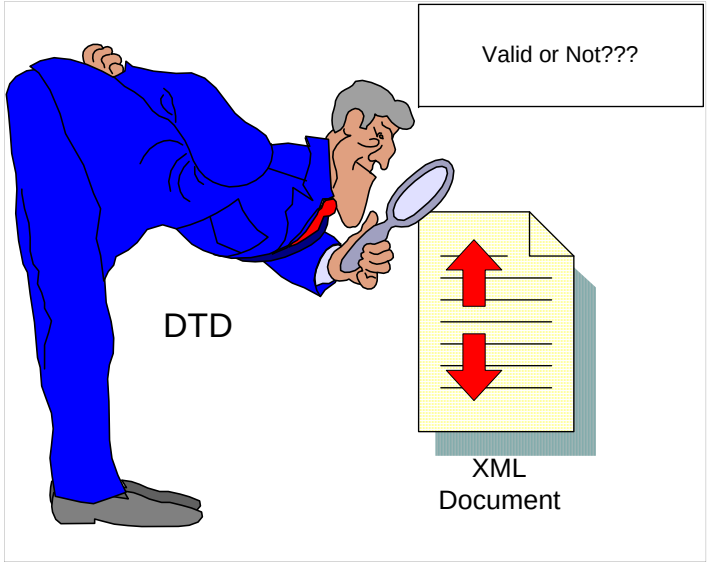
Web Databases and Applications

DTD and XML Schema

Data Type Definition (DTD)

- DTD is used to define and document the markup language.
- XML document is compared against a DTD to check for its validity
- In other words, DTD is like a dictionary that defines how a valid XML document should be formatted and the valid elements, tags, attributes.
  - E.g. A book DTD dictates that it should contain only one TITLE, one ISBN, and one or more AUTHORS



Parsing XML Documents

- Parsers
  - Validating
    - Able to read DTD
    - Determine whether XML document conforms to DTD
      - Valid document conforms to DTD
        - Document is then well formed, by definition
        - Documents can be well formed, but not valid
  - Nonvalidating
    - Able to read DTD
    - Cannot check document against DTD for conformity

Document Type Declaration

- Document Type Declaration
  - Placed in XML document's prolog
  - Begins with **`<!DOCTYPE`**
  - Ends with **`>`**
  - Can point to
    - External subsets
      - Declarations outside document
      - Exist in different file
        - typically ending with **`.dtd`** extension
    - Internal subsets
      - Declarations inside document
      - Visible only within document in which it resides

Why use DTD

- With a DTD, each of your XML files can carry a description of its own format.
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data.
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid.
- You can also use a DTD to verify your own data.

Element Type Declarations

- Element type declarations
  - Declare elements in XML documents
  - Begin with **<!ELEMENT**
  - End with **>**

```
<!ELEMENT myElement ( #PCDATA )>
```

    - **myElement** is *generic identifier*
    - Parentheses specify element's content (*content specification*)
    - Keyword **PCDATA**
      - Element must contain parsable character data
        - All text treated as markup

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 6.1: intro.xml      -->
4  <!-- Using an external subset -->
5
6  <!DOCTYPE myMessage SYSTEM "intro.dtd">
7
8  <myMessage>
9     <message>Welcome to XML!</message>
10 </myMessage>
```

XML document declaring its associated DTD.

DOCTYPE **starts document type declaration**
Document type declaration is named myMessage
Keyword SYSTEM **specifies external subset**
intro.dtd **is DTD**

intro.dtd

<!ELEMENT MyMessage (message) >

<!ELEMENT message (#PCDATA) >

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 6.3 : intro-invalid.xml      -->
4  <!-- Simple introduction to XML markup -->
5
6  <!DOCTYPE myMessage SYSTEM "intro.dtd">
7
8  <!-- Root element missing child element message -->
9  <myMessage>
10 </myMessage>
```

Element **myMessage**'s structure does not adhere to that specified in **intro.dtd**

Non-valid XML document

Sequences, Pipe Characters and Occurrence Indicators

- Sequences
  - Specify order in which elements occur
  - Comma (**,**) used as delimiter

```
<!ELEMENT classroom ( teacher, student
)>
```

- Pipe characters (**|**)
  - Specify choices

```
<!ELEMENT dessert ( iceCream | pastry )>
```

