Managing XML Data on Relational Databases

Svetlana Vinnik

University of Konstanz
Dept. of Computer & Information Science

September 2004

Motivation

- **XML vs. Relational Representation:**
  - XML structure is less rigid than the traditional relational format (*semi-structured* data)
  - XML model is more convenient for some application domains (e.g. content management systems)
  - Growing amounts of data are readily available in the XML format
  - Databases are well-equipped to process relational data (storage management, index structures, query optimization…)
- **Idea:**
  - Benefit from the existing database technology by "shredding" XML documents into tuples and storing them in a relational DBMS

Motivation

- **What are the obstacles?**
  - The data model behind XML is the **tree.**
  - XML trees are navigated through by following a path

```
for $x in fn:doc("bib.xml")/bib/books/book[author = "John Doe"]
  where @price >= 42
  return
     <expensive-book> { $x/title/text() } </expensive-book>
```

  - Existing technology cannot directly be applied to XML data.
    - ? How can we represent trees as relations?
    - ? Can we benefit from index structures?
    - ? How can we implement tree navigation?
  - The W3C XQuery proposal poses additional challenges :
    - ✓ notion of ***order***
    - ✓ complex ***type*** system
    - ✓ ability to ***construct new tree nodes*** on the fly.

Basic Approaches

- **Two basic approaches have emerged:**
  - ***Schema-oblivious*** techniques universally store any tree without closer look at the document's structure.
  - ***Schema-based*** techniques take advantage of the information on the document's schema, but may be less flexible.
- **XPath Accelerator** (Torsten Grust, 2002; Grust et al., 2004)**:**
  - a purely relational storage structure developed with a close eye on the XPath semantics
  - the underlying mapping scheme is schema-oblivious
  - supports all XPath axes, rooted from any context node
  - simple to load and to query

XPath Expressions

- The ***path expression*** (or path) is XPath's core construct.
- A path features one or more ***steps*** s_i (evaluated left to right), syntactically separated by /:

$$s_0 / s_1 / \ldots / s_n$$

- Evaluation of a single step s on node sequence ns:

```
step (ns)
result ← ();
foreach c ∈ ns do
  result ← result U locstep (c,s); // c: context node
return sido (result);
```

- N.B. Result of path expression is duplicate-free node sequence in document order.

XPath Terminology

- ***XPath expressions*** operate on trees of element or attribute nodes.
- Inner nodes *a*, *b*, *c*, *f*, *g* ,and *h* represent XML ***element nodes***.
- Leaf nodes *d*, *e*, *g*, *I*, and *j* represent either ***element*** or ***attribute nodes***.
- Node *a* is the ***root*** of the tree.
- ***height(v)*** is the length of the longest path from v to a leaf in the sub-tree rooted at v.

```
<a>
    <b>
        <c>
            <d> </d><e> </e>
        </c>
    </b>
    <f>
        <g> </g>
        <h>
            <i> </i><j> </j>
        </h>
    </f>
</a>
```

- **XPath expressions consist of**
  (1) a ***context node***, which is the starting point of the traversal, and
  (2) a sequence of ***location steps*** syntactically separated by **/**, evaluated from left to right.

XPath Terminology

- The ***locstep (c, s)*** primitive function lies at the heart of the XPath processor:
  - Given the context node c and step s, compute the sequence of nodes reached by this step.
- Each step **s** is built from two components (separated by **::**):

$$\alpha :: \upsilon$$

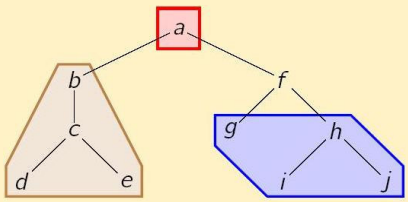
  - The axis α determines, based on the location of context node **c** in the document tree, a sequence of reachable nodes.
  - The ***node test*** υ filters this sequence to contain only specific types of nodes (e.g., only specifically named element nodes, only text / comment / pi nodes, etc.)
- Typical XPath queries thus look like

$$\alpha_0 :: \upsilon_0 / \alpha_1 :: \upsilon_1 / \ldots / \alpha_n :: \upsilon_n$$

