XML Data Stores: Emerging Practices

Athena Vakali • Aristotle University, Greece
Barbara Catania and Anna Maddalena • University of Genoa, Italy

Because XML is rapidly becoming the Internet standard for data representation and exchange, efficient XML document storage has become a core data management issue. Most early XML storage practices rely on conventional database management systems. However, such systems involve mappings and transformations between XML and the underlying database structure. More recent efforts are based on specific XML-tailored systems that provide ad hoc functionalities. This overview of emerging XML storage approaches highlights current practices along with prospective research and implementation trends.

XML’s wide adoption in both academia and industry has revealed a controversial fact. From one perspective, XML is characterized by interoperability, ease of use, and extensibility; from another, it creates difficulties when integrated into existing systems (such as typical database applications and database management systems). One result of this conflict is the arrival of native XML data stores, which involve XML at the internal architecture level.

XML is a hierarchical language, based on tags that imply parent–child relationships. In contrast, rather than using trees as basic storage elements, most DBMSs rely on tables (relational DBMSs), objects and class hierarchies (object-oriented DBMSs), or tables and objects (object-relational DBMSs). Researchers proposed native XML systems to address a specific problem: when relying on a standard DBMS, the system must factorize XML documents before it can store them in its data structures. This typically entails many format mappings and interchanges between XML’s hierarchical structure and the DBMS’s structures, which thus requires a high number of joins to reconstruct the XML document structure during query execution. Native XML data stores define a logical model to guide the storage of XML documents — similar to the way a table guides storage in a relational DBMS — and thus free the data store from DBMS restrictions. Native XML data stores have three key benefits:

• Scalability. Because native XML tools are based on an XML internal format, they can more easily scale to meet either client- or server-side requirements.
• Data-access speed. Native XML tools don’t require overhead time for mappings, interchanges, and joins between XML and other internal structures (such as tables).
• Reliability. Converting to and from XML can result in mismatches between the original XML structure hierarchy and the resulting table. Native tools avoid this problem in that they use XML for the internal document representation.

Here, we survey emerging native XML storage approaches and identify and highlight popular implementations tailored to XML’s “nature” and syntax. By understanding the storage practices of emerging native XML environments, programmers and software designers can better exploit the technology’s scalability and reliability benefits. Understanding the environments’ structure is also useful...
for users and clients, as it lets them take advantage of performance improvements in querying and data access.

**XML Storage: Existing Options**

Various application frameworks and technologies adopt XML, thus providing several options for how and where to store XML data. These options depend on the underlying framework’s particular level, which might range from primitive file systems to evolving DBMSs to the newly emerging directory servers.

**Current Policies**

Table 1 highlights the characteristics of existing XML data store practices. In file-system-oriented storage, the logical-level trees corresponding to the XML document structure are serialized into byte streams and stored either in the operating system’s file system as ASCII files or as binary large objects (Blobs) or character large objects (Clobs) in a DBMS. In the latter case, the Blob or Clob manager then distributes the byte streams over multiple disk pages on the basis of various criteria. This approach’s stream-based processing is easy to implement and has relatively efficient memory usage. However, the approach is less flexible than other approaches in rearranging or accessing XML document components because the whole document must be parsed, which means XML elements are not directly accessible.

In the directory-server-based approach, directory servers use the Lightweight Directory Access Protocol for data and representation. LDAP allows access to specialized databases, or directories, which are organized by a simple tree hierarchy and are thus suitable for storing XML documents. LDAP-based directory servers are becoming the Internet directory standard for accessing and integrating heterogeneous data sources, further supporting the representation of data in trees rather than tables.

In DBMS-oriented approaches, storage depends on several factors, including the DBMS data model and the system’s approach to mapping XML documents. Most existing approaches rely on the popular relational DBMSs, in which it’s quite easy to map XML data to relational tables. There are also a few object-relational and object-oriented DBMS options. The storage approach also depends on the DBMS type:

- **XML-enabled DBMSs** use DBMSs to provide interfaces for transforming XML data to the internal data model, and vice versa.
- **Native XML data stores** define a logical model for XML documents to efficiently support their storage and retrieval. The only interface to native XML database data is XML itself and its related technologies, including the Simple API for XML (SAX), Document Object Model (DOM), XML Path Language (XPath), and XML Query Language (XQuery).

