Chapter 2: Designing XML DTDs

References:

- Tim Bray, Jean Paoli, C.M. Sperberg-McQueen:
  Extensible Markup Language (XML) 1.0, 1998.
  [http://www.w3.org/TR/REC-xml] See also: [http://www.w3.org/XML].
- Elliotte R. Harold, W. Scott Means: XML in a Nutshell, 3rd Ed.
  O'Reilly, 2004, ISBN 0596007647.
- Didier Martin, Mark Birbeck, Michael Kay: Professional XML, 2nd Ed. Wrox, 2000.
- Henning Lobin: Informationsmodellierung in XML und SGML. Springer-Verlag, 1999.
- Erhard Rahm, Gottfried Vossen: Web & Datenbanken. Dpunkt Verlag, 2002.
- Meike Klettke, Holger Meyer: XML & Datenbanken. Dpunkt Verlag, 2002.
- Akmal B. Chaudhri et al.: XML Data Management. Addison-Wesley, 2003.
- Eve Maler, Jeanne El Andaloussi:
  Developing SGML DTDs: From Text to Model to Markup.
  Prentice Hall PTR, 1996.

Objectives

After completing this chapter, you should be able to:

- develop an XML DTD for a given application.
- translate a given Entity-Relationship-Diagram or relational database schema into an XML DTD.

Overview

Motivation (1)

- In order to use XML, one must specify the document/data file structure.
- This specification does not necessarily have to be in the form of a DTD, but DTDs are simple and there are many tools that work with DTDs.

  DTDs were inherited from SGML, and are more intended for documents. Databases have other restrictions that cannot be expressed in DTDs, therefore XML documents might be valid with respect to the specified DTD that do not correspond to a legal database state. XML Schema was developed as an alternative to DTDs that fulfills better the special requirements of databases.

Motivation (2)

- Often, XML is used as an exchange format between databases. Then it is clear that one must find an XML structure that corresponds to the given DB.
- There are a lot of methods, tools, and theory for developing database schemas.
- Therefore, even if one does not (yet) store the data in a database, it makes sense to develop first a DB schema in order to design an XML data structure.

  If XML is used as a poor man's database, and not for "real" documents which typically have a less stringent structure.

Example Database (1)

STUDENTS

SID	FIRST	LAST	EMAIL
101	Ann	Smith	···
102	Michael	Jones	(null)
103	Richard	Turner	···
104	Maria	Brown	···

EXERCISES

CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

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

Example Database (2)

- STUDENTS: one row for each student in the course.
  - ◇ SID: “Student ID” (unique number).
  - ◇ FIRST, LAST: First and last name.
  - ◇ EMAIL: Email address (can be null).
- EXERCISES: one row for each exercise.
  - ◇ CAT: Exercise category.

    E.g. 'H': homework, 'M': midterm exam, 'F': final exam.
  - ◇ ENO: Exercise number (within category).
  - ◇ TOPIC: Topic of the exercise.
  - ◇ MAXPT: Max. no. of points (How many points is it worth?).

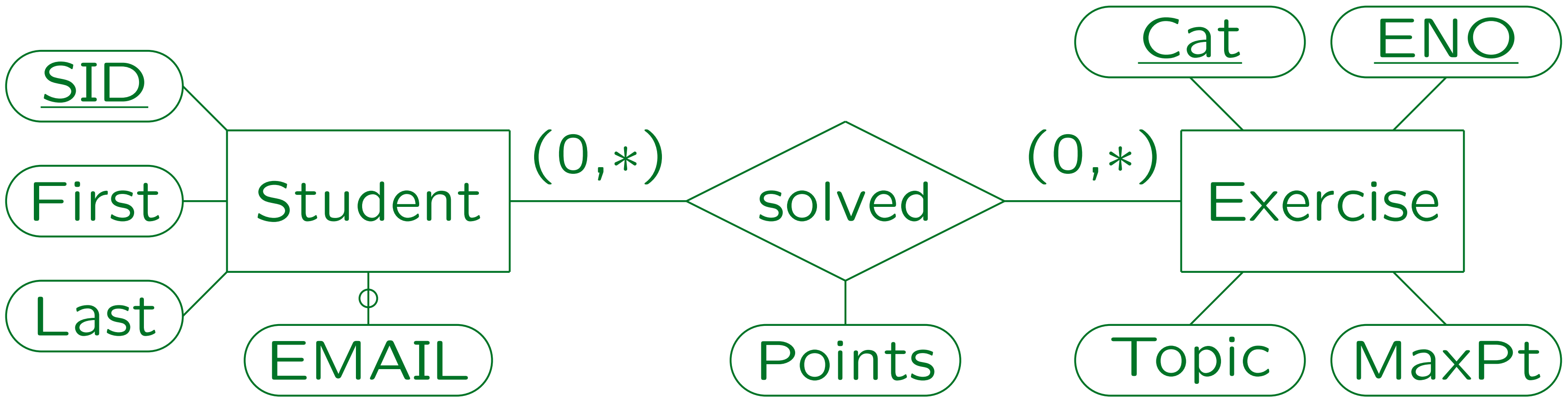
Example Database (3)

