

UNIT – 3 XML Schema

- Introduction to Schema
- Features
- DTD versus XML Schema
- XML Schema Type System
- Simple types
- Complex Types
- Grouping of Data
- Deriving Types
- Attributes

Introduction to Schema

- XML Schema is commonly known as XML Schema Definition (XSD).
- It is used to describe and validate the structure and the content of XML data.
- XML schema defines the elements, attributes and data types.
- Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Introduction to Schema



Attributes in the *<schema>* Element

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.test.com"
xmlns="http://www.xyz.com"
elementFormDefault="qualified">
...
...
</xs:schema>
```

- ***xmlns:xs="http://www.w3.org/2001/XMLSchema"*** indicates that the data types and elements used inside this schema come from this namespace and they should be prefixed with *xs:*
- ***targetNamespace="http://www.test.com"*** indicates that the user-defined XML elements, attributes etc come from this schema
- ***xmlns="http://www.xyz.com"*** is the default namespace
- ***elementFormDefault="qualified"*** mandates usage of namespaces for all elements in the current XML document

Introduction to Schema

Referencing a Schema from an XML File



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

```
<note xmlns="http://www.xyz.com" ← Default namespace  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.xyz.com note.xsd">
```

Namespace and schema file name

```
<to>Ram</to>  
<from>Krishna</from>  
<heading>Reminder</heading>  
<body>Please send me your XML notes!</body>  
</note>
```

Schema vs DTD

Schema	DTD
Schema document is an XML document i.e., the structure of an XML document is specified by another XML document	DTDs follow SGML syntax
supports variety of data types similar to programming language	In DTD everything is treated as text
creating relationship among elements is possible	This is not possible in DTD without invalidating existing documents
Grouping of elements and attributes are possible to form a single logical unit	Grouping of elements and attributes is not possible in DTD
it is possible to specify an upper limit for the number of occurrences of an element	It is not possible to specify an upper limit of an element in DTDs

Basic Example :

- Example 1 : Write an XML document that contains a single element to specify the name of the student. Provide a corresponding XML Schema.

(Student.xml)

```
<?xml version="1.0"?>
<Student xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Student.xsd">
  Harsh
</Student>
```

(Student.xsd)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Student" type="xsd:string"></xsd:element>
</xsd:schema>
```

XML Schema Type System

- Elements in schema can be divided into 2 categories :

1. Simple Elements :

- Simple elements can contain only text.
- They cannot have sub-elements or attributes.
- The text that they can contain, however , can be of various data types such as strings, numbers , dates etc.

2. Complex Elements :

- Complex elements, on the other hand, can contain sub-elements, attributes etc.
- Many times, they are made up of one or more simple element.

Let us understand the difference between the two types.

- Simple elements: Simple elements can contain only text. They cannot have sub-elements or attributes. The text that they can contain, however, can be of various data types such as strings, numbers, dates, etc.
- Complex elements: Complex elements, on the other hand, can contain sub-elements, attributes, etc. Many times, they are made up of one or more simple element. This is shown in Figure 4.6.

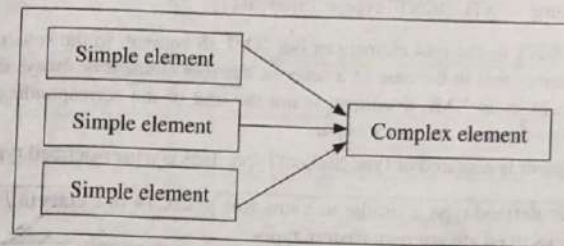


Figure 4.6 Complex element is made up of simple elements

Let us now consider an example. Suppose we want to capture student information in the form of the student's roll number, name, marks, and result. We can have all these individual blocks of information as simple elements. Then, we will have a complex element in the form of the root element. This complex element will encapsulate these individual simple elements. Figure 4.7 shows the resulting XML document, first.

```

<?xml version = "1.0"?>
<STUDENT xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="student.xsd">
  <ROLL_NUMBER> 100 </ROLL_NUMBER>
  <NAME> Pallavi Joshi </NAME>
  <MARKS> 80 </MARKS>
  <RESULT> Distinction </RESULT>
</STUDENT>
  
