XML & Query Languages for XML

XML Basics

DTD,XSD

XPath

XQuery

XSLT

Some of the slides are taken from ***Jennifer Widom's «Database Systems:The Complete Book», Standford Univ.***

eXtensible Markup Language : XML

- Text tabanlı, sıralı formatta, Genişletilebilir işaretleme dili : uygulamaya uygun olarak yeni işaretler eklenebilir.

```
<message>
        <text> Hello, XML! </text>
</message>
```

- Programlama dili DEĞİL.
- **S**tandard **G**eneralized **M**arkup **L**anguage (SGML) : elektronik dokümanların yapısı ve içeriğini tanımlamak için kullanılan uluslararası standartta kapsamlı bir işaretleme dili.
- XML ⊂ SGML *(HyperTextML ⊂ SGML)*
- XML: veri saklanması, organizasyonu ve sorgulanması için işaretleme dili. *HTML: sadece veri sunumu için işaretleme dili.*

- XML yaygın kullanımı:
  - Farklı platformlarda/sistemler arası veri iletişimi standardı
  - Veri tabanlarının entegrasyonu

XML syntax

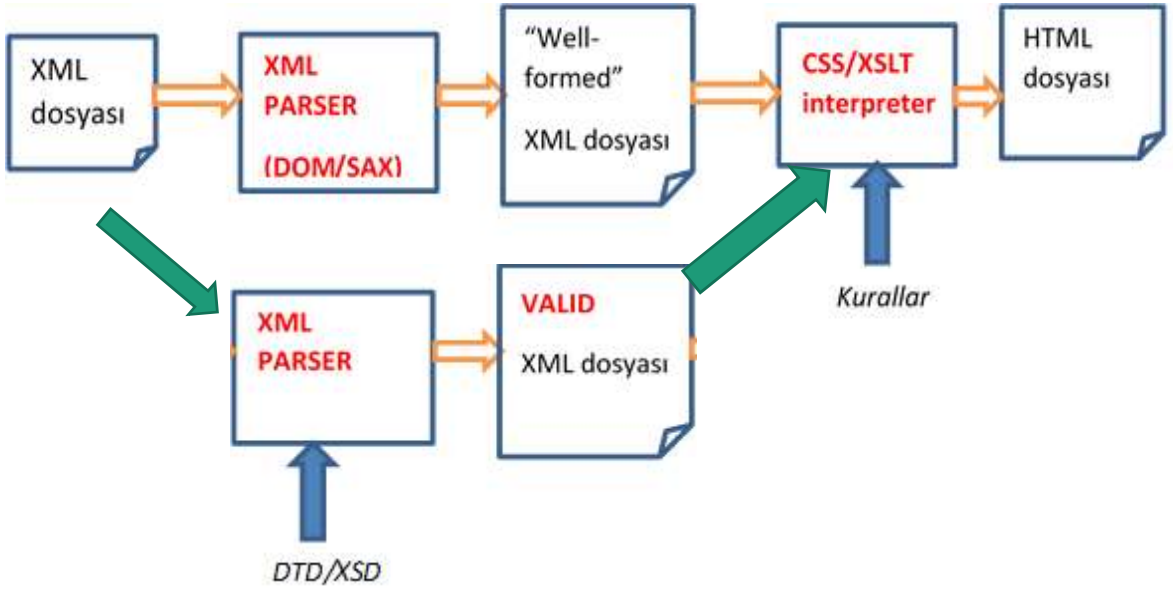
```
<?xml version = "1.0" encoding = "UTF-8"?>
<contact-info href = "http://www.mycompany.com/" >
        <company> MyCompany</company>
        <empty_tag/>
</contact-info>
```

- *İlk başlık satırı: XML beyanı (declaration)*
- XML-tags : start-tag, end-tag, empty-tag
- **XML-elements** == XML-nodes
- İçiçe (nested) element ler.
- **Kök element**
- Her zaman case-sensitive: <contact-info> ≠ <Contact-Info>
- XML-attributes

Well-formed XML & VALID XML

- Önceden tanımlı bir şemaya yok
- Tek kök olması
- start-tag ve end-tag'ların eşleşmesi
- Element nitelikleri (attribute), element içinde biricik olması

- İçerik yapılanması belli kurallara göre yapılması. Bunu belirleyen *(önceden tanımlı şema)* gramerler/diller:
  - DTD
  - XML Schema (XSD)



Ornek XML dosyası: (well formed)

```
<?xml version="1.0" ?>
<!--Bookstore with no DTD-->
<Bookstore>
    <Book ISBN="ISBN-0-13-713526-2" Price="85" Edition="3rd">
        <Title>A First Course in Database Systems</Title>
        <Authors>
            <Author>
                <First_Name>Jeffrey</First_Name>
                <Last_Name>Ullman</Last_Name>
            </Author>
            <Author>
                <First_Name>Jennifer</First_Name>
                <Last_Name>Widom</Last_Name>
            </Author>
        </Authors>
    </Book>
    <Book ISBN="ISBN-0-13-815504-6" Price="100">
        <Remark>  Buy this book bundled with "A First Course" - a great deal! </Remark>
        <Title>Database Systems: The Complete Book</Title>
        <Authors>
            <Author>
                <First_Name>Hector</First_Name>
                <Last_Name>Garcia-Molina</Last_Name>
            </Author>
            <Author>
                <First_Name>Jeffrey</First_Name>
                <Last_Name>Ullman</Last_Name>
            </Author>
            <Author>
                <First_Name>Jennifer</First_Name>
                <Last_Name>Widom</Last_Name>
            </Author>
        </Authors>
    </Book>
</Bookstore>
```

- Esnek içerik organizasyonu (flexible schema):
  - Edition bir kitapta var diğerinde yok !
  - Bir kitabın 2 yazarı var, diğerinin 3 yazarı var.
  - Bir kitabın Remark alt elementi var; diğerinin yok

RM - XML karşılaştırması

	RM	XML
Tasarım	ER modeli	**SSDM(yarı yapısal veri modeli): Graf**
Structure	Tablolar	Ağaç
Schema	Önceden sabit	Esnek (içerik ile beraber)
Query	SQL ☺	Xpath,Xquery ☹
Ordering	Yok	Sıralı text
İmpl.	Yerleşik,Çok başarılı	Eklenti tarzında

Structured, semi-structured, and unstructured data

- **Structured data**
  - Strict format (predefined schema)
  - Disadv: In real world, not all data collected is structured
  - **Ex: Relational Model**
- **Semi-structured data**
  - Data may have certain structure but not all information collected has identical structure
  - No exact pre-defined schema :
    - Semi-structured data *(names of attributes, relationships, and classes)* is mixed in with its schema (self-describing data)
    - **Can be displayed as a graph**
  - Some attributes may exist in some of the entities of a particular type but not in others
  - **Ex: XML**
- **Unstructured data**
  - Very limited indication of data type
  - No schema information
    - E.g., a simple text document
    - **HTML**

