XML:

- XML stands for eXtensible Markup Language.
- **Note:** XML is not a database.
- XML is useful for moving data between databases.
- XML was designed to store and transport data.
- XML was designed to be both human and machine-readable.
- **XPath** is a language for navigating in XML documents.
- **XQuery** is a language for querying XML documents.
- **XSLT** is a language for transforming XML documents.

Well-Formed XML:

- An XML document with the correct syntax is called **Well-Formed**.
- An XML document validated against a DTD is both Well-Formed and valid.
- **DTD** stands for **Document Type Definition**.
- A DTD defines the structure and the legal elements and attributes of an XML document.
- **Well-Formed XML** allows you to invent your own tags.
- Valid XML conforms to a certain DTD.
  You can think of DTD as a schema for your XML file.
- Start the document with a declaration, surrounded by **<?xml … ?>**.
- A normal declaration is: **<?xml version = ”1.0” standalone = “yes” ?>**.
  Note that standalone = “yes” means no DTD is provided.
  standalone = “no” means that a DTD is provided.
- The balance of document is a root tag surrounding nested tags.

XML Tags:

- Tags are normally matched pairs such as **<FOO> … </FOO>**.
- Unmatched tags are also allowed such as **<FOO/>**.
- Tags may be nested arbitrarily.
  E.g.

```
<Student>
        <Name> … </Name>
        <Student_ID> … </Student_ID>
</Student>
```

- XML tags are case-sensitive.

DTD Structure:

- The purpose of a DTD is to define the structure and the legal elements and attributes of an XML document.
- The structure of a DTD looks like this:

```
<!DOCTYPE <root tag> [
   <!ELEMENT <name> (<components>)>
   More elements
]>
```

