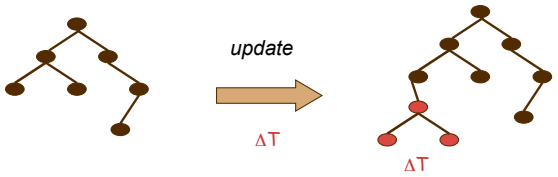
Querying and storing XML

Week 6
XML Updates
February 26-March 1, 2013

XML Updates



- Input: an XML tree ΔT and XML update T
- Output: updated XML tree T' = T + ΔT

Challenges

- The study of the following is still in its infancy:
- update languages for XML data
  - XQuery Update Facility: relatively new standard
- implementation of XML updates
  - native storage of XML data
  - validation, consistency and integrity
- concurrency control for XML "databases"; crash recovery
- Question: is there a method to
  - support updates commonly found in practice?
  - provide XML query engines with XML update support?
  - avoid the troubles of concurrency control, consistency checking, etc?

Update Support

- How to update?
  - Flat streams: overwrite document (slow)
  - Colonial: SQL updates? (hard to translate)
  - Native: DOM, proprietary APIs
- How do you know you have not violated schema?
  - Flat streams: re-parse document
  - Colonial: need to understand the mapping and translate/ maintain integrity constraints
  - Native: supported in some systems (e.g., eXcelon)

Atomic updates

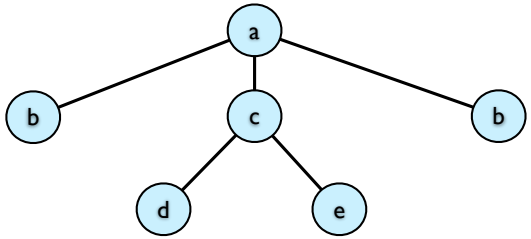
Atomic updates

- Basic changes that can be applied to tree

```
u ::= insertInto(n,t)
    |  insertAsFirstInto(n,t)
    |  insertAsLastInto(n,t)
    |  insertBefore(n,t)
    |  insertAfter(n,t)
    |  delete(n)
    |  replace(n,t)
    |  replaceValue(n,s)
    |  rename(n,a)
```

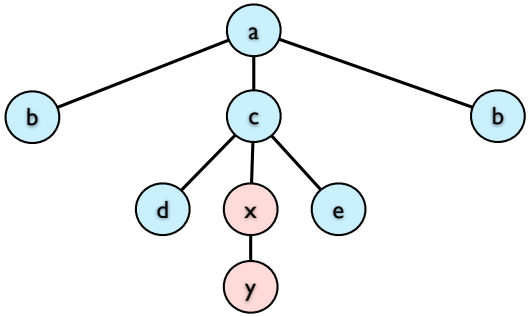

Atomic updates: insertion

- InsertInto (c,<x><y/></x>)



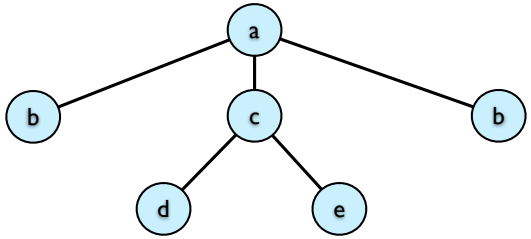
Atomic updates: insertion

- InsertInto (c,<x><y/></x>)



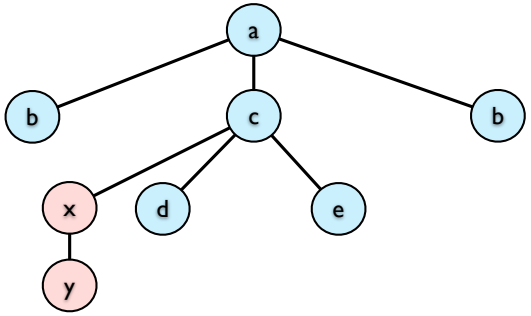
Atomic updates: insertion

- InsertAsFirstInto (c,<x><y/></x>)



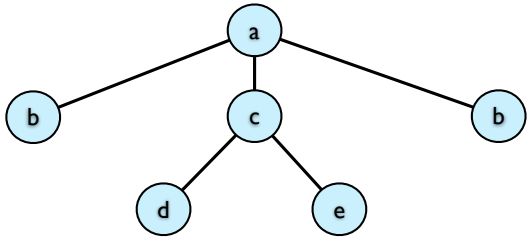
Atomic updates: insertion

- InsertAsFirstInto (c,<x><y/></x>)



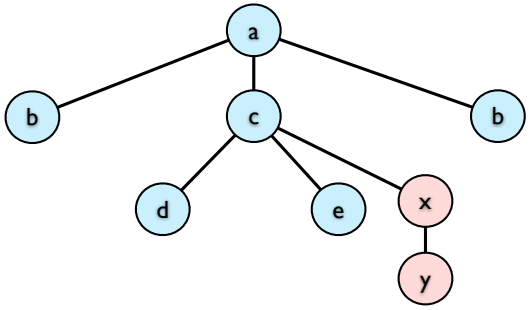
Atomic updates: insertion

- InsertAsLastInto (c,<x><y/></x>)



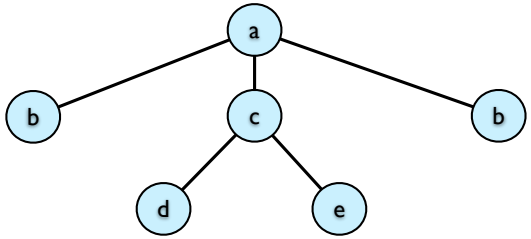
Atomic updates: insertion

- InsertAsLastInto (c,<x><y/></x>)



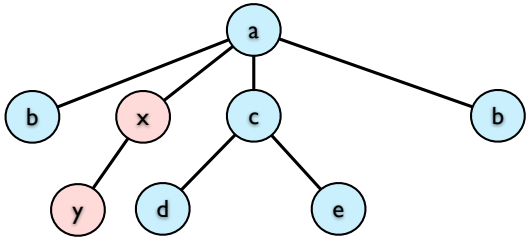
Atomic updates: insertion

- InsertBefore (c,<x><y/></x>)



Atomic updates: insertion

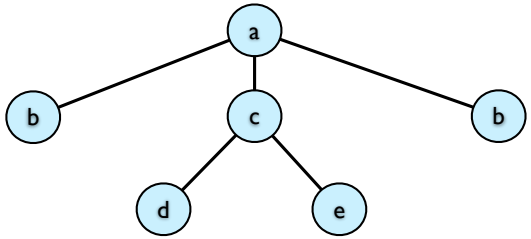
- InsertBefore (c,<x><y/></x>)



QSX February 26-March 1, 2013
10

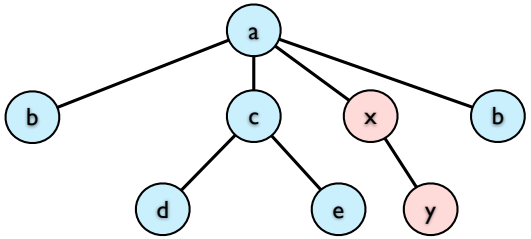
Atomic updates: insertion

- InsertAfter (c,<x><y/></x>)



Atomic updates: insertion

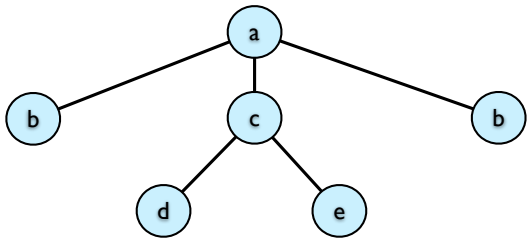
- InsertAfter (c,<x><y/></x>)



QSX February 26-March 1, 2013
11

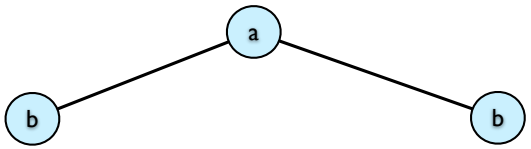
Atomic updates: deletion

- Delete (c)



Atomic updates: deletion

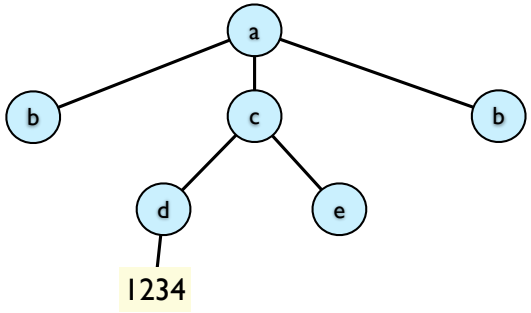
- Delete (c)



QSX February 26-March 1, 2013
12

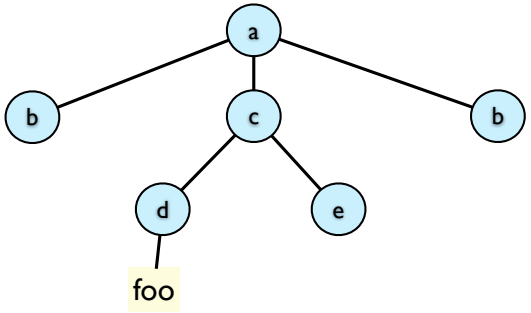
Atomic updates: replace text value

- ReplaceValue (d,"foo")



Atomic updates: replace text value

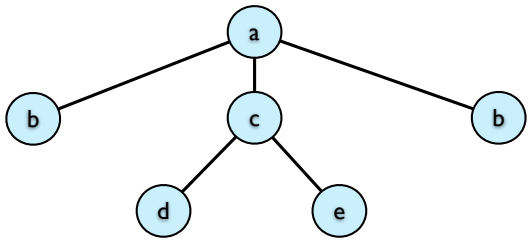
- ReplaceValue (d,"foo")



QSX February 26-March 1, 2013
13

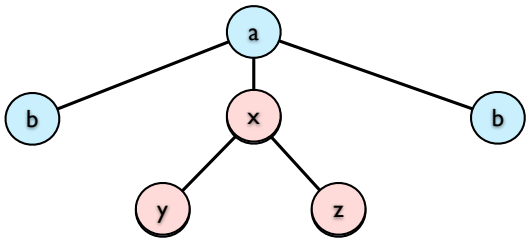
Atomic updates: replace subtree

- Replace (c,<x><y/><z/></x>)



Atomic updates: replace subtree

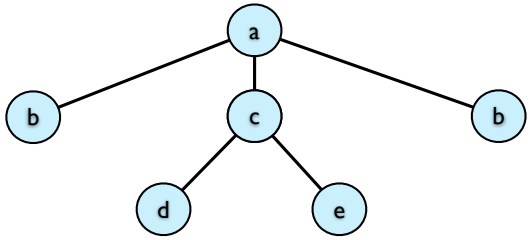
- Replace (c,<x><y/><z/></x>)



QSX February 26-March 1, 2013
14

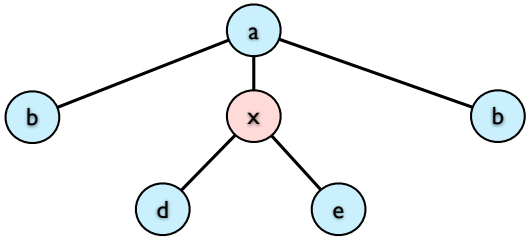
Atomic updates: rename

- Rename (c,x)



Atomic updates: rename

- Rename (c,x)



QSX February 26-March 1, 2013
15

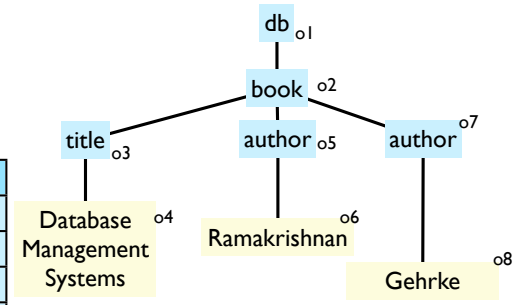
Updating XML stored in relations

Approaches

- We've considered several approaches to storing XML in relations
  - naive "edge relation:
  - shared inlining
  - Dewey Decimal
  - interval encoding
- How can we **update** data in these representations?
- What are the tradeoffs?