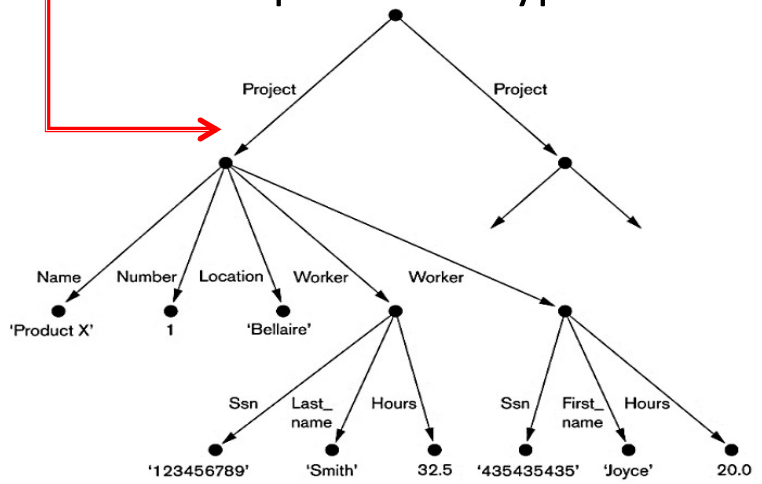


Figure 26.1
Representing semistructured data as a graph.

Unstructured data:

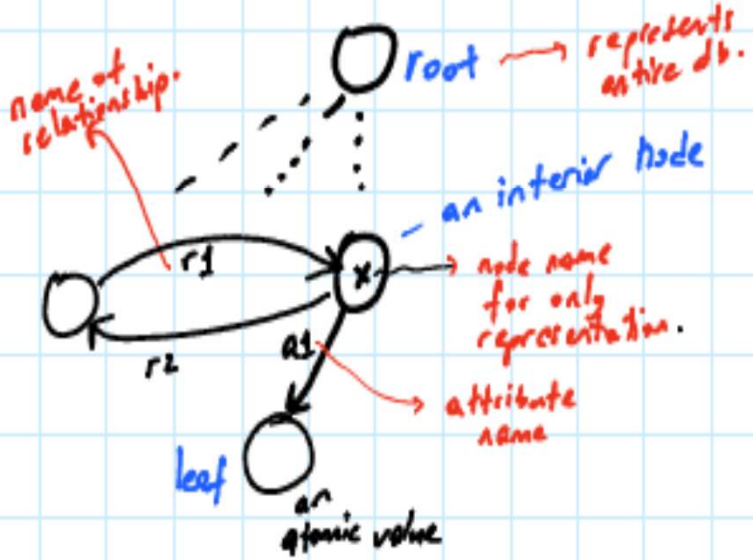
- Limited indication of data types
  - E.g., web pages in html contain some unstructured data
  - Figure shows part of HTML document representing unstructured data
- **Diffucult to interpret by computer programs BECAUSE no schema (type of data) information is known.**
  - **XML, conversely, provides easier interpretation and exchange Web documents b/w computers.**

```
<HTML>
  <HEAD>
  . . .
  </HEAD>
  <BODY>
    <H1>List of company projects and the employees in each project</H1>
    <H2>The ProductX project:</H2>
    <TABLE width="100%" border=0 cellpadding=0 cellspacing=0>
      <TR>
        <TD width="50%"><FONT size="2" face="Arial">John Smith:</FONT></TD>
        <TD>32.5 hours per week</TD>
      </TR>
      <TR>
        <TD width="50%"><FONT size="2" face="Arial">Joyce English:</FONT></TD>
        <TD>20.0 hours per week</TD>
      </TR>
    </TABLE>
    <H2>The ProductY project:</H2>
    <TABLE width="100%" border=0 cellpadding=0 cellspacing=0>
      <TR>
        <TD width="50%"><FONT size="2" face="Arial">John Smith:</FONT></TD>
        <TD>7.5 hours per week</TD>
      </TR>
      <TR>
        <TD width="50%"><FONT size="2" face="Arial">Joyce English:</FONT></TD>
        <TD>20.0 hours per week</TD>
      </TR>
      <TR>
        <TD width= "50%"><FONT size="2" face="Arial">Franklin Wong:</FONT></TD>
        <TD>10.0 hours per week</TD>
      </TR>
    </TABLE>
  . . .
  </BODY>
</HTML>
```

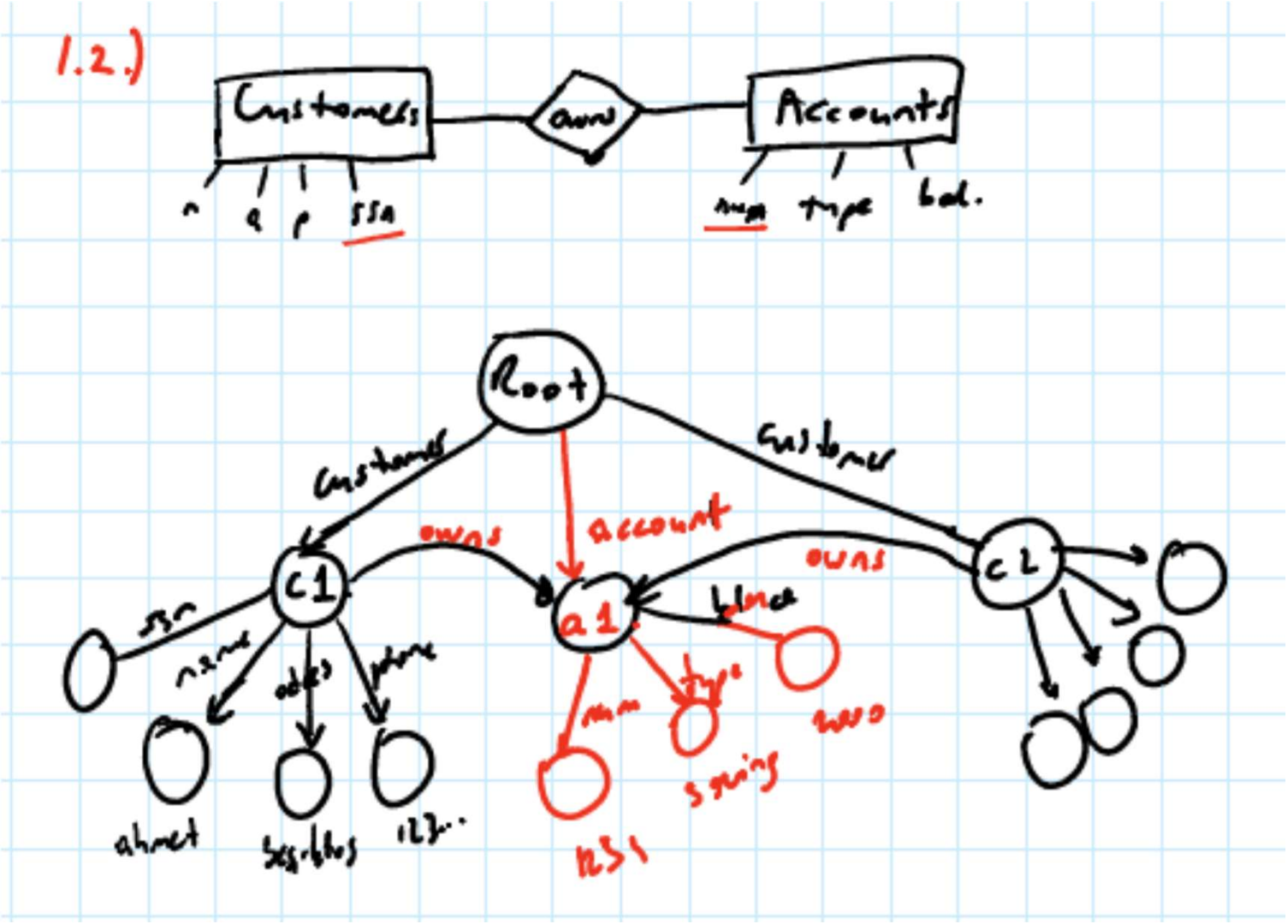
Figure 26.2
Part of an HTML document representing unstructured data.

