Semi-structured Data Model
XML

“Schema is implied by the data rather than being declared separately”

Semi-structured Data

- Suitable for integration of different databases
- Serves as underlying model for notations like XML
- Semi-structured data is schemaless
- Data is self-describing
- Harder Query Processor
- Can have arbitrary number of attribues

Semi-structured Data Model

- Data is a collection of nodes (leaf or interior)
- Leaf has data (Atomic type – numbers and srings)
- Interior – Arcs with laels
- Root node has no incoming arcs (interior node)

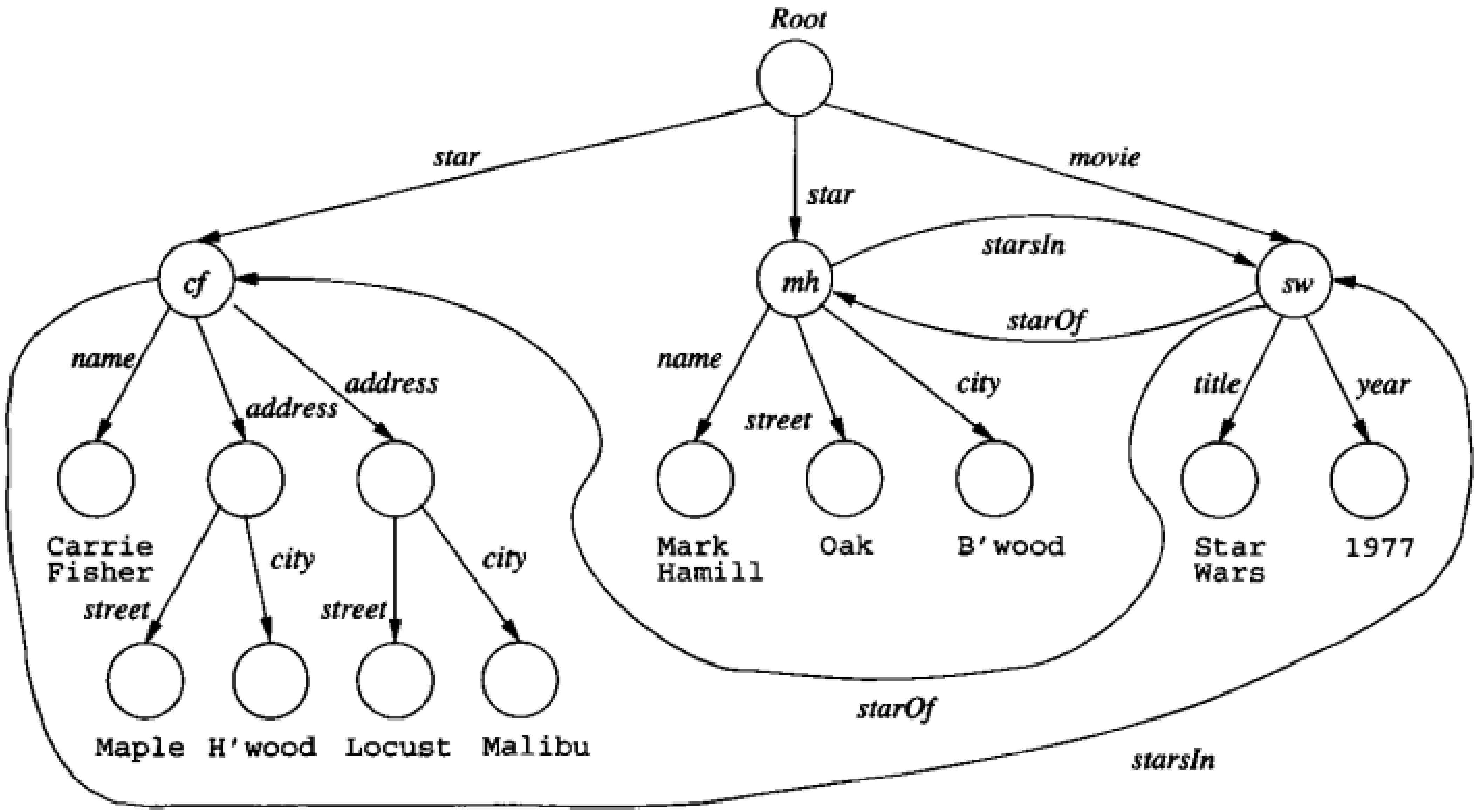


Figure 11.1: Semistructured data representing a movie and stars

Legacy Databases and Integration

- Once a database has been in existence for a while, it becomes impossible to disentangle it from the applications that grow up around it, so the database can never be decommissioned.