XPath Axes

- Given a context node, a step's *axis* establishes a subset of document nodes (a document region).
- This set of nodes, or *forest*, provides the context nodes for the next step
- **Semantics of axes a supported by XPath (step v/ a):**

Axis α	Result Forest
`child`	direct element child nodes of v
`descendant`	recursive closure of `child`
`descendant-or-self`	like `descendant`, plus v
`parent`	direct parent node of v
`ancestor`	recursive closure of `parent`
`ancestor-or-self`	like `ancestor`, plus v
`following`	nodes following v in doc. order
`preceding`	nodes preceding v in doc. order
`following-sibling`	like `following`, same parent as v
`preceding-sibling`	like `preceding`, same parent as v
`attribute`	attribute nodes of node v
`self`	v
`namespace`	namespace nodes of node v

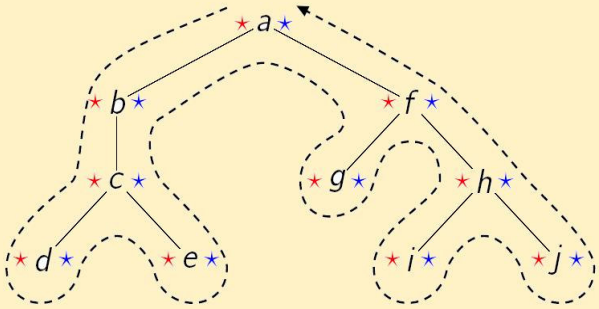


ancestor, preceding, and descendant nodes as seen from node *f*.

XPath Accelerator

- **Idea:**
  - The sequence of SAX events contains all necessary information for representing a tree.
  - Numbering according to this order is sufficient to answer all queries.
  - The SAX events arrive during a sequential read of the XML document.
  - This corresponds to a depth-first, left-to-right traversal of the XML tree.

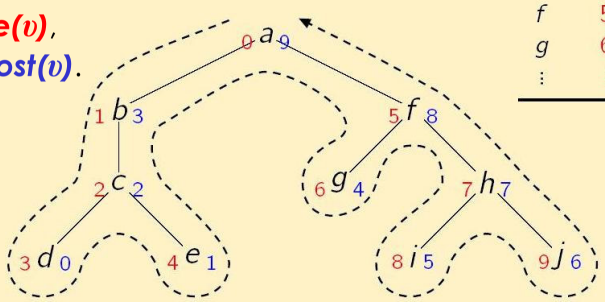
```
<a>
    <b>
        <c>
            <d> </d><e> </e>
        </c>
    </b>
    <f>
        <g> </g>
        <h>
            <i> </i><j> </j>
        </h>
    </f>
</a>
```



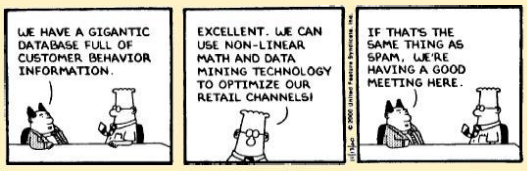
XPath Accelerator

- **For each node v , we receive:**
  - the *startElement* event before the events of any children of v , and
  - the *endElement* event after the events of any children of v .
- **We say that**
  - startElement events occur in ***pre-order***,
  - endElement events arrive in ***post-order***.
- **For each tree node v, the two values to be stored in a relational table**
  - pre-order rank ***pre(v)***,
  - post-order rank ***post(v)***.

