Kapitel 5: Datenaustausch mit XML

- Der E-Commerce beruht auf dem elektronischen Austausch von Dokumenten.
- Austauschformat ist XML.
- Die auszutauschenden Dokumente basieren typischerweise auf den Inhalten von Datenbanken.
- Exportieren von Daten einer Datenbank in XML; Importieren von XML in eine Datenbank.
- Gesamtmodell eines gemeinsamen elektronischen Marktes in XML.

5.1 Extensible Markup Language (XML)

Elemente

- XML ist eine *Auszeichnungssprache (Markup Language)*.
  `<aTagname>` *öffnender Tag*, `</aTagname>` *schließender* Tag.
- Öffnender und sein schließender Tag, zusammen mit dem durch beide umfassten Ausschnitt des Dokumentes, wird als *Element* bezeichnet; der *Elementname* ist der Name des Tags und der durch öffnenden und schließenden Tag umfasste Teil des Dokumentes ist der *Inhalt des Elementes*.
- Der Inhalt eines Elementes kann aus einer Zeichenkette (ohne Tags), weiteren Elementen, oder einer Mischung von beiden bestehen:
  - *Text* (Elementtext),[1]
  - *Element Content* (Elementinhalt),
  - *Mixed Content* (gemischter Inhalt).

[1] Der Standard redet hier nur von *Content*.

XML-Dokument

```
<Mondial>
  <Land LCode = "D">
    <LName>Germany</LName>
    <Provinz>
      <PName>Baden</PName>
      <Flaeche>15</Fläche>
      <Stadt>
        <SName>Freiburg</SName>
        <Einwohner>198</Einwohner>
      </Stadt>
      <Stadt>
        <SName>Karlsruhe</SName>
        <Einwohner>277</Einwohner>
      </Stadt>
    </Provinz>
    <Provinz>
      <PName>Berlin</PName>
      <Flaeche>0,9</Fläche>
      <Stadt>
        <SName>Berlin</SName>
        <Einwohner>3472</Einwohner>
      </Stadt>
    </Provinz>
    <Lage>
      <Kontinent>Europe</Kontinent>
      <Prozent>100</Prozent>
    </Lage>
    <Mitglied Organisation = "EU" Art = "member"/>
  </Land>
</Mondial>
```

Element mit gemischtem Inhalt

```
<Stadt>
    <SName>Freiburg i.Br.</SName>
    Eine der schönsten Städte Deutschlands.
    <Einwohner>198</Einwohner>
    Viele darunter sind ökologisch orientiert.
</Stadt>
```

Attribute

- Attribute:
  <aTagname attr$_1$ = "val$_1$"...attr$_k$ = "val$_k$">, $k \geq 1$.
- leeres Element:
  <aTagname attr$_1$ = "val$_1$"...attr$_k$ = "val$_k$"/>, $k \geq 0$.

  Beispiel:
  <Mitglied Organisation = "EU" Art = "member"/>.

wohlgeformte Elementstruktur

Eine Elementstruktur ist *wohlgeformt*,

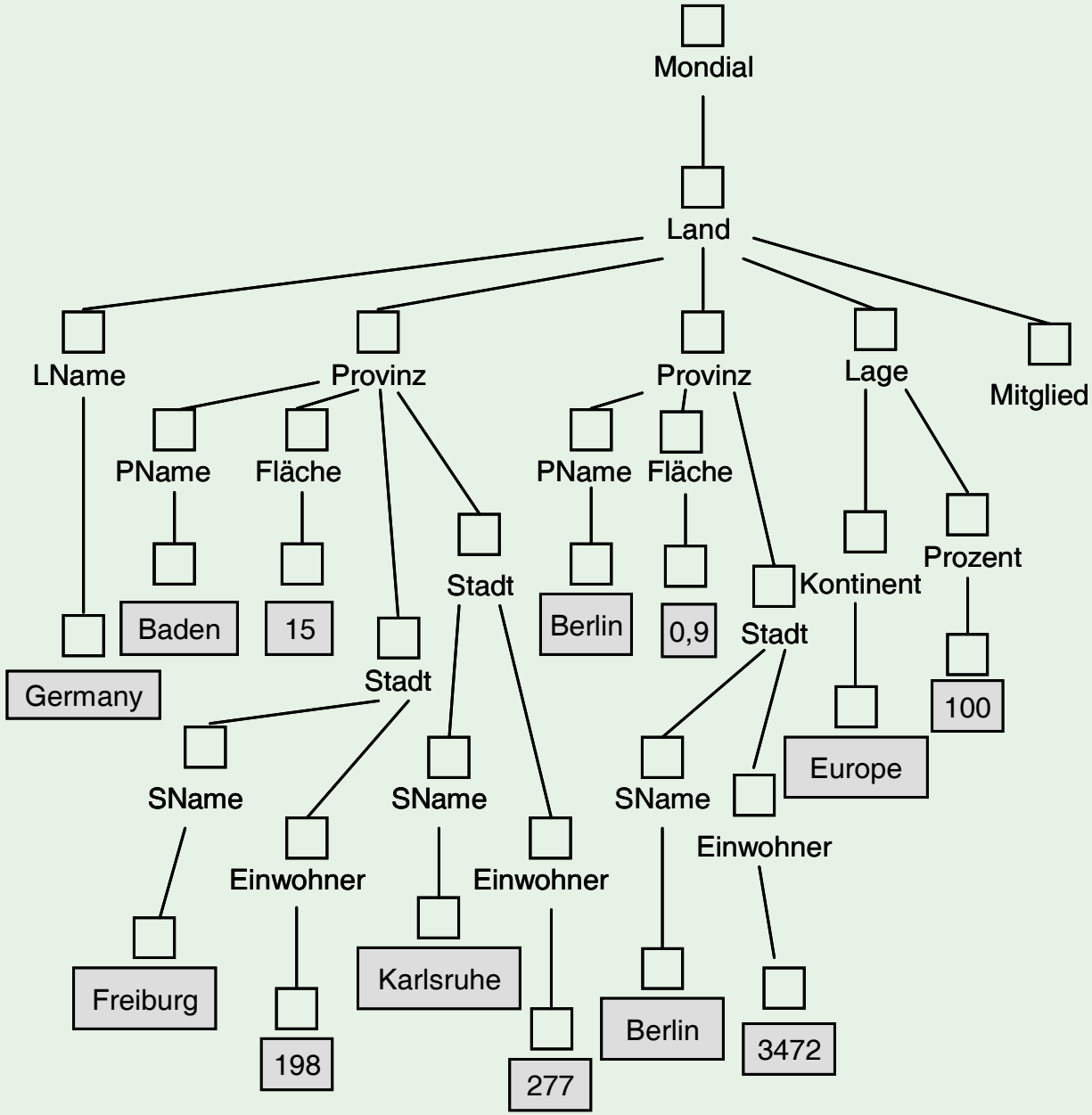
- wenn für je zwei Elemente mit Namen $\texttt{EName}_1$, $\texttt{EName}_2$ gilt: wenn $<\texttt{EName}_1>$ im Dokument vor $<\texttt{EName}_2>$, dann entweder auch $</\texttt{EName}_1>$ im Dokument vor $<\texttt{EName}_2>$, oder $</\texttt{EName}_2>$ im Dokument vor $</\texttt{EName}_1>$.
- Genau ein Element existiert, das in keinem anderen enthalten ist, *Dokumentelement*. Es ist das *Wurzelelement* des Dokumentes.

