

XML and Databases

Chapter 5: XML Schema

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Winter 2019/20

<http://www.informatik.uni-halle.de/~brass/xml19/>

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.

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 IDs 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.
 - [\[http://www.oasis-open.org/committees/relax-ng/spec-20011203.html\]](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.

Example Document (1)

STUDENTS			
<u>SID</u>	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	David	Jones	NULL
103	Paul	Miller	...
104	Maria	Brown	...

EXERCISES			
<u>CAT</u>	<u>ENO</u>	TOPIC	MAXPT
H	1	ER	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
<u>SID</u>	<u>CAT</u>	<u>ENO</u>	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

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.

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.

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>]

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 one wants to refer to the schema, 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>
```

Inhalt

- 1 Introduction, First Example
- 2 Schema Styles**
- 3 Attributes
- 4 Integrity Constraints
- 5 Advanced Constructs

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).”

[\[http://www.xfront.com/GlobalVersusLocal.html\]](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 reusable. 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

Inhalt

- 1 Introduction, First Example
- 2 Schema Styles
- 3 Attributes**
- 4 Integrity Constraints
- 5 Advanced Constructs

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>
```

Inhalt

- 1 Introduction, First Example
- 2 Schema Styles
- 3 Attributes
- 4 Integrity Constraints**
- 5 Advanced Constructs

Integrity Constraints (1)

- DTDs have **ID/IDREF** to permit a unique identification of nodes and links between elements.
- This mechanism is quite restricted:
 - The identification must be a single XML name.

A number cannot be used as identification. Composed keys are not supported. DTDs do not allow further restrictions of the possible values, e.g. one cannot enforce a certain format for the names.
 - The scope is global for the entire document.

One cannot state that the uniqueness only has to hold within an element (e.g., representing a relation). One cannot specify any constraints of the element type that is referenced with **IDREF**.
 - This works only for attributes, not for elements.

Integrity Constraints (2)

- XML Schema has mechanisms corresponding to keys and foreign keys in relational databases that solve the problems of **ID/IDREF**.

They are more complex than the relational counterparts, because the hierarchical structure of XML is more complex than the flat tables of the relational model. The simplicity of the relational model was one of its big achievements. This is given up in XML databases.

- The facets correspond to **CHECK**-constraints that restrict the value set of a single column.

Not all SQL conditions that refer to only one column can be expressed with facets. On the other hand, patterns in XML Schema are much more powerful than SQL's **LIKE**-conditions. It is strange that patterns refer to the external representation.

Integrity Constraints (3)

- Otherwise, XML Schema 1.0 is not very powerful with respect to constraints. This changed in Version 1.1.
 - E.g., CHECK-constraints in relational databases can state logical conditions between the column values of a table row, e.g. if one column has a certain value then another column must be not null. The facets of XML Schema constrain only single values.
- For example, XML Schema itself requires that the type-attribute of `element` is mutually exclusive with `simpleType/complexType-child` elements.
- This constraint cannot be specified in XML Schema 1.0.
 - One would expect that the schema for XML Schema can express the necessary requirements.

Integrity Constraints (4)

- XML Schema 1.1 (released April 5, 2012) introduced an Element **assert** that permits to specify arbitrary conditions in XPath 2.0.

However, there are not very many XML Schema 1.1 implementations yet.

- For instance, one can compare two attribute values of an element (attribute `min` must be \leq `max`):

```
<xs:complexType name="intRange">
  <xs:attribute name="min" type="xs:int"/>
  <xs:attribute name="max" type="xs:int"/>
  <xs:assert test="@min le @max"/>
</xs:complexType>
```

[<https://www.w3.org/TR/2012/REC-xmlschema11-1-20120405/>]

Unique/Key Constraints (1)

- Consider again the example:

```

<?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>

```

Unique/Key Constraints (2)

- SID-values uniquely identify the children of STUDENTS:

```

<xs:element name="STUDENTS">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="STUDENT"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:unique name="STUDENTS_KEY">
    <xs:selector xpath="*" />
    <xs:field xpath="SID" />
  </xs:unique>
</xs:element>

```

Unique/Key Constraints (3)

- There are three components to a **unique**-constraint (basically corresponds to relation, row, column(s)):

- The scope, which delimits the part of the XML document, in which the uniqueness must hold.

Every element of the type in which the **unique**-constraint is defined is one such scope.

- The elements which are identified.

The XPath-expression in **selector** specifies how to get from a scope-element to these elements (“target node set”).

- The values which identify these elements.

The XPath-expressions in one or more **field**-elements specify how to get from the identified elements to the identifying values.