v	*pre*	*post*
a	0	9
b	1	3
c	2	2
d	3	0
e	4	1
f	5	8
g	6	4
⋮	⋮	⋮



Encoding Example



dilbert.xml

```
1  <strip>
2    <prolog>
3      <series>Dilbert</series>
4      <author>Scott Adams</author>
5      <characters>
6        <character>The Pointy-Haired Boss</character>
7        <character>Dilbert</character>
8        <character>Wally</character>
9      </characters>
10   </prolog>
11   <panels>
12     <panel>
13       <scene>
14         Pointy-Haired Boss and Dilbert sitting
15         at table.
16       </scene>
17       <bubbles>
18         <bubble>
19           <speaker>The Pointy-Haired Boss</speaker>
20           <speech>
21             We have a gigantic database full of
22             customer behavior information.
23           </speech>
24         </bubble>
25       </bubbles>
26     </panel>
27     <panel>
28       <scene>
29         Dilbert, looking enthusiastic.
30       </scene>
31       <bubbles>
32         <bubble>
33           <speaker>Dilbert</speaker>
34           <speech>
35             Excellent. We can use non-linear math
36             and data mining technology to optimize
37             our retail channels!
38           </speech>
39         </bubble>
40       </bubbles>
41     </panel>
42     <panel>
43       <scene>
44         Pointy-Haired Boss and Dilbert sitting
45         at table.
46       </scene>
47       <bubbles>
48         <bubble>
49           ⋮
50         </bubble>
51       </bubbles>
52     </panel>
53   </panels>
54 </strip>
```


Encoding Example

- Resulting relational representation for ***dilbert.xml***:

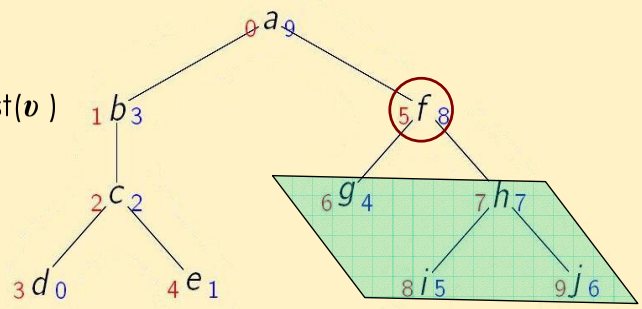
pre	*post*	*kind*	*prop*
0	32	doc	dilbert.xml
1	31	elem	strip
2	11	elem	prolog
3	1	elem	series
4	0	text	Dilbert
5	3	elem	author
6	2	text	Scott Adams
7	10	elem	characters
8	5	elem	character
9	4	text	The Pointy-Haired Boss
10	7	elem	character
11	6	text	Dilbert
12	9	elem	character
13	8	text	Wally
14	30	text	panels
⋮	⋮	⋮	⋮

XPath Steps

- The ***descendant*** axis:
- Node v' is a descendant of v , if
  we saw the start tag of v' ***after*** the start tag of v,
  we saw the end tag of v' ***before*** the end tag of v .
- In the ***pre/post*** encoding:

  $v' \in v$/descendant
  <=>
  pre(v') > pre(v) ∧ post(v') < post(v)

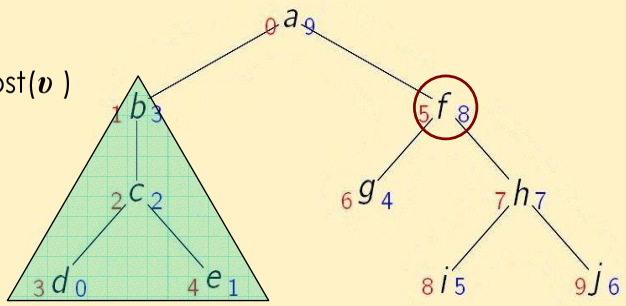
- Recursion turns into simple range predicates.



