Oracle Database 11g XML DB Technical Overview

An Oracle White Paper
July 2007

ORACLE

Oracle Database 11g XML DB Technical Overview

Oracle Database 11g XML DB Technical Overview

Oracle XML DB has been widely adopted by customers and partners for a broad spectrum of use cases with varied requirements since its introduction five years ago.

INTRODUCTION

Oracle XML DB has been widely adopted by customers and partners for a broad spectrum of use cases with varied requirements since its introduction five years ago. Oracle XML DB has also been making strides over the years to offer the most comprehensive standards-based XML storage, retrieval, and publishing capabilities on a high performance, reliable, available, scalable, and secure relational platform.

Why XML?

XML is an open standard that provides a unified model for data, content and metadata. It is self describing and easily readable by both humans and machines. It is vendor-neutral and extensible. These features have led to its adoption by a wide range of industries. Most XML usage falls into one of the following three categories, each of which is discussed below:

- Data exchange and integration
- XML-centric application development
- Management of content and metadata

In an enterprise environment, it is a common requirement to bring these usage categories together in information processing flows.

Data Exchange and Integration

By some estimates 40% of a typical IT budget is consumed by integration efforts. The use of loosely coupled, flexible, XML-based integration has consistently demonstrated significant cost savings and reduction in time-to-market when compared with more tightly coupled alternatives. The success of this approach is particularly visible in the growth of XML messaging in service-orientated architectures (SOA). Organizations are also starting to realize that they need to preserve some of this transient XML in order to meet regulatory and compliance requirements.

XML-Centric Application Development

XML has replaced proprietary EDI techniques as the de-facto standard for information exchange in a large number of vertical industries. The data structures defined by these standards are typically quite complex, containing recursive

structures and other complex constructs that have proven difficult to map to a purely relational model. These data structures are often described using the W3C XML Schema standard[1]. As the volume and complexity of the XML increases, the costs and overhead of mapping between the XML and relational representations of the data grows exponentially. Many organizations are looking to reduce development costs by storing this data directly as XML in the database.

Management of Content and Metadata

Over the past few years, several high-end document-authoring systems, including Adobe's Framemaker, have migrated from using proprietary formats or SGML to XML. More recently, desktop productivity tools such as Open Office[2] and Microsoft Office have switched to storing their documents as XML. This is leading to a growth in XML content that organizations need be able to manage effectively. The flexibility of XML also makes it attractive as a medium for managing metadata about other forms of content.

REQUIREMENTS FOR SERVER-BASED XML SUPPORT

Database customers need a platform that delivers performance and the ability to easily develop the next generation of XML-centric applications. They need to be able to use that platform together with the latest XML standards and application development techniques. As the volume and mission critical nature of the XML increases, organizations need the ability to search it in an efficient manner, and to manage it with the same rigor and ease as other types of corporate data. They need an XML platform that delivers the same levels of performance, reliability, availability, scalability, and security as their existing relational databases.

When existing database technology evolves to meet these XML requirements, the result is a hybrid platform, equally capable of managing XML and relational data. This reduces the complexity of data and business processes, leading to significant cost savings, and helps avoid the risks inherent in introducing new infrastructure into an organization. To be truly effective, such a hybrid database also needs to support the XML standards that make the process of generating, loading, updating, retrieving, transforming and indexing XML as simple and performant as possible.

Oracle Database, with Oracle XML DB, is a proven hybrid database for managing both XML and relational data. Oracle XML DB has been an integral part of Oracle Database for several years and four major database releases.

Oracle Database, with Oracle XML DB, is a proven hybrid database for managing both XML and relational data. Oracle XML DB has been an integral part of Oracle Database for several years and four major database releases. Customers have successfully deployed XML-centric applications on Oracle XML DB.

Oracle is leaping ahead in this release by introducing a new storage model – the binary XML storage model, and salient new capabilities such as XML indexes, in-place XML schema evolution, database-native web services, compound documents, and much more.

---

[1] http://www.w3c.org
[2] http://xml.openoffice.org/

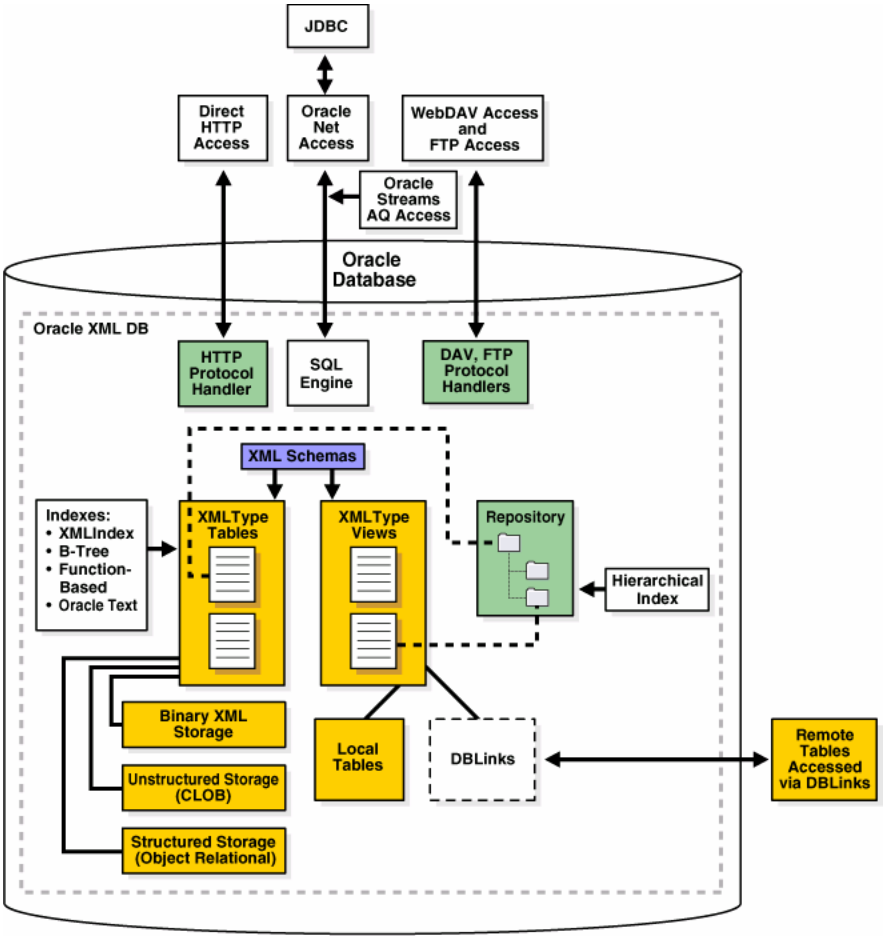
ORACLE XML DB ARCHITECTURE

Oracle XML DB is built on the core components of XMLType abstraction, XML DB repository, standard query languages, API support, and protocol support.

Oracle XML DB is built on the core components of XMLType abstraction, XML DB repository, standard query languages, API support, and protocol support. Each component provides a rich set of services for the essential aspects of Oracle XML DB. We will take a closer look at these components in the following sections.

XMLType Storage Models

Oracle XML DB supports comprehensive storage models (i.e., structured, unstructured, and binary XML storage models) to serve diverse XML use cases with different requirements. Although the majority of current customers' use cases are data-centric and hence structured storage is ideal. However, we have seen steady growth in document-centric use cases where binary XML and XML indexes will make a substantial difference.



As XML covers both structured and unstructured data for different use cases, Oracle XML DB customers have been using different storage and indexing options for XML with different characteristics to achieve optimal data storage and query. There is no one-size-fits-all solution for such variety of use cases.

