Bridging Relational Technology and XML

Jayavel Shanmugasundaram
Cornell University
(Work done at University of Wisconsin & IBM Almaden Research Center)

Business to Business Interactions

Tires R Us
Cars R Us

Internet

Order Fulfillment Application

Purchasing Application

Relational Database System

Relational Database System

Shift in Application Developers’ Conceptual Data Model

XML

XML application development

Code to convert XML data to relational data

Relational data manipulation

XML application development

Code to convert relational data to XML data

Why use Relational Database Systems?

• Highly reliable, scalable, optimized for performance, advanced functionality
  – Result of 30+ years of Research & Development
  – XML database systems are not “industrial strength” … and not expected to be in the foreseeable future

• Existing data and applications
  – XML applications have to inter-operate with existing relational data and applications
  – Not enough incentive to move all existing business applications to XML database systems

• Remember object-oriented database systems?

Are XML Database Systems the Answer?

Tires R Us
Cars R Us

Internet

Order Fulfillment Application

Purchasing Application

XML Database System

XML Database System

A Solution

Tires R Us
Cars R Us

Internet

Order Fulfillment Application

Purchasing Application

XML Translation Layer

Relational Database System

XML Translation Layer

Relational Database System
XML Translation Layer
(Contributions)

- Store and query XML documents
  - Harnesses relational database technology for this purpose [VLDB’99]
- Publish existing relational data as XML documents
  - Allows relational data to be viewed in XML terms [VLDB’00]

Outline

- Motivation & High-level Solution
- Background (Relations, XML)
- Storing and Querying XML Documents
- Publishing Relational Data as XML Documents
- Conclusion

SQL Query

Find all the items bought by “Cars R Us” in the year 1999

Select it.name
From PurchaseOrder po, Item it
Where po.customer = “Cars R Us” and
  po.year = 1999 and
  po.id = it.pid

XML Document

<PurchaseOrder id="2001", customer="Cars R Us">

<item name="Firestone Tire" cost="2000.00">
  <Quantity> 50 </Quantity>
</item>

<item name="Good year Tire" cost="8000.00">
  <Quantity> 200 </Quantity>
</item>

<Payment> 40% </Payment>
<Payment> 60% </Payment>
</PurchaseOrder>
XML Document

Self-describing tags

Nested structure

<PurchaseOrder id="2001" customer="Cars R Us">
  <Date>
    <Day> 10 </Day>
    <Month> June </Month>
    <Year> 1999 </Year>
  </Date>
  <Item name="Firestone Tire" cost="2000.00">
    <Quantity> 50 </Quantity>
  </Item>
  <Item name="Goodyear Tire" cost="8000.00">
    <Quantity> 200 </Quantity>
  </Item>
  <Payment> 40% </Payment>
  <Payment> 60% </Payment>
</PurchaseOrder>

XML Schema

PurchaseOrder — PurchaseOrder id=[integer] customer=[string]?

Date — Date
  — Day
  — Month: Month
  — Year

Item — Item name=[string] cost=[float]
  — Quantity

Order — Payment
  — 40%
  — 60%

… and so on

XML Schema (contd.)

PurchaseOrder — PurchaseOrder id=[integer] customer=[string]?

Date — Date
  — Day
  — Month: Month
  — Year

Item — Item name=[string] cost=[float]
  — Quantity

XML Query

Find all the items bought by "Cars R Us" in 1999

For $spo in /PurchaseOrder
Where $spo/@customer = "Cars R Us" and $spo/@year = 1999
Return $spo/Item
Outline

• Motivation & High-level Solution
• Background (Relations, XML)
• Storing and Querying XML Documents
• Publishing Relational Data as XML Documents
• Conclusion

Outline

• Motivation & High-level Solution
• Background (Relations, XML)
• Storing and Querying XML Documents
  – Relational Schema Design and XML Storage
  – Query Mapping and Result Construction
• Publishing Relational Data as XML Documents
• Conclusion

Desired Properties of Generated Relational Schema $\mathcal{R}$

• All XML documents conforming to XML schema should be “mappable” to tuples in $\mathcal{R}$
• All queries over XML documents should be “mappable” to SQL queries over $\mathcal{R}$
• Not Required: Ability to re-generate XML schema from $\mathcal{R}$

XML Query (contd.)

