Chapter 8 – XML Data Modeling

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Inhalt

Überblick

I. Objektorientierung und Erweiterbarkeit

1. Benutzerdefinierte Datentypen und getypte Tabellen
2. Objekt-relationale Sichten und Kollektionstypen
3. Benutzerdefinierte Routinen und Objektverhalten
4. Anbindung an Anwendungsprogramme
5. Objekt-relationales SQL und Java

II. Online Analytic Processing

6. Datenanalyse in SQL
7. Windows und Query Functions

III. XML

8. XML Datenmodellierung
9. SQL/XML
10. XQuery

XML Origin and Usages

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language, not a database language
  - Documents have tags giving extra information about sections of the document
  - For example:
    - <title> XML </title>
    - <slide> XML Origin and Usages </slide>
- Derived from SGML (Standard Generalized Markup Language)
  - standard for document description
    - enables document interchange in publishing, office, engineering, …
  - main idea: separate form from structure
- XML is simpler to use than SGML
  - roughly 20% complexity achieves 80% functionality

XML Origin and Usages (cont.)

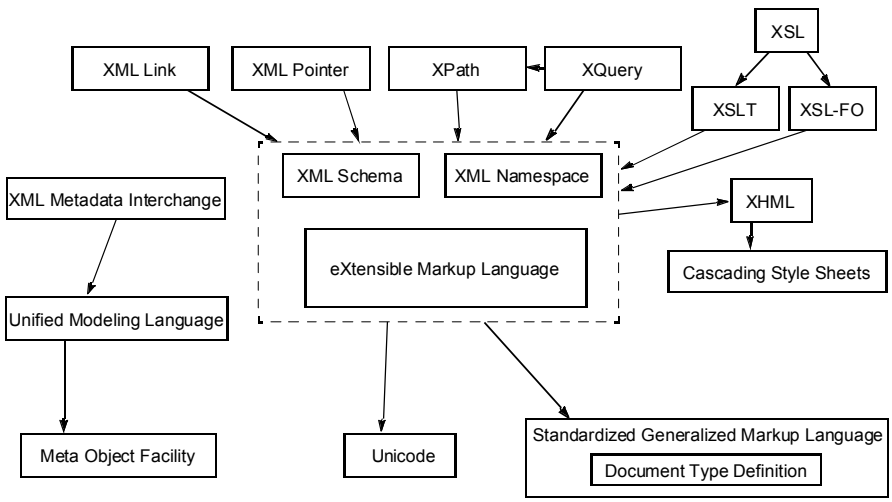
- XML documents are to some extent self-documenting
  - Tags can be used as metadata
  - Example

```
<bank>
    <account>
        <account-number> A-101     </account-number>
        <branch-name>      Downtown </branch-name>
        <balance>          500      </balance>
    </account>
    <depositor>
        <account-number> A-101    </account-number>
        <customer-name> Johnson </customer-name>
    </depositor>
</bank>
```

Forces Driving XML

- Document Processing
  - Goal: use document in various, evolving systems
  - structure – content – layout
  - grammer: markup vocabulary for mixed content
- Data Bases and Data Exchange
  - Goal: data independence
  - structured, typed data – schema-driven – integrity constraints
- Semi-structured Data and Information Integration
  - Goal: integrate autonomous data sources
  - data source schema not known in detail – schemata are dynamic
  - schema might be revealed through analysis only after data processing

XML Language Specifications



Describing XML Data: Basics - Elements

- **Tag:** label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly **nested**
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element
- Mixture of text with sub-elements is legal in XML
  - Example:

```
<account>
   This account is seldom used any more.
    <account-number> A-102</account-number>
    <branch-name> Perryridge</branch-name>
    <balance>400 </balance>
</account>
```

  - Useful for document markup, but discouraged for data representation

XML Document Example

```
<bank-1>
  <customer>
     <customer-name> Hayes </customer-name>
     <customer-street> Main </customer-street>
     <customer-city>    Harrison </customer-city>
     <account>
        <account-number> A-102 </account-number>
        <branch-name>     Perryridge </branch-name>
        <balance>          400 </balance>
     </account>
     <account>
        ...
     </account>
  </customer>
   .
   .
</bank-1>
```

Describing XML Data: Attributes

- **Attributes:** can be used to describe elements
- Elements can have attributes

```
<account acct-type = "checking" >
        <account-number> A-102 </account-number>
        <branch-name> Perryridge </branch-name>
        <balance> 400 </balance>
</account>
```

- Attributes are specified by *name=value* pairs inside the starting tag of an element
- Attribute names must be unique within the element

```
<account  acct-type = "checking"  monthly-fee="5">
```


Attributes vs. Subelements

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways
      - `<account  account-number = "A-101">  .... </account>`
      -
```
<account>
   <account-number>A-101</account-number> ...
</account>
```

