XML und Datenbanken

Stefan Brass
Martin-Luther-Universität Halle-Wittenberg

(Professor für Datenbanken, Certified Oracle8 DBA,
IBM Certified Advanced DBA: DB2 V8.1)

Forschungsgebiete: Deduktive Datenbanken,
Erkennung semantischer Fehler in SQL

Guten Abend. Meine Name ist Stefan Brass, ich vertrete den Bereich Datenbanken am Institut für Informatik.

Ich möchte heute über die Beziehung zwischen XML und Datenbanken sprechen. XML hat sich ja mit einer beeindruckenden Geschwindigkeit entwickelt. So oft, wie man davon hört, könnte man manchmal denken, daß es die ganze Informatik umkrempelt, und besonders natürlich den Datenbank-Bereich.

Ich glaube schon, daß XML einige interessante Aspekte hat, dies es erlauben, über Datenbanken vielleicht noch allgemeiner nachzudenken, und den Horizont zu erweitern. Lange Zeit waren ja die relationalen Datenbanken alles, was man kennen mußte. Das ist heute nicht mehr so. Relationale Datenbanken sind nachwievor die wichtigste Grundlage, aber es gibt ja inzwischen viele objektorientierte Erweiterungen, und die Bedeutung von XML als Datenaustauschformat ist ja wohl auch ziemlich klar.

Aber wir werden sehen, das XML als Datenbank nun nicht unbedingt alles besser macht, und in mancher Hinsicht auch einen Rückschritt bedeutet.

Inhalt

Wir wollen uns zunächst XML als Datenmodell anschauen.

Im zweiten Teil möchte ich dann auf XML Schema eingehen, was ja eingeführt wurde, weil den Datenbanklern die DTDs, die Dokumenttypdefinitionen, die ursprünglich mit XML mitkamen, nicht ausreichten. Man muß ja bedenken, daß XML ursprünglich einfach eine abgemagerte Version von SGML war, und somit zur Strukturierung von Dokumenten gedacht war. Früher haben die Datenbank- und die Dokumentverarbeitungs-Community auch kaum mit einander geredet. Das hat sich durch XML verbessert, weil sich da beide Seiten wiederfinden konnten.

Im dritten Teil wollen wir dann XPath und XQuery als Anfragesprachen für XML-Daten betrachten. Einer der wichtigsten Väter von SQL, Don Chamerlin, hat übrigens auch an der Entwicklung von XQuery mitgewirkt.

Der XML Hype hat auch die großen Datenbank-Hersteller, wie etwa Oracle und IBM DB2 nicht ruhen lassen, sie wollen ja keine Marktanteile abgeben. Sie haben ja schon objektorientierte Konstrukte in die ursprünglich relationalen Datenbanken eingebaut, so daß heute alle großen kommerziellen Datenbanken objektrelational sind, und die rein objektorientierten Datenbanken mit ganz wenigen Ausnahmen vom Markt verschwunden sind. Und so hat man jetzt also auch XML Unterstützung in die SQL-Datenbanken eingebaut, das wollen wir uns dann kurz im letzten Abschnitt anschauen.

Semistrukturierte Daten (1)

- Relationale Daten gelten als stark strukturiert:
  - ◇ Das Schema ist DBMS und Nutzern bekannt, sehr stabil (ändert sich nur minimal).
  - ◇ Die einzelnen Tabelleneinträge sind atomar, Auswertungen ohne manuelle Hilfe möglich.
- Texte (auch Bilder etc.) gelten als unstrukturiert:
  - ◇ DBMS-Sicht: nur Folge von Zeichen/Worten.
  - ◇ Die inhaltlich interessanten Strukturen sind dem DBMS nicht bekannt → Keine Hilfe bei Suche.

    Z.B. Warenbezeichnungen/zugehörige Preise, falls Katalog.

Bevor XML aufkam, gab es ja schon das Web (wenn auch das ja noch sehr neu ist, so seit 1991 or 1993, also ist es jetzt erst ungefährt 15 Jahre alt). Das Web war dann der Datenbankwelt Anregung dafür, semistrukturierte Datenmodelle zu strukturieren. Semistrukturiert bedeutet, daß es irgendwo zwischen den stark strukturierten relationalen Daten und unstrukturiertem Text liegt.

Bei relationalen Dankenbanken ist das Schema, also die Struktur der Daten, welche Tabellen es gibt und welche Spalten die haben, dem DBMS und den Nutzern bekannt. Das Schema ist auch sehr stabil, es ändert sich auch nur ganz selten und nur unwesentlich.

Ein wichtiger Punkt ist auch, daß die einzelnen Tabelleneinträge atomar sind, oder bei neueren Datenbanken jedenfalls klar definierte atomare Komponenten haben. Datenbanken können im wesentlichen nur Daten filtern und zusammenfassen, z.B. Summen berechnen. Die Aufspaltung, um etwa aus Zeichenketten bestimmte Informationen herauszusiehen, ist nur in einfachen Fällen möglich, und auch dann ist häufig nicht garantiert, daß alle Zeichenketten auch der erwarteten Struktur entsprechen, so daß dann wenigstens eine manuelle Nachbearbeitung nötig ist.

Das bringt uns jetzt zum anderen Ende des Spektrums, dem reinen Text, der aus Datenbanksicht im wesentlichen unstrukturiert ist, oder höchstens als Folge von Worten aufgefasst werden kann. Da das DBMS die Strukturen nicht kennt, kann es auch keine Hilfe bei der Suche geben. Man kann höchstens nach bestimmten Worten suchen. Nehmen wir an, der Text wäre ein Buchkatalog, und wir würden darin nach Goethe suchen. Dann würden wir sowohl Bücher von Goethe als auch Bücher über Goethe bekommen. Wenn wir jetzt noch davon das Buch mit minimalem Preis herausfinden wollen, wird es erst recht kompliziert. Wir müssen dabei ja unter anderem die Beziehung zwischen dem Titel und dem zugehörigen Preis herausfinden.

Zum Beispiel war es ein Erfolg der Braunschweiger Datenbankgruppe, daß sie aus wissenschaftlichen Arbeiten im Bereich der Bioinformatik, die nur als PDF vorlagen, die Beschriftung zu Bildern automatisch extrahieren konnten. Selbst das war recht kompliziert.

