Announcements

- Homework #3 due next Friday (November 7)
- Course project milestone 2 due on November 12

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”)

- Alternatives?
  - Well-formed XML → generic relational schema for graph-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>
```

<table>
<thead>
<tr>
<th>Element</th>
<th>ElementChild</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>tag</td>
</tr>
<tr>
<td>11</td>
<td>isbn</td>
</tr>
<tr>
<td>12</td>
<td>price</td>
</tr>
<tr>
<td>13</td>
<td>title</td>
</tr>
<tr>
<td>14</td>
<td>author</td>
</tr>
<tr>
<td>15</td>
<td>author</td>
</tr>
<tr>
<td>16</td>
<td>author</td>
</tr>
<tr>
<td>17</td>
<td>publisher</td>
</tr>
<tr>
<td>18</td>
<td>year</td>
</tr>
<tr>
<td>19</td>
<td>pos</td>
</tr>
<tr>
<td>20</td>
<td>child</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>value</td>
</tr>
<tr>
<td>1</td>
<td>Foundations of Databases</td>
</tr>
<tr>
<td>2</td>
<td>Abiteboul</td>
</tr>
<tr>
<td>3</td>
<td>Hull</td>
</tr>
<tr>
<td>4</td>
<td>Vianu</td>
</tr>
<tr>
<td>5</td>
<td>Addison Wesley</td>
</tr>
<tr>
<td>6</td>
<td>1995</td>
</tr>
</tbody>
</table>
Mapping queries

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

- **//section/title**
  
  ```sql
  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

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

  ```sql
  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 e2.eid = c3.eid AND c3.child = t.tid
  AND t.value = 'Abiteboul')
  AND e2.eid = a.eid
  AND a.attrName = 'price';
  ```

Mapping //

- **//book/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 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

- 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
    - 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.

```xml
<book ISBN="...">...
<publisher>...
<name>...</name><address>...</address>
</publisher>...
</book>
```

- With no inlining at all

```sql
publisher(id, name_id, address_id)
name(id, PCDATA_id)
address(id, PCDATA_id)
```

- With inlining

```sql
book(ISBN, publisher_name_PCDATA_value, publisher_address_PCDATA_value)
```

Queries


- `/title` (SELECT title FROM book) UNION ALL (SELECT title FROM section);
- `//section/title` SELECT title FROM section;
- `/book/title` (SELECT title FROM book) UNION ALL (SELECT title FROM section)

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