XTRACTOR

XML Storage (XDB)
White Paper

June 2002

Xtractor™'s XML storage – XDB – is an high-performance, scalable, robust, reliable, transactional, portable, XML document-based data storage system.

XDB combines the ***simplicity*** *and* ***flexibility*** *of XML with the* ***heavy-duty capabilities*** *of relational databases. It does so by both leveraging the strengths of traditional relational databases theory, practices and implementations and at the same time satisfying most of the properties that make native-XML databases attractive.*

However, differently from many products appearing in the market, XDB is not an unproved technological toy; but rather the result of an history of internal use in deploying ***high-volume transactional applications*** *within the Xtractor™ framework, combined with a clear-cut idea on approaching high-volume and document-centric XML data storage – and with a successful effort of specific design and engineering for simplicity and general-purpose usage.*

Therefore, it can both assume the same role that traditional (R)DBMSs have as data storage service providers for applications, and/or seamlessly integrate with leading relational storage systems.

On the other side, its technology does not stem from a relational-database vendor, whose interest is in preserving existing technology as much as in providing new XML-related functionality. Therefore, ***it does not suffer of complexities, idiosyncrasies, compromises and running costs*** *of most of the relational/XML hybrid storage systems currently available on the market. Portability across several platforms with no loss of performance or capabilities – a key requirement for the Xtractor™ framework – also avoids the vendor lock-in typical of relational/XML hybrids.*

In conclusion, XDB results in a mature, stable and reliable product that has been extensively field-tested, is actively used to support complex applications and satisfies real-world requirements in providing concurrent heavy-duty XML data storage services.

The following pages further outline the main rationales, properties and traits of the storage system to provide a high level yet reasonably complete overview of the product and its technology

A word about Xtractor™

The Xtractor™ application building framework is a set of coherent tools, based on the Sun Microsystems's J2EE™ architecture, which allows to easily integrate data from *related but isolated* data silos in an enterprise and to build applications using the integrated data.

The framework has been developed and used for applications ranging from building construction management (integrating project artifacts, pictures, project performance statistics and histories) to telephonic statistical reporting (integrating data from help desk, telecom hardware, cost databases), CRM applications, portfolio management and others.

Xtractor™ is available both as a product in itself and embedded in the off-the-shelf applications mentioned above, and is highly based on XML and related technologies. More information can be found in the *Xtractor™ Framework White Paper* and related documents.

A common factor to all these fields is ***the high volume of data to integrate and the heavy requirements on the applications accessing the integrated data*** (both web-based and client/server), in terms of speed, availability and concurrent users access. Being XML the core format in which data is approached by Xtractor, the need of efficient XML data base services is obvious.

XDB: rationales and XML data modeling approach

To support the Xtractor™ framework, a system is needed that is capable of handling efficiently large volumes of data across the disparate platforms supported by the Java2™ environment, while retaining the flexibility and advantages of XML structure-independency; and of allowing Xtractor™ developers to access data via the plethora of XML-related languages, technologies, tools and standards already available on the market and research arenas.

Xtractor™ approach elects to instantiate data items in XML as *complex elements*[1], optionally defined by a DTD/XSchema – a natural approach for XML document-centric data handling. Each complex element is generally a self-concluded piece of data with a logical meaning – very much like a row in an SQL table, or an object in an OODBMS. At the same time, its structure retains the fluidity of XML – for example, a certain instance of the complex element may present only some of the sub-elements or attributes, the structure of it can be expanded modified without necessarily disrupting existing data or requiring expensive export/import operations, and so on.

A direct consequence of this approach, however, is that high numbers of XML complex elements need to be efficiently stored, retrieved and made available for processing, even for systems of average complexity. The key insight is that each of these elements can actually be seen as a single short XML document[2]: therefore, ***XDB is designed around the idea of storing and indexing a large amount of relatively small documents***, while providing constructs and ways ***to address the entire set of documents as a whole***, performing queries and indexing over the set besides single documents.

Document-based rather than node-based storage	A *document* is the unit of storage and retrieval. Of course, a document may consist of a single element or node, or a complex element with subnodes, but is generally *short*.

---

[1] I.e., XML elements containing sub-elements and attributes. For example, an XML element as **<customer name="Donald"><email>donald@disney.com</email></customer>** is a simple "customer" instance in XML. See **Error! Reference source not found.** for further details.

[2] With some minor technical adjustments, mainly regarding XML namespaces and standard head declarations.

Document-set querying	XPath is used as a language to point to the set of documents to retrieve (considering the documents as sub-elements of a root element) or the insertion point for new documents. Arbitrary complex predicates and expressions can be used to identify the properties of the documents to retrieve or of the insertion position. Aggregation functions in XPath and XSLT can be executed over the entire set.