- E.g.

```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

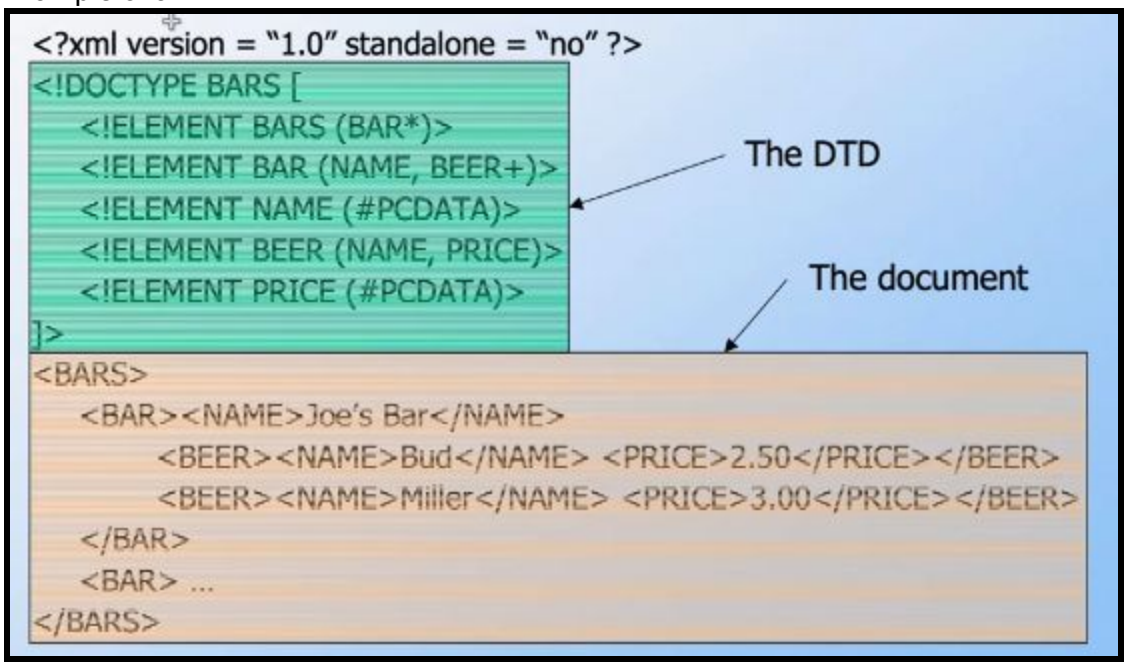
- The description of an DTD element consists of its name (tag), and a parenthesized description of any nested tags. It also includes the order of subtags and their multiplicity.

- Leaves (text elements) have #PCDATA (Parsed Character DATA) in place of nested tags.
- Subtags must appear in order shown.
- A tag may be followed by a symbol to indicate its multiplicity:
  - * = zero or more
  - + = one or more
  - ? = zero or one.
- The symbol | can connect alternative sequences of tags.
- E.g. In the example below, a name is an optional title, a first name, and a last name, in that order, or it is an IP address.

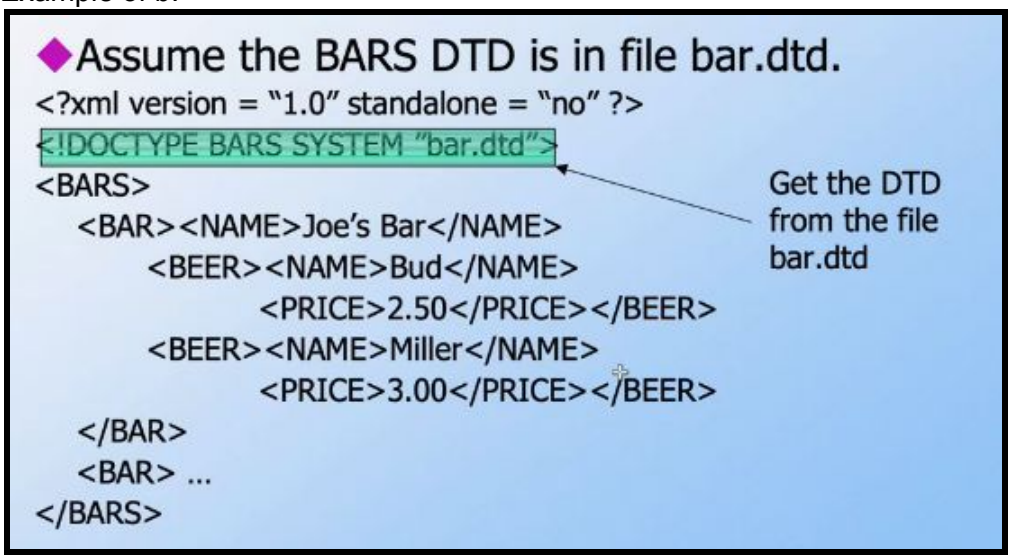
```
<!ELEMENT NAME (
  (TITLE?, FIRST, LAST) | IPADDR
)>
```

It is an optional title because we have a ? after TITLE.
It could just be an IP address because we have a |.

- If you want to use a DTD, you have to set standalone = “no”.
- We can either:
  a. Include the DTD as a preamble of the XML document.
     I.e. The DTD is included in the XML document.
     OR
  b. Follow DOCTYPE and the by SYSTEM and a path to the file where the DTD can be found.
     The DTD is a separate document but there’s a path to it in the XML document.
- Example of a:



- Example of b:



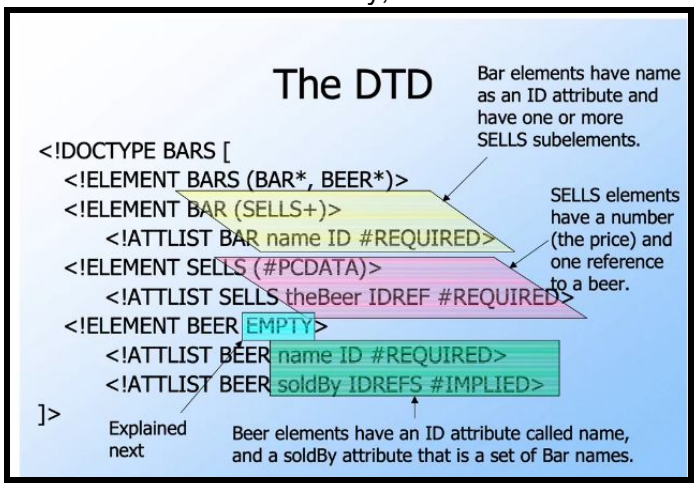
- An attribute declaration in DTD has the following syntax:
  <!ATTLIST element-name attribute-name attribute-type attribute-value>.
- E.g.
  <!ATTLIST payment type CDATA "check">
  XML example:
  <payment type="check"/>
- Table of attribute-type:

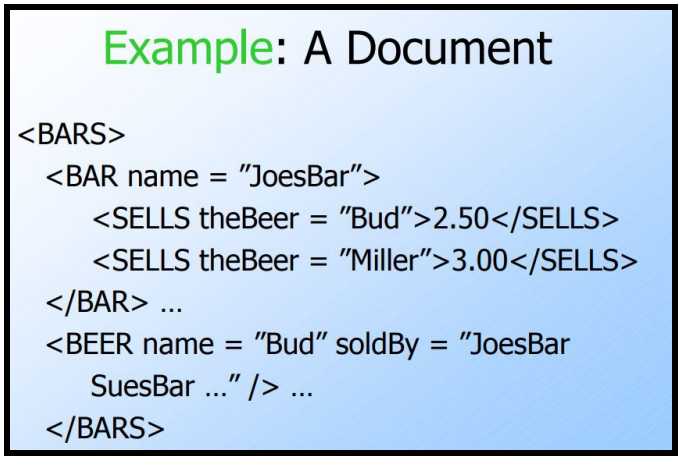
Type	**Description**
CDATA	The value is character data
(en1|en2|..)	The value must be one from an enumerated list
ID	The value is a unique id
IDREF	The value is the id of another element
IDREFS	The value is a list of other ids
NMTOKEN	The value is a valid XML name
NMTOKENS	The value is a list of valid XML names
ENTITY	The value is an entity
ENTITIES	The value is a list of entities
NOTATION	The value is a name of a notation
xml:	The value is a predefined xml value

- Table of attribute-value:

Value	**Explanation**
value	The default value of the attribute
#REQUIRED	The attribute is required
#IMPLIED	The attribute is optional
#FIXED value	The attribute value is fixed

- Attributes can be pointers from one object to another.
- IDs and IDREFs allow the structure of an XML document to be a general graph, rather than just a tree.
- E.g.
  A new BARS DTD includes both BAR and BEER subelements.
  BARS and BEERS have ID attributes name.
  BARS have SELLS subelements, consisting of a number (the price of one beer) and an IDREF theBeer leading to that beer.
  BEERS have attribute soldBy, which is an IDREFS leading to all the bars that sell it.



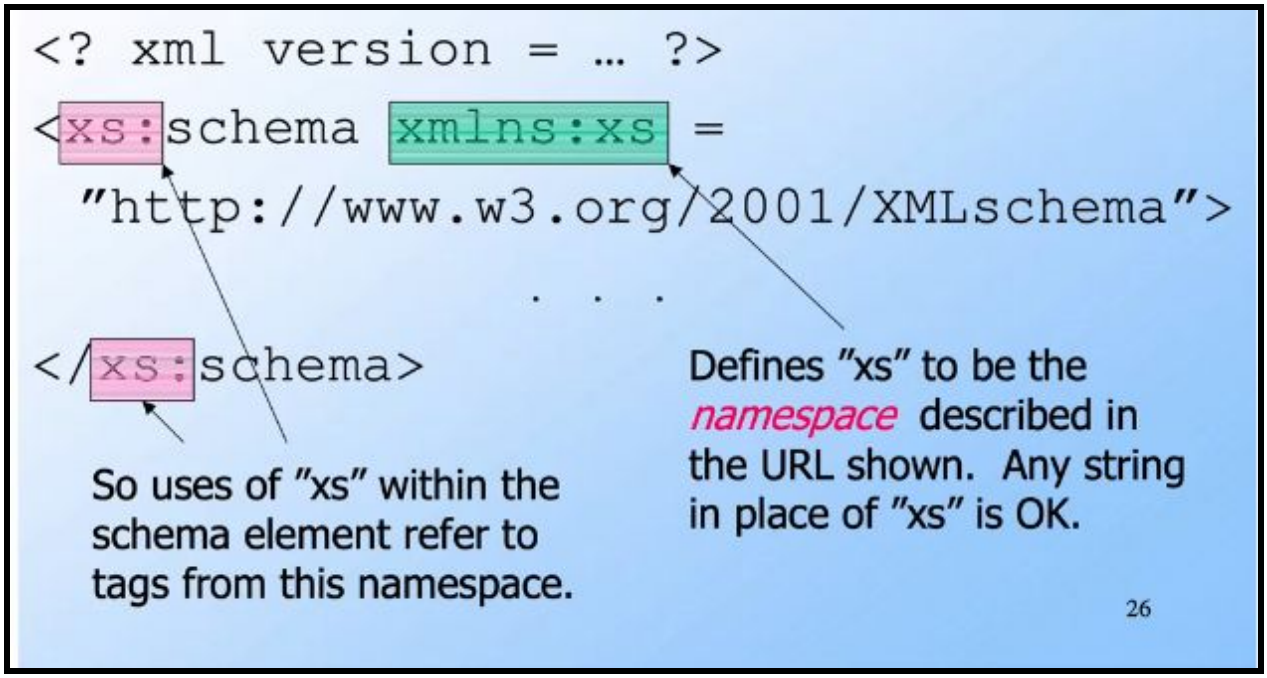


- Empty elements are declared with the category keyword EMPTY.

- E.g.
  <!ELEMENT element-name EMPTY>
- With empty elements, we can do all the work of an element in its attributes.

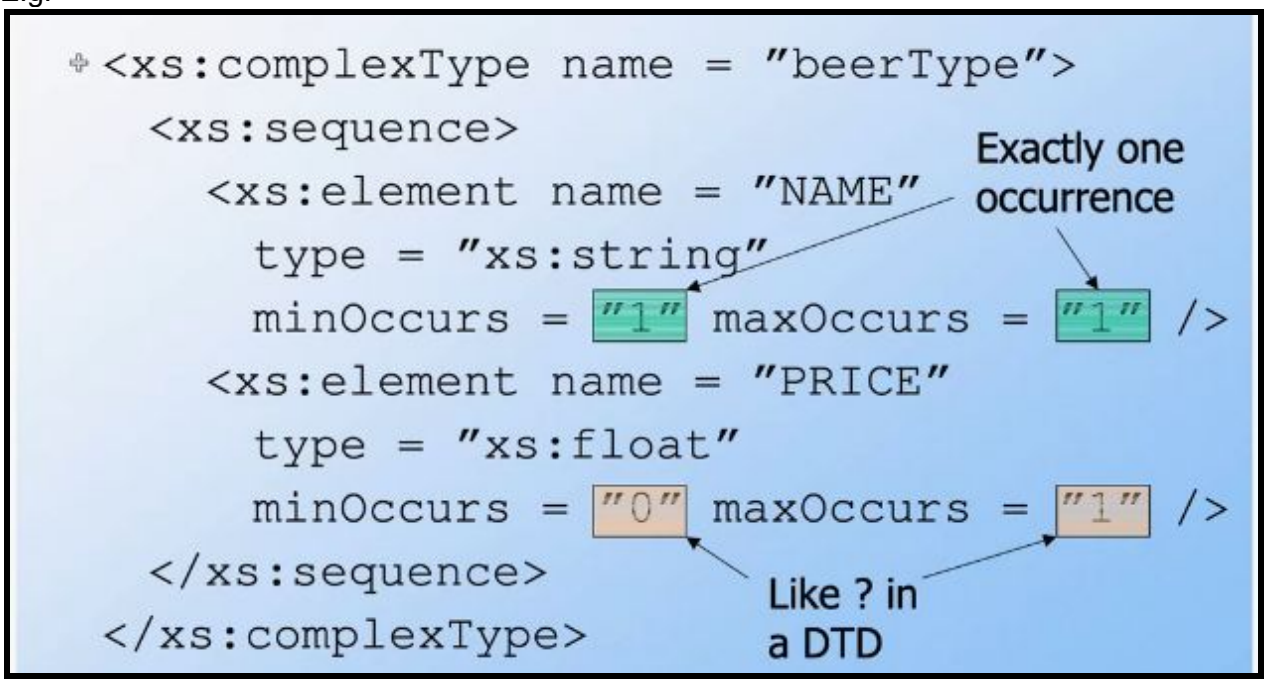
XML Schema:

- A more powerful way to describe the structure of XML documents.