```

Figure 4.7 XML document for Student example

Let us now immediately take a look at the corresponding schema file. Figure 4.8 shows this.

```

<?xml version = "1.0"?>
<xsd:schema xmlns:xsd = "http://www.w3org/2001/XMLSchema">
  <xsd:element name = "STUDENT" type = "StudentType"/>
  <xsd:complexType name = "StudentType">
    <xsd:sequence>
      <xsd:element name = "ROLL_NUMBER" type = "xsd:string"/>
      <xsd:element name = "NAME" type = "xsd:string"/>
      <xsd:element name = "MARKS" type = "xsd:integer"/>
      <xsd:element name = "RESULT" type = "xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
  
```

Figure 4.8 Schema for Student example

Let us understand our schema.

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

We know that the root element of the schema is a reserved keyword called as *schema*. Here too, it is the same. The namespace prefix *xsd* maps to the namespace URI *http://www.w3.org/2001/XMLSchema*, as earlier. In general, this will be true for any schema that we write.

2. `<xsd:element name = "STUDENT" type = "StudentType"/>`

This declares *STUDENT* as the root element of our XML document. In the schema, it is called as the top-level element. Remember that in the case of a schema, the root element is always the keyword *schema*. Therefore, the root element in an XML document is not the root of the corresponding schema. Instead, it appears in the schema after the root element *schema*.

The *STUDENT* element is declared of type *StudentType*. This is a user-defined type.

Conceptually, a user-defined type is similar to a structure in C/C++ or a class in Java (without the methods). It allows us to create our own custom types.

In other words, the schema specification allows us to create our own custom data types. For example, we can create our own types for storing information about employees, departments, songs, friends, sports games, and so on. We recognise this as a user-defined type because it does not have our namespace prefix *xsd*. Remember that all the standard data types provided by the XML schema specifications reside at the namespace *http://www.w3.org/2001/XMLSchema*, which we have prefixed as *xsd* in the earlier statement.

3. `<xsd:complexType name = "StudentType">`

Now that we have declared our own type, we must explain what it represents and contains. That is exactly what we are doing here. This statement indicates that we have used *StudentType* as a type earlier, and now we want to explain what it means. Also, note that we use a keyword *complexType* to designate that *StudentType* is a complex element. This is similar to stating struct *StudentType* or class *StudentType* in C++/Java.

4. `<xsd:sequence>`

Schemas allow us to force a sequence of simple elements within a complex element. We can specify that a particular complex element must contain one or more simple element in a strict sequence. Thus, if the complex element is A, containing two simple elements B and C, we can mandate that C must follow B inside A. In other words, the XML document must have:

```
<A>
  <B> ... </B>
  <C> ... </C>
</A>
```

This is accomplished by the sequence keyword.

5. `<xsd:element name = "ROLL_NUMBER" type = "xsd:string"/>`

This declaration specifies that the first simple element inside our complex element is *ROLL_NUMBER*, of type *string*. After this, we have *NAME*, *MARKS*, and *RESULT* as three more simple elements following *ROLL_NUMBER*. We will not discuss them. We will simply observe for now that *ROLL_NUMBER* has a different data type: an integer. We will discuss this in detail subsequently.

We will also not discuss the closure of the sequence, *ComplexType*, and schema tags.

Let us have a small exercise to build on these concepts.

Exercise 1

Write an XML document and a corresponding XML schema for maintaining the employee number, name, designation, and salary.

Solution**XML document (employee.xml)**

```
<?xml version = "1.0" ?>
<EMPLOYEE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="employee.xsd">
  <EMP_NO> 9662 </EMP_NO>
  <EMP_NAME> Atul Kahate </EMP_NAME>
  <DESIGNATION> Consultant </DESIGNATION>
  <SALARY> 1000 </SALARY>
</EMPLOYEE>
```

XML schema (employee.xsd)

```
<?xml version = "1.0"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <xsd:element name = "EMPLOYEE" type = "EmpType"/>

  <xsd:complexType name = "EmpType">
    <xsd:sequence>
      <xsd:element name = "EMP_NUMBER" type =
"xsd:integer"/>
      <xsd:element name = "EMP_NAME" type =
"xsd:string"/>
      <xsd:element name = "DESIGNATION" type =
"xsd:integer"/>
      <xsd:element name = "SALARY" type =
"xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

4.2.2 Specifying the Frequency: minOccurs and maxOccurs ✓ 6

Let us consider that we want to represent information about a book. The XML document depicting this information along with its corresponding schema is shown in Figure 4.9.

XML document (book.xml)

