

An Oracle White Paper
December 2009

Oracle XML DB : Choosing the Best XMLType Storage Option for Your Use Case

Introduction

XMLType is an abstract data type that provides different storage and indexing models to best fit your data and your use of it. As an abstract data type, your applications and database queries gain in flexibility: the same interface is available for all XMLType operations.

XML is being used in a variety of ways – e.g., sometimes XML is constructed from relational data sources, so it is relatively structured, sometimes it is used in the ETL scenario, which is also very structured, sometimes it is used for storing free-form documents like resumes, etc. In addition, the retrieval pattern is different for different kinds of data. The data-centric users usually have a fixed set of queries, whereas the document-centric users issue more ad-hoc queries.

Because the XML usage falls in a broad spectrum, there is no one-size-fits-all storage to offer the best performance and flexibility for each of these use cases. Hence Oracle offers three different storage models for XMLType. Because different storage (persistence) models are available, you can tailor performance and functionality to best fit the kind of XML data you have and the pattern of its use. Therefore, one key decision to make when using Oracle XML DB for persisting XML data as XMLType is which storage model to use for which kind of XML data. This paper guides you on how to choose the best storage option, given your use case.

Storage Options

XMLType tables and columns can be stored in the following ways:

- **Structured storage** – XMLType data is stored relationally. The biggest advantage of structured storage is the performance. This provides the best performance in structured cases – the query performance matches that of relational tables, and updates can be performed in-place. It also provides relational-like schema evolution capability.
- **Binary XML storage** – XMLType data is stored in a post-parsed binary format specifically designed for XML data. Binary XML is compact, post-parsed, and XML schema-aware XML. The biggest advantage of Binary XML storage is its flexibility – you can use it for schemaless documents, or when the schema allows for high variability.
- **Unstructured storage** – XMLType data is stored in Character Large Object (CLOB) instances. CLOB storage should only be used in limited cases. One use case is when you need “document fidelity”, which means that you need to maintain the original XML data, byte for byte – in particular, all original white space need to be preserved. Another use case is when you mostly retrieve full document – CLOB storage will provide fast insert and retrieval for full documents. Also, streams-based replication and Logical Standby are only supported for the Unstructured Storage option.

You can mix storage models, using one model for one part of an XML document and a different model for another part. The mixture of structured and unstructured storage is sometimes called **hybrid storage**. What is true about structured storage is also true about the structured part of hybrid storage. Likewise, what is true about unstructured storage is also true about the unstructured part of hybrid storage.

In addition, Oracle supports the following kinds of indexes on XMLType:

- B-Tree index on Structured storage
- XMLIndex with structured component on all storage options
- XMLIndex with unstructured component on Binary XML and Unstructured storage
- Secondary B-Tree indexes on the secondary tables created by XMLIndex (for both structured and unstructured components)

Each of these storage models and index combinations has its own advantages and disadvantages in different dimensions, such as performance and flexibility. No single storage or index is right for every use case. The advantages and disadvantages of different storage options are summarized in Appendix 1.

XMLType Use Cases

Most XMLType use cases fall into well defined categories listed below. If your use case falls into any of these categories, you could start by using the recommended solution for that use case. If the recommended solution doesn't satisfy your needs, then look at the remaining sections to fine-tune the storage/indexing model.

Note that this whitepaper is about the use cases where the data is persisted as XMLType. Although a common XML use case is "XML generation from relational data". This use case is not discussed here, because the storage is relational and the user is using XML generation functions to generate XML which is not persisted.

Use Case 1: An XML store with very little query requirements

There are several reasons why an application will want to store XML data and perhaps retrieve the full XML. In this use case, there is no requirement to update or query XML fragments. If in case XML fragments do need to be queried or updated, such operations are done in the application-tier, so the database is unaware of them. In such cases, the user has 2 options:

Option 1 is to store the XML into the XMLType CLOB storage.

Option 2 is to just store the XML into a relational BLOB or CLOB column, preferably as SecureFile.

Option 2 will have better performance than option 1. However, when using option 2, Oracle will not parse the XML, so we cannot guarantee the validity of the XML data. In addition, in option 2, users cannot perform XMLType operations on this column.