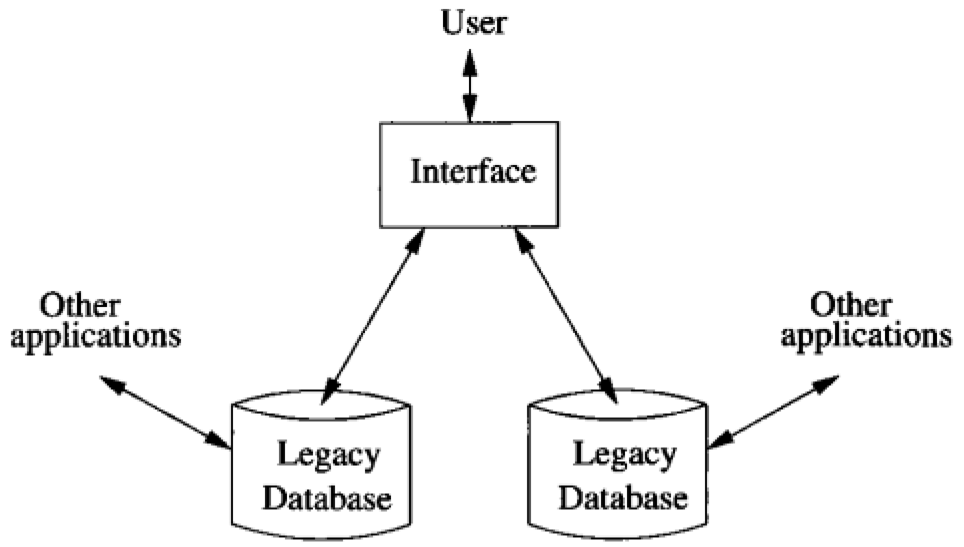


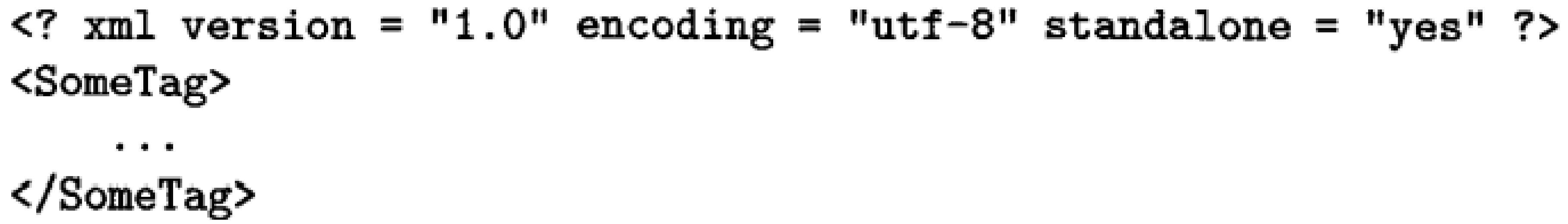
Figure 11.2: Integrating two legacy databases through an interface that supports semistructured data

Extensible Markup Language (XML)

- Tag based notation for marking documents.
- HTML talks abouts presentation while XML talks about meaning
- Opening and closing tags
- Single tag with no closing tag is allowed. It can only have attributes.
- Two modes of XML
  - Well-formed XML
    - Invent your own tags and follow their rules
  - Valid XML
    - Involves a “DTD” or “Document Type Definition”
    - Allowable Tags and gives a grammar for how they may be nested.
    - Intermediate model between relational and complete schema less.

Well Formed XML

- Starts with a declaration and has a root element.
- Encoding: UTF-8 “Unicode Transformation Format”. Uses one byte for ASCII characters.
- Standalone = “yes” indicates that there is no DTD for this document.



```
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<SomeTag>
    ...
</SomeTag>
```

```
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
    <Star>
        <Name>Carrie Fisher</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>
```

