Ready, Set, XML!
Using Oracle XML Data

John Jay King
King Training Resources
john@kingtraining.com

Download this paper and code examples from:
http://www.kingtraining.com
Session Objectives

• Understand how Oracle supports XML data
• Know how to use the XMLtype for both structured and unstructured XML
• Use XML-related functions to retrieve relational data in XML form
• Create XML views of relational data
• Understand how Oracle uses registered XML schemas
• Know how WebDAV may be used with Oracle data
Oracle XML DB

- Oracle's XML support is provided as XML DB:
  - W3C (Worldwide Web Consortium) XML compliance
  - XMLType is an Oracle-defined datatype storing XML data
    - Unstructured (CLOB underneath)
    - Structured ("Shredded" into relational columns and rows)
    - Binary XMLType (new with Oracle 11g)
  - The XML parser is part of the database
  - Oracle provides several XML-oriented SQL functions to support XML, some support the emerging ISO/ANSI SQLX initiative
  - Check the reference manual for complete information: "XML DB Developer’s Guide"
XML DB Features, page 1

- **XMLType**: XML document stored as CLOB (Character Large Object), or "shredded" and stored as structured XML.
- **DOM Fidelity**: Structured XML is stored without disrupting DOM (Document Object Model) hierarchy.
- **XML Schema**: Both “structured” and “unstructured” documents may use XML Schemas to constrain XML input.
- **XML Piecewise**: Allows use of XPath syntax to update specific elements and attributes without rewriting document.
- **XPath Search**: Allows use of XPath syntax to focus queries.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XML Indexes</td>
<td>Support quicker XPath searches</td>
</tr>
<tr>
<td>SQLX Operators</td>
<td>New ANSI/ISO SQLX functions</td>
</tr>
<tr>
<td>XQuery</td>
<td>XML Query</td>
</tr>
<tr>
<td>XSL Transformations</td>
<td>Use XSLT to transform XML documents via SQL</td>
</tr>
<tr>
<td>XML Views</td>
<td>Views of XML documents, document fragments, and relational data</td>
</tr>
<tr>
<td>Java Bean Interface</td>
<td>API providing access to structured XML data via Java Beans</td>
</tr>
<tr>
<td>XML Repository</td>
<td>XML content stored in a directory-like hierarchy</td>
</tr>
</tbody>
</table>
XMLType Datatype

- XMLType may be used to represent a document or document fragment in SQL
- XMLType has several built-in member functions to operate on XML content
- XMLType may be used in PL/SQL as variables, return values, and parameters
- XMLType APIs are provided for both PL/SQL and Java programming
- Beginning with Oracle9i Release 2 XMLType is also supported on the client via FTP, HTTP, and WebDav
XMLType Functions

- **XMLType member functions include:**
  - `createXML()` Create XMLType instance
  - `existsNode()` Checks if XPath can find valid nodes
  - `extract()` Uses XPath to return XML fragment
  - `isFragment()` Checks if document is a fragment
  - `getClobVal()` Gets document as a CLOB
  - `getStringVal()` Gets value as a string
  - `getNumberVal()` Gets numeric value as a number
  - `isSchemaBased` Returns 1 if schema based (0 if not)
  - `isSchemaValid` True if XMLType is valid
  - `schemaValidate` Validates XMLType using Schema
  - `Transform` Apply XSL Stylesheet to XMLType
  - `XMLType` Constructs an XMLType instance from CLOB, VARCHAR2 or object
Oracle URLs/URIs

- **URL** (Uniform Resource Locator), address of a complete document or specific location within a document
- **URI** (Uniform Resource Identifier), URLs with additional information
- Oracle supports three datatypes for URIs
  - **HttpUriType** URL beginning with “http://”
  - **DBUriType** URI points to a column, row, or set of rows in the database – object methods retrieve data linked using DBUriType
  - **XDBUriType** URI points to an XML document known to the Oracle XML DB Repository – object methods retrieve all or part of XML documents (resources) represented by XDBUriType
ISO-ANSI SQL/XML (SQLX)

- SQL/XML is an ISO-ANSI working draft for XML-Related Specifications (aka. SQLX)
- SQLX defines how SQL may be used with XML
- SQLX functions are used to generate XML from existing relational (and object relational) tables
- SQLX standard functions supported by Oracle:
  - XMLAgg()  - XMLExists()
  - XMLAttribute()  - XMLForest()
  - XMLCast()  - XMLParse()
  - XMLComment()  - XMLPI()
  - XMLConcat()  - XMLQuery()
  - XMLElement()  - XMLSerialize()
SQL/XML Functions, 1

- **XMLAgg(xmlTypeObject orderbyclause)**
  - Creates an aggregate forest of XML elements from a collection of XML elements
  - xmlTypeObject represents the set of xmlType data to be aggregated
  - If specified, ORDER BY clause dictates sequence of aggregation (Note: no comma)

- **XMLCast(expression as dataType)**
  - Converts SQL expression’s return as specified datatype

- **XMLComment(expression)**
  - Creates XML comment using specified expression

- **XMLConcat(xmlType,xmlType,…xmlType)**
  - Concatenates a series of XMLType objects (opposite of XMLElement)