XML-Baum

Die Struktur eines XML-Dokumentes kann durch einen Baum, genannt *XML-Baum*, dargestellt werden.

- Jedem Element des Dokumentes und jedem Elementtext wird ein Knoten zugeordnet.
- Das Dokumentelement ist die Wurzel des XML-Baumes.
- Ein Knoten p ist Elterknoten eines Knotens p', wenn das zu p' gehörende Element E' *direkt* in dem zu p gehörenden Element E enthalten ist, oder p' für den Elementtext von p steht.
- Ist ein Knoten mit einem Elementtext beschriftet, so bezeichnen wir ihn als *Textknoten* und seine Beschriftung als *Textinhalt*. Alle anderen Knoten sind *Elementknoten*.

XML-Baum


Mondial
Land
LName
Provinz
Provinz
Lage
Mitglied
PName
Fläche
PName
Fläche
Stadt
Kontinent
Prozent
Baden
15
Berlin
0,9
Stadt
Germany
Stadt
100
SName
SName
SName
Europe
Einwohner
Einwohner
Einwohner
Freiburg
Karlsruhe
Berlin
198
277
3472

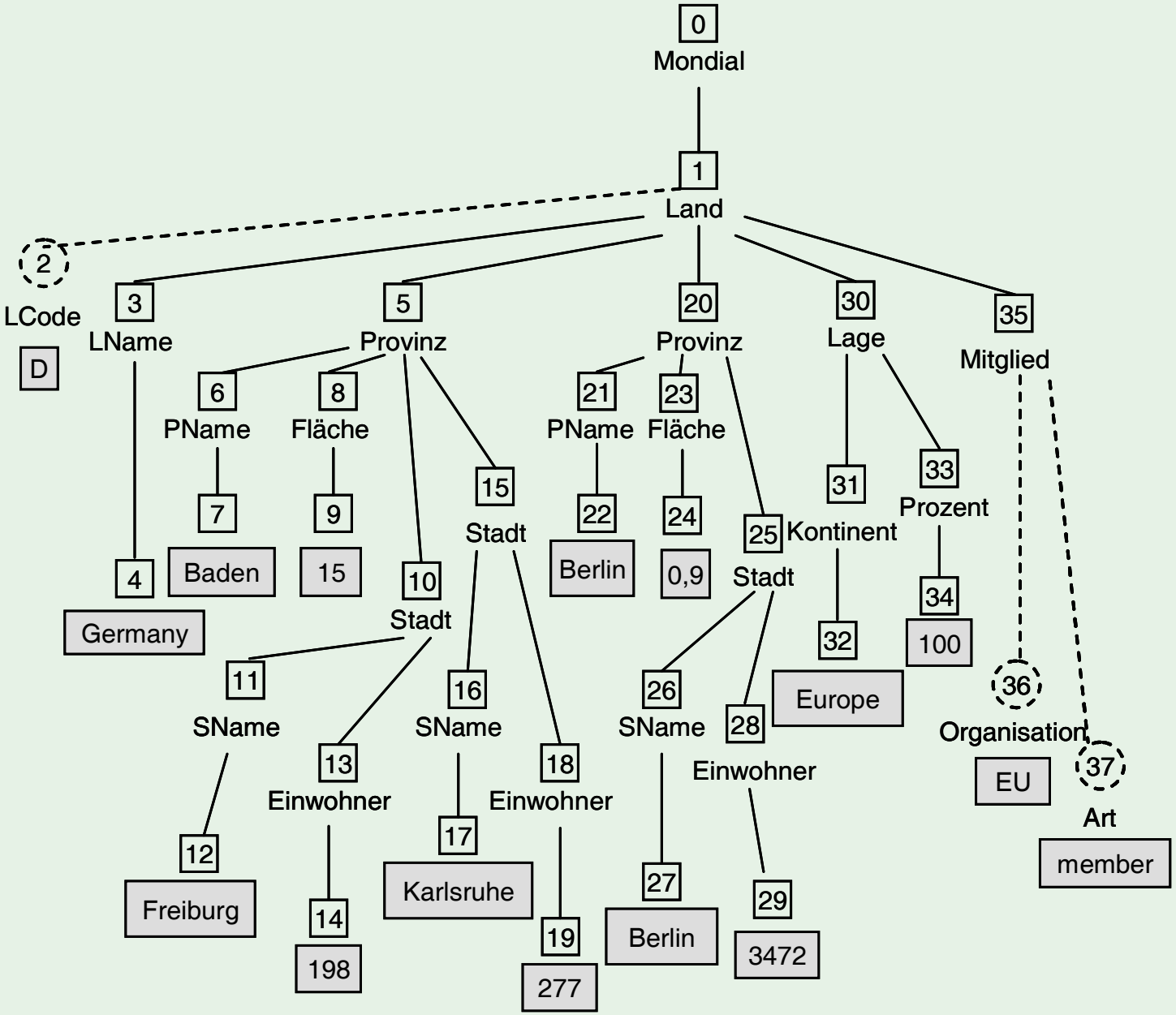
Dokumentordnung

- Die *Dokumentordnung* ergibt sich aus der Reihenfolge der öffnenden Tags.
- Ein XML-Baum ist *geordnet*, wenn die Tiefensuche des Baumes gerade die Dokumentordnung ergibt.
- Zu einem geordneten XML-Baum kann immer eine textuelle Darstellung, eine sogenannte *Serialisierung*, angeben werden, die der ursprünglichen Dokumentordnung entspricht.

Attribute

- Ein Element darf zu jedem Attribut maximal einen Wert haben.
- Die Attribute eines Elementes sind zueinander ungeordnet. Die Dokumentordnung auf den entsprechenden Elementen ist für die Attribute untereinander ohne Bedeutung.
- Attribute sind nicht Teil eines XML-Baumes. Aus Gründen der Einheitlichkeit werden im Folgenden auch Attribute als Knoten eines XML-Baumes betrachtet.
- XML-Baum mit Attributknoten ist ein *erweiterter* XML-Baum.

erweiterter XML-Baum



XML ist mehr

Processing Instructions, *Entity Declarations* und *Name Spaces* betrachten wir nicht.

XML-Prozessor

- Den Zugang zu Inhalt und Struktur eines XML-Dokumentes ermöglicht der *XML-Prozessor*.
- Mit diesem Modul kann mindestens die Wohlgeformtheit eines Dokumentes überprüft werden.
- Es ermöglicht des Weiteren das Parsen des Dokumentes.
  - Baumorientierter (DOM-)Parser,
  - Ereignisorientierter (SAX-)Parser.

5.2 Definition von Dokumenttypen (DTD)

Elementtypen und Attributtypen

- Eine DTD legt die zulässige Struktur eines Dokumentes fest.
- Es werden *Elementtypen* und *Attributtypen* definiert.
- Ein Dokument, das konform zu seiner DTD ist, heißt *gültig (valid)*.
- XML verlangt zwingend, dass Dokumente wohlgeformt sind; Gültigkeit ist eine optionale Eigenschaft.
- Die Gültigkeit eines Dokumentes, relativ zu einer betrachteten DTD, kann durch den *XML-Prozessor* überprüft werden.