Semistrukturierte Daten (2)

- Mit dem Web kamen semistrukturierte Daten auf:
  - ◇ Zum Teil sind inhaltlich interessante Strukturen mit Tags markiert, zum Teil einfach Text.
  - ◇ Daten entstehen häufig durch Integration autonomer Quellen.
  - ◇ Mit den Tags wird häufig sehr frei umgegangen, die Strukturen sind unregelmäßig.
  - ◇ Die Tags (Elementtypen) / das Schema sind nicht unbedingt vorab bekannt.
  - ◇ Das Schema ist ständiger Änderung unterworfen.

Mit dem Web kamen dann semistrukturierten Daten: Durch die Tags hatte man ja schon etwas Struktur, zum Teil konnte man sich durch die regelmäßige Struktur mancher Textteile (wenn sie ursprünglich aus einer relationalen Datenbank stammten) auch noch etwas mehr Struktur beschaffen.

Typisch für das Web-Zeitalter ist aber auch, daß man Daten aus mehreren autonomen Quellen benutzen möchte. Nun sind die aber alle ein bißchen unterschiedlich strukturiert, und man kann auch nicht verhindern, daß ein Autor einer sochen Quelle seine Daten gelegentlich umstrukturiert. Dadurch sind die Strukturen eher unregelmäßig, und das Schema ist nicht vorab bekannt.

Das ist auch ein ganz wichtiger Unterschied zwischen XML und relationalen Datenbanken, daß man XML auch ohne Schema nutzen kann. Man kann einfach Daten irgendwie strukturiert aufschreiben.

In relationalen Datenbanken muß man dagegen zunächst ein Schema definieren, bevor man die ersten Daten einfügen kann. Die spätere Änderung es Schema ist dann relativ aufwändig. Das hat durchaus auch den Vorteil, daß man über das Schema erstmal gründlich nachdenkt.

Bei XML wird dagegen häufig mit den Tags sehr frei umgegangen: Wenn Sie kein Schema haben, müssen Sie es ja garnicht merken, daß Sie die Strukturen nicht so ganz konsistent und einheitlich aufbauen.

Diese Charakteristika von semistrukturierten Datenbanken haben auch Einfluss auf die Anfragesprache. Zum Beispiel kann man sich bei relationalen Datenbanken in den Anfragen nur auf bekannte Spalten fester Tabellen beziehen. Relationale Datenbanken bieten zwar ein Data Dictionary, aus dem man sich diese Information verschaffen könnte, aber das ist jedenfalls ein getrennter Schritt. Man würde zuerst die Ergebnisse der Data Dictionary Abfrage benötigen, um dann die eigentliche Anfrage an die Daten formulieren zu können.

Bei XML-Daten würde es dagegen Sinn machen, daß man sich z.B. die Inhalte aller Tags anzeigen lassen möchte, die auf "name" enden, und das an beliebiger Stelle im Dokument.

Was bei relationalen Datenbanken praktisch überhaupt nicht geht, wäre die Anfragerichtung von den Daten zu den Meta-Daten. Z.B. möchte man vielleicht alle Spalten finden, in denen das Wort "Goethe" irgendwo in den Daten vorkommt.

Semistrukturierte Daten (3)

- In Anfragen an relationale Datenbanken kann man sich nur auf bekannte Spalten/Tabellen beziehen.

  Notfalls kann man das Data Dictionary abfragen, um sich die Schema-Information zu beschaffen. Dies ist aber ein getrennter Schritt.

- Bei XML-Daten macht es dagegen Sinn, daß man sich den Inhalt von beliebigen Tags, die auf "name" enden, anzeigen lassen möchte.
- Mit SQL könnte man auch nicht nach allen Tabellen und Spalten fragen, in deren Inhalt (den Daten) ein gegebenes Wort vorkommt.

  In einer XML-Datei könnte man mit einem Texteditor danach suchen.

In XML würde man erwarten, daß mindestens alles geht, was man auch mit der Suchfunktion eines besseren Texteditors machen könnte.

Ich kann es mir aber an dieser Stelle auch nicht verkneifen, nochmal etwas Werbung für deduktive Datenbanken zu machen. Das ist ja mein Lieblings-Forschungsthema. Wir haben ja gesehen, daß die Struktur von XML-Dateien eher unregelmäßig und uneinheitlich ist. Um jetzt doch relativ einfach Anfragen stellen zu können, braucht man ein mächtiges Sichtenkonzept. Das wäre eine der Stärken von deduktiven Datenbanken. Außerdem kann man dort auch kontextfreie Grammatikregeln formulieren und auf diese Art auch Textteile analysieren, die nicht durch Markup (Tags) strukturiert sind.

