

XML-Relational Mapping

Introduction to Databases
CompSci 316 Spring 2019



DUKE
COMPUTER SCIENCE

Announcements (Tue., Mar. 19)

- **Homework #3** due in 1 week 03/27 – next Wednesday
- **Project milestone #2** due in 1.5 week - next Friday 03/29 (extended by 3 days)
- Sudeepa's office hour 10-11 am, LSRC D325 for tomorrow 03/20

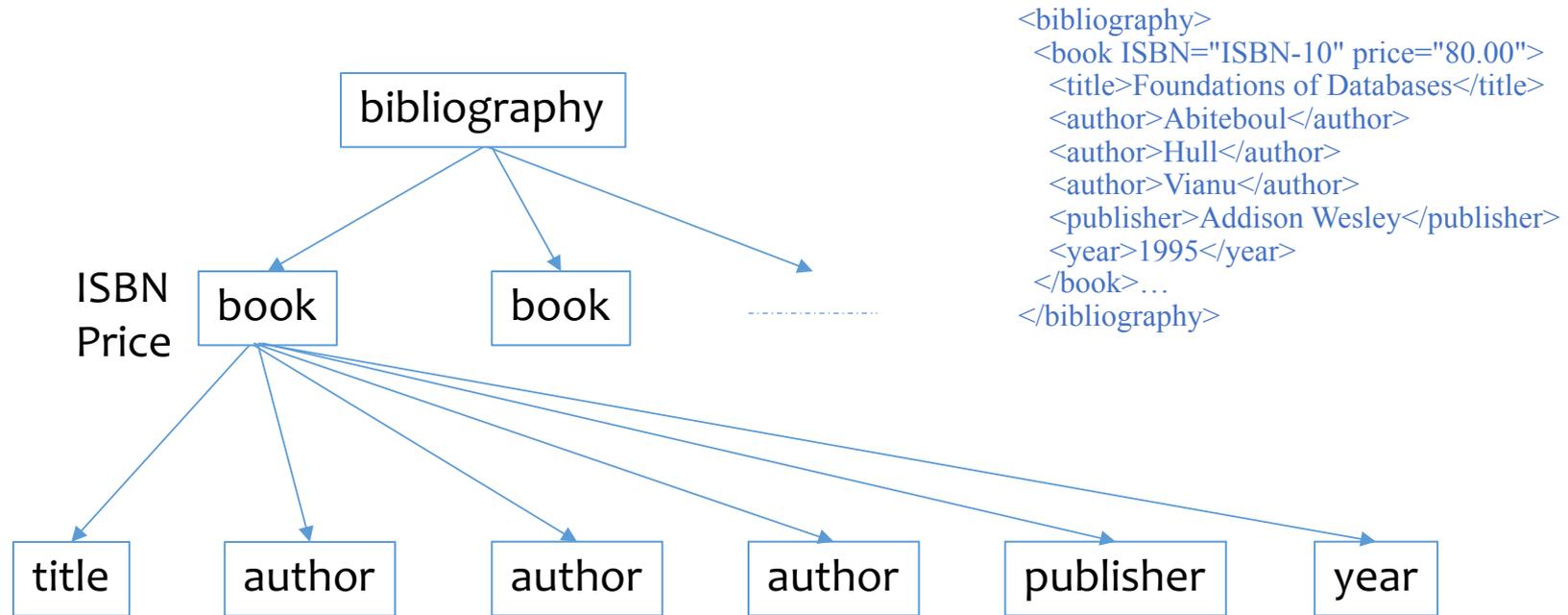
Approaches to XML processing

- Text files/messages
- Specialized XML DBMS
 - Tamino (Software AG), BaseX, eXist, Sedna, ...
 - 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

- Store XML in a column
 - Simple, compact
 - CLOB (Character Large Object) type + full-text indexing, or better, special XML type + functions
 - Poor integration with relational query processing
 - Updates are expensive
- Alternatives?
 - **Schema-oblivious mapping:** ← *Focus of this lecture*
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

Example – Node/Edge Based



- How would you translate it to a relational schema?
 - Element? Attribute? Parent-child relationship?
 - Keys?

