Oracle XML DB im Produktionseinsatz

Jan Golka
Data Design & Management GmbH

Bauernwaldstraße 126
70195 Stuttgart
Telefon: + 49 – 711 – 69 70 71 – 10
E-Mail: info@d-d-m.de
www.d-d-m.de

Agenda

- Einführung.
- Kurzbeschreibung der Aufgabe.
- Gewählter Lösungsweg.
- Ein vereinfachtes Beispiel.
- Beschreibung des realisierten Systems.
- Probleme, Beobachtungen etc.
- Fragen.

Warum XML?

- XML wird zunehmend benutzt, um Daten zwischen Anwendungen auszutauschen. Ein nicht seltenes Szenario ist, dass als Folge von Re-Engineering die bisher verwendeten CSV oder sonstigen „flat files“ durch XML ersetzt werden.

- Da XML wesentlich mehr Inhalt als eine „flache“ Datei übertragen kann, wird oft versucht, Schnittstellen zu vereinfachen und statt mehreren CSV Dateien (eine Datei pro Tabelle) nur eine XML Datei zu verwenden.

  - Der Vorteil: die (formelle) Datenintegrität wird durch XML gewährleistet. Sie kann auch einfach geprüft werden.

  - Der Preis: die Datengenerierung wird wesentlich komplexer.

- Andererseits kann man sich kaum vorstellen, dass eine komplexe Schnittstelle, über die der Inhalt von Hunderten von relationalen Tabellen übertragen werden muss, mit „flat files“ überhaupt fehlerfrei bedient werden kann .

- Mit anderen Worten, bleibt nichts anderes übrig, als sich mit dem Thema XML zu befassen.

