# XML and Databases

# Chapter 5: XML Schema

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Winter 2023/24

http://www.informatik.uni-halle.de/~brass/xml23/

# Objectives

After completing this chapter, you should be able to:

- explain why DTDs are not sufficient for many applications.
- explain some XML schema concepts.
- write an XML schema.
- check given XML documents for validity according to a given XML schema.

# Contents

# Introduction (1)

## Problems of DTDs:

- The type system is very restricted.

  E.g. one cannot specify that an element or an attribute must contain a number.

- Concepts like keys and foreign keys (known from the relational data model) cannot be specified.

  The scope of `ID` and `IDREF` attributes is global to the entire document.

  Furthermore, the syntax restrictions for `ID`s are quite severe.

- A DTD is not itself an XML document (i.e. it does not use the XML syntax for data).

- No support for namespaces.

- One cannot do everything with elements that can be done with attributes (e.g. enumeration types, `ID`/`IDREF`).

# Introduction (2)

- DTDs were probably sufficient for the needs of the document processing community, but do not satisfy the expectations of the database community.
- Therefore, a new way of describing the application-dependent syntax of an XML document was developed: XML Schema.
- In XML Schema, one can specify all syntax restrictions that can be specified in DTDs, and more (i.e. XML Schema is more expressive).

  Only entities cannot be defined in XML Schema.

# Introduction (3)

- The W3C began work on XML Schema in 1998.
- XML Schema 1.0 was published as a W3C standard ("recommendation") on May 2, 2001.

  A second edition appeared October 28, 2004.
- XML Schema 1.1 became a W3C recommendation on April 5, 2012.
- The Standard consists of:
  - Part 0: Tutorial introduction (non-normative).
  - Part 1: Structures.
  - Part 2: Datatypes.

# Introduction (4)

- A disadvantage of XML schema is that it is very complex, and XML schemas are quite long (much longer than the corresponding DTD).
- Quite a number of competitors were developed.

  E.g. XDR, SOX, Schematron, Relax NG.

  See: D. Lee, W. Chu: Comparative Analysis of Six XML Schema Languages. In ACM SIGMOD Record, Vol. 29, Nr. 3, Sept. 2000.
- Relax NG is a relatively well-known alternative.

  See: J. Clark, M. Makoto: RELAX NG Specification, OASIS Committee Specification, 3 Dec. 2001.
  [http://www.oasis-open.org/committees/relax-ng/spec-20011203.html]

# Introduction (5)

## Comparison with DBMS:

- In a (relational) DBMS, data cannot be stored without a schema.
- An XML document is self-describing: It can exist and can be processed without a schema.
- In part, the role of a schema in XML is more like integrity constraints in a relational DB.

  It helps to detect input errors. Programs become simpler if they do not have to handle the most general case.

- But in any case, programs must use knowledge about the names of at least certain elements.

# Contents

# Example Document (1)

**STUDENTS**

| SID | FIRST | LAST | EMAIL |
|---|---|---|---|
| 101 | Ann | Smith | ... |
| 102 | David | Jones | NULL |
| 103 | Paul | Miller | ... |
| 104 | Maria | Brown | ... |

**RESULTS**

| SID | CAT | ENO | POINTS |
|---|---|---|---|
| 101 | H | 1 | 10 |
| 101 | H | 2 | 8 |
| 101 | M | 1 | 12 |
| 102 | H | 1 | 9 |
| 102 | H | 2 | 9 |
| 102 | M | 1 | 10 |
| 103 | H | 1 | 5 |
| 103 | M | 1 | 7 |

**EXERCISES**

| CAT | ENO | TOPIC | MAXPT |
|---|---|---|---|
| H | 1 | ER | 10 |
| H | 2 | SQL | 10 |
| M | 1 | SQL | 14 |

# Example Document (2)

- Translation to XML with data values in elements:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
  <STUDENTS>
    <STUDENT>
      <SID>101</SID>
      <FIRST>Ann</FIRST>
      <LAST>Smith</LAST>
    </STUDENT>
    ...
  </STUDENTS>
  ...
</GRADES-DB>
```

# Example: First Schema (1)

- Part 1/4:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xs:schema
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="GRADES-DB">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="STUDENTS"/>
        <xs:element ref="EXERCISES"/>
        <xs:element ref="RESULTS"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

# Example: First Schema (2)

- Part 2/4:

```
<xs:element name="STUDENTS">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="STUDENT"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Example: First Schema (3)

- Part 3/4:

```
<xs:element name="STUDENT">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="SID"/>
      <xs:element ref="FIRST"/>
      <xs:element ref="LAST"/>
      <xs:element ref="EMAIL" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Example: First Schema (4)

- Part 4/4:

```
    <xs:element name="SID">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="100"/>
          <xs:maxInclusive value="999"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="FIRST" type="xs:string"/>
    <xs:element name="LAST" type="xs:string"/>
    <xs:element name="EMAIL" type="xs:string"/>
    ...
  </xs:schema>
```

# Example: First Schema (5)

## Namespace Prefix:

- The prefix used for the namespace is not important. E.g. sometimes one sees "`xsd:`" instead of "`xs:`".

## Simple vs. Complex Types:

- A complex type is a type that contains elements and/or attributes.
- A simple type is something like a string or number.

  A simple type can be used as the type of an attribute, and as the data type of an element (content and attributes). A complex type can only be the data type of an element (attributes cannot contain elements or have themselves attributes). Instead of "element", I should really say "element type", but that might be confusing (it is not an XML Schema type).

# Example: First Schema (6)

- In XML Schema, the sequence of declarations (and definitions, see below) is not important.

  The example contains many references to element types that are declared later. Actually, a schema can contain references to elements that are not declared at all, as long as these elements do not occur in the document, i.e. they are not needed for validation. Some validators even in this case print no error message: They use "lax validation" and check only for what they have declarations.

- It is necessary to use a one-element sequence (or choice) in the declaration of `STUDENTS`.

  One cannot use `xs:element` directly inside `xs:complexType`. This is similar to the content model in DTDs, which always needs "`(...)`".

# Example: First Schema (7)

- The default for `minOccurs` and `maxOccurs` is `1`.
  - `?` in DTD: `minOccurs="0"` (`maxOccurs` is 1 by default)
  - `+` in DTD: `maxOccurs="unbounded"` (`minOccurs` is 1)
  - `*` in DTD: `minOccurs="0" maxOccurs="unbounded"`
- In XML Schema, one cannot define what must be the root element type. E.g., a document consisting only of a `STUDENT`-element would validate.

  Every "globally" declared element type can be used. Global declarations are declarations that appear directly below `xs:schema`. As explained below, it is often possible to declare only the intended root element type globally, then there is no problem. Otherwise the application must check the root element type. Note that DTDs also do not define the root element type, this happens only in the `DOCTYPE`-declaration.

# Contents

# Validation (1)

Online Validators:

- Freeformatter

  [http://www.freeformatter.com/xml-validator-xsd.html]

- CoreFiling

  [https://www.corefiling.com/opensource/schemaValidate/]

- XML Validation

  [http://www.xmlvalidation.com/?L=2]

# Validation (2)

## Validators for Local Installation:

- Altova XML Community Edition

  [http://www.softpedia.com/get/Internet/Other-Internet-Related/AltovaXML.shtml]

- XSV

  [http://www.ltg.ed.ac.uk/~ht/xsv-status.html]

- BaseX

  [http://basex.org/] See also: [http://docs.basex.org/wiki/Validation_Module]
  Enter e.g. `validate:xsd-report("example.xml","example.xsd")` in the editor/query area and press the green execution arrow on top of this area.
  `validate:xsd-info(...)` returns the same result as a list of strings.

- There are plugins for Visual Studio Code and Eclipse.

# Validation (3)

Validating parser libraries:

- Apache Xerces

  [http://xerces.apache.org/]

- Libxml2

  [http://xmlsoft.org/]

- Oracle XDK

  [http://www.oracle.com/technetwork/developer-tools/xmldevkit/]

- Microsoft MSXML

  [http://msdn2.microsoft.com/en-us/xml/default.aspx]

- A small Java-Program suffices to use a library for validation.

  [https://www.torsten-horn.de/techdocs/java-xsd.htm#XSD-Java]

# Validation (4)

- Depending on the validator used, it is not necessary that the XML data file (the instance of the schema) contains a reference to the schema.
- If a schema reference is needed, this can be done as follows:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB xmlns:xsi=
      "http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="ex2.xsd">
  ...
</GRADES-DB>
```

There is also `schemaLLocation="NamespaceURI SchemaURI"`.