Beispiel

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Feuerwerksartikel>
    <Bombenrohr Herst="Diamond" Name="Silver Star">
        <Angebot Händler="Röder" Preis="5.49"/>
        <Bew Jahr="2005">Goldpalme! Toll.</Bew>
        <Bew Jahr="2006">Na ja, geht so.</Bew>
    </Bombenrohr>
    <Batterie Herst="Weco" Name="Tanz der Vampire"
            Schuss="12" Hoehe="45" Dauer="30">
        <Angebot Händler="Roeder" Preis="8.50"/>
        <Angebot Händler="Preisw-FW" Preis="7.69"/>
        <Bew Jahr="2006">Rote Blinker. Hübsch.</Bew>
    </Batterie>
</Feuerwerksartikel>
```

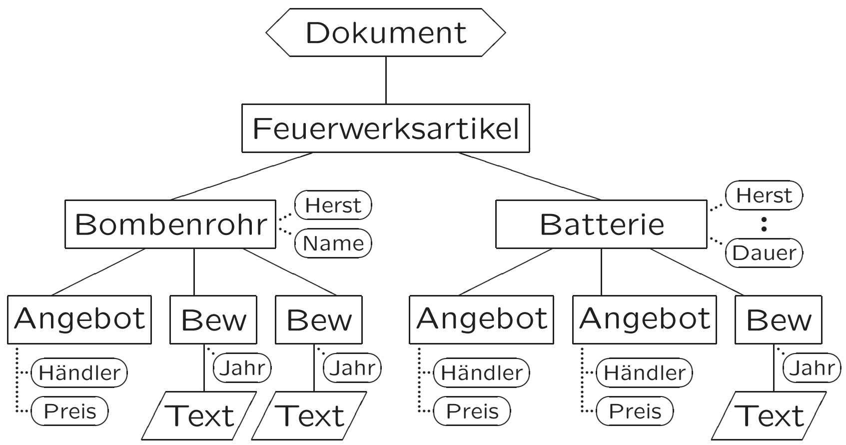

Nach dieser ganzen Vorrede wollen wir aber ein Beispiel betrachten, das man genausogut in einer relationalen Datenbank ablegen könnte. Sonst würden wir ja Äpfel mit Birnen vergleichen. Ich denke aber, es ist doch wichtig, daß XML nicht nur für solche stark strukturierten Datenbestände gedacht ist, sondern das ganze Spektrum bis zu reinem Text nur mit Formatieranweisungen abdecken kann.

Es sind zwar noch 178 Tage bis Silvester, aber Feuerwerk macht mir Spaß, und es ist vielleicht mal ein etwas interessanteres Beispiel als die Punkte-Datenbank, die ich sonst immer verwende. Hier sind also Daten von zwei Feuerwerksartikeln abgespeichert. Das erste ist ein Bombenrohr, was nur eine Kugelbombe hochschießt, die oben dann in eine Goldpalme zerplatzt, Das zweite ist eine Batterie mit zwölf Schüssen, die unter anderem rote Blinksterne an den Himmel zaubern.

Korrekterweise sollte ich noch sagen, daß Bombenrohre im Profi-Bereich eine Spezialform von römischen Lichtern sind, die aus einem Rohr mehrere Bomben hintereinander hochschießen. Aber da es das für Normalsterbliche wohl nicht gibt, ist es in den Online-Shops die übliche Bezeichnung für fertig verladene Einweg-Mörserrohre. Die Batterien heißen im Profi-Slang übrigens Cakeboxen.

In die Basisinformation zu den Feuerwerksartikeln geschachtelt haben wir nun Information über Angebote bei verschiedenen Händlern, da kann es mehrere geben. Ebenso kann es auch mehrere Bewertungen geben, meine Eindrücke vom Silvesterfeuerwerk in verschiedenen Jahren. Interessanterweise ist der Silver Star eine Goldpalme, 2005 hat es mich begeistert, 2006 hat es mich nicht mehr so beeindruckt. Das hängt vielleicht an der Erwartungshaltung, mit der man da rangeht, und daran, was man vorher schon gesehen hat, vielleicht hat auch der Wind gestört.

Interne Darstellung: XDM



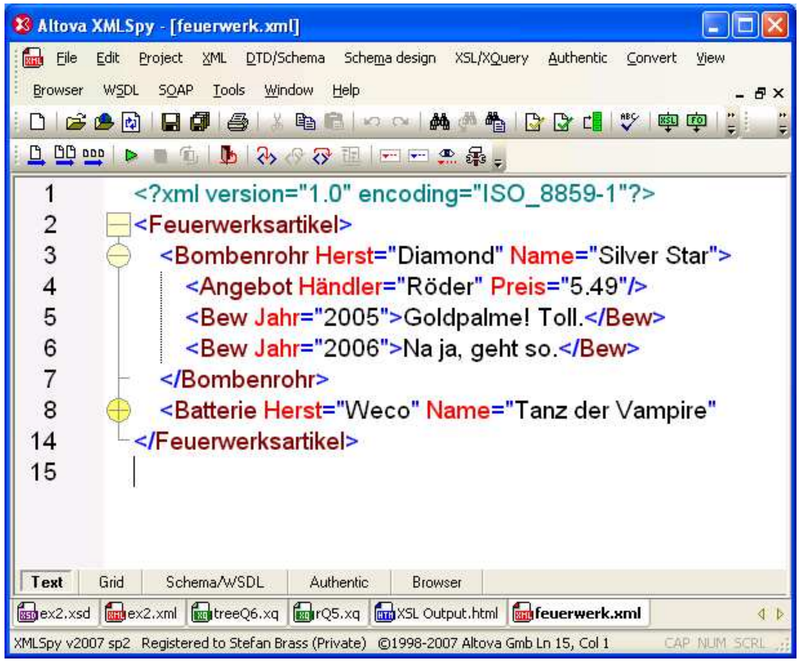
Nun sieht so eine XML-Datei als einfacher Text ja nicht besonders übersichtlich aus, hier ist die Struktur nochmal im XPath/XQuery Datenmodell (kurz XDM) gezeigt.

Durch die Schachtelung entsteht ja eine Baumstruktur. Neben dem Dokumentknoten als Wurzel haben wir dann auf oberster Ebene das Element "Feuerwerksartikel", das die ganze Liste von Einzelartikeln enthält. Für die Artikel habe ich je nach Kategorie verschiedene Elementtypen vorgesehen, da für verschiedene Kategorien eben etwas unterschiedliche Daten benötigt werden. In das Element für einen einzelnen Artikel sind dann Angebote und Bewertungen geschachtelt. Die kleinen blauen Knoten sind Attribute, bei den Bewertungen gibt es auch einen Text als Inhalt des Elementes.

Wenn man will, kann man XML auch verstehen als ein Objektmodell, mit dem man diese Datenstrukturen beschreiben kann, sowie einen Parser, der die Datenstrukturen aufbauen kann aus einer textuellen Beschreibung, und einen Unparser, der umgekehrt die Datenstrukturen wieder serialisieren, also als Text ausgeben kann.

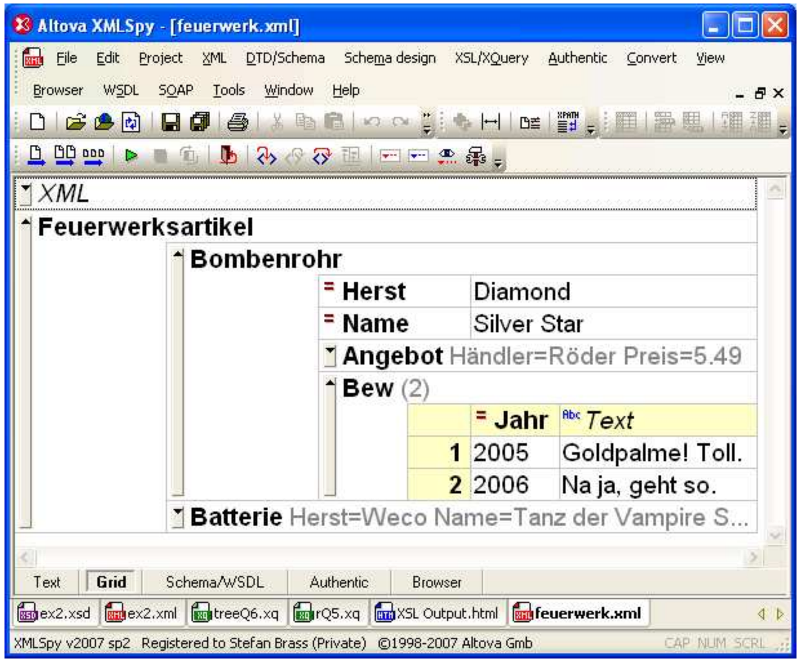
Manches passt gar nicht so gut zusammen, und das würde man heute wohl anders machen. Aber XML sollte ja eine Teilmenge von SGML sein, insofern ist es gar nicht so neu. SGML wurde 1986 standardisiert, also deutlich bevor es das Web gab. Dadurch war dann manches vorgegeben. Außerdem ist XML ja nicht ein Standard sondern viele Standards, die zu unterschiedlichen Zeiten von unterschiedlichen Leuten entwickelt wurden.

Nun ja, wer weiß. Vielleicht kann ich ja im nächsten Jahr über (m)einen Vorschlag zur Verbesserung der Situation berichten.