Updating XML: naive

- Remember this?



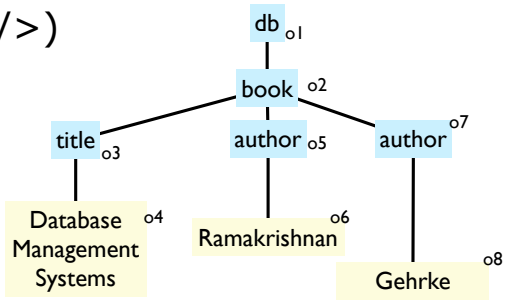
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

- InsertInto(o5,<foo/>)



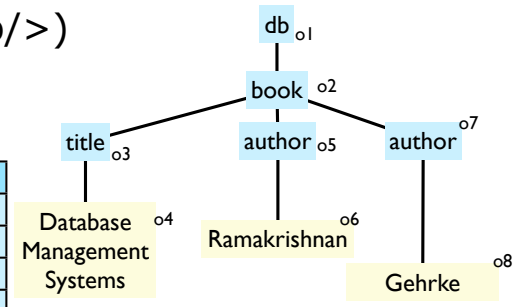
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
INSERT INTO Nodes
VALUE (o9,foo,ELT)
INSERT INTO Edges
VALUE (o5,o9)
```



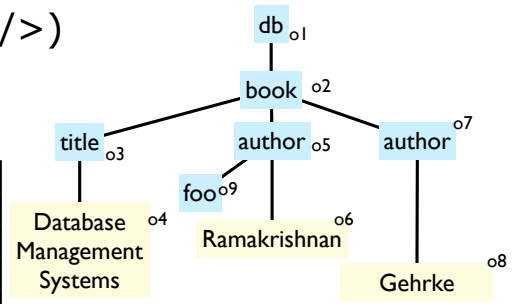
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
INSERT INTO Nodes
VALUE (o9,foo,ELT)
INSERT INTO Edges
VALUE (o5,o9)
```



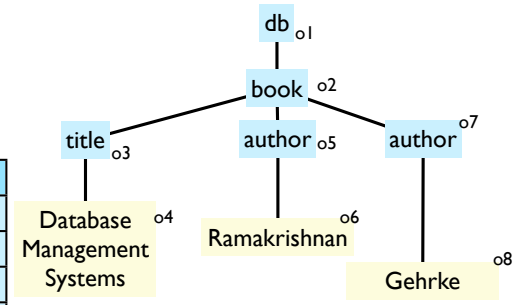
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8
o5	o9

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT
o9	foo	ELT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

- Delete(o5)



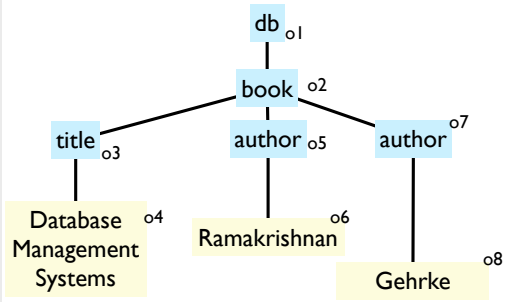
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
DELETE FROM Nodes
WHERE nodeId=o5;
<Trigger...>
DELETE FROM Edges
WHERE parent=o5
   OR child=o5;
DELETE FROM Nodes
WHERE nodeId=o6;
DELETE FROM Text
WHERE nodeId=o6
```



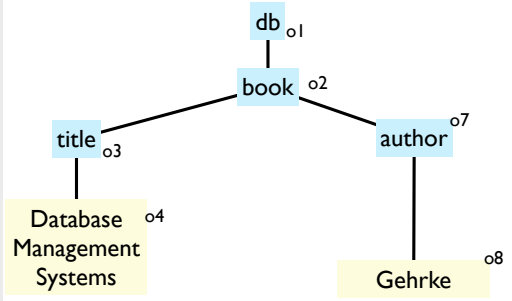
o2	o5
o5	o6
o2	o7
o7	o8

o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
DELETE FROM Nodes
WHERE nodeId=o5;
<Trigger...>
DELETE FROM Edges
WHERE parent=o5
   OR child=o5;
DELETE FROM Nodes
WHERE nodeId=o6;
DELETE FROM Text
WHERE nodeId=o6
```



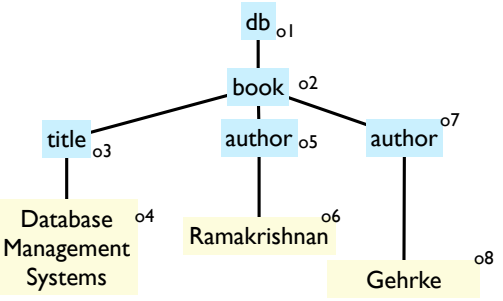
o2	o7
o7	o8

o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o8	Gehrke

Updating XML: naive

- Rename(o2,text)



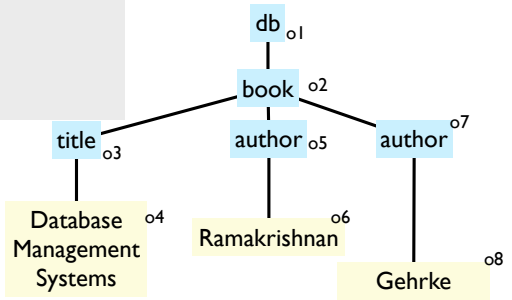
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
UPDATE Nodes
   SET tag='text'
 WHERE nodeId=o2
```



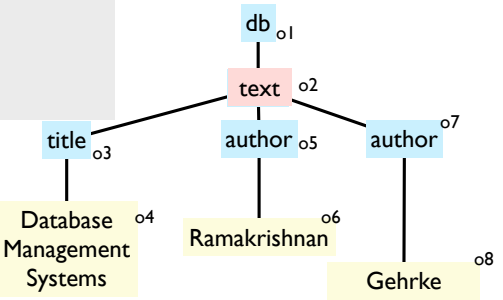
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
UPDATE Nodes
   SET tag='text'
 WHERE nodeId=o2
```



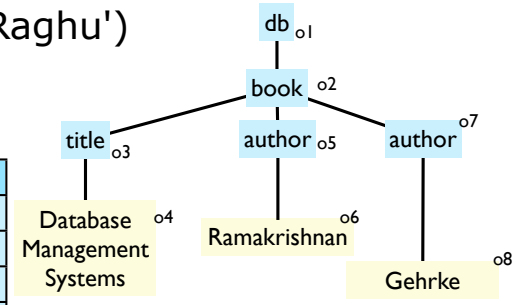
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	text	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

- ReplaceValue(o5,'Raghu')



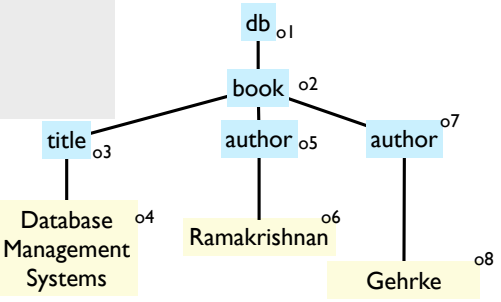
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
UPDATE Text
   SET text='Raghu'
 WHERE nodeId=o6
```



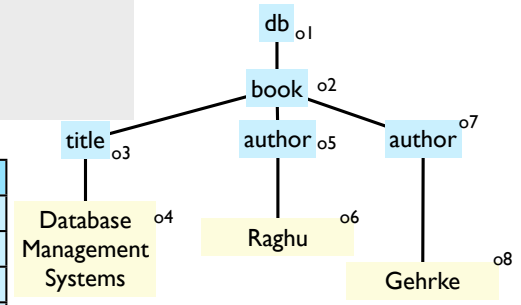
parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Ramakrishnan
o8	Gehrke

Updating XML: naive

```
UPDATE Text
   SET text='Raghu'
 WHERE nodeId=o6
```

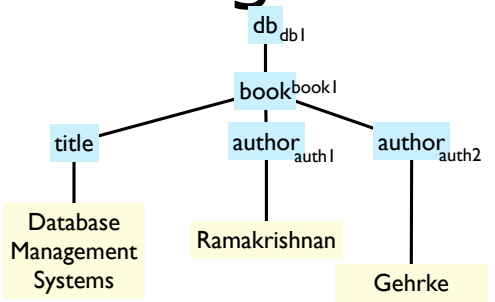


parent	child
o1	o2
o2	o3
o3	o4
o2	o5
o5	o6
o2	o7
o7	o8

nodeId	tag	type
o1	db	ELT
o2	book	ELT
o3	title	TEXT
o4		TEXT
o5	author	ELT
o6		TEXT
o7	author	ELT
o8		TEXT

nodeId	text
o4	Database Management Systems
o6	Raghu
o8	Gehrke

Updating XML: shared inlining



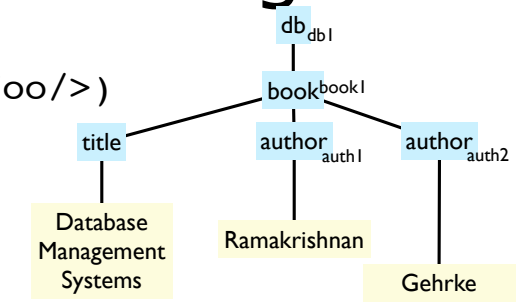
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
InsertInto(auth1,<foo/>)
```