More on XML Syntax

- Shortcut: elements without subelements or text content can be abbreviated by ending the start tag with a /> and deleting the end tag
  - <account number="A-101" branch="Perryridge" balance="200" />
- To store string data that may contain tags, without the tags being interpreted as subelements, use CDATA as below
  - <![CDATA[<account> … </account>]]>
    - Here, <account> and </account> are treated as just strings

XML Document Schema

- Metadata and database schemas constrain what information can be stored, and the data types of stored values
- Metadata are very important for data exchange
  - Guarantees automatic and correct data interpretation
- XML documents are not required to have associated metadata/schema
- Two mechanisms for specifying XML schema
  - **Document Type Definition (DTD)**
  - **XML Schema**

Describing XML Data: DTD

- Type and structure of an XML document can be specified using a DTD
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Example

  <! ELEMENT depositor (customer-name account-number)>
  <! ELEMENT customer-name(#PCDATA)>
  <! ELEMENT account-number (#PCDATA)>
- Subelement specification may have regular expressions

  <!ELEMENT bank ((account | customer | depositor)+)>
  - Notation:
    - "|" - alternatives
    - "?" - 0 or 1 occurrence
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences

Example: Bank DTD

```
<!DOCTYPE bank [
    <! ELEMENT bank ( ( account | customer | depositor)+)>
    <! ELEMENT account (account-number branch-name balance)>
    <! ELEMENT customer(customer-name customer-street
                                                customer-city)>
    <! ELEMENT depositor (customer-name account-number)>
    <! ELEMENT account-number (#PCDATA)>
    <! ELEMENT branch-name (#PCDATA)>
    <! ELEMENT balance(#PCDATA)>
    <! ELEMENT customer-name(#PCDATA)>
    <! ELEMENT customer-street(#PCDATA)>
    <! ELEMENT customer-city(#PCDATA)>
]>
```


Attribute Specification in DTD

- Attribute specification : for each attribute
  - Name
  - Type of attribute
    - CDATA
    - ID (identifier) or IDREF (ID reference) or IDREFS (multiple IDREFs)
  - Whether
    - mandatory (#REQUIRED)
    - default value (value),
    - or neither (#IMPLIED)
- Examples
  - <!ATTLIST account acct-type CDATA "checking">
  - <!ATTLIST customer
    customer-id ID #REQUIRED
    accounts IDREFS #REQUIRED >

IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  ➔ ID attribute (value) is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document
- IDs and IDREFs are untyped, unfortunately
  - Example below: The *owners* attribute of an account may contain a reference to another account, which is meaningless;
    owners attribute should ideally be constrained to refer to customer elements

Example: Extended Bank DTD

- Bank DTD with ID and IDREF attribute types

```
<!DOCTYPE bank-2[
    <!ELEMENT account (branch-name, balance)>
    <!ATTLIST account
            account-number   ID        #REQUIRED
            owners           IDREFS    #REQUIRED>
     <!ELEMENT customer(customer-name, customer-street,
                                                customer-city)>
     <!ATTLIST customer
            customer-id      ID                 #REQUIRED
            accounts         IDREFS     #REQUIRED>
      … declarations for branch, balance, customer-name,
                        customer-street and customer-city
]>
```

XML data with ID and IDREF attributes

```
<bank-2>
    <account account-number="A-401" owners="C100 C102">
        <branch-name> Downtown </branch-name>
        <balance>500 </balance>
    </account>
        . . .
    <customer customer-id="C100" accounts="A-401">
        <customer-name>Joe</customer-name>
        <customer-street>Monroe</customer-street>
        <customer-city>Madison</customer-city>
    </customer>
    <customer customer-id="C102" accounts="A-401 A-402">
        <customer-name> Mary</customer-name>
        <customer-street> Erin</customer-street>
        <customer-city> Newark </customer-city>
    </customer>
</bank-2>
```


Describing XML Data: XML Schema

- XML Schema is closer to the general understanding of a (database) schema
- XML Schema supports
  - Typing of values
    - E.g. integer, string, etc
  - Constraints on min/max values
  - Typed references
  - User defined types
  - Specified in XML syntax (unlike DTDs)
  - Integrated with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..
- BUT: significantly more complicated than DTDs

XML Schema Structures

- **Datatypes (Part 2)**
  Describes Types of scalar (leaf) values
- **Structures (Part 1)**
  Describes types of complex values (attributes, elements)
  - Regular tree grammars
    repetition, optionality, choice recursion
- **Integrity constraints**
  Functional (keys) & inclusion dependencies (foreign keys)
- **Subtyping (akin to OO models)**
  Describes inheritance relationships between types
- **Supports schema reuse**

XML Schema Structures (cont.)

