# XML and DTD

Week 7






## Learning Objectives

- Semistructured Data
- XML
- DTD
- XML Schema (Brief Introduction)

# Semistructured Data

- Structured data
  - Has schema
  - Stored in relational database (SQL systems)
- **Semistructured** data
  - Not all data has identical structure
  - Schema information mostly mixed with data
  - Self-describing
  - Stored in NoSQL systems
  - Example: **XML/JSON** documents, graphs, column-based, key-value pairs
- Unstructured data
  - Limited data type indication
  - Example: Web pages in HTML





# XML

- XML is a standard for data representation and exchange
  – W3C XML 1.0 and 1.1 http://www.w3.org/XML/
- It provides a common format for expressing both data structures and contents
- Markup language for structured information
- A markup language is a set of symbols that can be placed in the text of a document to create boundaries and label the parts of that document
- XML is a meta-markup language because it lets you create your own markup language
- XML allows author to *customise* own elements, i.e., their own tags
- The elements of the author's own choice are used to structure data and provide it meaning
- Native XML DBMSs: MarkLogic, Virtuoso, Oracle Berkeley DB
  ( https://db-engines.com/en/article/Native+XML+DBMS )

# XML vs HTML

- HTML has fixed tag semantics (defined by browser behaviour) and pre-defined tags (for static Web pages)
  - <H1>, <P>, <BODY>, <TD> …
  - W3C keeps extending this tag set and semantics (ie, what happens when displayed in a browser)
- XML specifies neither a tag set nor semantics (for dynamic Web pages)
  - It is a meta-language for describing (markup and other) languages
  - You can define your own tags, and the structure of these tags
  - Semantics are provided by applications that process XML documents or by style-sheets

5

5

# XML vs HTML : Simple Example

```
<TABLE>
  <TR>
    <TD>Thomas</TD><TD>Atkins</TD>
  </TR>
  <TR>
    <TD>age:</TD><TD>30</TD>
  </TR>
</TABLE>
```

HTML, using pre-defined tags which have pre-defined meanings wrt presentation in a browser

```
<Person>
  <Name>
    <First>Thomas</First>
    <Last>Atkins</Last>
  </Name>
  <Age>30</Age>
</Person>
```

XML, with user-defined tags which are chosen to convey intent of their content

6

6

# XML Elements and Syntax

- The example shows the following
  - **Boundaries**. Tags <section> and </section> surround collection of text and markup
  - **Roles**. <sectionname> .. </sectionname> has a different purpose from <ref> .. </ref>.
  - **Positions**. The first <ref> .. </ref> is placed logically after <sectionname> .. </sectionname> and sensibly would be printed to a browser like this.
  - **Containment**. Both <sectionname> and <ref> elements are nested within the <section> element.

```
<section>
    <sectionname>References</sectionname>
    <ref>Castro E., XML for the World Wide Web, Peachpit Press, 2000</ref>
    <ref>Deitel, H., et al., XML: How to Program, Prentice Hall, 2001</ref>
    <ref>Holzner, S., Inside XML, New Riders Publishing, 2001</ref>
</section>
```

7

7

# Well-formedness vs Validity

- All XML documents need to be
  - Well-formed (compulsory)
  - Valid against a Schema/DTD (optional)
- Parsers will check if document is well-formed
- There are Validating parsers and Non-Validating parsers
- With a validating parser, the validity is checked if the document refers to a specified DTD or XML Schema

8

8

# A Few Definitions

- Content
  - "The mobile phone revolution has just begun"
- Tags
  - <statement> The mobile phone … </statement>
  - <statement> = **Start tag**
  - </statement> = **End tag**
- Tags are case sensitive: <name>, <NAME>, <Name> are treated as different (**CAUSES MANY ERRORS**!)
- Element = Tag + Content

9

9

# Element

- An Element in XML consists of:
  - a start tag, content, and an end tag
  - e.g. <dateOfBirth>2003-01-01</dateOfBirth>
- A start tag may include one or more **attributes** of the element
  - <address **type**="permanent"> … </address>
- Empty elements without any content are allowed
  - <applause /> -- same as <applause> </applause>
  - XHTML: <br /> -- same as <br> </br>
  - Note: an empty element may have attributes