Yarı-yapısal Veri modeli tasarımı

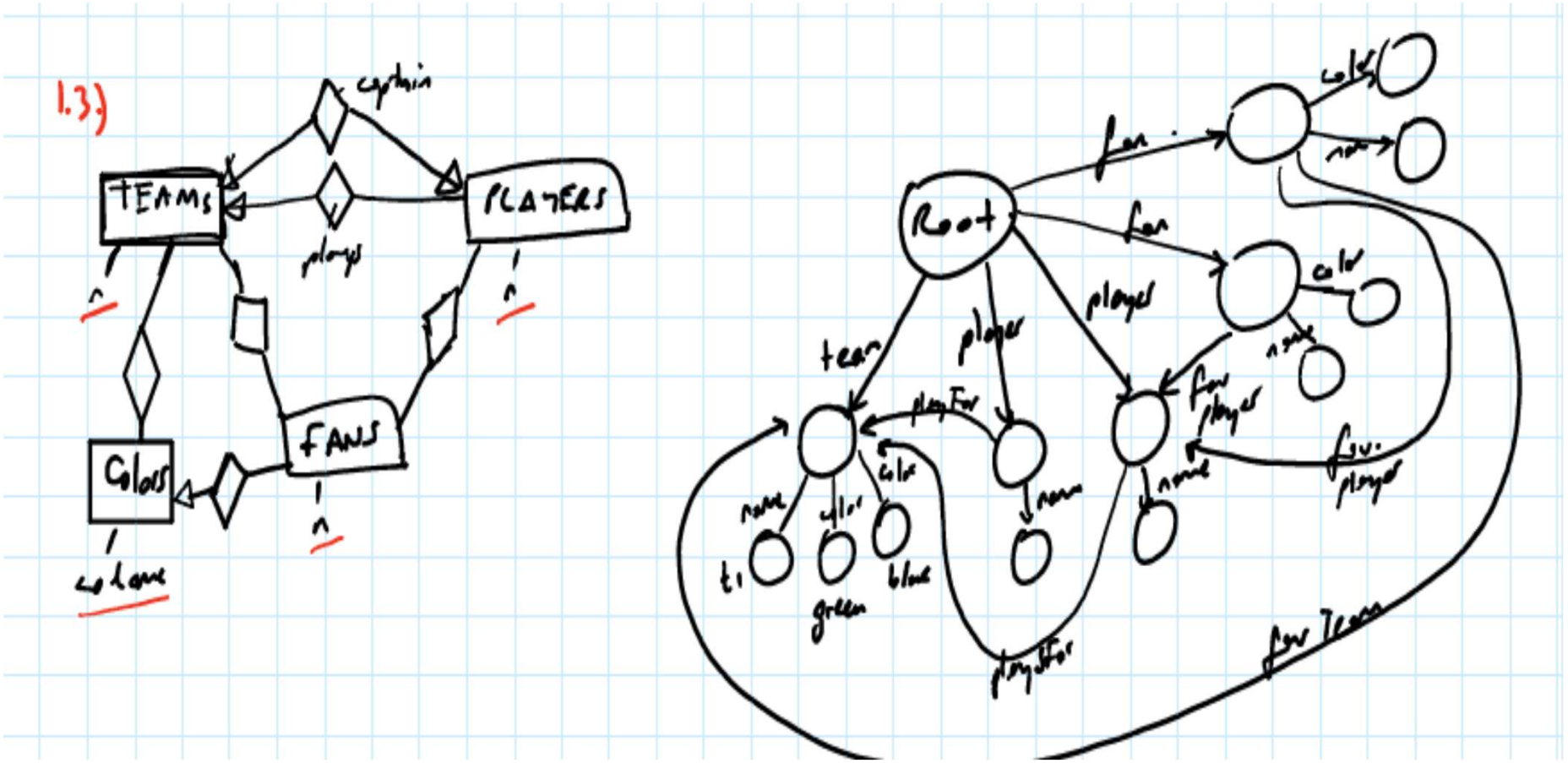
- SSD representation:
  - graph model with nodes & arcs. (UML is also graph model)
  - Nodes: leaf or interior
  - Leaf nodes hold atomic data: numeric or string. (UML nodes represent entity set)
  - Interior nodes have arcs b/w each other representing the relationships or the name of attributes.
  - Root represents entire db.



Örnek:


1.2.)
Customers
owns
Accounts
n
a
p
SSA
type
bal.
Root
customer
customer
owns
account
owns
C1
C2
a1
balance
name
address
phone
type
ahmet
balance
string
int

Örnek:



VALID XML : «veri iletişiminde XML»

- İçerik yapılanması belli kurallara göre yapılması:
  Elementlerin/niteliklerin belirlenmesi, nitelik tipleri, içiçe yapılandırılması, sıralanma durumları, adetleri, zorunlu/seçimli olmaları, anahtarların belirlenmesi, elementler arası imaların belirlenmesi:
  - DTD
  - XML Schema Descriptor (XSD)

	DTD	XSD
Genel yapı	Gramer esaslı	XML ile ifade
Tip	Veri tipi yok	Veri tipleri tanımlı
İma	İma edilen tip tanımlı değil	İma edilen tip tanımlı
Element Adetleri	*, ?	Daha detaylı belirlenebilir
…..	Karmaşık	Çok daha karmaşık

DTD (Document Type Definition)

```
<!DOCTYPE root-tag [
        <!ELEMENT element-tag (nested-element-tags)>
        ……
         <!ATTLIST  element-tag   att-name CDATA #REQUIRED
                                  ……..
                                   att-name CDATA #IMPLIED>
        …..
        <!ELEMENT element-tag (#PCDATA)>
        <!ELEMENT element-tag EMPTY>
]>
```

DTD- içiçe elementler, attribute tipleri

Reg.Exp.:

*: >= 0

+: >0

?: 0 veya 1

|: veya

<!ELEMENT Bookstore (Book | Magazine)*>

…

<!ELEMENT Book (Title, Authors, Remark?)>

<!ATTLIST Book ISBN ID #REQUIRED

Price CDATA #REQUIRED

Edition CDATA #IMPLIED

Authors IDREFS #IMPLIED

Genre (Comedy|drama|sci) >

- Nitelik tipleri:
  - CDATA
  - ID
  - IDREF(S)
  - ENUM

Örnek1:

```
<!DOCTYPE Bookstore [
  <!ELEMENT Bookstore (Book | Magazine)*>
  <!ELEMENT Book (Title, Authors, Remark?)>
  <!ATTLIST Book ISBN CDATA #REQUIRED
                 Price CDATA #REQUIRED
                 Edition CDATA #IMPLIED>
  <!ELEMENT Magazine (Title)>
  <!ATTLIST Magazine
            Month CDATA #REQUIRED
            Year CDATA #REQUIRED>
  <!ELEMENT Title (#PCDATA)>
  <!ELEMENT Authors (Author+)>
  <!ELEMENT Remark (#PCDATA)>
  <!ELEMENT Author (First_Name,Last_Name)>
  <!ELEMENT First_Name (#PCDATA)>
  <!ELEMENT Last_Name (#PCDATA)>
]>
```

```
<Bookstore>
  <Book ISBN="ISBN-0-13-713526-2" Price="85" Edition="3rd">
    <Title>A First Course in Database Systems</Title>
    <Authors>
      <Author>
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
      </Author>
      <Author>
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
      </Author>
    </Authors>
  </Book>
  <Book ISBN="ISBN-0-13-815504-6" Price="100">
    <Title>Database Systems: The Complete Book</Title>
    <Authors>
      <Author>
        <First_Name>Hector</First_Name>
        <Last_Name>Garcia-Molina</Last_Name>
      </Author>
      <Author>
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
      </Author>
      <Author>
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
      </Author>
    </Authors>
    <Remark>
      Buy this book bundled with "A First Course" - a great deal!
    </Remark>
  </Book>
</Bookstore>
```

Örnek2:

```
<!DOCTYPE Bookstore [
    <!ELEMENT Bookstore (Book*, Author*)>
    <!ELEMENT Book (Title, Remark?)>
    <!ATTLIST Book ISBN ID #REQUIRED
                   Price CDATA #REQUIRED
                   Authors IDREFS #REQUIRED>
    <!ELEMENT Title (#PCDATA)>
    <!ELEMENT Remark (#PCDATA | BookRef)*>
    <!ELEMENT BookRef EMPTY>
    <!ATTLIST BookRef book IDREF #REQUIRED>
    <!ELEMENT Author (First_Name, Last_Name)>
    <!ATTLIST Author Ident ID #REQUIRED>
    <!ELEMENT First_Name (#PCDATA)>
    <!ELEMENT Last_Name (#PCDATA)>]>
```

```
<Bookstore>
    <Book ISBN="ISBN-0-13-713526-2" Price="100" Authors="JU JW">
        <Title>A First Course in Database Systems</Title>
    </Book>
    <Book ISBN="ISBN-0-13-815504-6" Price="85" Authors="HG JU JW">
        <Title>Database Systems: The Complete Book</Title>
        <Remark> Amazon.com says: Buy this book bundled with
            <BookRef book="ISBN-0-13-713526-2" /> - a
            great deal!
        </Remark>
    </Book>
    <Author Ident="HG">
        <First_Name>Hector</First_Name>
        <Last_Name>Garcia-Molina</Last_Name>
    </Author>
    <Author Ident="JU">
        <First_Name>Jeffrey</First_Name>
        <Last_Name>Ullman</Last_Name>
    </Author>
    <Author Ident="JW">
        <First_Name>Jennifer</First_Name>
        <Last_Name>Widom</Last_Name>
    </Author>
</Bookstore>
```

The XPath/XQuery Data Model

- Corresponding to the fundamental "relation" of the relational model is: *sequence of items*.
- An *item* is either:
  1. A primitive value, e.g., integer or string.
  2. A *node*::::
     1. *Document nodes* represent entire documents.
     2. *Elements* are pieces of a document consisting of some opening tag, its matching closing tag (if any), and everything in between.
     3. *Attributes* names that are given values inside opening tags.

