"X-DATABASES" – THE INTEGRATION OF XML INTO ENTERPRISE DATABASE MANAGEMENT SYSTEMS

by Leah Davis

A Master's paper submitted to the faculty
of the School of Information and Library Science
of the University of North Carolina at Chapel Hill
in partial fulfillment of the requirements
for the degree of Master of Science in
Information Science.

Chapel Hill, North Carolina

October, 2000

Approved by:

_____

Gregory B. Newby
Advisor

ABSTRACT

Leah Davis. “X-Databases” – The Integration of XML into Enterprise Database Management Systems. A Master's paper for the M.S. in I.S. degree. October, 2000. 65 pages. Advisor: Gregory B. Newby.

An examination of how the eXtensible Markup Language (XML) and database management systems (DBMS) fit together, and current approaches to providing database technologies that support XML. Analysis of how XML is being deployed in four classes of XML Database (X-Database) applications provides a basis for understanding the direction of X-Database technology and associated standards.

In a simple implementation, an XML Document Type Definition (DTD) is mapped to relational structures, and XML data are stored in a DBMS (Oracle8i). Sample queries are presented to retrieve XML from the database. A middleware tool (XSQL Java Servlet) is used to transform query results into records on a Web page. The results demonstrate that relational databases require data to be rigidly mapped to relational structures.

The paper concludes by exploring future challenges to integrating XML and DTDs with X-Databases, which establishes the need for a more “native” integration approach.

Headings:

XML (Document Markup Language) - Database Management Systems (DBMS)

Relational Database – Oracle8i, XSQL, XSU

XML – XML Databases, Document Type Definitions (DTDs), Schemas, XQL

TABLE OF CONTENTS

INTRODUCTION

One of the most decisive issues in information technology (IT) today is the question of how to integrate the new metalanguage XML (eXtensible Markup Language) with typical database management tasks like the retrieval, storage, and querying of data. Some predict that XML will dramatically change the way enterprises work with data, driving a new generation of database and data integration solutions (X-Databases).

The question to be addressed is: What are current approaches to integrating XML in enterprise Database Management Systems (DBMS), and what products, standards, and integration issues are shaping the future direction of X-Database technology? The term enterprise refers to larger organizations that use DBMS technology, including *inter alia* corporations, government bodies, and non-profit institutions.

The paper opens with an overview of the concept of XML, its information structures, optional XML modules, and how XML is parsed and presented. This provides a background for understanding how XML is being used in both Internet and Database applications, and how XML and emerging “standards” are changing traditional approaches to data integration and exchange.

The review of how XML is being used in databases leads to a closer look of four classes of X-Database products deployed in enterprises today. Some products store and retrieve data directly as XML; others require some middle layer that performs a conversion or transformation to XML. Some technical possibilities and issues in using these X-Database products are discussed.

A simple implementation of XML in an Oracle8i database demonstrates how XML and a relational database can be integrated. An XML data transfer layer (middleware) is used to query the database and to present the results as XML-based information on a World Wide Web (Web) page.

Lastly, the paper explores future challenges to integrating XML with databases. It concludes with an analysis of the likely direction of X-Database technology, suggesting that there is a need to build more “native” X-Databases that are shaped by the essential characteristics of XML itself: the ability to be freely extensible.

WHAT IS XML?

Before one can start thinking about XML and databases, it is important to understand the concept of XML. This section provides an overview of XML, its principles and key concepts, and how XML documents are structured.

XML is a specification defined by the World Wide Web Consortium (W3C) that provides a set of grammar and syntax rules, guidelines and conventions for semantically describing the structure of data (W3C, 1998a). XML lets document authors define their own markup tags and attribute names to assign meaning to the data elements in a document. Further, hierarchical combinations of XML elements can be nested to relay information about data relationships.

The W3C formally ratified XML as an open Web standard in February 1998. An open standard means that the standard is not owned or controlled by one company. Since 1998, XML has become a widely used specification for defining networked data across a number of application domains (Henry, 1999).

XML is a subset or restricted form of the Standard Generalized Markup Language (ISO 8879) (SGML) (W3C, 1998a). SGML is an international standard metalanguage for text markup systems that allows documents to be self-describing through the specification of custom tag sets.

The Hypertext Markup Language (RFC 1866) (HTML) is also a subset of SGML; but it is not extensible like XML. HTML markup is limited to a predefined vocabulary of tags that describe how to present the data — for example, with headings, paragraphs, lists, images, etc. (and some provision for hypertext and multimedia.) One can control whether HTML document content is bold, italic, and indented. One can also exercise some basic control over structure, e.g. headings, paragraphs, and lists. However, it is not possible to assign semantics or to describe specific data in an HTML document (Simpson, 1999).

XML markup is not limited to defining formatting and visual presentation, as in HTML documents. Rather, XML markup provides descriptions of the data (metadata), allowing software to understand the meaning of the data automatically. Metadata is a definition or description of data (whereas metalanguage is a definition or description of

language) [1]. It is this ability to encode or “markup” text documents with tags that describe the data and a structure that the author defines that makes XML extensible.

Consider the following Example 1, which is an HTML document that represents the document structure of an employee record. A corresponding XML example follows in Example 2.

Example 1

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
   <head>
      <title>Employee List</title>
   </head>
   <body>
<table>
<tr><td>7788</td><td>JOHN</td><td>SCOTT</td><td>ANALYST</td><td>756
6</td><td>4/19/1987</td><td>3000</td><td>20</td></tr>
 </table>
   </body>
</html>
```

This example shows that HTML may reflect the document structure, but it cannot adequately represent the structure of data.

XML gets around this problem by allowing users to invent custom markup tags for identifying specific information within documents. The angle brackets <> and text inside are called tags, and each set of tags and its enclosed data are called an element. Consider Example 2, an example of an XML document describing an employee record. Note the addition of XML data tags and the nested structure.

Example 2

```
<?xml version="1.0" encoding="UTF-8"?>
<EMP>
  <EMPNO>7654</EMPNO>
  <ENAME>
     <FNAME>JOHN</FNAME>
     <LNAME>SCOTT</LNAME>
  </ENAME>
  <JOB>SALESMAN</JOB>
  <MGR>7566</MGR>
  <HIREDATE>4/19/1987</HIREDATE>
  <SAL>1250</SAL>
  <DEPTNO>20</DEPTNO>
</EMP>
```

This example clearly conveys a relationship between various data objects. Further, each conceptual piece of data is represented by its own XML element, such as <ENAME> and <SAL>. XML does not require tags or attributes to be pre-declared, and XML places no limits on how a user nests different elements.

A Document Type Definition (DTD) further extends the ability of users to describe the structure and nature of an XML document. A DTD is a file (or several files used together), written in XML's declaration syntax, that defines the document type and structural rules for a document's tags or attributes (St. Laurent, 1999a). A DTD may be internal — contained within the XML document — or external — in a separate file or Universal Resource Indicator (URI).

Metadata is published in a DTD file for reference by other systems. It is analogous to the Database Definition Language (DDL) file that is used to define the structure of a database, but with a different syntax (Finkelstein, 1999).

Unlike HTML, which has a DTD fixed by the W3C, XML enables authors to describe a DTD that is specifically tailored for different functions, providing the ability to:

- Describe a consistent "grammar" or structure for the type of document to be displayed, so that all documents that belong to a particular type will be constructed and named in a consistent manner. For example, in DTDs for specific industries or fields of knowledge, basic requirements can be forced on the structure of the documents (e.g. sequence, existence of particular elements) that are automatically verifiable via an XML parser. As requirements change, new tags can be added to the DTD to reflect new data types or to extend existing ones.

- Assign a context, meaning, and dependencies to a subset of tags. Without a DTD, one can use virtually any tag in an XML document (provided it complies with the XML syntax rules for well-formedness described below).

This flexibility to specify a DTD is really what differentiates XML from HTML. XML is extensible because a user gets to specify the DTD; HTML is not extensible because the DTD is fixed, at least for a particular HTML version like 3.2 or 4.0. For example, the HTML 4.01 specification includes a DTD with syntactic constraints for style sheets, scripting, embedding objects, etc. that cannot be modified.

SGML also provides the ability to specify a DTD; however, SGML is much more complex and rigid than XML. For example, SGML has more elaborate white space handling rules than XML, and SGML documents require that a DTD be present, whereas XML does not require a DTD.

An XML document may define its own markup informally by the simple existence and location of elements in an XML document (a DTDless XML document). XML omits the more complex and less-used parts of SGML in return for the benefits of self-description, extensibility, structure, and validation (Harold, 1999).

XML has been specifically designed for ease of implementation and for interoperability with both SGML and HTML (W3C, 1998). This makes XML a standard that can be used universally ("create once, read anywhere"). Any Internet object can be specified and integrated using XML.

Principles & key concepts behind XML

Physically, XML documents are made up of storage units called entities, which contain either parsed data or unparsed data (W3C, 1998a). Parsed data are made up of characters, some of which form character data and some markup.

Markup encodes a description of the document's storage layout and logical structure with tags (elements), entity references, comments, processing instructions, document type declarations, XML declarations, and CDATA section delimiters. The logical structure of an XML document refers to the prolog and the body of the document.

The prolog consists of the XML declaration, that is, the version number; a possible language encoding; other attributes (name="value" pairs); and an optional DTD. The prolog precedes the body of the document.

The body of the document contains the remainder of the XML document. The body starts with a single "root" element (DOCROOT) declaration, for example <EMP> in Example 2 above. This marks the beginning and end of the document and is considered the parent of all other elements, attributes, CDATA, entity references, comments, text, and processing instructions, which are nested within the root element's start-tag and end-tag.

There are optional modules that provide sets of tags and attributes for specific tasks. For example, CSS, XSL, and XML schemas. Other optional modules are the XML Linking Language (XLink), XML Pointer (XPointer) and XML namespaces.

XLink describes a standard way to add hyperlinks to an XML file [2]. XPointer is a syntax for pointing to parts of an XML document [3]. Together, they describe numerous ways to express complex but flexible linking relationships between XML documents. Since this paper focuses on the core XML specification, a detailed exploration of various proposed mechanisms for XLink and XPointer inter-document references is not warranted.

XML Namespaces is a specification that describes how you can associate a URI with every single tag and attribute in an XML document (W3C, 1999a). XML Namespaces are intended to prevent potential conflicts between identically named XML elements by associating a prefix that identifies an intended namespace with a URI.

XML documents and information structures

There are structural and notational rules that apply to all XML documents. XML documents that conform to these rules are considered well formed, and may be read and parsed by an XML application. The rules of well formedness are:

- All start-tags and end-tags must match up (omission is not allowed except for empty elements);
- Empty elements either end with '/>' or use the special XML syntax <empty/>;
- All the attribute values are quoted (e.g. <a href="http://www.w3.org/TR/REC-xml">);
- All the entities are declared (entities are re-usable chunks of data, much like macros, which is part of XML's inheritance from SGML);
- There must not be any isolated markup-start characters (< or &) in CDATA (they must be given as < and &); and
- Elements must nest inside each other properly (no overlapping markup).

Well-formed XML documents that conform to a DTD are considered valid. A valid XML document must refer to a DTD using a document type declaration that references the

DTD and its location [4]. As discussed earlier, a DTD is not required — an XML document may be DTDless.

An XML document that references a DTD specifies what names can be used for elements, attributes, and entities, where they may occur, and how they all fit together. Consider Example 3, a DTD for the employee data described in Example 2.

Example 3

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM?, DEPTNO)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (FNAME+, LNAME+, MNAME?)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
<!ELEMENT MNAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT MGR (#PCDATA)>
<!ELEMENT HIREDATE (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
<!ELEMENT COMM (#PCDATA)>
<!ELEMENT DEPTNO (#PCDATA)>
<!--================================================================-->
<!ENTITY XML "Extensible Markup Language">
<!ENTITY AUTHOR "Leah Davis - UNC-CH SILS">
```

The goal of this DTD is to provide structural rules that describe an employee record. If the XML document in Example 2 referenced the above DTD (for example, by including a declaration `<!DOCTYPE EMPLIST SYSTEM "emp.dtd">`, Example 2 would be a valid XML document.

XML APIs (tree-based API processing with DOM)

Once a DTD exists, a parser is used to read through the XML document (the input stream), check for conformity to the document type defined, and extract the data (content) from the tags.

A parser is a program that takes data and puts it in a format or language that the application can understand. A well-formed XML document is parsed to display the document (in a browser) or to read the data in the document (from a database).

Oracle8i is a DBMS that includes XML validating parsers for Java (versions 1.1 and 2), C, C++, and PL/SQL.

When the XML processor parses an input stream, it maps the result to the Document Object Model (DOM) — a tree-based hierarchical object — for further manipulation by the application. XML documents can be modified directly at the DOM level.

