

A comparison of data storage formats for the Voyager Learning Management System

Research report

**prepared by
Kim Hagen-Hall**

for 157.746 XML Databases and the Semantic Web

Abstract

This report compares potential data formats for Voyager, a web-based prototype Learning Management System based on the IMS Learning Design Specification. The report investigates the suitability of XML files, native XML databases, XML-enabled databases and relational databases.

While a native XML format would make importing and exporting IMS Learning Design packages simpler, data that is mapped to fields in an XML-enabled or relational database is likely to be more suitable for playing Learning Designs. While using a relational model to store IMS Learning Designs results in more a large number of related tables and joining tables, and hence a large number of joins, it also results in significantly less data redundancy than XML's hierarchical model. Considerations such as DBMS availability and cost are considered.

Given the varying data access methods used by Voyager, benchmark tests would be needed to provide a definitive answer as to the most efficient storage method. The most efficient method may be a hybrid solution, with some data stored in native XML format and some stored in relational format.

1. Introduction

As part of an ongoing research project for my Masters degree, I developed Voyager, a prototype web-based Learning Management System. Voyager is based on the IMS Learning Design Specification (IMS Global, 2003), an XML-based standard for sharing learning activities and resources. Although IMS Learning Design packages are therefore XML files (Tattersall, Manderveld, Hummel et al., 2004), Voyager was initially implemented using MySQL, a relational database (Hagen, Hibbert and Kinshuk, 2006).

This report examines that decision, comparing the commonly-used options for storing XML data and discussing whether a relational database is the best choice for the Voyager application, based on the comparative advantages discussed in the literature.

2. Literature Review

2.1. About Voyager

Voyager was designed to implement the IMS Learning Design Specification as closely as possible (Hagen et al., 2006). It includes both a Learning Design editor and a Learning Design player: the editor allows educators to import, store and create IMS Learning Designs and export IMS Learning Design packages; and the player presents the learning designs for learners to "play" (Hagen et al, 2006).

One of the key purposes of the IMS Learning Design Specification ("the Specification") is to enable educators to share and reuse learning designs (Koper and Olivier, 2004). Voyager must therefore be able to add downloaded or received Learning Design packages to its repository and create Learning Design packages

which comply with the Specification, which can then be published or sent to other educators.

Although IMS Learning Design packages are represented as XML files (Tattersall et al., 2004), the developers initially chose to implement Voyager using MySQL, a relational database. The developers chose a database-driven system over XML files “because it offers more efficient searching, the ability to tune performance and better and more easily administered security” (Hagen et al., 2006). However, given the need to import source data from XML files and export data to XML files, storage in XML format might be more efficient.

Kappel, Kapsammer and Retschitzegger (2004) suggest three main alternatives for storing XML data:

1. in an XML database;
2. in an object-oriented or object-relational database; or
3. in a relational database.

These options, and their comparative advantages, are discussed below.

2.2. XML Databases

XML data can be stored in XML files, in an XML-enabled database, or in a Native XML database.

2.2.1. Storing data in XML Documents

A database is defined as “an organised collection of logically related data” (Hoffer, Prescott & McFadden, 2007). A collection of XML files in a directory on the server can therefore validly be considered as a database. This is a simple solution and is technology-independent: it does not require an additional database management system to be present on the server.

Data stored in XML documents can be accessed through XML Query Engines or Wrappers. XML Query Engines are standalone engines that can query XML documents. Wrappers are software applications which treat XML documents as a source of relational data so enable SQL queries of XML documents (Bourret, 2005).

Document-based storage is easy to implement (Vakali, Catania, & Maddalena, 2005) and technology-independent. Bourret (2005) notes that it is possible to store XML data in documents “in environments with small amounts of data, few users, and modest performance requirements”. However he notes that it is not suitable for most production environments, “which have many users, strict data integrity requirements, and the need for good performance”. It is “verbose and access to the data is slow due to parsing and text conversion”, and lacks features typically found in DBMSes such as “efficient storage, indexes, security, transactions and data integrity, multi-user access, triggers, [and] queries across multiple documents”.

In most cases, therefore, it will be more suitable to store data in a database, rather than in files.

