MonetDB/XQuery — Consistent & Efficient Updates on the Pre/Post Plane

Peter Boncz Stefan Manegold Sjoerd Mullender Jan Flokstra Maurice van Keulen Torsten Grust Jan Rittinger Jens Teubner

Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands
http://monetdb-xquery.org/

University of Twente
Enschede - The Netherlands

Technische Universität München, Germany
http://pathfinder-xquery.org/

Abstract

This demonstration presents the second iteration of **MonetDB/XQuery**, now featuring efficient updates of XML documents in relational storage. While the first iteration of MonetDB/XQuery focused on high-performance and scalable (read-only) XQuery processing, this second iteration also implements structural and value updates following the **W3C XQuery Update Facility** proposal.

MonetDB/XQuery uses the region-based *pre/size/level* encoding (equivalent to the *pre/post* encoding), prohibiting straight-forward implementation of (structural) updates, as they would incur costs linear to the document size to update the *pre*-order rank of all successor as well as the *size* information of all ancestors.
Our implementation demonstrates how we apply techniques like delta-updates and carefully exploit the virtual column feature of MonetDB to avoid locking bottlenecks and reduce the physical cost to the minimum (i.e., linear to update volume). The result is an efficient implementation of W3C XQuery Update Facility-like updates that comes at an expense of less than 30% overhead in XQuery processing on the XMark benchmark.

Our XML Update Implementation: New XQuery Operators with Side Effects

Value Updates (map trivially to updates on the underlying relational tables)

```
fn:set-attr    (node,str,str)          : update // set a node attribute (2nd param) to a certain value (3rd param)
fn:set-attr    (node,str,str,str,str)  : update // set a node (attribute,prefix,URI) to a certain value (5th param)
fn:unset-attr  (node,str)              : update // remove an attribute (identified by 2nd param) from a node
fn:unset-attr  (node,str,str,str)      : update // remove an attribute (identified by 2nd-4th param) from a node
fn:set-text    (node,str)              : update // set value of a text node
fn:set-comment (node,str)              : update // set value of a comment node
fn:set-pi      (node,str,str)          : update // set (instruction,arguments) value of a processing instruction node
```

Structural Updates (The Challenge; see below)

```
fn:insert-first  (node,node) : update  // insert subtree (2nd param) as first child of 1st param
fn:insert-last   (node,node) : update  // insert subtree (2nd param) as last child of 1st param
fn:insert-before (node,node) : update  // insert subtree (2nd param) as immediate predecessor sibling of 1st param
fn:insert-after  (node,node) : update  // insert subtree (2nd param) as immediate successor sibling of 1st param
fn:delete        (node)      : update  // delete subtree rooted in the given node
```

- Similar functionality as proposed by XUpdate, UpdateX, XQuery!, W3C XQuery Update Facility
- Minimal changes to our XQuery parser before standard syntax is (was) proposed
- W3C XQuery Update Facility compliance is planned

Staircase Join on *pre*-view & on *rid*-table

pagesize $= 2^n$ allows fast *pre*⇔*rid* **swizzling** using the page-offset table *pg/off*:

$rid = pre \,\&\, 0x0000ffff + pg\_off[(pre \,\&\, 0xffff0000) \gg 16] \ll 16 \quad (n = 16)$

Example: `(b,f,k)/child::*`

on *pre*–view, using memory mapping trick as in MonetDB

	pre	size	level	
	0	18	0	a
	1	3	1	b
	2	2	2	c
	3	0	3	d
	4	0	3	e
	5	13	1	f
	6	9	2	g
	7	2	3	k
	8	0	4	l
	9	0	4	m
	10	5	null	
	11	4	null	
	12	3	null	
	13	2	null	
	14	1	null	
	15	0	null	
	16	2	2	h
	17	0	3	i
	18	0	3	j
	19	4	null	
	20	3	null	
	21	2	null	
	22	1	null	
	23	0	null	

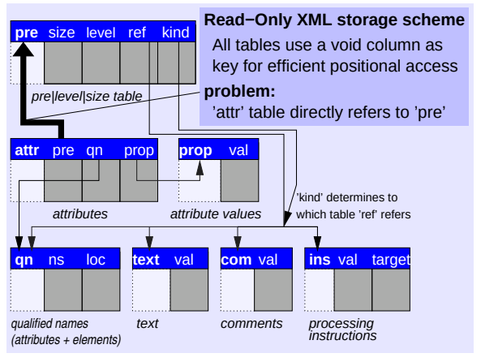
pre[b]=1; 1+3=4; 2+2+1=5>4; pre[f]=1; 5+13=18; 6+9+1=16; 7+2=9; 8+0+1=9; 9+0+1=10>9; 16+2+1=19>18

on *rid* –table, using explicit swizzling whenever a page boundary is crossed

	rid	size	level	
	0	18	0	a
	1	3	1	b
	2	2	2	c
	3	0	3	d
	4	0	3	e
	5	13	1	f
	6	9	2	g
	7	2	3	k
	8	2	2	h
	9	0	3	i
	10	0	3	j
	11	4	null	
	12	3	null	
	13	2	null	
	14	1	null	
	15	0	null	
	16	0	4	l
	17	0	4	m
	18	5	null	
	19	4	null	
	20	3	null	
	21	2	null	
	22	1	null	
	23	0	null	

rid[b]=swizzle(pre[b])=1; 1+3=4; 2+2+1=5>4; rid[f]=swizzle(pre[f])=5; 5+13=18; 6+9+1=16; 7+2=9; swizzle(7+1)=16; 16+2+1=19>18; 8+0+1=9; 9+0+1=10>9; swizzle(16)=8

