XML Technologies and Applications

Rajshekhar Sunderraman

Department of Computer Science
Georgia State University
Atlanta, GA 30302
raj@cs.gsu.edu

III. XML/Database Mappings

December 2005
Outline

- Introduction
- XML Basics
- XML Structural Constraint Specification
  - Document Type Definitions (DTDs)
  - XML Schema
- XML/Database Mappings
- XMLParsing APIs
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- XML Querying and Transformation
  - XPath
  - XQuery
  - XSLT
- XML Applications
Conversion from XML to Objects

**Straightforward:**

```xml
<Person Name="Joe">
    <Age>44</Age>
    <Address>
        <Number>22</Number>
        <Street>Main</Street>
    </Address>
</Person>
```

**Becomes:**

```javascript
(#345, [Name: "Joe",
    Age: 44,
    Address: [Number: 22, Street: "Main"]
])
```
Conversion from Objects to XML

- Also straightforward
- Non-unique:
  - Always a question if a particular piece (such as Name) should be an element in its own right or an attribute of an element
  - Example. A reverse translation could give

```xml
<Person>
  <Name>Joe</Name>
  <Age>44</Age>
  <Address>
    <Number>22</Number>
    <Street>Main</Street>
  </Address>
</Person>
```

This or this
SQL/XML – Extending Reach of SQL to XML Data

• In the past, SQL was extended with object-oriented features (SQL2)
  – added values for reference, tuple(row type), and collection(arrays), …
  – took over ODL and OQL standards of ODMG

• Currently, SQL is being extended for XML:
  – adding data types and functions to handle XML
  – Many vendors have already support for XML and extended SQL to query XML data
Why SQL/XML

• Publish contents of SQL tables or entire DB as XML documents – need convention for translating primitive SQL data types

• Create XML documents out of SQL query results – need extension of SQL queries to create XML elements

• Store XML documents in relational DBs and query them – need extension of SQL to use XPath to access the elements of tree structures
Publishing Relations as XML Documents

- Current proposal:
  - no built-in functions to convert tables to XML
  - but can create arbitrary XML documents using extended SELECT statements

- Encoding relational data in XML:
  - Entire relation: an element named after the relation
  - Each row: an element named row
  - Each attribute: an element named after the attribute
Publishing Relations as XML Doc: Tables

<table>
<thead>
<tr>
<th>Id</th>
<th>Name</th>
<th>DeptId</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>Bob Smith</td>
<td>CS</td>
</tr>
<tr>
<td>3093</td>
<td>Amy Doe</td>
<td>EE</td>
</tr>
</tbody>
</table>

<Professor>
  <row>
    <Id>1024</Id><Name>Bob Smith</Name><DeptId>CS</DeptId>
  </row>
  <row>
    <Id>3093</Id><Name>Amy Doe</Name><DeptId>EE</DeptId>
  </row>
...</Professor>
Publishing Relations as XML Documents

**SQL:**
```
CREATE TABLE Professor
  Id: INTEGER,
  Name: CHAR(50),
  DeptId: CHAR(3)
```

**XML Schema:**
```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://xyz.edu/Admin">
  <element name="Professor">
    <complexType>
      <sequence>
        <element name="row" minOccurs="0" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Id" type="integer"/>
              <element name="Name" type="CHAR_50"/>
              <element name="DeptId" type="CHAR_3"/>
            </sequence>
          </complexType>
        </row>
      </sequence>
    </complexType>
  </element>
</schema>
```
Publishing Relations as XML Doc: Schema

- **CHAR\_len**: standard conventions in SQL/XML for `CHAR(len)` in SQL.
  - For instance, `CHAR_50` is defined as

    ```xml
    <simpleType>
      <restriction base="string">
        <length value="50">
        </restriction>
      </restriction>
    </simpleType>
    ```

- A lot of the SQL/XML standard deals with such data conversion, and with user-defined types of XML, which are defined in SQL using `CREATE DOMAIN`.
Creating XML from Queries: Functions