Sequences, Pipe Characters and Occurrence Indicators

- Occurrence indicators
  - Specify element's frequency
  - Plus sign (**+**) indicates one or more occurrences

```
<!ELEMENT album ( song+ )>
```

  - Asterisk (*****) indicates optional element

```
<!ELEMENT library ( book* )>
```

  - Question mark (**?**) indicates element can occur only once

```
<!ELEMENT seat ( person? )>
```

EMPTY, Mixed Content and ANY

- Content specification types
  - **EMPTY**
    - Elements do not contain character data
    - Elements do not contain child elements

```
<!ELEMENT oven EMPTY>
```

    - Markup for **oven** element

```
<oven/>
```

EMPTY, Mixed Content and ANY

- Content specification types
  - Mixed content
    - Combination of elements and **PCDATA**

```
<!ELEMENT myMessage ( #PCDATA | message )*>
```

    - Markup for **myMessage**

```
<myMessage>Here is some text, some
   <message>other text</message>and
   <message>even more text</message>
</myMessage>
```

```
1  <?xml version = "1.0" standalone = "yes"?>
2
3  <!-- Fig. 6.5 : mixed.xml        -->
4  <!-- Mixed content type elements -->
5
6  <!DOCTYPE format [
7     <!ELEMENT format ( #PCDATA | bold | italic )*>
8     <!ELEMENT bold ( #PCDATA )>
9     <!ELEMENT italic ( #PCDATA )>
10 ]>
11
12 <format>
13    This is a simple formatted sentence.
14    <bold>I have tried bold.</bold>
15    <italic>I have tried italic.</italic>
16    Now what?
17 </format>
```

Example of a mixed-content element.

EMPTY, Mixed Content and ANY

- Content specification types
  - **ANY**
    - Can contain any content
      - **PCDATA**, elements or combination
      - Can also be empty elements
    - Commonly used in early DTD-development stages
      - Replace with specific content as DTD evolves

Attribute Declarations

- Attribute declaration
  - Specifies element's attribute list
  - Uses **`ATTLIST`** attribute list declaration

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 6.7: intro2.xml -->
4  <!-- Declaring attributes -->
5
6  <!DOCTYPE myMessage [
7     <!ELEMENT myMessage ( message )>
8     <!ELEMENT message ( #PCDATA )>
9     <!ATTLIST message id CDATA #REQUIRED>
10 ]>
11
12 <myMessage>
13
14    <message id = "445">
15       Welcome to XML!
16    </message>
17
18 </myMessage>
```

Attribute Types

- Attribute types
  - Strings (**`CDATA`**)
    - No constraints on attribute values
      - Except for disallowing **`<`**, **`>`**, **`&`**, **`'`** and **`"`** characters
  - Tokenized attributes
    - Constraints on permissible characters for attribute values
  - Enumerated attributes
    - Most restrictive
    - Take only one value listed in attribute declaration

Attribute Defaults

- (`#REQUIRED`, `#IMPLIED`, `#FIXED`)
- Attribute defaults
  - Specify attribute's default value
  - <!ATTLIST square width CDATA "0">
  - **`#IMPLIED`**
    - Use (application's) default value if attribute value not specified
  - **`#REQUIRED`**
    - Attribute must appear in element
    - Document is not valid if attribute is missing
  - **`#FIXED`**
    - Attribute value is constant
    - <!ATTLIST square width CDATA #FIXED "0">
    - Attribute value cannot differ in XML document

Tokenized Attribute Type

- (`ID`, `IDREF`, `ENTITY`, `NMTOKEN`) Tokenized attribute types
  - Restrict attribute values
  - **`ID`**
    - Uniquely identifies an element
  - **`IDREF`**
    - Points to elements with **`ID`** attribute

```
1  <?xml version = "1.0"?>
2
3  <!-- Fig. 6.8: IDExample.xml                          -->
4  <!-- Example for ID and IDREF values of attributes -->
5
6  <!DOCTYPE bookstore [
7     <!ELEMENT bookstore ( shipping+, book+ )>
8     <!ELEMENT shipping ( duration )>
9     <!ATTLIST shipping shipID ID #REQUIRED>
10    <!ELEMENT book ( #PCDATA )>
11    <!ATTLIST book shippedBy IDREF #IMPLIED>
12    <!ELEMENT duration ( #PCDATA )>
13 ]>
14
15 <bookstore>
16    <shipping shipID = "s1">
17       <duration>2 to 4 days</duration>
18    </shipping>
19
```