As XML covers both structured and unstructured data for different use cases, Oracle XML DB customers have been using different storage and indexing options for XML with different characteristics to achieve optimal data storage and query. There is no one-size-fits-all solution for such variety of use cases. The figure below shows a diagram of real world XML use cases and corresponding XML storage models and indexing options provided by Oracle XML DB.

Data-Centric	Hybrid (Object-Relational and CLOB) B-Tree, XML, and Full-Text Indexes	Object-Relational Storage B-Tree Indexes
Document-Centric	Binary XML or CLOB Storage XML and Full-Text Indexes	Binary XML or CLOB Storage XML and Full-Text Indexes

unstructured *structured*

Parts of the document

In the figure, the vertical column indicates whether the document is mainly data-centric or document-centric. The horizontal column indicates whether the document contains structured or unstructured components inside it. The right upper quadrant represents data-centric XML for which the object-relational schema-aware storage model along with B-tree indexes would be the right choice. The left upper quadrant represents data-centric XML with certain unstructured fragments. For this use case, using the object-relational storage as the overall storage model with unstructured parts stored as CLOB using XML indexes would be the ideal storage model. The left lower quadrant represents completely document-centric XML for which the binary XML or CLOB storage model with XML indexes are recommended. The right lower quadrant represents document-centric XML with certain structured parts. The binary XML or CLOB storage model along with a XML indexes for the structured part works best.

Oracle XML DB Repository

The relational model of the table-row-column metaphor is accepted as an effective mechanism for managing structured data. The model is not as effective for managing semi-structured and unstructured data, such as document-oriented XML. For example, a book is not easily represented as a set of rows in a table. It is more natural to represent a book as a hierarchy, book:chapter:section:paragraph, and to represent the hierarchy as a set of folders and subfolders.

Oracle XML DB Repository is a component of Oracle Database that is optimized for handling XML data as files in a file system.

Oracle XML DB Repository is a component of Oracle Database that is optimized for handling XML data as files in a file system. The Oracle XML DB repository contains resources, which can be either folders (directories, containers) or files. Each resource has these properties:

- It is identified by a path and name.
- It has content (data), which can be XML data but need not be.
- It has a set of system-defined metadata (properties), such as Owner and CreationDate, in addition to its content. Oracle XML DB uses this information to manage the resource.
- It might also have user-defined metadata: information that is not part of the content, but is associated with it.
- It has an associated access-control list that specifies who can access the resource, and for what operations.

Although Oracle XML DB Repository treats XML content specially, you can use Oracle XML DB Repository to store other kinds of data, besides XML; you can use the repository to access any data that is stored in Oracle Database.

You can access data in Oracle XML DB Repository in the following ways:

- Using SQL, through views RESOURCE_VIEW and PATH_VIEW
- Using PL/SQL, through the DBMS_XDB API
- Using Java, through the standard JSR 170 Content Repository API or the Oracle XML DB resource API for Java

Besides supporting APIs that access and manipulate data, Oracle XML DB Repository provides APIs for the following services:

- Versioning – Oracle XML DB uses the DBMS_XDB_VERSION PL/SQL package for versioning resources in Oracle XML DB Repository. Subsequent updates to a resource create a new version (the data corresponding to previous versions is retained). Versioning support is based on the IETF WebDAV standard.
- ACL Security – Oracle XML DB resource security is based on access-control lists (ACLs). Every resource in Oracle XML DB has an associated ACL that lists its privileges. Whenever resources are accessed or manipulated, the ACLs determine if the operation is legal. An ACL is an XML document that contains a set of access-control entries (ACEs). Each ACE grants or revokes a set of permissions to a particular user or group (database role). This access-control mechanism is based on the WebDAV specification.
- Foldering – Oracle XML DB Repository manages a persistent hierarchy of folder (directory) resources that contain other resources (files or folders). Oracle XML DB modules, such as protocol servers, the schema manager, and the Oracle XML DB RESOURCE_VIEW API, use foldering to map path names to resources.

Oracle XML DB Protocol Architecture

You can access XML documents in Oracle XML DB Repository using standard connect-access protocols such as FTP, HTTP(S), and WebDAV, in addition to SQL, PL/SQL, Java, and C languages.

You can access XML documents in Oracle XML DB Repository using standard connect-access protocols such as FTP, HTTP(S), and WebDAV, in addition to SQL, PL/SQL, Java, and C languages. The repository provides content authors and editors direct access to XML content stored in Oracle Database.

A resource in this context is a file or folder, identified by a URL. WebDAV is an IETF standard that defines a set of extensions to the HTTP protocol. It allows an HTTP server to act as a file server for a DAV-enabled client. For example, a WebDAV-enabled editor can interact with an HTTP/WebDAV server as if it were a file system. The WebDAV standard uses the term resource to describe a file or a folder. Each resource managed by a WebDAV server is identified by a URL. Oracle XML DB adds native support to Oracle Database for these protocols. The

protocols were designed for document-centric operations. By providing support for these protocols, Oracle XML DB allows Windows Explorer, Microsoft Office, and products from vendors such as Altova, Macromedia, and Adobe to work directly with XML content stored in Oracle XML DB Repository.

One key feature of the Oracle XML DB architecture is that HTTP(S), WebDAV, and FTP protocols are supported using the same architecture used to support Oracle Data Provider for .NET (ODP.NET) in a shared server configuration. The Listener listens for HTTP(S) and FTP requests in the same way that it listens for ODP.NET service requests. When the Listener receives an HTTP(S) or FTP request, it hands it off to an Oracle Database shared server process which services it and sends the appropriate response back to the client.You can use the TNS Listener command lsnrctl status to verify that HTTP(S) and FTP support has been enabled.

APIs for Oracle XML DB (Java, PL/SQL, ODP.NET, and C)

Oracle XML DB is accessible from popular programming language APIs, such as Java, PL/SQL, and C. It provides Java, PL/SQL, and C implementations of the DOM API.Applications that use JDBC, such as those based on servlets, need prior knowledge of the data structure they are processing. Oracle JDBC drivers allow you to access and update XMLType tables and columns, and call PL/SQL procedures that access Oracle XML DB Repository.Applications that use DOM, such as those based on XSLT transformations, typically require less knowledge of the data structure. DOM-based applications use string names to identify pieces of content, and must dynamically walk through the DOM tree to find the required information. For this, Oracle XML DB supports the use of the DOM API to access and update XMLType columns and tables. Programming to a DOM API is more flexible than programming through JDBC, but it may require more resources at run time.

ORACLE XML DB CAPABILITIES

With the architecture described in previous sections, we will now focus on invidual capabilities of Oracle XML DB.

XMLType

To provide flexibility and versatility, Oracle XML DB is built upon the XMLType abstraction for storing, querying, accessing, transforming, and manipulating XML data.

XMLType is a native data type for XML data. It provides methods that allow operations such as XML schema validation and XSL transformation on XML content.

XMLType is a native data type for XML data. It provides methods that allow operations such as XML schema validation and XSL transformation on XML content. You can use XMLType as you would any other data type. For example, you can use XMLType when you do any of the following:

- Creating a column in a relational table
- Declaring PL/SQL variables

- Defining and calling PL/SQL procedures and functions

XMLType is also an object type, so you can also create a table of XMLType instances. By default, an XMLType table or column can contain any well-formed XML document.

XMLType tables or columns can be constrained to conform to an XML schema. This has several advantages:

- The database ensures that only XML documents that validate against the XML schema are stored in the column or table.
- Since the contents of the table or column conform to a known XML structure, Oracle XML DB can use the information contained in the XML schema to provide optimized query and update processing of the XML.
- You have the option of storing the XML document content using structured storage. This decomposes the document and stores it as a set of object-relational objects. The object-relational model used to store the document is derived from the XML schema.

