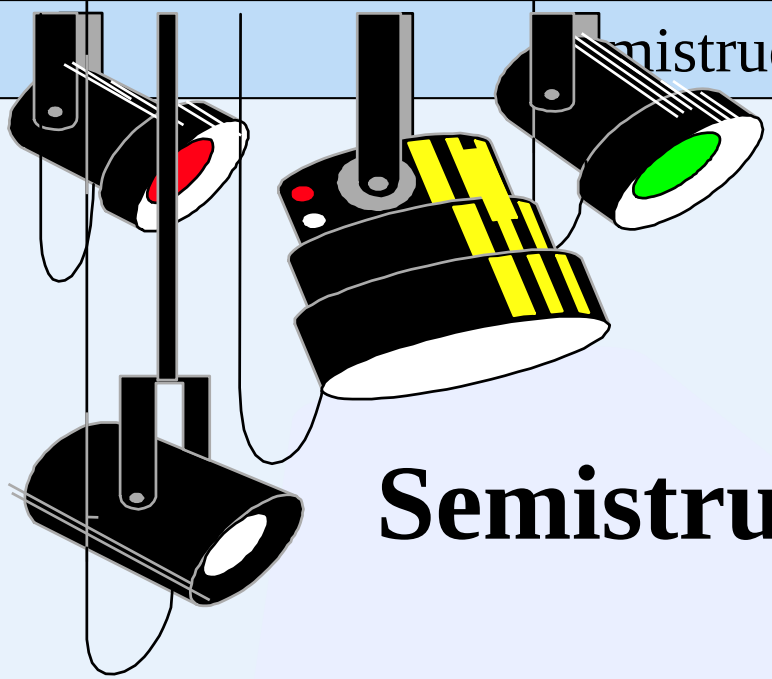


Semistructured-Data Model

- Semistructured data
- XML
- DTD (Document type definitions)
- XML schema

Semistructured Data

The semistructured-data model plays a special role in database systems:

1. It serves as a model suitable for integration of databases, i.e., for describing the data contained in two or more databases that contain similar data with different schemas.
2. It serves as the underlying model for notations such as XML that are being used to share information on the web.

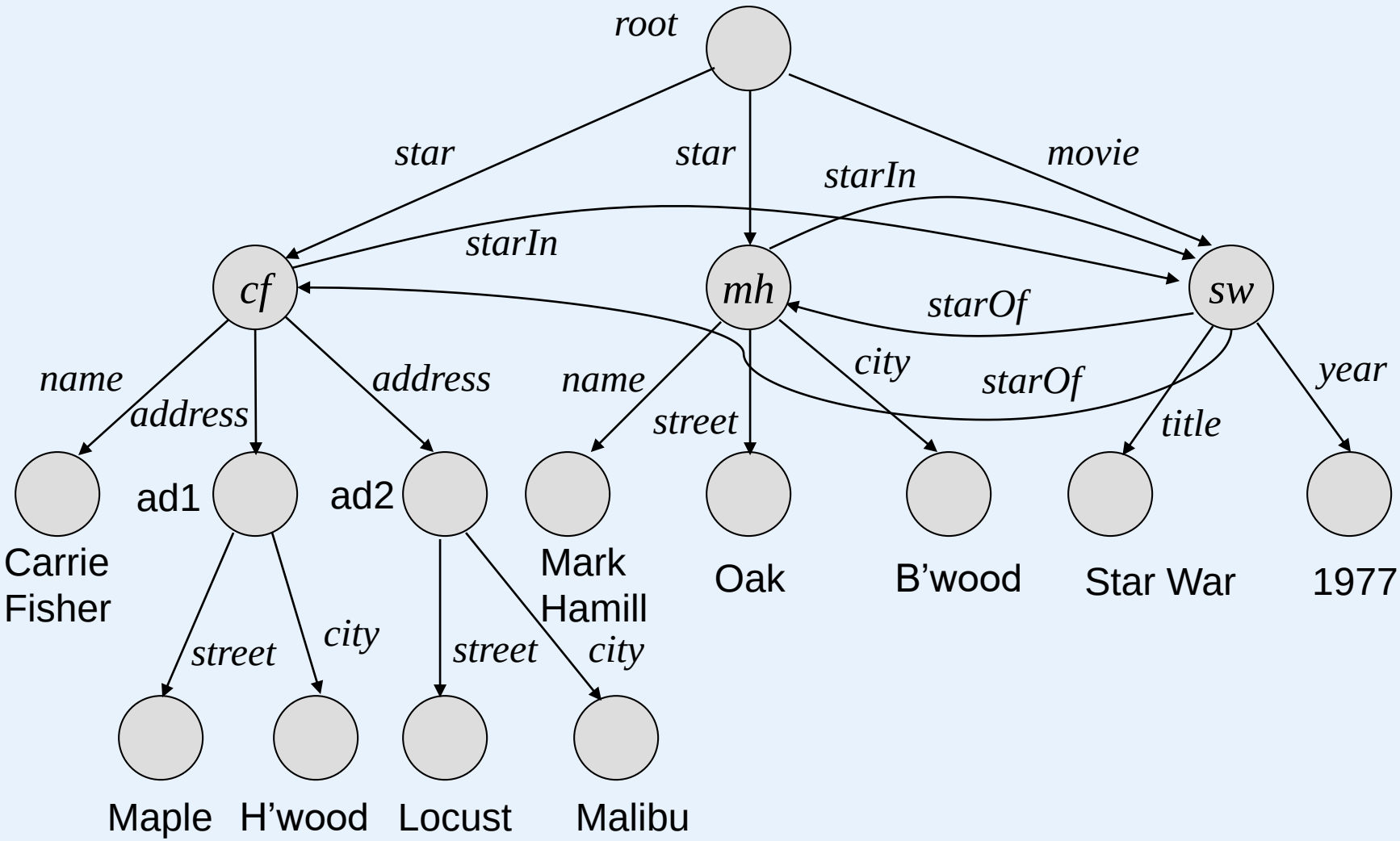
The semistructured data model can represent information more flexibly than the other models – E-R, UML, relational model, ODL (Object Definition Language).