XML-enabled DBMSs are advantageous in that they let the system exploit several DBMS features, such as scalability, concurrency control, and recovery services. However, this approach also factorizes XML content in several structures, which can make processing less efficient. In contrast, native XML data stores often use proprietary storage techniques that guarantee high flexibility and efficient space utilization.

---

**Table 1. Overview of popular XML storage approaches.**

<table>
<thead>
<tr>
<th>Framework</th>
<th>XML Storage Format</th>
<th>Main Advantages</th>
<th>Main Drawbacks</th>
</tr>
</thead>
<tbody>
<tr>
<td>File-system-oriented</td>
<td>ASCII files stored in the file system or database</td>
<td>Easy implementation</td>
<td>Accessing and updating are difficult</td>
</tr>
<tr>
<td></td>
<td>management system (DBMS) as binary large objects (Blobs)</td>
<td>Suitable for small XML sets</td>
<td></td>
</tr>
<tr>
<td>Relational DBMS</td>
<td>Tables</td>
<td>Scalability, reliability, and easy implementation</td>
<td>Requires many joins due to XML</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>document factorization</td>
</tr>
<tr>
<td>Object-relational DBMS</td>
<td>Tables and objects</td>
<td>Easy implementation</td>
<td>XML document factorization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Abstract data type support</td>
<td></td>
</tr>
<tr>
<td>Native XML</td>
<td>Ad hoc data models or typical database models</td>
<td>Flexibility</td>
<td>Less mature than conventional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Improved access performance</td>
<td>DBMSs (such as RDBMSs)</td>
</tr>
<tr>
<td>Directory servers</td>
<td>Tree structure</td>
<td>Optimized for queries</td>
<td>Low update performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Effective data retrieval</td>
<td></td>
</tr>
</tbody>
</table>
The Role of Schemas

An XML document’s structure is individually defined, or else it adheres to a document type definition (DTD) or (more recently) to an XML schema. A DTD defines an XML document’s legal building blocks, typically describing each allowable element within the document. That is, it describes each element’s attributes and (optionally) attribute values; it might also describe element hierarchies and occurrences.

XML Schema (www.w3.org/XML/schema.html) is a DTD successor that expresses shared vocabularies and provides a guide for characterizing an XML document’s structure, content, and semantics. XML Schema Definition (XSD) is a much more powerful instance of XML Schema that developers increasingly favor over DTDs because it’s written in XML (and is thus more direct), and is also queryable, self-documented, and supports automatic schema creation.

Both developers and researchers have begun using schema information to facilitate data querying, storage, and circulation. Frameworks such as Supex,8 OrientStore,9 and Sedna (www.modis.ispras.ru/Development/sedna.htm) focus on schema-conscious XML document management. The advantages of this approach are evident in several new applications:

- As we noted, developers have exploited LDAP’s support for tree-like structures to store XML documents, typically by mapping the DTD to the LDAP schema.
- Topic-specific applications based on particular schemas are becoming more common. These applications use markup languages such as

