CS157A:
Introduction to Database Management Systems

Chapter 11: The Semi-Structured Data Model

Suneuy Kim

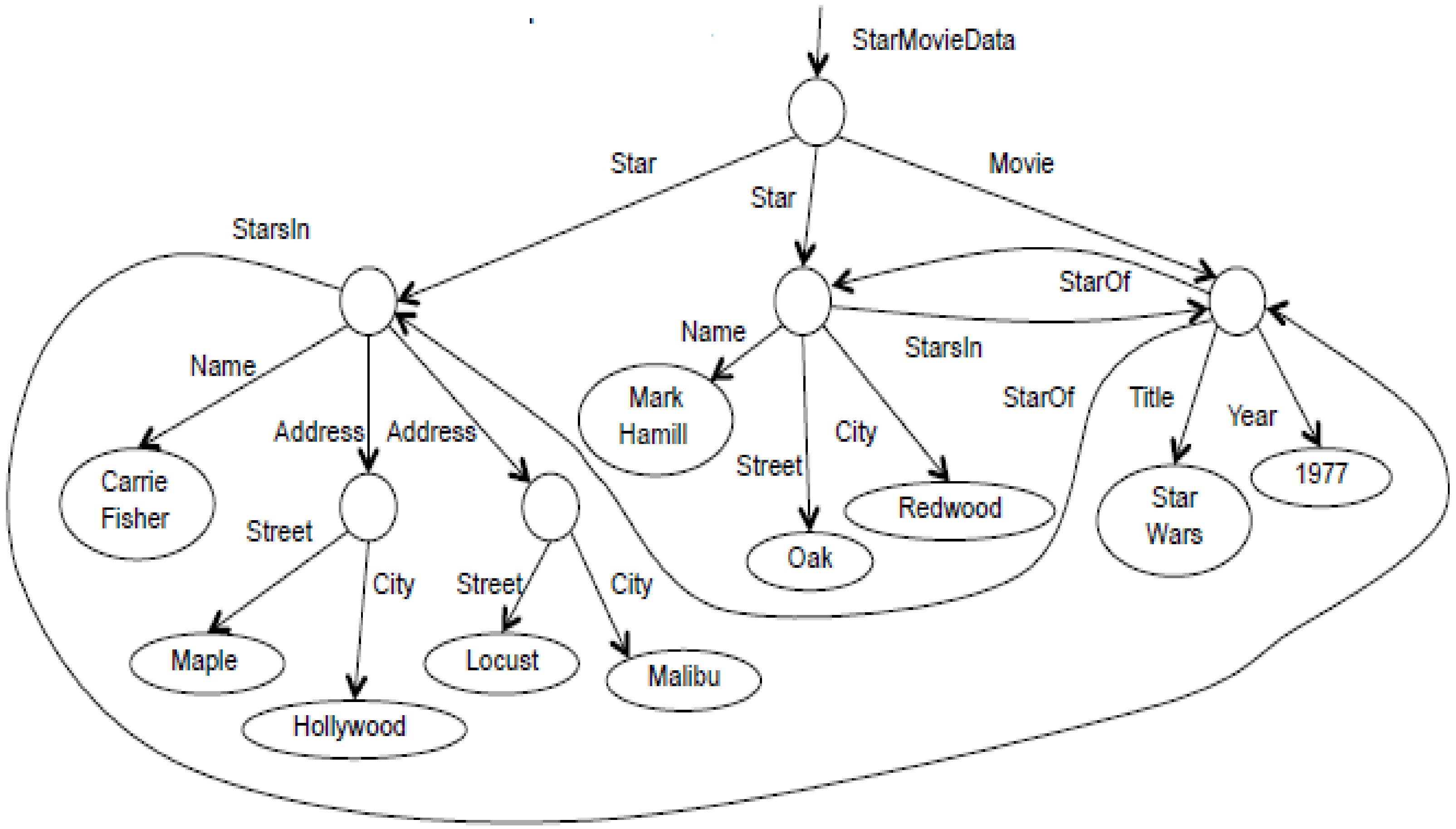
Semistructured Data Model

Role of semistructured data model in database systems:

- Integration of databases (flexible and self-describing)
- Underlying model for notations such as XML.

