Staircase Join: Teach a Relational DBMS to watch its (Axis) Steps.

Authors:
Torsten Grust
Maurice van Keulen
Jens Teubner

Presented by
Sanjay Kulhari
Agenda

- **Background**
  - XML and Relational Databases
  - XPath

- **XPath Accelerator**
  - Pre/Post Plane
  - SQL Based XPath evaluation

- **Staircase Join**
  - Pruning
  - Partitioning
  - Algorithm
XML and Relational Databases

- Specialized data type for XML.
- No. of methods associated with this data type.
- Methods access XML Document Object Model.
- Methods uses XPath expression as argument to search and retrieve nodes.
XPath

- XPath is a specialized expression language used to parse through XML.
  - State/City[Population > 100000]
- XPath nodes
  - Document, Element, Attribute, Text
- XPath Axes
  - Define and allow access to any node within XML document.
  - Major XPath axes
    - Ancestor
    - Descendent
    - Following
    - Preceding
XPath Axes

- Preceding
- Ancestor
- Descendant
- Following
XPath Accelerator

- Relational XML encoding.
  - Document is represented as a relational table.
  - Indexed using indexed structure native to the RDBMS.
  - Queried using relational language.
Pre/Post Plane

Pre order traversal

Post order traversal

Ancestor | Following | Preceding | Descendant

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>4</td>
</tr>
<tr>
<td>f</td>
<td>5</td>
</tr>
<tr>
<td>g</td>
<td>6</td>
</tr>
<tr>
<td>h</td>
<td>7</td>
</tr>
<tr>
<td>i</td>
<td>8</td>
</tr>
<tr>
<td>j</td>
<td>9</td>
</tr>
</tbody>
</table>

Postorder rank

Preorder rank

(0,0)
SQL-based XPath evaluation

(c)/following/descendant = (f, g, h, i, j)

| (v)/descendant | = post (v) – pre (v) + level (v) 

<= h

AND v2.pre <= v1.post + h AND v2.post >= v1.pre + h

SELECT DISTINCT v2.pre
FROM doc v1, doc v2
WHERE v1.pre > pre(c)
AND v1.pre < v2:pre
AND v1.post > post(c)
AND v1.post > v2.post
ORDER BY v2.pre
Staircase Join

Basic idea: Join is made between set of context nodes and the pre/post relational table by using knowledge of the pre/post plane.

Ancestor-or-self for \((d, e, f, h, i, j)\)

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d)</td>
<td>3</td>
</tr>
<tr>
<td>(e)</td>
<td>4</td>
</tr>
<tr>
<td>(f)</td>
<td>5</td>
</tr>
<tr>
<td>(i)</td>
<td>8</td>
</tr>
<tr>
<td>(h)</td>
<td>7</td>
</tr>
<tr>
<td>(j)</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0</td>
</tr>
<tr>
<td>(b)</td>
<td>1</td>
</tr>
<tr>
<td>(c)</td>
<td>2</td>
</tr>
<tr>
<td>(d)</td>
<td>3</td>
</tr>
<tr>
<td>(e)</td>
<td>4</td>
</tr>
<tr>
<td>(f)</td>
<td>5</td>
</tr>
<tr>
<td>(g)</td>
<td>6</td>
</tr>
<tr>
<td>(h)</td>
<td>7</td>
</tr>
<tr>
<td>(i)</td>
<td>8</td>
</tr>
<tr>
<td>(j)</td>
<td>9</td>
</tr>
</tbody>
</table>

Use predicates based on pre/post knowledge

Result

<table>
<thead>
<tr>
<th>Pre</th>
<th>Post</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>0</td>
</tr>
<tr>
<td>(d)</td>
<td>3</td>
</tr>
<tr>
<td>(e)</td>
<td>4</td>
</tr>
<tr>
<td>(f)</td>
<td>5</td>
</tr>
<tr>
<td>(h)</td>
<td>7</td>
</tr>
<tr>
<td>(i)</td>
<td>8</td>
</tr>
<tr>
<td>(j)</td>
<td>9</td>
</tr>
</tbody>
</table>
Staircase Join (Cont.)

- **Pruning**

  Ancestor-or-self for \( (d,e,f,h,i,j) \)

  \( (d,a), (e,a),(f,e,a), (h,f,e,a),(i,e,a),(j,i,e,a) \) \textbf{11 duplicates}

  Final result \( (a,d,e,f,h,i,j) \)

  Ancestor-or-self for \( (d,h,j) \)

  \( (d,a), (h,f,e,a),(j,i,e,a) \) \textbf{3 duplicates}

  Final result \( (a,d,e,f,h,i,j) \)
Staircase Join (Pruning)

(a) descendant axis
(b) ancestor axis

Overlapping regions
Staircase Join (Pruning)

Overlapping regions
Staircase Join (Pruning)

Pruning procedure for descendent axis

\[
\text{prunecontext_desc} \ (\text{context} : \text{TABLE} (\text{pre}, \text{post})) \equiv
\]

BEGIN
\[
\begin{align*}
\text{result} & \leftarrow \text{NEW TABLE} (\text{pre}, \text{post}); \ \text{prev} \leftarrow 0; \\
\text{FOREACH} \ c \ \text{IN context} & \ \text{DO} \\
& \quad \text{IF} \ c.post > \text{prev} \ \text{THEN} \\
& \quad \quad \text{APPEND} \ c \ \text{TO result}; \\
& \quad \text{prev} \leftarrow c.post; \\
\text{RETURN} \ & \text{result};
\end{align*}
\]

END

\(c1, c2\) and \(c3\) relate to each other on preceding/following axis

Context establishes a boundary that resembles a staircase.

Removal of nodes from overlapping regions
Staircase Join (Pruning)

(a, b)/following = S U T U W
= T U W
= (b)/following

(a) Nodes $a$ and $b$ relate to each other on the ancestor/descendant axis.
(b) Nodes $a$ and $b$ relate to each other on the preceding/following axis.

Empty regions in pre/post plane
The partitions \([p0; p1), [p1; p2), [p2; p3)\) of the ancestor staircase separate the ancestor-or-self paths in the document tree.
Staircase Join (Algorithm)

Characteristics

2. Scans the doc and context table sequentially

3. Scans both the tables only once for the entire context sequence.

4. Never duplicate nodes.

5. Result nodes are produced in document order.
Staircase Join (Skipping)

No node beyond v contributes to the result.
Region between pre(v) and pres(c2) is skipped

(c1,c2)/descendant
Experimental results

(a) Avoiding duplicates (Q2)  
(b) Staircase join performance (Q2)
(c) Effectiveness of skipping

(d) Effectiveness of skipping
Conclusion

Increased tree awareness can lead to significantly improved XPath performance.
Future research

- To experiment in a commercial disc based RDBMS.
- Use larger documents $> 1$GB
- Parallel XPath execution strategy
Thank You