---

**Table 2. Popular native XML data stores and their practices.**

<table>
<thead>
<tr>
<th>Native XML Framework</th>
<th>Querying</th>
<th>Data Organization</th>
<th>Storage Manager</th>
<th>Storage Practice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natix11</td>
<td>✓</td>
<td>✓</td>
<td>Divides XML documents into subtrees according to the physical disk page size (each subtree is a record)</td>
<td>Schema-independent</td>
</tr>
<tr>
<td>Ipedo <a href="http://www.ipedo.com">www.ipedo.com</a></td>
<td>✓</td>
<td>✓</td>
<td>Maintains, on disk, the physical data files that store collections and metadata associated with collections</td>
<td>Supports XML schema storage</td>
</tr>
<tr>
<td>Xindice <a href="http://xml.apache.org/xindice">http://xml.apache.org/xindice</a></td>
<td>✓</td>
<td>✓</td>
<td>Document-based storage that stores the entire XML document as a single record</td>
<td>Schema-independent</td>
</tr>
<tr>
<td>eXist <a href="http://exist.sourceforge.net">http://exist.sourceforge.net</a></td>
<td>✓</td>
<td>✓</td>
<td>Stores documents either in the internal XML store or on an external relational database, and backs stored data with a multiroot B+-Tree</td>
<td>Schema-independent</td>
</tr>
<tr>
<td>dbXML <a href="http://www.dbxml.com">www.dbxml.com</a></td>
<td>✓</td>
<td>✓</td>
<td>Offers either XML document-based storage or binary streams (records)</td>
<td>Schema-independent</td>
</tr>
<tr>
<td>Toronto XML server (ToX) <a href="http://www.cs.toronto.edu/tox">www.cs.toronto.edu/tox</a></td>
<td>✓</td>
<td>✓</td>
<td>Document-based storage maps XML into flat files, or in a relational or object-oriented structure</td>
<td>Supports XML schema storage</td>
</tr>
<tr>
<td>Timber <a href="http://www.eecs.umich.edu/db/timber">www.eecs.umich.edu/db/timber</a></td>
<td>✓</td>
<td>✓</td>
<td>Uses Shore (<a href="http://www.cs.wisc.edu/shore">www.cs.wisc.edu/shore</a>) for disk management, buffering, and concurrency control (each element is the unit of storage)</td>
<td>Schema-independent</td>
</tr>
<tr>
<td>XUpdate <a href="http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html">http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html</a></td>
<td>✓</td>
<td>✓</td>
<td>Document-based storage in which the entire XML document is a single record</td>
<td>Supports XML schema storage</td>
</tr>
</tbody>
</table>
Directory Services Markup Language (DSML), an XML-schema-definition language for working with directories that use XML-oriented syntax and tools; MathML, used for displaying mathematical notation and expressions on the Web (www.w3.org/Math); and NewsML for global news exchange (www.newsml.org/pages/index.php).

- Some applications use specific XML document data sets, based on particular schema definitions. Popular data sets such as ACM SIGMOD Record (www.dia.uniroma3.it/Araneus/Sigmod) and the DBLP Computer Science Bibliography (http://dblp.uni-trier.de/xml/dblp.xml) are based either on particular DTDs or XML schema definitions.

Typical RDBMSs and OODBMSs don’t require a schema to store or manage XML documents. Moreover, as Table 2 shows, most current native XML data stores are schema-independent and don’t require that XML documents include XML schemas (or DTDs). Even the developers of the Tamino XML server, which handles XML-schema-based documents, state that “it is mandatory that schemas remain optional” and that XML documents be stored regardless of whether they have a schema.7 Popular XML data sets such as DBLP, however, have specific DTDs, and schema-based XML document storage can improve querying and reduce I/Os for data retrieval. As Table 2 shows, some native XML data stores do support schema storage. Furthermore, the LegoDB storage engine has proposed physical XML schemas, which extend XML schemas to include statistics and storage configurations.10 (See www.rpbourret.com/xml/XMLDatabaseProds.htm#native for a full list of commercial and open-source native XML data stores).

Schema-driven XML documents offer:

- Reduced space consumption, by replacing element tags with schema nodes, which take less space than typical tag names. This is suitable for applications like iPlanet’s XMLLDAP directory server (www.sun.com/software/products/directory_srvr/home_directory.html), which includes an XML tag library that developers can use to perform LDAP operations.
- XML query validation, by exploiting XML query language syntax to translate relative paths into absolute paths. Again, this is helpful in LDAP services, where DSML services transport DSML-formatted queries (under HTTP), and specific components (such as XML2LDAP) translate them into a particular schema language syntax.
- Identification of parent–child relationships, which improves performance in solving XML queries for applications that require detection of these and other ancestor–descendant relationships. Because such applications are navigational in nature, they can easily exploit XML’s hierarchical structure.
- Improved indexing, by using indexing techniques optimized for the document schema. This speeds up the querying and data accessing times.

Given the variety and diversity of the available frameworks, choosing a particular XML data store is not easy. Native XML data storage tools and implementations are a more recent and evolving choice, because they’ve emerged to resolve and address the various performance problems encountered when accessing XML under conventional file systems or DBMSs.