10

10

## Elements are Used to Contain

- Elements
- Data (Typical case)
- Character references
- Entity references
- Comments
- Processing instructions
- CDATA Sections

11

11

## XML Character Data

- Character data is text in the document, not markup tags
  - May be of any allowable Unicode value
  - Certain characters are reserved or they are not part of the ASCII character set and must be entered using character or entity references
- Element content is often referred to as parsed-character data (PCDATA)
- PCDATA is any “well-formed” text string, most text strings are “well-formed” except those that contain symbols reserved for XML, such as < > &

12

12

# Entity References

- Used to place *string literals* into elements/attributes
  - Start with &, End with ;
  - e.g. &
- Some pre-defined entity references are:

| | |
|---|---|
| &amp; | & |
| &lt; | < |
| &gt; | > |
| &apos; | ' |
| &quot; | " |

Can define own entity references

13



# Character References

- A *character reference* is where use is made of a decimal or hexadecimal number to represent a character able to be stored in XML data and be displayable (for instance, in a Web browser), e.g., ©
- Such a character is not able to be placed in its displayable form into a document because it is not available from the input device, e.g., a Japanese character trying to be entered in a Turkish word processor
- Allows to represent Unicode characters
- Allowed forms of references are:
  - Decimal: &#DDDDD; (1 to 5 digits), e.g., ©
  - Hexadecimal: &#xHHHH; (1 to 4 digits), e.g., ©
  - Both the above represent ©
- Hexadecimal form is preferred
- Cannot use character references for element / attribute names

14

## Comments, PIs and CDATA

- Comments are like in HTML
  - `<!-- This is a comment -->`
- Processing Instructions (PIs) - processing hint, script code or presentational info can be indicated to a parser
  - `<?` Some processing instruction `?>`
  - is parser-specific, so not all parsers will respond to a PI's in the same way
  - Examples:

```
<?xml-stylesheet href="style.css" type="text/css"?>
```

- CDATA sections are used to capture any data that may confuse the parser. They will not be parsed
  - Example:

```
<![CDATA[
   <html><body>Content</body></html>
]]>
```

15

15

## XML Documents Contain

- an optional *prolog*, which contains
  - XML declaration
  - Miscellaneous statements or comments
  - Document type declaration
  - This order has to be followed or the parser will generate an error message

```
<?xml version=1.0 encoding="UTF-8" standalone="no" ?>
<!-- This document describes one book -->
<!DOCTYPE book SYSTEM "book.dtd">
```

- a body with a single root element

```
<book>
  <title>Long live XML</title>
</book>
```

- An optional *epilog* – seldom used

16

16

## Well-formedness Rules

- An XML document must follow "Document" *production*, i.e., document contains a *prolog*, a root element and a miscellaneous part to which the following rules apply
- One root element containing all other elements
- Elements must have both a start and end tag, except that *empty* elements end in "/>"
- Elements must nest, i.e., elements do not overlap, e.g.:
  <section><sectionname> …</sectionname> … </section>
- Attribute values in double or single quotes, e.g.:
  <book settext="no">Castro E., XML for the World Wide Web, Peachpit Press, 2000.</book>
- Markup characters (eg, **< > &**) do not appear in parsed content, use entity reference instead, e.g.: <eqn>1 < 3</eqn> for showing 1 < 3
- Element names may begin with letter or underscore or ":"and remaining characters include alphanumeric, "_", "-", ":" and "."
- Cannot use the same name for two or more attributes of the same element

17

17

## classes.xml

```
<?xml version="1.0"?>
<classes>
   <class classID="CS115">
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Adams</instructor><title>Programming Concepts</title>
   </class>
   <class classID="CS205" semester="fall">
      <department>ComputerScience</department><credits req="yes">3</credits>
      <instructor>Dykes</instructor><title>JavaScript</title>
   </class>
   < class classID="CS255" semester="fall">
      <department>ComputerScience</department><credits req="no">3</credits>
      <instructor>Brunner</instructor><title>Java</title>
   </class>
</classes>
```