//Item

//Item[5]

//Item Before //Payment

//Item/(Item/Payment)*/(Payment|Item)*/Date

Storing and Querying XML Documents
[Shanmugasundaram et. al., VLDB '99]

Outline

• Motivation & High-level Solution
• Background (Relations, XML)
• Storing and Querying XML Documents
• Publishing Relational Data as XML Documents
• Conclusion

XML Schema

PurchaseOrder  <PurchaseOrder id={integer} customer={string}>
  (Date | Payment)* (Item (Item Item)* Payment)*
</PurchaseOrder>
Simplifying XML Schemas

- XML schemas can be “simplified” for translation purposes
- All without undermining storage and query functionality!

Why is Simplification Possible?

- Structure in XML schemas can be captured:
  - Partly in relational schema
  - Partly as data values
- Order field to capture order among siblings
- Sufficient to answer ordered XML queries
  - `PurchaseOrder/Item` AFTER `PurchaseOrder/Payment`
  - Sufficient to reconstruct XML document

Simplification Desiderata

- Simplify structure, but preserve differences that matter in relational model:
  - Single occurrence (attribute)
  - Zero or one occurrences (nullable attribute)
  - Zero or more occurrences (relation)

Translation Normal Form

- An XML schema production is either of the form:
  
  \[ P \rightarrow \langle P \text{ attr}=\langle \text{type} \rangle \ldots \text{attr}=\langle \text{type} \rangle \rangle \ldots \text{attr}=\langle \text{type} \rangle \rangle \ldots \text{attr}=\langle \text{type} \rangle \rangle \langle \text{type} \rangle \langle /P \rangle \]
  
  where \( a_i \neq a_j \)
  
  - \ldots or of the form:
  
  \[ P \rightarrow \langle P \rangle \]  

Example Simplification Rules

\[
\begin{align*}
(e_1 \parallel e_2) & \rightarrow e_1 ? e_2 ? \\
(Date | (Payment)* (Item Item)* Payment)* & \rightarrow \langle Date \rangle \langle Item \rangle \langle Payment \rangle \\
Date ? (Item)* (Item Item)* Payment)* & \rightarrow \langle Date \rangle \\
Date ? (Payment)* (Item Item)* Payment)* & \rightarrow \langle Date \rangle \\
\end{align*}
\]

Simplified XML Schema

\[
\begin{align*}
\text{PurchaseOrder} & \rightarrow \langle \text{PurchaseOrder id}=\langle \text{integer} \rangle \text{ customer}=\langle \text{string} \rangle \rangle \\
\langle Date \rangle \langle Item \rangle \langle Payment \rangle & \rightarrow \langle Date \rangle \\
\langle Date \rangle & \rightarrow \langle Date \rangle \langle Day \rangle \langle Month \rangle \langle Year \rangle \\
\langle Day \rangle & \rightarrow \langle Day \rangle \langle integer \rangle \\
\langle Month \rangle & \rightarrow \langle Month \rangle \langle string \rangle \\
\langle Year \rangle & \rightarrow \langle Year \rangle \langle integer \rangle \\
\langle Item \rangle & \rightarrow \langle Item \rangle \langle item name=\langle \text{string} \rangle \rangle \langle cost=\langle \text{float} \rangle \rangle \langle \text{Quantity} \rangle \\
\end{align*}
\]

… and so on
### Outline

- Motivation & High-level Solution
- Background (Relations, XML)
- Storing and Querying XML Documents
  - Relational Schema Design and XML Storage
  - Query Mapping and Result Construction
- Publishing Relational Data as XML Documents
- Conclusion

### XML Query

Find all the items bought by “Cars R Us” in 1999

For $\text{ spo in /PurchaseOrder}$
Where $\text{ spo@customer = “Cars R Us”}$ and $\text{ spo@date@year = 1999}$
Return $\text{ spo@Item}$
Path Expression Automata (Moore Machines)

XML Schema Automaton (Mealy Machine)

Intersected Automaton

Generated SQL Query

Recursive XML Query

Recursive Automata Intersection
Recursive SQL Generation

ResultItems (id, name, quantity) as (  
Select it.id, it.name, it.quantity  
From PurchaseOrder po, Item it  
Where po.customer = "Cars R Us"  
and po.id = it.pid  
Union all  
Select it.id, it.name, it.quantity  
From ResultItems rit, PurchaseOrder po, Item it  
Where rit.id = po.pid and po.id = it.pid  
)

SQL Generation for Path Expressions (Completeness)