**XMLELEMENT, XMLATTRIBUTES**

- An SQL query does not return XML directly. Produces *tables* that can have columns of type **XML**.

```sql
SELECT P.Id, XMLELEMENT
  (NAME "Prof", XMLATTRIBUTES (P.DeptId AS "Dept"), P.Name)
  AS Info
FROM Professor P
```

Produces tuples of the form

1024, <Prof Dept="CS">Bob Smith</Prof>
3093, <Prof Dept="EE">Amy Doe</Prof>
Creating XML Using Queries:
Functions XMLELEMENT, XMLATTRIBUTES

- XMLELEMENT can be nested:

```sql
SELECT XMLELEMENT (NAME "Prof"
    XMLELEMENT(NAME "Id", P.Id),
    XMLELEMENT(NAME "Name", P.Name),
    XMLELEMENT(NAME "DeptId", P.DeptId),
) AS ProfElement
FROM Professor P
```

Produces tuples of the form

```xml
<Prof>
    <Id>1024</Id><Name>Bob Smith</Name><DeptId>CS</DeptId>
</Prof>
<Prof>
    <Id>3093</Id><Name>Amy Doe</Name><DeptId>EE</DeptId>
</Prof>
```
Creating XML Using Queries: Function XMLQUERY

SELECT XMLQUERY ('<Prof>
   <Id>{$I}</Id><Name>{$N}</Name><DeptId>{$D}</DeptId>
</Prof>' -- template with placeholder variables
PASSING BY VALUE
   P. Id AS I, -- values of I substitute for placeholders
   P. Name AS N,
   P. DeptId AS D
RETURNING SEQUENCE
) AS ProfElement
FROM Professor P
• Placeholder can occur in positions of XML elements and attributes
• Expressions can be XML-generating expressions or SELECT statements
  – In the example above, could have
    SELECT QUERY(''<Prof>
       {$I} <Name>{$N}</Name> …
    </Prof>'
    PASSING BY VALUE XMLELEMENT(NAME "Id", P. Id) AS I
    …
• In general, the argument to XMLQUERY can include any XQuery expression
  (XPath or a full query)
Storing XML in Relational DB: Data Type XML

- Not stored as a string, but natively as a *tree structure*. Supports navigation via efficient storage and indexing.

```sql
CREATE TABLE StudentXML (
    Id INTEGER,
    Details XML
)
```

where `Details` attribute contains things of the form

```xml
<Student>
    <Name><First>Amy</First><Last>Doe</Last></Name>
    <Status>U4</Status>
    <CrsTaken CrsCode="305" Semester="F2003"/>
    <CrsTaken CrsCode="336" Semester="F2003"/>
</Student>
```
Storing XML in Relational DB: Data Type XML

- To validate, use

```sql
CREATE TABLE StudentXML (  
    Id INTEGER,  
    Details XML  
    CHECK(Details IS VALID ACCORDING TO SCHEMA 'http://xyz.edu/student.xsd')  
)
```

assuming the schema is stored at

http://xyz.edu/student.xsd
Modifying Data in SQL/XML: XMLPARSE

XML stored as appropriately indexed tree structure, but in SQL is specified as a sequence of characters – so need to parse:

```
INSERT INTO StudentXML( Id, Details ) VALUES
( 12343, XMLPARSE( 
'&lt;Student&gt;
 &lt;Name&gt;
 &lt;First&gt;Bob&lt;/First&gt;
 &lt;Last&gt;Smith&lt;/Last&gt;
 &lt;/Name&gt;
 &lt;Status&gt;U4&lt;/Status&gt;
 &lt;CrsTake CrsCode=&quot;CS305&quot; Semester=&quot;F2003&quot;&gt;
 &lt;CrsTake CrsCode=&quot;CS339&quot; Semester=&quot;S2004&quot;&gt;
 &lt;/Student&gt;'  )
)
```
Modifying Data in SQL/XML:
IS VALID ACCORDING TO SCHEMA

