Storing XML in a RDBMS

Approaches
1. Use generic db schema,
   [Florescu, Kossman 1999]
2. Use DTD to derive db schema
   [Shanmugasundaram, et al. 1999]
XML Storage: 1. Ternary Relation

[Florescu, Kossman 1999]

• Use generic relational schema for graphs (independent of the XML schema):

$$\textbf{Edge}(\text{source}, \text{label}, \text{dest})$$

$$\textbf{Leaf}(\text{node}, \text{value})$$
XML Storage: Ternary Relation

```xml
<paper year="1986">
    <title>calculus</title>
    <author>Bob</author>
    <author><first>Jane</first><last>Doe</last></author>
</paper>
```

### Table Edge

<table>
<thead>
<tr>
<th>Source</th>
<th>Label</th>
<th>Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;01</td>
<td>paper</td>
<td>&amp;02</td>
</tr>
<tr>
<td>&amp;02</td>
<td>title</td>
<td>&amp;03</td>
</tr>
<tr>
<td>&amp;02</td>
<td>author</td>
<td>&amp;04</td>
</tr>
<tr>
<td>&amp;02</td>
<td>author</td>
<td>&amp;05</td>
</tr>
<tr>
<td>&amp;02</td>
<td>@year</td>
<td>&amp;06</td>
</tr>
</tbody>
</table>

### Table Leaf

<table>
<thead>
<tr>
<th>Node</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;03</td>
<td>“Calculus”</td>
</tr>
<tr>
<td>&amp;04</td>
<td>“Bob”</td>
</tr>
<tr>
<td>&amp;07</td>
<td>“Jane”</td>
</tr>
<tr>
<td>&amp;08</td>
<td>“Doe”</td>
</tr>
<tr>
<td>&amp;06</td>
<td>1986</td>
</tr>
</tbody>
</table>
XML Storage: Ternary Relation

**Xpath to SQL translation:**

- **Xpath:**
  
  \[$root/paper[year/text()=“1986”]/author\]

- **SQL:**

  ```sql
  SELECT  e3.dest AS current
  FROM   Edge e1, e2, e3,   Leaf v
  WHERE  e1.source=\&01 and e1.label=“paper” and
  e1.dest=e2.source and e2.label=“year” and
  e2.dest=v.node and v.value=1986 and
  e1.dest=e3.source and e3.label=“author”
  ```
XML Storage: Ternary Relation

Xpath to Datalog translation:

- **Xpath:**
  
  \[
  \text{root/paper[year/text()}=\text{“1986”}]/\text{author}
  \]

- **Datalog**

  \[
  \text{path(Start,Current)}: \text{-edge(Start,paper,}X) ,
  \text{edge(}X,\text{year,}Y) , \text{leaf(}Y,1986) ,
  \text{edge(}X,\text{author,}Current).
  \]

  invoked as \text{path(\&01,Current)}
XML Storage: Ternary Relation

*In general*, if we’ve been able to translate an XPath expression $p$ into Datalog query $p(\text{Start}, X)$, then the expression $p/\text{elem}$, where $\text{elem}$ is some element name, would be translated into

\[
p/\text{elem}(\text{Start}, \text{Current}) \leftarrow p(\text{Start}, X), \text{edge}(X, \text{elem}, \text{Current})
\]

Similarly, we can systematically consider other XPath constructs, and show how to recursively translate

- `@attrib`
- `*`
- `..` (parent)
- `//` (descendant)

To translate into SQL, convert the Datalog to SQL queries. (Inefficient.)
XML Storage: Ternary Relation

Xpath to Datalog translation:

- try to consider systematically all the XPath constructors, to see if you can find a way to say them in SQL

  - /yr

    \[
    \text{path/"yr"'(Start,Current) :- path(Start,X), edge(X,\text{yr},Current)}
    \]

  - @price

    \[
    \text{path/"@price"'(Start,Current) :- path(Start,X), edge(X,\text{@price},Current)}
    \]

  - *

    \[
    \text{path/""'(Start,Current) :- path(start,X), edge(X,\_ ,Current)}
    \]
XML Storage: Ternary Relation

Xpath to Datalog translation:

- try to consider systematically all the XPath constructors, to see if you can find a way to say them in Datalog (excercise for you)

  - path/'..'(Start,Current) :- path(Start,X),

  - //yr

    define edgechain( ) so that
    path//'yr'(Start,Current) :- path(Start,X), edgeChain(X, yr, Current)
2. XML Storage: DTD to DB Schema

