

XML-Relational Mapping

CPS 116

Introduction to Database Systems

Announcements (October 28)

- ❖ Homework #2 has been graded
- ❖ Homework #3 out last Thursday; due next Thursday
- ❖ Project milestone #2 due in 2 weeks
 - Check your email for feedback

Approaches to XML processing

- ❖ Text files (!)
- ❖ Specialized XML DBMS
 - Lore (Stanford), Strudel (AT&T), Timber (Michigan), MonetDB/XQuery (CWI, Netherlands), Tamino (Software AG), eXist, Sedna, Apache XIndex, XML:DB API initiative...
 - Still a long way to go
- ❖ Object-oriented DBMS
 - ObjectStore, ozone, ...
 - Not as mature as relational DBMS
- ❖ Relational (and object-relational) DBMS
 - Middleware and/or object-relational extensions

Mapping XML to relational

- ❖ Store XML in a GLOB (Character Large Object) column
 - Simple, compact
 - Full-text indexing can help (often provided by DBMS vendors as object-relational "extensions")
 - Poor integration with relational query processing
 - Updates are expensive
- ❖ Alternatives?
 - Schema-oblivious mapping: well-formed XML → generic relational schema
 - Node/edge-based mapping for graphs
 - Interval-based mapping for trees
 - Path-based mapping for trees
 - Schema-aware mapping: valid XML → special relational schema based on DTD

Node/edge-based: schema

- ❖ $Element(eid, tag)$
- ❖ $Attributes(eid, attrName, attrValue)$ *Key: (eid, attrName)*
 - Attribute order does not matter
- ❖ $ElementChild(eid, pos, child)$ *Keys: (eid, pos), (child)*
 - *pos* specifies the ordering of children
 - *child* references either $Element(eid)$ or $Text(tid)$
- ❖ $Text(tid, value)$
 - *tid* cannot be the same as any *eid*
- ⚡ Need to "invent" lots of *eid*'s
- ⚡ Need indexes for efficiency, e.g., $Element(tag)$, $Text(value)$

Node/edge-based: example

```

<bibliography>
  <book ISBN="15261-101" price="20.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Hull</author>
    <author>Vian</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>
</bibliography>
    
```

Attributes		
eid	attrName	attrValue
e1	ISBN	15261-10
e1	price	20

Text	
tid	value
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vian
t4	Addison Wesley
t5	1995

Element	
eid	tag
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

ElementChild		
eid	pos	child
e0	1	e1
e1	1	e2
e1	2	e3
e1	3	e4
e1	4	e5
e1	5	e6
e1	6	e7
e2	1	t0
e2	1	t1
e2	1	t2
e2	1	t3
e2	1	t4
e2	1	t5

Node/edge-based: simple paths

- ❖ `//title`
 - `SELECT eid FROM Element WHERE tag = 'title';`
- ❖ `//section/title`
 - `SELECT e2.eid`
`FROM Element e1, ElementChild c, Element e2`
`WHERE e1.tag = 'section'`
`AND e2.tag = 'title'`
`AND e1.eid = c.eid`
`AND c.child = e2.eid;`
- ☞ Path expression becomes joins!
 - Number of joins is proportional to the length of the path expression

Node/edge-based: more complex paths

- ✦ `//bibliography/book[author="Abiteboul"]/@price`
 - `SELECT a.attrValue`
`FROM Element e1, ElementChild c1,`
`Element e2, Attribute a`
`WHERE e1.tag = 'bibliography'`
`AND e1.eid = c1.eid AND c1.child = e2.eid`
`AND e2.tag = 'book'`
`AND EXISTS (SELECT * FROM ElementChild c2,`
`Element e3, ElementChild c3, Text t`
`WHERE e2.eid = c2.eid AND c2.child = e3.eid`
`AND e3.tag = 'author'`
`AND e2.eid = c3.eid AND c3.child = t.tid`
`AND t.value = 'Abiteboul')`
 - `AND e2.eid = a.eid`
`AND a.attrName = 'price';`

Node/edge-based: descendent-or-self

- ❖ `//book//title`
 - Requires SQL3 recursion
 - `WITH ReachableFromBook(id) AS`
`((SELECT eid FROM Element WHERE tag = 'book')`
`UNION ALL`
`(SELECT c.child`
`FROM ReachableFromBook r, ElementChild c`
`WHERE r.eid = c.eid))`
`SELECT eid`
`FROM Element`
`WHERE eid IN (SELECT * FROM ReachableFromBook)`
`AND tag = 'title';`