Node/edge-based: schema

- *Element*(*eid*, *tag*)
 - *Attribute*(*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 *id*’s
- ☞ Need indexes for efficiency, e.g., *Element*(*tag*), *Text*(*value*)

Node/edge-based: example

```

<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

<i>eid</i>	<i>attrName</i>	<i>attrValue</i>
e1	ISBN	ISBN-10
e1	price	80

Text

<i>tid</i>	<i>value</i>
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Element

<i>eid</i>	<i>tag</i>
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

ElementChild

<i>eid</i>	<i>pos</i>	<i>child</i>
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

- `//title`

<i>eid</i>	<i>attrName</i>	<i>attrValue</i>
e1	ISBN	ISBN-10
e1	price	80

Attribute

- `//section/title`

<i>eid</i>	<i>tag</i>
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

Element

<i>eid</i>	<i>pos</i>	<i>child</i>
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

ElementChild

Text

<i>tid</i>	<i>value</i>
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

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

<i>eid</i>	<i>attrName</i>	<i>attrValue</i>
e1	ISBN	ISBN-10
e1	price	80

Attribute

<i>eid</i>	<i>tag</i>
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

Element

<i>eid</i>	<i>pos</i>	<i>child</i>
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

ElementChild

Text

<i>tid</i>	<i>value</i>
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Node/edge-based: complex paths

- `//bibliography/book[author="Abiteboul"]/@price`

<i>eid</i>	<i>attrName</i>	<i>attrValue</i>
e1	ISBN	ISBN-10
e1	price	80

Attribute

<i>eid</i>	<i>tag</i>