Unique/Key Constraints (4)

- In the example:
 - The scope is the **STUDENTS**-element.

In the example, there is only one **STUDENTS**-element. If there were more than one, the uniqueness has to hold only within each single element.
 - The elements that are identified are the children of **STUDENTS** (the **STUDENT**-elements).

One could also write `xpath="STUDENT"`.
 - The value that identifies the elements is the value of the **SID**-child.

Unique/Key Constraints (5)

- The correspondence of the scope to a relation is not exact:
 - In the example, it is also possible to define the entire document as scope, but to select only **STUDENT**-elements (see next slide).
 - In contrast to the **ID**-type, it is no problem if other keys contain the same values.

Even if the scope is global, the uniqueness of values must hold only within a key (i.e. one could say that the scope is the key).
- Only values of simple types can be used for unique identification.

Unique/Key Constraints (6)

- SID-values uniquely identify STUDENT-elements:

```

<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:unique name="STUDENTS_KEY">
    <xs:selector xpath="STUDENTS/STUDENT"/>
    <xs:field xpath="SID"/>
  </xs:unique>
</xs:element>

```

Unique/Key Constraints (7)

- Example with composed key:

```

<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:unique name="EXERCISES_KEY">
    <xs:selector xpath="EXERCISES/*"/>
    <xs:field xpath="CAT"/>
    <xs:field xpath="ENO"/>
  </xs:unique>
</xs:element>

```

Unique/Key Constraints (8)

- Suppose we store the data in attributes:

```
<EXERCISE CAT='H' ENO='1'
      TOPIC='Rel. Algeb.' MAXPT='10' />
```

- Attributes as fields are marked with "@":

```
<xs:element name="GRADES-DB">
  ...
  <xs:unique name="EXERCISES_KEY">
    <xs:selector xpath="EXERCISES/*" />
    <xs:field xpath="@CAT" />
    <xs:field xpath="@ENO" />
  </xs:unique>
</xs:element>
```

Unique/Key Constraints (9)

- Example with exercise info nested in categories:

```

<EXERCISES>
  <CATEGORY CAT="H">
    <EX ENO="1" TOPIC="Rel. Algeb." MAXPT="10"/>
    <EX ENO="2" TOPIC="SQL" MAXPT="10"/>
  </CATEGORY>
  <CATEGORY CAT="M">
    <EX ENO="1" TOPIC="SQL" MAXPT="14"/>
  </CATEGORY>
</EXERCISES>

```

- XML Schema supports only a subset of XPath.
In particular, one cannot access ancestors in `xs:field`.
But the unique identification of `EX` needs `CAT`.

Unique/Key Constraints (10)

- The problem is solved by defining two keys:
 - One key ensures that the **CAT**-value uniquely identifies **CATEGORY**-elements.
 - The other key is defined within the **CATEGORY** element type (thus, there is one instance of the key, i.e. scope, for every category element). This key ensures the unique identification of **EX**-elements by the **ENO** within each **CATEGORY** element.
- However, in this way no foreign keys can be specified that reference **EX**-elements by **CAT** and **ENO**.

Unique/Key Constraints (11)

- Key on **CATEGORY**:

```

<xs:element name="GRADES-DB">
  ...
  <xs:unique name="CATEGORY_KEY">
    <xs:selector xpath="EXERCISES/CATEGORY"/>
    <xs:field xpath="@CAT"/>
  </xs:unique>
</xs:element>

```

The XPath-expression in **selector** could also be **EXERCISES/*** (because **EXERCISES** has only **CATEGORY**-elements as children).

One could define the key also under **EXERCISES** (instead of **GRADES-DB**) since the document contains only one element of type **EXERCISES**, and all elements to be identified are nested within this element.

Unique/Key Constraints (12)

- Key on **EX**-elements within **CATEGORY**:

```

<xs:element name="CATEGORY">
  ...
  <xs:unique name="EX_KEY">
    <xs:selector xpath="*" />
    <xs:field xpath="@ENO" />
  </xs:unique>
</xs:element>

```

- It is no problem that there are two **EX**-elements with the same **ENO** (e.g., 1) as long as they are nested within different **CATEGORY**-elements.
- This is similar to a weak entity.

Unique/Key Constraints (13)

- For a given “context node” (in which the key is defined), the selector defines a “target node set”.
- For each node in the target node set, the XPath-expression in each field must return 0 or 1 values. It is an error if more than one value is returned.
- The target nodes, for which each field has a value (that is not nil), form the “qualified node set”.
- The unique identification is required only for the qualified node set. Multiple elements with undefined or partially defined key values can exist.

