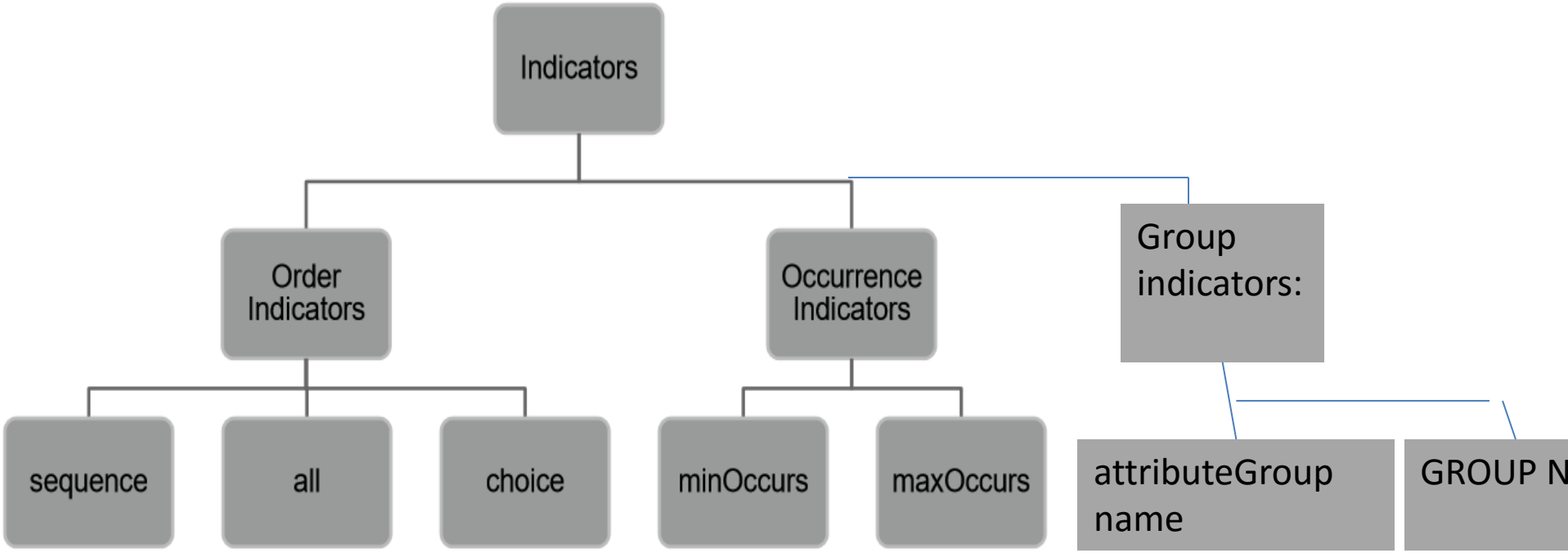
Cont…

Complex Elements



- Indicators

  Sequence indicator :

  Ensures that all the sub elements are defined and they are defined in the same order as given in the XSD

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

Choice indicator :

defines that either one of the child element must occur within the element

- Xsd:

```
<xs:element name="employee">
        <xs:complexType>
            <xs:choice>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="age" type="xs:integer"/>
            </xs:choice>
        </xs:complexType>
</xs:element>
```

XML:

```
<employee>
<name> Johan </name>
</employee>
OR
<employee>
<age> 28 </age>
</employee>
```

The <all> indicator :

specifies that the child elements can appear in any order, and that each child element must occur only once:

```
<xs:element name="employee">
    <xs:complexType>
        <xs:all>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="age" type="xs:integer"/>
        </xs:all>
    </xs:complexType>
</xs:element>
```

Ex. Xml:

```
<employee>
<name> Tom </name>
<age> 28 </age>
</employee>
```

Note: When using the <all> indicator you can set the <minOccurs> indicator to 0 or 1 and the <maxOccurs> indicator can only be set to 1 (the <minOccurs> and <maxOccurs> are described later).

Occurrence indicator: defines the number of times an element can occur

```
<xs:element name="employee">
        <xs:complexType>
         <xs:sequence>
              <xs:element name="name" type="xs:string"/>
              <xs:element name="childname" type="xs:string"
             minOccurs=”0” maxOccurs="5"/>
         </xs:sequence>
         </xs:complexType>
 </xs:element>
```

To allow an element to appear an unlimited number of times, use the maxOccurs="unbounded" statement:

- **Group Indicators**
- Group indicators are used to define related sets of elements.
- **Element Groups**
- Element groups are defined with the group declaration, like this:
- <xs:group name="groupname">
  ...
  </xs:group>

You must define an all, choice, or sequence element inside the group declaration.

After you have defined a group, you can reference it in another definition, like this:

```
<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>
```

- **Attribute Groups**

After you have defined an attribute group, you can reference it in another definition, like this:

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="firstname" type="xs:string"/>
  <xs:attribute name="lastname" type="xs:string"/>
  <xs:attribute name="birthday" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="person">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

Ques.

