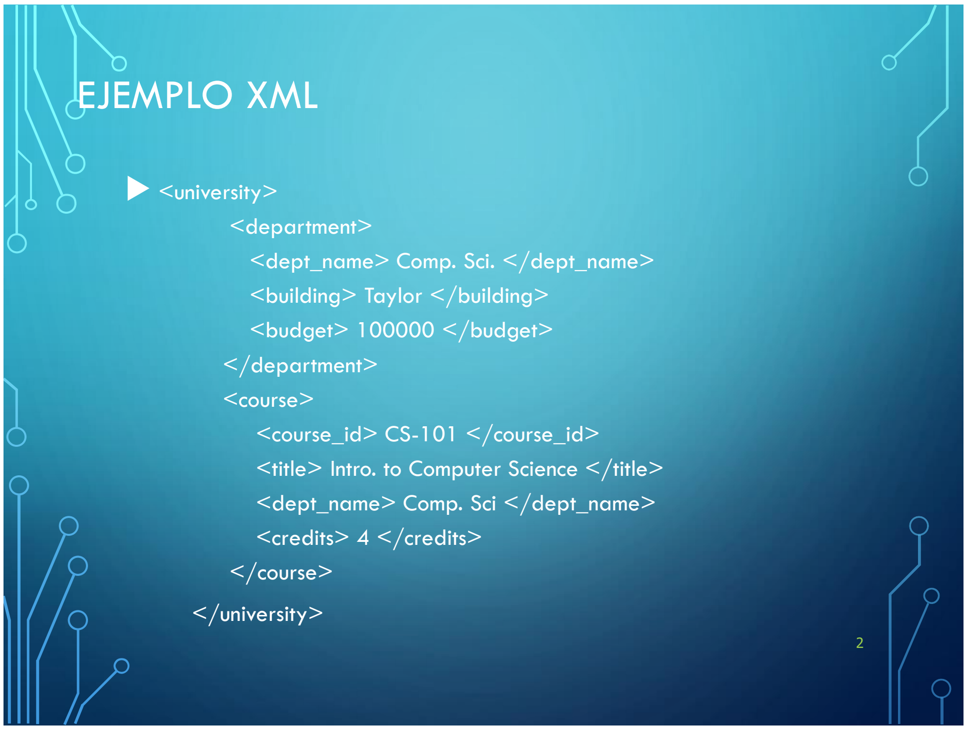
XML

Base de Datos

1

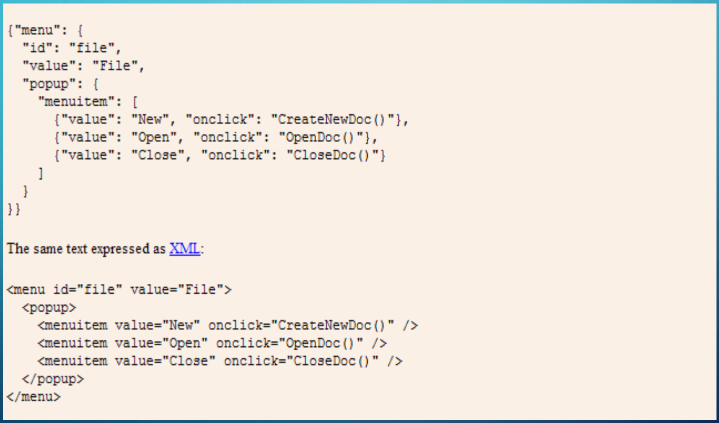


EJEMPLO XML

```
▶ <university>
        <department>
          <dept_name> Comp. Sci. </dept_name>
          <building> Taylor </building>
          <budget> 100000 </budget>
       </department>
       <course>
          <course_id> CS-101 </course_id>
          <title> Intro. to Computer Science </title>
          <dept_name> Comp. Sci </dept_name>
          <credits> 4 </credits>
        </course>
    </university>
```

