Storing and Querying XML into a Relational Database using FDewey Encoding

Irvanizam Zamanhuri

Free University of Bozen-Bolzano
Faculty of Computer Science

March 24, 2010
1 Introduction
   • Introduction
   • Approach
   • Contribution

2 Tree Related Concepts in XML
   • XML Tree Properties
   • Edit Operations in XML Tree
   • Limitations of Interval, Dewey, ORDPATH

3 FDewey Encoding
   • FDewey Encoding
   • Inserting a node
   • Deleting a node
   • Implementation
   • Experiments

4 Conclusions and Future Work
Introduction

- What is XML?
- XML has repeatedly increased in popularity for data representation
  - Richly structured documents could be used over the web
- An XML document can be viewed as a tree with XML elements, attributes, and text-values.
- Efficiency approach for inserting, querying, and removing XML elements/attributes has become a crucial issue.
Approach: the use of standard relational databases
- We first shred XML documents into relations.
- We translate XML queries into SQL statements over these relations.

The key issues of the approach are
- how to effectively update and query XML elements for a large XML document.
- how to resolve the conflict between hierarchical, ordered XML data model, and unordered relational data model.

Labeling Schemes: Interval, Dewey, ORDPATH encodings
We review Dewey, Interval, and ORDPATH encodings.

We propose a new encoding: Float Dewey encoding (FDewey).
The XML document can be viewed as a tree consisting of:

- **Element nodes**: with element tag name, the text-value, and some additional information.
- **Attribute nodes**: with attribute name, the text-value, and some additional information.

**Element nodes contain**:

- nodes of their sub-elements
  - nodes of their attributes

**Attribute nodes contain a node with their text-value**

**Many possibilities** - In this thesis we define/select nodes into two ways:

- Interval-label tree
- Dewey-based label tree
An Interval node consists of a left and right values as an interval value.

**Interval-Label Tree:** \((\textit{nid, left, right, value, tag})\)
- **\textit{nid}:** the node’s Identification Number (ID)
- **\textit{left}:** the left value of the node
- **\textit{right}:** the right value of the node
- **\textit{value}:** the text-value
- **\textit{tag}:** the element tag or attribute names

We assign an incremental preorder (document order) number to every node’s left and right columns.
- **\textit{left}** column: when we visit the node (find an opening XML tag) for the first time,
- **\textit{right}** column: when we visit the node (find a closing XML tag) for the second time.
XML as a Dewey-based Label Tree

- ORDPATH and FDewey are based on Dewey encoding.
- Dewey-based Label Tree: \((nid, nodePath, value, tag)\)
  - \(nid\): the node’s Identification Number (ID)
  - \(nodePath\): the node path from the document root to a node
  - \(value\): the text-value
  - \(tag\): the element tag or attribute names
- the node path depends on the algorithm of dewey-based encodings
Example: Interval and Dewey-based Label Trees

(a) A labeled Tree for Interval encoding

(b) A labeled Tree for Dewey encoding

Fig. 2: A tree labeled with Interval and Dewey

A simple XML document.
XML Tree Properties

- **Ancestor(i)** : a, b, e
- **Parent(i)** : e
- **Preceding(i)** : c, d, f, g, h
- **Preceding siblings(i)** : f, h
- **Descendants(i)** : j, k, l, m, n, o, p, q, r
- **Children(i)** : j, k, n, o, p
- **Following(i)** : s, t, u, v, w
- **Following siblings(i)** : s, t

Source: from paper [2]
XML Tree Properties

- Ancestor(i) : a, b, e
- Parent(i) : e
- Preceding(i) : c, d, f, g, h
- Preceding siblings(i) : f, h
- Descendants(i) : j, k, l, m, n, o, p, q, r
- Children(i) : j, k, n, o, p
- Following(i) : s, t, u, v, w
- Following siblings(i) : s, t

Source: from paper [2]
XML Tree Properties

- Ancestor(i) : a, b, e
- Parent(i) : e
- Preceding(i) : c, d, f, g, h
  - Preceding siblings(i) : f, h
- Descendants(i) : j, k, l, m, n, o, p, q, r
  - Children(i) : j, k, n, o, p
  - Following(i) : s, t, u, v, w
  - Following siblings(i) : s, t