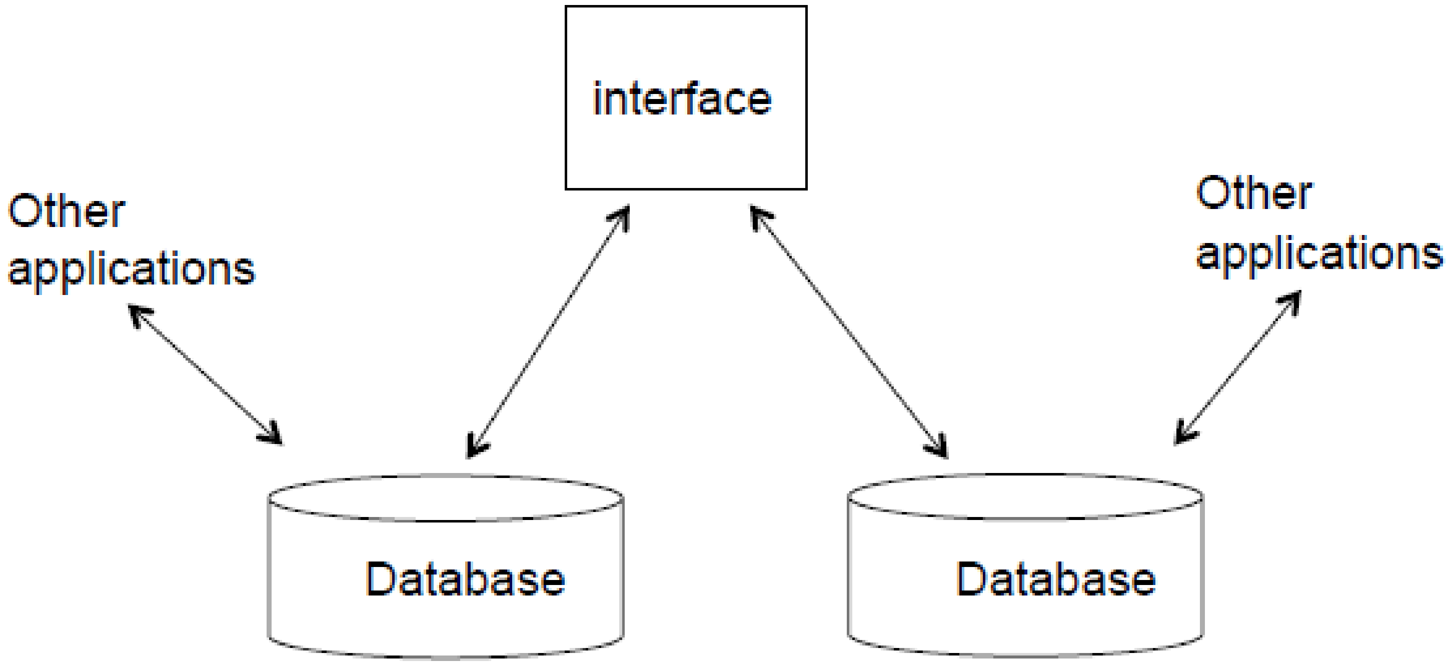
Semi structured data representation

- A database of semi structured data is a collection of nodes.
- Root represents the entire database.
- Immediate children of roots represents central entities.
- Leaf nodes have data
- A label on an arc from node N to node M
  - name of the attribute or the sub element
  - relationship


StarMovieData
Star
Star
Movie
StarsIn
StarOf
Name
Name
StarsIn
Mark Hamill
Carrie Fisher
Address
Address
StarOf
Title
Year
City
Street
Street
City
Street
City
Redwood
Oak
Star Wars
1977
Maple
Locust
Malibu
Hollywood

Information integration via semi structured data

“legacy-database problem”



XML (Extensible Markup Language)

- Standard for data representation and exchange
- Basic constructs
  - Tagged elements (can be nested)
  - Attributes
  - Text
- Tags
  - Play the same role as the labels on the arcs of semi structured-data graph.
  - HTML tags describe formatting
  - XML tags describe content, that is, meaning of data