2

JSON VS. XML



```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

The same text expressed as XML:

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

3

3

DATOS ESTRUCTURADOS Y SEMIESTRUCTURADOS

- **Datos estructurados**
  - Tienen un formato estricto
  - Ejemplo : tablas
- **Datos semiestructurados**
  - Tienen una cierta estructura
  - No todos los “registros” tienen la misma estructura

4

DATOS ESTRUCTURADOS Y SEMIESTRUCTURADOS (CONT.)

- La información sobre la estructura esta mezclada con los datos
- Son “autodescriptivos”
- Se pueden representar como grafos

5

DATOS ESTRUCTURADOS Y SEMIESTRUCTURADOS (CONT.)

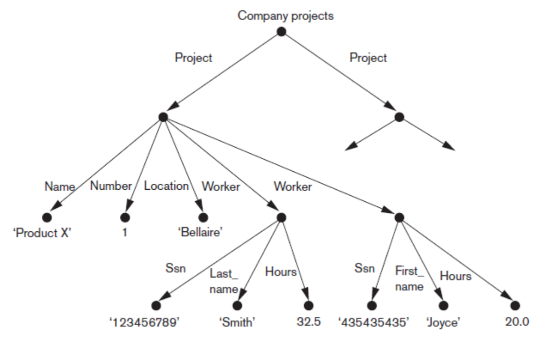
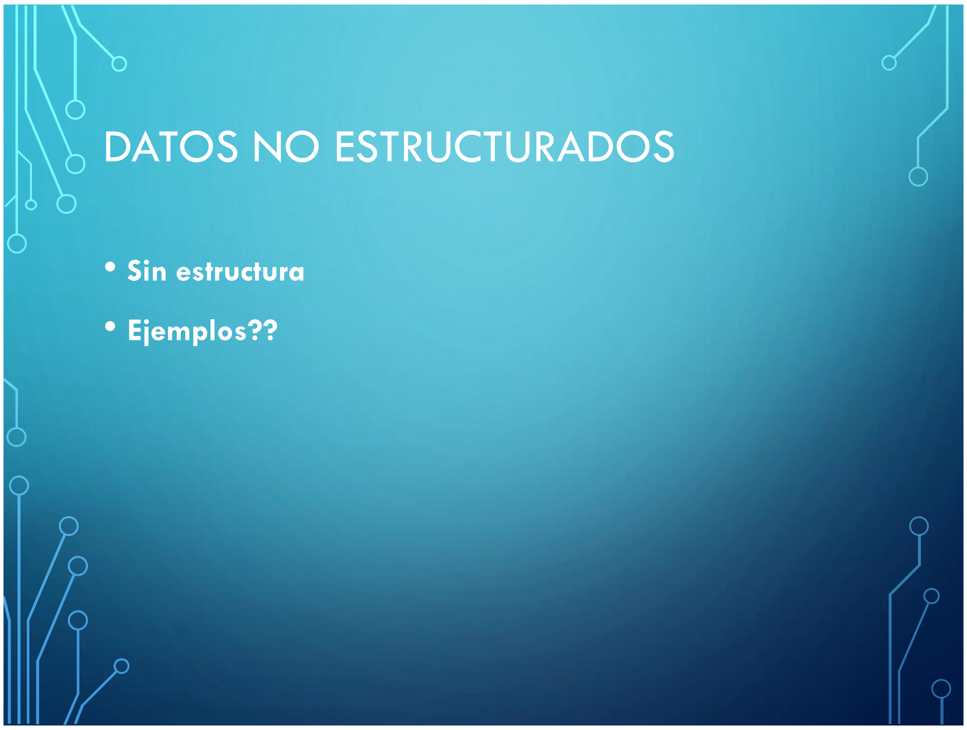


Figure 12.1
Representing semistructured data as a graph.

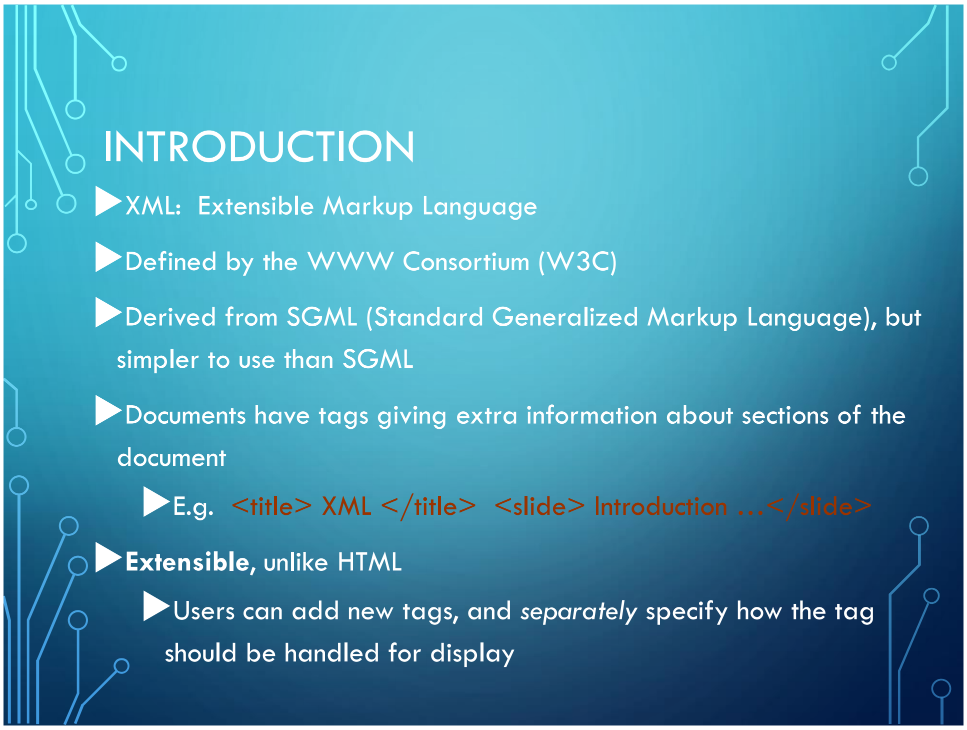
6



DATOS NO ESTRUCTURADOS

- **Sin estructura**
- **Ejemplos??**

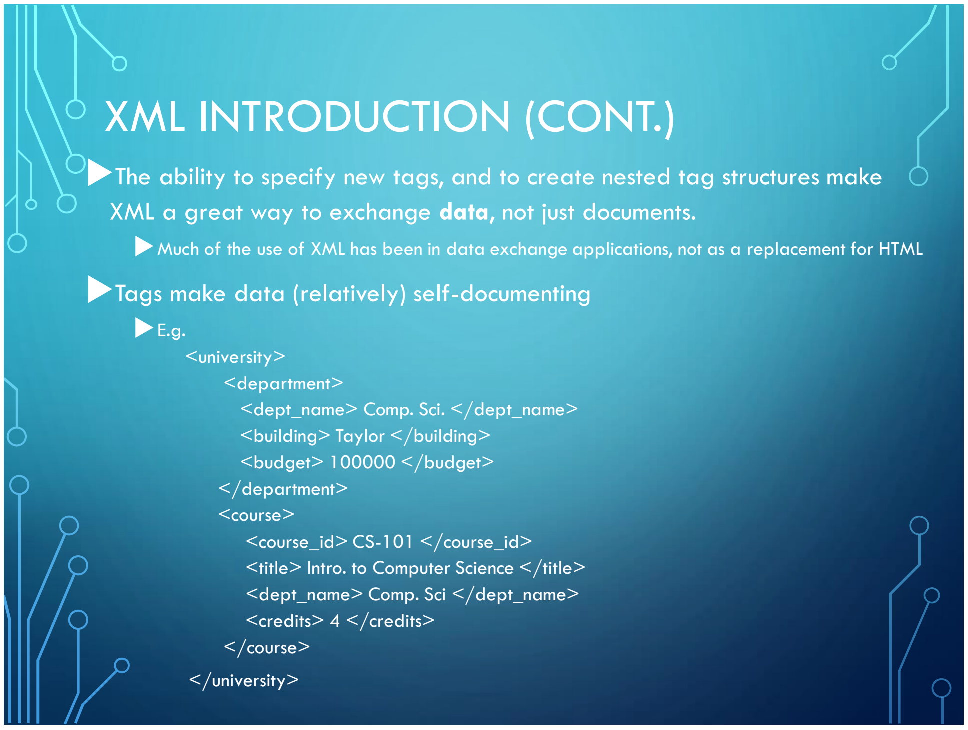
7



INTRODUCTION

- XML: Extensible Markup Language
- Defined by the WWW Consortium (W3C)
- Derived from SGML (Standard Generalized Markup Language), but simpler to use than SGML
- Documents have tags giving extra information about sections of the document
  - E.g. <title> XML </title> <slide> Introduction …</slide>
- **Extensible**, unlike HTML
  - Users can add new tags, and *separately* specify how the tag should be handled for display

8

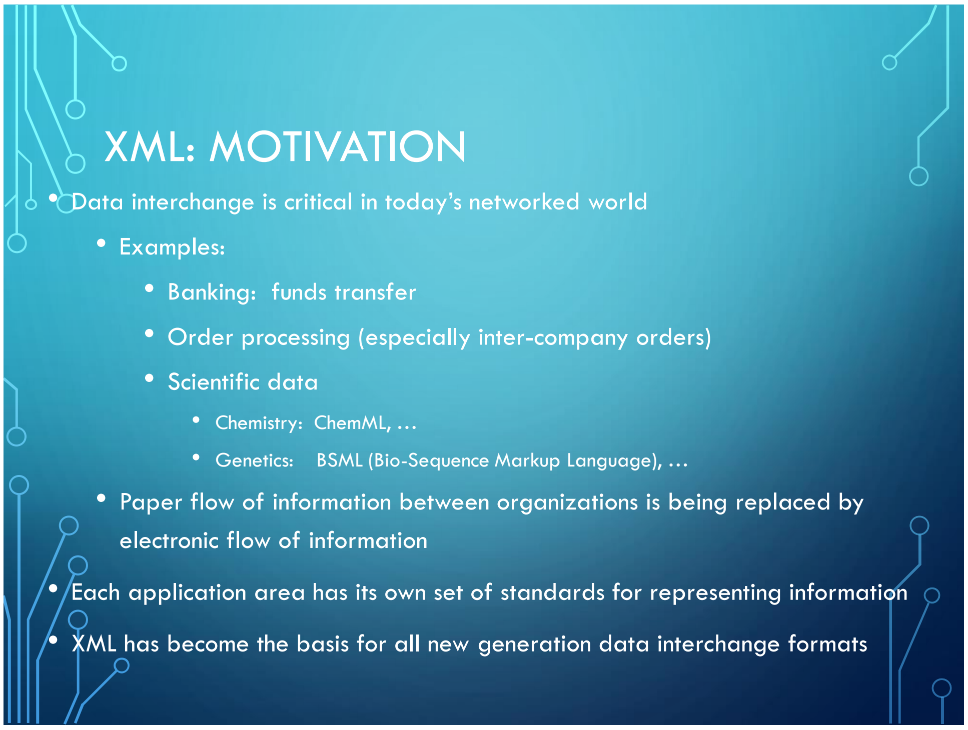


XML INTRODUCTION (CONT.)

- The ability to specify new tags, and to create nested tag structures make XML a great way to exchange **data**, not just documents.
  - Much of the use of XML has been in data exchange applications, not as a replacement for HTML
- Tags make data (relatively) self-documenting
  - E.g.