Use Case 2: ETL: XML used only as a staging area

ETL stands for "extract-transform-load" and refers to XML use cases where customers need to persist XML in the database before transforming it into their operational systems (mainly relational). The XML data is highly structured and conforms to an XML schema. Producing relational values from XML as well as generating XML from relational data is covered under this category. The storage we recommend for ETL is XMLType Structured Storage, also known as Object Relational storage.

Use Case 3: XML persistence requiring interoperability with relational systems (including updates)

This use case is similar to use case 2 above, except that the XMLType data require updates. These updates could update partial XML data (known as "piecewise updates"). The storage we recommend for this use case is the XMLType Structured Storage, also known as the Object Relational storage, since it has excellent support for piecewise updates.

Use Case 4: Semi-structured XML persistence that includes updates

In this use case, either the schema is variable, or large portions of schema are not well defined. Hence the use of XMLType Structured Storage is not feasible. Binary XML is the ideal storage option for this use case. For value searches, use structured XMLIndex when paths are known, and use path-subsetted unstructured XMLIndex when paths are not known beforehand.

Use Case 5: Business intelligence

SQL constructs such as order-by, group-by, and window enable powerful business intelligence (BI) queries over relational data. The XMLTable function allows values in XML to be projected out as a virtual table. Order-by, group-by, and window constructs can operate on columns of the virtual table. Structured XMLIndex internally organizes its storage tables in a manner that reflects the virtual table(s) exposed by XMLTable. Therefore, structured XMLIndex is well suited for indexing XML data in a way that makes such XMLTable-based queries very efficient. A query that uses the XMLTable function can be rewritten to simple access of the relational tables of a structured XMLIndex. This means that order-by, group-by, and window constructs operating on columns of the virtual table are translated to order-by, group-by, and window constructs operating on the corresponding physical columns of the structured XMLIndex tables.

For BI-style queries, we recommend that the user store their data as binary XML, with structured XMLIndex on it. Furthermore, the user should create relational views over XML using XMLTable, where the views project all columns of interest to the BI application. Application queries should be written against these relational views. If structured XMLIndex is created in one-to-one correspondence to these views, Oracle RDBMS will make sure that queries over the views are seamlessly translated into queries over the relational tables of the structured XMLIndex, thereby providing relational performance.

Use Case 6: XML content repository with full text searches

If your application needs to do full text searches on XML data, you should use binary XML with Oracle Text index on it. You may want to create a combination of unstructured XMLIndex and/or structured XMLIndex on top of the binary XML to satisfy value-searches and namespace awareness since Oracle Text index is not namespace aware.

Use Case 7: Data integration from diverse data sources to allow a uniform query interface

If your XML data comes from several different data sources, each having its own schema, then you should store it in the binary XML format.

There are two different flavors of this use case:

1. If data from different data sources share some structured islands that can be normalized, XMLIndex structured component can be created over these structured islands. An RSS news aggregator is a good example of such use case. [The Oracle XML DB Hands-on Lab Part 2](#) offered during the recent Oracle OpenWorld 2009 San Francisco demonstrated this use case.
2. If there is no common structured islands from different data sources, XMLIndex unstructured component can be created.

If your use case doesn't fit into one of the buckets described above, you need to take additional considerations detailed in the following sections on how to choose your storage.

The Rule of Thumb – Data Centric vs. Document Centric

The first thing to consider when choosing an XMLType storage model is the nature of your XML data and the ways you use it. A spectrum of XML use cases has data-centric use of highly structured data at one end and document-centric use of highly unstructured data at the other. The first question to ask yourself is whether your use case primarily data-centric or document-centric. These considerations are summarized in Figure 1 followed by detailed descriptions.

| | Data-Centric | | Document-Centric | |
|----------------------|---|---|---|------------------------------|
| Use Case | XML schema-based data, with little variation and little structural change over time | XML schema-based data, with some embedded variable data | Variable, free-form data, with some fixed embedded structures | Variable, free-form data |
| Typical Data | Employee record | Employee record that includes a free-form resume | Technical article, with author, date, and title fields | Web document or book chapter |
| Storage Model | Structured | Hybrid | CLOB (Unstructured) or Binary XML | |
| Indexing | B-tree index | Index the structured and unstructured parts separately | Function-based index | XMLIndex index |

FIGURE 1: XML USE CASES AND XMLTYPE STORAGE MODELS

- Data-centric – Your data is, in general, highly structured, with relatively static and predictable structure, and your applications take advantage of this structure. Your data also conforms to an XML schema. This kind of data typically follows an entity relation (ER) model.

