a user wants to store particular XML patterns ⇒
    - We search for similar fragments
    - We use the knowledge in adaptive enhancing

⇒ General idea: Emphasis on user-given information

Content

1. Overview of existing approaches
2. Motivation for improvements
3. Proposed improvement
4. Experiments
5. Conclusion
Why User-Given Information? (Example 1)

- Situation: Documents with XHTML fragments
- Problem: Shredding into tables = inefficient fragment reconstructions
  - XHTML DTD contains complete graphs on up to 10 nodes
- What if the real complexity is much simpler?
  - Statistical analysis: Yes, it is much simpler!

⇒ Simpler storage strategy (CLOB)

Why User-Given Information? (Example 2)

- Situation: Updatability of data vs. fast query evaluation
- Problem: Amount of mutual relationships information
  - Fast querying $\Rightarrow$ additional indices, numbering schemes
  - Fast updates $\Rightarrow$ the simplest information of mutual relationships
  - Fast querying, fast updates $\Rightarrow$ compromise
Why User-Given Information? (Example 3)

- Situation: Data redundancy
- Question: Is it always necessary to strictly follow the rules of normal forms?
  - No, it is not.
  - Optimal XML-to-relational storage strategy = 4NF
    - No null values, no redundancy

⇒ In all the cases we need additional information given by a user
- XML data, XML queries, annotations…
Content

1. Overview of existing approaches
2. Motivation for improvements
3. Proposed improvement
4. Experiments
5. Conclusion
Basic Ideas

• Searching for similar fragments in the not annotated schema parts
  • The user is not forced to annotate all schema fragments
  • The system can reveal new structural similarities
• Searching for optimal mapping strategy for the remaining schema fragments
  • Adaptive strategy
Step 1. Annotated Schema
Step 2. Searching for Similar Fragments
Step 3. Adaptive Strategy
Step 4. Mapping to Relations
Open Issues

• What similarity function is used? Can we optimize the exhaustive search for similar fragments?
• What types of annotations, i.e. fixed mapping strategies, are supported? How are they combined?
• Should we use a classical adaptive strategy?
Similarity Function and Search Algorithm (1)

- Closely related
  - No knowledge of function
    ⇒ Few ways how to avoid exhaustive search
    • Clustering (expensive preprocessing)
- Idea: Exploit knowledge of the similarity function
- Modification of a classical approach:
  - A set of matchers = partial similarity functions
    • Similarity of a particular feature
      • e.g. depth, number of nodes, complexity of content…
  - A composite similarity function = aggregates the results
    • Weighted sum
Similarity Function and Search Algorithm (2)

- **Features of matchers** ⇒ **a bottom-up strategy**
  - Knowledge for child nodes ⇒ knowledge for parent node
    - e.g. depth, number of nodes

- **Idea**: Searching can terminate if the similarity function reaches its optimum

- **Hard to define a function having a single optimum**

- **Heuristics**: Searching can terminate if reasonable amount of matchers exceed their single optimum
  - e.g. similarity of depths
    - With growing number of nodes the depth grows until it reaches the depth of the searched schema fragment

Types of Annotations

- Particular mapping methods influence versatility of implementation
- CLOB, shredding to tables, indices, numbering schemes...
- Key aspect: Intersection of annotated fragments
  - **Redundant** - both methods are applied on intersection
    - XHTML fragments ⇒ shredding + CLOB
  - **Overriding** - only one of the methods is applied
    - Classical user-driven strategies
  - **Influencing** - both methods are combined into one storage strategy
    - Shredding + indices/numbering schemes

Adaptive Strategy (1)

- Classical approach: Target DB schema is adapted to sample XML data and queries
  - + annotations = too much information
- Idea:
  - Queries = How the data are typically manipulated
  - Data = How complex are XML documents
  ⇒ How to store the data
  - Annotations = How particular schema fragments should be stored

⇒ Annotations can be reused ⇒ no need for additional information
Adaptive Strategy (2)

• **Key operations:**
  • Contraction = replaces each annotated fragment with an auxiliary node
  • Expansion = all auxiliary nodes are expanded to original schema fragments

• **Algorithm:**
  1. The searching for similar fragments and operation contraction repeats until there are no identified candidates for annotating
  2. The resulting schema is expanded

• **Intersection of original and new annotations:** Newly defined are overridden
Content

1. Overview of existing approaches
2. Motivation for improvements
3. Proposed improvement
4. Experiments
5. Conclusion
Experiments

• Experimental implementation *UserMap*
• First idea: Analysis of efficiency of the resulting mapping? Useless.
  • Basic idea: user assigns a mapping strategy to a schema fragment suitable for the actual application
  • Can be highly inefficient in the general case
• Aim: Analysis of behaviour of the strategy on real-world data
  • Can we find similar schema fragments?
  • Can we find any in contracted graphs?
  • How many contractions can be applied, if any?
• Assumption: Reliable and tuned similarity function

Real-World Data

- 99 real-world XML schemes used in statistical analysis
  - Divided into 5 categories - database, document, exchange, report, research
    - Classical data and document centric + finer division
- Testing set of real-world schema fragments
  - 5 data-centric, 5 document-centric, 3 relational, 3 DNA
- Modification of algorithm: Testing set represents annotations, used for all 5 categories of schemes

Results

(1)

- Number of iterations is reasonable
- Multiple contractions are performed
- No extreme values
  - Results correspond to depth of documents
- Schemes are usually not fully contracted
- Default mapping is necessary
- More information should be used

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>dat</th>
<th>doc</th>
<th>ex</th>
<th>rep</th>
<th>res</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average number of iterations</td>
<td>2.7</td>
<td>3.9</td>
<td>2.9</td>
<td>4.1</td>
<td>4.3</td>
</tr>
<tr>
<td>Average % of not annotated nodes</td>
<td>2.1</td>
<td>53.4</td>
<td>13.5</td>
<td>25.6</td>
<td>31.1</td>
</tr>
<tr>
<td>% of fully contracted schemes</td>
<td>93.7</td>
<td>22.2</td>
<td>81.1</td>
<td>0.0</td>
<td>28.6</td>
</tr>
</tbody>
</table>
Results (2)

- The percentage of annotated nodes is usually highest in the first iteration
- Exception: Report, research - low number of sample data, irregular structure
- No relation between types of schema fragments and iterations ⇒ no degenerated schemes
Content

1. Overview of existing approaches
2. Motivation for improvements
3. Proposed improvement
4. Experiments
5. Conclusion
Conclusions and Future Work

• Experiments ⇒ user-given information can and should be further exploited
• Current and future research
  • Combination with classical adaptive methods
    • Exploitation of sample data and queries
    • Too many input data vs. fully contracted schemes and better results
  • Efficient querying over the resulting schema
    • Query plans vs. intersecting annotations
Thank you