An Oracle White Paper
September 2009

Oracle Database 11g Release 2 XML DB New Features

ORACLE

Executive Overview 3
Introduction 3
XMLIndex Structured Component 4
XMLIndex Structured Component Data Type Mapping 5
Partitioning for XMLType Storage Options 7
Partitioning for XML Information Lifecycle Management 8
Equi-Partitioning for the Object-Relational Storage Option 8
Partitioning Binary XML Data Using Virtual Columns 10
XMLIndex Partitioning and Parallelism 11
XML Schema Management Optimization 12
Binary XML Performance Enhancements 12
Object-Relational Performance Enhancements 13
Cost-Based XQuery Optimization 14
Oracle XML DB Repository Enhancements 14
Read/Write Performance Enhancements 14
Access Control Enhancements 15
Oracle XML Developer's Kit (XDK) Enhancements 15
Performance Proof Points 15
XQuery and SQL/XML Operations 16
Binary XML Operations 16
XMLIndex Unstructured Component 17
XML Schema Management 17
Object-Relational Operations 17

XML DB Repository Operations 18

Conclusion 18

Executive Overview

As the volume of XML data rapidly multiplies and the workload grows, performance, scalability, XML schema management, and information lifecycle management have become the leading requirements for XML databases. To meet these increasing needs, new capabilities (e.g., XMLIndex structured component, partitioning) and numerous enhancements are introduced in Oracle Database 11g Release 2 XML DB to dramatically improve its performance, scalability, XML schema management, and XML information lifecycle management.

Introduction

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. XML usage scenarios fall into one of three categories:

- XML-centric applications
- Data exchange and integration
- Management of content and metadata

Oracle XML DB provides comprehensive capabilities to support all three usage categories operating in mission critical enterprise information management environments, where high performance, scalability, XML schema management, and information lifecycle management are essential. Oracle XML DB supports high performance and scalable XML data storage and retrieval by implementing innovative and optimized storage options, indexes, and standards (e.g., W3C, IETF, ISO/ANSI). In the following sections, new capabilities of Oracle XML DB in Oracle Database 11g Release 2 including XMLIndex structured component, partitioning, XML schema management, binary XML performance enhancements, object-relational performance enhancements, cost-based XQuery rewrite, repository read/write performance and access control enhancements, and XDK enhancements will be discussed in the context of high performance, scalability, as well as simplified XML schema and information lifecycle management. Finally, proof points of significant performance improvements over the previous release will also be discussed.

XMLIndex Structured Component

XMLIndex was introduced in Oracle Database 11g Release 1 to improve query performance of XML documents. Based on a path-value index structure along with asynchronous maintenance and path subsetting, XMLIndex is the indexing scheme of choice for use cases requiring binary XML or CLOB storage options where the XML document structure is highly variable or not known beforehand.

To provide even higher performance for use cases where queries are over structured components (e.g., authors and titles of books) of XML documents, Oracle Database 11g Release 2 introduces an XMLIndex for structured components. Structured XMLIndex uses relational content tables to organize structured islands of XML content. With these, structured XMLIndex is data-type aware, and each column of these index content tables can also be of a different scalar data type.

Creating the structured component of an `XMLIndex` index can be viewed as decomposing a structured portion of XML documents into a relational format. However, this is different from the object-relational storage option of `XMLType` in the following respects:

- A structured XMLIndex explicitly decomposes commonly queried portions of XML documents specified by a user. In contrast, object-relational `XMLType` storage involves automatic decomposition of an entire `XMLType` table or column.
- The structured component of an `XMLIndex` index applies to both XML schema-based and non-schema-based data, while object-relational `XMLType` storage applies only to data that is based on an XML schema.
- The decomposed data for a structured `XMLIndex` index is stored in addition to the `XMLType` data, as an index, rather than being the storage option for the `XMLType` data itself.

- Using a structured XMLIndex, the same data can be projected multiple times, as columns of different data type.

Since the index content tables for a structured XMLIndex are regular relational tables, secondary indexes on the columns of these index content tables can be created by using any standard relational indexes, including indexes that satisfy primary-key and foreign-key constraints. Domain indexes, such as an Oracle Text Context index, can also be created. However, the index content tables are transparent and read-only. Other than to DESCRIBE them and to create secondary indexes on them, they cannot be accessed directly.