- Document-centric – there are two cases:
 - Your data is relatively structured, but your applications do not take advantage of that structure: they treat the data as if it were without structure.
 - Your data is generally without structure or with a variable structure. Document structure can vary over time (evolution) and the content is mixed (semi-structured) with many elements contain both text nodes and child elements. Furthermore, many XML elements can be absent or can appear in different orders. Finally, documents might or might not conform to an XML schema.
- Semi-structured – Your data has structured (data-centric) and unstructured (document-centric) parts. The primary document could be structured, with islands of unstructured content, or the primary document could be unstructured, with structured islands.

Once you have determined the data-centric or document-centric half of the spectrum appropriate for your use case and data, consider whether your case is at an end of the spectrum or closer to the middle. That is, just how data-centric or document-centric is your case?

- Employ structured storage for purely data-centric uses. A typical example of this use case would be an employee record (fields employee number, name, address, and so on). The structured storage is an entity-relationship decomposition of the XML. Use B-tree indexing with structured storage. This storage model gives the best performance for data-centric cases as the metadata (i.e., tags) is pulled out into column level, and hence queries can do a metadata lookup, which is extremely fast.
- Employ hybrid storage if your data is composed primarily of invariable XML structures, but it does contain some variable data; that is, it contains a predictably few mixed-content elements. Hybrid storage is structured storage where the unstructured parts are stored as CLOB. A typical example of this use case would be an employee record that includes a free-form resume. Index the structured parts of your data using the B-tree/Bitmap index and unstructured parts of your data using XMLIndex.
- Employ binary XML storage for all document-centric use cases. This option gives the storage flexibility because the metadata (i.e., tags) is stored with the data, so schema changes are easily handled. XMLIndex is the indexing method of choice here.
 - For general indexing of document-centric XML data, use XMLIndex indexes with unstructured components. A typical example of this use case would be an XML Web document or a book chapter.
 - If your data contains some predictable, fixed structures that you query frequently, then you can use XMLIndex indexes with structured components on those parts. A typical example of this use case would be a free-form specification, with author, date, and title fields.

- A single XMLIndex index can have both structured and unstructured components, to handle islands of structure within generally unstructured content. A use case where you might use both components would be to support queries that extract an XML fragment from a document whenever some structured data is present. The unstructured index component is used for the fragment extraction; the structured component is used for the predicate that checks for the structured data (e.g., the SQL WHERE clause).
- In all cases, you can additionally use Oracle Text indexing for full-text queries. This is especially useful for document-centric cases.

These considerations are summarized in Figure 1 above. The figure shows the spectrum of use cases, from most data-centric, at the left, to most document-centric, at the right. The table in the figure classifies use cases and shows the corresponding storage models and indexing methods.

Query pattern

Another important consideration in choosing your storage and indexing option is your query patterns. The question to ask yourself is whether you have a single root hierarchy or a multi-root hierarchy.

- Single root hierarchy – This happens when you have purely content data, and you always query from the root down the tree to the leaves. For example, your XML instance is a book, and you always query from the root down to the text.
- Multi-root-hierarchy – This happens when your query can originate from different elements in the schema. For example, you have a Department-Employee-Project schema, and sometimes your query searches using the “department id”, sometimes using “project id”, and sometimes using “employee id”. This case frequently happens when XML is used as a data-exchange vehicle.

If you have a multi-root hierarchy, then structured storage will give you the best performance because it will perform relational-style lookups starting from any storage table to the parent/child tables. If, for some reason, you are unable to use the structured storage, the next best choice is to use Binary XML with structured XML Index. Single root hierarchy case is amenable to different storage options, as specified in the “Advanced considerations” section below.

If your use case is primarily structured, when do you choose Structured storage or Binary storage with Structured XMLIndex ? Structured XMLIndex lets you leverage the flexibility of Binary XML while maintaining relational performance. One way to determine if Structured XMLIndex is right for you, is to ask yourself – are your queries known ahead of time? If the queries are known ahead of time, and the list of Xpaths queried is known,

you can create a structured XMLIndex on those paths. Note that the queries can change over time, in which case you can ALTER your structured XMLIndex. However, if the queries are not known ahead of time, you are better off choosing the object-relational storage.