Definition von Typen

- Eine DTD ist formal eine Grammatik.
- Definition von Elementtypen:

  <!ELEMENT EName Content>,
- Definition von Attributtypen

  <!ATTLIST EName $\texttt{Attr}_1$ $\texttt{AttrType}_1$ $\texttt{Default}_1$ ... $\texttt{Attr}_k$ $\texttt{AttrType}_k$ $\texttt{Default}_k$>.
- Elementtypen sind global innerhalb einer DTD. Attributdefinitionen sind an ein Element gebunden und somit für dieses lokal.

Content

- `EMPTY`: Elemente von diesem Typ haben keinen Inhalt.
- `ANY`: Es sind beliebiger Elementtext oder beliebiger anderweitig in der DTD definierter Elementinhalt erlaubt.
- `(#PCDATA)`: Elemente von diesem Typ haben als Inhalt Elementtext. `#PCDATA` steht für *Parsed Character Data*; die einen Elementtext repräsentierende Zeichenkette wird nicht in Hochkommata eingefasst.
- Ein durch $'('$ und $')'$ umfasster regulärer Ausdruck über Elementtypen, der eine innere Struktur zu `EName` definiert, das *Inhaltsmodell*. Existiert zu einem Element ein Inhaltsmodell, so hat das Element *Elementinhalt*.

Inhaltsmodell

- Das Inhaltsmodell wird durch einen regulären Ausdruck α definiert, der gemäß der folgenden Regeln gebildet ist:
  - Jeder Elementtyp *E* ist ein Inhaltsmodell α.
  - Wenn α ein Inhaltsmodell ist, dann sind auch $\alpha*$, $\alpha+$ und $\alpha?$ Inhaltsmodelle.
  - Wenn α_1, α_2 Inhaltsmodelle sind, dann sind auch $(\alpha_1 \mid \alpha_2)$ und (α_1, α_2) Inhaltsmodelle.
- Die Zeichen $'|'$, $','$, $'*'$, $'+'$ und $'?'$ innerhalb eines Inhaltsmodells stehen für die zur Bildung des Elementinhaltes anwendbaren Operatoren:
  - Auswahl zwischen α_1 und α_2,
  - Konkatenation (Sequenz) von α_1 und α_2,
  - beliebig häufige (einschließlich keinmal) Konkatenation von α mit sich selbst,
  - beliebig häufige (ausschließlich keinmal) Konkatenation,
  - optionales Auftreten (höchstens einmal) von α.

Beispiel

```
<!DOCTYPE Mondial[
  <!ELEMENT Mondial (Land+)>
  <!ELEMENT Land (LName?, Provinz*, Lage*, Mitglied*)>
  <!ELEMENT LName (#PCDATA)>
  <!ELEMENT Provinz (PName, Flaeche, Stadt*)>
  <!ELEMENT PName (#PCDATA)>
  <!ELEMENT Flaeche (#PCDATA)>
  <!ELEMENT Stadt (SName, Einwohner)>
  <!ELEMENT SName (#PCDATA)>
  <!ELEMENT Einwohner (#PCDATA)>
  <!ELEMENT Lage (Kontinent, Prozent)>
  <!ELEMENT Kontinent (#PCDATA)>
  <!ELEMENT Prozent (#PCDATA)>
  <!ELEMENT Mitglied EMPTY>

  <!ATTLIST Land LCode ID #REQUIRED>
  <!ATTLIST Mitglied Organisation CDATA #REQUIRED Art CDATA "member">
]>
```

- Die Definition einer DTD wird durch <!DOCTYPE aName[und]> umfasst, wobei aName den Elementtyp des Wurzelelementes festlegt.
- Überflüssige Klammerungen innerhalb eines Inhaltsmodells lassen wir weg; anstatt $((\alpha_1, \alpha_2), \alpha_3)$ schreiben wir beispielsweise $(\alpha_1, \alpha_2, \alpha_3)$.
- Die Regeln einer DTD können *rekursiv* sein. Damit können beispielsweise auch Zusammenhänge der Form einer Teilehierarchie <!ELEMENT Teil (Teil*)> oder auch einer Baumstruktur <!ELEMENT Baum (Baum+ | Blatt)> definiert werden.

Attribute

- Definition des Typs `attrType` des Attributes `attr` eines Elementtyps `EName`

  `<!ATTLIST EName` $\texttt{attr}_1$ $\texttt{attrType}_1$ $\texttt{default}_1$ `...` $\texttt{attr}_k$ $\texttt{attrType}_k$ $\texttt{default}_k$`>`
- `attrType` ist von einer der folgenden Formen:
  - `CDATA`: Die zulässigen Werte des Attributs `attr` sind beliebige Zeichenketten, die in Hochkommata eingeschlossen werden.
  - `("val`$_1$`"| ...| "val`$_n$`")`: Die zulässigen Werte sind aufgezählt.
  - `ID`: Die Werte des Attributes sind über dem ganzen Dokument identifizierend. Die `ID`-Werte aller Vorkommen (beliebiger) Attribute vom Typ `ID` sind eindeutig.
  - `IDREF`: Der Wert des Attributes ist der `ID`-Wert eines anderen Attributes; er referenziert ein anderes Element innerhalb des Dokumentes.
  - `IDREFS`: Der Wert des Attributes ist eine Folge von `IDREF`-, d.h. `ID`-Werten.

- `default` ist von einer der folgenden Formen:
  - `#REQUIRED`: Das Attribut muss explizit aufgeführt werden.
  - `#IMPLIED`: Das Attribut ist optional; es ist kein Default-Wert vorgesehen.
  - `val`: Ist das Attribut im Dokument selbst nicht aufgeführt, dann wird durch den XML-Prozessor dieser Wert wie ein *Standardwert (Default-Wert)* für das betreffende Attribut angenommen.
  - `#FIXED val`: Das Attribut muss immer den angegebenen Wert haben.

Bemerkungen

- Eine DTD bietet an Datentypen lediglich einen allgemeinen Datentyp für Zeichenketten an.
- Datenbanken bieten einen reichhaltigen Vorrat an Basis-Datentypen zur Repräsentation von Zahlen, Zeichenketten, oder auch zeitlichen Zusammenhängen an.
- Es kann nicht unterschieden werden, ob `#PCDATA` der Inhalt einer beliebigen Zeichenkette, eine Zahl, oder ein Datum ist. Entsprechendes gilt für den Datentyp `CDATA` zur Darstellung von Attributwerten.
- Attributwerte können vom Typ `ID`, `IDREF` oder auch `IDREFS` sein. Der XML-Prozessor gewährleistet, dass `ID`-Werte im gesamten Dokument eindeutig sind und dass unter den `IDREF`-Werten nur solche sind, die als `ID`-Wert im Dokument auftauchen.
- Es ist nicht ausdrückbar, welcher Elementtyp mittels `IDREF` referenziert werden soll. Die Qualität der Referenzierung ist somit nicht vergleichbar zur referentiellen Integrität in Datenbanken.