XPath Steps

- The ***preceding*** axis:
- Node v' is a preceding of v , if
  we saw the start tag of v' ***before*** the start tag of v,
  we saw the end tag of v' ***before*** the end tag of v .
- In the ***pre/post*** encoding:

  $v' \in v$/preceding
  <=>
  pre(v') < pre(v) ∧ post(v') < post(v)



XPath Steps

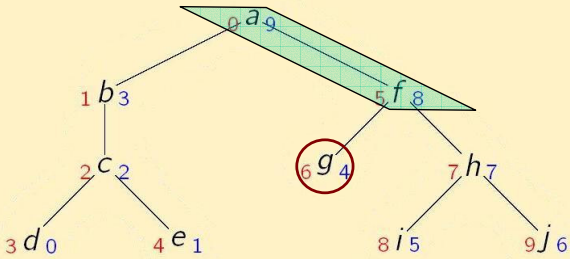
- The ***ancestor*** axis:

  Contrary to *descendant* relationship:

  $v' \in v/\text{ancestor}$

  $\Leftrightarrow$

  $\text{pre}(v') < \text{pre}(v) \wedge \text{post}(v') > \text{post}(v)$



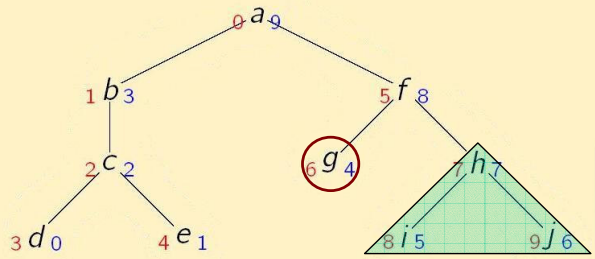
- The ***following*** axis:

  Contrary to *preceding* relationship:

  $v' \in v/\text{following}$

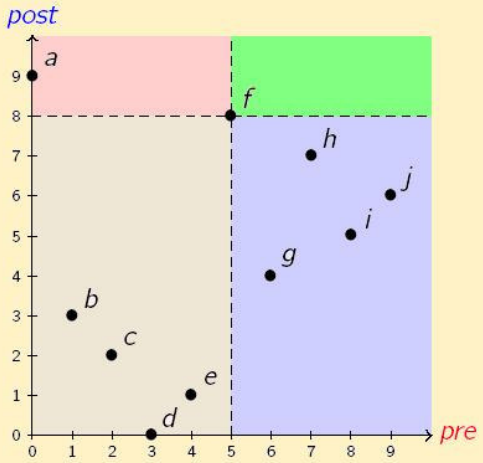
  $\Leftrightarrow$

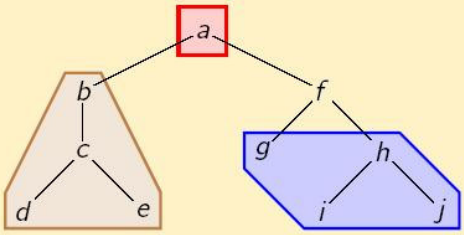
  $\text{pre}(v') > \text{pre}(v) \wedge \text{post}(v') > \text{post}(v)$



pre/post plane

- Regard pre and post values of nodes as points in the two-dimensional ***pre/post*** plane.
- XPath axes can be represented as windows in the plane:





ancestor, preceding, and descendant nodes as seen from node *f*.

XPath Evaluation Scheme

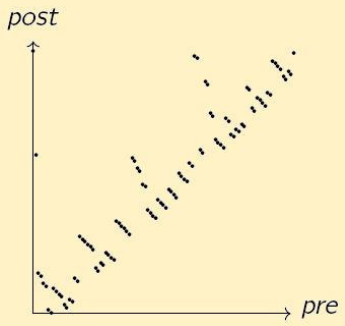
- Tree-structured document is mapped into a 2-dimensional space.
- Formerly recursive queries turn into region queries on that plane.
- Database technology can efficiently support n-dimensional data:
  - R-trees as an n-dimensional index structure;
  - A combination of two B+-trees can be used for two dimensions.
- Other node properties can be mapped to additional dimensions.
- To evaluate an XPath expression, just repeatedly evaluate window query on the pre/post plane:
  - load doc nodes into a descriptor table *accel*