Source: from paper [2]
XML Tree Properties

- Ancestor(i) : a,b,e
- Parent(i) : e
- Preceding(i) : c,d,f,g,h
- Preceding siblings(i) : f,h
- Descendants(i) : j,k,l,m,n,o,p,q,r
- Children(i) : j,k,n,o,p
- Following(i) : s,t,u,v,w
- Following siblings(i) : s,t

Source: from paper [2]
Ancestor(i) : a,b,e
Parent(i) : e
Preceding(i) : c,d,f,g,h
Preceding siblings(i) : f,h
Descendants(i) : j,k,l,m,n,o,p,q,r
Children(i) : j,k,n,o,p
Following(i) : s,t,u,v,w
Following siblings(i) : s,t

Source: from paper [2]
XML Tree Properties

- Ancestor(i) : a, b, e
- Parent(i) : e
- Preceding(i) : c, d, f, g, h
- Preceding siblings(i) : f, h
- Descendants(i) : j, k, l, m, n, o, p, q, r
- Children(i) : j, k, n, o, p
- Following(i) : s, t, u, v, w
- Following siblings(i) : s, t

Source: from paper [2]
Ancestor(i) : a, b, e
Parent(i) : e
Preceding(i) : c, d, f, g, h
Preceding siblings(i) : f, h
Descendants(i) : j, k, l, m, n, o, p, q, r
Children(i) : j, k, n, o, p
Following(i) : s, t, u, v, w
Following siblings(i) : s, t

Source: from paper [2]
XML Tree Properties

- Ancestor(i) : a, b, e
- Parent(i) : e
- Preceding(i) : c, d, f, g, h
- Preceding siblings(i) : f, h
- Descendants(i) : j, k, l, m, n, o, p, q, r
- Children(i) : j, k, n, o, p
- Following(i) : s, t, u, v, w
- Following siblings(i) : s, t

Source: from paper [2]
Edit Operations in XML Tree (1/2)

- **Rename:**
  
  \[ \text{rename}(N_{id}, value, tag) \],
  
  - \( N_{id} \): the node’s ID
  - \( value \): the text-value
  - \( tag \): the element tag or attribute names

- **Example:**
  
  \[ \text{rename}(3, z, new) \]
Insert: \( \text{insert}(N_{id}, P_{id}, i, k, value, tag) \)

- \( N_{id} \): the node's ID  
- \( P_{id} \): the parent's ID  
- \( i \): a child position  
- \( k \): the number of children  
- \( value \): the text-value  
- \( tag \): the element tag or attribute names

Delete: \( \text{delete}(N_{id}) \),

Insert and Delete Operation for a gray node.

Insert Operation for a node 4 at the 2-nd position of the node 1 where the inserted node has no child.
Interval encoding is based on a range value, consisting of *left* and *right* values.

- Idea: interval contains other intervals
- Use preorder and an incremental counter
- Assign *left* value when a node is first visited
- Assign *right* value when a node is last visited
Interval encoding is based on a range value, consisting of *left* and *right* values

Idea: interval contains other intervals

- use preorder and an incremental counter
- assign *left* value when a node is first visited
- assign *right* value when a node is last visited
Interval encoding is based on a range value, consisting of *left* and *right* values.

**Idea:** interval contains other intervals

*use preorder and an incremental counter*

*assign* *left* *value* when a node is first visited

*assign* *right* *value* when a node is last visited

Limitations of Interval (1/4)
Interval encoding is based on a range value, consisting of *left* and *right* values.

Idea: interval contains other intervals.