- XML-Schema declarations are themselves XML documents.
  They describe “elements” and the things doing the describing are also “elements.”
- Here is the structure of an XML schema:



- A **simple element** is an XML element that can contain only text. The text can be of many different types, such as string, decimal, integer, boolean, etc.
  The syntax for defining a simple element is:
  <xs:element name="xxx" type="yyy"/>
  where xxx is the name of the element and yyy is the data type of the element.
  The data type could be the name of a type defined in the document itself.
  XML Schema has a lot of built-in data types. The most common types are:
  - xs:string
  - xs:decimal
  - xs:integer
  - xs:boolean
  - xs:date
  - xs:time
- E.g.
  <xs:element name = ”NAME” type = ”xs:string” />
- To describe elements that consist of subelements, we use xs:complexType.
- The complexType element defines a complex type. A complex type element is an XML element that contains other elements and/or attributes.
- The attribute name gives a name to the type.
- A typical subelement of a complex type is xs:sequence, which itself has a sequence of xs:element subelements.
- We can use the minOccurs and maxOccurs attributes to control the number of occurrences of an xs:element.

- E.g.



- xs:attribute elements can be used within a complex type to indicate attributes of elements of that type.
- attributes of xs:attribute:
  - name and type as for xs:element.
  - use = ”required” or ”optional”.
- E.g.
  <xs:attribute name="lang" type="xs:string" use="required"/>
- E.g.

