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

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: (This 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: (Next 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

Plethora of PL/SQL XML Tools

- **XDK – XML Developers Kit**
  - XMLPARSER, XMLDOM, XSLPROCESSOR
- **XSU – XML SQL Utility**
  - DBMS_XMLSAVE, DBMS_XMLQUERY
- **XMLType – Special XML Oracle Object**
  - .extract(), .extractValue(), .exsistsNode(), etc.
- **C-Based Packages**
  - DBMS_XMLGEN, SYS_XMLGEN, SYS_SMLAGG, DBMS_XMLDOM, DBMS_XSLPROCESSOR, DBMS_XMLPARSER, DBMS_XMLTRANSFORM, etc.
- **XML/SQL – Open standard SQL extension**
  - XMLElement, XMLAgg, XMLForest
- **XML DB – XML Database**
  - Oracle's Marketing label for all of this!

History of XML in Oracle

- **Oracle 8i**
  - XML Support started with Oracle 8i
    - XDK – XML Developers Kit
  - Java based code, Not installed by default
    - You had to install JVM, load Java XDK, load XDK PL/SQL
- **Oracle 9i Release 1**
  - First release of XMLType
  - C-based packages (can process XML w/o JVM)
    - DBMS_XMLGEN, SYS_XMLGEN, SYS_SMLAGG
    - Does not scale well because XMLTypes stored as CLOBs

History of XML in Oracle

- **Oracle 9i Release 2**
  - XML DB concept introduced
  - Schema-based storage for XMLType
    - Allows for faster access than CLOB-based storage
  - Other performance improvements
  - New XMLType member functions
    - .transform()
  - SQL/XML support
    - XMLElement, XMLAgg, XMLForest
  - Other Oracle SQL extensions
    - UPDATEXML(), XMLSEQUENCE()
  - C-based XML PL/SQL packages (don't need JVM):
    - DBMS_XMLDOM, DBMS_XSLPROCESSOR, DBMS_XMLPARSER, DBMS_XMLTRANSFORM, etc.

History of XML in Oracle

- **Oracle 10g Release 1**
  - XML Schema improvements
    - Schema evolution
  - New C-based PL/SQL package:
    - DBMS_XMLSTORE
  - Java based packages depreciated
    - XMLPARSER, XMLDOM, XSLPROCESSOR
  - While not depreciated, should be avoided
    - DBMS_XMLQUERY, DBMS_XMLSAVE
    - Use DBMS_XMLGEN & DBMS_XMLSTORE instead

Where do I start?

- **Why are there so many different tools?**
  - Different tools released with 8i, 9i, 10g, etc.
    - Older methods remain for backwards compatibility
  - Java based, C-Based, New standards
  - Different XML storage methods
- **Which ones should I use?**
  - What performance do you need?
  - Are you storing XML in the database?
  - Do you want to install and use the JVM?
  - Which version of the database?

Where do I get documentation?

- **How do I find out about these tools?**
- **Java-Java-Java**
  - Every time I look for help on the Oracle website it seems that all I see is tools for Java developers
  - Even the XML DB documentation is mostly aimed at Java developers
  - XML usage *is* logically tied with Java development
- **There *IS* PL/SQL XML help in the Oracle Docs**
  - SQL Reference→Functions →XMLElement & etc.
  - PL/SQL Packages and Types Reference→DBMS_XML*
  - PL/SQL Packages and Types Reference→XMLType
  - XML DB Developer's Guide→PL/SQL API for XMLType
  - XML Developers Kit Programmers Guide→Using XSU
  - 9i Docs : XML Developer's Kits Guide XDK→XDK for PL/SQL

Where do I get documentation?

- **There IS help on Oracle Website**
  - Oracle XML Technology Center

    http://www.oracle.com/technology/tech/xml/index.html
  - XML DB How-To Documents

    http://www.oracle.com/technology/sample_code/tech/java/codesnippet/xmldb/index.html
  - PL/SQL XML Programming Forum

    http://forums.oracle.com/forums/forum.jspa?forumID=157&start=0
- **Useful XML Book:**
  - XML & SQL: Design, Build & Manage XML Applications in Java, C, C++, PL/SQL
    - Around 30% of the book is relevant to PL/SQL & SQL XML

XML is Great…In Its Place!

- **XML is a great format for mediating business events**
  - Great for messaging… AQ, MQ, etc.
  - Output of web services, RSS feeds, etc.
- **XML Can also be useful for limited data transfer**
  - Not efficient for large amounts of data
- **XML is also useful for:**
  - Metadata – Data that controls how an application works
  - Non-queried content for the web, especially when used with XSL
- **XML is *NOT* a good format for data access!**
  - If you need to access the data, store it in relational tables!
  - XML is not really useful as a database (in spite of what many seem to think!)
- **Therefore:**
  - This presentation spends much more time on getting XML data into and out of Oracle tables

XML is Great…In Its Place!

- **XML is *NOT* a good format for data access!**
  - If you need to access the data, store it in relational tables!
  - XML is not really useful as a database (in spite of what many seem to think!)
- **XML Should really not be used for:**
  - Transactional data
  - Highly queried data
  - Large amounts of data
- **Therefore:**
  - This presentation spends much more time on getting XML data into and out of Oracle tables than on storing XML

Different XML techniques in Oracle

- **There are many different ways to manipulate XML from within Oracle**
- **PL/SQL (and SQL) Methods for Producing XML documents from Normalized Tables:**
  - SQL/XML
  - Oracle extensions to SQL/XML
  - DBMS_XMLGEN
  - DOM Model APIs

Different XML techniques in Oracle

- **PL/SQL Methods for storing XML data into Normalized Oracle Tables:**
  - XPATH access to XMLType
  - DBMS_XMLPARSER
  - DOM Model API (DBMS_XMLDOM)
  - DBMS_XMLSTORE (inverse of DBMS_XMLGEN)
- **Also called "shredding" XML data**
- **This is not talking about storing XML as XML in the database…**
  - Go to next presentation for this.

Let's Start with Producing XML

- **We are going to start with techniques to generate XML Documents**
- **This is quite often the first requirement you will get**
- **Important Oracle Design Maxim:**
  - If you can do it in SQL, do it in SQL
  - If you can't do it in SQL, do it in PL/SQL
  - Only use client code (i.e. Java, C, VB) if you can't do it in SQL or PL/SQL
    - Of course there are exceptions!
- **So…We are going to start with SQL/XML**

XML from SQL: SQL/XML

- **There are a set of SQL functions that can be used to build complex XML from SQL**
  - Available in 9i and 10g
  - Based on an open standard SQL extension
  - Supported by Oracle, DB2, SQLServer, others
- **XMLElement: The primary function**
  - Accepts a name and a value and produces an XML "node"
  - Usually Nested with XMLElement calls to produce complex XML
- **Combined with other functions:**
  - XMLAgg – For aggregation of results
  - XMLAttributes – Sets XML tag attributes
  - XMLForest – Converts a series of columns into XML tags
  - Others -

XMLElement - Basics

- **Basic functionality is very simple**

identifier

value expression

```
SQL> SELECT XMLElement("TestElement", 1) as xml_output
       FROM DUAL
```

```
XML_OUTPUT
---------------------------------------------
<TestElement>1</TestElement>
```

XMLElement – Quoting identifier

- **Why use double quotes (“”)?**
  - **The *identifier* is *not* a parameter value**
  - **In the SQL it is more like a “column name”**
  - **The quotes can be left off, for example:**

```
SQL> SELECT XMLElement(TestElement, 1) as xml_output
       FROM DUAL
```

```
XML_OUTPUT
---------------------------------------------
<TESTELEMENT>1</TESTELEMENT>
```

Without Quotes, tags are always UPPERCASE!

XMLElement – Quoting identifier

- **It's best to always use double quotes**
  - **Most XML uses "CamelCase" tags**
- **The *value* should use single quotes**
  - **The *value* *is* like a parameter**

```
SQL> SELECT XMLElement("camelCaseTag", 'Some Value') as xml_output
       FROM DUAL
```

```
XML_OUTPUT
-------------------------------------------------------
<camelCaseTag>Some Value</camelCaseTag>
```

XMLElement – Column as value

- The *value expression* is usually a column from a table in the SQL query:

```
SQL> SELECT XMLElement("customerName", c.customer_name)
       FROM customer c
      WHERE c.customer_id = 1
```

```
XML_OUTPUT
--------------------------------------------------------
<customerName>Spacely Sprockets</customerName>
```

XMLElement - Chaining

- To be useful, XMLElement calls are “chained” together in a hierarchy.
  - In this case a parent “customer” tag is added:

```
SQL> SELECT XMLElement("customer"
           ,XMLElement("customerName", c.customer_name)
           ) as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
-------------------------------------------------------------------
<customer><customerName>Spacely Sprockets</customerName></customer>
```

XMLElement – Multiple elements

- **Multiple elements can be added at any level.**
  - **In this case “customerNumber” has been added:**

```
SQL> SELECT XMLElement("customer"
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
          ) as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
-------------------------------------------------------------------
<customer><customerName>Spacely
Sprockets</customerName><customerNumber>A12345</customerNumber></custome
r>
```

This format is a mess though!

XMLElement – Nice Format

- **Nicer format output can be displayed using the "extract" function of XMLType output**
  - How does this work?

```
SQL> SELECT XMLElement("customer"
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
          ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
-------------------------------------------------------------------
<customer>
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
```

XMLElement – Returns xmlType

- **The XMLElement functions returns an Oracle Type of xmlType**
  - SQL*Plus automatically converts output to CLOB for display

```
SQL> CREATE OR REPLACE VIEW testxmlview AS
SELECT XMLElement("customer"
          ,XMLElement("customerName", c.customer_name)
          ) as xml_output
  FROM customer c
 WHERE c.customer_id = 1
/
SQL> describe testxmlview

 Name                                      Null?    Type
 ----------------------------------------- -------- -----------------
 XML_OUTPUT                                         SYS.XMLTYPE
```

XMLElement – XMLType Methods

- **Since output is XMLType objects, the standard XMLType methods are available:**
  - extract() – Extracts a sub-portion of the XMLType as an XMLType.
    - One parameter: Xpath expression. '/' returns the whole thing
    - Useful because it formats the output nicely
  - getClobVal() – Returns CLOB instead of XMLType
    - Useful if calling method cannot use XMLType
  - getStringVal() – Returns VARCHAR2(4000)
    - Useful if calling method cannot use XMLType or CLOB
    - However…. The entire XML has to be under 4000 characters!