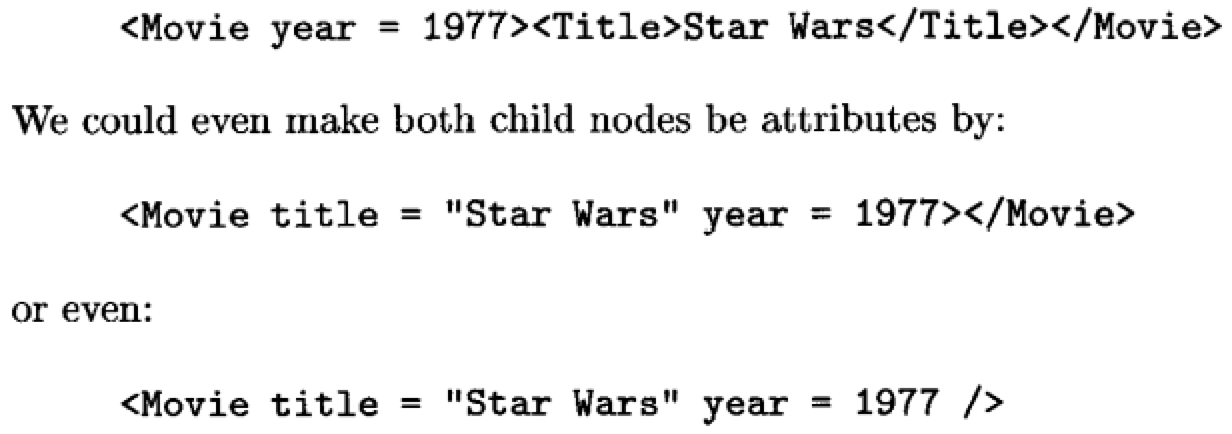
Figure 11.3: An XML document about stars and movies

```
<Star>
    <Name>Mark Hamill</Name>
    <Street>Oak</Street>
    <City>Brentwood</City>
    <Movie>
        <Title>Star Wars</Title>
        <Year>1977</Year>
    </Movie>
    <Movie>
        <Title>Empire Strikes Back</Title>
        <Year>1980</Year>
    </Movie>
</Star>
```

Figure 11.4: Nesting movies within stars

Attributes

- Name-value pairs of tags



```
<Movie year = 1977><Title>Star Wars</Title></Movie>
```

We could even make both child nodes be attributes by:

```
<Movie title = "Star Wars" year = 1977></Movie>
```

or even:

```
<Movie title = "Star Wars" year = 1977 />
```

Attributes as Identifiers and References

```
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
    <Star starID = "cf" starredIn = "sw">
        <Name>Carrie Fisher</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>
```

Namespaces

- In certain situations, XML data may involve tags that come from two or more different sources, and which may therefore have conflicting names.
- URI (Universal Resource Identifier) is a URL referring to a document that describes the meaning of the tags in the namespace.

```
xmlns:name="URI"
```

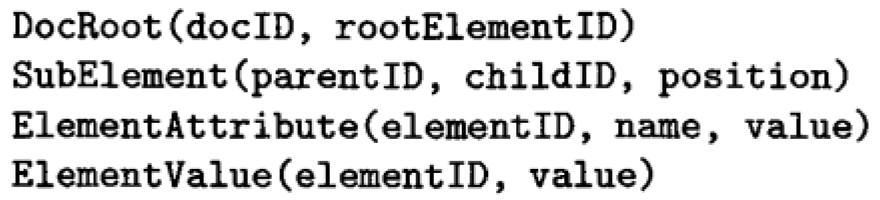
Example 11.7: Suppose we want to say that in element `StarMovieData` of Fig. 11.5 certain tags belong to the namespace defined in the document `infolab.stanford.edu/movies`. We could choose a name such as `md` for the namespace by using the opening tag:

```
<md:StarMovieData xmlns:md=
    "http://infolab.stanford.edu/movies">
```

Our intent is that `StarMovieData` itself is part of this namespace, so it gets the prefix `md:`, as does its closing tag `/md:StarMovieData`. Inside this element, we have the option of asserting that the tags of subelements belong to this namespace by prefixing their opening and closing tags with `md:`. □

XML as Communication

- Store the XML data in a parsed form, and provide a library of tools to navigate the data. Two common standards
  - SAX (Simple API for XML)
  - DOM (Document Object Model)
- Represent the documents and their elements as relations



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

Document Type Definition (DTD)

- Rules (DTD)
- Components
  - #PCDATA
    - Parsed Character Data
    - PCDATA after an element name means that element has a value that is text, and it has no elements nested within.
  - EMPTY
    - It has no subelements nor text.