Native XML Data Storage

Native XML data stores aim mainly to provide robust XML document storage and manipulation.1 The XML:DB Initiative (www.xmldb-org.sourceforge.net) has officially defined a native XML database as one that:

- uses the XML document as the fundamental unit of logical storage,
- uses a logical model for the XML document itself, and
- requires a particular underlying physical storage model.

Given this, it’s important to understand how native XML databases organize data and what criteria to consider when selecting one.

Data Organization

Figure 1 (next page) shows an abstract view of native XML data-store components, including:

- a native XML interface to map the particular application to the underlying framework, and
- a storage manager to manage data access for querying or updating.

In terms of querying, XML Query (XQuery;
www.w3.org/xml/query) and XPath (www.w3.org/tr/xpath) are the dominant choices. Existing native XML data stores use various indexing and storage-manager practices. Table 2 highlights the most indicative practices for querying, data organization, indexing, and storage.

Native XML data stores can work with any underlying physical storage model (relational, object-relational, or object-oriented databases) or proprietary storage format. 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 nonXML documents (such as graphic files that don’t need DTDs or schemas).

As Figure 2 illustrates, native XML data stores accept XML documents in the following formats:

- **Text**. Data stores keep entire XML documents in textual format and manages them through (often limited) database capabilities.
- **Tree-based**. Using DOM, data stores keep documents in a tree-based representation under an existing or custom database, and then map the binary model to the underlying database format (such as RDBMS tables).

To fully obtain a native XML framework’s benefits, we must understand its storage requirements and practices. Typically, native XML data stores organize data in terms of trees or collections. Trees are popular in research-oriented implementations and are the most obvious choice for representing XML documents. The store maps the trees into a physical data unit, such as Natix or Timber. Natix splits the logical tree representing each XML document into subtrees, which become the basic storage and query units. Timber transforms the XML document into a parse tree, which it stores as an atomic unit in the underlying storage manager.

To manage a large tree’s physical structure, the store might need additional nodes. The storage manager can map a logical tree to a physical one using one of three node types: aggregate (inner nodes), literal (leaf nodes), and proxy (nodes that point to different records). The store manages tree storage by semantically splitting the XML document structure into a set of subtrees that eliminate the XML document structure’s waste (when storing it in a “flat” storage typical of Blobs). Each record thus stores exactly one subtree. Natix supports dynamic physical tree maintenance, and its performance results have been quite promising in terms of supporting updates and space utilization.

Collections are typically found in commercial databases, including Ipdoo, Xindice, and dbXML. Such tools group related XML documents into one of two collection types:

- **Typed collections** contain a schema (based on a DTD or an XML schema), and all documents within the collection must conform to that schema.
- **Untyped collections** contain any number of XML documents, regardless of their schemas and how they relate.

Researchers have also proposed hierarchical collections, which are organized like file system directories. These collections can be nested, and, because each contains a root collection, they’re considered part of an XPath query string. In Ipdoo, for example, the storage manager maintains the collections (the top-level logical data units) on the physical disk pages that store collections and their associated metadata. Xindice and eXist use similar allocation methods, arranging all XML documents within collections on

---

Figure 1. Abstract view of a native XML data management framework. The store includes two key components: a native XML interface and a storage manager.
physical disk pages where they can be queried as a whole.

Tree-based implementations are beneficial for navigation-oriented applications, because the tree corresponding to the XML document is typically easier to parse and process than multiple tables, which require join operations. The collection-based approach is beneficial for applications that involve querying and indexing data sets or lists.

### Table 3. Different approaches for XML data stores.

<table>
<thead>
<tr>
<th>XML Data Store</th>
<th>Approach</th>
<th>Storage</th>
<th>Suitable Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-oriented</td>
<td>Text-based</td>
<td>Document as a whole (file, Blob, or Clob)</td>
<td>To access and retrieve whole documents (or specific text within a document)</td>
</tr>
<tr>
<td>Binary model-based</td>
<td>Tree-based</td>
<td>Internal tree based on the XML data’s hierarchical structure</td>
<td>To refer to specific parts of a document or set of documents</td>
</tr>
</tbody>
</table>

**Figure 2. XML data representations.** (a) Text-based representations typically store whole documents, whereas (b) tree-based approaches store trees based on the data’s hierarchical structure.

