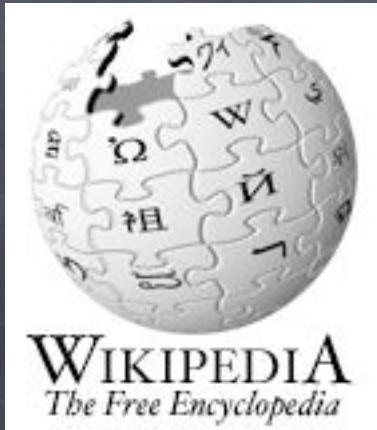


# Chalk Talk

Xml and Databases

Stephen Maderak



XML (Extensible Markup Language) is a W3C recommendation for creating special-purpose markup languages. It is a simplified subset of SGML, capable of describing many different kinds of data. Its primary purpose is to facilitate the sharing of structured text and information across the Internet. Languages based on XML (for example, RDF, SMIL, MathML, XSIL and SVG) are themselves described in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form.

## Pros

- compatibility with web and internet protocols
- simultaneously human- and machine-readable format
- the ability to represent general data structures (records, lists and trees)
- the format is self-documenting (meta data)
- hierarchical structure suitable for most (but not all) types of document
- plain text files, unencumbered by licenses or restrictions
- platform-independent, and so relatively immune to changes in technology

## Cons

- XML syntax is fairly verbose and partially redundant. This can hurt application efficiency. It can also make XML difficult to apply in cases where bandwidth is limited.
- XML still often requires further parsing to extract individual values.
- Modelling overlapping (non-hierarchical) data structures requires extra effort.
- Mapping XML to the object oriented or relational paradigms may be cumbersome..

## Table of Contents:

- 1.0 Introduction
- 2.0 Is XML a Database?
- 3.0 Why Use a Database?
- 4.0 Data versus Documents
  - 4.1 Data-Centric Documents
  - 4.2 Document-Centric Documents
  - 4.3 Data, Documents, and Databases
- 5.0 Storing and Retrieving Data
  - 5.1 Mapping Document Schemas to Database Schemas
    - 5.1.1 Table-Based Mapping
    - 5.1.2 Object-Relational Mapping
  - 5.2 Query Languages
    - 5.2.1 Template-Based Query Languages
    - 5.2.2 SQL-Based Query Languages
    - 5.2.3 XML Query Languages
  - 5.3 Storing Data in a Native XML Database
  - 5.4 Data Types, Null Values, Character Sets, and All That Stuff
    - 5.4.1 Data Types
    - 5.4.2 Binary Data
    - 5.4.3 Null Data
    - 5.4.4 Character Sets
    - 5.4.5 Processing Instructions and Comments
    - 5.4.6 Storing Markup
  - 5.5 Generating XML Schemas from Relational Schemas and Vice Versa
- 6.0 Storing and Retrieving Documents
  - 6.1 Storing Documents in the File System
  - 6.2 Storing Documents in BLOBs
  - 6.3 Native XML Databases
    - 6.3.1 What is a Native XML Database?
    - 6.3.2 Native XML Database Architectures
      - 6.3.2.1 Text-Based Native XML Databases
      - 6.3.2.2 Model-Based Native XML Databases
    - 6.3.3 Features of Native XML Databases
      - 6.3.3.1 Document Collections
      - 6.3.3.2 Query Languages
      - 6.3.3.3 Updates and Deletes
      - 6.3.3.4 Transactions, Locking, and Concurrency
      - 6.3.3.5 Application Programming Interfaces (APIs)
      - 6.3.3.6 Round-Tripping
      - 6.3.3.7 Remote Data
      - 6.3.3.8 Indexes
      - 6.3.3.9 External Entity Storage
    - 6.3.4 Normalization, Referential Integrity, and Scalability
      - 6.3.4.1 Normalization
      - 6.3.4.2 Referential Integrity
      - 6.3.4.3 Scalability
  - 6.4 Persistent DOMs (PDOMs)
  - 6.5 Content Management Systems
- 7.0 XML Database Products
- 8.0 Additional Links
- 9.0 Comments and Feedback

## 2.0 Is XML a Database?

A more useful question to ask is whether XML and its surrounding technologies constitute a "database" in the looser sense of the term -- that is, a database management system (DBMS). The answer to this question is, "Sort of." On the plus side, XML provides many of the things found in databases: storage (...), schemas (...), query languages (...), programming interfaces (...), and so on. On the minus side, it lacks many of the things found in real databases: efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, queries across multiple documents, and so on.