5.3 XPath

- XPath ist primär eine Sprache zum Addressieren (*Lokalisieren*) von Teilen eines erweiterten XML-Baumes mittels Pfaden.
- Zunächst Version 1.0 von XPath; später Unterschiede zu Version 2.0.
- Pfadausdrücke in XPath werden *Lokationspfade* genannt.
- Ein Lokationspfad besteht aus einer Folge von *Lokationsschritten*.
- Jeder Lokationsschritt enthält eine *Achse*, einen *Knotentest* und ein oder mehrere *Prädikate*.

Welche Knoten werden lokalisiert?

(1) `/child::Mondial/child::Land`

(2) `/child::Mondial/child::Land[child::LName = "Germany"]`

(3)
```
/child::Mondial/child::Land[child::LName = "Germany"]/
      child::Provinz
```

- Ein *Lokationspfad* in XPath wird relativ zu einem *Kontext* ausgewertet und liefert als Resultat eine Menge von Knoten.
- Kontext: *Kontextknoten*; später *Kontextposition*, *Kontextgröße*.
- Ein Lokationspfad heißt *absolut*, wenn ihm '/' vorangestellt ist und anderenfalls *relativ*.
- Der Kontextknoten eines absoluten Lokationspfades ist der Wurzelknoten des XML-Dokumentes; der Kontextknoten eines relativen Lokationspfades wird anderweitig gegeben.
- Der *Wurzelknoten des XML-Dokumentes* repräsentiert Eigenschaften des gesamten XML-Dokumentes und verweist auf das Dokumentelement (Wurzelelement) des XML-Dokumentes, d.h. den *Wurzelknoten des betreffenden XML-Baumes*.

Lokationspfad

- Ein Lokationspfad P besteht aus einer durch '/' getrennten Folge von *Lokationsschritten*.
- Eine solche Folge wird von links nach rechts induktiv ausgewertet.
- Das Ergebnis der Auswertung ist die durch den Pfad P *lokalisierte* Menge von Knoten K_P des betrachteten XML-Baumes.

Menge der lokalisierten Knoten

- Sei n die Anzahl Schritte eines Lokationspfades, $n \geq 1$.
- Sei p_i, $1 \leq i \leq n$, ein Lokationsschritt.
- Sei L_{i-1} die Menge der durch den Lokationsschritt p_{i-1} lokalisierten Knoten; ist $i = 1$, dann gilt $L_0 = \{r\}$, wobei r der Wurzelknoten.
- Sei $k \in L_{i-1}$. Der Lokationsschritt p_i *lokalisiert* in Abhängigkeit von k eine Menge von Knoten $L_i(k)$.
- Jeder Knoten $k' \in L_i(k)$ ist ein zu betrachtender Kontextknoten für den folgenden Lokationsschritt p_{i+1}.
- Die Menge der zu betrachtenden Kontextknoten für p_{i+1} ist somit

$$L_i = \bigcup_{k \in L_{i-1}} L_i(k).$$

- Ist p_i ist der letzte Lokationsschritt eines Lokationspfades P, dann gilt $K_P = L_i$.

Lokationsschritt

- Ein Lokationsschritt besteht aus einer *Achse*, einem *Knotentest* und, optional, aus einem oder mehreren *Prädikaten*: *Achse::Knotentest[Prädikat]*.
- Achse und Knotentest legen die durch diesen Schritt lokalisierte Menge von Knoten des XML-Baumes relativ zu dem betrachteten Kontextknoten des Lokationsschrittes, zur Struktur des XML-Baumes und zum Typ der Knoten fest.
- Prädikate wirken als ein zusätzlicher Filter.

Achse	definierte Menge von Knoten
`attribute`	alle Attribute des Kontextknotens
`child`	alle direkten Nachfolger des Kontextknotens
`descendant`	alle direkten und indirekten Nachfolger des Kontextknotens
`descendant-or-self`	wie `descendant`, jedoch einschließlich Kontextknoten
`parent`	der direkte Vorgänger des Kontextknotens
`ancestor`	alle Knoten, außer dem Kontextknoten, auf dem Pfad vom Kontextknoten zur Wurzel
`ancestor-or-self`	wie `ancestor`, jedoch einschließlich Kontextknoten
`following`	alle Knoten, die in der Dokumentordnung nach dem Kontextknoten stehen, jedoch außer den durch `descendant` definierten Knoten
`following-sibling`	alle Knoten, die denselben direkten Vorgänger haben wie der Kontextknoten und in der Dokumentordnung nach dem Kontextknoten stehen
`preceding`	alle Knoten, die in der Dokumentordnung vor dem Kontextknoten stehen, jedoch außer den durch `ancestor` definierten Knoten
`preceding-sibling`	alle Knoten, die denselben direkten Vorgänger haben wie der Kontextknoten und in der Dokumentordnung vor dem Kontextknoten stehen
`self`	Kontextknoten

Achse und Knotentest

- Die *Achsen* eines Lokationspfades legen fest, in welcher Weise der XML-Baum durchlaufen werden soll.
- Relativ zu den jeweiligen Kontextknoten der einzelnen Schritte des Pfades wird die Menge der für die Lokalisierung weiter zu betrachtenden Knoten festgelegt.
- Eine Achse ist eine *Vorwärtsachse*, wenn ihre vom Kontextknoten verschiedenen Knoten in der Dokumentordnung nach dem Kontextknoten stehen;
- sie ist eine *Rückwärtsachse*, wenn ihre vom Kontextknoten verschiedenen Knoten vor dem Kontextknoten stehen.
- Die Achse `self` ist gleichermaßen Vorwärts- und Rückwärtsachse.

Lokalisation durch Achse und Knotentest

- Knotentest eine Attribut oder Elementtyp:
  - Achse `attribute`: aus der Menge der Attributknoten der Achse wird nur der Knoten des Attributs mit dem als Knotentest angegebenen Namen weiter betrachtet.
  - Achse sonstig: die Knoten der Achse werden auf diejenigen eingeschränkt, deren Elementtyp gerade gleich dem als Knotentest gewählten Elementtyp ist.
- Knotentest '$*$', dann ist jedes Attribut, bzw. jeder Elementtyp, zulässig,
- Knotentest `text()` schränkt die Knoten auf alle Textknoten ein,
- Knotentest `node()` bedeutet keine Einschränkung.

Welche Knoten werden lokalisiert?

(4) `/descendant::Stadt`

(5) `/descendant::*[self::Stadt]`

Lokalisiere die Knoten der Städte in Baden.

(6) .....

(7) .....

Lokalisiere alle Attributknoten Organisation mit Wert "EU".

(8)

.....

(9) `/descendant::Stadt/descendant::text()`

Kurzschreibweise

- `self::node()`: '.',
- `parent::node()`: '..',
- `attribute::`: '@',
- Angabe einer Achse fehlt: `child`-Achse,
- Pfad enthält '//': `/descendant-or-self::node()/`.

zur Kurzschreibweise.

