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

Inhalt

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.

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.

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.

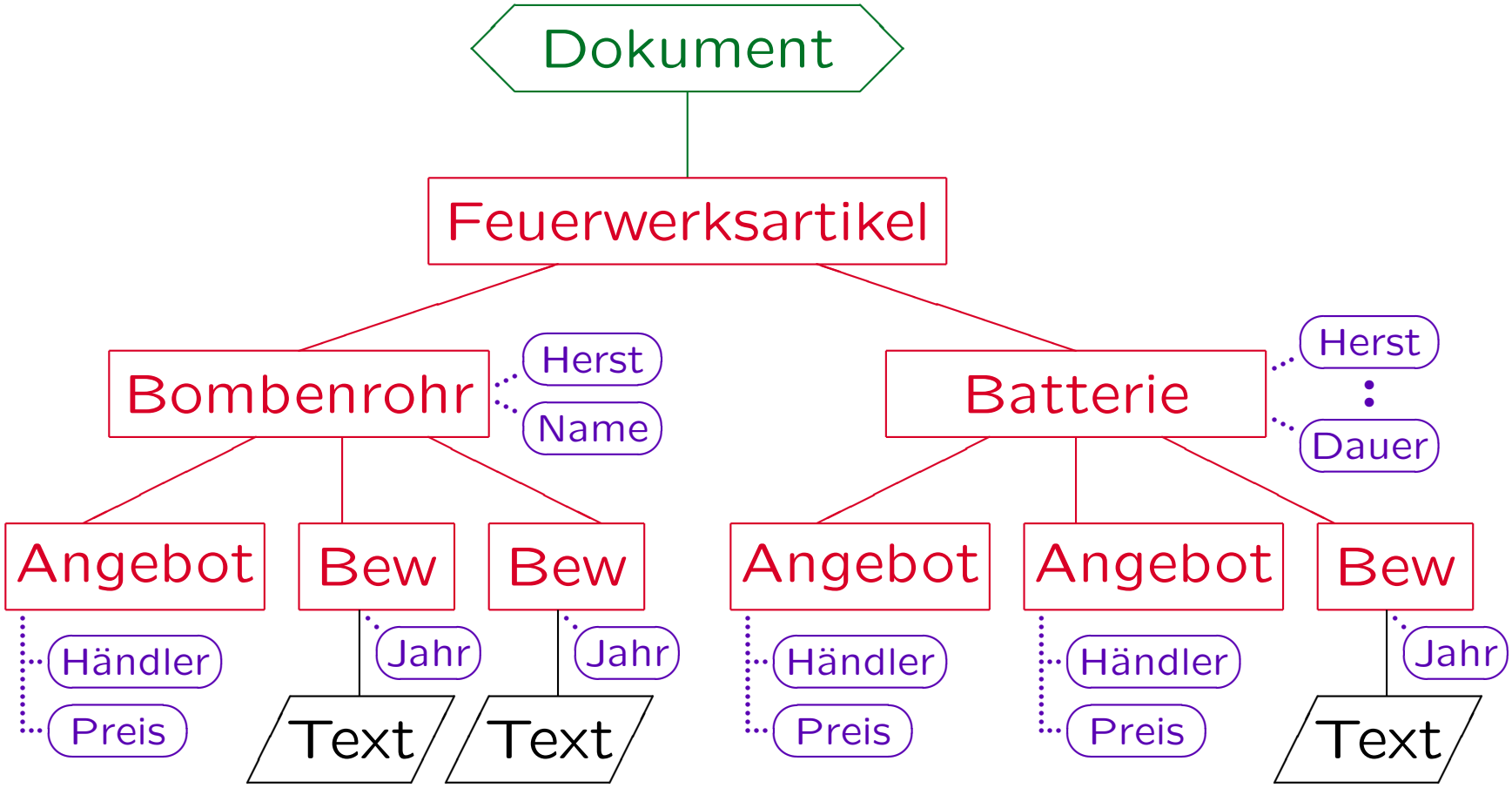
- Mit SQL könnte man auch nicht nach allen Tabellen und Spalten fragen, in deren Inhalt (den Daten) ein gegebenes Wort vorkommt.

