
ODTUG 2007
KALEIDOSCOPE
WOW! Wide Open World,
Wide Open Web!
JUNE 18 - 21, 2007
Preconference Hands-On Training JUNE 16 - 17
Hilton Daytona Beach Oceanfront Resort
Daytona Beach, Florida
FEATURING
Oracle Fusion Symposium all day June 18
"Seriously Practical" Application Express Training June 18 and 19

XML Data Into and Out of Oracle – Using PL/SQL

Ken Atkins
Creative Design Associates

Introductions

- **Who am I?**
  - Creative Design Associates
- **Who are you?**
  - How many are Developers? DBAs?
  - Experience in PL/SQL? Java?
  - Experience with XML?
  - Who has worked with XML in PL/SQL?

XML tasks to be done in PL/SQL

Part I: (Previous Presentation):

1. Introduction to Oracle PL/SQL Tools for XML
2. Producing XML documents from data in normalized Oracle tables
3. Storing data from XML documents into normalized Oracle tables

Part II: (This Presentation):

4. (Continuted) Storing data from XML into Oracle Tables
5. Storing XML documents in the database as XML
   - (CLOBs, XMLType, XMLDB tables)
6. Querying data within XML stored in database
7. Validating XML in the database

Shredding XML Data into Tables

(Review from Part I)

- **"Shredding" is the process of parsing XML and separating it into different elements**
- **PL/SQL Methods for "Shredding" XML:**
  - Xpath Access via XMLTypes – *Covered in Part I*
  - DBMS_XMLSTORE (inverse of DBMS_XMLGEN) – *Covered in Part I*
  - DBMS_XMLPARSER
  - DOM Model API
- **Again, this part is not about storing XML *as XML* in the database**

Accessing XML Data in PL/SQL

(Review from Part I)

- **Two Steps:**
  1. Parsing: Converting XML from files or CLOB into accessible format
  2. Accessing: Access portions of the XML from the parsed representation
- **Parsing – Two options:**
  - XMLType constructor – *Covered in Part I*
  - DBMS_XMLPARSE – Into DOM model
- **Accessing – Two options:**
  - XPATH References – *Covered in Part I*
  - DOM API

Overview of the XML DOM Model

(Review from Part I)

- **Document Object Model (DOM) view of XML**
- **The entire thing is called a "Document"**
- **Access to an XML document as a "Tree"**
- **Each "level" of the tree is a "Node"**
- **The "leaf" nodes of the tree are the actual text values.**
  - Also identified as the "elements"
- **The DOM tree can be accessed using "tree-walking" methods.**

Overview of the XML DOM Model

(Review from Part I)

- **The data is accessed via a hierarchy of Oracle Types:**
  - Document: DBMS_XMLDOM.DomDocument
  - Node: DBMS_XMLDOM.DomNode
  - Element: DBMS_XMLDOM.DomElement
  - Text: DBMS_XMLDOM.DomText
- **Can be used to Create XML from scratch**
  - But XML/SQL or DBMS_XMLGEN is usually easier
- **Usually used to access existing XML**
- **Can be used to merge/split XML fragments**

Understanding DOM Nodes

(Review from Part I)

- **To work with the DOM model, you need to understand some things about "Nodes"**
  1. There is a main Node that is for the entire document
     - This is not the node for the top tag, it is *above* that
  2. There is a node for each level of the XML hierarchy
  3. There is a special "text" node for each tag value
     - This is in addition to the node for the tag itself!
  4. The nodes are connected in a hierarchy
     - Child nodes are "appended" onto parent nodes

Overview of the XML DOM Model

(Review from Part I)

Root Node

1st Level Node

2nd Level (etc.)

Leaf Node (elements)

Text Node

```
<orderMessage>
  <order>
    <ORDER_NUMBER>Z1234-57A</ORDER_NUMBER>
    <ORDER_DATE>2007-01-15</ORDER_DATE>
    <orderLines>
      <orderLine>
        <ITEM_NUMBER>
          A1122
        </ITEM_NUMBER>
        <ITEM_QUANTITY>
          55
        </ITEM_QUANTITY>
        <ITEM_PRICE>2.65</ITEM_PRICE>
      </orderLine>
      <orderLine>
        <ITEM_NUMBER>A1342</ITEM_NUMBER>
        <ITEM_QUANTITY>32</ITEM_QUANTITY>
        <ITEM_PRICE>2.25</ITEM_PRICE>
      </orderLine>
    </orderLines>
  </order>

...
</orderMessage>
```

PL/SQL DOM Model APIs

- **DBMS_XMLDOM is the basic DOM API**
  - Works well with XMLType (no re-parsing)
  - Has functions to work with the DOM model
  - Works with the DOM XML object

DBMS_XMLPARSER

- **DBMS_XMLPARSER parses XML documents**
- **Supports various input types:**
  - VARCHAR2
  - CLOB
  - Files
- **DBMS_XMLParser Returns:**
  - DBMS_XMLDOM.DOMDocument Object

DBMS_XMLPARSER – From CLOB

- **Using DBMS_XMLPARSER**
  - Multiple steps: Create parser, configure, parse, free

```
FUNCTION XMLPARSER_ExampleCLOB(pi_Msg IN CLOB) RETURN DBMS_XMLDOM.DOMDocument IS
  v_Parser DBMS_XMLPARSER.Parser;
  v_XMLDoc DBMS_XMLDOM.DOMDocument;
BEGIN
   v_Parser := DBMS_XMLPARSER.newParser;
   DBMS_XMLPARSER.setValidationMode(v_Parser, False);
   DBMS_XMLPARSER.setPreserveWhitespace(v_Parser, true);
   DBMS_XMLPARSER.parseClob(v_Parser, pi_Msg);
   v_XMLDoc := DBMS_XMLPARSER.getDocument(v_Parser);
   DBMS_XMLPARSER.freeParser(v_Parser);
   RETURN(v_XMLDoc);
END;
```

- **Using XMLType Parser**
  - Simpler, but slower – Something like 3 times slower!

```
PROCEDURE XMLPARSER_ExampleCLOB(pi_Msg IN CLOB) RETURN DBMS_XMLDOM.DOMDocument IS
   v_XMLDoc DBMS_XMLDOM.DOMDocument;
BEGIN
   v_XMLDoc := DBMS_XMLDOM.newDOMDocument(XMLType(pi_Msg));
   RETURN(v_XMLDoc);
END;
```

DBMS_XMLPARSER – From File

- **Using DBMS_XMLPARSER to parse from Files**
  - Specify directory with setBaseDir()
  - Use parse() to get file

```
FUNCTION XMLPARSER_ExampleFILE(pi_fileName IN VARCHAR2) RETURN
DBMS_XMLDOM.DOMDocument IS
   v_Parser DBMS_XMLPARSER.Parser;
   v_XMLDoc DBMS_XMLDOM.DOMDocument;
   v_BaseDir VARCHAR2(80) := 'C:\Docs\Papers\PLSQL_XML\xml_files';
BEGIN
   v_Parser := DBMS_XMLPARSER.newParser;
   DBMS_XMLPARSER.setValidationMode(v_Parser, False);
   DBMS_XMLPARSER.setPreserveWhitespace(v_Parser, true);

   DBMS_XMLPARSER.setBaseDir(v_Parser, v_BaseDir);
   DBMS_XMLPARSER.parse(v_Parser, pi_fileName);

   v_XMLDoc := DBMS_XMLPARSER.getDocument(v_Parser);
   DBMS_XMLPARSER.freeParser(v_Parser);

   RETURN(v_XMLDoc);
END;
```

DOM Model API

- **We already talked about DOM concepts**
- **In addition to producing XML, the DOM API can be used to access data from XML**
  - The DOM model is used for this more often
- **Using the DOM model you have a lot of flexibility and control**
- **Great if you don't know the structure already**
  - XPATH access kind of assumes the structure