```
<xs:complexType name = "beerType">
  <xs:attribute name = "name"
     type = "xs:string"
     use = "required" />
  <xs:attribute name = "price"
     type = "xs:float"
     use = "optional" />
</xs:complexType>
```

- An xs:element can have an xs:key subelement. This means that within this element, all subelements reached by a certain selector path will have unique values for a certain combination of fields.
- An xs:keyref subelement within an xs:element says that within this element, certain values (defined by selector and field(s), as for keys) must appear as values of a certain key.

XPath:

- In XPath, there are seven kinds of nodes: element, attribute, text, namespace, processing-instruction, comment, and document nodes.
- XML documents are treated as trees of nodes.
  The topmost element of the tree is called the **root element**.
- E.g. Consider the XML snippet below.

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

  <bookstore> is the root element node.
  <author>J K. Rowling</author> is an element node.
  lang="en" is an attribute node.

- Table of XPath expressions

Expression	Description
node-name	Select all nodes with the given name "nodename"
/	Selection starts from the root node.
//	Selection starts from the current node that matches the selection.
.	Selects the current node.
..	Selects the parent of the current node.
@	Selects attributes.

- E.g. Consider the XML snippet below.

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
```

```
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>

</bookstore>
```

The below table shows examples of XPath expressions.

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore".
/bookstore	Selects the root element bookstore. **Note:** If the path starts with a slash (/) it always represents an absolute path to an element.
bookstore/book	Selects all book elements that are children of the bookstore.
//book	Selects all book elements no matter where they are in the document.
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element.
//@lang	Selects all attributes that are named lang.

XQuery:

- XQuery is to XML what SQL is to databases.
- XQuery is designed to query XML data.
- XQuery is an expression language.
  Each expression operates on and returns sequences of elements.
- XQuery is case-sensitive.
- XQuery elements, attributes, and variables must be valid XML names.
- An XQuery string value can be in single or double quotes.
- An XQuery variable is defined with a $ followed by a name. E.g. $bookstore
- XQuery comments are delimited by (: and :). E.g. (: This is a XQuery Comment :)
- XQuery is built on XPath expressions.
- **FLWOR Expression:**
  - Syntax:
    For $var in expr
    Let $var := expr
    Where condition
    Order By expr
    Return Expr
  - For: Selects a sequence of nodes.
  - Let: Binds a sequence to a variable.
  - Where: Filters the nodes.
  - Order by: Sorts the nodes.
  - Return: What to return (gets evaluated once for every node).
  - **Note:** Everything except the Return clause is optional.
  - **Note:** The For and Let clauses can be repeated and interleaved.