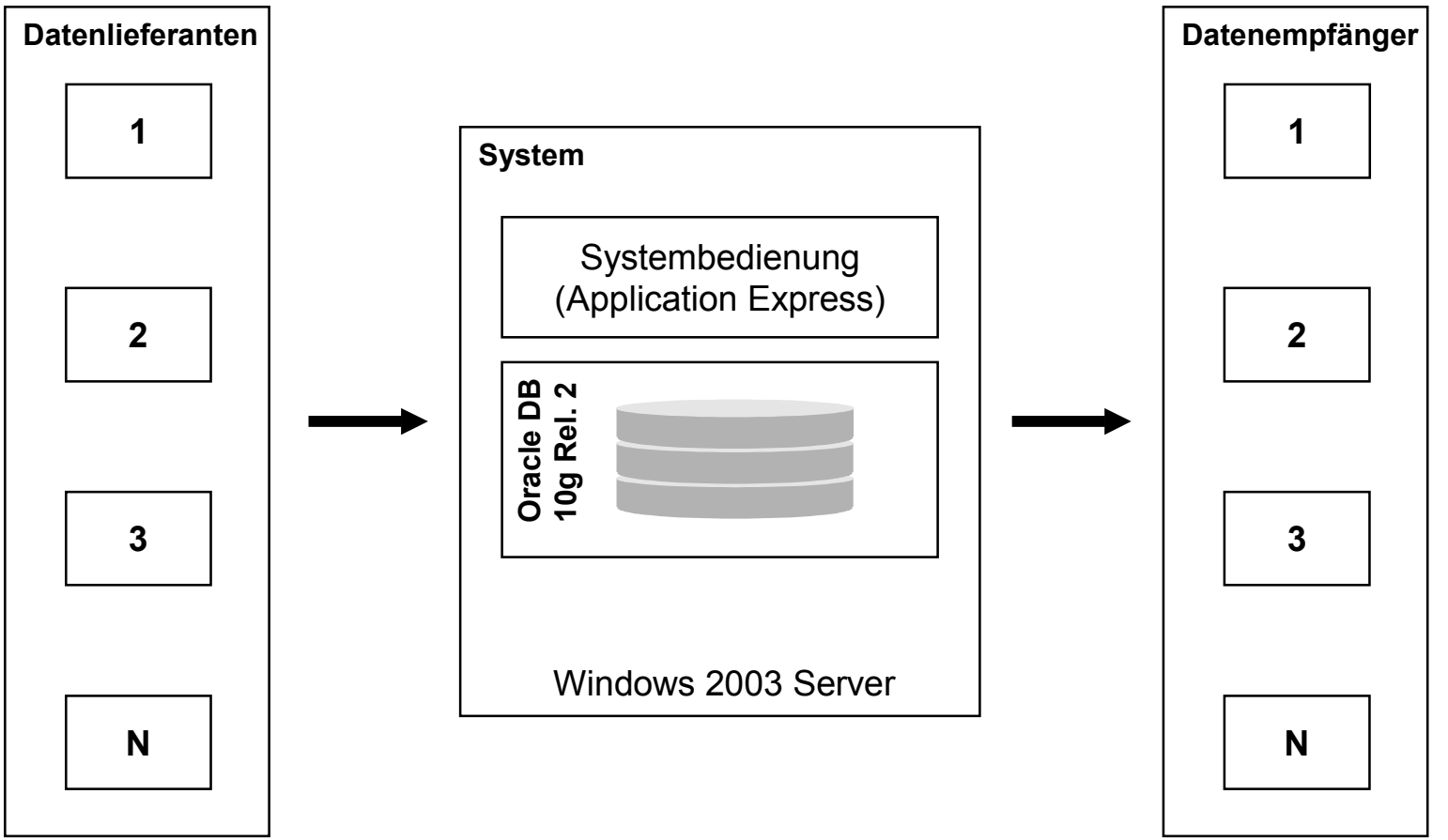
Zu der Aufgabe

- Der Teilinhalt einer Oracle Datenbank muss periodisch in Form von XML Dateien ausgegeben werden.
- Die hierarchische Struktur der XML Datei ist mit Hilfe einer sog. „DTD Datei“ („Document Type Definition“) festgelegt.
- Die relationale Datenbankstruktur ist durch ihr Datenmodell definiert.
- Die beiden Strukturen sind vollkommen unterschiedlich, und auf den ersten Blick ist es ist nicht ganz klar, wie die eine in die andere transformiert werden kann.
- Erschwerend kommt noch hinzu, dass dieses Thema – zumindest auf der XML Seite – wenig Beachtung findet. Im Blickpunkt liegt dort eher die Verarbeitung der (relativ kleinen) Dokumente, deren Transformation, usw.
    - Die gängigen XML Tools machen alles bevorzugt im Hauptspeicher, was im Fall einer Datei mit mehreren hundert MB nicht unbedingt von Vorteil ist.

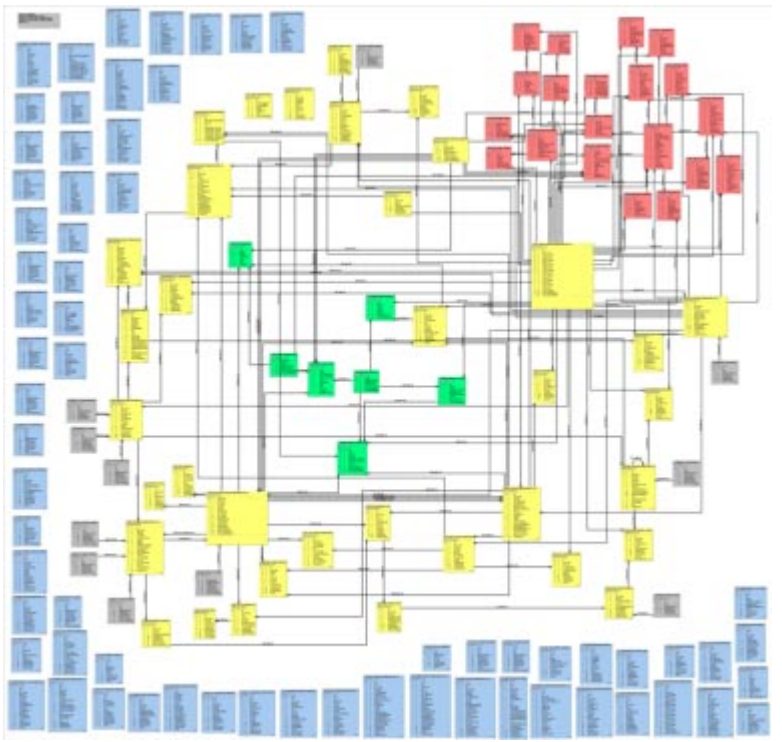
Entwicklungsaufgabe

- Ein System für die periodische Belieferung der Schnittstellenpartner mit komplexen XML Massendaten sollte erstellt werden.
- Datenquelle:
  - Eine Oracle 10g Rel. 2 Datenbank (Betriebssystemplattform: Windows 2003 Server).
  - In Abhängigkeit von der Schnittstelle, sind Daten aus bis zu 150 Tabellen zu exportieren.
  - Das Volumen der XML Daten beträgt, je nach Schnittstelle, bis zu ca. 2,2 GB in über hundert Dateien.
- Datensenke:
  - Ein für jede Lieferung neu anzulegendes Verzeichnis auf der gleichen Hardware.