Structured, Binary XML, and Unstructured Storage Models

XMLType is an abstract data type that allows for different storage models to best fit your data and your use case. As an abstract data type, your applications and database queries gain in flexibility, because the same interface is available for all XMLType operations. Oracle XML DB supports different storage (persistence) models for you to optimize performance and functionality by selecting a storage model that 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 XML data.

You can change XMLType storage from one model to another, using database import/export. Your application code does not need to be changed. You can change XML storage options when tuning your application; each storage option has its own benefits.

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

- Structured storage – XMLType data is stored as a set of objects. This is also referred to as object-relational storage and object-based persistence.
- Binary XML storage – XMLType data is stored in a post-parse, binary format specifically designed for XML data. Binary XML is compact, post-parse, XML schema-aware format. This is also referred to as post-parse persistence.
- Unstructured storage – XMLType data is stored in Character Large Object (CLOB) instances. This is also referred to as CLOB storage and text-based persistence.

In addition, the structured and the unstructured storage can also be used together in a **hybrid** storage. The structured part of a hybrid storage takes advantage of the

structured storage capabilities, and the unstructured part of a hybrid storage makes use of the unstructured storage advantages.

Both the binary XML and the unstructured storage support two LOB-storage options, SecureFile and BasicFile. SecureFile is a revolutionary new capability in this release to provide high performance and scalable storage and retrieval of semi-structured and unstructured data.

Each storage model has its advantages, depending on the use case and the characteristics of the XML data.

Structured Storage

Structured storage advantages over unstructured storage include optimized memory management, reduced storage requirements, B-tree indexing, and in-place updates. These advantages are at a cost of increased processing overhead during ingestion and retrieval of XML data, and reduced flexibility in the structure of the XML that can be managed by a given XMLType table or column. Structural flexibility is reduced because data and metadata (such as column names) are separated in object-relational storage; instance structures cannot vary easily. Structured storage is particularly appropriate for highly structured data that does not change often, provided that this maps to a manageable number of database tables and joins.

Binary XML Storage

Binary XML storage provides more efficient database storage, updating, indexing, and fragment extraction than unstructured storage.

Binary XML storage provides more efficient database storage, updating, indexing, and fragment extraction than unstructured storage. It can provide better query performance than unstructured storage — it does not suffer from the XML parsing bottleneck (it is a post-parse persistence model). Like structured storage, binary XML storage can also be made aware of XML Schema data types and can take advantage of native database data types. Like unstructured storage, no data conversion is needed during database insertion or retrieval. Like structured storage, binary XML storage allows for piecewise updates. Because binary XML data can also be used outside the database, it can serve as an efficient XML exchange medium, and you can offload work from the database to increase overall performance in many cases. Like unstructured storage, binary XML data is kept in document order. Like structured storage, data and metadata can, using binary storage, be separated at the database level, for efficiency. Like unstructured storage, however, binary storage allows for intermingled data and metadata, which lets instance structures vary. Binary XML storage allows for very complex and variable data, which in the structured-storage model could necessitate using many database tables and joins. Unlike the other XMLType storage models, you can use binary storage for XML schema-based data even if the XML schema is not known beforehand, and you can store multiple XML schemas in the same table and query across common elements.

When you insert XML schema-based data into binary XMLType columns or tables, the data is fully validated against the XML schema. Insertion fails if the data is invalid.

Binary XML storage is the closest thing to a universal storage model for XML data — you can use it effectively for a very wide range of use cases, from document-centric to data-centric.

Streaming XML Schema Validation

XML schema validation had been a costly step in XML processing. With binary XML storage option, schema validation is performed with a new streaming implementation. Streaming XML schema validation requires much less memory. Therefore it performs much more efficiently and scales well when validating large number of XML documents or large sized XML documents.

Streaming XPath processing

The streaming XPath engine is implemented as an NFA (non-deterministic finite automata). The set of XPaths can be compiled away into a state machine and then multiple instance documents can be fed one after the other to evaluate all the XPaths in one streaming pass.

Partial updates

Users can update XML documents stored in binary format using the regular SQL UPDATE statements along with the partial update SQL operators such as UPDATEXML, INSERTXMLBEFORE, APPENDCHILDXML and DELETEXML. When the user updates only a small part of the original document, the system optimizes the update by replacing only the affected portions of the document instead of replacing the entire document. This functionality relies on the ability to perform sliding inserts (i.e. updating a portion of the underlying BLOB without replacing the entire CLOB value) using the SecureFile storage option. The partial update optimization is entirely transparent in terms of user syntax and semantics. The benefits are better performance due to lesser I/O and logging overheads.

Unstructured Storage

Unstructured storage allows higher throughput than structured storage when inserting and retrieving entire XML documents. No data conversion is needed, so the same format can be used outside the database. Unstructured storage also provides greater flexibility than structured storage in the structure of the XML that can be stored. Unstructured storage is particularly appropriate for document-centric use cases. These advantages can come at the expense of certain aspects of intelligent processing. For example, in the absence of indexing, there is little that the database can do to optimize queries or updates on XML data that is stored in a CLOB instance. In particular, the cost of XML parsing (often implicit) can

significantly impact query performance. Indexing with XMLIndex can improve the performance of queries against unstructured storage.

Guidelines for Choosing the Storage Models

Relative advantages of each storage model, guidelines for choosing storage models for given use cases.

	Structured Storage	Binary Storage	Unstructured Storage
Throughput	– XML decomposition can result in reduced throughput when ingesting retrieving the entire content of an XML document.	++ High throughput. Fast DOM loading.	+ High throughput when ingesting and retrieving the entire content of an XML document.
Space efficiency (disk)	++ Extremely space-efficient.	+ Space-efficient.	– Consumes the most disk space, due to insignificant white spaces 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	– Relatively inflexible. Data and metadata are stored separately. Cannot use multiple XML schemas for the same XMLType	++ Flexible. Can store data and metadata together or separately. Can use multiple XML schemas for the same XMLType table.	+ Flexible. Data and metadata are stored together. Cannot use multiple XML schemas for the same XMLType table.

	Structured Storage	Binary Storage	Unstructured Storage
	table.		
XML fidelity	– DOM fidelity: A DOM created from an XML document that has been stored in the database will be identical to a DOM created from the original document. However, insignificant whitespace may be discarded.	– DOM fidelity (see structured storage description).	+ Document fidelity: Maintains the original XML data, byte for byte. In particular, all original whitespace is preserved.
Update operations (DML)	++ In-place, piecewise update.	+ 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.
XQuery- and XPath-based queries	++ XPath operations can often be evaluated using XPath rewrite, leading to significantly improved performance, particularly with large collections of documents.	+ Streaming XPath evaluation avoids DOM construction and allows evaluation of multiple XPath expressions in a single pass. Navigational XPath evaluation is significantly faster than with unstructured storage. XMLIndex indexing can further improve query	– XPath operations are evaluated by constructing a DOM from the CLOB data and using functional evaluation. Expensive when performing operations on large documents or large collections of documents. XMLIndex indexing can further improve

	Structured Storage	Binary Storage	Unstructured Storage
		performance.	query performance.
SQL constraint support	+ SQL constraints are supported.	+ SQL constraints are supported.	– SQL constraints are not available.
Support for SQL scalar data types	+ Yes	+ Yes	– No
Indexing support	B-tree, Oracle Text, and function-based indexes.	XMLIndex, function-based, and Oracle Text indexes.	XMLIndex, function-based, and Oracle Text indexes.
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 is fully validated when it is inserted.	XML schema-based data is partially validated when it is inserted.

Using Indexes to Improve Performance

For different storage models, Oracle XML DB supports the creation of a variety of indexes on XML content:

- **B-Tree indexes.** When the XMLType table or column is based on structured storage techniques, conventional B-Tree indexes can be created on underlying SQL types.

- **XML indexes.** XMLIndex provides a general, XML-specific index that indexes the internal structure of XML data using binary XML, unstructured, and hybrid storage.