The structured component of an XMLIndex index can also be viewed as a generalized function-based index. A function-based index is similar to a structured XMLIndex component that has only one relational column. Instead of creating multiple function-based indexes, a structured XMLIndex index should be considered along with secondary B-tree indexes on the columns of the structured XMLIndex content table. These columns are efficiently populated in a single scan of the input base document, which cannot be done with virtual columns. In addition, structured XMLIndex works well in cases where the XML has collections. On the other hand, while virtual columns can be defined for partitioning and constraining binary XML data, they cannot be used to project values within an XML collection.

Finally, Oracle XML DB supports the ANSI/ISO SQL/XML standard XMLTable function to enable relational access for an application even though the data is stored in XML. It allows business intelligence applications to use SQL GROUP BY and ORDER BY clauses, and window functions over XML. Structured XMLIndex can be used to speed up access of such queries, which are typically expressed as XMLTable views over XML.

XMLIndex Structured Component Data Type Mapping

While the relational index content tables of an XMLIndex structured component use SQL data types, XQuery expressions in queries use XML data types (XML Schema data types and XQuery data types). For a structured XMLIndex component to be applied to a query, its SQL data types must correspond to XML data types of XQuery expressions of a query by following the mapping rules in Table 1.

TABLE 1 XML AND SQL DATA TYPE MAPPING FOR XMLINDEX

XML DATA TYPE	SQL DATA TYPE
xs:decimal	INTEGER or NUMBER
xs:double	BINARY_DOUBLE
xs:float	BINARY_FLOAT
xs:date	DATE, TIMESTAMP WITH TIMEZONE

XML DATA TYPE	SQL DATA TYPE
xs:dateTime	TIMESTAMP, TIMESTAMP WITH TIMEZONE
xs:dayTimeDuration	INTERVAL DAY TO SECOND
xs:yearMonthDuration	INTERVAL YEAR TO MONTH

Be aware that XQuery typing rules can automatically change the data type of a subexpression to ensure coherence and type-checking. For example, if a non-schema-based document is queried using the XPath expression `/PurchaseOrder/LineItem [@ItemNumber = 25]`, the subexpression `@ItemNumber` is untyped, and it is then automatically cast to `xs:double` by the XQuery = comparison operator. To index this data using an `XMLIndex` structured component, `BINARY_DOUBLE` must be used as the SQL data type.

If the XML and SQL data types involved do not have a built-in one-to-one correspondence, then they must be specified to correspond according to Table 1, in order for the index to be picked up for a query. There are two ways to do this:

- Make the index correspond to the query:
  Define (or redefine) the column in the structured index component, so that it corresponds to the XML data type. For example, if a query uses the XML data type `xs:double`, then define the index to use the corresponding SQL data type, `BINARY_DOUBLE`.
- Make the query correspond to the index:
  In a query, explicitly cast the relevant parts of an XQuery expression to data types that correspond to the SQL data types used in the index content table.

Examples 1 and 2 show how to cast an XQuery expression in a query to match the SQL data type used in the index content table.

Example 1. Making Query Data Compatible with Index Data – SQL Cast

```
SELECT count(*) FROM purchaseorder WHERE
XMLCast(XMLQuery('$p/PurchaseOrder/LineItem/@ItemNumber' PASSING
OBJECT_VALUE AS "p" RETURNING CONTENT) AS INTEGER)= 25;
```

Example 2. Making Query Data Compatible with Index Data – XQuery Cast

```
SELECT count(*) FROM purchaseorder WHERE
XMLExists('$p/PurchaseOrder/LineItem[xs:decimal(@ItemNumber) =
25]' PASSING OBJECT_VALUE AS "p");
```

In Example 1, the `XMLQuery` result is cast to SQL type `INTEGER`, which is compared with the SQL integer value 25. In Example 2, the value of attribute `ItemNumber` is cast in XQuery to the XML data type `xs:decimal`, which corresponds to the SQL data type

`INTEGER` used for the index. Both kinds of data-type conversion in these examples convert query data to make it type-compatible with the index content table.

Partitioning for XMLType Storage Options

Partitioning addresses key issues in supporting very large tables and indexes by allowing them to be decomposed into smaller and more manageable pieces called partitions, which are entirely transparent to an application. SQL queries and DML statements do not need to be modified in order to access partitioned tables. However, after partitions are defined, DDL statements can access and manipulate individual partitions rather than entire tables or indexes. This is how partitioning can improve the manageability of large database objects. Although each partition of a table or index must have the same logical attributes, such as column names, data types, and constraints, each partition can have separate physical attributes such as compression, physical storage settings, and tablespaces.