- Elements : tag name & simple or complex type
  <xs:element **name**="sponsor" **type**="xsd:string"/>
  <xs:element **name**="action" **type**="Action"/>
- Attributes : tag name & simple type
  <xs:attribute **name**="date" **type**="xsd:date"/>
- Complex types

```
<xs:complexType name="Action">
    <xs:sequence>
        <xs:elemref name ="action-date"/>
        <xs:elemref name ="action-desc"/>
    </xs:sequence>
</xs:complexType>
```

XML Schema Structures (cont.)

- Sequence

```
<xs:sequence>
    <xs:element name="congress" type=xsd:string"/>
    <xs:element name="session" type=xsd:string"/>
</xs:sequence>
```

- Choice

```
<xs:choice>
    <xs:element name="author" type="PersonName"/>
    <xs:element name="editor" type="PersonName"/>
</xs:choice>
```

- Repetition

```
<xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs element name ="section" type="Section"/>
</xs:sequence>
```


Namespaces

- A single XML document may contain elements and attributes defined for and used by multiple software modules
  - Motivated by modularization considerations, for example
- Name collisions have to be avoided
- Example:
  - A **Book** XSD contains a Title element for the title of a book
  - A **Person** XSD contains a Title element for an honorary title of a person
  - A **BookOrder** XSD reference both XSDs
- Namespaces specifies how to construct universally unique names

Namespaces (cont.)

- Namespace is a collection of names identified by a URI
- Namespaces are declared via a set of special attributes
  - These attributes are prefixed by xmlns - Example:

    <BookOrder **xmlns:**Customer="http://mySite.com/Person"
    xmlns: Item="http://yourSite.com/Book">
  - Namespace applies to the element where it is declared, and all elements within its content
    - unless overridden
- Elements/attributes from a particular namespace are prefixed by the name assigned to the namespace in the corresponding declaration of the using XML document
  - ...Customer:Title='Dr'...
  - ...Item:Title='Introduction to XML'...
- Default namespace declaration for fixing the namespace of unqualified names
  - Example:

    <BookOrder **xmlns**="http://mySite.com/Person"
    xmlns:Item="http://yourSite.com/Book">

Namespaces and XML Schema

- XML-Schema elements and data types are imported from the XML-Schema namespace http://www/w3/org/2001/XMLSchema
  - **xsd** is generally used as a prefix
- The vocabulary defined in an XML Schema file belongs to a target namespace
  - declared using the **targetNamespace** attribute
  - declaring a target namespace is optional
    - if none is provided, the vocabulary does not belong to a namespace
    - required for creating XML schemas for validating (pre-namespace) XML1.0 documents
- XML document using an XML schema
  - declares namespace, refers to the target namespace of the underlying schema
  - can provide additional hints where an XML schema (xsd) file for the namespace is located
    - schemaLocation attribute

XML Schema Version of Bank DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banks.org"
            xmlns ="http://www.banks.org" >
<xsd:element name="bank" type="BankType"/>
<xsd:element name="account">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="account-number" type="xsd:string"/>
            <xsd:element name="branch-name"    type="xsd:string"/>
            <xsd:element name="balance"        type="xsd:decimal"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>                  ..... definitions of customer and depositor ....
<xsd:complexType name="BankType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element ref="account"/>
        <xsd:element ref="customer"/>
        <xsd:element ref="depositor"/>
    </xsd:choice>
</xsd:complexType>
</xsd:schema>
```


XML Document Using Bank Schema

```
<bank   xmlns="http://www.banks.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.banks.org  Bank.xsd">
    <account>
        <account-number> … </account-number>
        <branch-name> … </branch-name>
        <balance> … </balance>
    </account>
    …
</bank>
```

Assertions in XML-Schema

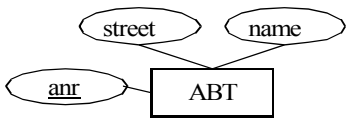
- Uniqueness: UNIQUE-Element, KEY-Element
  - forces uniqueness of attribute or element values
    - <field> elemen(s)
  - can be applied to/declared for specific parts of the XML document
    - <selector> element
  - Example: zip-codes should be unique for a combination of region and part number

```
<unique name="constraint1">
  <selector xpath="//regions/zip" />
  <field xpath="@code" />
  <field xpath="part/@number" />
</unique>
```

- Some remarks
  - NULL value semantics: **nillable** at the schema level, **nil** in the document
  - *<key>* equivalent to *<unique> and nillable="false"*
  - uniqueness can be reduced to XML fragments
  - composite keys/unique elements

Mapping ER-Model -> XML Schema

- Mapping Entities
  - 1:1 mapping to XML elements
  - use <key> to represent ER key attributes