To validate inserted document:

```
INSERT INTO StudentXML(Id, Details)
VALUES(12343,
XMLPARSE(
'<!-- Student -->
  <Name>
    <First>Bob</First>
    <Last>Smith</Last>
  </Name>
  <Status>U4</Status>
  <CrsTake CrsCode="CS305" Semester="F2003"/>
  <CrsTake CrsCode="CS339" Semester="S2004"/>
<!-- Student -->')
IS VALID ACCORDING TO SCHEMA
'http://xyz.edu/Students.xsd');
```
Querying XML in Relations: Oracle SQL

Consider the following relation:

```sql
create table xorders (  	ono number(5) primary key,  
order SYS.XMLType  
);
```

and the following data:

```sql
insert into xorders values  
(1020,sys.XMLType.createXML('  
<order>  
  <takenBy>1000</takenBy>  
  <customer>1111</customer>  
  <recDate>10-DEC-94</recDate>  
  <shDate>12-DEC-94</shDate>  
  </order>');
```

```xml
<items>  
  <item>  
    <partNumber>10506</partNumber>  
    <quantity>1</quantity>  
  </item>  
  <item>  
    <partNumber>10507</partNumber>  
    <quantity>1</quantity>  
  </item>  
  <item>  
    <partNumber>10508</partNumber>  
    <quantity>2</quantity>  
  </item>  
  <item>  
    <partNumber>10509</partNumber>  
    <quantity>3</quantity>  
  </item>  
</items>
```
Oracle XMLType Querying

Get names and street address of customers who have placed orders

```sql
select distinct c.cname, c.street
from   xorders o, xcustomers c
where
    c.cno=o.order.extract('/order/customer/text()').getnumberval();
```

Get number and names of employees who have not taken a single order

```sql
select e.eno, e.ename
from   xemployees e
where  e.eno not in
    (select o.order.extract('/order/takenBy/text()').getnumberval()
     from   xorders o);
```
Oracle XMLType Querying

Get number and names of customers who have ordered part 10506

```sql
select distinct c.cno, c.cname
from   xorders o, xcustomers c
where  c.cno=o.order.extract('/order/customer/text()').getnumberval() and
       instr(o.orderr.extract('/order/items/item/partNumber'),
            '10601') > 0;
```

Get order number of orders that have not yet shipped (illustrates use of `existsNode` function)

```sql
select o.ono
from   xorders o
where  o.orderr.existsNode('/order/shippedDate') = 0;
```
Representing XML using Relations: 
Shared-Inlining Algorithm

Question: Given a DTD (schema), is it possible to map it into an equivalent Relational Schema.

Based on node graph constructed for the given DTD
- nodes are the elements and attributes
- edges are the parent-child relationships

• Nodes with an in degree of zero (roots!) are made into a separate relation
• Nodes with an in degree of one are inlined
• Relational databases do not support set valued attributes – so a separate relation is created for “multiple” occurrences of sub-elements (* and + elements)
• Each tuple in a relation is uniquely identified by a field called ‘id’
• The child relation has a field called ‘parentid’ that references the parent relation’s ‘id’ field
<!ELEMENT books (book*, library)>  
<!ELEMENT book (booktitle, year)>  
  <!ATTLIST book author CDATA #REQUIRED>  
<!ELEMENT booktitle (bookname,header*,color)>  
<!ELEMENT year (monthpub, datepub)>  
<!ELEMENT monthpub (#PCDATA)>  
<!ELEMENT datepub (#PCDATA)>  
<!ELEMENT bookname (#PCDATA)>  
<!ELEMENT header (hdrsize)>  
<!ELEMENT color (#PCDATA)>  
<!ELEMENT hdrsize (#PCDATA)>  
<!ELEMENT library (#PCDATA)>
Shared-Inlining will produce 3 tables: books, book, and header
create table books(
    booksid int,
    library varchar (25),
    primary key (booksid)
);

create table header(
    headerid int,
    hdrsize varchar(25),
    parentid int,
    primary key (headerid),
    foreign key (parentid) references book
);
create table book(
    bookid int,
    author varchar(25),
    bookname varchar(25),
    color varchar(25),
    monthpub varchar(25),
    datepub varchar(25),
    parentid int,
    primary key (bookid),
    foreign key (parentid) references books
);
<?xml version="1.0"?>
<!DOCTYPE books SYSTEM "book.dtd">
<books>

