CSE 544: Lecture 5

XML

4/15/2002

Review: Views

- What is a view in SQL ?
- What is the difference between:
  - *Virtual* view
  - *Materialized* view
- How are views used ?
- What is an *updateable* view ?

Outline

XML

- Motivation:
  - *content* v.s. *presentation*
  - Data applications on the internet
- Syntax
  - lots of details, will skip some slides
- DTDs
- XML Schema

Will skip many irrelevant details. Best source: www.w3.org

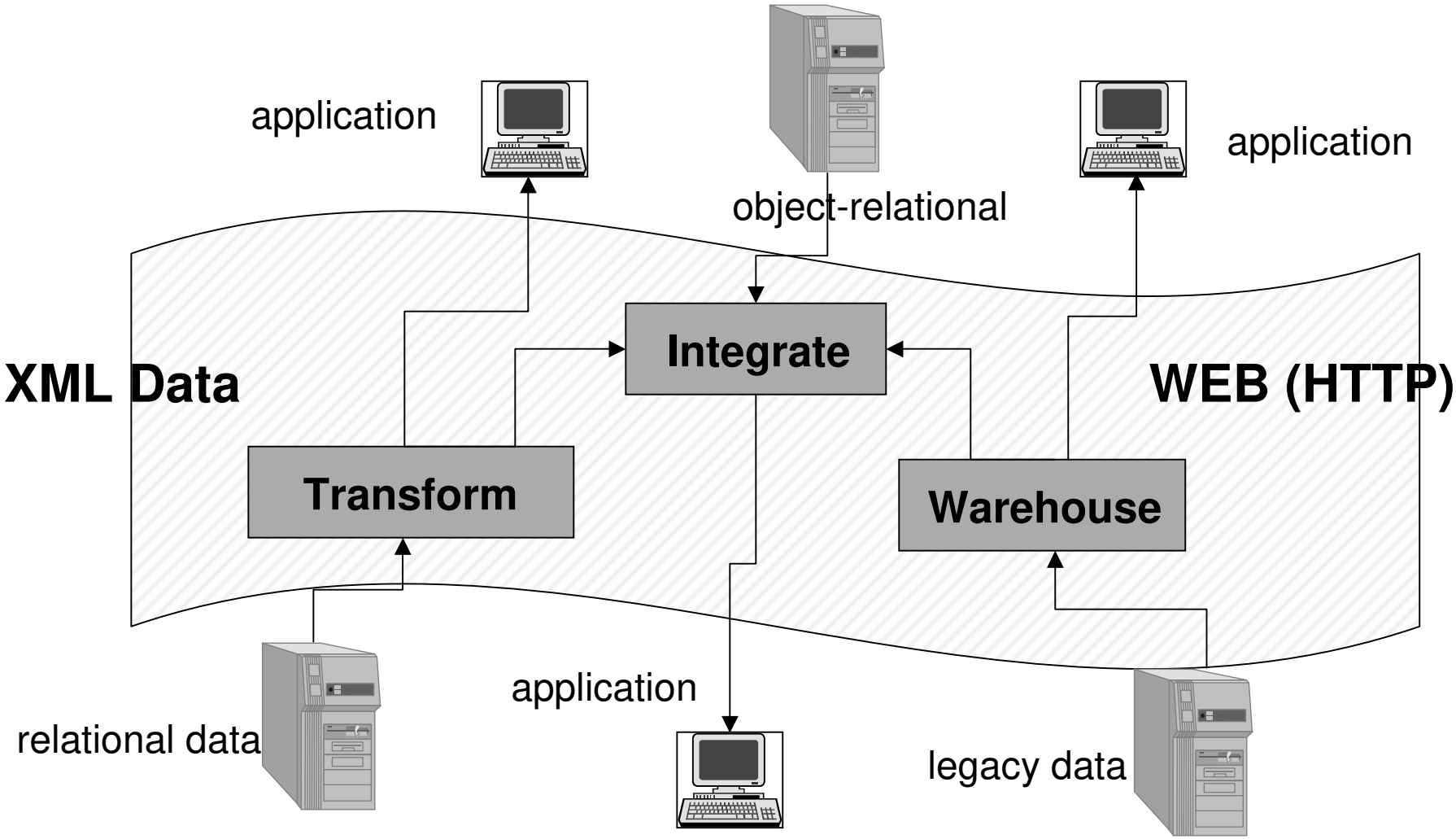
XML

- eXtensible Markup Language
- XML 1.0 – a recommendation from W3C, 1998
- Roots: SGML (a very nasty language).
- After the roots: a format for sharing *data*