- We can mix query results with xml data.
  E.g. <result> {query result} </result>
- **Comparison Operators:**

	Value Comparison	General Comparison
equals	eq	=
not equals	ne	!=
less than	lt	<
greater than	gt	>
less than or equal to	le	<=
greater than or equal to	ge	>=

- **General comparison operators** can be used to compare atomic values, sequences, or any combination of the two.
- When you are comparing two sequences by using general comparison operators and a value exists in the second sequence that compares True to a value in the first sequence, the overall result is True. Otherwise, it is False.
  E.g. (1, 2, 3) = (3, 4) is True, because the value 3 appears in both sequences.
- **Value comparison operators** are used to compare atomic values.
- If the two values compare the same according to the chosen operator, the expression will return True. Otherwise, it will return False. If either value is an empty sequence, the result of the expression is False.

XSLT:

- **XSL (eXtensible Stylesheet Language)** is a styling language for XML.
- XSLT stands for XSL Transformations.
- It is similar to XML as CSS is to HTML.

XSLT Template:

- The <xsl:template> element is used to build templates.
- Syntax:

```
<xsl:template
  name = Qname
  match = Pattern
  priority = number>
...
</xsl:template>
```

- The match attribute is used to associate a template with an XML element.
  The value of the match attribute is an XPath expression.
  Note: match="/" defines the whole document.
- Table of attributes:

Name	Description
Name	The name of the element on which the template is to be applied. If this is present, the match attribute becomes optional.
Match	The match attribute is used to associate a template with an XML element.

Priority	If there are several xsl:template elements that all match the same node, the one that is chosen is determined by the optional priority attribute. The template with highest priority wins. The priority is written as a floating-point number. The default priority is 1. If two matching templates have the same priority, the one that appears last in the stylesheet is used.

XSLT Value-of:

- The <xsl:value-of> element is used to extract the value of a selected node.
- The <xsl:value-of> element can be used to extract the value of an XML element and add it to the output stream of the transformation.
- Syntax:
  <xsl:value-of select = XPath_Expression/>
- Table of attributes:

Name	Description
Select	The XPath expression to be evaluated in the current context. I.e. The select attribute contains an XPath expression.

- E.g.
  <xsl:value-of select="title"/>

XSLT For-Each:

- The <xsl:for-each> element allows you to do looping in XSLT.
- The XSL <xsl:for-each> element can be used to select every XML element of a specified node-set.
- Syntax:
  <xsl:for-each select = XPath_Expression>
  ...
  </xsl:for-each>
- Table of attributes:

Name	Description
Select	The XPath Expression to be evaluated in the current context to determine the set of nodes to be iterated. I.e. The value of the select attribute is an XPath expression.

- E.g.

```
<xsl:for-each select="catalog/cd">
        <xsl:value-of select="title"/>
        <xsl:value-of select="artist"/>
</xsl:for-each>
```

XSLT If:

- The <xsl:if> element is used to put a conditional test against the content of the XML file. I.e. The <xsl:if> tag specifies a conditional test against the content of nodes.
- Syntax:

```
<xsl:if test = boolean-expression>
...
</xsl:if>
```

- Table of attributes:

Name	Description
Test	The condition in the xml data to test. The value of the required test attribute contains the expression to be evaluated.

- E.g.

```
<xsl:if test="price &gt; 10">
        <xsl:value-of select="title"/></td>
        <xsl:value-of select="artist"/></td>
        <xsl:value-of select="price"/></td>
</xsl:if>
```

XSLT Comparison Operators:

Operator	Description
$gt;	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
=	Equals
!=	Not equal