

# XML

## Document Type Definitions

# XML

- XML stands for eXtensible Markup Language.
- XML was designed to describe data.
- XML has come into common use for the interchange of data over the Internet.

# Well-Formed and Valid XML

- *Well-Formed XML* allows you to invent your own tags.
- *Valid XML* conforms to a certain DTD (Document Type Definition).

# Well-Formed XML

- Start the document with a *declaration*, surrounded by `<?xml ... ?>` .

- Normal declaration is:

```
<?xml version = "1.0" standalone = "yes" ?>
```

- “standalone” = “no DTD provided.”

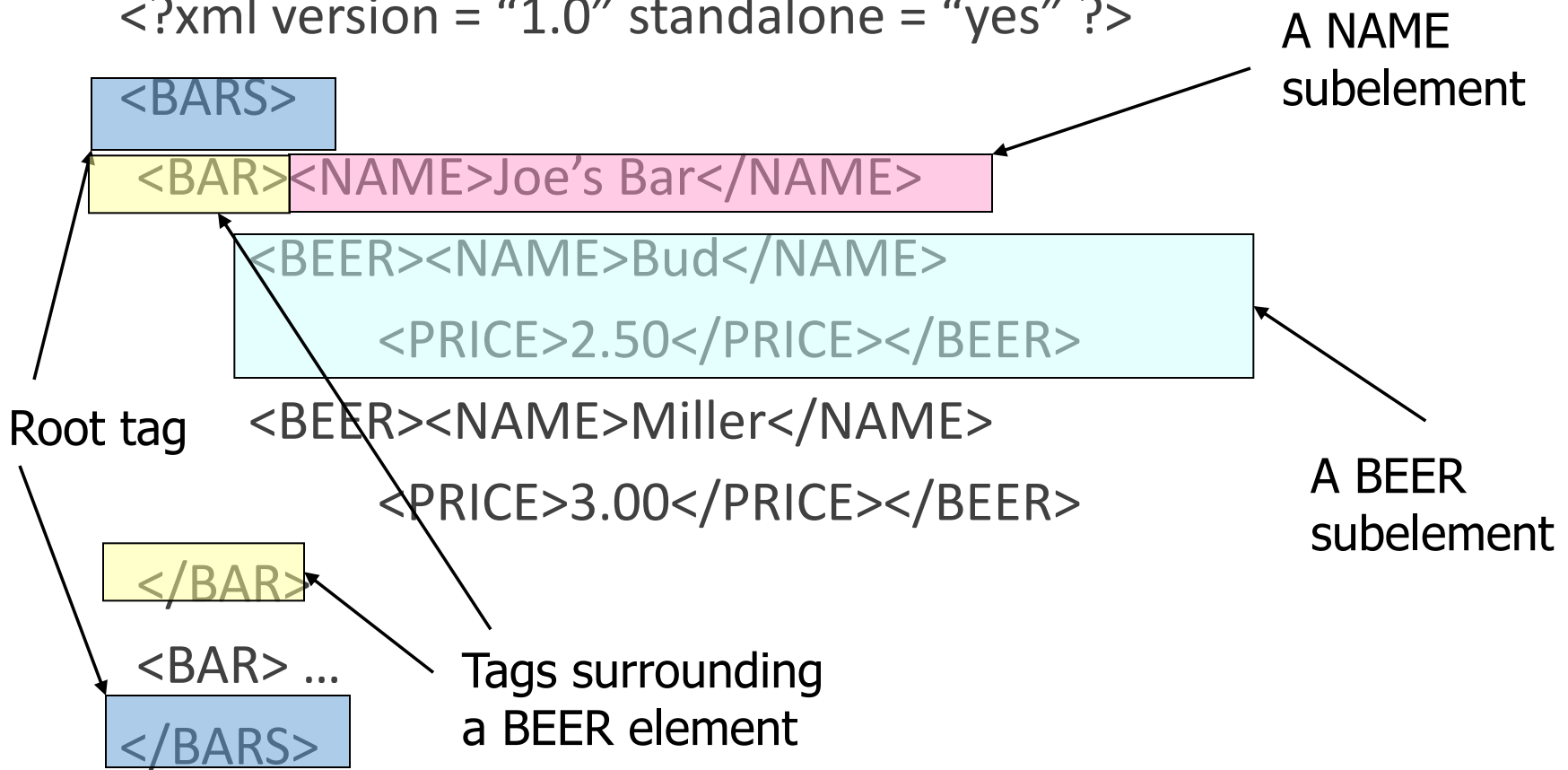
- Balance of document is a *root tag* surrounding nested tags.