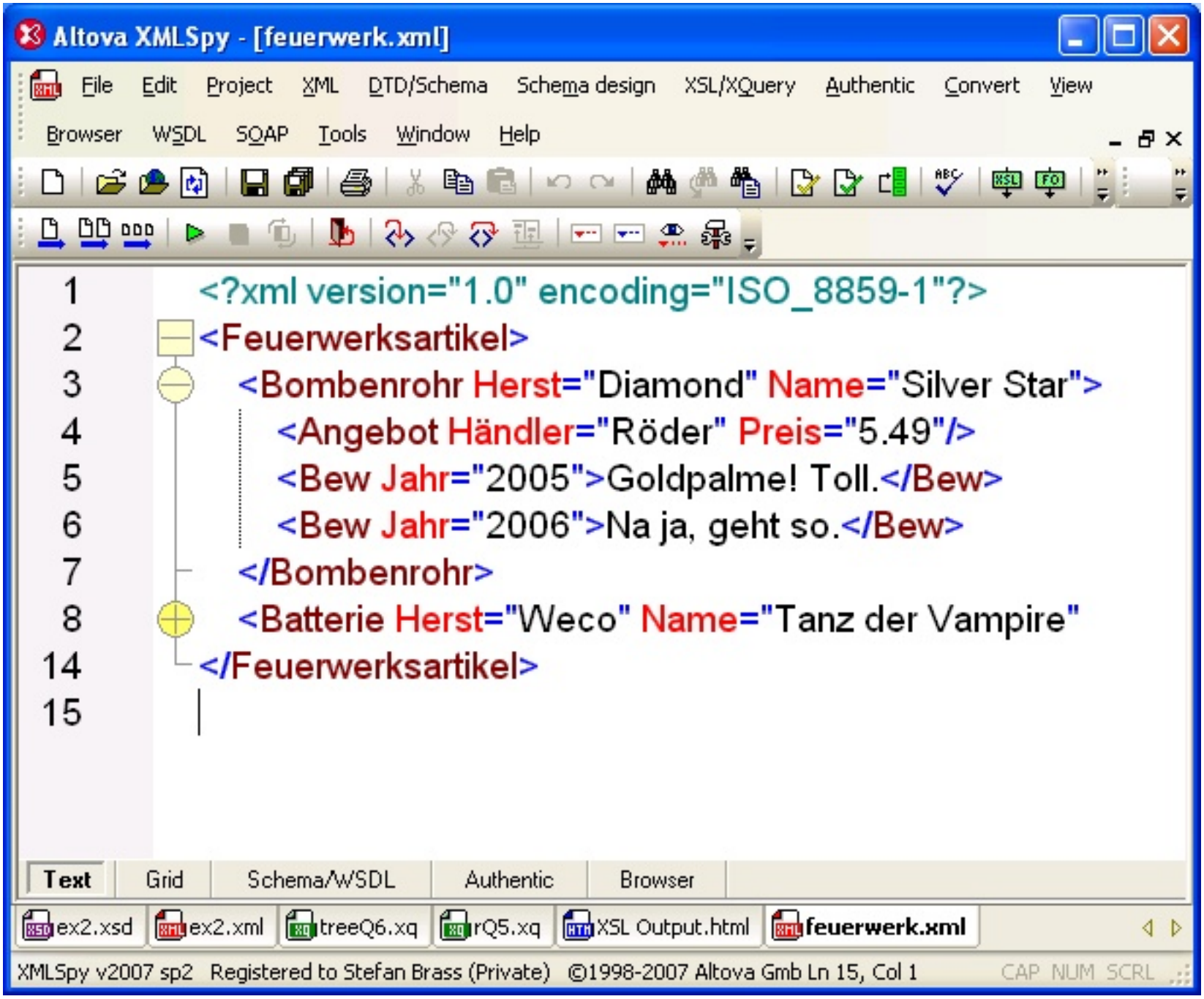
  In einer XML-Datei könnte man mit einem Texteditor danach suchen.

