MONASH University

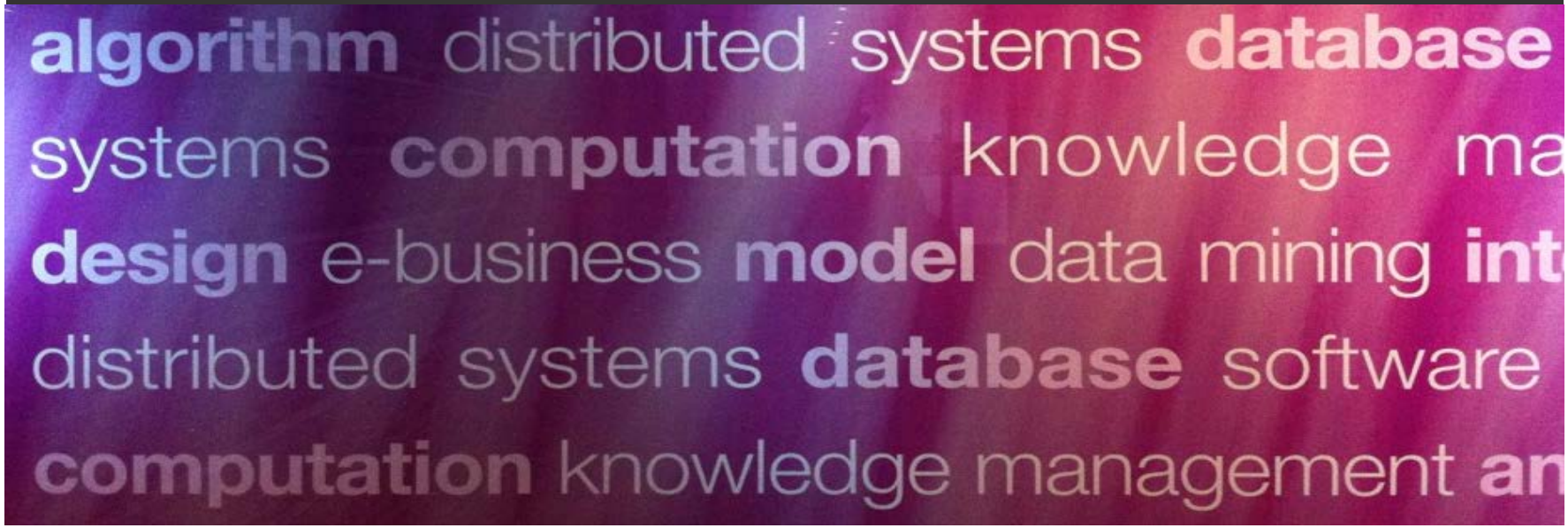
Information Technology

FIT3176 Advanced Database Design

Topic 8: XML Schemas

Dr. Minh Le

Minh.Le@monash.edu



*Adapted from slides developed by A/Prof David Taniar & Lindsay Smith

This unit covers…

1. Advanced Database Design
   - E/R is not complete enough
   - EER, covering superclass and subclass

2. SQL and PL/SQL
   - Trigger, Procedures/Functions and Packages

3. XML and XML DB
   - XML Documents
   - **XML Schema** (*this week*)
   - XML Database

4. JSON and JSON DB
   - JSON Documents and Schema
   - JSON in Oracle

Learning Objectives

By the end of this week you should be able to:

- describe how an XML document is validated by a Document Type Definition (DTD) or schema
- create schemas and use schema syntax to
  - create simple type declarations, and
  - create complex type declarations
- use a variety of approaches to structuring a schema and be aware of the consequences of each approach:
  - Russian Doll
  - Flat
  - Venetian
- derive customised data types

References

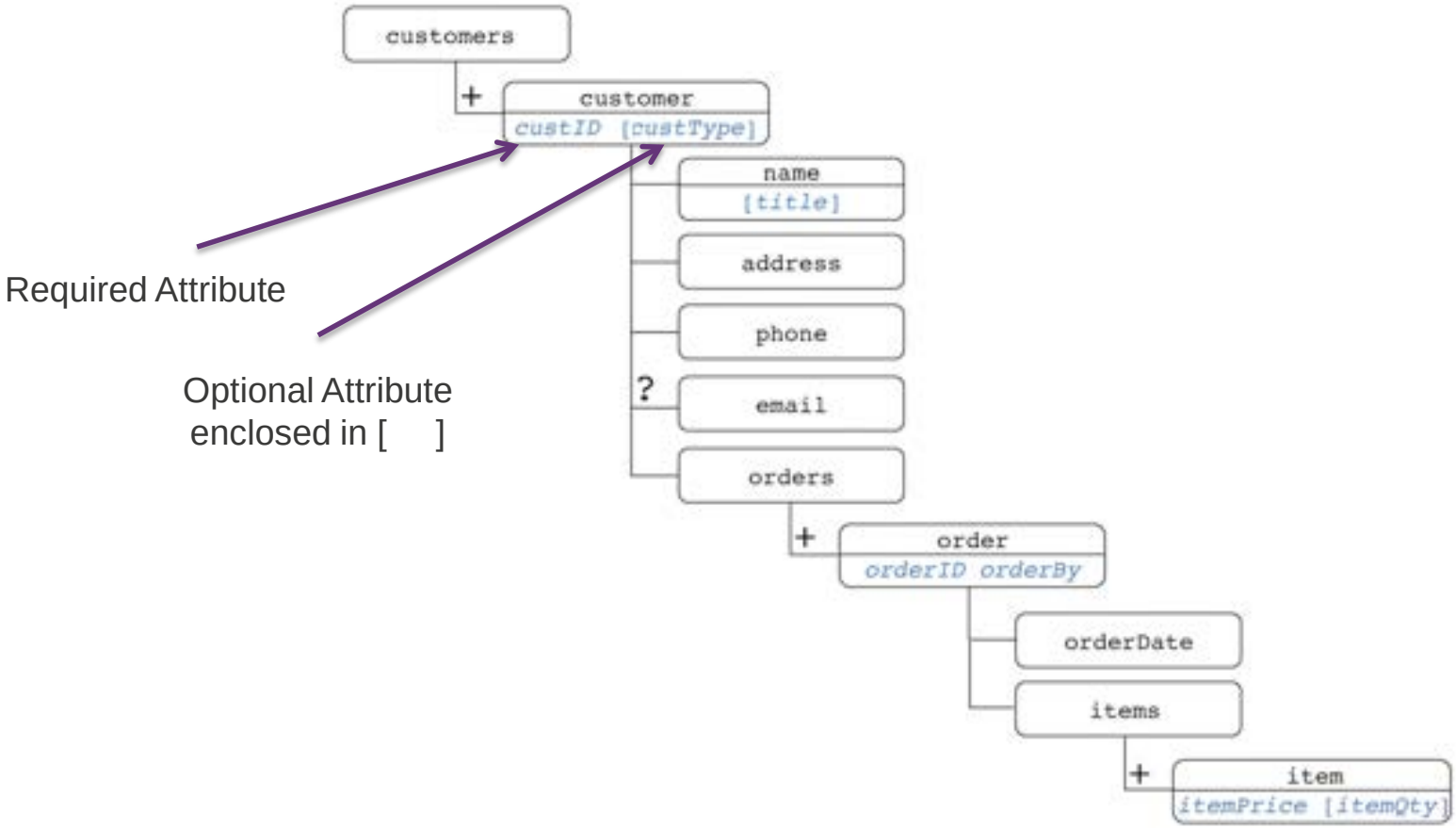
1. Carey, P., New Perspectives on XML 2nd Edition Comprehensive, 2007, Thomson Course Technology, Tutorial 3 & 4 (Available in the library)
2. Carey, P. and Vodnik, S., *New Perspectives on XML 3rd Edition Comprehensive*, 2015, Thomson Course Technology
3. Coronel, Morris, & Rob, Database Systems: Design, Implementation & Management, 11th Edition 2015, Thomson Course Technology. Chapter 14, Section 14.3