IT-Landschaft



Strukturkonflikt: Relational ←→ Hierarchisch



Lösungsmöglichkeiten

- Es gibt prinzipiell zwei Möglichkeiten:
  - XML direkt in der Datenbank zu generieren.
  - externe Tools einzusetzen (hauptsächlich Java).
- Ein Oracle Entwickler wird oft den ersten Weg versuchen, um im bekannten Umfeld zu bleiben. Auch hier gibt es zwei Wege:
  - PL/SQL mit Oracle XML Erweiterungen einsetzen.
  - Ein Pro*C/C++ oder Java/SQLJ Programme zu schreiben.
- Ein PL/SQL Enthusiast versucht selbstverständlich, seine bevorzugte Programmiersprache einzusetzen.
- Der Vortrag zeigt, wie das alles funktioniert.

Hierarchisch organisierte Daten aus der Oracle Datenbank: ein Widerspruch?

- Die Antwort von Oracle: „XML DB“ mit neuen Elementen wie
  - SQL/XML („XML extensions of SQL“), XQuery, etc.
- Detailfragen:
  - Wie gehe ich vor?
  - Wie werden die relationalen Daten in die hierarchische Struktur umgewandelt?
  - Hilft es, wenn object-Views erstellt werden?
  - Ist es vorteilhaft, ein „XML Schema“ zu verwenden?
  - Welche PL/SQL Programmierschnittstelle soll verwendet werden?
- Nicht zuletzt ist auch das „Kosten – Nutzen“ Thema sehr wichtig.

Oracle XML DB in Stichworten

- „Oracle XML DB is the name for a set of Oracle Database technologies related to high-performance XML storage and retrieval. It provides native XML support by encompassing both SQL and XML data models in an interoperable manner.“
- Eingeführt in Oracle 9i; (wesentliche) Erweiterungen in Oracle 10g Rel. 2.
- Die wichtigsten XML DB Konstrukte sind:
  - **XMLTYPE**: „a native server datatype that lets the database understand that a column or table contains XML“.
  - **XMLTYPE View**: „wrap existing relational data in XML formats“, mit anderen Worten, sorgt für das Mapping zwischen relationalen und hierarchischen Strukturen.

Lösungsweg zusammengefasst

- Zuerst werden die Ausgangsdaten vorbereitet.
- XMLTYPE Views sorgen für das notwendige relational-hierarchisch Mapping.
- Die generierten XML Daten werden in einem XMLTYPE Tabellenattribut zwischengespeichert.
- Die XML Erstellung passiert in einer CURSOR SELECT Loop:
  - „SELECT VALUE(xv) FROM < XMLTYPE View> xv „
  - „INSERT INTO < XMLTYPE Table> VALUES ...“.
- Nach Select-Loop-Ende werden die XML Daten um den XML-Prolog etc. ergänzt und – als schon fertig formatierte XML Datei – in ein CLOB Attribut geschrieben.
- Am Ende findet ein Spooling der XML Daten aus CLOBs in Betriebssystemfiles statt.

Aufbereitung der Ausgangsdaten

- Gemäß Vorgaben war es notwendig, die Ausgangsdaten zu aufbereiten. Zu diesem Zweck wurde ein Datenaufbereitungsschema eingeführt, welches auch benutzt wurde, um – mit Hilfe Tabellen und Views – die Daten an die XML anzupassen:
  - Die hierarchiebildenden Fremdschlüssel wurden zusätzlich mitgeführt.
  - Attributnamen wurden an die XML Namen angeglichen.
  - Datentypen und deren Darstellungsformat wurden an das XML Format weitestgehend angepasst.
    - Z. B., die numerischen Datentypen wurden in Zeichenstrings konvertiert und mit führenden Nullen aufgefüllt etc.
  - Einige Teile wurden zusammengefügt und dadurch denormalisiert.
- Diese Hilfsmaßnahmen sind nicht notwendig. Es stellte sich jedoch schnell heraus, dass diese XML-nahe Datenaufbereitung sehr vorteilhaft für die Fehlersuche ist.

