Exploring XML for Data Exchange in the Context of an Undergraduate Database Curriculum

Suzanne W. Dietrich Susan D. Urban Hua Ma Yang Xiao Shama Patel
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-8809
dietrich@asu.edu s.urban@asu.edu

ABSTRACT

The relationship between XML and database management systems has become an important topic for coverage at the undergraduate level. This paper presents an approach to teaching the use of XML through the study of data exchange. After a brief review of XML, the paper provides a tutorial on the different features that are provided in major relational database products for the import and export of XML, providing a discussion of how these features can be used as implementation exercises for students. In addition to addressing the use of XML for data exchange in relational systems, the paper also provides an overview of several teaching tools that are also used in the study of XML for object-oriented data and also for the exchange of object-oriented and object-relational data.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems – *relational databases, object-oriented databases, textual databases.*

General Terms

Languages

Keywords

XML, data exchange, undergraduate database curriculum, relational databases, object-oriented databases, object-relational databases.

1. INTRODUCTION

The Extensible Markup Language (XML) [13] is an important topic to address when teaching database concepts to undergraduates. The SQL:2003 standard [8] includes a new part on XML-related specifications, called SQL/XML [6]. Further advancements to SQL/XML can be found in [7]. However, there is varying support in database products for features described in the standard. Database educators are now faced with the challenge of upgrading course content based on the evolving SQL/XML standards and understanding how to use XML-enabled database products as a teaching tool in the classroom.

At Arizona State University, we have developed an advanced database concepts course (http://www.eas.asu.edu/~cse494db) for undergraduates that assumes a prerequisite course on relational database systems (e.g., http://www.eas.asu.edu/~cse412). A recent textbook [5] provides detailed coverage of the advanced database topics addressed in this course. A companion Web site (http://www.eas.asu.edu/~advdb) gives curriculum examples that support the concepts covered in the book. One module of the course includes coverage of XML and its relationship to data management. The standard for XML is introduced, including Document Type Definitions (DTDs) and XML Schema for representing the valid content of an XML document. The use of XML is addressed in the context of both object-oriented and relational data, with a specific focus on how XML can be used to support data exchange between different applications and database systems. Related work on the use of XML in the classroom has been reported in [12], describing the use of XML for exercises involving data modeling, converting an XML schema to a relational schema, using XPath to query an Oracle XMLType, and parsing XML documents.

This paper provides an overview of our approach to using XML for data exchange as a means for teaching students the relationship between XML and databases. The paper is presented in a tutorial style, outlining the XML features of several different relational database products that we have explored for use in the classroom, including Microsoft Access, Oracle, SQL Server 2000, and the new SQL Server 2005 Express [9]. In addition to discussing how these features can be used for exercises involving relational data exchange, we also provide an overview of several teaching tools that are also used in the study of XML for object-oriented data (using Objectivity/DB) and for the exchange of object-oriented and object-relational data between Objectivity/DB and Oracle implementations of the same conceptual design. Students therefore experience the use of XML with several different database paradigms.

The remainder of this paper is structured as follows. Section 2 provides a brief overview of XML. Section 3 then describes the data exchange support for XML in the four relational systems mentioned above, providing a discussion of how these features can be used as a teaching tool. Section 4 addresses our experience with the use of XML in an object-oriented database, whereas Section 5 focuses on a tool that we have developed for using XML for data exchange between object-relational and object-oriented databases. Section 6 concludes the paper with a discussion of our exploration.

2. A BRIEF INTRODUCTION TO XML

XML is widely used for representing textual knowledge. The markup language allows for user-defined tags to provide the semantics of the data contained in the document. Consider the schema of a simple employee table [3]:

employee(eID, eLast, eFirst, eTitle, eSalary)

Figure 1 shows a sample XML document representing a canonical table-based mapping of the employee relational table in XML based on the format discussed in [6]. A *tag* is a label that is contained within a < and >. A label is case sensitive in XML and cannot contain white space. Each opening tag must be matched with a closing tag that has the same label except that the label is preceded by a / character. The term *element* refers to an opening tag, the enclosed text, and the corresponding ending tag. A well-formed XML document must contain a distinguished root element, which is employee in the example shown in Figure 1. Additionally, a well-formed XML document must contain a proper nesting of all elements. As shown in Figure 1, each tuple of the table is enclosed within a row element, which has elements corresponding to each column in the employee table.