Semistructured Data representation

A database of semistructured data is a collection of nodes.

- Each node is either a leaf or interior
- Leaf nodes have associated data; the type of this data can be any atomic type, such as numbers and strings.
- Interior nodes have one or more arcs out. Each arc has a label, which indicates how the node at the head of the arc relates to the node at the tail.
- One interior node, called the root, has no arcs entering and represents the entire database.

Semistructured-Data Model



Semistructured Data representation

A label *L* on the arc from node *N* to node *M* can play one of two roles.

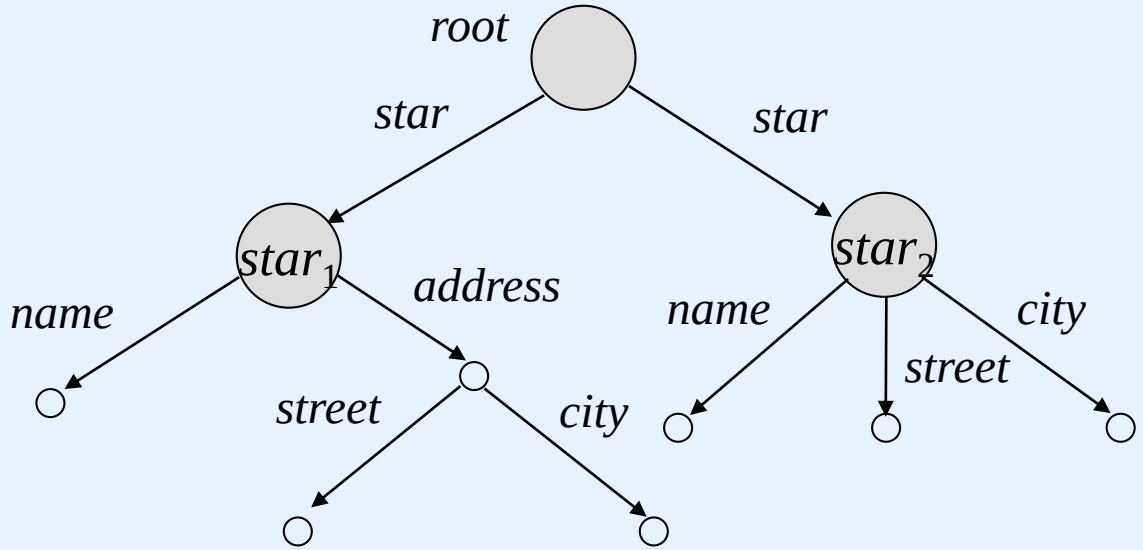
1. It may be possible to think of *N* as representing an object or entity, while *M* represents one of its attributes. Then, *L* represents the name of the attribute.
2. We may be able to think of *N* and *M* as objects or entities and *L* as the name of a relationship from *N* to *M*.

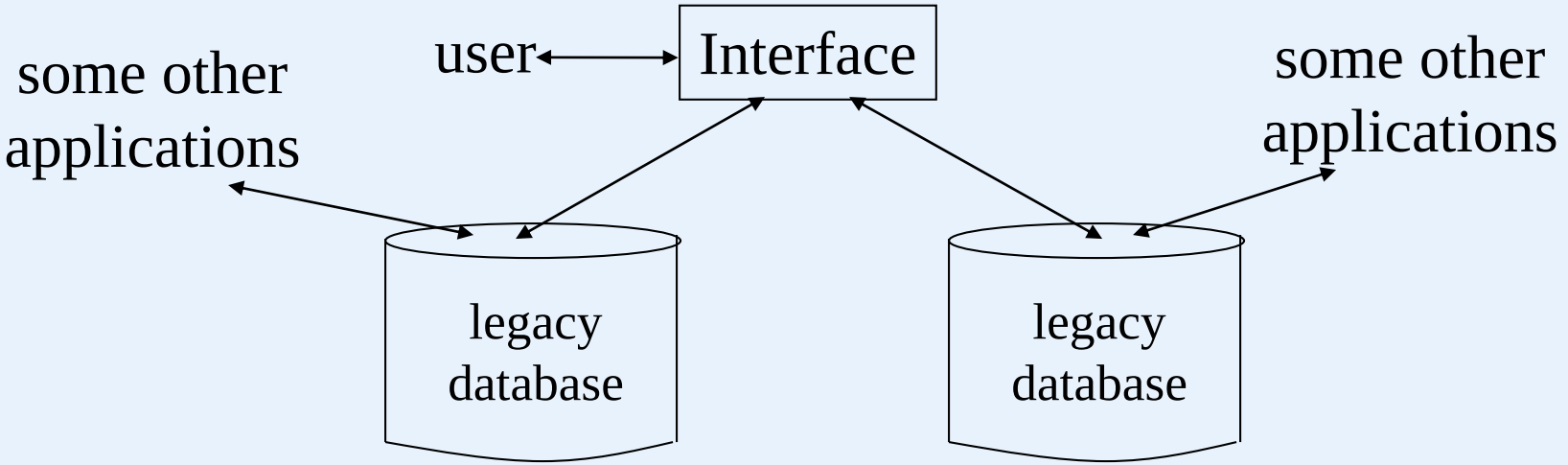
Semistructured Data model can be used to integrate information

Legacy-database problem: Databases tend over time to be used in so many different applications that it is impossible to turn them off and copy or translate their data into another database, even if we could figure out an efficient way to transform the data from one schema to another.