Partitioning offers these advantages:

- It enables data management operations such as data loading, index creation and rebuilding, and backup/recovery at the partition level rather than on the entire table. This results in significantly reduced times for these operations.
- It improves query performance. In many cases, the results of a query can be retrieved by accessing a subset of partitions, rather than the entire table. For some queries, this technique (called partition pruning) can provide order-of-magnitude gains in performance.
- It significantly reduces the impact of scheduled downtime for maintenance operations. Partition independence for partition maintenance operations allows concurrent maintenance operations on different partitions of the same table or index. Concurrent query and DML operations can be executed against partitions that are unaffected by maintenance operations.
- It increases the availability of mission-critical databases if critical tables and indexes are divided into partitions to reduce the maintenance windows, recovery times, and impact of failures.
- Parallel execution provides advantages to optimize resource utilization, and minimize execution time. Parallel execution against partitioned objects is crucial for scalability in a clustered environment. Parallel execution is supported for queries as well as for DML and DDL operations.

Therefore, partitioning is useful for many different types of applications, particularly applications that manage large volumes of data. OLTP systems often benefit from

improvements in manageability and availability, while data warehousing systems benefit from performance and manageability.

Partitioning for XML Information Lifecycle Management

Information Lifecycle Management (ILM) is a set of processes and policies for managing data throughout its useful life. One important component of an ILM strategy is determining the most appropriate and cost-effective medium for storing data at any point during its life time: newer data used in day-to-day operations is stored on the fastest, most highly-available storage tier, while older data which is accessed infrequently may be stored on a less-expensive and less-performant storage tier. Older data may also be updated less frequently in which case it makes sense to compress the data and store it as read-only.

Partitioning provides the fundamental technology that enables data in tables to be stored in different partitions. The most basic of ILM operations, archiving older data and purging or removing it from the database, can be orders of magnitude faster when using partitioning.

With rapidly multiplying volumes of XML data in the enterprises, XML information lifecycle management has become a high priority for database administrators. To this end, Oracle Database 11g Release 2 introduces partitioning support for XML data stored with either the object-relational or the binary XML storage option.

Equi-Partitioning for the Object-Relational Storage Option

When an `XMLType` table or a relational table with an `XMLType` column is stored object-relationally and is partitioned using list, range, or hash partitioning, any ordered collection tables (OCTs) within the data are automatically partitioned accordingly, by default. This equi-partitioning means that the partitioning of an OCT follows the partitioning scheme of its parent (base) table. There is a corresponding OCT partition for each partition of the base table. A child element is stored in the OCT partition that corresponds to the base-table partition of its parent element.

Storage attributes for a base table partition are, by default, also used for the corresponding child OCT partitions. These storage attributes for a given OCT partition can be overridden by a user. Similarly, by default, the name of an OCT partition is the same as its base (parent) table, but an OCT partition name can also be specified.

Partitioning information for an `XMLType` base table can be specified in two ways:

- During XML schema registration, using XML Schema annotation `xdb:tableProps`
- During table creation using `CREATE TABLE` as shown in Example 3

Example 3. Specifying Partitioning Information during Table Creation

```
CREATE TABLE purchaseorder OF XMLType XMLSCHEMA
"http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.
xsd"
  ELEMENT "PurchaseOrder"
  VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE
lineitem_table
((PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
  PARTITION BY RANGE (XMLDATA.Reference)
      (PARTITION p1 VALUES LESS THAN (1000)
VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE
lineitem_p1 (STORAGE (MINEXTENTS 13)),
       PARTITION p2 VALUES LESS THAN (2000)
VARRAY "XMLDATA"."LINEITEMS"."LINEITEM" STORE AS TABLE
lineitem_p2 (STORAGE (MINEXTENTS 13)));
```

Example 3 also shows how object storage options for the individual OCT partitions can be specified using the `STORAGE` clauses.

Partition Maintenance

When partition maintenance is performed on the base (parent) table, corresponding maintenance is automatically performed on the OCT (child) tables as well.