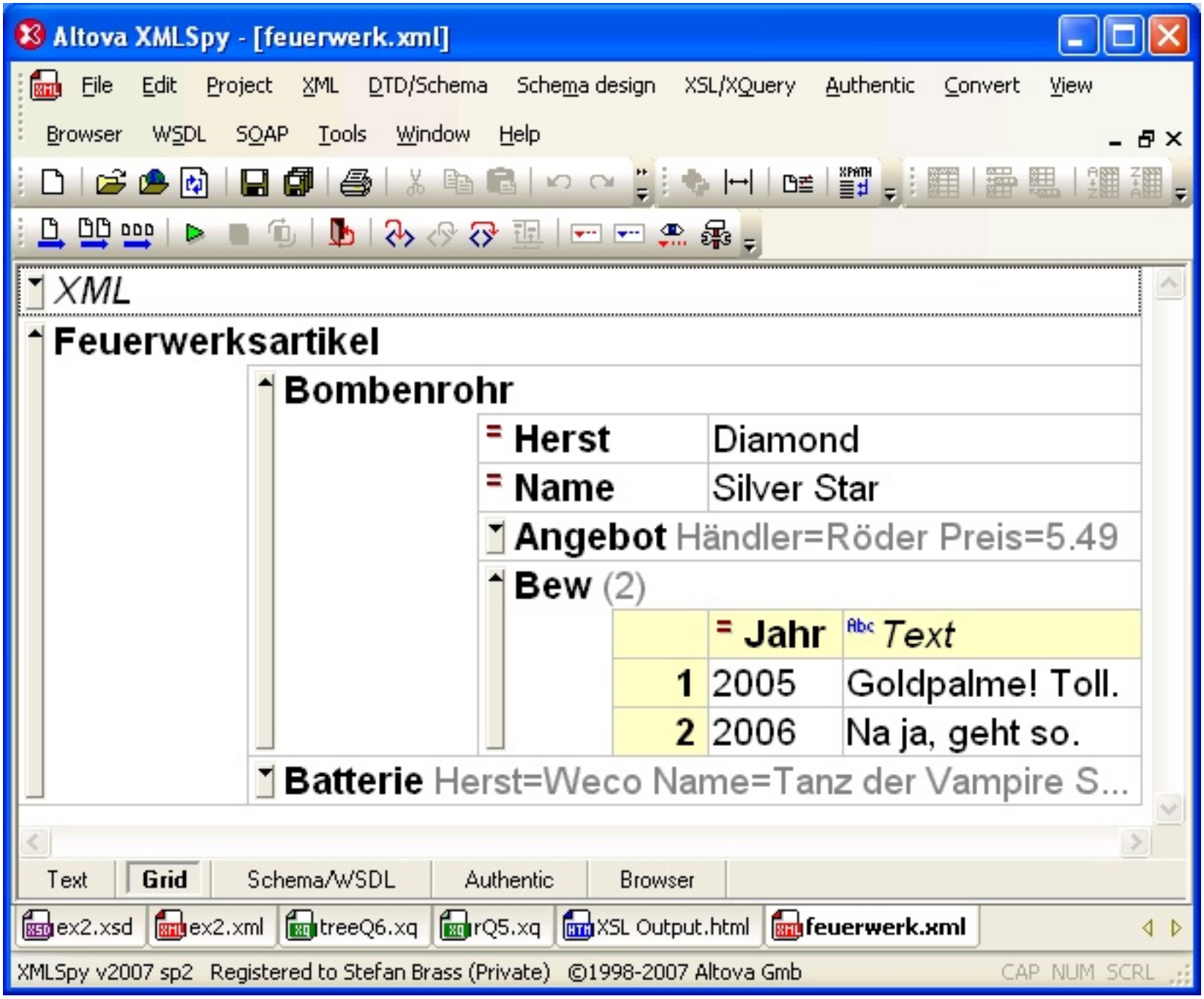
Beispiel

```
<?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>
```