**Selection Criteria**

When deciding when and if a native XML data store would be right for your organization or application, consider several questions.

**Is the application data-centric or document-centric?** Data-centric applications tend to do better with XML-enabled DBMSs because data management is supported by the data-oriented DBMS.
capabilities. For document-centric applications, in which data structure is often less regular and fixed, native XML data stores are a better choice.\(^1\,\text{13}\) That said, given the diversity and variety of current application data formats, the distinction between data- and document-centric applications is not always clear. In a hospital patient-management application, for example, a patient’s record document has specific data that’s fine-grained and data-centric (such as name and personal information). At the same time, the document might contain unstructured data (such as a patient’s case history or the doctor’s case evaluation).

Is a DTD or an XML schema available? Although schema information can help map an XML document to the underlying database schema, it doesn’t help store XML documents under native XML management systems. Thus, the absence of a schema generally favors native XML data stores, which are more suitable for semistructured and configuration-free data.

What XML capabilities are available in the environment? XML-enabled DBMSs have recently introduced many XML capabilities. In terms of XML queries, for example, XML-enabled data stores offer XML query language implementations with many advanced capabilities, including full-text and name indexing. Moreover, with respect to query processing, XML-enabled RBMSs behave better for relational queries, whereas native XML data stores behave better for navigational and document queries.

What are the performance requirements? Demands for improvements in both information-retrieval times and storage space are increasing due to advances in applications and services. Researchers have developed various experimental benchmarks to compare existing practices.\(^7\,\text{13}\) These benchmarks consider different types of queries, and by studying their response times, the user can realize which storage technology is more beneficial for his or her needs.

What is the application’s interactivity level? Current XML-focused applications include Web applications (such as servlets, active server pages, and search engines), directory services (such as LDAP), enterprise applications, and, more recently, Web services. Many of these applications are characterized by their navigational nature and interactivity requirements, making them more suited to native XML data stores.

The research and implementation trends for native XML data stores indicate a likely focus on three topics:

- **Distributed querying and updates.** The wide-ranging availability and circulation of XML data has increased the popularity of push and pull strategies for data dissemination. As a result, cooperative and distributed XML document updates are a key research area.\(^1\text{4}\)
- **XML data caching.** XML’s ubiquity makes improving query-response and data-retrieval times critical. Caching appropriate parts of XML documents now requires the identification of semantic information — such as frequent XML query patterns — and caching their results to improve XML management system performance.\(^1\text{5}\) Using cached queries, for example, distributed semantic-cache-oriented frameworks such as ACE-XQ (http://davis.wpi.edu/dsrg/Ace-XQ2002) apply innovative query containment and rewriting techniques when executing user queries. Furthermore, it’s challenging to consider prefetching parts of XML documents to meet future user requests.
- **Directory services.** These services are likely to emerge to overcome the limitations of predefined schemas and support XML-based data structures with hierarchical directories.

Research and implementations in native XML data stores are quite promising and will certainly advance the technology’s performance. Several challenges lie ahead, however, including tuning indexing and storage, supporting both navigational and data-centric applications, and investigating distributed querying over directory services.

References


Athena Vakali is an assistant professor in the Department of Informatics of the Aristotle University of Thessaloniki, Greece. Her research interests include the Web, XML, multimedia data management, Web caching, and content delivery. Vakali has an MSc in computer science from Purdue University and a PhD in computer science from the Aristotle University. Contact her at avakali@csd.auth.gr.

Barbara Catania is an associate professor in the Department of Computer and Information Sciences at the University of Genoa, Italy. Her research interests include deductive and constraint databases, spatial databases, XML and Web databases, pattern management and data mining, indexing techniques, and database security. Catania received an MSc in information sciences from the University of Genoa, Italy, and a PhD in computer science from the University of Milan. Contact her at catania@disi.unige.it.

Anna Maddalena is a PhD student in the Department of Computer and Information Sciences at the University of Genoa, Italy. Her research interests include pattern management and data mining, data warehousing, and XML query processing. Maddalena earned an MSc in computer science from the University of Genoa, Italy. Contact her at maddalena@disi.unige.it.