Why XML is of Interest to Us

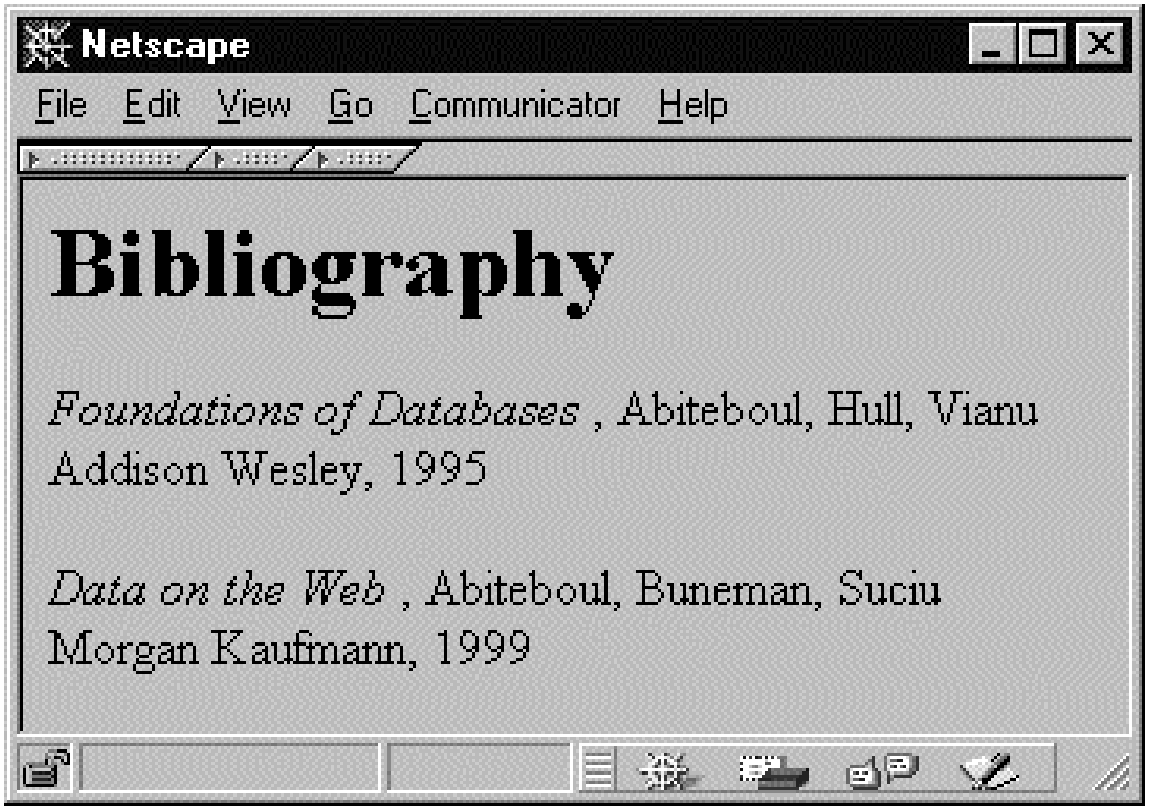
- XML is just syntax for data
  - Note: we have no syntax for relational data
  - But XML is not relational: *semistructured*
- This is exciting because:
  - Can translate *any* legacy data to XML
  - Can ship XML over the Web (HTTP)
  - Can input XML into any application
  - Thus: data sharing and exchange on the Web

XML Data Sharing and Exchange



Specific data management tasks

From HTML to XML



HTML describes the presentation

HTML

```
<h1> Bibliography </h1>
<p> <i> Foundations of Databases </i>
        Abiteboul, Hull, Vianu
        <br> Addison Wesley, 1995
<p> <i> Data on the Web </i>
        Abiteoul, Buneman, Suciu
        <br> Morgan Kaufmann, 1999
```

XML

```
<bibliography>
      <book>    <title> Foundations… </title>
                <author> Abiteboul </author>
                <author> Hull </author>
                <author> Vianu </author>
                <publisher> Addison Wesley </publisher>
                <year> 1995 </year>
      </book>
      …
</bibliography>
```