18

18

# Document Type Definition (DTD)

- Well-formedness (basic requirement of an XML document) doesn't mean that the data is useful or even correct, e.g.,
  - certain contents of data may be incorrect, e.g., contain wrong characters
  - certain elements should not be found in their parent elements or are shown in the wrong order within parent elements
- A *DTD* is validation method that uses rules of formal grammar, *validity constraints*, to define syntax, structure and other details of a document type. It can be used to:
  - Define hierarchy of elements in tree
  - Ensure required elements are present, are children of correct parent and found in right order in parent
  - Prevent unexpected elements from being used
  - Ensure attributes used correctly by elements, e.g., have correct types of values
  - Allow for default values of attributes
  - Describe data that is non-XML

19

19

# Declaring a DTD

- Only one DTD can be used with a document; however, this DTD consists of both an *internal* and *external* subset

```
<!DOCTYPE doc_el SYSTEM location
[
  internal_subset
]>
```

- An *internal subset* is declarations placed in the same file as the document content.
- An *external subset* is located in a separate file (with .dtd extension) – generally much more useful!
- Can have only internal, only external or combination of both
- If you place the DTD within the document, it is easier to compare the DTD to the document's content. However, the real power of XML comes from an external DTD that can be shared among many documents written by different authors.
- If a document contains both an internal and an external subset, the internal subset takes precedence over the external subset if there is a conflict between the two.
- This way, the external subset would define basic rules for all the documents, and the internal subset would define those rules specific to each document.

20

20

# Internal Declaration – Simple Example

```
<?xml version="1.0"?>
<!DOCTYPE exam
  [
    <!ELEMENT exam (weighting)>
    <!ELEMENT weighting (#PCDATA)>
  ]
>
<exam>
  <weighting>60%</weighting>
</exam>
```

Note that the DOCTYPE is called "exam". The root element of an XML document that conforms to this DOCTYPE must be "exam".

21

21

# External Declaration

Save previous internal-subset of the DTD declaration in assess.dtd.
Here we specify that the DTD is to be found in that external file

```
<!ELEMENT exam (weighting)>
<!ELEMENT weighting (#PCDATA)>
```

```
<?xml version="1.0"? standalone="no">
<!DOCTYPE exam SYSTEM "assess.dtd">
<exam>
   <weighting>60%</weighting>
</exam>
```

Note that the root element is still "exam", but because the dtd is external, the word "SYSTEM" is added in its declaration.

22

22

## Mixed Declaration

```
<!ELEMENT exam (weighting)>
<!ELEMENT weighting (#PCDATA)>
```

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<!DOCTYPE asst SYSTEM "assess.dtd"
   [<!ELEMENT asst (exam, assign1) >
    <!ELEMENT assign1 (weighting)>
   ]
>
<asst>
  <exam>
    <weighting>60%</weighting>
  </exam>
  <assign1>
    <weighting>20%</weighting>
  </assign1>
</asst>
```

Note : An ELEMENT that is defined in an external DTD cannot be re-defined in the internal definitions.

23

23

## Example : The dtd (file outline2.dtd)