XMLTYPE View: relational-XML Mapping

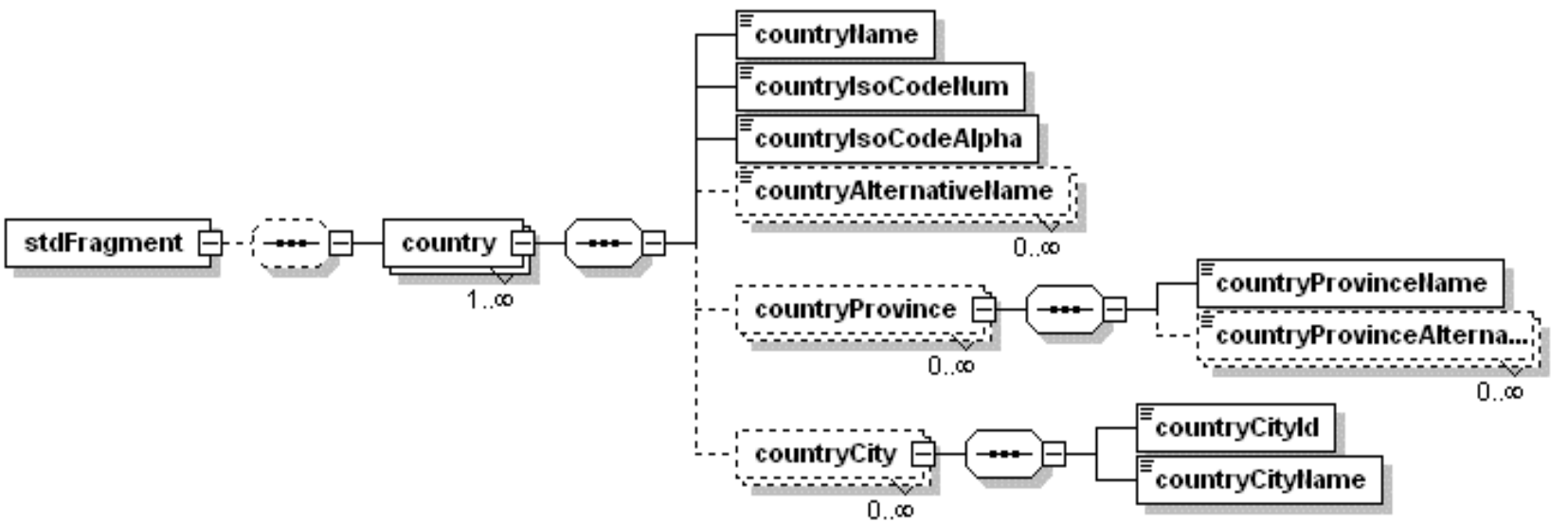
- Die zentrale Stelle des Vorhabens ist der Aufbau der XMLTYPE Views, die für das Mapping der Tabellen und Tabellenattribute auf die XML Attribute sorgen.
- Ein XMLTYPE View verwendet die Oracle SQL/XML sog. „Publishing Functions“:
  - XMLELEMENT: kreiert ein XML-Element. Die XMLELEMENT-Funktionen können ineinander verschachtelt werden und eine Hierarchie bilden.
  - XMLFORERST: produziert einen „Wald“ von XML-Elementen (eine Vielzahl der Stammknoten auf der gleichen Hierarchieebene).
  - XMLAGG: erzeugt einen Wald von XML-Elementen aus einer Sammlung von XML-Werten.
- Die richtige Anwendung dieser Funktionen stellt sicher, dass ein SELECT aus diesem View die korrekt strukturierten XML Daten liefert.
  - Das vollständige Mapping ist statisch als „View“ definiert. Zur Laufzeit finden keine programmatischen Umwandlungen etc. statt.

Vereinfachtes Beispiel

- Ein einfaches Schema mit fünf Tabellen, "country", "countryAlternative", "countryProvince", "countryCity" und "countryProvinceAlternative ".
- Die Beschreibung dieser Tabellen: Tabellenstruktur .
- Die zu erstellende XML Datei ist durch die folgende DTD Datei definiert:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT stdFragment (country+)?>
<!ELEMENT country (countryName, countryIsoCodeNum, countryIsoCodeAlpha,
                   countryAlternativeName*, countryProvince*, countryCity*)>
<!ELEMENT countryName (#PCDATA)>
<!ELEMENT countryIsoCodeNum (#PCDATA)>
<!ELEMENT countryIsoCodeAlpha (#PCDATA)>
<!ELEMENT countryAlternativeName (#PCDATA)>
<!ELEMENT countryProvince (countryProvinceName,
                           countryProvinceAlternativeName*)>
<!ELEMENT countryProvinceName (#PCDATA)>
<!ELEMENT countryProvinceAlternativeName (#PCDATA)>
<!ELEMENT countryCity (countryCityId, countryCityName)>
<!ELEMENT countryCityId (#PCDATA)>
<!ELEMENT countryCityName (#PCDATA)>
```

XML Struktur: grafische Darstellung



Beispiel von XMLType View