DOM is a standard set of function calls for manipulating XML from a programming language (W3C, 1998b). The DOM provides an abstract Application Programming Interface (API) for constructing, accessing, and manipulating XML (and HTML) documents. DOM defines specialized interfaces for documents, elements, text, attributes, entities, and other abstractions (Birbek, 2000).

DOM appears to be the most widely accepted API (Ogbuji, 1999). Oracle, Microsoft and other vendors have bound DOM to their proprietary programming languages to provide APIs with the ability to query and manipulate XML documents in memory. DOM Level 2 is the most recent version (W3C Candidate Recommendation as of May 2000).

Alternatively, other APIs can be used. Below is a list of alternatives — each has appropriate uses in particular environments:

- SAX: The Simple API for XML: an event-based interface for XML parsers, which differs from the tree-based DOM approach to parsing. SAX is a public domain API, which is used as the basis for a number of Java APIs. Most XML parsers support the May 1998 version of SAX, SAX1. Many, but not all, major parsers support the current version, SAX2, released in May 2000.

- JAXP: Java API for XML Parsing: Sun's JAXP, the Java API for XML Parsing, originally released in April 2000, provides a standard interface to XML for Java applications. It does not require use of Sun's Java Project X parser, although this is the default.

- Java Project X: Java Project X provides full XML processing capabilities, including an XML parser with optional validation, an in-memory object model tree that supports the W3C DOM Level 1 recommendation, and basic support for JavaBeans integration with XML.

- SAXON: The SAXON package is a collection of tools for processing XML documents. SAXON includes an XSLT processor (which implements the Version 1.0 XSLT and XPath Recommendations) and a Java library.

- Namespace APIs: Namespace interfaces give information relating to the XML document namespaces identified by URI references that qualify element and attribute names and location resources on different machines or in different XML documents (Chang, 2000). Essentially, namespace APIs allow identical names for elements and attributes to be qualified with URIs to differentiate the names.

Oracle8i's XML parsers support DOM, SAX, and Namespace APIs (Ramalho, 2000).

Presenting XML using XSL style sheets

After the system has parsed the XML input into the DOM object nodes (hierarchical, parent/child node relationships), an application (or multiple applications) can access the objects for further processing or presentation (Robie, 1998). Further processing might include rendering the XML data for a variety of media and application targets, such as a Web page, e-mail, pager display, cell phone, or formatting for input and integration with a database or an enterprise resource planning (ERP) system.

For content rendered in a Web browser, CSS, the Cascading Stylesheet Specification (CSS), can be used to assign styles to elements. XML separates structure and content from presentation — the XML data defines the structure and content, and a stylesheet defines the presentation. A stylesheet means a document that specifies how to process and format the elements of another document.

However, only the Extensible Stylesheet Language (XSL) specification has the ability to parse once and transform, organize and render XML content to many different media and application targets by applying stylesheets (W3C, 2000a). For example, with a DBMS is possible to transfer data from the database into a format specified by a device or user without having to modify the underlying database code. This is an important capability for allowing different databases and applications to share data.

Physically, an XSL stylesheet is an XML file that uses XML syntax and that combines formatting features from CSS.

An adjunct to XSL is the Extensible Stylesheet Language Transformation (XSLT), a transformation language for rearranging, adding or deleting tags and attributes. XSL is based on XSLT. XSLT transforms XML documents into other types of XML documents.

An XSLT processor is built in to the Oracle8i XML processor to use in conjunction with XSL style sheets for rendering an XML stream into different formats for different targets (Chang, 2000). For example, Oracle's XSLT processor correlates patterns in an XSL stylesheet and then applies commands (e.g. HTML tags to surround data found) to output the results as a new XML document.

Schemes and DTDs

XML has no data types — it is just text. Because DTDs are designed for use with text, they have no mechanism for defining the content of elements in terms of data types (typed attributes). Thus, a DTD can only be used to specify markup rules — it cannot be used to specify numeric ranges or to define limitations or checks on the text content.

Schemas provide a better means of specifying typed attributes. They provide criteria for validating the content of elements and markup. Schemas also provide more precise descriptions of valid structures and types of data than is achievable with DTD.