- **XMLElement(elementName,elementValue)**
  - Creates series of XML fragments using specified element name and values as attributes sometimes using the XMLAttributes clause

- **XMLExists(xmlQuery)**
  - Returns TRUE if xmlQuery returns non-null result
SQL/XML Functions, 2

- **XMLForest(expression)**
  - Creates a set of XML elements (forest) using fragments from a list of arguments/parameters

- **XMLParse(doctype,expression)**
  - Parses and generates document using expression

- **XMLPI(piName,piValue)**
  - Inserts a Processing Instruction (PI) as directed

- **XMLQuery(xQueryExpression)**
  - Returns result of XQuery specified

- **XMLSerialize(expression)**
  - Returns string or LOB containing results of expression
Oracle SQL/XML Extensions

- **XMLCdata**
  - Generate cdata section from specified expression
- **XMLColAttVal**
  - Oracle SQLX extension creates series of XML fragments using an element name of "column" and column names and values as attributes
- **XMLDiff**
  - Compare two XML documents and return difference(s) as a document
- **XMLPATCH**
  - Patches XMLType using second XMLType
- **XMLRoot**
  - Generate XML identification line (PI)
- **XMLSequence**
  - Creates Varray of XMLType instances
- **SYS_XMLGEN**
  - Convert specified database row and column into an XML document
- **SYS_XMLAGG**
  - Generate single XML document from aggregate of XML data specified by "exp"
Other XML Functions

- APPENDCHILDXML
- DELETEXML
- DEPTH
- EXTRACT (XML)
- EXISTSNODE
- EXTRACTVALUE
- INSERTCHILDXML
- INSERTXMLBEFORE
- PATH
- SYS_DBURIGEN
- SYS_XMLAGG
- SYS_XMLGEN
- UPDATEXML
- XMLTransform
SYS_XMLGEN

- Uses a single input expression representing a particular row/column (scalar value or user-defined type)
  - A single XML element representing scalar values is returned
  - XML elements representing each of a user-defined type’s data items is returned
  - Returns an instance of SYS.XMLType data that is an XML document
- The example below uses getStringVal since XMLType data returns as CLOB

```sql
select sys_xmlgen(ename).getStringVal() Name
  from emp
  where job = 'ANALYST'

NAME
----------------------------------------
<?xml version="1.0"?>
  <ENAME>FORD</ENAME>
<?xml version="1.0"?>
  <ENAME>SCOTT</ENAME>
```
SYS_XMLAGG

- SYS_XMLAGG aggregates all XML documents (or fragments) in an expression to produce a single document
  - ROWSET is the default tag name used
  - SYS.XMLGenFormatType may be used to change a tag name

- The example below uses the SYS_XMLGEN function to generate an XML document
  (example uses getClobVal to add XML PI)

```sql
select sys_xmlagg(SYS_XMLGEN(Ename)).getClobVal() emps
from emp
where deptno = 10

EMPS
------------------------------------------------------
<ROWSET>
<ENAME>KING</ENAME>
<ENAME>CLARK</ENAME>
<ENAME>MILLER</ENAME>
</ROWSET>
```
• XMLElement is used to define Elements

  XMLElement("MyElementName", valueExp)

  – MyElementName may be any valid XML name
  – valueExp may be a literal, column name, or expression providing the value for the element (May be nested)

• XMLAttributes is used to define Element Attributes; it should be used inside XMLElement and precede any SubElements for the chosen Element

  XMLAttributes("MyAttributeName", valueExp)

  – MyAttributeName may be any valid XML name
  – valueExp may be a literal, column name, or expression providing the value for the element
Most-Used XML Functions, 2

- XMLForest works like nested XMLElements

  `XMLForest(valExp1, valExp2 AS "MyElement2")`

  - `valExp1` may be a literal, column name, or expression providing the value for the element
  - `valExp2` may be a literal, column name, or expression providing the value for the element
  - `MyElement2` may be any valid XML name

- XMLAgg aggregates calls to XMLElement, XMLAttribute, and XMLForest (and others) to create an XML document

- Column name used if Element and/or Attribute not explicitly named
**XMLElement**

- `XMLElement(name,exp)` Generates an XML element using `name` and `exp` as data:

```sql
select xmlelement("employee",
    xmlelement("empid",empno),
    xmlelement("empname",ename)) myxml
from emp

<employee> <empid>7369</empid>
    <empname>SMITH</empname> </employee>
<employee> <empid>7499</empid>
    <empname>ALLEN</empname> </employee>
<employee> <empid>7521</empid>
    <empname>WARD</empname> </employee>
<employee> <empid>7566</empid>
    <empname>JONES</empname> </employee>
<employee> <empid>7654</empid>
    <empname>MARTIN</empname> </employee>
```
XMLAttributes

- Generates XML attributes using an expression list:

```sql
select xmlelement("employee",
    xmlelement("emp", xmlattributes(empno as "empno",
    ename as "ename")),
    xmlelement("job",job),
    xmlelement("hiredate",hiredate),
    xmlelement("pay", xmlattributes(nvl(sal,0) "sal",
    nvl(comm,0) as "comm"))
) as myxml
from emp;

<employee>
    <emp empno="7782" ename="CLARK"/>
    <job>MANAGER</job>
    <hiredate>09-JUN-81</hiredate>
    <pay sal="2450" comm="0"/>
</employee>
*** More like the above ***
```
XMLForest