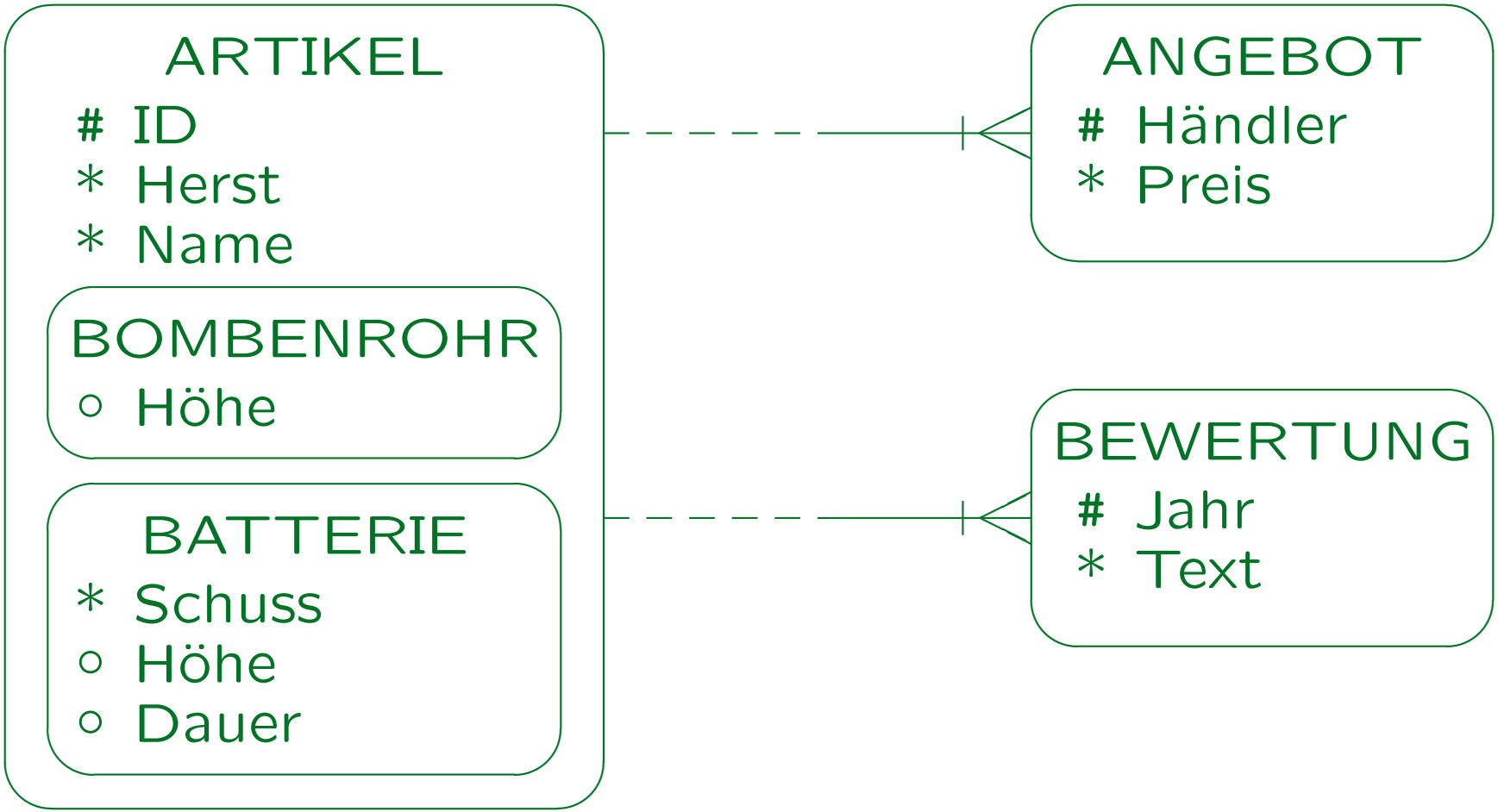
Interne Darstellung: XDM




Altova XMLSpy - [feuerwerk.xml]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View
Browser WSDL SOAP Tools Window Help
1 <?xml version="1.0" encoding="ISO_8859-1"?>
2 <Feuerwerksartikel>
3 <Bombenrohr Herst="Diamond" Name="Silver Star">
4 <Angebot Händler="Röder" Preis="5.49"/>
5 <Bew Jahr="2005">Goldpalme! Toll.</Bew>
6 <Bew Jahr="2006">Na ja, geht so.</Bew>
7 </Bombenrohr>
8 <Batterie Herst="Weco" Name="Tanz der Vampire"
14 </Feuerwerksartikel>
15
Text Grid Schema/WSDL Authentic Browser
ex2.xsd ex2.xml treeQ6.xq rQ5.xq XSL Output.html feuerwerk.xml
XMLSpy v2007 sp2 Registered to Stefan Brass (Private) ©1998-2007 Altova Gmb Ln 15, Col 1
CAP NUM SCRL