Insert Records - DOM Version

```
PROCEDURE XMLDOM_DOMWalk_Example_INS(pi_XMLDoc IN DBMS_XMLDOM.DOMDocument) IS
   v_TopNode     DBMS_XMLDOM.DOMNode; v_TopNodeList DBMS_XMLDOM.DOMNodeList;
   v_Node1       DBMS_XMLDOM.DOMNode; v_Node1Name VARCHAR2(80);  v_Node1List   DBMS_XMLDOM.DOMNodeList;
   v_Node2       DBMS_XMLDOM.DOMNode; v_Node2Name VARCHAR2(80);  v_Node2List   DBMS_XMLDOM.DOMNodeList;
   v_Node3       DBMS_XMLDOM.DOMNode; v_Node3Name VARCHAR2(80);  v_Node3List   DBMS_XMLDOM.DOMNodeList;
   v_Node4       DBMS_XMLDOM.DOMNode; v_Node4Name VARCHAR2(80);  v_Node4List   DBMS_XMLDOM.DOMNodeList;
   v_Node5       DBMS_XMLDOM.DOMNode;  v_Node5Name VARCHAR2(80); v_Node5List   DBMS_XMLDOM.DOMNodeList;
   v_TextNode    DBMS_XMLDOM.DOMNode;   v_TextValue   VARCHAR2(80);
   v_LocationCnt INTEGER := 0;          v_ItemCnt     INTEGER := 0;
   v_AttrNode    DBMS_XMLDOM.DOMNode;   v_AttrMap     DBMS_XMLDOM.DOMNamedNodeMap;
   r_inventoryUpdate INVENTORY_UPDATE%ROWTYPE;
   v_CustNumber   CUSTOMER.CUSTOMER_NUMBER%TYPE;
   v_customer_ID  CUSTOMER.customer_id%TYPE;
   v_locLabel     CUSTOMER_LOCATION.LOCATION_LABEL%TYPE;
   v_itemNumber   ITEM.ITEM_NUMBER%TYPE;  v_itemID ITEM.ITEM_ID%TYPE;
   v_inventory_update_id INVENTORY_UPDATE.INVENTORY_UPDATE_ID%TYPE;
   b_itemOK BOOLEAN;   v_error_svr  VARCHAR2(1);    v_error_txt VARCHAR2(240);
BEGIN
   v_TopNode := DBMS_XMLDOM.makeNode(DBMS_XMLDOM.getDocumentElement(pi_XMLDoc));
   p(DBMS_XMLDOM.getNodeName(v_TopNode));
   v_TopNodeList := DBMS_XMLDOM.getChildNodes(v_TopNode);
   -- Loop through children of top node
   FOR n1 IN 0..DBMS_XMLDOM.getLength(v_TopNodeList)-1 LOOP
      v_Node1 := DBMS_XMLDOM.item(v_TopNodeList, n1);
      v_Node1Name := DBMS_XMLDOM.getNodeName(v_Node1);
      v_Node1List := DBMS_XMLDOM.getChildNodes(v_Node1);
      -- Determine specific processing for each child by name
      CASE v_Node1Name
         WHEN 'messageSentTime' THEN  -- Simply save this text element into the record
            v_TextNode := DBMS_XMLDOM.item(v_Node1List, 0);
            r_inventoryUpdate.Inventory_Update_Date
                   := xml_to_date(DBMS_XMLDOM.getNodeValue(v_TextNode));
```

Insert Records - DOM Version

```
WHEN 'customer' THEN
   -- This node has children, so loop through them
   FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
      v_Node2 := DBMS_XMLDOM.item(v_Node1List, n2);
      v_Node2Name := DBMS_XMLDOM.getNodeName(v_Node2);
      v_Node2List := DBMS_XMLDOM.getChildNodes(v_Node2);
      CASE v_Node2Name  -- More specific processing by name
         WHEN 'custNumber' THEN
            v_TextNode := DBMS_XMLDOM.item(v_Node2List, 0);
            v_CustNumber := DBMS_XMLDOM.getNodeValue(v_TextNode);
            -- Do a lookup based on this data element, save for later
            v_customer_ID := tapi$customer.get_id(v_CustNumber);
         WHEN 'custName' THEN  -- This one is not actually used
            v_TextNode := DBMS_XMLDOM.item(v_Node2List, 0);
         ELSE
            NULL; -- I don't care about any other elements
      END CASE;
   END LOOP;
WHEN 'locationList' THEN
   -- Loop through the locations specified in the file
   FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
      v_Node2 := DBMS_XMLDOM.item(v_Node1List, n2);
      v_Node2Name := DBMS_XMLDOM.getNodeName(v_Node2);
      v_Node2List := DBMS_XMLDOM.getChildNodes(v_Node2);
      CASE v_Node2Name
         WHEN 'location' THEN  -- This is the only node for this one
            v_LocationCnt := v_LocationCnt + 1;
            FOR n3 IN 0..DBMS_XMLDOM.getLength(v_Node2List)-1 LOOP
               v_Node3 := DBMS_XMLDOM.item(v_Node2List, n3);
               v_Node3Name := DBMS_XMLDOM.getNodeName(v_Node3);
               v_Node3List := DBMS_XMLDOM.getChildNodes(v_Node3);
               CASE v_Node3Name
                  WHEN 'locName' THEN
                     v_TextNode := DBMS_XMLDOM.item(v_Node3List, 0);
                     v_LocLabel := DBMS_XMLDOM.getNodeValue(v_TextNode);
                     -- Do a lookup based on this data element
                     r_inventoryUpdate.location_id
                         := tapi$customer_location.get_id(v_locLabel);
```

Insert Records - DOM Version

```
WHEN 'itemList' THEN
   v_ItemCnt := 0;
   -- Loop through the item nodes under itemList
   FOR n4 IN 0..DBMS_XMLDOM.getLength(v_Node3List)-1 LOOP
      v_Node4 := DBMS_XMLDOM.item(v_Node3List, n4);
      v_Node4Name := DBMS_XMLDOM.getNodeName(v_Node4);
      v_Node4List := DBMS_XMLDOM.getChildNodes(v_Node4);
      CASE v_Node4Name
         WHEN 'item' THEN
            -- Reset previous item values, since r_inventoryUpdate is bein
            -- The other fields in this record stay the same from item t
            r_inventoryUpdate.Inventory_Update_Id := NULL;
            r_inventoryUpdate.Item_Id := NULL;
            r_inventoryUpdate.On_Hand_Quantity := NULL;
            r_inventoryUpdate.Optimum_Quantity := NULL;
            r_inventoryUpdate.Per_Month_Usage_Quantity := NULL;

            v_ItemCnt := v_ItemCnt + 1;
            -- The item node has an attribute
            v_AttrMap := DBMS_XMLDOM.getAttributes(v_Node4);
            v_AttrNode := DBMS_XMLDOM.getNamedItem(v_AttrMap, 'number');
            v_itemNumber := DBMS_XMLDOM.getNodeValue(v_AttrNode);

            -- Do a lookup to the ITEM table, using the "number" attr,
            BEGIN
               r_inventoryUpdate.Item_Id  := tapi$item.get_id(v_itemNumber);
               b_itemOK := True;
            EXCEPTION
               WHEN tapi$util.e_invalid_mnemonic THEN
               DBMS_OUTPUT.put_line('ERROR!! - Could not find item: '
                         ||v_itemNumber);
               b_itemOK := False;
            END;
            END;
```

Insert Records - DOM Version

```
            -- Loop through the item elements
            FOR n5 IN 0..DBMS_XMLDOM.getLength(v_Node4List)-1 LOOP
               v_Node5 := DBMS_XMLDOM.item(v_Node4List, n5);
               v_Node5Name := DBMS_XMLDOM.getNodeName(v_Node5);
               v_Node5List := DBMS_XMLDOM.getChildNodes(v_Node5);
               v_TextNode  := DBMS_XMLDOM.item(v_Node5List, 0);
               v_TextValue := DBMS_XMLDOM.getNodeValue(v_TextNode);
               -- Based on the name of the element determine which column
               CASE v_Node5Name
                  WHEN 'onHandQty' THEN
                     r_inventoryUpdate.On_Hand_Quantity := v_TextValue;
                  WHEN 'optimumQty' THEN
                     r_inventoryUpdate.Optimum_Quantity := v_TextValue;
                  WHEN 'monthlyUseQty' THEN
                     r_inventoryUpdate.Per_Month_Usage_Quantity
                           := v_TextValue;
                  ELSE
                     NULL;
               END CASE;
            END LOOP;
         ELSE
            NULL;
      END CASE;
      -- Insert record after every item node
      IF b_itemOK THEN
         tapi$inventory_update.add(pi_table_rec => r_inventoryUpdate
             ,po_error_svr => v_error_svr
             ,po_error_txt => v_error_txt
             );
         IF v_error_svr = 'S' THEN
            DBMS_OUTPUT.PUT_LINE('......Inserting record - ID: '
                ||r_inventoryUpdate.Inventory_Update_Id);
         ELSE
            DBMS_OUTPUT.put_line('ERROR!! - Could not insert record - '
                     ||v_error_txt);
         END IF;
```

Insert Records - DOM Version

```
                              END IF;
                           END LOOP;  -- item loop
                        ELSE
                           NULL;
                     END CASE;
                  END LOOP; -- of location loop
               ELSE  -- not location
                  NULL; -- I don't care about any other nodes
            END CASE;
         END LOOP;
      ELSE
         p('.....Unhandled Node');
   END CASE;

END LOOP;
END;
```

DOM Walk Example

- Starting with a DOMDocument object (already parsed!)

```
PROCEDURE XMLDOM_DOMWalk_Example_INS(pi_XMLDoc IN DBMS_XMLDOM.DOMDocument) IS
```

- Create a node reference for the entire document

```
v_TopNode := DBMS_XMLDOM.makeNode(DBMS_XMLDOM.getDocumentElement(pi_XMLDoc));
```

- Retrieve a list of the children nodes for the document

```
v_TopNodeList := DBMS_XMLDOM.getChildNodes(v_TopNode);
```

- Loop through the list of children nodes

```
 FOR n1 IN 0..DBMS_XMLDOM.getLength(v_TopNodeList)-1 LOOP
…
 END LOOP;
```

