
ORACLE®

ORACLE®

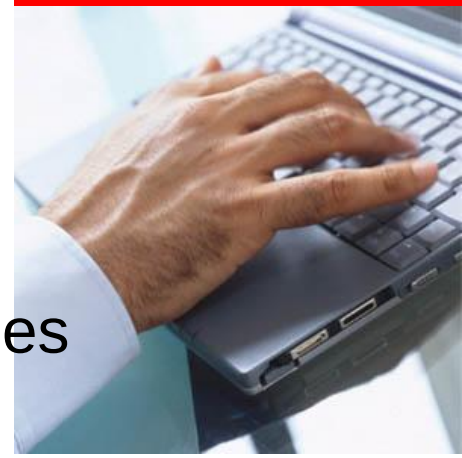
Managing XML Content with XML DB: Getting the Best Bang for the Buck

Sivasankaran Chandrasekar, XMLDB Development

Mark Drake, XMLDB Product Manager

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda



- XQuery, SQL/XML Best Practices & Guidelines
- Structured Storage Guidelines
- Binary XML with XMLIndex Storage Guidelines
- Q & A

XQuery, SQL/XML Best Practices

ORACLE

Using XMLQuery(), XMLExists()

SELECT XMLQuery('…' passing T.doc RETURNING CONTENT)
FROM table T
WHERE XMLExists('…' passing T.doc)

- Use XMLExists() to search
  - Typical database task to locating matching documents
  - XQuery expression must be index friendly to obtain good performance
  - Split complex XQuery expressions into index friendly and index unfriendly expressions joined by "AND"
- Use XMLQuery() to transform resulting documents
  - Typically runs on matching documents (much smaller set)
  - Can contain complex expressions and still obtain good performance

Performing XML DML operations

UPDATE Table t SET t.doc = DELETEXML(…)
WHERE XMLExists('…' passing t.doc)

- Use XMLExists() to identify documents being updated
  - As before, XQuery expression should take advantage of indexes to obtain good performance
- Use XPaths in DML operator to identify portion of document to update
- Piece-wise update operations are optimized
  - For Structured Storage, many operations rewritten to directly update relational columns
  - For Binary Storage, updates evaluated in streaming fashion and only updated portion changed on disk

Use XMLTable() for relational access

- XML documents are hierarchical
  - Can contain many master-detail relationalships
- Projecting out relational columns a common requirement

```
SELECT li.description, li.lineitem
FROM purchaseorder T, XMLTable('$p/PurchaseOrder/LineItems/LineItem'
                                        PASSING T.X AS “p”
                    COLUMNS lineitem NUMBER PATH '@ItemNumber',
                                description VARCHAR2(30) PATH 'Description',
                                partid NUMBER PATH 'Part/@Id',
                                unitprice NUMBER PATH 'Part/@UnitPrice',
                                quantity NUMBER PATH 'Part/@Quantity') li
WHERE li.unitprice > 30 and li.quantity < 20);
```

Use XMLTable() for relational access

- XQuery expressions should be storage/index friendly for good performance
- For structured storage, expressions should map down to relational columns for best performance
- For Binary XML storage with XMLIndex, expressions should be indexed
  - In particular, expression in predicates (WHERE clause)
- Multi level master-detail can be achieved by chaining multiple XMLTable clauses

Use XMLCast() for ORDER BY, GROUP BY

```
SELECT XMLCAST(XMLQUERY("$p/PurchaseOrder/@poDate"
            PASSING T.X
RETURNING CONTENT) AS DATE), COUNT(*)
FROM purchaseorder T
WHERE …
GROUP BY XMLCAST(XMLQUERY("$p/PurchaseOrder/@poDate"
                    PASSING T.X RETURNING CONTENT) AS DATE)
```

- If ordering or grouping multiple XPaths, use them in an XMLTable clause
- XMLTable allows expressions to be encapsulated in views for further BI analysis