Die Baumstruktur von XML Dokumenten kann man natürlich unterschiedlich visualisieren, was dann auch in verschiedenen speziellen XML-Editoren ausgenutzt wird. Dies wäre etwa die Text-Sicht von XML Spy. Das ist noch recht klassisch, das Markup wird unterschiedlich farbig dargestellt, man kann Teilbäume ausblenden, und die Schachtelungstiefe wird mit den dünnen Linien angezeigt.

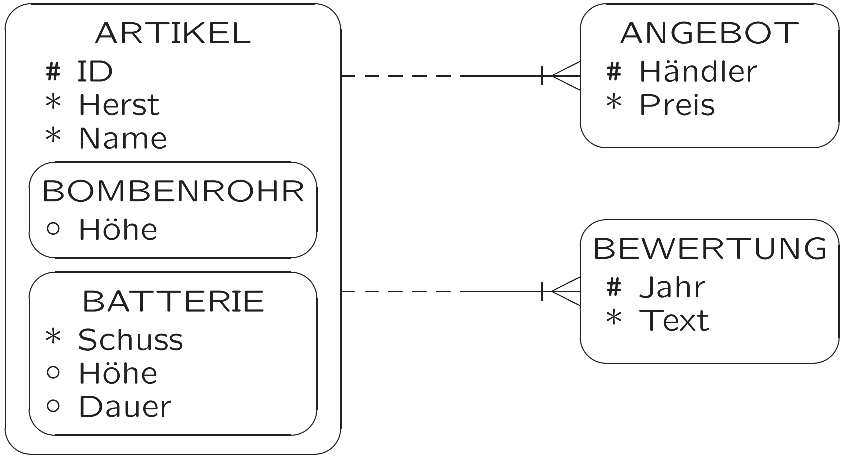
Diese XML-Entwicklungsumgebung ist übrigens recht teuer, die billigste Lizenz kostet 500 Euro. Ich hatte davon auch nur eine Testlizenz, als ich diesen Screenshot gemacht habe. Da viel Werbung dafür gemacht wird, wollte ich das mal ausprobieren. Ansonsten ist viel XML Software ja kostenlos. Tatsächlich kann man den XML Schema Validator und den XQuery Auswerter, der in XMLSpy eingebaut ist, auch kostenlos bekommen. Nur für den Editor und einige Zusatzwerkzeuge muß man bezahlen.



Hier noch einmal eine ganz andere Visualisierung der gleichen Daten. Dies wird die "Grid-View" genannt. Zum Beispiel wird der Elementtyp hier nur einmal angezeigt, während er in der eigentlichen XML-Syntax sowohl im Start-Tag als auch im End-Tag stehen muß. Das ist ja redundant.

Interessant ist hier auch, daß für Bewertung eine tabellarische Syntax gewählt wird, da hat man sich eben doch am relationalen Modell orientiert.

Klassischer DB-Entwurf



Wir wollen diese XML-Datei jetzt ja mit einer relationalen Datenbank vergleichen. Für eine relationale Datenbank würden wir wahrscheinlich zuerst einen Datenbank-Entwurf im ER-Modell machen.

Da sehen wir dann, daß wir eigentlich eine Oberklasse für Artikel haben, mit Bombenrohr und Batterie als zwei Beispiele von Unterklassen, die jeweils spezielle Attribute haben. Natürlich gibt es in der Realität weit mehr Arten von Feuerwerksartikeln, aber das ist ja nicht das Thema hier.

Angebot und Bewertung sind dann jeweils schwache Entities, die über eins-zu-viele Beziehungen mit dem Artikel verknüpft sind. Schwaches Entity bedeutet, daß die Schlüsselattribute nur im Kontext eines anderen Entities eindeutig sind, das passt sehr schön zu der Schachtelung.

Relationale Datenbank

ARTIKEL						
ID	HERST	NAME	TYP	HOEHE	SCHUSS	DAUER
1	Diamond	Silver Star	BO	60		
2	Weco	Tanz der Vampire	BA	45	12	30

ANGEBOT		
ID	HAENDLER	PREIS
1	Röder	5.49
2	Röder	8.50
2	Preisw.F.	7.69

BEWERTUNG		
ID	JAHR	TEXT
1	2005	Goldpalme! Toll.
1	2006	Na ja, geht so.
2	2006	Rote Blinker. Hübsch.

Das klassische relationale Modell hat ja nun leider keine Subklassen, das ist etwas, was bei den objektrelationalen Datenbanken dazu kommen würde. Ich habe die Subklassen hier über den Typ dargestellt, und Nullwerte für die nicht zutreffenden Spalten eingesetzt.

XML als Datenbank (1)

Vorteile:

- Es geht viel schneller, Daten im XML-Format zu erfassen, als eine relationale Datenbank anzulegen.

  Solange die Datensammlung klein ist, reicht ein Texteditor. Trotzdem können die Daten so strukturiert sein, daß man mit XQuery alles an Anfragen/Auswertungen berechnen könnte, was auch mit einem relationalen DBMS möglich wäre. Das Risiko ist allerdings, daß die Daten im Laufe der Zeit immer schlechter strukturiert werden (wenn man nicht bewußt eine DTD/ein Schema entworfen hat und die Einhaltung erzwingt — dann ist der Zeitvorteil aber nicht mehr klar).

- Anerkanntes Datenaustausch-Format.

  Die Dateien meiner relationalen Datenbank wären höchstens für jemanden lesbar, der genau das gleiche DBMS hat.

XML als Datenbank (2)

Vorteile, Forts.:

- XML unterstützt komplex strukturierte Objekte.

  Am relationalen Modell wird kritisiert, daß man die Objekte zur Speicherung in einfache Tupel zerlegen muß.

- Fließender Übergang von mehr datenzentrierten zu mehr dokumentenzentrierten Strukturen.

  Die Zeichenketten, die man in relationalen Datenbanken speichern kann, haben aus Sicht des relationalen DBMS keine weitere Struktur. Darüber hinaus haben relationale DBMS bei sehr langen Zeichenketten (Dokumenten) häufig starke Einschränkungen (z.B. keine Substring-Suche, nur reiner Datenspeicher).

- Viele freie Werkzeuge.

XML als Datenbank (3)

Nachteile:

- Die Struktur relationaler Datenbanken ist viel einfacher.
- XML ist recht ähnlich zum hierarchischen Datenmodell, das die Daten ebenfalls in Bäumen strukturierte.