Altova XMLSpy - [feuerwerk.xml]
File Edit Project XML DTD/Schema Schema design XSL/XQuery Authentic Convert View
Browser WSDL SOAP Tools Window Help
XML
Feuerwerksartikel
Bombenrohr
Herst Diamond
Name Silver Star
Angebot Händler=Röder Preis=5.49
Bew (2)
Jahr Text
1 2005 Goldpalme! Toll.
2 2006 Na ja, geht so.
Batterie Herst=Weco Name=Tanz der Vampire S...
Text Grid Schema/WSDL Authentic Browser
ex2.xsd ex2.xml treeQ6.xq rQ5.xq XSL Output.html feuerwerk.xml
XMLSpy v2007 sp2 Registered to Stefan Brass (Private) ©1998-2007 Altova Gmb
CAP NUM SCRL

Klassischer DB-Entwurf



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.

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.

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)

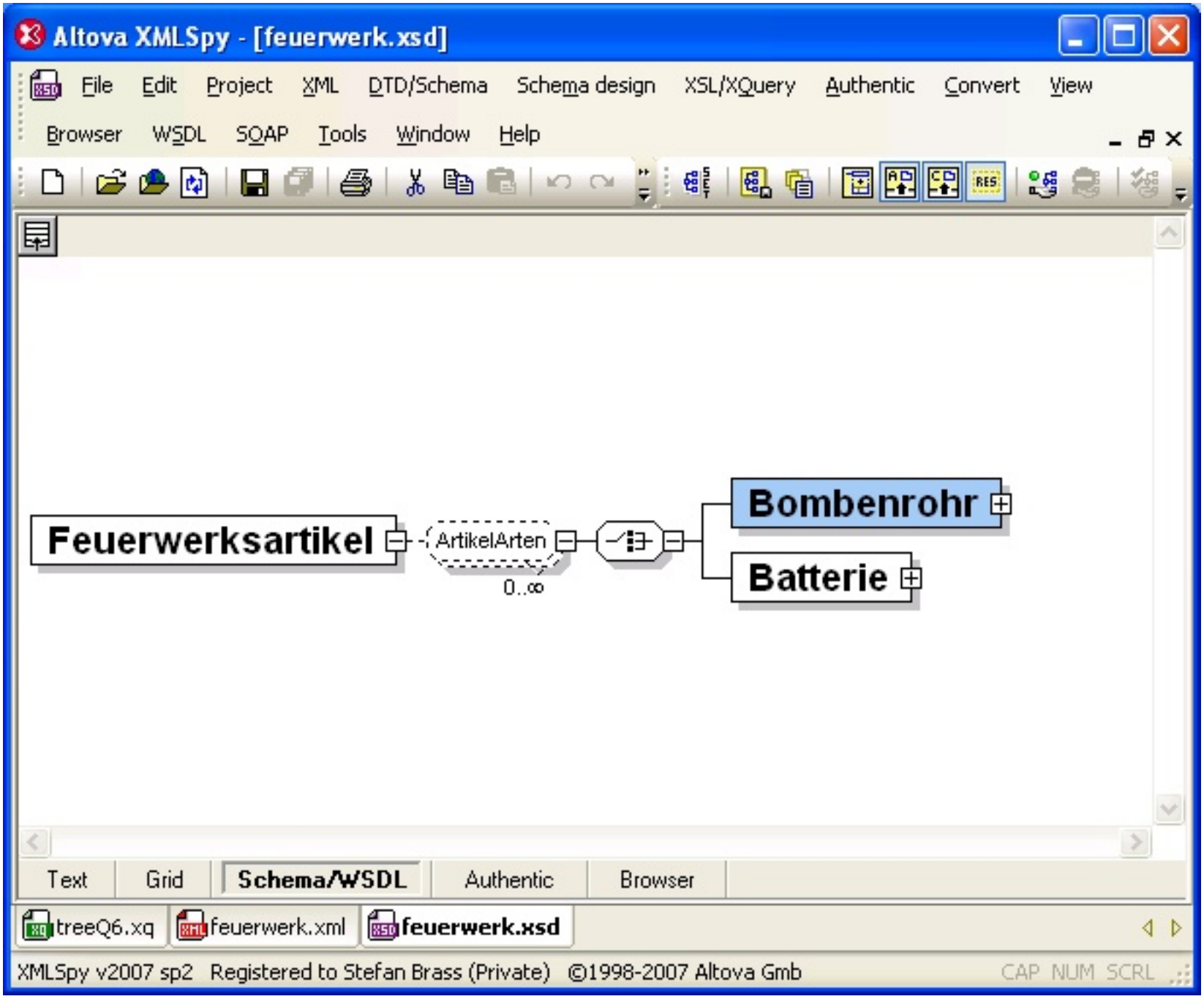
```
<xs:complexType name="BombenrohrTyp">
        <xs:annotation>
            <xs:documentation>
                Bei Klasse II (hier): Einschüsser
            </xs:documentation>
        </xs:annotation>
    <xs:complexContent>
        <xs:extension base="ArtikelTyp">