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

<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>
    <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>
```

Associating XML with XSD

Example

- Define an XSD to create an XML file which contains employee's information like name, department, salary and email.

- There can be many employee details present in the XML file)

employee.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
<xs:element name="employees">
<xs:complexType>
<xs:sequence>
<xs:element name="employee" minOccurs="1" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="name" type="xs:string"></xs:element>
<xs:element name="department" type="xs:string"></xs:element>
<xs:element name="salary" type="xs:float"></xs:element>
<xs:element name="email" type="xs:string"></xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:positiveInteger"></xs:attribute>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Associating XML with XSD

```
<?xml version="1.0"?>
<employees xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="employee.xsd">
<employee id="101">
<name> Tom </name>
<department> CSA </department>
<salary> 35000 </salary>
<email> tom.peter@gmail.com</email>
</employee>
<employee id="102">
<name>Sam</name>
<department>AC</department>
<salary>45000</salary>
<email>sam.johan@gmail.com</email>
</employee>
</employees>
```

Dividing the XML schema

The previous XML Schema is very simple

But it becomes very difficult to read it, and maintain the XML document.

- Avoid this by dividing the XML Schema as –

define the elements and attributes first and then

–make use of them using the “ref” keyword.

Dividing

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema" >
<!-- defining simple elements -->
<xs:element name="name"
type="xs:string"/>
<xs:element name="department"
type="xs:string"/>
<xs:element name="salary"
type="xs:float"/>
<xs:element name="email"
type="xs:string"/>
<!-- defining attributes -->
<xs:attribute name="id"
type="xs:positiveInteger"/>
<!-- defining Complex element -->
<xs:element name="employee">
<xs:complexType>
<xs:sequence>
<xs:element ref="name"/>
<xs:element ref="department" />
<xs:element ref="salary" />
<xs:element ref="email" />
</xs:sequence>
<xs:attribute ref="id"></xs:attribute>
</xs:complexType>
</xs:element>
<xs:element name="employees">
<xs:complexType>
<xs:sequence>
<xs:element ref="employee"
minOccurs="1"
maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

the XML schema

- **Using Named Types**
- This design method defines types, that enables you to reuse element definitions.
- This is done by giving names to the simpleTypes and complexTypes elements
- Then make them point through the type attribute of the element.

Using Named Types

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
<xs:simpleType name="stringtype">
<xs:restriction base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="floattype">
<xs:restriction base="xs:float">
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="idtype">
<xs:restriction base="xs:positiveInteger">
<xs:pattern value="[0-9]{3}"></xs:pattern>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="emailtype">
<xs:restriction base="xs:string">
<xs:maxLength value="30"/>
</xs:restriction>
</xs:simpleType>
```

Using Named Types

```
 <xs:complexType name="employeetype">
<xs:sequence>
<xs:element name="name" type="stringtype"/>
<xs:element name="department" type="stringtype"/>
<xs:element name="salary" type="floattype"/>
<xs:element name="email" type="emailtype"/>
</xs:sequence>
<xs:attribute name="id" type="idtype"/>
</xs:complexType>
<xs:complexType name="employeestype">
<xs:sequence>
<xs:element name="employee" type="employeetype"
maxOccurs="unbounded" minOccurs="1"/>
</xs:sequence>
</xs:complexType>
<xs:element name="employees" type="employeestype"/>
</xs:schema>
```

```
1 <?xml version = "1.0"?>
23
<!-- Fig. 7.21 : planner.xml -->
4 <!-- Day Planner XML document -->
56
<planner xmlns = "x-schema:planner-
schema.xml">
7 <year value = "2000">
8 <date month = "7" day = "15">
9 <note time = "1430">Doctor&apos;s
appointment</note>
10 <note time = "1620">Physics class at
BH291C</note>
11 </date>
12
13 <date month = "7" day = "4">
14 <note>Independance Day</note>
15 </date>
17 <date month = "7" day = "20">
18 <note time = "0900">General Meeting
room 32-A</note>
19 </date>
20
21 <date month = "7" day = "20">
22 <note time = "1900">Party at
Joe&apos;s</note>
23 </date>
24
25 <date month = "7" day = "20">
26 <note time = "1300">Financial Meeting
room 14-C</note>
27 </date>
28 </year>
29 </planner>
```

Fig. 7.21 Day planner XML document that conforms to Fig. 7.20 (part 1 of 2).

XSD Data Types

1. String Data Types:

<xs:string> Example

- Element declaration in xsd –
- <xs:element name = "name" type = "xs:string"/> Element usage in xml –
- <name>Dinkar</name> <name>Dinkar Kad</name>

XSD - Date Time

Date and Time data types are used to represent date and time in the XML documents.

The <xs:date> data type is used to represent date in YYYY-MM-DD format

<xs:date> Example

- Element declaration in XSD –
- <xs:element name = "birthdate" type = "xs:date"/> Element usage in XML –
- <birthdate>1980-03-23</birthdate>

<xs:time> data type

- The <xs:time> data type is used to represent time in hh:mm:ss format.