The attribute value consists of two parts, separated by a space: The target namespace URI and the link to the schema. Validation is done only for elements of that namespace.

# Contents

# Schema Styles (1)

- The same restrictions on XML documents can be specified in different ways in XML.

  I.e. there are equivalent, but very differently structured XML schemas.

- The above XML schema is structured very similar to a DTD:
  All element types are declared with global scope.
  No named types (see below) are used.

- This style is called “Salami Slice”.

  The schema is constructed in small pieces on equal level.
  “‘Salami slice’ caputes both the disassembly process, the resulting flat look of the schema, and implies reassembly as well (into a sandwich).”
  [http://www.xfront.com/GlobalVersusLocal.html]

# Schema Styles (2)

- One can also nest element declarations.
- Element declarations that are not defined as children of `xs:schema` cannot be referenced.

  They are local declarations in contrast to the global ones used above.
- In this way, one can have elements with the same name, but different content models in different contexts within one document.

  This is impossible with DTDs. It might be useful for complex documents, especially if the schema is composed out of independently developed parts. In relational DBs, different tables can have columns with the same name, but different types. Then the above XML translation of a relational schema cannot be done in "Salami Slice" style.

# Schema Styles (3)

- XML Schema in "Russian Doll" style:

```
<xs:element name="GRADES-DB">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="STUDENTS">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="STUDENT"
                minOccurs="0"
                maxOccurs="unbounded">
              <xs:complexType>
                <xs:sequence>
                  <xs:element name="SID">
                    ...
```

# Schema Styles (4)

- Advantages of "Russian Doll" style:
  - The structure of the schema is similar to the structure of the document.
  - In "Russian Doll" style, there is only one global element, thus the root element type is enforced.
- Disadvantages:
  - The declaration of equal subelements has to be duplicated.
  - Recursive element types are not possible.
  - No reuse of schema components.

# Schema Styles (5)

- Actually, in XML schema, one
  - defines (data) types and
  - declares elements to have a (data) type.

    A declaration binds names that occur in the XML data file (the instance) to (data) types. A definition introduces names that can be used only in the schema.

- In the above examples, all types are anonymous.
  In "Venetian Blind" design, explicit types are used.

  At least for elements with similar content models. Elements are declared locally as in the "Russian Doll" style.
  "'Venetian Blind' captures the ability to expose or hide namespaces with a simple switch, and the assembly of slats captures reuse of components."
  [http://www.xfront.com/GlobalVersusLocal.html]

# Schema Styles (6)

- XML Schema in "Venetian Blind" style, Part 1/4:

```
<xs:simpleType name="SIDType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="100"/>
    <xs:maxInclusive value="999"/>
  </xs:restriction>
</xs:simpleType>
<!-- Continued on next three slides -->
```

# Schema Styles (7)

- "Venetian Blind" Style, Part 2/4:

```
<xs:complexType name="StudentType">
  <xs:sequence>
    <xs:element name="SID" type="SIDType"/>
    <xs:element name="FIRST" type="xs:string"/>
    <xs:element name="LAST" type="xs:string"/>
    <xs:element name="EMAIL" type="xs:string"
        minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
```

# Schema Styles (8)

- "Venetian Blind" Style, Part 3/4:

```
<xs:complexType name="StType">
  <xs:sequence>
    <xs:element name="STUDENT" type="studentType"
        minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
```

# Schema Styles (9)

- "Venetian Blind" Style, Part 4/4:

```
<xs:complexType name="GradesType">
  <xs:sequence>
    <xs:element name="STUDENTS" type="StType"/>
    <xs:element name="EXERCISES" type="ExType"/>
    <xs:element name="RESULTS" type="ResType"/>
  </xs:sequence>
</xs:complexType>

<xs:element name="GRADES-DB" type="GradesType">
```

# Schema Styles (10)

- Remarks about "Venetian Blind" style:
  - There is only one global element declaration, thus the root element type is enforced.