In this case, we will define a semistructured data model over all the legacy databases, working as an interface for users. Then, any query submitted against the interface will be translated according to local schemas.

Semistructured-Data Model





Stars(name, address(street, city))

Stars(name, street, city)

XML (*Extensible Markup Language*)

XML is a tag-based notation designed originally for *marking* documents, much like HTML. While HTML's tags talk about the presentation of the information contained in documents – for instance, which portion is to be displayed in italics or what the entries of a list are – XML tags intended to talk about the meanings of pieces of the document.

Tags:

opening tag - < …. >, e.g., <Foo>
closing tag - </ … >, e.g., </Foo>

A pair of matching tags and everything that comes between them is called an *element*.

XML with and without a schema

XML is designed to be used in two somewhat different modes:

1. *Well-formed* XML allows you to invent your own tags, much like the arc-labels in semistructured data. But there is no predefined schema. However, the nesting rule for tags must be obeyed, or the document is not well-formed.
2. *Valid* XML involves a DTD (Document Type Definition) that specifies the allowed tags and gives a grammar for how they may be nested. This form of XML is intermediate between the strict-schema such as the relational model, and the completely schemaless world of semistructured data.

Semistructured-Data Model

```
<? Xml version = “1.0” encoding = “utf-8” standalone = “yes” ?>
<StarMovieData>
    <Star>
        <Name>Carrie Fishes</Name>
        <Address>
            <Street>123 Maple St.</Street><City>Hollywood</City>
        </Address>
        <Address>
            <Street>5 Locust Ln.</Street><City>Malibu</City>
        <Address>
    </Star>
    <Star>
        <Name>Mark Hamill</Name><Street>456 Oak Rd.</Street>
        <City>Brentwood</City>
    </Star>
    <Movie>
        <Title>Star Wars</title><Year>1977</Year>
    </Movie>
</StarMovieData>
```

