XML and Databases

Lecture 10
XPath Evaluation using RDBMS

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2009

Outline

1. Recall pre / post encoding
2. XPath with //, ancestor, @, and text()
3. XPath with / and following-sibling: use pre / size / level encoding
4. Optimization through tree-aware join: “Staircase Join”
   Prune
   Partition and Skip

Outline - Lectures

1. Introduction to XML, Encodings, Parsers
2. Memory Representations for XML: Space vs Access Speed
3. RDBMS Representation of XML
4. DTDs, Schemas, Regular Expressions, Ambiguity
5. XML Validation using Automata
8. Node Selecting Queries: XPath
7. Tree Automata for Efficient XPath Evaluation, Parallel Evaluation
8. XPath Properties: backward axes, containment test
9. Streaming Evaluation: how much memory do you need?
10. XPath Evaluation using RDBMS
11. XSLT – stylesheets and transform
12. XQuery – XML query language

Outline - Assignments

2. SAX Parse into memory structure: Tree and DAG
3. Map XML into
4. XPath evaluation
5. XPath into SQL Translation → 1. June

Questions

What is the benefit of storing the LEVEL of a node in the table?
Which accessor functions become easier?

→ It can also be useful to store the size of the subtree at a node in the table. Do you see advantages of doing that?

… from Lecture 3:

Later in this course, we will use the PRE/POST encoding again.

→ We will find a systematic way to map queries on XML (XPath) into SQL queries.

Assignment 5 is about programming this mapping.
Assignment 3: generate tables, send to DB, run query & print results

XML document

<table>
<thead>
<tr>
<th>book_tbl</th>
<th>1</th>
<th>12</th>
<th>book</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>title</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>null</td>
<td>TCP/IP Illustrated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>author</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>last</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>null</td>
<td>Stevens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>first</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>null</td>
<td>John</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>null</td>
<td>Addison-Wesley</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>price</td>
<td>65.95</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book_attr</th>
<th>1</th>
<th>isbn</th>
<th>1-2345-6789-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>year</td>
<td>1994</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>currency</td>
<td>USD</td>
<td></td>
</tr>
</tbody>
</table>

Assignment 5
add extra root node (pre=0)
Æ Generate SQL queries from Xpath
(1) if node tests: " and ", etc
(2) add / and ancestor
(3) add filters

Bonus: add @-axis

1. Recall: Pre/Post Encoding

Add post order

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
</tbody>
</table>

CREATE VIEW descendant AS
SELECT r1.pre, r2.pre FROM doc_tbl r1, R r2
WHERE r1.pre < r2.pre
AND r1.post > r2.post

“structural join”

XPath Accelerator encoding
XML fragment f and its skeleton tree

Pre/post encoding of f: table accel

<table>
<thead>
<tr>
<th>pre</th>
<th>post</th>
<th>tag</th>
<th>type</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>name</td>
<td>b</td>
<td>NULL</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>name</td>
<td>a</td>
<td>NULL</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>name</td>
<td>f</td>
<td>NULL</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>name</td>
<td>e</td>
<td>NULL</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>name</td>
<td>d</td>
<td>NULL</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>name</td>
<td>c</td>
<td>NULL</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>name</td>
<td>b</td>
<td>NULL</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
<td>name</td>
<td>a</td>
<td>NULL</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
<td>name</td>
<td>f</td>
<td>NULL</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>name</td>
<td>e</td>
<td>NULL</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
<td>name</td>
<td>d</td>
<td>NULL</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>name</td>
<td>c</td>
<td>NULL</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>name</td>
<td>b</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Our Tables: Assignment 3 + extra root node

XML document

<table>
<thead>
<tr>
<th>book_tbl</th>
<th>1</th>
<th>12</th>
<th>book</th>
<th>null</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>title</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>null</td>
<td>TCP/IP Illustrated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>author</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>last</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>null</td>
<td>Stevens</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>first</td>
<td>null</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>null</td>
<td>John</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>null</td>
<td>Addison-Wesley</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>price</td>
<td>65.95</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book_attr</th>
<th>1</th>
<th>isbn</th>
<th>1-2345-6789-0</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>year</td>
<td>1994</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>currency</td>
<td>USD</td>
<td></td>
</tr>
</tbody>
</table>

Add root node with
Æ pre = 0
Æ post = |nodes| + 1
Æ tag = null
Æ text = null
2. XPath with //, ancestor, @, and text()

Formally: do you remember the correct definition of the abbreviation "//"?