Figure 1 illustrates an element-based representation of the XML data. The opening tags of elements may contain attributes, which have a name and a value that is enclosed in quotes. The following is an alternative attribute-based representation of a table row in XML:

```
<row  eID="456" eLast="Last456" eFirst="First456"
      eTitle="Software Engineer" eSalary="45456" />
```

3. RELATIONAL SYSTEMS

This section provides an overview of the support for XML in the four relational systems that we have explored: Access, SQL Server 2000, Oracle and the recently announced SQL Server 2005 Express. Initial support for representing XML in relational database products used CLOBs (Character Large OBjects) or varying length character strings (varchar), while more recent releases have introduced an explicit type for XML.

3.1 Access

Access has the capability to export and import XML data. To export the XML representation of data in a relational table, select the table in Access and then select the Export item from the File menu. (Another alternative is to use the shortcut of a right-mouse click on the table to open a pop-up window that shows the Export option.) Choose the "Save as Type XML Documents" option. Access generates several files on the export of a table:

- *xml*: The xml data corresponding to the table, where the XML distinguished root element is dataroot and each tuple of the table is enclosed by an element given by the table name.
- *xsd*: The XML Schema Definition for the generated XML data.
- *xsl*: An Extensible Stylesheet Language file that provides a transformation of the XML to HTML for viewing in a browser.

There are several options for importing XML data into Access. From the File menu, select Get External Data and then Import. There is an Options button that allows one of the following choices:

```
<employee>
   <row>
      <eID>456</eID>
      <eLast>Last456</eLast>
      <eFirst>First456</eFirst>
      <eTitle>Software Engineer</eTitle>
      <eSalary>45456</eSalary>
   </row>
<!-- This is an XML comment: other rows are not shown -->
   <row>
      <eID>999</eID>
      <eLast>Last999</eLast>
      <eFirst>First999</eFirst>
      <eTitle>Manager</eTitle>
      <eSalary>100999</eSalary>
   </row>
</employee>
```

Figure 1. XML Data for an employee Table

- *structure and data*: The structure of the XML document and its corresponding data is imported into a table, given by the name of the element enclosing each row. By default, the fields corresponding to imported XML data are defined in Access as text with maximum length of 255.
- *structure only*: Only the structure of the XML document is used to define a table.
- *append to an existing table*: By defining a table with the appropriate field types first, the import of XML data can be appended into the table and the textual data will be imported as the predefined type.

3.2 SQL Server 2000

SQL Server 2000 uses strings (varchar) for representing XML data. The FOR XML clause of the SELECT statement provides the capability to return XML data. The Transact-SQL (T-SQL) programming language provides programmatic support with the OPENXML function for importing XML data into a relational table.

The SELECT statement allows a FOR XML clause to return the result of a query in XML format:

```
SELECT ... FROM ... FOR XML mode [,ELEMENTS]
```

The *mode* specification indicates the structure of the resulting XML. By default, an attribute-based XML representation for the data is provided, where the columns of the table are represented as attributes in XML. The use of the ELEMENTS option changes the XML representation to be element-based, where each column of the table is represented as an XML element. The choices for the *mode* specification are: RAW, AUTO and EXPLICIT. The following query will be used to illustrate the XML result based on the *mode*:

```
SELECT * FROM employee WHERE eID='456' FOR XML
```

The RAW mode returns each tuple of the resulting query in a generic row element:

```
<row eID="456" eLast="Last456" eFirst="First456"
     eTitle="Software Engineer" eSalary="45456" />
```