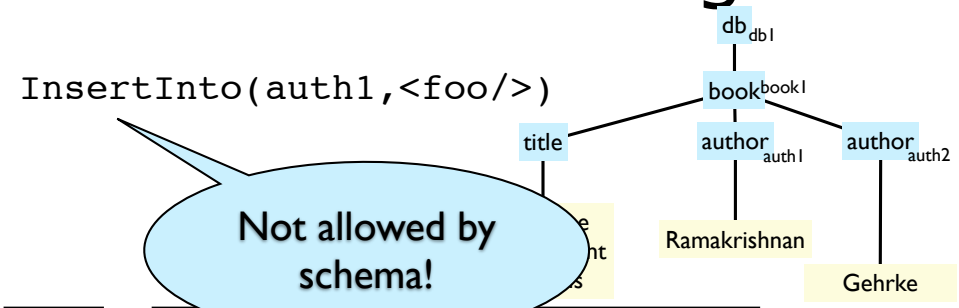
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
InsertInto(auth1,<foo/>)
```



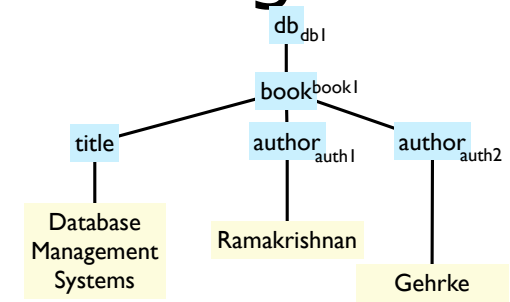
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
Delete(auth1)
```



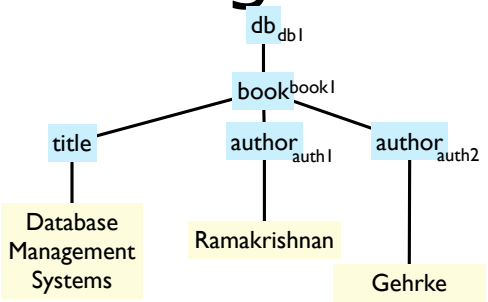
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
DELETE FROM authors
 WHERE authorId=auth1
```



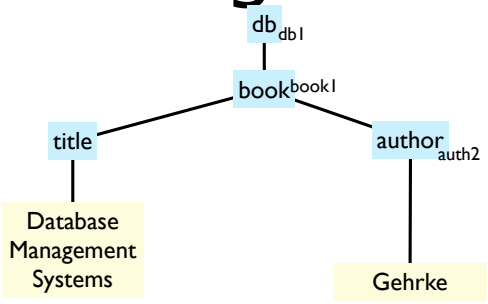
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
DELETE FROM authors
 WHERE authorId=auth1
```



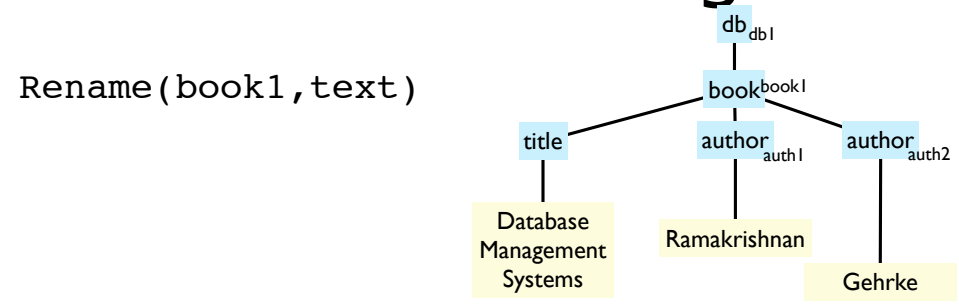
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth2	book1	Gehrke

Updating XML: shared inlining

```
Rename(book1,text)
```



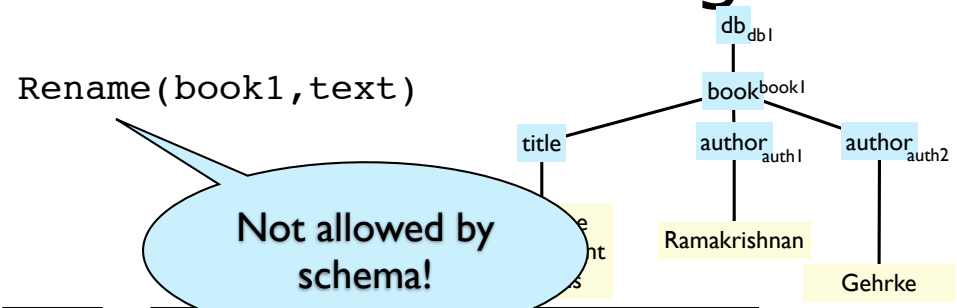
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
Rename(book1,text)
```



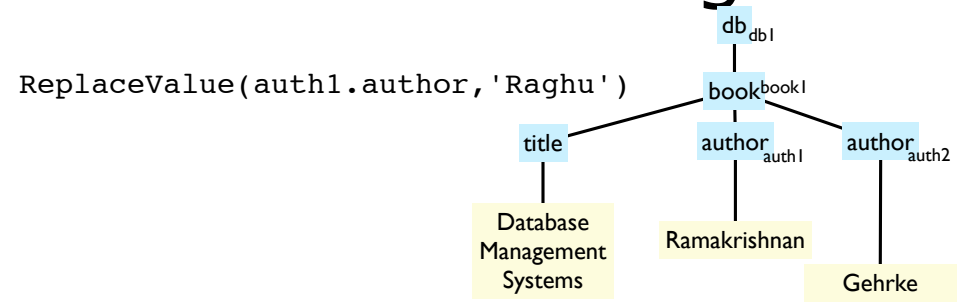
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
ReplaceValue(auth1.author,'Raghu')
```



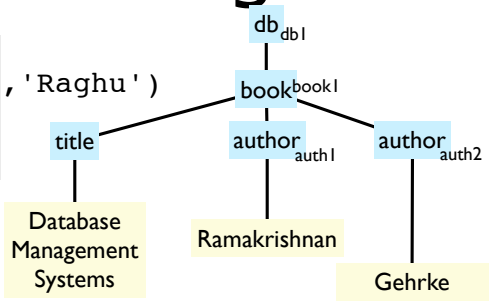
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
UPDATE authors
   SET author='Raghu'
 WHERE authorId=auth1
```



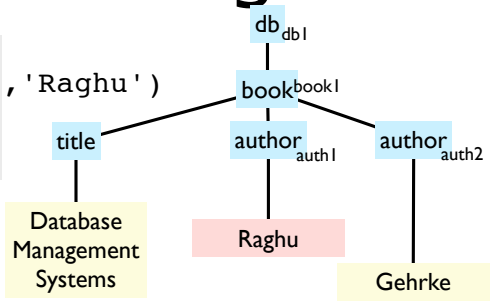
dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

authorID	bookID	author
auth1	book1	Ramakrishnan
auth2	book1	Gehrke

Updating XML: shared inlining

```
UPDATE authors
   SET author='Raghu'
 WHERE authorId=auth1
```



dbID
db1

bookID	parentID	code	title
book1	dbID	0	Database Management Systems

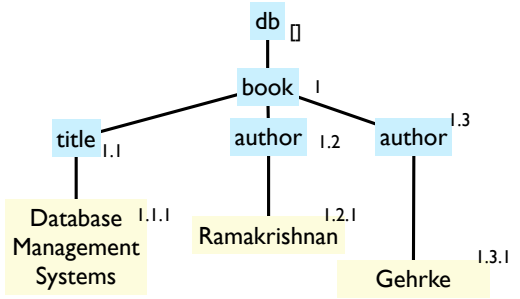
authorID	bookID	author
auth1	book1	Raghu
auth2	book1	Gehrke

Naive/Shared inlining: summary

- Rename, replace value, insert relatively straightforward
  - when allowed by schema (how to check this?)
- Delete can require recursion or triggers
  - can avoid this for a non-recursive schema (or delete of an element with no recursive children)
  - since there is a fixed upper bound on subtree depth

Updating XML: Dewey Decimal

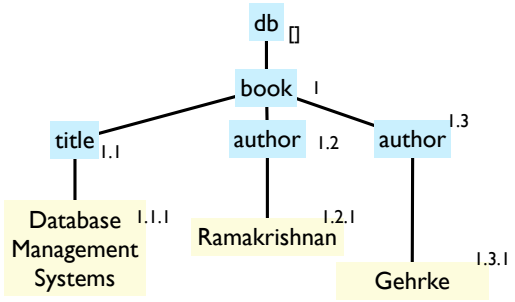
- Remember this?



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

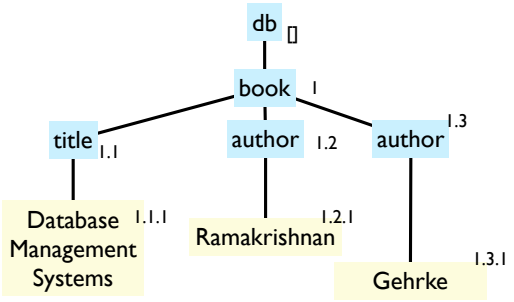
- InsertAsFirstInto(1.2,foo)



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

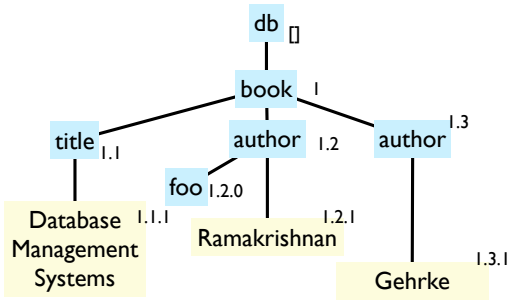
```
INSERT INTO Nodes
VALUE (1.2.0,'foo','ELT')
```



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

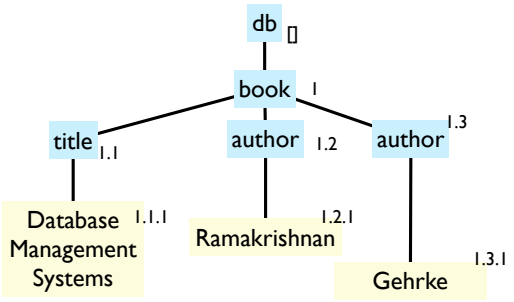
```
INSERT INTO Nodes
VALUE (1.2.0,'foo','ELT')
```



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT
1.2.0	foo	ELT

Updating XML: Dewey Decimal

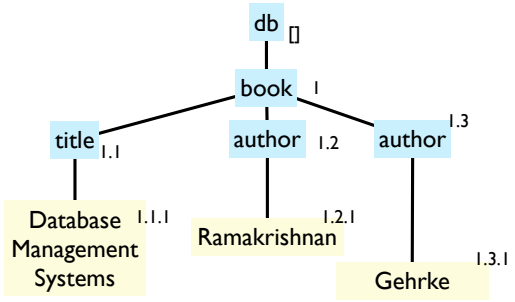
- InsertBefore(1.2,foo)



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

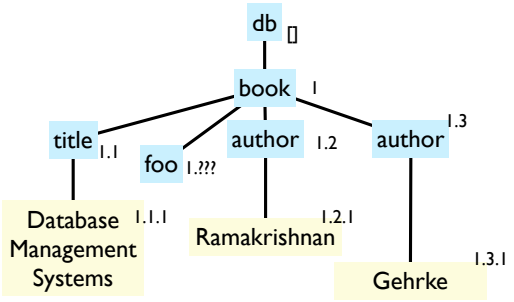
```
INSERT INTO Nodes
VALUE (1.???,'foo','ELT')
```



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

