

XML-Relational Mapping

CompSci 316  
Introduction to Database Systems

---

---

---

---

---

---

---

---

Announcements (Tue. Nov. 5) <sup>2</sup>

- ❖ Homework #3
  - Gradiance part due today
  - Non-Gradiance part due Thursday
- ❖ Project milestone #2 due in 9 days

---

---

---

---

---

---

---

---

Approaches to XML processing <sup>3</sup>

- ❖ Text files (!)
- ❖ Specialized XML DBMS
  - Lore (Stanford), Strudel (AT&T), Timber (Michigan), MonetDB/XQuery (CWI, Netherlands), Tamino (Software AG), BaseX, eXist, Sedna, ...
  - Still some way to go
- ❖ Object-oriented DBMS
  - ObjectStore, ozone, ...
  - Not as mature as relational DBMS
- ❖ Relational (and object-relational) DBMS
  - Middleware and/or extensions
  - IBM DB2's pureXML, PostgreSQL's XML type/functions...

---

---

---

---

---

---

---

---

## Mapping XML to relational

4

- ❖ Store XML 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?
  - 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

5

- ❖ *Element(eid, tag)*
  - ❖ *Attribute(eid, attrName, attrValue)*
    - Attribute order does not matter
  - ❖ *ElementChild(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 *id*'s
- ☞ Need indexes for efficiency, e.g., *Element(tag)*, *Text(value)*

---

---

---

---

---

---

---

---

## Node/edge-based: example

6

```
<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**

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
e3	1	t1
e4	1	t2
e5	1	t3
e6	1	t4
e7	1	t5

---

---

---

---

---

---

---

---

## Node/edge-based: simple paths <sup>7</sup>

### ❖ //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 <sup>8</sup>

### ❖ //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 e3.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 <sup>9</sup>

### ❖ //book//title

- Requires SQL3 recursion
- WITH RECURSIVE 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

10

### ❖ *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

### ❖ *Text(left, right, level, value)*

- Key is

### ❖ *Attribute(left, attrName, attrValue)*

- Key is

---

---

---

---

---

---

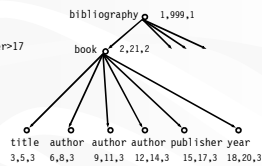
---

---

## Interval-based: example

11

```
1<bibliography>
2<book ISBN="ISBN-10" price="80.00">
3<title>4Foundations of Databases</title>5
6<author>7Abiteboul</author>8
9<author>10Hu1</author>11
12<author>13W1</author>14
15<publisher>16Addison Wesley</publisher>17
18<year>191995</year>20
</book>21_
</bibliography>999
```



### ☞ Where did *ElementChild* go?

- *E1* is the parent of *E2* iff:

---

---

---

---

---

---

---

---

## Interval-based: queries

12

### ❖ *//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

- ❖  $Element(pathid, left, right, \dots), Path(pathid, path), \dots$ 
  - $path$  is a label path starting from the root
  - Why are  $left$  and  $right$  still needed?

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

---

---

---

---

---

---

---

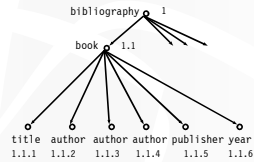
---

## 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”



*Element(dewey\_pid, tag)*

*Text(dewey\_pid, value)*

*Attribute(dewey\_pid, attrName, attrValue)*

---

---

---

---

---

---

---

---

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

---

---

---

---

---

---

---

---

## Handling \* and + in DTD

19

- ❖ 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)*

---

---

---

---

---

---

---

---

## Inlining

20

- ❖ An *author* element just has a PCDATA child
- ❖ Instead of using foreign keys
  - *book\_author(ISBN, author\_pos, author\_id)*
  - *author(id, PCDATA\_id)*
  - *PCDATA(id, value)*
- ❖ Why not just “inline” the string value inside *book*?
  - *book\_author(ISBN, author\_pos, author\_PCDATA\_value)*
  - *PCDATA* table no longer stores author values

---

---

---

---

---

---

---

---

## More general inlining

21

- ❖ 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
  - ❖ With inlining
- |   |  |
|---|--|
| <i>book(ISBN, publisher_id)</i>           | <i>book(ISBN,</i>                      |
| <i>publisher(id, name_id, address_id)</i> | <i>publisher_name_PCDATA_value,</i>    |
| <i>name(id, PCDATA_id)</i>                | <i>publisher_address_PCDATA_value)</i> |
| <i>address(id, PCDATA_id)</i>             |  |

---

---

---

---

---

---

---

---

## Queries

22

- ❖ *book*(ISBN, price, title, publisher, year),  
*book\_author*(ISBN, author\_pos, author),  
*book\_section*(ISBN, section\_pos, 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;
- ❖ //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)

These queries only work for the given DTD

---

---

---

---

---

---

---

---

## Pros and cons of inlining

23

- ❖ Not always applicable
  - \* and +, recursive schema (e.g., section)
- ❖
- ☞ Heuristic: do not inline elements that can be shared

---

---

---

---

---

---

---

---

## Result restructuring

24

- ❖ Simple results are fine
  - Each tuple returned by SQL gets converted to an element
- ❖ Simple grouping is fine (e.g., books with multiple authors)
  - Tuples can be returned by SQL in sorted order; adjacent tuples are grouped into an element
- ❖ Complex results are problematic (e.g., books with multiple authors and multiple references)
  - One SQL query returns one table, whose columns cannot store sets
  - Option 1: return one table with all combinations of authors and references → bad
  - Option 2: return two tables, one with authors and the other with references → join is done as post processing
  - Option 3: return one table with all author and reference columns; pad with NULL's; order determines grouping → messy

---

---

---

---

---

---

---

---



## Comparison of approaches

### ❖ Schema-oblivious

- Flexible and adaptable; no DTD needed
- Queries are easy to formulate
  - Translation can be easily automated
- Queries involve lots of join and are expensive

### ❖ Schema-aware

- Less flexible and adaptable
- Need to know DTD to design the relational schema
- Query formulation requires knowing DTD and schema
- Queries are more efficient
- XQuery is tougher to formulate because of result restructuring

---

---

---

---

---

---

---

---