    <book author="james">
        <booktitle>
            <bookname>Marine Bio</bookname>
            <header>
                <hdrsize>20</hdrsize>
            </header>
            <header>
                <hdrsize>10</hdrsize>
            </header>
            <color>blue</color>
        </booktitle>
        <year>
            <monthpub>may</monthpub>
            <datepub>18th</datepub>
        </year>
    </book>

    <book author="Foster">
        <booktitle>
            <bookname>Mass Comm</bookname>
            <header>
                <hdrsize>30</hdrsize>
            </header>
            <header>
                <hdrsize>20</hdrsize>
            </header>
            <color>Orange</color>
        </booktitle>
        <year>
            <monthpub>may</monthpub>
            <datepub>25th</datepub>
        </year>
    </book>

</books>
<book author="Kimberley">
  <booktitle>
    <bookname>Chemistry</bookname>
    <header>
      <hdrsize>30</hdrsize>
    </header>
    <header>
      <hdrsize>15</hdrsize>
    </header>
    <color>Red</color>
  </booktitle>
  <year>
    <monthpub>jun</monthpub>
    <datepub>18th</datepub>
  </year>
</book>

<book author="Jacob">
  <booktitle>
    <bookname>Philosophy</bookname>
    <header>
      <hdrsize>20</hdrsize>
    </header>
    <color>Grey</color>
  </booktitle>
  <year>
    <monthpub>feb</monthpub>
    <datepub>19th</datepub>
  </year>
</book>

</books>
## Relational Data

### Books Table

<table>
<thead>
<tr>
<th>BOOKSID</th>
<th>LIBRARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Central Library</td>
</tr>
<tr>
<td>BOOKID</td>
<td>AUTHOR</td>
</tr>
<tr>
<td>--------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>James</td>
</tr>
<tr>
<td>2</td>
<td>Foster</td>
</tr>
<tr>
<td>3</td>
<td>Kimberly</td>
</tr>
<tr>
<td>4</td>
<td>Jacob</td>
</tr>
</tbody>
</table>
## Relational Schema

### Header Table

<table>
<thead>
<tr>
<th>HEADERID</th>
<th>HDRSIZE</th>
<th>PARENTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>4</td>
</tr>
</tbody>
</table>
Translating XML Queries to SQL

XPath (will study this later)

/books/book/booktitle/bookname

the equivalent SQL query is

select bookid, bookname, book.parentid
from book

Answer generated by SQL to XML utility (Oracle 9i)

<?xml version = '1.0'?>

<ROWSET>
  <ROW num="1">
    <BOOKID>1</BOOKID>
    <BOOKNAME>Marine Biology</BOOKNAME>
    <PARENTID>1</PARENTID>
  </ROW>
  ...
</ROWSET>
Translating XML Queries to SQL

XPath Query

/books/book[booktitle|header|hdrsize='15']/year/monthpub

Equivalent SQL query is

```sql
select bookid, monthpub, book.parentid
from book, header
where header.hdrsize='15' and
  header.parentid=book.bookid;
```

Answer generated by SQL to XML utility (Oracle 9i)

```xml
<?xml version = '1.0'?>
<ROWSET>
  <ROW num="1">
    <BOOKID>3</BOOKID>
    <MONTHPUB>jun</MONTHPUB>
    <PARENTID>1</PARENTID>
  </ROW>
</ROWSET>
```