XMLElement - Attributes

- **Add XMLAttributes call within XMLElement**
  - Put the call *before* children elements
  - By default, the attribute name will be the same as the column name

```
SELECT XMLElement("customer"
             ,XMLAttributes(c.customer_id)
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
          ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
-------------------------------------------------------------------
<customer CUSTOMER_ID="1">
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
```

XMLElement - Attributes

- **Use "*AS*" to override attribute name**
  - Use quotes to allow mixed case attribute names

```
SELECT XMLElement("customer"
             ,XMLAttributes(c.customer_id as "id")
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
          ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
-----------------------------------------------------------------
<customer id="1">
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
```

XMLForest – Shortcut

- **XMLForest is a shortcut for multiple XMLElements**
  - Accepts a list of value expressions (usually columns)

```
SELECT XMLElement("customer"
          ,XMLElement("customerID", c.customer_id)
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
      ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

Can be replaced with this

```
SELECT XMLElement("customer"
          ,XMLForest(c.customer_id, c.customer_name, c.customer_number)
      ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

XMLForest – Default tags

- By default XMLForest uses the column names as tags

```
SELECT XMLElement("customer"
          ,XMLForest(c.customer_id, c.customer_name, c.customer_number)
      ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
----------------------------------------------------------------
<customer>
  <CUSTOMER_ID>1</CUSTOMER_ID>
  <CUSTOMER_NAME>Spacely Sprockets</CUSTOMER_NAME>
  <CUSTOMER_NUMBER>A12345</CUSTOMER_NUMBER>
</customer>
```

XMLForest – Tag override

- **Tag names can be specified with AS clauses**

```
SELECT XMLElement("customer"
          ,XMLForest(c.customer_id     AS "customerID"
                    ,c.customer_name   AS "customerName"
                    ,c.customer_number AS "customerNumber")
      ).extract('/') as xml_output
  FROM customer c
 WHERE c.customer_id = 1
```

```
XML_OUTPUT
----------------------------------------------------------------
<customer>
  <customerID>1</customerID>
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
```

XMLElement – Multiple Rows

- **What happens if there are multiple rows?**
  - The output *also* returns multiple rows.

```
SELECT rownum
      ,XMLElement("customer"
             ,XMLAttributes(c.customer_id as "id")
          ,XMLElement("customerName", c.customer_name)
          ,XMLElement("customerNumber", c.customer_number)
          ).extract('.') as xml_output
  FROM customer c
```

```
ROWNUM XML_OUTPUT
------- ------------------------------------------------------------
      1 <customer id="1">
          <customerName>Spacely Sprockets</customerName>
          <customerNumber>A12345</customerNumber>
        </customer>

      2 <customer id="2">
          <customerName>Cogswells Cogs</customerName>
          <customerNumber>A2765</customerNumber>
        </customer>
```

XMLElement – Multiple Rows

- **Are multiple rows OK?**
  - It depends on what you are doing….
  - OK of each row will return a separate XML document
  - But… If the entire query should return a single XML document we have a problem….
- **An XML document should be "Well-Formed"**
  - Every tag needs an end tag
  - Tags should be nested correctly
  - And….A single root tag in entire document

XMLElement – Combining Rows

- OK, so let's add a "root" tag

```
SELECT XMLElement("customerReport"
         ,XMLElement("customer", XMLAttributes(c.customer_id AS "id")
             ,XMLElement("customerName", c.customer_name)
             ,XMLElement("customerNumber", c.customer_number)
         )
       ).extract('.') as xml_output
  FROM customer c
```

This did not work! There are two "customerReport" tags

```
XML_OUTPUT
--------------------------------------------------
<customerReport>
  <customer id="1">
    <customerName>Spacely Sprockets</customerName>
    <customerNumber>A12345</customerNumber>
  </customer>
</customerReport>

<customerReport>
  <customer id="2">
    <customerName>Cogswells Cogs</customerName>
    <customerNumber>A2765</customerNumber>
  </customer>
</customerReport>
```

XMLAgg – "GROUP BY" for XML

- **XMLAgg()**
  - A SQL/XML function that groups multiple rows within a single tag
  - Kind of like a "Group by" for SQL/XML
  - Can work with GROUP BY
  - Takes XMLType output of XMLElement as input
  - Can be used multiple times in a SELECT using in-line sub-queries (example later)

XMLAgg – Combining Rows

- **Add an XMLAgg call to the list:**

```
SELECT XMLElement("customerReport"
          ,XMLAgg(
             XMLElement("customer", XMLAttributes(c.customer_id AS "id")
                ,XMLElement("customerName", c.customer_name)
                ,XMLElement("customerNumber", c.customer_number)
             )
          )
      ).extract('/') as xml_output
  FROM customer c
```

Now there is only ONE "customerReport" tag

```
XML_OUTPUT
-------------------------------------------------
<customerReport>
  <customer id="1">
    <customerName>Spacely Sprockets</customerName>
    <customerNumber>A12345</customerNumber>
  </customer>
  <customer id="2">
    <customerName>Cogswells Cogs</customerName>
    <customerNumber>A2765</customerNumber>
  </customer>
</customerReport>
```

XMLAgg – Using with GROUP BY

- **Add a GROUP BY to the query…**
  - Now the XMLAgg works with the group by to aggregate each group.

```
SELECT XMLElement("customer"
          ,XMLForest(c.customer_id AS "id"
                    ,c.customer_name as "customerName"
                    ,c.customer_number AS "customerNumber")
          ,XMLElement("orders"
             ,XMLAgg(
                XMLElement("order"
                   ,XMLForest(h.order_number as "orderNumber"
                   ,h.order_date AS "orderDate")
                )
             )
          )
      ).extract('/') as xml_output
  FROM customer c, order_header h
 WHERE c.customer_id = h.customer_id
 GROUP BY c.customer_id, c.customer_name, c.customer_number
```

GROUP BY Columns *before* XMLAgg

XMLAgg – Using with GROUP BY

```
XML_OUTPUT
--------------------------------------------------
<customer>
  <id>1</id>
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
  <orders>
    <order>
      <orderNumber>Z1234-57A</orderNumber>
      <orderDate>2007-01-15</orderDate>
    </order>
    <order>
      <orderNumber>Z5299-23C</orderNumber>
      <orderDate>2007-02-27</orderDate>
    </order>
  </orders>
</customer>

<customer>
  <id>2</id>
  <customerName>Cogswells Cogs</customerName>
  <customerNumber>A2765</customerNumber>
  <orders>
    <order>
      <orderNumber>Y1299-14C</orderNumber>
      <orderDate>2007-02-27</orderDate>
    </order>
  </orders>
</customer>

2 rows selected.
```

There is one row per group

Columns from Detail Records specified within XMLAgg

XMLAgg – Multiple Aggregations

- But what if I want more than one aggregation?
- For instance you need an order report with Order header and order lines…

```
break on order_number on order_date skip 1
SELECT h.order_number, h.order_date, i.item_number, l.item_quantity), l.item_price)
  FROM order_line l, order_header h, item i
 WHERE l.order_id = h.order_id  AND l.item_id = i.item_id
 ORDER BY h.order_date, h.order_number, i.item_number
/

ORDER_NUMBER    ORDER_DATE ITEM_NUMBER     ITEM_QUANTITY ITEM_PRICE
--------------- ---------- --------------- ------------- ----------
Z1234-57A       15-JAN-07  A1122                      55       2.65
                           A1342                      32       2.25
                           W1234                    1927         13
                           W1272                     521        8.9
                           W3472                     625       6.05
Y1299-14C       27-FEB-07  A1122                      92       2.65
                           A1342                      23       2.25
                           W1234                    1627         13
                           W1272                     123        8.9
                           W3472                     205       6.05
```

- You want an outer "<orderMessage>" tag as well as grouping order lines within orders…

XMLAgg – Nesting SELECTs

- **Add two XMLAgg() calls:**
  1. One to group ALL orders within the <orderMessage>

2. Another to group order lines within each order

```
SELECT XMLElement("orderMessage"
          ,XMLAgg(
             XMLElement("order"
                ,XMLForest(h.order_number, h.order_date)
                ,(SELECT XMLElement("orderLines"
                     ,XMLAgg(
                        XMLElement("orderLine"
                           ,XMLForest(i.item_number
                                     ,l.item_quantity
                                     ,l.item_price)
                        ) ) )
                    FROM order_line l, item i
                   WHERE l.order_id = h.order_id
                     AND i.item_id = l.item_id
                 ) ) )
       ).extract('.') xml_output
  FROM order_header h
 ORDER BY h.order_date
```

XMLAgg – Nesting SELECTs

1. <order>s are grouped within <orderMessage> because of 1st XMLAgg()

```
XML_OUTPUT
---------------------------------------------
<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>
  <order>
    <ORDER_NUMBER>Z5299-23C</ORDER_NUMBER>
    <ORDER_DATE>2007-02-27</ORDER_DATE>
    <orderLines>
      <orderLine>
        <ITEM_NUMBER>A1122</ITEM_NUMBER>
        <ITEM_QUANTITY>92</ITEM_QUANTITY>
        <ITEM_PRICE>2.65</ITEM_PRICE>
      </orderLine>

...
```

2. <orderLine>s are grouped within <orders> because of 2nd XMLAgg()

Dealing with Optional Data

- **XMLElements without data still generate tags.**
  - For example, the address lines on the following query

```
SELECT XMLElement("customers"
          ,XMLAgg(
             XMLElement("customer"
                ,XMLElement("customerID", c.customer_id)
                ,XMLElement("customerName", c.customer_name)
                ,(SELECT XMLElement("locations",
                           XMLAgg(
                              XMLElement("location"
                                 ,XMLElement("locationLabel", l.location_label)
                                 ,XMLElement("locationAddress1", l.address_line_1)
                                 ,XMLElement("locationAddress2", l.address_line_2)
                                 ,XMLElement("locationAddress3", l.address_line_3)
                                 )
                             )
                         )
                   FROM customer_location l
                  WHERE c.customer_id = l.customer_id
                )
             )
          )
       ).extract('/')
  FROM customer c
 WHERE customer_id = 1
 ORDER BY c.customer_id
```

Dealing with Optional Data

```
XML_OUTPUT
---------------------------------------------
<customers>
  <customer>
    <customerID>1</customerID>
    <customerName>Spacely Sprockets</customerName>
    <locations>
      <location>
        <locationLabel>LOUISVILLE-A</locationLabel>
        <locationAddress1>Building 534C</locationAddress1>
        <locationAddress2>1234 Some Industrial Lane</locationAddress2>
        <locationAddress3/>
      </location>
      <location>
        <locationLabel>INDIANAPOLIS</locationLabel>
        <locationAddress1>1234 Another Lane</locationAddress1>
        <locationAddress2/>
        <locationAddress3/>
      </location>
    </locations>
  </customer>
</customers>
```

Address tags for columns without values still appear

- **This might be OK… It depends on what you want**
- **You *can* suppress the tags with CASE statements**

Suppressing Tags for Optional Data

- Use CASE to suppress tags:

```
SELECT XMLElement("customers"
          ,XMLAgg(
             XMLElement("customer"
                ,XMLElement("customerID", c.customer_id)
                ,XMLElement("customerName", c.customer_name)
                ,(SELECT XMLElement("locations",
                           XMLAgg(
                              XMLElement("location"
                                 ,XMLElement("locationLabel", l.location_label)
                                 ,XMLElement("locationAddress1", l.address_line_1)
                                 ,CASE
                                     WHEN l.address_line_2 IS NULL THEN NULL
                                     ELSE XMLElement("locationAddress2", l.address_line_2)
                                   END
                                 ,CASE NVL(l.address_line_3,'NULL')
                                     WHEN 'NULL' THEN NULL
                                     ELSE XMLElement("locationAddress3", l.address_line_3)
                                   END
                                 )
                             )
                         )
                   FROM customer_location l
                  WHERE c.customer_id = l.customer_id
                )
...
```

Suppressing Tags for Optional Data

```
XML_OUTPUT
---------------------------------------------
<customers>
  <customer>
    <customerID>1</customerID>
    <customerName>Spacely Sprockets</customerName>
    <locations>
      <location>
        <locationLabel>LOUISVILLE-A</locationLabel>
        <locationAddress1>Building 534C</locationAddress1>
        <locationAddress2>1234 Some Industrial Lane</locationAddress2>
      </location>
      <location>
        <locationLabel>INDIANAPOLIS</locationLabel>
        <locationAddress1>1234 Another Lane</locationAddress1>
      </location>
    </locations>
  </customer>
</customers>
```

Empty tags disappear!

Suppressing Tags for Optional Data

- You can suppress parent tags also

```
SELECT XMLElement("customers"
          ,XMLAgg(
             XMLElement("customer"
                ,XMLElement("customerID", c.customer_id)
                ,XMLElement("customerName", c.customer_name)
                ,(SELECT XMLElement("locations",
                           XMLAgg(
                              XMLElement("location"
                                 ,XMLElement("locationLabel", l.location_label)
                                 ,CASE WHEN l.address_line_1 IS NULL THEN NULL
                                    ELSE
                                       XMLElement("address"
                                          ,XMLElement("locationAddress1", l.address_line_1)
                                          ,CASE WHEN l.address_line_2 IS NULL THEN NULL
                                           ELSE XMLElement("locationAddress2"
                                                        , l.address_line_2)
                                           END
                                          ,XMLElement("state", l.state)
                                          ,XMLElement("zip", l.postal_code)
                                       )
                                   END
                                 ) ) )
                   FROM customer_location l
                  WHERE c.customer_id = l.customer_id

...
```

Dealing with Optional Data

Entire <address> node is gone!

```
XML_OUTPUT
---------------------------------------------
<customers>
  <customer>
    <customerID>1</customerID>
    <customerName>Spacely Sprockets</customerName>
    <locations>
      <location>
        <locationLabel>LOUISVILLE-A</locationLabel>
        <address>
          <locationAddress1>Building 534C</locationAddress1>
          <locationAddress2>1234 Some Industrial Ln</locationAddress2>
          <state>KY</state>
          <zip>40031</zip>
        </address>
      </location>
      <location>
        <locationLabel>LOUISVILLE-B</locationLabel>
        <address>
          <locationAddress1>Building 600</locationAddress1>
          <locationAddress2>1234 Some Industrial Ln<locationAddress2>
          <state>KY</state>
          <zip>40031</zip>
        </address>
      </location>
      <location>
        <locationLabel>INDIANAPOLIS</locationLabel>
      </location>
    </locations>
  </customer>
</customers>
```

Where's the <?XML?> tag?