```
20    <shipping shipID = "s2">
21       <duration>1 day</duration>
22    </shipping>
23
24    <book shippedBy = "s2">
25       Java How to Program 3rd edition.
26    </book>
27
28    <book shippedBy = "s2">
29       C How to Program 3rd edition.
30    </book>
31
32    <book shippedBy = "s1">
33       C++ How to Program 3rd edition.
34    </book>
35 </bookstore>
```

Using ID and IDREF Attributes

```
<!DOCTYPE family [
  <!ELEMENT family   (person)*>
  <!ELEMENT person  (name)>
  <!ELEMENT name    (#PCDATA)>
  <!ATTLIST  person
       id ID #REQUIRED
       mother IDREF #IMPLIED
       father IDREF #IMPLIED
       children IDREFS #IMPLIED>
]>
```

Some Conforming Data

```
<family>
    <person  id="lisa"  mother="marge" father="homer">
      <name> Lisa Simpson </name>
    </person>
    <person  id="bart"  mother="marge" father="homer">
      <name> Bart Simpson </name>
    </person>
    <person id="marge" children="bart lisa">
        <name> Marge Simpson </name>
    </person>
    <person id="homer" children="bart lisa">
        <name> Homer Simpson </name>
    </person>
</family>
```

Limitations of ID References

- The attributes mother and father are references to IDs of other elements.
- However, those are not necessarily person elements!
- The mother attribute is not necessarily a reference to a female person.

An Alternative Specification

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE family [
    <!ELEMENT family (person)*>
    <!ELEMENT person (name, mother?, father?,
                children?)>
     <!ATTLIST person id ID #REQUIRED>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT mother EMPTY>
    <!ATTLIST mother idref IDREF #REQUIRED>
    <!ELEMENT father EMPTY>
      <!ATTLIST father idref IDREF #REQUIRED>
    <!ELEMENT children EMPTY>
      <!ATTLIST children idrefs IDREFS #REQUIRED>
]>
```

Empty sub-elements instead of attributes

The Revised Data

```
<family>

<person id="marge">
 <name>Marge Simpson</name>
 <children
     idrefs="bart lisa"/>
 </person>

 <person id="homer">
 <name>Homer Simpson</name>
 <children
   idrefs="bart lisa"/>
</person>
```

```
<person id="bart">
 <name>Bart Simpson</name>
 <mother idref="marge"/>
 <father idref="homer"/>
</person>

<person id="lisa">
 <name>Lisa Simpson</name>
 <mother idref="marge"/>
 <father idref="homer"/>
</person>

</family>
```

Consistency of ID and IDREF Attribute Values

- If an attribute is declared as ID
  - The associated value must be distinct, i.e., different elements (in the given document) must have different values for the ID attribute.
  - Even if the two elements have different element names
- If an attribute is declared as IDREF
  - The associated value must exist as the value of some ID attribute (no dangling “pointers”)
- Similarly for all the values of an IDREFS attribute
- ID, IDREF and IDREFS attributes are *not* typed

Tokenized Attribute Type

- **`ENTITY`** tokenized attribute type
  - Entities are variables used to define common text.
  - Entity declaration

    ```
    <!ENTITY digits “0123456789”>
    ```
  - Entity may be used as follows:

    ```
    <useAnEntity>&digits;</useAnEntity>
    ```
  - Entity reference **`&digits;`** replaced by its value

    ```
    <useAnEntity>0123456789</useAnEntity>
    ```

External Entity Declaration

- DTD
  - <!ENTITY entity-name SYSTEM "URI/URL">
- Example:
  - <!ENTITY writer SYSTEM "http://www.w3schools.com/dtd/entities.dtd">
  - <!ENTITY copyright SYSTEM "http://www.w3schools.com/dtd/entities.dtd">

- XML example:
  - <author>&writer;&copyright;</author>

Tokenized Attribute Type

- **`NMTOKEN`** tokenized attribute type
  - "Name token"
  - Value consists of letters, digits, periods, underscores, hyphens and colon characters

Enumerated Attribute Types

- Enumerated attribute types
  - Declare list of possible values for attribute

```
<!ATTLIST person gender ( M | F ) “F”>
```

    - Attribute **gender** can have either value **M** or **F**
    - **F** is default value

DTDs

- Do not provide any mechanism to restrict what a data element can contain
  - E.g. specific that a particular date item must be numeric
- Syntax of DTD comes from old SGML

So what's wrong?