- For each child node, retrieve the node, get its name, and list of children

```
 v_Node1 := DBMS_XMLDOM.item(v_TopNodeList, n1);
 v_Node1Name := DBMS_XMLDOM.getNodeName(v_Node1);
 v_Node1List := DBMS_XMLDOM.getChildNodes(v_Node1);
```

DOM Walk Example

- **Different child notes need different handling…**

```
 CASE v_Node1Name
         WHEN 'messageSentTime' THEN
…
         WHEN 'customer' THEN
…
 END CASE
```

- **For the "messageSentTime" node, simply retrieve its value**
  - Retrieve the actual node into *v_TextNode*
  - Then get the value of the text node and convert it to a date

```
WHEN 'messageSentTime' THEN  -- Simply save this text element into the record
           v_TextNode := DBMS_XMLDOM.item(v_Node1List, 0);
           r_inventoryUpdate.Inventory_Update_Date
                 := xml_to_date(DBMS_XMLDOM.getNodeValue(v_TextNode));
```

- **For the "customer" node we will need to loop through *its* children**

```
WHEN 'customer' THEN
    -- This node has children, so loop through them
    FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
   …
    END LOOP;
```

DOM Walk Example

- This same type of processing is repeated at every level

```
FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
   v_Node2 := DBMS_XMLDOM.item(v_Node1List, n2);
   v_Node2Name := DBMS_XMLDOM.getNodeName(v_Node2);
   v_Node2List := DBMS_XMLDOM.getChildNodes(v_Node2);
   CASE v_Node2Name  -- More specific processing by name
      WHEN 'custNumber' THEN
         v_TextNode := DBMS_XMLDOM.item(v_Node2List, 0);
         v_CustNumber := DBMS_XMLDOM.getNodeValue(v_TextNode);
          -- Do a lookup based on this data element, save for later
         v_customer_ID := tapi$customer.get_id(v_CustNumber);
      WHEN 'custName' THEN  -- This one is not actually used
         v_TextNode := DBMS_XMLDOM.item(v_Node2List, 0);
      ELSE
         NULL; -- I don't care about any other elements
   END CASE;
END LOOP;
```

- Retrieving attribute values is slightly different

```
   -- The item node has an attribute
   v_AttrMap := DBMS_XMLDOM.getAttributes(v_Node4);
   v_AttrNode := DBMS_XMLDOM.getNamedItem(v_AttrMap, 'number');
   v_itemNumber := DBMS_XMLDOM.getNodeValue(v_AttrNode);
```

DOM Walk Example

- **Retrieve information for data updates when it is encountered…**

```
FOR n1 IN 0..DBMS_XMLDOM.getLength(v_TopNodeList)-1 LOOP
  ...
  CASE v_Node1Name
  ...
      r_inventoryUpdate.Inventory_Update_Date
                    := xml_to_date(DBMS_XMLDOM.getNodeValue(v_TextNode));
    WHEN 'locationList' THEN
     FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
       ...
       CASE v_Node2Name
         WHEN 'location' THEN
           FOR n3 IN 0..DBMS_XMLDOM.getLength(v_Node2List)-1 LOOP
             ...
             CASE v_Node3Name
               WHEN 'locName' THEN
                 ...
                   r_inventoryUpdate.location_id
                                  := tapi$customer_location.get_id(v_locLabel);
              WHEN 'itemList' THEN
              FOR n4 IN 0..DBMS_XMLDOM.getLength(v_Node3List)-1 LOOP
                ...
                  CASE v_Node4Name
                    WHEN 'item' THEN
                      ...
                        v_itemNumber := DBMS_XMLDOM.getNodeValue(v_AttrNode);
                       r_inventoryUpdate.Item_Id  := tapi$item.get_id(v_I…
                        ...
                       FOR n5 IN 0..DBMS_XMLDOM.getLength(v_Node4List)-1 LOOP
                         ...
                         CASE v_Node5Name
                           WHEN 'onHandQty' THEN
                            r_inventoryUpdate.On_Hand_Quantity := v_TextValue;
```

DOM Walk Example

- **Insert the data in the correct location in the loops…**

```
FOR n1 IN 0..DBMS_XMLDOM.getLength(v_TopNodeList)-1 LOOP
  ...
  CASE v_Node1Name
  ...
    WHEN 'locationList' THEN
      FOR n2 IN 0..DBMS_XMLDOM.getLength(v_Node1List)-1 LOOP
        ...
        CASE v_Node2Name
          WHEN 'location' THEN
            FOR n3 IN 0..DBMS_XMLDOM.getLength(v_Node2List)-1 LOOP
              ...
              CASE v_Node3Name
                ...
                WHEN 'itemList' THEN
                  FOR n4 IN 0..DBMS_XMLDOM.getLength(v_Node3List)-1 LOOP
                    ...
                      CASE v_Node4Name
                        WHEN 'item' THEN
                          ...
             tapi$inventory_update.add(pi_table_rec => r_inventoryUpdate...
                                ...
                             END LOOP;  -- item loop
                          ELSE
                             NULL;
                          END CASE;
                       END LOOP; -- of location loop
                    ELSE  -- not location
                       NULL; -- I don't care about any other nodes
                 END CASE;
              END LOOP;
           ELSE
```

Another approach: Loop through list

```
PROCEDURE XMLDOM_ALL_ELEM_EXAMPLE(pi_XMLDoc IN DBMS_XMLDOM.DOMDocument) IS
   v_NodeList  DBMS_XMLDOM.DOMNodeList;
   v_NodeName  VARCHAR2(80);
   v_Node      DBMS_XMLDOM.DOMNode;
   v_ChildNode DBMS_XMLDOM.DOMNode;
   v_Value     VARCHAR2(80);
   v_AttrNode    DBMS_XMLDOM.DOMNode;
   v_AttrMap     DBMS_XMLDOM.DOMNamedNodeMap;
   TYPE UpdateRec_tabtype IS TABLE OF inventory_update%ROWTYPE INDEX BY BINARY_INTEGER;
   r_invUpdate inventory_update%ROWTYPE;
   t_invUpdate UpdateRec_tabtype;
   i_invUpdate INTEGER := 0;
 v_error_svr  VARCHAR2(1);    v_error_txt VARCHAR2(240);
BEGIN
   -- Get a list of all of the elements
   v_NodeList := DBMS_XMLDom.getElementsbyTagName(pi_XMLDoc, '*');
   FOR i IN 0..DBMS_XMLDOM.getLength(v_NodeList)-1 LOOP
      v_Node      := DBMS_XMLDOM.item(v_NodeList, i);
      v_NodeName  := DBMS_XMLDOM.getNodeName(v_Node);
      v_ChildNode := DBMS_XMLDOM.getFirstChild(v_Node);
      IF DBMS_XMLDOM.getNodeType(v_ChildNode) = DBMS_XMLDOM.TEXT_NODE THEN
         v_Value := DBMS_XMLDOM.getNodeValue(v_ChildNode);
      ELSE
         v_Value := NULL;
      END IF;
      CASE v_NodeName
         WHEN 'messageSentTime' THEN
            r_invUpdate.inventory_update_date := xml_to_date(v_Value);
         WHEN 'locName' THEN
            r_invUpdate.location_id := tapi$customer_location.get_id(v_Value);
         WHEN 'itemList' THEN
            NULL;
```

Another approach: Loop through list

```
         WHEN 'item' THEN
            v_AttrMap  := DBMS_XMLDOM.getAttributes(v_Node);
            v_AttrNode := DBMS_XMLDOM.getNamedItem(v_AttrMap, 'number');
            v_Value    := DBMS_XMLDOM.getNodeValue(v_AttrNode);
            -- Initialize new record
            BEGIN
               i_InvUpdate := i_InvUpdate + 1;
               t_invUpdate(i_invUpdate).item_id               := tapi$item.get_id(v_Value);
               t_invUpdate(i_invUpdate).location_id           := r_invUpdate.location_id;
               t_invUpdate(i_invUpdate).inventory_update_date := r_invUpdate.inventory_update_date;
            EXCEPTION
               WHEN tapi$util.e_invalid_mnemonic THEN
                   i_InvUpdate := i_InvUpdate - 1;
                   DBMS_OUTPUT.put_line('..............ERROR!! - Could not find item: '||v_Value);
            END;
         WHEN 'onHandQty' THEN
            t_invUpdate(i_invUpdate).on_hand_quantity := v_Value;
         WHEN 'optimumQty' THEN
            t_invUpdate(i_invUpdate).optimum_quantity := v_Value;
         WHEN 'monthlyUseQty' THEN
            t_invUpdate(i_invUpdate).per_month_usage_quantity := v_Value;
         ELSE
            NULL;
      END CASE;
   END LOOP;
   FOR i IN 1..i_InvUpdate LOOP
      tapi$inventory_update.add(pi_table_rec => t_InvUpdate(i)
                               ,po_error_svr => v_error_svr, po_error_txt => v_error_txt
                               );
      IF v_error_svr = 'S' THEN
          DBMS_OUTPUT.PUT_LINE('......Inserting record - ID: '||t_InvUpdate(i).Inventory_Update_Id);
      ELSE
          DBMS_OUTPUT.put_line('ERROR!! - Could not insert record - '||v_error_txt);
      END IF;
   END LOOP;
END;
```

DOM Error Types

```
EXCEPTION
     WHEN DBMS_XMLDOM.INDEX_SIZE_ERR THEN

     WHEN DBMS_XMLDOM.DOMSTRING_SIZE_ERR THEN

     WHEN DBMS_XMLDOM.HIERARCHY_REQUEST_ERR THEN

     WHEN DBMS_XMLDOM.WRONG_DOCUMENT_ERR THEN

     WHEN DBMS_XMLDOM.INVALID_CHARACTER_ERR THEN

     WHEN DBMS_XMLDOM.NO_DATA_ALLOWED_ERR THEN

     WHEN DBMS_XMLDOM.NO_MODIFICATION_ALLOWED_ERR THEN

     WHEN DBMS_XMLDOM.NOT_FOUND_ERR THEN

     WHEN DBMS_XMLDOM.NOT_SUPPORTED_ERR THEN

     WHEN DBMS_XMLDOM.INUSE_ATTRIBUTE_ERR THEN
```

Which shredding method should I use?