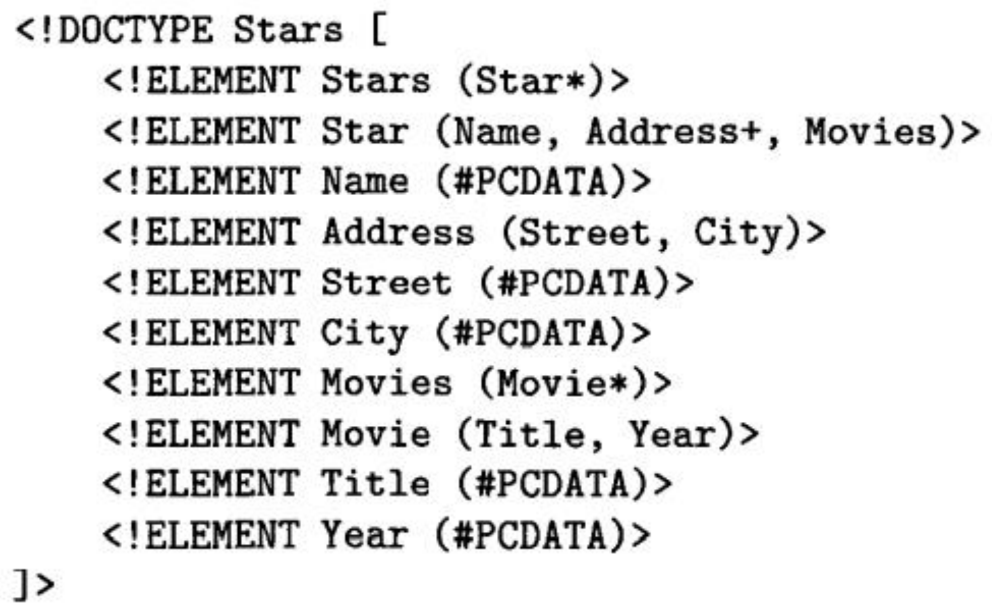
```
<!DOCTYPE root-tag [
    <!ELEMENT element-name (components)>
        more elements
]>
```

```
<!ELEMENT Title (#PCDATA)>
```

```
<!ELEMENT Foo EMPTY>
```

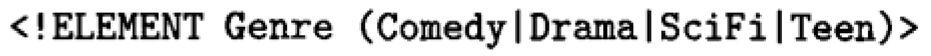
DTD Format

- (Stars*) – 0 or more stars
- (Address+) – 1 or more
- ? Following an element means that the element may occur either zero or one times, but no more.
- (or) – only 1 of the possibilities
- () – group components

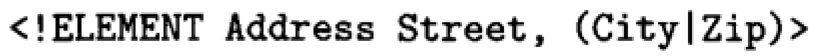


```
<!DOCTYPE Stars [
    <!ELEMENT Stars (Star*)>
    <!ELEMENT Star (Name, Address+, Movies)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address (Street, City)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movies (Movie*)>
    <!ELEMENT Movie (Title, Year)>
    <!ELEMENT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

Figure 11.6: A DTD for movie stars



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



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

```
<Stars>
    <Star>
        <Name>Carrie Fisher</Name>
        <Address>
            <Street>123 Maple St.</Street>
            <City>Hollywood</City>
        </Address>
        <Address>
            <Street>5 Locust Ln.</Street>
            <City>Malibu</City>
        </Address>
        <Movies>
            <Movie>
                <Title>Star Wars</Title>
                <Year>1977</Year>
            </Movie>
            <Movie>
                <Title>Empire Strikes Back</Title
                <Year>1980</Year>
            </Movie>
            <Movie>
                <Title>Return of the Jedi</Title>
                <Year>1983</Year>
            </Movie>
        </Movies>
    </Star>
    </Star>
    <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 Strikes Back</Title>
                <Year>1980</Year>
            </Movie>
            <Movie>
                <Title>Return of the Jedi</Title>
                <Year>1983</Year>
            </Movie>
        </Movies>
    </Star>
</Stars>
```

Figure 11.7: Example of a document following the DTD of Fig. 11.6

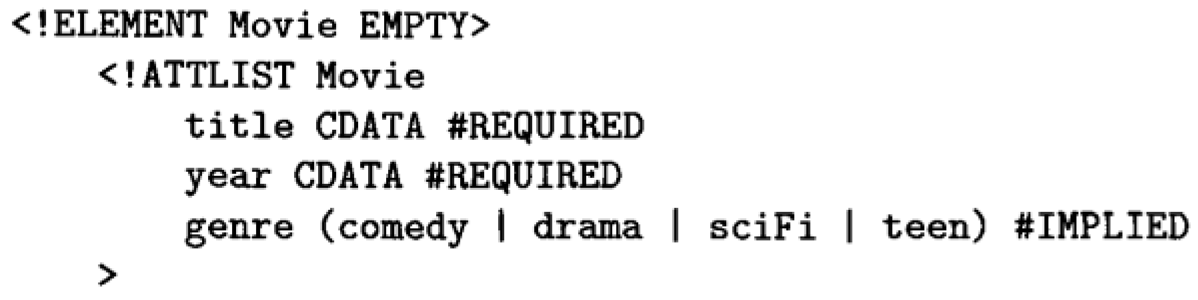
Using a DTD

- If a document conforms to a certain DTD, we can
  - Include the DTD itself as a preamble to the document
  - Refer to the DTD in the opening line

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

Attribute Lists of DTD

- Named Attribute can be an attribute of the named element
- Several attributes can be defined in one ATTLIST
- Type of Attributes
    - CDATA – Character string data
    - Enumerated type, which is a list of possible strings, surrounded by parentheses and separated by |’s.
    - #REQUIRED – must be present
    - #IMPLIED - optional



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

Figure 11.8: Data about movies will appear as attributes

Identifiers and References (Attribute Type)

- Attributes can be used as identifiers for elements.
- In DTD, these attributes have the type ID
- Other attributes have values that are references to these element ID’s; these attributes may be declared to have type IDREF. (pointer to the ID)
- IDREFS: the value of the attribute is a string consisting of a list of ID’s, separated by whitespaces