- **Most XML requires an <?XML> tag**
  - The standard SQL/XML functions do not provide this.
- **Oracle provides some functions that *do* produce the <?XML> tag:**
  - SYS_XMLGEN() – Similar to XMLElement
  - SYS_XMLAGG() – Similar to XMLAGG()
- **You can use these as the "top" level call to add:**
  - Top <?xml> tag:

```
<?xml version="1.0" ?>
```

  - Other <?xml> tags:

```
<?xml-stylesheet href="customerReport.xsl" type="text/xsl" ?>
<?xml ?>
```

Adding an <?XML?> tag?

- **Oracle provides another function: SYS_XMLGEN**
  - Similar to XMLElement
  - But can add <?XML> tags

```
SELECT SYS_XMLGEN(
          XMLAgg(
             XMLElement("customer", XMLAttributes(c.customer_id AS "id")
                ,XMLElement("customerName", c.customer_name)
                ,XMLElement("customerNumber", c.customer_number)
                )
           )
          ,XMLFormat.createformat('customerReport','NO_SCHEMA', null)
      ) as xml_output
  FROM customer c
```

XMLFormat.createformat used to:
1) Specify tag name
2) Generate various <?XML> tags

Adding an <?XML?> tag?

From SYS_XMLGEN

```
XML_OUTPUT
--------------------------------------------------
<?xml version="1.0"?>
<customerReport>
<customer id="1">
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
<customer id="2">
  <customerName>Cogswells Cogs</customerName>
  <customerNumber>A2765</customerNumber>
</customer>
</customerReport>
```

Comes from inner XMLAgg and XMLElements

Adding an <?XML?> tag?

- **XMLFormat.createformat**
  - Adding additional <?xml> tags

```
SELECT SYS_XMLGEN(
          XMLAgg(
             XMLElement("customer", XMLAttributes(c.customer_id AS "id")
                ,XMLElement("customerName", c.customer_name)
                ,XMLElement("customerNumber", c.customer_number)
                )
           ).extract('/')
          ,XMLFormat.createformat('customerReport','NO_SCHEMA'
               ,null, null, null
               , '<?xml-stylesheet href=htmlRend.xsl" type="text/xsl" ?>')
      ) as xml_output
  FROM customer c
```

Adds an additional tag as specified

Adding an <?XML?> tag?

Additional tag

```
XML_OUTPUT
--------------------------------------------------------
<?xml version="1.0"?>
<?xml-stylesheet href=htmlRend.xsl" type="text/xsl" ?>
<customerReport>
<customer id="1">
  <customerName>Spacely Sprockets</customerName>
  <customerNumber>A12345</customerNumber>
</customer>
<customer id="2">
  <customerName>Cogswells Cogs</customerName>
  <customerNumber>A2765</customerNumber>
</customer>
</customerReport>
```

XMLFormat parameters

- **XMLFormat - Oracle Type**
  - Specifies tag name, and other XML properties
- **XMLFormat.createFormat() method parameters:**
  - enclTag - Specifies tag name
  - schemaType – Specifies XML Schema type
    - NO_SCHEMA, USE_GIVEN_SCHEMA, GEN_SCHEMA_INLINE, GEN_SCHEMA_OUTOFLINE
  - schemaName – Used with USE_GIVEN_SCHEMA
  - targetNameSpace – Used with certain schema types
  - dbURLPrefix – Used with OUTOFLINE schemas
  - processingIns – Varies by type… For NO_SCHEMA can be used to specify additional <?xml> tags
- **XML Schemas out of scope for this presentation**

Oracle Provided SQL/XML

- **Oracle provides additional functions**
  - Not defined in the SQL/XML standards
- **SYS_XMLGEN()**
  - Like XMLElement but with <?XML?> prolog
- **XMLSEQUENCE()**
  - Returns a collection of XMLType s in an array
- **SYS_XMLAGG()**
  - Like XMLAgg, but with <?XML?> prolog
- **XMLCOLATTVAL()**
  - Generates a set of <column> elements from a query
- **Others…**

SQL/XML Limitations

- **Size Limitations (due to XMLType)**
  - XMLType nodes limited to 64K
  - If using getStringVal()
    - XML Size limited to 4K from SQL
    - XML Size limited to 32K from PL/SQL
  - If using getClobVal(), XML Node Size limited to 64K
- **Performance**
  - Mainly an issue of generating the large amount of text inherent in XML
  - Slower than XMLGEN, faster than DOM

Format SQL/XML for maintenance

- **Do you think this would be easy to maintain?**

```
SELECT XMLElement("customers", XMLAgg(XMLElement("customer"
      ,XMLElement("customerID", c.customer_id), XMLElement("customerName",
            c.customer_name), (SELECT XMLElement("locations", XMLAgg(
       XMLElement("location", XMLElement("locationLabel", l.location_label)
      ,XMLElement("locationAddress1", l.address_line_1)
      ,XMLElement("locationAddress2", l.address_line_2)
      ,XMLElement("locationAddress3", l.address_line_3))))
       FROM customer_location l WHERE c.customer_id = l.customer_id)))
   ).extract('/')
  FROM customer c
 WHERE customer_id = 1
 ORDER BY c.customer_id
```

- **It is very easy to get the parentheses and commas out of kilter!**
- **Better to use some sort of indenting**
  - **Like I have been doing in the examples.**

Format SQL/XML for maintenance

```
SELECT XMLElement("customers"
          ,XMLAgg(
             XMLElement("customer"
                ,XMLElement("customerID", c.customer_id)
                ,XMLElement("customerName", c.customer_name)
                ,(SELECT XMLElement("locations",
                            XMLAgg(
                               XMLElement("location"
                                  ,XMLElement("locationLabel", l.location_label)
                                  ,XMLElement("locationAddress1", l.address_line_1)
                                  ,XMLElement("locationAddress2", l.address_line_2)
                                  ,XMLElement("locationAddress3", l.address_line_3)
                               )
                            )
                         )
                   FROM customer_location l
                  WHERE c.customer_id = l.customer_id
                )
             )
          )
       ).extract('/')
  FROM customer c
 WHERE customer_id = 1
 ORDER BY c.customer_id
```

- **I indent every level by 3 characters**
  - Helps keep track of parentheses matching
  - Easy to see which tags produced for each level

Best Practice: Wrap SQL with PL/SQL

- **This is a PL/SQL conference! That's just SQL!**
  - The XML/SQL method is probably the easiest way to generate XML, even in PL/SQL.
- **All generation of XML should be "wrapped" by PL/SQL anyway**
  - No application should just run a complex query like these directly!
  - Encapsulate ALL SQL in Views or PL/SQL.
    - More on XMLType views later

Using SQL/XML from PL/SQL

- **Put SQL into cursor with bind variables**
- **Various designs for various purposes**
  - Function to get single XML message
  - Procedure to output multiple messages
  - Where will output go?
    - More on this later
- **Use standard PL/SQL best practices!**
- **Make the interface flexible**
  - Provide either XMLType or CLOB output
  - Include <?XML?> or exclude it

Function to Return Single Message

```
FUNCTION get_order_message(pi_order_date IN DATE) RETURN xmltype IS
        CURSOR order_message_cur(pc_order_date IN DATE) IS
        SELECT XMLElement("orderMessage"
              ,XMLAgg(
                 XMLElement("order"
                    ,XMLForest(h.order_number, h.order_date)
                    ,(SELECT XMLElement("orderLines"
                               ,XMLAgg(
                                  XMLElement("orderLine"
                                    ,XMLForest(i.item_number
                                        ,SUM(l.item_quantity) as "ITEM_QUANTITY"
                                        ,SUM(l.item_price) as "ITEM_PRICE")
                                  ) ) )
                        FROM order_line l, item i
                       WHERE l.order_id = h.order_id
                         AND i.item_id = l.item_id
                        GROUP BY i.item_Number
                     ) ) )
           ).extract('.') xml_output
      FROM order_header h
     ORDER BY h.order_date;
     x_ordermessage XMLtype;
  BEGIN
     OPEN order_message_cur(pi_order_date);
     FETCH order_message_cur
        INTO x_ordermessage;
     CLOSE order_message_cur;
     RETURN(x_orderMessage);
  END get_order_message;
```

Return Single Message: Examples

```
SELECT xmlexample.get_order_message(TO_DATE('01/17/2007','MM/DD/YYYY'))
  FROM dual

XML_OUTPUT
---------------------------------------------------------------
<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>

• • •
```

```
DECLARE
  x_myMessage XMLType;
BEGIN
  x_myMessage := xmlexample.get_order_message(TRUNC(sysdate));
  -- Now do something with the XMLType.....
END;
```

Procedure to Write XML Files

```
PROCEDURE write_orders(pi_order_date IN DATE) IS
    CURSOR order_xml_cur(pc_order_date IN DATE) IS
       SELECT h.order_id
              ,XMLElement("order"
                ,XMLForest(h.order_number, h.order_date)
                ,(SELECT XMLElement("orderLines"
                            ,XMLAgg(
                               XMLElement("orderLine"
                                 ,XMLForest(i.item_number
                                           ,SUM(l.item_quantity) as "ITEM_QUANTITY"
                                           ,SUM(l.item_price) as "ITEM_PRICE")
                               ) ) )
                    FROM order_line l, item i
                   WHERE l.order_id = h.order_id
                     AND i.item_id = l.item_id
                    GROUP BY i.item_Number
                 )
            ).extract('.') xml_order
     FROM order_header h
    ORDER BY h.order_date;
    v_DomDoc DBMS_XMLDOM.DOMDocument;
    v_FileName VARCHAR2(80);
    v_OutputDir VARCHAR2(80) := 'C:\Docs\Papers\PLSQL_XML\xml_files\';
 BEGIN
   FOR rc_order IN order_xml_cur(pi_order_date) LOOP
      v_DomDoc   := DBMS_XMLDOM.newDOMDocument(rc_order.xml_order);
      v_FileName := v_OutputDir || 'order' || rc_order.order_id || '.xml';
      DBMS_XMLDOM.writeToFile(v_DomDoc, v_FileName);
   END LOOP;
 END;
```

Procedure to Write XML Files

```
BEGIN
  XMLExample.write_orders('17-JAN-07');
END;
```

C:\Docs\Papers\PLSQL_XML\xml_files

File Edit View Favorites Tools Help

Back Search Folders

Folders: Desktop, My Documents, My Computer, SQ003713 (C:), ail, ARCSOFT, Docs

Name	Size	Type	Date Modified
order44.xml	1 KB	XML Document	2/6/2007 10:01 PM
order45.xml	1 KB	XML Document	2/6/2007 10:01 PM
order46.xml	1 KB	XML Document	2/6/2007 10:01 PM

```
<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>
```

Make it more Flexible

- **Version to return CLOB as well as XMLType**

```
FUNCTION get_order_message_clob(pi_order_date IN DATE) RETURN CLOB IS
    x_Message XMLType;
    v_Message CLOB;
 BEGIN
    x_Message := get_order_message(pi_order_date);
    v_Message := x_Message.getClobVal();
    RETURN(v_Message);
 END;
```