Why an XML/Relational hybrid: native XML power with relational efficiency

However, it's a well-known fact that in the field of handling data efficiently relational databases have an huge history of research, development and successes that cannot be ignored; however, an internal research action produced the conclusion that *most current XML/relational hybrid solutions in the market are not easily portable, complex and expensive* to acquire, use and maintain.

Also, relational databases are one of the most widespread and well-known technology in the IT commercial arena – and the possibility of building over existing installations, know-how and skills for administration operations (backups, copies, etc) is not to be disregarded either.

Therefore, an architectural decision was taken to implement XDB by using relational (or optionally, object-oriented) databases as back ends, and provide interface layers to allow clients to reason in terms of documents/complex elements sets – for example by querying by XPath/XQuery expressions[3] and obtaining *node sets or scalar values* as results.

However, the presentation to the client code is **natively XML**. Client code and developers do not need to know, configure or set up anything related to the relational support.

Albeit ***the relational support is in general hidden to the client, XDB leverages on the strengths of existing RDBMS products and technology*** for locating quickly the required subset of documents/complex elements that satisfy a query, and completes the processing as necessary in memory. Depending on the features of the RDMBS at hand, certain operations (for example, XPath aggregation functions) may be configured to be executed on the database server or by XDB.

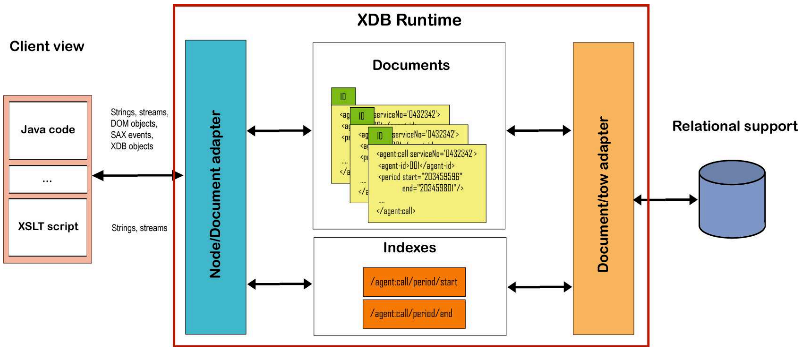


Figure 1 - Isolation levels in XDB implementation

Further, extensive indexing capabilities (that can be fully defined and maintained – conceptually and concretely – at XML level) allow for ***easy applicative-level optimizations and tuning for the specific queries employed by a certain application***.

Xtractor™ requirements include accessibility of the data for further processing both from traditional programming language level (Java/C++) and from more XML-specific scripted modules (XSLT transformations).

XDB makes the data available as XML equally easily to both client environments: XSLT transformations will "see" results as either strings or byte or character stream to transform; Java modules can receive also standard DOM trees, SAX events or directly XDB™ internal object model[4] to process.

Also, DTD/XSchema specification support for the documents is available (and used if present) but optional. Even if in line of principle it's good to have a specification, practical

---

[3] XQuery support is still experimental.

[4] Less generic but more efficient than generic DOM trees.

experience suggests that in the early stages of development[5], the XML representation of data is very fluid – and the possibility of using the storage without having to update the DTD/Schema on every modification is actually valuable.

XML/Relational hybrid	A relational model is used to store documents, each with an opaque internal id, indexing data, configuration and other properties – and to take advantage of the high levels of performance some RDBMS can deliver. Extensive caching and a query optimizer minimizes the relational lookup time for query. The relational back end is completely hidden.
Full XML client view	XML query languages: XPath, XQuery, XUpdate; XSLT can access directly the storage via a library of extension functions. Indexing is conceptually at XML level – an XPath to index is provided to the storage to index all the nodes matching that path, by value, structure or free-text contents.
Optional DTD/XSchema support	Documents and elements may or may not be specified via a DTD or XSchema. If they are, usual XML validation is performed as necessary.
Plasticity towards the database server	Depending on the features and performance of the relational support, certain complex expressions critical to the overall performance can transparently be set to execute either on the database server (translated into SQL) or in the application server (computed in the runtime). For example, • aggregation functions • index merging and union calculations
Programming and scripting views of the results	The data can be accessed both as strings or streams usable by XSLT engines; or directly as objects (DOM or XDB-specific types) or SAX events for usage from object-oriented language level.

Quality by design

Functional features are important, but system-level properties as reliability and little or null downtime are as essential for real-world applications. XDB is the result of both a research effort for best-approaches and designs, and the deployment and validation of resulting implementations in real-world production cases.

Three major and several minor versions have been produced in two years, engineering various designs[6] to implement the hybrid architecture, each version production-quality and progressively more refined – and every one exposing the same well-defined and simple client interface.

Tests are conducted by deploying the various versions in internal replicas of delivered applications, with production-level data volumes; suitable versions are deployed in production environments as betas, leading quickly to stability and maturity.