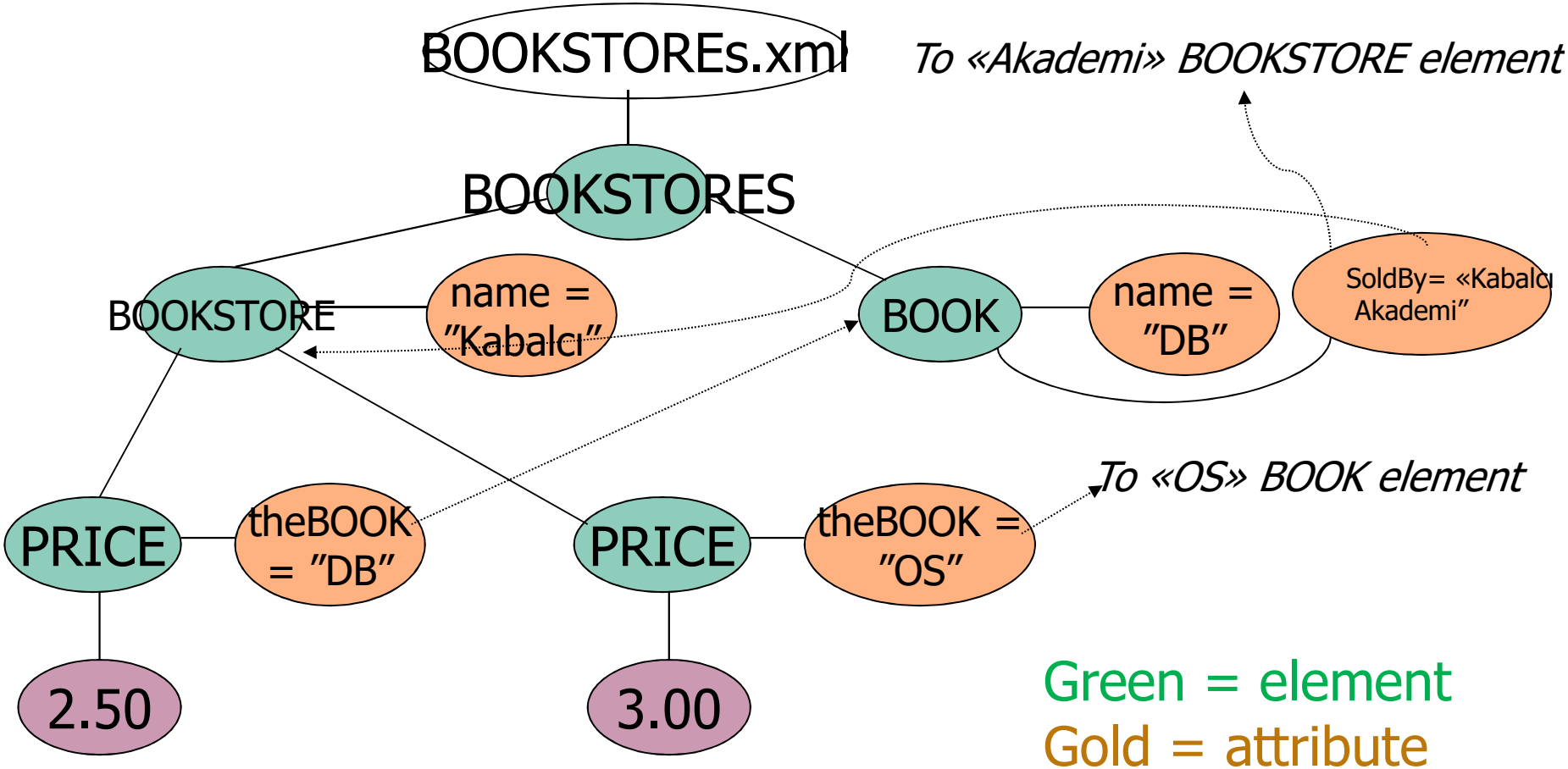
Document Nodes

- **Formed by doc(URL)** or document(URL).
- Example: doc(/usr/class/…/BOOKSTOREs.xml)
- All XPath (and XQuery) queries refer to a **doc node**, either explicitly or implicitly.

DTD for Running Example: BOOKSTORES database

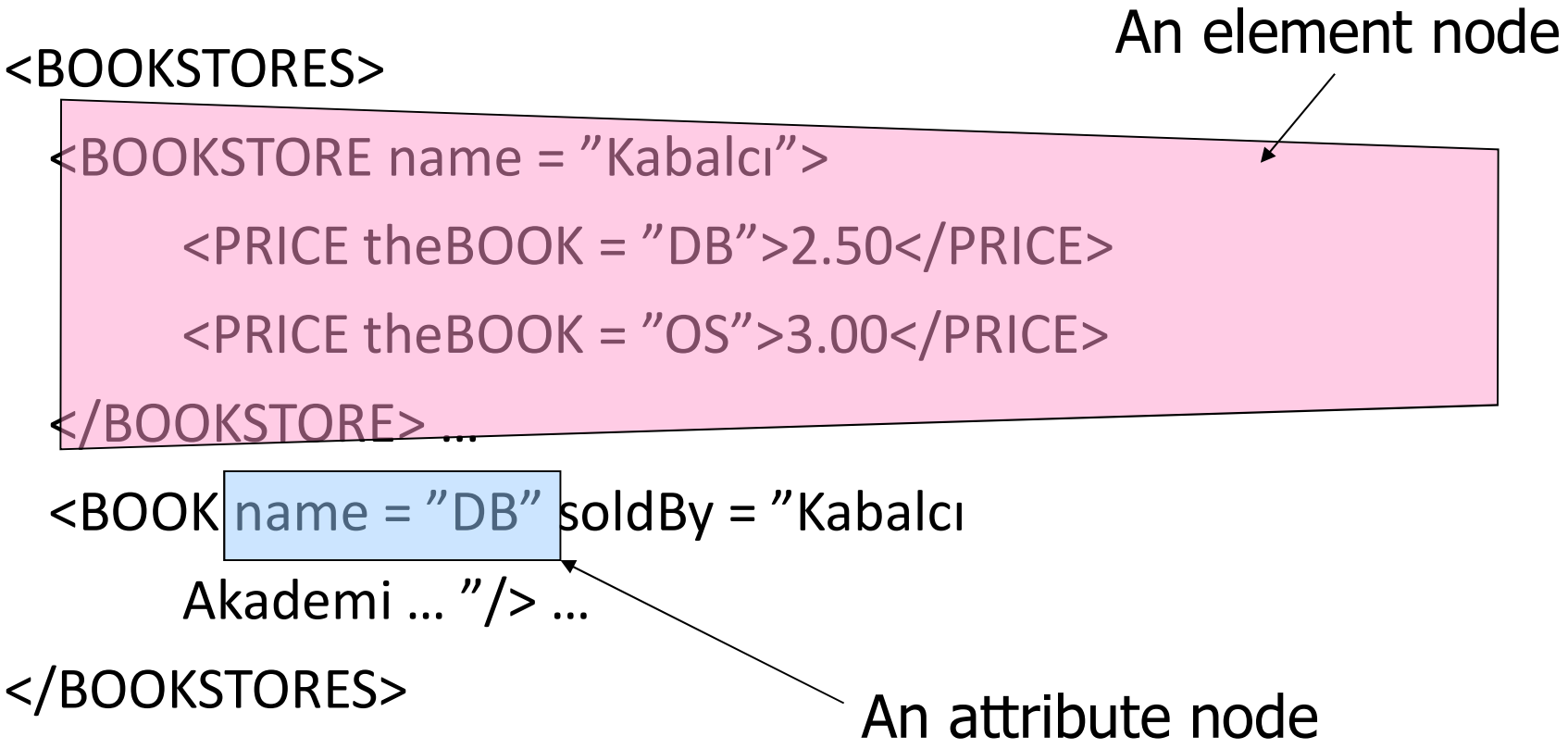
```
<!DOCTYPE BOOKSTORES [
  <!ELEMENT BOOKSTORES (BOOKSTORE*, BOOK*)>
  <!ELEMENT BOOKSTORE (PRICE+)>
        <!ATTLIST BOOKSTORE name ID #REQUIRED>
  <!ELEMENT PRICE (#PCDATA)>
        <!ATTLIST PRICE theBOOK IDREF #REQUIRED>
  <!ELEMENT BOOK EMPTY>
        <!ATTLIST BOOK name ID #REQUIRED>
        <!ATTLIST BOOK soldBy IDREFS #IMPLIED>
]>
```