Thus, while it may be possible to use an XML document or documents as a database in environments with small amounts of data, few users, and modest performance requirements, this will fail in most production environments, which have many users, strict data integrity requirements, and the need for good performance.

## 5.2 Query Languages

Many products transfer data directly according to the model on which they are built. Because the structure of the XML document is often different from the structure of the database, these products often include or are used with XSLT...

The long term solution to this problem is the implementation of query languages that return XML...

...most of these languages rely on SELECT statements embedded in templates. This situation is expected to change when XQuery and SQL/XML are finalized, as major database vendors are already working on implementations.

SQL-based query languages use modified SELECT statements, the results of which are transformed to XML. A number of proprietary SQL-based languages are currently available. The simplest of these uses nested SELECT statements, which are transformed directly to nested XML according to the object-relational mapping.

Unlike template-based query languages and SQL-based query languages, which can only be used with relational databases, XML query languages can be used over any XML document. To use these with relational databases, the data in the database must be modeled as XML, thereby allowing queries over virtual XML documents.

## 5.3 Storing Data in a Native XML Database

It is also possible to store data in XML documents in a native XML database. There are several reasons to do this. The first of these is when your data is semi-structured. That is, it has a regular structure, but that structure varies enough that mapping it to a relational database results in either a large number of columns with null values (which wastes space) or a large number of tables (which is inefficient). Although semi-structured data can be stored in object-oriented and hierarchical databases, you can also choose to store it in a native XML database in the form of an XML document.

The obvious drawback in this case is that the increased speed applies only when retrieving data in the order it is stored on disk. If you want to retrieve a different view of the data, such as a list of customers and the sales orders that apply to them, performance will probably be worse than in a relational database. Thus, storing data in a native XML database for performance reasons applies only when one view of the data predominates in your application.

One problem with storing data in a native XML database is that most native XML databases can only return the data as XML. (A few support the binding of elements or attributes to application variables.) If your application needs the data in another format (which is likely), it must parse the XML before it can use the data.

## 5.5 Generating XML Schemas from Relational Schemas and Vice Versa

To generate a relational schema from an XML schema:

1. For each complex element type, create a table and a primary key column.
2. For each element type with mixed content, create a separate table in which to store the PCDATA, linked to the parent table through the parent table's primary key.
3. For each single-valued attribute of that element type, and for each singly-occurring simple child element, create a column in that table. If the XML schema has data type information, then set the data type of the column to the corresponding type. Otherwise, set the data type to a pre-determined type, such as CLOB or VARCHAR(255). If the child element type or attribute is optional, make the column nullable.
4. For each multi-valued attribute and for each multiply-occurring simple child element, create a separate table to store values, linked to the parent table through the parent table's primary key.
5. For each complex child element, link the parent element type's table to the child element type's table with the parent table's primary key.

To generate an XML schema from a relational schema:

1. For each table, create an element type.
2. For each data (non-key) column in that table, as well as for the primary key column(s), add an attribute to the element type or a PCDATA-only child element to its content model.
3. For each table to which the primary key is exported, add a child element to the content model and process the table recursively.
4. For each foreign key, add a child element to the content model and process the foreign key table recursively.

## References:

<http://www.rpbouret.com/xml/XMLAndDatabases.htm>

<http://en.wikipedia.org/wiki/Xml>

Other web searches (i.e. Oracle and xml)

For more information read the document at the first link.