- Different syntax than regular XML
- No data types and complex to extend
  - Length, occurrences
  - 10 data types
- Namespaces not well support
- No inheritance (no *kind-of* information, only *part-of* information)
- Insufficient checking: too much work is left to application

XML Schema

- XML Schema was originally proposed by Microsoft, but is now a W3C recommendation
- Can be used to specify the schema of a particular class of documents
- Uses XML syntax
  - Not required to learn a completely new syntax just to describe your grammar
  - You still need to learn how to declare elements and attributes using XML Schema.
- www.w3.org/XML/Schema

What is an XML Schema?

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

An Example

Note.dtd

```
<!ELEMENT note (to, from, heading, body)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT heading (#PCDATA)>
 <!ELEMENT body (#PCDATA)>
```

Note.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://www.w3schools.com"
           xmlns="http://www.w3schools.com" elementFormDefault="qualified">
 <xs:element name="note">
        <xs:complexType>
              <xs:sequence>
               <xs:element name="to" type="xs:string"/>
              <xs:element name="from" type="xs:string"/>
              <xs:element name="heading" type="xs:string"/>
              <xs:element name="body" type="xs:string"/>
               </xs:sequence>
        </xs:complexType>
 </xs:element>
</xs:schema>
```

XML Sample with DTD Reference

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM
        "http://www.w3schools.com/dtd/note.dtd">
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

XML Sample with XML Schema Reference

```
<?xml version="1.0"?>
        <note xmlns="http://www.w3schools.com"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3schools.com note.xsd">

        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

XML Scheme Template

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

A Simple Element

- XML Schema has a lot of built-in data types.
- The most common types are:
  - xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time

```
<xs:element name="lastname" type="xs:string“ />
<xs:element name="age" type="xs:integer"/>
<xs:element name="dateborn" type="xs:date"/>
<xs:element name="color" type="xs:string" default="red"/>
<xs:element name="color" type="xs:string" fixed="red"/>

<lastname>Refsnes</lastname>
 <age>36</age>
<dateborn>1970-03-27</dateborn>
```

Types

- The XML Schema supports two basic types:
  - Complex, and
  - Simple
- Complex types allow elements and attributes in their contents
- Simple types can not have elements or attributes in their contents
- A simple type can be either:
  - Built in, or
  - Derived

Restrictions on Values

```
<xs:element name="age">
    <xs:simpleType>
    <xs:restriction base="xs:integer">
          <xs:minInclusive value="0"/>
          <xs:maxInclusive value="120"/>
    </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restrictions on Values

To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

```
<xs:element name="car">
    <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:enumeration value="Audi"/>
        <xs:enumeration value="Golf"/>
        <xs:enumeration value="BMW"/>
    </xs:restriction>
    </xs:simpleType>
</xs:element>
```

```
<xs:element name="car" type="carType"/>
<xs:simpleType name="carType">
<xs:restriction base="xs:string">
<xs:enumeration value="Audi"/>
<xs:enumeration value="Golf"/>
<xs:enumeration value="BMW"/>
 </xs:restriction>
</xs:simpleType>
```

Restrictions on a Series of Values

```
<xs:element name="letter">
    <xs:simpleType>
    <xs:restriction base="xs:string">
        <xs:pattern value="[a-z]"/>
    </xs:restriction>
    </xs:simpleType>
</xs:element>
```

```
<xs:element name="initials">
        <xs:simpleType>
        <xs:restriction base="xs:string">
        <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        <xs:pattern value="[a-zA-Z0-9]{8}"/>
        <xs:length value="8"/>
        </xs:restriction>
        </xs:simpleType>
</xs:element>
```

Complex Elements

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text