• XMLForest is similar to using XMLElement several times

```sql
select xmlelement("employee",
    xmlelement("emp", xmlattributes(empno as "empno",
                                   ename as "ename")),
    xmlforest(job, hiredate, sal, deptno)
  ) as myxml
from emp;

<employee>
  <emp empno="7839" ename="KING"/>
  <JOB>PRESIDENT</JOB>
  <HIREDATE>17-NOV-81</HIREDATE>
  <SAL>5000</SAL>
  <DEPTNO>10</DEPTNO>
</employee>

*** More like the above ***
• XMLConcat concatenates XML data without adding higher-level elements

```sql
SELECT XMLConcat( XMLElement("empno",empno),
                  XMLElement("empname",ename),
                  XMLElement("hiredate",hiredate),
                  XMLElement("salary",to_char(sal,'99,999.99'))) FROM emp;
```

```xml
<empno>7839</empno>
<empname>KING</empname>
<hiredate>17-NOV-81</hiredate>
<salary>5,000.00</salary>
*** More like the above ***
XMLAgg

- XMLAgg is used when a GROUP BY is aggregating the SQL data

```sql
SELECT XMLELEMENT("Department",
    XMLAGG(XMLForest(deptno, empno, ename, sal)))
FROM emp
GROUP BY deptno;

<Department>
    <DEPTNO>20</DEPTNO>
    <EMPNO>7566</EMPNO>
    <ENAME>JONES</ENAME>
    <SAL>2975</SAL>
    *** more dept 20 rows ***
    <DEPTNO>20</DEPTNO>
    <EMPNO>7902</EMPNO>
    <ENAME>FORD</ENAME>
    <SAL>3000</SAL>
</Department>
*** More like the above ***
select xml element("employee",
    xmlagg(xmlelement("emp",xmlattributes(empno as "empno",
        ename as "ename"),
        xmlelement("job",job),
        xmlelement("hiredate",hiredate),
        xmlelement("pay",
            xmlattributes(nvl(sal,0) as "sal",
                nvl(comm,0) as "comm")))))
    from emp;
<employee>
    <emp empno="7839" ename="O'BRIAN">
        <job>PRESIDENT</job>
        <hiredate>17-NOV-81</hiredate>
        <pay sal="5000" comm="0"/>
    </emp>
    <emp empno="7698" ename="BLAKE">
        <job>MANAGER</job>
    *** More like above ***
</employee>
DBMS_XMLGEN

- DBMS_XMLGEN package creates XML documents from an SQL query
- DBMS_XMLGEN function setMaxRows may be used to control the maximum number of rows fetched
- DBMS_XMLGEN function setSkippedRows may be used to control the number of rows to be fetched
- Additional functions/procedures are provided to allow changing the default root element name from ROWSET to a chosen value and to change or omit the ROW identifier
- C language program
- First available with Oracle8i (8.1.6 and later)
dbms_xmlgen Functions/Procedures

- `newContext()` Creates new context handle
- `setRowTag()` Set root element name (default=ROWSET)
- `setRowSetTag` Set row element name (default=ROW)
- `getXML()` Use query to get XML document
- `getNumRowsProcessed()` Gets the number of SQL rows in last getXML call
- `setMaxRows()` Set max. rows to be fetched each time
- `setSkipRows()` Set number of rows to skip before generating XML (default = 0)
- `setConvertSpecialChars()` “Escape” special characters
- `convert()` Converts XML as “escaped”/”unescaped”
- `useItemTagsForColl()` Defines a collection column name appended by _ITEM
- `restartQUERY()` Restarts query at the beginning
- `closeContext()` Closes context and releases all resources
DBMS_XMLGEN, 1

- DBMS_XMLGEN may be used to process a query as follows