2.2.2. XML-enabled databases

Several popular commercial database systems, including Oracle, Microsoft SQL Server, and DB2, now include support for XML (Kappel et al., 2004). These “XML-enabled” DBMSs provide interfaces for transforming XML data to the internal data model (Vakali et al., 2005), which might be relational or object-relational. The XML:DB initiative (n.d.) defines an XML-Enabled database as “a database that has an added XML mapping layer ... [which] manages the storage and retrieval of XML data”. Data that is mapped into the database is mapped into application specific formats. For example, the XML solution from Oracle maps the XML data into an object-relational format (ref), and the solution in Microsoft SQL Server 2000 (onwards) maps XML data into a relational format.

Some “XML-enabled” features in these DBMSs involve simply storing an entire XML document in a field in the database - Oracle and DB2 have this option as well as actually mapping XML data to their underlying data model. This method is not discussed here, and is considered further in section 2.3.

The XML-enabled databases which map XML data to their internal data model exploit several DBMS features such as scalability, concurrency control, and recovery services (Vakali et al.), and also allows integration with existing relational data (Kappel et al., 2004). On the other hand, this approach factorizes XML content in several structures, which can make processing less efficient (Vakali et al., 2005), and in some solutions the original XML meta-data and structure may be lost (XML:DB Initiative, n.d.).

Vakali et al. (2005) and Nambiar et al. (2002) recommend that data-centric applications, which capture structured data such as a product catalogue (Chaudhri, Rashid & Zikari, 2003), use an XML-enabled DBMSs rather than a Native XML DBMS, because of the data management support provided.

When benchmarking a range of XML-enabled and Native XML databases, Nambiar et al. (2002) found that XML-enabled relational database system performs better than Native XML databases when processing simple relational queries, such as selecting a specific record using a number or string, but Native XML databases perform better when processing navigational and document queries.

2.2.3. Native XML databases

Native XML databases are databases designed especially to store XML documents. Their internal model is based on XML, rather than another model such as the relational model (Bourret, 2005). The commonly-used definition is from the XML:DB initiative (n.d.), who define a native XML database as one that:

a) defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according to that model;

b) has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage; and

c) is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files.

Bourret (2005) comments that this last point is analogous to stating that the physical storage format used by a relational database is unrelated to whether that database is relational. In eXist and Tamino, for example, “XML documents are stored either in the internal native XML database or an external RDBMS environment, which can also manage no XML documents” (Vakali et al., 2005)

Commercial databases such as Tamino support the features expected of other database systems such as transactions, security, multi-user access, programmatic APIs, query languages (Bourret, 2005) and indexing (Chaudhri, Rashid & Zikari, 2003). However Chaudhri et al. note that locking is often at the level of entire documents, rather than at the level of individual nodes, reducing multi-user concurrency.

The XML format may be more suitable than other formats for storing semi-structured data, data whose structure is unknown, or data which has a regular structure but the structure varies significantly, such as biological or financial data. This last category can result in a lot of columns with null values or a large number of tables in a relational database. Native XML databases handle schema changes more easily than relational databases (Bourret, 2005), so are useful for schemata which are evolving (Vakali, 2005).

If XML documents will be regularly received and/or created, native XML databases will import and export data more quickly than other options as they do not need overhead time for mappings, integration and joins between XML and other internal structures such as tables (Vakali et al., 2005).

Data reliability is an important consideration. Native tools also preserve document order, which might be lost in conversion to another format (Nambiar et al., 2002, Schoning, 2001). Converting to and from XML and other storage models can also result in mismatches between the original XML structure hierarchy and the resulting table (Vakali et al., 2005), and comments and processing instructions that might have been in the original document tags are lost (Schoning, 2001).

The relational model does not adapt well to data with complex nesting (Nambiar et al., 2002), although this can be handled by object-relational and object-oriented databases.

Data retrieval in a native XML database may be faster if the whole XML document is retrieved and if the database stores the document together physically or uses physical (rather than logical) pointers between parts of the document. This allows the documents to be retrieved either without joins or with physical joins, both of which are faster than the logical joins used by relational databases (Bourret, 2005). Vakali et al. (2005) recommend that Native XML data stores are a good choice of document-centric applications, which store data such as emails or articles (Chaudhri et al., 2003), and often have a less regular data structure. Nambiar et al. (2002) found that native XML implementations typically processed navigational and document queries more efficiently than XML-enabled databases.