To force an XML file to adopt a schema instead of a DTD, one needs to add an "xmlns attribute" to the root element. For Example 2, the syntax may read:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsd:schema PUBLIC "-//W3C//DTD XMLSCHEMA 19991216//EN" ""
[
  <!ENTITY % p 'xsd:'>
  <!ENTITY % s ':xsd'>
]>
<xsd:schema xmlns:xsd="http://www.w3.org/1999/XMLSchema">
  <xsd:element name="EMPLIST ">
```

There are a number of schema proposals under development. The W3C has a Working Group for a schema language named XML-Schema. The goal of XML-Schema is to define a more global and robust way of structuring and formatting the content and semantics of XML documents (W3C, 2000b). Microsoft has developed its own proprietary schema named BizTalk, which is specifically oriented toward data and

business process integration. At this point it is difficult to determine which proposal will become widely ratified.

In this section, the structural and notational rules of XML were explained, and XML was distinguished from HTML and SGML. This provides a background for the next section, which describes how XML is being used in Internet and database applications to identify and describe data and typed attributes. The next section also takes a closer look at the proposals and standards being developed to define approaches to integrating and exchanging XML-based information.

HOW IS XML BEING USED?

There are many potential uses of XML. This section captures the current state of how XML is being used in Internet and database applications.

Uses of XML in Internet applications

It appears that XML is being used in the five major types of Internet applications:

1. Electronic commerce: To define, process, and manage cross-platform data formats for Internet-based Business-to-Business (B2B) and Business-to-Consumer (B2C) data interchange. The expectation seems to be that XML will enable developers to build more portable, scalable, and interoperable distributed Web applications that leverage open Internet standards (Oracle, 1999).

2. Internet content management and delivery: To provide native support for storage, retrieval, and querying of XML content, enabling database-driven Web sites to serve up more relevant search results and user-driven content.

3. Messaging and applications integration: To integrate business processes and systems in a transparent manner. For example, integration with back-end ERP and Customer Resource Management (CRM) systems from multiple vendors, systems from partners in the supply chain, and data warehouses [5].

4. Data warehousing: To enable automated exchange and transformation of enterprise metadata between enterprise database repositories, and business intelligence

applications (Hackathron, 1999). For example, XML is being used in Data Warehousing technologies for corporate portals, also called Enterprise Portals (EPs) or Enterprise Information Portals (EIPs), to integrate both structured and unstructured data throughout an enterprise [6].

5. Application development: To develop customized and extensible Web applications that make it easy to build web-database driven Internet applications.

XML models complex, hierarchical data. It is simple. It allows the use of domain specific tags (domain-specific vocabularies) that all browsers with XML parsers can understand. It provides a scaleable open platform, making it easier to integrate the process of gathering, processing and disseminating data on the Internet.

Enterprises increasingly see the value in this extensibility. XML gives enterprises an opportunity to create a single, consolidated view of corporate data — for example, to gain a more complete view of each customer in order to improve customer relations and speed up responsiveness [7]. An enterprise's ability to access and use its knowledge resources in this way can directly affect its competitive advantage and future prosperity.

XML is of particular importance to web-based electronic business (e-commerce), where large amounts of information must be exchanged within and outside an enterprise in a way that ensures high performance and transaction processing. As a result, productivity tools, knowledge management tools and office suites are beginning to support XML. For example, Microsoft Office 2000 uses XML to maintain the internal formats and styles used by Word, Excel and PowerPoint when converted to HTML — so that the same originating source products can later open those HTML documents again without losing relevant formatting detail. XML development tools such as the XML Developers Kit (XDK) are also being released so that XML applications can be developed more easily.

For many years, a great deal of effort has gone into the definition and implementation of Electronic Data Interchange (EDI) standards to address the problem of intercommunication between dissimilar systems and databases. EDI has been widely used to exchange documents between commercial partners to a transaction. It works

well, but it requires expensive proprietary software and it is complex. As a result, it is cost-justifiable generally only for large corporations (Finkelstein, 1999).

There are now moves to enable EDI data to travel inside XML. EDI vendors have become aware that XML is a more inexpensive and standardized way to do data translation and transformation. Once an organization's metadata is defined and documented, all programs can use XML to communicate. Further, optional modules like XSL style sheets provide a means to render the same data on different devices such as palm pilots and WebTV. This capability allows companies to focus more on using XML for business operations, without concern for the kind of output devices that will present the data.

The other thing that makes XML important for use in Internet applications is that it provides much better metadata about individual content elements created and stored in an XML DTD. If one adds XML and a DTD together, there is a powerful ability to not only create, but also to repurpose data. For example, XML provides the ability to build Internet applications that publish output to the Internet and port data to another application.

There are, however, some limitations to using XML in Internet applications. DTDs, schemes, domain-specific vocabularies and XML industry standards are still developing in selected industry areas. Over the last decade there has been a rush to build networks and to develop de facto technical standards to exploit Web industries. For example, Microsoft has developed its own proprietary schema for handling XML data (the XML Reduced Data Schema). XML-based protocols have been promulgated, such as the Information Content Exchange (ICE) protocol, which is designed to provide automated methods of repurposing and syndicating data. Numerous proprietary APIs have also been developed that are controlled by a particular company (for example, Sun's Java API for XML Parsing).

In this competitive environment, there is a concern that software, information, services and products may be lost in the cycles of technological change. There is also a concern that no one single organization should have control over the future of the Web. Instead, XML standards and protocols should promote the Web's evolution and ensure interoperability. Standards directly enhance compatibility and interoperability, reducing costs for both vendors (development costs and implementation time) and users

(collective switching costs arising from implementation of new protocols, installation of new software and retraining for new products).

No one entity is charged with developing standards for the Internet in general. Instead, communities such as the Internet Engineering Task Force (IETF) and consortia such as W3C are working cooperatively on developing standards for the smooth operation of the Internet.

The IETF is a large open international community of network designers, operators, vendors, and researchers. By contrast, the W3C is a consortium of organizations that pay a membership fee to support its operation (membership is open to any organization). W3C Member organizations include vendors of technology products and services, content providers, corporate users, research laboratories, standards bodies, and governments.

The W3C plays a central role in developing technical specifications, called Recommendations, which promote interoperable, accessible, universal, and portable communication standards on the Internet. These Recommendations are effectively standards documents. While they are not enforceable on W3C members (or anyone else), the Recommendations are widely supported in Internet industry products.

W3C has a formal structured process for assigning defined groups of experts to develop Recommendations. A member organization submits a proposed technology or other idea (Proposal) for consideration by W3C. A Note is issued to acknowledge the submission of the Proposal — though the Note does not guarantee any further action by W3C. If W3C decides to develop the Proposal, a W3C Working Group is established to create and publish a Working Draft. A Working Group consists of representative researchers and engineers working for W3C Member organizations. Requests for comments are made on the Working Draft (with limited opportunities for non-Member participation), and the Working Draft is revised and republished about every three months. Some Working Drafts may be dropped as active work — there are no guarantees. Final drafts of the Recommendations (Proposed Recommendations) are finally put up for Members to cast ballots on their acceptance or revision. If a Proposed Recommendation gets past the membership and the W3C Director (who has final say), it becomes a W3C Recommendation.

In just over two years, the W3C has developed more than four (4) Recommendations relating to XML, including the XML1.0 specification itself, XML Namespaces (W3C, 1999a), Associating Style Sheets with XML documents (W3C, 1999b), and XSLT for XML transformations (W3C, 1999c).

W3C currently has a Working Draft named the Resource Description Framework (RDF), which describes a language for using XML syntax for activities including sitemaps, content ratings, stream channel definitions, search engine data collection (web crawling), digital library collections, and distributed authoring (W3C, 2000c). RDF was initiated by the W3C to build standards for XML applications so they can inter-operate and communicate more easily using XML as an interchange syntax.

A number of W3C Proposals are also in the pipeline. Examples include:

- The eXtensible Query Language (XQL) (September 1998), a recent (and evolving) W3C proposal for using XSL as a general-purpose query language for making database-style queries on XML documents [8].

- A W3C Note entitled XML-Data (January 1998), which provides recommendations on how to make XML “database-ready” especially for three-tier architectures and hetergenous databases. XML-Data suggests syntax for schemes and a model for extending XML elements by adding data types, presentation rules, inheritance, etc. XML-Data will be described in more detail below [9].

- A W3C Note entitled Simple Object Access Protocol 1.1 (SOAP) (May 2000), which proposes a common remote procedure call (RPC) mechanism for basic information access across any operating system or object interface protocols [10]. Until recently there hasn’t been a common RPC mechanism for any operating system, language, or object interface (for example, Microsoft COM vs. enterprise Java bean environment). SOAP may provide a bridge between Windows, Java, and scripting environments like Python and Perl.

While W3C Recommendations are considered stable — at least until the next revision — and thus safe to implement, the specifications continue to evolve. The XML 1.0 Recommendation itself is still maturing. The most recent version is Version 1.6 (July 2000). Vendors may be behind or ahead of the W3C in releasing tools that support

these specifications in their products and application interfaces. For example, not all standard Web browsers currently support XML.

Clearly, XML has momentum in terms of its integration into Internet applications; however, standardization has a way to go. At this point, XML is really just a collection of emerging specifications and related vocabularies based on the XML syntax. With rapidly changing standards and middleware vendors competing on functionality, it is a tenous environment in which to develop or adopt X-Database products or any other XML-supported Internet application. It will take some time before standards mature and platforms converge to provide seamless interoperability and integration of data processes.

Uses of XML in Database Applications

XML is providing a platform for databases to integrate different data types and to share data sources, facilitating an unprecedented opportunity for large-scale data/document integration and exchange.

Integration of Data Types

One of the main database applications enabled by XML is the integration of data types. Data type refers to a set of data with values having predefined characteristics — for example, integers, floating points numbers, characters, strings, and pointers.

XML documents fall into two broad categories of data types: data-centric and document-centric. XML enables both data- and document-centric data types to be easily integrated in relational and object-oriented databases [11], where this was difficult before.

Most relational and object-oriented databases are built for storing *structured* (data-centric) data that are optimally stored in database tables (Ju, 1997). Structured data tends to exist primarily in databases and data files that are used by current and older enterprise databases (legacy systems). Data-centric documents commonly use XML for data transport and their physical structure is often unimportant (Bourret, 1999).

Content management systems have been primarily used for storing document-centric or u*nstructured* (document-centric) data, especially when documents had to be output in a human readable format (Bourret, 1999). Document-centric documents are

characterized by irregular structure and mixed content, and their physical structure is important.

Chart 1 characterizes the differences between data- and document-centric data types:

Chart 1 – Structured & Unstructured Data Types

Data type	Structure	Used for	Granularity	Examples
Data-centric	*Highly regular* structure which is constructed from known, regular sets of data	Storing, querying, analyzing, and manipulating data by different kinds of applications and for machine processing	Fine-grained — smallest independent unit of data at the level of a PCDATA-only element or an attribute	Data transport (e.g. sales orders, customer records and scientific data) Dynamic web pages constructed from known, regular sets of data
Document-centric	*Irregular* structure and mixed content (text and multimedia). The physical structure of the content is important	Searching in specialized ways specific to media types in order to produce human-readable dynamic documents	Entities and metadata are significant (as well as the structure of sibling elements and PCDATA) — smallest independent unit of data at the level of an element	User manuals, reports, email, graphics, images, audio, video

While legacy databases optimally store data-centric content, Finkelstein (1999) estimates that in most enterprises, structured data comprises only 10% of the data, information and knowledge resources of the business. The other 90% exists as unstructured data in textual documents, reports or email, or as graphics and images, or in audio or video formats that are not easily stored in the data-centric architecture of a database (Finkelstein, 1999). A key advantage of using XML as a data source is that XML enables both structured and unstructured data sources to be integrated (even in legacy systems), because it does not differentiate between data types. As discussed earlier, data in an XML document is just text, even if it represents another data type

(such as a date). Therefore, XML can be used in both data- and document-centric formats.

With structured documents, such as text or RTF files, XML can be used to describe data elements within an XML document. With unstructured documents, such as graphics, video and multimedia files, XML can be used to describe the file as a whole or its sub-elements, so it can be used with other documents.

New versions of enterprise databases and corporate portals/EPs are beginning to use XML to integrate both structured and unstructured data, for easy access to information throughout the enterprise (Hackathorn, 1999). XML is being used in databases so that both structured and unstructured data can be:

- Stored, queried, processed and retrieved more efficiently in large amounts (including Internet content);
- Published in more “dynamic” customizable formats based on client and user-specified parameters; and
- Distributed more easily between applications [12].

An example is Oracle8i, which has introduced content management-like “views” to retrieve both data- and document-centric content from the database for sharing with other applications (Ramalho, 2000). The underlying physical storage of the content is hidden from the end-user, and the appropriate view of content specific to the task at hand is delivered as an integrated XML document. These views effectively transform the structure of one or more underlying tables into a more meaningful structure for the demands of a specific application.

In summary, XML is blurring the distinction between structured and unstructured content by providing the ability to mix traditional data elements with free text in XML, and to encode data ranging from structured to unstructured. The result is a subtle but substantial shift in thinking about the formerly separate technologies of content management systems and databases, and thus what an “XML database” is.

Exchange of XML Data/Documents

Since XML provides a standard syntax for representing data, is text-based (Unicode), simple to create, easily parsed, and self-describing, XML is perceived to be a key enabling technology for the exchange of data on the Internet and within enterprises.

XML data can be:

- Transformed and rendered (with XML DTDs and XSL transformations) as desired using simple XSL specifications;
- Stored, retrieved, and exchanged without having to manage and interpret proprietary or incompatible data formats;
- Queried in sophisticated, dynamic ways;
- Presented as HTML pages with XSL stylesheets;
- Searched using XML-based query languages; and
- Used as a data-exchange or data-storage format.

XML can be created and republished on any platform, providing an unprecedented opportunity for application interoperation and data integration on the Internet. There is no need to convert XML data to other formats because XML focuses on the data and its context without tying it to specific formats, transfer storage systems, or network communication protocols.

It is not necessary to add individual extensions to XML, because XML is a metalanguage and extensions can be realized on the basis of XML. No prior knowledge of the sender application is required because the syntax of an XML document instance describes the relationships among the various elements — either explicitly via a DTD or implicitly by means of element context.

XML can be used in both object-oriented and relational-databases. XML encoding is compatible with the structure and semantics of both database models. This is because entities created in a DTD comprise a hierarchical structure. When parsed, these entities can be treated as either objects or tables in a database.

When storing data from an XML document in a database, the DTD can be used to validate its structure ensuring its data elements will map to the corresponding columns

in the database table. A table refers to a single result set (when transferring data from the database to XML) or a single table or updateable view (when transferring data from XML to the database). Where an XML document is generated by reading data from the database and constructed based upon a DTD, the resultant document is implicitly valid.

Lastly, XML is being used to make searching databases (and the Internet) more efficient. Databases have been used for years to augment HTML by querying and automatically generating data content on Web pages (Web databases). A problem is that data embedded within HTML pages needs to be preprocessed by special-purpose, page-specific parsers before meaningful queries can be posed (Widom, 1999). Further, queries are commonly limited to simple parameterized keyword-based searches that understand documents as streams of words. Only a small fraction of records containing the keywords or search phrase may be relevant, yet each record must be manually investigated to assess its content. With XML, not only can data be searched for, but also the context associated with the data can be added to the search.

In a nutshell, XML is ideally suited to data integration and data exchange. The structure and "meaning" of the data (at least to the extent that meaning can be embodied in tags), as well as the data itself, is readily parsable and available through multiple APIs, facilitating the use of powerful queries. Further, no one “owns” the XML specification, so XML’s functionality can be added to as needed (without waiting on a company to take action).

As a consequence, there has been a rush to integrate XML data in enterprise databases. The problem is that, without agreed upon DTDs, XML does nothing to support integration at the semantic level because the names and meanings of the tags used in XML documents are arbitrary (Levy, 1999). XML in itself (as a syntax only) only partially advances the prospects of data integration. There is a need for a DTD or schema for describing the content and capabilities of data sources.