```
<!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)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movie (Title, Year)>
        <!ATTLIST Movie
            movieId ID #REQUIRED
            starsOf IDREFS #IMPLIED
        >
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

Figure 11.9: A DTD for stars and movies, using ID's and IDREF's

```
<!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)>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
    <!ELEMENT Movie (Title, Year)>
        <!ATTLIST Movie
            movieId ID #REQUIRED
            starsOf IDREFS #IMPLIED
        >
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Year (#PCDATA)>
]>
```

Figure 11.9: A DTD for stars and movies, using ID's and IDREF's

```
<? xml version = "1.0" encoding = "utf-8" standalone = "yes" ?>
<StarMovieData>
    <Star starID = "cf" starredIn = "sw">
        <Name>Carrie Fisher</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>
        <Address>
            <Street>456 Oak Rd.</Street>
            <City>Brentwood</City>
        </Address>
    </Star>
    <Movie movieID = "sw" starsOf = "cf mh">
        <Title>Star Wars</Title>
        <Year>1977</Year>
    </Movie>
</StarMovieData>
```

XML Schema

Introduction

- Alternative way to provide a schema for XML documents
- Powerful than DTD
- Allows arbitrary restrictions on the number of occurrences of sub-elements.
- Allows to declare types, such as integers or float for simple elements
- Ability to declare keys and foreign keys.

The Form of XML Schema

- XS: ELEMENT
  - Type: Simple or Complex
- Simple
  - xs: integer
  - xs:string
  - xs:Boolean
  - No subelements
- Complex Type
  - Sequence of elements
  - minOccurs (no fewer than given)
  - maxOccurs (no more than given) (infinite: unbounded)
  - Default: 1 occurrence
  - Xs:all (each of the elements between opening and closing tag must occur in any order exactly once each)
  - Xs:choice (exactly one of the elements found between the opening and closing will appear)