Interval-based: schema

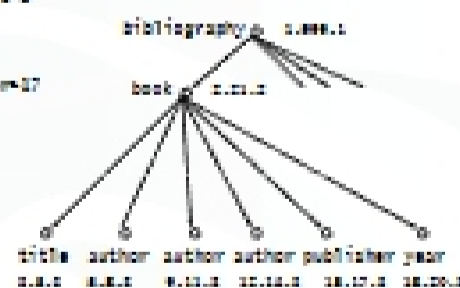
- ❖ `Element(left, right, level, tag)`
 - *left* is the start position of the element
 - *right* is the end position of the element
 - *level* is the nesting depth of the element (strictly speaking, unnecessary)
 - *Key* is *left*
- ❖ `Text(left, right, level, value)`
 - *Key* is *left*
- ❖ `Attribute(left, attrName, attrValue)`
 - *Key* is (*left, attrName*)

Interval-based: example

```

bibliography
  <book ISBN="0201-00" price="20.00">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Horn</author>
    <author>Korth</author>
    <publisher>Addison Wesley</publisher>
    <year>1995</year>
  </book>
</bibliography>

```



☞ Where did *ElementChild* go?

- *E1* is the parent of *E2* iff:
 $[E1.left, E1.right] \supset [E2.left, E2.right]$, and
 $E1.level = E2.level - 1$

Interval-based: queries

- ❖ `//section/title`
 - `SELECT e2.left`
`FROM Element e1, Element e2`
`WHERE e1.tag = 'section' AND e2.tag = 'title'`
`AND e1.left < e2.left AND e2.right < e1.right`
`AND e1.level = e2.level - 1;`
 - ☞ Path expression becomes "containment" joins!
 - Number of joins is proportional to path expression length
- ❖ `//book//title`
 - `SELECT e2.left`
`FROM Element e1, Element e2`
`WHERE e1.tag = 'book' AND e2.tag = 'title'`
`AND e1.left < e2.left AND e2.right < e1.right;`
 - ☞ No recursion!

Summary of interval-based mapping

13

- ❖ Path expression steps become containment joins
- ❖ No recursion needed for descendent-or-self
- ❖ Comprehensive XQuery-SQL translation is possible

A path-based mapping

14

Label-path encoding

- ❖ *Element(pathid, left, right, ...)*, *Path(pathid, path), ...*
 - *path* is a label path starting from the root
 - Why are *left* and *right* still needed? To preserve structure

Element				Path	
pathid	left	right	...	pathid	path
1	1	999	...	1	/bibliography
2	2	21	...	2	/bibliography/book
3	3	9	...	3	/bibliography/book/title
4	5	8	...	4	/bibliography/book/author
5	9	11
6	12	14
...

Label-path encoding: queries

15

- ❖ Simple path expressions with no conditions
 - `//book//title`
 - Perform string matching on *Path*
 - Join qualified *pathid*'s with *Element*
- ❖ `//book[publisher='Prentice Hall']/title`
 - Evaluate `//book/title`
 - Evaluate `//book/publisher[text()='Prentice Hall']`
 - How to ensure title and publisher belong to the same book?
 - ☞ Path expression with attached conditions needs to be broken down, processed separately, and joined back

Another path-based mapping

16

Dewey-order encoding

- ❖ Each component of the id represents the order of the child within its parent
 - Unlike label-path, this encoding is "lossless"



Dewey-order encoding: queries

17

- ❖ Examples:
 - `//title`
 - `//section/title`
 - `//book//title`
 - `//book[publisher='Prentice Hall']/title`
 - Works similarly as interval-based mapping
 - Except parent/child and ancestor/descendant relationship are checked by prefix matching
 - Serves a different purpose from label-path encoding
 - Any advantage over interval-based mapping?

Schema-aware mapping

18

- ❖ Idea: use DTD to design a better schema
- ❖ Basic approach: elements of the same type go into one table
 - Tag name → table name
 - Attributes → columns
 - If one exists, ID attribute → key column; otherwise, need to "invent" a key
 - IDREF attribute → foreign key column
 - Children of the element → foreign key columns
 - Ordering of columns encodes ordering of children

```

<!DOCTYPE bibliography [
  <ELEMENT book (title, _)>
  <!ATTLIST book ISBN ID REQUIRED>
  <!ATTLIST book price CDATA FINLIZED>
  <ELEMENT title (PCDATA)>
]>
book(ISBN, price, title_id, ...)
title(id, PCDATA_id)
PCDATA(id, value)
    
```