```
INSERT INTO Nodes
VALUE (1.???,'foo','ELT')
```

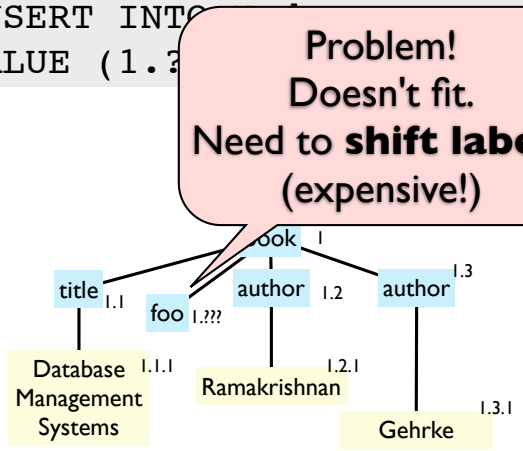


nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT
1.???	foo	ELT

Updating XML: Dewey Decimal

```
INSERT INT
VALUE (1.?
```

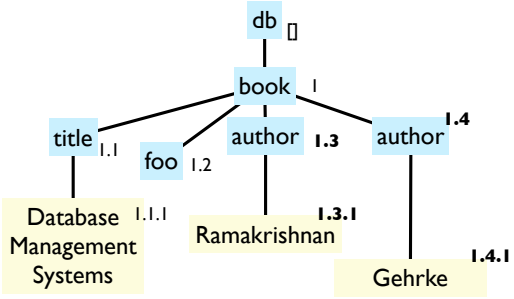
Problem!
Doesn't fit.
Need to **shift labels**
(expensive!)



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT
1.???	foo	ELT

Updating XML: Dewey Decimal

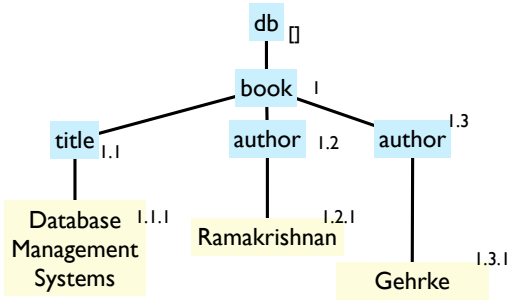
```
UPDATE <nodeIDs>
INSERT INTO Nodes
VALUE (1.2,'foo','ELT')
```



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.3	author	ELT
1.3.1		TEXT
1.4	author	ELT
1.4.1		TEXT
1.2	foo	ELT

Updating XML: Dewey Decimal

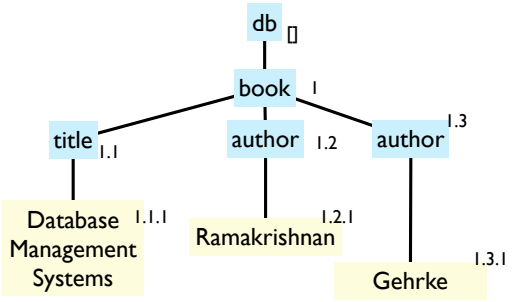
- Delete(1.2)



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

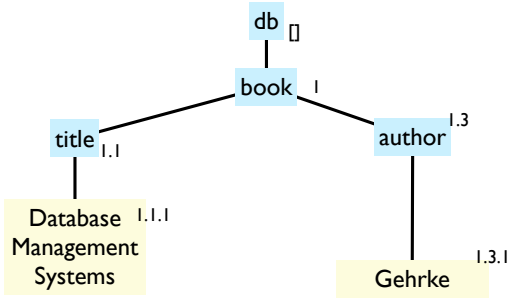
```
DELETE x from Nodes
 WHERE PREFIX([1.2],x.nodeId)
```



tag	type
db	ELT
book	ELT
title	ELT
	TEXT
author	ELT
	TEXT
author	ELT
	TEXT

Updating XML: Dewey Decimal

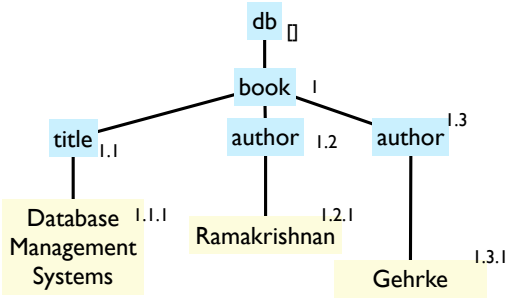
```
DELETE x from Nodes
 WHERE PREFIX([1.2],x.nodeId)
```



tag	type
db	ELT
book	ELT
title	ELT
	TEXT
author	ELT
	TEXT

Updating XML: Dewey Decimal

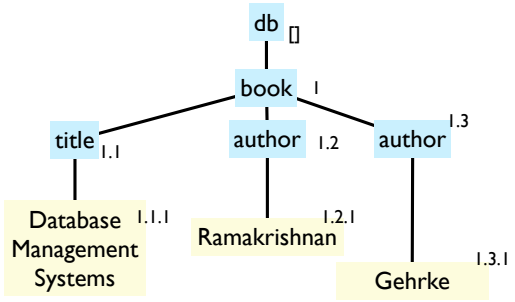
- Rename(1,'text')



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

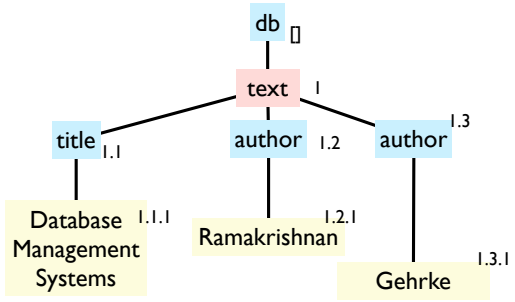
```
UPDATE Nodes x
   SET x.tag = 'text'
 WHERE x.nodeId = 1
```



nodeID	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

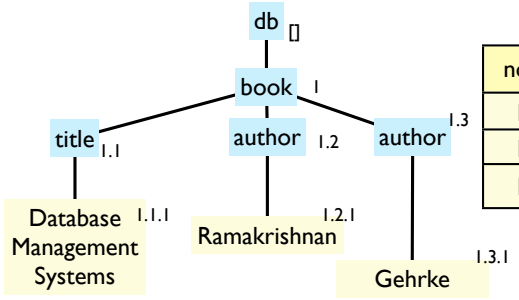
```
UPDATE Nodes x
   SET x.tag = 'text'
 WHERE x.nodeId = 1
```



nodeID	tag	type
[]	db	ELT
1	text	ELT
1.1	title	ELT
1.1.1		TEXT
1.2	author	ELT
1.2.1		TEXT
1.3	author	ELT
1.3.1		TEXT

Updating XML: Dewey Decimal

- ReplaceValue(1.2.1,Raghu)

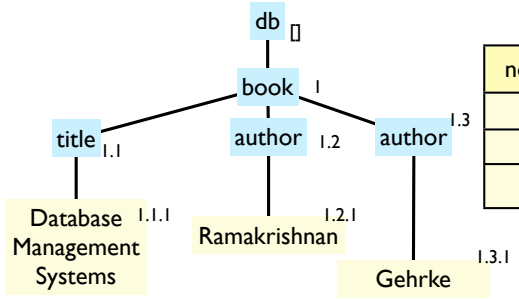


nodeId	tag	type
[]	db	ELT
1	book	ELT
1.1	title	ELT
1.3.1		TEXT

nodeId	text
1.1.1	Database Management Systems
1.2.1	Ramakrishnan
1.3.1	Gehrke

Updating XML: Dewey Decimal

```
UPDATE Text x
   SET x.txt = 'Raghu'
 WHERE x.nodeId = '1.2'
```



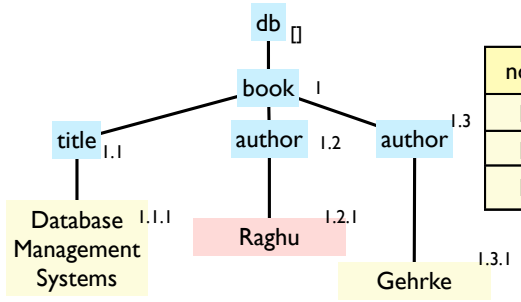
tag	type
db	ELT
book	ELT
title	ELT
	TEXT

nodeId	text
1.1.1	Database Management Systems
1.2.1	Ramakrishnan
1.3.1	Gehrke

1.3.1		TEXT

Updating XML: Dewey Decimal

```
UPDATE Text x
   SET x.txt = 'Raghu'
 WHERE x.nodeId = '1.2'
```

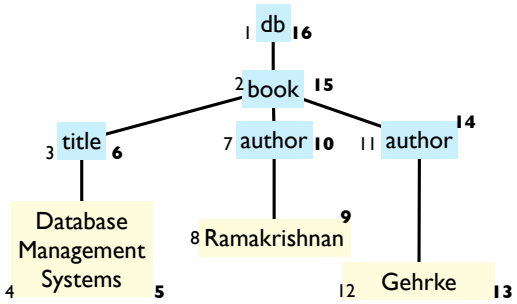


nodeId	text
1.1.1	Database Management Systems
1.2.1	Raghu
1.3.1	Gehrke

	tag	type
	db	ELT
	book	ELT
1.1	title	ELT
1.3.1		TEXT

Updating XML: interval encodings

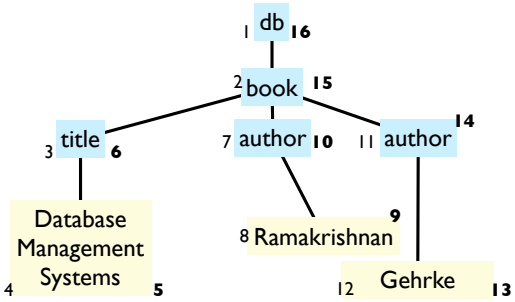
- Remember the interval approach:



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

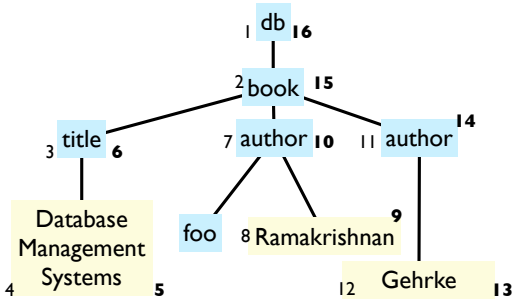
- InsertBefore(8,9,<foo/>) - trickier



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

- InsertBefore(8,9,<foo/>) - trickier

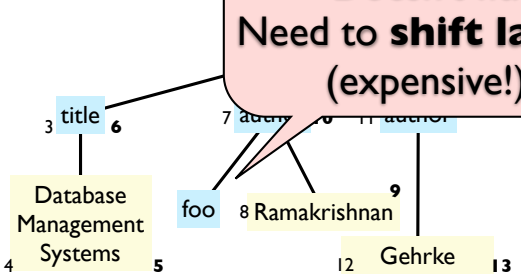


begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

- InsertB… …trickier

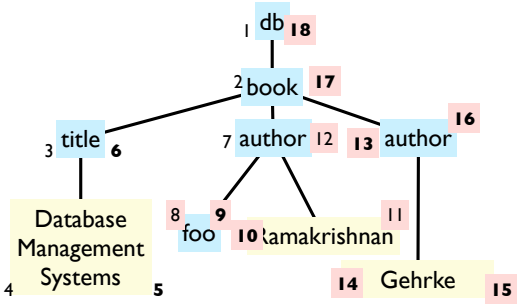
Problem!
Doesn't fit.
Need to **shift labels**
(expensive!)



	end	par	tag	type
	16		db	ELT
	15	1	book	ELT
	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

- InsertBefore(8,9,<foo/>) - trickier



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