```
<?xml version = "1.0" ?>
<BOOK xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="book.xsd">
  <TITLE> Cryptography and Network Security </TITLE>
  <AUTHOR> Atul Kahate </AUTHOR>
  <PUBLISHER> TMH </PUBLISHER>
  <PRICE> 250 </PRICE>
</BOOK>
```

XML schema (book.xsd)

```
<?xml version = "1.0"?>
<xsd:schema xmlns:xsd = "http://www.w3org/2001/XMLSchema">
  <xsd:element name = "BOOK" type = "BookType"/>

  <xsd:complexType name = "BookType">
    <xsd:sequence>
      <xsd:element name = "TITLE" type = "xsd:string"/>
      <xsd:element name = "AUTHOR" type = "xsd:string"/>
      <xsd:element name = "PUBLISHER" type = "xsd:integer"/>
      <xsd:element name = "PRICE" type = "xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Figure 4.9 Book XML and schema

There is no problem with this example. However, now imagine a situation where we want to provide support for a book XML document that can have multiple authors. Would the same schema serve the purpose? Figure 4.10 shows this situation. Please note that the schema declaration in this figure is incorrect. We have shown it merely to explain the problem.

XML document (book.xml)

```

<?xml version = "1.0" ?>
<BOOK xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="book.xsd">
  <TITLE> Web Technologies </TITLE>
  <AUTHOR> Achyut Godbole </AUTHOR>
  <AUTHOR> Atul Kahate </AUTHOR>
  <PUBLISHER> TMH </PUBLISHER>
  <PRICE> 250 </PRICE>
</BOOK>

```

XML schema (book.xsd)

```

<?xml version = "1.0"?>
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema">
  <xsd:element name = "BOOK" type = "BookType"/>

  <xsd:complexType name = "BookType">
    <xsd:sequence>
      <xsd:element name = "TITLE" type = "xsd:string"/>
      <xsd:element name = "AUTHOR" type = "xsd:string"/>
      <xsd:element name = "PUBLISHER" type = "xsd:integer"/>
      <xsd:element name = "PRICE" type = "xsd:integer"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Figure 4.10 Book XML and incorrect schema

We now have two authors in the XML document. However, the corresponding schema talks about the author element only once. This is not legal in XML. We must use either or both of the minOccurs and maxOccurs attributes in such situations.

The minOccurs attribute specifies the minimum number of occurrences that an element can have.

On the other hand, the maxOccurs attribute specifies the maximum number of occurrences.

Our requirement is to have two authors in this case. Therefore, we only require maxOccurs with a value of 2. Therefore, the declaration of the AUTHOR element in our schema would change to the following:

```

<xsd:element name = "AUTHOR" type = "xsd:string"
  maxOccurs = "2"/>

```

Nothing else needs to change. The above declaration specifies that we can at the most have two authors.

The default value of both minOccurs and maxOccurs is 1. Therefore, if we do not specify either of them, the element is deemed to occur exactly once.

The above declaration is equivalent to the following:

```
<xsd:element name = "AUTHOR"           type = "xsd:string"
              minOccurs = "1"           maxOccurs = "2"/>
```

There is a specific value called as unbounded, which means infinite occurrences. Whenever we wish to specify that the upper limit for an element occurrence is infinite (that is, there is no upper limit), we can specify it as unbounded. For example, if our book can have a minimum of one author or an infinite number of authors, our declaration would change to:

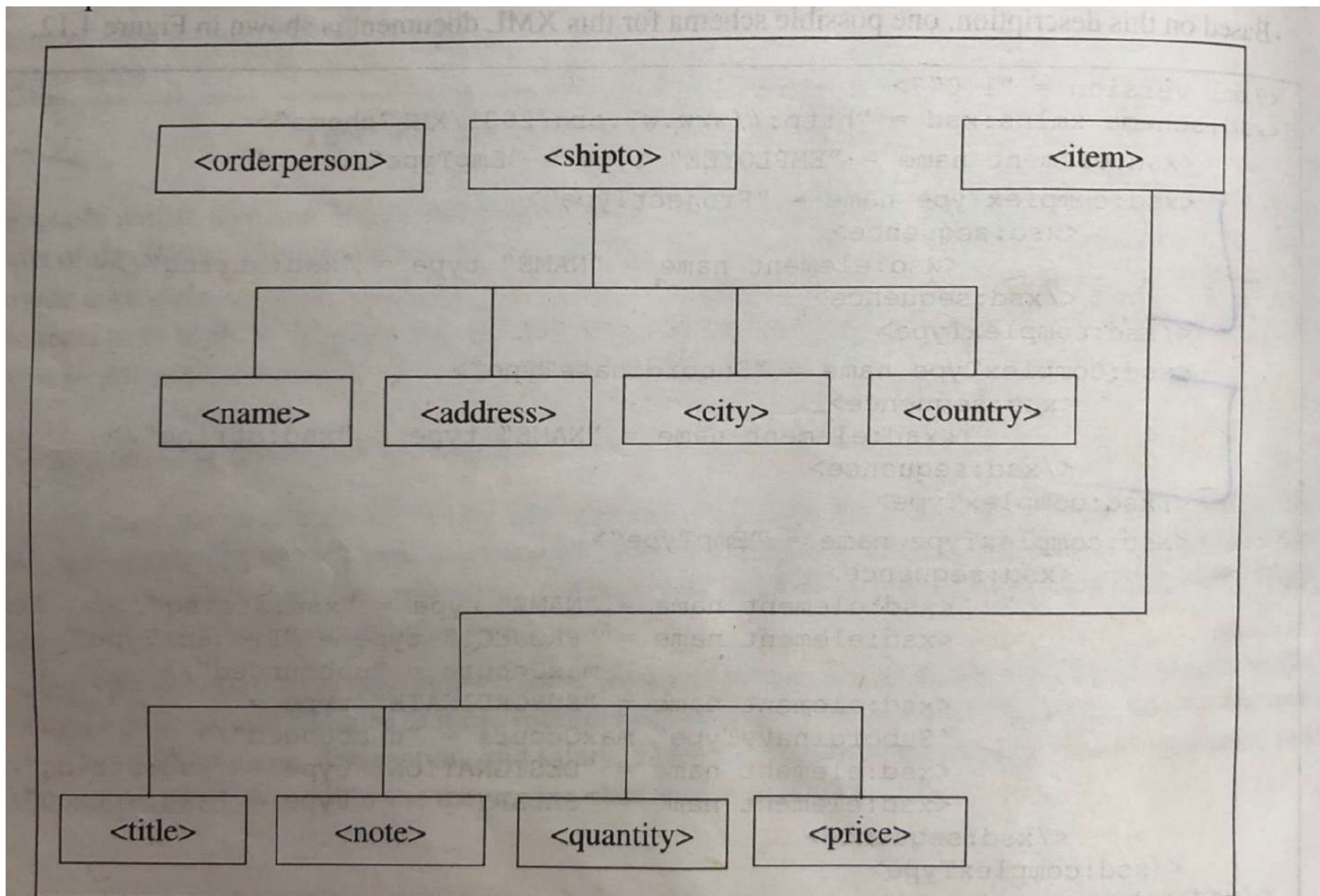
```
<xsd:element name = "AUTHOR"           type = "xsd:string"
              minOccurs = "1"           maxOccurs = "unbounded"/>
```

Based on our requirements, we can set minOccurs and maxOccurs attributes to various values. These are summarised in Table 4.1.

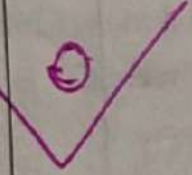
Requirement	Set minOccurs to	Set maxOccurs to
An element should <u>occur exactly once</u>	1	1
An element should <u>occur at least once</u> and <u>possibly many more times</u>	1	Unbounded
An element is <u>optional</u> (may not occur at all), or may <u>occur for any number of times</u>	0	Unbounded
An element may <u>not occur at all</u> , or may <u>occur only once</u>	0	1

Table 4.1 Usage of minOccurs and maxOccurs

We would realize that minOccurs and maxOccurs are similar to, but far more effective than the ?, *, and + symbols of the DTDs. The minOccurs and maxOccurs are not only easier to read and understand, but they also provide a lot more accurate precision. For example, suppose that one manager can manage any number of employees, from eight to 20. Then we can specify minOccurs as eight and maxOccurs as 20. There is no such accurate precision available in the case of DTD declarations.