            <xs:attribute name="Höhe"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

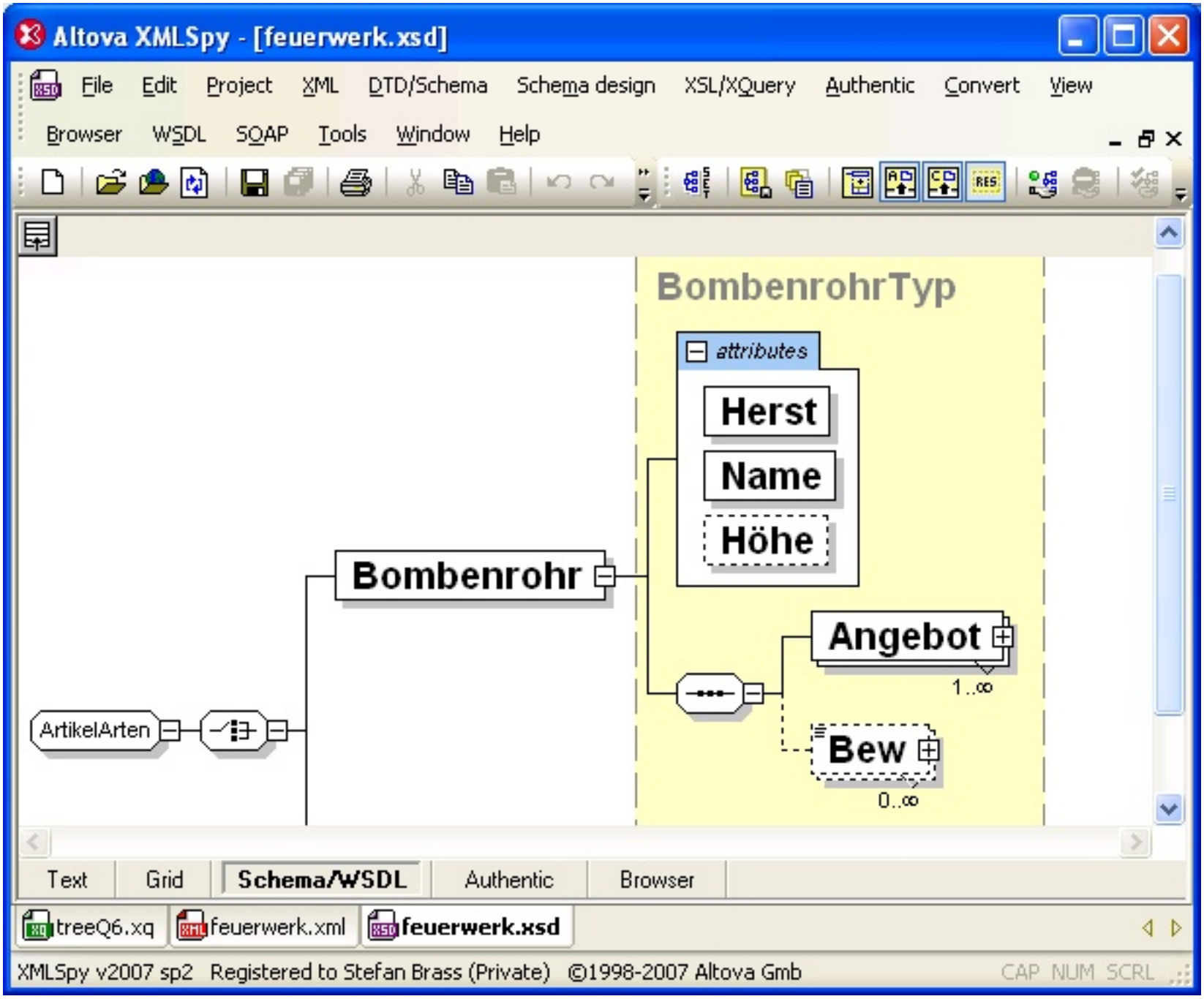
XML Schema: Beispiel (5)

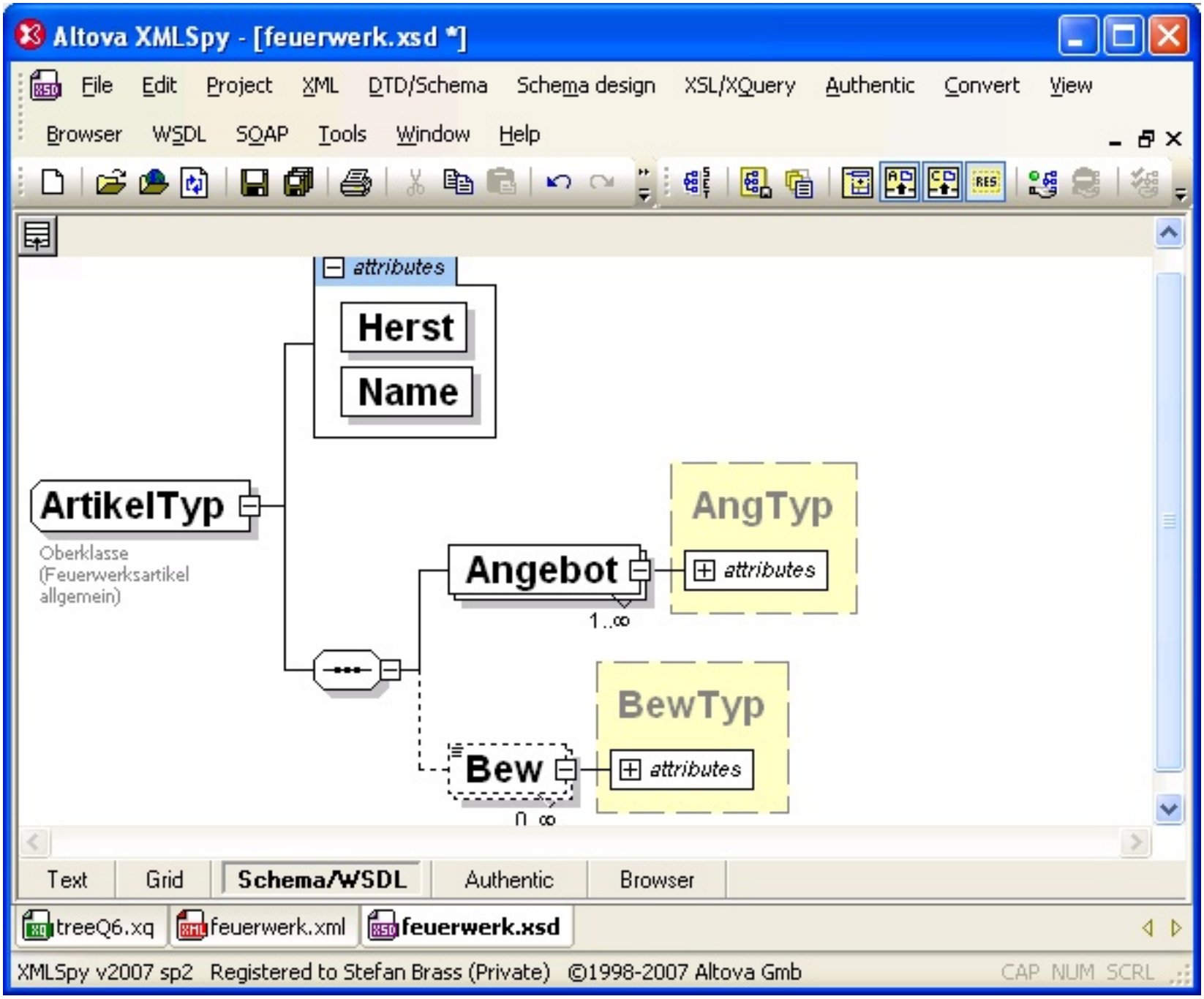
```
<xs:complexType name="BatterieTyp">
    <xs:complexContent>
        <xs:extension base="ArtikelTyp">
            <xs:attribute name="Schuss"
                type="xs:nonNegativeInteger"
                use="required"/>
            <xs:attribute name="Höhe"
                type="xs:nonNegativeInteger"/>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