Advanced considerations

Of course, not all use cases are easy to classify into the spectrum outlined in the above sections. Even when they are, other constructs of the schema may dictate the storage model. Three storage solutions are described below. Following that, we give a flowchart guiding you on how to choose a storage model.

The storage solutions

Here are the three storage options Oracle XMLDB provides. Please see “The flowchart” in the next subsection to decide which solution is right for you.

Solution A: Choose Structured storage.

- Create B-tree and bitmap indexes just like you would for relational storage.

Solution B: Choose Binary XML storage. Choose your indexing options:

- Does your data have predictable structured islands in it?
 - Yes → Choose Structured XML Index for the structured islands.
- Choose Unstructured XML index for free-form part of the data.
- Do you need to support full-text queries?
 - Yes → Create Text index.

Solution C: Choose Unstructured (CLOB) storage. Choose your indexing options:

- Does your data have predictable structured islands in it?
 - Yes → Choose Structured XML Index for the structured islands.
- Choose Unstructured XML index for free-form part of the data.
- Do you need to support full-text queries?
 - Yes → Create Text index.

Note: This solution consumes higher disk space. Update to any part of the document will be written back the whole document. There will also be high memory consumption for SELECT statements due to DOM construction. In short, solution C should be chosen sparingly.

The Flowchart

This flowchart guides you on how to choose the storage. The solutions refer to Solutions A, B and C in the subsection above.

1. Do you need Document Fidelity for your XMLType? In other words, do you want to maintain the original XML data, byte for byte, with all original white spaces preserved?
Yes → Stop. Choose “Solution C”.
2. Do you usually just insert and select the entire XML data? In other words, do you rarely select or update part of the XML?
Yes → Stop. Choose “Solution C”.
3. Do you want to store XMLType instances conforming to multiple schemas in 1 table/column? (Note: This kind of usage is not recommended because your queries will not be able to take advantage of the schema to make optimizations.)
Yes → Stop. Choose “Solution B”
4. Do you have a schema for your XMLType?
Yes → Go to Step 6.
5. (You do not have a schema.) Is your XMLType data-centric? Is it possible for you to generate a schema using a schema generation tool?
No → Stop. Choose “Solution B”
6. If you came here, it means that your data conforms to a single schema, so you will get the best performance out of using the Structured storage. However, there are several schema constructs that do not give the best performance for structured storage. You will need to massage your schema and/or use case to make it structured-storage-friendly. Here are some factors that may affect your choice.

6.1. Schema evolution:

Note: It is conceivable that your use case during product development may be different from that of a production product. For example, it is conceivable that your data is structured, and your schema may evolve frequently during product development. However, once your product is released, it may evolve infrequently. In this case, it is important to consider the production time schema evolution (as opposed to development time schema evolution).

6.1.1. Will the XML Schema be evolved very frequently?

No → Go to 6.2

- 6.1.2. Can you take advantage of in-place evolve, or will you need to do copy-evolve?
In-place evolve → Go to 6.2
 - 6.1.3. Are you OK with taking the downtime when the data is being copied for copy-evolve?
Yes → Go to 6.2.
No → Stop. Choose “Solution B”.
 - 6.2. **Sparseness of data:** Is your data extremely sparse (like HL7, XBRL) ?
Yes → Stop. Choose “Solution B”.
 - 6.3. **Use of ANY, Choice:** A lot of automatic schema generators add constructs like ANY, Choice in the schema to make it more flexible. Many times, these are not strictly needed. These constructs make it hard to register the schema as OR.
 - 6.3.1. Does your schema use constructs that make it hard to register as OR, e.g., ANY, Choice etc?
No → Go to 6.4
 - 6.3.2. Is it possible to modify your schema to remove these constructs?
Yes → Modify the schema to remove such constructs and go to 6.4
No → Stop. Choose “Solution B”.
 - 6.4. Stop. Choose “Solution A”.
7. If you are still unable to decide what storage / indexing option is right for you, try out both Solutions A & B, and run performance experiments with your workload to see what works best for you.