```
<university>
      <department>
         <dept_name> Comp. Sci. </dept_name>
         <building> Taylor </building>
         <budget> 100000 </budget>
     </department>
     <course>
         <course_id> CS-101 </course_id>
         <title> Intro. to Computer Science </title>
         <dept_name> Comp. Sci </dept_name>
         <credits> 4 </credits>
      </course>
</university>
```

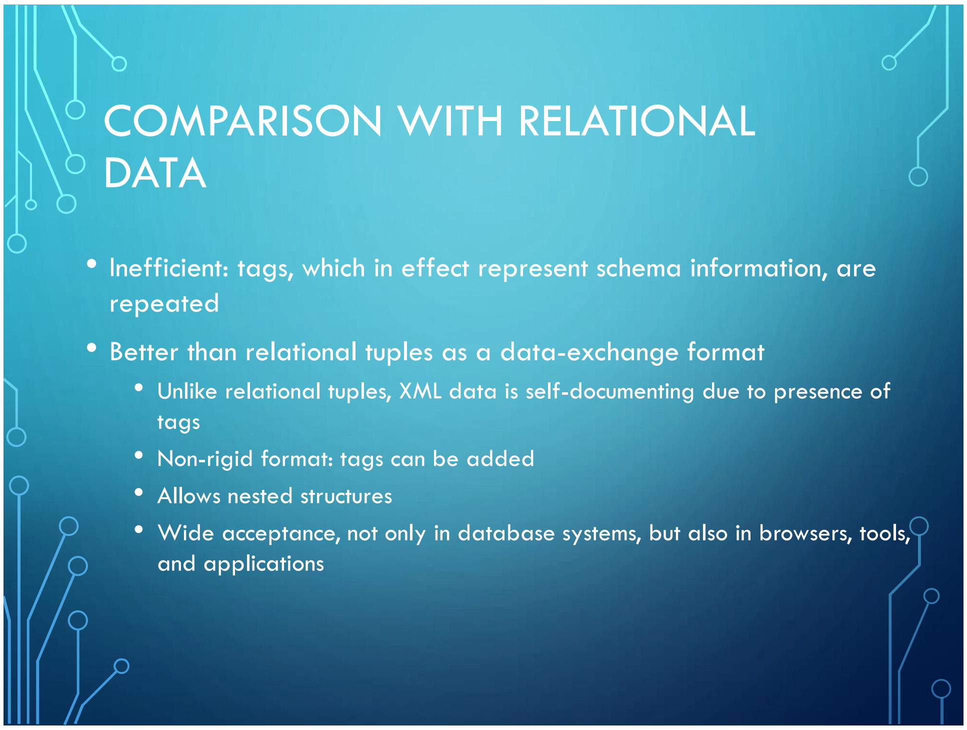
9



XML: MOTIVATION

- Data interchange is critical in today's networked world
  - Examples:
    - Banking: funds transfer
    - Order processing (especially inter-company orders)
    - Scientific data
      - Chemistry: ChemML, …
      - Genetics: BSML (Bio-Sequence Markup Language), …
  - Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats

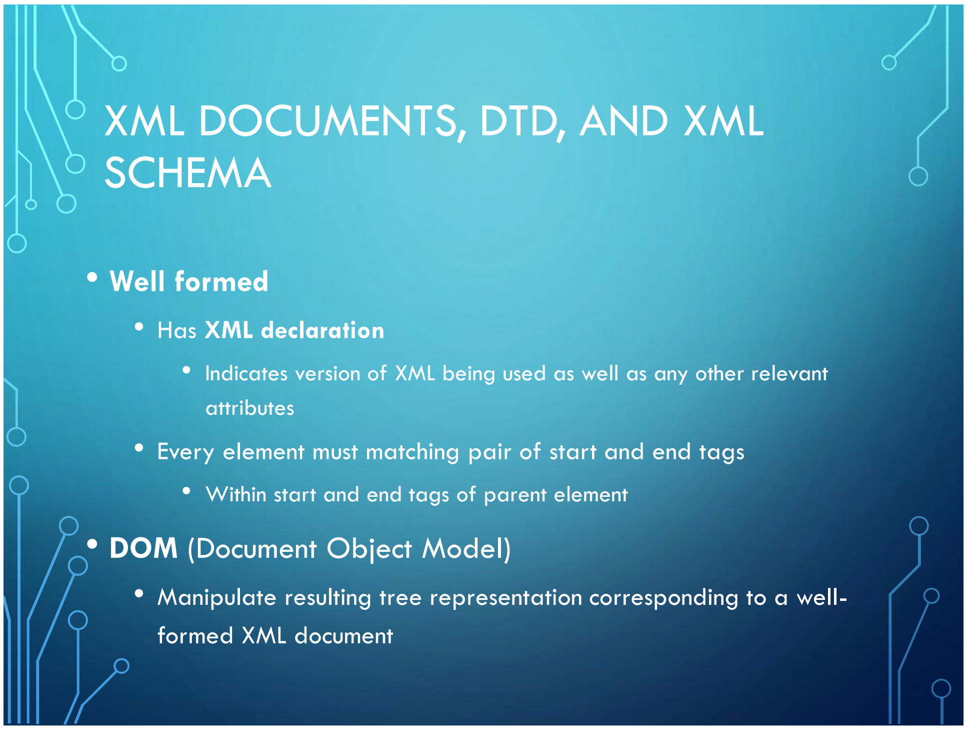
10



COMPARISON WITH RELATIONAL DATA

- Inefficient: tags, which in effect represent schema information, are repeated
- Better than relational tuples as a data-exchange format
  - Unlike relational tuples, XML data is self-documenting due to presence of tags
  - Non-rigid format: tags can be added
  - Allows nested structures
  - Wide acceptance, not only in database systems, but also in browsers, tools, and applications

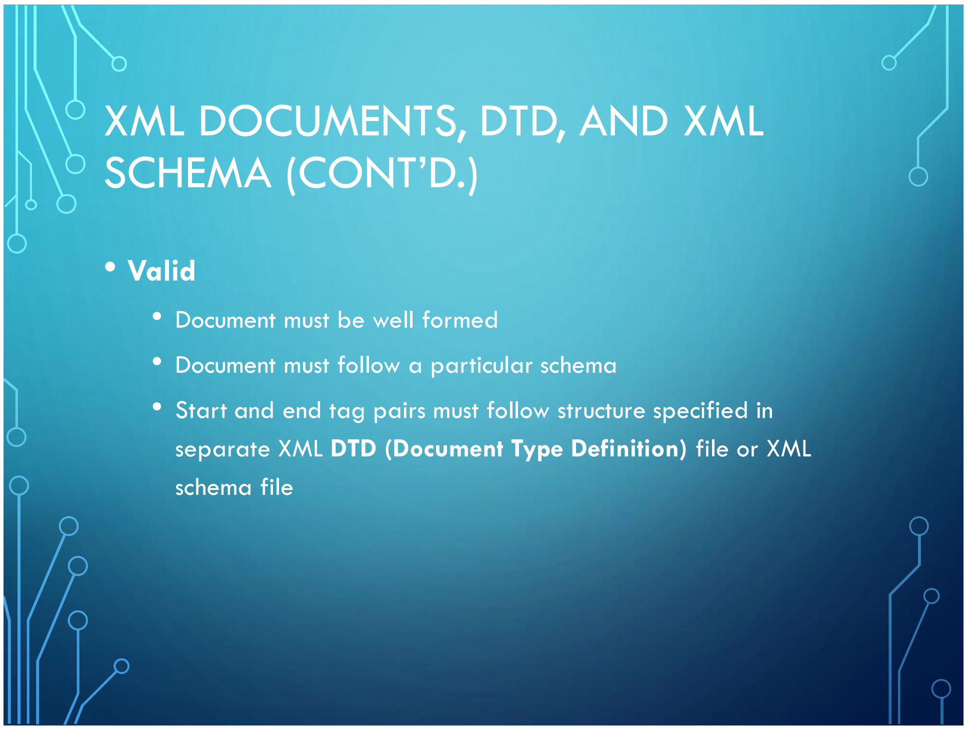
11



XML DOCUMENTS, DTD, AND XML SCHEMA

- **Well formed**
  - Has **XML declaration**
    - Indicates version of XML being used as well as any other relevant attributes
  - Every element must matching pair of start and end tags
    - Within start and end tags of parent element
- **DOM** (Document Object Model)
  - Manipulate resulting tree representation corresponding to a well-formed XML document

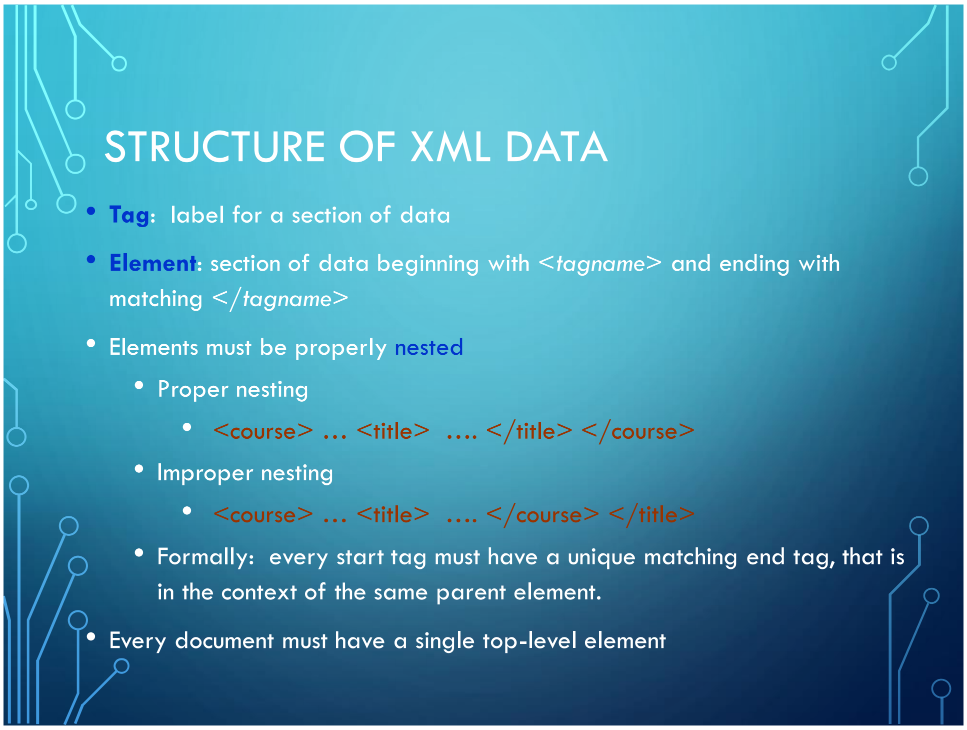
12



XML DOCUMENTS, DTD, AND XML SCHEMA (CONT'D.)

- **Valid**
  - Document must be well formed
  - Document must follow a particular schema
  - Start and end tag pairs must follow structure specified in separate XML **DTD (Document Type Definition)** file or XML schema file

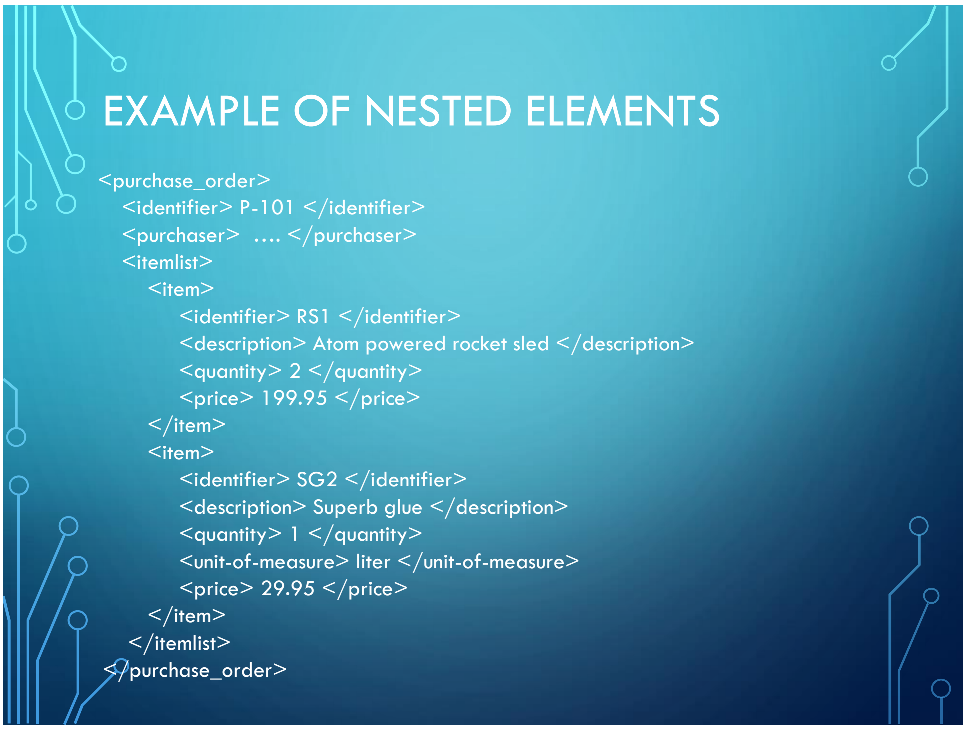
13



STRUCTURE OF XML DATA

- **Tag**: label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly nested
  - Proper nesting
    - <course> … <title> …. </title> </course>
  - Improper nesting
    - <course> … <title> …. </course> </title>
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element

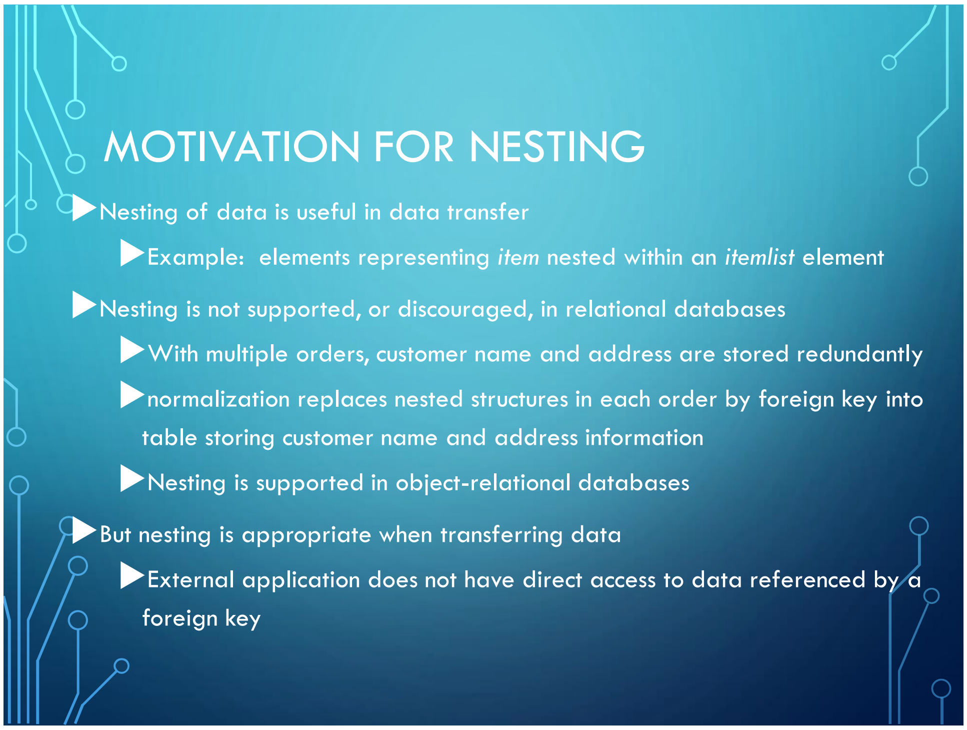
14



EXAMPLE OF NESTED ELEMENTS