# Tags

- Tags are normally matched pairs, as `<FOO> ... </FOO>`.
- Unmatched tags also allowed, as `<FOO/>`
- XML tags are case-sensitive.

# Example: Well-Formed XML

<?xml version = "1.0" standalone = "yes" ?>



# DTD Structure

```
<!DOCTYPE <root tag> [  
  <!ELEMENT <name> (<components>) >  
  ... more elements ...  
>
```

# DTD Elements

- The description of an element consists of its name (tag), and a description of any nested tags.
  - Includes order of subtags and their multiplicity.
- Leaves (text elements) have #PCDATA (*Parsed Character DATA*) in place of nested tags.



## Example: DTD

```
<!DOCTYPE BARS [
```

```
<!ELEMENT BARS (BAR*)>
```

```
<!ELEMENT BAR (NAME, BEER+)>
```

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

```
<!ELEMENT BEER (NAME, PRICE)>
```

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

```
]>
```

A BARS object has zero or more BAR's nested within.

A BAR has one NAME and one or more BEER subobjects.

NAME and PRICE are text.

A BEER has a NAME and a PRICE.

# Element Descriptions

- 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.
- Symbol | can connect alternative sequences of tags.

## Example: Element Description

- A name is an optional title (e.g., “Prof.”), a first name, and a last name, in that order, or it is an IP address:

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

# Use of DTD's

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)

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE BARS [  
  <!ELEMENT BARS (BAR*)>  
  <!ELEMENT BAR (NAME, BEER+)>  
  <!ELEMENT NAME (#PCDATA)>  
  <!ELEMENT BEER (NAME, PRICE)>  
  <!ELEMENT PRICE (#PCDATA)>  
>
```

The DTD



The document



```
<BARS>  
  <BAR><NAME>Joe's Bar</NAME>  
    <BEER><NAME>Bud</NAME> <PRICE>2.50</PRICE></BEER>  
    <BEER><NAME>Miller</NAME> <PRICE>3.00</PRICE></BEER>  
  </BAR>  
  <BAR> ...  
</BARS>
```

## Example: (b)

- Assume the BARS DTD is in file bar.dtd.

```
<?xml version = "1.0" standalone = "no" ?>
```

```
<!DOCTYPE BARS SYSTEM "bar.dtd">
```

```
<BARS>
```

```
  <BAR><NAME>Joe's Bar</NAME>
```

```
    <BEER><NAME>Bud</NAME>
```

```
      <PRICE>2.50</PRICE></BEER>
```

```
    <BEER><NAME>Miller</NAME>
```

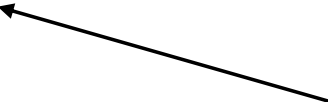
```
      <PRICE>3.00</PRICE></BEER>
```

```
  </BAR>
```

```
  <BAR> ...
```

```
</BARS>
```

Get the DTD  
from the file  
bar.dtd



# Attributes

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

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

declares attributes for element *E*, along with its datatype.

# Example: Attributes

- Bars can have an attribute `kind`, a character string describing the bar.