Conclusion

XML has diverse use cases ranging from data-centric to document-centric, so there is no one-size-fits-all storage model to give the best performance and flexibility for each of these cases. XMLType is an abstract data type that provides different storage and indexing models to best fit your data and your use of it. One key decision to make when using Oracle XML DB for persisting XML data as XMLType is which storage model to use for which kind of XML data. This paper has guided you in looking at various properties of your XML data to decide the best storage and indexing option, given your specific use case.

Appendix 1: Storage options relative advantages

The advantages and disadvantages of different storage options are summarized in Table-1 below:

| TABLE 1 XMLTYPE STORAGE MODELS: RELATIVE ADVANTAGES | | | |
|---|--|---|--|
| QUALITY | STRUCTURED STORAGE | BINARY XML STORAGE | UNSTRUCTURED (CLOB) STORAGE |
| Throughput | – XML decomposition can result in reduced throughput when ingesting or retrieving the entire content of an XML document. | + High throughput. Fast DOM loading. There is a slight overhead from the binary encoder / decoder. | ++ High throughput when ingesting and retrieving the entire content of an XML document. |
| Indexing support | ++ B-tree, Bitmap, Oracle Text, XMLIndex, and function-based indexes. | + XMLIndex, function-based, and Oracle Text indexes. | + XMLIndex, function-based, and Oracle Text indexes. |
| Queries | ++ Extremely Fast. Relational query performance. Users can create B-tree indexes on the exploded columns. | + Fast when using XMLIndex. User queries which cannot use the index use streaming Xpath evaluation, which is reasonably fast as well. | –. Fast when using XMLIndex. Parts of query which can't use the index cannot be optimized. |
| Update operations (DML) | ++ Extremely fast. Relational column gets updated in-place. | + In-place, piecewise update for SecureFile LOB storage. | – When any part of the document is updated, the entire document must be written back to disk. |
| Space efficiency (disk) | ++ Extremely space-efficient. | + Space-efficient. | – Consumes the most disk space, due to insignificant white space and repeated tags. |
| Data flexibility | – Limited flexibility. Only documents that conform to the XML schema can be stored in the XMLType table or column. | + Flexibility in the structure of the XML documents that can be stored in an XMLType column or table. | + Flexibility in the structure of the XML documents that can be stored in an XMLType column or table. |
| XML schema flexibility | – One XMLType table can only store documents conforming to one schema. Also provides relational-like in-place schema evolution capability. | ++ Can store schemaless or schema based documents. An XMLType table can store documents conforming to any of the registered schemas. | ++ Can store schemaless or schema based documents. Cannot use multiple XML schemas for the same XMLType table. |
| XML fidelity | + DOM fidelity: A DOM created from an XML document that has been stored in the database will be | + DOM fidelity (see structured storage description). | ++ Document fidelity: Maintains the original XML data, byte for byte. In particular, all original white space is |

| TABLE 1 XMLTYPE STORAGE MODELS: RELATIVE ADVANTAGES | | | |
|---|---|---|--|
| QUALITY | STRUCTURED STORAGE | BINARY XML STORAGE | UNSTRUCTURED (CLOB) STORAGE |
| | identical to a DOM created from the original document. However, insignificant white space may be discarded. | | preserved. |
| Optimized memory management | + XML operations can be optimized to reduce memory requirements. | + XML operations can be optimized to reduce memory requirements. | – XML operations on the document require creating a DOM from the document. |
| Validation upon insert | + XML data is partially validated when it is inserted. | ++ XML schema-based data can be fully validated when it is inserted, though this is an expensive operation. | + XML schema-based data is partially validated when it is inserted. |
| Partitioning | ++ Available | + Partition based on virtual columns. | + XMLType columns can be partitioned when the partitioning key is a relational column. |
| Streams based replication | - Not available | - Not available | ++ Available |
| Compression and Encryption | Each element/attribute can be compressed / encrypted individually | Binary XML with SecureFile storage can be compressed / encrypted | Cannot be compressed / encrypted. |



Oracle XML DB : Choosing the Best XMLType
Storage Option for Your Use Case

December 2009

Author: Geeta Arora

Contributing Authors: Oracle XML DB

Development and Product Management Team

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com



| Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, 2010, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.