The resulting current document-centric design provides for a clean implementation which satisfies performance requirements, scalability, transactional properties, while retaining full XML flexibility and ease of extensibility of the implementation.

Active best-design research	A number of versions has been produced based on external and internal research efforts, to produce the best design satisfying the architectural choices and the properties necessary for the document-centric approach.
Internal and field testing	Each alpha version is tested internally on specific test beds (current test platforms allow for various combinations of leading relational database and application server) mirroring high-volume applications; beta versions are put in production, monitored and stabilized quickly.

---

[5] With the Xtractor™ framework.

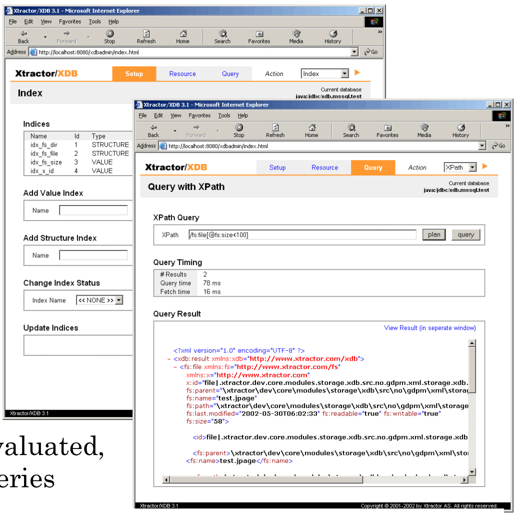
[6] For example, modeling XML with relational entities at node or document level; using relational-level indexing or defining storage-level indexing; using different flavors and patterns of Enterprise Java Beans™ to manage transactional access and data persistence; and so on.

Administrative tools

Any system, to be mature, must allow external access for administration and tuning, without requiring programming or access to its internals. Any mature system, to be *good*, must allow users to do so in a simple and effective way.

XDB does not require explicit access or tuning of the underlying relational support; it rather provides sets of administrative tools for XML-level administration – both as console commands and by accessing a comprehensive web-based administrative application.

In both ways, the current state of the storage can be monitored, indexes maintained and evaluated, statistical data collected and examined, queries executed, etc – all the typical tasks that lead quickly a specific application to production-level efficiency.



Address http://localhost:8080/xdbadmin/index.html

Action Index

Index

Current database java:jdbc/xdb.mssql.test

Indices

Name	Id	Type
idx_fs_dir	1	STRUCTURE
idx_fs_file	2	STRUCTURE
idx_fs_size	3	VALUE
idx_x_id	4	VALUE

Add Value Index

Name

Add Structure Index

Name

Change Index Status

Index Name << NONE >>

Update Indices

Address http://localhost:8080/xdbadmin/index.html

Action XPath

Query with XPath

Current database java:jdbc/xdb.mssql.test

XPath Query

XPath /fs:file[@fs:size<100]

Query Timing

# Results	2
Query time	78 ms
Fetch time	16 ms

Query Result

View Result (in seperate window)

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <xdb:result xmlns:xdb="http://www.xtractor.com/xdb">
  - <fs:file xmlns:fs="http://www.xtractor.com/fs"
      xmlns:x="http://www.xtractor.com"
      x:id="file|.xtractor.dev.core.modules.storage.xdb.src.no.gdpm.xml.storage.xdb.
      fs:parent="\xtractor\dev\core\modules\storage\xdb\src\no\gdpm\xml\storag
      fs:name="test.jpage"
      fs:path="\xtractor\dev\core\modules\storage\xdb\src\no\gdpm\xml\storage
      fs:last.modified="2002-05-30T06:02:33" fs:readable="true" fs:writable="true"
      fs:size="58">

