XML and Databases

Lecture 13
Update Languages for XML

Sebastian Maneth
NICTA and UNSW

CSE@UNSW -- Semester 1, 2010

Outline

1. Update Languages for XML
   - XQuery Update Facility: delete, insert, replace, rename, remove
   - type issues
   - snapshot semantics

2. The physical site
   - how to update a DAG?
   - how to update PRE/POST encoding?
   - other storage schemes?

XML Updates -- History

Updates = write operations, e.g., delete, insert, replace, rename, etc
Want to have Update Language, i.e., a formalism for “update programs”.

Currently, there is no accepted standard XML Update Language

- XUpdate (XML:DB, working draft from 9/2000)
- XQuery! (by the implementors of the Galax XQuery engine)
- XQuery Update Facility (W3C Candidate Recommendation, 09 June 2009)

plus lots of other smaller projects…

Example updates for XML data

(1) delete subtree rooted at node x

Explicit examples
Delete the last author of the first book in a given bibliography.

\[ \text{do delete fn:doc("bib.xml")/books/book[1]/author[last()]} \]

Delete all email messages that are more than 365 days old.

\[ \text{do delete /email/message[fn:currentDate()-date > xs:dayTimeDuration("P365D")]} \]

Use XPath to specify the nodes x to be deleted.

Note
Every node has an “identity” = a unique identifier.
Also: there may be attributes of type “ID”.

XML Updates

Example updates for XML data

(2) insert subtree “t” as first of node x

Explicit examples

Note
Every node has an “identity” = a unique identifier.
Also: there may be attributes of type “ID”.

XML Updates

Example updates for XML data

(3) replace subtree rooted at node x with “t”
XML Updates

Example updates for XML data

(2) insert subtree "t" as first of node x

Note: Every node has an "identity" = a unique identifier. Also, there may be attributes of type "ID"!

Question: Can t be arbitrary? For which t should the insert fail?

Note: Every node has an "identity" = a unique identifier. Also, there may be attributes of type "ID"!

(3) insert subtree "t" as last of node x

Note: All insert operations: "subtree t can easily be generalized to a sequence of subtrees (t_1, t_2, t_3, ..., t_n)"
XML Updates

Example updates for XML data

(5) insert subtree "t" after node x

Explicit examples

Insert a year element after the publisher of the first book.

```xml
do insert <year>2005</year> after fn:doc("bib.xml")/books/book[1]/publisher
```

Navigating by means of several bound variables, insert a new police report into the list of police reports for a particular accident.

```xml
do insert $new-police-report as last into fn:doc("insurance.xml")/policies/policy[id = $pid]/driver[license = $license]/accident[date = $accdate]/police-reports
```

(6) rename node x as name

Explicit examples

Rename the first author element of the first book to principal-author.

```xml
```

Rename the first author element of the first book to the QName that is the value of the variable $newname.

```xml
```

(7) replace node x with ( n_1 n_2 n_3 … n_m )

Explicit examples

Replace the publisher of the first book with the publisher of the second book.

```xml
```

(8) replace value of node x with "some string"

Explicit examples

Replace the publisher of the first book with the publisher of the second book.

```xml
```

Note

The replace-value-of op. preserves node identity!

Æ If x is a text-node, then text-content of x becomes "string1"

Æ If x is an attribute node, then attribute value becomes "string1"
XML Updates

Example updates for XML data

(8) replace value of node x with “some string”

Explicit examples

Increase the price of the first book by ten percent.

```
fn:doc("bib.xml")/books/book[1]/price
```

Semantical issues: doc changes after first update, this might affect the subsequent updates! How to deal with this?

Questions

What about the different node types

Can I insert an attribute node at any position?

Can I replace an attribute node by an element node, or vice versa?

etc

Do we really need so many different operations?

Which operation can be simulated by other ones?

How to generalize the target, from a node to an XPath expression?

(bulk updates, using one operation)

Snapshot Semantics

Semantics of this on the document "Jonny Pizzicato"?

Insert <phone>02 83060405</phone> as last into //address/name[text()="Jonny Pizzicato"]/ tiền in $e in phone $e as "telephone"

Semantics for all n_i, append upd:delete(n_i) to pending update list

Snapshot Semantics

Each update operation is logically applied to a separate snapshot of the original document.

Updates are applied independently from each other to the original document. They don't see each others' effects.

The order of the update operations is irrelevant.

Type Issues

do delete TargetExpr

must eval. to a sequence (n_1…n_m) of nodes.

Otherwise: Type Error!

Semantics for all n_i, append upd:delete(n_i) to pending update list

Type Issues

do delete TargetExpr

must eval. to a sequence (n_1…n_m) of nodes.

Otherwise: Type Error!

Semantics for all n_i, append upd:delete(n_i) to pending update list

Type Issues

do delete TargetExpr with ExprSingle

must eval. to a sequence of attribute nodes followed by non-att nodes

Semantics for all n_i, append upd:delete(n_i) to pending update list

Type Issues

do delete TargetExpr

must eval. to a sequence of nodes.

Otherwise: Type Error!