(10) `//Stadt`

(11) `//*[self::Stadt]`

(12) `//Stadt[../PName = "Baden"]`

(13) `//Provinz[PName = "Baden"]/Stadt`

(14) `//@Organisation[. = "EU"]`

(15) `//Stadt//text()`

Prädikate

- Prädikate sind XPath-Ausdrücke, denen ein Wahrheitswert zugeordnet werden kann.
- Jedem Lokationspfad kann ein Wahrheitswert zugeordnet werden; er hat den Wert `true()`, wenn er eine nicht-leere Menge von Knoten lokalisiert und ansonsten den Wert `false()`.
- Wird ein Lokationspfad innerhalb eines Vergleichsprädikates verwendet, dann wird jeder Knoten durch seinen *String-Value* ersetzt; für einen Elementknoten ergibt sich der String-Value aus der Konkatenation aller Inhalte der Textknoten in Dokumentordnung, die in dem durch ihn identifizierten Teilbaum liegen.
- Prädikate können über Konjunktionen und Disjunktionen von XPath-Ausdrücken definiert werden.

Welche Knoten werden lokalisiert?

(16) `//Stadt[Einwohner > 500]`

(17) `//Stadt[Einwohner]`

(18) `//Stadt[../PName = //Stadt[SName = "Freiburg"]/../PName]`

(19) `//Provinz[Stadt/SName != "Freiburg"]`

Lokalisiere alle Knoten der Provinzen, die keine Stadt mit Namen "Freiburg"enthalten.

(20) .....

weitere Beispiele zu Prädikaten

- `//*[@Art | @LCode]`
- `//*[@Art and @Organisation]`
- `//*[@Art][@Organisation]`

Kontextposition

- Auswertung nicht nur relativ zu einem Kontextknoten, sondern auch relativ zur *Kontextposition* und zur *Kontextgröße*.
- Grundlage für Kontextgröße und Kontextposition ist die durch Achse und Knotentest eines Lokationsschrittes bzgl. eines Kontextknotens definierte Knotenmenge, auf die die Prädikate angewendet werden sollen. Die Zählung der Knoten beginnt dabei bei 1.
- Die *Kontextgröße* ist gegeben durch die Mächtigkeit der Knotenmenge.
- Die *Kontextposition* durch die Position eines Knotens in dieser Menge relativ zur Dokumentordnung. Ist die betrachtete Achse eine Rückwärtsachse, dann wird die umgekehrte Dokumentordnung betrachtet.

Welche Knoten werden lokalisiert?

(21) `//Stadt[position() = 2]`

(22) `//Land/Provinz[1]/Stadt[last()]`

Lokalisiere den Knoten derjenigen Stadt, die im Dokument direkt auf Freiburg folgt.

(23) .....

(24) `//Stadt[(preceding::Stadt)[1]/SName = "Freiburg"]/SName`

XPath Version 1.0 und 2.0

- XPath ist weniger geeignet als Anfragesprache; jeder lokalisierte Knoten identifiziert einen Teilbaum des XML-Baumes, dieser Teilbaum kann jedoch nicht in seiner Struktur verändert werden.
- Im Zusammenhang mit der Standardisierung von XQuery wurde mit XPath Version 2.0 eine in einigen Punkten grundlegend geänderte und erweiterte Version von XPath entwickelt.
- Das Konzept einer Knotenmenge (*Node Set*) wird in der Version 2.0 durch das allgemeinere Sprachkonzept einer *Sequenz* ersetzt.
- In einigen Fällen ergeben sich unterschiedliche Semantiken von XPath Version 1.0 Ausdrücken, wenn sie nach der Version 2.0 interpretiert werden.
- Darüber hinaus enthält XPath Version 2.0 eine Reihe von Sprachkonstrukten wie `if...then...else`, oder auch `for`, die keine Entsprechung in Version 1.0 haben.

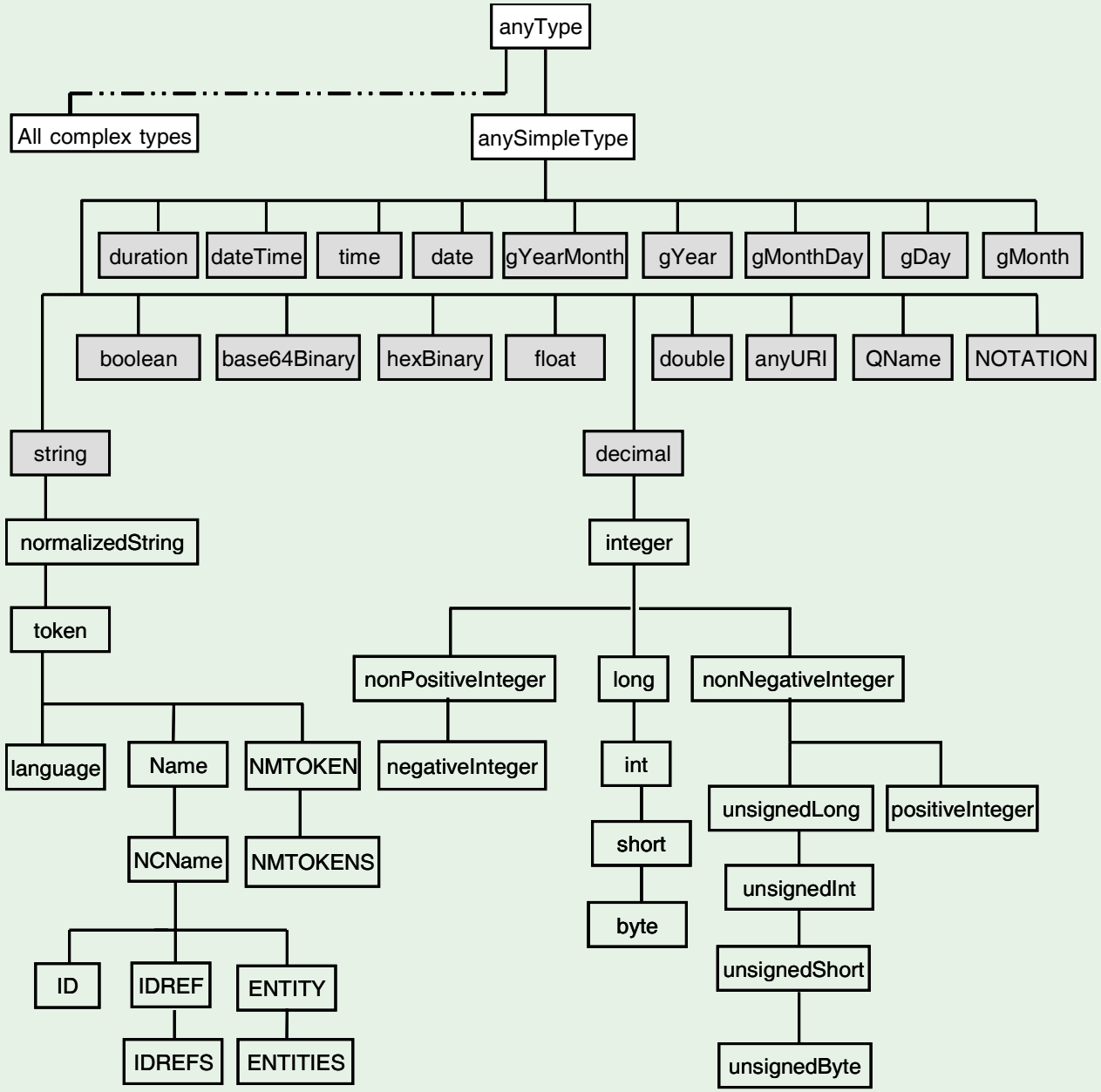
5.4 XML-Schema

- XML-Schema behält die prinzipielle Mächtigkeit einer DTD zur Definition von Inhaltsmodellen bei und ergänzt diese um eine reichhaltige Möglichkeit, Datentypen zu definieren.
- Es werden Typdefinitionen zur Verwendung in XML-Dokumenten bereitgestellt, gegen die die Dokumente dann validiert werden können.
- Schemata gemäß XML-Schema sind selbst XML-Dokumente.
- Wir beschränken uns auf vor einem Datenbankhintergrund wesentliche Zusammenhänge.