```sql
CREATE OR REPLACE FUNCTION getAllEmps
RETURN clob
IS
    mySql VARCHAR2(500);
BEGIN
    mySql := 'select ename,job,empno,sal,deptno ' || ' from emp order by ename,empno ';
    RETURN dbms_xmlgen.getxml(mySql);
END;
select getAllEmps from dual;
<?xml version="1.0"?>
<ROWSET>
    <ROW>
        <ENAME>ADAMS</ENAME>
        <JOB>CLERK</JOB>
        <EMPNO>7876</EMPNO>
        <SAL>1100</SAL>
        <DEPTNO>20</DEPTNO>
    </ROW>
    ...
</ROWSET>
```
CREATE OR REPLACE FUNCTION getEmpData (inEmpno in varchar2)
RETURN clob IS
    mySql VARCHAR2(500);
BEGIN
    mySql := 'select ename,job,empno,sal,deptno ' || 'order by ename,empno ';
    RETURN dbms_xmlgen.getxml(mySql);
END;

select getEmpData(7788) from dual;

<?xml version="1.0"?>
<ROWSET>
    <ROW>
        <ENAME>SCOTT</ENAME>
        <JOB>ANALYST</JOB>
        <EMPNO>7788</EMPNO>
        <SAL>3000</SAL>
        <DEPTNO>20</DEPTNO>
    </ROW>
</ROWSET>
CREATE OR REPLACE FUNCTION getEmpData2
RETURN clob
IS
  mySql dbms_xmlgen.ctxHandle;
BEGIN
  mySql := dbms_xmlgen.newContext('select ename,job,empno,sal,deptno ' || ' from emp ' || ' order by ename,empno ');
  dbms_xmlgen.setRowsetTag(mySql,'personnelData');
  dbms_xmlgen.setRowTag(mySql,'employee');

  return dbms_xmlgen.getxml(mySql);
END;

select getEmpData2 from dual;

<?xml version="1.0"?>
<personnelData>
  <employee>
    <ENAME>ADAMS</ENAME>
    <JOB>CLERK</JOB>
    <EMPNO>7876</EMPNO>
    <SAL>1100</SAL>
    <DEPTNO>20</DEPTNO>
  </employee>
</personnelData>
DBMS_XMLQUERY

- Java program to create XML from an SQL query
- Like DBMS_XMLGEN, but, not as fast (use DBMS_XMLQUERY)
DBMS_XMLQUERY Procedures/Functions

- closeContext() Closes query context
- getDTD() Generate DTD
- getNumRowsProcessed() Gets number of rows processed
- getXML() Generate the XML document
- newContext() Creates a query context
- removeXSLTParam() Removes a stylesheet parameter
- setBindValue() Sets value for bind name
- setDateFormat() Sets format of generated dates in the XML
- setEncodingTag() Set XML encoding instructions
- setMaxRows() Sets maximum number of rows to be converted to XML
- setRowIdAttrName() Sets name of id attribute
- setRowIdAttrValue() Specifies column to provide row id attribute
- setRowTag() Sets element name for each row
- setRowsetTag() Sets root element tag name
- setSkipRows() Sets the number of rows to skip
- setStylesheetHeader() Sets stylesheet header
- setTagCase() Specify case of generated XML tags
- setXSLT() Registers stylesheet to be applied to XML
- setXSLTParam() Sets value of stylesheet parameter
- useNullAttributeIndicator() Whether or not to indicate null
CREATE OR REPLACE FUNCTION getAllEmpsXQ
RETURN clob
IS
    mySql VARCHAR2(500);
    mySqlCtx DBMS_XMLQuery.ctxType;
    returnXML CLOB;
BEGIN
    mySql := 'select ename, job, empno empid, sal, deptno ' || ' from emp order by ename, empno ';
    mySqlCtx := DBMS_XMLQuery.newContext(mySql);
    dbms_xmlquery.setRowIdAttrName(mySqlCtx,null);
    dbms_xmlquery.setRowsetTag(mySqlCtx,'personnelData');
    dbms_xmlquery.setRowTag(mySqlCtx,'employee');
    returnXML := DBMS_XMLQuery.getXML(mySqlCtx);
    DBMS_XMLQuery.closeContext(mySqlCtx);
    RETURN returnXML;
END;
select getAllEmpsXQ from dual;
xml version = '1.0'?
<personnelData>
  <employee>
    <ENAME>ADAMS</ENAME>
    <JOB>CLERK</JOB>
    <EMPID>7876</EMPID>
    <SAL>1100</SAL>
    <DEPTNO>20</DEPTNO>
  </employee>
  <employee>
    <ENAME>ALLEN</ENAME>
    <JOB>SALESMAN</JOB>
    <EMPID>7499</EMPID>
    <SAL>1600</SAL>
    <DEPTNO>30</DEPTNO>
  </employee>
  *** more ***
</personnelData>
XML Schema Support

• The W3C (World-Wide Web Consortium) XML Schema standard provides a middle ground between data modeling and document modeling
• XML schemas may be used to automatically create tables and types, or, to validate updates and inserts
• XML schemas may be used as the basis for XMLType tables and columns (but, schemas are not required to store XMLType data)
• XML schemas must be registered in the database where they are stored as CLOBs
• Once registered, XML schemas may be referenced using URL notation
• Registered XML schemas may be used to map XML documents to structured or unstructured database storage
More on XML Schema

- Beginning with Oracle9i Release 2 some of the XML schema related activity permitted includes:
  - XMLType objects may be built based upon XML schema
  - XMLType objects may be validated using XML schema
  - Tables may be created based upon XML schema automatically creating storage structures
    (does not require specific column definition)
  - XML schemas may be registered in the database using DBMS_XMLSCHEMA package
  - Registered schemas may be shared
  - Schema registration may create Java beans and default tables
  - Schemas allow pre-parsing of incoming XML documents and their direction to the appropriate tables
  - XML documents and instances may be validated using XMLType’s XMLIsSchemaValid() method
  - extractValue is used to extract part of the XML document
Schema Validation

- XML Schemas are powerful, but, do not provide some features we take for granted in the database such as UNIQUE key and FOREIGN key constraints.
- By default, Oracle9i does not completely validate documents as they are inserted into the database; restriction facets such as minLength, maxLength, and patterns are ignored (11g and 10g add more-complete validation).

Schema validation may be enabled for individual schemas via Check Constraints or Triggers.
DBMS_XMLSCHEMA

- Oracle9i added the DBMS_XMLSchema package, Oracle 10g enhanced it
- DBMS_XMLSchema provides procedures used to manager schemas in the XDB repository
  - COMPILESCHEMA Recompile an already registered schema
  - COPYEVOLVE Evolve registered schema(s)
  - DELETESCHEMA Remove schema from the database
  - GENERATEBEAN Generate the Java bean code corresponding to a registered schema
  - GENERATESCHEMEA Generate XML schema from an Oracle type
  - GENERATESCHEMAS Generate several XML schemas from an Oracle type
  - REGISTERSCHEMA Register schema in repository
  - REGISTERURI Register XMLSchema specified by URI
PROCEDURE registerSchema(
    schemaURL IN varchar2,
    schemaDoc IN SYS.URIType,
    local IN BOOLEAN := TRUE,
    genTypes IN BOOLEAN := TRUE,
    genBean IN BOOLEAN := FALSE,
    force IN BOOLEAN := FALSE,
    owner IN VARCHAR2 := null);

- schemaURL URL to be used within Oracle for schema
- schemaDoc Schema (varchar, bfile, blob, clob, XMLtype, URIType
- local true, register as /sys/schemas/<username>/*
  false, register as /sys/schemas/PUBLIC/* (default = true)
- genTypes true, generate object types (default = true)
- genBean true, generate Java beans (default = false)
- genTables true, generate default tables (default = true)
- force true, do not raise registration errors probably building
  invalid schema object (default = false)
- owner User who owns schema object
  (default = user registering schema)
- csid Character set of schema, if 0 current rule for "text/xml"
  (no default)
Registering Schema

- Schemas must be created and tested (use an appropriate XML editor), then, register them with DBMS_XMLSCHEMA.