```
<xs:element name="employee">
        <xs:complexType>
                <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                </xs:sequence>
        </xs:complexType>
 </xs:element>
```

Complex Elements

- Serveral elements reference the same type

```
<xs:element name="employee" type="personinfo"/>
 <xs:element name="student" type="personinfo"/>
 <xs:element name="member" type="personinfo"/>


<xs:complexType name="personinfo">
            <xs:sequence>
                        <xs:element name="firstname" type="xs:string"/>
                        <xs:element name="lastname" type="xs:string"/>
            </xs:sequence>
 </xs:complexType>
```

Complex Elements

- Re-use previous definitions

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
        <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
        </xs:sequence>
 </xs:complexType>

<xs:complexType name="fullpersoninfo">
        <xs:complexContent>
         <xs:extension base="personinfo">
                <xs:sequence>
                        <xs:element name="address" type="xs:string"/>
                         <xs:element name="city" type="xs:string"/>
                         <xs:element name="country" type="xs:string"/>
                </xs:sequence>
         </xs:extension>
         </xs:complexContent>
</xs:complexType>
```

Complex Elements

- Only contains simple contents: text, attributes

```
<xs:element name="shoesize">
        <xs:complexType>
                <xs:simpleContent>
                <xs:extension base="xs:integer">
                 <xs:attribute name="country" type="xs:string" />
                </xs:extension>
                </xs:simpleContent>
         </xs:complexType>
 </xs:element>
```

Complex Elements

- Contains mixed contents

```
<xs:element name="letter">
        <xs:complexType mixed="true">
        <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="orderid" type="xs:positiveInteger"/>
        <xs:element name="shipdate" type="xs:date"/>
        </xs:sequence>
        </xs:complexType>
</xs:element>
```

MinOccurs and MaxOccurs

- The **maxOccurs** attribute
  - a constraint rule, specifying the maximum number of times that a sub-element may appear.
  - Valid values for **maxOccurs** include integers and "*", which indicates that an unrestricted number of elements may appear.
  - The default value for **maxOccurs** is "1"; however, when content="mixed", the default value is "*".

MinOccurs and MaxOccurs

- **minOccurs** can specify a minimum number of times a sub-element may appear with.
  - E.g. to make a sub-element optional, set **minOccurs** to "0".
  - The default value for minOccurs is 1.
- These attributes can be used for both **element** and **group** declarations.

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="full_name" type="xs:string"/>
 <xs:element name="child_name" type="xs:string" maxOccurs="10" minOccurs="0"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
```

Sub-Element Order

- The element indicates the ordering of the sub-elements of an element
  - "sequence" value: sub-elements must appear in the order listed in the schema.
  - “choice" value : only one sub-element can be used from a list of sub-elements
  - “all" value: sub-elements may appear in any order, and occur only once

Indicating Order

```
<xs:element name="person">
 <xs:complexType>
 <xs:choice>
 <xs:element name="employee" type="employee"/>
 <xs:element name="member" type="member"/>
 </xs:choice>
 </xs:complexType>
 </xs:element>
```

```
<xs:element name="person">
 <xs:complexType>
 <xs:all>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 </xs:all>
 </xs:complexType>
 </xs:element>
```

Group

- Enables you to specify constraints on a specific set of sub-elements
- Accepts the **order**, **minOccurs**, and **maxOccurs** attributes

Group

- Re-using a group definition

```
<xs:group name="persongroup">
        <xs:sequence>
                <xs:element name="firstname" type="xs:string"/>
                <xs:element name="lastname" type="xs:string"/>
                <xs:element name="birthday" type="xs:date"/>
        </xs:sequence>
 </xs:group>