// is abbreviation for /descendant-or-self::node()/

/book is abbreviation for /descendant-or-self::node()//title
XPath with //, ancestor, @, and text()

/\book//author[@text()="Knuth"]

location step 3: "descendant :: author"

```
SELECT r3.pre FROM book_tbl r1, r2, r3
WHERE r1.pre=0
AND r2.pre>r1.pre
AND r2.post<r1.post
AND r2.tag="book"
AND r3.pre>r2.pre
AND r3.post<r2.post
AND r3.tag="author"
AND r2.text="Knuth"
```

Plus filter

```
ORDERED BY r3.pre
```

Correct: returns result nodes in document order.

What about duplicates? E.g., /\author/ancestor::*

always remove duplicates!

What about other axes? E.g., child, parent etc??

Easy to add following and preceding axes.

online step 3: "descendant :: author"

XPath with //, ancestor, @, and text()

/\book//author[@name="Knuth"]

```
SELECT r3.pre FROM book_tbl r1, r2, r3, book_attr a
WHERE r1.pre=0
AND r2.pre>r1.pre
AND r2.post<r1.post
AND r2.tag="book"
AND r3.pre>r2.pre
AND r3.post<r2.post
AND r3.tag="author"
AND r3.pre=a.pre
AND a.attr="name"
AND a.value="Knuth"
```

```
ORDERED BY r3.pre
```

What about duplicates? E.g., /\author/ancestor::*

Always remove duplicates!
3. XPath with / and following-sibling

3. XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling

XPath with / and following-sibling
XPath with / and following-sibling

How to select ALL children of a node?

maybe "descendant, i.e., larger pre and smaller post, AND no other descendant in between" can be done in SQL, but is expensive!

... AND NOT EXIST (SELECT ...)

Using LEVEL this is easy:

Select all descendants at LEVEL + 1 ...

firstChild (pr, po) = left-most node, below and to the right of (pr, po)

e.g., not c- and d-node (because b-node is in between..)

Can be done similar as before, using a complicated SQL query.

Expensive

PRE / SIZE / LEVEL Encoding

<table>
<thead>
<tr>
<th>pre</th>
<th>size</th>
<th>level</th>
<th>lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>root</td>
<td>root</td>
</tr>
<tr>
<td>3</td>
<td>b</td>
<td>1</td>
<td>a</td>
</tr>
<tr>
<td>4</td>
<td>c</td>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>5</td>
<td>d</td>
<td>3</td>
<td>c</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>4</td>
<td>c</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>2</td>
<td>b</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>2</td>
<td>c</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>3</td>
<td>b</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>c</td>
</tr>
</tbody>
</table>

Question

Can you compute the post-value of a node, from its (pre, size, level) values?
**Pre / Size / Level Encoding**

Useful relationships & approximations

Knowledge taken from specific properties of pre/post ranks to shrink window

\[ \text{size}(e) = \text{level}(e) - \text{post}(e) + \text{pre}(e) \]  

Why?

For the right most leaf of the sub tree we can say

\[ \text{size}(e) = \text{pre}(h) - \text{pre}(e) \]

THUS  

\[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

number \( x \) of nodes

\( \text{pre}(n) = x + \text{level}(n) \)

\( \text{post}(n) = x + \text{size}(n) \)

\( \text{pre}(n) - \text{post}(n) = \text{level}(n) - \text{size}(n) \)

---

**Pre / Size / Level Encoding**

Useful relationships & approximations

THUS  

\[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

number \( x \) of nodes

\( \text{pre}(n) = x + \text{level}(n) \)

\( \text{post}(n) = x + \text{size}(n) \)

\( \text{pre}(n) - \text{post}(n) = \text{level}(n) - \text{size}(n) \)

---

**Pre / Size / Level Encoding**

Useful relationships & approximations

THUS  

\[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

number \( x \) of nodes

\( \text{pre}(n) = x + \text{level}(n) \)

\( \text{post}(n) = x + \text{size}(n) \)

\( \text{pre}(n) - \text{post}(n) = \text{level}(n) - \text{size}(n) \)

---

**Pre / Size / Level Encoding**

Useful relationships & approximations

THUS  

\[ \text{pre}(n) + \text{size}(n) = \text{post}(n) + \text{level}(n) \]

Why?

number \( x \) of nodes

\( \text{pre}(n) = x + \text{level}(n) \)

\( \text{post}(n) = x + \text{size}(n) \)

\( \text{pre}(n) - \text{post}(n) = \text{level}(n) - \text{size}(n) \)

---

**Pre / Size / Level Encoding**

Useful relationships & approximations

From equation (2) formed to

\[ \text{pre}(n) = \text{pre}(e) + \text{size}(e) \]

and replacing size(e) with (1) leads to:

\[ \text{pre}(n) - \text{post}(n) = \text{level}(e) - \text{size}(e) \]