Unique/Key Constraints (14)

- If one writes `xs:key` instead of `xs:unique`, the fields must exist.
 - In this case, it is an error if the XPath-expression in `xs:field` returns no values.
 - And it is always an error if it returns more than one value.
 - Furthermore, neither the identified nodes nor the identifying fields may be nillable.
- Note that value equality respects the type:
 - For a field of type `integer`, "03" and "3" are the same (so the uniqueness would be violated).
 - For a field of type `string`, they are different.

Key References (1)

- A “key reference” identity constraint corresponds to a foreign key in relational databases.
- It demands that certain (tuples of) values must appear as identifying values in a key constraint.

“Key constraint” means **key** or **unique**.

- Example: For each **SID**-value in a **RESULT** element, there must be a **STUDENT**-element with the same **SID** (one can store points only for known students).

As in relational databases, it is not required that the two fields have the same name.

Key References (2)

- **SID-values in RESULT** reference **SID-values in STUDENT**:

```

<xs:element name="GRADES-DB">
  ...
  <xs:key name="STUDENT_KEY">
    <xs:selector xpath="STUDENTS/STUDENT"/>
    <xs:field xpath="SID"/>
  </xs:key>

  <xs:keyref name="RESULT_REF_STUDENT"
    refer="STUDENT_KEY">
    <xs:selector xpath="RESULTS/RESULT"/>
    <xs:field xpath="SID"/>
  </xs:keyref>
</xs:element>

```

Key References (3)

- The referenced key must be defined in the same node or in a descendant node (i.e. “below”) the node in which the foreign key constraint is defined.

I would have required the opposite direction, because on the way up, there could be only one instance of the referenced key, on the way down, there can be several (see below). But the committee certainly had reasons, probably related to the parsing/checking algorithms.

- The standard explains that “node tables” which map key values to the identified nodes are computed bottom-up.

The standard talks of “key sequence” instead of “key values” to include also composed keys (with more than one field).

Key References (4)

- It is possible that several instances of the referenced key exist below the foreign key.
- In that case, the union of the node tables is taken, with conflicting entries removed.

I.e. if two instances of the referenced key contain the same key value with different identified nodes, that key value is removed from the table: It cannot be referenced (the reference would not be unique).

The situation is even more complicated, if the key is defined in an element type that has descendants of the same type. Then key value-node pairs originating in the current node take precedence over pairs that come from below. Values that come from below are only entered in the node table if they do not cause a conflict.

Key References (5)

- Fields of key and foreign key are matched by position in the identity constraint definition, not by name (as in relational databases).
- Normally, the types of corresponding fields (of the key and the foreign key) should be the same.
- However, if the types of both columns are derived from the same primitive type, it might still work (for values in the intersection of both types).
- But values of unrelated types are never identical: E.g. the string “1” is different from the number “1”.

Inhalt

- 1 Introduction, First Example
- 2 Schema Styles
- 3 Attributes
- 4 Integrity Constraints
- 5 Advanced Constructs**

Derived Complex Types (1)

- There are two ways to derive complex types:
 - by extension, e.g. adding new elements at the end of the content model, or adding attributes,
 - by restriction, e.g. removing optional elements or attributes, or restricting the data type of attributes, etc.
- Derived simple types are always restrictions.

One can extend a simple type by adding attributes, but then it becomes a complex type.

Derived Complex Types (2)

- Extension looks very similar to subclass definitions in object-oriented languages.

There all attributes from the superclass are inherited to the subclass, and additional attributes can be added.

- However, a basic principle in object-oriented languages is that a value of a subclass can be used wherever a value of the superclass is needed.

- In XML, it depends on the application, whether it breaks if there are additional elements/attributes.

Since XML Schema has this feature, future applications should be developed in a way that tolerates possible extensions.

Derived Complex Types (3)

- Additional attributes are probably seldom a problem, since attributes are typically accessed by name (not in a loop).
- It was tried to minimize the problems of additional child elements by allowing them only at the end of the content model.
- Formally, the content model of the extended type is always a **sequence** consisting of
 - the content model of the base type,
 - the added content model (new child elements).

Derived Complex Types (4)

- Consider a type for **STUDENT**-elements:

```

<xs:complexType name="STUDENT_TYPE">
  <xs:sequence>
    <xs:element name="SID" type="SID_TYPE"/>
    <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>

```

- Suppose that exchange students must in addition contain the name of the partner university.