Make it more Flexible

- Version to optionally include <?XML?> tag

```
FUNCTION get_order_message(pi_order_date IN DATE
                          ,pi_include_XML IN VARCHAR2 := 'N') RETURN xmltype IS
      CURSOR order_message_cur(pc_order_date IN DATE) IS
         SELECT
                XMLAgg(
                  XMLElement("order"
                     ,XMLForest(h.order_number, h.order_date)
                . . .
                ).extract('.') xml_output
       FROM order_header h
      ORDER BY h.order_date;
      x_orderMsg XMLtype;
   BEGIN
      OPEN order_message_cur(pi_order_date);
      FETCH order_message_cur INTO x_orderMsg;
      CLOSE order_message_cur;

      IF pi_include_XML = 'Y' THEN
         SELECT SYS_XMLGEN(x_orderMsg ) INTO x_orderMsg FROM DUAL;
      ELSE
         SELECT XMLElement("orderMessage", x_orderMsg) INTO x_orderMsg FROM DUAL;
      END IF;
      RETURN(x_orderMsg);
   END get_order_message;
```

SQL/XML Errors

- **Example XMLType conversion error:**

```
ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00225: end-element tag "updateSentTime" does not match start-element tag
"messageSentTime"
Error at line 2
ORA-06512: at "SYS.XMLTYPE", line 254
```

- **You can usually tell the problem from the text of the "LPX" message**
- **For *producing* XML, you usually don't have to do much error handling**
  - Your errors are almost always due to bugs in your SQL

Using DBMS_XMLGEN

- **DBMS_XMLGEN**
  - Generates XML in CLOBs or XMLTypes
  - C – based (better performance)
  - Similar functionality to SQL/XML functions
  - Can be called from PL/SQL or SQL
- **Uses "Canonical" mapping**
  - Maps SQL Query to XML – Easy to use!
  - All rows within a <ROWSET> tag
  - Each row within a <ROW> tag
  - Each column has its own tag – Defaults to column name

Simple Use of DBMS_XMLGEN

```
FUNCTION get_customers RETURN CLOB IS
   v_Context DBMS_XMLGEN.ctxHandle;
   v_xml CLOB;
BEGIN
   v_Context := DBMS_XMLGEN.newContext('SELECT * FROM customer');
   v_xml := DBMS_XMLGEN.getXML(v_Context);
   RETURN(v_xml);
END;
```

```
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <CUSTOMER_ID>1</CUSTOMER_ID>
  <CUSTOMER_NUMBER>A12345</CUSTOMER_NUMBER>
  <CUSTOMER_NAME>Spacely Sprockets</CUSTOMER_NAME>
 </ROW>
 <ROW>
  <CUSTOMER_ID>2</CUSTOMER_ID>
  <CUSTOMER_NUMBER>A2765</CUSTOMER_NUMBER>
  <CUSTOMER_NAME>Cogswells Cogs</CUSTOMER_NAME>
 </ROW>
</ROWSET>
```

DBMS_XMLGEN: Specifying tags

```
FUNCTION get_customers RETURN CLOB IS
   v_Context DBMS_XMLGEN.ctxHandle;
   v_xml CLOB;
BEGIN

   v_Context := DBMS_XMLGEN.newContext(
          'SELECT c.customer_id      AS "customerID"
                 ,c.customer_number AS "customerNumber"
                 ,c.customer_name   AS "customerName"
             FROM Customer c');
   DBMS_XMLGEN.setrowsettag(v_Context, 'customers');
   DBMS_XMLGEN.setRowTag(v_Context, 'customer');
   v_xml := DBMS_XMLGEN.getXML(v_Context);
   RETURN(v_xml);
END;
```

Set column tags using AS

Rename <ROW> and <ROWSET>

```
<?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>
```

DBMS_XMLGEN: Using REF cursor

- **newContext() accepts REF Cursors as well as SQL**

```
FUNCTION query_customers(pi_customer_id IN customer.customer_id%TYPE)
         RETURN SYS_REFCURSOR IS
   v_cust_cur SYS_REFCURSOR;
BEGIN
   OPEN v_cust_cur FOR
      'SELECT * FROM CUSTOMER WHERE customer_id = :ID' USING pi_customer_id;
   RETURN(v_cust_cur);
END;

FUNCTION get_customer(pi_customer_id IN customer.customer_id%TYPE) RETURN CLOB IS
   rc_cust SYS_REFCURSOR;
   v_Context DBMS_XMLGEN.ctxHandle;
   v_xml CLOB;
BEGIN
   rc_cust := query_customers(pi_customer_id);
   v_Context := DBMS_XMLGEN.newcontext(rc_cust);
   v_xml     := DBMS_XMLGEN.getXML(v_context);
   RETURN(v_xml);
END;;
```

DBMS_XMLGEN: Bind Variables

- You can also use Bind variables in SQL input:

```
FUNCTION get_customer(pi_customer_id IN customer.customer_id%TYPE) RETURN CLOB IS
   v_Context DBMS_XMLGEN.ctxHandle;
   v_xml CLOB;
BEGIN
   v_Context := DBMS_XMLGEN.newcontext(
                        'SELECT * FROM Customer WHERE customer_id = :ID');
   DBMS_XMLGEN.setBindValue(v_context, 'ID', pi_customer_id);
   v_xml     := DBMS_XMLGEN.getXML(v_context);
   RETURN(v_xml);
END;
```

- Using bind variables increases performance
  - Especially for code that is run frequently

DBMS_XMLGEN: Special Characters

- **By default, DBMS_XMLGEN "escapes" special characters**
  - For instance, &, <, >, etc. have special meaning in XML
  - They are replaces with & < > etc

```
SELECT DBMS_XMLGEN.getXML('SELECT ''This is an ampersand: &'' AS "amper"
                                 ,''How  about a tag: <TAG>'' AS "testTag"
                            FROM DUAL')
  FROM DUAL
```

```
<?xml version="1.0"?>
<ROWSET>
 <ROW>
  <amper>This is an ampersand: &amp;</amper>
  <testTag>How  about a tag: &lt;TAG&gt;</testTag>
 </ROW>
</ROWSET>
```

- This can be turned off with setSpecialCharacters()

DBMS_XMLGEN: Functions

- **setPrettyPrinting()**
  - Can be used to turn on or off "pretty" or indented format
- **setIndentationWidth()**
  - Sets the number of characters each indentation level
- **getXML()**
  - Returns output as CLOB
- **getXMLType()**
  - Returns output as XMLType
- **setConvertSpecialChars()**
  - Turns on or off escaping of special characters

DBMS_XMLGEN Errors

- **Most common: Invalid SQL used**

```
ORA-19202: Error occurred in XML processing
ORA-00942: table or view does not exist
ORA-06512: at "SYS.DBMS_XMLGEN", line 7
ORA-06512: at "SYS.DBMS_XMLGEN", line 147
ORA-06512: at "XMLEX.XMLEXAMPLE", line 875
```

- **You will see the SQL error in the error stack**
  - For instance: "table or view does not exist"
- **No special handling needed if SQL is static**
- **However, if the SQL is dynamic…**
  - Add an exception handler that parses the error stack
  - Return the actual SQL error nicely

XMLType – Object Concepts

- **XMLType – An Oracle Object "Type"**
- **Uses "Object Oriented" concepts**
  - Method calls: Tacked on to end of variable..Example:
  - Constructors:
    - Accept XML text as input as VARCHAR2, CLOB, etc.
    - Also accepts BFILE, REF Cursor
    - Parses XML into internal structure
  - Can be used as PL/SQL Variable or parameter
  - Can be used as a column datatype

Overview of the XML DOM Model

- **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

- **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

- **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

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

Using DBMS_XMLDOM

```
FUNCTION DOMExample1 RETURN CLOB IS
   v_DOMDoc   DBMS_XMLDOM.DOMDocument;
   v_RootElem DBMS_XMLDOM.DOMElement;
   v_CustNode DBMS_XMLDOM.DOMNode;
   v_NameNode DBMS_XMLDOM.DOMNode;
   v_NameTextNode DBMS_XMLDOM.DOMNode;
   v_Output   CLOB;
   v_MainNode DBMS_XMLDOM.DOMNode;
   v_RootNode DBMS_XMLDOM.DOMNode;
   v_CustElem DBMS_XMLDOM.DOMElement;
   v_NameElem DBMS_XMLDOM.DOMElement;
   v_NameText DBMS_XMLDOM.DOMText;
   v_XML XMLType;
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );


   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));

   v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, 'Oracle Corporation');
   v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode, DBMS_XMLDOM.makeNode(v_NameText));

   DBMS_LOB.createtemporary(v_Output, False, DBMS_LOB.SESSION);
   DBMS_XMLDOM.writeToClob(v_DOMDoc, v_Output);
   RETURN(v_Output);
END;
```

```
 <customers>
  <customer ID="1">
    <customerName>Oracle Corporation</customerName>
  </customer>
</customers>
```

Using DBMS_XMLDOM

- **Create the actual document with newDOMDocument()**
  - You will need to declare a variable of this type to hold the document
  - If you had passed in an XMLType or text with XML content, then the document would have been created from that content
  - If nothing is passed in, it creates an "empty" object of type DOMDocument

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_DOMDoc   DBMS_XMLDOM.DOMDocument;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );

   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

Using DBMS_XMLDOM

- **Create the "main" Node with makeNode()**
  - Creates an object of type DOMNode
  - Initially, your document has *no* nodes!
  - You need to explicitly create the top or "main" node
  - This node will have no elements or values, it is just a handle for all of the other nodes.

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_MainNode DBMS_XMLDOM.DOMNode;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );

   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

Using DBMS_XMLDOM

- **Create an Element for the root tag with createElement()**
  - Creates an object of type DOMElement
  - This creates an element for the document, but it is not attached! (at this point outputing the document would return nothing)
  - You can set additional properties for this element (example later)

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_RootElem DBMS_XMLDOM.DOMElement;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );

   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

Using DBMS_XMLDOM

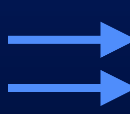
- **Add the element to the document with appendChild()**
  - This attaches the element to the correct node.
  - Returns an object of type DOMNode
  - Notice the makeNode() call to convert the element to a node

    (You don't always have to create a variable for each component)

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_RootNode DBMS_XMLDOM.DOMNode;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode,DBMS_XMLDOM.makeNode(v_RootElem));

   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

Using DBMS_XMLDOM

- **Create another element and attach it to the first**
  - The v_CustElem with a tag of <customer> becomes a child of the v_RootElem with a tag of <customers>
  - This same type of attachments can be made for the whole hierarchy by just repeating the logic



```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_CustNode DBMS_XMLDOM.DOMNode;        v_CustElem DBMS_XMLDOM.DOMElement;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode,DBMS_XMLDOM.makeNode(v_RootElem));

    v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
    v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

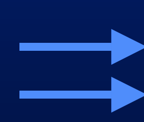
Using DBMS_XMLDOM

- **Set attributes for the tag with setAttribute()**
  - Operates on a DOMElement type object
  - This call adds the "ID" attribute with a value of "1"
  - Multiple attributes can be set with multiple calls to setAttribute()
  - setAttribute() does not return an object, the object to operate on is passed in.

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_CustNode DBMS_XMLDOM.DOMNode;        v_CustElem DBMS_XMLDOM.DOMElement;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode,DBMS_XMLDOM.makeNode(v_RootElem));

   v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
   v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
…
```

Using DBMS_XMLDOM

- **Add another element and node for an actual data element**
  - This only creates the node and names it
  - The value of the data element is not set yet.



```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_NameNode DBMS_XMLDOM.DOMNode;        v_NameElem DBMS_XMLDOM.DOMElement;
 …
BEGIN
…
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

    v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
    v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));

   v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, 'Oracle Corporation');
   v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode, DBMS_XMLDOM.makeNode(v_NameText));

   DBMS_LOB.createtemporary(v_Output, False, DBMS_LOB.SESSION);
   DBMS_XMLDOM.writeToClob(v_DOMDoc, v_Output);
   RETURN(v_Output);
END;
```

Using DBMS_XMLDOM

- **Use createTextNode() to create the actual text for the tag**
  - This command creates an DOMText type
  - The value of the DOMText is set
  - The text is not attached to anything yet

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_NameText DBMS_XMLDOM.DOMText;
 …
BEGIN
…
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));

   v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, 'Oracle Corporation');
   v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode, DBMS_XMLDOM.makeNode(v_NameText));