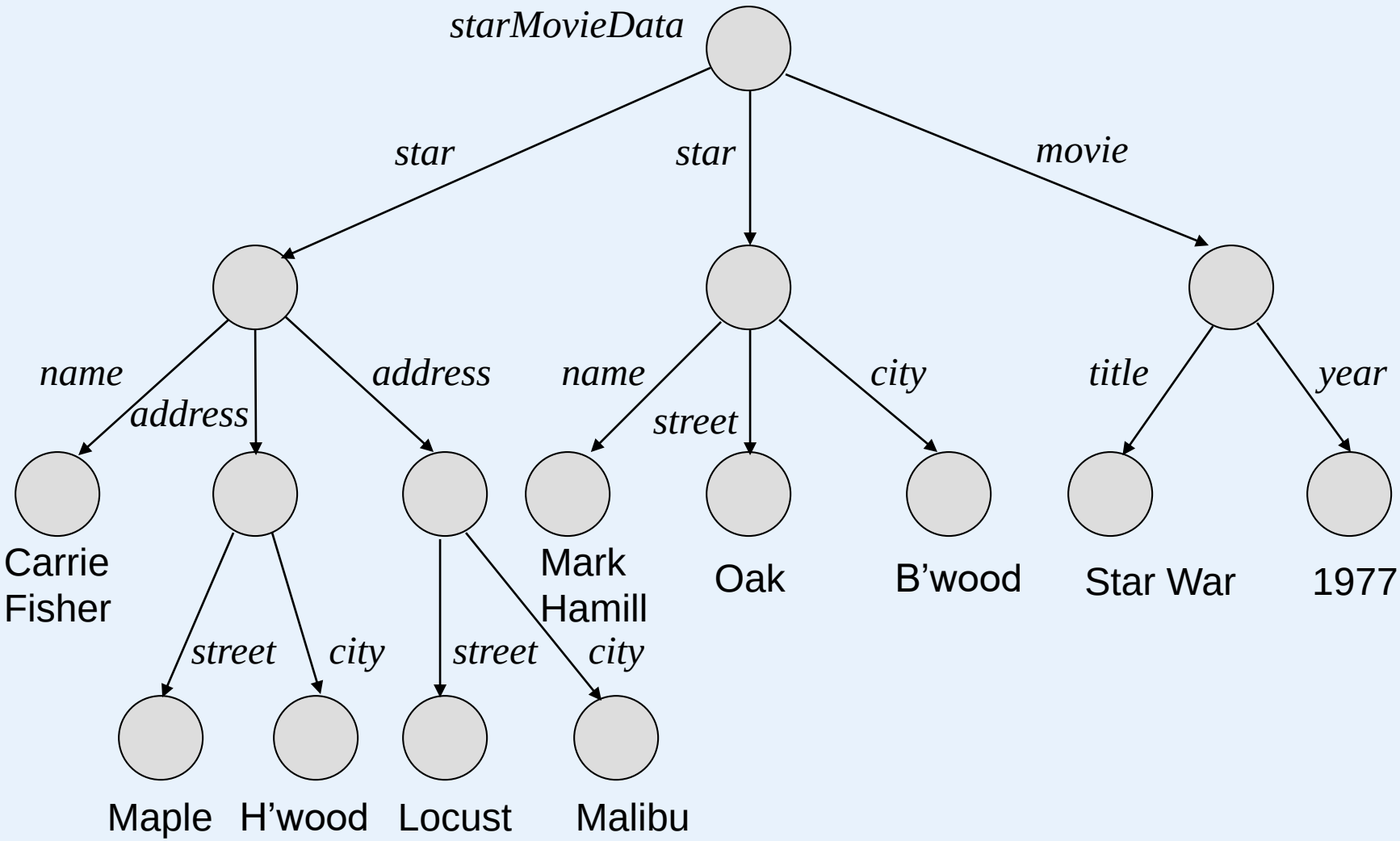
prologue

Semistructured-Data Model

<? Xml version = “1.0” encoding = “utf-8” standalone = “yes” ?>

- Xml – indicate that the file is an XML document
- Version = “1.0” – the first version of the document
- encoding = “utf-8” – utf (Unicode Transformation Format) is a common choice of encoding for characters because it is compatible with ASCII.
- standalone = “yes” – indicate that there is no DTD for this document. i.e., it is well-formed XML.

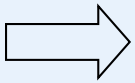
Semistructured-Data Model



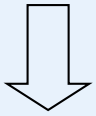
Attributes

As in HTML, an XML element can have attributes (name-value pairs) with its opening tag. An attribute is an alternative way to represent a leaf node of semistructured data. Attributes, like tags, can represent labeled arcs in a semisructured-data graph.

<Movie>
  <Title>“Star Wars”</title>
  <Year>1977</Year>
</Movie>



<Movie year = 1977>
  <Title>“Star Wars”</title>
</Movie>



<Movie title = “Star War” year = 1977>
</Movie>

Attributes that connect elements

An important use for attributes is to represent connections in a semistructured data graph that do not form a tree.

```
<? Xml version = “1.0” encoding = “utf-8” standalone = “yes” ?>
<StarMovieData>
    <Star starID = “cf” starredIn = “sw”>
            … …
    </Star>
    <Star starID = “mh” starredIn = “sw”>
            …  …
    </Star>
    <Movie movieID = “sw” starsOf = “cf”, “mh”>
        <Title>Star Wars</title><Year>1977</Year>
    </Movie>
</StarMovieData>
```

Namespace

There are situations in which XML data involves tags that come from two or more different sources. So we may have conflicting names. For example, we would not want to confuse an HTML tag used in a text with an XML tag that represents the meaning of that text. To distinguish among different vocabularies for tags in the same document, we can use a *namespace* for a set of tags.

To indicate that an element's tag should be interpreted as part of a certain space, we use the attribute xmlns in its opening tag:

xmlns: name = <Universal Resource Identifier>

Example:

<md : StarMoviedata xmlns : md = http://infolab.stanford.edu/movies>

XML storage

There are three approaches to storing XML to provide some efficiency:

1. Store the XML data in a parsed form, and provide a library of tools to navigate the data in that form. Two common standards are called SAX (Simple API for XML), and DOM (Document Object Model), MongoDB.
2. MongoDB – non-tabular databases

In Mongo DB, a document is stored as a set of property-value pairs (JSON format).

```
[   { title : “post1”,
    body: “body of post 1”,
    category: “news”,
    time: Date( )
    }
    { title : “post2”,
    body: “body of post 2”,
    category: “events”,
    time: Date( )
    }   ]
```

Semistructured-Data Model

3. Represent the document and their elements as relations, and use a conventional, relational DBMS to store them.

In order to represent XML documents as relations, we should give each document and each element of a document a unique ID. For each document, the ID could be its URL or its path in a file system.

A possible relational database schema:

DocRoot(docID, rootElmentID)
ElementValue(elementID, value)
SubElement(parentID, childID, position)
ElementAttribute(elementID, name, value)

Semistructured-Data Model

```
<? Xml version = “1.0” encoding = “utf-8” standalone = “yes” ?>
< md : StarMovieData xmlns : md = http://infolab.stanford.edu/movies >
    <Star starID = “cf” starredIn = “sw”>
        <Name>Carrie Fishes</Name>
        <Address>
            <Street>123 Maple St.</Street><City>Hollywood</City>
        </Address>
        <Address>
            <Street>5 Locust Ln.</Street><City>Malibu</City>
        <Address>
    </Star>
    <Star starID = “mh” starredIn = “sw”>
        <Name>Mark Hamill</Name><Street>456 Oak Rd.</Street>
        <City>Brentwood</City>
    </Star>
    <Movie movieID = “sw” starsOf = “cf”, “mh”>
        <Title>Star Wars</title><Year>1977</Year>
    </Movie>
</StarMovieData>
```

