Overview

- Motivation
- Input and Output Models
- Proposed Techniques
  - Flat Translation (FT)
  - Nesting-based Translation (NeT)
  - Translation using Inclusion Dependency (NeT<sup>C</sup>)
- Conclusion
**Motivation**

- Many database products support conversion from RDB to XML documents
  - IBM DB2 XML Extender, XML-DBMS, Oracle9i, ...
- Given a relational schema $R$ and XML schema $X$, generate XML documents conforming to $X$

![Diagram showing R and X connected by mapping rules]

**Motivation (contd.)**

- Users have to provide a target XML schema $X$ first
- Then, provide mapping rules from $R$ to $X$
- Problematic when dealing with
  - Large relational schema
  - Old relational schema
- Goal: translate a relational schema $R$ to a “GOOD” XML schema $X$ so that users can use $X$ for other purposes
Input and Output Model

- Concise and precise notations for input and output
- Dozen XML schema language proposals
  - Proposals from W3C: DTD, XML-Schema
  - Proposals related to ISO/OASIS: RELAX, TREX, RELAX+TREX=?
  - Industry: XDR (Microsoft), DSD (AT&T), SOX (CommerceOne), ...
  - Misc.: Schematron, DDML, ...
- Different focus, different expressive power

Input and Output Model (contd.)

- Use a formalism instead of one schema language proposal
  - Borrow structure from DTD and RELAX
  - Borrow type from XML-Schema
  - Borrow constraints from XML-Schema and Schematron
  - Borrow notations from [Fan & Simeon; PODS 00]

- In the talk, however, I will use DTD as output notation for simplicity
Flat Translation (FT)

- **R -> X**
  - Table of R to element of X
  - Column of R to
    - Attribute of X (in attribute-oriented mode)
    - Element of X (in element-oriented mode)

- Simple and straightforward translation
- But relational model is “flat” while XML model is “hierarchical”
- Need to take advantage of such hierarchical feature to capture the original schema more accurately
  - XML model provides RE (?, *, +, |)

FT Example: R (K, A, B, C)

**Element-oriented**

```xml
<!ELEMENT R (A, B, C)>
<!ATTLIST R  K ID #REQUIRED>
<!ELEMENT A (#PCDATA)>
<!ELEMENT B (#PCDATA)>
<!ELEMENT C (#PCDATA)>

<R K="1">
  <A/> <B/> <C/>
</R>
<R K="2">
  <A/> <B/> <C/>
</R>
<R K="3">
  <A/> <B/> <C/>
</R>
```

**Attribute-oriented**

```xml
<!ELEMENT R (EMPTY)>
<!ATTLIST R K ID #REQUIRED>
<!ELEMENT A CDATA>
<!ELEMENT B CDATA>
<!ELEMENT C CDATA>

<R K="1" A="…” B="…” C="…”/>
<R K="2" A="…” B="…” C="…”/>
<R K="3" A="…” B="…” C="…”/>
```
Problems?

ER Model \rightarrow Relational Model \rightarrow XML Model

ER Model \rightarrow Relational Model 

Relational Model \rightarrow XML Model

Nesting

- Nested relational model: allows non-1NF
- Nest operator: nesting on column $X$ collects all tuples that agree on the remaining columns into a set
Nesting (contd.)

\[ \text{nest}_A(t) \]

\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
1 & a & 10 \\
1 & a & 20 \\
2 & a & 10 \\
3 & a & 10 \\
4 & b & 10 \\
4 & b & 20 \\
5 & b & 20 \\
\hline
\end{array}
\]

\[ \text{nest}_C(t) \]

\[
\begin{array}{|c|c|c|}
\hline
A+ & B & C \\
\hline
\{1,2,3\} & a & 10 \\
1 & a & 20 \\
4 & b & 10 \\
\{4,5\} & b & 20 \\
\hline
\end{array}
\]

\[ \text{nest}_A(nest_C(t)) \]

\[
\begin{array}{|c|c|c|}
\hline
A & B & C+ \\
\hline
1 & a & \{10,20\} \\
2 & a & 10 \\
3 & a & 10 \\
4 & b & \{10,20\} \\
5 & b & 20 \\
\hline
\end{array}
\]