XML describes the content

XML Terminology

- tags: book, title, author, …
- start tag: <book>, end tag: </book>
- elements: <book>…<book>,<author>…</author>
- elements are nested
- empty element: <red></red> abbrv. <red/>
- an XML document: single *root element*

well formed XML document: if it has matching tags

More XML: Attributes

```
<book price = “55” currency = “USD”>
   <title> Foundations of Databases </title>
   <author> Abiteboul </author>
    …
   <year> 1995 </year>
</book>
```

attributes are alternative ways to represent data

More XML: Oids and References

```
<person id="o555">  <name> Jane </name> </person>

<person id="o456">  <name> Mary </name>
                        <children idref="o123 o555"/>
</person>

<person id="o123" mother="o456"><name>John</name>
</person>
```

oids and references in XML are just syntax

More XML: CDATA Section

- Syntax: <![CDATA[.....any text here...]]>

- Example:

```
<example>
   <![CDATA[ some text here </notAtag> <>]]>
</example>
```

More XML: Entity References

- Syntax: &entityname;
- Example:
  <element> this is less than < </element>
- Some entities:

<	<
>	>
&	&
'	‘
"	“
&	Unicode char

More XML: Processing Instructions

- Syntax: <?target argument?>
- Example:

```
•<product> <name> Alarm Clock </name>
                <?ringBell 20?>
                <price> 19.99 </price>
</product>
```

- What do they mean ?

More XML: Comments

- Syntax <!-- .... Comment text... -->
- Yes, they are part of the data model !!!