Elementtypen

- XML-Schema beruht auf einem erweiterbaren Typkonzept.
- Eine Anzahl von Datentypen ist *vordefiniert*, sogenannte *Built-in Types*.
  - *primitive*,
  - *abgeleitete*.
- Basierend auf den primitiven Datentypen können neue Typen definiert werden.
  - *einfache*, d.h. Typen ohne Attribute und Kindelemente,
  - *komplexe*, d.h. Typen mit Attributen und Kindelementen.

XML-Schema Datentypen



einfache Datentypen

- *Wertebereich*, *Repräsentationsraum* und *Aspekte*.
- Der Repräsentationsraum eines Datentyps sieht zu jedem Wert des Wertebereichs mindestens eine lexikalische Darstellung (*Literal*) vor.
- Mehr Literale als Werte: beipielsweise Zeittypen oder Gleitkommazahlen (100 oder 10E1).
- Wertebereich und Repräsentationsraum können durch *Aspekte* mittels eines regulären Ausdrucks eingeschränkt werden: Unter-/Obergrenzen der Werte, Längenbeschränkungen und Aufzählungen der erlaubten Werte.
- Einfache Typen können aus einfachen Typen mittels Listenbildung (`list`) und Vereinigung (`union`) erzeugt werden.

Beispiele für einfache Datentypen

```
<simpleType name = "Prozent">
    <restriction base = "number">
       <fractionDigits value = "2"/>
       <minInclusive value = "0.00"/>
       <maxInclusive value = "100.00"/>
    </restriction>
</simpleType>
<simpleType name = "Autonummer">
    <restriction base = "string">
       <pattern value = "[A-Z]+[0-9]+"/>
    </restriction>
</simpleType>
<simpleType name = "LandCode">
    <restriction base = "NMTOKEN">
       <enumeration value = "D"/>
       <enumeration value = "A"/>
       <enumeration value = "CH"/>
    </restriction>
</simpleType>
<simpleType name = "AlleAutonummern">
    <list itemType = "Autonummer"/>
</simpleType>
```

Attribute

- Attribute dürfen lediglich einfache Typen besitzen.
- Ein Attribute `attr` eines Elementtyps kann mittels der Attribute `use, default` und `fixed` weiter charakterisiert werden.
  - `use` kann die Werte `optional,prohibited` und `required` annehmen.
    Wird `use` nicht definiert, so wird `optional` angenommen.
  - Die Verwendung von `default` und `fixed` ist alternativ.

Beispiel

```
<attribute name = "LCode" type = "string"
    use = "required" default = "Atlantis"/>
```

komplexe Elementtypen

- Erweiterung eines einfachen Typs, oder durch Definition eines Inhaltsmodells aus einfachen und komplexen Typen.
- Inhaltsmodelle werden mittels Reihung (`sequence`), Auswahl (`choice`) und Konjunktion (`all`) definiert; Kardinalitäten werden durch `minOccurs` und `maxOccurs` ausgedrückt.
- Bei Verwendung von `all` dürfen alle angegebenen Typen maximal einmal in beliebiger Reihenfolge auftreten.
- Mittels `minOccurs = "0"`, kann ein Element als optional für ein Inhaltsmodell definiert werden.

  Mittels `nillable = "true"` und innerhalb eines Dokumentes `nil = "true"` wird ein Nullwert *explizit* ausgedrückt.

globale Elemente und Typen

- Typdefinitionen und Elementdeklarationen auf der obersten Ebene eines XML-Schema Dokumentes sind *global*.
- Globale Elementdeklarationen können mittels `ref` referenziert werden.
- Globale Typdefinitionen haben einen Namen und können zur Deklaration von Elementen verwendet werden.
- Typdefinitionenohne einen Namen sind *anonym*.

Beispiel

```
<complexType name = "Einwohner">
    <simpleContent>
       <extension base = "positiveInteger">
          <attribute name = "Jahr" type = "gYear"
             use = "required"/>
       </extension>
    </simpleContent>
</complexType>
<complexType name = "Land">
    <sequence>
       <element name = "LName" type = "string" nillable = "true"/>
       <element name = "Einwohner" type = "Einwohner"/>
       <element ref = "Provinz" minOccurs = "0" maxOccurs = "unbounded"/>
       <element ref = "Lage" minOccurs = "0" maxOccurs = "unbounded"/>
       <element ref = "Mitglied" minOccurs = "0" maxOccurs = "unbounded"/>
    </sequence>
    <attribute name = "LCode">
       <simpleType>
          <restriction base = "NMTOKEN">
             <enumeration value = "D"/>
             <enumeration value = "A"/>
             <enumeration value = "CH"/>
          </restriction
       </simpleType>
    </attribute>
</complexType>
```

Eindeutigkeitsbedingungen

- Eindeutigkeit kann für Element- und Attributinhalte, oder auch für Kombinationen hiervon, verlangt werden.
- Die `unique`-Klausel verlangt Eindeutigkeit für vorhandene Werte.
- Bei Anwendung der `key`-Klausel muss ein Wert vorhanden und eindeutig sein.

- Eindeutigkeit bezieht sich relativ auf das direkt übergeordnete Element (Kontext), in dem die Bedingung definiert ist.
  - Mittels `selector` wird die zu betrachtende Knotenmenge des Dokumentes relativ zum Kontext festgelegt.
  - Mittels `field` wird die Kombination der konkreten Werte relativ zum Selektor festgelegt.
  - Mittels `field` lokalisierte Elemente müssen einen einfachen Typ besitzen.
- `selector` und `field` werden durch eingeschränkte XPath-Ausdrücke – im Wesentlichen dürfen nur Vorwärtsachsen verwendet werden, es sind keine Prädikate zulässig und als Knotentest sind `node()` und `text()` nicht erlaubt – festgelegt.
- Eine Eindeutigkeitsbedingung ist erfüllt, wenn die Werte der durch `field` lokalisierten Knoten bzgl. der Menge der durch `selector` lokalisierten Knoten eindeutig sind.