XQuery on PL/SQL variable

```
DECLARE
   v_x XMLType;
   NumAcc NUMBER;
BEGIN
   v_x := XMLType(…); /* initialize xmltype variable */
   SELECT /*+ NO_XML_QUERY_REWRITE */
            XMLCAST(XMLQUERY('declare default element namespace
            "http://custacc";for $cust in $cadoc/Customer return
            fn:count($cust/Addresses/Address)'
   PASSING v_x AS "cadoc" RETURNING CONTENT) AS NUMBER)
   INTO NumAcc
   FROM DUAL;
END;
```

- Hint allows efficient DOM based evaluation
- XMLExists() can be used similarly

Datatype considerations

- XQuery specifies xs:untypedAtomic for non-schema documents
  - Different for schema-based documents
  - Can lead to confusing semantics
- Use explicit casting in XQuery and PASSING clause for numeric comparisons

$po/purchaseOrder[xs:decimal(@id)=$id]

PASSING T.X AS "po", CAST(:1 AS NUMBER) as "id“

- For non-numeric datatypes, XQuery can automatically cast to appropriate type of RHS

$po/purchaseOrder[@podate =xs:date($d)]

Accessing XML DB repository

- Repository data can be accessed using fn:doc and fn:collection
- Approach 1: Use SQL/XML and RESOURCE_VIEW

```
SELECT XMLQuery(…)
FROM RESOURCE_VIEW rv, purchaseorder p
WHERE ref(p) = XMLCast(XMLQuery('declare default element namespace
    "http://xmlns.oracle.com/xdb/XDBResource.xsd"; (: :) fn:data
    (/Resource/XMLRef)' PASSING rv.RES RETURNING CONTENT) AS REF
    XMLType) AND
equals_path(rv.RES, '/home/mydocs/podocs/1924.xml') = 1;
```

- Approach 2: Use XQuery pragma

```
for $doc in (#ora:defaultTable PURCHASEORDER #)
    {fn:doc("/home/mydocs/podocs/1924.xml")} …
```

XMLQuery() vs XMLTable() for top level XQuery

- Two variants for top level XQuery
  - SELECT * FROM XMLTable()
  - SELECT XMLQuery() FROM DUAL
- All rewrite optimizations are performed and appropriate indexes are leveraged
- Difference in semantics though:
  - XMLTable() construct produces many rows (one for each result item)
  - Avoids materializing large sequences and maps better to SQL's iterator model
  - XMLQuery() produces 1 row (the entire result sequence)

Structured Object Relational Storage Best Practices

ORACLE

Default tables, Nested tables & Indexes

- Use annotation xdb:defaultTable
- Allows your tables to be recognized in the explain plan output
- Nested tables can be named using the VARRAY STORE AS clause
- If tables already created, use DBMS_XMLSCHEMA_MANAGE RENAMECOLLECTIONTABLE procedure
- Create B-tree indexes on underlying relational tables and columns

Indexing predicates

- Create Indexes on XPaths used in predicates (WHERE clause)
- Using the XPath directly (approach 1):

```
CREATE INDEX po_reference_ix ON purchaseorder
(XMLCast(XMLQuery ('$p/PurchaseOrder/Reference'
PASSING po.OBJECT_VALUE AS "p" RETURNING CONTENT)
AS VARCHAR2(128)));
```

- If function based index created, use approach 2 to obtain name of indexed column and then create index

```
select XDB.DBMS_MANAGE_XMLSTORAGE.xpath2TabColMapping(
'PURCHASEORDER_TAB',NULL, '/ipo:purchaseOrder/ Reference ',
'"http://www.example.com/IPO" as "ipo"') from dual;
```

Indexing predicates

- Create indexes on collection tables
- Composite index
  - The column in the collection table corresponding to XML attribute or element that needs to be indexed
  - The NESTED_TABLE_ID column

WHERE
XMLExists('$p/PurchaseOrder/LineItems/LineItem/Part[@Id="717951002372"]'

SELECT XDB.DBMS_MANAGE_XMLSTORAGE.xpath2TabColMapping(
'PURCHASEORDER_TAB', NULL, '/ipo:purchaseOrder/items/item/Part/Id',
'"http://www.example.com/IPO" as "ipo"')
FROM dual)

CREATE INDEX xxx ON tab_name (col_name, NESTED_TABLE_ID)