<xs:element name="person" type="personinfo"/>

<xs:complexType name="personinfo">
        <xs:sequence>
                <xs:group ref="persongroup"/>
                <xs:element name="country" type="xs:string"/>
        </xs:sequence>
 </xs:complexType>
```

Global Elements

- Global elements are declared as the children of the `schema` element
- A global element may appear as a top-level (root) element in an XML document
- A global element can be referenced using `ref` attribute, but it can not contain references
- A global element can not contain structural constraints (but elements referencing it can)

`ref` Attribute

- Instead of declaring a new element it is possible to use an existing one by referencing it using the ref attribute
- The referenced element has to be a global one
- The consequence is that an element called comment may appear in an instance of the ClassType

Attributes

- Attributes are more limited
- Attributes cannot contain sub-elements nor to appear in any particular order;
- Allow to specify whether an attribute is required or optional, but an attribute may only appear once per element.
- Attributes may limit their legal values to a small set of strings, and may indicate a value to be inferred if the attribute is omitted from an element.
- Different element types may have attributes with the same name.

Attributes

- Builtin-types
  - xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time
- Support includes primitive data types common to programming languages as well as the special attribute types included in the XML Language Specification
  - E.g. ID, IDREF, and NMTOKEN
  - msdn.microsoft.com/xml/reference/schema/datatypes.asp

```
<xs:attribute name="xxx" type="yyy"/>
<xs:attribute name="lang" type="xs:string"/>
<xs:attribute name="lang" type="xs:string" default="EN"/>
<xs:attribute name="lang" type="xs:string" fixed="EN"/>
<xs:attribute name="lang" type="xs:string" use="required"/>
```

Complex Empty Element

- Only attribute content

```
<xs:element name="product">
        <xs:complexType>
                <xs:complexContent>
                <xs:restriction base="xs:integer“>
                <xs:attribute name="prodid" type="xs:positiveInteger"/>
                 </xs:restriction>
                </xs:complexContent>
        </xs:complexType>
 </xs:element>
```

Working Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:noNamespaceSchemaLocation="shiporder.xsd">

 <orderperson>John Smith</orderperson>
 <shipto>
     <name>Ola Nordmann</name>          <address>Langgt 23</address>
     <city>4000 Stavanger</city>        <country>Norway</country>
 </shipto>
<item>
     <title>Empire Burlesque</title>    <note>Special Edition</note>
     <quantity>1</quantity>             <price>10.90</price>
</item>
<item>
     <title>Hide your heart</title>      <quantity>1</quantity>
     <price>9.90</price>
 </item>
</shiporder>
```

Shiporder.xsd - First

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
    <xs:complexType>
        <xs:sequence>
        <xs:element name="orderperson" type="xs:string"/>
        <xs:element name="shipto">
        <xs:complexType>
            <xs:sequence>
             <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
             </xs:sequence>
        </xs:complexType>
        </xs:element>
```

Shiporder.xsd - First

```
            <xs:element name="item" maxOccurs="unbounded">
                    <xs:complexType>
                            <xs:sequence>
                            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
             <xs:element name="quantity" type="xs:positiveInteger"/>
                            <xs:element name="price" type="xs:decimal"/>
                            </xs:sequence>
                    </xs:complexType>
             </xs:element>
            </xs:sequence>
            <xs:attribute name="orderid" type="xs:string" use="required"/>
      </xs:complexType>
 </xs:element>
</xs:schema>
```

Shiporder.xsd - Second

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
 <xs:element name="address" type="xs:string"/>
 <xs:element name="city" type="xs:string"/>
 <xs:element name="country" type="xs:string"/>
 <xs:element name="title" type="xs:string"/>
 <xs:element name="note" type="xs:string"/>
 <xs:element name="quantity" type="xs:positiveInteger"/>
 <xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>
```

Shiporder.xsd - Second

```
<!-- definition of complex elements -->
<xs:element name="shipto">
    <xs:complexType>        <xs:sequence>
            <xs:element ref="name"/>
             <xs:element ref="address"/>
            <xs:element ref="city"/>
             <xs:element ref="country"/>
    </xs:sequence>            </xs:complexType>
 </xs:element>
 <xs:element name="item">
    <xs:complexType>        <xs:sequence>
            <xs:element ref="title"/>
             <xs:element ref="note" minOccurs="0"/>
            <xs:element ref="quantity"/>
            <xs:element ref="price"/>
    </xs:sequence>            </xs:complexType>
 </xs:element>
```

Shiporder.xsd - Second

```
<xs:element name="shiporder">
      <xs:complexType>
            <xs:sequence>
                  <xs:element ref="orderperson"/>
                  <xs:element ref="shipto"/>
                   <xs:element ref="item" maxOccurs="unbounded"/>
            </xs:sequence>
           <xs:attribute ref="orderid" use="required"/>
     </xs:complexType>
 </xs:element>
</xs:schema>
```