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

Element

<i>eid</i>	<i>pos</i>	<i>child</i>
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

ElementChild

Text

<i>tid</i>	<i>value</i>
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Node/edge-based: 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 e3.eid = c3.eid AND c3.child = t.tid
AND t.value = 'Abiteboul')

AND e2.eid = a.eid
AND a.attrName = 'price';

eid	attrName	attrValue
e1	ISBN	ISBN-10
e1	price	80

Attribute

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

Element

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

ElementChild

Text

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

Node/edge-based: descendent-or-self

- `//book//title`

<i>eid</i>	<i>attrName</i>	<i>attrValue</i>
e1	ISBN	ISBN-10
e1	price	80

Attribute

<i>eid</i>	<i>tag</i>
e0	bibliography
e1	book
e2	title
e3	author
e4	author
e5	author
e6	publisher
e7	year

Element

<i>eid</i>	<i>pos</i>	<i>child</i>
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

ElementChild

Text

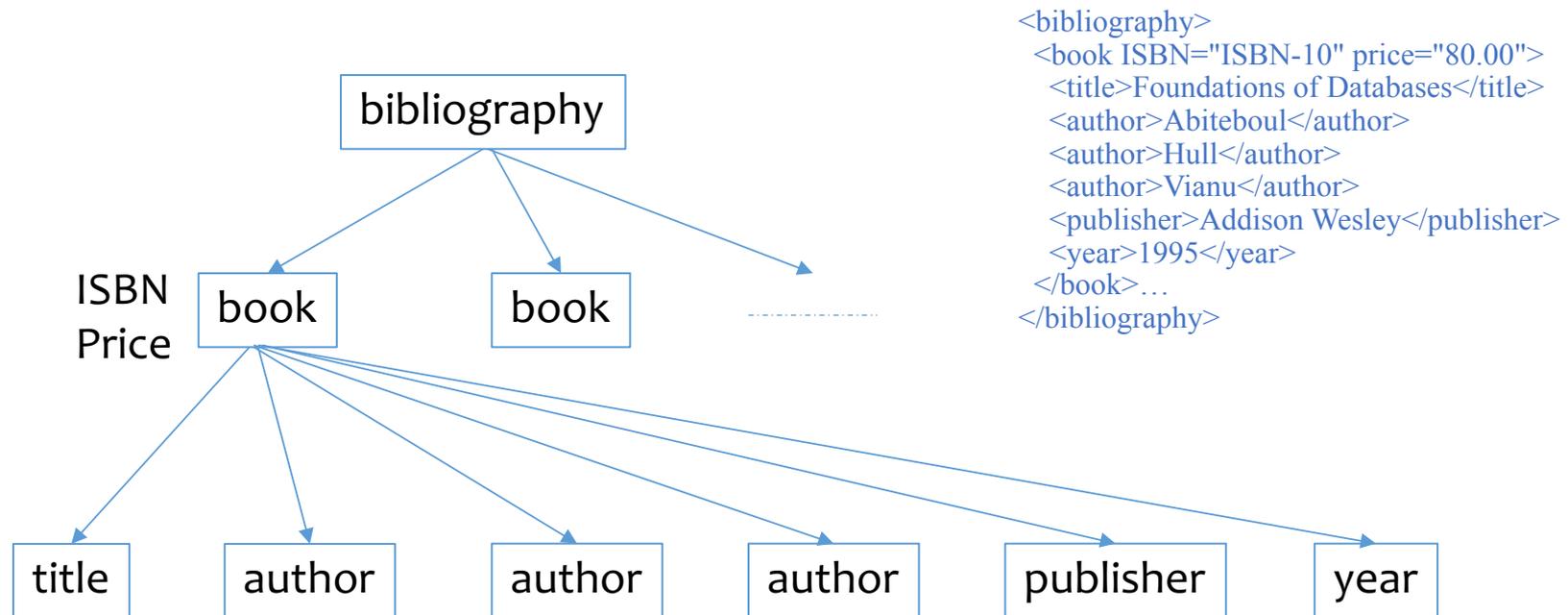
<i>tid</i>	<i>value</i>
t0	Foundations of Databases
t1	Abiteboul
t2	Hull
t3	Vianu
t4	Addison Wesley
t5	1995

Node/edge-based: descendent-or-self

- `//book//title`

- Requires SQL3 recursion
- WITH RECURSIVE `ReachableFromBook(id)` AS
((SELECT eid FROM Element WHERE tag = 'book')
UNION
(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';

Example – Interval Based



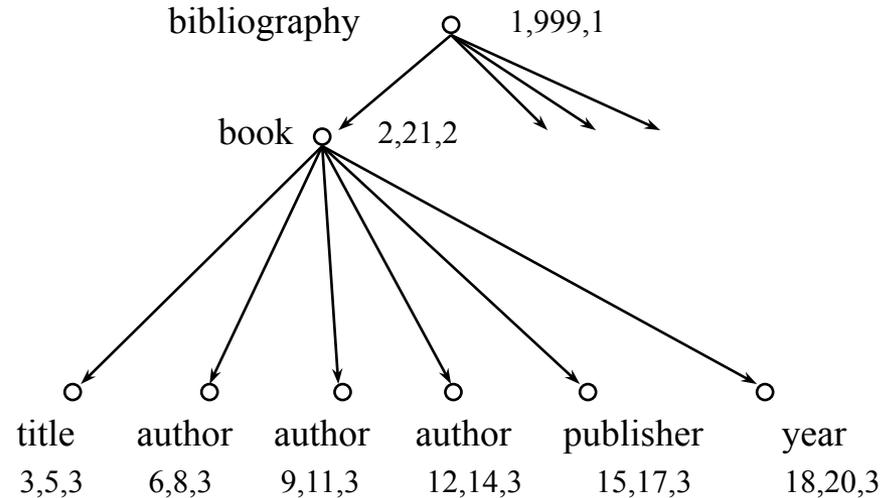
- How would you translate it to a relational schema?
 - Using intervals!

Interval-based: example

