

# 7 XML: Giving Messages a Structure

## (some) limitations of Relational Databases

- ▶ Documents: how, for example, to store a book or a system handbook in a relational table?  
As one huge string? By introducing for each book chapter a respective attribute?  
For each paragraph? ...
- ▶ Data Interchange: for example, for eBusiness, in which format shall messages be interchanged between partners?  
By writing emails? By exporting/importing relations?

- ▶ A language to structure data in a flexible way would be helpful.
- ▶ A language to semantically annotate data would be helpful.
- ▶ This should be done in a way sensible for humans and machines.

That's XML!

# XML Document

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

## XML Basics

- ▶ XML is a *Markup Language*; XML is a subset of SGML, which is a metalanguage standardized 1986 used to define the structure and content of documents. (Note: HTML is a specific application of SGML.)

Markup is indicated by a *start-tag* `<aTagname>` followed by an *end-tag* `</aTagname>`, both enclosing the markued part of the document.

- ▶ Start- and corrsponding end-tag, including the enclosed part of the document, is called *element*; the name of the element is the name of the tag and the *content* of the element is the enclosed part.
- ▶ The content of an element may contain simply text without any further tags, only (other) elements, or a mixture of both.

If the content of an element does not contain any further tags, the content is called *element text*.

If the content of an element is built out of other elements only, it is called *element content*.

In the remaining case it is called *mixed content*.

- ▶ A tag without content is called *empty tag* with shorthand notation `<aTagname/>`
- ▶ Start-tags may be further described using attributes.

## Attributes

- ▶ Elements with attributes:

`<aTagname attr1 = "val1"...attrk = "valk">, k ≥ 1.`

- ▶ empty element with attributes:

`<aTagname attr1 = "val1"...attrk = "valk" />, k ≥ 1.`

Example:

`<Mitglied Organisation = "EU" Art = "member"/>.`

- ▶ Each element may contain to each attribute at most one value.

## well-formed elements

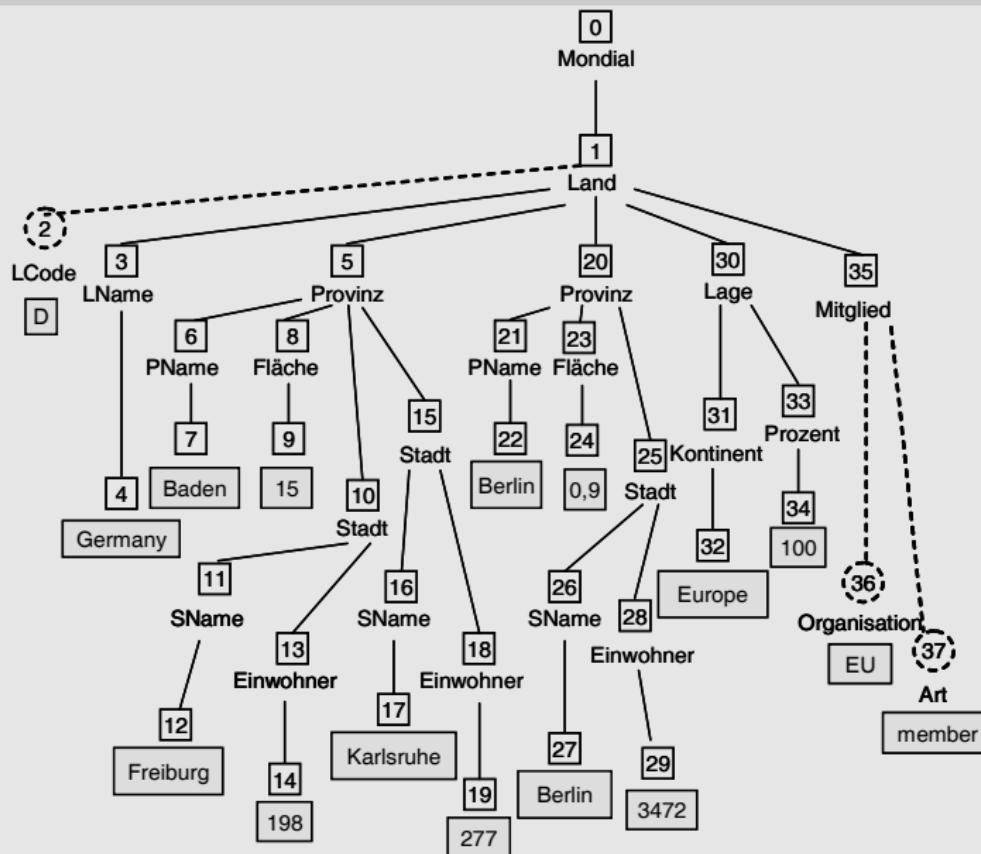
A document is called *wellformed*, if there holds