```
<!ELEMENT BAR (NAME BEER*) >
```

```
<!ATTLIST BAR kind
```

```
#IMPLIED>
```

```
CDATA
```

Character string  
type; no tags

Attribute is optional  
opposite: `#REQUIRED`



## Example: Attribute Use

- In a document that allows BAR tags, we might see:

```
<BAR kind = "sushi">
```

```
  <NAME>Homma's</NAME>
```

```
  <BEER><NAME>Sapporo</NAME>
```

```
    <PRICE>5.00</PRICE></BEER>
```

```
  . . .
```

```
</BAR>
```

# ID's and IDREF's

- Attributes can be pointers from one object to another.
- Allows the structure of an XML document to be a general graph, rather than just a tree.

# Creating ID's

- Give an element  $E$  an attribute  $A$  of type ID.
- When using tag  $\langle E \rangle$  in an XML document, give its attribute  $A$  a unique value.
- Example:

$\langle E \quad A = \text{"xyz"} \rangle$

# Creating IDREF's

- To allow elements of type  $F$  to refer to another element with an ID attribute, give  $F$  an attribute of type IDREF.
- Or, let the attribute have type IDREFS, so the  $F$ -element can refer to any number of other elements.

## Example: ID's and IDREF's

- 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.

# The DTD

```
<!DOCTYPE BARS [  
  <!ELEMENT BARS (BAR*, BEER*)>  
  <!ELEMENT BAR (SELLS+)>  
    <!ATTLIST BAR name ID #REQUIRED>  
  <!ELEMENT SELLS (#PCDATA)>  
    <!ATTLIST SELLS theBeer IDREF #REQUIRED>  
  <!ELEMENT BEER EMPTY>  
    <!ATTLIST BEER name ID #REQUIRED>  
    <!ATTLIST BEER soldBy IDREFS #IMPLIED>  
>
```

Bar elements have name as an ID attribute and have one or more SELLS subelements.

SELLS elements have a number (the price) and one reference to a beer.

Beer elements have an ID attribute called name, and a soldBy attribute that is a set of Bar names.

Explained next

## Example: A Document

<BARS>

<BAR name = "JoesBar">

<SELLS theBeer = "Bud">2.50</SELLS>

<SELLS theBeer = "Miller">3.00</SELLS>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar

SuesBar ..." /> ...

</BARS>

# Query Languages for XML

XPath

XQuery



# 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* (defined next).

# Principal Kinds of Nodes

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.

# DTD for Running Example

```
<!DOCTYPE BARS [  
  <!ELEMENT BARS (BAR*, BEER*)>  
  <!ELEMENT BAR (PRICE+)>  
    <!ATTLIST BAR name ID #REQUIRED>  
  <!ELEMENT PRICE (#PCDATA)>  
    <!ATTLIST PRICE theBeer IDREF #REQUIRED>  
  <!ELEMENT BEER EMPTY>  
    <!ATTLIST BEER name ID #REQUIRED>  
    <!ATTLIST BEER soldBy IDREFS #IMPLIED>  
>
```

## Example Document

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