        <id>file|.xtractor.dev.core.modules.storage.xdb.src.no.gdpm.xml.storage.xdb

        <fs:parent>\xtractor\dev\core\modules\storage\xdb\src\no\gdpm\xml\sto
      <fs:name>test.jpage</fs:name>
```

Copyright © 2001-2002 by Xtractor AS. All rights reserved.

Figure 2 - Screenshots of the administration interface

XML-level administrative tools	Both command line and web-based administration tools are available (and equivalent) to monitor contents set up, update or monitor index performance and usage. A querying tools allows immediate verification of expressions or queries. Indexes can be enabled/disabled to monitor their effectiveness.		
Tuning	**Indexes**	**Structure indexes**	information about relationships between specific nodes in the documents
		Value indexes	information about text nodes and attributes
		Free text indexes	Associates text tokens to documents and nodes containing them
	Query execution plans[7]	Various execution plans are generated, estimated and stored by the storage using current statistic information and index availability. Specific plans can be forced if necessary or desired.	
Configuration	Configuration, properties and state can be entirely controlled by administration tools and changed transparently at runtime.		

Standards adherence

It may be a trivial observation nowadays, but one of the key factors in the success of XML and its related technologies has been the fact that precise and publicly available standards have been existing since their inception.

XDB fully adheres to such standards, implementing W3C recommendations and drafts either directly or by employing standard libraries – and guaranteeing interoperability with existing tools and XML-based applications. For example, full XPath support is offered for locating XML nodes within the storage, both for inserting and querying.

Also, XDB makes use of existing and open-source implementations and libraries whenever feasible, concentrating design, coding and specific optimizations to the critical parts; and therefore reducing the effort needed to keep compatibility with the quickly evolving XML standards.

---

[7] This feature is still experimental.

Full standards compatibility	Full XML/XPath support for querying and updating. Efficient third-part standard-compliant components are selected (and not re-implemented from scratch) for certain steps of the processing (parsing, XSLT processing, DOM generation, etc); critical steps are always implemented referencing the current state of recommendations and drafts at W3C.

Transactional properties and portability

XDB ensures database-level portability by not using specific XML features of a certain database; only minor and localized changes, if any, are necessary to comply with the SQL dialects and semantic differences among the leading RDBMS implementations.

By defining a simple yet very effective relational model for storing document sets, and building querying, updating, inserting and indexing over that model, most RDBMSs can be 'plugged-in' easily to support XDB. The current product for example supports MS SQL Server, DB2 and Oracle in combination with BEA and JBoss application servers. Concurrent access to the storage and transactions are supported at two levels: by a design which employs JTA resource managers and transaction controllers and separates clearly in-core manipulation of documents from their submission in the storage; and by leveraging the J2EE™ declarative transaction approach, using Enterprise Java Beans™ and the collaboration of the application server to coordinate data replication and concurrent updating of documents.

Using Java™ and J2EE™ Enterprise Java Beans™ as programming language and distribution architecture and a design carefully crafted not to depend directly from specific application server or database features (or isolating such dependencies in the few cases they're necessary) ensures quick portability on any platform capable of supporting the Java 2™ environment.

Scalability has also been carefully considered in the design, leveraging on the state-replication and distribution features of leading J2EE™ application server implementations.

Portability: independency from RDMBS support	RDBMS-specific features are avoided or carefully isolated, and provided as services of the core runtime. JDBC access and usage of standard application server data services support ensures compatibility with any JDBC-compliant database.
Portability: platform independency	The storage is 100% Java 2™ and J2EE™ based. Isolation of application-server specific characteristics ensures cross- compatibility and testing over different J2EE™ application servers.
Transactional behavior	A change/store separation design, together with J2EE™ declarative transactions are employed to guaranteed ACID properties to concurrent queries and insertions in the storage and ensuring a clear result of (and a consistent state after) each transaction.
Scalability	XDB design allows for replication and minimum-cost synchronization of the state of the storage and the data across multiple physical processing nodes; J2EE™ architectural features (and their implementation by leading application servers) are used as building blocks to ensure scalability.

Final statement

XDB provides real-world performance for accessing large sets of XML documents/elements in a concurrent heavy-duty environment. See the fact sheet for numbers on production-level applications.

Bibliography

[1] *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, October 2000 (http://www.w3.org/TR/REC-xml)

[2] *XML Path Language (XPath) Version 1.0*, W3C Recommendation, November 1999 (http://www.w3.org/TR/xpath)

[3] *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, November 1999 (http://www.w3.org/TR/xslt)

[4] *XUpdate - XML Updating Language - Working Draft*, XML:DB Initiative, August 2000 (http://www.xmldb.org/xupdate/xupdate-wd.html)

[5] *XQuery 1.0: An XML Query Language*, W3C Working Draft, April 2002 (http://www.w3.org/TR/2002/WD-xquery-20020430/)

[6] *XParent: An Efficient RDBMS-Based XML Database System*, Haifeng Jiang, Hongjun Lu, Wei Wang

[7] *Storing and querying XML Data using an RDBMS*, Daniela Florescu, Donald Kossmann, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1999

[8] *DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases*, Roy Goldman, Jennifer Widom, Standford University, 1999

[9] *Approximate DataGuides*, Roy Goldman, Jennifer Widom, Standford University, 1999

[9] *High Performance XML Storage/Retrieval System*, Yasuo Yamane, Nobuyuki Igata, Isao Namba, FUJITSU Sci. Tech. Journal, 2000

[10] *An XML Indexing Structure with Relative Region Coordinate*, Dao Dinh Kha, Masatoshi Yoshikawa, Shunsuke Uemura, GSIS, Nara Institute of Science and Technology, 2001

[11] *Persistent DOM: An architecture for XML repositories in relational databases*, Richard Edwards, Sian Hope, School of Informatics, University of Wales, Bangor, 2001