- ▶ for any two elements with names  $EName_1$ ,  $EName_2$ , if  $\langle EName_1 \rangle$  precedes  $\langle EName_2 \rangle$ , than either  $\langle /EName_1 \rangle$  precedes  $\langle EName_2 \rangle$ , or  $\langle /EName_2 \rangle$  precedes  $\langle /EName_1 \rangle$ .
- ▶ there is exactly one element in the document, which is not contained in any other element; this element is called *document-element*, it is the *root-element* of the document.
- ▶ If not stated differently, XML-documents are well-formed in the sequel.

## XML-tree

- ▶ Because of the well-formedness of an XML-document, it has a natural representation as a tree.
- ▶ Attributes are not per se part of an XML-tree; we call an XML-tree *extended*, if it contains the attributes as well.
- ▶ XML parser: constructs the tree representation of a given XML-document.

## (extended) XML-tree



## Document-order

- ▶ Tags of a document are ordered; the *document-order* is defined by the sequence in which the start-tags inside the document appear.
- ▶ Attributes are not ordered; there does not exist a document-order on attributes.
- ▶ A XML-tree is *ordered*, if depth-first search of the tree produces the document-order.
- ▶ To each ordered XML-tree there exists a textual representation called *serialization*, which reflects the document-order.

Note: *A node in the tree represents the corresponding elements start- and end-tag; the element content and the tags and contents of the respective subtree are included.*

## Name Spaces

XML-documents may contain other XML-documents as element-content. For example, a message written in XML may contain an XML-document as its data. Name conflicts may appear!

- ▶ Element- and attribute-names get a prefix denoting the intended (*name space*).
- ▶ Prefixes are URI's.
- ▶ Definition of Mondial-namespace:

```
<... xmlns:Mondial="http://www.inf.uni-fr.de/dbis/mondial">
```

Example: <Mondial:Stadt>...</ Mondial:Stadt>

- ▶ Default-namespace definition:

```
<... xmlns="http://www.inf.uni-fr.de/dbis/mondial">
```

Elements and attributes used without prefix are assumed to be taken from the default-namespace.

- ▶ Namespace-declarations are valid for the respective element - they may be redefined.

XML documents can be typed using DTD's (Document Type Definition) or XML Schema. Important issue - not discussed in this course!

Example: Mondial DTD. Observe the similarity to regular expressions!

```
<!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">  
]>
```

# XPath

Towards querying XML documents.

- ▶ XPath is a language to locate subtrees of an extended XML-tree by means of *location paths*.
- ▶ A location path is a sequence of '/'-separated *location steps*.
- ▶ Each location step is formed out of an *axis*, a *nodetest* and none or several *predicates*:

**axis::node-test[predicate]**

See for a detailed presentation:

<http://www.w3.org/TR/xpath/> which gives a concise specification of XPath 1.0,

<http://www.w3.org/TR/xpath-31/> which specifies the most recent version.

Some (very) simple location paths; short notation.

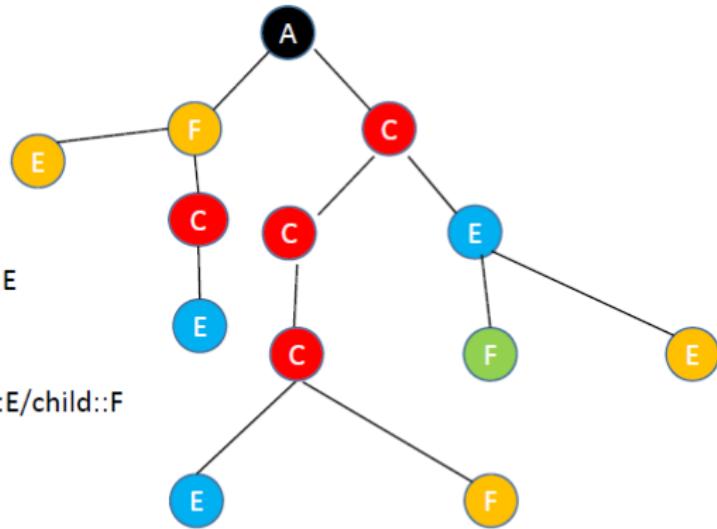
● /child::A  
/A

● /child::A/descendant::C  
/A//C

● /child::A/descendant::C/child::E  
/A//C/E

● /child::A/descendant::C/child::E/child::F  
/A//C/E/F

● node not located



- ▶ A location path  $P$  is evaluated relative to a respective context and locates a set of nodes, i.e. a set of root nodes of subtrees.
- ▶ A context is given by a *context node*, a *context position* and a *context size*.
- ▶ A location path starting with '/' is *absolut* and otherwise *relative*.