However, there are a few exceptions to the general rule of performing partition maintenance only on the base table. In the following cases, maintenance should be performed on an OCT table:

- Modifying the default physical storage attributes of a collection partition
- Modifying the physical storage attributes of a collection partition
- Moving a collection partition to a different segment, possibly in a different tablespace
- Renaming a collection partition

For example, if the tablespace of a base table is changed, that change is not cascaded to its child OCT partitions. `ALTER TABLE MODIFY PARTITION` must be used manually on the OCT partitions to change their tablespace.

Other than these exceptional operations, all partition maintenance tasks are performed on the base table only. This includes operations such as adding, dropping, and splitting a partition.

Online partition redefinition is also supported for OCTs; that is, non-partitioned OCTs can be copied to partitioned OCTs during online redefinition of a base table. Typically, parameter values `copy_indexes => 0` and `copy_constraints => false` are specified for PL/SQL procedure `DBMS_REDEFINITION.copy_table_dependents`, to protect the indexes and constraints of the newly defined OCTs.

Partitioning Binary XML Data Using Virtual Columns

XML data has its own structure, which, except for object-relational storage of `XMLType`, is not reflected directly in database structure. That is, individual XML elements and attributes are not mapped to individual database columns or tables. To partition XML data stored as binary XML, virtual columns must be created to represent the XML data of interest and then used to define the partitions as needed. Range, list, and hash partitioning are supported for XML data stored as binary XML using a virtual column. To be discussed further in the next section, XMLIndex indexes local to range or list partitioned binary XML data can also be created to improve query and DML performance. However, an XMLIndex index or an Oracle Text Context index cannot be created for hash partitioned binary XML data in the current release. Note also that `XMLType` is an abstract data type, virtual columns created on an `XMLType` table are therefore hidden. For example, they do not show up in `DESCRIBE` statements.

A virtual column based on an XML element or attribute can be created by defining it in terms of a SQL expression that involves that element or attribute. SQL/XML functions `XMLCast` and `XMLQuery` are used to do this, as shown in Example 4. The XQuery expression argument to function `XMLQuery` must be a simple XPath expression that uses only the child and attribute axes.

Example 4. Partitioning a Binary XML Table using Virtual Columns

```
CREATE TABLE po_binaryxml OF XMLType
  XMLTYPE STORE AS BINARY XML
  VIRTUAL COLUMNS
    (DATE_COL AS (XMLCast(XMLQuery('/PurchaseOrder/@orderDate'
         PASSING OBJECT_VALUE RETURNING CONTENT) AS DATE)))
  PARTITION BY RANGE (DATE_COL)
    (PARTITION orders2001 VALUES LESS THAN (to_date('01-JAN-
2002')),
     PARTITION orders2002 VALUES LESS THAN (MAXVALUE));
```

Example 4 partitions an `XMLType` table using a virtual column, `DATE_COL`, which targets the `orderDate` element in a purchase-order document. The guidelines below should be followed:

- To use a virtual column for partitioning, its data type must be constant. In the case where the `XMLType` data in the column or table is mixed, with some documents being encoded using an XML schema and others being encoded without using any schema, the functional expression must be cast to ensure that the same data type is used for all rows in the virtual column.
- For best performance, choose the partitioning key using an XPath expression whose target occurs within 32 K bytes of the beginning of the XML document.

- A relational table that has an XMLType column cannot be partitioned by using that column to define virtual columns of XML data.

XMLIndex Partitioning and Parallelism

In addition to partitioning an XMLType table, or a table with an XMLType column using range or list partitioning, a locally partitioned XMLIndex index can be created on the partitioned base table by specifying the keyword LOCAL. When created, the index and all of its storage tables are locally equi-partitioned with respect to the base table. If the keyword LOCAL is not used, then an XMLIndex index cannot be created on a partitioned table. If a table is hash-partitioned, then an XMLIndex index cannot be created on it in the current release.

In addition, a PARALLEL clause (with optional degree) can be used when creating or altering an XMLIndex index to ensure that index creation and maintenance are carried out in parallel. If the base table is partitioned or enabled for parallelism, then this can improve the performance for both DML operations (INSERT, UPDATE, DELETE) and index DDL operations (CREATE, ALTER, REBUILD). Although specifying parallelism for an index can also consume more storage as a result of storage parameters being applied separately to each query server process, a partitioned base table enabled for parallelism does offer more opportunities to improve performance.