```
UPDATE x from Nodes
SET x.begin=x.begin+2
WHERE x.begin >= 8
UPDATE x from Nodes
SET x.par = x.par+2
WHERE x.par >= 8
UPDATE x FROM Nodes
SET x.end = x.end+2
WHERE x.end >= 8

INSERT INTO Nodes
VALUES (8,9,7,foo,ELT)
```

- …>) - trickier

begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

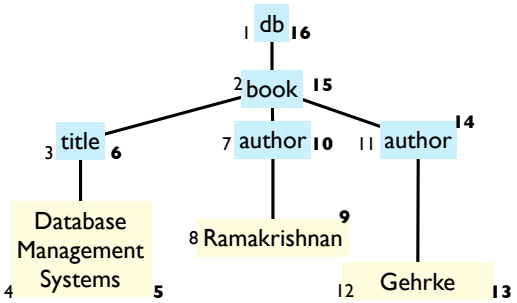
```
UPDATE x from Nodes
SET x.begin=x.begin+2
WHERE x.begin >= 8
UPDATE x from Nodes
SET x.par = x.par+2
WHERE x.par >= 8
UPDATE x FROM Nodes
SET x.end = x.end+2
WHERE x.end >= 8

INSERT INTO Nodes
VALUES (8,9,7,foo,ELT)
```

- …>) - trickier

begin	end	par	tag	type
1	18		db	ELT
2	17	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	12	2	author	ELT
8	9	7	foo	ELT
10	11	7		TEXT
13	16	2	author	ELT
14	15	13		TEXT

Updating XML: interval encodings

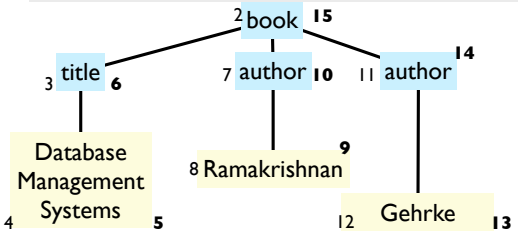
- Delete(7,10) - easy



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

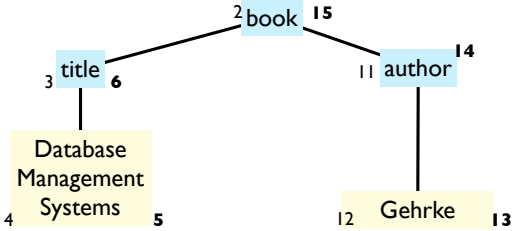
```
DELETE x from Nodes
 WHERE 7 <= x.begin
   AND x.end <= 10
```



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

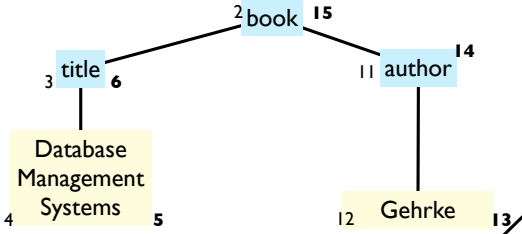
```
DELETE x from Nodes
 WHERE 7 <= x.begin
   AND x.end <= 10
```



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

```
DELETE x from Nodes
 WHERE 7 <= x.begin
   AND x.end <= 10
```

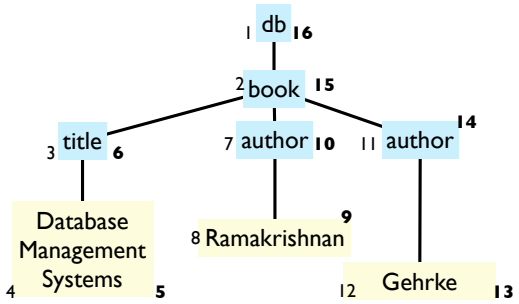


gaps are fine

begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

- Rename(2,15,"text") - easy

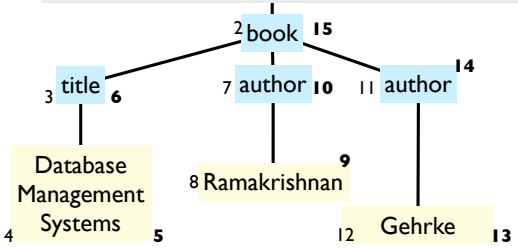


begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

```
UPDATE x from Nodes
SET x.tag='text'
WHERE x.nodeId = 8
```

easy

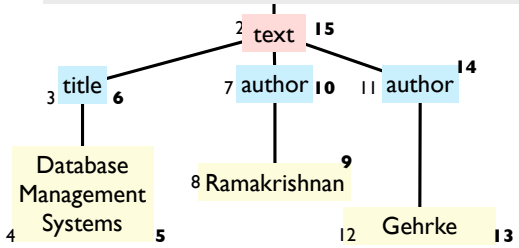


begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

```
UPDATE x from Nodes
SET x.tag='text'
WHERE x.nodeId = 8
```

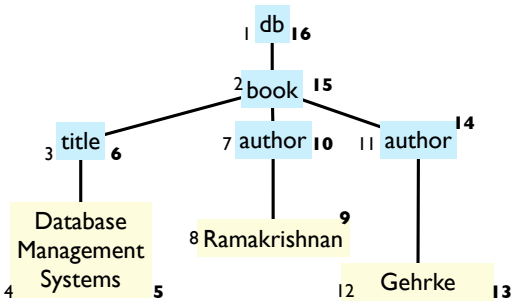
easy



begin	end	par	tag	type
1	16		db	ELT
2	15	1	text	ELT
3	6	2	title	ELT
4	5	3		TEXT
7	10	2	author	ELT
8	9	7		TEXT
11	14	2	author	ELT
12	13	11		TEXT

Updating XML: interval encodings

- ReplaceVal(8,9,"Raghu") - easy



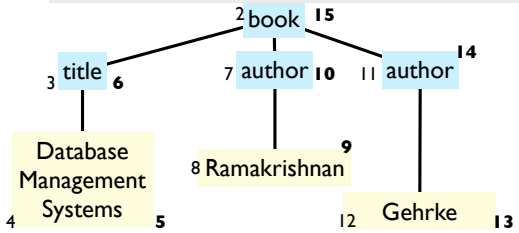
begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT
12	13	11		TEXT

nodeId	text
4	Database Management Systems
8	Ramakrishnan
12	Gehrke

Updating XML: interval encodings

```
UPDATE x from Nodes
SET x.text='Raghu'
WHERE x.nodeId = 8
```

- easy



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT

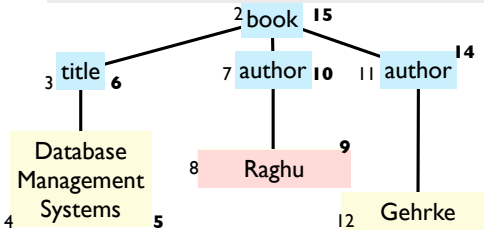
nodeId	text
4	Database Management Systems
8	Ramakrishnan
12	Gehrke

12	13	11		TEXT

Updating XML: interval encodings

```
UPDATE x from Nodes
SET x.text='Raghu'
WHERE x.nodeId = 8
```

- easy



begin	end	par	tag	type
1	16		db	ELT
2	15	1	book	ELT
3	6	2	title	ELT

nodeId	text
4	Database Management Systems
8	Raghu
12	Gehrke

12	13	11		TEXT

Updating interval encodings: summary

- Replace = delete + insert
  - can recycle "gap" left by delete, if inserted tree smaller
- Over time, deletes + inserts lead to "gaps"
  - Naive: export DB as XML and re-import (O(n) traversal of db though).
  - periodically clean up by finding gaps and shrinking them?
- Some work on leaving "holes" to decrease amortized cost of insertion

Atomic updates: summary

- Atomic updates: primitive change operators on XML data
- How can they be implemented?
  - Shared inlining: using recursive updates/triggers
  - Dewey: can translate to SQL using prefix, length
  - Interval: can translate to SQL using <
- All of these are expensive for some operations
  - Some recent work on "dynamic DDE" avoids this for Dewey approach
  - at cost of making XPath steps more complex
- There has been no comprehensive comparison of these approaches (AFAIK)

XQuery Update Facility

Bulk updates

- Atomic updates allow for changing one thing at a time
  - Widely supported
- But tedious to:
  - delete all 2012 students
  - increase all salaries where employee is in top 10% of sales
- Solution: **XQuery Update Facility**
  - extends XQuery to allow updating

Compare with SQL

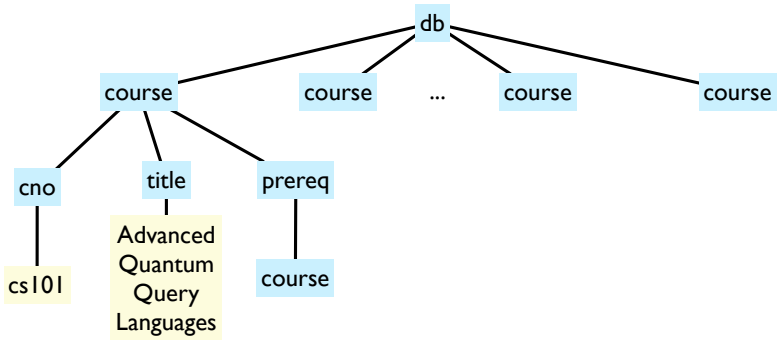
- Besides queries, SQL has "Data Manipulation Language"
  - i.e., bulk updates
  - which we've already seen in action
- Table updates:
  - `INSERT INTO table VALUES r1,...,rn`
  - `DELETE FROM table WHERE condition`
  - `UPDATE table SET t.a = v ... WHERE condition`
- But also create/delete tables (`CREATE TABLE, DROP TABLE`), add/remove columns, (`ALTER TABLE`) etc.
- For XML, no built-in distinction between table, record, field: need *uniform* mechanism for updates

Updating XML

[Tatarinov et al.]

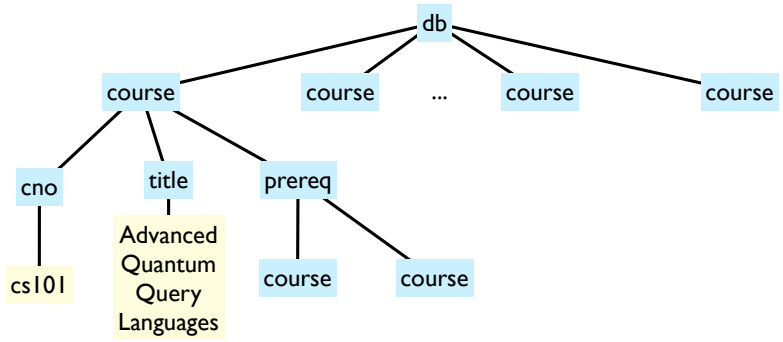
- First paper to propose XQuery extensions for XML updates
- Idea:
  - use XPath/XQuery operations to *select* target nodes of updates (including conditional tests)
  - use XQuery to *construct* subtrees (for insert/replace)
  - Create a *pending update list* that is applied in one shot
- XQuery Update Facility uses similar approach

Example XPath-based updates