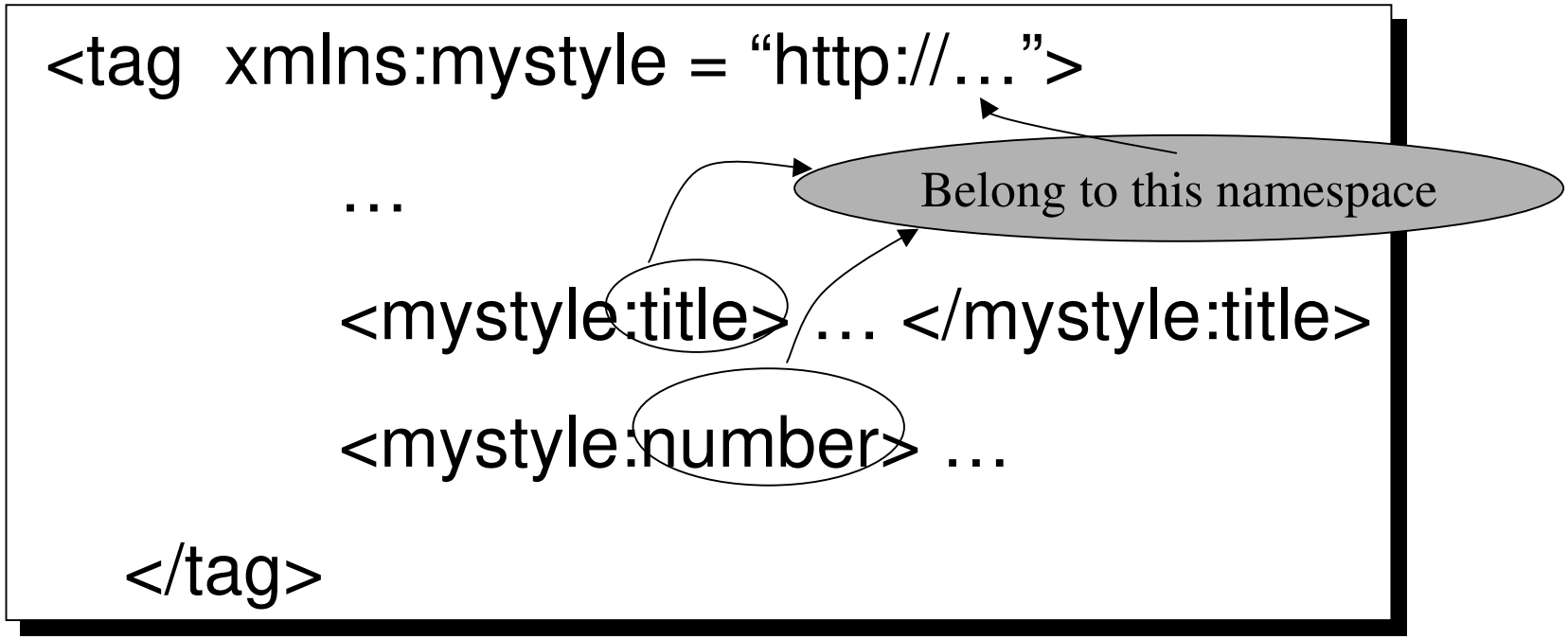
XML Namespaces

- http://www.w3.org/TR/REC-xml-names (1/99)
- name ::= [prefix:]localpart

```
<book xmlns:isbn="www.isbn-org.org/def">
    <title> … </title>
    <number> 15 </number>
    <isbn:number> …. </isbn:number>
</book>
```

XML Namespaces

- syntactic: <number> , <isbn:number>
- semantic: provide URL for schema

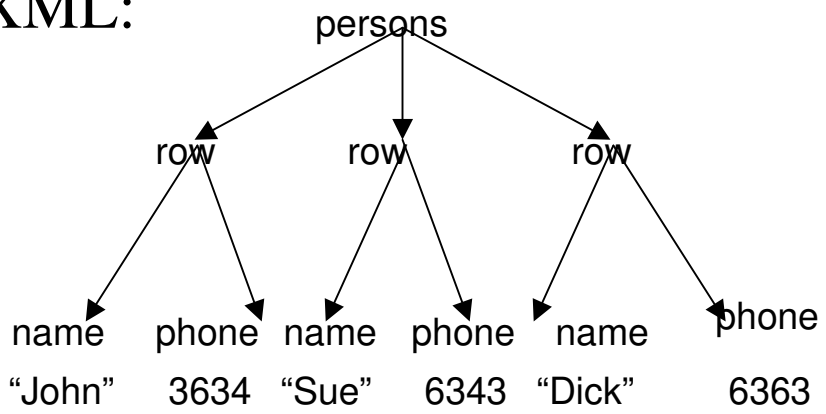


XML for Representing Data

persons

name	phone
John	3634
Sue	6343
Dick	6363

XML:




```
<persons>
     <row> <name>John</name>
           <phone> 3634</phone></row>
     <row> <name>Sue</name>
           <phone> 6343</phone>
     <row> <name>Dick</name>
           <phone> 6363</phone></row>
</persons>
```

XML vs Data Models

- XML is self-describing
- Schema elements become part of the data
  - Reational schema: persons(name,phone)
  - In XML <persons>, <name>, <phone> are part of the data, and are repeated many times
- Consequence: XML is much more flexible
- XML = semistructured data

Semi-structured Data Explained

- Missing attributes:

```
<person>   <name> John</name>
           <phone>1234</phone>
 </person>

<person>  <name>Joe</name>
</person>
```

← no phone !

- Could represent in a table with nulls

name	phone
John	1234
Joe	-

Semi-structured Data Explained

- Repeated attributes

```
<person> <name> Mary</name>
         <phone>2345</phone>
         <phone>3456</phone>
</person>
```

← two phones !

- Impossible in tables:

name	phone	
Mary	2345	3456

???

Semistructured Data Explained

- Attributes with different types in different objects

```
<person> <name>  <first> John </first>
                 <last> Smith </last>
         </name>
         <phone>1234</phone>
</person>
```

← structured name !

- Nested collections (no 1NF)
- Heterogeneous collections:
  - <db> contains both <book>s and <publisher>s

XML Data Models

Several competing models:

- Document Object Model (DOM):
  - http://www.w3.org/TR/2001/WD-DOM-Level-3-CMLS-20010209/ (2/2001)
  - class hierarchy (**node**, **element**, **attribute**,…)
  - objects have behavior
  - defines API to inspect/modify the document
- XPath data model
- XML Query data model
- Infoset
  - PSV (post schema validation)

XML Data v.s. E/R, ODL, Relational

- Q: is XML better or worse ?

- A: serves different purposes
  - E/R, ODL, Relational models:
    - For centralized processing, when we control the data
  - XML:
    - Data sharing between different systems
    - we do not have control over the entire data
    - E.g. on the Web
- Do NOT use XML to model your data ! Use E/R, ODL, or relational instead. Use it to *exchange* data instead.