(fixed mapping 1)

[Shanmugasundaram’ 99]

• DTD:

```xml
<!ELEMENT paper (title, author*, year?)>
<!ELEMENT author (firstName, lastName)>
```

Given

• Possible relational schema:

```
PaperNodes(pNodeId, title, aNodeId, year)
AuthorNodes(aNodeId, firstName, lastName)
```

How is space wasted in this representation?
2. XML Storage: DTD to DB Schema

(fixed mapping 2)

[Shanmugasundaram’ 99]

• DTD:

Given

<!ELEMENT paper (title, author*, year?)>
<!ELEMENT author (firstName, lastName)>

• Another relational schema:

PaperNodes(pNodeId, title, firstName, lastName, year)

What is bad about this representation?
2. XML Storage: DTD to DB Schema
(fixed mapping 3)

- **DTD:**

```xml
given
<!ELEMENT paper (title, author*, year?)>
<!ELEMENT author (firstName, lastName)>
```

- **Third relational schema:**

```sql
PaperNd(pNodeId, title, year)
AuthorNd(aNodeId, firstName, lastName, parent_id)
parent_id references PaperNd
```
Example
DTD to DB : example document mapping 3

<!ELEMENT paper (title, author*, year?)>
<!ELEMENT author (firstName, lastName)>

PaperNode( pnid, title, year)
(&02,"Calculus",2010)
(&021,"Nausea", 1938)
...

AuthorNode(anid, firstName, lastName, parent_pnid)
(&05, “Joe” , “Cool” , &02)
(&041, “Joe” , “Cool”, &021)
(&051, “Al” , “Camus”, &021)

Note how this avoids update anomaly!
XML Storage: DTD to DB Schema

Xpath to SQL translation

• Xpath:

  /paper[year/text()="1986"]/author

• SQL:

  select a.aid
  
  from PaperNode p, AuthorNode a

  where p.year=1986 and p.pid=a.parent_pid
General algorithm when DTD is "flat" (no nesting in parentheses)

So the only constructors are * + ? and ,

(1). Treat + like *, and ignore ?

*Result*: you can still store old document under new DTD
DTD to DB Schema: Finding mapping(2)

(2) Create DTD graph
   - nodes: elements, attributes
   - edge(A,B) if <! element A (...B...)>
   - label edge by * if <! element A (...B*...)> 

(3) Use graph to decide for which nodes to make tables.
Suppose no * edges:
   - Simplest: every node; with columns “pointing to” 1-children/values
     table Paper(pid,tid,yid,aid)
     table Title(tid,value)
     table Year(yid,value)
     table Editor(eid,fnid,lnid) ...
   - But tables for Title, Year, ... are wasted! So “in-line” them:
     table Paper(pid,title,year,aid)
     table Editor(eid,FirstN,LastN)
   - In fact, do this repeatedly
     table Paper(pid,title,year,editor_FirstN,editor_LastN)
     so there are columns only for atomic-valued leaves (saves joins!)
DTD to DB Schema: Finding mapping (3)

- What if the graph is not a tree? Currently, in-lining would construct:

  table `Paper(pid, Title, Year, Editor_FirstN, Editor_LastN)`
  table `Thesis(tid, Editor_FirstN, Editor_LastN, Advsr)`

- Problem: *splits editors into several tables, making it harder to find //editor*

- Improved rule: create separate table for nodes of in-degree greater than 1

  table `Paper(pid, Title, Year)`
  table `Thesis(tid, Advisor)`
  table `Editor (aid, FirstN, LastN)`

  But how can you tell which author goes with which paper/thesis? Add *parent-link*, pointing to either

  table `Editor (aid, FirstN, LastN, parentID)`
DTD to DB Schema: Finding mapping (3)

- **What if we have a node with * edge to it?** Cannot in-line since we don’t know how many repetitions needed. If we repeat the paper row for each author, we get redundancy.

- So create separate Author table.

  ```sql
  table Paper(pid, Title, Year)
  table Author(aid, FirstN, LastN, parentID)
  ```
XML to DB using DTD: summary

1. Create graph
2. Mark the following as ‘base nodes’
   • in-degree 0 or 2+
   • child of * edge
3. Create tables for base nodes, each with id column to be filled by system generated identifiers
4. Create columns for leafs (name it with sequence of tags) and add them to the tables of lowest base node ancestor
5. If a base node is a child of some other node, then add column parentID, to be filled with foreign key(s)