- **Function-based indexes.** These can be created on any XMLType table or column. Its usage is limited to pre-defined XPath queries on unstructured storage.
- **Full-Text indexes.** These can be created on any XMLType table or column.

B-Tree Index

B-tree indexes are typically created by using SQL function extractValue, although it is also possible to create indexes based on other functions such as existsNode. During the index creation process Oracle XML DB uses XPath rewrite to determine whether it is possible to map between the nodes referenced in the XPath expression used in the CREATE INDEX statement and the attributes of the underlying SQL types. If the nodes in the XPath expression can be mapped to attributes of the SQL types, then the index is created as a conventional B-Tree index on the underlying SQL objects. If the XPath expression cannot be restated using object-relational SQL then a function-based index is created.

XML Indexes

XMLIndex provides a general, XML-specific index that indexes the internal structure of XML data.

XMLIndex provides a general, XML-specific index that indexes the internal structure of XML data. One of its main purposes is to overcome the indexing limitation presented by unstructured and hybrid storage of XML data, that is, CLOB storage. It does this by indexing the XML tags of your document and identifying document fragments based on XPath expressions that target them. It can also index scalar node values, to provide quick lookup based on individual values or ranges of values. It also records document hierarchy information for each node it indexes relations of parent–child, ancestor–descendant, and sibling.

Index size reduction from path subsetting

One of the advantages of XMLIndex is that it is very general: you need not specify which XPath locations to index; you need no prior knowledge of the XPath expressions that will be queried. By default, XMLIndex indexes all possible XPath locations in your XML data.

However, if you are aware of the XPath expressions that you are most likely to query, you can narrow the focus of XMLIndex indexing and thus improve performance. Having fewer unnecessary indexes means that less space is required for indexing, which improves index maintenance during DML operations. Having fewer indexed nodes improves DDL performance, and having a smaller path table improves query performance.

You narrow the focus of indexing by pruning the set of XPath expressions (paths) corresponding to XML fragments to be indexed, specifying a subset of all possible paths. You can do this in two alternative ways:

- Exclusion – Start with the default behavior of including all possible XPath expressions, and exclude some of them from indexing.

- Inclusion – Start with an empty set of XPath expressions to be used in indexing, and add paths to this inclusion set.

You can specify path subsetting either when you create an XMLIndex index using CREATE INDEX or when you modify it using ALTER INDEX. In both cases, you provide the subsetting information in the PATHS parameter of the statement's PARAMETERS clause. For exclusion, you use keyword EXCLUDE. For inclusion, you use keyword INCLUDE for ALTER INDEX and no keyword for CREATE INDEX (list the paths to include). You can also specify namespace mappings for the nodes targeted by the PATHS parameter.

Asynchronous Index Maintenance

By default, XMLIndex indexing is maintained at each DML operation, so that it remains in sync with the base table. In some situations, you might not require this, and using possibly stale indexes might be acceptable. In that use case, you can decide to defer the cost of index maintenance, performing at commit time only or at some time when database load is reduced. This can improve DML performance. It can also improve index maintenance performance by enabling bulk loading of unsynchronized index rows when an index is synchronized.

Using XMLIndex on Oracle XML DB Repository

A database administrator can create an XMLIndex index on resources in Oracle XML DB Repository to improve querying of XML data or metadata (system-defined or user-defined). The DBMS_XDB_ADMIN package includes procedures for creating and maintaining XMLIndex indexes on Oracle XML DB Repository. After an XMLIndex has been created, a query on resources will be able to take advantage of it for optimal query execution.

XML Schema

Support for the Worldwide Web Consortium (W3C) XML Schema Recommendation is a key feature in Oracle XML DB. XML Schema specifies the structure, content, and certain semantics of a set of XML documents.

W3C Schema for Schemas

The W3C Schema Working Group publishes an XML schema, often referred to as the "schema for schemas". This XML schema provides the definition, or vocabulary, of the XML Schema language. An XML schema definition (XSD) is an XML document, that is compliant with the vocabulary defined by the schema for schemas. An XML schema uses vocabulary defined by W3C XML Schema Working Group to create a collection of type definitions and element declarations that declare a shared vocabulary for describing the contents and structure of a new class of XML documents.

XML Schema Base Set of Data Types Can be Extended

The XML Schema language provides strong typing of elements and attributes. It defines numerous scalar data types. This base set of data types can be extended to define more complex types, using object-oriented techniques such as inheritance and extension. The XML Schema vocabulary also includes constructs that you can use to define complex types, substitution groups, repeating sets, nesting, ordering, and so on. Oracle XML DB supports all of the constructs defined by the XML Schema Recommendation, except for redefines.

XML schemas are commonly used as a mechanism for checking (validating) whether XML instance documents conform with their specifications. Oracle XML DB includes XMLType methods and SQL functions that you can use to validate XML documents against an XML schema.

XML Schema Unifies Document Modeling and Data Modeling

In Oracle XML DB, you can use XML Schema to automatically create database tables and data types for storing XML data. This allows you to use a standard data model for all of your data, whether the data is structured, unstructured, or semi-structured.

Create XMLType Tables and Columns, Ensure DOM Fidelity

You can create XML schema-based XMLType tables and columns and optionally specify that they conform to pre-registered XML schemas, maintaining DOM fidelity.

Schema Evolution

A major challenge for developers using an XML schema with Oracle XML DB is how to deal with changes in the content or structure of XML documents. In some environments, the need for changes may be frequent or extensive, arising from new regulations, internal needs, or external opportunities. For example, new elements or attributes may need to be added to an XML schema definition, a data type may need to be modified, or certain minimum and maximum occurrence requirements may need to be relaxed or tightened.

In such cases, you need to "evolve" the XML schema so that new requirements are accommodated, while any existing instance documents (the data) remain valid (or can be made valid), and existing applications can continue to run.

If you do not care about any existing documents, you can of course simply drop the XMLType tables that are dependent on the XML schema, delete the old XML schema, and register the new XML schema at the same URL. In most cases, however, you need to keep the existing documents, possibly transforming them to accomodate the new XML schema.

Oracle XML DB supports two kinds of schema evolution.

Oracle XML DB supports two kinds of schema evolution. Each approach has its own PL/SQL procedure: DBMS_XMLSCHEMA.copyEvolve for copy-based evolution and DBMS_XMLSCHEMA.inPlaceEvolve for in-place evolution.

In-place Schema Evolution

In-place XML schema evolution makes changes to an XML schema without requiring that existing data be copied, deleted, and reinserted. In-place evolution is thus much faster than copy-based evolution. In general, in-place evolution is permitted if you are not changing the storage model and if the changes do not invalidate existing documents (that is, if existing documents are conformant with the new schema or can be made conformant with it).

Copy-based Schema Evolution

With copy-base schema evolution, all instance documents that conform to the schema are copied to a temporary location in the database, the old schema is deleted, the modified schema is registered, and the instance documents are inserted into their new locations from the temporary area. Although it does not have the same restrictions as does the in-place schema evolution, copy-based schema evolution can be time-consuming if there are a large number of associated instance documents.

Using XQuery to Query, Transform, and Access XML and Relational Data

XQuery 1.0 is the W3C standard language designed for querying, transforming, and accessing XML and relational data.

XQuery 1.0 is the W3C standard language designed for querying, transforming, and accessing XML and relational data. It is similar to SQL in many ways, but just as SQL is designed for querying structured, relational data, XQuery is designed especially for querying semistructured, XML data from a variety of data sources. You can use XQuery to query XML data wherever it is found, whether it is stored in database tables, available through Web Services, or otherwise created on the fly. In addition to querying XML data, XQuery can be used to construct XML data. In this regard, XQuery can serve as an alternative or a complement to both XSLT and the other SQL/XML publishing functions, such as XMLElement.