Example 5 creates an XMLIndex index with a parallelism degree of 10. If the base table is partitioned, then this index is equi-partitioned.

Example 5. Creating an XMLIndex Index in Parallel

```
CREATE INDEX po_xmlindex_ix ON sale_info (sale_po_clob) INDEXTYPE
IS XDB.XMLIndex
  LOCAL PARALLEL 10;
```

In Example 5, the path table and the secondary indexes are created with the same parallelism degree as the XMLIndex index itself, 10, by inheritance. Different parallelism degrees for these can be specified by using separate PARALLEL clauses. Example 6 demonstrates this. Again, because of keyword LOCAL, if the base table is partitioned, then this index is equi-partitioned.

Example 6. Using Different PARALLEL Degrees for XMLIndex Internal Objects

```
CREATE INDEX po_xmlindex_ix ON sale_info (sale_po_clob) INDEXTYPE
IS XDB.XMLIndex
  LOCAL PARAMETERS ('PATH TABLE po_path_table (PARALLEL 10)
                     PIKEY INDEX po_pikey_ix
                     VALUE INDEX po_value_ix (PARALLEL 5)')
NOPARALLEL;
```

In Example 6, the `XMLIndex` index itself is created serially, because of `NOPARALLEL`. The secondary index `po_pikey_ix` is also populated serially, because no parallelism is specified explicitly for it, it inherits the parallelism of the `XMLIndex` index. The path table itself is created with a parallelism degree of 10, and the secondary index value column, `po_value_ix`, is populated with a degree of 5, due to their explicit parallelism specifications.

Any parallelism specified for an `XMLIndex` index, its path table, or its secondary indexes is exploited during subsequent DML operations and queries. Note that there are two places where parallelism can be specified for `XMLIndex`: within the `PARAMETERS` clause parenthetical expression and after it. Note that parallel specification inside the parentheses applies only to the `XMLIndex` unstructured component in the current release.

XML Schema Management Optimization

As complex industry schemas proliferate, efficient XML schema management has become a key requirement for an XML database. In Oracle Database 11g Release 2, the XML schema management operations have been improved in the following areas that resulted in significant improvements in elapsed time, memory usage, and cursor sharing.

- Schema Validator Cache
  - Significantly improves XML schema validation, especially for small documents. Note that validation is essential for accurate binary XML encoding.
- Schema registration performance
  - Eliminated internal and external memory fragmentation
  - Optimized schema loading
  - Able to handle complex industry schemas
  - Elapsed time and memory usage improved by 200 times in some cases

These enhancements have dramatically streamlined XML schema registration for complex industry schemas including US GAAP, FixML, ACORD, SDMX, FPML, OAGIS, MPEG7, NIEM, Global JXDM, GML, HL7, MPEG-7, and KML. Performance of XML schema-based binary XML and object-relational storage options also benefit from these enhancements.

Binary XML Performance Enhancements

Binary XML was introduced in Oracle Database 11g Release 1 as a new storage option for XMLType. Its goals include schema flexibility, better support for the document

abstraction, improved end-to-end performance for client applications, and the ability to handle all artifacts of the XML information model with high performance. It allows users to store and process a wide variety of XML documents efficiently without knowing about their exact structure. In addition to XPath table caching, schema caching enhancements and improved full document retrieval, the following enhancements have also been introduced in Oracle Database 11g Release 2 to further improve the performance and scalability of binary XML storage option.

- More efficient decoder: XPaths are pushed down to the decoder so that it can look for the desired XPath rather than decoding the entire document
- Schema-optimized streaming XPath evaluation engine: Takes advantage of schema constraints to speed up XPath evaluation
- Document-level summary: Enables more efficient traversal of forward XPath axes during streaming XPath evaluation
- XPath caching: Identifies XPaths that are evaluated repeatedly and caches their results within the context of a single query
- Enhanced query rewrite optimizations: Pushes more query operations into the streaming XPath engine, thereby simplifying the query and improving query performance.

Object-Relational Performance Enhancements

In Oracle Database 11g Release 2, query rewrite of XQuery constructs related to XML schema complex type with inheritance has been implemented for the structured storage option. A predicate involving "xsi:type" and the XQuery construct "instance of" are now rewritten to underlying object-relational constructs to improve performance.