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

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

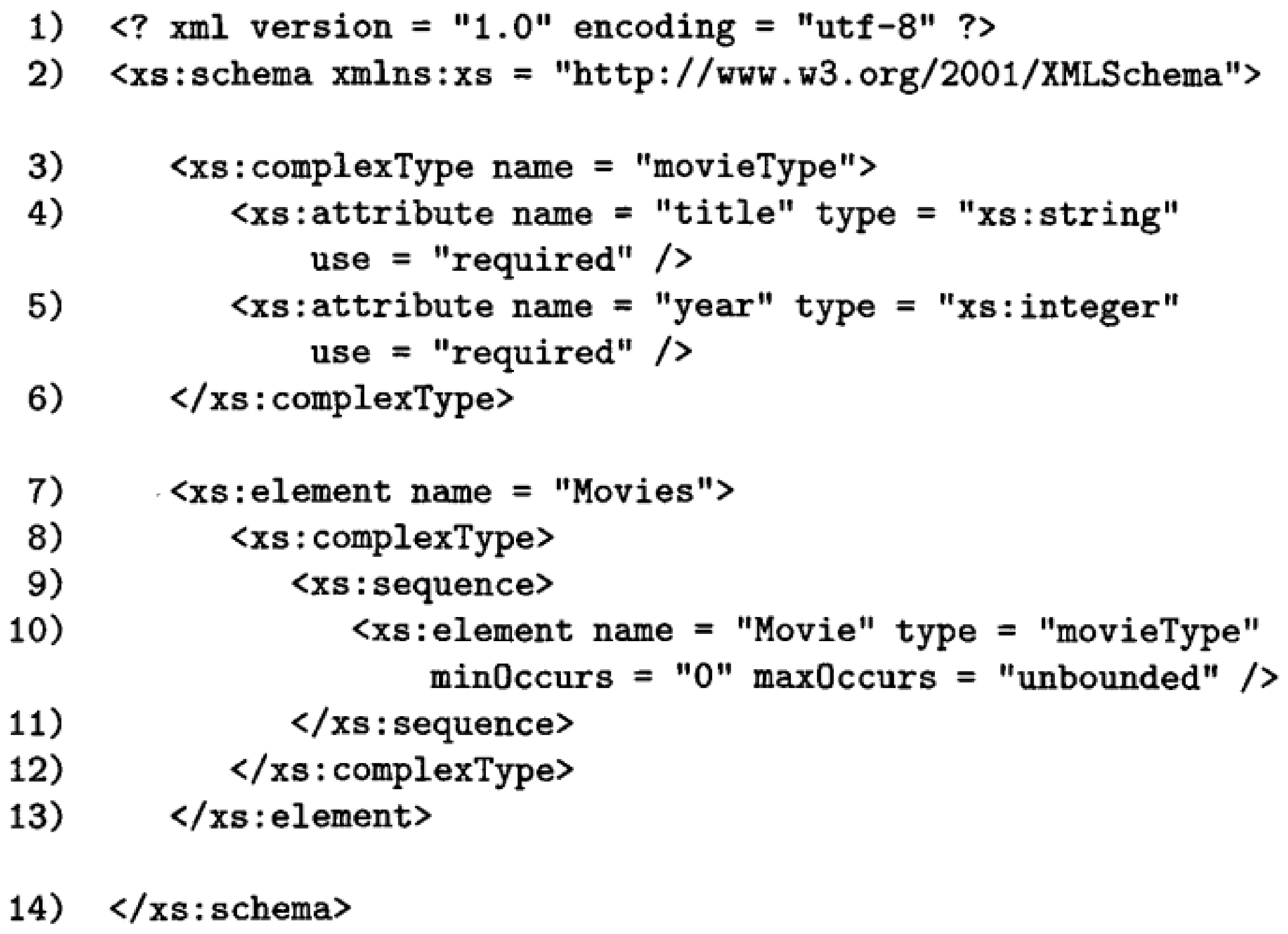
```
<xs:element name = "Title" type = "xs:string" />
<xs:element name = "Year" type = "xs:integer" />
```

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

```
<? 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:complexType>
         <xs:sequence>
            <xs:element name = "Movie" type = "movieType"
                minOccurs = "0" maxOccurs = "unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Figure 11.12: A schema for movies in XML Schema

Attributes



```
<? 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:complexType>
         <xs:sequence>
            <xs:element name = "Movie" type = "movieType"
                minOccurs = "0" maxOccurs = "unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>

</xs:schema>
```

Restriction in SimpleType

```
<xs:simpIeType 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

```
<? 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:sequence>
         <xs:element name = "Title" type = "xs:string" />
         <xs:element name = "Year" type = "xs:integer" />
         <xs:element name = "Genre" type = "genreType"
            minOccurs = "0" maxOccurs = "1" />
      </xs:sequence>
   </xs:complexType>

   <xs:element name = "Movies">
      <xs:complexType>
         <xs:sequence>
            <xs:element name = "Movie" type = "movieType"
               minOccurs = "0" maxOccurs = "unbounded" />
         </xs:sequence>
      </xs:complexType>
      <xs:key name = "movieKey">
         <xs:selector xpath = "Movie" />
         <xs:field xpath = "Title" />
         <xs:field xpath = "Year" />
      </xs:key>
   </xs:element>

</xs:schema>
```

Foreign Key

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

<xs:element name = "Stars">

   <xs:complexType>
      <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 = "unbounded">
                     <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>

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

</xs:element>
```

Figure 11.20: Stars with a foreign key