Enterprises seem to have acknowledged the need to agree on descriptions that provide the semantic mapping between the data in the source and the relations in the database schema (Date, 2000).

As yet, there are no W3C Recommendations for describing XML data sources. However, in many enterprise sectors, DTDs and schemes are being established which enable and optimize the access and exchange of XML content. Examples are:

- XML-Data/XML-Data Reduced — As discussed earlier, XML-Data is a W3C Note (January 1998) based on a proposal that Microsoft and others submitted to the W3C. This schema proposal is used in Microsoft's BizTalk framework. XML-Data provides a large set of data types appropriate to database and data-centric content interchange. It also proposes a vocabulary for defining and documenting object classes that are strictly syntactic (for example, XML) or those that indicate concepts and relations among concepts (as used in relational databases and RDF). The former are called "syntactic schemas;" the latter "conceptual schemas."

- Document Content Description (DCD) — DCD is a W3C Note (July 1998) created in a joint effort between IBM and Microsoft. DCD uses some ideas from XML-Data and some syntax from the W3C RDF project described earlier [13].

- Schema for Object-Oriented XML (SOX) - SOX is a W3C Note (July 1999) developed by CommerceOne that provides functionality like inheritance to XML structures [14]. SOX has gone through multiple versions; the latest is SOX version 2.

- Document Description Markup Language (DDML) - DDML is another W3C Note (January 1999) developed on the XML-dev mailing list, which creates a schema language with a subset of DTD functionality [15].

- XML-Schema — As discussed earlier, XML-Schema is the W3C Working Draft that specifies a formal XML Schema definition language (W3C, 2000b). XML-Schema offers facilities for describing the structure and constraining the contents of XML documents.

  XML-Schema is divided into two documents: the W3C Schema Structures Document (April 2000) [16] and the W3C XML Schema Data-Type Document (April 2000) [17]. The XML-Schema Structures Document specifies ways to identify and handle data in the XML space, and the XML-Schema Data-Type Document addresses areas that are currently lacking in DTDs, including the definition and hierarchical validation of data types.

While a number of schemes and DTDs are now supported in XML parsers, St. Laurent (1999b) suggests that using the W3C XML-Schema is probably the safest long-term

solution. XML Schema supports namespaces and provides data-centric data types in addition to the more document-centric data types, making XML more suitable for data interchange applications. Built-in data types include strings, booleans, and time values, and the XML-Schema draft provides a mechanism for generating additional data types.

An appropriate query language (or set of languages) for XML also needs to be defined. An XML query language could provide improvements over current approaches to full-text searching described earlier. An XML query language could make it easier to process large collections of XML documents and extract relevant information. It might also increase the precision of searching in documents because queries could utilize the document's structure.

There is increasing interest in developing an XML-aware query language to take advantage of XML's data model, while enabling the kinds of applications that the Structured Query Language (SQL) provides for databases and the Object Query Language (OQL) provides for objects stored in an object-oriented database [18]. Both SQL and OQL are well established and standardized languages that are designed for retrieving relational and object-oriented data.

The XML Query Language (XQL) appears to be the main contender to standardize a query language that understands XML documents in the way that SQL understands a relational database.

XQL is a SQL-like notation for addressing and filtering the specific elements and text of XML documents. The basic constructs of XQL correspond directly to the basic structures of XML, and XQL is closely related to XPath, the common locator syntax used by XSL and XPointers [19]. XQL provides the ability to extract information from an XML dictionary, rather than using an ad hoc index.

The XQL proposal was submitted in September 1998 to the W3C and an XSL Working Group has been formed to develop the proposal.

In summary, XML breaks down the barriers between databases, document/content management systems and data types. XML has become a common ground for data integration and data interchange between heterogeneous databases. Standards, schemes and query languages are still being established to enable and optimize the access, integration and exchange of XML data/documents.

The next section provides an overview of products that are using XML with databases. It should be noted that new products are entering the market weekly, so the coverage is by no means exhaustive. Products have been gathered from Web sites, product reviews, XML Webzines, and other XML resource guides.

XML DATABASES

This section provides an overview of products that are using XML to process documents/data with databases.

What is an XML Database?

While it is clear that XML will become a major part of DBMS development, as yet there is no definitive answer to "What is an XML database"? At present, there appear to be two main approaches to storing and retrieving XML-based content:

1. A database (relational or object-oriented) to store XML data and middleware (built-in or third party) to perform data transfers between the database and XML document;

2. An XML server, which is an application that produces XML based upon an initial query of some sort (e.g. an e-commerce platform for building distributed system applications that use XML for data transfer). Typically, these are referred to as content management systems.

The common denominator with both approaches is that XML provides a bridge between structured and unstructured data in all classes of X-databases described below.

XML Database Solutions

Deployment of XML databases

Without conducting an extensive survey, it is difficult to determine the extent to which X-Databases are being deployed in enterprises. Online literature indicates that deployments to date are minimal, and that most XML database applications are using XML in the middle tier to integrate data from various back-end databases (Bourret,

2000). Nearly all X-Databases run under Windows and many are beginning to support multiple platforms.

Bourret (2000) identifies six primary classes of current X-Database solutions:

1. Middleware: Software called from an application that transfers data between XML documents and database
2. XML-enabled databases: Relational/object-oriented databases that transfer data between XML documents and the database;
3. XML servers: A platform for serving data in the form of XML documents to and from distributed applications. May also include a platform for building distributed applications that use XML for data transfer;
4. XML-enabled web servers: A Web server that serves XML — usually built from dynamic Web pages — to browsers “on the fly”;
5. Content management systems: A system for storing human-readable XML documents (document-centric data) and for managing fragments of documents.
6. Persistent DOM implementations: DOM implementations that use a database for speed and to avoid memory limits. These usually also support an XML query language.

In this paper, X-Database solutions are classified into four generic classes that focus more on how XML is integrated:

1. Object-relational databases with an added layer of XML support;
2. Native XML databases;
3. XML middleware components to perform data transfers; and
4. Content management systems.

Most databases in these classes have integrated XML servers, which means that the database has the ability to publish XML data/documents to a Web page.

Next, the paper takes a closer look at X-Database products in the four classes above, and it identifies issues in using XML with such products.

(1) Databases with relational database kernel + XML layer (XML ® Middleware ® Database)

The first class consists of XML-enabled databases (usually relational) that have an extended XML data transfer layer (middleware) for transferring data between XML documents and themselves.

A relational database consists of a set of tables, which in turn consist of records, which in turn consist of fields (W3C, 1997). A record is a set of fields and each field is a pair field-name/field-value (Date, 2000). An extended XML data transfer layer refers to some middleware software that transfers data between an XML document and the database.

These databases are generally designed to store and retrieve data-centric documents; however, most databases in this class can store document-centric documents in a single column and use text-processing extensions for queries (Kroenke, 2000). Databases approach storage of document-centric content as a single, intact object with its tags in a Character Large Object (CLOB) or Binary Large Object (BLOB). These objects can be thought of as searchable content "chunks".

XML documents are not stored in their native format. The database maps an XML document structure to a relational model, creating context to the data through tables, columns, joins, etc.

When an XML query is submitted to the database, the database translates the query with a transformation language — such as SQL or the XML Query Language. This transformation process is necessary in order for the database kernel to process the data and create a set of results. The results generally describe XML documents as fields in tree-like database tables (sometimes with tuples and attributes) [19].

It should be noted that this class of X-Database is not purely relational. Most of the databases listed below offer object-relational extensions for indexing, searching, and managing both data and documents. Accordingly, this class may be more properly referred to as object-relational DBMSs + an added XML layer.

Examples are Oracle's Oracle8i, Informix Internet Foundation.2000, IBM Universal DB2, and Microsoft's SQL Server.

Database	Description	DBMS Type	Platform(s)	Supports
Oracle8i	Data/application-integration server	Relational: with object-relational extensions	Commercial: Windows NT, UNIX, LINUX, SOLARIS, HP-UX, AIX, Sequent	SQL, PL/SQL, JDBC, CORBA, EJB, C, C++, SAX, DOM, XSLT, JavaBeans
Informix Internet Foundation.2000	Database server with XML-enabled Web server	Relational: with object-relational extensions	Commercial: UNIX, Windows	JDBC, SQLJ, SQL, ODBC
IBM DB2 Universal Database	Object-relational plug-in/extender for XML	Relational: with object-relational extensions	Commercial: Windows 95, Windows 98, Windows NT, Linux, OS/2, AIX, Solaris, HP-UX	Java, JDBC, SQLJ, OLE DB, LDAP
Microsoft SQL Server 2000	IIS ISAPI extension to IIS Web server with direct access to SQL Server	Relational: No XML integration yet. Preview to SQL Server 7.0 only	Commercial: Windows NT, Windows 2000	SQL, XSL

XML support in Oracle8i consists of three key components:

1. An Internet File System (iFS) that allows XML documents to be mapped to tables in the database, using custom configuration files to define how elements and attributes are mapped to tables and columns (Oracle, 1998). IFS is a Java application that resembles a content management system. It enables the end user to access the data stored in the iFS through a Web browser, FTP client, e-mail client or Windows Explorer — giving users "write once, read anywhere" access to data.
2. Next, Oracle8i provides “XML-Enabled Object Views Over Relational Data” (Oracle, 1998). As discussed earlier, this means that data in relational tables can be rendered dynamically as structured XML “views” of information in the database. XML content can be inserted into the database through these object views.

3. Lastly, Oracle8i has XML-enabled "section searching" in Oracle interMedia. Oracle interMedia can automatically index and search XML documents and document Fragments of any size up to 4 Gigabytes each (Oracle, 1998). This search functionality includes hierarchical element containership, data type discrimination, and searching on XML attributes. InterMedia also enables Oracle8i to manage multimedia content, including audio, video, text and location information for multimedia files and documents. It is designed to provide more precise searches over structured documents (Muench, 2000).

Middleware components — that actually allow one to serve XML (from the database) to a Web page — have to be “added on” to Oracle 8i. These components include the Oracle XSQL Servlet (which includes an XML parser and the XML SQL Utility for Java) and a Web server. These components are described in more detail below. They are not bundled with Oracle8i itself, and must be downloaded and installed separately.

XML support in Informix Internet Foundation.2000 (IIF.2000) consists of an XML DataPort, a new data type (XML Data) and an XML-enabled web server named DataBlade that can serve up XML and HTML via a SQL interface [20]. IIF.2000 offers the ability to install custom indexes and custom access methods. It also offers a “Virtual Table Interface” that provides a hierarchical table view of documents stored as BLOBs or CLOBs. The new XML Data type provides mapping between tree nodes and columns in tables, and IIF.2000 users can store information about an XML document's structure to provide efficient structure-based queries.

IBM DB2 Universal Database (UDB) has an object-relational plug-in for XML called an Extender [21]. The XML Extender can act as an XML DTD and support structural text searching with DB2 Text Extender. UDB offers the ability to store individual documents in a column (Xcolumn), or decompose documents into tables (Xcollection tables). The XML Extender also supports SQL queries that use XPath expressions, and it obtains document structure information from a Document Access Definition (DAD), an XML document that contains seven types of nodes (`root_node`, `element_node`, `attribute_node`, `text_node`, `namespace_node`, `comment_node`, and `processing_ instruction_node`). One creates the DAD to define the custom mapping between XML elements and attributes and database columns.

Microsoft SQL Server 7.0 does not yet provide integral support for XML. However, Microsoft has released a technology preview (the Microsoft SQL Server XML Technology Preview) of the next version of SQL Server (SQL Server 2000), code-named Shiloh [22]. The preview provides XML integration with and direct URL access to SQL Server 7.0 using Microsoft's web server, Internet Information Server (IIS). Essentially, the preview is an IIS ISAPI extension that provides HTTP access to SQL Server and XML data formatting and updating capabilities. This allows queries to be sent directly to SQL Server 7.0 via a URL with the results returned as XML formatted documents. The preview also enables "canned" queries, such as stored procedure calls, to be stored on an IIS server. Microsoft plans to incorporate the XML preview in SQL Server 2000, which is scheduled to be released to manufacture (RTM) in August 2000 [23].

The format of the XML returned by the Microsoft SQL Server preview can be customized (probably with XSL) and includes the ability to include schema information either in a DTD or XML-Data format [24]. The Microsoft ADO 2.5 (and ADO 2.1) engine produces the XML format. The resultant XML document is written in Microsoft's proprietary schema (Biztalk), which works by specifying the data types and similar characteristics of the schema (primary key, etc.) from the database and placing this content in the first half of the document. A similar process then extracts the data to place XML in row nodes.

An advantage of using X-Databases in this class is that in many cases the XML layer has been implemented as part of a product upgrade — for example, the upgrade from Oracle 8 to Oracle8i or from SQL Server 7.0 to SQL Server 2000. For corporations that have already invested heavily in relational databases and SQL-based technology, XML can be used to augment their DBMS without investing in a new DBMS and tools.

Issues:

To transfer data between an XML document and a database in this class, one must map the XML document structure to a relational database schema and vice versa. Relational data are flat, whereas XML is a tagged hierarchical representation (with nested structures and many levels). In this respect XML differs from relational schemas.

For example, in Oracle8i, XML documents may be stored as:

1. A single document with tags in a CLOB;
2. Data by decomposing it, and distributing the data untagged across multiple tables or columns; or
3. Fragments of XML documents as CLOBS and the rest of it as multiple tables.

These storage options create some limitations. Storing an XML document in one piece — for example, as a single column in a table such as a CLOB column — enables full-text indexing and content-based queries that return some information about the document. However, it is not possible to index or update individual elements. The database sees the XML document as a single <root> element, and sub-elements in the structure beneath that <root> element are ignored.

Decomposing the document into single elements indexed separately in multiple tables and columns enables document section searches that return specific content from documents. However, retrieval operations must then read all elements individually and join them together.

Another limitation is that relational database store an XML document untagged (i.e. without markup). The database discards the document name and DTD, entity declarations, CDATA sections, encoding, and the order in which attribute values and sibling elements occur. This is done because of the way in which binary data are stored (Bourret, 1999), and because the structure and encoding are not regarded as relevant if data are simply being transferred. A potential problem of discarding the documents markup and physical structure is “round-tripping” a document — that is, storing data from a document in the database and then reconstructing the document from that data — may result in a different document (Bourret, 1999).

A key issue is that the X-Databases in this class need additional XML middleware support components to bridge the Relational Database Management System (RDBMS) infrastructure. This means that some kind of data transfer middleware is needed to convert the data from text (in the XML document) to other types (in the database) and vice versa. The result is that data cannot be stored in the database without utilizing a query language like SQL or XQL, and content-specific database tools (which are often proprietary).

The query language is used to map the flat relational schema to XML structures. Then complex join operations must be performed (in order to make the XML data available in its original form). Finally, a content-specific processor is used to take relational data from the tables, transform it into XML, and then implement the structure conveyed by the markup in each XML document. This mapping and transformation process involves some administrative overhead — especially in setting up the multiple table joins (Harold, 2000).

The mapping process (from XML to a relational schema) also makes it difficult to store XML content with a unique structure. DTDs and schemes are stored as hierarchical data in an internal form using object-relational tables, and relational databases store, index, and construct joins upon only those tables that correspond to the schema of the table. Thus, if an XML document to be inserted into the database contained different data types or column lengths, it would be necessary to rewrite the database table(s) and update the relational schema to reflect the structure of the document.

There appear to some other scalability issues with this class of X-Database. Scalability refers to how well a system can adapt to increased demands. XML Parsers tend to put the entire XML document in memory (or its parsed DOM tree form), before data extraction begins [26]. If the XML data format is complex (perhaps even if there are only a few hierarchical levels), the process of mapping the XML structures to a relational schema may involve a heavy amount of parsing in a significant DBMS. This may degrade performance — for example, lead to unexpected application behavior, or lack of maintainability.

Lastly, it may be argued that the use of proprietary content-specific processing applications such as Oracle8i's XSQL Servlet does not promote the interoperability that underlies the XML specification. Vendors of native XML databases argue that it is better to use XML "natively" since it is a more open universal standard [27].

(2) Native XML Databases
(XML ® Database ® XML)

The second class is the "native" XML DBMS. A native XML database does not use translation routines to convert an XML document to another format. Nor is it necessary to map the database schema to the structure of an XML document. XML can be stored,

integrated, processed and served directly from the database through an information server.

These databases tend to use a different programming paradigm from the more traditional RDMS. They often allow for object-oriented queries that return XML classes as result sets.