Nodes as Semistructured Data



Green = element
Gold = attribute
Purple = primitive value

Example Document



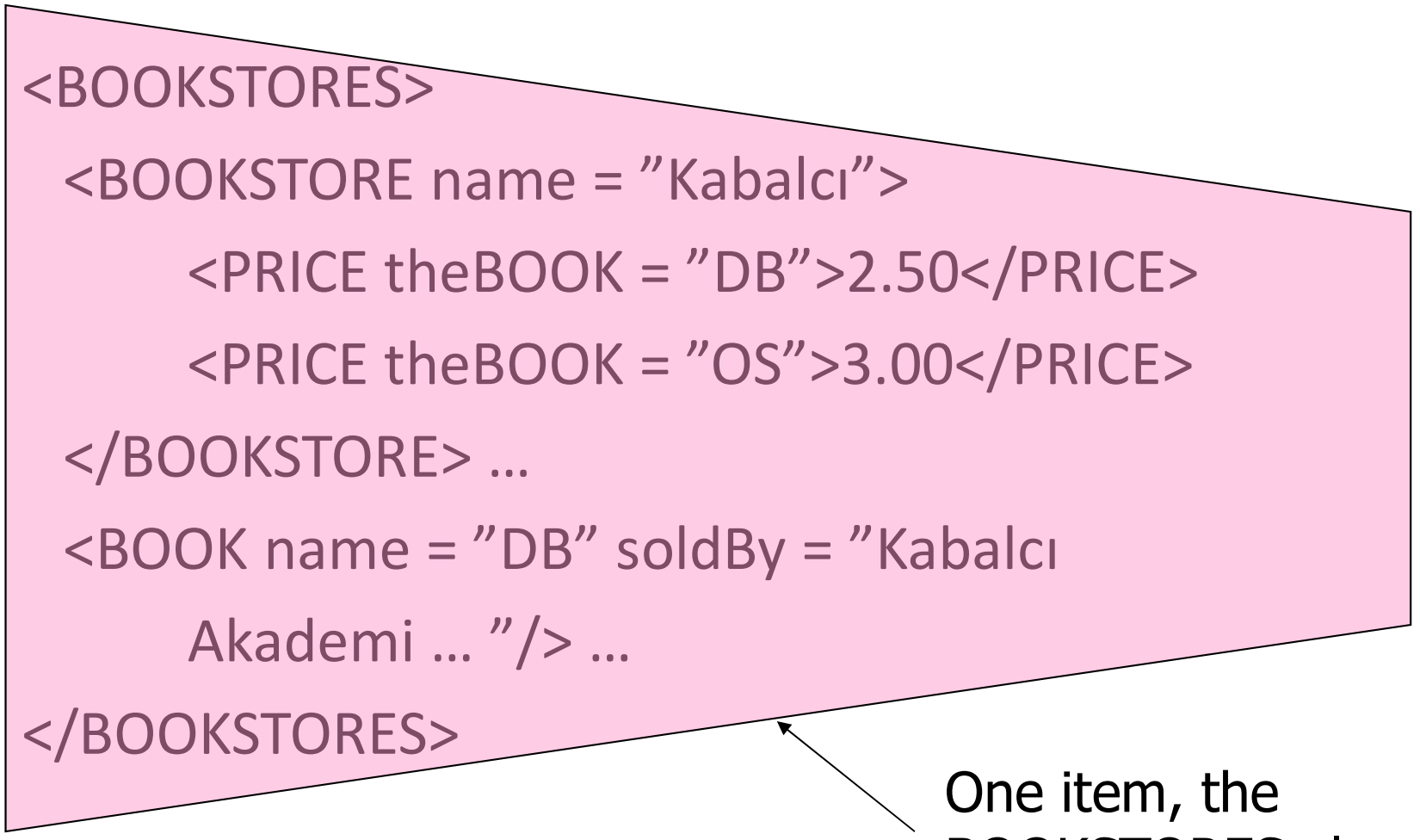
Document node is all of this, **plus the header (<? xml version…).**

Paths in XML Documents

- XPath is a language **for describing paths** in XML documents.
- The result of the described path is **a sequence of items.**
- sequences of slashes (/) and tags, starting with /.
  - Example: /BOOKSTORES/BOOKSTORE/PRICE
  - starting with just the doc node, first tag is the root, and processing each tag from the left to right end.
  - For each item that is an element node, replace the element by the ALL subelements with tag *X*.

Mesela örnekte; BOOKSTORES kökü altındaki bütün BOOKSTORE'ların altındaki bütün PRICE element'leri

Example: /BOOKSTORES



Example: /BOOKSTORES/BOOKSTORE

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK =”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

<BOOK name = ”DB” soldBy = ”Kabalcı Akademi …”/> …

</BOOKSTORES>

This BOOKSTORE element followed b
all the other BOOKSTORE elements

Example: /BOOKSTORES/BOOKSTORE/PRICE

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK =”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

<BOOK name = ”DB” soldBy = ”Kabalcı

Akademi …”/> …

</BOOKSTORES>

These PRICE elements followed by the PRICE elements of all the other BOOKSTOREs.

Paths that Begin Anywhere

- The path starts from the document node by default and if it begins with //*X*, then the first step can begin at the tag with '*X*'.

Example: //PRICE

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK =”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

<BOOK name = ”DB” soldBy = ”Kabalcı

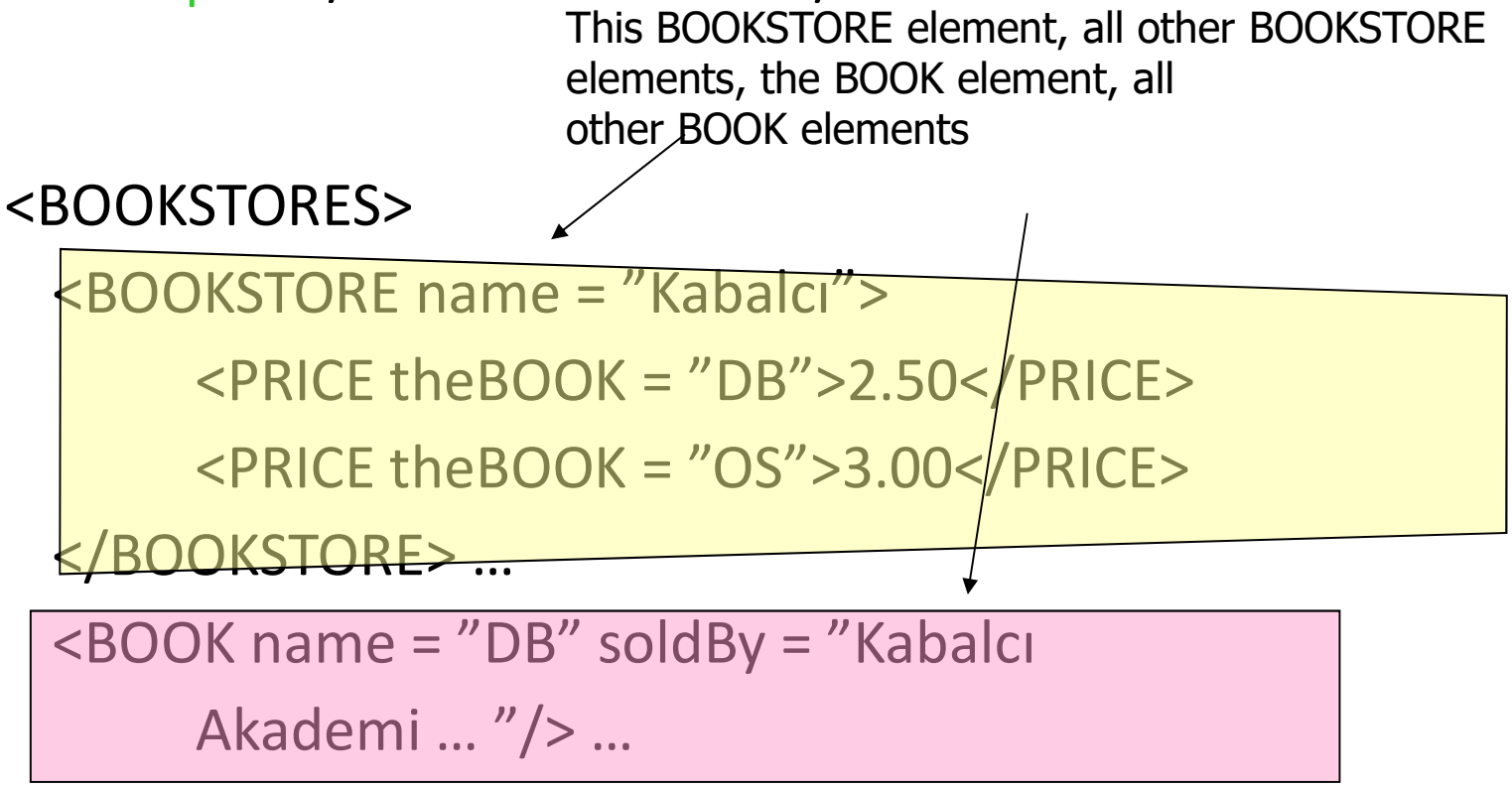
Akademi …”/> …

</BOOKSTORES>

These PRICE elements and any other all PRICE elements in the entire document

Wild-Card *

- A star (*) in place of a tag represents any one tag.
- Example: /*/*/PRICE represents all price objects at the third level of nesting.
- Example: /BOOKSTORES/*