Semistructured-Data Model

DocRoot

Doc-id	rootElementID
1	1

elementValue

Doc-id	element-id	value
1	1	starMovieData
1	2	Star
1	3	Star
1	4	movie
…	…	…

subElement

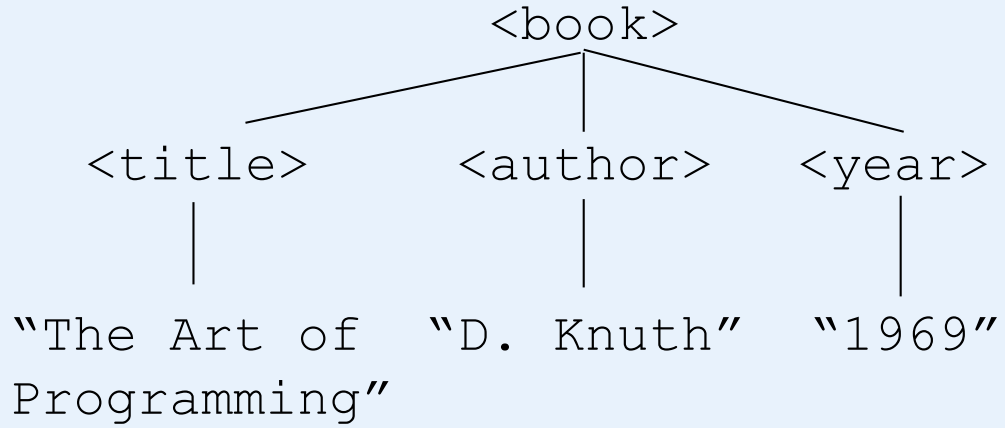
parentId	childId	position
1.1	1.2	1
1.1	1.3	2
1.1	1.4	3
… …	… ...	… …

elementAttribute

elemenAttId	attName	value
1.1	xmlns : md	http://... …
1.2	starId	“mf”
1.2	starId	“mh”
1.3	starredIn	“sw”
1.3	starredIn	“sw”
1.4	movieId	“sw”
1.4	starsOf	“sf”, “mh”

Transform an XML document to a tree

```
<book>
     <title>
        “The Art of Programming”
     </title>
      <author>
         “D. Knuth”
      </author>
      <year>
         “1969”
      </year>
</book>
```

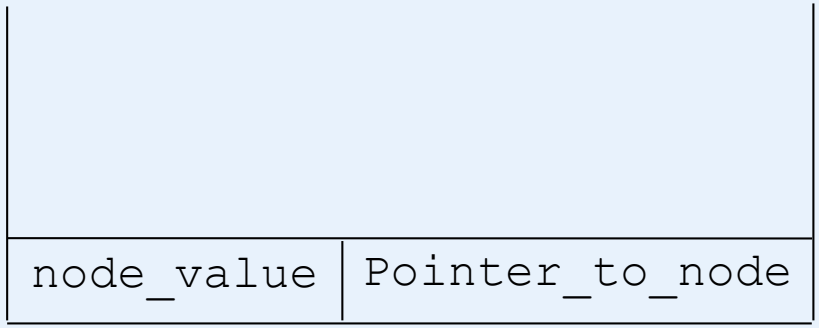


Transform an XML document to a tree

Read a file into a character array `A`:

<	b	o	o	k	>	<	t	i	t	l	e	>	“	T	h	e		A	r	t	…

stack S:



Transform an XML document to a tree

```
Algorithm:

Scan array A; Let A[i] be the character currently
encountered;
If A[i] is '<' and A[i+1] is a character then {
      generate a node x for A[i..j],
      where A[j] is '>' directly after A[i];
      let y = S.top().pointer_to_node;
      make x be a child of y; S.push(A[i..j], x);
If A[i] is ' " ', then {
      genearte a node x for A[i..j],
      where A[j] is ' " ' directly after A[i];
      let y = S.top().pointer_to_node;
      make x be a child of y;
If A[i] is '<' and A[i+1] is '/',
then S.pop();
```

Generating a node for an opening tag.

Generating a leaf node for a string value.

Popping out the stack when meeting a closing tag.

Document Type Definition (*DTD*)

A DTD is a set of grammar-like rules to indicate how elements can be nested.

DTD general form:

```
<!DOCTYPE root-tag [
    <!ELEMENT element-name (components)>
        … …
    ]>
```

Stars.dtd

```
<!DOCTYPE Stars [
    <!ELEMENT Stars (Star*)>
    <!ELEMENT Star (Name, Address+, Movies)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

Semistructured-Data Model

```
<Stars>
    <Star>
        <Name>Carrie Fishes</Name>
        <Address>
            <Street>123 Maple St.</Street>
            <City>Hollywood</City>
        </Address>
        <Movies>
            <Movie>
                <Title>Star Wars</Title>
                <Year>1977</Year>
            </Movie>
            <Movie>
                <Title>Empire Striker</Title>
                <Year>1980</Year>
            </Movie>
            <Movie>
                <Title>Return of the Jedi</Title><Year>1983</Year>
            </Movie>
        </Movies>
    </Star>
```

```
<!DOCTYPE Stars [
    <!ELEMENT Stars (Star*)>
    <!ELEMENT Star (Name, Address+, Movies)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

Semistructured-Data Model

```
<Star>
    <Name>Mark Hamill</Name>
    <Address>
        <Street>456 Oak Rd.</Street>
        <City>Brentwood</City>
    </Address>
    <Movies>
        <Movie>
            <Title>Star Wars</Title>
            <Year>1977</Year>
        </Movie>
        <Movie>
            <Title>Empire Wars</Title>
            <Year>1980</Year>
        </Movie>
        <Movie>
            <Title>Return of the Jedi</Title>
            <Year>1983</Year>
        </Movie>
    </Movie>
</Star>
</Stars>
```

```
<!DOCTYPE Stars [
    <!ELEMENT Stars (Star*)>
    <!ELEMENT Star (Name, Address+, Movies)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

Semistructured-Data Model

```
<!DOCTYPE Stars [
    <!ELEMENT Stars (Star*)>
    <!ELEMENT Star (Name, Address+, Movies)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

```
<? Xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<Stars>
    <Star>
        <Name>Carrie Fishes</Name>
        <Address>
            <Street>123 Maple St.</Street>
            <City>Hollywood</City>
        </Address>
        <Address>
            <Street>5 Locust Ln.</Street>
            <City>Malibu</City>
        <Address>
    </Star>
    <Star>
        <Name>Mark Hamill</Nam>
        <Street>456 Oak Rd.</Street>
        <City>Brentwood</City>
    </Star>
    <Movie>
        <Title>Star Wars</title><Year>1977</Year>
    </Movie>
</Stars>
```

This document does not confirm to the DTD. ⟶

Terminologies and notations in DTD:

1. #PCDATA means that an element has a value that is a text, and it has no element nested within. Parsed character data may be thought of as HTML text. A formatting character like < must be escaped by &lt. For instance,

   <!ELEMENT Title (#PCDATA)>

   say that between <Title> and </Title> tags a character string can appear.