The context node of an absolut location path is the root node of the document.  
The root node of the document links to the root element of the document, which  
is the root node of the respective XML tree.

result of an absolute location path  $P = /p_1/p_2/\dots/p_n$

- ▶ Let  $n, n \geq 1$  and  $p_i, 1 \leq i \leq n$ , the  $i$ -th location step.
- ▶ Let  $L_0 = \{r\}$ , where  $r$  the root node of the document;  
Let  $L_i(k)$ ,  $i \geq 1$  be the set of nodes determined by  $p_i$  with respect to context  
node  $k \in L_{i-1}$ .

Then, for  $i \geq 1$  we define  $L_i = \bigcup_{k \in L_{i-1}} L_i(k)$ .

- ▶  $L_n$  is called the result of the location path  $P$ .

Axis:<sup>1</sup>**axis**::node-test [predicate]

- ▶ **child** axis contains the children of the context node,
- ▶ **descendant** axis contains the descendants of the context node; a descendant is a child or a child of a child and so on,
- ▶ **parent** axis contains the parent of the context node, if there is one,
- ▶ **ancestor** axis contains the ancestors of the context node; the ancestors of the context node consist of the parent of context node and the parent's parent and so on,
- ▶ **following-sibling** axis contains all the following siblings of the context node,
- ▶ **preceding-sibling** axis contains all the preceding siblings of the context node,
- ▶ **following** axis contains all nodes in the same document as the context node that are after the context node in document order, excluding any descendants,
- ▶ **preceding** axis contains all nodes in the same document as the context node that are before the context node in document order, excluding any ancestors,
- ▶ **attribute** axis contains the attributes of the context node,
- ▶ **self** axis contains just the context node itself,
- ▶ **descendant-or-self** axis contains the context node and the descendants of the context node,
- ▶ **ancestor-or-self** axis contains the context node and the ancestors of the context node.

---

<sup>1</sup>Cf. W3C XPath Recommendation <http://www.w3.org/TR/xpath>.

Locating nodes inside the document.