• (Almost) all path expressions can be translated to SQL
• SQL does not support
  – Nested recursion
  – Meta-data querying
• Meta-data query capability provided in the XML translation layer

Constructing XML Results

("Firestone Tire", 2000.00, 50)  
("Goodyear Tire", 8000.00, 200)

Complex XML Construction

Relational Schema and Data

Outline

• Motivation & High-level Solution
• Background (Relations, XML)
• Storing and Querying XML Documents
• Publishing Relational Data as XML Documents
• Conclusion
XML Document

```xml
<PurchaseOrder id="200I" customer="Cars R Us">
  <Date>
    <Day> 10 </Day>
    <Month> June </Month>
    <Year> 1999 </Year>
  </Date>
  <Item name="Firestone Tire" cost="2000.00">
    <Quantity> 50 </Quantity>
  </Item>
  <Item name="Goodyear Tire" cost="8000.00">
    <Quantity> 200 </Quantity>
  </Item>
</PurchaseOrder>
```

Naïve Approach

- Issue many SQL queries that mirror the structure of the XML document to be constructed
- Tag nested structures as they are produced

Late Tagging, Early Structuring

- **Structured** XML document content produced
  - In document order
  - “Sorted Outer Union” approach
- Tagger just adds tags
  - In constant space

Sorted Outer Union Approach

```
1 2 3
```

```
Sort 1
Union
```

```
200I null null null null null null null null 2
200I "Cars R Us", 10, June, 1999, null null null null null 1
200I null null null null null null null null 2
200I "Firestone Tire", 2000.00, 50 null null null 1
200I null null null null "Goodyear Tire", 8000.00, 200 null null 1
1 2 3
Sort 1
Union
```

```
200I null null null null null null null null 2
200I "Cars R Us", 10, June, 1999, null null null null null 1
200I null null null null null null null null 1
200I "Firestone Tire", 2000.00, 50 null null null 1
200I null null null null null null null null 2
200I "Goodyear Tire", 8000.00, 200 null null 1
1 2 3
Sort 1
Union
```

Relations to XML: Issues

[Shanmugasundaram et. al., VLDB’00]

- Two main differences:
  - Ordered nested structures
  - Self-describing tags
- Space of alternatives:
  - Early Tagging
  - Late Tagging

```
Early Structuring
```

```
Late Structuring
```

Problem 1: Too many SQL queries
Problem 2: Fixed (nested loop) join strategy
XML Document Construction (Completeness and Performance)

- Any nested XML document can be constructed using “sorted outer union” approach
- 9x faster than previous approaches [VLDB’00]
  - 10 MB of data
  - 17 seconds for sorted outer union approach
  - 160 seconds for “naïve XML application developer” approach

Outline

- Motivation & High-level Solution
- Background (Relations, XML)
- Storing and Querying XML Documents
- Publishing Relational Data as XML Documents
- Conclusion

Conclusion

- XML has emerged as the Internet data format
- But relational database systems will continue to be used for data management tasks
- Internet application developers currently have to explicitly bridge this “data model gap”
- Can we design a system that automatically bridges this gap for application developers?

For Scus in /Customer
Where Scus/nname = “Jack”
Return Scus

Conclusion (Contd.)

- Yes! XPERANTO is the first such system
- Allows users to …
  - Store and query XML documents using a relational database system
  - Publish existing relational data as XML documents
  - … using a high-level XML query language
- Also provides a dramatic improvement in performance
<table>
<thead>
<tr>
<th>Industry Impact</th>
<th>Relational Database System Vendors</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Sorted outer union approach is used in the</td>
<td>• IBM, Microsoft, Oracle, Informix, …</td>
</tr>
<tr>
<td>DB2 XML Extender product (beta version)</td>
<td>– SQL extensions for XML</td>
</tr>
<tr>
<td>• XPERANTO is now an IBM initiative</td>
<td>– XML Translation Layer</td>
</tr>
<tr>
<td></td>
<td>– “Pure XML” philosophy … provides high-level XML query interface</td>
</tr>
<tr>
<td></td>
<td>• SQL extensions for XML, while better than writing applications, is still low-level</td>
</tr>
<tr>
<td></td>
<td>– More powerful than XML-extended SQL</td>
</tr>
<tr>
<td></td>
<td>• SQL just not designed with nifty XML features in mind</td>
</tr>
</tbody>
</table>