6. Datenbanken und XML

6.0 Einführung

XML (e**X**tended **M**arkup **L**anguage) definiert ein Format für Textdateien, das sich sehr gut als Austauschformat zwischen verschiedenen Softwaresystemen (z.B. zwischen Datenbanken und anderen Systemen) eignet. Jedes Standarddatenbanksystem bietet eine Exportschnittstelle an, um die Inhalte eines Datenbanksegments als XML-Datei zu exportieren.

BSP.1: Der Inhalt einer RDB Tabelle LAND wird mittels einer XML–Exportschnitt-stelle als XML-Datei exportiert (Diese Aktivität wurde im 1. Praktikum mittels eines RDBMS-Exportwerkzeugs ausgeführt):

a) Inhalt der Tabelle LAND:

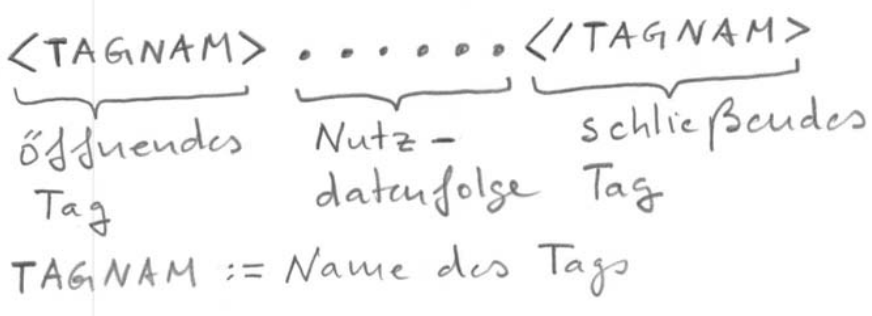
```
1   USA      1,3784   USD
2   Schweiz  1,2343   SFR
3   Japan    134,65   YEN
```

b) Inhalt der XML-Datei LAND.XML:

```
<?xml version="1.0"  encoding="ISO-8859-1">
<TAB NAME="LAND">
     <ROW>
          <LANDNR>1</LANDNR>
          <LBEZ>USA</LBEZ>
          <WAEHFAKTOR>1.3784</WAEHFAKTOR>
          <WAEHKENN>USD</WAEHKENN>
     </ROW>
     <ROW>
          <LANDNR>2</LANDNR>
          <LBEZ>Schweiz</LBEZ>
          <WAEHFAKTOR>1.2343</WAEHFAKTOR>
          <WAEHKENN>SFR</WAEHKENN>
     </ROW>
     <ROW>
          <LANDNR>3</LANDNR>
          <LBEZ>Japan</LBEZ>
          <WAEHFAKTOR>134.65</WAEHFAKTOR>
          <WAEHKENN>YEN</WAEHKENN>
     </ROW>
</TAB>
```

XML ist wie HTML eine **Markup**-Sprache, die mit Markups bzw. **Tags** (engl.: Tauende) ein Textdokument strukturiert. Während bei HTML Tags sowohl für die „schöne Gestaltung“ (**Presentation**, z.B. Fettdruck, Zeilenabstände, Größe von Überschriften) als auch für die inhaltliche Gliederung (**Content**, z.B. Teilung eines HTML-Dokuments in <HEAD> und <BODY>) verwendet werden, dienen **die Tags in XML nur zur inhaltlichen Gliederung**.

BSP.2: Der Aufbau eines einfachen XML-Elements, das nur aus einem Tag-Paar und einer Nutzdatenfolge besteht:
a) Allgemeine Struktur eines einfachen XML-Elements:

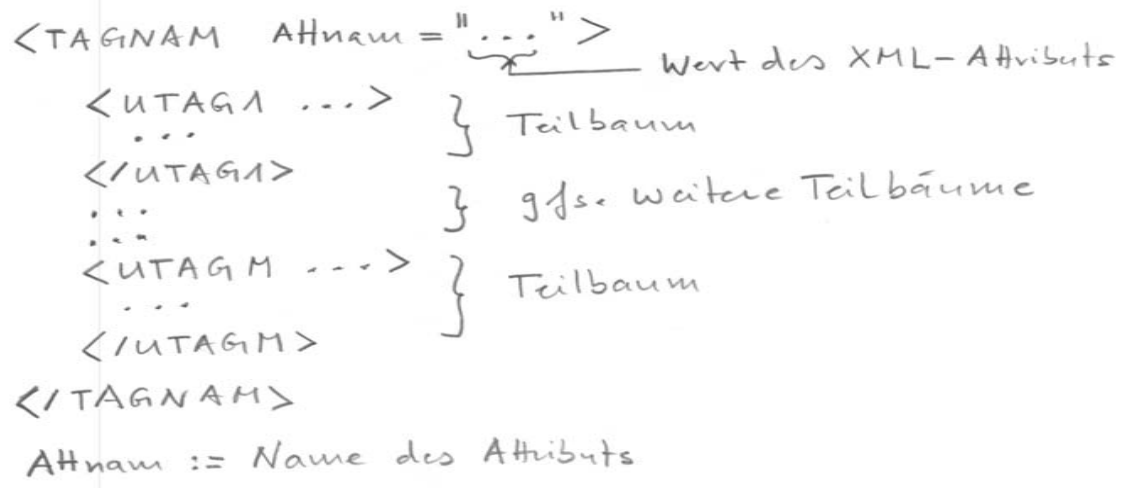


b) Ein einfaches XML-Element im BSP.1:

```
<WAEHFAKTOR>1.3784</WAEHFAKTOR>
```

BSP.3: Ein komplexes XML-Element, das eine Folge von einfacheren XML-Elemente (Teilbäumen) und ein XML-Attribut enthält.

a) Allgemeine Struktur eines komplexen XML-Elements:



b) Ein komplexes XML-Element im BSP.1:

```
<TAB NAME="LAND">
     <!-- Teilbaum 1 -->
     <ROW> . . . . . </ROW>
     <!-- Teilbaum 2 -->
     <ROW> . . . . . </ROW>
```

```
    <!-- Teilbaum 3 -->
    <ROW> . . . . . . </ROW>
</TAB>
```

Def.1: Ein XML-Element der Form **<!-- . . . -->** ist ein **Kommentar**. Die Zeichenfolge **. . .** ist ein beliebiger Kommentartext. Kommentare dürfen nicht innerhalb eines öffnenden oder schließenden Tags stehen.

BSP.4: Beispiel eines Kommentars:

```
    <!-- Dieser Satz ist ein Kommentar -->
```

Ein komplexes XML-Element, das eine Folge von einfacheren XML-Elemente (Teilbäumen) und ein XML-Attribut enthält.

BSP.5: Ein komplexes XML-Element **<LIT>** für eine vollständige Literaturangabe:

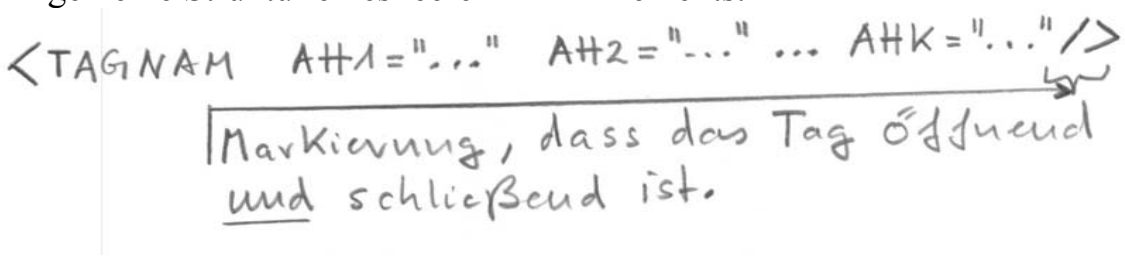
```
<LIT ISBN="978-3-642-38238-3">
    <VERFASSER>
        <AUTOR AUNR="1">
            <NAME>Alten</NAME>
            <VNAM>Heinz-Wilhelm</VNAM>
            <TITEL>Prof. Dr.</TITEL>
        </AUTOR>
        <AUTOR AUNR="8">
            <NAME>Wußing</NAME>
            <VNAM>Hans</VNAM>
```

```
            <TITEL>Prof. Dr.</TITEL>
        </AUTOR>
    </VERFASSER>
    <TITEL>4000 Jahre Algebra</TITEL>
    <UTITEL>Geschichte – Kulturen – Menschen</UTITEL>
    <AUFL NR="2">aktualisierte und ergänzte Auflage</AUFL>
    <EJAHR JAHR="2014"/>
    <EORT ORT1="Berlin" ORT2="Heidelberg"/>
    <VERLAG>Springer-Verlag</VERLAG>
</LIT>
```

Anm.1: Dem Designer einer XML-Struktur ist freigestellt, ob er seine Nutzdaten als Nutzdatenfolge innerhalb eines Tag-Paares unterbringt oder ob er sie als Wert eines XML-Attributs verwalten möchte.