   DBMS_LOB.createtemporary(v_Output, False, DBMS_LOB.SESSION);
   DBMS_XMLDOM.writeToClob(v_DOMDoc, v_Output);
   RETURN(v_Output);
END;
```

Using DBMS_XMLDOM

- **Attach the DOMText to the Node with appendChild()**
  - Remember, there is one node for the tag, and another for the text
  - This command creates the node for the text, assigning the value specified

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
 v_NameTextNode DBMS_XMLDOM.DOMNode;
 …
BEGIN
…
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));

   v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, 'Oracle Corporation');
   v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode,DBMS_XMLDOM.makeNode(v_NameText));

   DBMS_LOB.createtemporary(v_Output, False, DBMS_LOB.SESSION);
   DBMS_XMLDOM.writeToClob(v_DOMDoc, v_Output);
   RETURN(v_Output);
END;
```

Using DBMS_XMLDOM

- **Use writeToClob() to convert the DOMDocument object to a CLOB**
  - Allows the function (in this case) to return A CLOB for output
  - Can also use other output methods:
    (writeToFile(), writeToBuffer()

```
FUNCTION DOMExample1 RETURN CLOB IS
 …
   v_Output  CLOB;
 …
BEGIN
…
   DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', 1);

   v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
   v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));

   v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, 'Oracle Corporation');
   v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode, DBMS_XMLDOM.makeNode(v_NameText));

   DBMS_LOB.createtemporary(v_Output, False, DBMS_LOB.SESSION);
   DBMS_XMLDOM.writeToClob(v_DOMDoc, v_Output);
   RETURN(v_Output);
END;
```

Using DBMS_XMLDOM: Add Query

- **Now put the code into a cursor loop to retrieve data**
  - Also added a 2nd data element

```
CURSOR customer_cur IS
 …
     SELECT c.customer_id, c.customer_number, c.customer_name
        FROM customer c
       ORDER BY customer_id;
  BEGIN
 …
  FOR rc_cust IN customer_cur LOOP
     v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
     v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));

     DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', rc_cust.customer_id);

     v_NumbElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerNumber');
     v_NumbNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NumbElem));
     v_NumbText := DBMS_XMLDOM.createTextNode(v_DOMDoc, rc_cust.customer_number);
     v_NumbTextNode := DBMS_XMLDOM.appendChild(v_NumbNode, DBMS_XMLDOM.makeNode(v_NumbText));

     v_NameElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName');
     v_NameNode := DBMS_XMLDOM.appendChild(v_CustNode, DBMS_XMLDOM.makeNode(v_NameElem));
     v_NameText := DBMS_XMLDOM.createTextNode(v_DOMDoc, rc_cust.customer_name);
     v_NameTextNode := DBMS_XMLDOM.appendChild(v_NameNode, DBMS_XMLDOM.makeNode(v_NameText));

  END LOOP;
 …
```

DBMS_XMLDOM: Query Results

- **Now the output now looks like this:**

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

DBMS_XMLDOM: Variables

- **You do not have to declare all of the intermediate objects as variables**
  - The function calls themselves can be passed in as parameters to other functions
  - Reduces the number of variables to declare and manage
  - You only need to create a variable if you are going to modify the object
    - For instance: adding an attribute, or adding a child node
  - May lead to less readable or maintainable code though
- **You can also reuse object variables**
  - If you are not going to use an object later, you can reuse the variables.
  - Once an object is in the document, reusing the variable does not effect a previous use

DBMS_XMLDOM: Variables

```
FUNCTION DOMExample3 RETURN CLOB IS
   v_DOMDoc   DBMS_XMLDOM.DOMDocument;      v_MainNode DBMS_XMLDOM.DOMNode;
   v_RootElem DBMS_XMLDOM.DOMElement;       v_RootNode DBMS_XMLDOM.DOMNode;
   v_CustNode DBMS_XMLDOM.DOMNode;          v_CustElem DBMS_XMLDOM.DOMElement;
   v_TextNode DBMS_XMLDOM.DOMNode;          v_Output   CLOB;
 …
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();

   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );
   FOR rc_cust IN customer_cur LOOP
      v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
      v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
      DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', rc_cust.customer_id);

      v_TextNode := DBMS_XMLDOM.appendChild(
                       DBMS_XMLDOM.appendChild(v_CustNode
                          ,DBMS_XMLDOM.makeNode(
                             DBMS_XMLDOM.createElement(v_DOMDoc, 'customerNumber')))
                                ,DBMS_XMLDOM.makeNode(
                                   DBMS_XMLDOM.createTextNode(v_DOMDoc
                                      ,rc_cust.customer_number)));
      v_TextNode := DBMS_XMLDOM.appendChild(
                       DBMS_XMLDOM.appendChild(v_CustNode
                          ,DBMS_XMLDOM.makeNode(
                             DBMS_XMLDOM.createElement(v_DOMDoc, 'customerName')))
                               ,DBMS_XMLDOM.makeNode(
                                  DBMS_XMLDOM.createTextNode(v_DOMDoc
                                     ,rc_cust.customer_name)));
   END LOOP;
 …
```

Needed as a variable to add children

Needed as a variable to set ID attribute

Calls are chained together instead of using variables

The same variable is re-used because it is not really needed

DBMS_XMLDOM: Write utilities!

- **Replace repetitive code with utilities!**
  - Creating "add_text_node" replaces the complicated nested, DBMS_XMLDOM functions with a single line of code

```
PROCEDURE add_text_node(pi_Doc IN OUT DBMS_XMLDOM.DOMDocument
                       ,pi_ParentNode IN OUT DBMS_XMLDOM.DOMNode
                       ,pi_ElemName   IN VARCHAR2
                       ,pi_Value      IN VARCHAR2
                       ) IS
   v_Node DBMS_XMLDOM.DOMNode;
BEGIN
   v_Node := DBMS_XMLDOM.appendChild(
                DBMS_XMLDOM.appendChild(pi_ParentNode
                  ,DBMS_XMLDOM.makeNode(
                     DBMS_XMLDOM.createElement(pi_Doc, pi_ElemName)))
                        ,DBMS_XMLDOM.makeNode(
                           DBMS_XMLDOM.createTextNode(pi_Doc, pi_Value)));

END;
```

Much more readable code!

```
    FOR rc_loc IN location_cur(rc_cust.customer_id) LOOP
       v_LocParNode := add_child_node(v_DOMDoc, v_CustNode, 'locations');
       v_LocNode   := add_child_node(v_DOMDoc, v_LocParNode, 'location');

       add_text_node(v_DOMDoc, v_LocNode, 'locationLabel', rc_loc.location_label);
       add_text_node(v_DOMDoc, v_LocNode, 'locationName' , rc_loc.location_name);

    END LOOP;
```

DBMS_XMLDOM: Nested Cursors

- **Walk a data hierarchy with nested cursors…**

```
FUNCTION DOMExample5 RETURN CLOB IS
…
 CURSOR customer_cur IS
         SELECT c.customer_id, c.customer_number, c.customer_name
           FROM customer c ORDER BY customer_id;
 CURSOR location_cur(pc_customer_id IN customer.customer_id%TYPE)  IS
         SELECT cl.customer_id, cl.location_id, cl.location_label, cl.location_name
           FROM customer_location cl
          WHERE cl.customer_id = pc_customer_id ORDER BY cl.location_label;
BEGIN
   v_DOMDoc := DBMS_XMLDOM.newDOMDocument();
   v_MainNode := DBMS_XMLDOM.makeNode(v_DOMDoc);
   v_RootElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customers');
   v_RootNode := DBMS_XMLDOM.appendChild(v_MainNode, DBMS_XMLDOM.makeNode(v_RootElem) );
   FOR rc_cust IN customer_cur LOOP
      v_CustElem := DBMS_XMLDOM.createElement(v_DOMDoc, 'customer');
      v_CustNode := DBMS_XMLDOM.appendChild(v_RootNode, DBMS_XMLDOM.makeNode(v_CustElem));
      DBMS_XMLDOM.setAttribute(v_CustElem, 'ID', rc_cust.customer_id);

      add_text_node(v_DOMDoc, v_CustNode, 'customerNumber', rc_cust.customer_number);
      add_text_node(v_DOMDoc, v_CustNode, 'customerName', rc_cust.customer_name);

      FOR rc_loc IN location_cur(rc_cust.customer_id) LOOP
         v_LocParNode := add_child_node(v_DOMDoc, v_CustNode, 'locations');
         v_LocNode  := add_child_node(v_DOMDoc, v_LocParNode, 'location');
         add_text_node(v_DOMDoc, v_LocNode, 'locationLabel', rc_loc.location_label);
         add_text_node(v_DOMDoc, v_LocNode, 'locationName' , rc_loc.location_name);
      END LOOP;
   END LOOP;
 …
```

DOM w/Nested Cursors: Result

```
<customers>
  <customer ID="1">
    <customerNumber>A12345</customerNumber>
    <customerName>Spacely Sprockets</customerName>
    <locations>
      <location>
        <locationLabel>INDIANAPOLIS</locationLabel>
        <locationName>Indianapolis plant</locationName>
      </location>
    </locations>
    <locations>
      <location>
        <locationLabel>LOUISVILLE-A</locationLabel>
        <locationName>Louisville plant - Bldg 534C</locationName>
      </location>
    </locations>
    <locations>
      <location>
        <locationLabel>LOUISVILLE-B</locationLabel>
        <locationName>Louisville plant - Bldg 600</locationName>
      </location>
    </locations>
  </customer>
  <customer ID="2">
    <customerNumber>A2765</customerNumber>
    <customerName>Cogswells Cogs</customerName>
    <locations>
      <location>
        <locationLabel>PORTLAND</locationLabel>
 …
```

Mix and Match – Combine Methods

- **You can actually mix and match these methods**
- **XMLTypes can be converted to DOMDocument types and visa versa**
- **DBMS_XMLGEN output can be concatenated together using DOM methods**
- **Example on following slide:**
  - Used two DBMS_XMLGEN calls to produce two XML fragments
  - Uses DBMS_XMLDOM methods to insert the 2nd XMLGEN result *into* the first XMLGEN result

Mix and Match - Example

```
FUNCTION DOM_and_XMLGEN RETURN CLOB IS
     v_custCtx DBMS_XMLGEN.ctxHandle;      x_cust XMLType;
     v_locCtx DBMS_XMLGEN.ctxHandle;       x_loc XMLType;
     v_DOMDoc  DBMS_XMLDOM.DOMDocument;    v_DOMSrc  DBMS_XMLDOM.DOMDocument;
     v_parentNode DBMS_XMLDOM.DOMNode;     v_ImpNode DBMS_XMLDOM.DOMNode;
     v_locElem DBMS_XMLDOM.DOMElement;     v_DOMNode DBMS_XMLDOM.DOMNode;
     v_Output  CLOB;
  BEGIN
     -- Create top nodes as a simple select from parent table, place results into XMLType var
     v_custCtx := DBMS_XMLGEN.newContext('SELECT * FROM CUSTOMER WHERE customer_id = 1');
     DBMS_XMLGEN.setrowsettag(v_custCtx, 'customers');
     DBMS_XMLGEN.setRowTag(v_custCtx, 'customer');
     x_cust := XMLType(DBMS_XMLGEN.getXML(v_custCtx));
     -- Create initial DOM document from the XMLType produced above
     v_DOMDoc := DBMS_XMLDOM.newDOMDocument(x_Cust);
     -- To add a child node from another query, need to get the node handle
     --   this gets the <customer> node since it is a grandchild of the root node
     v_ParentNode := DBMS_XMLDOM.getFirstChild( DBMS_XMLDOM.getFirstChild(
                             DBMS_XMLDOM.makeNode(v_DOMDoc)));
     -- Now create an XML fragement from another XMLGEN query
     v_locCtx := DBMS_XMLGen.newContext('SELECT * FROM CUSTOMER_LOCATION WHERE customer_id = 1');
     DBMS_XMLGEN.setrowsettag(v_locCtx, 'locations');
     DBMS_XMLGEN.setRowTag(v_locCtx, 'location');
     x_loc := XMLType(DBMS_XMLGEN.getXML(v_locCtx)).extract('.');
     -- Convert the XML fragment to an XMLDOM document
     v_DOMSrc := DBMS_XMLDOM.newDOMDocument(x_loc);
     -- Extract an element reference from the document
     v_locElem := DBMS_XMLDOM.getDocumentElement(v_DOMSrc);
     -- Import the node into the target document from the element reference in the source doc
     v_ImpNode := DBMS_XMLDOM.importNode(v_DOMDoc
                     ,DBMS_XMLDOM.makeNode(v_locElem), true);
     -- Connect the imported node to the desired location.
     v_DOMNode := DBMS_XMLDOM.appendChild(v_ParentNode, v_ImpNode);
…
```