Why XML Schemas…

Well-formed vs. **Valid** XML documents

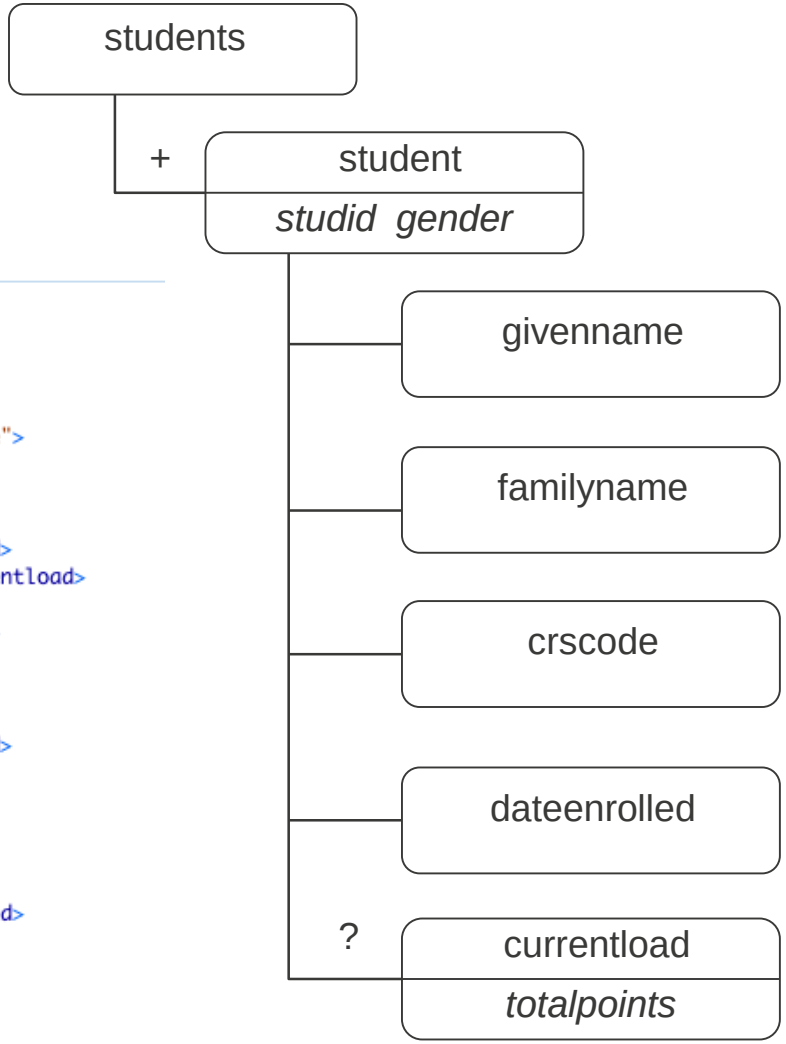
```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <!--  [5 lines]
9
10 <students>
11     <student studid="S12345678" gender="Female">
12         <givenname>Mary</givenname>
13         <familyname>Jones</familyname>
14         <crscode>3334</crscode>
15         <dateenrolled>27/02/2015</dateenrolled>
16         <currentload totalpoints="24">4</currentload>
17     </student>
18     <student studid="S22223344" gender="Male">
19         <givenname>Sung Hoon</givenname>
20         <familyname>Xiue</familyname>
21         <crscode>3830</crscode>
22         <dateenrolled>2014-07-02</dateenrolled>
23     </student>
24     <student studid="S33451087">
25         <familyname>Olson</familyname>
26         <givenname>Phyllis</givenname>
27         <Crscode>2380</Crscode>
28         <dateenrolled>16-Feb-2014</dateenrolled>
29     </student>
30 </students>
31
```

Showing required and optional attributes



Students XML document

```xml
<?xml version="1.0" encoding="UTF-8"?>

<!--  [5 lines]

<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>27/02/2015</dateenrolled>
        <currentload totalpoints="24">4</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2014-07-02</dateenrolled>
    </student>
    <student studid="S33451087">
        <familyname>Olson</familyname>
        <givenname>Phyllis</givenname>
        <Crscode>2380</Crscode>
        <dateenrolled>16-Feb-2014</dateenrolled>
    </student>
</students>
```



Main Reference:

- Carey, P., *New Perspectives on XML 2nd Edition Comprehensive*, 2007, Thomson Course Technology, Tutorial 4, **pages 145-224**, (Available in the library)

1. Starting a Schema File (p150)
2. Understanding Simple and Complex Types (p152)
3. Working with Simple Type Elements (p153)
4. Declaring an Attribute (p154)
5. Associating Attributes and Elements (p155)
6. Referencing an Element or Attribute (p157)
7. Working with Child Elements (p159)
8. Applying a Schema (p165)
9. Working with XML Schema Data Types (p167)
10. Deriving New Data Types (p174)
11. Deriving a Restricted Data Type (p176)
12. Working with Character Patterns (p183)
13. Working with Named Types (p190)
14. Structuring a Schema (p192)
15. Placing a Schema in a Namespace (p195)
16. Validating a Compound Document (p201)

1. Starting a Schema File

- An XML Schema is placed in a separate file, with an **extension .xsd**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   schema content
</xs:schema>
```

- Namespace prefix is either **xs** or xsd

- Don’t forget the **prolog** in the XML Schema file:

```
<?xml version=“1.0” encoding=“UTF-8” ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   schema content

</xs:schema>
```

2. Understanding Simple and Complex Types (1)

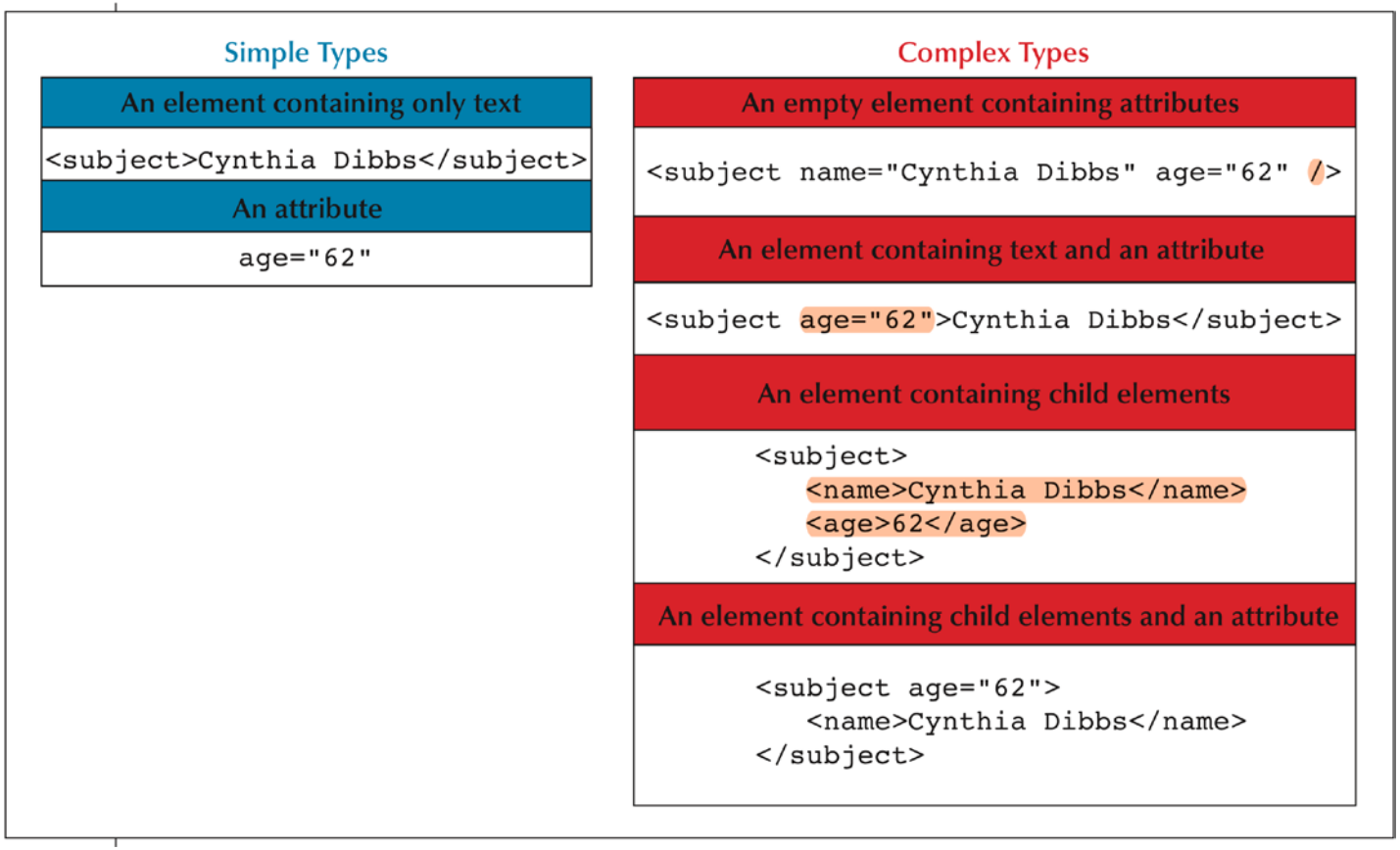
Q1: In the students XML document below currentload is ___ content type.

A. Simple
B. Complex
C. Mixed
D. Standalone

```
<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>2012-02-27</dateenrolled>
        <currentload totalpoints="30">5</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2013-07-02</dateenrolled>
    </student>
    <student studid="S33451087" gender="Female">
        <givenname>Phyllis</givenname>
        <familyname>Olson</familyname>
        <crscode>2380</crscode>
        <dateenrolled>2014-02-16</dateenrolled>
    </student>
</students>
```

2. Understanding Simple and Complex Types (2)

- XML Schema support two types of content: **simple** and **complex**



3. Working with Simple Type Elements

- Simple Type Elements:

```
<xs:element name="name" type=”xs:type" />
```

- An example:

```
<xs:element name="lastName" type="xs:string" />
```

Declaring simple type elements

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="lastName" type="xs:string" />
   <xs:element name="firstName" type="xs:string" />
   <xs:element name="dateOfBirth" type="xs:string" />
   <xs:element name="age" type="xs:string" />
   <xs:element name="stage" type="xs:string" />
   <xs:element name="comment" type="xs:string" />
</xs:schema>
```

simple type elements

4. Declaring an Attribute (1)

- An attribute is another example of a simple type:

```
<xs:attribute name="name" type="type" default="default"
fixed="fixed" />
```

- An example:

```
<xs:attribute name="Gender" type="xs:string" default="female"
/>
```

Declaring attributes

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="lastName" type="xs:string" />
   <xs:element name="firstName" type="xs:string" />
   <xs:element name="dateOfBirth" type="xs:string" />
   <xs:element name="age" type="xs:string" />
   <xs:element name="stage" type="xs:string" />
   <xs:element name="comment" type="xs:string" />

   <xs:attribute name="patID" type="xs:string" />
   <xs:attribute name="onStudy" type="xs:string" />
   <xs:attribute name="scale" type="xs:string" />

</xs:schema>
```

attribute declarations

4. Declaring an Attribute (2)

Q2: Given the following XML Schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="lastName" type="xs:string" />
    <xs:element name="firstName" type="xs:string" />
    <xs:element name="dateOfBirth" type="xs:string" />
    <xs:element name="age" type="xs:string" />
    <xs:element name="stage" type="xs:string" />
    <xs:element name="comment" type="xs:string" />

    <xs:attribute name="patID" type="xs:string" />
    <xs:attribute name="onStudy" type="xs:string" />
    <xs:attribute name="scale" type="xs:string" />

</xs:schema>
```