```
CREATE OR REPLACE VIEW COUNTRY_XML_VIEW OF XMLTYPE
WITH OBJECT IDENTIFIER
(extract(OBJECT_VALUE, '/country/aus_id/text()').getnumberval()) AS
SELECT XMLELEMENT( "country„
    , XMLFOREST( a."countryName„
                , a."countryIsoCodeNum"
                , a."countryIsoCodeAlpha")
    , ( SELECT XMLAGG(XMLFOREST( s."countryAlternativeName"))
        FROM "countryAlternative" s WHERE s.aus_id = a.aus_id)
    , ( SELECT XMLAGG( XMLELEMENT( "countryProvince"
                                  , XMLFOREST( p."countryProvinceName")
                                  , ( SELECT XMLAGG( XMLFOREST(
                                                pa."countryProvinceAlternativeName"))
                                      FROM "countryProvinceAlternative" pa
                                         WHERE pa.apr_id = p.apr_id)
      )) FROM "countryProvince" p WHERE p.aus_id = a.aus_id)
    , ( SELECT XMLAGG( XMLELEMENT( "countryCity", XMLFOREST( cc."countryCityId"
                                                           , cc."countryCityName")
      )) FROM "countryCity" cc WHERE cc.aus_id = a.aus_id)
) FROM "country" a ORDER BY "countryName";
```

1) XML aus relationalen Daten

- Die Tabelle XML_DATA mit einem XMLTYPE Attribut:

```
CREATE TABLE XML_DATA ( name VARCHAR2(255), xml XMLTYPE );
```

- Mit Hilfe eines SELECT Befehls wird diese Tabelle mit den Daten aus dem XMLTYPE-View geladen. Die XML Daten gehen in das Attribut XML:

```
INSERT INTO xml_data (name, xml) SELECT
   cv.extract('/country/countryName/text()').getstringval(),
   VALUE(cv) FROM country_xml_view cv;
```

- Die vollständige und korrekt formatierte XML Datei wird als ein CLOB Attribut in der Tabelle CLOB_XML_DATA gespeichert:

```
CREATE TABLE clob_xml_data ( xml CLOB );
```

- Der entsprechende SELECT- Befehl sieht folgendermaßen aus:

2) Fertige XML Datei in CLOB