Nesting (contd.)

\[ \text{nest}_C(t) \]

\[
\begin{array}{|c|c|c|}
\hline
A & B & C+ \\
\hline
1 & a & \{10,20\} \\
2 & a & 10 \\
3 & a & 10 \\
4 & b & \{10,20\} \\
5 & b & 20 \\
\hline
\end{array}
\]

\[ \text{nest}_A(nest_C(t)) \]

\[
\begin{array}{|c|c|c|}
\hline
A+ & B & C+ \\
\hline
1 & a & \{10,20\} \\
\{2,3\} & a & 10 \\
4 & b & \{10,20\} \\
5 & b & 20 \\
\hline
\end{array}
\]
Nesting (contd.)

- Some properties [Jaeschke & Schek; PODS 82]
  - nest_a(nest_b(t)) ≠ nest_b(nest_a(t))
  - nest_a(nest_b(t)) = nest_b(t)
- Functional Dependencies are preserved against nesting [Fischer et al; JCSS 85]
- Applying nest operator on a non-prime column X yields no changes

- For a table t with n columns and m prime columns (m ≤ n), max # T of necessary nesting is:
  \[ T = m + m(m-1) + \ldots + m(m-1)\ldots(2)(1) \]

Nesting-based Translation (NeT)

- R -> X
  - For each table t in R, apply nesting repeatedly until no nesting succeeds.
  - If no nesting succeeds, do FT
  - Otherwise, for each column c where nesting was succeeded, convert c to
    - c* if c was nullable
    - c+ if c was not nullable
NeT Example

- R (A,B,C)
- Nesting was succeeded on columns B & C

FT: `<!ELEMENT R (A,B,C)>`
NeT: `<!ELEMENT R (A,B*,C*)>`

NeT Example (contd.)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>a</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>a</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>b</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>b</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>

FT: R (A,B,C)
- `<t> <A>1</A> <B>a</B> <C>10</C> </t>`
- `<t> <A>1</A> <B>a</B> <C>20</C> </t>`
- `<t> <A>2</A> <B>a</B> <C>10</C> </t>`
- `<t> <A>3</A> <B>a</B> <C>10</C> </t>`
- `<t> <A>4</A> <B>b</B> <C>10</C> </t>`
- `<t> <A>4</A> <B>b</B> <C>20</C> </t>`

NeT: R (A*,B,C*)
- `<t> <A>1</A> <B>a</B> <C>10</C> <C>20</C> </t>`
- `<t> <A>2</A> <A>3</A> <B>a</B> <C>10</C> </t>`
- `<t> <A>4</A> <B>b</B> <C>10</C> <C>20</C> </t>`
- `<t> <A>4</A> <B>b</B> <C>20</C> </t>`
- `<t> <A>5</A> <B>b</B> <C>20</C> </t>`
Translation using INDs (NeTC)

- NeT
  - Is only applicable for a single table at a time
  - Cannot draw a big picture when multiple tables exist in a schema
- NeTC
  - Uses inclusion dependencies to derive a more intuitive schema
  - For tables s and t with columns X and Y, and an IND s[A] ⊆ t[B],

\[ \text{Pull-up}(s, t) \]

- If A is a super key, there is an 1:1 relationship btw. s and t.
  \[ <\text{ELEMENT } t (Y, s)> \]
- If A is not a super key, there is an n:1 relationship from s to t.
  \[ <\text{ELEMENT } t (Y, s*)> \]
  \[ <\text{ELEMENT } s (X-A)> \]

NeTC Example

student  (Sname, Course, Advisor, Gender)
professor (Pname, Age)