- **XPATH with XMLType**
  - Useful when you only need to parse part of the file
  - Use when you know the XML format for sure
  - Slower than DBMS methods
- **DBMS_XMLPARSE & DBMS_XMLDOM**
  - Very flexible, therefore useful for complicated XML
  - Good when you have to parse the entire XML
  - Better ability to inspect XML format
    - Write your own XML utilities!
    - Useful when you have to parse multiple similar structures
  - Faster! (C based code in kernel)
- **DBMS_XMLSTORE**
  - Easiest to use for single table data
  - Most useful if you control the XML layout
  - More powerful when combined with XSL or XMLDOM

Simple XML Shred Performance Test

Example XML to shred:

```
<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
`         <itemList>
            <item number="W1234">
 …
            </item>
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
```

```
.
Number of runs: 1000
.
XPATH access to XMLType Parse    - elapsed time: 07.531 Sec
.
XPATH access to XMLType No Parse - elapsed time: 06.509 Sec
.
XMLDOM access to DOMDocument     - elapsed time: 04.857 Sec
.
XMLPARSER - Parse only           - elapsed time: 00.360 Sec
.
XMLType - Parse only             - elapsed time: 00.732 Sec
```

XML Storage Methods in Oracle

- **There are many different ways to store XML data in Oracle:**
  - Shredded into your own tables
    - Covered previously
  - CLOB columns
  - XMLType columns (XDB)
  - XMLType tables (as LOB)
  - XML Schema-based table Storage (XDB)
  - External XML files (XDB Repository)
- **There are also different ways to access the data once stored:**
  - XPATH Based searches
  - Oracle Text searches

Storing XML in CLOB columns

- **XML Can be stored as CLOB columns in Oracle**
- **Advantages**
  - Non-Oracle Specific storage
    - Client access may not recognize XMLType columns
  - XML does not need to be valid to store
  - Preserves original format
  - Easily allows multiple XML formats in same column
    - I.e. for schema evolution, etc.
  - Easy to tie XML into relational model via other columns in table
- **Disadvantages**
  - Incur parsing overhead to access XML data
  - Poor performance for queries
    - Though can use Oracle Text queries and function indexes to improve
  - Limited size of XML
    - Caused because CLOB → XMLType limited to 64K
  - CLOB memory management needed

Example Loading into CLOB cols

```
CREATE TABLE XML_DATA_CLOB
   (XML_DATA_ID   NUMBER(10)    NOT NULL
   ,CONTENT_TYPE  VARCHAR2(20)  NOT NULL
   ,CONTENT_DESCR VARCHAR2(240)
   ,XML_DATA CLOB               NOT NULL
   );


INSERT INTO Xml_Data_Clob(xml_data_id, Content_Type, Content_Descr, Xml_Data)
  VALUES(xml_data_seq.nextval
        ,'INVENTORY_UPDATE'
        ,'Test inventory Update'
        ,
'<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
         <itemList>
            <item number="W1234">
               <onHandQty>256</onHandQty>
               <optimumQty>1000</optimumQty>
               <monthlyUseQty>523</monthlyUseQty>
            </item>
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
');
1 row inserted
```

Example Loading into CLOB cols

```
INSERT INTO Xml_Data_Clob(xml_data_id, Content_Type, Xml_Data)
  VALUES(xml_data_seq.nextval
        ,'CUSTOMER_REPORT'
        ,
'<customers>
  <customer>
    <customerID>1</customerID>
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
  </customer>
  <customer>
    …
  </customer>
</customers> ');

select xml_data_id, content_type, xml_data FROM XML_Data_Clob;

XML_DATA_ID CONTENT_TYPE         XML_DATA
----------- -------------------- ----------------------------------------------------------------
          3 INVENTORY_UPDATE     <inventoryUpdate>
                                    <messageSentTime>2007-01-15:12:15Z</messageSentTime>
                                    <customer>
                                       <custNumber>A12345</custNumber>
                                       <custName>Spacely Sprockets</custName>
                                    </customer>
                                    <locationList>
                                       <location>
                                          <locName>INDIANAPOLIS</locName>
                                          …

         10 CUSTOMER_REPORT      <customers>
                                   <customer>
                                     <customerID>1</customerID>
                                     <customerNumber>A12345</customerNumber>
                                     <customerName>Spacely Sprockets</customerName>
                                   </customer>
                                   <customer>
                                     <customerID>2</customerID>
                                     …
```

Storing XML in XMLType columns

- **XML Can be stored in XMLType columns**
  - Not really XDB, whole XML still in single columns
- **Advantages**
  - Better performance than CLOB (pre-parsed)
  - Also allows multiple XML formats in same column
  - Also easy to tie into relational model via other columns
  - No need for CLOB memory management (I.e. temporary CLOBs, DBMS_LOB, etc)
- **Disadvantages**
  - Oracle-specific datatype – May not work with some clients
  - XML *must* be well behaved (I.e. valid)
  - Does not preserve original format
  - Poor performance for queries
    - Though can use Oracle Text queries and function indexes to improve
  - Limited size of XML
    - Caused because CLOB → XMLType limited to 64K

Example Loading into XMLType cols

```
CREATE TABLE XML_DATA_XMLTYPE
 (XML_DATA_ID NUMBER(10) NOT NULL
 ,CONTENT_TYPE VARCHAR2(20) NOT NULL
 ,CONTENT_DESCR VARCHAR2(240)
 ,XML_DATA XMLTYPE NOT NULL
 );

SQL> desc xml_data_xmltype;
Name          Type          Nullable Default Comments
------------- ------------- -------- ------- --------
XML_DATA_ID   NUMBER(10)
CONTENT_TYPE  VARCHAR2(20)
CONTENT_DESCR VARCHAR2(240) Y
XML_DATA      XMLTYPE

INSERT INTO Xml_Data_XMLType(xml_data_id, Content_Type, Content_Descr, Xml_Data)
  VALUES(xml_data_seq.nextval
        ,'INVENTORY_UPDATE'
        ,'Test inventory Update'
        ,XMLType('<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
         <itemList>
            …
            </item>
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
'));

1 row inserted
```

Example Loading into XMLType cols

```
INSERT INTO Xml_Data_XMLType(xml_data_id, Content_Type, Xml_Data)
  VALUES(xml_data_seq.nextval
        ,'CUSTOMER_REPORT'
        ,XMLType('<customers>
  <customer>
    <customerID>1</customerID>
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
  </customer>
  <customer>
    …
  </customer>
</customers> '));

select xml_data_id, content_type, xml_data FROM XML_Data_Clob;

XML_DATA_ID CONTENT_TYPE         XML_DATA
----------- -------------------- ----------------------------------------------------------------
          3 INVENTORY_UPDATE     <inventoryUpdate>
                                    <messageSentTime>2007-01-15:12:15Z</messageSentTime>
                                    <customer>
                                       <custNumber>A12345</custNumber>
                                       <custName>Spacely Sprockets</custName>
                                    </customer>
                                    <locationList>
                                       <location>
                                          <locName>INDIANAPOLIS</locName>
                                          …

         10 CUSTOMER_REPORT      <customers>
                                   <customer>
                                     <customerID>1</customerID>
                                     <customerNumber>A12345</customerNumber>
                                     <customerName>Spacely Sprockets</customerName>
                                   </customer>
                                   <customer>
                                     <customerID>2</customerID>
                                     …
```

Storing XML in XMLType Table

- **XML Can be stored in XMLType Table**
  - An XML Type table, not a column in a table
- **Advantages**
  - Better performance than CLOB (pre-parsed)
  - Also allows multiple XML formats in same column
  - No need for CLOB memory management (I.e. temporary CLOBs, DBMS_LOB, etc)
- **Disadvantages**
  - Oracle-specific datatype – May not work with some clients
  - XML *must* be well behaved (I.e. valid)
  - Does not preserve original format
  - Poor performance for queries
    - Though can use Oracle Text queries and function indexes to improve
  - Limited size of XML
    - Caused because CLOB → XMLType limited to 64K
  - More difficult to tie into relational model (i.e. no other columns)