Example: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<Movies>
  <Movie title = "King Kong">
     <Version year ="1933">
       <Star>Far Wray</Star>
     </Version>
     <Version year = "1976">
       <Star>Carrie Fisher</Star>
       <Star>Jessica Lange</Star>
     </Version>
   </Movie>
   <Movie title = "Footloose">
    <Version year = "1984">
       <Star> Kevin Bacon</Star>
       <Star>John Lithgow</Star>
       <Star>Sarah Jessica Parker</Star>
    </Version>
   </Movie>
</Movies>
```

Semantic Tag

- Tags are normally matched pairs, as

```
<Foo> Element </Foo>
```

- Element: A pair of matching tags and everything that comes between them
- `<Foo/>` a single tag cannot have any other elements. It can have attributes.

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

- Tags may be nested arbitrarily.
- XML tags are case-sensitive.

Attributes

An alternative way to represent a leaf node

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

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

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

Attributes

- Attributes that represent the identifier of an element
- Attributes that can connect elements
- Attribute names are case sensitive.

<Star starID = “cf” starredIn = “sw”> </Star>

<Star starID = “mh” starredIn = “sw” > </Star>

<Movie movieID = “sw” starsOf = “cf mh” > </Movie>

Example: attribute serving as id

- StarMovieData.xml

Namespaces

To distinguish among different vocabularies for tags in the same document

```
<md:StarMovieData xmlns:md =

"http://infolab.stanford.edu/movies">
```

URI: URL that refers to a document describing the meaning of the tags in the name space.

XML with and without a Schema

- Well-formed XML
  - You can invent your own tags – no predefined schema
  - The nesting rule for tags must be obeyed.
- Valid XML
  - Conforms to a certain DTD (Document Type Definition) or a XML Schema
  - DTD/XML Schema specifies the allowable tags and a grammar about how they may be nested.

Well-formed XML

The minimum requirement of well-formed XML: indication that this is a xml document and a root element.

```
<? xml version = “1.0” encoding = “utf-8”
standalone = “yes” ?>
<some  tag>
```

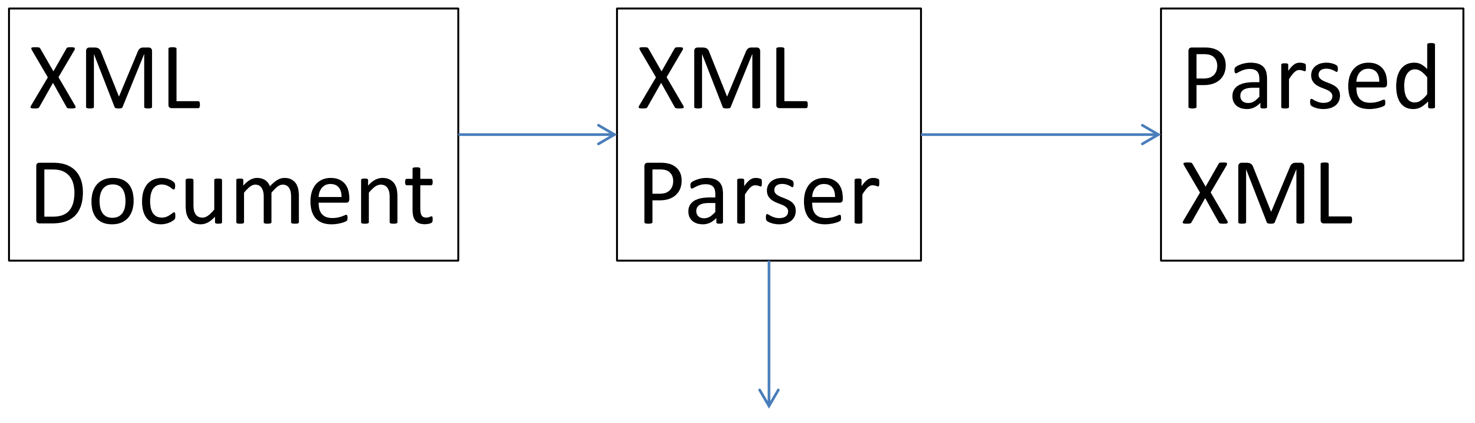
$\leftarrow$ tag for the root element