While XML database technology can provide some clear advantages over the relational model, it is not yet mature. The relational model still provides some advantages over this emerging technology.

2.3. Relational Databases

Kappel et al. (2004) prefer the option of storing XML data to a relational database because much existing data is already in relational databases, as this option allows the XML data to be integrated with existing relational data. Relational databases are scalable, reliable and easy to implement (Vakali et al., 2005).

There are 3 primary ways in which XML data can be stored in a relational database (Kappel et al., 2004):

1. XML documents can be loaded into a field in a database table. In this case XML data is not integrated with existing data, and only one user can update the document at a time, as the entire field will be locked. However it does facilitate the inclusion of documents whose structure is not known in advance.
2. XML documents can be “shredded” or “decomposed” - each line of an XML document, including its tags, is loaded into a row in a database table. This also allows the inclusion of documents whose structure is not known in advance, but still does not integrate the XML data with existing data and queries may be cumbersome, needing several joins to reconstruct the document.
3. XML data is mapped to a corresponding relational attribute. In this case the structure of the data must be known in advance, but it allows XML data to be integrated with relational data.

The first two options do not allow data to be integrated with existing data and do not exploit the efficient updating and query facilities of relational databases. The third option does exploit these capabilities thoroughly, but mapping between the XML and Relational models can be difficult (Kappel et al., 2004) and the resulting schema may require many joins to reassemble the data (Vakali et al., 2005).

In order to transfer between XML documents and a database (whether relational, object-oriented or hierarchical) the structure of the document must be known in advance (Bourret, 2005, Schoning, 2001). Mapping logic must either be hard coded in import-export programmes or stored in a mapping tool (Kappel et al., 2004). Hard coding is inflexible (Kappel et al., 2004), whereas the use of middleware mapping tools adds an additional layer of complexity.

Bourret (2005) also notes that ““round-tripping” a document - storing the data from a document in the database and then reconstructing the document from that data - often results in a different document”. Whether this is a problem depends on the needs of the application.

However relational databases “have efficient storage and retrieval techniques and can evaluate queries using various indexing mechanisms” (Nambiar et al., 2002) and there are also disadvantages to storing data in the hierarchical XML format which would make a relational model more suitable.

While it may be quicker to retrieve hierarchical data in the order in which it was stored, if you want to retrieve a different view of the data performance will “probably be worse than in a relational database” (Bourret, 2005). This may not be a problem if the application typically uses only one view of the data.

Hierarchies can lead to redundancy (Obsanjo, 2004) – for example, a customer’s shipping address may be stored in each order. Obsanjo also notes that in the real world there are often multiple groups to which a datum belongs, which often cannot be modelled with a single hierarchy.

Finally, Obsanjo (2004) also notes that data is tightly coupled in a hierarchy – deleting a customer record may delete their entire order history.

2.4. Object-Oriented and Object-Relational Databases

Object-oriented and object-relational models are a candidate for storing XML data because they support nested database objects, and therefore allow XML elements to be mapped directly to corresponding nested database objects, simplifying data mapping and preserving the structure of the original XML data. They describe sequential and hierarchical relationships more easily than relational databases (Yen, Huang and Ku, 2002). However all of the other problems with mapping data that were discussed above still apply to object-oriented and object-relational databases.

3. Comparing the Options for Voyager

Given all of the advantages and disadvantages discussed, what would be the best data storage model for Voyager?

Storing Voyager’s data in XML format would allow easy inclusion of received and downloaded Learning Design packages, and simple production of new Learning Design packages.

Storing data in the form of XML files is an option, because this is the form that external packages will be received in. This would be useful because it is simple to implement and technology-independent, so the system can be moved easily to a different server.

However while XML files would be suitable for a proof-of-concept system, the literature suggests that it would not be suitable for classroom evaluations which will involve multiple concurrent users, and where slow performance would effect participants’ perceptions of the system’s usability. The literature suggests that a database would therefore be more suitable.

An XML-enabled database which stores data in a field does not allow multiple users, and shredded documents do not provide good performance, so data would therefore need to be stored either in an XML-enabled database which maps XML data to its internal model, in a Native XML database, or be mapped to fields in a relational database.