2. The keyword Empty, with no parentheses, indicates that the element is one of those that has no matched closing tag. It has no subelements, nor does it have a text as a value. For example,

   <!ELEMENT Foo Empty>

   say that the only way the tag Foo can appear is as <Foo some attributes />.

Terminologies and notations in DTD:

1. A * following an element means that the element may occur any number of times, including zero times.
2. A + following an element means that the element may occur either one or more times.
3. A ? following an element means that the element may occur either zero times or one time, but no more.
4. We can connect a list of options by the 'or' symbol | to indicate that exactly one option appears. For example, if <Movie> element has <Genre> subelement, we might declare these by

<!ELEMENT Genre (Comedy | Drama | SciFi | Teen)>

To indicate that each <Genre> element has one of these four subelements.

<!ELEMENT Stars (Star*)
<!ELEMENT Star (Name, Address^{+}, Movies)>

Terminologies and notations in DTD:

5. Parentheses can be used to group components, For example, if we declare address to have the form:

```
<!ELEMENT Address (Street, (City | Zip))>
```

Then, <Address> elements would each have <Street> subelement followed by either a <City> or <Zip> subelement, but not both.

Using a DTD

If a document is intended to conform to a certain DTD, we

a) Include the DTD itself as a preamble to the document, or

b) In the opening line, refer to the DTD, which must be stored separately in the file system accessible to the application that is processing the document.

```
<?xml version = "1.0" encoding = "utf-8" standalone = "no"?>
<!DOCTYPE Star SYSTEM "star.dtd">
```

SYSTEM – keyword indicating that the DTD can be find in file star.dtd (this can also be a valid URL if the .dtd file is remote.)

Semistructured-Data Model

```
<?xml version="1.0" ?>
 <!DOCTYPE r [
<!ELEMENT r ANY >
<!ELEMENT a ANY >
<!ELEMENT b ANY >
<!ELEMENT c (a*)>
<!ELEMENT d (b*)>
]>
<r>
        <a>
                <b>
                        <a></a><a></a><b></b>
                </b>
                <c>
                        <a>
                                <b></b>
                        </a>
                </c>
                <a>
                        <a></a><b></b>
                </a>
        </a>
</r>
```

A DTD is included as a preamble.

Semistructured-Data Model

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
        <name>Tanmay Patil</name>
        <company>TutorialsPoint</company>
        <phone>(011) 123-4567</phone>
</address>
```

Attribute Lists

An element may be associated with an attribute list:

<!ATTLIST *element-name attribute-name type*>

```
<!ELEMENT Movie EMPTY>
    <!ATTLIST       Movie
        title     CDATA         #REQUIRED
        year      CDATA         #REQUIRED
        genre     (comedy | drama | sciFi | teen) #IMPLIED
    >
```

<Movie title = “Star Wars” year = “1977” genre = “sciFi”/>

Semistructured-Data Model

```
<!DOCTYPE StarMovieData [
  <!ELEMENT StarMovieData          (Star*, Movie*)>
  <!ELEMENT Star                   (Name, Address+)>
    <!ATTLIST Star
      starId          ID           #REQUIRED
      StarredIn       IDREFS       #IMPLIED
    >
  <!ELEMENT Name                   (#PCDATA)>
  <!ELEMENT Address                (Street, City)>
  <!ELEMNT Street                  (#PCDATA)>
  <!ELEMENT City                   (#PCDATA)>
  <!ELEMENT Movie                  (Title, Year)>
    <!ATTLIST       Movie
      movieId         ID           #REQUIRED
      startOf         IDREFS       #REQUIRED
    >
  <!ELEMENT Title                  (#PCDATA)>
  <!ELEMENT Year                   (#PCDATA)>
]>
```

Identifiers and Reference

Semistructured-Data Model

```
<? Xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
    <Star starID = "cf" starredIn = "sw">
        <Name>Carrie Fishes</Name>
        <Address>
            <Street>123 Maple St.</Street><City>Hollywood</City>
        </Address>
        <Address>
            <Street>5 Locust Ln.</Street><City
        <Address>
    </Star>
    <Star starID = "mh" starredIn = "sw">
        <Name>Mark Hamill</Name>
        <address>
            <Street>456 Oak Rd.</Street>
            <City>Brentwood</City>
        </address>
    </Star>
    <Movie movieID = "sw" starOf = "cf mh">
        <Title>Star Wars</title><Year>1977</Ye
    </Movie>
</StarMovieData>
```