Anm.2: Im obigen Beispiel treten auch XML-Elemente, die weder einen Teilbaum noch eine Nutzdatenfolge enthalten, die vereinfacht gesagt **leer** sind.

a) Allgemeine Struktur eines leeren XML-Elements:

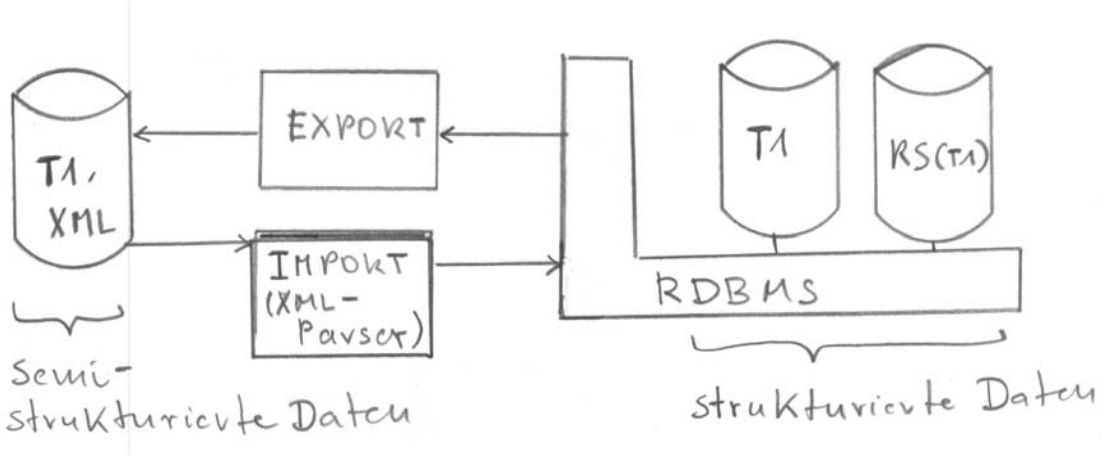


b) Leere XML-Elemente im BSP.5:

```
<EJAHR JAHR="2014"/>
<EORT ORT1="Berlin" ORT2="Heidelberg"/>
```

6.1 XML und Datenbanken (Export / Import)

Aufgrund der Tabellenstruktur bei einem RDBMS ist der **Export** einer Tabelle in eine XML-Datei nicht schwer zu programmieren. Es gilt: Die Segmentstruktur in der Datenbank bestimmt die XML-Struktur. Die folgende Abbildung veranschaulicht den Export / Import von einer / in eine DB-Tabelle:



(Abb.1: Export / Import DB-Tabelle / XML-Datei)

Alle Nutzdaten in einer Datenbank sind **strukturierte Daten**. Für jeden Wert **wij** einer **Tabelle T1** gilt: Er liegt in einer Zeile **i** und unterliegt einem Attribut **Aj**. Für das Attribut Aj ist im **Relationenschema RS(T1)** der Eintrag **(Aj, dt(Aj), Wj)** hinterlegt. D.h. für den Wert **wij** ist durch **Aj** die **Semantik** und durch **dt(Aj)** der Datentyp bestimmt. Daten heißen **strukturiert** genau dann, wenn ihre Semantik und ihr Datentyp bestimmt ist. D. h. **Daten in einer DB sind strukturierte Daten.**

Nutzdaten in einem XML-Element liegen standardmäßig in einer Nutzdatenfolge vor, die durch ein öffnendes und ein schließendes Tag gleichen Namens eingeschlossen sind

(z.B. **<TAGNAM> wij </TAGNAM>**). Dadurch bestimmt der **Tag-Name** formal die **Semantik** des Werts. Werte haben in XML keine besonderen Datentypen. Werte sind Zeichenfolgen. D. h. sie sind alle, grob betrachtet, vom Typ String. Daten heißen **semistrukturiert** genau dann, **wenn sie eine Semantik, aber keinen Datentyp haben**. D. h. **XML-Daten sind semistrukturierte Daten**.

