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
• XMLIndex Guidelines
• XML Storage Guidelines
• Using XMLDB Repository for XML document management Guidelines
• Common use-cases for XML DB
• Q & A
XQuery, SQL/XML
Best Practices
XML generation from relational data

• Application Use case
  • Data are in relational form with mature relational access paradigm
  • Need to generate different hierarchical XML shape for XML data presentation and XML data exchange
  • Need to generate complex XML report from relational data

• Recommendations
  • Use XMLElement(), XMLForest(), XMLAgg() SQL/XML generation function to define SQL/XML views over relational data
  • Not to use DBMS_XMLGEN(), DBMS_XMLQUERY(), XSU packages as they are less performing and less declarative than that of SQL/XML generation functions
  • Use pure XQuery with ora:view() for complex XML Report Generation
Generating Hierarchical XML using SQL/XML generation functions

- Mater-detail hierarchical XML views using XMLAGG with correlated sub-query

```sql
CREATE OR REPLACE VIEW DEPT_HV AS
SELECT XMLElement("dept", XMLAttributes(deptno as "deptno"),
(SELECT XMLAGG(XMLElement("emp",
XMLForest(empno, ename, job)))
FROM EMP e
WHERE e.DEPTNO = d.DEPTNO)) AS dept-emps
FROM DEPT d;
SELECT dept-emps FROM DEPT_HV;
```
JSP style XML Report generation using pure XQuery with ora:view() on table

```
SELECT XMLQUERY(''
<counties>
{for $c in ora:view("COMMERCE")
let $coc_county := $c/ROW/COC_COUNTY/text(),
$coc_name := $c/ROW/COC_NAME,
$coc_phone := $c/ROW/COC_PHONE/text()
order by $coc_county
return <county>
 <name>{$coc_county}</name><chamber
 phone="{$coc_phone}">{$coc_name/text()}</chamber>
 <attractions>{for $a in ora:view("ATTR_XMLT") where $coc_county
 = $a/attraction/county/text() return $a}          </attractions>
</county>}
</counties>' RETURNING CONTENT) FROM DUAL;
```
Querying Persistent XML using SQL/XML embedded with XQuery

- **Application Use case**
  - XML data are stored using structured (object relational) storage or binary XML storage in XMLType table or XMLType column
  - Need to search XML data using XQuery and then apply transformation using XQuery on search results or modifying XML from the research results

- **Recommendations**
  - Use XQuery embedded XMLQuery(), XMLExists(), XMLCast() standard SQL/XML querying functions
  - Not to use extract(), existsNode(), extractValue() as they are XPath 1.0 based with non-standard behavior
Querying Persistent XML using SQL/XML embedded with XQuery

• Use XMLExists() in WHERE clause to locate qualified XML document rows within the table (finding needles in the haystack)
  • Use explain plan to ensure relational index (structured storage) or XMLIndex (binary XML storage) access methods are applied for WHERE clause

• Use XMLQuery() in SELECT clause to apply XQuery operations on each qualified XML documents

```sql
SELECT XMLQuery('<paper_info>{$p/title, $p/author, $p/abstract, $p/affiliation}</paper_info>') PASSING object_value AS "p" RETURNING CONTENT
FROM DOCUMENT_TAB doc
WHERE XMLExists('/Paper[title="xmlquery" and author="zhen"]' PASSING object_value)
```
Full Text search on Persistent XML

- Use SQL CONTAINS() in WHERE clause to do full text search and XMLExists() embedded XQuery to do XML search
- Use explain plan to ensure proper combination of Text Index and XMLIndex/relational index are used

```
SELECT ...
FROM DOCUMENT_TAB doc
WHERE XMLExists('/Paper[title="xmlquery" and author="zhen"]'
               PASSING object_value)
AND CONTAINS(doc, 'optimization') > 0
```
Modifying XML persistence using XML Modification Functions

- Use `DELETEXML()`, `UPDATEXML()`, `INSERTXML()` etc operators on each qualified XML documents selected by `XMLExists()`
- For updating large XML document, use explain plan to ensure DML rewrite occurs

```sql
UPDATE DOCUMENT_TAB doc
SET object_value = DELETEXML(object_value, 'delete $p/afflication')
WHERE XMLExists('/Paper[title="xmlquery" and author="zhen"]' PASSING object_value)
```
Relational views over XML data using XMLTable construct

• Application UseCase
  • XML data are stored using structured (object relational) storage or binary XML storage in XMLType table or XMLType column
  • Need to provide relational views over XML to integrate with relational applications
  • Rich BI SQL query (group by, order by, window function) access of XML data using relational paradigm