Loading large documents

- Large XML documents are loaded without building complete DOM in memory
- Pipelined storage directly into collection tables as document is scanned
- Can handle arbitrarily large documents (proven to many GB size)
- Use FTP PUT or CreateResource() for optimal performance
  - Ensure that xdb:defaultTable is specified correctly for root element in schema document
  - Ensure all collections are mapped to tables (default)
- Can also use SQL Insert statement (11gR2)

Configuring large document load

- Configuration file /xdbconfig.xml has parameters that control the amount of memory used by the loading operation
- Can tune the memory usage and performance of a load (or retrieval) operation by varying:
  - xdbcore-loadableunit-size – indicates the maximum size to which a ***loadable unit*** (partition) can grow in Kilobytes. Default is 16 KB.
  - xdbcore-xobmem-bound – indicates the maximum size in kilobytes that a document is allowed to occupy in memory. Default is 1024 KB.

```
DECLARE
  v_cfg XMLType;
BEGIN
  SELECT updateXML(DBMS_XDB.cfg_get(),
                   '/xdbconfig/sysconfig/xdbcore-xobmem-bound/text()',
                   '65536',
                   '/xdbconfig/sysconfig/xdbcore-loadableunit-size/text()',
                   '1024')
    INTO v_cfg FROM DUAL;
  DBMS_XDB.cfg_update(v_cfg);
  COMMIT;
END;
/
```

Binary XML Storage with XMLIndex Best Practices

ORACLE

Binary XML Streaming evaluation

- Multiple XPaths evaluated in a single streaming access over XML document
- Automatically used for XMLExists(), XMLQuery() and XMLTable() constructs
- Execution plan shows “XPATH EVALUATION”
- Reverse axes should be converted by user to more optimal forward axes

  $p/PurchaseOrder//a[@id=”abc1”]/..*

  ➔

  $p/PurchaseOrder/[a/@id=”abc1”]*
- Avoid descendant and wildcard axes if exact XPaths can be used.
  - Especially for large documents to avoid scanning unnecessary portions

Binary XML DML considerations

- For DML heavy workloads, enable caching on writes of underlying LOB column
- Use securefile for Binary XML storage if:
  - Documents are very large
  - Piece-wise updates form an important part of the workload
  - XMLIndex has been created on the XML column

XMLIndex considerations

- Query paradigm can determine choice of index
- XMLIndex (structured component)
  - Ideal for scalar value lookups
  - Speeding up queries on islands of structure
  - Author, Date, Title fields for example
  - Captures the “attributes” of an “entity” together using E/R Model
- XMLIndex (unstructured component)
  - Can handle wide variety of queries
  - Scalar value lookups and fragment identification/retrieval
  - Can index desired sub-trees including hierarchies

XMLIndex considerations

- Queries suited to XMLIndex (structured component)
  - Applications with stable XPaths
  - Query hierarchy is expressable as XMLTable constructs
  - Key value search having data types (dates, numbers)
- Queries suited to XMLIndex (unstructured component)
  - Applications with ad-hoc queries
  - Exact list of paths cannot be predicted (path subsetting required)
  - Queries requiring hierarchy computations
- XMLIndex can have both components
  - Mix of either class of queries

Structured XMLIndex guidelines

- Use Structured XMLindex in place of multiple functional indexes
  - All expressions can be accessed and populated efficiently
- Index and Query datatypes should correspond
- Use structured XMLindex for relational paradigm over XML data
  - Relational views can be defined over XML using XMLTable()
  - XMLTable() construct can be efficiently indexed
- Create secondary indexes for predicates

Structured XMLIndex guidelines

- Split up fragment extraction from document identification
  - Fragment extraction in SELECT list using XMLQuery()
  - Can efficiently use Binary XML streaming evaluation
  - Document identification in WHERE clause using XMLExists()
  - Can use structured XMLindex
- Use SQL ORDER BY instead of XQuery “order by”
  - Project out ordering keys as columns in XMLTable()
  - Use them in SQL ORDER BY clause
  - Better match for picking up structured XMLindex