As IMS Learning Design packages comply with the pre-defined Specification, the schema is known in advance and they have a regular structure. However the elements included can vary significantly between packages – for example, some may

contain a simple list of activities to be done while others contain complex multi-user “plays” with multiple role parts with different activities, and several “Acts” (like a play in a theatre). In Voyager’s relational database these optional elements were mapped to related tables, so if they are not present, no row is created. However mapping these recurring elements and nested elements to related tables resulted in a large number of tables in the database and a large number of joining tables, reducing query and update performance (Hagen et al., 2006). A Native XML database would not need the large number of joins required in the relational database, so imports, exports, updates and queries could well be quicker. Alternatively, an object-oriented or object-relational database would also preserve nested elements.

Document order is preserved by Voyager’s import process by storing a sequence number with each element which could occur more than once within a document, such as activities. However this increases the overhead when importing or exporting documents. It also relies on the programmer getting the code right.

Meta data about IMS Learning Designs is stored in meta-data elements (IMS Global, 2003), and these are imported into Voyager’s relational database, preserving comments and instructions which are properly documented using these tags. However if authors use other XML features to record meta-data, this will be lost during the current import process.

Educators will use the Learning Design Editor in Voyager to create and update Learning Designs, so the platform must allow new documents to be created and existing documents to be updated. While I think this would mainly involve retrieving and storing whole documents, new Learning Designs might be created from fragments of existing Learning Designs – such as Learning Objects that are used in other Learning Designs, or Acts that are part of other Plays. This means that relational queries, which select data from parts of a document or from several documents, must also be efficient.

However the main use of the system will be to play learning designs. This is true for its planned use as an evaluation tool, but would also be true if it went into production. In this case the whole Learning Design document would be loaded into retrieved, and the learner would navigate through the document to see the activities that they should do. They might navigate sequentially through the activities, or might want to access the activities in an ad hoc order, but would generally be navigating within one document at a time. This indicates that the platform needs to process navigational and document queries efficiently. At the same time, however, the system updates the learner’s profile to record whether the user has started or completed an activity and/or Learning Design.

Voyager is a data-centric application rather than document-centric. While the import and export processes would be more efficient if the data is stored in native XML format, the Learning Design Editor and Learning Design Player will be used far more often. Some of those processes access the whole document and perform navigational queries, suggesting that Native XML databases might be more efficient than XML-enabled or relational databases. However some of its processes require ad-hoc access, where an XML-enabled database might be more efficient.

Because a Learning Design package is designed to stand alone, the document will contain all the data it needs, and this data will be repeated in other Learning Design packages. Learning Designs can include Learning Objects, which are learning resource such as documents, pictures or applications (Advanced Distributed Learning Initiative, 2004). Storing these multiple times in the database will result in very large amounts of data redundancy – suggesting that the relational or object-relational model might be more suitable.

“Round tripping” is required in Voyager – IMS Learning Design packages might be imported, and later exported again. This has not been thoroughly tested, but I think that the exported document generally matches the imported document, because the sequence numbers added to the database tables store the order of repeating elements.

As a learning management system, in a production environment Voyager would be likely to be linked to student enrolment and results systems. As many existing applications use relational databases (Kappel et al., 2004), this would be simpler if Voyager also used a relational database.

One final consideration is the availability of suitable XML databases. Tamino is a full commercial database, and supports locking, concurrency control and indexing, but the cost of its license fees make it unsuitable for a research project. Timber is available free of charge, but as it is under development it may not perform as well as a commercial relational database. Using Timber would also mean creating and maintaining a specialised web server, as web hosts do not generally have Timber servers in their standard packages.

A cut-down version of Oracle 9i is available free of charge, and offers most of the features of the enterprise system. Oracle’s object-relational model would allow for simpler imports and exports than a relational database, but web hosts do not often offer Oracle as a standard database server either.

Microsoft SQL Server is provided as database servers by some web hosts, and has some XML-enabled features which could be useful for Voyager. Unfortunately it can only store an XML document in a field in the database (Kappel et al., 2004), rather than map the XML data to relational fields. As discussed above, this would not provide efficient access for Voyager’s purposes, so new learning designs would have to be stored in the relational tables instead. There is therefore no advantage for the import process. The XML view over the database could simplify the export function, but on the other hand it was not difficult to create the export function from the relational database. Developer editions of Microsoft SQL Server are currently available free of charge.