```
...
<element name="ABT">
  <complexType>
        <attribute name="anr" type="string" />
        <attribute name="street" type="string" />
        <attribute name="name" type="string" />
  </complexType>
</element>
...
<key name="abt_pk">
  <selector xpath=".//ABT/" />
  <field xpath="@anr" />
</key>
...
```

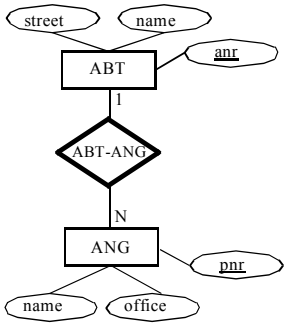
Mapping 1:N Relationships

- Mapping alternatives
  - Nesting

```
<element name="ABT">
    <complexType>
      <sequence>
        <element name="ANG">
          <complexType>
            <attribute name="name" type="string"/>
            <attribute name="office" type="string"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="street" type="string"/>
      <attribute name="name" type="string"/>
    </complexType>
</element>
```

  - Flat representation

```
<element name="ABT">
    <complexType>
      <sequence>
        <element ref="ANG">
      </sequence>
      <attribute name="street" type="string" />
      <attribute name="name" type="string" />
    </complexType>
</element>
```




```
...
<element name="ANG">
    <complexType>
      <attribute name="name" type="string" />
      <attribute name="office" type="string" />
    </complexType>
</element>
```


Primary/Foreign Keys

- Problem
  - nesting alone is not sufficient for modeling a 1:n relationship
  - element identity is required to avoid duplicate entries
- Foreign Keys
  - guarantee referential integrity: <key> / <keyref> elements

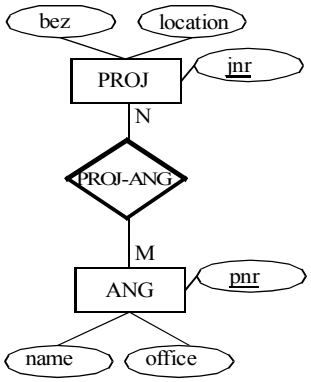
```
<element name="ABT">
    <complexType>
      <sequence>
        <element name="ANG">
          <complexType>
            <attribute name="pnr" type="string"/>
            <attribute name="name" type="string"/>
            <attribute name="office" type="string"/>
            <attribute name="abtid" type="string"/>
          </complexType>
        </element>
      </sequence>
      <attribute name="anr" type="string"/>
      <attribute name="name" type="string"/>
      <attribute name="street" type="string"/>
    </complexType>
</element>
...
```

```
...
<key name="abt_pk">
        <selector xpath=".//ABT" />
        <field xpath="@anr" />
</key>
...
<key name="ang_uniq">
        <selector xpath=".//ABT/ANG" />
        <field xpath="@pnr" />
</unique>
...
<keyref name="abt_fk" refer="abt_pk">
        <selector xpath=".//ABT/ANG" />
        <field xpath="@abtid" />
</key>
...
```

Primary/Foreign Keys

- Advantages over ID/IDREF
  - based on equality of data types
  - composite keys
  - locality, restricting scope to parts of the XML document
- Mapping of N:M – relationships
  - use <key>/<keyref> elements
  - flat modeling plus "pointers"
  - addition of helper element similar to mapping to relational model

```
<element name="PROJ_ANG">
    <complexType>
      <attribute name="pnr" type="string" />
      <attribute name="jnr" type="string" />
    </complexType>
</element>
```



Example: Mapping N:M-Relationships

- Element mapping

```
<element name="ANG">
    <complexType>
        <attribute name="pnr" type="string" />
        <attribute name="name" type="string" />
        <attribute name="office" type="string" />
        <attribute name="jnr" type="string" />
    </complexType>
</element>
```

```
<element name="PROJ">
    <complexType>
        <attribute name="jnr" type="string" />
        <attribute name="bez" type="string" />
        <attribute name="location" type="string" />
        <attribute name="pnr" type="string" />
    </complexType>
</element>
```

- Referential constraints, composite keys

```
<key name="ang_pk">
    <selector xpath=".//ANG" />
    <field xpath="@pnr" />
</key>
...
<key name="proj_pk">
    <selector xpath=".//PROJ" />
    <field xpath="@jnr" />
</key>
...
```

```
<unique name="proj_ang_uq">
    <selector xpath=".//PROJ_ANG" />
    <field xpath="@pnr" />
    <field xpath="@pnr" />
</key>
...
```

```
<keyref name="proj_fk" refer="proj_pk">
    <selector xpath=".//PROJ_ANG" />
    <field xpath="@jnr" />
</keyref>
...
<keyref name="ang_fk" refer="ang_pk">
    <selector xpath=".//PROJ_ANG" />
    <field xpath="@pnr" />
</keyref>
...
```