  Der bekannteste Vertreter des hierarchischen Modell ist das System IMS von IBM. Es erschien 1968. Das hierarchische Modell war zwei Generationen vor dem relationalen Modell (das Netzwerk-Modell war noch dazwischen). Fairerweise muß man aber sagen, daß das hierarchische Modell keine deklarative Anfragesprache hatte, die für XML mit XQuery heute zur Verfügung steht.

Heute wird vieles von der schönen Einfachheit, die das relationale Modell gebracht hat, kaputt gemacht: Alles wird immer komplexer (meiner persönlichen Meinung nach häufig ohne rechte Not).

XML als Datenbank (4)

Nachteile, Forts.:

- Solange man nur mit einem Texteditor arbeitet, fehlen wichtige DBMS-Eigenschaften:
  - ◇ Synchronisation paralleler Zugriffe (Sperren)
  - ◇ Unterschiedliche Zugriffsrechte auf unterschiedliche Teile des Dokuments.
  - ◇ Verschiedene Sichten auf die gleichen Daten
  - ◇ Verwaltung von Datenbeständen, die wesentlich größer als der Hauptspeicher sind.
  - ◇ Recovery bei Plattenschäden.

Inhalt

Zweck eines Schemas (1)

- Bei relationalen Datenbanken legt das Schema die Datenstrukturen zur Abspeicherung der Daten fest, die Daten wären ohne Schema nicht interpretierbar.

  Diese Funktion hat das Schema bei XML nicht, da die Daten dort selbstbeschreibend sind. XML kann ohne Schema verwendet werden.

- Verhinderung von (bestimmten) Eingabefehlern

  Natürlich können nur wirklich sinnlose Daten erkannt werden.

- Langfristige Sicherung einer konsistenten Struktur

  Man kann nicht für gerade einzugebene Daten spontan ein neues "Tag" verwenden, sondern muß bewußt zuerst das Schema ändern.

- "Vertrag" zwischen Partnern beim Datenaustausch

Zweck eines Schemas (2)

- Definition von Bezeichnern (Einstiegspunkten), die in Programmen benötigt werden.
- Programme werden einfacher, wenn sie sich auf bestimmte Strukturen verlassen können, und nicht den allgemeinsten Fall behandeln müssen.
- Dokumentation/Beschreibung der Datenstrukturen
- Platz für Parameter allgemeiner Werkzeuge

  Man kann in XML Schema beliebige Anwendungsinformation hinterlegen. Dies könnte z.B. für Werkzeuge nützlich sein, die zwischen XML-Datei und relationaler Datenbank transformieren, oder die formularbasierte Eingabemasken für XML-Daten erstellen.

Zweck eines Schemas (3)

- Anreicherung der Daten mit Defaultwerten sowie Typ-Information.

  Der Standard spricht vom "Post-Schema-Validation-Infoset" (PSVI), das eine ganze Reihe von Zusatz-Informationen zum XML Infoset hat. Durch die automatische Eintragung von Default-Werten braucht die Anwendung nicht den Fall eines fehlenden Wertes zu behandeln.

- Normalisierung von Attribut-/Datenwerten.

  Die Anwendung muß so weniger Varianten in der Eingabe behandeln.

- Statische Typprüfung von Stylesheets/Anfragen

  Wenn dagegen Fehler erst dynamisch bei der Ausführung bemerkt werden können, ist es immer möglich, daß der Fehler beim Testen nicht auftritt, bei der ersten offiziellen Präsentation dagegen schon.

XML Schema (1)

- Im XML Standard sind DTDs als Mittel zur Strukturbeschreibung von XML-Daten definiert.

  DTD: "Dokument Typ Definition"

- Diese sind von SGML übernommen, und eignen sich zwar für Dokumente, sind aber aus Datenbank-Sicht unzureichend.
- Einige Einschränkungen von DTDs:
  - ◇ Wenig Typ-Information (und nur für Attribute)
  - ◇ Keine Schlüssel/Fremdschlüssel.

    Es gibt zwar `ID`/`IDREF`-Attribute, aber ihr Wertebereich ist zu eingeschränkt und sie beziehen sich global auf die ganze Datei.

XML Schema (2)

- XML Schema ist recht komplex, es bietet u.a.
  - Alle Möglichkeiten von DTDs (außer Entities)
  - Unterstützung von Namensräumen
  - Umfangreiche Auswahl von Basistypen
  - Spezialisierung und Erweiterung zur Definition abgeleiteter Typen
  - Gleichbehandlung von Attributen und Elementen mit einfachem Inhalt
  - Schlüssel, Fremdschlüssel
  - Lokale Elementdeklarationen
  - Substitution von Elementtypen

XML Schema: Beispiel (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:complexType name="AngTyp">
        <xs:attribute name="Händler" type="xs:string"/>
        <xs:attribute name="Preis">
            <xs:simpleType>
                <xs:restriction base="xs:decimal">
                    <xs:fractionDigits value="2"/>
                    <xs:minInclusive value="0"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
```