The AUTO mode returns each tuple in an element named by the table name:

```
<employee   eID="456" eLast="Last456" eFirst="First456"
            eTitle="Software Engineer" eSalary="45456" />
```

```
CREATE PROCEDURE insertEmployeeXML
  @myxml varchar(2000)
AS
DECLARE
  @iTree int
  EXEC sp_xml_preparedocument @iTree OUTPUT, @myxml
  INSERT employee(eID, eLast, eFirst, eTitle, eSalary)
     SELECT * FROM OPENXML(@iTree,'/dataroot/employee',2)
        WITH (eID          varchar(5)    'eID',
              eLast        varchar(20)   'eLast',
              eFirst       varchar(20)   'eLast',
              eTitle       varchar(50)   'eTitle',
              eSalary      float         'eSalary')
  EXEC sp_xml_removedocument @iTree
```

Figure 2. Inserting XML Data in SQL Server 2000

The EXPLICIT mode provides the capability to specify the structure of the resulting XML. Due to space limitations, the full details of the EXPLICIT specification are beyond the scope of this paper. The ELEMENTS option returns an element-based XML representation:

```
<employee>
    <eID>456</eID>
    <eLast>Last456</eLast>
    <eFirst>First456</eFirst>
    <eTitle>Software Engineer</eTitle>
    <eSalary>45456</eSalary>
</employee>
```

Figure 2 illustrates the population of the employee table in SQL Server 2000. The OPENXML function parses and queries an XML document to return values in the form of rows and columns. The sp_xml_preparedocument and sp_xml_removedocument are two system procedures that are associated with the OPENXML function. The sp_xml_preparedocument procedure takes an XML string as a parameter and parses it to build an XML internal tree representation (pointed to by @iTree). Once the document is prepared, OPENXML can be used to query the document and return a result set. The sp_xml_removedocument procedure should be executed after XML processing to release the allocated memory.

3.3 Oracle

Oracle (Version 9) introduced a data type for XML, called XMLType. The attributes of a table can be defined to be of type XMLType, allowing the storage of XML within a table. Oracle provides two packages for supporting XML: DBMS_XMLGEN and DBMS_XMLSave.

DBMS_XMLGEN is a PL/SQL package that supports the creation of XML from an SQL query. To generate an XML document, first create a context handle by passing the query to the parameter of the newContext method, where employeeCtx is declared to be of type DBMS_XMLGEN.ctxhandle:

```
employeeCtx :=
   DBMS_XMLGEN.newContext('SELECT * FROM employee');
```

By default, Oracle uses an element-based representation of the table having the element ROWSET as the distinguished root and each tuple of the table is enclosed in a ROW element. The setRowSetTag and setRowTag methods allow for changing the defaults to the second parameter, where the first parameter is the XML context handle:

```
DBMS_XMLGEN.setRowSetTag(employeeCtx, 'employees');
DBMS_XMLGEN.setRowTag(employeeCtx, 'employee');
```

The XML results can be stored in either a CLOB or an XMLTYPE, using the getXML or getXMLType methods, respectively:

```
empXMLclob := DBMS_XMLGEN.getXML(employeeCtx);
```

or

```
empXMLtype := DBMS_XMLGEN.getXMLType(employeeCtx);
```

where empXMLclob is defined to be of type CLOB and empXMLtype is defined to be of type XMLType. A procedure can then be written to store the result in a file.

The import of XML data to be stored as a relational table requires calling the methods provided by the DBMS_XMLSave package in Oracle. Assuming that the table employee is already defined (but not populated), a context named empCtx is defined as type DBMS_XMLSave.ctxType and assigned as the context for the save of the XML data to the employee table:

```
empCtx := DBMS_XMLSave.newContext(employee);
```

The setRowTag method allows for the specification of the row tag that encloses each tuple. In this example, it is assumed to be called employeeRow:

```
DBMS_XMLSave.setRowTag(empCtx, 'employeeRow');
```