<xs:time> Example

Element declaration in XSD −

<xs:element name = "startTime" type = "xs:time"/>

Element usage in XML −

<startTime>10:20:15</startTime>

<xs:datetime> data type

- The <xs:datetime> data type is used to represent date and time in YYYY-MM-DDThh:mm:ss format.

<xs:datetime> Example

- Element declaration in XSD −
- <xs:element name = "startTime" type = "xs:datetime"/> Element usage in XML −
- <startTime>1980-03-23T10:20:15</startTime>

YYYY − represents year

MM − represents month

DD − represents day

T − represents start of time section

hh − represents hours

mm − represents minutes

ss − represents seconds

<xs:duration> data type

- The <xs:duration> data type is used to represent time interval in PnYnMnDTnHnMnS format. Each component is optional except P.
- **P** – represents start of date section
- **nY** – represents year
- **nM** – represents month
- **nD** – represents day
- **T** – represents start of time section
- **nH** – represents hours
- **nM** – represents minutes
- **nS** – represents seconds

- **<xs:duration> Example**
- Element declaration in XSD −
- <xs:element name = "period" type = "xs:duration"/> Element usage in xml to represent period of 6 years, 3 months, 10 days and 15 hours.
- <period>P6Y3M10DT15H</period>

Date Data Types

- Following is the list of commonly used date data types.

Name & Description	
Date :	**Represents a date value**
dateTime:	**Represents a date and time value**
Duration:	**Represents a time interval**
gDay:	**Represents a part of a date as the day (DD)**
gMonth:	**Represents a part of a date as the month (MM)**
gMonthDay:	**Represents a part of a date as the month and day (MM-DD)**
gYear:	**Represents a part of a date as the year (YYYY)**
gYearMonth:	**Represents a part of a date as the year and month (YYYY-MM)**
Time:	**Represents a time value**

Restrictions

- Following types of restrictions can be used with Date data types –
- enumeration
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- whiteSpace

XSD - Numeric Data Types

Numeric data types are used to represent numbers in XML documents.

<xs:decimal> data type

- The <xs:decimal> data type is used to represent numeric values. It supports decimal numbers up to 18 digits.
- **<xs:decimal> Example**
- Element declaration in XSD –
- <xs:element name = "score" type = "xs:decimal"/> Element usage in XML –
- <score>9.12</score>

Numeric Data Types

Following is the list of commonly used numeric data types.

- **byte** A signed 8 bit integer
- **decimal** A decimal value
- **int** A signed 32 bit integer
- **integer** An integer value
- **long** A signed 64 bit integer
- **negativeInteger** An integer having only negative values (..,-2,-1)
- **nonNegativeInteger** An integer having only non-negative values (0,1,2,..)
- **nonPositiveInteger** An integer having only non-positive values (..,-2,-1,0)
- **positiveInteger** An integer having only positive values (1,2,..)
- **short** A signed 16 bit integer
- **unsignedLong** An unsigned 64 bit integer
- **unsignedInt** An unsigned 32 bit integer
- **unsignedShort** An unsigned 16 bit integer
- **unsignedByte** An unsigned 8 bit integer

Restrictions

Following types of restrictions can be used with Date data types −

- enumeration
- fractionDigits
- maxExclusive
- maxInclusive
- minExclusive
- minInclusive
- pattern
- totalDigits
- whiteSpace

XSD has a few other important data types, such as **Boolean, binary,** and **anyURI.**

The <xs:boolean> data type is used to represent true, false, 1 (for true) or 0 (for false) value.

Binary data types

The Binary data types are used to represent binary values. Two binary types are common in use.

- **base64Binary** − represents base64 encoded binary data
- **hexBinary** − represents hexadecimal encoded binary data

<xs:hexbinary> Example

Element declaration in XSD −

<xs:element name = "blob" type = "xs:hexBinary"/> Element usage in XML −

<blob>9FEEF</blob>

<xs:anyURI> data type

- The <xs:anyURI> data type is used to represent URI.

<xs:anyURI> Example

- Element declaration in XSD –
- <xs:attribute name = "resource" type = "xs:anyURI"/> Element usage in XML –
- <image resource = "http://www.xyz/images/smiley.jpg" />

Name & Description

byte: A signed 8 bit integer

decimal: A decimal value

int: A signed 32 bit integer

integer: An integer value

long: A signed 64 bit integer

negativeInteger: An integer having only negative values (..,-2,-1)

nonNegativeInteger: An integer having only non-negative values (0,1,2,..)

nonPositiveInteger: An integer having only non-positive values (..,-2,-1,0)

positiveInteger: An integer having only positive values (1,2,..)

Short: A signed 16 bit integer

unsignedLong: An unsigned 64 bit integer

unsignedInt: An unsigned 32 bit integer

unsignedShort: An unsigned 16 bit integer

unsignedByte: An unsigned 8 bit integer

- **Restrictions**
- Following types of restrictions can be used with Miscellaneous data types except on boolean data type –
- enumeration
- length
- maxLength
- minLength
- pattern
- whiteSpace