```
<?xml version="1.0"?>
<shiporder xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation="shiporder.xsd"
            orderid="88923">
    <orderperson>Narendra Limaye</orderperson>
    <shipto>
        <name>Rahul Khatri</name>
        <address>Wakad Phata</address>
        <city>Pune</city>
        <country>India</country>
    </shipto>
    <item>
        <title>TCP/IP</title>
        <note>Special discount</note>
        <quantity>1</quantity>
        <price>200</price>
    </item>
    <item>
        <title>C++</title>
        <quantity>1</quantity>
        <price>300</price>
    </item>
</shiporder>
```



```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="shiporder">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="orderperson" type="xsd:string"/>
        <xsd:element name="shipto">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="name" type="xsd:string"/>
              <xsd:element name="address" type="xsd:string"/>
              <xsd:element name="city" type="xsd:string"/>
              <xsd:element name="country" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="item" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="title" type="xsd:string"/>
              <xsd:element name="note" type="xsd:string"
                minOccurs="0"/>
              <xsd:element name="quantity" type="xsd:Integer"/>
              <xsd:element name="price" type="xsd:decimal"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute name="orderid" type="xsd:string" use="required"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```



```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="EMPLOYEE" type="EmpType"/>
  <xsd:complexType name="EmpType" mixed="true">
    <xsd:sequence>
      <xsd:element name="FIRST_NAME" type="xsd:string"/>
      <xsd:element name="LAST_NAME" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Figure 4.23 Schema with mixed content (emp2.xsd)

Note that we have dropped the TITLE and the MIDDLE_NAME elements. Also, we have added an attribute *mixed* with a value of *true* for the *EmpType* complex type. As a result, we cannot use the TITLE or the MIDDLE_NAME elements in our XML document. However, we can still specify the values for the title and the middle name elements as placeholders as shown in Figure 4.24. This is what mixed content allows us to do.

```

<?xml version = "1.0"?>
<EMPLOYEE xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation = "emp2.xsd">
  Mr <FIRST_NAME> Atul</FIRST_NAME> S. <LAST_NAME>Kahate</LAST_NAME>
</EMPLOYEE>

```

Figure 4.24 XML document with mixed content (emp2.xml)

4.3 GROUPING OF DATA

Thus far, we have been dealing with schemas that mandate a specific order of elements inside an XML document. That is, we have seen cases where element A must follow element B inside a document. In reality, this is always not the case. At times, we just want to make sure that an element exists inside an XML document – where, is not so important. This is where grouping of elements comes into picture.

The schema syntax provides support for three grouping constructs that also govern the sequence of elements inside an XML document. These three constructs are:

The `xsd:all` grouping specifies that all the elements in a group must occur at the most once, but their ordering is not significant.

The `xsd:choice` grouping allows us to specify that only one element from the group can appear. Alternatively, we can also specify that out of n elements in a group, m should appear in any order.

The `xsd:sequence` grouping mandates that every element in a group must appear exactly once, and also in the same order in which the elements are listed.

Let us discuss these now.

4.3.1 Mandating All Elements

When we use `xsd:all`, we mean that an element may occur. If it occurs, it must occur only once. The order of elements is not significant.

Consider the example shown in Figure 4.25.

```
<xsd:complexType name = "EmpType">
  <xsd:sequence>
    <xsd:element name = "NAME">
      <xsd:complexType>
        <xsd:all>
          <xsd:element name = "FIRST_NAME" type = "xsd:string"
            minOccurs = "1" maxOccurs = "1"/>
          <xsd:element name = "LAST_NAME" type = "xsd:string"
            minOccurs = "1" maxOccurs = "1"/>
        </xsd:all>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>
```

Figure 4.25 Usage of `all`

In our example, the complex type `NAME` contains elements `FIRST_NAME` and `LAST_NAME`. Both `FIRST_NAME` and `LAST_NAME` must occur exactly once. Their order is not significant. This is because they are contained inside the `<xsd:all>` tag.

We must mention that we can also specify a value of zero for `minOccurs` or `maxOccurs`. That is, we can allow an element to not occur at all. In that sense, *all* is a misnomer. For instance, we can change the declaration of `FIRST_NAME` to the following:

```
<xsd:element name = "FIRST_NAME" type = "xsd:string"
  minOccurs = "0" maxOccurs = "1"/>
```

We now allow `FIRST_NAME` to be missing from the XML document. This is perfectly all right.

We also need to note that we cannot specify an arbitrary number of occurrences in the case of *all*. That is, both `minOccurs` and `maxOccurs` can have only a value of zero or one. We cannot, for instance, say `minOccurs = 2` and `maxOccurs = 4`. This is illegal.

Exercise

Write an XML schema and show the corresponding XML document for the following: It should contain information about a credit card so that the credit card can be validated.

Solution

XML schema (card.xsd)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CreditCard">
    <xsd:complexType>
      <xsd:all>
        <xsd:element name="CardType"
          type="xsd:string"/>
        <xsd:element name="CardNumber"
          type="xsd:string"/>
        <xsd:element name="CardHolder"
          type="xsd:string"/>
        <xsd:element name="CardValidTill"
          type="xsd:string"/>
      </xsd:all>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML document (card.xml)

```
<?xml version = "1.0"?>
<CreditCard xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation = "card.xsd">
  <CardHolder>Sonia Kapoor </CardHolder>
  <CardType> Visa </CardType>
  <CardValidTill>Feb-2010</CardValidTill>
  <CardNumber> 1234567890123456 </CardNumber>
</CreditCard>
```

Note: The order of elements in the XML document is different from the once specified in the schema. As we have mentioned earlier, this is allowed in the case of all.

4.3.2 Making Choices

We know that in the case of a DTD, we can use the pipe (|) symbol to signify selection. In schema, the corresponding functionality is achieved by using the `xsd:choice` syntax. When we embed more than one element inside a *choice* boundary, exactly one of them must occur in the XML document.

For example, suppose that we want to store the information about the result of examination as Pass or Fail along with the percentage of marks obtained. Clearly, only one of these should be allowed. We can make use of the *choice* element, as shown in Figure 4.26.

Schema file (result.xsd)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Result">
    <xsd:complexType>
      <xsd:choice>
        <xsd:element name="Pass" type="xsd:string"/>
        <xsd:element name="Fail" type="xsd:string"/>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML document (result.xml)

```
<?xml version="1.0"?>
<Result xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="result.xsd">
  <Pass>75%</Pass>
</Result>
```

Schema file (result1.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="StudentExamResult">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="RollNumber" type="xsd:integer"/>
        <xsd:element name="StudentName" type="xsd:string"/>
        <xsd:element name="TotalMarks" type="xsd:integer"/>
        <xsd:element name="Result">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="Pass"
                type="xsd:string"/>
              <xsd:element name="Fail"
                type="xsd:string"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

XML document (result1.xml)

```
<?xml version="1.0"?>
<StudentExamResult xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="result1.xsd">
  <RollNumber>100</RollNumber>
  <StudentName>Abhiroop Sharma</StudentName>
  <TotalMarks>545</TotalMarks>
  <Result>
    <Pass>75%</Pass>
  </Result>
</StudentExamResult>
```


Exercise

Write an XML schema and show the corresponding XML document for storing information about lunch. It should consist of a starter, a main course, and a dessert. There should be options in each of the categories.

Solution

XML schema (lunch.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Lunch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="Starter">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="Soup"
                type="xsd:string"/>
              <xsd:element name="Juice"
                type="xsd:string"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="MainCourse">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="VegLunch"
                type="xsd:string"/>
              <xsd:element name="NonVegLunch"
                type="xsd:string"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="Dessert">
          <xsd:complexType>
            <xsd:choice>
              <xsd:element name="IceCream"
                type="xsd:string"/>
              <xsd:element name="FruitSalad"
                type="xsd:string"/>
            </xsd:choice>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

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

XML document (lunch.xml)

```
<?xml version="1.0"?>
<Lunch xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="lunch.xsd">
    <Starter>
        <Juice>Apple</Juice>
    </Starter>
    <MainCourse>
        <VegLunch>Thali</VegLunch>
    </MainCourse>
    <Dessert>
        <IceCream>Vanilla</IceCream>
    </Dessert>
</Lunch>
```