XML Data Management (3)

Indexierung von XML
- Path surrogates
- Patricia Tries
- Dataguides

Indexing for native XML data store

- Performance matters!
- Naive solution:
  - Parse document, traverse tree
- (O)RDBS: use standard techniques (B+ tree etc)
  - CLOB with inverted text index, XML type enhancement
- Principal Problem: how to index trees / graphs / semi-structured data
  - Why difficult??
- Approaches:
  - Support retrieval
    - Value index / text index / tag index
  - Support navigation
Indexing Path expressions

- **Example**

![XML tree diagram](image)

- **Linear context**
  - for each value store root path in posting list
  
  XML: `<book, chapter, section, title>; adr1`  
  `<book, chapter, title>; adr2`

  Expensive: full path must be stored for every occurrence of the word in the posting list

  - Linear context: no label names, no order

- **Def.:** linear context of a leaf value $v$ is the bit set $(b_j)$ of all labels $l_i$ in the tree, in which $b_j = 1$ iff $l_j$ is on the root path to $v$.

  - If number of different labels is large, compression by hashing is used
Linear context

- Example from above:
  \[(b, i) = (\text{book, year, author, name, chapter, section, title})\]

```
XML: (1000111); adr1
     (1000101); adr2
Bradley: (1011000); adr3
Smith: (1011100); adr4
     (1011000); adr5
John: ....
```

Search \[/section/title="XML"\]
Second occurrence of search term can be excluded bit since section is 0

- No navigation support, only text retrieval

Path indexing

- Navigational support using Patricia tries

- Basic idea
  - Encode a document tree as a set of paths

```
Director
  \|-- Movie
    \|-- Title
        \|-- First
            \|-- Andy
        \|-- Last
            \|-- Wachovsky

MDFAndy
MDLWachovsky
MTMatrix
```

- Use index structure appropriate for string search
"Fabric index": example

(a) 
```
<invoice>
  <buyer>
    <name>ABC Corp</name>
  </buyer>
  <seller>
    <name>Acme Inc</name>
  </seller>
</invoice>
```

(b) 
```
<table>
<thead>
<tr>
<th>Document 1</th>
<th>Document 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISBN 2 Acme Rd.</td>
<td>IT C 4</td>
</tr>
<tr>
<td>IT drill</td>
<td>IT N tail</td>
</tr>
<tr>
<td>IT saw</td>
<td></td>
</tr>
<tr>
<td>IT C 2</td>
<td></td>
</tr>
</tbody>
</table>

Note: false drops possible

Patricia trie (see Alp 3)

Note: false drops possible
Search in a Patricia

\[ s = w_1w_2...w, \text{ Patricia } P \]

1. start traversing of \( P \) at root node
2. If \( w \) [value of node] = c follow path labeled c, exit with failure, if no such path exists
3. If node is leaf, check value = \( s \)?
   else goto 2

Example (cont.)

How can large trees be mapped to disk?

Partitioning and indexing the partitions (blocks)
Partitioning

- Separate Patricia trie as index to partitions of tree at level 0
- "direct pointer" points to the partition on same level
- "far pointer" to partition on next level

Result:
- two connected Pat trees

1. **Partitioning**

If partition on level 1 is too large: split and make a further level 2.
Searching the Fabric Index

Search (word \( w_1, \ldots, w_r \) )
level = highest level;
while \( (w_i \text{ of word fits to Pat tree edge}) \) traverse same level or far pointer;
If (no edge labeled) \( w_i \) follow direct pointer; level = level+1;
search (word \( w_{i+1}, \ldots, w_r \) )

Lorel / OEM / XML

- Lorel: experimental DBS for semistructured data
  (Stanford DB group / J. Widom, ~1996-99)
- OEM data model ("Object Exchange Model")
  - In a nutshell:
    - Graph \( G = (N, E) \) with edge labels from \( L \)
    - Different edges may have the same label
    - Nodes have identity

(label graph, objects omitted)
OEM and XML

- XML documents seen as an OEM DB
  XML tags → edge labels
  Tree nodes → OEM nodes which represent a subtree

Abstract* XML document represented as an OEM DB

* Attributes and elements:
Inner nodes of "XML tree"

Indexing a OEM / XML DB

- Dataguides*
  Idea:
  - store all paths of the data graph
  - this constitutes an index, if for each path the reachable objects are stored

Lore-System (Stanford ’97)
Abiteboul et al.: Indexing semistructured data,
ftp://db.stanford.edu/pub/papers/semiindexing98.ps
**Dataguide**

- **Definition**
  - Semistructured DB (e.g. XML DB) (typically hierarchic, extensible to graphs)
  - A Dataguide for DB is the following graph:
    - Every path in DB exists in G (only node name, not values!)
    - Every path in G exists in DB
    - Paths in G are unique (does not hold for DB!)

**Example: Constructing data guides**

The "database" graph – construct data guide
Dataguides: non unique

More than one correct Dataguide for one DB

Constructing datagraph $d$ from source graph $g$ equivalent to constructing a DFA from a NDFA.

Tree $\Rightarrow$ linear time

Using dataguides as an index structure

First variant of dataguide index
Dataguides as indexes

Strong dataguide

- "Strong Dataguide": equivalent path in guide are equivalent in DB
- Or: for all label path l in DG the set of associated object to l is equal to the set of objects reachable in DB

Does not hold for minimal DG of the example: \{x, z, z'\} \neq \{z, z'\}

=> Minimal datagraph can be useless as an index structure
Constructing data guides: example

Algorithm for "Index construction": dataguide G:

```plaintext
node (G) = { root }
edges (G) = ∅
while change do {
    s = nonDetSelect (nodes (G))
    while not all l ∈ Labels(s) processed {
        l = Select (Labels(s));
        nodes (G) += {s′={y|x in s, exists (x→y with label l) in edges(DB)}}
        edges (G) += (x→y, label l)
    }
}
```

Data guide G
May be LARGE
If DB tree: number of nodes G <= number of nodes in DB

construction of deterministic
from nondeterministic automaton
## Dataguides as an index structure

- **Principal idea:**
  - Given a graph / tree structured DB, construct index of all different label paths

- **Space requirement**
  - Worst case: exponential in number of nodes

- **Build time for dataguide index**
  - Worst case exponential
    (construction of DFA using powerset of NDFA states)

- **Useful for lookup of path targets:**
  - `/x/y/z` (like XPath)

- **Not appropriate for // subexpressions**

- **Typically used (in prototyp systems!) together with other index types e.g. full text index**

- **Lessons learned:** Indexing structured data is difficult, no "optimal" solution visible up to now

---

**Diagram:**
- Node `a` connected to `x` and `y`
- Node `b` connected to `y` and `b`

---

*HS/DBSII-03-XML-4*