Semantics for all n_i, append upd:delete(n_i) to pending update list

Type Issues
**Type Issues**

- **do delete TargetExpr**
  - must eval. to a sequence of nodes. Otherwise: Type Error!
  - must eval. to a sequence of attribute nodes followed by non-att nodes

- **do insert SourceExpr (as (first | last) into) before | after TargetExpr**

- **do replace TargetExpr with ExprSingle**
  - evals to $n_1 n_2 \ldots n_p u_1 u_2 \ldots u_p$
  - $\Rightarrow$ TargetExpr must eval. to single node (called $\text{Target}$) and must have a parent ($\text{Parent}$)

- If $\text{Target}$ is attribute node, then
  - appending to pending update list
  - $\text{upd: insertAttributes(}\text{Parent, }\text{alist);}$
  - $\text{upd: insertBefore(}\text{Parent, }\text{clist);}$
  - $\text{upd: delete} \text{Target;}$

- Ambiguity

  - If $\text{Target}$ is element, text, comment, or PI node, then
    - **do replace TargetExpr with ExprSingle**
    - is the same as
    - **do insert ExprSingle before TargetExpr**
    - **do delete TargetExpr**

  - Many more data-dependent ambiguities
    - insert as last = insert as first, if there are no children
    - insert as first = insert before on the first child, if that exists
    - insert as last = insert after on the last child, if that exists

**Challenges: Physical Updates**

**Questions**

- How to do updates on a DAG?
  - What will be different?
  - Are incremental updates possible?

- How to do updates on a PRE/POST-encoding?
  - What will be different?
  - Are incremental updates possible?

**XUpdate: Text node updates**

- Obviously, the kind of $c$ determines the overall impact on the updated tree and its encoding.

<table>
<thead>
<tr>
<th>XUpdate: replacing text by text</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b$</td>
</tr>
<tr>
<td>$b/foid = 1$</td>
</tr>
<tr>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;/a&gt;$</td>
</tr>
<tr>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;a&gt;$</td>
</tr>
<tr>
<td>$&lt;a&gt;$</td>
</tr>
</tbody>
</table>

- New content $c$: a text node

**XUpdate: Structural updates**

<table>
<thead>
<tr>
<th>XUpdate: inserting a new subtree</th>
</tr>
</thead>
<tbody>
<tr>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;c&gt;&lt;d&gt;&lt;e/c&gt;&lt;f/&gt;g&lt;//c&gt;$</td>
</tr>
<tr>
<td>$&lt;$</td>
</tr>
<tr>
<td>$&lt;/a&gt;$</td>
</tr>
</tbody>
</table>

- Question: What are the effects w.r.t. our structure encoding...?
Updates and fixed-width encodings

Theoretical result [Milo et al., PODS 2002]
There is a sequence of updates (subtree insertions) for any persistent tree encoding scheme $E$, such that $E$ needs labels of length $\Omega(N)$ to encode the resulting tree of $N$ nodes.

- Fixed-width tree encodings (like XPath Accelerator) are inherently static.
  - Non-solutions:
    - Gaps in the encoding.
    - Encodings based on decimal fractions.

$\Omega(N)$ is not so much a problem of cost but of locking. Why?

A variable-width tree encoding: ORDPATH

Here we look at a particular variant of a hierarchical numbering scheme, optimized for updates.

- The ORDPATH encoding (used in MS SQL Server$^\text{TM}$) assigns node labels of variable length.

ORDPATH labels for an XML fragment

- The fragment root receives label 1.
- The $n$th ($n = 1, 2, \ldots$) child of a parent node labeled $p$ receives label $p.(2^n - 1)$.
- Internally, ORDPATH labels are not stored as <Separated ordinals but using a prefix-encoding (similarities with Unicode).>
Processing XQuery and ORDPATH

Is ORDPATH a suitable encoding for XPath?

Mapping core operations of the XQuery processing model to operations on ORDPATH labels:

\[ v/parent::node() \]

- Let \( p, m, n \) denote \( v \)'s label (\( n \) is odd).
- If the rightmost ordinal (\( m \)) is even, remove it. Go to 0.

In other words: the cares (\( . \)) do not count for ancestry.

\[ v/descendant::node() \]

- Let \( p, n \) denote \( v \)'s label (\( n \) is odd).
- Perform a lexicographic index range scan from \( p \) to \( p \cdot (n + 1) \)—the virtual following sibling of \( v \).

ORDPATH: Variable-length node encoding

- Using (4 byte) integers for all numbers in the hierarchical numbering scheme is an obvious waste of space!
- Fewer (and variable number of) bits are typically sufficient;
- they may bear the risk of running out of new numbers, though. In that case, even ORDPATH cannot avoid renumbering.
  - In principle, though, no bounded representation can absolutely avoid the need for renumbering.
- Several approaches have been proposed so as to alleviate the problem, for instance:
  - use a variable number of bits/bytes, akin to Unicode,
  - apply some (order-preserving) hashing schemes to shorten the number,
  - ...

For a 10MB XML sample document, the authors of ORDPATH observed label lengths between 6 and 12 bytes (using Unicode-like compact representations).

Since ORDPATH labels encode root-to-node paths, node labels share common prefixes.

**ORDPATH** labels of `<cl/>` and `<a/>`

<table>
<thead>
<tr>
<th>label</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.5.4.1</td>
<td>4 bytes</td>
</tr>
<tr>
<td>1.5.4.3</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

- Label comparisons often need to inspect encoding bits at the far right.
- MS SQL Server™ employs further path encodings organized in reverse (node-to-root) order.
- **Note:** Fixed-length node IDs (such as, e.g., preorder ranks) typically fit into CPU registers.

END Lecture 13