XML Technologies

XML Databases: Encodings

Christian Grün
Database & Information Systems Group
Universität Konstanz
Introduction

In this and the next meetings, we will...

- trace back the *roots* of today’s XML databases
- analyze various representations (*encodings*) of XML documents
- look at how XML is broken down to single *bits and bytes*
- find out how data can be further *compressed*
- build up *indexes* for the resulting data structures
XML Structure

XML documents are nothing else than trees!

- more formally: connected acyclic graphs
- tree algorithms can be used for processing the document structure
- can directed or cyclic graphs be represented as well?
DB Architecture

Which DBMS components do exist?

- interfaces (APIs)
- query processor: parsing, compilation, evaluation
- buffer management, caching
- transaction & locking
- recovery management
- data, indexes, meta information

How do XML databases look like?
DB Architecture: LORE

Lightweight Object REpository

- developed at Stanford University
- based on object-oriented databases
- uses OEM (Object Exchange Model; looks similar to XML)
- novelty: database schema (DataGuide) is derived from input data
- novelty (at that time): query processor works \(\backslash\) schema-oblivious

McHugh et al. [1997]

```xml
<DB: Eats { // A Sample Database
<Restaurant {
   <Name "Darbar">
   <Entree {
      <Name str "Masala Dosa">
   }>
   "Credit Card" "Visa">
}>
```
DB Architecture: STORED, NATIX

STORED: Semistructured To RELational
Mixed schema, derived from OEM input
  • regular structures are stored relationally
  • irregular structures are stored as graphs

Deutsch et al. [1999]

NATIX

  • maps XML documents to physical storage
  • distributes XML snippets across disks blocks

Kanne & Moerkotte [2000]
DB Architecture

XML databases today:
- result of ten years of development
- well-established concepts

1. XML-enabled databases
- relational DBMS that also support XML

2. Native XML DBMS
- offer specially tailored data structures
- optimized for X-languages (XPath, XQuery, ...)

XML Encodings

**Desired encoding $\mathcal{E}$ must...**

1. be able to map XML to a *database* and back to XML ($\mathcal{E}^{-1}$)
2. reflect *node order* (essential for semi-structured data!)
3. support efficient *traversal* of the tree structure

Alternative wordings: *create, build, shred, ...* documents

**Three picked solutions**

1. DOM: representation of XML nodes as *objects*
2. Hierarchic Labelings: Dewey, ORDPATH
3. Numbering Schemes: *pre, post, size, dist, ...*
**XML Encodings: DOM**

**Document Object Model**
- popular **W3C** standard
- first effort to standardize the representation of HTML/XML documents in internet browsers (Netscape, Internet Explorer)
- XML nodes are mapped to **main-memory** objects
- node objects are equipped with functions to query and modify the tree structure
XML Encodings: DOM

DOM Example in Java

```java
// create document instance
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
DocumentBuilder db = dbf.newDocumentBuilder();
Document doc = db.parse("addressbook.xml");

// query document nodes
NodeList nl = doc.getElementsByTagName("address");
int l = nl.getLength();
System.out.println("Result: " + l);
```

**Drawbacks**

- How could this be expressed in XPath?
- occupies *lots of space* •
- operates in *main-memory* •
XML Encodings: DOM

Persistent DOM

- binary, persistent representation of DOM objects
- node objects and references to related objects are mapped to disk

Edwards & Hope [2000]: Persistent DOM – An architecture for XML repositories
Lv et al. [2002]: Performance evaluation of a DOM-based XML database

Probst [2010]: Processing Arbitrarily Large XML using a Persistent DOM
XML Encodings: Hierarchic Labelings

Dewey Decimal Classification (DCC)

- attempt to organize all human knowledge in a *hierarchical scheme*
- used worldwide in *library catalogs*
- model can be arbitrarily *extended*

Dewey IDs

- general concept of assigning hierarchical numbers to *descendant tree nodes*
- drawback: updates are expensive (💡 why?)

The Lancashire cotton industry: a study in economic development

Assigned DDC Code: 338.4767721094276

3  Economics, Education, Society
33  Economics and Management
338  Industries, Products
338.1 – 338.4 Specific kinds of industries
338.4  Secondary Industries and Services
338.47  Goods and Services
338.471 – 338.479  Subdivisions for Goods and Services
338.476  Technology
338.4767  Manufacturing
338.47677  Textiles
338.476772  Textiles of Seed hair fibres
338.4767721  Cotton
338.47677210  Facet Indicator for Standard Subdivision
338.476772109  Historical, geographic, persons treatment
338.4767721094  Europe
338.47677210942  Western Europe
338.476772109429  England and Wales
338.476772109429  Northwestern England and Isle of Man
338.4767721094276  Lancashire
XML Encodings: Hierarchic Labelings

**ORDPATH**
O’Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- *odd numbers* are assigned to initial labels
- *even numbers* are used as *insertion markers*
- no renumbering required when inserting nodes
- insert node between 1.2.1 and 1.3!