Beispiel

```
<element name = "Provinz">
    <complexType>
       <sequence>
          <element name = "PName" type = "string"/>
          <element name = "Fläche" type = "float"/>
          <element name = "Stadt"
             minOccurs = "0" maxOccurs = "Unbounded"/>
       </sequence>
    </complexType>
</element>
<element name = "Land" type = "Land"/>
<element name = "Mondial">
    <complexType>
       <sequence>
          <element ref = "Land"
             minOccurs = "0" maxOccurs = "Unbounded"/>
       </sequence>
    </complexType>
    <key name = "PrimaryKeyForLand">
       <selector xpath = "Land"/>
       <field xpath = "@LCode"/>
    </key>
    <key name = "PrimaryKeyForProvinz">
       <selector xpath = "Land/Provinz"/>
       <field xpath = "PName"/>
    </key>
</element>
```

Fremdschlüsselbedingungen

- Referenzbeziehung werden mittels der `keyref`-Klausel definiert.
- Mit `refer` wird der Bezug zu einem eindeutigen Kriterium, typischerweise ein Schlüssel, festgelegt.
- Mittels `selector` und `field` wird der Fremdschlüssel festgelegt.
- Der Kontext der Fremdschlüsseldefinition muss hierbei den Kontext der betreffenden Schlüsseldefinition enthalten.
- XML-Schema erlaubt auch den ID/IDREF-Mechanismus einer DTD. `ID`, `IDREF` und `IDREFS` sind zulässige einfache vordefinierte Typen und haben dieselbe Semantik wie bei Verwendung einer DTD.

Beispiel

```
<element name = "Land"><complexType><sequence>
          <element name = "LName" type = "string" nillable = "true"/>
          <element name = "Einwohner" type = "Einwohner"/>
          <element ref = "Lage" minOccurs = "0" maxOccurs = "Unbounded"/>
          <element ref = "Mitglied" minOccurs = "0" maxOccurs = "Unbounded"/>
       </sequence>
       <attribute name = "LCode" type = "string"/>
</complexType></element>
<element name = "Provinz"><complexType><sequence>
          <element name = "PName" type = "string"/>
          <element name = "Stadt" minOccurs = "0" maxOccurs = "Unbounded"/>
       </sequence>
       <attribute name = "LCode" type = "string"/>
</complexType></element>
<element name = "Mondial"><complexType><sequence>
          <element ref = "Land" minOccurs = "0" maxOccurs = "Unbounded"/>
          <element ref = "Provinz" minOccurs = "0" maxOccurs = "Unbounded"/>
    </sequence></complexType>
    <key name = "PrimaryKeyForLand">
       <selector xpath = "Land"/><field xpath = "@LCode"/></key>
    <key name = "PrimaryKeyForProvinz">
       <selector xpath = "Provinz"/>
       <field xpath = "PName"/><field xpath = "@LCode"/></key>
    <keyref name = "ForeignKeyForLand" refer = "PrimaryKeyForLand">
       <selector xpath = "Provinz"/><field xpath = "@LCode"/></keyref>
</element>
```

Bemerkungen

- Innerhalb XML-Schema können keine allgemeinen Integritätsbedingungen formuliert werden.
- Schlüssel- und Fremdschlüsselbedingungen gehen über den ID/IDREF-Mechanismus weit hinaus, haben jedoch nur innerhalb eines Dokumentes eine Bedeutung.

5.5 XQuery

Die Ziele von *XML Query*, kurz *XQuery*, sind prägnant beschrieben auf der Homepage des XQuery-Projektes:[2]

The mission of the XML Query project is to provide flexible query facilities to extract data from real and virtual documents on the World Wide Web, therefore finally providing the needed interaction between the Web world and the database world. Ultimately, collections of XML files will be accessed like databases.

[2] http://www.w3.org/XML/Query

Allgemeines

- XQuery ist, vergleichbar zur Relationenalgebra, eine funktionale Sprache.
- Eine Anfrage in XQuery wird deklarativ als Ausdruck definiert, der von einem XQuery-System zu einem Wert, der *Antwort*, ausgewertet werden kann.
- XQuery ist darüber hinaus eine streng getypte Sprache, so dass Typfehler bereits zur Übersetzungszeit erkannt werden können.
- Der Wert eines XQuery-Ausdrucks ist eine *Sequenz* von keinem oder mehreren sogenannten *Items*.
- Ein Item ist entweder ein atomarer Wert, d.h. ein Wert von einem einfachen Typ gemäß XML-Schema, oder ein Knoten (Element-, Attribut- und Textknoten).

Kreieren von Elementknoten

- Der Ausdruck

```
<Stadt PLZ = "79100">
{"Freiburg i.Br."}
</Stadt>
```

  erzeugt ein Element `Stadt` und weist ihm das Attribut `PLZ` mit Wert 79100 und den Elementtext `"Freiburg"` zu.

- Verwendung von *Konstruktoren* `element` und `attribute`:

```
element Stadt {attribute PLZ {"79100"}, "Freiburg i.Br."}
```

XPath 2.0

- Anfragen in Form von XQuery-Ausdrücken beziehen sich typischerweise auf XML-Bäume und extrahieren und kombinieren durch Knoten identifizierte Teilbäume.
- Zur Lokalisierung von Knoten eines gegebenen XML-Baumes verwendet XQuery XPath 2.0.
- Beispiel für Unterschiede zu XPath 1.0:

  Der Ausdruck `aName = true()` ist gemäß XPath 1.0 wahr, sofern `aName` eine nicht-leere Knotenmenge lokalisiert. Derselbe Ausdruck ist gemäß XPath 2.0 nur dann wahr, wenn die durch `aName` lokaliserte Menge mindestens ein Element enthält, das den Wahrheitswert `true` besitzt.

Beispiel

```
<Städte>
{ doc("Mondial.xml")//Stadt }
</Städte>
```

Der Ausdruck `doc("Mondial.xml")` lokalisiert hierbei den Wurzelknoten des XML-Dokumentes.

FLWOR-Ausdruck

- Ein FLWOR-Ausdruck besteht im Allgemeinen aus einer `for`-, `let`-, `where`-, `order`- und `return`-Klausel.
- Mittels einer `for`-Klausel werden eine oder mehrere Variablen an Ausdrücke gebunden und erzeugen so einen Strom von Tupeln.
- Durch eine `let`-Klausel werden eine oder mehrere Variable jeweils an das gesamte Resultat eines Ausdrucks gebunden.
- Die `where`-Klausel dient zu Filterung der erzeugten Tupel.
- Mittels der `order by`-Klausel kann eine erwünschte Sortierung der Tupel erreicht werden.
- Mittels der `return`-Klausel wird das Resultat des gesamten Ausdrucks definiert.

Beispiel

```
for $a in (1,2,3),
    $b in ("a", "b", "c"),
    $c in (10, 20, 30)
return
<Z>{$a, $b, $c}</Z>
```

Beispiel

```
<Städte>
{
  for $a in
    doc("Mondial.xml")//Provinz[PName = "Baden"]/Stadt/SName
  return
    <Stadt> { $a } </Stadt>
}
</Städte>
```

Beispiel

```
<Städte>
{
  let $a := doc("Mondial.xml")//Land[@LCode = "D"]
  for $b in $a//Provinz[PName = "Baden"]/Stadt/SName
  return
    <Stadt> { $b, element Land {$a/LName/text()} } </Stadt>
}
</Städte>
```

Beispiel

```
for $a in doc("Mondial.xml")//Provinz/Stadt
order by $a/SName descending
return
<Stadt>
  { $a/SName }
  <Provinz>
  {
    for $b in doc("Mondial.xml")//Provinz
    where $b/Stadt = $a
    return $b/PName
  }
  </Provinz>
</Stadt>
```