XML Schema: Beispiel (2)

```
<xs:simpleType name="JahrTyp">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="1900"/>
        <xs:maxExclusive value="3000"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="BewTyp">
    <xs:simpleContent>
        <xs:extension base="xs:string">
            <xs:attribute name="Jahr"
                          type="JahrTyp"/>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
```


XML Schema: Beispiel (3)

```
<xs:complexType name="ArtikelTyp">
    <xs:sequence>
        <xs:element name="Angebot" type="AngTyp"
                maxOccurs="unbounded"/>
        <xs:element name="Bew" type="BewTyp"
                minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="Herst" type="xs:string"
            use="required"/>
    <xs:attribute name="Name"  type="xs:string"
            use="required"/>
</xs:complexType>
```

XML Schema: Beispiel (4)

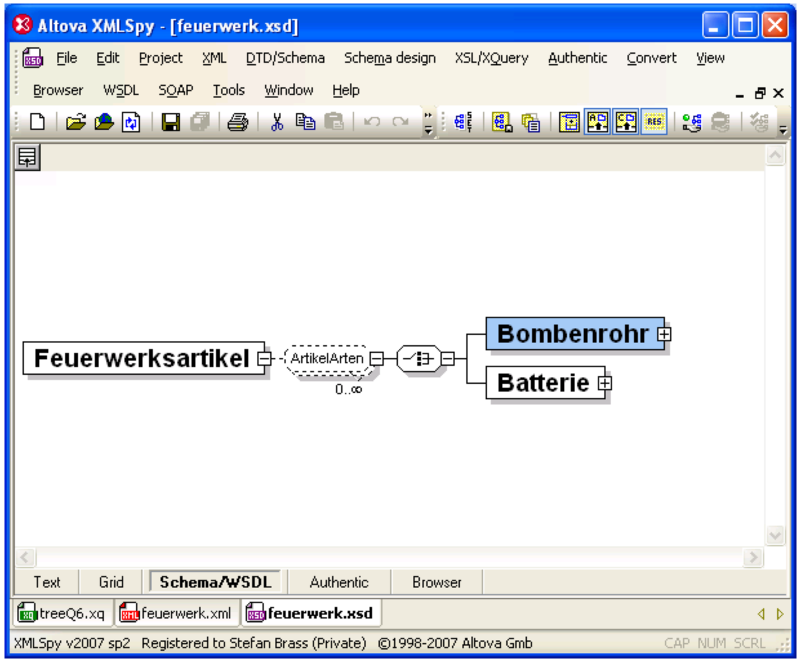
```
<xs:complexType name="BombenrohrTyp">
        <xs:annotation>
            <xs:documentation>
                Bei Klasse II (hier): Einschüsser
            </xs:documentation>
        </xs:annotation>
    <xs:complexContent>
        <xs:extension base="ArtikelTyp">
            <xs:attribute name="Höhe"
                type="xs:nonNegativeInteger"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

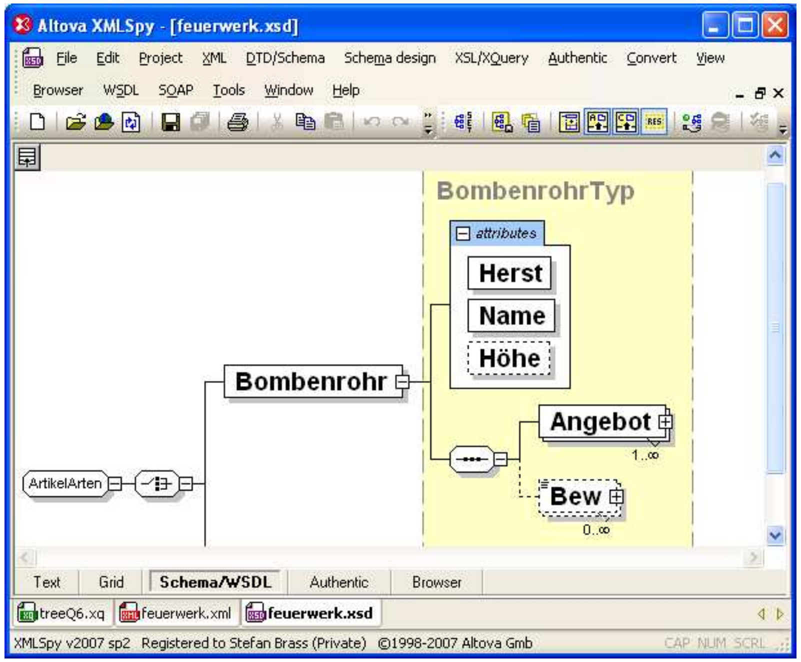

XML Schema: Beispiel (5)

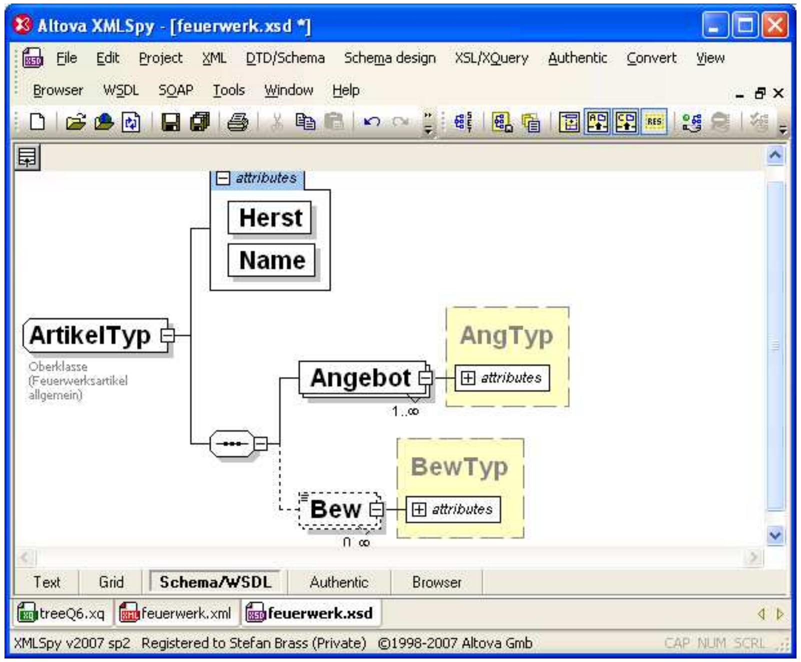
```
<xs:complexType name="BatterieTyp">
    <xs:complexContent>
        <xs:extension base="ArtikelTyp">
            <xs:attribute name="Schuss"
                type="xs:nonNegativeInteger"
                use="required"/>
            <xs:attribute name="Höhe"
                type="xs:nonNegativeInteger"/>
            <xs:attribute name="Dauer"
                type="xs:nonNegativeInteger"/>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