```
INSERT INTO clob_xml_data (xml)
SELECT '<?xml version="1.0"?>'||chr(10)||                                    ← (2)
       '<!DOCTYPE stdFragment SYSTEM "stdFragment.dtd">'||chr(10)||          ← (2)
       '<stdFragment>'||chr(10)||                                            ← (3)
       '<!-- all countries -->'||chr(10)||                                   ← (4)
       xmlagg(XML).getClobVal()||                                            ← (1)
       '</stdFragment>'                                                      ← (3)
       FROM xml_data ORDER BY name;
```

- Im SELECT-Befehl werden gleichzeitig mehrere Aktionen unternommen:
  - Die SQL Funktion XMLAGG verwandelt eine Sammlung der XML Elemente in einen XML Wald (1).
  - Gleichzeitig wird mit Hilfe der „getClobVal()“ Methode (des XMLTyps) das XML Ergebnis in ein CLOB umgewandelt (1).
  - Die restlichen Teile des SELECT-Befehls liefern den XML Prolog (2), das oberste Element (3) sowie den Kommentar (4).

3) Abschluss der XML Generierung

- Zuletzt werden die in CLOBs gespeicherten XML Dateien als Betriebssystemdateien ausgegeben.
- Dabei werden auch – im Sinne des XML Standards – die Umlaute, „ß“ etc. entsprechend umgewandelt.
- Die fertige Beispieldatei (die realen XML Dateien sind wesentlich komplexer und länger) sieht folgendermaßen aus: XmlspyAuslandXml .
- Die oben gezeichneten Schritte beschreiben vollständig das Prinzip des realisierten Anwendungssystems.
- Im folgenden wird das realisierte System kurz vorgestellt.

Datenquelle

- Das Arbeits- und Aufbereitungsschema besteht aus ca. 90 Tabellen und ca. 170 Views, die aus ca. 150 Tabellen des Ausgangsschemas gespeist sind.
  - Die Anzahl der Zeilen pro Tabelle schwankt zwischen einigen 1.000 und weit über 1.000.000.
  - Die umfangreichste Datenlieferung besteht aus knapp über 100 XML Dateien mit Gesamtvolumen von ca. 2,2 GB.
  - Eine typische XML Datei hat Volumen von ca. 20 MB, das maximale Datenvolumen einer Datei liegt bei ca. 55 MB.
- Auch die verwendeten XMLTYPE Views sind etwas komplexer als im Beispiel: XmlTypeView.

PL/SQL Programme und sonst. Systemmerkmale

- Das gesamte System ist in PL/SQL programmiert.
  - Eine Ausnahme sind die Verzeichnismanipulationsfunktionen wie z. B. „makedir“, die in Java programmiert wurden. Sie wurden in PL/SQL Programme integriert.
- Für die Bedienung des Systems wurde mit Hilfe von „Application Express“ eine Bedienoberfläche erstellt.
- Ein besonderer Merkmal des Systems ist, dass es sich um eine produktionsnahe Anwendung handelt.
  - Die Hardware steht nicht im Rechenzentrum, sondern in einer Produktionsstätte.
  - Das System wird ausschließlich remote bedient und gewartet.
  - Seit der Produktionsaufnahme läuft das System praktisch wartungsfrei.

Systemsteuerung

- Die interne Systemsteuerung läuft vollständig über die Oracle Jobs und Jobqueues (PL/SQL Package „DBMS_SCHEDULER“).
- Es besteht auch die Möglichkeit, die Aufgaben parallel durchzuführen. Diese kam jedoch wegen der 2 GB Prozessraumgrenze unter 32-Bit Windows nicht zum Einsatz.
  - Parallelisiert wurden zunächst nur die Aufgaben, die nicht viel Oracle Prozessspeicher verlangen, wie z. B. das Spooling der fertigen CLOBs in die XML Dateien.
- Auch die Prozesse, die direkt aus der Bedienoberfläche gestartet werden, laufen letztendlich asynchron unter Kontrolle des Oracle Schedulers.
  - Über die Bedienoberfläche werden nur die Jobparameter eingegeben sowie, per Scheduleraufruf, die Prozesse gestartet.
- Der Prozessablauf wird sowohl in der Datenbank als auch in Textdateien protokolliert.

Probleme und Beobachtungen

- Der XML Generierungsvorgang verbraucht sehr viel Prozessspeicher. In der Oracle 10g Rel. 1 hatten wir damit folgendes Problem:
  - *ORA-04030: out of process memory when trying to allocate 66060 bytes (pga heap,kllcqgf:kllsltba)*. Dies passierte im SELECT aus XMLTYPE View.
  - Der Hintergrund ist, dass die Intel 32 Bit Prozessoren nur einen 2 GB Adressraum pro Prozess erlauben. Damit ist auch die Oracle maximale Taskgröße auf 2 GB beschränkt. Der Versuch, diese Grenze zu überschreiten, führt dann zum o. g. Fehler.
  - Wir haben einen Metalink TAR geöffnet, dies aber nicht weiterverfolgt, weil
- es in Oracle 10g Rel. 2 dieses Problem nicht mehr gab.
  - Allerdings kann auch in diesem Release eine falsche PGA Konfiguration den ORA-04030 erzeugen.

Fazit

- Oracle XML DB hat den „structure clash“ zwischen der relationalen und der hierarchischen Datensicht wesentlich gemildert:
  - Es ist damit **sehr einfach**, auch sehr komplex strukturierte XML Daten zu generieren.
  - Außer PL/SQL und einigen elementaren Begriffen aus der XML Welt sind dazu keine weiteren Kenntnisse notwendig.
- Allerdings kann die Vielfalt der neuen Begriffe, Tools und Möglichkeiten auch den erfahrenen Oracle Anwendungsentwickler schnell verwirren, zumal die sehr umfangreiche Dokumentation eher die Einzelaspekte und seltener das Gesamtbild vermittelt.
- Vielen Dank für Ihre Aufmerksamkeit. Fragen?