Loading into XMLType table

```
CREATE TABLE xml_data_xmltype_table of XMLType;

SQL> desc Xml_Data_XMLType_Table;
Name            Type    Nullable Default Comments
--------------- ------- -------- ------- --------
SYS_NC_ROWINFO$ XMLTYPE Y

INSERT INTO Xml_Data_XMLType_Table
  VALUES(XMLType('<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
         <itemList>
            …
            </item>
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
'));

1 row inserted
INSERT INTO Xml_Data_XMLType_Table
  VALUES(XMLType('<customers>
  <customer>
    <customerID>1</customerID>
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
  </customer>
  <customer>
    …
  </customer>
</customers> '));
```

Loading into XMLType table

```
SELECT *
  FROM xml_data_xmltype_table;

SYS_NC_ROWINFO$
--------------------------------------------------------------------------------
<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
         <itemList>
            <item number="W1234">
         ...
      </location>
      <location>
         <locName>LOUISVILLE-A</locName>
         <itemList>

<customers>
  <customer>
    <customerID>1</customerID>
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
  </customer>
  <customer>
    <customerID>2</customerID>
    <customerNumber>A2765</customerNumber>
    <customerName>Cogswells Cogs</customerName>
  </customer>
</customers>
```

Storing XML in Schema-Based Tables

- **XDB has a method to store XML "Automatically shredded" in Oracle tables**
  - Based on an XML Schema
- **Advantages**
  - Much better performance
    - XPATH queries automatically rewritten to SQL queries
  - Allows larger XML sizes
- **Disadvantages**
  - You need to have an XML Schema
  - Changes in XML format much more difficult
    - (I.e. High cost for schema evolution)
  - Only store a single XML type in each object
  - More difficult to integrate with existing relational data
  - XML *must* be well-behaved (I.e. valid XML)
  - Limited data-replication support

What is XDB?

- **XDB = XML Database**
- **XDB is a set of Oracle XML utilities**
  - PL/SQL packages
  - Tables
  - XMLType
- **XMLType is the primary XML manipulation tool**
- **XDB Allows XML to be stored in relational tables**
  - Faster Access to pre-shreded data
  - DOM Model loaded on the fly when accessing
- **XML Repository**
  - An internet repository for XML files

Steps needed to store in XDB

1. **Register XML Schema**
   - You need to have an XML Schema defined
   - It gets registered in XDB
2. **Create XMLType table or column**
   - Associated with registered schema
   - Option: use DEFAULT TABLE
3. **Actually load the data into the XDB object**
   - Ends up being stored in multiple objects

Example XML Schema

```
<?xml version = '1.0'?>
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns:xdb="http://xmlns.oracle.com/xdb" version="1.0">
      <xsd:element name="customers" xdb:defaultTable="CUSTOMER_XDB_TABLE">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="customer" minOccurs="0" maxOccurs="unbounded">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="customerID" type="xsd:long" minOccurs="0"/>
                        <xsd:element name="customerNumber" nillable="true" minOccurs="0">
                           <xsd:simpleType>
                              <xsd:restriction base="xsd:string">
                                 <xsd:maxLength value="20"/>
                              </xsd:restriction>
                           </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="customerName" nillable="true" minOccurs="0">
                           <xsd:simpleType>
                              <xsd:restriction base="xsd:string">
                                 <xsd:maxLength value="40"/>
                              </xsd:restriction>
                           </xsd:simpleType>
                        </xsd:element>
                     </xsd:sequence>
                     <xsd:attribute name="num" type="xsd:integer"/>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
```

Register the XML Schema

Some code I added to delete any existing version of the schema

Reads the XSD file shown in the previous slide into an XMLType variable

The "schemaURL" is the unique name of the schema.

```
DECLARE
   x_XML XMLType;
   v_Dummy VARCHAR2(1);
BEGIN
   BEGIN
      SELECT 'Y' INTO v_Dummy
        FROM user_xml_schemas xs
       WHERE xs.schema_url = 'http://xmlns.cda-llc.com/customerReport.xsd';
       DBMS_XMLSchema.deleteSchema(
                 schemaURL => 'http://xmlns.cda-llc.com/customerReport.xsd'
                 ,delete_option => DBMS_XMLSchema.DELETE_CASCADE_FORCE
                 );
       DBMS_OUTPUT.PUT_LINE('Deleted existing version of schema');
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         NULL;
   END;

   x_XML := XMLType(bfilename('XML_DIR'
                              ,'customerReport.xsd')
                              ,nls_charset_id('AL32UTF8') );



   DBMS_XMLSchema.registerSchema(
                 schemaURL => 'http://xmlns.cda-llc.com/customerReport.xsd'
                ,schemaDoc => x_XML
                );
   DBMS_OUTPUT.PUT_LINE('Registered schema from file');


END;
/
```

What happened when I did this?

- **The Schema was listed as a registered schema**
  - Data Dictionary Views: ALL_XML_SCHEMAS
- **Many Oracle artifacts are created**
  - An Object Table
  - A trigger on the table
  - Many Oracle Types
  - Many LOBs for storage….
- **XML Stored in type will be auto-shredded into these artifacts.**

What happened when I did this?

```
SELECT owner, schema_url FROM ALL_XML_schemas;

OWNER                SCHEMA_URL
-------------------- ----------------------------------------------------
XMLEX                http://xmlns.cda-llc.com/customerReport.xsd
XDB                  http://xmlns.oracle.com/xdb/dav.xsd
XDB                  http://xmlns.oracle.com/xdb/XDBStandard.xsd
<…More…>

SELECT object_name, object_type, o.created
  from user_objects o
 WHERE o.created > (sysdate - 1);

OBJECT_NAME                    OBJECT_TYPE
------------------------------ -------------------
customer382_T                  TYPE
customer383_COLL               TYPE
customers381_T                 TYPE
CUSTOMER_XDB_TABLE             TABLE
SYS_C0020386                   INDEX
SYS_LOB0000079631C00008$$      LOB
SYS_LOB0000079631C00007$$      LOB
SYS_LOB0000079631C00005$$      LOB
SYS_LOB0000079631C00004$$      LOB
CUSTOMER_XDB_TABLE$xd          TRIGGER
```

Description of Objects Created

```
desc CUSTOMER_XDB_TABLE;

Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
TABLE of SYS.XMLTYPE(XMLSchema "http://xmlns.cda-llc.com/customerReport.xsd" Element "customers")
    STORAGE Object-relational TYPE "customers381_T"


desc "customers381_T"

Name                                      Null?    Type
----------------------------------------- -------- ----------------------------
SYS_XDBPD$                                         XDB.XDB$RAW_LIST_T
customer                                           customer383_COLL

desc "customer383_COLL"

 "customer383_COLL" VARRAY(2147483647) OF customer382_T
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SYS_XDBPD$                                         XDB.XDB$RAW_LIST_T
 num                                                NUMBER(38)
 customerID                                         NUMBER(20)
 customerNumber                                     VARCHAR2(20 CHAR)
 customerName                                       VARCHAR2(40 CHAR)

desc "customer382_T"

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 SYS_XDBPD$                                         XDB.XDB$RAW_LIST_T
 num                                                NUMBER(38)
 customerID                                         NUMBER(20)
 customerNumber                                     VARCHAR2(20 CHAR)
 customerName                                       VARCHAR2(40 CHAR)
```

Example loading data into XDB

Some XML that conforms to the registered schema

Simply insert an XMLType into the Object Table

```
DECLARE
  v_CLOB CLOB;
BEGIN
  v_CLOB :=
'<?xml version="1.0"?>
<customers>
   <customer>
      <customerID>1</customerID>
      <customerNumber>A12345</customerNumber>
      <customerName>Spacely Sprockets</customerName>
   </customer>
   <customer>
      <customerID>2</customerID>
      <customerNumber>A2765</customerNumber>
      <customerName>Cogswells Cogs</customerName>
   </customer>
</customers>';


  INSERT INTO customer_xdb_table
     VALUES(XMLTYpe(v_CLOB));

END;
/
```

Example Loading Data into XDB

```
SELECT * FROM CUSTOMER_XDB_TABLE;

SYS_NC_ROWINFO$
--------------------------------------------------------------------------------
<?xml version="1.0"?>
<customers>
  <customer>
    <customerID>1</customerID>
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
  </customer>
  <customer>
    <customerID>2</customerID>
    <customerNumber>A2765</customerNumber>
    <customerName>Cogswells Cogs</customerName>
  </customer>
</customers>
```

XML *Must* Conform to Schema

```
DECLARE
  v_CLOB CLOB;
BEGIN
  v_CLOB :=
'<?xml version="1.0"?>
<customers>
   <customer>
      <customerID>1</customerID>
      <customerNo>A12345</customerNo>
      <customerName>Spacely Sprockets</customerName>
   </customer>
   <customer>
      <customerID>2</customerID>
      <customerNo>A2765</customerNo>
      <customerName>Cogswells Cogs</customerNo>
   </customer>
</customers>';
--  v_CLOB := xmlexample.get_customers_XMLGEN;
  DBMS_OUTPUT.PUT_LINE(v_CLOB);

  INSERT INTO customer_xdb_table
     VALUES(XMLTYpe(v_CLOB));

END;
/

ORA-30937: No schema definition for 'customerNo' (namespace '##local') in parent
'/customers/customer[1]'

ORA-06512: at line 21
```