The following call to the setIgnoreCase method tells Oracle to ignore any case differences in matching XML element names to database attribute names, since the database convention is to ignore case sensitivity in attribute names:

```
DBMS_XMLSave.setIgnoreCase(empCtx, 1);
```

A call to the insertXML method inserts the employee data represented in XML as a CLOB into the employee table, returning the number of rows inserted into the table:

```
numberOfRows :=
     DBMS_XMLSave.insertXML(empCtx, empXMLclob);
```

The closeContext method releases the resources associated with the context:

```
DBMS_XMLSave.closeContext(empCtx);
```

The new XMLType in Oracle allows for storing XML directly in the database. Consider the following table definition:

```
CREATE TABLE sampleXMLtable
     (xmlColumn XMLType);
```

Assume that the variable myXMLclob contains a CLOB representation of XML data, the following code snippet shows how to create an instance myXMLdata of an XMLType from a CLOB and insert the XML into the table:

```
myXMLdata := XMLType.createXML(myXMLclob);
INSERT INTO sampleXMLtable VALUES (myXMLdata);
```

3.4 SQL Server 2005 Express

The recently announced SQL Server 2005 Express [9] also supports a new type called XML, extending the features discussed for SQL Server 2000 with inherent support for XML. The stored procedure sp_xml_preparedocument has been extended to support a parameter of type XML rather than a string (varchar) and the FOR XML clause has been extended with several new features.

The FOR XML clause of the select statement allows a TYPE option, returning an instance of type XML, which is assigned to the myxml variable:

```
DECLARE @myxml XML
SET @myxml =
        (SELECT * FROM employee WHERE eID='456'
         FOR XML, ELEMENTS, TYPE)
```

The FOR XML clause of the select statement has also been extended to allow a PATH mode and ROOT directive for specifying a more complex XML structure. Consider as a motivational example, the creation of an XML representation of the employee data that

- returns the eID as an id attribute in XML,
- encloses the eLast and eFirst attributes within a name element,
- encloses each tuple within an element named employeeTuple, and
- names the distinguished root of the XML document as employeeDataRoot.

The following select statement illustrates the query specification:

```
SELECT      eID AS '@id',
            eLast AS 'name/eLast',
            eFirst AS 'name/eFirst',
            eTitle,
            eSalary
FROM        employee
FOR XML     PATH('employeeTuple'),
            ROOT('employeeDataRoot')
```

The following indicates the resulting XML (with only one employee tuple shown for brevity of presentation):

```
<employeeDataRoot>
    <employeeTuple id="456">
      <name>
        <eLast>Last456</eLast>
        <eFirst>First456</eFirst>
      </name>
      <eTitle>Software Engineer</eTitle>
      <eSalary>45456</eSalary>
    </employeeTuple>
…
</ employeeDataRoot >
```

SQL Server 2005 Express also has the capability to store XML data as an attribute value. Consider the following table definition:

```
CREATE TABLE sampleXMLtable
        (xmlColumn XML);
```

The INSERT statement can be used to populate a column of type XML:

```
INSERT INTO sampleXMLtable VALUES (@xmlColumnValue);
```

where the variable @xmlColumnValue can be declared either as type XML or varchar, and is assumed to have the value of an XML document. The system automatically converts a varchar argument to type XML for inserting the values.

3.5 Relational Data Exchange

The features described in the previous subsections provide useful tools that can be used in exercises to explore XML through data exchange between different database products. Exercises can be designed to use the export/generation feature of one product and then use the import/loading feature of the other product. Depending on the products available, exercises can either focus on the transfer of data between a relational table and XML format or the storage of an XML file as an XML type in a relational table. For example, the use of the FOR XML, ELEMENTS option of the SELECT statement in SQL Server (2000 or 2005 Express) results in an element-based representation of the table, with the table name as the element enclosing each tuple. The Oracle DBMS_XMLSave package can then be used to load the XML data into a table, specifying the table name as the row tag in the setRowTag method.