```
<!DOCTYPE StarMovieData [
    <!ELEMENT StarMovieData          (Star*, Movie*)>
        <!ELEMENT Star               (Name, Address+)>
            <!ATTLIST Star
                starId      ID       #REQUIRED
                StarredIn   INREFS   #IMPLIED
            >
    <!ELEMENT Name                   (#PCDATA)>
    <ELEMENT Address                 (Street, City)>
    <!ELEMNT Street                  (#PCDATA)>
    <!ELEMENT City                   (#PCDATA)>
    <!ELEMENT Movie                  (Title, Year)>
        <!ATTLIST Movie
                movieIn     ID       #REQUIRED
                startOf     IDREFS   #REQUIRED
            >
    <!ELEMENT Title  (#PCDATA)>
    <!ELEMENT Year   (#PCDATA)>
]>
```

XML Schema

XML Schema is an alternative way to provide a schema for XML documents.

More powerful – give the schema designer extra capabilities.

- allow us to declare types, such as integers or float for simple elements.
- allow arbitrary restriction on the number of occurrences of subelements.
- give us the ability to declare keys and foreign keys.

The Form of an XML schema

- An XML schema description of a schema is itself an XML document. It uses the namespace at the URL

  http://www.w3.org/2001/XMLSchema

  that is provided by the World-Wide-Web Consortium.
- Each XML-schema document has the form:

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

Elements

An important component in an XML schema is the element, which is similar to an element definition in a DTD.

The form of an element definition in XML schema is:

```
<xs: element name = element name type = element type>
      constraints and/or structure information
</xs: element>
```

<xs: element name = “Title” type = “xs: string” />
<xs: element name = “Year” type = “xs: integer” />

DTD

```
<!DOCTYPE root-tag [
        … …
    <!ELEMENT Title (#PCDATA)>
     <!ELEMENT Year (#PCDATA)>
        … …
    ]>
```

Complex Types

A *complex type* in XML Schema can have several forms, but the most common is a sequence of elements.

```
<xs: complexType name = type name >
      <xs: sequence>
            list of element definitions
      </xs: sequence>
</xs: complexType>
```

```
<xs: complexType name = type name >
            list of attribute definitions
      </xs: complexType>
```

DTD

```
<!DOCTYPE root-tag [
      <!ELEMENT element-name (components)>
            ... ...
      ]>
```

Semistructured-Data Model

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

    <xs:complexType name = "movieType">
        <xs: sequence>
            <xs: element name = "Title" type = "xs: string" />
            </xs: element name = "Year" type = "xs: integer" />
        </xs: sequence>
    </xs: complexType>

    <xs: element name = "Movies">
        <xs: complexTyp>
            <xs: sequence>
                <xs: element name = "Movie" type = "movieType"
                    minOccurs = "0" maxOcurs = "unbouned" />
            </xs: sequence>
        </xs: complexTyp>
    </xs: element>
</xs: schema>
```

```
<xs: complexType name = type name >
    <xs: sequence>
        list of element definitions
    </xs: sequence>
</xs: complexType>
```

A schema for movies in XML schema. Itself is a document.

The above schema (in XML schema) is equivalent to the following DTD.

```
<!DOCTYPE Movies [
    <!ELEMENT      Movies (Movie*) >
    <!ELEMENT      Movie (Title, Year) >
    <!ELEMENT      Title (#PCDATA) >
    <!ELEMENT      Year (#PCDATA) >
]>
```

Attributes

A *complex type* can have *attributes*. That is, when we define a complex type *T*, we can include instances of element <xs: attribute>. Thus, when we use *T* as the type of an element *E* (in a document), then *E* can have (or must have) an instance of this attribute. The form of an attribute definition is:

<xs: attribute name = *attribute name* type = *type name*
other information about attribute />

<xs: attribute name = “title” type = “xs: integer” default = “0” />
<xs: attribute name = “year” type = “xs: integer” use = “required” />

Semistructured-Data Model

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

    <xs: complexType name = "movieType">
        <xs: attribute name = "title" type = "xs: string" use = "required" />
        <xs: attribute name = "year" type = "xs: integer" use = "required" />
    </xs: complexType>

    <xs: element name = "Movies">
        <xs: complexTyp>
            <xs: sequence>
                <xs: element name = "Movie" type = "movieType"
                    minOccurs = "0" maxOcurs = "unbouned" />
            </xs: sequence>
        </xs: complexTyp>
    </xs: element>
</xs: schema>
```

```
<xs:complexType name = "movieType">
    <xs: sequence>
        <xs: element name = "Title" type = "xs: string" />
        </xs: element name = "Year" type = "xs: integer" />
    </xs: sequence>
</xs: complexType>
```

A schema for movies in XML schema.
Itself is a document.

The above schema (in XML schema) is equivalent to the following DTD.

```
<!DOCTYPE Movies [
    <!ELEMENT      Movies (Movie*) >
    <!ELEMENT      Movie EMPTY />
        <!ATTLIST  Movie
            Title      CDATA #REQUIRED
            Year       CDATA #REQUIRED
        >
]>
```

```
<!DOCTYPE Movies [
      <!ELEMENT      Movies (Movie*) >
      <!ELEMENT      Movie (Title, Year) >
      <!ELEMENT      Title (#PCDATA) >
      <!ELEMENT      Year (#PCDATA) >
]>
```

Restricted Simple Types

It is possible to create a restricted version of a simple type such as integer or string by limiting the values the type can take. These types can then be used as the type of an attribute or element.

1. Restricting numerical values by using minInclusive to state the lower bound, maxInclusive to state the upper bound.
2. Restricting values to an numerated type.