Derived Complex Types (5)

- Example for type extension:

```

<xs:complexType name="EXCHANGE_STUDENT_TYPE">
  <xs:complexContent>
    <xs:extension base="STUDENT_TYPE">
      <xs:sequence>
        <xs:element name="PARTNER_UNIV"
          type="UNIV_TYPE"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

```

- The effective content model is now:

((SID, FIRST, LAST, EMAIL?), (PARTNER_UNIV))

Derived Complex Types (6)

- In the same way, one can add attributes. Suppose that `STUDENT_TYPE2` has attributes `SID`, `FIRST`, `LAST`, `EMAIL` (and empty content).
- Then a new attribute is added as follows:

```
<xs:complexType name="EXCHANGE_STUDENT_TYPE2">
  <xs:complexContent>
    <xs:extension base="STUDENT_TYPE2">
      <xs:attribute name="PARTNER_UNIV"
        type="UNIV_TYPE" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Derived Complex Types (7)

- Let us return to the case where **STUDENT** has child elements **SID**, **FIRST**, **LAST**, **EMAIL**.
- The type of **EMAIL** might be a simple type:

```
<xs:simpleType name="EMAIL_TYPE">
  <xs:restriction base="xs:string">
    <xs:maxLength value="80"/>
  </xs:restriction>
</xs:simpleType>
```

- Suppose that an attribute must be added that indicates whether emails can be formatted in HTML or must be plain text.

Derived Complex Types (8)

- When an attribute is added to a simple type, one gets a complex type:

```
<xs:complexType name="EMAIL_TYPE2">
  <xs:simpleContent>
    <xs:extension base="EMAIL_TYPE">
      <xs:attribute name="HTML_OK"
        type="xs:boolean" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

- Example (element `EMAIL` of type `EMAIL_TYPE2`):

```
<EMAIL HTML_OK="false">brass@acm.org</EMAIL>
```

Derived Complex Types (9)

- If one uses restriction to define a derived type, it is guaranteed that every value of the derived type is also a valid value of the original type.
- If one wants to restrict a content model, one must repeat the complete content model.

I.e. also the unmodified parts must be listed. The restricted content model does not have to be structurally identical. E.g. groups with only a single element can be eliminated (if `minOccurs` and `maxOccurs` are both 1), a `sequence` group with `minOccurs="1"` and `maxOccurs="1"` can be merged with an enclosing `sequence` group, the same for `choice`-groups. However, for `all` and `choice` groups, subgroups must be listed in the same order, although the sequence is semantically not important.

Derived Complex Types (10)

- If one wants to restrict an attribute, it suffices to repeat only this attribute.
- Consider again `STUDENT_TYPE2` with attributes `SID`, `FIRST`, `LAST`, `EMAIL`. The optional attribute `EMAIL` can be removed as follows:

```

<xs:complexType name="STUDENT_TYPE3">
  <xs:complexContent>
    <xs:restriction base="STUDENT_TYPE2">
      <xs:attribute name="EMAIL"
        use="prohibited"/>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

```

Derived Complex Types (11)

- The same change for the type **STUDENT** with child elements **SID**, **FIRST**, **LAST**, **EMAIL** (`minOccurs="0"`):

```

<xs:complexType name="STUDENT_TYPE4">
  <xs:complexContent>
    <xs:restriction base="STUDENT_TYPE">
      <xs:sequence>
        <xs:element name="SID" type="SID_TYPE"/>
        <xs:element name="FIRST" type="xs:string"/>
        <xs:element name="LAST" type="xs:string"/>
      </xs:sequence>
    </xs:restriction>
  </xs:complexContent>
</xs:complexType>

```

Derived Complex Types (12)

- Possible restrictions for complex types:
 - Optional attribute becomes required/prohibited.
 - The cardinality of elements or model groups becomes more restricted (`minOccurs` ↑, `maxOccurs` ↓).
 - Alternatives in `choice`-groups are reduced.
 - A restricted type can be chosen for an attribute or a child element.
 - A default value can be changed.
 - An attribute or element can get a fixed value.
 - Mixed content can be forbidden.

Documentation, App. Info (1)

- Documentation about the schema can be stored within the XML Schema definition.

And not only as XML comments: Many XML tools suppress comments, and very little formatting can be done there.

- This is one purpose of the `annotation` element type, which is allowed

- as first child of every XML Schema element type

But it cannot be nested, i.e. it cannot be used within `annotation` or its children `documentation` and `appinfo`.

- anywhere as child of `schema` and `redefine`.

There, multiple `annotation` elements are allowed. Inside all other element types, only one `annotation` element is permitted.