```
<purchase_order>
   <identifier> P-101 </identifier>
   <purchaser>  .... </purchaser>
   <itemlist>
      <item>
          <identifier> RS1 </identifier>
          <description> Atom powered rocket sled </description>
          <quantity> 2 </quantity>
          <price> 199.95 </price>
      </item>
      <item>
          <identifier> SG2 </identifier>
          <description> Superb glue </description>
          <quantity> 1 </quantity>
          <unit-of-measure> liter </unit-of-measure>
          <price> 29.95 </price>
      </item>
    </itemlist>
</purchase_order>
```

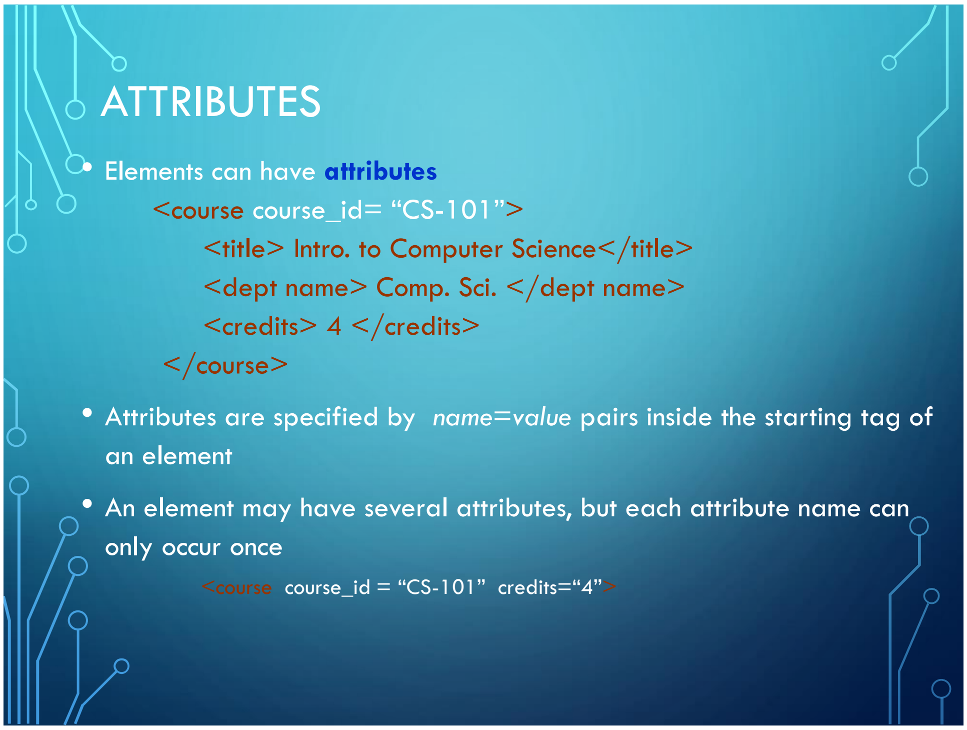
15



MOTIVATION FOR NESTING

- Nesting of data is useful in data transfer
  - Example: elements representing *item* nested within an *itemlist* element
- Nesting is not supported, or discouraged, in relational databases
  - With multiple orders, customer name and address are stored redundantly
  - normalization replaces nested structures in each order by foreign key into table storing customer name and address information
  - Nesting is supported in object-relational databases
- But nesting is appropriate when transferring data
  - External application does not have direct access to data referenced by a foreign key

16



ATTRIBUTES

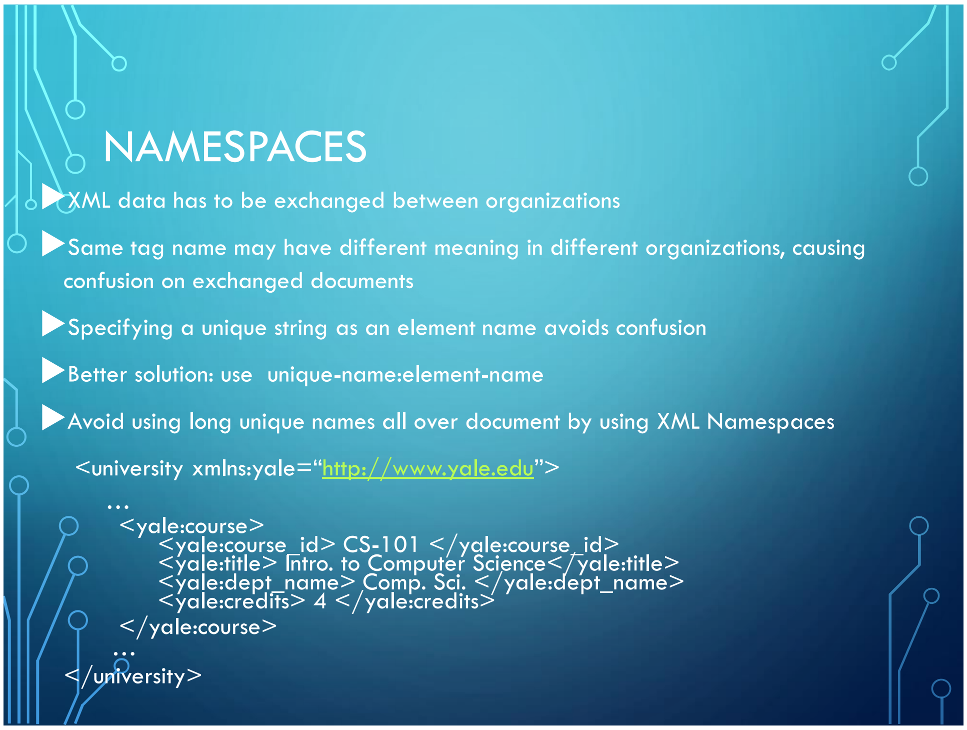
- Elements can have **attributes**