In addition, VARRAYs in XMLType columns and tables created separately from schema registration are now stored as order collection tables (OCTs) by default in Oracle Database 11g Release 2. This improves query performance significantly in most cases. To override this default behavior, the user can specify "STORE ALL VARRAYS AS [LOBS|TABLES]" syntax. Note that for default storage tables created during schema registration, VARRAYs are stored as OCTs by default starting in Oracle Database 11g Release 1.

Finally, fast-path insert for XMLType has been enhanced to improve scalability for ingesting XML documents into object-relational storage. The performance of XML fragment retrieval for structured storage has also been improved significantly.

Cost-Based XQuery Optimization

Several competing optimization alternatives can exist for queries with XQuery expressions, depending on various factors such as the `XMLType` storage option and indexing that are used. By default, Oracle XML DB follows a prioritized set of rules to determine which of the possible optimizations should be used for any given query and context. This behavior is referred to as rule-based XML query rewrite.

Optionally, Oracle XML DB can use cost-based XML query rewrite. In this mode, Oracle XML DB estimates the performance of the various XML optimization alternatives for a given query and chooses the combination that is expected to be most performant. Cost-based XQuery optimization can be imposed by using the optimizer hint `/*+ COST_XML_QUERY_REWRITE */`. In Oracle Database 11g Release 2, cost-based XQuery Optimization is enabled only for XMLType stored as structured storage.

Oracle XML DB Repository Enhancements

Oracle XML DB Repository was introduced in Oracle9i Release 2. New capabilities of Oracle XML DB Repository have been introduced in subsequent releases to further improve its performance, scalability, flexibility, and security. In Oracle Database 11g Release 2, additional enhancements for Oracle XML DB Repository include read/write performance and access control functionality.

Read/Write Performance Enhancements

A typical repository read/write operation may involve repository maintenance operations, event handling, and index maintenance operations. The following enhancements have been implemented in Oracle Database 11g Release 2.

- All repository contents are now stored in SecureFiles
- Create Operations
- Improved creation performance of documents conforming to system schemas (ACLs etc).
- Performance improvement of 1.5 times in creating of smaller documents.
- Retrieval Operations
- Improved read operations of repository objects for the class of queries where predicates involve `equals_path` and `under_path`.
- Full retrieval of documents improved by 2-3 times.
- Delete

- ACL OID Index for ACLs to accelerate lookup

Access Control Enhancements

Access control lists (ACLs) have also been enhanced extensively to provide fine-grained access control that a user can customize. Specifically,

- Privileges can now be defined and associated with users and roles with more flexibility
- Security classes are introduced as named collections of privileges
- Inheritance is now available for both ACLs and security classes
- Access control entries (ACEs) can stipulate start and end dates
- Access for application users and roles can be controlled differently from database users and roles

Oracle XML Developer's Kit (XDK) Enhancements

Oracle XML Developer Kit (XDK) is a set of components, tools and utilities implemented in Java, C, and C++ that are available in Oracle Database, Oracle Fusion Middleware, and Oracle JDeveloper releases. Oracle XDK simplifies the task of developing and deploying XML-enabled applications in an enterprise computing environment.

In Oracle Database 11g Release 2, streaming XPath and streaming XSLT features have been added to Java XDK. For XPath, the implementation of JAXP 1.3 XPath API now performs streaming processing for XPath expressions containing child, descendant, attribute, text, parent, and ancestor axes. For XSLT, common XSLT elements that can be executed in a streaming fashion are now automatically detected and processed more efficiently using less memory. In cases where the entire stylesheet can be processed via streaming, memory consumption can be further reduced by configuring Scalable DOM with FORWARD_READ access. The combination of Scalable DOM with FORWARD_READ and scalable XSLT allows processing of very large input XML documents using constant memory.

Performance Proof Points

In addition to new capabilities discussed in previous sections, an exhaustive list of enhancements was also implemented throughout Oracle XML DB to improve overall performance significantly over the previous release. In the following subsections, specific enhancements and corresponding performance proof points will be discussed.

XQuery and SQL/XML Operations

XQuery and SQL/XML operations are essential to XML applications. As a result of numerous refinements in XQuery rewrites and query processing, performance improvements have been observed in real world performance benchmarks as shown below.

TEST CASE	IMPROVEMENT
XMark	3.9 X
XML generation with large XQuery	60 X
XML generation with SQL/XML	1.1 X

Focused effort on enhancing specific XQuery and SQL/XML operations has resulted in performance improvements, as measured by using the XQuery Technology Compatibility Kit (TCK).