Documentation, App. Info (2)

- Many relational databases also have the possibility to store comments about tables and columns in the data dictionary.

Of course, this is usually pure text, quite short and without formatting.

- The other purpose of the annotation element is to store information for tools (programs) that process XML Schema information within the schema.

E.g. tools that compute a relational schema from an XML schema, and map data between the two, or tools that generate form-based data entry programs out of the schema data.

- This makes XML Schema extensible.

Documentation, App. Info (3)

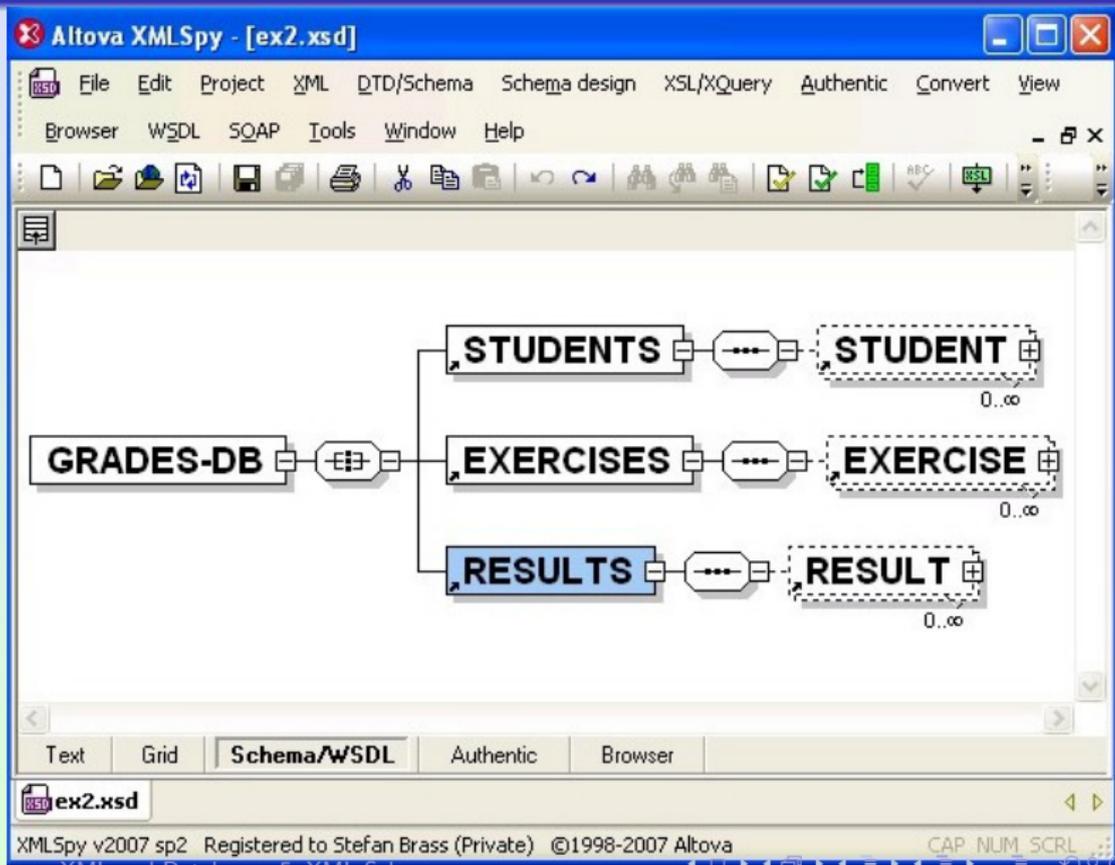
- Example:

```

<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:doc="http://doc.org/d1"
  xmlns:xsi=
    "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://doc.org/d1 doc.xsd">
<xs:element name="GRADES-DB">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      <doc:title>Grades Database</doc:title>
      This is the root element.
      ...
    <xs:complexType>
      ...
    </xs:complexType>
  </xs:annotation>
</xs:element>
</xs:schema>

```

Visualization of Schema Structure



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.
[\[http://www.w3.org/XML/Schema\]](http://www.w3.org/XML/Schema)
- David C. Fallside, Priscilla Walmsley: XML Schema Part 0: Primer.
W3C, 28. October 2004, Second Edition.
[\[http://www.w3.org/TR/xmlschema-0/\]](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
[\[http://www.w3.org/TR/xmlschema-1/\]](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.w3.org/TR/xmlschema-2/)
- [\[http://www.w3schools.com/schema/\]](http://www.w3schools.com/schema/)