```
<course course_id= "CS-101">
    <title> Intro. to Computer Science</title>
    <dept name> Comp. Sci. </dept name>
    <credits> 4 </credits>
</course>
```

- Attributes are specified by *name*=*value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once

```
<course course_id = "CS-101" credits="4">
```

17

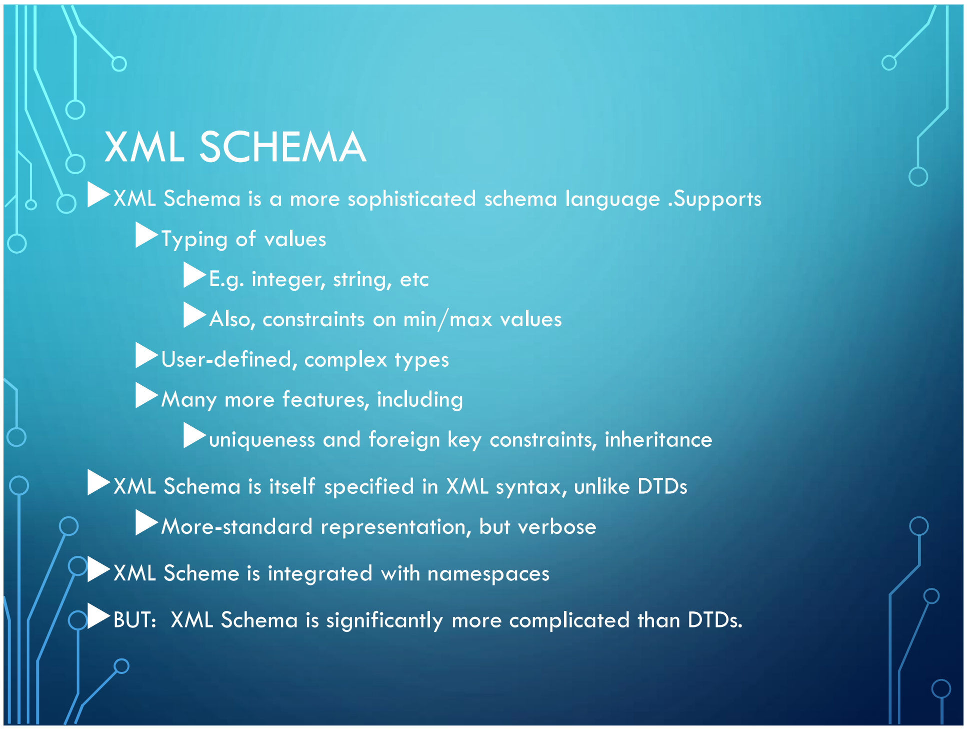


NAMESPACES

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name
- Avoid using long unique names all over document by using XML Namespaces