```
insert node <course>...</course>
into //course[cno='cs101']/prereq
```

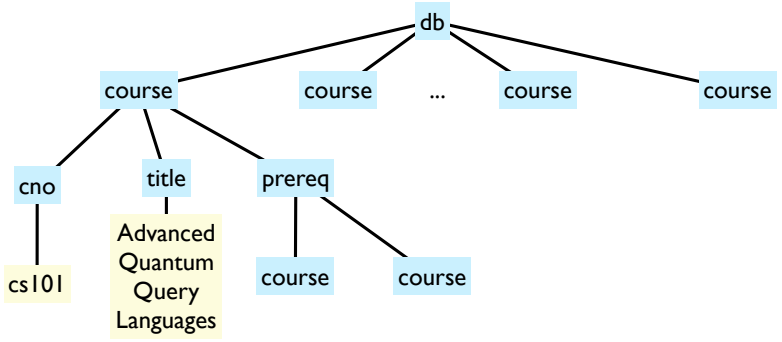

Example XPath-based updates




```
insert node <course>...</course>
into //course[cno='cs101']/prereq
```

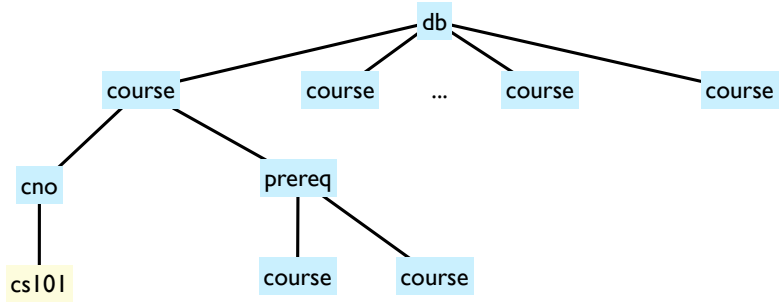
QSX February 26-March 1, 2013
47

Example XPath-based updates




```
delete nodes //course[cno='cs101']/title
```


Example XPath-based updates




```
delete nodes //course[cno='cs101']/title
```

QSX February 26-March 1, 2013
48

XQuery Update Facility

- Extends queries with updating expressions

```
u ::= insert node(s) exp
      (as first|as last) into path
    | insert node(s) exp
      (before|after) path
    | delete node(s) path
    | replace node(s) path with exp
    | replace value of node path with exp
    | rename node path as name
```

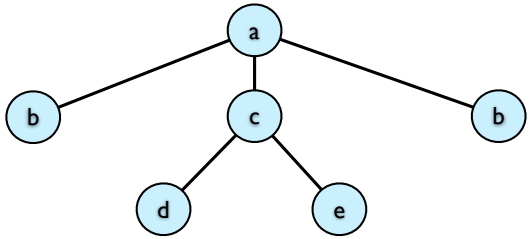

Insert Into

```
insert node(s) exp
(as first|as last) into path
```

- where *exp* is an XQuery expression that constructs subtree value
- and *path* is an XPath expression that selects node(s)
  - these will be **parents** of inserted exp
  - "as first"/"as last" govern where nodes inserted
  - if not specified, it's up to implementation

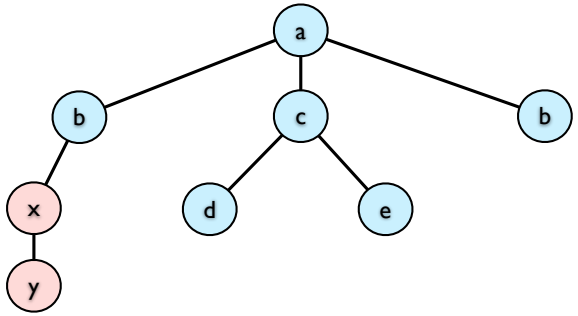
Example

```
insert node <x><y/></x>
as first into /a/*
```



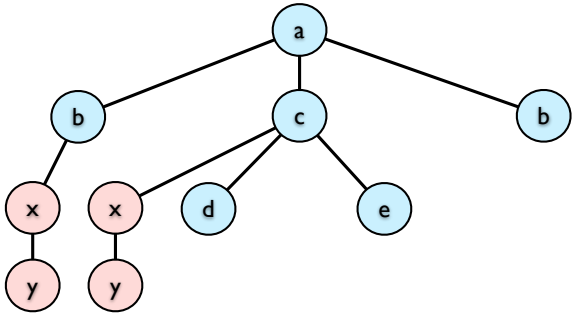
Example

```
insert node <x><y/></x>
as first into /a/*
```



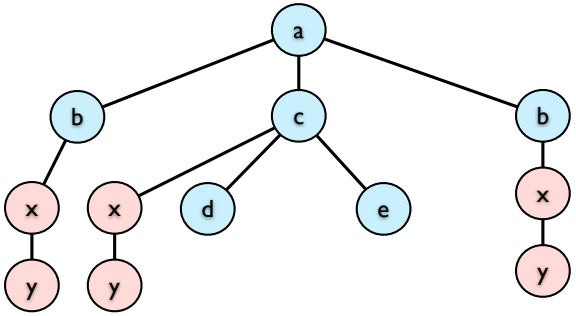
Example

```
insert node <x><y/></x>
as first into /a/*
```



Example

```
insert node <x><y/></x>
as first into /a/*
```



QSX February 26-March 1, 2013
51

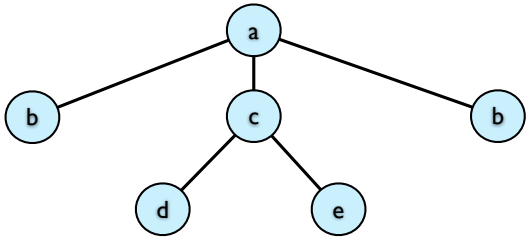
Insert Before/After

insert node(s) *exp*
(before|after) into *path*

- where *exp* is an XQuery expression that constructs subtree value
- and *path* is an XPath expression that selects node(s)
  - these will be **siblings** of inserted exp
  - "as first"/"as last" govern where nodes inserted
  - if not specified, it's up to implementation

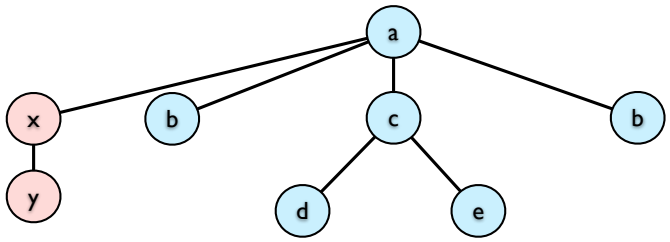
Example

```
insert node <x><y/></x>
before /a/b
```



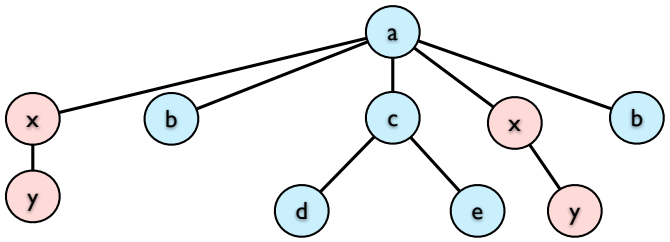
Example

```
insert node <x><y/></x>
before /a/b
```



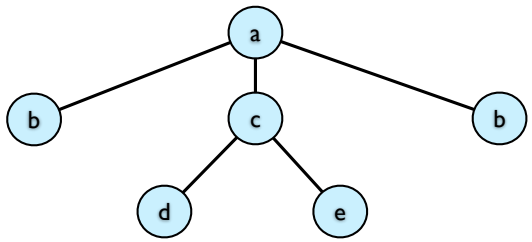
Example

```
insert node <x><y/></x>
before /a/b
```



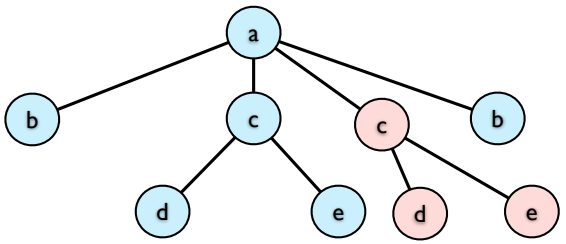
Example:Copy-paste