• Recommendations
  • Use XMLTable construct to define master-detail relational view over XML data
  • Not to use table(xmlsequence()) as it is XPath 1.0 based with non-standard behavior
BI Query over relational view of XML data

• Use XMLTable chaining to step master-detail

```sql
CREATE OR REPLACE VIEW PapRec AS
SELECT v1.*, v2.*
FROM DOCUMENT_TAB,
XMLTABLE('$p/paper'  PASSING object_value AS "p"
    COLUMNS
    title varchar(100) PATH 'title',
    pubdate date PATH 'pubdate',
    affiliation varchar(20)  PATH 'affilation',
    authorList XML PATH 'authorList' XMLType) v1,
XMLTABLE('.,' PASSING v1.authorList
    COLUMNS
    author_name varchar(20) PATH 'authorName') v2;

SELECT p.author_name, count(*)
FROM PapRec p
WHERE p.pubdate > "2003-02-04" and p.affilation = "Oracle"
GROUP BY p.author_name
ORDER BY p.author_name;
```
Tuning BI Query over relational view of XML data

- Use explain plan to ensure query plan accessing XMLTable view is rewritten to access the underlying relational storage tables
  - For object relational storage of XML, the underlying internal relational tables are accessed
  - For binary storage of XML, use structured XMLIndex so that the underlying relational tables managed by structured XMLIndex are accessed
XQuery Usage Best Practices

- Use schema based storage to catch invalid XPath and leveraging additional query optimization based on presence of schema

  `SELECT XMLQuery('/paper/Pubdate' PASSING object_value RETURNING CONTENT) – invalid XPath FROM tab;`

- Use XQuery built-in type functions `xs:date()`, `xs:decimal()` etc to apply proper datatype search, in particularly useful for non-schema based document

  `SELECT * FROM tab
  WHERE XMLExists('/paper[pubDate > xs:date("2002-02-03")]' passing object_value)`

- Avoid using parent/ancestor axis, sibling axis, positional predicate in XPath whenever possible
XMLIndex Guidelines

Structured Component - New in 11gR2

Unstructured Component – Available in 11gR1 & 11gR2
UseCase: XML with structured component

- A document having structured data component, such as document title, document date, document affiliation, document author lists ... even though the overall XML is content driven
- Typical query want to find document with specific structured component data value
- Example query:

```sql
SELECT *
FROM DOCUMENT_TAB  doc
WHERE XML EXISTS(
  '$doc//document [ title = "indexing XML Techniques" and pubdate > xs:date("2007-03-01") and pubdate < xs:date("2007-12-31") and affiliation = "Oracle" ]' PASSING VALUE(doc) AS "doc")
```
Solution using Structured XMLIndex

• **What if we leverage** the spirit of structured object relational storage – *(schema-aware based de-composition)*?

• **Create a side pivot table with**
  • title, pubdate, affiliation, are pivoted as three columns of the table.

• **The example query can be rewritten using the side pivot table**

  ```sql
  SELECT *
  FROM DOCUMENT_TAB doc
  WHERE EXISTS(
    SELECT 1
    FROM PIVOT_TAB p
    WHERE p.title = "indexing XML Technique" AND
    p.pubdate > to_date("2007-03-01") AND
    p.pubdate < to_date("2007-12-31") AND
    p.affiliation = "Oracle" AND
    p.ROWID = doc.ROWID)
  ```
Structured XMLIndex Creation

- Example

```
CREATE INDEX paper_info ON DOCUMENT_TAB indextype is xdb.xmlindex
PARAMETERS(XMLTABLE('//document' PIVOT_TAB
  COLUMNS
  title varchar(100) PATH 'title',
  pubdate date PATH 'pubdate',
  affiliation varchar(20) PATH 'affiliation'))
```

- An XPath ‘//document’ used to identify nodes stored in each row of the table
- Multiple leaf data is projected out as columns of XMLTABLE
- Syntax similar to XMLTABLE construct in SQL/XML
Structured XML Index Layout

### XML data

<table>
<thead>
<tr>
<th>Row ID</th>
<th>Title</th>
<th>Affiliation</th>
<th>Pubdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Indexing XML Techniques</td>
<td>Oracle</td>
<td>2007-04-10</td>
</tr>
<tr>
<td>20</td>
<td>Object relational storage</td>
<td>Oracle</td>
<td>2003-03-15</td>
</tr>
</tbody>
</table>
Mater-detail Aspect of Structured XMLIndex