XML *Must* Conform to Schema

```
DECLARE
  v_CLOB CLOB;
BEGIN
  v_CLOB :=
'<?xml version="1.0"?>
<customers>
   <customer>
      <customerID>1</customerID>
      <customerNumber>THIS_VALUE_IS_WAY_TOO_LONG</customerNumber>
      <customerName>Spacely Sprockets</customerName>
   </customer>
   <customer>
      <customerID>2</customerID>
      <customerNumber>A2765</customerNumber>
      <customerName>Cogswells Cogs</customerName>
   </customer>
</customers>';
--  v_CLOB := xmlexample.get_customers_XMLGEN;
  DBMS_OUTPUT.PUT_LINE(v_CLOB);

  INSERT INTO customer_xdb_table
     VALUES(XMLTYpe(v_CLOB));

END;
/

ORA-30951: Element or attribute at Xpath /customers/customer[1]/customerNumber exceeds
maximum length
ORA-06512: at line 21
```

XML Schema Annotations

- **You can control how the XDB objects are created with XML Schema annotations**
- **Special annotations in the XML Schema:**
  - xdb:defaultTable – Sets the object table name
  - xdb:tableProps – Specifies table properties
  - xdb:SQLName – Specifies name of object that maps to the element (I.e. column name or type name)
  - xdb:SQLType – Specifies the type of the object
  - xdb:SQLCollType – Specifies the collection names (I.e. can override the "customer383_coll" name
  - More……

Which storage method should I choose?

- **It depends upon many factors**
  - Do I need to query the contents a lot?
  - Do I really want to get into all that XDB stuff?
  - What sort of performance do I need?
  - How large are my XML documents?
  - How many documents do I need to store?
  - Do all of the documents have the same structure?
  - Is the structure going to change over time?

Getting XML Documents into Oracle

- **Client call to PL/SQL**
  - Client (VB, Java, C) reads XML file and inserts it into table
- **Reading files from server**
  - UTL_FILE or XML methods to get file from server
- **Reading XML from websites or web services**
  - HTTP access methods can be used to retrieve XML
- **Using SQL*Loader**
  - SQL*Loader has special XML methods
- **Oracle AQ Messages**

Reading files from server Example

```
CREATE OR REPLACE DIRECTORY XML_DIR as 'C:\Docs\Papers\PLSQL_XML\xml_files';


Directory created

INSERT INTO xml_data_xmltype(xml_data_id, content_type, xml_data)
  VALUES(xml_data_seq.nextval
        ,'XML Order'
        ,XMLType(bfilename('XML_DIR', 'order44.xml'), nls_charset_id('AL32UTF8') )
        );


1 row inserted


INSERT INTO xml_data_xmltype(xml_data_id, content_type, xml_data)
  VALUES(xml_data_seq.nextval
        ,'XML Order'
        ,XMLType(bfilename('XML_DIR', 'order45.xml'), nls_charset_id('AL32UTF8') )
        );


1 row inserted

SELECT xml_data_id, content_type, xml_data
 from xml_data_xmltype;


XML_DATA_ID CONTENT_TYPE         XML_DATA
----------- -------------------- ------------------------------------------------------------
         43 XML Order            <order>
                                   <ORDER_NUMBER>Z1234-57A</ORDER_NUMBER>
                                   <ORDER_DATE>2007-01-15</ORD

         44 XML Order            <order>
                                   <ORDER_NUMBER>Z5299-23C</ORDER_NUMBER>
                                   <ORDER_DATE>2007-02-27</ORD
```

Reading XML from websites Example

```
DECLARE
   x_OTNFeed XMLType;
BEGIN
   x_OTNFeed :=
SYS.HTTPUriType.createURI('http://www.oracle.com/technology/syndication/rss_otn_news.xml').getXML();
   INSERT INTO XML_Data_XMLType(Xml_Data_Id, Content_Type, Xml_Data)
      VALUES(xml_data_seq.nextval, 'OTN RSS Feed', x_OTNFeed);
END;
/
SELECT xml_data_id, content_type, xml_data
  FROM XML_Data_XMLType
/

XML_DATA_ID CONTENT_TYPE         XML_DATA
----------- -------------------- --------------------------------------------------------------------
         69 OTN RSS Feed         <?xml version="1.0" encoding="UTF-8"?>
                                 <rss version="2.0"><channel><title>Oracle Technology Network Headli
                                 nes</title><link>http://www.oracle.com/technology</link><descriptio
                                 n>New services and resources to help developers, DBAs, and architec
                                 ts build, deploy, manage, and optimize applications using Oracle pr
                                 oducts and industry-standard technologies. </descrip
```

Using XML Data Stored in Oracle

- **Now that you have the XML in Oracle Tables what do you do with it?**
  - Pull the XMLType or XML CLOB into PL/SQL and process it with the methods already discussed
  - Query the XML Data directly with SQL
- **Next, some examples of SQL access to XML in Oracle tables**
  - Not exactly PL/SQL, but remember the Oracle Design Maxim!

Querying XML data with XPATH

- **We talked about XPATH in Part I (shredding)**
- **XPATH queries can be used against XML data in the database**
  - XMLType in the database instead of XMLType variables
- **Uses XMLType.extract() method**
- **Often more elaborate XPATH than simply shredding**
- **Can be used in SELECT as well as WHERE**

Understanding XPATH references

(Review from Part I)

- **XPATH is a syntax for accessing an XML Doc**
- **Works like file system paths**
- **Each node is accessed by node name**
- **Special "text()" syntax to get actual value**
- **Examples:**
  - Top Node: /
  - Node Reference: /inventoryUpdate/customer
  - Text Reference: /inventoryUpdate/customer/custName/text()
  - Attribute: /inventoryUpdate/customer/@attr
  - Predicates: /inventoryUpdate/customer[1]
- **This is just an overview… there is more detail**
  - Look for XPATH tutorials on the internet (try www.w3schools.com/xpath/)

Data for XPATH examples

- Data like the following XML will be used in the following examples
- Each row in the tables will have a different order
- We will query some data from the tables using XPATH

```
<order>
  <ORDER_NUMBER>Z1234-57A</ORDER_NUMBER>
  <ORDER_DATE>2007-01-15</ORDER_DATE>
  <orderLines>
    <orderLine>
      <ITEM_NUMBER>A1122</ITEM_NUMBER>
      <ITEM_QUANTITY>55</ITEM_QUANTITY>
      <ITEM_PRICE>2.65</ITEM_PRICE>
    </orderLine>
    <orderLine>
      <ITEM_NUMBER>A1342</ITEM_NUMBER>
      <ITEM_QUANTITY>32</ITEM_QUANTITY>
      <ITEM_PRICE>2.25</ITEM_PRICE>
    </orderLine>
    <orderLine>
      <ITEM_NUMBER>W1234</ITEM_NUMBER>
      <ITEM_QUANTITY>1927</ITEM_QUANTITY>
      <ITEM_PRICE>13</ITEM_PRICE>
    </orderLine>
    <orderLine>
      <ITEM_NUMBER>W1272</ITEM_NUMBER>
      <ITEM_QUANTITY>521</ITEM_QUANTITY>
      <ITEM_PRICE>8.9</ITEM_PRICE>
    </orderLine>
    <orderLine>
      <ITEM_NUMBER>W3472</ITEM_NUMBER>
      <ITEM_QUANTITY>625</ITEM_QUANTITY>
      <ITEM_PRICE>6.05</ITEM_PRICE>
    </orderLine>
  </orderLines>
</order>
```

Simple XPATH example

- **Use the *extract()* method with an XPATH to extract discrete values from a CLOB column**

2. XPATH query text passed in to *extract()*

4. getStringVal() is needed to convert the output to a VARCHAR2

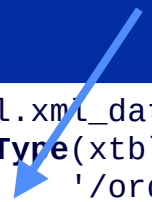
```
SELECT xtbl.xml_data_id
      ,XMLType(xtbl.xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal() order_number
  FROM xml_data_clob xtbl
/
```