```
<university xmlns:yale="http://www.yale.edu">
  …
  <yale:course>
      <yale:course_id> CS-101 </yale:course_id>
      <yale:title> Intro. to Computer Science</yale:title>
      <yale:dept_name> Comp. Sci. </yale:dept_name>
      <yale:credits> 4 </yale:credits>
  </yale:course>
  …
</university>
```

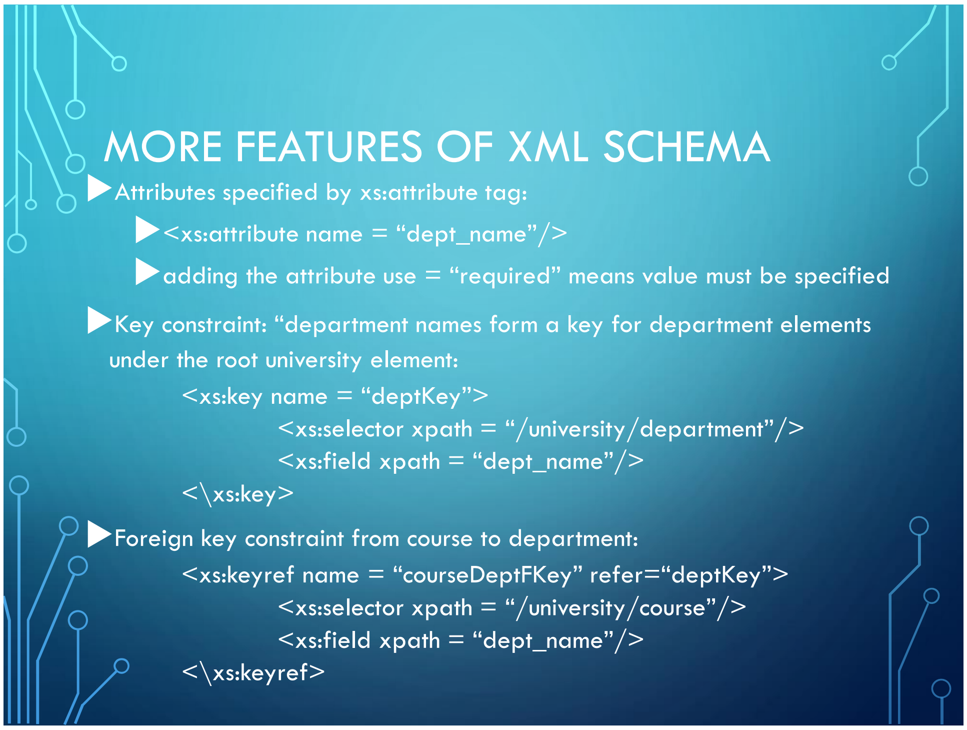
18



XML SCHEMA

- XML Schema is a more sophisticated schema language .Supports
  - Typing of values
    - E.g. integer, string, etc
    - Also, constraints on min/max values
  - User-defined, complex types
  - Many more features, including
    - uniqueness and foreign key constraints, inheritance
- XML Schema is itself specified in XML syntax, unlike DTDs
  - More-standard representation, but verbose
- XML Scheme is integrated with namespaces
- BUT: XML Schema is significantly more complicated than DTDs.

19



MORE FEATURES OF XML SCHEMA

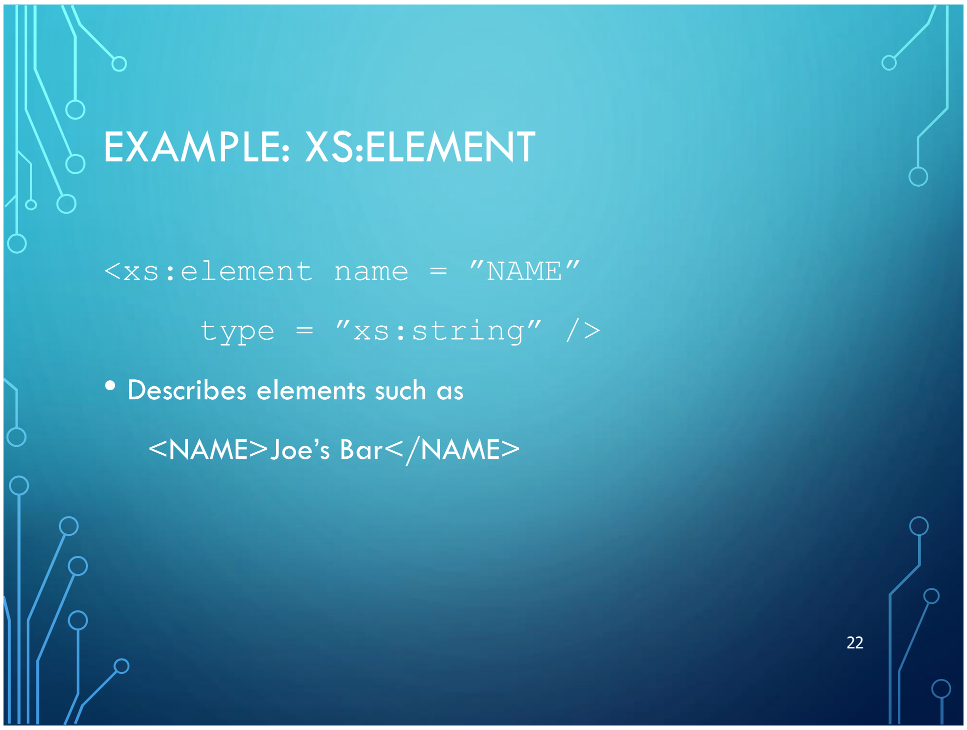
- Attributes specified by xs:attribute tag:
  - <xs:attribute name = “dept_name”/>
  - adding the attribute use = “required” means value must be specified
- Key constraint: “department names form a key for department elements under the root university element:

```
<xs:key name = “deptKey”>
        <xs:selector xpath = “/university/department”/>
        <xs:field xpath = “dept_name”/>
<\xs:key>
```

- Foreign key constraint from course to department:

```
<xs:keyref name = “courseDeptFKey” refer=“deptKey”>
        <xs:selector xpath = “/university/course”/>
        <xs:field xpath = “dept_name”/>
<\xs:keyref>
```

20



EXAMPLE: XS:ELEMENT

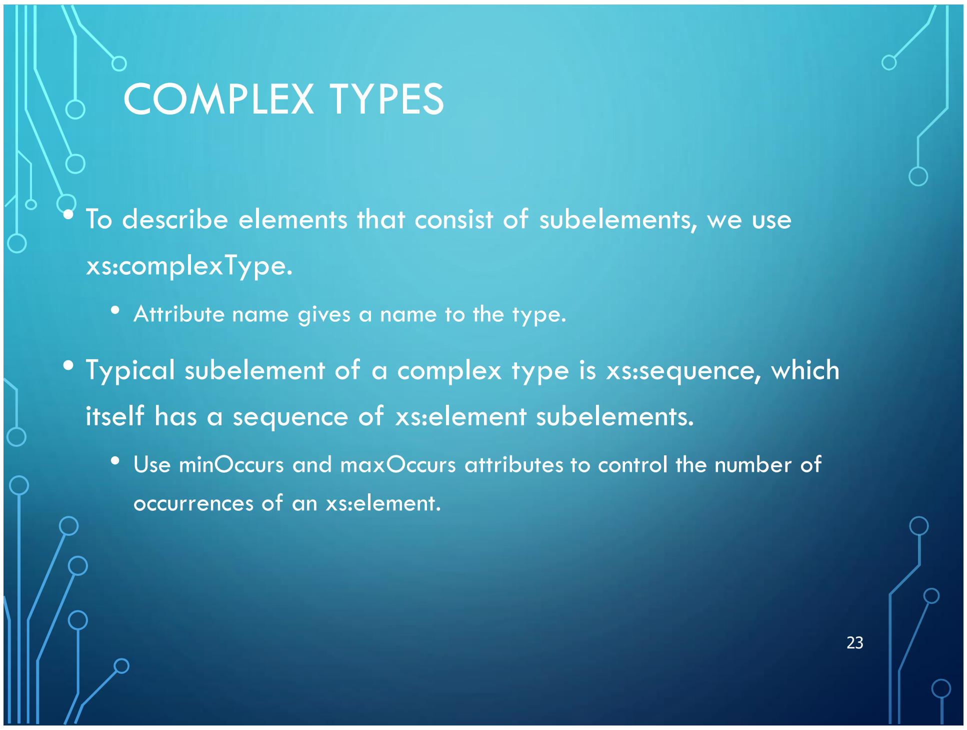
```
<xs:element name = ”NAME”
     type = ”xs:string” />
```

- Describes elements such as

  <NAME>Joe’s Bar</NAME>

22

22

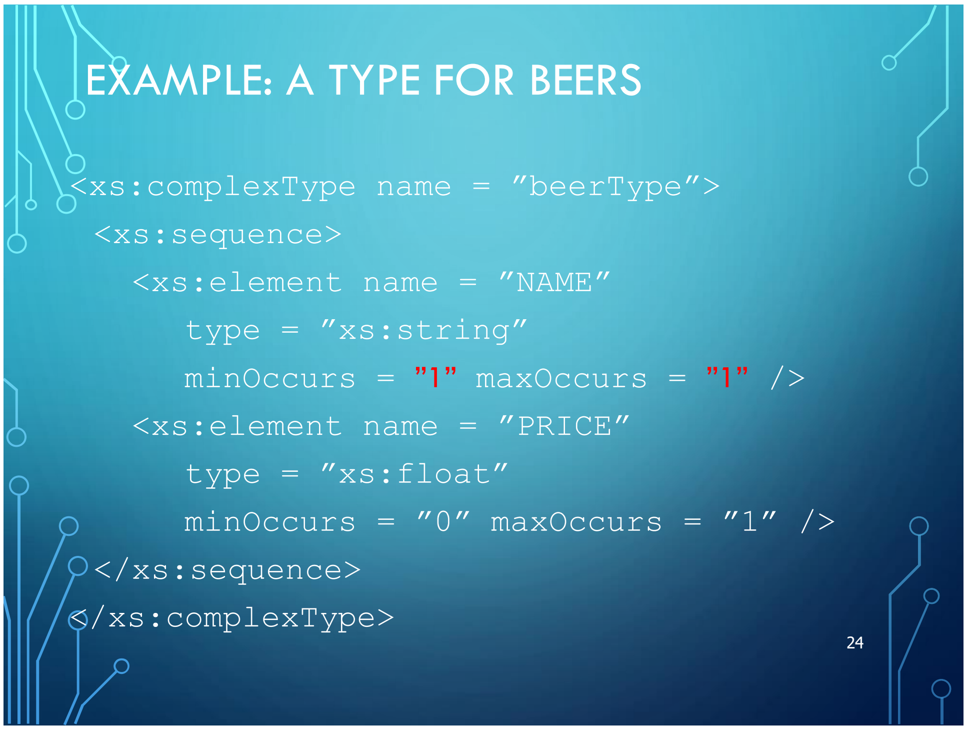


COMPLEX TYPES

- To describe elements that consist of subelements, we use xs:complexType.
  - Attribute name gives a name to the type.
- Typical subelement of a complex type is xs:sequence, which itself has a sequence of xs:element subelements.
  - Use minOccurs and maxOccurs attributes to control the number of occurrences of an xs:element.

23

23

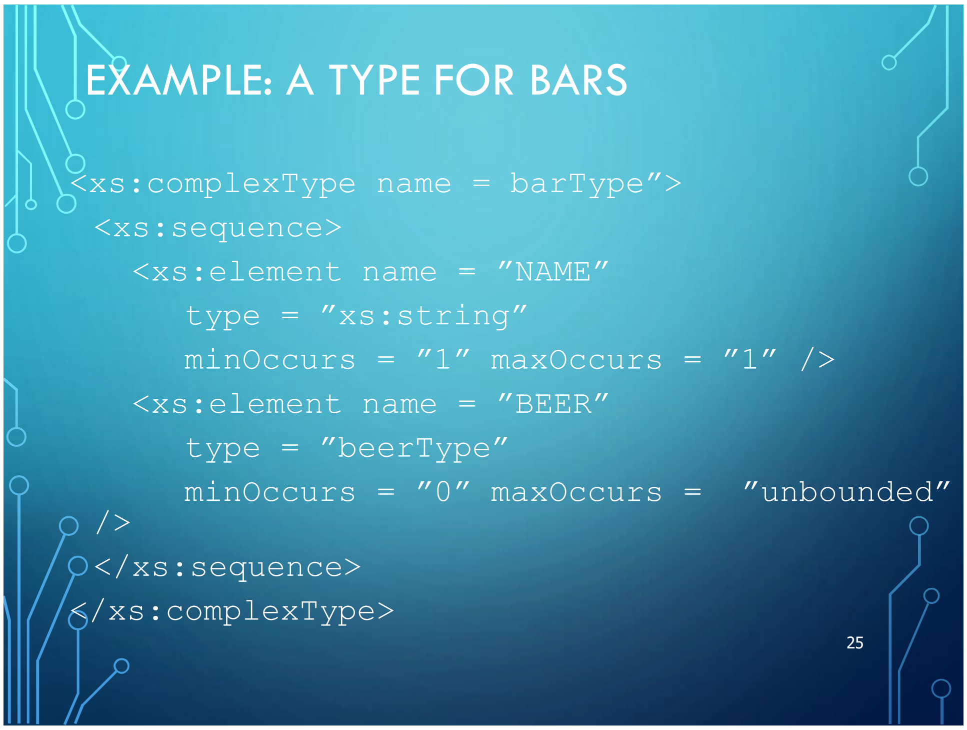


EXAMPLE: A TYPE FOR BEERS