Beispiel

```
for $a in doc("Mondial.xml")//Provinz/Stadt,
  $b in doc("Mondial.xml")//Provinz/Stadt
where $a/../PName < $b/../PName
return
<Paar ProvinzA = "{ $a/../PName }"
  ProvinzB = "{ $b/../PName }">
  { $a/SName/text(), ", ", $b/SName/text() }
</Paar>
```

Quantoren some, every und bedingte Ausdrücke

```
for $a in doc("Mondial.xml")//Provinz
where some $b in $a//Stadt
satisfies ($b/Einwohner > 1000)
return $a/PName
```

```
for $a in doc("Mondial.xml")//Provinz
return
if (count($a/Stadt) > 1)
then
  <HauptProvinz>
  { $a/PName }
  </HauptProvinz>
else ()
```

Positionen einer Sequenz

```
for $a at $i in doc("Mondial.xml")//SName
return <Stadt Nummer = "{$i}">{$a}</Stadt>
```

Speichern von Zwischenresultaten

```
let $a :=
  for $b in doc("Mondial.xml")//Provinz[PName = "Baden"]/Stadt
    return
    <Stadt> { $b/SName } </Stadt>
let $c :=
  for $d in doc("Mondial.xml")//Provinz[PName = "Berlin"]/Stadt
    return
    <Stadt> { $d/SName } </Stadt>
return ($c, $a)
```

benutzerdefinierte Funktionen

Die Deklaration einer Funktion besteht aus einem *Funktionskopf*, der aus dem Funktionsbezeichner, der Liste der Parameter und der Angabe des Rückgabeparameters besteht, gefolgt von einem XQuery-Ausdruck, dem *Funktionsrumpf*.

```
declare function StadtmitNummer($a,$i)
as item()*
{
       let $b := ($a//SName)[$i]
       return
       (
          <Stadt Nummer = "{ $i }"> { $b } </Stadt>,
          if ($i < count($a//Stadt))
             then StadtmitNummer($a, $i+1) else ()
       )
};

for $c in doc("Mondial.xml")/Mondial/Land return
    ($c/LName, StadtmitNummer($c,1))
```

Sequenzen

- Werte sind *Sequenzen* von Items.
- Der Ausdruck `(1,"eins")` hat den Wert `1 eins`.
- Der Ausdruck

  ```
  (1, "eins", doc("Mondial.xml")//Land/@LCode)
  ```

  hat den Wert `1 eins LCode = "D"`.
- Ist eine Sequenz durch einen Ausdruck definiert, der selbst einen Ausdruck enthält, der eine Sequenz definiert, so wird die innere Sequenz aufgelöst (*flattened*). Der Ausdruck

  ```
  (1,
  for $i in doc("Mondial.xml")//Provinz
  return $i/Stadt/SName/text(),
  2)
  ```

  hat den Wert `1 Freiburg Karlsruhe Berlin 2`.

Funktion data()

Mittels data() wird einer Sequenz eine Sequenz atomarer Werte zugeordnet.

- Attributknoten und Textknoten werden durch ihren zugeordneten atomaren Wert repräsentiert.

```
data((1, "eins", doc("Mondial.xml")//Land/@LCode))
```

- Einem Elementknoten wird als Wert die Konkatenation aller in ihm enthaltenen Textknoten in Dokumentordnung zugeordnet.

```
data((1, "eins",
   doc("Mondial.xml")/(//Provinz/Stadt)[last()]))
```

Funktion `distinct-values()`

Eine Sequenz kann Duplikate enthalten. Der Ausdruck `(1,2,1)` definiert beispielsweise die Sequenz `1 2 1`.

Die Funktion distinct-values() eliminiert Duplikate aus Sequenzen.

- `distinct-values((1,2,1))`
- mit Duplikaten:

```
data(doc("Mondial.xml")//Provinz
   [PName = "Berlin"]//node()[text()])
```

- ohne Duplikate:

```
distinct-values(doc("Mondial.xml")//Provinz
   [PName = "Berlin"]//node()[text()])
```

Standardfunktionen

XQuery bietet weit über 100 Standardfunktionen an, die in einem separaten Standardisierungsdokument definiert werden. Darunter

- Aggregierungs-Funktionen, wie `avg()`, `max()` und `count()`,
- Funktionen für spezielle XML-Schema Datentypen wie `date`,
- Funktionen zur Typkonversion oder auch zum Pattern-Matching,
- Funktionen für spezielle XML-Konstrukte wie `ID`, `IDREF` und Namensräume.

Beispiel

```
<Mondial>
{
for $a in doc("Mondial1.xml")//Land
return
    <Land>
    {
       $a/LName,
       for $b in $a/Provinz
       let $b1 := $b/PName,
          $b2 := $b/Fläche,
          $b3 := $b/Stadt
       return
          element Provinz {attribute Einwohner {sum($b//Einwohner)},
             ($b1, $b2, $b3)}
    }
    </Land>
}
</Mondial>
```

Operatoren

Zusätzlich zu den üblichen arithmetischen Operatoren, wie `+`,`-`,`*`,`div` und `mod`, besitzt XQuery unterschiedliche Typen von Vergleichsoperatoren.

- Die Wertevergleichsoperatoren `eq`,`ne`,`lt`,`le`,`gt`,`ge` erlauben den Vergleich zweier atomarer Werte.
- Die Operatoren für allgemeinen Vergleich `=`,`!=`,`<`,`<=`,`>`,`>=` erlauben den Vergleich zweier Sequenzen atomarer Werte.

  Die Semantik ist hierbei so festgelegt, dass der Vergleichsausdruck dann `true()` liefert, wenn irgendein Wert der einen Sequenz mit irgendeinem Wert der anderen Sequenz in der gewünschten Relation steht.

  Damit gilt `(1,2)=(2,3)` und `(2,3)=(3,4)`, aber nicht der transitive Zusammenhang `(1,2)=(3,4)`.

weitere Operatoren

- Der Vergleich zweier Knoten ist bzgl. `is` und `is not` möglich, wobei `is` gleiche Knotenidentität meint.

```
doc("Mondial.xml")/(//Stadt)[last()] is
   doc("Mondial.xml")//Provinz[last()]/Stadt[last()]
```

- Die Operatoren <<,>> erlauben einen Vergleich zweier Knoten bzgl. der Dokumentordnung.

```
doc("Mondial.xml")//Provinz[PName = "Baden"] <<
   doc("Mondial.xml")//Provinz[PName = "Berlin"]
```

- Operatoren `union`,`intersect`,`except` liefern angewandt auf zwei Sequenzen eine Resultat-Sequenz ohne Duplikatknoten in Dokumentordnung.

```
(doc("MondialTest3.xml")//Stadt)[1] union
  doc("MondialTest3.xml")//Provinz[PName="Baden"]/Stadt[1]
```

Gleichheit von Sequenzen: `fn:deep-equal()`

Zwei Sequenzen sind gleich, wenn sie

- dieselbe Anzahl Einträge besitzen,
- Einträge an derselben Position in einer für ihren Typ definierten Weise gleich sind,
- diese Gleichheit bei Einträgen eines komplexen Typs rekursiv für die Kindknoten gilt.

Beispiel

```
fn:deep-equal(
    (doc("MondialTest3.xml")//Stadt)[last()],
    doc("MondialTest3.xml")//Provinz[PName="Baden"]/Stadt[1] )
```