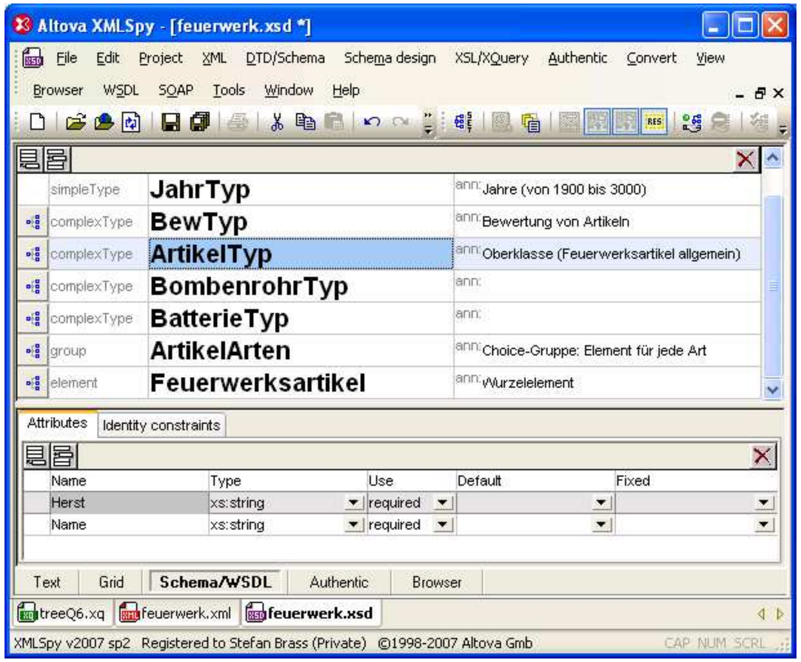

XML Schema: Beispiel (6)

```
    <xs:group name="ArtikelArten">
        <xs:choice>
            <xs:element name="Bombenrohr"
                        type="BombenrohrTyp"/>
            <xs:element name="Batterie"
                        type="BatterieTyp"/>
        </xs:choice>
    </xs:group>
    <xs:element name="Feuerwerksartikel">
        <xs:complexType>
            <xs:group ref="ArtikelArten"
                minOccurs="0" maxOccurs="unbounded"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```


Altova XMLSpy - [feuerwerk.xsd]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View
Browser WSDL SOAP Tools Window Help
BombenrohrTyp
attributes
Herst
Name
Höhe
Bombenrohr
Angebot
1..∞
ArtikelArten
Bew
0..∞
Text Grid Schema/WSDL Authentic Browser
treeQ6.xq feuerwerk.xml feuerwerk.xsd
XMLSpy v2007 sp2 Registered to Stefan Brass (Private) ©1998-2007 Altova Gmb
CAP NUM SCRL


Altova XMLSpy - [feuerwerk.xsd *]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View
Browser WSDL SOAP Tools Window Help
attributes
Herst
Name
ArtikelTyp
Oberklasse (Feuerwerksartikel allgemein)
AngTyp
Angebot
attributes
1..∞
BewTyp
Bew
attributes
0..∞
Text Grid Schema/WSDL Authentic Browser
treeQ6.xq feuerwerk.xml feuerwerk.xsd
XMLSpy v2007 sp2 Registered to Stefan Brass (Private) ©1998-2007 Altova Gmb
CAP NUM SCRL


Altova XMLSpy - [feuerwerk.xsd *]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View
Browser WSDL SOAP Tools Window Help
simpleType JahrTyp ann: Jahre (von 1900 bis 3000)
complexType BewTyp ann: Bewertung von Artikeln
complexType ArtikelTyp ann: Oberklasse (Feuerwerksartikel allgemein)
complexType BombenrohrTyp ann:
complexType BatterieTyp ann:
group ArtikelArten ann: Choice-Gruppe: Element für jede Art
element Feuerwerksartikel ann: Wurzelelement
Attributes Identity constraints
Name Type Use Default Fixed
Herst xs:string required
Name xs:string required
Text Grid Schema/WSDL Authentic Browser
treeQ6.xq feuerwerk.xml feuerwerk.xsd
XMLSpy v2007 sp2 Registered to Stefan Brass (Private) ©1998-2007 Altova Gmb
CAP NUM SCRL

Inhalt

XPath (1)

- XPath ist eine Sprache für einfache Selektionen und Berechnungen auf XML-Bäumen (liefert Folgen von Werten oder Knoten).
- XPath-Ausdrücke sind auch XQuery Anfragen.

  XPath wird aber z.B. auch in XSLT und XPointer eingesetzt.
- Einfache XPath-Ausdrücke sind ähnlich zu Pfad-Ausdrücken im UNIX-Dateisystem (XPath hat aber noch viel mehr Möglichkeiten).
- Beispiel: Liefere (alle) Bombenrohre.

```
/Feuerwerksartikel/Bombenrohr
```


XPath (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Feuerwerksartikel>
    <Bombenrohr Herst="Diamond" Name="Silver Star">
        <Angebot Händler="Röder" Preis="5.49"/>
        <Bew Jahr="2005">Goldpalme! Toll.</Bew>
        <Bew Jahr="2006">Na ja, geht so.</Bew>
    </Bombenrohr>
    <Batterie Herst="Weco" Name="Tanz der Vampire"
            Schuss="12" Hoehe="45" Dauer="30">
        <Angebot Händler="Roeder" Preis="8.50"/>
        <Angebot Händler="Preisw-FW" Preis="7.69"/>
        <Bew Jahr="2006">Rote Blinker. Hübsch.</Bew>
    </Batterie>
</Feuerwerksartikel>
```


XPath (3)

- Beispiel: Liefere die Namen aller Feuerwerksartikel:

```
/Feuerwerksartikel/*/@Name
```

  ◇ `*` passt auf beliebiges Element.

  ◇ `@` markiert Zugriff auf Attribut.
- Es werden die Attribut-Knoten geliefert.

  Umwandlung in ein Dokument mit XQuery s.u.
- Bei manchen XQuery-Implementierungen muß man das Eingabedokument so nennen:

```
doc("E:\feuerwerk.xml")/Feuerwerksartikel/*/@Name
```


XPath (4)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Feuerwerksartikel>
    <Bombenrohr Herst="Diamond" Name="Silver Star">
        <Angebot Händler="Röder" Preis="5.49"/>
        <Bew Jahr="2005">Goldpalme! Toll.</Bew>
        <Bew Jahr="2006">Na ja, geht so.</Bew>
    </Bombenrohr>
    <Batterie Herst="Weco" Name="Tanz der Vampire"
            Schuss="12" Hoehe="45" Dauer="30">
        <Angebot Händler="Roeder" Preis="8.50"/>
        <Angebot Händler="Preisw-FW" Preis="7.69"/>
        <Bew Jahr="2006">Rote Blinker. Hübsch.</Bew>
    </Batterie>
</Feuerwerksartikel>
```

XPath (5)

- Beispiel: Welche Händler bieten das Bombenrohr "Silver Star" an?

  `//Bombenrohr[@Name="Silver Star"]/Angebot/@Händler`

  - `//` sucht überall im Baum
  - `[...]` ist ein zusätzlicher Test (Filter).