Mix and Match – Example Results

This part comes from 1st XMLGEN call

This part comes from 2nd XMLGEN call

Also from 1st call

```
<?xml version="1.0"?>
<customers>
  <customer>
    <CUSTOMER_ID>1</CUSTOMER_ID>
    <CUSTOMER_NUMBER>A12345</CUSTOMER_NUMBER>
    <CUSTOMER_NAME>Spacely Sprockets</CUSTOMER_NAME>
    <locations>
      <location>
        <LOCATION_ID>21</LOCATION_ID>
        <CUSTOMER_ID>1</CUSTOMER_ID>
        <LOCATION_LABEL>LOUISVILLE-A</LOCATION_LABEL>
        <LOCATION_NAME>Louisville plant - Bldg 534C</LOCATION_NAME>
        <ADDRESS_LINE_1>Building 534C</ADDRESS_LINE_1>
        <ADDRESS_LINE_2>1234 Some Industrial Lane</ADDRESS_LINE_2>
        <CITY>Louisville</CITY>
        <STATE>KY</STATE>
        <POSTAL_CODE>40031</POSTAL_CODE>
      </location>
      <location>
        <LOCATION_ID>22</LOCATION_ID>
        <CUSTOMER_ID>1</CUSTOMER_ID>
        <LOCATION_LABEL>LOUISVILLE-B</LOCATION_LABEL>
        <LOCATION_NAME>Louisville plant - Bldg 600</LOCATION_NAME>
        <ADDRESS_LINE_1>Building 600</ADDRESS_LINE_1>
        <ADDRESS_LINE_2>1234 Some Industrial Lane</ADDRESS_LINE_2>
        <CITY>Louisville</CITY>
        <STATE>KY</STATE>
        <POSTAL_CODE>40031</POSTAL_CODE>
      </location>
   </customer>
</customers>
```

XMLType Views

- **Sometimes it is convenient to use view to access specific XML data**
- **The view can present a column with XML data**
- **The XML data can be presented with other columns for query**
- **The column returned can be:**
  - CLOB with XML
  - XMLType column
- **Use PL/SQL to present XML Type**

XMLType View Example

```
 CREATE OR REPLACE VIEW customer_xml_view  AS
  SELECT c.customer_id
        ,xmlexample.get_customer_as_clob(c.customer_id) customer_xml
    FROM customer c
/

SELECT * from customer_xml_view;

CUSTOMER_ID CUSTOMER_XML
----------- -----------------------------------------------------------------------
          1 <customer>
              <customerID>1</customerID>
              <customerNumber>A12345</customerNumber>
              <customerName>Spacely Sprockets</customerName>
            </customer>

          2 <customer>
              <customerID>2</customerID>
              <customerNumber>A2765</customerNumber>
              <customerName>Cogswells Cogs</customerName>
            </customer>
```

Getting XML Data Out of PL/SQL

- **SQL*Plus spooling**
  - Possible, but not the best except for testing
- **AQ or other Messaging**
  - AQ can accept XMLType objects directly
  - Or messages can be sent as CLOBs into AQ or others (like MQ, etc.)
  - AQ methods out of scope for this paper
- **Sent directly to web with HTP**
  - You can build a web service that returns XML
  - For example, an RSS feed

Getting XML Data Out of PL/SQL

- **Writing to a file on the server**
  - Using UTL_FILE
  - Using build in XML functions:
    - DBMS_XMLDOM.WriteToFile()
- **Extracted by Client tool**
  - VB.Net, Java, C, etc. can make a call to PL/SQL
    - PL/SQL Returns CLOB to client
    - Client is responsible for writing to file, putting on queue, etc
- **XMLType Views**
- **Stored locally in CLOBs or XDB**
  - More on this in next session

Which method should I use?

- **DBMS_XMLGEN**
  - Best for simple single level XML files
  - Often for one table only, but can use multi-table joins
  - Useful if table or query is not known beforehand
  - Not as useful for many "Real" XML generation
  - Fastest method
- **DBMS_XMLDOM**
  - Very flexible, therefore useful for complicated XML
  - Can be difficult to understand and use
- **XML/SQL**
  - Easiest to use (in my opinion)
  - Easiest to support for simple enough XML
  - Can be cumbersome for large complex XML
  - Faster than DBMS_XMLDOM

Simple Performance Test

```
 Example XML produced:
<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>

.
Number of runs: 1000
.
XML/SQL elapsed time: 00.841000000
.
XMLDOM elapsed time:  01.643000000
.
XMLGEN elapsed time:  00.400000000
.
```

Shredding XML Data into Tables

- **"Shredding" is the process of parsing XML and separating it into different elements**
- **PL/SQL Methods for "Shredding" XML:**
  - Xpath Access via XMLTypes
  - DBMS_XMLSTORE (inverse of DBMS_XMLGEN)
  - DBMS_XMLPARSER
  - DOM Model API
- **Again, this presentation is not about storing XML *as XML* in the database**
- **Only covering the shredding process**
  - Methods for getting the XML into PL/SQL in next session

Accessing XML Data in PL/SQL

- **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
  - DBMS_XMLPARSE – Into DOM model
- **Accessing – Two options:**
  - XPATH References
  - DOM API

Parsing XML with XMLType

- **XMLType has *constructors* which parse XML into XMLType structure.**
- **Simply pass the XML into XMLType**
  - Can be passed as VARCHAR2, CLOB, BLOB, BFile
- **In order to parse, An XML document should be "Well-Formed":**
  - Every tag needs an end tag
  - Tags should be nested correctly
  - A single root tag in entire document

Example Parsing with XMLType

- **Simply pass the XML into the XMLType object**
  - The XMLType constructor converts the input and returns an XMLType "object"

```
DECLARE
  x_Example XMLType;
BEGIN
  x_Example := XMLType('<inventoryUpdate>
                           <messageSentTime>2007-01-15:12:15Z</messageSentTime>
                       </inventoryUpdate>');
END;
/
PL/SQL procedure successfully completed
```

- **No errors when the XML is well-formed**

Example Parsing with XMLType

- **If the XML has problems, an error is raised…**
  - In this example, the end tag for <inventoryUpdate> is incorrect

```
DECLARE
  x_Example XMLType;
BEGIN
  x_Example := XMLType('<inventoryUpdate>
                           <messageSentTime>2007-01-15:12:15Z</messageSentTime>
                       </inventoryUpd>');
END;

ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00225: end-element tag "inventoryUpd" does not match start-element tag "inventoryUpdate"
Error at line 3
ORA-06512: at "SYS.XMLTYPE", line 301
ORA-06512: at line 4
```

- **The errors are fairly descriptive..**

Example Parsing with XMLType

- **Another example of an error…**
  - In this example, the end tag for <inventoryUpdate> is missing

```
DECLARE
  x_Example XMLType;
BEGIN
  x_Example := XMLType('<inventoryUpdate>
                            <messageSentTime>2007-01-15:12:15Z</messageSentTime>
                        ');
END;

ORA-31011: XML parsing failed
ORA-19202: Error occurred in XML processing
LPX-00007: unexpected end-of-file encountered
ORA-06512: at "SYS.XMLTYPE", line 301
ORA-06512: at line 4
```

- **Write an error handler that traps and processes errors**
  - Using an exception handler for ORA-31011
  - Parse the error stack
  - Handle the specific LPX errors

Extracting XML Data with XPATH

- **XMLType has an *extract()* method to extract XML information**
- ***extract()* uses XPATH references to extract XML fragments**
- **These fragments can be large portions of the document, or single text fields**
- **Used with get..Val() methods to extract data from XML**
  - getStringVal(), getClobVal(), getNumberVal()

Understanding XPATH references

- **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/)

Example XML to Shred

Top level Tag

Data element directly under root tag

2nd level Tag

3rd level Tag

Element Value

```
<?xml version="1.0"?>
<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>
            <item number="A1122">
               <onHandQty>15</onHandQty>
               <optimumQty>60</optimumQty>
               <monthlyUseQty>12</monthlyUseQty>
            </item>
         </itemList>
      </location>
      <location>
         <locName>LOUISVILLE-A</locName>
         <itemList>
            <item number="W1234">
               <onHandQty>512</onHandQty>
…
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
```

Extracting XML Fragments

/inventoryUpdate/customer

```
<?xml version="1.0"?>
<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
…
```

```
PROCEDURE parse_by_XMLType_example1(pi_XMLIn CLOB) IS
  x_Example XMLType;
  x_Fragment XMLType;
BEGIN
  x_Example := XMLType(pi_XMLIn);
  x_fragment := x_Example.extract('/inventoryUpdate/customer');
 DBMS_OUTPUT.PUT_LINE(x_Fragment.getClobVal());
END;
```

```
<customer>
  <custNumber>A12345</custNumber>
  <custName>Spacely Sprockets</custName>
</customer>
```

Extracting XML Leaf Nodes

/inventoryUpdate/customer/custNumber

```
<?xml version="1.0"?>
<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
…
```

```
PROCEDURE parse_by_XMLType_example1(pi_XMLIn CLOB) IS
  x_Example XMLType;
  x_Fragment XMLType;
BEGIN
  x_Example := XMLType(pi_XMLIn);
  x_fragment := x_Example.extract('/inventoryUpdate/customer/custNumber');
  DBMS_OUTPUT.PUT_LINE(x_Fragment.getClobVal());
END;
```

```
<custNumber>A12345</custNumber>
```

Is this really what you want? It is more likely you want the "A12345"

Extracting XML Values

/inventoryUpdate/customer/custNumber/text()

```
<?xml version="1.0"?>
<inventoryUpdate>
   <messageSentTime>2007-01-15:12:15Z</messageSentTime>
   <customer>
      <custNumber>A12345</custNumber>
      <custName>Spacely Sprockets</custName>
   </customer>
   <locationList>
      <location>
         <locName>INDIANAPOLIS</locName>
…
```

```
PROCEDURE parse_by_XMLType_example1(pi_XMLIn CLOB) IS
  x_Example XMLType;
  v_Value VARCHAR2(40);
BEGIN
  x_Example := XMLType(pi_XMLIn);
  v_Value := x_Example.extract('/inventoryUpdate/customer/custNumber/text()');
 DBMS_OUTPUT.PUT_LINE(v_Value);
END;
```

```
A12345
```

Extracting XML – Ordinal Predicates

/inventoryUpdate/customer/locationlist/location[1]

/inventoryUpdate/customer/locationlist/location[2]

```
<?xml version="1.0"?>
<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>
            <item number="A1122">
               <onHandQty>15</onHandQty>
               <optimumQty>60</optimumQty>
               <monthlyUseQty>12</monthlyUseQty>
            </item>
         </itemList>
      </location>
      <location>
         <locName>LOUISVILLE-A</locName>
         <itemList>
            <item number="W1234">
               <onHandQty>512</onHandQty>
…
         </itemList>
      </location>
   </locationList>
</inventoryUpdate>
```

Extracting XML Attribute Values

/inventoryUpdate/customer/locationList/location[1]/itemList/item[1]/@number

```
…
      <location>
         <locName>INDIANAPOLIS</locName>
         <itemList>
            <item number="W1234">
               <onHandQty>256</onHandQty>
               <optimumQty>1000</optimumQty>
               <monthlyUseQty>523</monthlyUseQty>
            </item>
            <item number="A1122">
               <onHandQty>15</onHandQty>
…
```

```
PROCEDURE parse_by_XMLType_example1(pi_XMLIn CLOB) IS
  v_XML_Clob CLOB := get_xml_example1;
  x_Example XMLType;
  x_Fragment XMLType;
  XML_Parse_Failed EXCEPTION;
  PRAGMa Exception_Init(XML_Parse_Failed, -31011);
BEGIN
  x_Example := XMLType(pi_XMLIn);
   v_Value :=
x_Example.extract('/inventoryUpdate/locationList/location[1]/itemList/item[1]/@number').getStringVal
();
 DBMS_OUTPUT.PUT_LINE(v_Value);
END;
```

```
W1234
```