```
for $x in /a/c
insert node $x
before /a/*[3]
```



Example:Copy-paste

```
for $x in /a/c
insert node $x
before /a/*[3]
```



Other updates

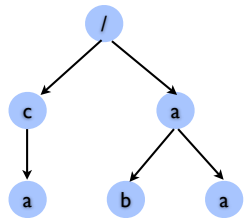
- delete: obvious (delete selected nodes)
- replace: similar to delete + insert
  - replace value allows changing string values
- rename: changes name while keeping structure fixed

Evaluating complex updates

- Update evaluation is multi-stage / snapshot:
  - Evaluate query & update expressions to form **pending update list** (PUL)
    - all constructed values are built using **old version**
  - Check PUL is minimally sensible
    - e.g. does not rename same node to two different names
  - Finally, **reorder & apply** updates
- Good in that it avoids strange behaviors
- But this may not do what you expect!

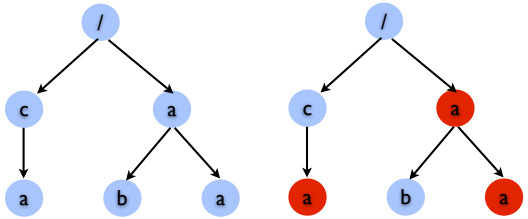
Example

$x



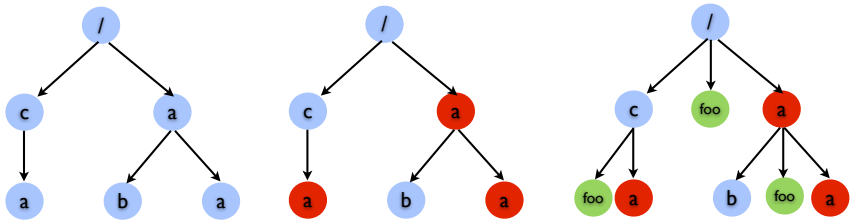

```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```


First **collect** updates



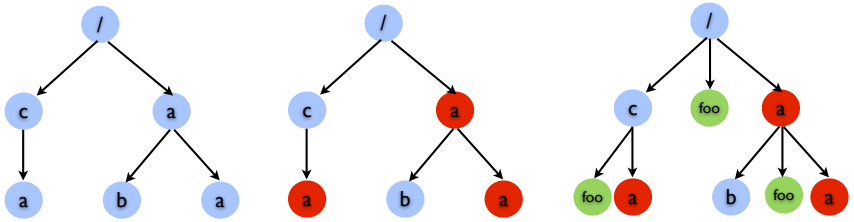

```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```

First **collect** updates



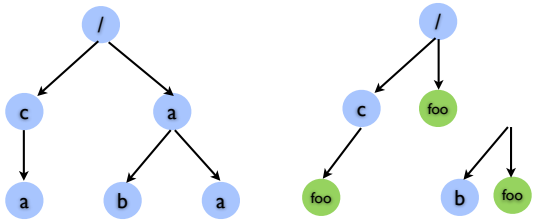

```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```


Then **reorder** & **apply**



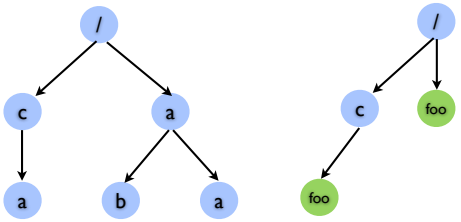

```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```


Then **reorder** & **apply**




```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```


Then **reorder** & **apply**




```
delete $x//a,
insert <foo>bar</foo>
  before $x//a
```

XQuery Update Facility: Transforms

- Queries can perform updates **locally**

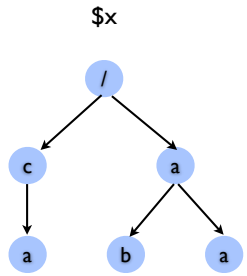
```
u ::= insert node(s) exp
      (as first|as last) into path
    | insert node(s) exp
      (before|after) path
    | delete node(s) path
    | replace node(s) path with exp
    | replace value of node path with exp
    | rename node path as name
q ::= copy $x := exp, ...
      modify u
      return exp
```


Transform queries (hypothetical)

```
copy $x1 := exp1, ..., $xn := expn
modify u
return exp
```

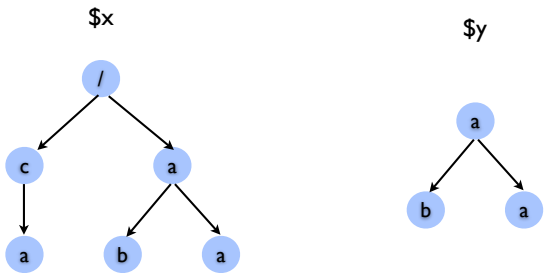
- Evaluate exp_1, bind to $\$x_1$, ...
- Evaluate exp_n, bind to $\$x_n$
- Apply update *u*
  - only $\$x_1 \ldots \$x_n$ are mutable, other vars cannot be updated
- Evaluate & return *exp*
- Key point: No side-effects on database

Example



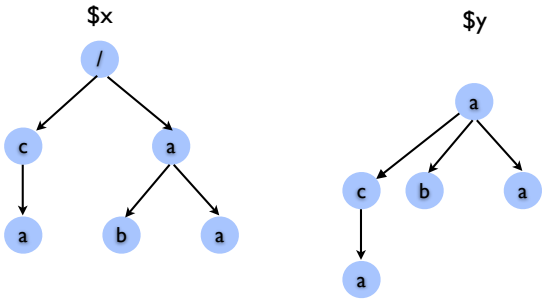

```
copy $y := $x/a
modify insert nodes $x/c
       as first into $y
return <result>$y</result>
```


Example: Make copy



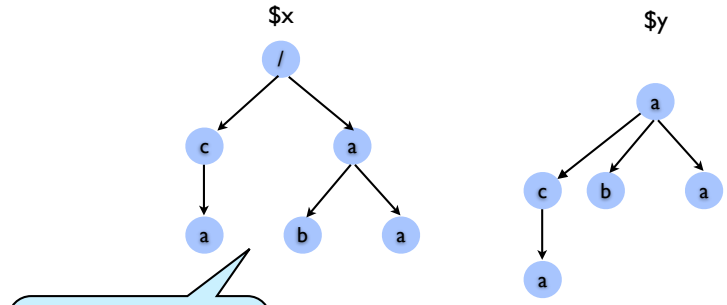

```
copy $x := $doc/a
modify insert nodes $doc/c
       as first into $x
return <result>$x</result>
```

Example: Apply updates to copy




```
copy $x := $doc/a
modify insert nodes $doc/c
       as first into $x
return <result>$x</result>
```

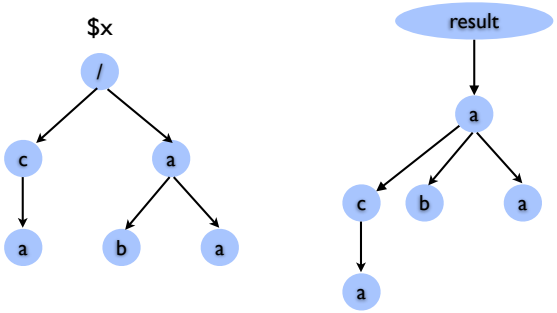

Example: Apply updates to copy



No side effects on $x

```
copy $x := $doc/a
modify insert nodes $doc/c
       as first into $x
return <result>$x</result>
```

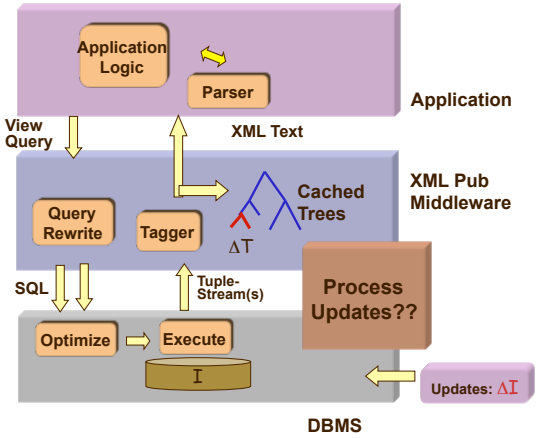

Example: Final result




```
copy $x := $doc/a
modify insert nodes $doc/c
       as first into $x
return <result>$x</result>
```


Incremental maintenance of XML views

Goal: Incremental Update

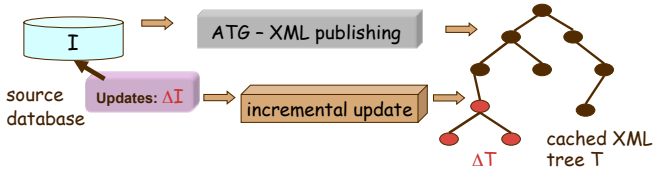


Why incremental update?

[Bohannon et al. 2004]

- Goal: update external materialized XML tree in response to changes ΔI to the underlying database
- Batch computation: recompute the entire tree from scratch;
  - large XML views may take multiple hours or days to produce!
- Incremental computation: compute XML change ΔT
  - Idea: the new view T′ = the old view T + ΔT
  - Why? the new view T′ often differs **only slightly** from the old view T – reuse partial results computed earlier
  - Typically more efficient to compute ΔT (small) and update the old view T with ΔT
- Incremental computation: an effective technique with a wide range of applications

Coping with source updates



- Problem: the underlying database may be updated constantly (ΔI)
  - e.g. movies database - new movies coming out all the time
- Goal: update the published (materialized) XML tree in response to source changes ΔI -- updating the treatment hierarchies
- Incrementally compute XML change ΔT
- such that the new view T′ = the old view T + ΔT

Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid        $year = $Movie.year
$title := $Movie.title   $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```

Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Example

```
insert Movies(42,Star Wars,1977)
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Example

```
insert Movies(42,Star Wars,1977)
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```

Example

```
insert Movies(42,Star Wars,1977)
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

doc -

```
$Movie
```

Movie

```
$id :=
$title
```

Actor

```
$Actor :=
```

Actor

```
$id :=
$lname
$fname
```

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
<Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="42">
 <Title>Star Wars</Title>
 <Year>1977</Year>
</Movie>
```


Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

{+Movies(42,Star Wars,1977)}

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

{+Movies(42,Star Wars,1977)}

doc -> Movie*

```
$Movie := +<Movie>...</Movie>e,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```

Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

{+Movies(42,Star Wars,1977)}

```
doc -> Movie*
$Movie := +<Movie>...</Movie>e,year
          from Movies
```

+@id='42',<Title>Star Wars</Title><Year>1977</Year>...

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

```
Actors -> Actor*
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

```
Actor -> id,lname,fname
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005
42	Star Wars	1977

Appears

mid	aid
11	1
11	2
32	2

{+Movies(42,Star Wars,1977)}

```
doc -> Movie*
$Movie := +<Movie>...</Movie>e,year
          from Movies
```

+@id='42',<Title>Star Wars</Title><Year>1977</Year>...

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> A + no new actors

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

```
Actor -> id,lname,fname
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


More complex example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
doc -> Movie*
$Movie := select mid,title,year
          from Movies
```

```
Movie -> id,title,year,Actors
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

```
Actors -> Actor*
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

```
Actor -> id,lname,fname
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


More complex example

```
insert Actors(3,'Bloom','Orlando')
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten
3	Bloom	Orlando

Movies

mid	title	year
32	Elizabethtown	2005

```
insert Appears(32,3)
delete Appears(11,2)
```

Appears

mid	aid
11	1
32	2
32	3

```
doc -> Movie*
$Movie := select mid,title,year
          from Movies
```

```
Movie -> id,title,year,Actors
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

```
Actors -> Actor*
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

```
Actor -> id,lname,fname
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```