Examples are Software AG's Tamino, Stanford's Lore, dbXML Group's dbXML, and DataChannel's DataChannel Server:

Database	Description	DBMS Type	Platform(s)	Supports
Tamino	Data/application-integration server	Hierarchical / Native XML. Relational and others through X-Node	Commercial: Windows NT, UNIX, LINUX, IBM OS/390	ODBC, JDBC, OLE DB. SQL
Lore	Native XML DBMS	Semi-structured Native XML	Open source	XML
DbXML core and Enterprise Server	Data management system	Native XML	Open source (GNU Public License): dbXML core Commercial: dbXML Enterprise Server — Linux, UNIX, Win32	ODBC/JDBC(TM)-like drivers, CORBA, and HTTP
DataChannel Server (DCS) 4.0	Enterprise Information Portal and data repository	Hierarchical / Native XML	Commercial: Windows NT, Sun Solaris	JDBC, HTTP

XML support in Tamino consists of an information server with an integrated XML-based DBMS that allows native XML document storage, integration and exchange of XML-data. Tamino is based on kernel technology (named X-Machine) that enables it to process XML natively (Software AG, 2000) [28].

Tamino is specifically designed to store, manage and process structured and unstructured data. It stores XML as objects without transformation to another data format and offers the same XML objects as output. In this sense Tamino is a database; however, Tamino can also manage content and process conventional data types — including mixed data types such as text, images, and sound.

Tamino is able to store well-formed documents and data not previously defined by a Tamino schema. This means that Tamino can handle unexpected changes in the format of a data stream — for example, the introduction of a new data type — and the data are processed directly on the embedded markup.

Interestingly, Tamino is also capable of managing relational data structures. It is equipped with its own SQL engine (SQL2) that, on the one hand, can be used to store tables embedded in XML objects. On the other hand, this SQL engine can be used to feed applications running on the same platform as Tamino that exclusively uses relational data schemes. This means that Tamino users can access existing external sources (like a relational database or even a Microsoft office product like Word or Excel) via X-Node through standard open interfaces: ODBC, JDBC and OLE DB. The result is that existing databases can be easily integrated with X-Node. Users can continue to work with existing (and even legacy) applications while accessing their data through Tamino and simultaneously integrating it into XML-objects. Tamino is customizable and allows programming of server extensions.

Stanford University's Lore, is a native XML DBMS that has been under development at Stanford University since 1995. Lore was originally developed for a data model named OEM, which is a similar concept to XML without the notion of DTDs and schemes. The DBMS prototype was extended to include full support for native XML. Lore decomposes XML data into its individual elements and attributes, storing a graph of the data physically on disk. In Lore, DTDs are not required; that is, the XML loaded into Lore need not conform to any predefined grammar or schema. Lore also includes a query language, multiple indexing techniques, a cost-based query optimizer, multi-user support, logging, and recovery (Goldman, 1997).

DbXML is a data management system designed specifically for large collections of XML documents [29]. This DBMS does not utilize a traditional RDBMS or object-oriented database management system (OODBMS). Instead, XML documents are organized as

tree structures. The dbXML core also provides some content management functionality, including document insertion, updates, deletions, triggers, stored procedures and object oriented business logic for XML documents. It also enables remote data access via ODBC/JDBC(TM)-like drivers, CORBA, and HTTP.

Finally, DataChannel Server (DCS). This product is described as an XML-enabled Enterprise Information Portal (EIP) for efficiently publishing, managing, and retrieving information [30]. DCS also provides XML-native support to store pure XML-tagged data as XML objects. The server has a hierarchical architecture named the Intelligent eXtensible Architecture (IXA), which can store metadata and content in “data, application, and access layers” [30]. DCS can also query the object store for specific XML objects and conduct full meta-content searches. DCS also includes an XQL query interpreter and HTTP addressable interface for remote data access.

Clearly, the big advantage of a native X-Database is that there is no need to convert data to and from XML using some extra middleware layer. There is no transformation process to map XML document structures to relational schemes, and there is no need to transform XML information into other physical data structures. Instead, XML data are stored in the existing structures and made available for further processing. Both data- and document-centric XML content (relational data, graphics, sound, video or plain text) can be processed based on its embedded markup. New XML tags can be introduced on the fly, and unexpected changes in the format of a data stream can be processed without having to update the database schema.

Issues:

One potential limitation may be the object-oriented (OO) design of many native XML databases. It has been suggested that the OO design may be inconsistent with the extensibility of XML (net.uniqueness, 1999). While object-oriented databases are well suited to present the structures of XML documents, the use of well bounded objects forces some association with a certain behavior of data within an object. Yet, XML markup may describe complex and multiple simultaneous relationships or behavior of data — necessitating objects to express the effect of relationships and behavior suited to each of the perspectives encoded in the markup (net.uniqueness, 1999). An example is document-centric data — this data type cannot be represented as a simple

object, which is inconsistent with the fundamental object-oriented design goal of encapsulation.

Lastly, while vendors like Software AG claim that products like Tamino can be integrated with existing databases, it is not clear how easy it will be to integrate Tamino's object-oriented application server with relational databases. Further research is needed to determine how Tamino integrates XML content stored in existing RDBMSs.

(3) XML Middleware Components (XML ® Database ® Middleware ® XML)

The third class of X-Database solutions is Middleware — the XML data transfer software that transfers data in the middle tier between XML documents (or some XML-enabled application) and a back-end database(s). It appears that most middleware is integrated with a Web server, especially if it is accessing a distributed/remote data source.

The following products are just a small sampling of the XML middleware. The examples include Oracle's XML SQL Utility for Java and XSQL Servlet, Stonebroom's ASP2XML, Beanstalk's Transparency, IBM's DatabaseDom, DBIx::XML_RDB, XML Software Corporation's InterAccess, ObjectDesign's eXcelon, Bourret's XML-DBMS, Manuel Lemos' Metabase, and Bluestone's Bluestone XML-Server.

Database	Description	DBMS Type	Platform(s)	Supports
XSQL Servlet (+ XML SQL Utility for Java)	Java Servlet and set of Java classes for transferring data between a relational database and an XML document	Relational (JDBC)	Commercial: all Java platforms	SQL, Java / JDBC
ASP2XML	COM object for transferring data between XML document and RDBMS	Relational (ODBC or OLE-DB)	Commercial: Windows NT	ASP

Beanstalk	Object-relational engine that generates XML from back-end relational databases (B2B orientation).	Object-relational (ODBC)	Commercial	Java/JDBC
DatabaseDom	JavaBean for transferring data between a DOM tree and a JDBC database	Relational (JDBC)	Commercial: all Java platforms	Java/JDBC
DBIx::XML_RDB	PERL module for transferring data between XML and DBI databases	Relational (DBI)	Open Source	Perl
InterAccess	Client/server package for accessing ODBC/OLE DB databases via the Internet	Relational (ODBC, OLE DB)	Commercial: Windows NT	SQL, ODBC, OLE DB
XML-DBMS	Java classes for transferring data between a RDBS and an XML document	Relational (JDBC)	Open Source	Java / JDBC, Perl
eXcelon	B2B Portal Server aka data/application-integration server with mid-tier development environment	Integrated object-relational DBMS (ODBC and OLE-DB) (ObjectStore)	Commercial: Win32	SQL, XQL, DOM, Java, COM, and server-side XSLT and XPath., ODBC, OLE-DB
Metabase	Set of PHP classes for database interactivity	Relational (SQL)	Open source	PHP, SQL, PostgreSQL, Mini-SQL
Bluestone XML-Server	B2B Information integration server	Relational (JDBC) and non-relational through Data Source Integration Modules	Commercial: all Java platforms	HTML, DHTML, XML, Java, JavaScript, ActiveX, VBScript, VRML, any Java class / applet

XSQL Servlet is a proprietary Java servlet that transfers data from a relational database to an XML document. The servlet is template-driven, which means that there is no predefined mapping between document structure and database structure (Bourret, 1999). Instead, retrieval commands are embedded in a template as <query> elements. (Bourret, 2000). The servlet provides support for passing query parameters through HTTP and for processing the output document through a Web server (Oracle, 1999).

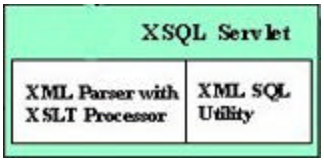
The concept behind the XSQL Servlet is that no programming is required to transfer XML data between a user and the database — the Java code in the servlet handles all XML processing.

As shown in Figure 1, the Servlet has two components:

- XML Parser for Java: An XML parser to process the SQL queries, query the database, obtain database results, and parse the XML plus, and an XSLT Processor — to transform the data into any format using XSL stylesheets; and

- XML SQL Utility: A set of Java classes to write data from XML input directly into a database table or view, and to pass a query to the database and generate an XML document (text or DOM) from the results. These classes may be used through one of the provided front ends or in a user-written (Java) application [31]. The product is model-driven, which means that a data model of the structure of the XML is created and mapped to the structures in the database (and vice versa) (Bourret, 1999). When transferring data from the database to XML, the user provides either a SELECT statement or a JDBC result set; and the results are returned as an XML document or DOM Document.

Figure 1



ASP2XML provides an interface between ODBC or OLE-DB compliant databases and any XML-enabled client. It is designed for use either with Microsoft Active Server Pages (ASP) scripts, or as a stand-alone COM-compliant ActiveX DLL component. The product is model-driven and the XML document is modeled as a single table. Data are transferred from the database to XML using a single SELECT statement and both the input and output contain ASP2XML-specific tags (that are required for processing).

BeanStalk 1.0 is an object-relational engine that can access any relational database or other ODBC data source and provide SQL queries via Java/JDBC. Query results are output as an XML document using a tree-like object model, where nesting in the result sets translates to nesting in the XML document. The user can specify whether to return results as elements or attributes, as well as the names of those elements and attributes. Beanstalk claims to have a patented (OLAP) message-based technology that can process large or complicated queries faster than a conventional relational database, even while accessing the same data over ODBC.

DatabaseDom is a combination of Java JDBC, IBM Data Access Bean and DOM programming. An XML template file defines the database and XML structure. A JavaBean reads this, and creates XML from the results of a database query, and also updates the database based on a new or modified XML structure [32].

DBIx::XML_RDB creates XML data from DBI data sources. It allows one to extract data from a database, and manipulate later using XML::Parser [33].

InterAccess enables access to any ODBC/OLE DB compliant SQL database via the Internet remotely without the need to have any ODBC drivers on the client. All database access is done via the InterAccess server using Microsoft ADO (the ODBC/OLEDB drivers are located there). The server will store and retrieve data as XML, or receive data in XML format from the client [34].

XML-DBMS is a set of Java packages for transferring data between XML documents and relational databases. It views the XML document as a tree of objects in which element types are generally viewed as classes and attributes, and PCDATA as properties of those classes. It then uses an object-relational mapping to map these objects to the database. An XML-based mapping language is used to define the view and map it to the database (Bourret, 2000).

eXcelon is a midtier, XML-based data-integration server that utilizes an object-oriented database called ObjectStore. eXcelon's goal is to manage, distribute, and cache large amounts of data- and document-centric data in the middle tier. The ObjectStore eXcelon data sever stores XML in ObjectStore. XML is parsed and then stored in ObjectStore in its parsed format; that is, its building elements are stored as individual objects for performance and reuse of XML elements. It is possible to add new elements to any node on the fly. EXcelon can integrate structured, semi-structured and unstructured data in spreadsheets, COBOL files and Web pages.

eXcelon caches XML and provides a query engine for accessing subsets of XML. It also handles transactions and synchronization with multiple back-end data sources. The cache is synchronized with all connected databases. A limiting factor is that eXcelon cannot write changes made by a client back to a relational database backend — only to its own data cache. EXcelon also features a complete development environment including an editor, graphical tools and an in-built browser.

Metabase is a set of classes for the PHP scripting language that provide DBMS independent access and management of databases using XML. The package contains a set of functions that call the selected DBMS driver objects functions, and a parser class that can interpret DBMS independent database schema file defined in a custom XML format. This class enables the database structure and contents to be discarded in Metabase's XML format parser, and thus gives the ability to move data between databases of different DBMS vendors.

Bluestone XML-Server is an XML-based information integration platform for deploying B2B data interchange. Bluestone automates the data exchange process by providing a platform to build integration objects that represent applications and data sources. The server has the ability to dynamically generate, interpret and receive XML documents from distributed applications. The application development environment appears to include, *inter alia*, Visual-XML (facilitates dynamic XML document management), XML-Contact (a contact management application for the Palm Pilot) and XwingML (an application to build XML documents that define the complete Java Swing Graphical User Interface).

The benefit of products in this class is that they make it easier to integrate XML with a variety of databases (especially legacy DBMSs) in the back-end; perhaps making it easier develop a scalable database system that supports XML.

Apart from the plethora of choices in this X-Database class, the only issue appears to be that most middleware products only support a specific type or range of database connectivity.

(4) Content Management Systems

(XML ® Content / Documents ® XML)

Content management systems are specifically designed for storing, retrieving, and assembling documents from document fragments (content) (Bourret, 2000). They generally include such features as editors, version control, and multi-user access. Most use a back-end database, although this is generally transparent to the user.

Examples of content management systems that support XML are Chrystal Software's Astoria, Inso's DynaBase, and Eidon's XMLBase:

Content Management System	Description	Type	Platform(s)
Astoria	Document Management System for technical publications	Based on ObjectStore, ObjectDesign's object-oriented database	Commercial: Windows NT, Win 95 / 98, UNIX, Sun Solaris, LINUX, IBM OS/390
DynaBase	Content repository for enterprise collaboration + integrated Web server	Based on ObjectStore	Commercial: Windows NT 4.0, Win 95 / 98, Sun Solaris 2.6 Macintosh OS 8.1

Astoria is a content management solution for enterprises producing technical publications and other kinds of structured documents. XML support in Astoria consists of new media and structured file formats for both SGML and XML.

Dynabase combines XML-enabled content management capabilities with an integrated Web server. It is designed for use by distributed work groups throughout an enterprise,

enabling co-workers to collaborate directly on Web site projects over the Internet and intranets.

Issues

Products in this class are designed to manage XML documents regardless of format or source. Most products have a “content repository” that stores collections of documents, which can include multimedia and document files and the individual data structures within them — allowing that data to be indexed, searched, and extracted on demand.

The real benefit of using XML-enabled content management systems is the ability to efficiently gather, manage and dynamically deliver vast quantities of semi-structured and unstructured document-centric content from a back-end database on a wide array of Web sites and other electronic information-sharing mechanisms. Obviously, these products are not database solutions in themselves; they are more oriented toward XML document management and delivery.

The next section presents a simple implementation of an X-Database in Oracle8i using the middleware Java servlet, XSQL Servlet.

IMPLEMENTATION – RETRIEVING & STORING XML IN ORACLE8I

A complete description of how XML is used with above-mentioned X-Database products is beyond the scope of this paper. Instead, this section provides proof-of-concept of technical integration issues in using: (1) Oracle8i, which falls into the first class of X-Database products above; and (2) XSQL Servlet, middleware that transforms XML into Oracle8i's RDBMS.

The main questions of interest were:

- How easy or difficult is it to implement XML in Oracle8i?
- How well do the typical document structures of XML information objects map to the storage and transmission framework of relational tables and columns?

The implementation involved developing a simple XML application in Oracle8i for managing employee records. The object was to store data in the database and then to expose that data as XML on a Web page using XSQL pages.

The steps involved in installing Oracle8i, designing the database schema, mapping the XML document structure to the database schema, and the process to retrieve XML from the database, are described in detail below.

Oracle8i - Installation and Process Overview

Installing Oracle8i was a non-trivial task in itself. Attempts were made to install Oracle8i on Red Hat Linux 6.1 and Microsoft Windows NT 4.0.

In a typical installation, an initial database is installed by default. During the initial database installation, the Oracle Universal installer set up some initial listeners under the SID of ORCL — without any kind of warning/notification. The listener is an HTTP daemon (httpd) that listens for requests for database service.

At an earlier point in the installation, a global database name and SID were entered for the Oracle8i database server, creating a home of LCD and SID of LCD. The database server was also configured to use a TCP/IP listener connection named LCD.

After the server was installed, the Database Creation Assistant automatically created a database under Oracle8i. Several permission and connection errors occurred during the database creation process. Subsequently, when the Oracle8i Net8 utility was used to test the listener, there was no response from the database. It appears that there was a conflict between the initial listener set up under the ORCL SID and the LCD SID.

After several unsuccessful attempts at creating a database in Oracle8i Personal Edition, Oracle8i (Enterprise Edition 8.1.6) was finally installed on an NT Workstation 4.0 machine. All default settings used in a “typical” installation were accepted. The initial database was installed accepting all default settings. The global database name and SID “ORCL” were used.

For the successful installation of the Oracle XSQL Servlet, additional software needed to be installed, including:

- An external Java Virtual Machine – version 1.1.8 of the Java Development Kit (JDK) [35];

- The Java Servlet Development Kit (JSDK) – version 2.0, which consists of a self-extracting and installing executable;

- A Web Server that supports Java servlets to serve up the XML pages – Apache 1.3.12. The Web server is needed to invoke the XSQL Servlet with an XSQL page, and then deliver the results to a Web browser. Apache was configured as an NT service so that it would start automatically in Windows NT.

- The Jserv module for Apache, so that it can run the Oracle XSQL servlet – Jserv 1.1.

The Java Virtual Machine was installed first, then JSDK 2.0, then Apache 1.3.12. A directory alias was configured in the Apache configuration to access XML-related pages as though they were part of the Apache home directory structure. In the httpd.conf file, the following line was added to set the DocumentRoot and Directory to “c:\xsql”:

```
Alias /xsql/ “C:/xsql/”
```

Jserv 1.1 was installed accepting all defaults. The result was a Web Server that supports both Java servlets and an Oracle8I database.

The last step was to install the XSQL Servlet. The XSQL Servlet executable includes JDBC drivers, XML parser for Java and the XML SQL Utility classes in one downloadable package from Oracle. The XSQL Servlet file (the distribution) was extracted to the root directory (C:\). When the distribution extracted in WinZip, it created subdirectory named “xsql” (C:\xsql).

Next, the Jserv module’s configuration was edited to utilize the new Java classes installed earlier. This involved editing C:\Program Files\Apache Jserv 1.1\conf\jserv.properties and inserting the following lines:

```
wrapper.classpath=C:\xsql\lib\oraclexsql.jar
wrapper.classpath=C:\xsql\lib\classes111.zip
wrapper.classpath=C:\xsql\lib\xmlparserv2.jar
wrapper.classpath=C:\xsql\lib\oraclexmlsql.jar
wrapper.classpath=C:\xsql\lib
```

These wrapper.classpath directives were added after existing directives in jserv.properties.

Next, jserv.conf (in the same path as jserv.properties) was edited to add ApJServAction lines for xsql as follows:

```
ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

Finally the $ORACLE_HOME\xsql\lib\XSQLConfig.xml file, which contains information about database connections, was edited to add connection information for the employee records database using the global database name and SID “ORCL”.

Note: The XSQL Servlet installation was tested by running the xsql.bat file and then checking whether or not the demo connections on the default page worked at http:\\localhost:7070\xsql\index.html.

Database Design

Step 1 - Designing Database Schema

With Oracle8i and its additional components installed, the next step was to create a database schema for storing employee in the database. The following DTD was developed (Figure 3 - emp.dtd):

Figure 3 – emp.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!--PURPOSE: This DTD describes pages containing employee records-->
<!--============================================================-->
<!ELEMENT EMPLIST (EMP)*>
<!ELEMENT EMP (EMPNO, ENAME, JOB, MGR, HIREDATE, SAL, COMM?,
DEPTNO)>
<!ELEMENT EMPNO (#PCDATA)>
<!ELEMENT ENAME (#PCDATA)>
<!ELEMENT JOB (#PCDATA)>
<!ELEMENT MGR (#PCDATA)>
<!ELEMENT HIREDATE (#PCDATA)>
<!ELEMENT SAL (#PCDATA)>
<!ELEMENT COMM (#PCDATA)>
<!ELEMENT DEPTNO (#PCDATA)>
<!--============================================================-->
```

Note that <EMP> is defined as the root element, or basic structural element. Within <EMP> tag, are child elements that define more granular aspects of an employee record.

Step 2 - Mapping XML document structure to database schema

The next step was to map the DTD, which represents the structure of XML documents, to a database schema. At this point, it was necessary to look at the data types to be stored.

As discussed above, Oracle8i stores XML documents as:

1. Structured data (employee records data) in one location decomposed within database tables — data-centric approach; and/or
2. Related unstructured data (for example, employee reviews) within a CLOB — document-centric approach.

Given the structured nature of the data and the limitations of the second CLOB storage model discussed above (i.e. the granularity of metadata are the complete document), the first storage model was used. The DTD was mapped directly to tables in the database.

The DTD in Figure 3 was mapped to a database schema by creating a single EMP table. SQL was used to create the tables in Oracle8i's text editor, PL*SQL. PL*SQL enables the creation of database objects at the command line from an sql> prompt. The following SQL code shows the structures of the database tables:

```
create table emp (
  empno number primary key,
  ename varchar2(30) not null,
  job varchar2(30),
  mgr number,
  hiredate date,
  sal   number,
  comm   number
  deptno number not null,
);
```

This table, EMP, was created using a direct copy of a similarly named table owned by the user ORACLE (by utilizing the SQL SELECT command). The table was thus populated with sample employee records from a table installed by default during the Oracle8i installation.

Database to XML: Retrieving XML

Step 3 - Retrieve XML from Database using a Web Form

This step involved designing and building a Web page to retrieve a list of employee records from the Oracle8i database. The XSQL Servlet was used to take the source XML and transform it into a viewable HTML page.

Here are the steps followed to retrieve a list of all employees from the database arranged by job type, together with details of the employee who has the highest salary:

1. Created an XML Page named emp.xsql (Figure 4). Emp.xsql is simply a well-formed XML document with the extension .xsql. The XSU allows one to provide either a

SELECT statement or a JDBC result set. This particular XML Page defines a set of queries to retrieve all employees by job type (JOB) and highest salary (SAL) as a SELECT (SQL) statement.

Figure 4 – emp.xsql

```
<?xml version="1.0"?>
<page connection="demo" xmlns:xsql="urn:oracle-xsql">
<xsql:include-request-params/>
<xsql:query  find="%"
             sort="JOB"
   null-indicator="yes" >

       SELECT *
         FROM EMP
       WHERE JOB LIKE UPPER('%{@find}%')
      ORDER BY {@sort}

</xsql:query>
<xsql:query sort="ENAME"
   null-indicator="yes" >

       SELECT ENAME, EMPNO, SAL
         FROM EMP
       WHERE SAL IN (SELECT MAX (SAL) FROM EMP)
      ORDER BY {@sort}

</xsql:query>
</page>
```

This page conforms to the basic structure of an XSQL Page as follows:

- The file starts with the standard XML header: <?xml version="1.0" ?>
- It has an optional XSL stylesheet tag: <?xml-stylesheet type="text/xml" href="emp.xsl"?> to specify the format of the XML in the Web browser (or on some other client device)
- The <xsql:query> tag identifies the components needed to complete the query, including the root document element (EMP), table row element (JOB), and the actual query. The query is regular <SQL>. The tag also identifies a namespace for defining the xsql keyword and tells the XSQL Servlet to use the (predefined) database connection named demo. Oracle's Java, C & C++ XML parsers all have integrated support for such namespaces.

2. Used an XSL stylesheet named emp.xsl (provided by Oracle as a demo in the XSQL installation), to format the XML documents that represent the query results. The stylesheet, emp.xsl, references another stylesheet, rowcol.xsl, which was installed by default in the path `../common/rowcol.xsl`.

3. In a Web browser (Internet Explorer 5.0), entered a URL that points to the XSQL Page emp.xsql (http://localhost:7070/xsql/emp.xsql). The .xsql extension indicated to the Web server to use the XSQL Servlet to handle this request.

4. The XSQL Servlet passed the emp.xsql to the XML Parser for Java (in the form of a string). Emp.xsql was then parsed and processed by the XML Parser and XSLT Processor. The Servlet found all elements with an <xsql:query> tag, and submitted all SELECT statements in the <xsql:query> tags as SQL statements to the underlying XML SQL Utility (XSU), which then passed the queries to the Oracle8i database via JDBC (when processed, these query tags were replaced by the result of the queries).

5. The database used the information contained in the <connection> attribute to log on. The attribute `connection="demo"` defines where to find the information needed to connect to the database. The XSQL Servlet uses the configuration file, XSQLConnections.xml. This file can have multiple entries where each one represents a separate database connection. The default database connection entry, demo, in the XSQLConnections.xml file was defined as follows:

```
<?xml version="l.O" ?>
<connectiondefs dumpallowed="no">
      <connection name="demo">
            <username>scott</username>
            <password>tiger</password>
            <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
            <driver>oracle.jdbc.driver.OracleDriver</driver>
      </connection>
</connectiondefs>
```

   Note that the default user is SCOTT (with TIGER as the password).

   Database returned the query results to the XSU, which wrapped the data in its schema creating the XML result returned to the Servlet. The Servlet formatted the query results, and then passed the completed document back to the Web browser.

6. The Web server returned a canonical document, containing a list of records. The browser then received the response to the requested emp.xsql page in HTML format. Figure 5 shows the results:

Figure 5

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7788	SCOTT	ANALYST	7566	4/19/1987 0:0:0	3000		20
7902	FORD	ANALYST	7566	12/3/1981 0:0:0	3000		20
7369	SMITH	CLERK	7902	12/17/1980 0:0:0	800		20
7876	ADAMS	CLERK	7788	5/23/1987 0:0:0	1100		20
7934	MILLER	CLERK	7782	1/23/1982 0:0:0	1300		10
7900	JAMES	CLERK	7698	12/3/1981 0:0:0	950		30
7566	JONES	MANAGER	7839	4/2/1981 0:0:0	2975		20
7782	CLARK	MANAGER	7839	6/9/1981 0:0:0	2450		10
7698	BLAKE	MANAGER	7839	5/1/1981 0:0:0	2850		30
7839	KING	PRESIDENT		11/17/1981 0:0:0	5000		10
7499	ALLEN	SALESMAN	7698	2/20/1981 0:0:0	1600	300	30
7654	MARTIN	SALESMAN	7698	9/28/1981 0:0:0	1250	1400	30
7844	TURNER	SALESMAN	7698	9/8/1981 0:0:0	1500	0	30
7521	WARD	SALESMAN	7698	2/22/1981 0:0:0	1250	500	30

ENAME	EMPNO	SAL
KING	7839	5000

The top table in Figure 5 shows the results of the query to retrieve all employees by job type. The bottom table shows the results of the query to select the employee with the highest salary.

This example demonstrates how the XSQL Servlet generates XML from data sets. The XSQL Servlet, separate middleware software, is entirely responsible for transforming the XML document with a stylesheet. It is the XSQL Servlet that handles the tasks of establishing a connection to the database, submitting the query, and transforming and formatting the results into HTML (or some other format). The Web Server merely sends the transformed document back to the web browser.

If the line declaring the stylesheet emp.xsl were not included in emp.xsql, the XSQL Servlet would return the following XML document (Figure 6):

Figure 6