For a useful estimation we can also for sure say that:

\[ \text{Level}(e) \leq \text{height}(t) \]

Using height(t) instead of level(e):

Node b has max preorder and node f has min post-order rank:

\[ \text{pre}(b) \leq \text{post}(e) + \text{height}(t) \]

\[ \text{post}(f) - \text{pre}(e) = \text{height}(t) \]

The value height(t) of a document tree is assigned at document loading time.
descendant( pr, si, le ) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre >  pr
AND r1.pre <= pr + si

ancestor( pr, si, le ) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre <  pr
AND r1.pre + r1.size >= pr

child( pr, si, le ) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.level = le + 1

parent( pr, si, le ) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.level = le - 1

preceding( pr, si, le ) =
SELECT r1.pre FROM doc_tbl r1
WHERE r1.pre + r1.size < pr
4. Staircase Join

Most of the following slides are by Maurice van Keulen (Univ. of Twente, The Netherlands)

See http://edbtss04.dia.uniroma3.it/VanKeulen.pdf
4. Staircase Join

Technique 1: Avoiding Duplicates

- Problem 1: Might have MANY duplicates here!

- SQL Query:

  ```sql
  SELECT DISTINCT r2.pre
  FROM context c, doc_tbl r1, r2
  WHERE r1.pre > c.pre
  AND r1.post > c.post
  AND r2.pre > r1.pre
  AND r2.post < r1.post
  ORDER BY r2.pre
  ```

- IBM DB2 Query Plan:

  - `SELECT DISTINCT` clause selects unique predecessor values.
  - `FROM` clause specifies the tables involved.
  - `WHERE` clause filters the data to avoid duplicates.
  - `ORDER BY` clause orders the results.

4. Staircase Join

Technique 1: Pruning

- How to Prune depends on axis

  - C: set of context nodes
  - ax: XPath axis

  1. If `ancestor`, then
     - Prune(C,ax) = lowest (=bottom-most) independent nodes in C
  2. If `descendant`, then
     - Prune(C,ax) = highest (=top-most) independent nodes in C

Pruning requires only a single sequential scan over the pre/post table.
4. Staircase Join

Technique 1  Pruning

→ How to Prune depends on axis

C  set of context nodes
ax  XPath axis

(1) If ax=ancestor, then
Prune(C,ax) = lowest independent nodes in C

(2) If ax=descendant, then
Prune(C,ax) = highest independent nodes in C

Qu: give pruning rule for
(3) ax = following

Hint
For any two context nodes N1 and N2:

following-nodes(N1) ⊆ following-nodes(N2) OR
following-nodes(N2) ⊆ following-nodes(N1)

Why is that?
Consider arbitrary N1, N2
Either one is a descendant of the other or…

Technique 2  Partitioning

Problem
Still 2-scan area that generates duplicates

(b,c,d) ∈ doc
content doc = begin
  _partition_table(pre, post);
  /* partition pre ... c, d, pre */
  case pre C nodes in context:
  pre < post ∧ c nodes in context:
  if c nodes < c nodes then
    else
      _compartition_des(c nodes, pre = 1, c nodes, pre = 1,
                      c nodes, post);
      c nodes = 0;
      node node in doc:
      _compartition_des(c nodes, pre = 1, d nodes, pre = 1,
                        c nodes, post);
      return result;
      end
  _compartition_des(pre, pre2, post)
begin
  for i from pre to pre2 do
    if doc[i] post ∧ c node then
      append doc[i] to result;
  end

1) It scans doc and context tables sequentially
2) It scans both tables only once for an entire context node sequence
3) It never delivers duplicate nodes
4) Result nodes are produced in document order
5) Input for staircase join can be any node sequence
   (3) + (4) ⇒ no post-processing (unique/sort) is needed to comply to XPath semantics
4. Staircase Join

**Technique 2: Partitioning & Skipping**

\[(b, c, b) / \text{descendant...}^*\]

- Never touch more than \[|content| + \text{preorder nodes}\]
- A characterization of the location of \(T\) can be found before scan starts (estimated skipping)

CAVE: only true if there are no value comparisons in filters...

More...

**Skip empty regions in the plane**

*For any two nodes ‘a’ and ‘b’*

- Why? \(a\)'s following nodes are also following nodes of \(b\).
- \(a\) and \(b\) do not have common descendants.

More...

**Skip empty regions in the plane**

*For any two nodes ‘a’ and ‘b’*

- Why? \(a\) cannot be a descendant of a preceding node of \(b\).

More...

**Skip empty regions in the plane**

*For any two nodes ‘a’ and ‘b’*

- Why? \(b\) cannot be a descendant of a preceding node of \(a\).
END
Lecture 10