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 $\rightarrow$ generic relational schema for tree-structured data
  - Valid XML $\rightarrow$ 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

<table>
<thead>
<tr>
<th>Bibliography</th>
<th>Element</th>
<th>ElementChild</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eid</td>
<td>Tag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISBN-10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price 80.00</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bibliography</td>
</tr>
<tr>
<td></td>
<td></td>
<td>title</td>
</tr>
<tr>
<td></td>
<td></td>
<td>author Abiteboul</td>
</tr>
<tr>
<td></td>
<td></td>
<td>author Hull</td>
</tr>
<tr>
<td></td>
<td></td>
<td>author Vianu</td>
</tr>
<tr>
<td></td>
<td></td>
<td>publisher Addison Wesley</td>
</tr>
<tr>
<td></td>
<td></td>
<td>year 1995</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributess</th>
<th>Element</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eid</td>
<td>attrName</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ISBN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>price</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Texts</th>
<th>ElementChild</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Eid pos child</td>
</tr>
<tr>
<td></td>
<td>40 1 41</td>
</tr>
<tr>
<td></td>
<td>41 1 42</td>
</tr>
<tr>
<td></td>
<td>42 1 43</td>
</tr>
<tr>
<td></td>
<td>43 1 44</td>
</tr>
<tr>
<td></td>
<td>44 1 45</td>
</tr>
<tr>
<td></td>
<td>45 1 46</td>
</tr>
<tr>
<td></td>
<td>46 1 47</td>
</tr>
<tr>
<td></td>
<td>47 1 48</td>
</tr>
<tr>
<td></td>
<td>48 1 49</td>
</tr>
<tr>
<td></td>
<td>49 1 50</td>
</tr>
<tr>
<td></td>
<td>50 1 51</td>
</tr>
<tr>
<td></td>
<td>51 1 52</td>
</tr>
<tr>
<td></td>
<td>52 1 53</td>
</tr>
<tr>
<td></td>
<td>53 1 54</td>
</tr>
<tr>
<td></td>
<td>54 1 55</td>
</tr>
<tr>
<td></td>
<td>55 1 56</td>
</tr>
<tr>
<td></td>
<td>56 1 57</td>
</tr>
<tr>
<td></td>
<td>57 1 58</td>
</tr>
</tbody>
</table>

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

- \(/\text{bibliography/book[@author='Abiteboul']}/\text{price}\)
  - 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 //

- \(/\text{book/title}\)
  - 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

```xml
<!DOCTYPE bibliography […]
<!ELEMENT book (title, …)
<!ATTLIST book ISBN CDATA #REQUIRED
<!ATTLIST book price CDATA #IMPLIED
<!ELEMENT title (#PCDATA)…>

book(ISBN, price, title_id, …)
title(id, PCDATA_id)
PCDATA(value)
Handling * and + in DTD

- What if an element can have any number of children?
- Example: Book can have multiple authors
- 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
  - How about book?

Inlining

- An author element just has a PCDATA child
- Instead of using foreign keys
  - 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
  - <publisher>
    - <name>…</name><address>…</address>
  - </publisher>
- With no inlining at all
  - 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),
  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
- //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