The following patient.xml is:

A. Correct

B. Incorrect

C. I don't know

```
<patients>
   <patient patID="MR890-041-02" onStudy="TBC-080-5">
      <lastName>Dibbs</lastName>
      <firstName>Cynthia</firstName>
      <dateOfBirth>1945-05-22</dateOfBirth>
      <age>62</age>
      <stage>II</stage>
      <performance scale="Karnofsky">0.81</performance>
   </patient>
   <patient patID="MR771-121-10" onStudy="TBC-080-5">
      <lastName>Wilkes</lastName>
      <firstName>Karen</firstName>
      <dateOfBirth>1959-02-24</dateOfBirth>
      <age>48</age>
      <stage>II</stage>
      <comment>Dropped out of study.</comment>
      <performance scale="Karnofsky">0.84</performance>
   </patient>
   <patient patID="MR701-891-05" onStudy="TBC-080-5">
      <lastName>Sanchez</lastName>
      <firstName>Olivia</firstName>
      <dateOfBirth>1958-08-14</dateOfBirth>
      <age>49 years old</age>
      <stage>II</stage>
      <comment>Possibly stage I/II</comment>
      <comment>Karnofsky performance rating unavailable.</comment>
      <performance scale="Bell">0.89</performance>
   </patient>
   <patient patID="MR805-891-08" onStudy="TBC-080-5">
      <lastName>Russell</lastName>
      <firstName>Alice</firstName>
      <dateOfBirth>1952-9-14</dateOfBirth>
      <age>55</age>
      <stage>II</stage>
      <performance scale="Karnofsky">1.76</performance>
   </patient>
   <patient patID="mr815-741-03" onStudy="tbc-080-5">
      <lastName>Browne</lastName>
      <firstName>Brenda</firstName>
      <dateOfBirth>1964-04-25</dateOfBirth>
      <age>39</age>
      <stage>I</stage>
      <performance scale="Karnosfky">0.88</performance>
   </patient>
</patients>
```

5. Associating Attributes and Elements (1)

Q3: totalpoints (using string data) in the students xml would be declared the following in the XML Schema:

A.
```
<xs:element name="totalpoints" type="xs:string"/>
```

B.
```
<xs:attribute name="totalpoints" type="xs:string"/>
```

C.
```
<xs:attribute name="totalpoints" type="xs:string" use="required"/>
```

D.
```
<xs:complexType>
  <xs:attribute name="totalpoints"
    type="xs:string" use="required"/>
</xs:complexType>
```

```
<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>2012-02-27</dateenrolled>
        <currentload totalpoints="30">5</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2013-07-02</dateenrolled>
    </student>
    <student studid="S33451087" gender="Female">
        <givenname>Phyllis</givenname>
        <familyname>Olson</familyname>
        <crscode>2380</crscode>
        <dateenrolled>2014-02-16</dateenrolled>
    </student>
</students>
```

5. Associating Attributes and Elements (2)

- Four complex type elements that usually appear in an instance document are the following:
    1. The element is an **empty element** and contains only **attributes**.
    2. The element contains **textual content and attributes** but **no child** elements.
    3. The element contains **child elements** but **not attributes**.
    4. The element contains both **child elements and attributes**

5. Associating Attributes and Elements (3)

Q4: What content type is the students element?

A. The element is an empty element and contains only attributes.
B. The element contains textual content and attributes but no child elements.
C. The element contains child elements but not attributes.
D. The element contains both child elements and attributes.

```
<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>2012-02-27</dateenrolled>
        <currentload totalpoints="30">5</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2013-07-02</dateenrolled>
    </student>
    <student studid="S33451087" gender="Female">
        <givenname>Phyllis</givenname>
        <familyname>Olson</familyname>
        <crscode>2380</crscode>
        <dateenrolled>2014-02-16</dateenrolled>
    </student>
</students>
```

5. Associating Attributes and Elements (3)

Elements and Attributes of the Student Document

- Four complex type elements that usually appear in an instance document are the following:
  1. The element is an **empty element** and contains only **attributes**.
  2. The element contains **textual content and attributes** but **no child** elements.
  3. The element contains **child elements** but **not attributes**.
  4. The element contains both **child elements and attributes**

ITEM	CONTENT	CONTENT TYPE	COMPLEX TYPE
students	element	complex	3
student	element	complex	4
studid	*attribute*	*simple*	
gender	*attribute*	*simple*	
givenname	*element*	*simple*	
familyname	*element*	*simple*	
crscode	*element*	*simple*	
dateenrolled	*element*	*simple*	
currentload	element	complex	2
totalpoints	*attribute*	*simple*	

```
<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>2012-02-27</dateenrolled>
        <currentload totalpoints="30">5</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2013-07-02</dateenrolled>
    </student>
    <student studid="S33451087" gender="Female">
        <givenname>Phyllis</givenname>
        <familyname>Olson</familyname>
        <crscode>2380</crscode>
        <dateenrolled>2014-02-16</dateenrolled>
    </student>
</students>
```

5. Associating Attributes and Elements (4)

Declaring Empty Elements and Attributes

```
<xs:element name="name">
  <xs:complexType>
     attributes
  </xs:complexType>
</xs:element>
```

- Four complex type elements that usually appear in an instance document are the following:
  1. The element is an **empty element** and contains only **attributes**.
  2. The element contains **textual content and attributes** but **no child** elements.
  3. The element contains **child elements** but **not attributes**.
  4. The element contains both **child elements and attributes**

- The XML is:

```
<subject name="Cynthia Dibbs" age="62" />
```

- The XML Schema is:

```
<xs:element name="subject">
  <xs:complexType>
    <xs:attribute name="name" type="xs:string" />
    <xs:attribute name="age" type="xs:string" />
  </xs:complexType>
</xs:element>
```

- The order of the attribute declarations is unimportant!!

5. Associating Attributes and Elements (5)

Declaring Simple Content Elements and Attributes

```
<xs:element name="name">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="type">attributes</xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- The XML is:

```
<performance scale="Karnofsky">0.81</performance>
```

- The XML Schema is:

```
<xs:element name="performance">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="scale" type="xs:string" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

5. Associating Attributes and Elements (6)

Specifying the Use of Attributes

- The use of attribute has the following values:
  1. **required** – the attribute must always appear with the element
  2. **optional** – the use of the attribute is optional with the element
  3. **prohibited** – the attribute cannot be used with the element

- An Example:

```
<xs:attribute name="scale" type="xs:string" use="required" />
```

- The default value is optional

6. Referencing an Element or Attribute (1)

Q5: The below XML code to declare an attribute is

A. Correct

B. Incorrect

C. I don't know

```
<xs:attribute name="scale" type="xs:string" use="required" />
<xs:element name="performance">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="scale" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

6. Referencing an Element or Attribute (2)

- Reusability of an attribute declaration, by referencing the attribute declaration, rather than by nesting it:

```
<xs:attribute name="scale" type="xs:string" />
<xs:element name="performance">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute ref="scale" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- The “use” declaration must be in the nested form.

7. Working with Child Elements (1)

Using Compositors

- There are three types of compositors:
  - **sequence** – defines a specific order for the child elements
  - **choice** – allows by one of the child elements to appear in the instance document
  - **all** – allows any of the child elements to appear in any order in the instance document, however, they must appear either only once or not at all
- A **sequence** example (the elements inside address must be in this order)

```
<xs:element name="address">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="street" type="xs:string"/>
      <xs:element name="city" type="xs:string"/>
      <xs:element name="state" type="xs:string"/>
      <xs:element name="country" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

7. Working with Child Elements (2)

Q6: Given the following XML Schema

```
<xs:element name="Family">
  <xs:complexType>
    <xs:all>
      <xs:element name="Father" type="xs:string"/>
      <xs:element name="Mother" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Is the following XML:

```
<Family>
   <Mother>Jane</Mother>
   <Father>Jim</Father>
</Family>
```

A. Correct
B. Incorrect
C. I don't know

7. Working with Child Elements (3)

- A **choice** example (choose one, not both)

```
<xs:element name="sponsor">
  <xs:complexType>
    <xs:choice>
      <xs:element name="parent" type="xs:string"/>
      <xs:element name="guardian" type="xs:string"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- An **all** example (all but no particular order)

```
<xs:element name="Family">
  <xs:complexType>
    <xs:all>
      <xs:element name="Father" type="xs:string"/>
      <xs:element name="Mother" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

7. Working with Child Elements (4)

- Compositors can be nested and combined with each other:

```
<xs:element name="Account">
  <xs:complexType>
    <xs:sequence>
      <xs:choice>
        <xs:element name="Person" type="xs:string"/>
        <xs:element name="Company" type="xs:string"/>
      </xs:choice>
      <xs:choice>
        <xs:element name="Cash" type="xs:string"/>
        <xs:element name="Credit" type="xs:string"/>
      </xs:choice>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- **Restriction!!**: The **all** compositor cannot contain other compositors as children within an XSD, and each element within an **all** compositor has maxOccurs=1. This makes the **all** compositor relatively restrictive, so sequence and choice types are more common.

7. Working with Child Elements (5)

Q7: Is the following XML Schema:

A. Correct

B. Incorrect

C. I don't know

```
<xs:element name="Account">
  <xs:complexType>
    <xs:all>
      <xs:choice>
        <xs:element name="Person" type="xs:string"/>
        <xs:element name="Company" type="xs:string"/>
      </xs:choice>
      <xs:choice>
        <xs:element name="Cash" type="xs:string"/>
        <xs:element name="Credit" type="xs:string"/>
      </xs:choice>
    </xs:all>
  </xs:complexType>
</xs:element>
```

7. Working with Child Elements (6)

Specifying the Occurrences of an Item

- Use **minOccurs** and **maxOccurs** attribute

```
<xs:element name="patient" type="xs:string" minOccurs="1"
maxOccurs="3"/>
```

- Rules:
  - minOccurs="0" means that the declared item is optional
  - maxOccurs can be a positive value, or "unbounded"
  - If there is minOccurs but no maxOccurs, maxOccurs = minOccurs
  - Default values: minOccurs=maxOccurs=1
  - Like the "use" value, minOccurs and maxOccurs must be declared locally within a complex type element

7. Working with Child Elements (7)

- minOccurs and maxOccurs for the entire sequence of items:

```
<xs:element name="Customer">
  <xs:complexType>
    <xs:sequence minOccurs="1" maxOccurs="unbounded">
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="MiddleName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- The **sequence** of three child elements (FirstName, MiddleName, LastName) can be repeated countless times within the Customer element.