```sql
begin
    dbms_xmlschema.registerschema('myBooks.xsd',
        '<?xml version="1.0" encoding="UTF-8"?>
        <xs:schema
            xmlns:xs="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified">
            <xs:element name="myBooks">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="book" maxOccurs="unbounded"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:schema>',true,true,false,false);
end;
/
```
10g Schema Annotations

- Oracle 10g adds add schema annotations providing information to the XDB schema compiler

```xml
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0"
            xdb:storeVarrayAsTable="true">
    <xs:element name="PurchaseOrder"
                type="PurchaseOrderType"
                xdb:defaultTable="PURCHASEORDER" />
    <xsd:complexType name="PurchaseOrderType"
                    xdb:SQLType="XDBPO_TYPE">
        - Namespace "http://xmlns.oracle.com/xdb" is required
        - Attributes include:
          • storeVarrayAsTable Direct database to store as a table
          • defaultTable Names table representing Schema's data
          • SQLType Defines SQL type (number, varray, etc...) of related database column
          • SQLName Name used for SQL attribute
          • SQLSchema Names registered schema
```
DBMS/XMLSCHEMA Dictionary

- ALL_XML_SCHEMAS: Schemas available to user
- ALL_XML_TABLES: XMLType tables available to user
- ALL_XML_TAB_COLS: XMLType table columns in tables available to user
- ALL_XML_VIEWS: XMLType views available to user
- ALL_XML_VIEW_COLS: XMLType view columns in views available to user
Schema with XMLType Tables

• To create a table using the schema type the following

```sql
create table myBooksType of xmltype
    xmlns schema "myBooks.xsd" element "myBooks";
```
Schema with XMLType Columns

• To create an XMLType column using the schema definition

```sql
create table myBooks
(id number,
 books xmltype)
xmltype column books
    xmlschema "myBooks.xsd" Element "myBooks"
```
Oracle 10g CopyEvolve

- Oracle 10g added the ability to evolve registered schemas so that existing XML data remains valid
- **Careful!** Backup all schemas and documents before executing COPYEVOLVE, it deletes all conforming documents
- COPYEVOLVE is very busy, it:
  - Copies XMLType tables to temporary storage tables
  - Drops old tables
  - Deletes old schemas
  - Registers new schemas
  - Creates new XMLType tables
  - Populates new tables with data from temporary tables (constraint, triggers, indexes are lost)
  - Drops temporary storage tables
• Oracle 11g adds the capability of In-Place schema evolution

• In-Place Schema Evolution **does not** require copying, deleting, and inserting existing data making it faster than copy-based evolution

• In-Place Schema Evolution is restricted to cases where:
  – Storage model is not changing
  – Existing documents are valid using the new schema

• Different PL/SQL procedures are used for each:
  – DBMS_XMLSCHEMA.copyEvolve (Copy-Based)
  – DBMS_XMLSCHEMA.inPlaceEvolve (In-Place)
Unstructured XML Data

- XMLType data stored without a schema is "unstructured" and stored as CLOB

```
CREATE TABLE my_books OF XMLTYPE;

CREATE TABLE my_books ( books XMLTYPE );
```
Unstructured Insert

- Data inserted into an unstructured XMLType table or column is stored in CLOB form

```sql
INSERT INTO my_books
VALUES
    (XMLTYPE ('
        <book>
            <name>Learning XML</name>
            <author>Eric T. Ray</author>
            <publisher>O’Reilly</publisher>
            <isbn>0-596-00046-4</isbn>
        </book>
    '));
```
• It is also possible to insert (both structured and unstructured) XML using an input file