```
<Bombenrohr Herst="Diamond" Name="Silver Star">
    <Angebot Händler="Röder" Preis="5.49"/>
    <Bew Jahr="2005">Goldpalme! Toll.</Bew>
    <Bew Jahr="2006">Na ja, geht so.</Bew>
</Bombenrohr>
```


XQuery (1)

- Ein wichtiges XQuery-Konstrukt sind "FLWOR"-Ausdrücke (man spricht es "Flower", obwohl das "`order by`" falsch steht):

```
for $⟨var⟩ in ⟨Ausdruck⟩, ...
let $⟨var⟩ := ⟨Ausdruck⟩, ...
where ⟨Bedingung⟩
order by ⟨Sortierung⟩
return ⟨Ausdruck⟩
```

- Man kann `for` und `let` mehrfach in beliebiger Reihenfolge verwenden. Eins von beiden ist nötig.
- `where` und `order by` sind optional.

XQuery (2)

- Vergleich mit SQL:
  - `for` entspricht `FROM` (Schleife über möglichen Variablenbindungen)
  - `return` entspricht `SELECT`
  - Variablen werden in XQuery mit "`$`" markiert.
- `let` vs. `for`:
  - Bei `let` wird die Sequenz, die Ergebnis des Ausdrucks ist, als Ganzes an die Variable gebunden.
  - Bei `for` wird jeweils ein Element der Sequenz an die Variable gebunden (in einer Schleife).

XQuery (3)

- Ein wichtiges Konzept ist auch die Schachtelung von Ausdrücken:
  - Mit "<" beginnt "literaler Inhalt", der so in die Ausgabe übernommen wird.

    Man will ja XML-Dokumente als Anfrageergebnis konstruieren, dafür möchte man die XML-Syntax verwenden.
  - Darin kann man dann mit `{...}` Teile markieren, die wieder ausgewertet werden sollen.

    Falls man wirklich geschweifte Klammern haben will, muß man sie ggf. verdoppeln.

XQuery (4)

- Beispiel (Namen von Feuerwerksartikeln als Liste):

```
<ul>
    {for $n in doc("E:\feuerwerk.xml")
            /Feuerwerksartikel/*/@Name
    return <li>{string($n)}</li>
</ul>
```

- Ausgabe:

```
<ul>
    <li>Silver Star</li>
    <li>Tanz der Vampire</li>
</ul>
```


XQuery Software (1)

- IPSI XQ

  In Java, nichtkommerzielle Nutzung ist kostenlos.
  [http://www.ipsi.fraunhofer.de/oasys/projects/ipsi-xq/]

- AltovaXML

  Der Validator und XSLT/XQuery Auswerter, den auch XML Spy benutzt, ist kostenlos. [http://www.altova.com/altovaxml.html]

- Galax

  Open source, von einigen Herausgebern der XQuery Specification.
  [http://www.galaxquery.org/]

- Saxon

  Von Michael Kay, herausgeber der XSLT 2.0 Spezifikation.
  Die Basisversion ist Open Source. [http://saxon.sourceforge.net/]

XQuery Software (2)

- X-HIVE

  Kommerzielles XML-DBMS, Online Demo (beliebige Anfragen an vorgegebenen Datenbestand). [http://support.x-hive.com/xquery/].

- Qizx/open

  Open Source, in Java. [http://www.axyana.com/qizxopen/]
  Online Demo: [http://www.xmlmind.com:8080/xqdemo/xquery.html]

- eXist (open source native XML database)

  Native XML DBMS, Open Source [http://exist.sourceforge.net/]
  Online demo: [http://demo.exist-db.org/sandbox/sandbox.xql]

Inhalt

1. Semistrukturierte Daten, XML als DB

2. XML Schema

3. XQuery

4. XML und SQL

Oracle 9i (1)

- Man kann XML in Spalten vom Typ `SYS.XMLTYPE` abspeichern.
- Mit

  `SYS.XMLTYPE.create('...')`

  kann man Werte dieses Typs eingeben, dabei wird die Wohlgeformtheit geprüft.
- Mit

  `Spalte.extract('...').getStringVal()`

  kann man einen XPath-Ausdruck ... auf die in einer Spalte abgespeicherten Werte anwenden.

Oracle 9i (2)

- Man kann Ergebnisse von SQL-Anfragen als XML ausgeben (mit der XML SQL Utility XSU).
- Dabei werden strukturierte Typen, Referenz- und Kollektionstypen auf entsprechende Strukturen in XML abgebildet.
- Auch umgekehrt ist eine Abbildung von XML auf objektrelationale Strukturen möglich.
- Außerdem werden XML Parser, XSLT-Prozessor, XML Schema Validator, und ein XML Class Generator for Java mitgeliefert.

DB2 V9 (1)

- IBM DB2 Version 9 enthält einen Auswerter für XQuery.

  Sie sind besonders stolz darauf, daß dieser "nativ" ist, und es nicht einfach nach SQL übersetzt.
- Man kann XQuery und SQL-Anfragen wechselseitig in einander schachteln.
- Der XML-Datentyp heißt hier "`xml`".
- XQuery-Anfragen werden durch das vorangestellte Schlüsselwort "`xquery`" gekennzeichnet.

DB2 V9 (2)

- Auf Spalten wird von XQuery aus mit

  `db2-fn:xmlcolumn('Tabelle.Spalte')`

  zugegriffen, auf Ergebnisse von SQL-Anfragen mit

  `db2-fn:sqlquery('SELECT ... ')`

  Die SQL-Anfrage muß natürlich Werte vom Typ `XML` liefern.
- Für den Aufruf von XQuery aus SQL gibt es die Funktionen `XMLQuery('...')` und `XMLExists('...')`.

Schlussbemerkung

- XML ist wichtig geworden.
- Alles wird immer komplizierter (bis irgendwann ein genialer neuer Vorschlag kommt).
- XML als eine Art von Datenbank zu sehen kann zu einer Horizonterweiterung führen.
- Die Vorlesung "XML und Datenbanken II" findet am 1. und 2. Oktober als Kompaktkurs statt.

  Behandelt u.a. XQuery, setzt XML Schema und XPath voraus.