7. Working with Child Elements (8)

Q8: What compositor should be used for students?

A. sequence

B. choice

C. all

D. maxOccurs = "unbounded"

```
<students>
    <student studid="S12345678" gender="Female">
        <givenname>Mary</givenname>
        <familyname>Jones</familyname>
        <crscode>3334</crscode>
        <dateenrolled>2012-02-27</dateenrolled>
        <currentload totalpoints="30">5</currentload>
    </student>
    <student studid="S22223344" gender="Male">
        <givenname>Sung Hoon</givenname>
        <familyname>Xiue</familyname>
        <crscode>3830</crscode>
        <dateenrolled>2013-07-02</dateenrolled>
    </student>
    <student studid="S33451087" gender="Female">
        <givenname>Phyllis</givenname>
        <familyname>Olson</familyname>
        <crscode>2380</crscode>
        <dateenrolled>2014-02-16</dateenrolled>
    </student>
</students>
```

7. Working with Child Elements (9)

Working with Child Elements and Attributes

- An example: the patient element contains two attributes (patID and onStudy) and seven child elements (lastName, firstName, dateOfBirth, age, stage, comment, and performance)

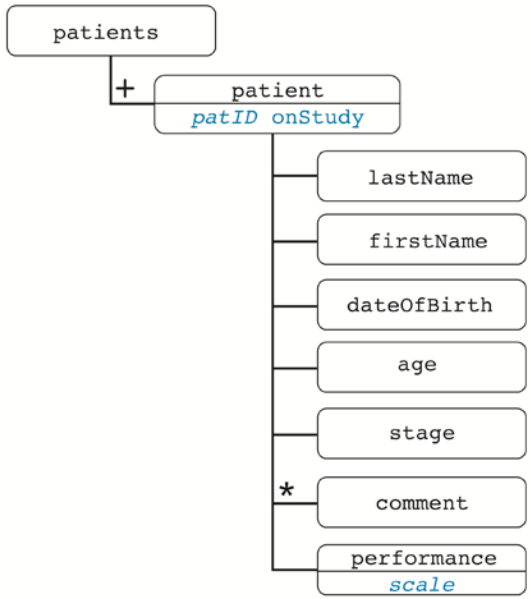
```
<xs:element name="patient">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="lastName"/>
      <xs:element ref="firstName"/>
      <xs:element ref="dateOfBirth"/>
      <xs:element ref="age"/>
      <xs:element ref="stage"/>
      <xs:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element ref="performance"/>
    </xs:sequence>
    <xs:attribute ref="patID" use="required"/>
    <xs:attribute ref="onStudy" use="required"/>
  </xs:complexType>
</xs:element>
```

7. Working with Child Elements (10)

Complete schema for the patient data

patient element definition

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="patients">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="patient" minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="patient">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="lastName"/>
            <xs:element ref="firstName"/>
            <xs:element ref="dateOfBirth"/>
            <xs:element ref="age"/>
            <xs:element ref="stage"/>
            <xs:element ref="comment" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element ref="performance"/>
         </xs:sequence>
         <xs:attribute ref="patID" use="required"/>
         <xs:attribute ref="onStudy" use="required"/>
      </xs:complexType>
   </xs:element>

   <xs:element name="lastName" type="xs:string" />
   <xs:element name="firstName" type="xs:string" />
   <xs:element name="dateOfBirth" type="xs:string" />
   <xs:element name="age" type="xs:string" />
   <xs:element name="stage" type="xs:string" />
   <xs:element name="comment" type="xs:string" />

   <xs:attribute name="patID" type="xs:string" />
   <xs:attribute name="onStudy" type="xs:string" />
   <xs:attribute name="scale" type="xs:string" />

   <xs:element name="performance">
      <xs:complexType>
         <xs:simpleContent>
            <xs:extension base="xs:string">
               <xs:attribute ref="scale" use="required" />
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>
   </xs:element>

</xs:schema>
```



7. Working with Child Elements (11)

Specifying Mixed Content

- An element may contain both text and child elements
- Add the mixed attribute to the <complexType> tag, and set to “true”

```
<element name="Summary">
  <complexType mixed="true">
    <sequence>
      <element name="Name" type="string"/>
      <element name="Study" type="string"/>
    </sequence>
  </complexType>
</element>
```

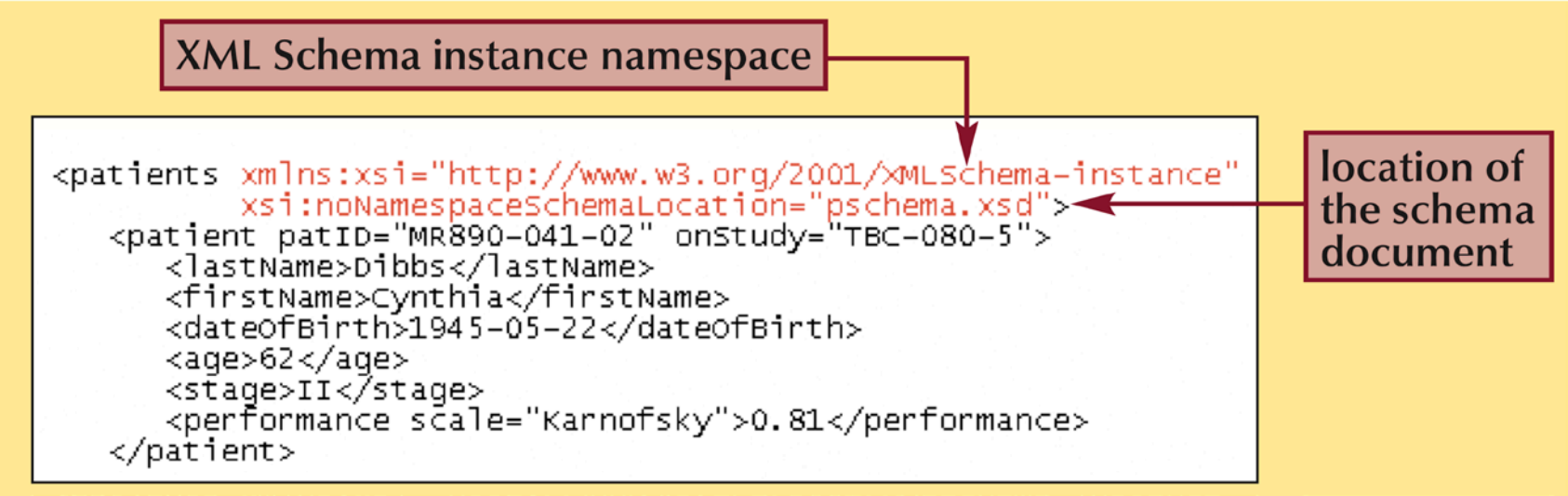
- The XML is:

```
<Summary>
   Patient <Name>Cynthia Davis</Name> was enrolled in
   the <Study>Tamoxifen Study</Study> on 8/15/2003.
</Summary>
```

- Note that XML Schema allows content **text to appear** before, between, and after any of the child elements.

8. Applying a Schema (1)

- To attach a schema to the XML document:
  1. Declare a namespace for XML Schema in the XML document
  2. Indicate the location of the schema file

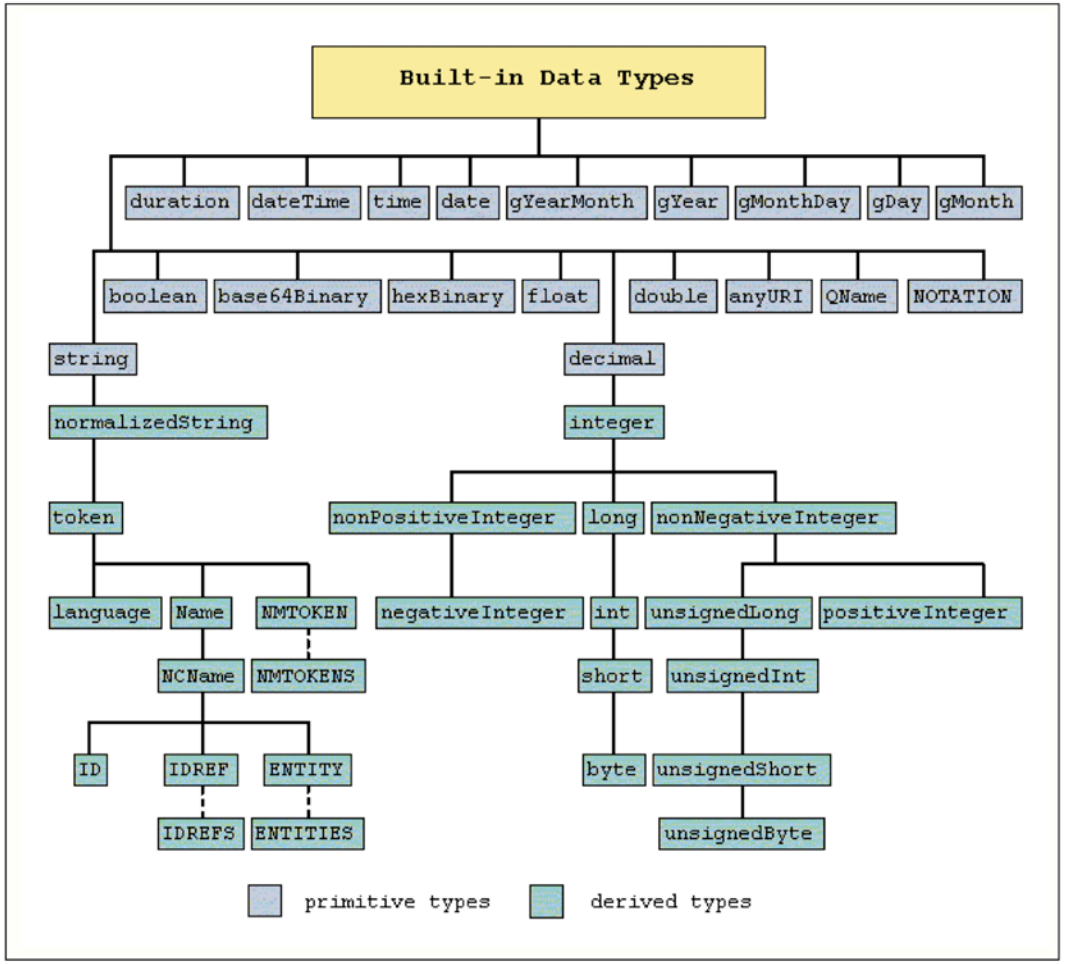


8. Applying a Schema (2)

- **Final notes:**
  - Specifying a schema in an XML document is treated only as a hint by validating parsers, and some parsers ignore specified schemas.
  - An XHTML editor may have a built-in XHTML schema that it uses in place of any schema you might specify for a document.
  - In commerce programs, parsers are often written to accept validation only from approved schemas, and do not allow document authors to specify their own. This is done to prevent improper financial documents from being fraudulently submitted to a server attached to an unsupported schema.

9. Working with XML Schema Data Types (1)

- **Built-in data types:**
  - *Primitive* data types (or base types): 19
  - *Derived* data types: 25, based on the 19 primitive data types
  - Total = 44
- **User-derived data types**



9. Working with XML Schema Data Types (2)

String Data Types

- For example:

```
<xs:attribute name="patID" type="xs:ID" />
<xs:attribute name="onStudy" type="xs:string" />
<xs:attribute name="scale" type="xs:string" />
```

String data types

Data Type	Description
xs:string	A text string containing all legal characters from the ISO/IEC character set, including all white space characters
xs:normalizedString	A text string in which all white space characters are replaced with blank spaces
xs:token	A text string in which blank spaces are replaced with a single blank space; opening and closing spaces are removed
xs:NMTOKEN	A text string containing valid XML names with no white space
xs:NMTOKENS	A list of NMTOKEN data values separated by white space
xs:Name	A text string similar to the NMTOKEN data type except that names must begin with a letter or the character ":" or "-"
xs:NCName	A "noncolonized name," derived from the Name data type but restricting the use of colons anywhere in the name
xs:ID	A unique ID name found nowhere else in the instance document
xs:IDREF	A reference to an ID value found in the instance document
xs:IDREFS	A list of ID references separated by white space
xs:ENTITY	A value matching an unparsed entity defined in a DTD
xs:ENTITIES	A list of entity values matching an unparsed entity defined in a DTD

9. Working with XML Schema Data Types (3)

Numeric Data Types

- Based on four primitive data types: *decimal*, *double*, *float*, and *boolean*

```
<xs:element name="lastName" type="xs:string" />
<xs:element name="firstName" type="xs:string" />
<xs:element name="dateOfBirth" type="xs:string" />
<xs:element name="age" type="xs:positiveInteger" />
<xs:element name="stage" type="xs:string" />
<xs:element name="comment" type="xs:string" />

<xs:attribute name="patID" type="xs:ID" />
<xs:attribute name="onStudy" type="xs:string" />
<xs:attribute name="scale" type="xs:string" />

<xs:element name="performance">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
            <xs:attribute ref="scale" use="required" />
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

Numeric data types

Data Type	Description
xs:decimal	A decimal number in which the decimal separator is always a dot (.) with a leading + or - character allowed; no non-numeric characters are allowed, nor is exponential notation.
xs:integer	An integer
xs:nonPositiveInteger	An integer less than or equal to zero
xs:negativeInteger	An integer less than zero
xs:nonNegativeInteger	An integer greater than or equal to zero
xs:positiveInteger	An integer greater than zero
xs:float	A floating point number allowing decimal values and values in scientific notation; infinite values can be represented by -INF and INF, non-numeric values can be represented by NaN.
xs:double	A double precision floating point number
xs:boolean	A Boolean value that has the value true, false, 0, or 1

9. Working with XML Schema Data Types (4)

Q9: XML Schema:

```
<xs:element name=”examdate” type=“xs:date” />
<xs:element name=”examtime” type=“xs:time” />
```

XML instance document:

```
1. <examdate>2018-04-14</examdate>
2. <examtime>15:00</examtime>
```

A. 1 and 2 are correct
B. 1 and 2 are incorrect
C. 1 is correct, 2 is incorrect
D. 1 is incorrect, 2 is correct
E. I don’t know

9. Working with XML Schema Data Types (5)

Dates and Times

- The date format is ***yyyy-mm-dd*** (01-12, and 01-31)
- If you want to use any other date formats, such as Apr 8, 2016, you need to a user-defined data type

- The time format is 24hr time format: ***hh:mm:ss*** (hh is 0-23, mm and ss 00-59)
- ss is not optional

```
<xs:element name="lastName" type="xs:string" />
<xs:element name="firstName" type="xs:string" />
<xs:element name="dateOfBirth" type="xs:date" />
<xs:element name="age" type="xs:positiveInteger" />
<xs:element name="stage" type="xs:string" />
<xs:element name="comment" type="xs:string" />
```

Date and time data types

Data Type	Description
xs:datetime	A date and time entered in the format *yyyy-mm-ddThh:mm:ss* where *yyyy* is the four-digit year, *mm* is the two-digit month, *dd* is the two-digit day, *T* is the time zone, *hh* is the two-digit hour, *mm* is the two-digit minute, and *ss* is the two-digit second
xs:date	A date entered in the format *yyyy-mm-dd*
xs:time	A time entered in the format *hh:mm:ss*
xs:gYearMonthDay	A date based on the Gregorian calendar entered in the format *yyyy-mm-dd* (equivalent to xs:date)
xs:gYearMonth	A date entered in the format *yyyy-mm* (no day is specified)
xs:gYear	A year entered in the format *yyyy*
xs:gMonthDay	A month and day entered in the format *--mm-dd*
xs:gMonth	A month entered in the format *--mm*
xs:gDay	A day entered in the format *---dd*
xs:duration	A time duration entered in the format PyYmMdDhHmMsS where *y*, *m*, *d*, *h*, *m*, and *s* are the duration values in years, months, days, hours, minutes, and seconds; an optional negative sign is also permitted to indicate a negative time duration

10. Deriving New Data Types (1)

- New data types are simple data types because they contain values like attributes and simple type elements.

```
<xs:simpleType name="name">
  Rules
</xs:simpleType>
```

- User-derived data types fall into three general categories: **list**, **union**, and **restriction**.

10. Deriving New Data Types (2)

Q10: We would like to record the mark of each exam question.

XML Schema:

```
1 <xs:element name=”examMark" type=”xs:examMarkList" />
2 <xs:simpleType name=”examMarkList">
3   <xs:list itemType="xs:decimal" />
4 </xs:simpleType>
```

Is the following XML instance document:

```
5 <examMark>7.5; 10; 3; 10; 8; 8; 9; 5.5; 0; 0</examMark>
```

A. Error: line 1
B. Error: line 2
C. Error: line 5
D. Errors: lines 1, 2
E. Errors: lines 2, 5
F. Errors: lines 1, 5
G. No errors

10. Deriving New Data Types (3)

Deriving a List Data Type

- List of values separated by white space, in which each item in the list is derived from an established data type

```
<xs:simpleType name="name">
  <xs:list itemType="type" />
</xs:simpleType>
```

- For example, in the XML instance document (must use **whitespace**):

```
<wbc>15.1 15.8 12.0 9.3 7.1 5.2 4.3 3.4</wbc>
```

- Declare a simple type in the schema:

```
<xs:simpleType name="wbcList">
  <xs:list itemType="xs:decimal" />
</xs:simpleType>
```

- Add this new data type to the wbc element:

```
<xs:element name="wbc" type="wbcList" />
```

- Note: the type value does not have the xs prefix, because wbcList is not part of the XML Schema namespace.

10. Deriving New Data Types (4)

Deriving a Union Data Type

- A union data type is composed of the value and/or lexical spaces from any number of base types. Each of the base types is known as a member type.

```
<xs:simpleType name="name">
  <xs:union memberTypes="type1 type2 type3 ..." />
</xs:simpleType>
```

- Union from nested simple types:

```
<xs:simpleType name="name">
  <xs:union>
    <xs:simpleType>rules1</xs:simpleType>
    <xs:simpleType>rules2</xs:simpleType>
  </xs:union>
</xs:simpleType>
```

10. Deriving New Data Types (5)

Deriving a Union Data Type

- An example of XML instance document:

```
<wbc>15.9 high 14.2 9.8 normal low 5.3</wbc>
```

- XML Schema:

```
<xs:simpleType name="wbcType">
  <xs:union memberTypes="xs:decimal xs:string"/>
</xs:simpleType>

<xs:simpleType name="wbcList">
  <xs:list itemType="wbcType"/>
</xs:simpleType>

<xs:element name="wbc" type="wbcList" />
```

- Union data types are often used for multilingual documents in which the data content is expressed in different languages, but must be validated based on a single schema.