TECHNOLOGY IMPROVEMENT	PERFORMANCE IMPROVEMENT
Rewrite enabled for XQuery up to 32K	8X increase in size and complexity of supported XQuery operations
A newly implemented XML virtual machine (XVM) architecture	More than 20X improvement for functional evaluation of XQuery
Optimized execution of XQuery and SQL/XML operators	Up to 60X improvement for XML generation from relational data

Binary XML Operations

There are extensive improvements in binary XML implementation in Oracle Database 11g Release 2. Using the XMark benchmark with 10 MB documents, improvements in memory usage and elapsed time have been observed as shown below. Note that reduced memory usage directly translates to improved scalability.

OPERATION	MEMORY USAGE IMPROVEMENT	ELAPSED TIME IMPROVEMENT
Query	2X – 6X for large documents	5X – 15X for large documents
Insert/Load	1.4X	1.4X
Entire Document Retrieval	1X	3.5X
Update	2.3X	4.6X

End-to-end binary XML operations have also been improved for the Thin and Thick (OCI) JDBC drivers.

OPERATION	THIN DRIVER	OCI DRIVER
Schema-Based	3.8X	5.2X
Non-Schema-Based	4X	3.3X

XMLIndex Unstructured Component

Using XMark benchmark operations, both the elapsed time and the memory usage for unstructured XMLIndex have shown substantial improvements.

XMARK BENCHMARK OPERATIONS	ELAPSED TIME IMPROVEMENT	MEMORY USAGE IMPROVEMENT
Query	5.1X	1.9X
Insert/Load	1.6X	1.2X
Update	2X	3X
Bulk DML	3X – 25X for large number of documents	2X

Furthermore,

- Queries improve by 5 times on average for the XMark benchmark.
- Four queries improved up to 100 times.
- Asynchronous DML performance improved up to 2.5 times.
- Parallel index load and parallel query were enabled on a 4-node cluster improved scalability 75%.
- 40% compression was observed on XMLIndex when columnar compression was enabled.

XML Schema Management

XML schema management operations have been improved in several areas, resulting in significant improvements in elapsed time, memory usage, and cursor sharing.

OPERATION	ELAPSED TIME IMPROVEMENT	MEMORY USAGE IMPROVEMENT	CURSOR SHARING IMPROVEMENT
Registration	10.5X	2.7X	1.2X
Validation	6.4X	1.7X	5.2X
Insert/Load	4.1X	4X	5.2X
Update	7.9X	2.4X	1.2X

Object-Relational Operations

With implementation of fast-path insert and better query rewrite for DML operations and complex types with inheritance, performance of operations for the object-relational storage option has improved substantially as shown below.

OPERATION	PERFORMANCE IMPROVEMENT
Load	Up to 8X
Document Retrieval	Up to 1.5X

OPERATION	PERFORMANCE IMPROVEMENT
Validation	Up to 1.4X
Update	Up to 1.25X

XML DB Repository Operations

Because of the Oracle XML DB Repository enhancements introduced in Oracle Database 11g Release 2, performance improvements have been observed as shown below.

OPERATION	NON-XML	BINARY XML	ACLS
Create	1.5X	1.5X	14X
Delete	1.1X	1X	26X
Update	1.6X	1.4X	2.3X
Query (equals_path)	3X	3X	6X
Full Read	3.3X	5X	2X

Conclusion

As the leader among major database vendors since its introduction, Oracle XML DB has provided comprehensive support for diverse use cases by offering flexible, versatile, high performance, scalable, XML schema management, and information lifecycle management capabilities for its repository and multiple storage options. Numerous performance enhancements and new capabilities, such as structured XMLIndex and partitioning dramatically improve performance, scalability, XML schema management, and XML information lifecycle management. The extensive list of proof points demonstrated the comprehensive improvements in performance and scalability in Oracle Database 11g Release 2 XML DB. Consequently, XML application developers will be able to take advantage of expanded high performance capabilities, while database administrators will be able to achieve higher performance and scalability, as well as simplified XML schema management and XML information lifecycle management with Oracle Database 11g Release 2 XML DB.

Oracle Database 11g Release 2 XML DB New Features
September 2009
Author: Geoff Lee
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, 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.

0109