Unstructured XMLIndex guidelines

- Consists of path table (path, value pairs) and secondary indexes (PIKEY, VALUE)
- Execution plan should show usage of unstructured XMLIndex
  - The Path table name should appear
  - One of the secondary indexes should be used
- Drop PIKEY index if only simple XPath filtering is needed
  - Instead create PATHID index directly on PATHID column
- Create datatype aware secondary indexes
  - CreateNumberIndex and CreateDateIndex

Path subsetting considerations

- Path subsetting reduces index size
  - Faster population
  - Faster queries
- The explain plan output should be used to verify that path subsetted index is being picked
- Ensure paths used in predicates are indexed
- Exclude paths that are better evaluated by Binary XML streaming evaluation

Using SQL/XML for Text Searches

- Example query:

```
SELECT XMLQuery()
FROM DocTable
WHERE XMLExists()
AND CONTAINS()
```

- Create XMLIndex on *DocTable* with optional paths (structured or unstructured)
- Create Text Index on *DocTable* with desired settings
- Optimizer uses appropriate combination of indexes

Text Search Considerations

- Queries
  - Use SQL operator CONTAINS
- Path restriction
  - If complete document need not be indexed, use custom data source
  - Use INPATH inside CONTAINS for path restricted search (PATH_SECTION_GROUP)
  - Keywords can also be matched inside a particular complex element (XML_SECTION_GROUP with tags)
- Disk space usage
  - Optimal space usage since keywords are present only in text index
  - XML structure and values only in XMLIndex

Organizing Documents

- Documents can be stored in binary XML table
- Can be queried using XQuery
- In addition, XML DB repository can be used if:
  - Documents need to be organized and searched using a hierarchical file/folder metaphor
  - Documents need to be accessed using path/URL based protocols like FTP, HTTP, WebDaV etc.
  - Document lifecycle needs to be managed using Content Management models:
    - Security policies using ACLs
    - Simple versioning

Organizing Schema-based Documents

- Documents conforming to XML schema controlled by xdb:defaultTable annotation
- Automatically route document to binary XML table
  - Using DBMS_XDB.CreateResource
  - Using Protocols

DocSchema.xsd

```
<element name="DocRoot" xdb:defaultTable="DOC_TAB"…>
```

Doc.xml

```
<DocRoot xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance "
                xsi:schemaLocation="DocSchema.xsd" >
    …..
```

Organizing Schema-less Documents

- Document metadata in XML DB repository
- Document contents in user's binary XML table
- 2 ways of creating hierarchy
  - Using repository events
    - User's PLSQL or Java code
    - Triggered during repository create operations
    - Can store content in desired binary XML table
    - XML DB repository stores "REF" to content
  - Staging table with path, document key
    - Can store content in desired binary XML table using regular options like SQL-Loader etc.
    - XML DB repository stores "REF" to content

Oracle XML DB DEMOgrounds Booths

- **Come by our DEMOgrounds booths to have one-on-one conversation with our team members**
  - Moscone West: W-41, W-44, and W-61

Tuesday Sessions

S317480: Managing XML Content with Oracle XML: Getting the Best Bang for the Buck
Moscone South, Rm 200

2:00 PM – 3:00 PM

S317428: ProQuest Use Case
Moscone South, Rm 200

5:00 PM – 6:00 PM

Wednesday Sessions

S317650 : S&P Use Case
Hotel Nikko, Nikko Ballroom I

10:00 AM – 11:00 AM

S319105: Interfacing with Your Database via Oracle XML DB

Hotel Nikko/Bay View

11:30 AM – 12:30 PM

S317648: PolarLake Use Case, XDK, and XQJ
Hotel Nikko Nikko Ballroom I

1:00 PM – 2:00 PM

Thursday Sessions

S317504: Waters Use Case and Structured XMLIndex
Moscone South, Rm 200

10:30 AM – 11:30 AM

S317528: Working with Complex XML Schemas: Not as Hard as You Might Think
Hotel Nikko Nikko Ballroom I

2:00 PM – 3:00 PM

S317657: XBRL Expert Panel - Using Oracle Database as an XBRL Repository
Hotel Nikko Nikko Ballroom I

3:30 PM – 4:30 PM