XQuery builds on the Post-Schema-Validation Infoset (PSVI) data model, which unites the XML Information Set (Infoset) data model and the XML Schema type system. XQuery defines a new data model based on sequences: the result of each XQuery expression is a sequence. XQuery is all about manipulating sequences. This makes XQuery similar to a set-manipulation language, except that sequences are ordered and can contain duplicate items. XQuery sequences differ from the sequences in some other languages in that nested XQuery sequences are always flattened in their effect.

In many cases, sequences can be treated as unordered, to maximize optimization – where this is available, it is under your control. This unordered mode can be applied to join order in the treatment of nested iterations (for), and it can be applied

to the treatment of XPath expressions (for example, in /a/b, the matching b elements can be processed without regard to document order).

An XQuery sequence consists of zero or more items, which can be either atomic (scalar) values or XML nodes. Items are typed using a rich type system that is based upon the types of XML Schema. This type system is a major change from that of XPath 1.0, which is limited to simple scalar types such as Boolean, number, and string.

XQuery is a functional language. As such, it consists of a set of possible expressions that are evaluated and return values (which, in the case of XQuery, are sequences). As a functional language, XQuery is also referentially transparent, generally: the same expression evaluated in the same context returns the same value.

Exceptions to this desirable mathematical property include the following:

- XQuery expressions that derive their value from interaction with the external environment. For example, an expression such as fn:current-time(...) or fn:doc(...) does not necessarily always return the same value, since it depends on external conditions that can change (the time changes; the content of the target document might change).

  In some cases, like that of fn:doc, XQuery is defined to be referentially transparent within the execution of a single query: within a query, each invocation of fn:doc with the same argument results in the same document.

- XQuery expressions that are defined to be dependent on the particular XQuery language implementation. The result of evaluating such expressions might vary between implementations. Function fn:doc is an example of a function that is essentially implementation-defined.

Referential transparency applies also to XQuery variables: the same variable in the same context has the same value. Functional languages are like mathematics formalisms in this respect and unlike procedural, or imperative, programming languages. A variable in a procedural language is really a name for a memory location; it has a current value, or state, as represented by its content at any time. A variable in a declarative language such as XQuery is really a name for a static value.

FLWOR Expressions

FLWOR is the most general expression syntax in XQuery. FLWOR (pronounced "flower") stands for for, let, where, order by, and return. A FLWOR expression has at least one for or let clause and a return clause; single where and order by clauses are optional.

- for – Bind one or more variables each to any number of values, in turn. That is, for each variable, iterate, binding the variable to a different value for each iteration.

At each iteration, the variables are bound in the order they appear, so that the value of a variable $earlier that is listed before a variable $later in the for list, can be used in the binding of variable $later. For example, during its second iteration, this expression binds $i to 4 and $j to 6 (2+4):

for $i in (3, 4), $j in ($i, 2+$i)

- let – Bind one or more variables.

  Just as with for, a variable can be bound by let to a value computed using another variable that is listed previously in the binding list of the let (or an enclosing for or let). For example, this expression binds $j to 5 (3+2):

  let $i := 3, $j := $i + 2

- where – Filter the for and let variable bindings according to some condition. This is similar to a SQL WHERE clause.
- order by – Sort the result of where filtering.
- return – Construct a result from the ordered, filtered values. This is the result of the FLWOR expression as a whole. It is a flattened sequence.

Expressions for and let function similarly to a SQL FROM clause; where acts like a SQL WHERE clause; order by is similar to ORDER BY in SQL; and return is like SELECT in SQL. In other words, except for the two keywords whose names are the same in both languages (where, order by), FLWOR clause order is more or less opposite to the SQL clause order, but the meanings of the corresponding clauses are quite similar.

Note that using a FLWOR expression (with order by) is the only way to construct a sequence in any order other than document order.

SQL/XML XQuery Functions

Oracle XML DB supports SQL/XML 2006 standard XMLQuery, XMLTable, and XMLExists functions for passing XQuery expressions into SQL statements.

- XMLQuery – used in the SQL SELECT clause for generating XML from queries.
- XMLTable – used in the SQL FROM clause for mapping XML data into a relational table format.
- XMLExists – used in the SQL WHERE clause for processing a query based on criteria specified in XQuery expression.

Rewrite for XQuery

Oracle's query rewrite technology for XQuery expressions in SQL/XML functions brings high performance and scalability to XML applications.

Oracle's query rewrite technology for XQuery expressions in SQL/XML functions brings high performance and scalability to XML applications. SQL/XML functions and their corresponding XMLType methods use XQuery expressions to search collections of XML documents and to access a subset of the nodes contained

within an XML document. In many cases, Oracle XML DB is able to rewrite such expressions to code that executes directly against the underlying database objects.

Oracle XML DB provides the following ways of evaluating XQuery expressions that operate on XMLType columns and tables, depending on the XML storage method used:

- Structured storage – Oracle XML DB attempts to translate the XQuery expression in a SQL/XML function into an equivalent SQL query. The SQL query references the object-relational data structures that underpin a schema-based XMLType. This process is referred to as XQuery rewrite. It can occur when performing queries and UPDATE operations. In addition, B-tree indexes, full-text indexes, and function-based indexes on the underlying object-relational tables, can be used to evaluate XQuery expressions for structured storage.
- Binary XML storage – Oracle XML DB can evaluate XPath expressions in different ways: using a function-based index, using XMLIndex, and using single-pass streaming. Single-pass streaming means evaluating a set of XPath expressions in a single scan of binary XML data. During query compilation, the cost-based optimizer picks the fastest combination of the methods. A secondary full-text index can also be created on the VALUE column of the XMLIndex path table to allow optimal full-text query execution of XQuery ora:contains function.
- Unstructured storage – XMLIndex, and function-based indexes can be used to evaluate XPath expressions for unstructured storage. In addition:
    - In the absence of an XMLIndex index, Oracle XML DB evaluates the XPath expression using functional evaluation. Functional evaluation builds a DOM tree for each XML document, and then resolves the XPath programmatically using the methods provided by the DOM API. If the operation involves updating the DOM tree, the entire XML document must be written back to disk when the operation is completed.
    - If an XMLIndex can be used, then it is used instead of functional evaluation. Further, if a secondary full-text index is present on the VALUE column of the XMLIndex path table, it will be used for optimal full-text query execution of XQuery ora:contains function.

A major objective in designing Oracle XML DB is to provide XML/SQL duality.

SQL/XML Duality

A major objective in designing Oracle XML DB is to provide XML/SQL duality. XML programmers can leverage the power of the relational model when working with XML content and SQL programmers can leverage the flexibility of XML when working with relational content. This lets you use the most appropriate tools for a particular business problem.

XML/SQL duality means that the same data can be exposed as rows in a table and manipulated using SQL and XQuery, or exposed as nodes in an XML document and manipulated using techniques such as XQuery rewrite, DOM, and XSL transformation. Access and processing techniques are optimized to take advantage of the underlying storage formats and indexes.

These features provide simple solutions to common business problems. For example:

- You can use Oracle XML DB SQL/XML functions to generate XML data directly from a SQL query. You can then transform the XML data into other formats, such as HTML, using the database-resident XSLT processor.
- You can access XML content, without converting between different data formats, for use in SQL queries, on-line analytical processing (OLAP), and business-intelligence/data warehousing operations.
- You can perform text, spatial, and multimedia operations on XML content.

SQL Standard SQL/XML Functions

SQL/XML 2006 standard defines mapping between SQL, XML, and XQuery constructs, as well as functions for querying, accessing, and generation of XML.

Oracle XML DB provides the SQL functions defined in the SQL/XML standard.