- What about collection Element Value?
- Store them in a separated nested table
- Structured XMLIndex with chaining option

```sql
CREATE INDEX paper_info ON PAPER_TAB indextype is xdb.xmlindex
XMLTABLE('//document' PIVOT_TAB
COLUMNS
  title varchar(100) PATH 'title',
  pubdate date PATH 'pubdate' ,
  affiliation varchar(20)  PATH 'affilation',
authorList XML PATH '//authorList' VIRTUAL
XMLTABLE '.' PIVOT_NTAB
COLUMNS
  authorname varchar(20) PATH 'authorName')
```
Master-detail Structured XMLIndex Query Rewrite

- Queries over the base XML storage can be “rewritten” to go against the XMLTable Index tables

```xml
$doc//document [ title = "indexing XML Techniques" and pubdate > xs:date("2007-03-01") and pubdate < xs:date("2007-12-31") and affiliation = "Oracle" and authorList/authorName = "J.Chan" ]

Can be rewritten as
```
SELECT .. FROM DOCUMENT_TAB doc
WHERE EXISTS( SELECT 1 FROM
    PIVOT_TAB p
    WHERE title = "indexing XML Technique" AND
        pubdate > to_date("2007-03-01") AND
        pubdate < to_date("2007-12-31") AND
        affiliation = "Oracle" AND p.ROWID = doc.ROWID
    AND EXISTS (SELECT 1 FROM PIVOT_NTAB nt
        WHERE pnt.authorName = "J.Chan" AND nt.nid =
p.nid))
```
Structured XMLIndex with Secondary Index

- Secondary relational indexes can be created on top of structured xmlindex tables
  - Bitmap index can be created on column with few number of distinct values
  - B+ tree composite indexes can be created on multi-columns
  - Accurate statistics can be built and maintained for different indexes
  - Value statistic collection is precise compared with unstructured XMLindex approach

- Text Index can be created on projected text column
  - Speed up text search in XML content
Summary for structured XMLIndex

- **Efficient Value Search of structured components of any XML**
  - No path matching needed during run time, value search has relational query performance
  - XML Storage & Schema Independent

- **Index size is small and light-weight**
  - No path information is stored in the indexed tables

- **Smooth Integration of XML with existing relational applications using** XMLTable Design Pattern
Unstructured XMLIndex

- Available since 11gR1
- Organizes required paths and values in single path table
- Allows easy indexing of interesting sub-trees
- Whole spectrum possible – single leaf element to everything
- Allows asynchronous maintenance
- Updates to document result in piece-wise index updates
## Unstructured XML Index Layout

<table>
<thead>
<tr>
<th>RID</th>
<th>PATHID</th>
<th>ORDER KEY</th>
<th>LOCATOR</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>/Document</td>
<td>1</td>
<td>Locator to get binary content</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>/Document/Title</td>
<td>1.1</td>
<td>Locator to get binary content</td>
<td>Indexing XML Techniques Oracle</td>
</tr>
<tr>
<td>10</td>
<td>/Document/Affiliation</td>
<td>1.2</td>
<td>Locator to get binary content</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>/Document/pubDate</td>
<td>1.3</td>
<td>Locator to get binary content</td>
<td>2007-04-10</td>
</tr>
<tr>
<td>20</td>
<td>/Document</td>
<td>1</td>
<td>Locator to get binary content</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>/Document/Title</td>
<td>1.1</td>
<td>Locator to get binary content</td>
<td>Object relational storage</td>
</tr>
</tbody>
</table>
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
Scalable XMLIndex Management

- Partitioning for large data sets
  - XMLIndex can be equi-partitioned with the document table
  - Leverages partition pruning for queries
  - Document and index partitions can be managed together

- Taking advantage of multiple CPUs
  - Building an index can be parallelized
  - Can leverage partitioning
  - Many queries can be run in parallel
  - Document ingestion can be run in parallel with indexing
    - Using asynchronous indexing
Incorporating Text searches in XML DB
Oracle Text Index

• Many content repositories need full text searches within XML documents
• XML DB leverages Oracle Text Index
• Using SQL/XML with XQuery for XML search and CONTAINS for text search
• Create XMLIndex on XMLType with optional paths (structured or unstructured)
• Create Text Index on XMLType with desired settings
• Optimizer uses appropriate combination of indexes
Using SQL/XML for Text Searches

- Example query:
  ```sql
  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
XML Storage Guidelines
XML Use cases

Structured

“Data Centric”
Static XML Schema
Limited Variability
No “any” or “mixed”

Semi Structured

Complex XML Schema Collections
Volatile XML Schemas
Islands of “any”
Or
Islands of Structure

Unstructured

“Document Centric”
No XML Schema
Very flexible XML Schema
Repeating Choice, “any” and “mixed”
XML Storage