- use preorder and an incremental counter
- assign *left* value when a node is first visited
- assign *right* value when a node is last visited

```
(4,4,5,v4,i) (5,6,7,v5,j) (7,10,11,v7,k) (8,12,13,v8,l)
```

```
(2,2,15,v2,b) (9,16,17,v9,c) (10,18,21,v10,d)
```

```
(1,1,22,v1,a) (3,3,8,v3,e) (6,9,14,v6,g)
```

```
(11,19,20,v11,h)
```
Limitations of Interval (1/4)

- Interval encoding is based on a range value, consisting of \textit{left} and \textit{right} values
- Idea: interval contains other intervals
- use preorder and an incremental counter
- assign \textit{left} value when a node is first visited
- assign \textit{right} value when a node is last visited
Insert new node with text-value $v_{12}$:

\[ \text{insert}(12,2,2,0,v_{12},f) \]

- relabel the ancestor of the new node
Limitations of Interval (3/4)

- Insert new node with text-value $v_{12}$: $\text{insert}(12, 2, 2, 0, v_{12}, f)$
  - relabel the ancestor of the new node
  - relabel the nodes following of the new node in preorder
Limitations of Interval (4/4)

- Insert new node with text-value $v_{12}$:
  \[ \text{insert}(12,2,2,0,v_{12},f) \]

- relabel the ancestor of the new node
- relabel the nodes following of the new node in preorder
- insert single node
Limitations of Dewey (1/3)

- Dewey Decimal Classification:
  - used in libraries to classify books
  - developed by Melvil Dewey in 1876

- Delete node with ID = 3: delete(3)
  - delete single node
Limitations of Dewey (2/3)

- Delete node with ID = 3: \textit{delete}(3)
  - delete single node
  - update descendants of node ID=3
Limitations of Dewey (3/3)

- Delete node with ID = 3: \textit{delete}(3)
  - delete single node
  - relabel descendants of node ID=3
  - relabel following siblings of node ID=3 and their descendants

```
(1,1,v1,a)
(2,1.1,v2,b)
(4,1.1.1,v4,i)
(5,1.1.2,v5,j)
(6,1.1.3,v6,g)
(7,1.1.3.1,v7,k)
(8,1.1.3.2,v8,l)
(9,1.1.3.2.1,v9,k)
(10,1.1.3.2.1.1,v10,k)
(11,1.2,v11,c)
(12,1.3,v12,d)
(13,1.3.1,v13,h)
```
Limitations of ORDPATH (1/2)

- ORDPATH[3]: Based on Dewey and use only odd numbers at the initial labeling
- insert-friendly XML node
  - leftmost: adding -2 to the last component of first child
  - between two nodes: creating an even component falling between the odd components of the two nodes, then following this with a new odd component, usually 1.
  - rightmost: adding 2 to the last component of last child
Limitations of ORDPATH (2/2)

- no compact component size
- Sibling(3) = (15,1.1.2.1, 1.1.2.1, n1,e), (16,1.1.2.3, 1.1.2.3, n2,e), (6,1.1.3, 1.1.3, v6,g)
FDewey encoding

- Based on Dewey and similar to Dewey at the initial labeling
- use float numbers when a new node is inserted
- Similar to Dewey, each FDewey label also represents a unique path from the root to a node
- use character ‘*’ to separate among components of a node path
  - example: 1*3*1.25, 1*1.5, etc.
Initial Labeling

- Example: "1*1*3", the node is a 3\textsuperscript{rd} child of the parent node which is 1\textsuperscript{st} child of the root node.
Adding a new node

- **leftmost insertion.** It occurs when a new node is inserted before node $N_k$ where $N_k: a_1*a_2*a_3*...*a_n$ is the first child of a node, we label the new node as $a_1*a_2*a_3*...*(a_n/2)$.

- **rightmost insertion.** It occurs when a new node is inserted after node $N_k$ where $N_k: a_1*a_2*a_3*...*a_n$ is the last child of a node, we label the new node as $a_1*a_2*a_3*...*(a_n+1)$.

- **between two consecutive siblings.** It occurs when a new node is inserted between two consecutive sibling nodes, $N_{k-1}: a_1*a_2*...*a_n$ and $N_k: a_1*a_2*...*b_n$, at any position in a set of children of a node, we assign label the new node as $a_1*a_2*a_3*...*((a_n+b_n)/2)$.
we insert a new node without relabeling any other nodes
there are some spaces in between two consecutive sibling nodes
Inserting a node with relabeling nodes

- need to relabeling nodes when the distance between the new node and its nearest right sibling $\leq$ a given epsilon
- the epsilon value is $10^{-38}$
Deleting a node with relabeling nodes