Oracle XML DB provides the SQL functions defined in the SQL/XML standard. This standard is defined by specifications prepared by the International Committee for Information Technology Standards (INCITS) Technical Committee H2. INCITS is the main standards body for developing standards for the syntax and semantics of database languages, including SQL.

SQL/XML functions fall into two categories:

- Functions that you can use to query and access XML content as part of normal SQL operations. XMLQuery, XMLTable, and XMLExists functions belong to this category.
- Functions that you can use to generate XML data from the result of a SQL query. These include XMElement, XMLAttributes, XMLAgg, and XMLForest functions.

With SQL/XML functions you can address XML content in any part of a SQL statement. These functions use XQuery or XPath expressions to traverse the XML structure and identify the nodes on which to operate. The ability to embed XQuery and XPath expressions in SQL statements greatly simplifies XML access.

Relational Views from XML Data and XML Views from Relational Data

Supported by Oracle's XQuery write technology, Oracle XML DB provides seamless integration of XML and relational data. Oracle's implementation of SQL/XML enables high performance XML publishing and enables XQuery to

operate directly on relational data. Conversely, relational views of XML content also allow SQL operations to be performed efficiently on XML.

Use XML Views to Wrap Relational Data

You can choose to wrap existing relational and object-relational data into XML format using XMLType views. XMLType views can be created by using SQL/XML XMLQuery or XML generation functions (e.g., XMLAgg, XMLForest). Using SQL XMLQuery function with XQuery ora:view function provides flexible means for publishing relational data into complex-structured XML.

Map XML Data to Relational Views

While XML data has become prevalent, there are a large number of existing relational database applications that cannot process XML data. Oracle XML DB provides efficient support of SQL/XML XMLTable function and its COLUMNS clause for mapping XML data into relational views. By taking advantage of XQuery rewrite technology, storage models, and indexing schemes, downstream processing on a relational view created with XMLTable function can approach pure-relational performance.

XMLType Update, Transformation, and Diff/Patch Operations

Oracle XML DB provides comprehensive and efficient support for all aspects of XML processing. In addition to storage, indexing, and querying of XML content, updating , transforming, comparing, and patching XML documents are also essential operations in XML applications.

Update XML Content Stored in Oracle XML DB

Oracle XML DB allows fine-grained update operations on XML content using either structured or SecureFile binary XML storage models. Update operations can either replace the entire contents of a document or parts of a document. Partial updates on XML documents is very powerful, particularly when you make small changes to large documents, as it can significantly reduce the amount of network traffic and disk input-output required to perform the update.

There are several SQL functions that you can use to update XML data incrementally without replacing the entire surrounding XML document.

- updateXML – Replace XML nodes of any kind.
- insertChildXML – Insert XML element or attribute nodes as children of a given element node.
- insertXMLbefore – Insert XML nodes of any kind immediately before a given node (other than an attribute node).
- appendChildXML – Insert XML nodes of any kind as the last child nodes of a given element node.

- deleteXML – Delete XML nodes of any kind.

XSL Transformation and Oracle XML DB

The W3C XSLT Recommendation defines an XML language for specifying how to transform XML documents from one form to another. Transformation can include mapping from one XML schema to another or mapping from XML to some other format such as HTML or XHTML.

XSL transformation can be expensive in terms of the amount of memory and processing required. Both the source document and the style sheet need to be parsed and loaded into memory structures that allow random access to different parts of the documents. Most XSL processors use DOM to provide the in-memory representation of both documents. The XSL processor then applies the style sheet to the source document, generating a third document.

Oracle XML DB implements a high performance XSLT processor that lets XSL transformations be performed inside the database.

Oracle XML DB implements a high performance XSLT processor that lets XSL transformations be performed inside the database. In this way, Oracle XML DB can provide XML-specific memory optimizations that significantly reduce the memory required to perform the transformation. It can also eliminate overhead associated with parsing the documents. These optimizations are only available when the source for the transformation is a schema-based XML document.

Oracle XML provides three ways to invoke the XSL processor:

- SQL function XMLtransform
- XMLType method transform()
- PL/SQL package DBMS_XSLPROCESSOR

Of these different ways to transform XML data, DBMS_XSLPROCESSOR has the best performance, because the style sheet is parsed only once.

Each of these XML transformation methods takes as input a source XML document and an XSL style sheet in the form of XMLType instances. For SQL function XMLtransform and XMLType method transform(), the result of the transformation can be an XML document or a non-XML document, such as HTML. However, for PL/SQL package DBMS_XSLPROCESSOR, the result of the transformation is expected to be a valid XML document. This means that any HTML generated by a transformation using package DBMS_XSLPROCESSOR must be XHTML, which is both valid XML and valid HTML.

Comparing and Patching XML Documents with XMLDiff and XMLPatch

You may need to compare two versions of XML documents to see what has changed. It is also often necessary to apply the same changes to a large number of XML documents. To simplify these tasks, Oracle XML DB is providing a pair of new SQL functions – XMLDiff and XMLPatch.

- XMLDiff – this function compares the trees that represent the two input XML documents to determine differences.
- XMLPatch – this function takes an instance document generated by XmlDiff , and follows the instructions in the instance document to modify other XML documents as specified.

Full-Text Search of XML Data

Oracle supports full-text search on documents. If your documents are XML, then you can use the XML structure of the document to refine the full-text search.

Oracle supports full-text search on documents. If your documents are XML, then you can use the XML structure of the document to refine the full-text search. For example, you may want to find all purchase orders that contain the word "electric" using full-text search. If the purchase orders are in XML form, then you can restrict the search by finding all purchase orders that contain the word "electric" in a comment, or by finding all purchase orders that contain the word "electric" in a comment under line items. If your XML documents are of type XMLType, then you can project the results of your query using the XML structure of the document. For example, after finding all purchase orders that contain the word "electric" in a comment, you may want to return just the comments, or just the comments that contain the word "electric".

There are two ways to do a search that includes full-text search and XML structure:

- Include the structure inside the full-text predicate, using SQL function contains:

  ... WHERE contains(doc, 'electric INPATH (/purchaseOrder/items/item/comment)') > 0 ...

  Function contains is an extension to SQL, and can be used in any query. It requires a CONTEXT full-text index.

- Include the full-text predicate inside the structure, using the ora:contains XPath function:

  ... '/purchaseOrder/items/item/comment[ora:contains(text(), "electric")>0]' ...
  The ora:contains XPath function is an extension to XPath, and can be used in any call to existsNode, extract, or extractValue.

Oracle XML DB Repository

As one of the key differentiator of Oracle XML DB, the repository supports a rich set of services and APIs for managing XML content.

As one of the key differentiator of Oracle XML DB, the repository supports a rich set of services and APIs for managing XML content.

Foldering

Foldering is the keystone Oracle XML DB repository that supports storing content in the database in hierarchical structures, as opposed to traditional relational database structures. Foldering lets applications access hierarchically indexed content in the database using the FTP, HTTP(S), and WebDAV protocols as if the

database content were stored in a file system. Along with foldering support, Oracle XML DB repository adheres to the following terminology.

Resource – Any object or node in the repository hierarchy. Resources are identified by URLs.

Folder – A resource that can contain other resources. Sometimes called a directory.

Links – In addition to containing resources, a folder resource can contain links to other resources (files or folders). These repository links, sometimes called folder links, are not to be confused with document links, which correspond to the links provided by the XLink and XInclude standards, and which are also supported by Oracle XML DB. Repository links are navigational, folder–child links among repository resources; document links are arbitrary links among documents that are not necessarily repository resources.

Path name – A hierarchical name representing an absolute path to a resource. It is composed of a slash (/) representing the repository root, followed by zero or more path components separated by slashes. A path component cannot be only . or .., but a period (.) can otherwise be used in a path component.

Resource name (or link name) – The name of a resource within its parent folder. This is the rightmost path component of a path name. Resource names must be unique within their immediately containing folder, and they are case-sensitive.