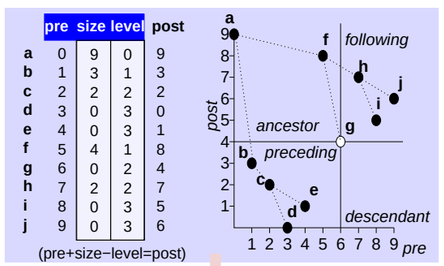
Read-Only vs. Updatable Schema



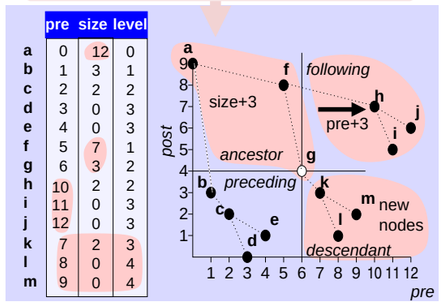
Structural Updates on *pre/size/level* XML Storage

Problem:
Shifts in *pre* ⇒ $O(size(doc))$ to keep *pre* sorted

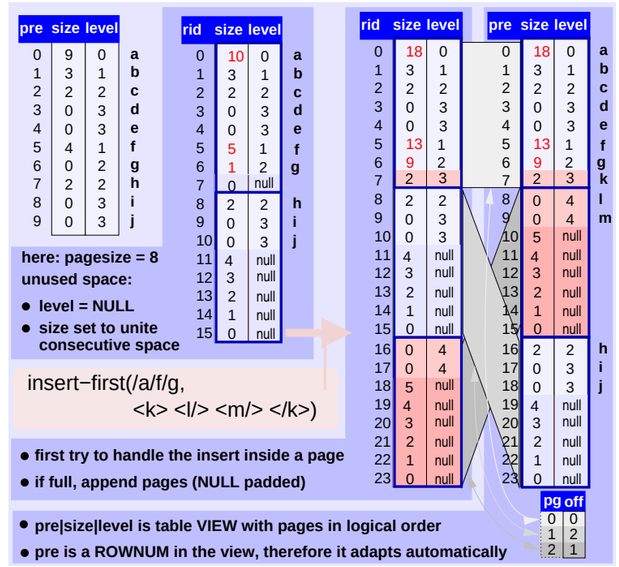
Solution:
Use Logical Pages ⇒ $O(size(update))$ (*pagesize* $= 2^n$ tuples, $n \geq 16$)



insert−first(/a/f/g , <k><l/><m/></k>)



Read–Only vs. Updatable Representation



Consistent Bulk Processing

- **Consistency:**
  - **Snapshot Semantics:** Pre-image determines **which nodes** to update with **what values**
  - **XQuery Semantics:** Use XQuery sequence order to maintain the relative order between separate update operations
- **Problem:**
  - Conflicts with bulk relational evaluation (optimization!)
- **Solution:**
  - Update operators initially produce a **tape** of **intended** updates
  - Tape is represented as XQuery item sequence ⇒ ensures correct order
  - Update tape is optimized and applied only after XQuery evaluation has finished
  - Similar to the idea of monad-based I/O in purely functional programming languages (*e.g.*, Haskell)

Example: Updates on the XMark document

- Add a new bid to open auction "open_auction9":

```
let $inc:=1.23
let $bid:=<bidder><date>03/28/2006</date><time>13:30:00</time>
          <personref person="007"/><increase>{$inc}</increase></bidder>
let $oa:=doc("auctions.xml")/site//open_auction[@id="open_auction9"]
return (insert-after(exactly-one($oa/bidder[last()]),$bid),
set-text(exactly-one($oa/current),"{exactly-one($oa/current)+$inc}"))
```

- Close auction "open_auction0":

```
let $site:=doc("auctions.xml")/site
let $oa:=exactly-one($site//open_auction[@id="open_auction0"])
let $ca:=exactly-one($site/closed_auctions)
return (insert-last($ca,
<closed_auction>{$oa/(seller,itemref,quantity,type,annotation)}
<price>{$oa/current}</price><date>{$oa/interval/end}</date> <buyer person="{$oa/bidder[last()]/personref/@person}"/></closed_aution>),
delete($oa))
```

- Change author of "open_auction4711"'s annotation:

```
let $oa:=doc("auctions.xml")/site//open_auction[@id="open_auction4711"]
let $au:=$oa/annotation/author
return set-attr(exactly-one($au),"author","person007")
```