self



descendent



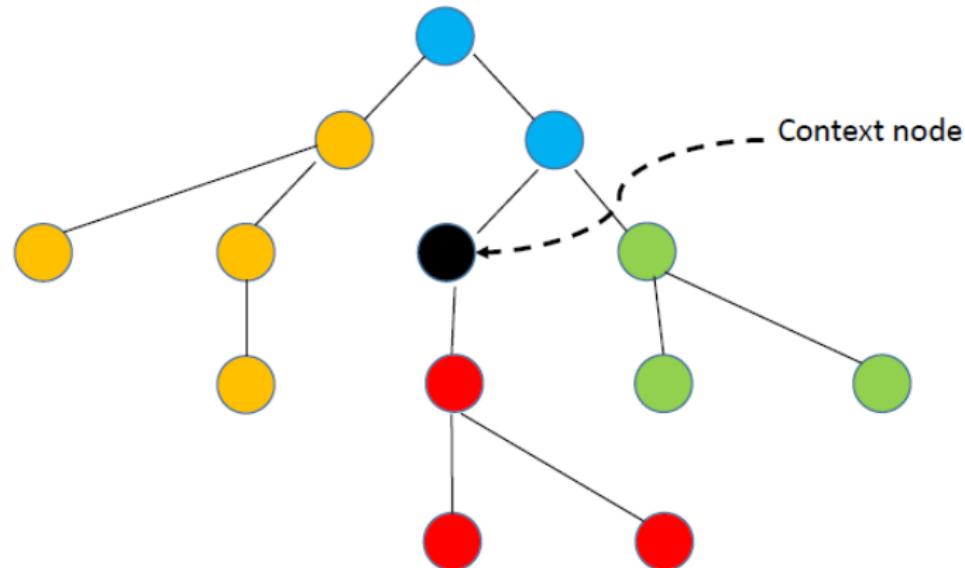
parent

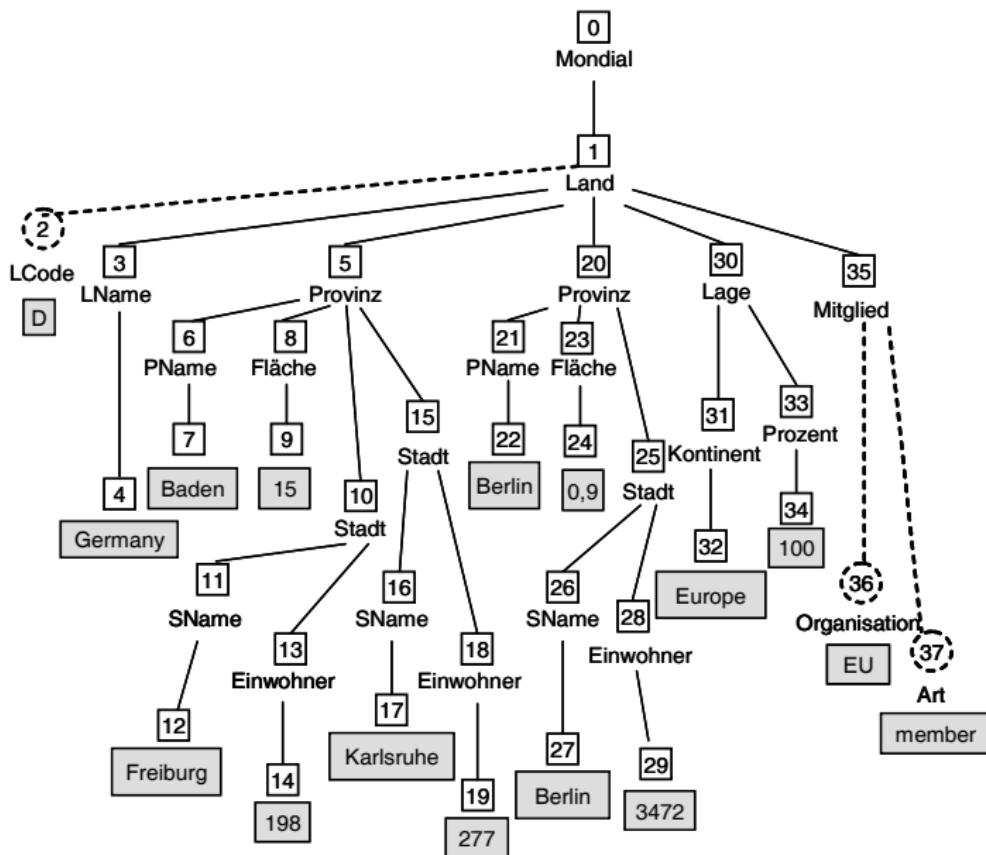


following



preceding





## Node test

axis::**node-test** [predicate]

- ▶ Element name, resp. attribute name
- ▶ '\*': any attribute or element
- ▶ `text()` only text nodes,
- ▶ `node()` no restriction

## Examples

- ▶ `/child::Mondial/child::Land`
- ▶ `/descendant::Stadt`
- ▶ `/descendant::text()`

## Predicates

axis::node-test [predicate]

- ▶ Predicates are XPath-expressions with can be assigned a boolean value.
- ▶ A location path has value `true()`, if it locates at least one node; otherwise `false()`.
- ▶ To evaluate comparison predicates, each node in the result of a location path is replaced by the string built out of the concatenation of all text nodes of the subtree it locates.
- ▶ Boolean expressions built by negation *not*, conjunction *and* or disjunction *|*.

## Examples

- ▶ `/descendant::*[child::Stadt]`
- ▶ `/descendant::*[Stadt]`
- ▶ `/descendant::*[self::Stadt]`
- ▶ `/child::Mondial/child::Land[child::LName = "Germany"]/child::Provinz`
- ▶ `/descendant::Stadt[parent::node()/child::PName = "Baden"]`
- ▶ `/descendant::Provinz[PName = "Baden"]/child::Stadt`

## context position

- ▶ *Context size* and *context position* are defined relative to a node set.
- ▶ The context size is the cardinality of the node set.
- ▶ The context position of a node is given by its position relative to document order; for backward axis the reverse document order is the basis.

## Examples:

- ▶ `//Stadt[position() = 2]`
- ▶ `//Land/Provinz[1]/Stadt[last()]`

## Function Library

- ▶ Node Set Functions, e.g. `last()`, `position()`, ...,
- ▶ String Functions, e.g. `contains(string, string)`, `concat(string, string, string)`, ...,
- ▶ Boolean Functions, e.g. `not(boolean)`, `true()`, `false()`, ...,
- ▶ Number Functions, e.g. `sum(node-set)`, `round(number)`, ....