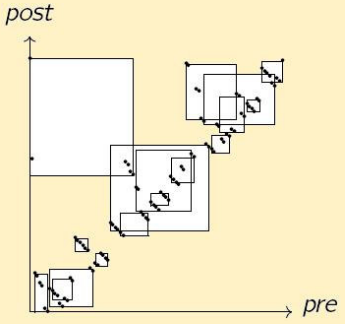
pre	post	par	attr	tag

  - Set-oriented XPath translation scheme targeting SQL

$$query(p/\alpha) = \begin{cases} SELECT \quad v'.* \\ FROM \quad query(p)\ v,\ accel\ v' \\ WHERE \quad v'\ INSIDE \quad window(\alpha, v) \end{cases}$$

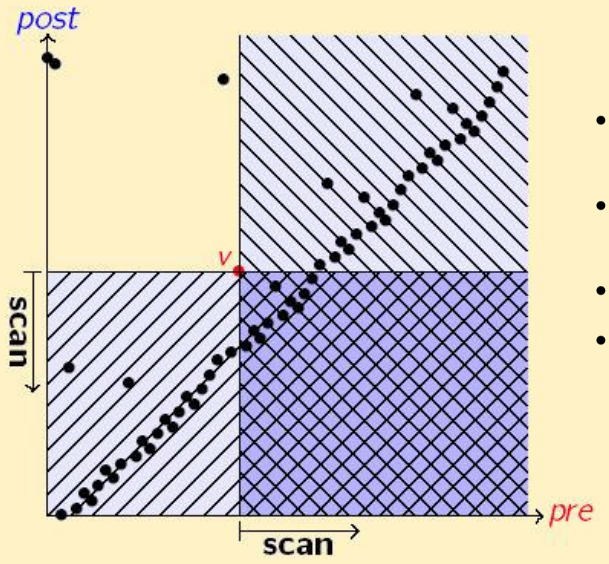
Employing R-Trees





- **Evaluating XPath location steps**
  - Node descriptors (*pre, post, par, attr, tag*) are points in a 5-dimensional space
  - R-Trees: incomplete space partitioning, adapt well to node distribution.
  - 5-dimensional R-Tree evaluates XPath axis and name test in parallel.
- **Preorder Packed R-Trees**
  - Storage utilization in R-Tree leaf pages maximized
  - Coverage and overlapping of leaves minimized
  - Preorder packing preserves doc order on retrieval

Employing B$^+$-Trees



- Index pre and post columns with a B$^+$-index.
- A window query takes two index scans.
- Lots of false hits in either scan.
- Can we do better?

Further Optimizations

- Notice that

$$\text{pre}(v) = |\, v\,/\text{ancestor}\,| + |\, v\,/\text{preceding}\,|$$
$$\text{post}(v) = |\, v\,/\text{preceding}\,| + |\, v\,/\text{descendant}\,|$$
$$\text{pre}(v) - \text{post}(v) = \underbrace{|\, v\,/\text{ancestor}\,|}_{\text{level}(v)} - \underbrace{|\, v\,/\text{descendant}\,|}_{\text{size}(v)}$$

- The pre value of the last descendant node v' is $\text{pre}(v') = \text{pre}(v) + \text{size}(v)$.

$$v' \in v/\text{descendant}$$
$$\Leftrightarrow$$
$$\text{pre}(v') > \text{pre}(v) \wedge \text{pre}(v') \leq \text{post}(v) + \text{level}(v)$$
$$\wedge$$
$$\text{post}(v') < \text{post}(v) \wedge \text{post}(v') \geq \text{pre}(v) - \text{level}(v)$$

- Similar ideas apply to other axes.