11. Deriving a Restricted Data Type (1)

- XML Schema provides **twelve constraining facets** that can be used to derive new data types

```
<xs:simpleType name="name">
  <xs:restriction base="type">
    <xs:facet1 value="value1" />
    <xs:facet2 value="value2" />
  </xs:restriction>
</xs:simpleType>
```

For example (Range):

```
<xs:simpleType name="ageType">
  <xs:restriction base="xs:integer">
    <xs:minInclusive value="21"/>
  </xs:restriction>
</xs:simpleType>
```

For example (Enumeration):

```
<xs:simpleType name="stageType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="I" />
    <xs:enumeration value="II" />
  </xs:restriction>
</xs:simpleType>
```

Constraining facets

Facet	Description
enumeration	Constrains the data type to a specified list of values
length	Specifies the length of the data type in characters (for text strings) or items (for lists)
maxLength	Specifies the maximum length of the data type in characters (for text strings) or items (for lists)
minLength	Specifies the minimum length of the data type in characters (for text strings) or items (for lists)
pattern	Constrains the lexical space of the data type to follow a specific character pattern
whiteSpace	Controls the use of blanks in the lexical space of the data type; the whiteSpace facet has three values: preserve (preserve all white space) replace (replace all tabs, carriage returns, and line feed characters with blank spaces) collapse (collapse all consecutive occurrences of white space to a single blank space, remove any leading or trailing white space)
maxExclusive	Constrains the data type to be less than a maximum value
maxInclusive	Constrains the data type to be less than or equal to a maximum value
minExclusive	Constrains the data type to be greater than a minimum value
minInclusive	Constrains the data type to be greater than or equal to a minimum value
fractionDigits	Specifies the maximum number of decimal places to the right of the decimal point in the data type's value
totalDigits	Specifies the maximum number of decimals in the data type's value

11. Deriving a Restricted Data Type (2)

Constrain the list to 10 exam marks:

```
<xs:simpleType name="examMarks">
    <xs:restriction base="MarksList">
        <xs:length value="10"/>
    </xs:restriction>
</xs:simpleType>
```

Constrain the size of each value to 1 - 10:

```
<xs:simpleType name="ZeroToTen">
    <xs:restriction base="xs:decimal">
        <xs:maxInclusive value="10"/>
    </xsd:restriction>
</xs:simpleType>
<xs:simpleType name="MarksList">
    <xs:list itemType="ZeroToTen"/>
</xs:simpleType>
```

Declare the <examMark> element:

```
<xs:element name="examMark" type="examMarks" />
```

Then, in the XML document, we can have:

```
<examMark>7.5 10 3 10 8 8 9 5.5 0 0</examMark>
```

12. Working with Character Patterns (1)

Regular Expressions

```
<xs:simpleType name="name">
  <xs:restriction base="type">
    <xs:pattern value="regex"/>
  </xs:restriction>
</xs:simpleType>
```

For example (1):

```
<xs:pattern value="\d\d\d"/>
<xs:pattern value="\D\D\D"/>
```

12. Working with Character Patterns (2)

Regular Expressions

```
<xs:simpleType name="name">
  <xs:restriction base="type">
    <xs:pattern value="regex"/>
  </xs:restriction>
</xs:simpleType>
```

For example (1):

```
<xs:pattern value="\d\d\d"/>
<xs:pattern value="\D\D\D"/>
```

For example (2):

```
<xs:pattern value="[dog]"/>
<xs:pattern value="[a-z]"/>
<xs:pattern value="[A-Z]"/>
<xs:pattern value="[a-zA-Z]"/>
<xs:pattern value="[0-9]"/>
<xs:pattern value="[0-9a-zA-Z]"/>
<xs:pattern value=”[1-5]"/>
```

12. Working with Character Patterns (3)

Regular Expressions

```
<xs:simpleType name="name">
  <xs:restriction base="type">
    <xs:pattern value="regex"/>
  </xs:restriction>
</xs:simpleType>
```

For example (1):

```
<xs:pattern value="\d\d\d"/>
<xs:pattern value="\D\D\D"/>
```

For example (2):

```
<xs:pattern value="[dog]"/>
<xs:pattern value="[a-z]"/>
<xs:pattern value="[A-Z]"/>
<xs:pattern value="[a-zA-Z]"/>
<xs:pattern value="[0-9]"/>
<xs:pattern value="[0-9a-zA-Z]"/>
<xs:pattern value=”[1-5]"/>
```

For example (3):

```
<xs:pattern value="\d{3}" />
<xs:pattern value="[A-Z]*"/>
<xs:pattern value="[A-Z]{0,10}"/>
```

12. Working with Character Patterns (4)

Applying a Regular Expression

patID = **MR###-###-###**

where # is digit 0 to 9

onStudy = **AAA-###-#**

where A is an uppercase letter, and # is digit 0 to 9.

```
<xs:attribute name="patID" type="mrType" />
<xs:attribute name="onStudy" type="studyType" />
<xs:attribute name="scale" type="scaleType" />

<xs:simpleType name="mrType">
   <xs:restriction base="xs:ID">
      <xs:pattern value="MR\d{3}-\d{3}-\d{2}" />
   </xs:restriction>
</xs:simpleType>

<xs:simpleType name="studyType">
   <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z]{3}-\d{3}-\d" />
   </xs:restriction>
</xs:simpleType>
```

13. Working with Named Types (1)

- Previous sections worked with simpleType element to create customized data types
- This section focuses on creating customized complex types
- An example of a complex type (this is an **unnamed complex type** or an **anonymous complex type**)

```
<xs:element name="client">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstName" type="xs:string"/>
      <xs:element name="lastName" type="xs:string" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

13. Working with Named Types (2)

- We can **name** a complex type

```
<xs:complexType name="fullName">
  <xs:sequence>
    <xs:element name="firstName" type="xs:string"/>
    <xs:element name="lastName" type="xs:string" />
  </xs:sequence>
</xs:complexType>
```

- Then we can reuse this “fullName” complex type

```
<xs:element name="client" type="fullName" />
<xs:element name="salesperson" type="fullName" />
```

13. Working with Named Types (3)

Named Model Groups

- An example:

```
<xs:group name="fullName">
  <xs:sequence>
    <xs:element name="firstName" type="xs:string" />
    <xs:element name="lastName" type="xs:string" />
  </xs:sequence>
</xs:group>
```

- The element declarations must be enclosed within a sequence, choice, or all compositor

```
<xs:element name="client">
  <xs:complexType>
    <xs:group ref="fullName"/>
  </xs:complexType>
</xs:element>
```

13. Working with Named Types (4)

Working with Named Attribute Groups

- An example in an XML instance document:

```
<doctor DRID="DR251" dept="Pediatrics"> Curt Hurley</doctor>
```

- Assume both the DRID and dept attributes may need to be used in other elements

- Then in the XML Schema:

```
<xs:attributeGroup name="DRInfo">
  <xs:attribute name="DRID" type="xs:string" use="required" />
  <xs:attribute name="dept" type="xs:string" use="required" />
</xs:attributeGroup>

<xs:element name="doctor" type="deptData"/>
<xs:complexType name="deptData">
  <xs:simpleContent>
    <xs:extension base="string">
      <xs:attributeGroup ref="DRInfo"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

14. Structuring a Schema

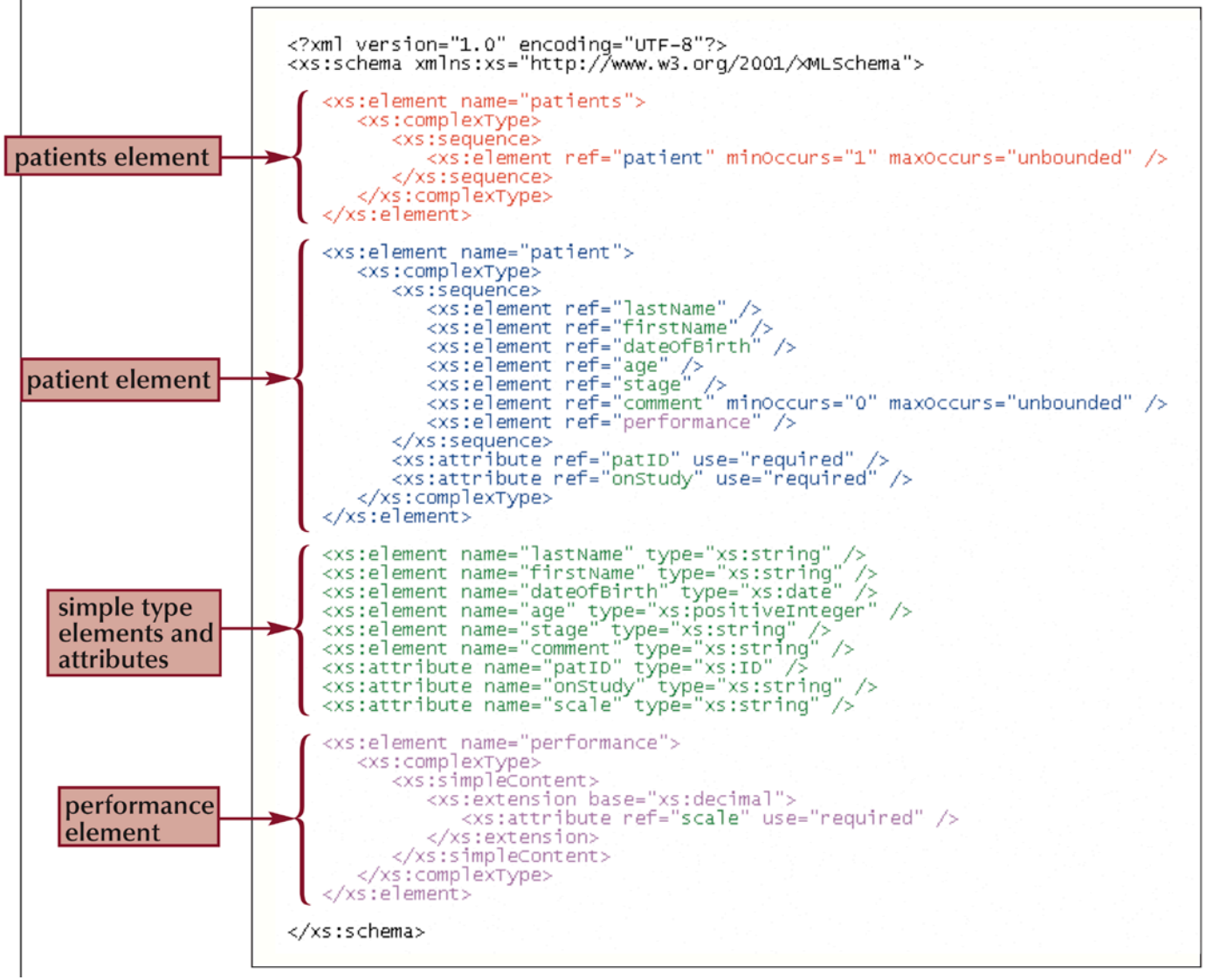
- Three basic schema design:

1. Flat Catalog Design (or Salami Slice Design)
   All declarations are made globally

2. Russian Doll
   It has only one global element with everything else nested inside it

1. Venetian Blind
   Similar to a flat catalog, except that instead of declaring elements and attributes globally, it creates named types and references those types within a single global element

A flat catalog design



A Russian Doll design

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="patients">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="patient" minOccurs="1" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="lastName" type="xs:string" />
              <xs:element name="firstName" type="xs:string" />
              <xs:element name="dateOfBirth" type="xs:date" />
              <xs:element name="age" type="xs:positiveInteger" />
              <xs:element name="stage" type="xs:string" />
              <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
              <xs:element name="performance">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:decimal">
                      <xs:attribute name="scale" type="xs:string" use="required" />
                    </xs:extension>
                  </xs:simpleContent>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="patID" type="xs:ID" use="required" />
            <xs:attribute name="onStudy" type="xs:string" use="required" />
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

A Venetian blind design

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="patients">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="patient" type="pType" minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:complexType name="pType">
      <xs:group ref="childElements" />
      <xs:attributeGroup ref="patientAtt" />
   </xs:complexType>

   <xs:group name="childElements">
      <xs:sequence>
         <xs:element name="lastName" type="xs:string" />
         <xs:element name="firstName" type="xs:string" />
         <xs:element name="dateOfBirth" type="xs:date" />
         <xs:element name="age" type="xs:positiveInteger" />
         <xs:element name="stage" type="xs:string" />
         <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
         <xs:element name="performance" type="perfType" />
      </xs:sequence>
   </xs:group>

   <xs:attributeGroup name="patientAtt">
      <xs:attribute name="patID" type="xs:ID" />
      <xs:attribute name="onStudy" type="xs:string" />
   </xs:attributeGroup>

   <xs:complexType name="perfType">
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
            <xs:attribute name="scale" type="xs:string" use="required" />
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

</xs:schema>
```

A flat catalog design

patients element

patient element

simple type elements and attributes

performance element

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="patients">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="patient" minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:element name="patient">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="lastName" />
            <xs:element ref="firstName" />
            <xs:element ref="dateOfBirth" />
            <xs:element ref="age" />
            <xs:element ref="stage" />
            <xs:element ref="comment" minOccurs="0" maxOccurs="unbounded" />
            <xs:element ref="performance" />
         </xs:sequence>
         <xs:attribute ref="patID" use="required" />
         <xs:attribute ref="onStudy" use="required" />
      </xs:complexType>
   </xs:element>

   <xs:element name="lastName" type="xs:string" />
   <xs:element name="firstName" type="xs:string" />
   <xs:element name="dateOfBirth" type="xs:date" />
   <xs:element name="age" type="xs:positiveInteger" />
   <xs:element name="stage" type="xs:string" />
   <xs:element name="comment" type="xs:string" />
   <xs:attribute name="patID" type="xs:ID" />
   <xs:attribute name="onStudy" type="xs:string" />
   <xs:attribute name="scale" type="xs:string" />

   <xs:element name="performance">
      <xs:complexType>
         <xs:simpleContent>
            <xs:extension base="xs:decimal">
               <xs:attribute ref="scale" use="required" /
            </xs:extension>
         </xs:simpleContent>
      </xs:complexType>
   </xs:element>

</xs:schema>
```

A Venetian blind design

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

   <xs:element name="patients">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="patient" type="pType" minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

   <xs:complexType name="pType">
      <xs:group ref="childElements" />
      <xs:attributeGroup ref="patientAtt" />
   </xs:complexType>

   <xs:group name="childElements">
      <xs:sequence>
         <xs:element name="lastName" type="xs:string" />
         <xs:element name="firstName" type="xs:string" />
         <xs:element name="dateOfBirth" type="xs:date" />
         <xs:element name="age" type="xs:positiveInteger" />
         <xs:element name="stage" type="xs:string" />
         <xs:element name="comment" type="xs:string" minOccurs="0" maxOccurs="unbounded" />
         <xs:element name="performance" type="perfType" />
      </xs:sequence>
   </xs:group>

   <xs:attributeGroup name="patientAtt">
      <xs:attribute name="patID" type="xs:ID" />
      <xs:attribute name="onStudy" type="xs:string" />
   </xs:attributeGroup>

   <xs:complexType name="perfType">
      <xs:simpleContent>
         <xs:extension base="xs:decimal">
            <xs:attribute name="scale" type="xs:string" use="required" />
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>

</xs:schema>
```

14. Comparing Schema Structures

Feature	Russian Doll	Flat Catalog (Salami Slice)	Venetian Blind
Global and local declarations	The schema contains one single global element; all other declarations are local.	All declarations are global.	The schema contains one single global element; all other declarations are local.
Nesting of elements	Element declarations are nested within a single global element.	Element declarations are not nested.	Element declarations are nested within a single global element referencing named complex types, element groups, and attribute groups.
Reusability	Element declarations can only be used once.	Element declarations can be reused throughout the schema.	Named complex types, element groups, and attribute groups can be reused throughout the schema.
Interaction with namespaces	If a namespace is attached to the schema, only the root element needs to be qualified in the instance document.	If a namespace is attached to the schema, all elements need to be qualified in the instance document.	If a namespace is attached to the schema, only the root element needs to be qualified in the instance document.

15. Placing a Schema in a Namespace (1)

Targeting a Namespace

- To associate a schema with a namespace, you first declare the namespace and then make that namespace the target of the schema. To do this, you add the following attributes to the schema's root element:

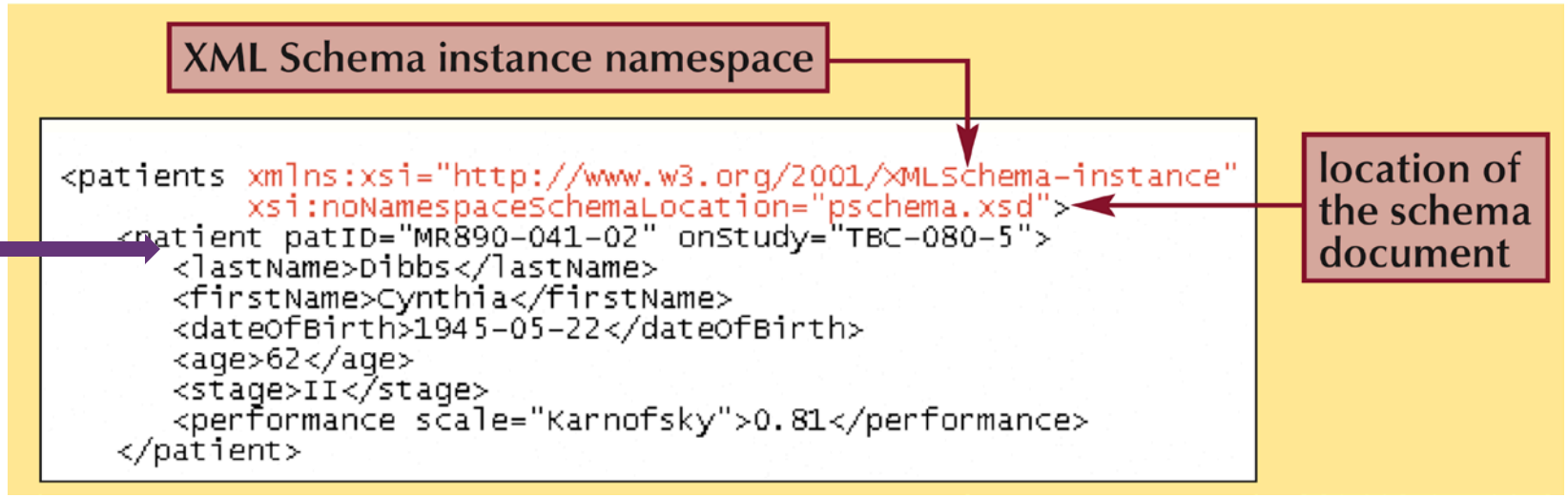
```
prefix:xmlns="uri"
targetNamespace="uri"
```

  where prefix is the prefix of the XML Schema namespace and uri is the URI of the schema's target. If no prefix is specified, XML Schema is the default namespace of the schema file.
- Namespace prefixes in the schema do not have to match the prefixes used in the instance document; only the URIs must match.
- Any customized data types, named types, elements, element groups, or attributes are placed in the target namespace.

Applying a namespace to a schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://uhosp.edu/patients/ns"
           targetNamespace="http://uhosp.edu/patients/ns">

   <xs:element name="patients">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="patient" type="pType" minOccurs="1" maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
```