- `RESULTS`: one row for each submitted solution to an exercise.
  - ◇ `SID`: Student who wrote the solution.

    This references a row in `STUDENTS`.
  - ◇ `CAT`, `ENO`: Identification of the exercise.

    Together, this uniquely identifies a row in `EXERCISES`.
  - ◇ `POINTS`: Number of points the student got for the solution.
  - ◇ A missing row means that the student did not yet hand in a solution to the exercise.

Example Database (4)



- This is an equivalent schema in the ER-Model.

  ER = Entity-Relationship. Entities are another name for objects (object types / classes are shown as boxes in the ER-diagram). Relationships between objects (object types) are shown as diamonds. Attributes are pieces of data that are stored about objects or relationships (shown as ovals). Optional attributes are marked with a circle. Key attributes (which uniquely identify objects) are underlined.

Overview

Table Rows: Method I

- A simple and natural way to encode relational data is to use one empty element per table row:

```
<STUDENT SID='101' FIRST='Ann' LAST='Smith'
         EMAIL='smith@acm.org'/>
```

- This could be declared as follows:

```
<!ELEMENT STUDENT EMPTY>
<!ATTLIST STUDENT SID   CDATA #REQUIRED
                  FIRST CDATA #REQUIRED
                  LAST  CDATA #REQUIRED
                  EMAIL CDATA #IMPLIED>
```

- See next slide for the data type of `SID`.

Data Types, Keys (1)

- In SGML, SID can be declared as NUMBER (instead of CDATA). NUMBER-values are sequences of digits (≥ 0).

  In XML, this is not supported. The nearest one could come would be NMTOKEN, but that would also permit letters (as well as -, _, :, .). This might make the real data type even less clear.

- If references to students are needed (see below), ID might be the right type for the attribute SID.

  This is supported in SGML and XML. However, now the restriction is lost that SID is a number.

- But note that ID-values must start with a letter.

  Or "_" or ":". Thus, the data values have to be changed, e.g. "S101" instead of "101".

Data Types, Keys (2)

- Note also that `ID`-values must be globally unique in an XML document.

  In contrast, key values have to be unique only within a relation (corresponding to an element type in this translation).

- Finally, composed keys (e.g., `CAT` and `ENO`) cannot be directly translated to `ID`-attributes.

  In the example, one could concatenate the two attributes, this would also solve the problem that `ID`-values must start with a letter: E.g., `H1`, `H2`, `M1`. The problem with this is that it is now more difficult to access category and exercise number separately.

- It might be good to choose the attribute name `ID` instead of `SID` (purpose clear even without DTD).

Data Types, Keys (3)

- These problems to represent data types in XML has led to the XML Schema proposal.

  Specifications in XML Schema are an alternative to DTDs. XML Schema permits basically all that is possible in classical databases (and more), but it is much more complicated than DTDs. Whereas DTDs use a different syntax than the XML data syntax, XML Schema specifications are valid XML documents. Unfortunately, this also means that XML Schema specifications are significantly longer than the corresponding DTD.

- When the XML data are only an export from a database, and not directly modified, it is unnecessary to specify all constraints also for the XML file.

  They are automatically satisfied.