1. XMLType() parses the CLOB into an XMLType object

3. Need the "text()" path to extract the actual value

Multiple extracts() at once

TO_DATE used to convert value to a DATE

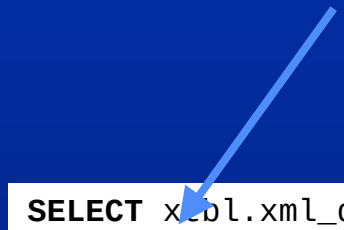


```
SELECT xtbl.xml_data_id
      ,XMLType(xtbl.xml_data).extract(
              '/order/ORDER_NUMBER/text()').getStringVal() order_number
      ,TO_DATE(XMLType(xtbl.xml_data).extract(
              '/order/ORDER_DATE/text()').getStringVal(),'YYYY-MM-DD') order_date
  FROM xml_data_clob xtbl
/

XML_DATA_ID ORDER_NUMBER         ORDER_DATE
----------- -------------------- -----------
        258 H0701262             1/26/2007
        259 H0701263             1/26/2007
        260 H0701264             1/26/2007
        261 H0701265             1/26/2007
        262 H0701266             1/26/2007
        263 H0701267             1/26/2007
        264 H0701271             1/27/2007
        265 H0701272             1/27/2007
        266 H0701273             1/27/2007
        267 H0701274             1/27/2007
        268 H0701275             1/27/2007
        269 H0701276             1/27/2007
```

XPATH against XMLType Column

- **The XMLType parsing is not needed**



```
SELECT xtbl.xml_data_id
      ,xtbl.xml_data.extract(
             '/order/ORDER_NUMBER/text()').getStringVal() order_number
      ,TO_DATE(xtbl.xml_data.extract(
             '/order/ORDER_DATE/text()').getStringVal(),'YYYY-MM-DD') order_date
  FROM xml_data_XMLType xtbl
/
```

XPATH against XMLType Table

- There is an internal column name for the Object Type Table: SYS_ NC_ROWINFO$

```
SELECT tbl.sys_nc_rowinfo$.extract(
               '/order/ORDER_NUMBER/text()').getStringVal() order_number
     ,TO_DATE(tbl.sys_nc_rowinfo$.extract(
               '/order/ORDER_DATE/text()'),'YYYY-MM-DD') order_date
  FROM xml_data_xmltype_table tbl
/
```

- Note: The query would be the same for XMLTypes stored in XDB

More complex XPATH examples

The [#] syntax can be used to get row values

```
SELECT xtbl.xml_data.extract('/order/ORDER_NUMBER/text()').getStringVal() order_number
      ,xtbl.xml_data.extract('/order/orderLines/orderLine[1]/ITEM_NUMBER/text()').getStringVal() item_number_1
      ,xtbl.xml_data.extract('/order/orderLines/orderLine[2]/ITEM_NUMBER/text()').getStringVal() item_number_2
      ,xtbl.xml_data.extract('/order/orderLines/orderLine[3]/ITEM_NUMBER/text()').getStringVal() item_number_3
      ,xtbl.xml_data.extract('/order/orderLines/orderLine[4]/ITEM_NUMBER/text()').getStringVal() item_number_4
  FROM xml_data_xmltype xtbl

ORDER_NUMBER ITEM_NUMBER_1 ITEM_NUMBER_2 ITEM_NUMBER_3 ITEM_NUMBER_4
------------ ------------- ------------- ------------- -------------
H0701261     W1257         W1272         W3472
H0701262     A1122         W3434         W3472
H0701263     W1272         W3472
H0701264     A1342         W1272         W3457         W3472
H0701265     A1342         W3457
H0701266     W3434         W3457
H0701267     W1257         W3457         W3472
H0701271     A1122         W1234         W1272
H0701272     A1342         W3457         W3472
H0701273     A1122         W1234         W1257         W1272
H0701274     W1257         W3472
H0701275     W1234         W1272
H0701276     A1122         A1342
```

More complex XPATH examples

extract() can be used in predicates

```
SELECT xtbl.xml_data.extract(
                '/order/ORDER_NUMBER/text()').getStringVal() order_number
              ,TO_DATE(xtbl.xml_data.extract(
                '/order/ORDER_DATE/text()').getStringVal(),'YYYY-MM-DD') order_date
  FROM xml_data_xmltype xtbl
 WHERE xtbl.xml_data.extract('/order/ORDER_NUMBER/text()').getStringVal() = 'H0701035'


ORDER_NUMBER ORDER_DATE
------------ -----------
H0701035     1/3/2007
```

```
SELECT xtbl.xml_data.extract(
                '/order/ORDER_NUMBER/text()').getStringVal() order_number
              ,TO_DATE(xtbl.xml_data.extract(
                '/order/ORDER_DATE/text()').getStringVal(),'YYYY-MM-DD') order_date
  FROM xml_data_xmltype xtbl
 WHERE existsNode(xtbl.xml_data, '/order[ORDER_NUMBER="H0701262"]') > 0
```

More complex XPATH examples

Other XPATH syntax can be used creatively

```
SELECT xtyp.xml_data.extract('/order/ORDER_NUMBER/text()').getStringVal() order_number
      ,xtyp.xml_data.extract('/order/orderLines/orderLine[ITEM_PRICE<3]') order_lines
  FROM xml_data_xmltype xtyp
 WHERE existsNode(xtyp.xml_data,'/order/orderLines/orderLine[ITEM_PRICE<3]') > 0

ORDER_NUMBER    ORDER_LINES
--------------- ------------------------------------------
H0701084        <orderLine>
                  <ITEM_NUMBER>A1122</ITEM_NUMBER>
                  <ITEM_QUANTITY>511</ITEM_QUANTITY>
                  <ITEM_PRICE>2.65</ITEM_PRICE>
                </orderLine>

H0701085        <orderLine>
                  <ITEM_NUMBER>A1122</ITEM_NUMBER>
                  <ITEM_QUANTITY>990</ITEM_QUANTITY>
                  <ITEM_PRICE>2.65</ITEM_PRICE>
                </orderLine>
                <orderLine>
                  <ITEM_NUMBER>A1342</ITEM_NUMBER>
                  <ITEM_QUANTITY>503</ITEM_QUANTITY>
                  <ITEM_PRICE>2.25</ITEM_PRICE>
                </orderLine>
```

XPATH can be used to return XML Fragments

More complex XPATH examples

```
select t.sys_nc_rowinfo$.getClobVal()
      ,t.sys_nc_rowinfo$.extract('/customers/customer/customerNumber/text()').getStringval() customer_number
      ,t.sys_nc_rowinfo$.extract('/customers/customer/customerName/text()').getStringval() customer_name
  from customer_xdb_table t
 WHERE existsNode(t.sys_nc_rowinfo$, '//*[contains(.,"Space")]') > 0
```

XPATH operators, wildcards, etc.

XPATH Queries and Namespaces

- **If your XML contains namespace references, *extract()* will not work without an additional parameter**

```
SELECT xtbl.xml_data_id
      ,xtbl.xml_data.extract(
             '/xs:order/xs:ORDER_NUMBER/text()'
             ,'xmlns:xs="http://www.cda-llc.com/test.xsd'
             ).getStringVal() order_number
      ,TO_DATE(xtbl.xml_data.extract(
             '/xs:order/xs:ORDER_DATE/text()'
             ,'xmlns:xs="http://www.cda-llc.com/test.xsd'
             ).getStringVal(),'YYYY-MM-DD') order_date
  FROM xml_data_XMLType xtbl
/
```

A Shortcut: XMLTable()

- **XMLTable is an Oracle Type that can be used to map XML to a "table" structure**
  - Best for single table XML
  - But can be used for more complex XML

```
ELECT xtyp.xml_data_id, ord.order_number, ord.order_date, ord.item_number_1, ord.item_number_2
  FROM xml_data_xmltype xtyp
      ,XMLTABLE('/order'
         PASSING xtyp.xml_data
         COLUMNS
             order_number  VARCHAR2(20) PATH '/order/ORDER_NUMBER'
            ,order_date    DATE         PATH '/order/ORDER_DATE'
            ,item_number_1 VARCHAR2(20) PATH '/order/orderLines/orderLine[1]/ITEM_NUMBER'
            ,item_number_2 VARCHAR2(20) PATH '/order/orderLines/orderLine[2]/ITEM_NUMBER'
            ,item_number_3 VARCHAR2(20) PATH '/order/orderLines/orderLine[3]/ITEM_NUMBER'
            ,item_number_4 VARCHAR2(20) PATH '/order/orderLines/orderLine[4]/ITEM_NUMBER'
      ) ord
WHERE ord.order_number like 'H0701%';


XML_DATA_ID ORDER_NUMBER         ORDER_DATE  ITEM_NUMBER_1        ITEM_NUMBER_2
----------- -------------------- ----------- -------------------- --------------------
        259 H0701263             1/26/2007   W1272                W3472
        260 H0701264             1/26/2007   A1342                W1272
        261 H0701265             1/26/2007   A1342                W3457
        262 H0701266             1/26/2007   W3434                W3457
        263 H0701267             1/26/2007   W1257                W3457
        264 H0701271             1/27/2007   A1122                W1234
        265 H0701272             1/27/2007   A1342                W3457
```

XPATH Performance Comparison

```
Example Query:

SELECT xtbl.xml_data.extract(
                 '/order/ORDER_NUMBER/text()').getStringVal() INTO v_orderNumber
    FROM xml_data_xmltype xtbl
   WHERE existsNode(xtbl.xml_data, '/order[ORDER_NUMBER="H0701262"]') > 0;


Rows in table: 7467


NO PERFORMANCE INDEXES


CLOB Column XPATH access - elapsed time:            08.1720 Sec
.
XMLType Column XPATH access - elapsed time          02.2130 Sec
.
XMLType Table XPATH access - elapsed time:          02.7440 Sec
.
XMLType Schema Storage XPATH access - elapsed time: 00.6710 Sec
```