Remember: Item Sequences

- Until now, all item sequences have been
  - sequences of elements.
- Next: sequence of values of primitive type, such as strings in the next example. *(When a path expression ends in an attribute)*
- Instead of going to subelements with a given tag, you can go to an attribute of the elements you already have.
  - An attribute is indicated by putting @ in front of its name.

Example: /BOOKSTORES/BOOKSTORE/PRICE/@theBOOK

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK = ”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

<BOOK name = ”DB” soldBy = ”Kabalcı Akademi …”/> …

</BOOKSTORES>

These attributes contribute ”DB” ”OS” to the result, followed by other theBOOK Values if any.

Selection Conditions

- A condition inside [...] may follow a tag.
- If so, then only paths that have that tag and also satisfy the condition are included in the result of a path expression.
  - Query: /BOOKSTORES/BOOKSTORE/PRICE[. < 2.75]

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK = ”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

The condition that the PRICE be < $2.75 makes this price but not the OS price part of the result.

Example: Attribute in Selection

Example: /BOOKSTORES/BOOKSTORE/PRICE[@theBOOK = ”DB”]

<BOOKSTORES>

<BOOKSTORE name = ”Kabalcı”>

<PRICE theBOOK = ”DB”>2.50</PRICE>

<PRICE theBOOK = ”OS”>3.00</PRICE>

</BOOKSTORE> …

Now, this PRICE element is selected, along with any other prices for DB.

Axes

- In general, path expressions allow us to start at the root and execute steps to find a sequence of nodes at each step.
- At each step, we may follow any one of several *axes*.
- The default axis is child:: --- go to all the children of the current set of nodes.
  - /BOOKSTORES/BOOK is really shorthand for
    /BOOKSTORES/ child::BOOK .
- @ is really shorthand for the attribute:: axis.
  - Thus, /BOOKSTORES/BOOK[@name = ”DB”] is shorthand for
    /BOOKSTORES/BOOK[attribute::name = ”DB”]

More Axes

- Some other useful axes are:
  1. parent:: = parent(s) of the current node(s).
  2. descendant-or-self:: = the current node(s) and all descendants.
     - Note: // is really shorthand for this axis.
  3. ancestor::, ancestor-or-self, etc.
  4. self (the dot) .

XQuery

- XQuery extends XPath to a query language **that has power similar to SQL.**
- Uses the same sequence-of-items data model.
- XQuery is an expression language.
  - Like relational algebra --- any XQuery expression can be an argument of any other XQuery expression.

More About Item Sequences

- XQuery will sometimes form sequences of sequences.
- All sequences that is coming from different iterations are flattened.
- Example: (1 2 () (3 4) <st>data</data>) =

  (1 2 3 4 <st>data</data>).

Empty sequence

FLWR Expressions

1. One or more for and/or let clauses.
2. Then an optional where clause.
3. A return clause. (must)
   - Return clause (not statement!). Because it may be executed many times in for-loops.

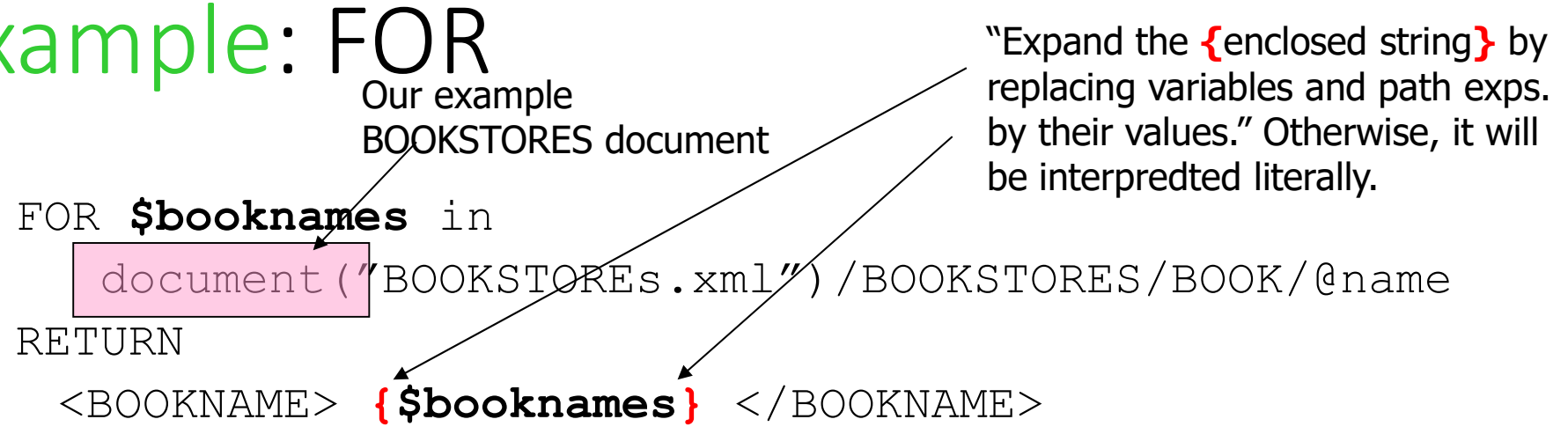
- LET produces only a local definition.
- Each FOR creates a loop.
- **At each iteration of the nested loops**, if any, evaluate the where clause.
- If the where clause returns TRUE, invoke the return clause, and append its value to the output. Thus, **result is constructed in stages..**

FOR Clauses

`FOR <variable> in <expression>, . .`

- Variables begin with $.
- A FOR-variable **takes on each item in the sequence** denoted by the expression, in turn.
- **Whatever follows this FOR is executed once for each value of the variable.**

Example: FOR



- **$booknames** ranges over the **name attributes of all BOOKs** in our example document.
- Result is a sequence of BOOKNAME elements:

<BOOKNAME>DB</BOOKNAME>

<BOOKNAME>OS</BOOKNAME> . . .

Same as above:

```
FOR $B in
  document("BOOKSTOREs.xml")/BOOKSTORES/BOOK
RETURN
  <BOOKNAME> {$B/@name} </BOOKNAME>
```

- **$B** ranges over the **all BOOKs** in our example document.