- Oracle XML DB provides 2 main storage options
- Structured (Object Relational) Storage
  - O-R mapping derived from XML schema
  - Relational performance for structured use cases
  - In Oracle XML DB since 9iR2
- Binary XML Storage
  - Post-parsed binary representation stored in BLOB
  - Handles wide variety of use cases
  - In Oracle XML DB since 11gR1
XML Storage considerations

- Data Characteristics & Access paradigm can guide choices of storages

- Structured (object relational) storage
  - Data centric XML having schema modeling E/R model (well defined hierarchy with relational like schema evolution)
  - XQuery extracting, updating leaf level value
  - Many XMLTable relational views to provide SQL Access

- Binary storage
  - Data has No XML schema or has XML Schema with high flexibility (repeating choices, mixed content, any content)
  - XQuery retrieve or update large document fragments
  - Query with more wildcard, descendant ad-hoc XPaths
Object Relational Storage Practices Guidelines

- Use PL/SQL utility package from OTN to annotate XML Schema properly before schema registration
  - Use nested table for collection element by default
  - Use out-of-line storage for recursive element
  - For large dictionary like XML schema, use annotation to avoid unnecessary table creations
- To load large XML document into XMLType table, use FTP
- Build proper B+ tree or bitmap index and analyze explain plan
- Avoid XPath that can not be rewritten to directly query the underlying relational storage tables
When should you use Binary?

- **When application requires Schema flexibility**
  - If the XML Schema is not known a-priori.
  - Document centric use-cases with flexible structures.

- **Schema-less use cases**
  - Use Binary XML instead of CLOB storage
  - Avoids repeated XML parsing
  - Good performance even without indices
  - Data-type aware storage removes expensive conversions.
  - Compression of XML documents to avoid text bloat

- **Simplicity, Ease of management**
  - Single BLOB
  - XML Schemas need not be annotated
Binary XML Storage Practices Guidelines

• Use of XML Schema
  • XML Schema is optional for binary XML storage
  • But using XML Schema provides better compaction, improved query performance, data typing for queries
  • The main downside is that schema evolutions are limited to backward compatible changes

• Use SecureFiles
  • Can leverage capabilities like compression, dedup, encryption
  • Piece-wise updates of XML content

• Loading & Retrieving Binary XML in Mid-tier
  • Applications can directly manipulate XML in binary format using XDK, thus offloading DB CPU cycles
Binary XML Storage Practices Guidelines

• Follow the XMLIndex guidelines to build proper structured/unstructured XMLIndex, analyze execution plan
• Use explicit datatyping in XQuery when querying mix of schema based and schema-less XML documents in the same column
XML DB Repository Guidelines
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
Querying Documents in Repository

- **SQL/XML method**
  - Join RESOURCE_VIEW and DOC_TAB
  - Folder/path restriction on RESOURCE_VIEW
  - XMLExists or XMLQuery on DOC_TAB

- **Utilizing XML and Text Indexes on DOC_TAB.**

  ```sql
  SELECT XMLQuery('let $val := $DOC/PurchaseOrder/LineItems/LineItem[@ItemNumber = 19]
  return $val' PASSING OBJECT_VALUE AS "DOC" RETURNING CONTENT)
  FROM RESOURCE_VIEW rv, purchaseorder x
  WHERE ref(x) = 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/OE/PurchaseOrders/2002/Sep/VJONES-20021009123337583PDT.xml')
  = 1;
  ```
Common XML DB
Use cases
Use-cases: Structured XML Persistence

• Requirements
  • XML needs to be persisted in the database
    • Conforms to a highly structured XML schema
  • Queries tend to extract relational values from XML
  • Need to interact with other relational systems

• Configuration
  • Object Relational table for base storage
  • B-Tree and Bitmap indexes just like relational systems
  • Provides custom E-R performance comparable to any relational system, while preserving XML abstraction
Use-cases: Semi-structured XML Persistence

• Requirements
  • XML needs to be persisted in the database
    • May or may not have an XML schema
  • Queries tend to extract relational values as well as fragments from XML
  • Need to interact with BI systems

• Configuration
  • Binary XML table for base storage
  • XML Views based on XMLTable allows extraction of scalar values
  • XMLIndex (structured and/or unstructured) for indexing precise structure
  • Provides data/schema flexibility while preserving custom E-R performance on structured portions
Use-cases: XML Document Mgmt

• **Requirements**
  - XML documents need to be stored in a database repository
  - Queries involve combination of XML searches and keyword searches with language aware features
  - Need to interact with Web-based systems

• **Configuration**
  - Binary XML table for base storage
  - XMLIndex for XML searches
  - Text Index on document table
    - Can perform basic structure aware keyword searches
  - XML DB repository for file/folder organization