XPATH – Other syntax

//	All descendants of node
.	Current Node
..	Parent Node
[last()]	Last child of Node
[position()<3]	First two children
[price>35.00]	All elements where price > 35
*	All nodes

Putting it to Use: Insert Records

```
PROCEDURE save_inventory_updateXPATH(pi_XMLIn IN CLOB) IS
   x_InvUpdate    XMLType;
   v_TopNodeName  VARCHAR2(80) := '/inventoryUpdate';
   v_CustNodeName VARCHAR2(80) := v_TopNodeName||'/customer';
   v_LocNode      VARCHAR2(80) := v_TopNodeName||'/locationList/location';
   v_ItemNode     VARCHAR2(80) := '/itemList/item';
   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_inventory_update_id INVENTORY_UPDATE.INVENTORY_UPDATE_ID%TYPE;
   i_LocNode      INTEGER;      v_CurrLocNode  VARCHAR2(80);
   i_ItemNode     INTEGER;      v_CurrItemNode VARCHAR2(80);
   b_customerOK   BOOLEAN;      b_locationOK   BOOLEAN;       b_itemOK       BOOLEAN;
BEGIN
   x_InvUpdate := XMLType(pi_XMLIn);
   -- Parse header information
   r_inventoryUpdate.Inventory_Update_Date
       := xml_to_date(x_InvUpdate.extract(v_TopNodeName||'/messageSentTime/text()').getStringVal());
   v_CustNumber := x_InvUpdate.extract(v_CustNodeName||'/custNumber/text()').getStringVal();

   BEGIN
      SELECT customer_id INTO v_customer_ID
        FROM Customer
       WHERE customer_number = v_CustNumber;
       b_customerOK := True;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.put_line('ERROR!! - Could not find customer number: '||v_custNumber);
         b_customerOK := False;
   END;
…
```

Putting it to Use: Insert Records

```
FUNCTION get_xml_text(pi_XMLIn IN XMLType
                     ,pi_XPATH IN VARCHAR2) RETURN VARCHAR2 IS
BEGIN
   RETURN(pi_XMLIn.extract(pi_XPATH||'/text()').getStringVal());
END;

PROCEDURE save_inventory_updateXPATH(pi_XMLIn IN CLOB) IS
…
BEGIN
   x_InvUpdate := XMLType(pi_XMLIn);
   -- Parse header information
   r_inventoryUpdate.Inventory_Update_Date
       := xml_to_date(x_InvUpdate.extract(v_TopNodeName||'/messageSentTime/text()').getStringVal());
  v_CustNumber := get_xml_text(x_InvUpdate,v_CustNodeName||'/custNumber');

   BEGIN
      SELECT customer_id INTO v_customer_ID
        FROM Customer
       WHERE customer_number = v_CustNumber;
       b_customerOK := True;
   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         DBMS_OUTPUT.put_line('ERROR!! - Could not find customer number: '||v_custNumber);
         b_customerOK := False;
   END;
…
```

Putting it to Use: Insert Records

```
PROCEDURE save_inventory_updateXPATH(pi_XMLIn IN CLOB) IS
…
BEGIN
   …
   i_LocNode := 1;
   v_CurrLocNode := v_LocNode||'['||i_LocNode||']';
   WHILE x_invUpdate.existsNode(v_CurrLocNode) = 1 LOOP
      v_LocLabel := x_invUpdate.extract(v_CurrLocNode||'/locName/text()').getStringVal();
      BEGIN
         SELECT location_id INTO r_inventoryUpdate.Location_Id
           FROM customer_location loc
          WHERE loc.customer_id = v_customer_id
            AND loc.location_label = v_locLabel;
         b_LocationOK := True;
      EXCEPTION
         WHEN NO_DATA_FOUND THEN
            DBMS_OUTPUT.put_line('ERROR!! - Could not find location: '||v_locLabel);
            b_locationOK := False;
      END;
      DBMS_OUTPUT.put_line('Cust node: '||i_LocNode||' - '||v_CurrLocNode||' Name: '||v_LocLabel);
…
```

Putting it to Use: Insert Records

```
PROCEDURE save_inventory_updateXPATH(pi_XMLIn IN CLOB) IS
   …
BEGIN
   …
      i_itemNode := 1;
      v_currItemNode := v_CurrLocNode || v_ItemNode || '['||i_itemNode||']';
      WHILE x_invUpdate.existsNode(v_CurrItemNode) = 1 LOOP
         v_itemNumber := x_invUpdate.extract(v_CurrItemNode||'/@number').getStringVal();
         DBMS_OUTPUT.put_line('...Item node: '||i_ItemNode||'-'||v_CurrItemNode||'-'||v_itemNumber);
         BEGIN
            SELECT item_id INTO r_inventoryUpdate.Item_Id
              FROM item
             WHERE item.item_number = v_itemNumber;
             b_ItemOK := True;
         EXCEPTION
            WHEN NO_DATA_FOUND THEN
               DBMS_OUTPUT.put_line('ERROR!!  - Could not find item: '||v_itemNumber);
               b_itemOK := False;
         END;
         r_inventoryUpdate.On_Hand_Quantity
             := x_invUpdate.extract(v_CurrItemNode||'/onHandQty/text()').getStringVal();
         r_inventoryUpdate.Optimum_Quantity
             := x_invUpdate.extract(v_CurrItemNode||'/optimumQty/text()').getStringVal();
         r_inventoryUpdate.Per_Month_Usage_Quantity
             := x_invUpdate.extract(v_CurrItemNode||'/monthlyUseQty/text()').getStringVal();
…
```

Putting it to Use: Insert Records

```
PROCEDURE save_inventory_updateXPATH(pi_XMLIn IN CLOB) IS
   …
BEGIN
   …
   IF b_customerOK AND b_locationOK AND b_itemOK THEN
            BEGIN
               INSERT INTO inventory_update(item_id, location_id, inventory_update_date
                                 ,on_hand_quantity, optimum_quantity, per_month_usage_quantity)
                  VALUES(r_inventoryUpdate.Item_Id
                        ,r_inventoryUpdate.Location_Id
                        ,r_inventoryUpdate.Inventory_Update_Date
                        ,r_inventoryUpdate.On_Hand_Quantity
                        ,r_inventoryUpdate.Optimum_Quantity
                        ,r_inventoryUpdate.Per_Month_Usage_Quantity
                        ) RETURNING inventory_update_id INTO v_inventory_update_id;
               DBMS_OUTPUT.PUT_LINE('......Inserting record - ID: '||v_inventory_update_id);
            EXCEPTION
               WHEN OTHERS THEN
                  DBMS_OUTPUT.put_line('Could not insert record - '||SQLERRM);
            END;
         END IF;
         i_itemNode := i_itemNode + 1;
         v_currItemNode := v_CurrLocNode || v_ItemNode || '['||i_itemNode||']';
      END LOOP;
      i_locNode := i_locNode + 1;
      v_CurrLocNode := v_LocNode||'['||i_LocNode||']';
   END LOOP;
END;
```

Putting it to Use: Insert Records

```
Found customer node: 1 - /inventoryUpdate/locationList/location[1] Name: INDIANAPOLIS
...Found item node: 1 - /inventoryUpdate/locationList/location[1]/itemList/item[1] - W1234
......Inserting record - ID: 41
...Found item node: 2 - /inventoryUpdate/locationList/location[1]/itemList/item[2] - A1122
......Inserting record - ID: 42
...Found item node: 3 - /inventoryUpdate/locationList/location[1]/itemList/item[3] - W1257
......Inserting record - ID: 43
Found customer node: 2 - /inventoryUpdate/locationList/location[2] Name: LOUISVILLE-A
...Found item node: 1 - /inventoryUpdate/locationList/location[2]/itemList/item[1] - W1234
......Inserting record - ID: 44
...Found item node: 2 - /inventoryUpdate/locationList/location[2]/itemList/item[2] - A342

ERROR!! - Could not find item: A342

...Found item node: 3 - /inventoryUpdate/locationList/location[2]/itemList/item[3] - W1272
......Inserting record - ID: 45
```

```
SELECT * FROM Inventory_Update
/
INVENTORY_UPDATE_ID     ITEM_ID LOCATION_ID INVENTORY_UPDATE_DATE ON_HAND_QUANTITY OPTIMUM_QUANTITY PER_MONTH_USAGE_QUANTITY
------------------- ----------- ----------- --------------------- ---------------- ---------------- ------------------------
                 41           1          23 1/15/2007 12:15:00 PM              256             1000                      523
                 42           7          23 1/15/2007 12:15:00 PM               15               60                       12
                 43           2          23 1/15/2007 12:15:00 PM              712             1200                      612
                 44           1          21 1/15/2007 12:15:00 PM              512             1200                      456
                 45           3          21 1/15/2007 12:15:00 PM              287              600                      292
```

Use Good Coding Practices!

- **Some good coding practices shown:**
  - Generic function for converting XML date formats
  - Only specify each node name *once!*
  - Generic *get_xml_text* function
- **Other Opportunities:**
  - Standard XML Error handler
  - Table API access to tables
  - Wrap XML access in a "Bean" package

Idea: PL/SQL "XML Bean"

- **Hard coding XPATH paths all over your code is not a good practice**
- **Also, what if multiple programs need to parse an XML document?**
- **How many places do you need to change if the XML format changes?**
- **Try using an "XML Bean"**
  - Each property has a getter and a setter
  - Other useful code (node exists for instance)

XML Bean Example

```
CREATE OR REPLACE PACKAGE BODY inventoryUpdateBean is
   v_TopNodeName  VARCHAR2(80) := '/inventoryUpdate';
   v_CustNodeName VARCHAR2(80) := v_TopNodeName||'/customer';
   v_LocNodeName  VARCHAR2(80) := v_TopNodeName||'/locationList/location';
   v_ItemNodeName VARCHAR2(80) := '/itemList/item';

   FUNCTION get_xml_text(pi_XMLIn IN XMLType
                        ,pi_XPATH IN VARCHAR2) RETURN VARCHAR2 IS
   BEGIN
      RETURN(pi_XMLIn.extract(pi_XPATH||'/text()').getStringVal());
   END;
   FUNCTION get_attribute(pi_XMLIn IN XMLType
                        ,pi_XPATH IN VARCHAR2
                        ,pi_attrName IN VARCHAR2) RETURN VARCHAR2 IS
   BEGIN
      RETURN(pi_XMLIn.extract(pi_XPATH||'/@'||pi_attrName).getStringVal());
   END;

   FUNCTION xml_to_date(pi_xml_value IN VARCHAR2) RETURN DATE IS
     v_Date DATE;
   BEGIN
     v_Date := TO_DATE(REPLACE(pi_xml_value,'Z',''), 'YYYY-MM-DD:HH24:MI:SS');
     RETURN(v_Date);
   EXCEPTION
     WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('Could not convert to a date of YYYY-MM-DD:HH24:MI:SS : '||pi_xml_value);
        RETURN(NULL);
   END;

   FUNCTION addIndex(pi_Index IN INTEGER) RETURN VARCHAR2 IS
   BEGIN
      RETURN('['||pi_Index||']');
   END;
   …
```

XML Bean Example

```
…
FUNCTION get_messageSentTime(pi_Msg IN OUT NOCOPY XMLType) RETURN DATE IS
BEGIN
   RETURN(xml_to_date(get_xml_text(pi_Msg, v_TopNodeName||'/messageSentTime')));
END;
FUNCTION get_custNumber(pi_Msg IN OUT NOCOPY XMLType) RETURN VARCHAR2 IS
BEGIN
   RETURN(get_xml_text(pi_Msg, v_CustNodeName||'/custNumber'));
END;
FUNCTION get_custName(pi_Msg IN OUT NOCOPY XMLType) RETURN VARCHAR2 IS
BEGIN
   RETURN(get_xml_text(pi_Msg, v_CustNodeName||'/custName'));
END;
FUNCTION get_locName(pi_Msg IN OUT NOCOPY XMLType, pi_LocIndex IN INTEGER) RETURN VARCHAR2 IS
BEGIN
   RETURN(get_xml_text(pi_Msg, v_LocNodeName||addIndex(pi_LocIndex)||'/locName'));
END;
FUNCTION get_itemNumber(pi_Msg IN OUT NOCOPY XMLType, pi_LocIndex IN INTEGER
                       ,pi_ItemIndex IN INTEGER) RETURN VARCHAR2 IS
BEGIN
   RETURN(get_attribute(pi_Msg, v_LocNodeName||addIndex(pi_LocIndex)
          ||v_ItemNodeName||addIndex(pi_itemIndex), 'number'));
END;

FUNCTION existsLocNode(pi_Msg IN OUT NOCOPY XMLType, pi_locIndex IN INTEGER) RETURN boolean IS
BEGIN
   IF pi_Msg.existsNode(v_LocNodeName||addIndex(pi_locIndex)) = 1 THEN
      RETURN(True);
   ELSE
      RETURN(False);
   END IF;
END;
…
```