```
<BARS>
```

```
<BAR name = "JoesBar">  
  <PRICE theBeer = "Bud">2.50</PRICE>  
  <PRICE theBeer = "Miller">3.00</PRICE>  
</BAR> ...
```

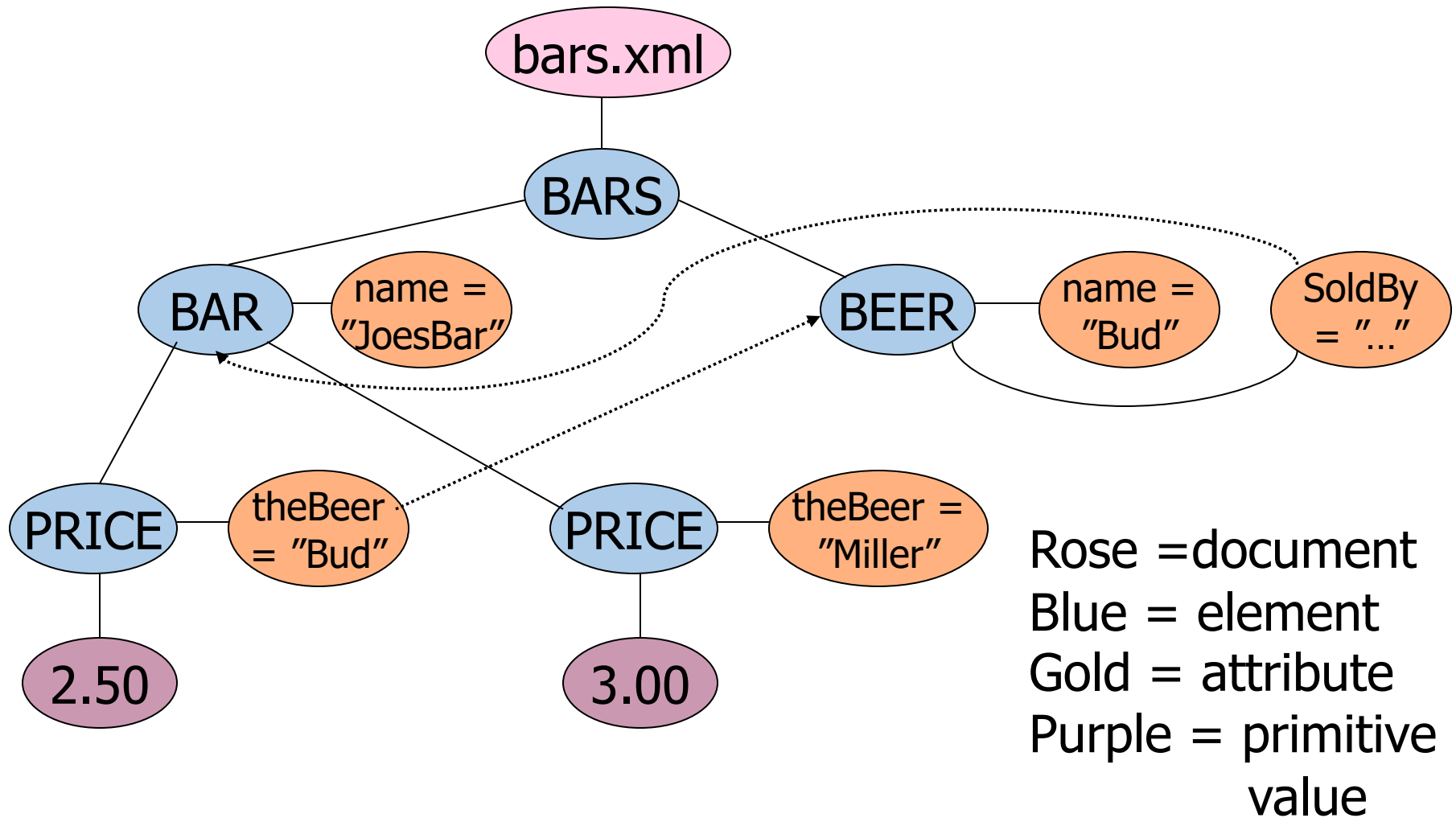
```
<BEER name = "Bud" soldBy = "JoesBar  
SuesBar ... "/> ...
```

```
</BARS>
```

An element node

An attribute node

# Nodes as Semistructured Data



# 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.

# Path Expressions

- Simple path expressions are sequences of slashes (/) and tags, starting with /.
  - **Example:** /BARS/BAR/PRICE
- Construct the result by starting with just the doc node and processing each tag from the left.

## Example: /BARS