```
<!ELEMENT OUTLINE (UNIT, OBJECTIVES, REFERENCES, ASSESSMENT)>
<!ELEMENT UNIT (TITLE, CODE+)>
<!ELEMENT TITLE (#PCDATA)>
<!ATTLIST TITLE Credit (SINGLE | DOUBLE) #REQUIRED>
<!ELEMENT CODE (#PCDATA)>
<!ELEMENT OBJECTIVES (OBJECTIVE+)>
<!ELEMENT OBJECTIVE (#PCDATA)>
<!ELEMENT REFERENCES (REFERENCE*)>
<!ELEMENT REFERENCE (#PCDATA)>
<!ELEMENT ASSESSMENT (ASSESSMENTTASK+)>
<!ELEMENT ASSESSMENTTASK (#PCDATA)>
<!ATTLIST ASSESSMENTTASK Type (Indiv | Group) #REQUIRED>
<!ATTLIST ASSESSMENTTASK Value CDATA #REQUIRED>
<!ATTLIST ASSESSMENTTASK DueDate CDATA #REQUIRED>
```

24

24

# Example : An XML file (WAD.xml)

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>
<!DOCTYPE OUTLINE SYSTEM "outline2.dtd">
<OUTLINE>
  <UNIT>
    <TITLE Credit="SINGLE"> Web Application Development </TITLE>
    <CODE> HIT3324 </CODE>
    <CODE> HIT8324 </CODE>
  </UNIT>
  <OBJECTIVES>
    <OBJECTIVE> Learn about Ajax </OBJECTIVE>
    <OBJECTIVE> Learn about XML </OBJECTIVE>
  </OBJECTIVES>
  <REFERENCES>
    <REFERENCE> Beginning Ajax</REFERENCE>
    <REFERENCE> Enterprise Ajax </REFERENCE>
  </REFERENCES>
  <ASSESSMENT>
    <ASSESSMENTTASK Type="Indiv" Value="10%" DueDate="7 Sept 2008"> Ass1 </ASSESSMENTTASK>
    <ASSESSMENTTASK Type="Indiv" Value="15%" DueDate="5 Oct 2008" > Ass2 </ASSESSMENTTASK>
    <ASSESSMENTTASK Type="Indiv" Value="25%" DueDate="7 Nov 2008" > Ass3 </ASSESSMENTTASK>
    <ASSESSMENTTASK Type="Indiv" Value="50%" DueDate="27 Nov 2008" > Exam </ASSESSMENTTASK>
  </ASSESSMENT>
</OUTLINE>
```

25

# Validation

- An XML file that specifies a DTD is **valid** if it satisfies the constraints of the DTD
- This is in addition to the XML being well-formed
- Validity can be checked by several on-line validity checkers
- A good editor to use to "play" with XML is **Oxygen**
- You can download a trial version of this – it lets you experiment with XSLT, as well as DTDs and XML schemas

26

26

# DTD Declarations

- Syntax of a declaration

  <!keyword param1 param2 … param *n*>

- Keywords used:

  1. ELEMENT – name of element and its sub-elements (if any)

  2. ATTLIST – attribute names for a particular element with possible values of attributes

27



# ELEMENT Declaration

- Two forms:

  <!ELEMENT elementName contentCategoryKeyword> (form 1)

  <!ELEMENT elementName (contentModel)> (form 2)

- Five categories of element content:

  1. Empty – no text or sub-elements

  2. Any – any content as long as it is well-formed

  3. Text – character data only

  4. Element – elements only

  5. Mixed – allowed to contain both text and elements

28

# EMPTY Content

- One of two types of form 1 element declaration
- Syntax

  <!ELEMENT elementName EMPTY>
- We know examples of these from HTML, e.g., <BR>
- Use empty elements where no data required for storage but *attributes* store relevant values. This is useful when we want to restrict stored values by using types.
- Thus, EMPTY element declarations are used with ATTLIST declaration (covered later)

29

29

# ANY Content

- The second of the two types of form 1 element declaration
- Syntax

  <!ELEMENT elementName ANY>
- Can contain within an element absolutely any well-formed XML: text, comments, any elements, PIs and CDATA sections, etc.
- **Not used often as idea is to use DTD to check data stringently**

30

30

# Text and Element

- Text only - the keyword #PCDATA stands for "parsed-character data" and is any well-formed text string.

  <!ELEMENT *element* (#PCDATA)>

- Elements that contain only child elements

  <!ELEMENT *element* (*sub-element-list*)>

  - Sequence list – sub-elements found within parent in order shown
  - Choice list – sub-elements are mutually exclusive in parent element; only one of them can be found in its parent

- Example:

<!ELEMENT el1 (el2, el3)> el1 has two children, el2 then el3

<!ELEMENT el2 (#PCDATA)> el2 has text data

<!ELEMENT el3 (el4 | el5)> el3 has one child; an el4 or an el5

31



# Mixed

- Mixed content contains any combination of text characters and elements, e.g.

  <!ELEMENT el4 (#PCDATA | el6)*>

This states that an el4 consists of a sequence of zero or more sub-elements, each of which is either a PCDATA string, or an el6 element.

