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:
- *ElementChild*(*eid*, *pos*, *child*)
  - *pos* specifies the ordering of children
  - *child* references either *Element*(*eid*) or *Text*(*tid*)
  - Keys:
- *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>
```

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

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

Mapping queries

- `//title`
  - `SELECT eid FROM Element WHERE tag = 'title';`
- `//section/title`
  -

- Path expression becomes joins!
  - Number of joins is proportional to the length of the path expression

Another query mapping example

- `//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;
    ```

Mapping //

- `//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

- 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 CDATA #REQUIRED>
  <!ATTLIST book price CDATA #IMPLIED>
  <!ELEMENT title (#PCDATA)>…
]>
```

book(<u>*ISBN*</u>, *price*, *title_id*, …)
title(<u>*id*</u>, *PCDATA_id*)
PCDATA(<u>*id*</u>, *value*)

Handling * and + in DTD

- 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

  - ☞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

- 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
  -
  -

More general inlining

- 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

- *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`
  -
- `//bibliography/book[author="Abiteboul"]/@price`
  - `SELECT price FROM Book WHERE author = 'Abiteboul';`
- `//book//title`
  -

These queries only work for the given DTD

Comparison of approaches

- 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