Resource content – The body, or data, of a resource. This is what you get when you treat the resource as a file and ask for its content. This is always of type XMLType.

XDBBinary element – An XML element that contains binary data. It is defined by the Oracle XML DB XML schema. XDBBinary elements are stored in the repository whenever unstructured binary data is uploaded into Oracle XML DB.

Access control list (ACL) – A set of database users that are allowed access to one or more specific resources.

Resource Versioning

Oracle XML DB versioning provides a way to create and manage different versions of a resource in Oracle XML DB.

Oracle XML DB versioning provides a way to create and manage different versions of a resource in Oracle XML DB. When you update a resource such as a table or column, Oracle XML DB stores the pre-update contents as a separate resource version. Oracle XML DB versioning supports the following:

- Version control on a resource. You can turn version control on or off for an Oracle XML DB Repository resource.
- Updating process of a version-controlled resource. When Oracle XML DB updates a version-controlled resource (VCR), it creates a new version of the resource. This new version will not be deleted from the database when you delete the version-controlled resource.

- Loading a VCR is similar to loading a resource in Oracle XML DB using the path name.
- Loading a version of the resource. To load a version of a resource, you must first find the resource object id of the version and then load the version using that id. The resource object id can be found from the resource version history or from the version-controlled resource itself.

Repository Resource Security

Object-level security is maintained for all resources stored in Oracle XML DB Repository.

Object-level security is maintained for all resources stored in Oracle XML DB Repository. Oracle XML DB uses a security mechanism that is based on access control lists (ACLs) to restrict access to any Oracle XML DB resource. An ACL is a list of access control entries (ACEs) that determine which users, roles, and groups have access to a given resource.

ACLs are a standard security mechanism that is used in some languages, such as Java, and some operating systems, such as Microsoft Windows. ACLs are also a part of the WebDAV standard, and Oracle XML DB resource ACLs act as WebDAV ACLs. Resource ACLs are enforced no matter how resources are accessed, whether by WebDAV, SQL, or any other way.

ACLs in Oracle XML DB are XML schema-based resources, stored and managed in Oracle XML DB. Each resource in Oracle XML DB Repository is protect ted by an ACL. Before a user performs an operation on a resource, the user privileges on the resource are checked. The set of privileges checked depends on the operation to be performed.

Repository Metadata

Data that you use is often associated with additional information that is not part of the content. To process it in different ways, you can use such metadata to group or classify data. For example, you might have a collection of digital photographs, and you might associate metadata with each picture, such as information about the photographic characteristics (color composition, focal length) or context (location, kind of subject: landscape, people).

An Oracle XML DB repository resource is an XML document that contains both metadata and data; the data is the contents of element Contents; all other elements in the resource contain metadata. The data of a resource can be XML, but it need not be.

You can associate resources in the Oracle XML DB repository with metadata that you define. In addition to such user-defined metadata, each repository resource also has associated metadata that Oracle XML DB creates automatically and uses (transparently) to manage the resource. Such system-defined metadata includes properties such as the owner and creation date of each resource.

Except for system-defined metadata, you decide which resource information should be treated as data and which should be treated as metadata. For a photo

resource, supplemental information about the photo is normally not considered to be part of the photo data, which is a binary image. For text, however, you sometimes have a choice of whether to include particular information in the resource contents (data) or keep it separate and associate it with the contents as metadata — that choice is often influenced by the applications that use or produce the data.

User-Defined Resource Metadata

User-defined resource metadata is itself represented as XML. It is XML data that is associated with other XML data, describing it or providing supplementary, related information.

User-defined metadata for resources can be either XML schema-based or not:

- Resource metadata that is schema-based is stored in separate (out-of-line) tables. These are related to the resource table by the resource OID, which is stored in the hidden object column RESID of the metadata tables.
- Resource metadata that is not schema-based is stored in a CLOB column in the resource table.

You can take advantage of schema-based metadata, in particular, to perform efficient queries and DML operations on resources. In this chapter, you will learn how to perform the following tasks involving schema-based resource metadata:

- Create and register an XML schema that defines the metadata for a particular kind of resource.
- Add metadata to a repository resource, and update (modify) such metadata.
- Query resource metadata to find associated content.
- Delete specific metadata associated with a resource and purge all metadata associated with a resource.

In addition, you will learn how to add non-schema-based metadata to a resource.

You can generally use user-defined resource metadata just as you would use resource data. In particular, versioning and access-control management apply.

Typical uses of resource metadata include workflow applications, enforcing user rights management, tracking resource ownership, and controlling resource validity dates.

Repository Events

You can use Oracle XML DB Repository to store and access data of any kind in the form of repository resources: files and folders. Repository resource operations include creating, deleting, locking, unlocking, rendering, linking, unlinking, placing under version control, checking in, checking out, unchecking out, opening, and updating. You can access data in the repository from any application. Sometimes

your application needs to perform certain actions whenever a particular repository operation occurs. For example, you might want to perform some move-to-wastebasket or other backup action whenever a resource is deleted.

Each repository operation is associated with one or more repository events. Your application can configure listeners for the events associated with resources it is concerned with.

Each repository operation is associated with one or more repository events. Your application can configure listeners for the events associated with resources it is concerned with. A repository event listener is a Java class or a PL/SQL package or object type. It comprises a set of PL/SQL procedures or Java methods, each of which is called an event handler. Each event handler processes a single event. A repository event listener can be configured for a particular resource or for the entire repository. A listener can be further restricted to apply only when a given node-existence precondition is met.

XLink and XInclude

A document-oriented, or content-management, application often tracks relationships, between documents, and those relationships are often represented and manipulated as links of various kinds. Such links can affect application behavior in various ways, including affecting the document content and the response to user operations such as mouse clicks.

W3C has two recommendations that are pertinent in this context, for documents that are managed in XML repositories:

- XLink – Defines various types of links between resources. These links can model arbitrary relationships between documents. Those documents can reside inside or outside the repository.
- XInclude – Defines ways to include the content of multiple XML documents or fragments in a single infoset. This provides for compound documents, which model inclusion relationships. Compound documents are documents that contain other documents; more precisely, they are file resources that include documents or document fragments. The included objects can be file resources in the same repository or documents or fragments outside the repository.

Using XLink and XInclude to represent document relationships provides flexibility for applications, facilitates reuse of component documents, and enables their fine-grained manipulation (access control, versioning, metadata, and so on). Whereas using XML data structure (an ancestor–descendents hierarchy) to model relationships requires those relationships to be relatively fixed, using XLink and XInclude to model relationships can easily allow for change in those relationships.

XLink and XInclude Links Model Document Relationships

XLink and XInclude links model arbitrary relationships among documents; the semantics of a relationship is determined by the applications that use the link; it is not inherent in the link itself. XLink and XInclude links can be mapped to Oracle XML DB document links. When document links target Oracle XML DB

Repository resources, they can (according to a configuration option) be hard or weak links; in this, they are similar to repository links in that context. Repository links can be navigated using file system-related protocols such as FTP and HTTP; document links cannot, but they can be navigated using the XPath 2.0 function fn:doc.

Oracle XML DB Repository APIs

Oracle XML DB is introducing a new XML DB Content Connector based on the Java Content Repository API (i.e., JSR 170).

Oracle XML DB is introducing a new XML DB Content Connector based on the Java Content Repository API (i.e., JSR 170). With this support, the standard content repository Java API can be used to access the Oracle XML DB Repository. Both Level 1 (read-only) and Level 2 (read/write) of JSR 170 are supported. In addition, Oracle implementation provides comprehensive mapping of the rich capabilities of Oracle XML DB Repository to extend this standard API.

JCR models the data in a content repository as a tree of nodes. Each node may have one or more child nodes. Every node has exactly one parent node, except for the root node, which has no parent.