As an example:

```
<el4>
    Here is some initial text before the element, el6.
    <el6>Just an ordinary element after some text</el6>
</el4>
```

32

# Cardinality

- Where no operator is used, it means the sub-element occurs once and must occur in the parent
- Three cardinality operators are used:
  1. **?** optionality: zero or one occurrence
  2. ***** zero or more occurrences
  3. **+** one or more occurrences

33



# Cardinality Example

**<!ELEMENT unicampus (name, faculty+, bank*, staffclub?)>**

Must have 1 Any number >=1 Any number >= 0 0 or 1

means the following XML can occur:

<unicampus>
  <name>Swinburne – Hawthorn</name>
  <faculty>ICT</faculty>
  <faculty>Business and Enterprise</faculty>
  <faculty>Engineering and Industrial Sciences</faculty>
  … <!-- more faculties -->
  <staffclub>Swinburne (Hawthorn) Staff Club</staffclub>
</unicampus>

It is ok that there is no "bank" element, because zero occurrences of "bank" is allowed

34

# Attribute Declaration

- Attribute-list declaration syntax:

```
<!ATTLIST elementName
   attribName1   attribType1   defaultKeyword1   defaultVal1
   attribName2   attribType2   defaultKeyword2   defaultVal2
   …
   attribName n  attribType n  defaultKeyword n  defaultVal n
>
```

- elementName and attribName are required, legal XML names
- defaultKeyword indicates something about the attribute and can take values #REQUIRED, #IMPLIED, #FIXED
- Data is CDATA

35

35

# Attribute Defaults: #REQUIRED

- Attribute must be *always* be specified for every use of element
- HTML example:

  <IMG SRC="file.gif">

  SRC is #REQUIRED in XHTML DTD specification
- XML example:

  **<!ATTLIST Textbook number CDATA #REQUIRED>**

  <Textbook number="1">Birbeck, M., et al, Professional XML, 2nd Edition, Wrox Press Ltd, 2001.</Textbook>
- Attribute *number* of a Textbook element must have a value
- There is no default value provided

36

36

# Attribute Defaults: #IMPLIED

- Attribute is optional (OK – maybe the word "IMPLIED" is not the best choice here, but we have to live with it!)
- HTML example:

  <IMG WIDTH="100">

  WIDTH is #IMPLIED in HTML specification (so need not be given)
- XML example:

  **<!ATTLIST Textbook rating CDATA #IMPLIED>**

  <Textbook number="1" rating="9.0">Birbeck, M., et al, Professional XML, 2nd Edition, Wrox Press Ltd, 2001.</Textbook>
- The Attribute *rating* is optional, so need not be given
- Like #REQUIRED, no default is specified

37

37

# Attribute Defaults: #FIXED

- The attribute is optional but if one is specified, it must match the default (otherwise the XML is not valid wrt the dtd)
- Example:

  **<!ATTLIST Textbook pubdate CDATA #FIXED "2001">**

  <Textbook number="1" pubdate="2001">Birbeck, M., et al, Professional XML, 2nd Edition, Wrox Press Ltd, 2001.</Textbook>
- Here, a default value is specified
- If the attribute is not specified, the validator will supply the default value
- #FIXED not used much

38

38

# Attribute Defaults: Default Values

- Attribute is optional, but if no value specified for attribute, it is given the default
- Example:

  **<!ATTLIST Book recommendation CDATA "no">**

  But could have the following code:

  <Book recommendation="yes">Verygood, J., "XML", Verygood Publishers, 2001</Book>

39

39

# Attribute Types

- CDATA (already addressed)
- Enumerated values
  - A list of values, one of which is used by the attribute
  - Each value has to be *nmtoken* type (all characters from *NameChar*: letter, digit, '_', '-', '.', ':')
- Example:

  **<!ATTLIST TeachingMethod hours (1 | 2) #REQUIRED>**

  <TeachingMethod hours="2">Lecture</TeachingMethod>
- Note : In the DTD, the options are specified without quotes. In the XML, they must be quoted.