```

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

```



First two fields denote the interval.. see next slide

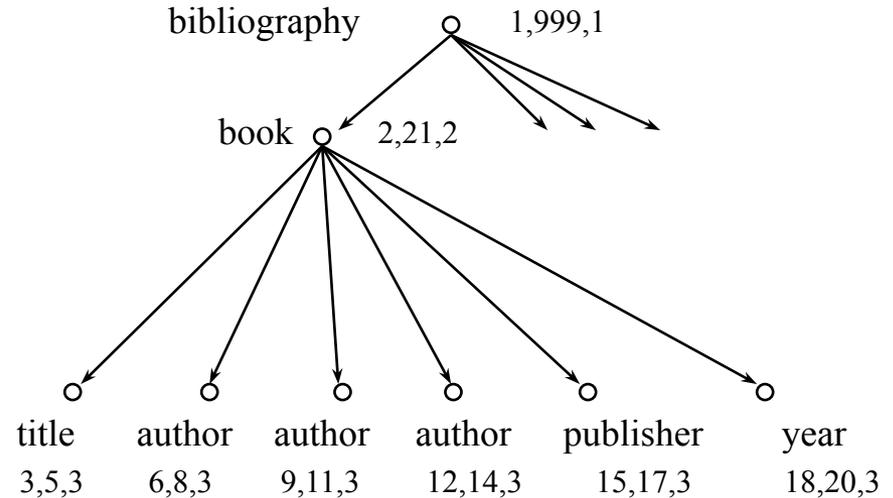
Interval-based: schema

- *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

```

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



☞ Where did *ElementChild* go?

- e_1 is the parent of e_2 iff:

$[e_1.left, e_1.right] \supset [e_2.left, e_2.right]$, and
 $e_1.level = e_2.level - 1$

Interval-based: queries

- `//section/title`

Element(left, right, level, tag)
Text(left, right, level, value)
Attribute(left, attrName, attrValue)

- `//book//title`

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;

Element(left, right, level, tag)
Text(left, right, level, value)
Attribute(left, attrName, attrValue)

☞ 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;

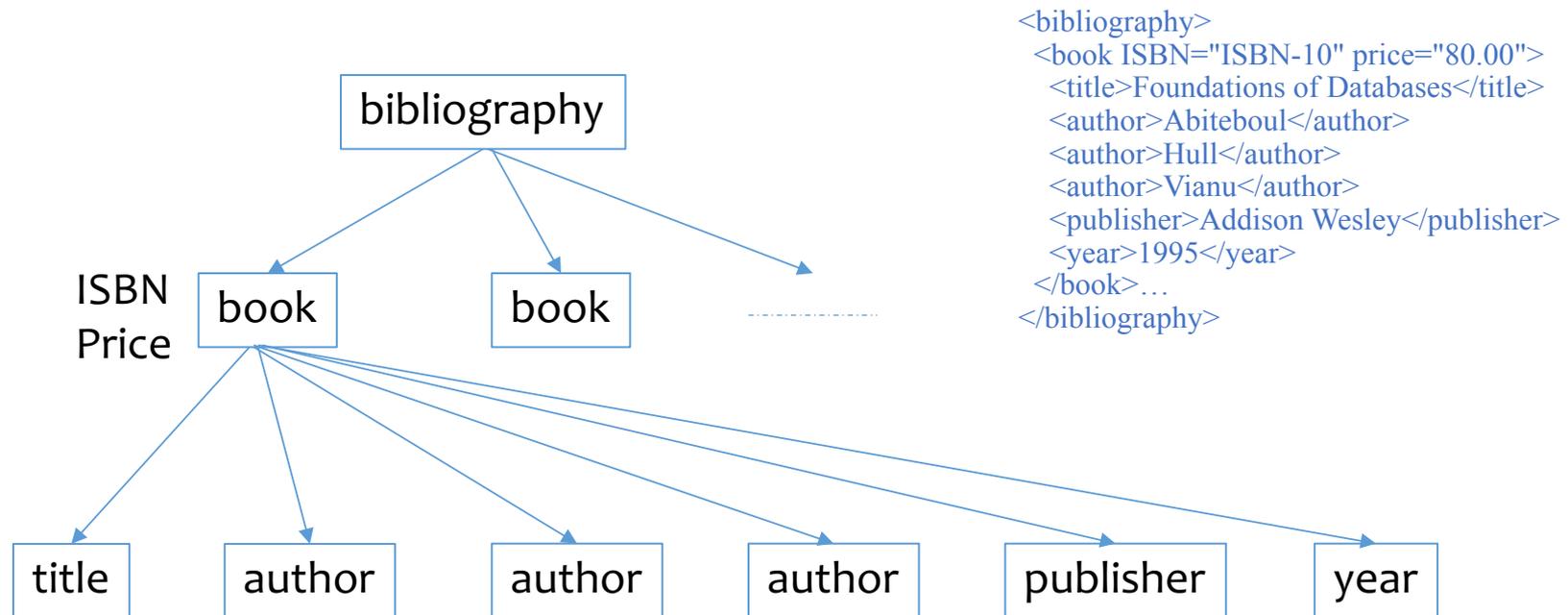
☞ No recursion!