## Examples:

- ▶ `sum(//Stadt//Einwohner)`
- ▶ `//Provinz[count(Stadt) >= 2]/PName`

# SQL/XML

## Outline

- ▶ How to *publish* (export) data in a relational database as XML documents?
- ▶ How can we *extract* data out of a XML document and store (import) it in a relational database?
- ▶ How can we *map* SQL queries such that they can be executed on a XML document?

## SQL/XML is part of the SQL:2003-Standard

- ▶ XML documents can be stored as character strings of type VARCHAR or CLOB (CHARACTER LARGE OBJECT).
- ▶ SQL/XML provides an additional type data *XML* (in addition to types like NUMERIC, VARCHAR,...)

## XML-Type

- ▶ XMLPARSE() parses a XML document (String) into XML-Type.
- ▶ XMLSERIALIZE() serializes a value of XML-Type into a String.
- ▶ SQL/XML defines additional functions which can be used within SFW-expressions in order to publish data from a relational database as XML.

## Example

```
create table ABC (xtest xmltype);
```

- ▶ **Insertion of XML as String**

```
insert into ABC ('<Mondial><Land LCode = "D">...</Land></Mondial>');
```

- ▶ **Insert of XML from relational data**

```
insert into ABC XMLTYPE(  
SELECT XMLEMENT ( NAME Land,  
    XMLATTRIBUTES (L.LCode AS LCode),  
    XMLEMENT (NAME LName, L.LName),  
    ( SELECT XMLAGG (  
        XMLEMENT (NAME Lage,  
        XMLEMENT (NAME Kontinent, La.Kontinent),  
        XMLEMENT (NAME Prozent, La.Prozent) )  
        FROM Lage La WHERE La.LCode = L.LCode) ... ) AS Mondial  
FROM Land L WHERE LCode = 'D');
```

## XPath Querying XML-Type Data with SQL Functions

- ▶ You can query XML-Type data and extract portions of it using SQL functions. These functions may use a subset of the W3C XPath recommendation to navigate the document.
- ▶ SQL Functions: EXISTSNODE, EXTRACT, EXTRACTVALUE.

### some queries

```
CREATE TABLE ABC (XTest xmltype);
```

- ▶ SELECT XTest.Erg FROM ABC;
- ▶ SELECT EXTRACT(XTest, '//LAGE') .Erg FROM ABC;
- ▶ SELECT EXTRACT(XTest, '//LAGE').getStringVal() .Erg FROM ABC;
- ▶ SELECT EXTRACT(XTest, '//LAGE').getStringVal() .Erg FROM ABC  
 WHERE EXISTSNODE (XTest, '//LAGE') = 1;
- ▶ SELECT EXTRACT(XTest, '//\*[text()]').getStringVal() .Erg FROM ABC  
 WHERE EXISTSNODE (XTest, '//LAGE') = 1;
- ▶ SELECT EXTRACTVALUE(XTest, '//LAGE[2][KONTINENT=''Asia'']/PROZENT') .Erg FROM ABC;
- ▶ SELECT EXTRACTVALUE(XTest, '//LAGE[position()=2 and KONTINENT=''Asia'']/PROZENT') .Erg  
 FROM ABC;
- ▶ SELECT EXTRACTVALUE(XTest, '//LAGE[2][KONTINENT=''Asia'']') .Erg FROM ABC;

# Summarize (1/2)

## Part 1: System Design

- ▶ Data Modeling with Entity-Relationship Diagram
- ▶ Entities, Relationships, Cardinalities
- ▶ Decomposition
- ▶ Mapping ER-Diagrams to Tables

## Part 2: Querying Relational Databases with SQL

- ▶ Joins, Nullvalues
- ▶ Aggregation and Grouping, Operations on Sets
- ▶ Aggregation Function and Subqueries
- ▶ Subqueries, Orthogonality of Syntax, Views
- ▶ Insert, Delete and Update
- ▶ Referential Integrity, Trigger

## Summarize (2/2)

### Part 3: Database Security

- ▶ Authorization by means of access rights
- ▶ GRANT and REVOKE rights

### Part 4: XML

- ▶ XML Documents
- ▶ Elements, Attributes
- ▶ XML trees, Document-order
- ▶ XPath
- ▶ SQL/XML