40

40

# Attribute Types: ID

- Used for identification of an element within the document
- Use a unique legal XML name
- A *legal name* starts with letter or underscore or ':', followed by nameChars. Thus "1114445" is illegal ID value.
- Attribute must be declared as #REQUIRED or #IMPLIED
- Example:

```
<!ATTLIST Student sid ID #REQUIRED>
<Students>
   <Student sid="s1114445"><Surname>Edward</Surname></Student>
   <Student sid="s1234321"><Surname>Howard</Surname></Student>
</Students>
```

41



# Attribute Types: IDREF

- Used for allowing some element to link to an element having an ID attribute
- Many such elements are allowed to link to the same element
- Example:

```
<!ATTLIST Top_student sid IDREF #REQUIRED>
<Subject>
   <Name> Web Development</Name>
   <Top_student sid="s1234321"/>
</Subject>
…
<Subject>
   <Name>Advanced Web Technologies</Name>
   <Top_student sid="s1234321"/>
</Subject>
```

42

# Attribute Types: IDREFS

- It is illegal to have:
  <Good_students sid="s1234321" sid="s1114445"/>
- XML allows this capability thru IDREFS
- Value of IDREFS attribute is list of whitespace-delimited ID values
- Example:
  **<!ATTLIST Good_students sids IDREFS #REQUIRED>**
  <Good_students sids="s1234321 s1114445"/>

43

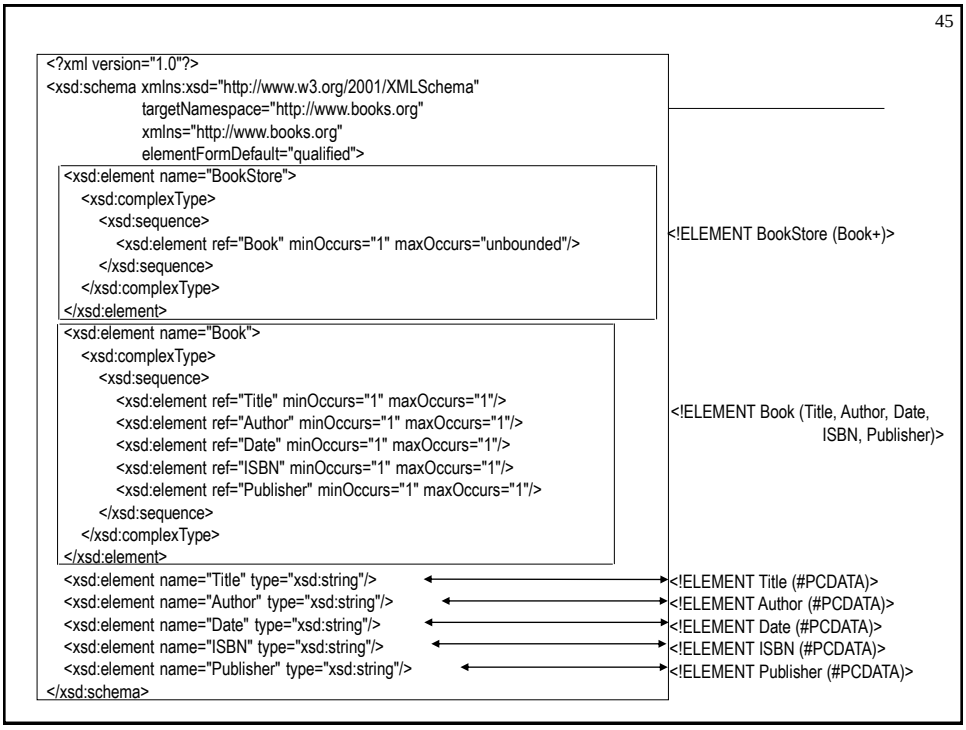


# From DTD to XML Schema

**BookStore.dtd**

```
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date, ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
```

44


<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.books.org"
xmlns="http://www.books.org"
elementFormDefault="qualified">
<xsd:element name="BookStore">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="Book" minOccurs="1" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Book">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="Title" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Author" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Date" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="ISBN" minOccurs="1" maxOccurs="1"/>
<xsd:element ref="Publisher" minOccurs="1" maxOccurs="1"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="Author" type="xsd:string"/>
<xsd:element name="Date" type="xsd:string"/>
<xsd:element name="ISBN" type="xsd:string"/>
<xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
<!ELEMENT BookStore (Book+)>
<!ELEMENT Book (Title, Author, Date,
ISBN, Publisher)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>

45