- Without relabeling nodes:
  - the deleted node has no children

- With relabeling nodes:
  - Relabeling descendants
    - when the distance between the last child of the deleted node’s children and its nearest right sibling $\leq$ epsilon
  - Relabeling descendants, following siblings and their descendants
    - when the distance $\geq$ epsilon
Deleting a node with relabeling nodes

- Without relabeling nodes:
  - the deleted node has no children

- With relabeling nodes:
  - Relabeling descendants
    - when the distance between the last child of the deleted node’s children and its nearest right sibling $\leq$ epsilon
  - Relabeling descendants, following siblings and their descendants
    - when the distance $\geq$ epsilon
FDewey Encoding (nodeID, nodePath, nodeValue, label)

Table 1: A FDeweyEncoding relational table.
Storing the FDewey path

Consideration:
- Storing FDewey path in preorder traversal
- Separator character: e.g. 1*11, 1*111.5, 1*12
  - Storing: not consistent with preorder (1*12 > 1*111.5)
- Concatenation between Fixed length and the digits of the float: e.g. 1*0000000011, 1*000000111.5, 1*0000000012
  - Sort order Ok (1*000000012 < 1*000000111.5)
  - It uses space inefficiently
- Variable length encoding (UTF-8)
  - UTF-8 proposed in [1]
Experiments: Inserting nodes at the same place

- Ordered insertions at rightmost.

- Ordered insertions at leftmost and between two consecutive siblings.
Experiments: Inserting nodes in random order

- Random insertion on a wide tree.
- Random insertion on a deep tree.
Experiments: Deleting nodes in random order

- Random deletion on a wide tree.

- Random deletion on a deep tree.
Experiments: Querying static XML document (1/3)

- Getting Children
- Getting parent
Experiments: Querying static XML document (2/3)

- Getting descendants
- Getting ancestors
Experiments: Querying static XML document (3/3)

- Getting siblings
Experiments: Querying dynamic XML document

- the queries for static documents.
- the queries for dynamic documents.
Conclusions

- **Interval:**
  - supports the delete performance without relabeling nodes,
  - but not for the insert performance,

- **Dewey:**
  - has high query performance but poor update performance,

- **ORDPATH:**
  - effectively supports node insertions at any position in the XML tree
  - but slightly decreases query performance particularly for dynamic XML document.

- **FDewey:**
  - it can frequently avoid re-labeling compared to two of the existing labeling schemes, Interval and Dewey.
  - The only node relabeling will occurs if the distance of two XML nodes is smaller than a given epsilon
  - high query performance
Conclusions

- Interval:
  - supports the delete performance without relabeling nodes,
  - but not for the insert performance,
- Dewey:
  - has high query performance but poor update performance,
- ORDPATH:
  - effectively supports node insertions at any position in the XML tree
  - but slightly decreases query performance particularly for dynamic XML document.
- FDewey:
  - it can frequently avoid re-labeling compared to two of the existing labeling schemes, Interval and Dewey.
  - The only node relabeling will occurs if the distance of two XML nodes is smaller than a given epsilon
  - high query performance
Conclusions

- **Interval:**
  - supports the delete performance without relabeling nodes,
  - but not for the insert performance,

- **Dewey:**
  - has high query performance but poor update performance,

- **ORDPATH:**
  - effectively supports node insertions at any position in the XML tree,
  - but slightly decreases query performance particularly for dynamic XML document.

- **FDewey:**
  - it can frequently avoid re-labeling compared to two of the existing labeling schemes, Interval and Dewey.
  - The only node relabeling will occur if the distance of two XML nodes is smaller than a given epsilon
  - high query performance
Conclusions

- **Interval:**
  - supports the delete performance without relabeling nodes,
  - but not for the insert performance,

- **Dewey:**
  - has high query performance but poor update performance,

- **ORDPATH:**
  - effectively supports node insertions at any position in the XML tree
  - but slightly decreases query performance particularly for dynamic XML document.

- **FDewey:**
  - it can frequently avoid re-labeling compared to two of the existing labeling schemes, Interval and Dewey.
  - The only node relabeling will occurs if the distance of two XML nodes is smaller than a given epsilon
  - high query performance
References