```sql
insert into my_books
values
(
    XMLType
    (
        bfilename('MYXDB','myBooks.xml'),
        nls_charset_id('AL32UTF8')
    )
)
```
CREATE or REPLACE PROCEDURE insertPurchaseOrderXMLOrder IS
    PurchaseOrderXML CLOB; -- CLOB to hold XML
BEGIN
    PurchaseOrderXML :=
        '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
        <PurchaseOrder>
            <POID>1234</POID>
            <Date>2003-0401</Date>
            <CustomerID>AA1234</CustomerID>
            <Company>King Training Resources</Company>
        </PurchaseOrder>'';
    -- Insert the Purchase Order XML into an XMLType column
    INSERT INTO purchaseOrderTable (purchaseOrder)
        VALUES (XMLTYPE(PurchaseOrderXML));
EXCEPTION
    WHEN OTHERS THEN
        raise_application_error(-20101,
            'Error loading purchaseOrderTable, SQLCODE='||SQLERRM);
END insertPurchaseOrderXMLOrder;
Unstructured Update

- Unstructured Updates look like this

```sql
update my_books xstuff
    set books = xmltype(''
        <book>
            <name>Learning XML</name>
            <author>Eric T. Ray</author>
            <publisher>O Reilly</publisher>
            <isbn>0-596-00046-4</isbn>
        </book>'
    )
where
    xstuff.books.extract('/myBooks/book/text()').getStringVal() = 'Wiley'
```
Unstructured Delete

• Unstructured Delete may also use XPath notation to identify parts of the document to be deleted

```xml
delete from my_books xstuff
where
xstuff.books.extract('/myBooks/book/text()').getStringVal() = 'Wiley'
```
Structured XML Data

- XMLType data stored using a schema is “structured” and is taken apart (shredded) into relational objects

CREATE TABLE my_books OF XMLTYPE
   XMLSCHEMA 'myBooks.xsd'
   ELEMENT 'myBooks';

CREATE TABLE my_books (  
   info XMLTYPE, desc VARCHAR2(100))
   XMLTYPE COLUMN info STORE AS OBJECT RELATIONAL
   XMLSCHEMA 'myBooks.xsd'
   ELEMENT 'myBooks';

CREATE TABLE my_books (  
   info XMLTYPE, desc VARCHAR2(100)  )
   XMLTYPE COLUMN info STORE AS CLOB
   XMLSCHEMA 'myBooks.xsd'
   ELEMENT 'myBooks';
Insert using Schema

- When inserting data, XML document using schemas are validated

```sql
INSERT INTO myBooksType VALUES
(xmltype.createxml('"\n   <myBooks xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:noNamespaceSchemaLocation="myBooks.xsd">
      <book>
         <name>Definitive XML Schema</name>
         <author>Prescilla Walmsley</author>
         <publisher>Prentice-Hall</publisher>
         <isbn>0-13-0655667-8</isbn>
      </book>
      <book>
         <name>Definitive XSLT and XPATH</name>
         <author>G. Ken Holman</author>
         <publisher>Prentice-Hall</publisher>
         <isbn>0-13-065196-6</isbn>
      </book>
   </myBooks>
'));
```
Selecting Using XPATH

• XPath notation may be used to select sets of data or particular rows

```
select value(xml)
    from myBookstype xml
    where existsnode(value(xml),
        '/myBooks/book[publisher="OReilly"]') = 1
```

```
select extract(object_value,'/PurchaseOrder/Reference')
    aref
from oe.PurchaseOrder
where extract(object_value,'/PurchaseOrder/Requestor')
    like '%Walsh%'`
Update Using XPath

- XPath may be used to alter portions of the structured XML document (piece-wise update)
  - Oracle 9i provided UpdateXML()
  - Oracle 10G R2 added InsertXML(), AppendChildXML(), InsertXMLBefore(), DeleteXML()

```sql
update myBookstype xstuff
set value(xstuff) =
  updateXML(value(xstuff),
    '/myBooks/book/publisher/text()', 'Shannon')
where existsnode(value(xstuff),
    '/myBooks/book[publisher="Wiley"]') = 1
```
Delete Using XPath

- Delete may also use XPath notation to identify portions of the XML document to be deleted

```xml
delete myBookstype xstuff
    where existsnode(value(xstuff),
        '/myBooks/book[publisher="Wiley"]') = 1
```
Structured Storage Review

• XML document is “shredded” into database objects
• Documents must conform to a registered XMLSchema; XML DB will use the XML Schema to generate SQL
• Structured Storage has several advantages:
  – Memory management is better than with CLOB
  – Storage requirements are reduced
  – Indexing is easier with structured data
  – Partial or in-place updates are possible
• Adding and retrieving XML documents to the database is slower when using Structured Storage
Binary XML

- Oracle continues its XML leadership in Oracle 11g
- Biggest change is the addition of a new “binary” XMLType
  - “binary xml” is a third method for storing XML data in the database
  - “structured” and “unstructured” XMLType still supported
  - Oracle 11g’s XML processors includes a binary XML encoder, decoder, and token manager
  - XML 1.0 text may be parsed via SAX events with or without a corresponding schema into “binary” XML form
  - “binary” XMLType allows optimization of some XML applications by reducing memory and CPU expense
11g Binary XMLType Pros