4. AN OBJECT-ORIENTED EXPLORATION

Our exploration of XML in the classroom includes the use of XML with object-oriented and object-relational database systems so that students also have an understanding of how to use XML to represent object-oriented data. The study of XML with the use of object-oriented databases makes use of the Object Manager tool developed at ASU and presented in [4]. The Object Manager is a graphical user interface tool for interacting with object-oriented databases such as Objectivity/DB. An XML file is used to communicate the schema of an object-oriented database to the Object Manager, allowing students to define the classes in the database, the attributes and keys of each class, inverse relationships between classes, and method names for getting and setting attributes and relationships. The Object Manager uses this schema information in the XML file to generate a generic user interface that allows students to create objects, delete objects, and manage relationships between objects. The Object Manager also supports importing of XML data into an object-oriented database as well as exporting of data to an XML file. Specific XML examples of schema and data files used with the Object Manager can be found in [4].

Our most current use of the Object Manager is integrated with a tool that we have developed known as the Object Database Generator [10]. The Object Database Generator, outlined in the following section on object-relational exploration, allows students to use XML to study similarities and differences between object-oriented and object-relational representations of data.

5. AN OBJECT-RELATIONAL EXPLORATION

XML for the representation of object-relational data is explored through the use of the Object Database Generator (ODG) [10]. The ODG was developed to support the portability of data between an Oracle object-relational database system and the Objectivity/DB object-oriented database system. Figure 3 shows the architecture of the ODG. Beginning with a database schema described using the Object Definition Language (ODL) of the Object Data Management Group (ODMG) standard [1], the ODG supports the creation of Objectivity and Oracle database implementations from the same ODL schema. For Objectivity, the ODG compiles an ODL schema to generate an XML file that conforms to the schema description required by the Object Manager tool. Students then implement the Java classes for the object-oriented database implementation. For Oracle, the ODG gives students a choice of bidirectional or unidirectional relationships for all inverse relationships in the ODL schema based on an object-relational mapping approach described in [5]. Furthermore, for the many side of a 1:N or a M:N relationship, the user must specify the implementation to be either a varray or a nested table, which are the choices for representing collections in Oracle. The ODG then generates the appropriate object type, object table, varray, and nested table definitions for the creation of an object-relational database. Students complete the implementation with the creation of the appropriate stored procedures and functions.

A subcomponent of the ODG is the Oracle-to-Objectivity Converter (OOC), originally developed in [11] and modified in [10] for use as part of the ODG. The OOC transforms data in the object-relational implementation of the ODL schema into an XML data file that conforms to the data format used by the Object Manager. As part of the transformation, the OOC generates the inverse data for unidirectional relationships. Students then experiment with exporting data from the Oracle object-relational implementation into the corresponding Objectivity database through the XML data import feature of the Object Manager. The ODG together with the OOC allows students to study the similarities and differences between the object-oriented and object-relational implementations, using XML to transfer data from one implementation to the other.

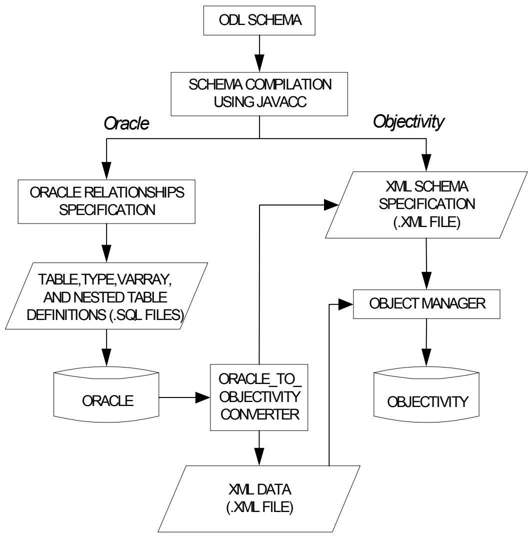
ODL SCHEMA
SCHEMA COMPILATION USING JAVACC
Oracle
Objectivity
ORACLE RELATIONSHIPS SPECIFICATION
XML SCHEMA SPECIFICATION (.XML FILE)
TABLE,TYPE,VARRAY, AND NESTED TABLE DEFINITIONS (.SQL FILES)
OBJECT MANAGER
ORACLE
ORACLE_TO_ OBJECTIVITY CONVERTER
OBJECTIVITY
XML DATA (.XML FILE)