    All other elements are known only locally within their type.
  - Probably, this is often the best style.

    The content model (and attributes) of equal subelements is specified only once (in the corresponding type). The components (types) are resuable. The reusability is even better than in the "Salami Slice" style, because the (data) types can be used with different element (type) names.
  - It is possible to define types and elements with the same name.

# Schema Styles (11)

Summary:

| Style | Element Decl. | Type Decl. |
|---|---|---|
| Salami Slice | Global | Anonymous, local (except predefined simple types) |
| Russian Doll | Local (except root) | Anonymous, local (except predefined simple types) |
| Venetian Blind | Local (except root) | Named, global |

# Contents

# Example with Attributes (1)

- Document:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
  <STUDENT SID='101' FIRST='Ann' LAST='Smith'/>
  <STUDENT SID='102' FIRST='David' LAST='Jones'/>
  ...
  <EXERCISE CAT='H' ENO='1' TOPIC='Rel. Algeb.'/>
  ...
  <RESULT SID='101' CAT='H' ENO='1' POINTS='10'/>
  ...
</GRADES-DB>
```

# Example with Attributes (2)

- Schema, Part 1/3:

```
<xs:element name="GRADES-DB">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="STUDENT"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="EXERCISE"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="RESULT"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Example with Attributes (3)

- Schema, Part 2/3:

```
<xs:element name="STUDENT">
  <xs:complexType>
    <xs:attribute name="SID" use="required">
      <xs:simpleType>
        <xs:restriction base="xs:integer">
          <xs:minInclusive value="100"/>
          <xs:maxInclusive value="999"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <!--- declaration continued on next slide -->
```

# Example with Attributes (4)

- Schema, Part 3/3:

```
        <xs:attribute name="FIRST"
            use="required"
            type="xs:string"/>
        <xs:attribute name="LAST"
            use="required"
            type="xs:string"/>
        <xs:attribute name="EMAIL"
            type="xs:string"/>
    </xs:complexType>
</xs:element> <!-- STUDENT -->
```

# Example with Attributes (5)

- The same (simple) data type can be used for attributes and for element content.

  In contrast, DTDs had some data types for attributes, but basically no data types for element content (only strings) (and of course content models, but that is a separate issue).

- In the example, the elements have empty content (`xs:complexType` contained no content model).
- If an element type has element content and attributes, inside `xs:complexType`, one must specify
  - first the content model (e.g., with `xs:sequence`)
  - and then declare the attributes.

# Example with Attributes (6)

- Element types with attributes and simple types as content, e.g.

```
<length unit="cm">12</length>
```

can be defined by extension of the simple type:

```
<xs:complexType name="lengthType">
  <xs:simpleContent>
    <xs:extension base="xs:integer">
      <xs:attribute name="unit" type="xs:string">
    </xs:extension>
  </xs:simpleContent>
<xs:complexType>
```

# References


- Harald Schöning, Walter Waterfeld: XML Schema.
  In: Erhard Rahm, Gottfried Vossen: Web & Datenbanken, Seiten 33-64.
  dpunkt.verlag, 2003, ISBN 3-89864-189-9.
- Priscilla Walmsley: Definitive XML Schema.
  Prentice Hall, 2001, ISBN 0130655678, 560 pages.
- W3C Architecture Domain: XML Schema.
  [http://www.w3.org/XML/Schema]
- David C. Fallside, Priscilla Walmsley: XML Schema Part 0: Primer.
  W3C, 28. October 2004, Second Edition.
  [http://www.w3.org/TR/xmlschema-0/]
- Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn:
  XML Schema Part 1: Structures.
  W3C, 28. October 2004, Second Edition
  [http://www.w3.org/TR/xmlschema-1/]
- Paul V. Biron, Ashok Malhotra: XML Schema Part 2: Datatypes.
  W3C, 28. October 2004, Second Edition
  [http://www.w3.org/TR/xmlschema-2/]
- [http://www.w3schools.com/schema/]