**Drawbacks**
- hierarchical labels have *variable length*
- structure might *degenerate* for certain insertion patterns
XML Encodings: Hierarchic Labelings

**ORDPATH: Compression**
O’Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- hierarchical labels are *bit-wise compressed* into a fixed-size representation
- compressed bit format:
  \[ N_0 \cdot N_1 \ldots N_k \rightarrow L_0 0 \cdot L_1 0 \ldots L_k 0 \]
  - \( L_i \): bitstring (encoded bit length of \( O_i \))
  - \( O_i \): bits of value \( N_i \)

What is the bit pattern for the ORDPATH label 1.8.33?
XML Encodings: Hierarchic Labelings

ORDPATH: Compression

O’Neil et al. [2004]: ORDPATHs: Insert-Friendly XML Node Labels

- the resulting bit sequences preserve node order and contain ancestry relationships
- \( m \) appears before \( n \) if its value is smaller:
  \[
  (m = 1.1.1 = 010010100101001 = 9513) < (n = 1.3 = 0100101100000 = 9568)
  \]
- \( m \) is ancestor of \( n \) if \( m \) starts with substring of \( n \):
  \( m = 1 : 01001 \) is a substring of \( n = 1.3 : 0100101011 \)

Drawback: deeply nested nodes lead to overflow
XML Encodings: Numbering Schemes

Preorder, postorder, inorder
Knuth [1968]: The Art of Computer Programming, Volume 1

- Traverse left subtree
- Traverse right subtree
- Visit the root

Which ordering is represented by which tree?

As all numbers are unique, they may also serve as node IDs
XML Encodings: Numbering Schemes

Pre/post encoding
Grust [2002]: Accelerating XPath Location Steps

```
<A>
  <B/>
  <C>
    <D/> <E/>
  </C>
  <F/>
</A>
```

- all pre and post values can be mapped to a two-dimensional plane
- each node partitions the plane into four regions
XML Encodings: Numbering Schemes

Pre/post encoding
Grust [2002]: Accelerating XPath Location Steps

- n' is an ancestor of n if:
  \[ \text{pre}(n') < \text{pre}(n) \land \text{post}(n') > \text{post}(n) \]
- n' is a descendant of n if:
  \[ \text{pre}(n') > \text{pre}(n) \land \text{post}(n') < \text{post}(n) \]
- n' is a following node of n if:
  \[ \text{pre}(n') > \text{pre}(n) \land \text{post}(n') > \text{post}(n) \]
- n' is a preceding node of n if:
  \[ \text{pre}(n') < \text{pre}(n) \land \text{post}(n') < \text{post}(n) \]

How can you find child nodes?
XML Encodings: Numbering Schemes

Pre/post/level encoding

- \( n' \) is parent of \( n \) if:
  \[
  \text{pre}(n') < \text{pre}(n) \land \text{post}(n') > \text{post}(n) \land \text{level}(n') = \text{level}(n) - 1
  \]

- \( n' \) is a child of \( n \) if:
  \[
  \text{pre}(n') > \text{pre}(n) \land \text{post}(n') < \text{post}(n) \land \text{level}(n') = \text{level}(n) + 1
  \]

- \( n' \) is a following sibling of \( n \) if it has \( p \) as parent and if:
  \[
  \text{pre}(n') > \text{pre}(n) \land \text{post}(n') > \text{post}(n) \land \text{post}(n') < \text{post}(p) \land \text{level}(n') = \text{level}(n)
  \]

- \( n' \) is a preceding sibling of \( n \) if it has \( p \) as parent and if:
  \[
  \text{pre}(n') < \text{pre}(n) \land \text{post}(n') < \text{post}(n) \land \text{pre}(n') > \text{pre}(p) \land \text{level}(n') = \text{level}(n)
  \]

- Can following sibling nodes be found without reference to parent node?
XML Encodings: Numbering Schemes

Pre/size/level encoding
Inspired by Li & Moon [2002]: Indexing and Querying XML Data...

The size value represents no. of descendants + 1.
It simplifies numerous axis traversals:

- n' is a descendant of n if:
  \( \text{pre}(n) < \text{pre}(n') < \text{pre}(n) + \text{size}(n) \)
- n' is a following sibling of n if:
  \( \text{pre}(n') = \text{pre}(n) + \text{size}(n) \land \text{level}(n') = \text{level}(n) \)
- n' is a preceding sibling of n if:
  \( \text{pre}(n') = \text{pre}(n) - \text{size}(n') \land \text{level}(n') = \text{level}(n) \)

Important relationship: \( \text{size}(n) = \text{post}(n) - \text{pre}(n) + \text{level}(n) \)
XML Encodings: Numbering Schemes

Pre/parent encoding
Grün et al. [2006]: Pushing XPath Accelerator to its Limits

- the parent value represents a direct mapping between children and their parents:
  \( n' \) is a parent of \( n \) if \( \text{pre}(n') = \text{parent}(n) \)

- Exercise: \( n' \) is a following sibling of \( n \) if...

- the pre/parent pair constitutes a minimal encoding to reconstruct the original document (no post, size or level needed)

- most update operations based on pre/parent are expensive (\( \text{why?} \))
XML Encodings: Numbering Schemes

Pre/dist/size encoding
Grün et al. [2009]: XQuery Full Text Implementation in BaseX

- the **dist** value is the *distance* to the parent node:
  \( n' \) is a parent of \( n \) if \( \text{pre}(n') = \text{pre}(n) - \text{dist}(n) \)

- in contrast to parent, **dist** is *update-invariant*: subtrees preserve the *original distance* value when moved or inserted

- the **size** property has been retained, as it speeds up some axis operations (see previous slides)
XML Encodings: Numbering Schemes: Updates