```
<xs:complexType name = ”beerType”>
 <xs:sequence>
   <xs:element name = ”NAME”
      type = ”xs:string”
      minOccurs = ”1” maxOccurs = ”1” />
   <xs:element name = ”PRICE”
      type = ”xs:float”
      minOccurs = ”0” maxOccurs = ”1” />
 </xs:sequence>
</xs:complexType>
```

24

24

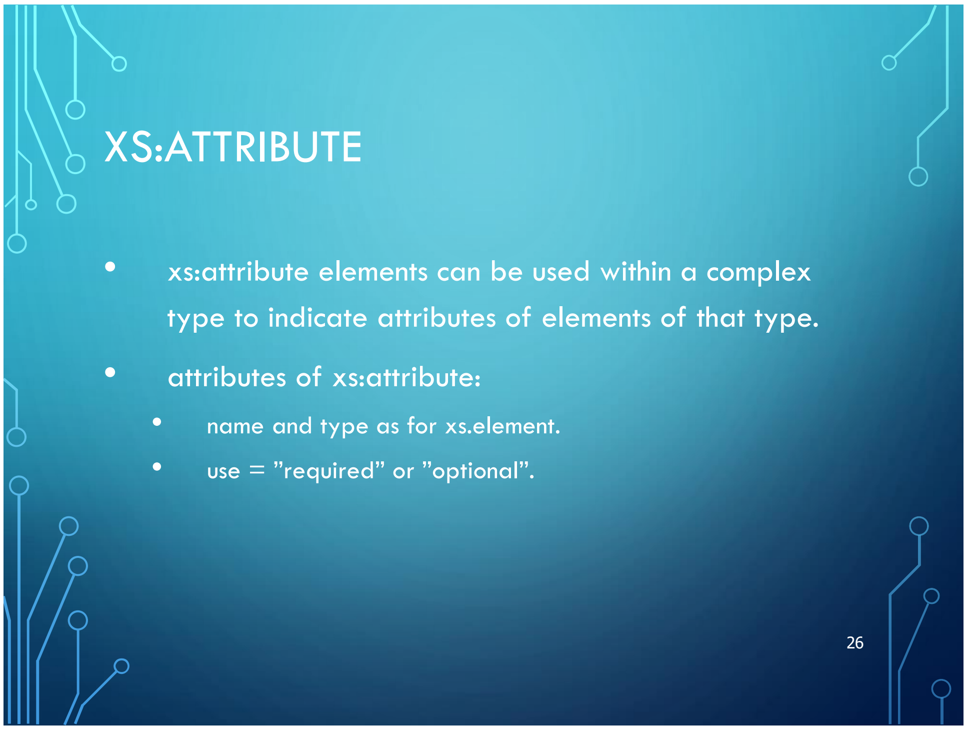


EXAMPLE: A TYPE FOR BARS