• XMLType data is stored in a binary format (known as post-parse)

• Binary XML is:
  – Smaller
  – Already-parsed
  – XML schema-aware
    (but does not require XML schema)
BINARY XML Storage

- When creating table/column of XMLType the type of storage may be specified

```
CREATE TABLE MYSTUFF OF XMLTYPE
XMLTYPE STORE AS BINARY XML
/
```
## XML Types Compared

<table>
<thead>
<tr>
<th></th>
<th>Structured</th>
<th>Unstructured</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Throughput</strong></td>
<td>Slower</td>
<td>Fast</td>
<td>Fast</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>Excellent</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td><strong>Flexible Data</strong></td>
<td>Match schema</td>
<td>Any</td>
<td>Any</td>
</tr>
<tr>
<td><strong>Mult. Schemas</strong></td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>Piecewise</td>
<td>Entire doc.</td>
<td>Piecewise (SecureFile only)</td>
</tr>
<tr>
<td><strong>XPath Query</strong></td>
<td>Excellent</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td><strong>Insert Validation</strong></td>
<td>Partial</td>
<td>Partial (if schema-based)</td>
<td>Full</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>B-tree, Function-based, Oracle-text</td>
<td>XMLIndex, function-based, Oracle-text</td>
<td>XMLIndex, function-based, Oracle-text</td>
</tr>
</tbody>
</table>
XML DB Components

• XMLType  Data type defining the column or table as XML data and including methods to allow operations on the XML such as XSL transformations and validation via XML Schema

• XMLSchema  Complete XML Schemas may be registered with XML DB to validate documents and to define how documents are stored by Oracle

• XML DB Repository  Provides mechanism for associated URIs with XPath notation to access XML data; supports interaction with HTTP, FTP, WebDAV clients

• SQL/XML  XML DB includes many operators that are part of the emerging SQL/XML standard
Viewing Folders

- The server’s Http server may be used to access the directory structure
What is WebDAV?

- WebDAV is an IETF (www.ietf.org) standard set of HTTP extensions allowing an HTTP Server to serve files to a WebDAV-enabled client.
- Any WebDAV-enabled product can read and update XML content stored in the XML DB Repository.
- Since both Microsoft Office (Office XP and beyond) and Oracle support WebDAV, they work together automatically.
- Some other WebDAV-enabled products: Microsoft Internet Explorer, Altova XMLSpy, Macromedia MX and others.
- XML’s promise of portable data is greatly facilitated by WebDAV.
Running Oracle’s XDB Demo

- Go to http://technet.oracle.com

If you don’t already belong:
  - No cost (other than an email address and occasional spam)
  - Most trial software available for download
  - Many white papers and demos

- Look under “XML” for the “XDBBasicDemo”
  - Download it
  - Download other software: XML editor, FTP package (not required, but makes installation of demo easier)
  - Try WebDAV by following the well-laid-out instructions
Indexing XML Content

• XML data may be indexed to increase efficiency like other Oracle tables

• Unstructured XML data (CLOB storage)
  – XML & text aware indexing and searching with Oracle Text

• Structured XML data (Object-Relational storage)
  – Automatic query rewrite enables all existing indexes types

```sql
create index book_author on myBooks
  (books.extract('/myBooks/book/author.text()').getStringVal())
```
11g XMLIndex

• Oracle 11g introduces a new index type for XMLType called XMLIndex
• XMLIndex can improve performance of XPath-based predicates and fragment extraction
• XMLIndex is a (logical) domain index consisting of underlying physical table(s) and secondary indexes (replaces CTXSYS.CTXXPath; Oracle recommends replacing any CTXXPath indexes with XMLIndex)
• Supported by PL/SQL DBMS/XMLINDEX package
• XML Publisher is a powerful report generation tool allowing creation of standardized reports containing Oracle (and other) data quickly and easily (replacement for SQL*Reports?)

• XML Publisher supports:
  – Creating Reports and Report Standardization
  – Report, Data, and Translation Templates
  – Support for regular files, XML files, and database data
  – Online Report Editor, Query Builder, and Analyzer
  – Simple Charts & Formatting Options
  – Row, Column, and Cell formatting
  – Number and Date formatting (e.g. internationalization)
  – Advanced reporting via XSL, SQL, and XSL-FO (including PDF)

• Originally intended to support Oracle' various ERP tools XML Publisher is now available as a separate product
XQuery

- XQuery is an XML query and processing language represented in manner similar to XML
- Some find XQuery programs easier to understand and maintain than XSLT
- XQuery language is small and powerful
- XQuery language is a W3C (World Wide Web Consortium) project in working draft form
- XQuery is part of ISO/ANSI SQL 2005
- XQuery may well become the normal way of accessing SQL data in the future
- XQuery supports XML Schema
XQuery Expressions - FLWOR

• Everything in XQuery is an “expression”
• Every expression returns a new value
• FLWOR expressions include:
  – For Range of values (like SQL FROM)
  – Let Define/set variables used by For
  – Where Filter conditions (like SQL WHERE)
  – Order-by Ordering (like SQL ORDER BY)
  – Return Output definition (like SQL SELECT)

• SQL*Plus now supports direct XQuery use
XMLQuery & XMLTable

• The Oracle (ISO) XMLQuery function
  – Uses XQuery to one read or more XML documents
  – Return is an XML document
  – XMLQuery is usually used in a Select list