More complex example

```
insert Actors(3,'Bloom','Orlando')
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten
3	Bloom	Orlando

Movies

mid	title	year
32	Elizabethtown	2005

```
insert Appears(32,3)
delete Appears(11,2)
```

Appears

mid	aid
11	1
32	2
32	3

doc

```
$Mov
```

Movi

```
$id :=
$title
```

Acto

```
$Actor
```

Acto

```
$id :=
$lname
$fname
```

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
<!-- deleted -->
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
 <Actor id="3">
  <LName>Bloom</LName>
  <FName>Orlando</FName>
 </Actor>
</Movie>
```


More complex example

```
insert Actors(3,'Bloom','Orlando')
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten
3	Bloom	Orlando

Movies

mid	title	year
32	Elizabethtown	2005

```
insert Appears(32,3)
delete Appears(11,2)
```

Appears

mid	aid
11	1
32	2
32	3

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


More complex example

```
insert Actors(3,'Bloom','Orlando')
```

Actors

aid	lname	fname

Trickier - need to find keys/paths to subtrees that need to change

Movies

mid	title	year
32	Elizabethtown	2005

Appears

mid	aid
11	1
32	2
32	3

doc -> Movie*

```
$Movie := select mid,title,year
          from Movies
```

Movie -> id,title,year,Actors

```
$id := $Movie.mid       $year = $Movie.year
$title := $Movie.title  $Actors = $Movie.mid
```

Actors -> Actor*

```
$Actor := select a.aid,a.lname,a.fname
          from actors a, appears app
          where app.mid=$Actors.mid,app.aid=a.aid
```

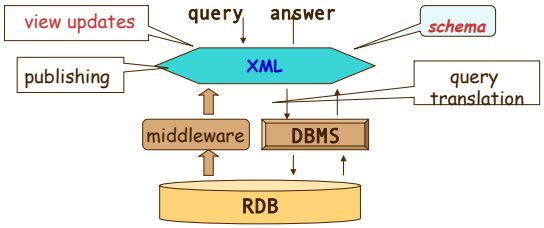
Actor -> id,lname,fname

```
$id := $Actor.aid
$lname := $Actor.lname
$fname := $Actor.fname
```


Updating XML views of relations

View updates

- XML view updates: propagation from XML to relations
  - Input: a mapping σ from DB R to XML T, and XML updates ΔT
  - Output: updated database R such that T + ΔT = σ (R + ΔR)
- Already hard for relational views



View updates: hard even for relational views

- Input: a relational view definition σ, an instance of relational database I of schema R, a view V = σ(I), and view updates ΔV
- Output: database updates ΔI such that V + ΔV = σ (I + ΔI)
- May not be updatable:
  - Schema: R(A, B), S(B, C);
  - View: Π_{AC} (R ⋈ S)
  - View delete: remove (a_1, c_1).
  - Not doable without side effect (the deletion of (a_1, c_2) or (a_2, c_1))

R:

A	B
a1	b1
a2	b1

S:

B	C
b1	c1
b1	c2

V:

A	C
a1	c1
a1	c2
a2	c1
a2	c2

More on relational view updates

- May not have a unique answer
  - Schema: R(A, B), S(B, C);
  - View: Π_{AC} (R ⋈ S)
  - View delete: remove (a_1, c_1).
  - Not doable without side effect (the deletion of (a_1, c_2) or (a_2, c_1))

R:

A	B
a1	b1
a1	b2

S:

B	C
b1	c1
b2	c1

V:

A	C
a1	c1

Complexity of relational updates

[Buneman et al. 2002]

- View updatability problem:
  - **given** a relational view definition σ, an instance of relational database I of schema R, a view V = σ(I), and view updates ΔV
  - decide whether the view is **updatable**, ie, whether there exists a **side-effect-free** database update ΔI such that V + ΔV = σ (I + ΔI)
  - **NP-hard** for relational views defined with Projection+Join (PJ) or Join +Union (JU) only, even for only deletions
- Minimal view update problem:
  - **given** a relational view definition σ, an instance of relational database I of schema R, a view V = σ(I), and view insertions (resp. deletions) ΔV
  - find the **smallest** database update ΔI such that V + ΔV = σ (I + ΔI)
  - **NP-hard** for relational views defined with PJ or JU only, even for only deletions

XML view updates

- Input: an ATG σ from DB R to XML T, and XML updates ΔT
- Output: updated database R such that T + ΔT = σ (R + ΔR)
- XML updates:
  - `insert e into p`
  - `delete p`
- where p: XPath query; e: an XML element/subtree
- Suppose that T is stored as edge relations – relational view V
- Approach:
  - Translate ΔT to ΔV
  - Resolve the relational view update problem: from ΔV to base relational updates ΔR

Example

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Example

```
delete /Movie[@id=11]/Actor[@id=2]
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Example

```
delete /Movie[@id=11]/Actor[@id=2]
```

Actors

delete Kirsten Dunst from Spiderman?

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```

Example

```
delete /Movie[@id=11]/Actor[@id=2]
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabeth	

Appears

mid	
11	1
11	2
32	2

One way: delete Appears(11,2)

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
 ...
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Example

```
delete /Movie[@id=11]/Actor[@id=2]
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	tit	
11	Spide	
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

Another way: delete Actors(2,Dunst,Kirsten)

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Example

```
delete /Movie[@id=11]/Actor[@id=2]
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year

But this has side-effects!

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 <Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Another tricky one

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 ...
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```

Another tricky one

```
insert nodes <Actor id='1'>...</Actor
into /Movie[@id='32']
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 ...
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
</Movie>
```


Another tricky one

```
insert nodes <Actor id='1'>...</Actor
into /Movie[@id='32']
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

insert Tobey Maguire into Elizabethtown?

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 ...
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
</Movie>
```


Another tricky one

```
insert nodes <Actor id='1'>...</Actor
into /Movie[@id='32']
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabethtown	2005

Appears

mid	aid
11	1
11	2
32	2

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 ...
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  <LName>Dunst</LName>
  <FName>Kirsten</FName>
 </Actor>
<Actor id="1">
  <LName>Maguire</LName>
  <FName>Tobey</FName>
 </Actor>
</Movie>
```


Another tricky one

```
insert nodes <Actor id='1'>...</Actor
into /Movie[@id='32']
```

Actors

aid	lname	fname
1	Maguire	Tobey
2	Dunst	Kirsten

Movies

mid	title	year
11	Spider-Man	2002
32	Elizabe	

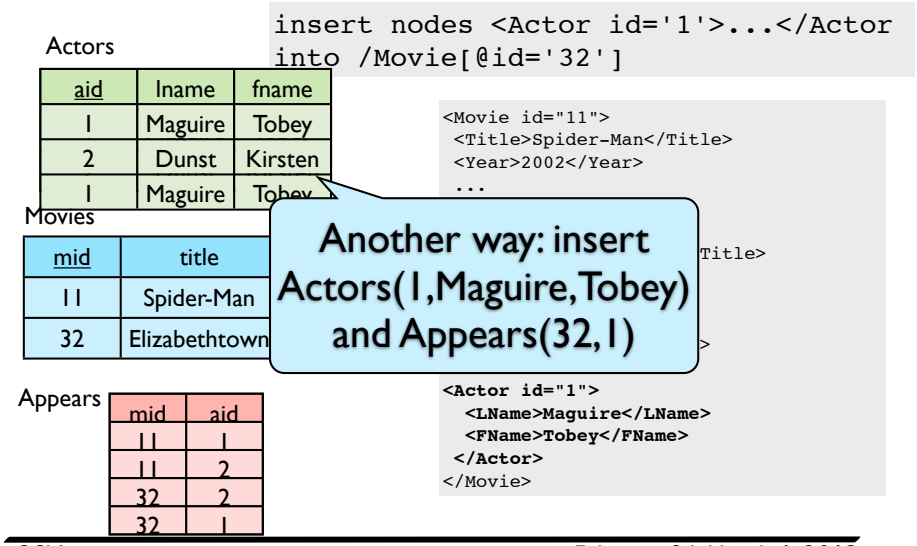
One way: add to Appears

Appears

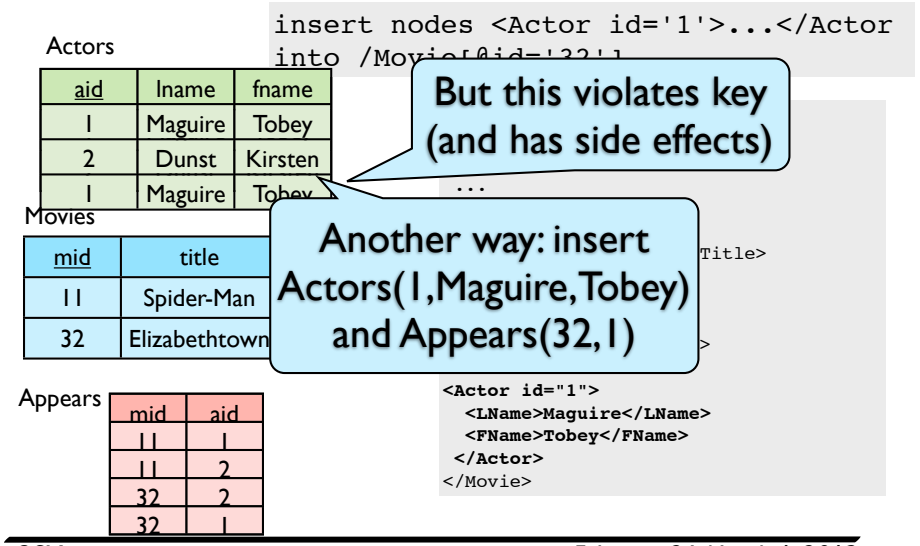
mid	aid
11	1
11	2
32	2
32	1

```
<Movie id="11">
 <Title>Spider-Man</Title>
 <Year>2002</Year>
 ...
</Movie>
<Movie id="32">
 <Title>Elizabethtown</Title>
 <Year>1999</Year>
 <Actor id="2">
  </LName>
  en</FName>
 </Actor>
<Actor id="1">
  re</LName>
  <FName>Tobey</FName>
 </Actor>
</Movie>
```

Another tricky one



Another tricky one



Summary

- XQuery Update Facility
  - allows for bulk updates
  - semantics somewhat complex
- Updating views of XML stored in relations
  - view update: challenging in general
  - special case: XPath on ATG-defined publishing views
- XML/semistructured updates & optimizations remain active research topics

Summary; research areas

- Controlling when/how updates performed
  - snapshot (all at once at end) vs. explicit commit
  - concurrency control
- Dynamic optimization of updates
- Translating XML updates to relational updates
- Typechecking results of updates
- Analyzing when update interferes with query
  - avoiding view recomputation

Next time

- XML typechecking & static analysis
  - XDuce: A statically typed XML processing language
  - Static analysis for path correctness of XML queries
  - Schema-based independence analysis for XML updates