Beim **Import** steht man insbesondere vor folgendem Problem: Man muss semistrukturierte Daten korrekt auf strukturierte Daten abbilden. Daher muss das Import-Programm unter Verwendung zusätzlicher Quellen den XML-Elementen Datentypen für ihre Nutzdaten hinzufügen, die auf Datentypen des Zieldatenbanksystems abbildbar sind. Eine solche Quelle ist z.B. eine XML-**Grammatik**, die **jedem XML-Element des Dokuments einen Datentyp zuordnet**. Dieses Manko von XML veranlasste das W3C das Konzept von **XML-Schema** zu entwickeln. In XML-Schema wird zu jedem XML-Dokumenttyp eine Grammatik beschrieben, die jedem Element ein Datentyp (ein Schemaeintrag) zuordnet. Ein strukturtreues Importprogramm verfügt über einen XML-Parser, der Schemaeinträge verarbeitet.

Literaturhinweise:

1) Th. Rottach / S. Groß: „XML Kompakt“, Heidelberg, Berlin (Spektrum), 2002, ISBN: 3-8274-1339-7 .
2) [XML-Standard]: http://www.w3c.org/TR/2004/REC-xml-20040204/ .

3) Ralph Steyer „XML und Java“, entwickler.press (Frankfurt), 2003, ISBN 3-935042-78-7.

6.2 XML-Syntax, Hierarchiemodell, Wohlgeformtheit

Def.2: Ein XML-Dokument heißt **wohlgeformt** genau dann, wenn es die Regeln der XML-Syntax erfüllt.

Def.3: Die **XML-Syntax** enthält folgende acht Regeln:
(R1) Das **XML-Dokument** besteht aus einem **Prolog** und einem **Dokument-Element**.

(R2) Das **Dokument-Element** ist die **Wurzel** des XML-Dokuments (d.h. jedes XML-Dokument enthält eine **Baumstruktur** mit genau einem Wurzelelement und Nutzdaten, die sich an den Blättern des Dokuments befinden. Der Baum kann eine beliebige Verschachtelungstiefe haben).

(R3) Das **Dokument-Element** ist ein **Element**.

(R4) Jedes **Element** kann **Kinderelemente** sowie **Nutzdaten** beinhalten. Nutzdaten sind **Zeichenfolgen**.

(R5) Jedes Element ist entweder ein Element mit leerem Eintrag oder es enthält ein **öffnendes** und ein **schließendes Tag**:

`<Elementname> ... </Elementname>` . Innerhalb eines solchen Tag-Paares stehen ein oder mehrere **Teilbäume** oder eine **Zeichenfolge**. Ein Element mit leerem Eintrag besteht nur aus einem Tag, das öffnend und schließend ist:
`<Elementname . . . />`

(R6) **Tags verschiedener Elemente** dürfen sich in ihrem Gültigkeitsbereich **nicht überlappen**.

(R7) **Öffnende Tags** können beliebig viele oder keine **Attribute** haben, deren Namen innerhalb des Tags eindeutig sein müssen. Ein Tag mit Attributen hat folgenden Aufbau: **`<Elementname att1="..." att2="..." ... attN="...">`**

(R8) Der **Prolog** enthält genau eine **XML-Deklaration**. Er kann darüber hinaus noch eine **Dokumenttyp-Deklaration**, **Kommentare** und **Processing Instructions** beinhalten. In der **XML-Deklaration** wird die gültige XML-Version und ein normierter Zeichensatz genannt, gemäß der das Dokument formuliert ist. In der **Dokumenttyp-Deklaration** wird die Verbindung zur Grammatik hergestellt, mit der das Dokument validierend geparst werden kann (vgl. Kap. 6.3). Durch **Processing Instructions** können z.B. Stylesheets, die der graphisch aufbereiteten Darstellung eines XML-Dokuments in einem Browser dienen, zugeordnet werden.