Figure 3. Architecture of the Object Database Generator

6. DISCUSSION

This paper has explored the use of XML for data exchange between relational systems as well as data exchange between object-oriented and object-relational database systems. It is important to note that many existing relational database systems also provide the capability to export a database in a format compatible for another product. For example, SQL Server 2000 has a tool, known as the Data Transformation Service, which transforms data from SQL Server to Oracle without the use of XML. Also, some products may also provide separate tools for mapping tables and views to XML documents, such as Oracle's XML SQL Utility. We have found, however, that an XML-based approach to data exchange is a useful learning tool, where students are exposed to the universal nature of XML as a means for representing data and its corresponding description, together with relevant database tools that support the import and export of XML data.

As the support for XML in commercial database products continues to grow, so will our approach to the incorporation of XML into the database curriculum. For example, some of the products discussed in this paper already have some form of support for querying XML [2]. We anticipate the expression of queries over XML data as the next module in our approach to teaching XML within an advanced database curriculum for undergraduates.

7. ACKNOWLEDGMENTS

We would like to thank Ingrid Biswas and Alexey Ushakov who supported earlier versions of this exploration. This research was supported by NSF Grant No. DUE-9980417 and the Microsoft Research Content and Curriculum Program.

8. REFERENCES

[1] Cattell, R. G. G. et al. (eds.) *The Object Database Standard: ODMG 3.0*. Morgan Kaufmann, San Franciso, 2000.

[2] Chaudhri, A. B., Rashid, A., and Zicari, R. (eds.) *XML Data Management: Native XML and XML-Enabled Database Systems*. Addison-Wesley, Boston, 2003.

[3] Dietrich, S. W. *Understanding Relational Database Query Languages*. Prentice Hall, Upper Saddle River, NJ, 2001.

[4] Dietrich, S. W., Suceava, D., Cherekuri, C., and Urban, S. D. A Reusable Graphical Interface for Manipulating Object-Oriented Databases Using Java and XML. In *Proceedings of the ACM Technical Symposium on Computer Science Education* (*SIGCSE '01*) (North Carolina, Feb. 2001). ACM Press, New York, NY, 2001, 362-366.

[5] Dietrich, S. W. and Urban, S. D. *An Advanced Course in Database Systems: Beyond Relational Databases*. Prentice Hall, Upper Saddle River, NJ, 2005.

[6] Eisenberg, A. and Melton, J. SQL/XML is making good progress. *ACM SIGMOD Record*, 31, 2 (Jun. 2002), 101-108.

[7] Eisenberg, A. and Melton, J. Advancements in SQL/XML. *ACM SIGMOD Record*, 33, 3 (Sep. 2004), 79-86.

[8] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J., and Zemke, F. SQL:2003 has been published. *ACM SIGMOD Record*, 33, 1 (Mar. 2004), 119-126.

[9] Microsoft Corporation. SQL Server 2005 Express Edition, 2004. http://lab.msdn.microsoft.com/express/sql/

[10] Patel, S. *The Object Database Generator*. M.C.S. Project, Department of Computer Science and Engineering, Arizona State University, 2003.

[11] Ushakov, A. *Oracle_to_Objectivity Converter*. Undergraduate Independent Study Report, Department of Computer Science and Engineering, Arizona State University, Fall 2002.

[12] Wagner, P. and Moore, T. Integrating XML into a Database Systems Course. In *Proceedings of the ACM Technical Symposium on Computer Science Education (SIGCSE '03)*, (Nevada, Feb. 2003). ACM Press, New York, NY, 2001, 26-60.

[13] World Web Wide Consortium. XML. http://www.w3.org/XML/