Summary so far

Node/edge-based vs. interval-based mapping

- Path expression steps
 - Equality vs. containment join
- Descendent-or-self
 - Recursion required vs. not required

Example – Path Based



- How would you translate it to a relational schema?
 - Using paths from the roots!

Path-based mapping: approach 1

Label-path encoding: paths as strings of labels

- *Element*(*pathid*, *left*, *right*, ...), *Path*(*pathid*, *path*),

...

- *path* is a string containing the sequence of labels on a path starting from the root
- Why are *left* and *right* still needed?

Element

<i>pathid</i>	<i>left</i>	<i>right</i>	...
1	1	999	...
2	2	21	...
3	3	5	...
4	6	8	...
4	9	11	...
4	12	14	...
...

Path

<i>pathid</i>	<i>path</i>
1	/bibliography
2	/bibliography/book
3	/bibliography/book/title
4	/bibliography/book/author
...	...

Label-path encoding: queries

- `//book//title`
- `//book[publisher='Prentice Hall']/title`

Element

<i>pathid</i>	<i>left</i>	<i>right</i>	...
1	1	999	...
2	2	21	...
3	3	5	...
4	6	8	...
4	9	11	...
4	12	14	...
...

Path

<i>pathid</i>	<i>path</i>
1	/bibliography
2	/bibliography/book
3	/bibliography/book/title
4	/bibliography/book/author
...	...

Label-path encoding: queries

- Simple path expressions with no conditions

`//book//title`

- Perform string matching on *Path*
- Join qualified *pathid*'s with *Element*

- `//book[publisher='Prentice Hall']/title`

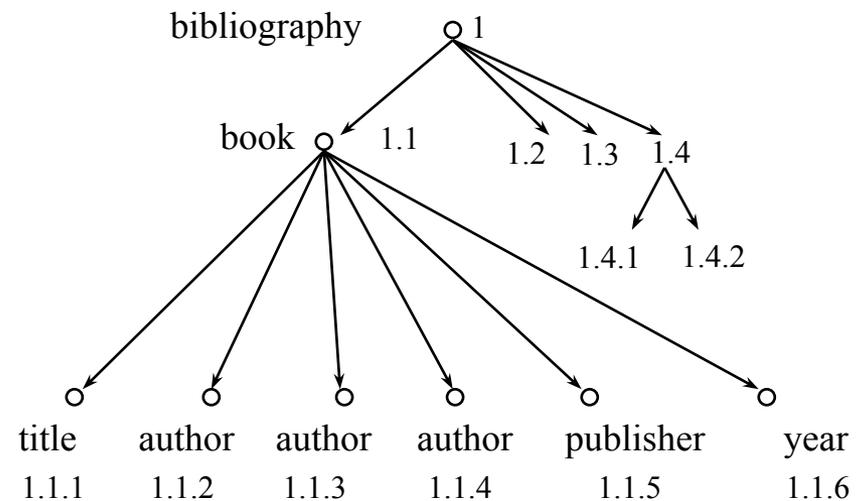
- Evaluate `//book/title`
- Evaluate `//book/publisher[text()='Prentice Hall']`
- Must then ensure title and publisher belong to the same book (how?)

☞ Path expression with attached conditions needs to be broken down, processed separately, and joined back

Path-based mapping: approach 2

Dewey-order encoding

- Each component of the id represents the order of the child within its parent



Element(dewey_pid, tag)

Text(dewey_pid, value)

Attribute(dewey_pid, attrName, attrValue)

Dewey-order encoding: queries

- 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

Summary

- XML data can be “shredded” into rows in a relational database
- XQueries can be translated into SQL queries
 - Queries can then benefit from smart relational indexing, optimization, and execution
- With schema-oblivious approaches, comprehensive XQuery-SQL translation can be easily automated
 - Different data mapping techniques lead to different styles of queries
- Schema-aware translation is also possible and potentially more efficient, but automation is more complex