LET Clauses

```
LET <variable> := <expression>, . . .
```

- Value of the variable becomes the *sequence* of items defined by the expression.
- **Note LET does not cause iteration; FOR does.**

```
LET $d := document("BOOKSTOREs.xml")
LET $booknames := $d/BOOKSTORES/BOOK/@name
RETURN
  <BOOKNAMES> {$booknames } </BOOKNAMES>
```

- **RETURN executes here only 1 time.** It returns one element with all the names of the BOOKs, like:

  <BOOKNAMES>DB OS …</BOOKNAMES>

Order-By Clauses

- FLWR is really FLWOR: an order-by clause can precede the return.
- Form: order by <expression>
  - With optional ascending or descending.
- The expression is evaluated for each assignment to variables.
- Determines placement in output sequence.

Example: Order-By

- List all prices for DB, lowest first.

```
LET $d := document("BOOKSTOREs.xml")
FOR $p in
  $d/BOOKSTORES/BOOKSTORE/PRICE[@theBOOK="DB"]
ORDER BY $p
RETURN $p
```

Order those bindings by the values inside the elements (auto-matic coersion).

Generates **bindings** for $p to PRICE elements.

Aside: SQL ORDER BY

- SQL works the same way; it's the result of the FROM and WHERE that get ordered, not the output.
- Example: Using R(a,b),

```
SELECT b
FROM R
WHERE b > 10
ORDER BY a;
```

Then, the b-values are extracted from these tuples and printed in the same order.

R tuples with b>10 are ordered by their a-values.

Predicates

- Normally, conditions imply existential quantification.
- Example: /BOOKSTORES/BOOKSTORE[@name] means “all the BOOKSTOREs that have a name.”
- Example: /BOOKSTORES/BOOK[@soldAt = ”Kabalcı”] gives the set of BOOKs that are sold at Kabalcı BOOKSTORE.

Example: Comparisons

- Query: Let us produce the PRICE elements (from all BOOKSTOREs) for all **the BOOKs that are sold by «Kabalcı» BOOKSTORE**. *(**Not** the BOOKS in Kabalcı store's book price list)*
- The output must be <bookAllPrices> elements with the **names of the BOOKSTORE and BOOK as attributes** and the price element as a subelement.

```
<BOOKSTORES>
   <BOOKSTORE name = "Kabalcı">
         <PRICE theBOOK = "DB">2.50</PRICE>
         <PRICE theBOOK = "OS">3.00</PRICE>
   </BOOKSTORE> …
   <BOOK name = "DB" soldAt = "Kabalcı Akademi … "/>
</BOOKSTORES>
```

Strategy

1. Must use a triple for-loop, with variables ranging over all BOOK elements, all BOOKSTORE elements, and all PRICE elements within those BOOKSTORE elements.
2. Check that the BOOK is sold at Kabalcı BOOKSTORE and that the name of the BOOK and theBOOK in the PRICE element match.
3. Construct the output element.

The Query

```
LET $BOOKSTOREs =
  doc("BOOKSTOREs.xml")/BOOKSTORES
FOR $BOOK in $BOOKSTOREs/BOOK
FOR $BOOKSTORE in $BOOKSTOREs/BOOKSTORE
FOR $price in $BOOKSTORE/PRICE
WHERE $BOOK/@soldAt = "Kabalcı" and
  $price/@theBOOK = $BOOK/@name
RETURN < bookAllPrices BOOKSTORE =
  {$BOOKSTORE/@name} BOOK =
  {$BOOK/@name}>{$price}</ bookAllPrices >
```

True if "Kabalcı" appears anywhere in the sequence

```
SAMPLE RESULT :
< bookAllPrices BOOKSTORE ='Kabalcı' BOOK='DB'>
     <PRICE theBOOK = "DB">2.50</PRICE>
</bookAllPrices>
```

Strict Comparisons

- To require that the things being compared are **sequences of only one element**, use the Fortran comparison operators:
  - **eq, ne, lt, le, gt, ge.**
- Example: $BOOK/@soldAt **eq** ”Kabalcı” is true only if Kabalcı is the only BOOKSTORE selling the BOOK.

Comparison of Elements and Values

- When an element is compared to a primitive value, the element is treated as its value (***coersion***), if that value is atomic.
- Example:
  `/BOOKSTORES/BOOKSTORE[@name="Kabalcı"]/PRICE[@theBOOK="DB"]` **`eq`** `"2.50"`

  is true if `Kabalcı` charges $2.50 for DB.

Comparison of Two «Elements»

- It is insufficient that two elements look alike.
- Example:

```
/BOOKSTORES/BOOKSTORE[@name="Kabalcı"]/PRICE[@theBOOK="DB"]
```

```
eq /BOOKSTORES/BOOKSTORE[@name="Akademi"]/PRICE[@theBOOK="DB"]
```

is false, even if `Kabalcı` and `Akademi` charge the same for DB.

- For elements to be equal, they must be the same, physically, in the implied document.
- Subtlety: elements are really pointers to sections of particular documents, not the text strings appearing in the section.

Getting Data From Elements

- Suppose we want to compare **the values of elements**, rather than their location in documents.
- To extract just the value (e.g., the price itself) from an element *E*, use **data(*E*)**.

Example: data()

```
<BOOKSTORES>
    <BOOKSTORE name = ”Kabalcı”>
        <PRICE theBOOK = ”DB”>2.50</PRICE>
        <PRICE theBOOK = ”OS”>3.00</PRICE>
    </BOOKSTORE> …
    <BOOK name = ”DB” soldAt = ”Kabalcı Akademi … ”/>
</BOOKSTORES>
```

- Suppose we want to modify the return for previous Query:“find the prices of BOOKs at BOOKSTOREs that sell a BOOK Kabalcı sells” to **produce an empty BBP element with price as one of its attributes**.

```
..
FOR $price in $BOOKSTORE/PRICE
WHERE ..
RETURN <BBP bookstore = {$BOOKSTORE/@name}
    book = {$BOOK/@name} price= {data($price)} />
```

SAMPLE RESULT :

```
< BBP bookstore='Kabalcı'
      book='DB'
      price=2.50
bookAllPrices/>
```

Used to extract just the value from an element

Used to avoid showing as literally in HTML tags. {} make it as expression

Eliminating Duplicates

- Use function `distinct-values` applied to a sequence.
- Subtlety: this function strips tags away from elements and compares the string values.
  - But it doesn't restore the tags in the result!.
  - Below shows all distinct prices.

```
RETURN distinct-values(
  LET $BOOKSTOREs = doc("BOOKSTOREs.xml")
  RETURN
     $BOOKSTOREs/BOOKSTORES/BOOKSTORE/PRICE
)
```

Remember: A query can appear any place a value can since XQuery is an expression language.

XSLT

- XSLT (*extensible stylesheet language – transforms*) is another language to process XML documents.
- Originally intended as a presentation language: transform XML into an HTML page that could be displayed.
- **It can also transform XML -> XML, thus serving as a query language.**
- Like XML Schema, an XSLT program is itself an XML document.
- XSLT has a special namespace of tags, usually indicated by xsl:.