4. Conclusion

The literature does not provide a clear answer as to whether a Native XML database, an XML-enabled database, or a relational database would be more suitable to the Voyager application.

IMS Learning Designs have a pre-defined structure, so data can be stored in a relational database, and the current import process into the relational database satisfactorily preserves document order and annotations for the current purposes.

The mapping of the logical XML model to the relational data model does result in a large number of joins to foreign tables, affecting query performance and increasing both the amount of data that needs to be stored and the complexity of the application. This would not happen in a native XML database. On the other hand, the storage of large Learning Objects within each learning design would lead to large amounts of redundant data in a native XML database.

A native XML database would simplify the import and export processes, but these are not anticipated to comprise the main use of the system. Creating and updating Learning Designs within the system would be more common, but the main use would be to “play” learning designs. This is likely to involve a mixture of document navigation queries, where a native XML database may be faster, and relational queries, where an XML-enabled or relational database is likely to be faster.

The initial choice of a relational database does not, therefore, appear to have been clearly “wrong”. Benchmark tests would be needed to establish which database model is more efficient in practice, and it may be that the most efficient implementation is a combination of data stored in native XML data and data stored in a relational format.

References

- Advanced Distributed Learning Initiative. (2004). *Shareable Content Object Reference Model (SCORM) 2004 3rd edition Overview*. Retrieved December 19, 2006, from <http://www.adlnet.gov/scorm/20043ED.cfm>
- Bourret, R. (2005). *XML and Databases*. Retrieved September 29, 2007, from <http://www.rpbourret.com/xml/XMLAndDatabases.htm>
- Chaudhri, A., Rashid, A., and Zikari, R. (Eds.) (2003). *XML data management: Native XML and XML-Enabled database systems*. Indianapolis: Addison-Wesley.
- Hagen, K., Hibbert, D., & Kinshuk. (2006). Developing a LMS based on the IMS learning design specification. In Kinshuk, R. Koper, P. Kommers, P. Kirschner, D. Sampson, W. Didden (Eds.). *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies* (pp420-424). Kerkrade, The Netherlands: IEEE.
- Hoffer, J., Prescott, M. & McFadden F. (2007) *Modern Database Management* (8th ed.). New Jersey: Pearson Education.
- IMS Global Learning Consortium, Inc. (2003). *IMS Learning Design Best Practice and Implementation Guide*. Retrieved May 25, 2005, from www.imsglobal.org
- Kappel, G., Kapsammer, E. & Retschitzegger W. (2004). Integrating XML and Relational Database Systems. *World Wide Web: Internet and Web Information Systems*, 7, 343–384.
- Koper, R. & Olivier, B. (2004) Representing the Learning Design of Units of Learning. *Educational Technology and Society*, 7(3), 97-111.
- Nambiar, U., Lacroix, Z., Bressan, S., Lee, M., & Li, Y. (2002). Current approaches to XML management. *IEEE Internet Computing*, July-Aug, 43–51.
- Obsanjo, D. (2004). *XML, the new database heresy*. Retrieved September 29, 2007, from <http://www.25hoursaday.com/weblog/PermaLink.aspx?guid=d28ce1fb-7b27-407d-b1a3-0b9a34831ca1>
- Schoning, H. (2001). Tamino - a DBMS Designed for XML IEEE. In *Proceedings of the 17th Annual Conference on Data Engineering*. Heidelberg, Germany: IEEE.
- Tattersall, C., Manderveld, J., Hummel, H., Sloep, P., Koper, R., & Vries, F. D. (2003). *IMS Learning Design Frequently Asked Questions*. Retrieved August 6, 2005, from www.imsglobal.org.
- Vakali, A., Catania, B. & Maddalena, A. (2005). XML Data Stores: Emerging Practices. *IEEE Internet Computing*, March – April, 62 – 69.
- XML:DB Initiative (n.d.) *Frequently Asked Questions*. Retrieved September 29, 2007, from <http://xmldb-org.sourceforge.net/faqs.html>
- Yen, D., Huang, S. & Ku, C. (2002) The impact and implementation of XML on business-to-business commerce. *Computer Standards & Interfaces*, 24, 347–362.