15. Placing a Schema in a Namespace (2)

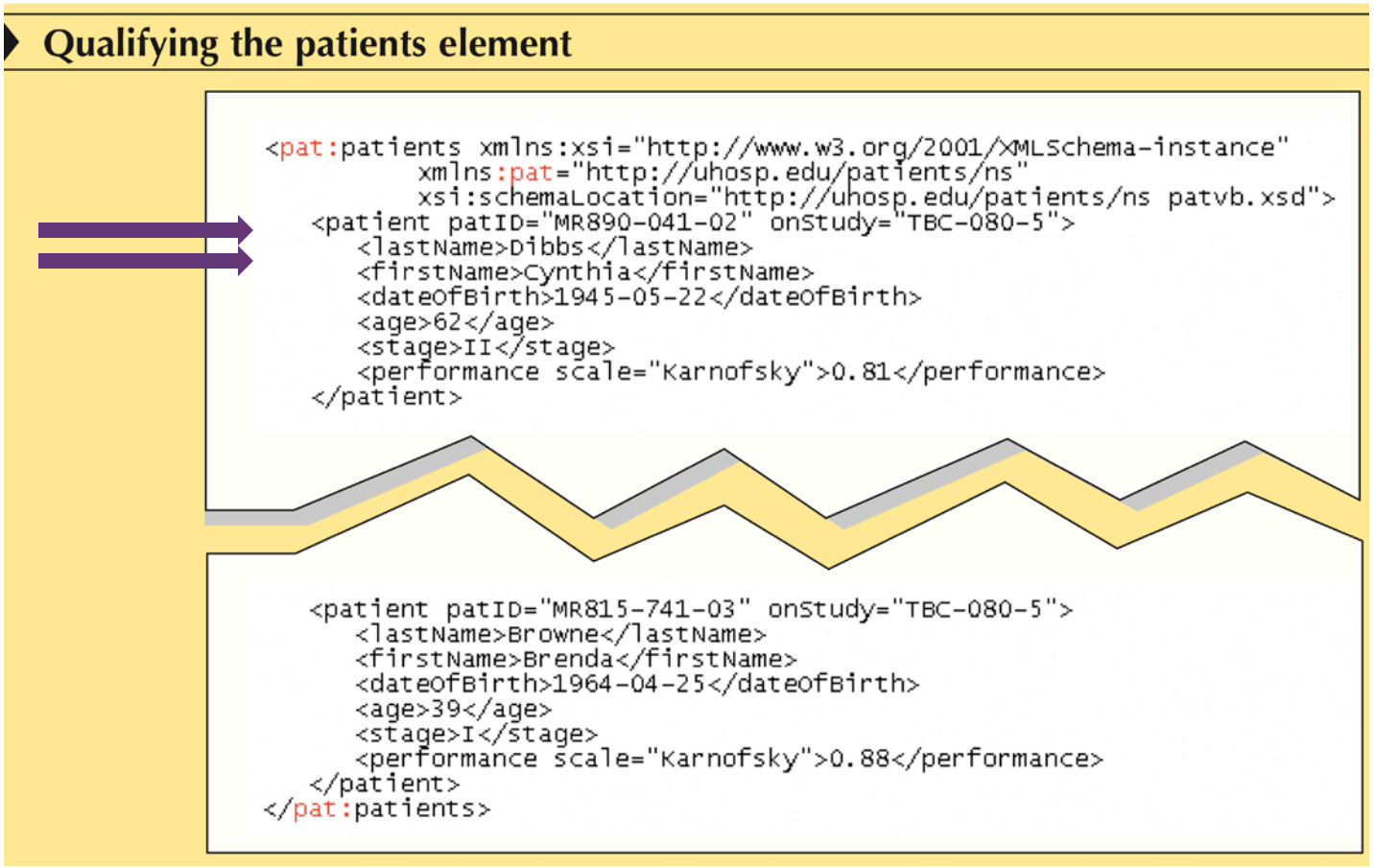


Attaching a schema namespace to the instance document

```
<patients xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns="http://uhosp.edu/patients/ns"
          xsi:schemaLocation="http://uhosp.edu/patients/ns patvb.xsd">
   <patient patID="MR890-041-02" onStudy="TBC-080-5">
      <lastName>Dibbs</lastName>
      <firstName>Cynthia</firstName>
      <dateOfBirth>1945-05-22</dateOfBirth>
      <age>62</age>
      <stage>II</stage>
      <performance scale="Karnofsky">0.81</performance>
   </patient>
```

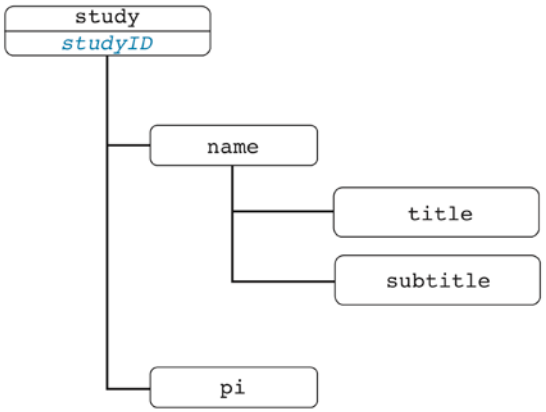
15. Placing a Schema in a Namespace (3)

Qualified and Unqualified Names



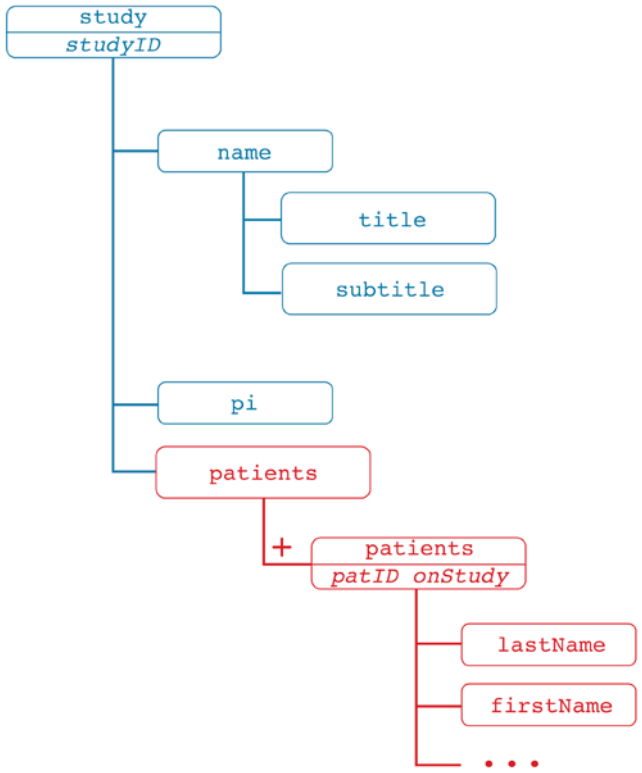
16. Validating a Compound Document (1)

study.xsd




```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://uhosp/studies/ns"
           targetNamespace="http://uhosp/studies/ns">

   <xs:element name="study">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="title" />
                     <xs:element name="subtitle" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="pi" type="xs:string" />
         </xs:sequence>
         <xs:attribute name="studyID" type="xs:ID" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```

16. Validating a Compound Document (2)



study + patient vocabularies

```
<std:study studyID="TBC-080-5"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:std="http://uhosp/studies/ns"
           xsi:schemaLocation="http://uhosp/studies/ns study.xsd">

   <name>
      <title>Tamoxifen Breast Cancer Study</title>
      <subtitle>Randomized Phase 3 Clinical Trial</subtitle>
   </name>
   <pi>Dr. Diane West</pi>

<pat:patients xmlns:pat="http://uhosp.edu/patients/ns"
              xsi:schemaLocation="http://uhosp.edu/patients/ns patvb.xsd">
   <patient patID="MR890-041-02" onStudy="TBC-080-5">
      <lastName>Dibbs</lastName>
      <firstName>Cynthia</firstName>
      <dateOfBirth>1945-05-22</dateOfBirth>
      <age>62</age>
      <stage>II</stage>
      <performance scale="Karnofsky">0.81</performance>
   </patient>

   <patient patID="MR815-741-03" onStudy="TBC-080-5">
      <lastName>Browne</lastName>
      <firstName>Brenda</firstName>
      <dateOfBirth>1964-04-25</dateOfBirth>
      <age>39</age>
      <stage>I</stage>
      <performance scale="Karnofsky">0.88</performance>
   </patient>
</pat:patients>

</std:study>
```

16. Validating a Compound Document (3)

Insert:

```
<xs:import
namespace="http://uhosp.edu
/patients/ns"
schemaLocation="patvb.xsd"
/>
```

study.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://uhosp/studies/ns"
           targetNamespace="http://uhosp/studies/ns">

   <xs:element name="study">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="title" />
                     <xs:element name="subtitle" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="pi" type="xs:string" />
         </xs:sequence>
         <xs:attribute name="studyID" type="xs:ID" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```

16. Validating a Compound Document (4)

The final **study.xsd**

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://uhosp/studies/ns"
           targetNamespace="http://uhosp/studies/ns"
           xmlns:pat="http://uhosp.edu/patients/ns">

<xs:import namespace="http://uhosp.edu/patients/ns"
           schemaLocation="patvb.xsd" />

   <xs:element name="study">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="name">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="title" />
                     <xs:element name="subtitle" />
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="pi" type="xs:string" />
            <xs:element ref="pat:patients" />
         </xs:sequence>
         <xs:attribute name="studyID" type="xs:ID" />
      </xs:complexType>
   </xs:element>

</xs:schema>
```