```



</some tag>
```

Well-Formed XML

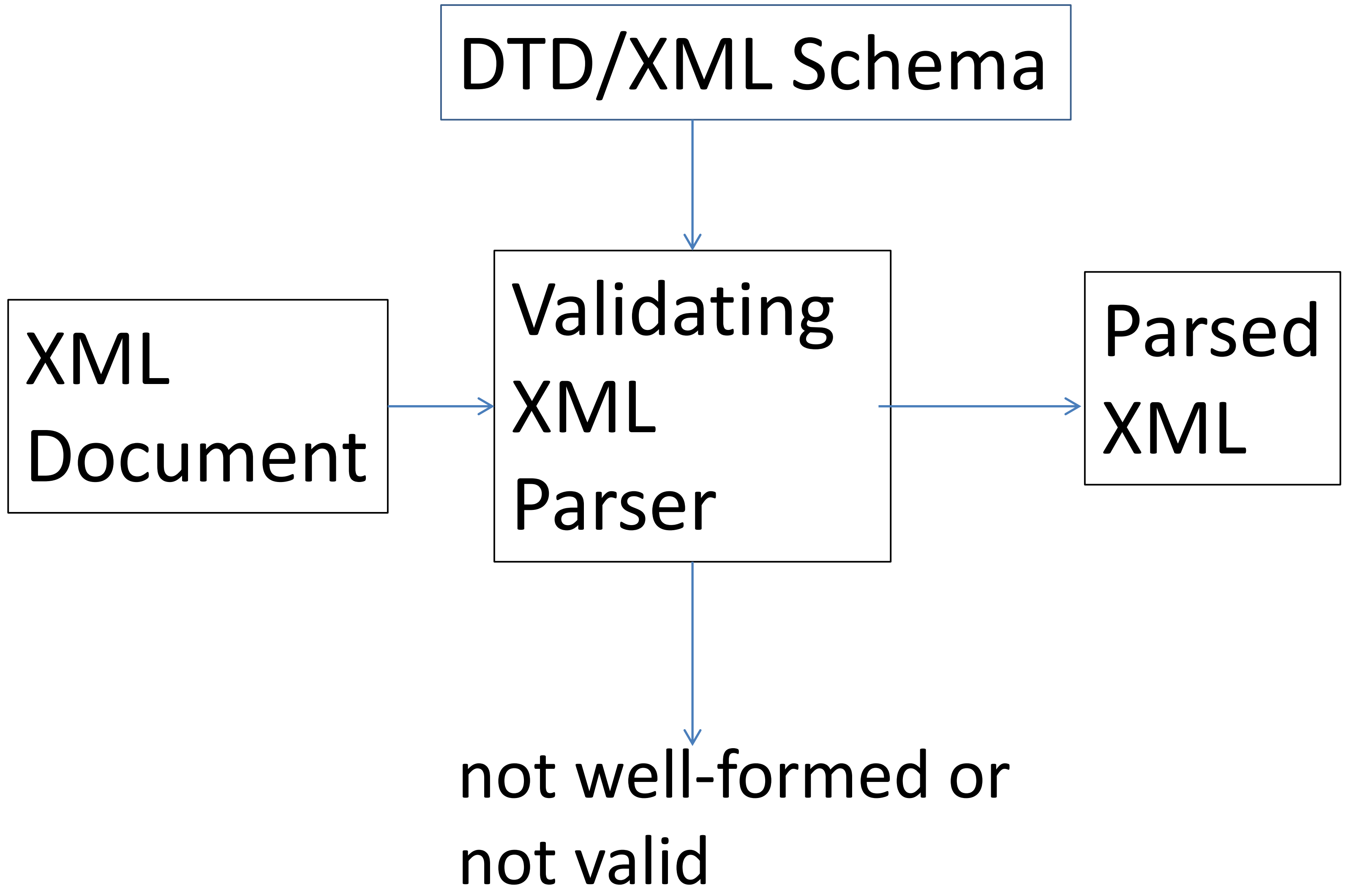


not well-formed :<

Valid XML

- Adheres to basic structural requirements
- Adheres to content-specific specification
  - Document Type Descriptor (DTD)
  - XML Schema (XSD)

Valid XML


DTD/XML Schema
XML Document
Validating XML Parser
Parsed XML
not well-formed or not valid

Valid vs. Well-formed XML

- Valid XML – benefit of typing
  - Application programs can assume structure
  - DTD/XSD can serve as specification for data exchange
  - Documentation
- Well-formed XML – flexibility, benefit of no-typing
  - Flexibility – ease of change
  - DTD/XSD can be messy for irregular data

DTD (Document Type Definitions)

- Language to describe XML schema by specifying elements, attributes, nesting, ordering and # of occurrences
- Also special attribute types for key and foreign key(s): ID and IDREF(s)

The form of a DTD

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

DTD Elements

- The description of an element consists of its name (tag), and a parenthesis containing any nested tags.
- Sub tags must appear in order shown
- Each tag may be followed by its multiplicity.
    - A*: any number of times including 0
    - A+: one or more times
    - A ?: either zero or one time, but no more
- Symbol | can connect alternative sequences of tags. Example: (A|B) means A or B, but not both.

DTD Elements: #PCDATA and EMPTY

- Leaves (text elements) have #PCDATA (*Parsed Character DATA*) in place of nested tags
  - The element has a text value and no nested element within it.
  - e.g.) <!ELEMENT Title (#PCDATA)>
- <!ELEMENT Foo EMPTY> means <Foo /> is the only available form of Foo.

Example: DTD Elements