Further Optimizations

- We may now even use either range, ***pre*** or ***post***, to evaluate the descendant axis.
- We thus extend our database table by a property ***level***:

pre	*post*	*level*	*kind*	*prop*
0	32	0	doc	dilbert.xml
1	31	1	elem	strip
2	11	2	elem	prolog
3	1	3	elem	series
4	0	4	text	Dilbert
5	3	3	elem	author
6	2	4	text	Scott Adams
7	10	3	elem	characters
8	5	4	elem	character
9	4	5	text	The Pointy-Haired Boss
10	7	4	elem	character
11	6	5	text	Dilbert
⋮	⋮	⋮	⋮	⋮

Further Optimizations

- If we do not have ***level (v)*** available, we can estimate it using

$$\textit{height}(t) \geq \textit{level}(v)$$

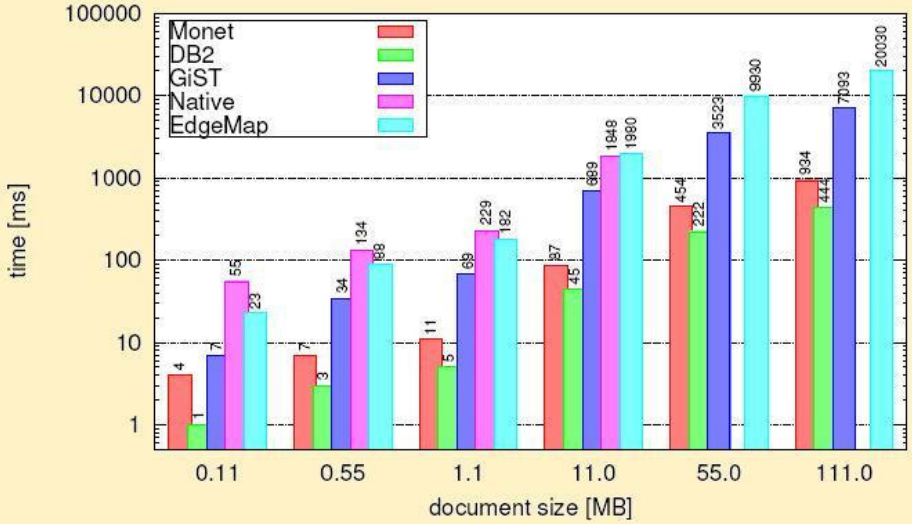
- The height of an XML document is typically independent of the document size.
- The estimation is very precise in practice, the speedup can amount to orders of magnitude.
- The level property also makes the non-recursive axes easier to express:

$$v' \in v/\text{child} \Leftrightarrow v' \in v/\text{descendant} \wedge \text{level}(v') = \text{level}(v) + 1$$

$$v' \in v/\text{parent} \Leftrightarrow v' \in v/\text{ancestor} \wedge \text{level}(v') = \text{level}(v) - 1$$

- To support the child and parent axes, as well as the *-sibling axes, storing each node's parent (by means of its pre value) can be beneficial.

Performance



- The chart compares an implementation of the XPath accelerator on the main-memory database "**MonetDB**", on the SQL system "**DB2**", and on a library (**GiST**) for multidimensional data against a commercial XML database system and the "naive" edge mapping approach for relational systems.

Literature

- Jens Teubner. Database Supported XML: *Lecture Notes, May 2004* .
- Torsten Grust, Maurice van Keulen, Jens Teubner. **Accelerating XPath Location Steps in Any RBMDS.** *ACM Transactions on Database Systems (TODS), 29(1), March 2004*.
- Torsten Grust. **Accelerating XPath Location Steps**. *In Proc. of the 21st Int'l ACM SIGMOD Conference on Management of Data, pages 109-120, Madison, Wisconsin, USA, June 2002*.
- Torsten Grust. XML : Lecture Notes, May 2002
- Torsten Grust. **Accelerating XPath Location Steps**. Technical Report 162/2001, Department of Computer and Information Science, Database Research Group, University of Konstanz, December 2001.