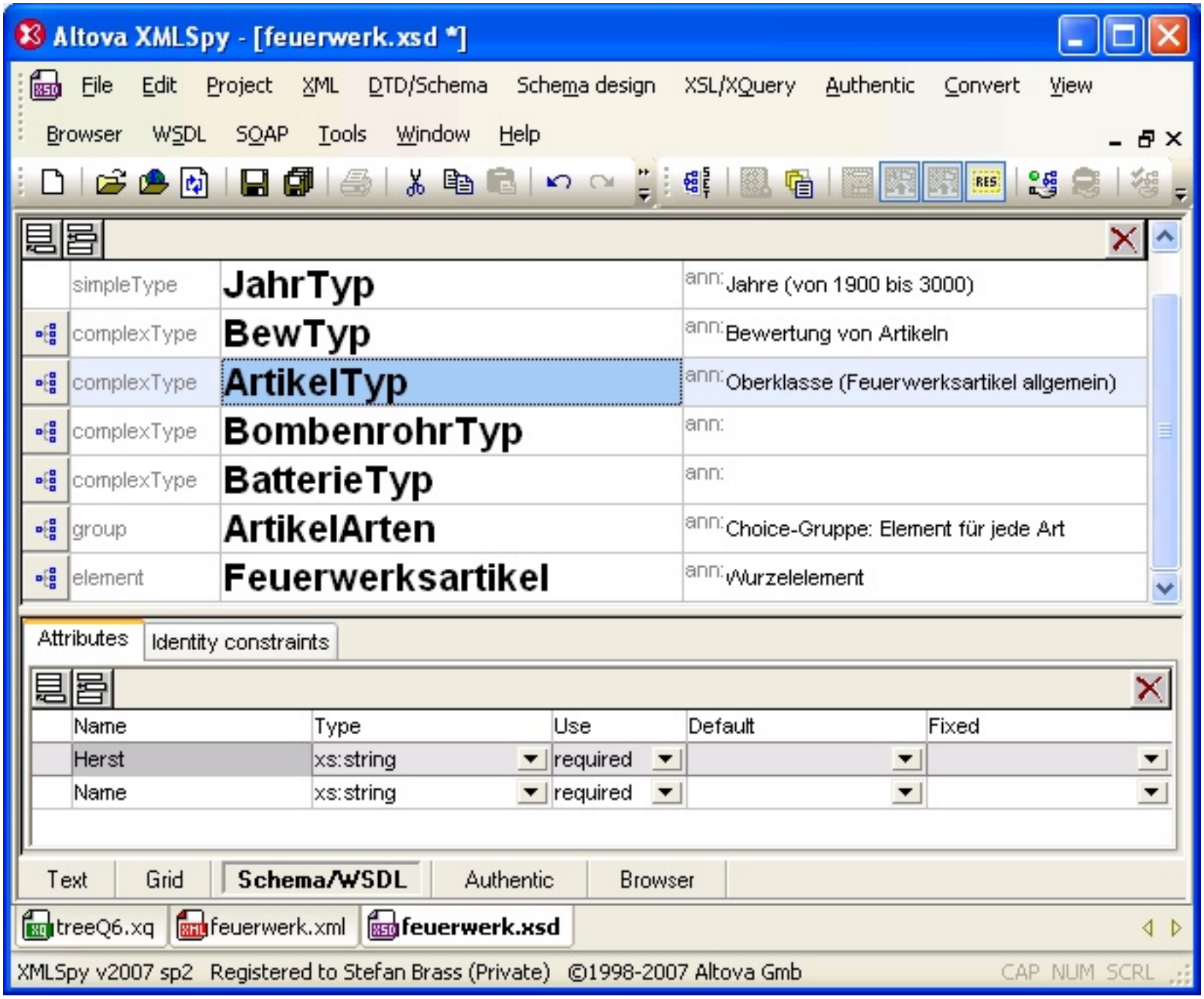
XML Schema: Beispiel (6)

```
<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
Feuerwerksartikel
ArtikelArten
0..∞
Bombenrohr
Batterie
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
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)
Angebot
1..∞
AngTyp
attributes
BewTyp
attributes
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
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

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.