```
<!ELEMENT Genre (Comedy| Drama|SciFi|Teen)>
<!ELEMENT Address (Street, (City|Zip))>
<!ELEMENT NAME ((TITLE?, FIRST, LAST) | IPADDR)>
```

Using a DTD

1. Set standalone = “no”.
2. Either:
   a) Include the DTD as a preamble of the XML document, or
   b) Follow DOCTYPE and the <root tag> by SYSTEM and a path to the file where the DTD can be found.

Example: (a) InternalDTD.xml

```
<?xml version = “1.0” standalone = “no” ?>
<!DOCTYPE Stars[
    <!ELEMENT Stars(Star*)>
    <!ELEMENT Star (Name, Address+)>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Address(Street, City )>
    <!ELEMENT Street (#PCDATA)>
    <!ELEMENT City (#PCDATA)>
]>
<Stars>
    <Star><Name>Carrie Fisher</Name>
        <Address><Street>123 Maple St. </Street> <City>Holly Wood</City></Address>
        <Address><Street>5 Locust Ln.</Street> <City>Malibu</City></Address>
    </Star>
    <Star> …
</Stars>
```

$\leftarrow$ The DTD

$\swarrow$ The XML document

Example: (b) ExternalDTD.xml

- Assume the Stars DTD is in file default.dtd.

```
<?xml version = “1.0” standalone = “no” ?>
<!DOCTYPE Stars SYSTEM ”default.dtd”>
<Stars>
     <Star><Name>Carrie Fisher</Name>
               <Address><Street>123 Maple St. </Street>
                                        <City>Holly Wood</City></Address>
               <Address><Street>5 Locust Ln.</Street>
                                        <City>Malibu</City></Address>
      </Star>
      <Star> …
</Stars>
```

Internal vs. External DTD

External DTD are better because of:

- possibility of sharing definitions between XML documents
- The documents that share the same DTD are more uniform and easier to retrieve

Attributes

- Opening tags in XML can have *attributes*.
- In a DTD,

`<!ATTLIST` *E* `...>`

declares attributes for element *E*, along with its data type.

Attributes

DTD:

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

XML:

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

Example: ATTLIST in DTD

- MoviesWithAttribute.dtd
- MoviesWithAttribute.xml

DTD types: ID and IDREF

DTD:

```
<!ATTLIST Star
    starID ID #REQUIRED
    starredIn  IDREF #IMPLIED >
```

XML:

```
<Star starID = “cf” starredIn “sw”>
```

DTD types: ID and IDREF

DTD:

```
<!ATTLIST Movie
   movieId  ID #REQUIRED
   starsOf   IDREFS   #IMPLIED
>
```

XML:

```
<Movie movieId = “sw”  starsOf = “cf  mh”>
```

Example: ID and IDREF

- StarMovieData.dtd
- StarMovieData.xml

XML Schema

- A more powerful way to describe the structure of XML documents.
  - Allows restrictions on the number of occurrences of sub elements
  - Allows type declarations
  - Ability to declare keys and foreign keys
- XML-Schema declarations themselves are XML documents.

Structure of an XML-Schema Document