```
<xs: simpleType name = type name >
    <xs: restriction base = base type >
        upper and/or lower bounds
    </xs: restriction>
</xs: simpleType>
```

```
<xs: enumeration value = some value />
```

Semistructured-Data Model

```
<xs: simpleType name = "movieYearType" >
    <xs: restriction base = "xs: integer" >
        <xs:minInclusive value = "1915" />
    </xs: restriction>
</xs: simpleType>
```

```
<xs: simpleType name = "genretype" >
    <xs: restriction base = "xs: string" >
        <xs: enumeration value = "comedy" />
        <xs: enumeration value = "drama" />
        <xs: enumeration value = "sciFi" />
        <xs: enumeration value = "teen" />
    </xs: restriction>
</xs: simpleType>
```

Keys in XML Schema

An element can have a *key declaration*, which is a field or several fields to uniquely identify the element among a certain class *C* of elements).

field: an attribute or a subelement.
selector: a path to reach a certain node in a document tree.

```
<xs: key name = key name >
      <xs: selector xpath = path description >
      <xs: field xpath = path description >
            more field specification
</xs: key>
```

```
Create table EMPLOYEE
            (…,
             DNO INT NOT NULL DEFAULT 1,
   CONSTRAINT EMPPK
            PRIMARY KEY(SSN),
   CONSTRAINT EMPSUPERFK
   FOREIGN KEY(SUPERSSN)
   REFERENCES
                        EMPLOYEE(SSN)
            ON DELETE SET NULL  ON
   UPDATE
                        CASCADE,
   CONSTRAINT EMPDEPTFK
   FOREIGN KEY(DNO) REFERENCES

            DEPARTMENT(DNUMBER)
             ON DELETE SET DEFAULT
            ON UPDATE CASCADE);
```

Semistructured-Data Model

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

  <xs: simpleType name = “genretype” >
    <xs: restriction base = “xs: string” >
      <xs: enumeration value = “comedy” />
      <xs: enumeration value = “drama” />
      <xs: enumeration value = “sciFi” />
      <xs: enumeration value = “teen” />
    </xs: restriction>
  </xs: simpleType>

  <xs: complexType name = “movieType”>
    <xs: attribute name = “title” type = “xs: string” />
    <xs: attribute name = “year” type = “xs: integer” />
    <xs: attribute name = “Genre” type = “genreType”
      minOccurs = “0” maxOccurs = “1” />
  </xs: complexType>
```

Semistructured-Data Model

```
<xs: element name = “Movies”>
  <xs: complexTyp>
    <xs: sequence>
      <xs: element name = “Movie” type = “movieType”
      minOccurs = “0” maxOcurs = “unbounded” />
    </xs: sequence>
  </xs: complexTyp>
  <xs: key name = “movieKey”>
    <xs: selector xpath = “Movie” />
    <xs: field xpath = “@Title” />
    <xs: field xpath = “@Year” />
  </xs: key>
</xs: element>
</xs: schema>
```

/Movies/Movie

/Movies/Movie@Title

/Movies/Movie@Year

```
<? Xml version = “1.0” encoding = “utf-8” standalone = “yes” ?>
  <Movies>
      … …
    <Movie Title = “Star Wars” Year = 1977 Genre = “comedy” />
      … …
  </Movies>
```

Foreign Keys in XML Schema

We can declare that an element has, perhaps deeply nested within it, a field or fields that serve as a reference to the key for some other element. It is similar to what we get with ID's and IDREF's in DTD.

In DTD: untyped references
In XML schema: typed references

```
<xs: keyref name = foreign-key name
      refer = key name>
      <xs: selector xpath = path description >
      <xs: field xpath = path description >
            more field specification
</xs: keyref>
```

```
Create table EMPLOYEE
            (…,
             DNO INT NOT NULL DEFAULT 1,
   CONSTRAINT EMPPK
            PRIMARY KEY(SSN),
   CONSTRAINT EMPSUPERFK
   FOREIGN KEY(SUPERSSN)
   REFERENCES
                        EMPLOYEE(SSN)
            ON DELETE SET NULL  ON
   UPDATE
                        CASCADE,
   CONSTRAINT EMPDEPTFK
   FOREIGN KEY(DNO) REFERENCES
            DEPARTMENT(DNUMBER)
             ON DELETE SET DEFAULT
            ON UPDATE CASCADE);
```

Semistructured-Data Model

```
<? Xml version = “1.0” encoding = “utf-8” ?>
<xs: schema xmlns: xs = “http://www.w3.org/2001/XMLSchema”>
<xs: element name = “Stars”>
  <xs: complxType>
    <xs: sequence>
      <xs: element name = “Star” minOccurs = “1” maxOccurs = “unbounded”>
        <xs: complexType>
          <xs: sequence>
            <xs: element name = “Name” type = “xs: string” />
            <xs: element name = “Address” type = “xs: string” />
            <xs: element name = “StarredIn” minOccurs = “0” maxOccurs = “1”>
              <xs: complexType>
                <xs: attribute name = “title” type = “xs: string” />
                <xs: attribute name = “year” type = “xs: integer” />
              </xs: complexType>
            </xs: element>
          </xs: sequence>
        </xs: complexType>
      </xs: element>
    </xs: sequence>
  </xs: complexType>
```

Semistructured-Data Model

```
<xs: keyref name = "movieRef" refers = "movieKey">
    <xs: selector xpath = "Star/StarredIn" />
    <xs: field xpath = "@title" />
    <xs: field xpath = "@year" />
</xs: keyref>
</xs: element>
</xs: schema>
```

```
<? Xml version = "1.0" encoding = "utf-8" standalone =
"yes" ?>
  <Stars>
    <Star>
        <Name>Mark Hamill</Name>
        <Address>456 Oak Rd. Brentwood</Address>
        <StarredIn title = "star war" year = "1977"/>
    </Star>
    … …

  </Stars>
```

About usage of XML schema

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

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

The following example is an XML Schema file called "note.xsd" that defines the elements of the above XML document ("note.xml"):

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