```
<?xml version = '1.0'?>
<page>
<request><parameters/><session><wtgbid>cd92uajt</wtgbid></session><
cookies/></request>
<ROWSET><ROW
num="1"><EMPNO>7876</EMPNO><ENAME>ADAMS</ENAME><JOB>CLERK</JOB><MGR
>7788</MGR><HIREDATE>5/23/1987 0:0:0</HIREDATE><SAL>1100</SAL><COMM
NULL="YES"/><DEPTNO>20</DEPTNO></ROW><ROW
num="2"><EMPNO>7499</EMPNO><ENAME>ALLEN</ENAME><JOB>SALESMAN</JOB><
MGR>7698</MGR><HIREDATE>2/20/1981
0:0:0</HIREDATE><SAL>1600</SAL><COMM>300</COMM><DEPTNO>30</DEPTNO><
/ROW><ROW
num="3"><EMPNO>7698</EMPNO><ENAME>BLAKE</ENAME><JOB>MANAGER</JOB><M
GR>7839</MGR><HIREDATE>5/1/1981
0:0:0</HIREDATE><SAL>2850</SAL><COMM
NULL="YES"/><DEPTNO>30</DEPTNO></ROW><ROW
num="4"><EMPNO>7782</EMPNO><ENAME>CLARK</ENAME><JOB>MANAGER</JOB><M
GR>7839</MGR><HIREDATE>6/9/1981
0:0:0</HIREDATE><SAL>2450</SAL><COMM
NULL="YES"/><DEPTNO>10</DEPTNO></ROW><ROW
num="5"><EMPNO>7902</EMPNO><ENAME>FORD</ENAME><JOB>ANALYST</JOB><MG
R>7566</MGR><HIREDATE>12/3/1981
0:0:0</HIREDATE><SAL>3000</SAL><COMM
NULL="YES"/><DEPTNO>20</DEPTNO></ROW><ROW
num="6"><EMPNO>7900</EMPNO><ENAME>JAMES</ENAME><JOB>CLERK</JOB><MGR
>7698</MGR><HIREDATE>12/3/1981 0:0:0</HIREDATE><SAL>950</SAL><COMM
NULL="YES"/><DEPTNO>30</DEPTNO></ROW><ROW
num="7"><EMPNO>7566</EMPNO><ENAME>JONES</ENAME><JOB>MANAGER</JOB><M
GR>7839</MGR><HIREDATE>4/2/1981
0:0:0</HIREDATE><SAL>2975</SAL><COMM
NULL="YES"/><DEPTNO>20</DEPTNO></ROW><ROW
num="8"><EMPNO>7839</EMPNO><ENAME>KING</ENAME><JOB>PRESIDENT</JOB><
MGR NULL="YES"/><HIREDATE>11/17/1981
0:0:0</HIREDATE><SAL>5000</SAL><COMM
NULL="YES"/><DEPTNO>10</DEPTNO></ROW><ROW
num="9"><EMPNO>7654</EMPNO><ENAME>MARTIN</ENAME><JOB>SALESMAN</JOB>
<MGR>7698</MGR><HIREDATE>9/28/1981
0:0:0</HIREDATE><SAL>1250</SAL><COMM>1400</COMM><DEPTNO>30</DEPTNO>
</ROW><ROW
num="10"><EMPNO>7934</EMPNO><ENAME>MILLER</ENAME><JOB>CLERK</JOB><M
GR>7782</MGR><HIREDATE>1/23/1982
0:0:0</HIREDATE><SAL>1300</SAL><COMM
NULL="YES"/><DEPTNO>10</DEPTNO></ROW><ROW
num="11"><EMPNO>7788</EMPNO><ENAME>SCOTT</ENAME><JOB>ANALYST</JOB><
MGR>7566</MGR><HIREDATE>4/19/1987
0:0:0</HIREDATE><SAL>3000</SAL><COMM
NULL="YES"/><DEPTNO>20</DEPTNO></ROW><ROW
num="12"><EMPNO>7369</EMPNO><ENAME>SMITH</ENAME><JOB>CLERK</JOB><MG
R>7902</MGR><HIREDATE>12/17/1980
0:0:0</HIREDATE><SAL>800</SAL><COMM
NULL="YES"/><DEPTNO>20</DEPTNO></ROW><ROW
num="13"><EMPNO>7844</EMPNO><ENAME>TURNER</ENAME><JOB>SALESMAN</JOB
><MGR>7698</MGR><HIREDATE>9/8/1981
0:0:0</HIREDATE><SAL>1500</SAL><COMM>0</COMM><DEPTNO>30</DEPTNO></R
```