Improving performance of XPATH

- **XPATH Queries can be slow**
  - Especially against large data sets
- **Performance can be improved using indexes**
  - B*tree or Bitmap indexes
  - Function based indexes
- **Full Text Search with Oracle Text**
  - Create a context index on the XMLType or CLOB column
  - Limitations:
    - Searches entire XML, not specific element
    - Index needs to be kept synchronized

Schema Based Storage: B*Tree Index

```
Example Query:

SELECT xtbl.xml_data.extract(
                 '/order/ORDER_NUMBER/text()').getStringVal() INTO v_orderNumber
    FROM order_xdb_table xtbl
   WHERE existsNode(xtbl.xml_data, '/order[ORDER_NUMBER="H0701262"]') > 0;


Rows in table: 7467

Before Index:

XMLType Schema Storage XPATH access - elapsed time: 00.6710 Sec


CREATE INDEX order_number_idx ON order_xdb_table.xmldata."ORDER_NUMBER")


After Index:

XMLType Schema Storage XPATH access - elapsed time: 00.0400 Sec
```

Function Based Index on CLOB

```
Example Query:

 SELECT XMLType(xtbl.xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal() INTO v_orderNumber
    FROM xml_data_clob xtbl
   WHERE XMLType(xtbl.xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal() = 'H0701262';


Rows in table: 7467


Before Index:

CLOB Column XPATH access - elapsed time: 08.6130 Sec



CREATE INDEX xml_data_clob_order
    ON xml_data_clob(XMLType(xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal());


After Index:

CLOB Column XPATH access - elapsed time: 00.0600 Sec
```

Function Based Index on XMLType

Example Query:

```
 SELECT XMLType(xtbl.xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal() INTO v_orderNumber
    FROM xml_data_clob xtbl
   WHERE XMLType(xtbl.xml_data).extract(
                '/order/ORDER_NUMBER/text()').getStringVal() = 'H0701262';
```

Rows in table: 7467

Before Index:

XMLType Column XPATH access - elapsed time: 02.2130 Sec

```
CREATE INDEX xml_data_xmltype_order
    ON xml_data_xmltype(xml_data.extract(
                '/order/ORDER_NUMBER/text()').getStringVal());
```

After Index:

XMLType Column XPATH access - elapsed time: 00.0400 Sec

Using XML Schemas in PL/SQL

- **What is an XML Schema?**
  - A special XML document that defines structural and validation rules for an XML document
- **How are XML Schemas used?**
  - Used to automatically validate the content of XML
  - XDB uses XML schemas to create automatically shredded XML storage.
- **How does this work in Oracle**
  - You have to register the schema
  - Then can be used for validation

Generating a Default Schema

- **How do you create an XML Schema?**
  - Use an XML tool like XML Spy
  - Create one manually in a text editor
  - Use a tool to generate a default schema from an existing XML file.
- **Oracle has a method to Generate a default schema**
  - Uses DBMS_XMLQUERY
    - PL/SQL Version of XSU… Similar to XMLGEN
  - Based on a SQL Statement (like XMLGEN)
  - Uses Oracle dictionary to determine datatype, optionality, etc.

Generating Default Schema Example

Pass in query, column aliases will become node names

By default, uses ROWSET and ROW tags. Can be overridden just like XMLGEN

DBMS_XMLQUERY.schema is a constant that tells getXML to generate a schema

These last few steps remove some extraneous content and allow the resultant schema to be registered in Oracle

```
DECLARE
   v_Context DBMS_XMLQUERY.ctxHandle;
   v_XSD     CLOB;
   x_Doc     XMLType;
   x_XSD     XMLType;
BEGIN
   v_Context := DBMS_XMLQUERY.newContext(
                        'SELECT customer_id AS "customerID"
                               ,customer_number AS "customerNumber"
                               ,customer_name   AS "customerName"
                          FROM CUSTOMER');


   DBMS_XMLQUERY.setRowsetTag(v_Context, 'customers');
   DBMS_XMLQUERY.setRowTag(v_Context, 'customer');




   v_XSD := DBMS_XMLQUERY.getXML(v_Context, DBMS_XMLQUERY.SCHEMA);
   DBMS_OUTPUT.PUT_LINE('Before cleanup:');
   DBMS_OUTPUT.PUT_LINE(v_XSD);

   x_Doc := XMLType(v_XSD);
   x_XSD := x_Doc.extract('/DOCUMENT/xsd:schema'
               , 'xmlns:xsd="http://www.w3.org/2001/XMLSchema"');
   DBMS_OUTPUT.PUT_LINE('After cleanup: ');
   DBMS_OUTPUT.PUT_LINE(x_XSD.getClobVal());

END;
```

Example Generated Schema

Extraneous content

```
Before cleanup:
<?xml version = '1.0'?>
<DOCUMENT xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="customers">
         <xsd:complexType>
            <xsd:sequence>
               <xsd:element name="customer" minOccurs="0" maxOccurs="unbounded">
                  <xsd:complexType>
                     <xsd:sequence>
                        <xsd:element name="customerID" type="xsd:long" minOccurs="0"/>
                        <xsd:element name="customerNumber" nillable="true" minOccurs="0">
                           <xsd:simpleType>
                              <xsd:restriction base="xsd:string">
                                 <xsd:maxLength value="20"/>
                              </xsd:restriction>
                           </xsd:simpleType>
                        </xsd:element>
                        <xsd:element name="customerName" nillable="true" minOccurs="0">
                           <xsd:simpleType>
                              <xsd:restriction base="xsd:string">
                                 <xsd:maxLength value="40"/>
                              </xsd:restriction>
                           </xsd:simpleType>
                        </xsd:element>
                     </xsd:sequence>
                     <xsd:attribute name="num" type="xsd:integer"/>
                  </xsd:complexType>
               </xsd:element>
            </xsd:sequence>
         </xsd:complexType>
      </xsd:element>
   </xsd:schema>
   <customers xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="#/DOCUMENT/xsd:schema[not(@targetNamespace)]">
      <customer num="1">
…
```

Example Generated Schema

Extraneous content is now removed

```
After cleanup:
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="customers">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="customer" minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="customerID" type="xsd:long" minOccurs="0"/>
              <xsd:element name="customerNumber" nillable="true" minOccurs="0">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="20"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>
              <xsd:element name="customerName" nillable="true" minOccurs="0">
                <xsd:simpleType>
                  <xsd:restriction base="xsd:string">
                    <xsd:maxLength value="40"/>
                  </xsd:restriction>
                </xsd:simpleType>
              </xsd:element>
            </xsd:sequence>
            <xsd:attribute name="num" type="xsd:integer"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Registering XML Schema to Oracle

- **XML Schemas have to be registered to Oracle to be used**
- **Uses the DBMS_XMLSCHEMA package**
- **Already covered registering schema**
  - Needed for XDB XML Schema Storage

Example Registering Schema

```
DECLARE
   x_XML XMLType;
BEGIN
 x_XML := XMLType(bfilename('XML_DIR'
                            ,'customerReport.xsd')
                            ,nls_charset_id('AL32UTF8') );
   DBMS_XMLSchema.registerSchema(
                 schemaURL => 'http://xmlns.cda-llc.com/customerReport.xsd'
                ,schemaDoc => x_XML
                );
END;
/

SELECT owner, schema_url, local
  FROM all_xml_schemas
/

OWNER           SCHEMA_URL                                        LOCAL
--------------- ------------------------------------------------- -----
XMLEX           http://xmlns.cda-llc.com/customerReport.xsd       YES
XMLEX           http://www.cda-llc.com/orderMessage.xsd           YES
XDB             http://xmlns.oracle.com/xdb/dav.xsd               NO
XDB             http://xmlns.oracle.com/xdb/XDBStandard.xsd       NO
```

Schema Evolution

- **Schemas change over time**
  - New requirements
  - New elements, etc.
- **Impact to various storage methods:**
  - Shredded into relational tables
    - Medium Impact
    - You will have to change your tables and the shredding code
  - Stored as CLOBs or XMLType columns
    - Low impact, just store both types
    - Maybe have a version column in the table.
  - Stored as XML Schema Objects
    - Need to update the registered schema

Conclusion

- **There are a lot of tools you can use to get XML data into and out of Oracle**
- **They each have their own strengths and weaknesses**
- **This presentation only introduces them….**
  - Each has more capabilities not shown here
  - Very little error handling shown here
  - Develop your own XML toolkit!

Papers Available for Download at:

Questions?


ODTUG 2007
KALEIDOSCOPE
WOW! Wide Open World,
Wide Open Web!
JUNE 18 - 21, 2007
Preconference Hands-On Training JUNE 16 - 17
Hilton Daytona Beach Oceanfront Resort
Daytona Beach, Florida
FEATURING
Oracle Fusion Symposium all day June 18
"Seriously Practical" Application Express Training June 18 and 19