Document Type Definitions DTD

- part of the original XML specification
- an XML document may have a DTD
- terminology for XML:
  - well-formed: if tags are correctly closed
  - valid: if it has a DTD and conforms to it
- validation is useful in data exchange

Very Simple DTD

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office, phone?)>
  <!ELEMENT ssn        (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
   <!ELEMENT description    (#PCDATA)>
]>
```

Very Simple DTD

Example of valid XML document:

```
<company>
   <person>  <ssn> 123456789 </ssn>
             <name> John </name>
             <office> B432 </office>
             <phone> 1234 </phone>
   </person>
   <person>  <ssn> 987654321 </ssn>
             <name> Jim </name>
             <office> B123 </office>
   </person>
   <product> ...  </product>
   ...
</company>
```

Content Model

- Element *content*: what we can put in an element (aka *content model*)
- Content model:
  - Complex = a regular expression over other elements
  - Text-only = #PCDATA
  - Empty = EMPTY
  - Any = ANY
  - Mixed content = (#PCDATA | A | B | C)*
    - (i.e. very restrictied)

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIS      person age CDATA  #REQUIRED>
```

```
<person age="25">
      <name> ....</name>
       ...
</person>
```

Attributes in DTDs

```
<!ELEMENT person  (ssn, name, office, phone?)>
<!ATTLIS      person age          CDATA  #REQUIRED
                     id           ID       #REQUIRED
                     manager    IDREF   #REQUIRED
                     manages    IDREFS   #REQUIRED
>
```

```
<person age="25"
        id="p29432"
        manager="p48293"  manages="p34982 p423234">
    <name> ....</name>
     ...
</person>
```

Attributes in DTDs

Types:

- CDATA = string
- ID = key
- IDREF = foreign key
- IDREFS = foreign keys separated by space
- (Monday | Wednesday | Friday) = enumeration
- NMTOKEN = must be a valid XML name
- NMTOKENS = multiple valid XML names
- ENTITY = you don’t want to know this

Attributes in DTDs

Kind:

- #REQUIRED
- #IMPLIED = optional
- value = default value
- value #FIXED = the only value allowed

Using DTDs

- Must include in the XML document
- Either include the entire DTD:
  - <!DOCTYPE rootElement [.....]>
- Or include a reference to it:
  - <!DOCTYPE rootElement SYSTEM "http://www.mydtd.org">
- Or mix the two... (e.g. to override the external definition)

DTDs as Grammars

```
<!DOCTYPE paper [
  <!ELEMENT paper   (section*)>
  <!ELEMENT section ((title,section*) | text)>
  <!ELEMENT title      (#PCDATA)>
  <!ELEMENT text       (#PCDATA)>
]>
```

```
<paper> <section> <text> </text> </section>
          <section> <title> </title> <section> … </section>
                                          <section> … </section>
          </section>
</paper>
```

DTDs as Grammars

- A DTD = a grammar
- A valid XML document = a parse tree for that grammar

DTDs as Schemas

Not so well suited:

- impose unwanted constraints on order

  <!ELEMENT person (name,phone)>

- references cannot be constrained
- can be too vague:

  <!ELEMENT person ((name|phone|email)*)>

From DTDs to Relational Schemas

Shanmugasundaram et al. *Relational databases for querying XML documents"*

Design a relational schema for:

```
<!DOCTYPE company [
  <!ELEMENT company   ((person|product)*)>
  <!ELEMENT person  (ssn, name, office?, phone*)>
  <!ELEMENT ssn       (#PCDATA)>
  <!ELEMENT name     (#PCDATA)>
  <!ELEMENT office     (#PCDATA)>
  <!ELEMENT phone    (#PCDATA)>
  <!ELEMENT product  (pid, name, description?)>
  <!ELEMENT pid    (#PCDATA)>
   <!ELEMENT description    (#PCDATA)>
]>
```

XML Schemas

- http://www.w3.org/TR/xmlschema-1/ 10/2000
- generalizes DTDs
- uses XML syntax
- two documents: structure and datatypes
  - http://www.w3.org/TR/xmlschema-1
  - http://www.w3.org/TR/xmlschema-2
- XML-Schema is very complex
  - often criticized
  - some alternative proposals