```
<? xml version = … ?>
<xs:schema xmlns:xs =
  "http://www.w3.org/2001/XMLschema">
                    . . .
</xs:schema>
```

Defines "xs" to be the *namespace* described in the URL shown.

Interpret the meaning of schema as part of the name space xs.

Elements of XML Schema

- <xs:element name = "…" type = "…" />
  - name: the tag-name of the element being defined.
  - type: the type of the element.
    - Simple type e.g., xs:string, xs:integer, and xs:boolean
    - Complex type and Restricted Simple type that are defined in the document itself
- Use minOccurs and maxOccurs attributes to control the number of occurrences of an xs:element.

minOccurs and maxOccurs

- minOccurs: no fewer than minOccurs
- maxOccurs: no more than maxOccurs
- If there is more than one, they must all appear consecutively.
- Unbounded: no upper bound limit
- Default is one occurrence.

xs:element

In XML Schema:

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

XML:

```
<Title> Star Wars</Title>
<Year> 1977 </Year>
```

Complex Types

Several ways to construct a complex type

- xs:sequence – order matters
- xs:all – the child elements can appear in any order and that all of the child elements occur once or none of them occur.
- xs:choice – any one of the elements will appear

Complex Types

name of the complex type

```
<xs:complexType name = “movieType”>
  <xs:sequence>
```

typical sub-element of complex type

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

Note: you need a name if you want to use it for the type of multiple elements.

Alternative: Complex Types defined in an Element

no type attribute

type of element Movies,
no type name

```
<xs: element  name = “Movies”>
  <xs:complexType>
  <xs:sequence>
       <xs:element name = “Movie”  type =  “movieType”
        minOccurs = “0”  maxOccurs = “unbounded” />
  </xs:sequence>
 </xs: complexType>
</xs:element>
```

A DTD for Movies

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

Example

- MoviesValidatedBySchema.xml
- MoviesValidatedBySchema.xsd
- MoviesValidatedBySchema.dtd

Example: xs:all

```
<xs:element name="person">
<xs:complexType>
   <xs:all minOccurs = 1>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

- Defines an element named "person" which must contain the “firstname" and the “lastname" elements. They can appear in any order but both elements MUST occur once and only once!
- If exists, maxOccurs must be 1, but minOccurs can be either 0 or 1
- With minOccurs=“0”, each element CAN appear zero or one time!

e.g.)
```
<person>
        <firstname>Kimberly</firstname>
         </person>
```
is NOT valid.

Example: xs:choice

```
<xs:element name="person">
  <xs:complexType>
    <xs:choice>
      <xs:element name="employee" type="employee"/>
      <xs:element name="member" type="member"/>
    </xs:choice>
  </xs:complexType>
</xs:element>
```

- Defines an element named "person" which must contain either a "employee" element or a "member" element, not both.
- minOccurs and maxOccurs can be defined per element.

Example

- Persons.xsd
- Persons.xml

xs:attribute

- xs:attribute elements can be used within a complex type to indicate attributes of elements of that type.
- attributes of xs:attribute:
  - name
  - type
  - use = ”required” or ”optional”.

With xs:attribute

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

With sub elements

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

Example

- MoviesWithAttribute.xsd
- MoviesWithAttribute.dtd
- MoviesWithAttribute.xml

Restricted Simple Type

- Restricted simple type can be the type of elements or attributes.
- xs:simpleType can describe enumerations and range-restricted base types.
- name is an attribute
- xs:restriction is a sub element.

<xs:restriction>

- Attribute base gives the simple type to be restricted, e.g., xs:integer.
- Subelements
  - xs:{min, max}{Inclusive, Exclusive} are four attributes that can give a lower or upper bound on a numerical range.

  or

  - xs:enumeration is a subelement with attribute value that allows enumerated types.