FT

\[ <\text{ELEMENT student } (Sname, Course, Advisor, Gender)> \]
\[ <\text{ELEMENT professor } (Pname, Age)> \]

NeTC

\[ <\text{ELEMENT student } (Sname, Course, Gender)> \]
\[ <\text{ELEMENT professor } (Pname, Age, student*)> \]
General NeT$^C$ Algorithm

- Given a set of INDs, creates a digraph s.t. there is an edge $i \rightarrow j$ for every IND $j [...] \subseteq i [...]$
- Identify top nodes
  - Source nodes
  - Node with highest outdegree among mutually recursive nodes
- For each top node $t$, do BFS until all nodes are visited. For each node $v \rightarrow w$, pull-up($w,v$)

Complex NeT$^C$ Example

**Tables**
- student ($Sid$, Name, Advisor)
- emp ($Eid$, Name, ProjName)
- prof ($Eid$, Name, Teach)
- course ($Cid$, Title, Room)
- dept ($Dno$, Mgr)
- proj ($Pname$, Pmgr)

**INDs**
- student[Advisor] $\subseteq$ prof[Name]
- emp[ProjName] $\subseteq$ proj[Pname]
- prof[Teach] $\subseteq$ course[Cid]
- prof[Eid,Name] $\subseteq$ emp[Eid,Name]
- dept[Mgr] $\subseteq$ emp[Eid]
- proj[Pmgr] $\subseteq$ emp[Eid]
INDs Graph

WebDB’01

Complex NeT^C Example (contd.)

```xml
<!ELEMENT course (Cid,Title,Room,prof*)>
<!ELEMENT prof (Name,student*)>
<!ATTLIST prof Eid ID>
<!ELEMENT student (Sid,Name)>
<!ELEMENT emp (Eid,Name,ProjName,dept*,proj*)>
<!ATTLIST emp Ref_prof IDREF>
<!ELEMENT dept (Dno)>  
<!ELEMENT proj (Pname)>
```

WebDB’01
Experimental Results

- Compared output with DB2XML v1.3
- Used MS Access NorthWind sample database

**DB2XML**

```xml
<!ELEMENT Orders (CustmerID,EmployeeID,ShipVia,ShipAddress,ShipCity,ShipCountry,ShipPostalCode)>  
<!ELEMENT CustomerID (#PCDATA)>  
<!ATTLIST CustomerID ISNULL (true|false) #IMPLIED>  
...  
<!ELEMENT ShipPostalCode (#PCDATA)>  
<!ATTLIST ShipPostalCode ISNULL (true|false) #IMPLIED>
```

**Net**

```xml
<!ELEMENT Orders (EmployeeID+,ShipVia*)>  
<!ATTLIST Orders CustomerID CDATA #REQUIRED  
ShipAddress CDATA #IMPLIED  
ShipCity CDATA #IMPLIED  
ShipCountry CDATA #IMPLIED  
ShipPostalCode CDATA #IMPLIED>  
<!ELEMENT EmployeeID (#PCDATA)>  
<!ELEMENT ShipVia (#PCDATA)>  
```

Related Work

- Conversion * - > DTD
  - XML Documents - > DTD: STORED [SIGMOD98], XTRACT [SIGMOD 00]  
  - View Description - > DTD: MIX [ICDE 99]  
  - Relational Schema - > DTD: XML-DBMS, DB2 Extender, Oracle 9i iFS, Informix ObjectTranslator, ...

- Conversion DTD - > R
  - XRel [TOIT 01], Inlining [VLDB 99], ...
  - XML-DBMS, ...

- SilkRoute, XPERANTO, ...
Conclusion

- Converting “flat” relational schema to “hierarchical” XML schema in 1-to-1 manner is not good

- NeT can find the hidden semantics from data and use them to generate more intuitive XML schema

- Improvements are ongoing
  - Nesting on multiple attributes
  - Information capacity aspects