```
OW><ROW
num="14"><EMPNO>7521</EMPNO><ENAME>WARD</ENAME><JOB>SALESMAN</JOB><
MGR>7698</MGR><HIREDATE>2/22/1981
0:0:0</HIREDATE><SAL>1250</SAL><COMM>500</COMM><DEPTNO>30</DEPTNO><
/ROW></ROWSET>
<ROWSET><ROW
num="1"><ENAME>KING</ENAME><EMPNO>7839</EMPNO><SAL>5000</SAL></ROW>
</ROWSET>
</page>
```

As displayed above, the result is an XML document containing data from the rows. In a nutshell, the mapping can be described as follows:

- Database table names → Document element (EMP) → ROWSET tag.
- Database column names → Top-level elements/tag names → ROW tag.
- Scalar columns → Elements with text-only content (ENAME) → Elements nested within the ROW node.

The output XML document contained no CDATA or entity usage, and the order in which sibling elements and attributes appear is the order in which the data was returned by the database. The database discarded all encoding information, including the document's name and DTD, and the order in which attribute values and sibling elements occurred.

Issues

In this implementation, XML data are transferred to and from database tables as fragments of decomposed XML documents — rather than tables specifically designed to model XML documents.

Unlike a content management system that views a set of XML documents as exactly that — a set of documents — the Oracle8i iFS treats each document as a database "load file," where all markup is discarded and document contents are merged into a single database table. The iFS stores only those XML documents that correspond to the schema of a database table. DTDs and schemes are integrated as hierarchical data in an internal form using object-relational tables.

The markup is discarded because the source XML document must be transformed into the precise physical structure needed for automatic insertion into the relational database tables. Bourret (1999) says this is acceptable because of the way in which

binary data are stored, and because the structure and encoding may not be relevant if the data are simply being transferred.

Organizing XML data in this tabular or “graph” fashion seems to work when it is more important to access the data itself (rather than the actual entities and encodings used within XML documents). However, it may be more difficult to store XML data that follows a unique schema (Widom, 2000). New elements or attributes cannot be added to the DTD to reflect new data types or extend existing ones without manually changing the database schema — for example, to add a new table for a new top level element.

This limitation may confine the extensibility of the DBMS later on in the lifecycle of the system, especially if the database schema needs to expand to include new data types or information objects that cannot be easily mapped to the storage and transmission framework of a relational database. For example, it might be difficult to integrate emerging industry standard DTDs/schemes or industry vocabularies into the database in the future. This is becoming a significant issue as industry- and domain-specific DTDs and schemes evolve. Retrofitting a schema onto a relational database schema is likely to be difficult if there are numerous variations in structure.

The clear answer to the question posed earlier — how well do XML document structures map to the storage and transmission framework of Oracle8i — is, in the author’s opinion, not well. It seems the XSQL Servlet really does nothing more than add another layer of complexity and overhead to the Oracle8i database.

It is definitely a drawback that the meaning and context of the data relies totally on further processing applications like the XSQL Servlet, and it may be argued that this may limit the future of X-Database technology in this product class.

Another limiting factor is that the database server has to make a new database connection for every HTTP request that requires database access. The browser submits the form request via HTTP. Then the XSU component of the XSQL Servlet has to convert all the input name/value elements into XML and the XSLT processor has to apply the XSL sylesheet to map that internal format into the database table (or view). With a large complex database the overall system may not be very scalable, and response time may be slowed by all the administrative overhead of establishing new database connections.

Given the less than straightforward implementation of Oracle8i and the XSQL Servlet, and the need to map XML documents to the underlying object-relational storage, it may be concluded that Oracle8i augments, rather than integrates, XML to the Oracle 8i RDBMS.

CONCLUSIONS & FUTURE CHALLENGES

XML presents fundamental changes to the ways organizations think about and utilize data and metadata. XML separates data from its presentation, so it is easy to change the look and feel of data simply by changing the XSL stylesheet. Since the data for the site can be stored in a database, it also has the advantage of being more easily maintained.

There are significant gains to be made in data integration and interchange from XML's open design. The XML model offers the opportunity to build flexible, interoperable XML databases that support both structured and unstructured data. It can extract information that provides context and semantics. It is portable over existing Internet protocols, and across distributed systems and operating systems. It provides the ability to converge and map data between disparate systems. These capabilities provide an unprecedented opportunity to exchange data between different database platforms (and Internet applications).

XML may also mitigate the complexity of developing custom database applications for the Internet. It's rich data structures and metadata enable users to select, specify, and manipulate different views of the same data. Accompanying technology like XSL can be used to render the same data on different internet-enabled devices like handheld devices.

The future challenges in integrating XML with databases are to:

1. Use XML for manipulating data without imposing an inflexible data structure or relationship expressed in a schema or DTD.

   The kinds of data structures that appear occur in XML are richer than in relational data. As the implementation demonstrates, RDBMS architectures like Oracle8i generally require data to be much more rigidly structured than XML. A database

schema must be designed carefully before any data is loaded. This assumes that a particular structure or patterns pre-exists within the XML data.

This approach makes it difficult to manage data that does not adhere to predefined table structures, and if the structure of the data changes in any way, then the database schema must be modified as well. Further, while object-oriented databases do not require rigid table structures, they still rely heavily on predefined, constant schemes.

There is no doubt that there is a need for some kind of DTD or schema within an X-Database (despite the fact that the XML specification does not require the structure of data to be fixed ahead of time). Without a structure for the tag and attributes patterns in a database, it could be difficult for users to formulate meaningful queries. The execution engine also needs some understanding of a database's structure in order to process a query efficiently.

However, this need for accompanying structural rules must be offset against the need to make X-Databases generalized and adaptable enough to respond — to XML records written in the database — by reading, writing, querying, and generally processing them in sync with the markup. The relationship between XML's optional DTDs and traditional database schemes must be extensible in order to reflect changes in markup.

2. Find more efficient ways to structure and tag document-centric data in relational databases — from one or more tables as a hierarchical XML document. Relational databases have no problem presenting structured XML (data-centric) data. Since most transactional data like invoices is data-centric, it is likely there will continue to be a market for relational databases that support XML.

3. Create efficient means to publish relational data/documents as XML without proprietary translation routines for querying XML content.

The integration of data storage and retrieval components into *native* XML databases promises to provide more seamless transparent integration of XML data types. Enterprise database vendors have acknowledged this. Oracle says that it plans to integrate storage and retrieval components natively into the Oracle Internet platform in the future (Wiseth, 2000).

Further research is needed to characterize and compare the performance of native X-Database alternatives. Vendors like Software AG claim that native XML storage is more scaleable; that it serves XML faster and more efficiently than relational databases. This research should evaluate how native databases handle large volumes of XML files, and how they scale up in serving native XML to clients from a number of data sources.

It seems likely that future X-Databases will use a declarative XML query language (and query processor) — perhaps XQL — that will be as important to XML applications as SQL has been to traditional enterprise database applications. As more and more data is exchanged and ultimately stored in XML, users and applications alike will benefit from the ability to query X-Databases.

In conclusion, it seems likely that future X-Databases will evolve in a way that reflects XML itself — they should be freely extensible. In other words, X-Databases should be defined and manipulated by XML markup, and be cast in a document structure that reflects (and adapts to changes in) an XML schema or DTD. They should have the ability to accept input data, and then maintain and present it for each user in a structure suited to that user's role and understanding of the data. Lastly, they should also support industry-wide standards (such as W3C Recommendations) and domain-specific vocabularies.

NOTES

[1] North, K. (1999). Modeling, Metadata, and XML (Web Techniques, June 1999) [Online]. Available: http://www.webtechniques.com/archives/1999/06/data. (August 7, 2000).

[2] XML Linking Language (XLink). W3C Working Draft 21-February-2000. Available: http://www.w3.org/TR/2000/WD-xlink-20000221/. (August 7, 2000).

[3] XML Pointer Language (XPointer) Version 1.0. W3C Candidate Recommendation 7 June 2000. Available: http://www.w3.org/TR/2000/CR-xptr-20000607. (August 7, 2000).

[4] The XML Specification 1.0 defines a Declaration for XML, which is fixed for all instances. The default attribute of the Declaration is `version="1.0"` and `encoding="UTF-8"`. An XML version of the specified DTD must be accessible to the XML processor, either locally or via the network (e.g. an external URL specified by a System Identifier such as `<!DOCTYPE advert SYSTEM "http://www.ils.unc.edu:7070/emp.dtd">`.

[5] Reimers, B. D. (2000). Consolidate Your Data in Seven Steps – How to Gain a Consolidated View [online]. Available: http://www.planetit.com/techcenters/docs/Database/Product/PIT19990125S0010. [August 7, 1999].

[6] Oracle Corporation. (2000). ORACLE ENHANCES INTERNET PLATFORM WITH XML SUPPORT [Online]. Available: http://www.uk.oracle.com/info/news/nov98xmi.html. (August 7, 2000).

[7] Murray, G. (1999). The Portal is the Desktop. Framingham, MA: International Data Corporation. This article is available from the Group Computing web site at http://www.groupcomputing.com/Issues/1999/MayJune1999/mayjune1999.html.

[8] XML Query Language (XQL) Proposal [Online]. Available: http://www.w3.org/Style/XSL/Group/1998/09/XQL-proposal.html. (August 7, 2000).

[9] XML-Data W3C Note 05 Jan 1998 [Online]. Available: http://www.w3.org/TR/1998/NOTE-XML-data/. (August 7, 2000).

[10] Simple Object Access Protocol (SOAP) 1.1 W3C Note 08 May 2000 [Online]. Available: http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. (August 7, 2000).

[11] An object-oriented database is a system offering DBMS facilities in an object-oriented programming environment. Data is stored as objects, and can be interpreted using methods specified by its class. The relationship between similar objects is preserved (inheritance) as are references between objects (Harrington, J. L. (2000). Object-Oriented Database Design Clearly Explained. Ap Professional.)

[12] Stanek, W. R. (1998). Structuring Data with XML [Online]. Available: http://www.zdnet.com/pcmag/pctech/content/17/10/tf1710.001.html. (August 7, 2000).

[13] Document Content Description for XML - Submission to the World Wide Web Consortium 31 July 1998 [Online]. Available: http://www.w3.org/TR/NOTE-dcd.

[14] Schema for Object-Oriented XML 2.0. W3C Note 30 July 1999 [Online]. Available: http://www.w3.org/TR/NOTE-SOX/.

[15] Document Definition Markup Language (DDML) Specification, Version 1.0 - W3C Note, 19 January 1999 [Online]. Available: http://www.w3.org/TR/NOTE-ddml].

[16] XML Schema Part 1: Structures. W3C Working Draft 7 April 2000 [Online]. Available: http://www.w3.org/TR/xmlschema-1/.

[17] XML Schema Part 2: Datatypes. W3C Working Draft 07 April 2000 [Online]. Available: http://www.w3.org/TR/xmlschema-2/.

[18] Rein, L. (1999). The Quest For An XML Query Standard [Online]. Available: http://www.xml.com/pub/1999/03/quest/index.html. (August 7, 2000).

[19] Robie, J. The Design of XQL [Online]. Available: http://www.texcel.no/whitepapers/xql-design.html (August 7, 2000).

[20] Informix. (2000). Informix Internet Foundation.2000: The Internet Future Comes Back to Informix [Online]. Available: http://www.informix.com/informix/whitepapers/aberdeen/iif_aberdeen.htm (August 8, 2000).

[21] IBM. (2000). DB2 Universal Database Version 7.1 [Online]. Available: http://www-4.ibm.com/software/data/db2/udb/v7/beta/. (August 8, 2000).

[22] Microsoft Corporation. Microsoft SQL Server XML Technology Preview [Online]. Available: http://msdn.microsoft.com/workshop/xml/articles/xmlsql/sqlxml_prev.asp. (August 8, 2000). Ricadela, A. (1999). Microsoft Outlines SQL Server 7 Upgrade Plans [Online]. InformationWeek Online - Monday, December 13, 1999. Available: http://www.informationweek.com/story/IWK19991213S0006. (August 8, 2000).

[23] Glascock, R. (2000). SQL Server 2000 Will RTM Next Week [Online]: Available: http://www.techweb.com/wire/story/TWB20000727S0007. (August 8, 2000).

[24] Ressler, J. (2000). SQL Server Insider: Developer Q&A [Online]. Available: http://microsoft.com/sql/techinfo/insiderwebfeatures2K.htm. (August 8, 2000).

[25] Trupin, J. (2000). SQL Server 2000: New XML Features Streamline Web-centric App Development [Online]. Available: http://msdn.microsoft.com/msdnmag/issues/0300/sql/sql.asp. (August 8, 2000).

[26] This is the case with Microsoft Internet Explorer 5.0's XML Parser. It should be noted that some Java parsers can process an XML document incrementally. Nevertheless, this is still an issue when there are a large amount (e.g. thousands) of upper-tier XML nodes/elements.

[27] Software AG. (2000). Why We Need XML Server Technology [Online]. Available: http://www.softwareag.co.uk/xml/mail1.html (August 8, 2000).

[28] Software AG. (2000). About Tamino – The Power Database for the Internet [Online]. Available: http://www.softwareag.com/tamino/product/strategy.htm. (August 8, 2000).

[29] dbXML.org. (2000). What is dbXML? [Online]. Available: core is a data management system designed specifically for large collections of XML documents. (August 8, 2000).

[30] DataChannel. (2000). DataChannelServer (DCS) 4.0 [Online]. Available: http://www.datachannel.com/products/. (August 8, 2000).

[31] Compare to an object-oriented database like Tamino, where the structure of the XML document would be mapped to the structure of the object database.

[32] IBM. (2000). DatabaseDom [Online]. Available: http://www.alphaworks.ibm.com/tech/databasedom. (August 8, 2000).

[33] Sergeant, M. (2000). DBIx-XML_RDB-0.03 [Online]. Available: http://theoryx5.uwinnipeg.ca/CPAN/data/DBIx-XML_RDB/XML_RDB.html. (August 8, 2000).

[34] XML Software Foundation. (2000). InterAccess [Online]. Available: http://www.xmlsoft.com.au/iaccess.html. (August 8, 2000).

[35] Although all of the Java components needed by XSQL Servlet are available within the Oracle8i installation, an external Java Virtual Machine (version 1.1.8) had to be installed to successfully run XSQL Servlet. The Oracle Universal Installer installs version 1.1.7 of the JDK (by default at C:\Program Files\Oracle\jre). Initially, this path was used as the Java environment variable for XSQL Servlet. XSQL Servlet would run using this environment variable, but it would not successfully connect to a database. The solution was to separately install version 1.1.8, and then to modify the appropriate values to reference this external JVM (installed by default at C:\jdk1.1.8).

REFERENCES

- Birbeck, M., Kay M., Livingstone, S., Mohr, S. F., Pinnock, J., Loesgen, B., Livingston, S., Martin, D., Ozu, N., Seabourne, M., Baliles, D. (2000). Professional XML - 1st edition. Birmingham, UK ; Chicago, US: Wrox Press Inc.

- Bourret, R. (1999). XML and Databases [Online]. Available: http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm (August 7, 2000).

- Bourret, R. (2000). XML-DBMS – Middleware for transferring Data between XML Documents and Relational Databases [Online]. Available: http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xmldbms/xmldbms.htm (undated).

- Chang, B., Scardina, M., Karun, K., Kiritzov, S., Macky, I., Novoselsky, A., Ramakrishnan, N. (2000). Oracle XML Handbook. Berkeley, California, U.S.: McGraw-Hill.

- Date, C.J. (2000). An Introduction to Database Systems – Seventh Edition. New York New York: Addison Wesley Longman.

- Finkelstein, C., Aiken, P., and Zachman, J. (1999). Building Corporate Portals with XML (Enterprise Computing). Sydney, Australia: McGraw-Hill.

- Goldman, R., McHugh, J., Abiteboul, S., Quass, D., and Widom, J. Lore: A Database Management System for Semistructured Data. SIGMOD Record, 26(3):54-66, September 1997.

- Hackathron, Richard D. (1999). Web Farming for the Data Warehouse. San Francisco, California: Morgan Kaufmann Publishers.

- Harold, E. R. (1999). XML Bible. Foster City, California: IDG Books Worldwide.

- Harold, E. R. (2000). The Advantages of XML for Database Integration [Online]. Available: http://metalab.unc.edu/xml/slides/sd2000west/xmlandjava/. (August 8,

2000).

- Henry, A. (1999). XML On Your Net [Online]. Available: http://www.nwfusion.com/buzz99/buzzxml.html. (August 8, 2000).

- Ju, P. (1997). Databases on the Web – Designing and Programming for Network Access. New York, New York: M&T Books.

- Kroenke, D. M. (2000). Database Processing: Fundamentals, Design, and Implementation. Upper Saddle River, New Jersey: Prentice-Hall.

- Muench, S. (2000). Using XML and Relational Databases for Internet Applications [Online]. Available: http://technet.oracle.com/tech/xml/info/htdocs/relational/index.htm#ID795. (August 8, 2000).

- Net.uniqueness. (1999). What is an XML Database? [Online]. Available: http://www.uniquness.net/whitepaper.html. (August 8, 2000).

- Ogbuji, U. (1999). Practice XML [Online]. Available: http://www.cnn.com/TECH/computing/9910/01/practical.xml.idg/index.html. (August 8, 2000).

- Oracle Corporation. (1998). Oracle Technical White Paper: XML Support in Oracle8i and Beyond [Online]. Available: http://technet.oracle.com/tech/xml/info/htdocs/xml_twp.html (August 8, 2000).

- Oracle. (1999). Using XML in Oracle Database Applications [Online]. Available: http://technet.oracle.com/tech/xml/info/index2.htm?Info&htdocs/otnwp/xml_custom_presentation.htm (August 8, 2000).

- Ramalho, J. (2000). Learn Oracle8i, Plano, Texas: Wordware Publishing.

- Robie J., Lapp J., and Schach D. XML Query Language (XQL), in Proceedings of QL '98: The Query Languages Workshop [Online]. Available: http://www.w3.org/TandS/

QL/QL98/. (August 7, 2000).

- Simpson, J.E. (1999). Just XML. PTR, New Jersey: Prentice Hall.
- St. Laurent, S. (1999a). XML: A Primer - Second Edition. Foster City, California: M&T Books, IDG Books Worldwide, Inc.
- St. Laurent, S. (1999b). Describing Your Data: DTDs and XML Schemas [Online]. Available: http://www.xml.com/pub/1999/12/dtd/. (August 7, 2000).
- The XML FAQ. (2000). Frequently Asked Questions about the Extensible Markup Language [Online]. Available: http://www.ucc.ie/xml/. (August 7, 2000).
- World Wide Web Consortium [W3C]. (1997). XML Representation of a Relational Database [Online]. Available: http://www.w3.org/XML/RDB.html. (August 7, 2000).
- W3C. (1998a). Extensible Markup Language (XML) 1.0 [Online]. Available: http://www.w3.org/TR/REC-xml. (August 7, 2000).
- W3C. (1998b). Document Object Model (DOM) Level 1 Specification [Online]. Available: http://www.w3.org/TR/REC-DOM-Level-1. (August 7, 2000).
- W3C. (1999a). Namespaces in XML – W3C Recommendation 14 January, 1999 [Online]. Available: http://www.w3.org/TR/REC-xml-names. (August 7, 2000).
- W3C. (1999b). Associating Style Sheets with XML documents Version 1.0 - W3C Recommendation 29 June, 1999 [Online]. Available: http://www.w3.org/1999/06/REC-xml-stylesheet-19990629/. (August 7, 2000).
- W3C. (1999c). XSL Transformations (XSLT) Version 1.0 – W3C Recommendation 16 November, 1999 [Online]. Available: http://www.w3.org/TR/1999/REC-xslt-19991116. (August 7, 2000).
- W3C. (2000a). Extensible Stylesheet Language (XSL) Version 1.0 - W3C Working Draft 27 March 2000 [Online]. Available: http://www.w3.org/TR/xsl/. (August 7,

2000).

- W3C. (2000b). XML Schema - W3C Working Draft [Online]. Available: http://www.w3.org/TR/xmlschema-1/. (August 7, 2000).

- W3C. (2000c). Resource Description Framework (RDF) [Online]. Available: http://www.w3.org/RDF/. (August 7, 2000).

- Widom, J. (1999). Data Management for XML - Research Directions. IEEE Data Engineering Bulletin, Special Issue on XML, 22(3):44-52. (September 1999).

- Wiseth, K. (2000). Industry Standard Introduction to XML [Online]. Available: http://www.oracle.com/oramag/oracle/00-Jan/10ind.html. (August 8, 2000).