```
<BARS>  
  <BAR name = "JoesBar">  
    <PRICE theBeer = "Bud">2.50</PRICE>  
    <PRICE theBeer = "Miller">3.00</PRICE>  
  </BAR> ...  
  <BEER name = "Bud" soldBy = "JoesBar  
    SuesBar ... "/> ...  
</BARS>
```

One item, the  
**BARS** element



## Example: /BARS/BAR

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar  
SuesBar ..."/> ...

</BARS>

This BAR element followed by  
all the other BAR elements

## Example: /BARS/BAR/PRICE

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar  
SuesBar ..."/> ...

</BARS>

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

# Attributes in Paths

- 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: /BARS/BAR/PRICE/@theBeer

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar

SuesBar ..."/> ...

</BARS>

These attributes contribute "Bud" "Miller" to the result, followed by other theBeer values.

# Paths that Begin Anywhere

- If the path starts from the document node and begins with `//X`, then the first step can begin at the root or any subelement of the root, as long as the tag is `X`.

## Example: //PRICE

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

<BEER name = "Bud" soldBy = "JoesBar  
SuesBar ..."/> ...

</BARS>

These PRICE elements and  
any other 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: /BARS/\*

This BAR element, all other BAR elements, the BEER element, all other BEER elements

<BARS>

```
<BAR name = "JoesBar">  
  <PRICE theBeer = "Bud">2.50</PRICE>  
  <PRICE theBeer = "Miller">3.00</PRICE>  
</BAR> ...
```

```
<BEER name = "Bud" soldBy = "JoesBar  
  SuesBar ... "/> ...
```

</BARS>



# 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.

## Example: Selection Condition

- /BARS/BAR/PRICE[. < 2.75]

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

The current element.

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

## Example: Attribute in Selection

- /BARS/BAR/PRICE[@theBeer = "Miller"]

<BARS>

<BAR name = "JoesBar">

<PRICE theBeer = "Bud">2.50</PRICE>

<PRICE theBeer = "Miller">3.00</PRICE>

</BAR> ...

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

# 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.
  - Similarly to SQL

# More About Item Sequences

- XQuery will sometimes form sequences of sequences.
- All sequences are flattened.
- **Example:** (1 2 () (3 4)) = (1 2 3 4).

↑  
Empty  
sequence

# FLWR Expressions

1. One or more **for** and/or **let** clauses.
2. Then an optional **where** clause.
3. A **return** clause.

# Semantics of FLWR Expressions

- Each **for** creates a loop.
  - **let** produces only a local definition.
- 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.

## Example: FOR

Our example  
BARS document

“Expand the enclosed string by replacing variables and path exps. by their values.”

for \$beer in document("bars.xml")/BARS/BEER/@name  
return

<BEERNAME>{\$beer}</BEERNAME>

- \$beer ranges over the name attributes of all beers in our example document.
- Result is a sequence of BEERNAME elements:  
<BEERNAME>Bud</BEERNAME>  
<BEERNAME>Miller</BEERNAME> . . .



# 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.

## Example: LET

```
let $d := document("bars.xml")
```

```
let $beers := $d/BARS/BEER/@name
```

```
return
```

```
<BEERNAMES> {$beers} </BEERNAMES>
```

- Returns one element with all the names of the beers, like:

```
<BEERNAMES>Bud Miller ...</BEERNAMES>
```

# 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 Bud, lowest first.

```
let $d := document("bars.xml")
```

```
for $p in $d/BARS/BAR/PRICE[@theBeer="Bud"]
```

```
order by $p
```

```
return $p
```

Order those bindings by the values inside the elements .

Generates bindings for \$p to PRICE elements.

Each binding is evaluated for the output. The result is a sequence of PRICE elements.

# Aside: SQL ORDER BY

- SQL works the same way; it's the result of the FROM and WHERE that get ordered.

- **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:** /BARS/BAR[@name] means “all the bars that have a name.”
- **Example:** /BARS/BEER[@soldAt = "JoesBar"] gives the set of beers that are sold at Joe's Bar.

# Actions

- Read Chapters 11 about XML DTD and 12 about Xpath and XQuery
- Review slides!