BSP.6: Eine **XML-Deklaration**:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

BSP.7: Eine **Dokumenttyp-Deklaration**:

```
<!DOCTYPE tabelleMa SYSTEM “tabelleMa.dtd“>
```

BSP.8: Ein minimales XML-Dokument:

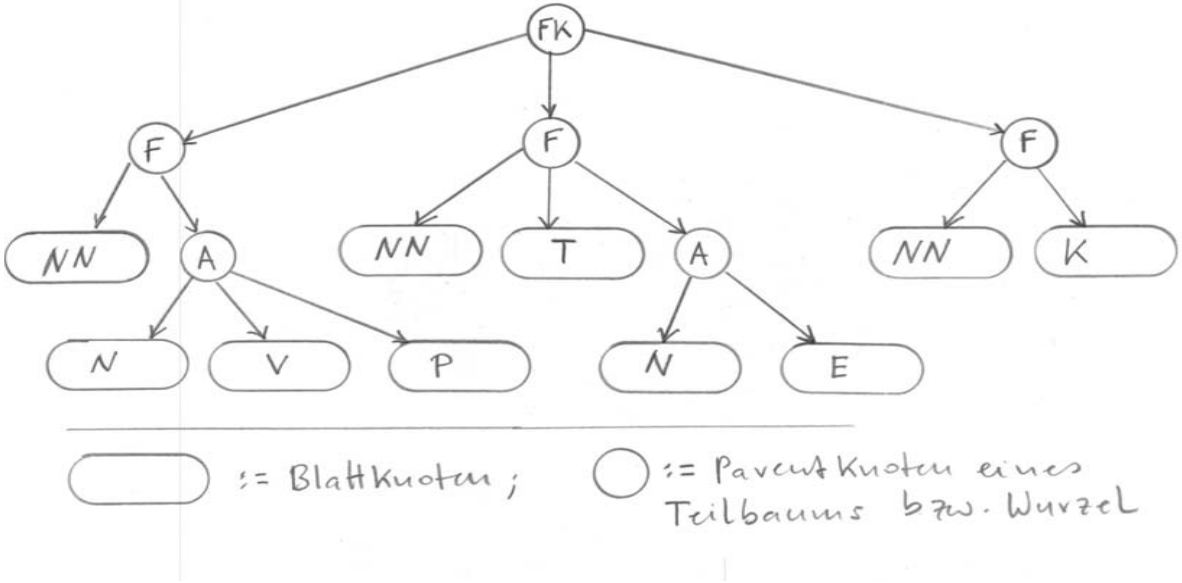
```
<?xml version=“1.0“ encoding=“UTF-8“?>
<Wurz1> Ein karges Dokument</Wurz1>
```

BSP.9: Ein Dokument mit Teilbäumen:

```
<?xml version=“1.0“ encoding=“ISO-8859-1“ ?>
<Freundeskreis>
    <Freund bk=“gut“>
        <NName> Fritz </NName>
        <Adr>
            <Name>Müller</Name>
            <Vname>Friederich</Vname>
            <Plz>50678</Plz>
        </Adr>
    </Freund>
    <Freund bk=“ferner“>
        <NName>Otto</NName>
        <Tel>474747</Tel>
        <Adr>
```

```
            <Name>Schmitz</Name>
            <Email>otto_schmitz@gmx.de</Email>
        </Adr>
    </Freund>
    <Freund bk="Bier">
        <NName>Kalle</NName>
        <Kneipe>Früh</Kneipe>
    </Freund>
</Freundeskreis>
```

Zum obigen XML-Dokument FREUNDESKREIS.XML ist in nachfolgender Abbildung die zugehörige Baumstruktur angegeben:



(Abb.2: Baumstruktur des Dokuments FREUNDESKREIS.XML)

BSP.10: Ein XML-Dokument, das strukturtreu alle Daten und Metadaten einer Tabelle T aus einem RDBS abbildet. Hier ist **T** eine Artikeltabelle mit dem Tabellenschema **RS(T)**:
RS(T) = { (artnr,int,PRIK) , (artbez,char(30),Ø) , (preis,decimal(7,2), preis >=0)}