In addition to child nodes, a node may also have one or more properties. A property is a simple name/value pair. For example, a node representing a particular file in the content repository has a property named jcr:created whose value is the date the file was created.

Each property has a property type. For example, the jcr:created property has the DATE property type, requiring its value to be a valid date/time.

Similarly, each node has a node type. For example, a node representing a file has node type nt:file. The node type controls what child nodes and properties the node may have or must have. For example, all nodes of type nt:file must have a jcr:created property.

Files and folders in Oracle XML DB Repository are represented as JCR nodes (and properties of those nodes). They can be created, retrieved, and updated using the JCR APIs.

Internet Protocols Support

Oracle XML DB also provides the Oracle XML DB protocol server. This supports standard Internet protocols, FTP, WebDAV, and HTTP(S), for accessing its hierarchical repository or file system. HTTPS provides secure access to Oracle XML DB Repository.

Session Pooling

Oracle XML DB protocol server maintains a shared pool of sessions. Each protocol connection is associated with one session from this pool. After a connection is closed the session is put back into the shared pool and can be used to serve later connections.

Session pooling improves performance of HTTP(S) by avoiding the cost of re-creating session states, especially when using HTTP 1.0, which creates new connections for each request. For example, a couple of small files can be retrieved by an existing HTTP/1.1 connection in the time necessary to create a database session. You can tune the number of sessions in the pool by setting session-pool-size in Oracle XML DB xdbconfig.xml file, or disable it by setting pool size to zero.

Native Oracle XML DB Web Services

Your applications can access Oracle Database using native Oracle XML DB Web services.

Service-oriented architecture has become the dominant computing architecture, and web services are the backbone of service-oriented architecture. Web services provide a standard way for applications to exchange information over the Internet and access services that implement business logic. Your applications can access Oracle Database using native Oracle XML DB Web services. One available service lets you issue SQL and XQuery queries and receive results as XML data. Another service provides access to all PL/SQL stored functions and procedures. You can customize the input and output document formats when you use the latter service; the WSDL is automatically generated by the native database Web services engine.

Simple Object Access Protocol (SOAP) 1.1 is the version supported by Oracle XML DB. Applications use the HTTP POST method to submit SOAP requests to native Oracle XML DB Web services. You can configure the locations of all native Oracle XML DB Web services and WSDL documents using the Oracle XML DB configuration file, xdbconfig.xml. You can also configure security settings for the Web services using the same configuration file.

End-to-End High Performance Programming Langauge APIs for Oracle XML DB

As a universal data format, XML has been broadly adopted in many use cases. However, with its verbose textual format, XML processing can become the critical path in an application's computing architecture. To provide scalable and high performance XML processing, Oracle solves this problem by introducing Binary XML Java API for the mid-tier to address the root cause of inefficiency in traditional XML processing. Furthermore, Oracle is also introducing the Scalable DOM API to optimize XML DOM processing by taking advantage of binary XML support along with sophisticated resource optimization schemes, such as data sharing, shadow copy, and lazy materialization.

ORACLE XML DB APPLICATION DEVELOPMENT TOOLS

Oracle XML DB is supported by a number of Oracle and 3rd party tools. SQL development for the database tier can benefit from Oracle SQL Developer. Web service development for service-oriented architecture will find Oracle JDeveloper to be a versatile tool.

Using Oracle SQL Developer for Database-Tier Development

Oracle SQL Developer is a free agile development tool for the development of the database-tier of XML applications. With SQL Developer, you can browse database objects, run SQL statements and SQL scripts, and edit and debug PL/SQL statements. You can also run any number of provided reports, as well as create and save your own. SQL Developer enhances productivity and simplifies your database development tasks.

Using Oracle JDeveloper for SOA Application Development

Oracle JDeveloper is a comprehensive development tool for XML applications. Client-server, N-tier, and SOA application architectures can be conveniently developed and deployed by using Oracle JDeveloper. It is a free integrated development environment with end-to-end support for modeling, developing, debugging, optimizing, and deploying Java and XML applications, and Web services.

ORACLE XML DB LIFE CYCLE MANAGEMENT TOOLS

Oracle XML DB provides a complete suite of tools for managing the life cycle of XML applications. The following sections describe the some of the essential tools for managing Oracle XML DB.

SQL*Loader Support

SQL*Loader supports loading XMLType tables. You can load XMLType data with SQL*Loader using either the conventional method or the direct-path method, regardless of how it is stored (structured, unstructured, or binary XML storage).

Export/Import Support

Oracle XML DB supports export and import of XMLType tables and columns that store XML data. You can export and import this data regardless of the XMLType storage format (structured, unstructured, or binary XML). However, Data Pump exports and imports XML data as text or binary XML data only. The underlying object-relational tables and columns used for structured storage of XMLType are thus not exported. Instead, they are converted to binary form and then exported as self-describing binary XML data. XMLType data stored as CLOB instances (unstructured storage) is exported as text.

Exporting and Importing Transportable Tablespaces

Using the transportable tablespace feature, you can move a set of tablespaces from one Oracle database to another, whether it is XML schema-based or non-schema-based. When you export using transportable tablespaces mode, only the metadata for tables (and their dependent objects) within a specified set of tablespaces are unloaded. You can then copy the tablespace data files to another Oracle database and perform a transport tablespace import. This is generally very fast, because it involves only copying the tablespace and re-creating the tablespace metadata.

Support for XMLType by Oracle Streams and Logical Standby

Oracle Streams and logical standby now support XMLType stored as CLOB. Both XML schema-based and non-schema-based XML data are supported. Streams can capture, propagate, and apply changes to XMLType data. Capture processes can capture changes to XMLType columns stored as CLOB columns, but capture processes cannot capture changes to XMLType columns stored object relationally or as binary XML. Apply processes can apply changes to XMLType columns stored as CLOB columns, stored object relationally, or stored as binary XML.

Oracle Enterprise Manager Support

As an integral component of an Oracle Database, XML application built with Oracle XML DB can be managed by DBAs with the Oracle Enterprise Manager tool. Oracle Enterprise Manager can now be used to manage the following Oracle XML DB features:

- configuration parameters
- repository resources
- repository access control lists (ACLs)
- XML Schemas
- XMLType tables and columns

CONCLUSION

With rapidly multiplying volumes and usage scenarios of XML data, you can no longer store and retrieve XML data in a file system or simply as LOBs in a database. Processing XML data in the mid-tier by building DOM trees has painfully met its limits. To build scalable web applications with XML, your database tier has to reduce the workload of your middle tier by processing XML data more intelligently and efficiently. Oracle XML DB meets this acute need of the information technology industry by offering a versatile array of capabilities.

XML DB in Oracle Database 11g provides the most comprehensive and efficient capabilities for developing, deploying, and managing highly scalable and versatile XML applications.

Built on the solid foundation of Oracle database, Oracle XML DB provides highly extensive capabilities for the efficient storage, retrieval, querying, generation, and management of massive volume of XML data. Further deepen its integration of XMLType, XML Schema processing, structured, binary, and unstructured storage, XML indexing, XQuery, content repository, SQL/XML, and management capabilities, XML DB continues to evolve in many key areas. In the new Oracle Database 11g, Oracle XML DB is forging ahead with a versatile and efficient binary XML storage model, powerful XML indexes, a high performance in-place schema evolution support, a vastly improved implementation of W3C XQuery 1.0 standard, a database-native web services, a standards-based (XLink and XInclude) support for compound documents, and more.

In short, XML DB in Oracle Database 11g provides the most comprehensive and efficient capabilities for developing, deploying, and managing highly scalable and versatile XML applications

Oracle Database 11g XML DB Technical Overview
July 2007
Author: Geoff Lee
Contributing Authors: Oracle XML DB Development 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