Query Processing for XML Data Stored in a Relational Database

A Masters Thesis Proposal

Bhaskar R. Rathod
with Prof. Paul Helman

CS @ UNM

To be presented at the Thesis committee meeting on Aug 26, 2004
Outline

- XML
- Example Scenario
- Discussion of issues
- Our interest
- Outline of what we intend to do
- Details of planned thesis work
- Deliverables
XML

- We start from where we left in the proposal
- Good for data interaction and document modeling
  - Structurally more flexible yet fairly typed
  - Plain-text
  - Parsing and transformation tools widely available
  - Widely used: SOAP, publishing frameworks
XML (cont’d)

- The flip side
  - Very verbose, plain-text => Not efficient to store
  - Big debate: Is it a data model?

- Relational DB systems
  - Mature research in indexing, storage, summarization, query processing
  - Levels of abstraction means flexibility
Example

UNM General Library’s Encoded Archival Documents (EAD)

- Size of each vary from a few kb up to a few mb
- Structure: An almost flat document description section, a highly hierarchical and a highly varied content section
- Stored as plain-text (ASCII, UTF-8) disk files on file system
- Querying done in two ways:
  - keyword search on command-line to the indexing engine
  - A separate XQuery processor application that can not take advantage of keywords
Issues

- It is a common scenario.
- The indexing engine does not index the semantic information in the structure of document.
- Index needs to be rebuilt separately for changed content
- Storage as disk files.
- Even a typical content management system would store the xml documents as disk files
- No benefit of typical database services like transaction management.
- Native XML databases can do some of above, but they are miles behind relational systems.
- Besides, practically used XML does not generally have mixed content.
So?

1. Storage:

2. Data Decomposition:
   a. Do the queries on XML data exhibit different properties than the queries on relational data?

3. Query processing:
   a. Can we query such efficiently?
   b. What role does the implementation of relational operators play?
   c. What role does summarization play?

❖ So, would EAD better off in an RDBMS?
Our Interest

If we store XML data in a relational system:

- How efficiently can we query it as if it were stored in a native XML database?
- How efficient is the storage?
- Are there any interesting characteristics of queries over such data?
- We want to address mainly 2-a, 3-a,b
The Bigger Picture

- Is it possible to modify relational systems such that transaction management, query processing and storage primitives can be used for “highly regular” XML data?
- Can it be efficient?
- If so, then it may solve many practical problems – like the one in example.
- If not, it is good to know; and we can move on.
- That’s all fine and dandy, but where’s the meat?
Work at a glance

1. Develop a metric for query complexity in terms of activity in the physical model – essentially, it will be in terms of tuple-attribute access.

2. For a set of SQL queries transformed from XQueries, determine their complexity in terms of above metric. Determine if a different physical implementation of relational algebra operators affect complexity in terms of disk i/o.

3. Analyze the complexity in disk i/o with structural properties of XML, ie non-flatness of data. Compare the results with the results from Native XML database eXist.

Well, That’s it! In a nutshell.
1. The Metric

- Logical relational operators can be implemented differently at the physical level.
- Each such implementation may impact query plans differently.
- Essentially, each query can be measured in terms of disk I/O of attributes and tuples of participating relations.
- So a relational algebra expression can be evaluated against various implementations.
- For a given configuration of implementations, the metric gives the complexity analysis of the expression.
1. The Metric (cont’d)

- SELECT variants: Unsorted, sorted, indexed
- PROJECT variants: sort-based, index-based
- Join variants: Block-Nested Loop, Simple Sort Merge, Sort Merge, Hash, Hybrid Hash, Hash Index-based
- Partitioning: Vertical, Horizontal
2. XQueries

- XMark – the XML Benchmark contains a set of XML queries against an XML repository
- We will model UNM General Library EAD documents collection along XMark
- We will create XQuery collection on EAD collection a-la XMark Xqueries
- Our benchmark is real-life, more hierarchical and data-rich
- We will transform the EAD XML collection to relational database (various approaches)
- We will transform EAD’s XQuery collection to SQL (various approaches)
2. XQueries (Cont’d)

- Transform SQL queries to relational algebra expressions
- Optimize/process relational algebra expressions
- Evaluate each expression wrt various implementation configurations
- Assign metric values
- Does not consider regeneration of XML twig, but only retrieval of all elements intended. (XSLT can do it)
- XML frameworks use XSLT over recordsets to generate XML twigs (we are a layer below that)
3. Analysis

- Performance in disk/IO versus:
  - Query categories
  - Structure in data
  - Structure in queries
  - Metric value and physical implementation operators
3. Comparison: eXist

- eXist: An open source native XML database (NXD)
- Stores XML documents as a collection
- Supports XQueries
- Provides full-text search options
- EAQ XQueries can be executed over eXist
- Results comparison
Deliverables

- The Metric
- EAD document collection, queries collection, transformed relational database, SQL queries
- Results and Analysis documents
Future Work

- Original idea was to devise a native XML database-like interface on top of a relational db storing XML data – *a la* a deductive db engine that “knows” XML structure and that can be queried upon using XQuery. This thesis will hopefully probe the effectiveness of such an approach.