

# Supporting XML in Relational Databases

CPS 196.3  
Introduction to Database Systems

# Approaches to supporting XML

- ❖ Text files
- ❖ Specialized XML DBMS
  - Lore (Stanford), Strudel (AT&T), eXist (open-source), Tamino/QuiP (Software AG), etc.
  - Still a long way to go
- ❖ Object-oriented DBMS
  - eXcelon (ObjectStore), ozone, etc.
  - Not as mature as relational DBMS
- ❖ Relational (and object-relational) DBMS
  - Middleware and/or object-relational extensions

# Mapping XML to relational

- ❖ Just store XML text in a CLOB (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?
  - Well-formed XML → generic relational schema for tree-structured data
  - Valid XML → special relational schema based on DTD

# Storing well-formed XML

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

# Mapping data

```
<bibliography>
<book ISBN="ISBN-10" price="80.00">
<title>Foundations of Databases</title>
<author>Abiteboul</author>
<author>Hull</author>
<author>Vianu</author>
<publisher>Addison Wesley</publisher>
<year>1995</year>
</book>
</bibliography>
```

Attribute	eid	attrName	attrValue
	e1	ISBN	ISBN-10
	e1	price	80

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

Element		ElementChild		
eid	tag	eid	pos	child
e0	bibliography	e0	1	e1
e1	book	e1	1	e2
e2	title	e1	2	e3
e3	author	e1	3	e4
e4	author	e1	4	e5
e5	author	e1	5	e6
e6	publisher	e1	6	e7
e7	year	e2	1	t0
		e3	1	t1
		e4	1	t2
		e5	1	t3
		e6	1	t4
		e7	1	t5

# Mapping queries

- ❖ `//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

## Another query mapping example

7

- ❖ //bibliography/book[author="Abiteboul"]/@price
  - SELECT a.attrValue
    - FROM Element e1, ElementChild c1,
    - Element e2, ElementChild c2,
    - Element e3,
    - Attribute a
  - WHERE e1.tag = 'bibliography'
  - AND e2.tag = 'book'
  - AND e3.tag = 'author'
  - AND a.attrName = 'price'
  - AND e1.eid = c1.eid AND c1.child = e2.eid
  - AND e2.eid = c2.eid AND c2.child = e3.eid
  - AND e2.eid = a.eid
  - AND e3.eid IN (SELECT eid FROM ElementChild, Text
    - WHERE child = tid
    - AND value = 'Abiteboul');

## Mapping //

8

- ❖ //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';

## Storing valid XML

9

- ❖ 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 #IMPLIED>
  <ELEMENT title (#PCDATA)>
]>
book(ISBN, price, title_id, ...)
title(id, PCDATA_id)
PCDATA(id, value)
```

## Handling \* and + in DTD

10

- ❖ What if an element can have any number of children?
- ❖ Example: Book can have multiple authors
  - *book(ISBN, price, title\_id, author\_id, publisher\_id, year\_id)?*
  - ☞ BCNF?
- ❖ Idea: create another table to track such relationships
  - *book(ISBN, price, title\_id, publisher\_id, year\_id)*
  - *book\_author(ISBN, author\_id)*
  - ☞ BCNF decomposition in action!
  - ☞ A further optimization: merge *book\_author* into *author*
- ❖ Need to add position information if ordering is important
  - *book\_author(ISBN, author\_pos, author\_id)*
  - ☞ How about *book*?

## Inlining

11

- ❖ An author element just has a PCDATA child
- ❖ Instead of using foreign keys
  - *book\_author(ISBN, author\_id)*
  - *author(id, PCDATA\_id)*
  - *PCDATA(id, value)*
- ❖ Why not just "inline" the string value inside *book*?
  - *book\_author(ISBN, author\_PCDATA\_value)*
  - *PCDATA* table no longer stores *author* values
- ❖ Pros and cons of inlining
  - Fewer joins!
  - May create "scattering": There is no longer any table containing all authors; author information is scattered across *book*, *article*, etc.

## More general inlining

12

- ❖ As long as we know the structure of an element and its number of children (and recursively for all children), we can inline this element where it appears
 

```
<book ISBN="...">
  <publisher>
    <name>...</name><address>...</address>
  </publisher>...
</book>
```
- ❖ With no inlining at all
  - book(ISBN, publisher\_id)*
  - publisher(id, name\_id, address\_id)*
  - name(id, PCDATA\_id)*
  - address(id, PCDATA\_id)*
- ❖ With inlining
  - book(ISBN,*
  - publisher\_name\_PCDATA\_value,*
  - publisher\_address\_PCDATA\_value)*

## Queries

13

- ❖ *book(ISBN, price, title, publisher, year),  
book\_author(ISBN, author), book\_section(ISBN, section\_id),  
section(id, title, text), section\_section(id, section\_pos, section\_id)*
- ❖ `//title`
  - `(SELECT title FROM book) UNION ALL  
(SELECT title FROM section);`
- ❖ `//section/title`
  - `SELECT title FROM section;`

These queries only work  
for the given DTD
- ❖ `//bibliography/book[author="Abiteboul"]/@price`
  - `SELECT price FROM book, book_author  
WHERE book.ISBN = book_author.ISBN AND author = 'Abiteboul';`
- ❖ `//book//title`
  - `(SELECT title FROM book) UNION ALL  
(SELECT title FROM section)`

## Comparison of approaches

14

- ❖ Generic relational schema
  - Flexible; no DTD needed
  - Queries are easy to formulate
    - Translation from XPath can be easily automated
  - Queries involve lots of join and are expensive
- ❖ DTD-based relational schema
  - Need to know DTD to design the relational schema
  - Query formulation requires knowing DTD and schema
  - Queries are more efficient