XML Bean Example

```
…
PROCEDURE set_xml_text(pi_XML       IN OUT NOCOPY XMLType
                      ,pi_XPATH     IN VARCHAR2
                      ,pi_NewValue IN VARCHAR2) IS
 BEGIN
    IF pi_XML.existsNode(pi_XPATH) = 1 THEN
       SELECT updateXML(pi_XML, pi_XPATH||'/text()', pi_NewValue) INTO pi_XML
         FROM DUAL;
    END IF;
 END;

 PROCEDURE set_messageSentTime(pi_Msg IN OUT NOCOPY XMLType, pi_NewValue IN DATE) IS
 BEGIN
    set_xml_text(pi_Msg, v_TopNodeName||'/messageSentTime', date_to_xml(pi_NewValue));
 END;

 PROCEDURE set_custNumber(pi_Msg IN OUT NOCOPY XMLType, pi_NewValue IN VARCHAR2) IS
 BEGIN
    set_xml_text(pi_Msg, v_CustNodeName||'/custNumber', pi_NewValue);
 END;

 PROCEDURE set_itemNumber(pi_Msg IN OUT NOCOPY XMLType, pi_LocIndex IN INTEGER
                         ,pi_ItemIndex IN INTEGER, pi_NewValue IN VARCHAR2) IS
 BEGIN
    set_attribute(pi_Msg, v_LocNodeName||addIndex(pi_LocIndex)||v_ItemNodeName
                  ||addIndex(pi_itemIndex), 'number', pi_NewValue);
 END;
 …
end inventoryUpdateBean;
```

Using the XML Bean package

```
PROCEDURE save_inventory_updateBean(pi_XMLIn IN CLOB) IS
   x_InvUpdate    XMLType;
   v_error_svr  VARCHAR2(1);    v_error_txt VARCHAR2(240);
   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_inventory_update_id INVENTORY_UPDATE.INVENTORY_UPDATE_ID%TYPE;
   i_LocNode      INTEGER;       i_ItemNode     INTEGER;
   b_customerOK   BOOLEAN;       b_locationOK   BOOLEAN;       b_itemOK       BOOLEAN;
BEGIN
   x_InvUpdate := XMLType(pi_XMLIn);
   r_inventoryUpdate.Inventory_Update_Date := inventoryUpdateBean.get_messageSentTime(x_InvUpdate);
   v_CustNumber                            := inventoryUpdateBean.get_custNumber(x_InvUpdate);

   v_customer_ID := tapi$customer.get_id(v_CustNumber);
   i_LocNode := 1;
   WHILE inventoryUpdateBean.existsLocNode(x_invUpdate, i_LocNode) LOOP
      v_LocLabel := inventoryUpdateBean.get_locName(x_invUpdate, i_LocNode);
      r_inventoryUpdate.location_id := tapi$customer_location.get_id(v_locLabel);

      DBMS_OUTPUT.put_line('Found customer node: '||i_LocNode||' Name: '||v_LocLabel);
      i_itemNode := 1;
      WHILE inventoryUpdateBean.existsItemNode(x_invUpdate, i_LocNode, i_itemNode) LOOP
         v_itemNumber := inventoryUpdateBean.get_itemNumber(x_invUpdate, i_LocNode, i_itemNode);
         DBMS_OUTPUT.put_line('...Found item node: '||i_ItemNode||' - '||v_itemNumber);
         BEGIN
            v_itemNumber := tapi$item.get_id(v_itemNumber);
         EXCEPTION
            WHEN tapi$util.e_invalid_mnemonic THEN
               DBMS_OUTPUT.put_line('ERROR!! - Could not find item: '||v_itemNumber);
               b_itemOK := False;
         END;
```

Using the XML Bean package

```
        r_inventoryUpdate.On_Hand_Quantity
                := inventoryUpdateBean.get_onHandQty(x_invUpdate, i_locNode, i_itemNode);
        r_inventoryUpdate.Optimum_Quantity
                := inventoryUpdateBean.get_optimumQty(x_invUpdate, i_locNode, i_itemNode);
        r_inventoryUpdate.Per_Month_Usage_Quantity
                := inventoryUpdateBean.get_monthlyUseQty(x_invUpdate, i_locNode, i_itemNode);
        IF b_customerOK AND b_locationOK AND 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;
        END IF;
        i_itemNode := i_itemNode + 1;
     END LOOP;
     i_locNode := i_locNode + 1;
  END LOOP;
END;
```

DBMS_XMLSTORE

- **Inverse of DBMS_XMLGEN**
- **Based on a canonical mapping of XML to table definition**
- **C Based code**
  - More efficient than Java based DBMS_XMLSave
  - Does not need to load JVM
- **Better memory usage**
- **Can use bind variables**
- **Can be used to INSERT, UPDATE, DELETE**
  - insertXML(), updateXML(), deleteXML()

Using DBMS_XMLSTORE

```
DECLARE
  v_Msg CLOB :=
  '<ROWSET>
     <ROW>
        <ITEM_NUMBER>X4723</ITEM_NUMBER>
        <ITEM_NAME>X Widget type 3</ITEM_NAME>
        <ITEM_DESCRIPTION>A special X type widget</ITEM_DESCRIPTION>
        <ITEM_LIST_PRICE>25.57</ITEM_LIST_PRICE>
     </ROW>
     <ROW>
        <ITEM_NUMBER>Y9999</ITEM_NUMBER>
        <ITEM_NAME>Y Widget</ITEM_NAME>
        <ITEM_DESCRIPTION>A Y type widget</ITEM_DESCRIPTION>
        <ITEM_LIST_PRICE>2.69</ITEM_LIST_PRICE>
     </ROW>
  </ROWSET>
  ';
   v_storeCtx DBMS_XMLStore.ctxType;
   i_RowsIns INTEGER;
BEGIN
   v_storeCtx := DBMS_XMLStore.newContext('XMLEX.ITEM');
   DBMS_XMLStore.clearUpdateColumnList(v_storeCtx);
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_NUMBER');
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_NAME');
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_DESCRIPTION');
   DBMS_XMLStore.setUpdateCOlumn(v_storeCtx, 'ITEM_LIST_PRICE');
   i_RowsIns := DBMS_XMLStore.insertXML(v_storeCtx, v_Msg);
   DBMS_XMLStore.closeContext(v_storeCtx);
   DBMS_OUTPUT.PUT_LINE(i_RowsIns||' rows inserted into ITEM');
END;
/
2 rows inserted into ITEM
```

DBMS_XMLSTORE - Limitations

- **You cannot easily override the column names**
  - You *can* override the ROW tag with setRowTag()
  - It does not really matter what the root tag is
  - But the items *have* to be the column names
- **You can do insert or update…not both**
  - All of the records in the file have to be inserts or updates
  - Not flexible for doing a merge
- **Debugging is difficult**
  - It is not always clear from the errors what the problem is.

Use XSL to map to Canonical

- **You can use an XSL stylesheet to convert incoming XML to the canonical format**
- **XMLType.convert() is a method to do this**
- **The XSL can do the following:**
  - Create canonical <ROWSET> and <ROW> tags
  - Map XML tags to canonical column names
  - Flatten out an XML hierarchy into the canonical single row format
  - Restrict rows that are mapped
  - Etc. Anything that XSL can do

Example XSL Mapping to Canonical

Incoming non-Canonical message with different column names, multiple levels etc:

```
x_Msg := XMLType(
'<itemList>
   <item>
      <itemIdentification>
         <itemNumber>X4723</itemNumber>
         <itemID>12345</itemID>
      </itemIdentification>
      <itemInfo>
         <itemName>X Widget type 3</itemName>
         <description>A special X type</description>
         <listPrice>25.57</listPrice>
         <discountPrice>24.00</discountPrice>
      </itemInfo>
   </item>
   <item>
      <itemIdentification>
         <itemNumber>Y9999</itemNumber>
         <itemID>12399</itemID>
      </itemIdentification>
      <itemInfo>
         <itemName>Y Widget</itemName>
         <description>A Y type widget</description>
         <listPrice>2.69</listPrice>
         <discountPrice>2.69</discountPrice>
      </itemInfo>
   </item>
</itemList>
');
```

Example XSL Mapping to Canonical

XSL Style sheet to create a canonical format:

```
X_XSL := XMLType(
'<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <ROWSET>
      <xsl:for-each select="/itemList/item">
      <ROW>
        <ITEM_NUMBER>
          <xsl:value-of select="current()/itemIdentification/itemNumber" />
        </ITEM_NUMBER>
        <ITEM_NAME>
          <xsl:value-of select="current()/itemInfo/itemName" />
        </ITEM_NAME>
        <ITEM_DESCRIPTION>
          <xsl:value-of select="current()/itemInfo/description" />
        </ITEM_DESCRIPTION>
        <ITEM_LIST_PRICE>
          <xsl:value-of select="current()/itemInfo/listPrice" />
        </ITEM_LIST_PRICE>
      </ROW>
      </xsl:for-each>
    </ROWSET>
  </xsl:template>
</xsl:stylesheet>
');
```

Example XSL Mapping to Canonical

Use the XSL style sheet with XMLType.transform() :

```
PROCEDURE XML_Store_Example3 IS
   …
   v_storeCtx DBMS_XMLStore.ctxType;
   i_RowsIns INTEGER;
   x_CanonicalMsg XMLType;
BEGIN
   x_CanonicalMsg := XMLType.transform(x_Msg, x_XSL);

   DBMS_OUTPUT.put_line('****AFTER XSL Transformation*****');
   DBMS_OUTPUT.put_line(x_CanonicalMsg.getStringVal());

   v_storeCtx := DBMS_XMLStore.newContext('XMLEX.ITEM');
   DBMS_XMLStore.clearUpdateColumnList(v_storeCtx);
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_NUMBER');
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_NAME');
   DBMS_XMLStore.setUpdateColumn(v_storeCtx, 'ITEM_DESCRIPTION');
   DBMS_XMLStore.setUpdateCOlumn(v_storeCtx, 'ITEM_LIST_PRICE');
   i_RowsIns := DBMS_XMLStore.insertXML(v_storeCtx, x_CanonicalMsg);
   DBMS_XMLStore.closeContext(v_storeCtx);
   DBMS_OUTPUT.PUT_LINE(i_RowsIns||' rows inserted into ITEM');

END;
```

Now DBMS_XMLSTORE can be used to save the data

Example XSL Mapping to Canonical

Executing the code, showing the transformed canonical format:

```
****AFTER XSL Transformation*****
<ROWSET>
  <ROW>
    <ITEM_NUMBER>X4723</ITEM_NUMBER>
    <ITEM_NAME>X Widget type 3</ITEM_NAME>
    <ITEM_DESCRIPTION>A special X type widget</ITEM_DESCRIPTION>
    <ITEM_LIST_PRICE>25.57</ITEM_LIST_PRICE>
  </ROW>
  <ROW>
    <ITEM_NUMBER>Y9999</ITEM_NUMBER>
    <ITEM_NAME>Y Widget</ITEM_NAME>
    <ITEM_DESCRIPTION>A Y type widget</ITEM_DESCRIPTION>
    <ITEM_LIST_PRICE>2.69</ITEM_LIST_PRICE>
  </ROW>
</ROWSET>

2 rows inserted into ITEM
```

DOM Model API

- Using DBMS_XMLPARSER to parse XML
- And DBMS_XMLDOM to read parsed XML
- Covered in part II….

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?

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