XML Schemas

```
<xsd:element name="paper" type="papertype"/>
<xsd:complexType name="papertype">
    <xsd:sequence>
        <xsd:element name="title" type="xsd:string"/>
        <xsd:element name="author" minOccurs="0"/>
        <xsd:element name="year"/>
        <xsd: choice> < xsd:element name="journal"/>
                        <xsd:element name="conference"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:element>
```

DTD: <!ELEMENT paper (title,author*,year, (journal|conference))>

Elements v.s. Types in XML Schema

```
<xsd:element name="person">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="name"
                   type="xsd:string"/>
      <xsd:element name="address"
                   type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

```
<xsd:element name="person"
             type="ttt">
<xsd:complexType name="ttt">
  <xsd:sequence>
    <xsd:element name="name"
                 type="xsd:string"/>
    <xsd:element name="address"
                 type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>
```

DTD: <!ELEMENT person (name,address)>

- Types:
  - Simple types (integers, strings, ...)
  - Complex types (regular expressions, like in DTDs)
- Element-type-element alternation:
  - Root element has a complex type
  - That type is a regular expression of elements
  - Those elements have their complex types...
  - ...
  - On the leaves we have simple types

Local and Global Types in XML Schema

- Local type:

```
<xsd:element name="person">
        [define locally the person's type]
</xsd:element>
```

- Global type:

```
<xsd:element name="person" name="ttt"/>

<xsd:complexType name="ttt">
        [define here the type ttt]
</xsd:complexType>
```

Global types: can be reused in other elements

Local v.s. Global Elements in XML Schema

- Local element:

```
<xsd:complexType name="ttt">
    <xsd:sequence>
        <xsd:element name="address" type="..."/>...
    </xsd:sequence>
</xsd:complexType>
```

- Global element:

```
<xsd:element name="address" type="..."/>

<xsd:complexType name="ttt">
    <xsd:sequence>
        <xsd:element ref="address"/>  ...
    </xsd:sequence>
</xsd:complexType>
```

Global elements: like in DTDs

Regular Expressions in XML Schema

Recall the element-type-element alternation:

```
<xsd:complexType name="....">
        [regular expression on elements]
</xsd:complexType>
```

Regular expressions:

- <xsd:sequence> A B C </...> = A B C
- <xsd:choice> A B C </...> = A | B | C
- <xsd:group> A B C </...> = (A B C)
- <xsd:... minOccurs="0" maxOccurs="unbounded"> ..</...> = (...)*
- <xsd:... minOccurs="0" maxOccurs="1"> ..</...> = (...)?

Local Names in XML-Schema

name has different meanings in **person** and in **product**

```
<xsd:element name="person">
  <xsd:complexType>
        . . . . .
     <xsd:element name="name">
          <xsd:complexType>
               <xsd:sequence>
                    <xsd:element name="firstname" type="xsd:string"/>
                    <xsd:element name="lastname" type="xsd:string"/>
          </xsd:sequence>
     </xsd:element>
      .  .  .  .
 </xsd:complexType>
</xsd:element>


<xsd:element name="product">
  <xsd:complexType>
        . . . . .
     <xsd:element name="name"   type="xsd:string"/>

  </xsd:complexType>
</xsd:element>
```

Subtle Use of Local Names

```
<xsd:element name="A" type="oneB"/>


<xsd:complexType name="onlyAs">
  <xsd:choice>
     <xsd:sequence>
        <xsd:element name="A" type="onlyAs"/>
        <xsd:element name="A" type="onlyAs"/>
     </xsd:sequence>
     <xsd:element name="A" type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
```

```
<xsd:complexType name="oneB">
  <xsd:choice>
     <xsd:element name="B" type="xsd:string"/>
     <xsd:sequence>
        <xsd:element name="A" type="onlyAs"/>
        <xsd:element name="A" type="oneB"/>
     </xsd:sequence>
     <xsd:sequence>
        <xsd:element name="A" type="oneB"/>
        <xsd:element name="A" type="onlyAs"/>
     </xsd:sequence>
  </xsd:choice>
</xsd:complexType>
```

Arbitrary deep binary tree with A elements, and a single B element

Summary of XML Schema

- Formal Expressive Power:
  - Can express precisely the regular tree languages (over unranked trees)
- Lots of other stuff
  - Some form of inheritance
  - A “null” value
  - Large collection of data types