Example (a)

```
<xs:simpleType name = "movieYearType">
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1915"/>
      <xs:maxInclusive value="2013"/>
    </xs:restriction>
  </xs:simpleType>
```

Example (b)

```
<xs:simpleType name = "movieGenreType">
  <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>
```

Example

- MoviesWithSimpleType.xml
- MoviesWithSimpleType.xsd

Keys in XML Schema

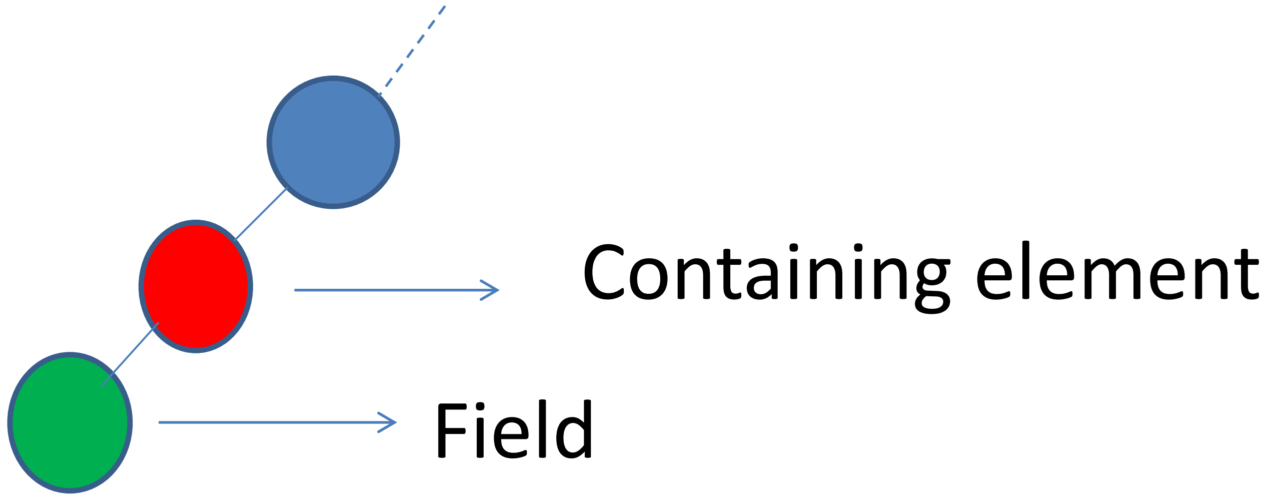
- An xs:element can have an xs:key subelement.

```
<xs:element name = element name>
   <xs:key name = key name>
      <xs:selector xpath = “path description” />
      <xs:field xpath = “path description” /> or
      <xs:field xpath = “@path description”/>
   </xs:key>
</xs:element>
```

- The key element MUST contain the following (in order):
  – one and only one selector element
  – one or more field elements to form a key. The field can be any sub element of the last element on the selector path or an attribute of the last element.

Keys in XML Schema

- Selector: Xpath to the containing element
- Field: Xpath to an attribute or element of which value (or set of values) must be a key within the containing element.



Example

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

Note: The key name “movieKey” will be used if it is restricted by a foreign key.

Example

- MoviesWithKey.xsd
- MoviesWithKey.xml

xs:key vs xs:unique

- xs:key

  The field must exist.
- xs:unique

  The field might not exist, and the constraint is only that they are unique if they exist.

Foreign Keys in XML Schema

- An xs:element can have an xs:keyref subelement.

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

- A foreign-key itself has a name.
- It refers to the name of some key or unique value.
- The selector and fields(s) are as for keys.

ID and IDREF vs <xs:keyref>

- ID’s and IDREF’s in a DTD: untyped references
- <xs:keyref>: to particular types of elements

Example:<xs:keyref>

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

Example:<xs:keyref>

- Within the Stars element, you can reach StarIn by following Star/StarIn. The Starred element contains two fields (in this case attributes): title and year. These fields together serves as a key called movieKey in the Movie element.

Example

- StarsWithKeyRef.xsd
- StarsWithKeyRef.xml
- MoviesWithKeySimple.xsd
- MoviesWithKeySimple.xml

[Open Question] Validation stopped because keyref 'movieRef's refers to out of scope key/unique. Why ?

	Relational Model	XML
Structure	Tables	Hierachical Tree
Schema	Fixed in advance, required	Flexible, “self-describing” optional
Queries	SQL	XPath, XQuery, XSLT
Ordering	None	Implied ordering