• Oracle (ISO) XMLTable Function
  – XMLTable() is used in the From clause of an SQL Expression
Example XMLQuery

```
SELECT XMLQuery('for $i in ora:view("OE","WAREHOUSES")/ROW
    return <Warehouse id="{$i/WAREHOUSE_ID}"/>
    <Location>
        {for $j in ora:view("HR", "LOCATIONS")/ROW
            where $j/LOCATION_ID eq $i/LOCATION_ID
            return ($j/STREET_ADDRESS,
                    $j/CITY, $j/STATE_PROVINCE)}
    </Location>
</Warehouse>'
RETURNING CONTENT) FROM DUAL;
```
Example XMLTable

```sql
SELECT mystuff.COLUMN_VALUE AS OrderTotal
FROM oe.purchaseorder,
    XMLTable(
        'for $i in /PurchaseOrder
        where $i/User = "AWALSH"
        return
        <OrderTotal>
            {$i/Reference}
        <Total>
            {fn:sum(for $j in $i/LineItems/LineItem/Part
                        return ($j/@Quantity*$j/@UnitPrice))}
        </Total>
    </OrderTotal>'
PASSING OBJECT_VALUE
) mystuff;
```
SQL*Plus XQuery

- SQLPlus now supports direct XQuery

```xquery
XQuery(
    'for $i in ora:view("OE", "WAREHOUSES")/ROW
       return
       <Warehouse id="{$i/WAREHOUSE_ID}">
          <Location>
            {for $j in ora:view("HR", "LOCATIONS")/ROW
               where $j/LOCATION_ID eq $i/LOCATION_ID
               return ($j/STREET_ADDRESS, $j/CITY,
                        $j/STATE_PROVINCE)}
          </Location>
       </Warehouse>'
/}
```
10g R1 XML Enhancements

- Export/Import support
- Schema evolution via DBMS_XMLSCHEMA.CopyEvolve
- C and C++ APIs allow XML modification
- DBMS_XMLGEN allows turning off "pretty print"
- Hierarchical queries (CONNECT) via DBMS_XMLGEN.newContextFromHierarchy
- DBMS_AQ support for XMLType data
- SQL*Loader support for both shredded and unshredded data
- Globalization support
10g R2 XML Enhancements

- XMLType in Java, C, C++ (Ent. XML Developers Kit)
- XSLT 2.0 XPath support
- XQuery XMLQUERY and XMLTABLE functions
- InsertXML(), AppendChildXML(), InsertXMLBefore(), DeleteXML() SQL functions added to UpdateXML()
- SOAP 1.2 support in C/C++
- SQL/XML 2003 standard support
- DBMS_XMLDOM, DBMS_XMLPARSER and DBMS_XSLPROCESSOR replace deprecated XMLDOM, XMLPARSER and XSL_PROCESSOR packages
- XML DB HTTPS support
11g XML Enhancements, 1

- New Binary XMLType storage model
- XMLIndex Index type to improve Xpath location
- XMLTypeOCT (XMLType Ordered Collection Tables)
- Supports both WebDAV and XML DB privileges
- Web Services Oracle queries with SQL or XQuery
- In-Place XML Schema Evolution
- Recursive XML Schemas
- XLink and XInclude support
- XQuery 1.0 Compliance
- SQL/XML Compliance
11g XML Enhancements, 2

- Large XML Nodes in PL/SQL, Java, and C
- Java XML APIs shared for XML DB and Oracle XDK
- XMLType support by Oracle Streams
- XMLType supported by Oracle Data Pump
- Oracle XDK Pull-Parser API
- XML-Update Performance improvements
- XQuery and SQL/XML Performance improvements
- XSLT Performance Enhancements
Wrapping it all Up

- XML DB provides several valuable tools for working with XML data
- XML may be produced from existing relational tables using XML-oriented functions and PL/SQL packages
- XML may be stored in the database in three ways:
  - Unstructured (entire XML document in one CLOB)
  - Structured (XML document uses schema to store data in object-relational database objects)
  - Binary (may use schema, but not required)
- XML data may be retrieved, inserted, updated and deleted like other database data
- WebDAV provides a direct interface to several other products
Oracle Development Tools User Group

A Real World User Group
For Real World Developers

WWW.ODTUG.COM
Evolution of the Developer: Middleware & Beyond

ODTUG Returns to . . . THE BIG EASY!
June 15-19, 2008
Sheraton New Orleans

Abstracts Due December 10th

• Fusion Middleware
• Java EE And SOA
• Oracle Tools
• Third Party Tools
• Business Intelligence/Hyperion
• Development DBA
• Methodology
• Professional Development
Training Days 2008
February 13-14 2008!

2 Days – Denver Colorado
$240 for RMOUG, IOUG, ODTUG
(or other Oracle user-group members)
$300 for non-user-group attendees

http://www.rmouug.org
(make it a Valentines weekend in the Rockies!)
Ready, Set, XML!
Using Oracle XML Data
Session: S290740

To contact the author:
John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727

Email: john@kingtraining.com

Today's slides and examples are on the web:
http://www.kingtraining.com

Thanks for your attention!