XML-Dokument mit Dokumenttyp-Deklaration:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>:
<!DOCTYPE tabelleMa SYSTEM "tabelleMa.dtd">
<tabelleMa>
    <tabname>Artikel</tabname>
    <Zeile>
        <artnr DT="int">4711</artnr>
        <artbez DT="char(30)">Parfüm</artbez>
        <preis DT="decimal(7,2)">5.95</preis>
    </Zeile>
    <Zeile>
        <artnr DT="int">4810</artnr>
        <artbez DT="char(30)">Seife</artbez>
        <preis DT="decimal(7,2)">0.45</preis>
    </Zeile>
    ...
</tabelleMa>
```

6.3 DTD, Validität

Eine **Document Type Definition** (=:**DTD**) ist eine Grammatik für eine Menge strukturgleicher XML-Dokumente. Grammatiken sind Grundlagen für Parser. Ein XML-Parser kann prüfen, ob ein XML-Dokument sich an die in der DTD vereinbarte Struktur hält. D.h. der Parser prüft, ob das XML-Dokument in Bezug auf die DTD korrekt oder nicht korrekt ist.

Def.4: Ein XML-Dokument heißt **valide** genau dann, wenn ihm eine DTD zugeordnet ist und wenn der Dokumentaufbau gemäß der DTD korrekt ist.

Anm.3: Die DTD wird einem XML-Dokument im Prolog zugeordnet (s. (R8) und BSP.7)

Eine DTD besteht aus Definitionen für:

B1) XML-Elemente,
B2) Attributlisten von XML-Elementen,
B3) XML-Entities,
B4) Processing Instructions,
B5) Notationen.

Anm.4: Zu B3) Eine XML-Entity ist ein Sonderzeichen oder ein XML-Teildokument. Entity-Deklarationen sind notwendig, falls Sonderzeichen wie '<', '>', ' “ ',... benötigt werden oder falls ein XML-Dokument auf mehrere Teildokumente verteilt werden soll.

Zu B4) Ein Processing Instruction ist z.B. erforderlich, wenn man einem XML-Dokument ein XSLT-Stylesheet für die grafische Gestaltung zuordnen will.

Zu B5)Notationen dienen dafür, um Dateien mit Nicht-XML-Dateien in ein XML Dokument zu integrieren.

Def.5: **DTD-Definition für XML-Elemente:** Bei dieser Deklaration wird unterschieden, (a) ob ein Element nur einen **leeren Eintrag** haben soll, (b) ob ein Element nur **eine Nutzdatenfolge** enthalten soll (Blattelement) oder (c) ob ein Element mindestens einen **XML-Teilbaum** enthalten soll.

a) Definition eines Elements mit einem **leeren Eintrag:**

```
<!ELEMENT tagname EMPTY>
```

BSP.11: Definition: `<!ELEMENT EJAHR EMPTY>`

Valides Element: `<EJAHR JAHR="2014"/>`

b) Definition eines Elements, das nur eine **Nutzdatenfolge** enthält:

```
<!ELEMENT tagname (#PCDATA)>
```

`#PCDATA` steht für **parsed character data**.

BSP.12: Definition: `<!ELEMENT TITEL (#PCDATA)>`

Valides Element: `<TITEL>4000 Jahre Algebra</TITEL>`

c) Definition eines Elements, das bereits definierte XML-Elemente bzw. einen **Teilbaum** enthält**:** Sind **E1, E2, …, EN** bereits definierte XML-Elemente, dann kann

ein neues Element entweder ein **Tupel** sein, der aus diesen Elementen gebildet wird oder das neue Element ist eine **Alternative** dieser Elemente. Die Elemente E1, E2, …, EN gehen mit **Kardinalitäten** (Vielfachheiten) **q1, q2, …, qN** in das neue Element ein. Für alle **Kardinalitäten** gilt ($1 \leq \mathbf{i} \leq N$):

qi = kein Eintrag	⇔	**genau 1**
qi= ?	⇔	**höchstens 1** (0 oder 1)
qi= +	⇔	**mindestens 1** (1...n)
qi= *	⇔	**0 oder mindestens 1** (0...n)

(c1) Definition eines **Tupel**-Elements:

```
<!ELEMENT tagname (E1 q1, E2 q2, … …, EN qN)>
```

BSP.13: Definition:
```
<!ELEMENT Zeile(artnr,artbez,preis)>
```
Valides Element:
```
<Zeile>
    <artnr DT="int">4810</artnr>
    <artbez DT="char(30)">Seife</artbez>
    <preis DT="decimal(7,2)">0.45</preis>
</Zeile>
```

BSP.14: Definition:
```
<!ELEMENT Adr(Name,Email ?, Vname ?, Plz ?)>
```
Valides Elemente: i)
```
<Adr>
    <Name>Müller</Name>
    <Vname>Friederich</Vname>
    <Plz>50678</Plz>
```

```
    </Adr>
```

ii)
```
<Adr>
    <Name>Huber</Name>
    <Email>willi.huber@hobby1.de</Email>
    <Plz>88991</Plz>
</Adr>
```

(c2) Definition einer **Alternative**:

```
<!ELEMENT tagname (E1 q1|E2 q2)>
```

BSP.15: Definition: `<!ELEMENT AdrPFSTR(Postfach|Strasse)>`

i) Valides Element:
```
<AdrPFSTR>
    <Postfach>5130</Postfach>
</AdrPFSTR>
```

ii) Valides Element:
```
<AdrPFSTR>
    <Strasse>Heinestr.51</Strasse>
</AdrPFSTR>
```

Def.6: **DTD-Definition für Attributlisten von XML-Elementen:** Ein XML-Element kann mehrere Attribute haben, die als Liste angeordnet sind. Für jedes Attribut folgende drei Angaben hinterlegt:
a) der Attributname
b) ein sog. XML-Datentyp
c) eine Voreinstellung für das Attribut

zu b) Der Standardattributdatentyp ist CDATA (:= Characterdata).
zu c) Unterschieden werden: c1) ein Muss-Attribut: **#REQUIRED**
c2) ein Kann-Attribut: **#IMPLIED**
c3) ein voreingestellter Wert: **“Defaultwert“**
(Der Fall c3) impliziert #IMPLIED)

Allgemeine Syntax der Attributlisten Deklaration:

```
<!ATTLIST elementname
    att1 DT1 v1
    att2 DT2 v2
     ...    ...    ...
    attn DTn vn
>
```

mit: **atti** : Name des i-ten Attributs ($1 \leq i \leq n$)
DTi : XML-Datentyp des i-ten Attributs
vi : Voreinstellung des i-ten Attributs

BSP.16: Definition:

```
<!ELEMENT EORT EMPTY>
<!ATTLIST EORT
    ORT1 CDATA #REQUIRED
    ORT2 CDATA #IMPLIED
```

```
                    ORT3 CDATA #IMPLIED
                >
```

BSP.17: XML-Schema zum XML-Dokument **tabelleMa.XML** (s. BSP.10). Dieses XML-Schema ist in der Datei **tabelleMa.DTD** gespeichert:

```
<!ELEMENT tabelleMa (tabname,zeile+)>
<!ELEMENT zeile (artnr,artbez,preis)>
<!ELEMENT tabname (#PCDATA)>
<!ELEMENT artnr (#PCDATA)>
<!ELEMENT artbez (#PCDATA)>
<!ELEMENT preis (#PCDATA)>
<!ATTLIST artnr
       DT CDATA #REQUIRED>
<!ATTLIST artbez
       DT CDATA #REQUIRED>
<!ATTLIST preis
       DT CDATA #REQUIRED>
```

BSP.18: XML-Schema zum XML-Dokument **Freundeskreis.XML** (s. BSP.9). Dieses XML-Schema ist in der Datei **Freundeskreis.DTD** gespeichert:

```
<!ELEMENT Freundeskreis(Freund+)>
<!ELEMENT Freund(NName,Tel?,Adr?,Kneipe?)>
<!ELEMENT Adr(Name,Email?,Vname?,Plz?,Ort?)>
```

```
<!ELEMENT NName (#PCDATA)>
<!ELEMENT Tel (#PCDATA)>
<!ELEMENT Kneipe (#PCDATA)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Vname (#PCDATA)>
<!ELEMENT Plz (#PCDATA)>
<!ELEMENT Ort (#PCDATA)>
<!ATTLIST Freund
       bk CDATA #IMPLIED>
```

Erste Lernziele zu Kap.6: Datenbanken und XML

1. **Erklären können, wann ein XML-Dokument wohlgeformt ist. Hierzu die acht wichtigen Regeln der XML-Syntax benennen können. Wohlgeformte XML-Dokumente editieren können.**
2. **Erklären können, wann ein XML-Dokument valide ist. Den Zweck einer DTD benennen können. DTD Definitionen für XML-Elemente und für Attributlisten ausführen können.**
3. **Zu einem wohlgeformten XML-Dokument eine DTD als Grammatik aufstellen können.**

4. **Den Unterschied zwischen semistrukturierten und strukturierten Daten erklären können. Eine DTD für ein XML-Dokument, mit dem der Inhalt einer RDB-Tabelle mit Spaltendatentypen exportiert werden soll, aufstellen können.**
5. **Den Zweck von XML als Datenaustauschformat zwischen verschiedenen Softwaresystemen erläutern können.**