Data Types, Keys (4)

- The most common XML data types for attributes are `CDATA` (strings), `ID` (unique identifiers), `NMTOKEN` (words/codes), and enumeration types.
- E.g., if it is clear that the only possible exercise categories are homeworks, midterm, and final, exercises could be represented as follows:

```
<!ELEMENT EXERCISE EMPTY>
<!ATTLIST EXERCISE CAT   (H|M|F) #REQUIRED
                   ENO    CDATA  #REQUIRED
                   TOPIC  CDATA  #REQUIRED
                   MAXPT  CDATA  #REQUIRED>
```

Special Characters

- If the XML file is generated by exporting data from a database, characters that are forbidden within attribute values must be escaped:
  - ◇ Replace "<" by "`&lt;`".
  - ◇ Replace "&" by "`&amp;`".
  - ◇ Replace an apostrophe (') by "`&apos;`".

    If this character is used as a string delimiter.
- Special national characters must be represented in UTF-8, or an XML declaration that specifies an encoding must be used.

Table Rows: Method II (1)

- An alternative is to use a nested structure with an element per attribute, plus one per row:

```
<STUDENT>
    <SID>101</SID>
    <FIRST>Ann</FIRST>
    <LAST>Smith</LAST>
    <EMAIL>smith@acm.org</EMAIL>
</STUDENT>
```

- Advantage (?): Only elements, no attributes.
- Disadvantage: Longer.

Table Rows: Method II (2)

- The declaration in an XML DTD would look as follows:

```
<!ELEMENT STUDENT (SID, FIRST, LAST, EMAIL?)>
<!ELEMENT SID     (#PCDATA)>
<!ELEMENT FIRST   (#PCDATA)>
<!ELEMENT LAST    (#PCDATA)>
<!ELEMENT EMAIL   (#PCDATA)>
```

- In SGML, one could use "&" instead of "," to permit an arbitrary sequence of the subelements.

  In XML, one would have to simulate this with "|". For a larger number of columns, this gives very long and complex content models.

Method I vs. Method II (1)

- In the document processing community, one usually puts the real text into the element content, so that one would still get the important information if all tags were removed.
- In the example, one could discuss whether `SID` and `EMAIL` should be attributes as in Method I, but at least first name and last name should be elements as in Method II.
- The database community has no such rules.

Method I vs. Method II (2)

- Example for a mixture of both methods:

```
<STUDENT ID='S101' EMAIL='smith@acm.org'>
    <FIRST>Ann</FIRST>
    <LAST>Smith</LAST>
</STUDENT>
```

- This can be declared as follows:

```
<!ELEMENT STUDENT (FIRST, LAST)>
<!ATTLIST STUDENT ID    ID    #REQUIRED
                  EMAIL CDATA #IMPLIED>
<!ELEMENT FIRST   (#PCDATA)>
<!ELEMENT LAST    (#PCDATA)>
```

Method I vs. Method II (3)

- Method II is of course advantageous if attributes values are XML data.

  If one does not know the structure, the content model `ANY` can be used (note that there are no parentheses around `ANY`).

- Enumeration values and unique identifications are only possible with Method I.
- Of course, also in Method II, special characters must be escaped. An alternative is to use a `CDATA`-section.

Overview

Grouping Rows (1)

- One possibility is to create one element for each relation/table, e.g.

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
  <STUDENTS>
    <STUD SID='101' FIRST='Ann' .../>
    <STUD SID='102' FIRST='Michael' .../>
    ...
  </STUDENTS>
  ...
</GRADES-DB>
```

Grouping Rows (2)

- An alternative is to have no such containers:

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

Grouping Rows (3)

- If there are no table groups, one can require that all rows of a single table are written consecutively:

```
<!ELEMENT GRADES-DB (STUDENT*,
                     EXERCISE*,
                     RESULT*)>
<!ELEMENT STUDENT   EMPTY>
...
```

- An alternative is to require no specific sequence:

```
<!ELEMENT GRADES-DB (STUDENT|EXERCISE|RESULT)*>
```

Grouping Rows (4)

- Within a table, one can group rows by an attribute:

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

- “EX” does not need an attribute “CAT”: Its value can be derived from the enclosing “CAT” element.

Ordered Data (1)

- In relational databases, the rows within a table have no specific sequence.
- If the sequence is important, and cannot be recovered by ordering the rows by one of the columns, a column must be added that encodes the sequence (e.g. a number).
- XML documents are always ordered.
- Therefore it might be possible to leave out columns that can be derived from the position in the file.

Ordered Data (2)

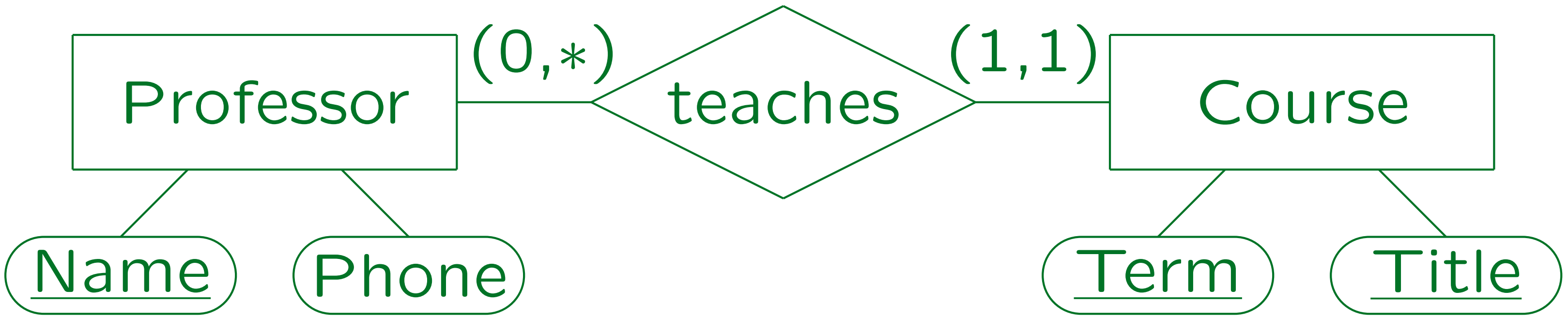
- If e.g. the exercise number is always sequential, it does not have to be stored explicitly:

```
<EXERCISES>
  <CAT LETTER='H'>
    <EX TOPIC='Rel. Algeb.' MAXPT='10'/>
    <EX TOPIC='SQL' MAXPT='10'/>
  </CAT>
  <CAT LETTER='M'>
    <EX TOPIC='SQL' MAXPT='14'/>
  </CAT>
</EXERCISES>
```

Overview

1:n Relationships (1)

- The example database contains only a many-to-many (n:m) relationship, which is discussed below.
- An example for a one-to-many (1:n) relationship is:



- One professor teaches many courses (between 0 and $* = \infty$, i.e. arbitrarily many), but each course is taught by exactly one professor (min 1, max 1).

1:n Relationships (2)

- In the relational model, the relationship “teaches” is implemented by adding the key of professor as a foreign key to the course table:

PROFESSORS

NAME	PHONE
Brass	55-24740
Zimmermann	55-24712

COURSES

TERM	TITLE	PROF
Summer 2004	Databases Design	Brass
Winter 2004	Foundations of the WWW	Brass
Summer 2004	Compiler Construction	Zimmermann

1:n Relationships (3)

- One-to-many relationships can be easily represented by nesting the elements for the "many" side (in this case Course) into the elements for the "one" side (Professor): See example on next slide.
- This nesting can be extended to arbitrary depth to represent a tree of one-to-many relationships.
- Example: Suppose that the times when the course meets (class) has to be stored for each course (courses can meet several times per week).

1:n Relationships (4)

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<COURSE-DB>
    <PROFESSOR NAME='Brass' PHONE='55-24740'>
        <COURSE TERM='Summer 2004'
                TITLE='Database Design'/>
        <COURSE TERM='Winter 2004'
                TITLE='Foundations of the WWW'/>
    </PROFESSOR>
    <PROFESSOR NAME='Zimmermann' PHONE='55-24712'>
        <COURSE TERM='Summer 2004'
                TITLE='Compiler Construction'/>
    </PROFESSOR>
</COURSE-DB>
```

1:n Relationships (5)

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<COURSE-DB>
    <PROFESSOR NAME='Brass' PHONE='55-24740'>
        <COURSE TERM='Summer 2004'
                TITLE='Database Design'>
            <CLASS DAY='MON' FROM='10' TO='12'/>
            <CLASS DAY='THU' FROM='16' TO='18'/>
        </COURSE>
        <COURSE TERM='Winter 2004'
                TITLE='Foundations of the WWW'>
            <CLASS DAY='WED' FROM='14' TO='16'/>
        </COURSE>
    ...
```

Foreign Keys (1)

- An alternative is to represent the professor-course relationship with ID- and IDREF-attributes:

```
<!ELEMENT COURSE-DB (PROFESSOR|COURSE)*>
<!ELEMENT PROFESSOR EMPTY>
<!ATTLIST PROFESSOR NAME  ID    #REQUIRED
                    PHONE CDATA #REQUIRED>
<!ELEMENT COURSE    EMPTY>
<!ATTLIST COURSE    TERM  CDATA #REQUIRED
                    TITLE CDATA #REQUIRED
                    PROF  IDREF #REQUIRED>
```

- This is very similar to the relational solution.

Foreign Keys (2)

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<COURSE-DB>
    <PROFESSOR NAME='Brass' PHONE='55-24740'/>
    <COURSE TERM='Summer 2004' PROF='Brass'
            TITLE='Database Design'/>
    <COURSE TERM='Winter 2004' PROF='Brass'
            TITLE='Foundations of the WWW'/>

    <PROFESSOR NAME='Zimmermann' PHONE='55-24712'>
    <COURSE TERM='Summer 2004' PROF='Zimmermann'
            TITLE='Compiler Construction'/>
</COURSE-DB>
```

Foreign Keys (3)

- IDREF-attributes are similar to foreign keys, but there are the following differences:
  - ◇ ID/IDREF-attribute values must be identifiers.

    E.g. "S. Brass" could not be used as value of an ID-attribute.
  - ◇ IDREF attributes can refer to any element that as an ID-attribute. One cannot specify that PROF in COURSE must point to a professor.

    In the example, only PROFESSOR has an ID-attribute. Then this problem does not occur.
  - ◇ The relational model permits keys that consist of several attributes (not supported in DTDs).

n:m Relationships (1)

- The student grades database contains a many-to-many (n:m) relationship:
  - ◇ One student can solve many exercises.
  - ◇ One exercise can be solved by many students.
- In this case references are unavoidable (at least if duplicate storage of the same entities is excluded):
  - ◇ Either the results are nested under students (then result elements must point to exercises),
  - ◇ or results are nested under exercises (then they must point to students).

n:m Relationships (2)

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
    <STUDENT SID='101' FIRST='Ann' LAST='Smith'
             EMAIL='smith@acm.org'>
        <RESULT CAT='H' ENO='1' POINTS='10'/>
        <RESULT CAT='H' ENO='2' POINTS='8'/>
        <RESULT CAT='M' ENO='3' POINTS='12'/>
    </STUDENT>
    ...
    <EXERCISE CAT='H' ENO='1' TOPIC='Rel. Algeb.'
              MAXPT='10'/>
    ...
</GRADES-DB>
```

n:m Relationships (3)

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<GRADES-DB>
    <STUDENT ID='S101' FIRST='Ann' LAST='Smith'
             EMAIL='smith@acm.org'/>
    ...
    <EXERCISE CAT='H' ENO='1' TOPIC='Rel. Algeb.'
              MAXPT='10'>
        <RESULT STUD='S101' POINTS='10'/>
        <RESULT STUD='S102' POINTS='9'/>
        <RESULT STUD='S103' POINTS='5'/>
    </EXERCISE>
    ...
</GRADES-DB>
```

n:m Relationships (4)

- As is well known in databases, one can replace a many-to-many relationship by an "association entity" (RESULTS) and two one-to-many relationships.
- One of the relationships is represented by nesting the elements, the other relationship is represented by references.

  Given an arbitrary ER-diagram, one would first replace the many-to-many relationships in this way by association entities, and then cut the resulting graph of one-to-many relationships into trees. References are needed for cutted edges, the trees are represented by nesting. This technique was already used for the very old hierarchical data model.