```
<xs:complexType name = barType”>
 <xs:sequence>
   <xs:element name = ”NAME”
      type = ”xs:string”
      minOccurs = ”1” maxOccurs = ”1” />
   <xs:element name = ”BEER”
      type = ”beerType”
      minOccurs = ”0” maxOccurs =  ”unbounded”
 />
 </xs:sequence>
</xs:complexType>
```

25

25

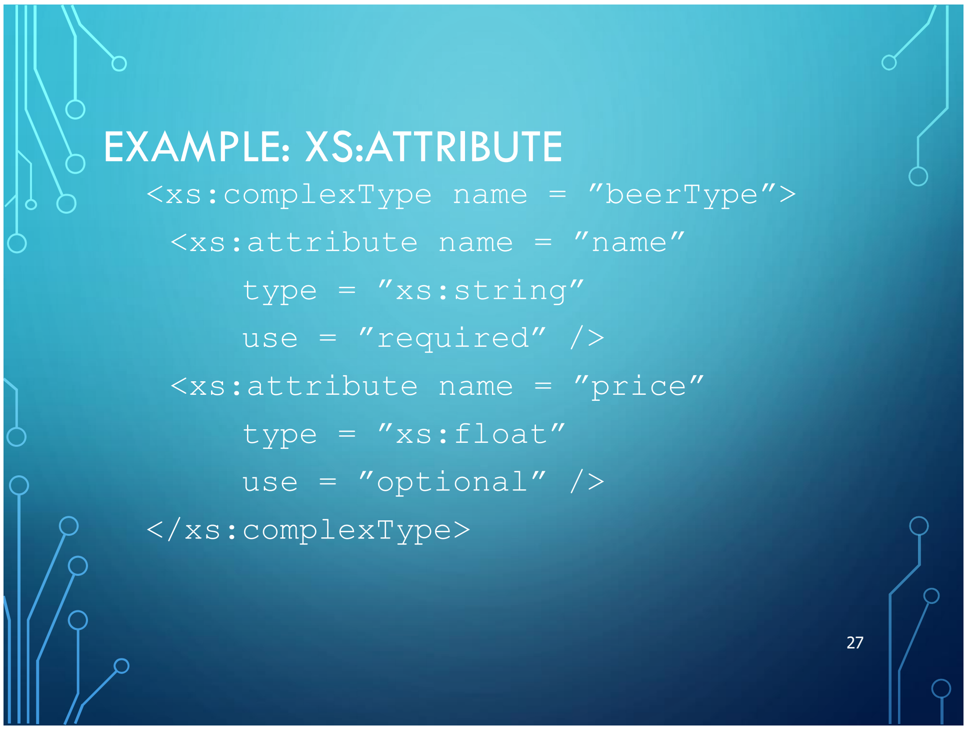


XS:ATTRIBUTE

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

26

26

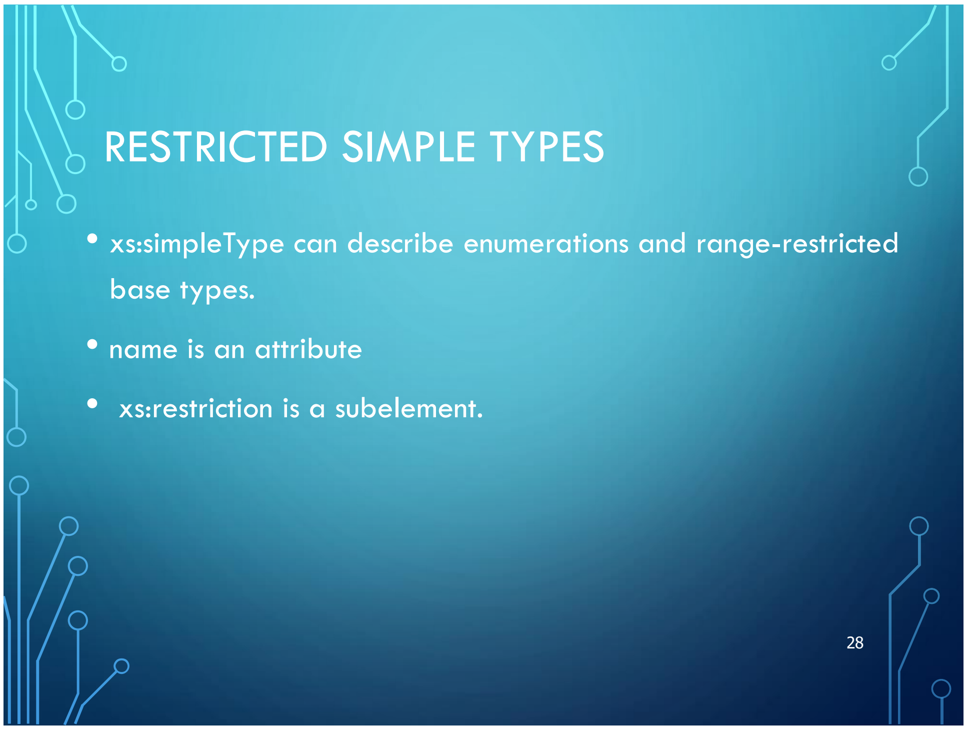


EXAMPLE: XS:ATTRIBUTE

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

27

27

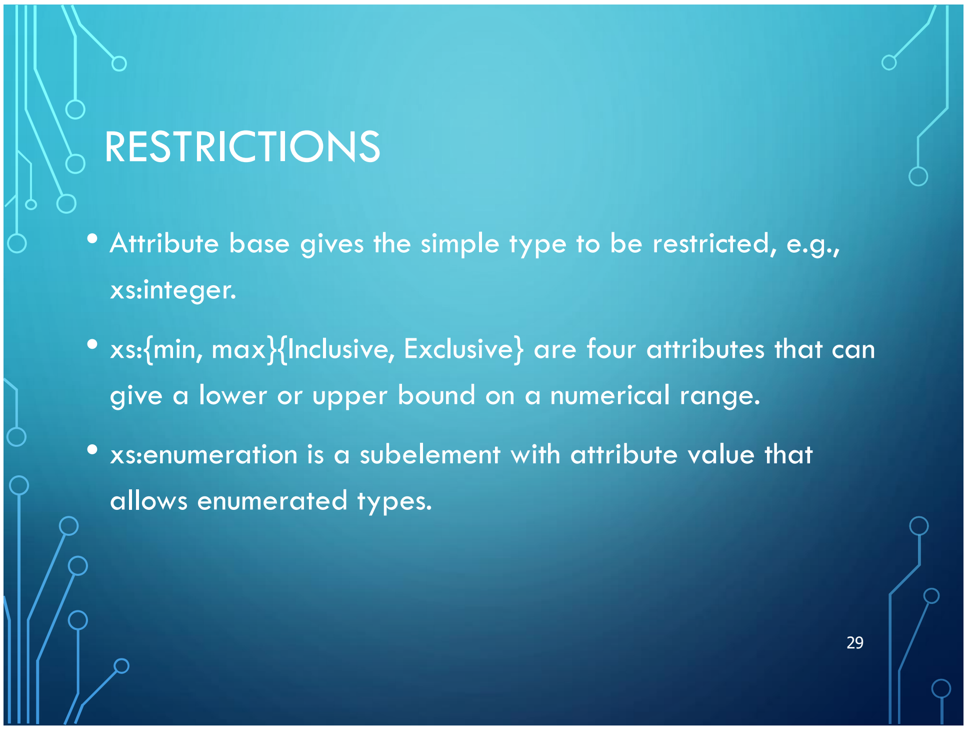


RESTRICTED SIMPLE TYPES

- xs:simpleType can describe enumerations and range-restricted base types.
- name is an attribute
- xs:restriction is a subelement.

28

28

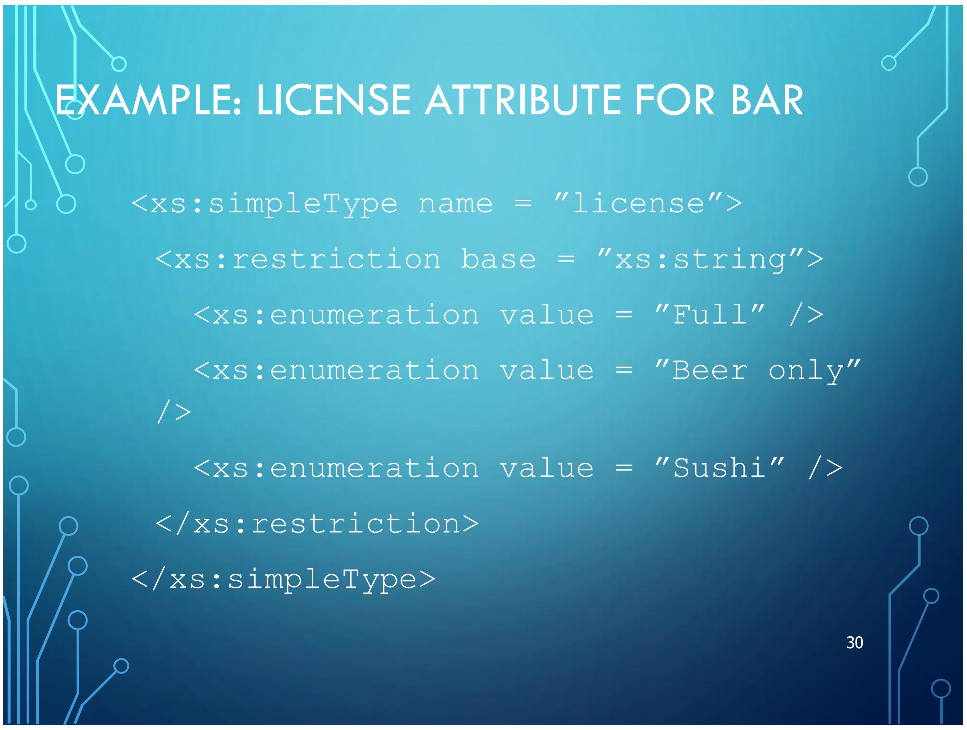


RESTRICTIONS

- Attribute base gives the simple type to be restricted, e.g., xs:integer.
- xs:{min, max}{Inclusive, Exclusive} are four attributes that can give a lower or upper bound on a numerical range.
- xs:enumeration is a subelement with attribute value that allows enumerated types.

29

29

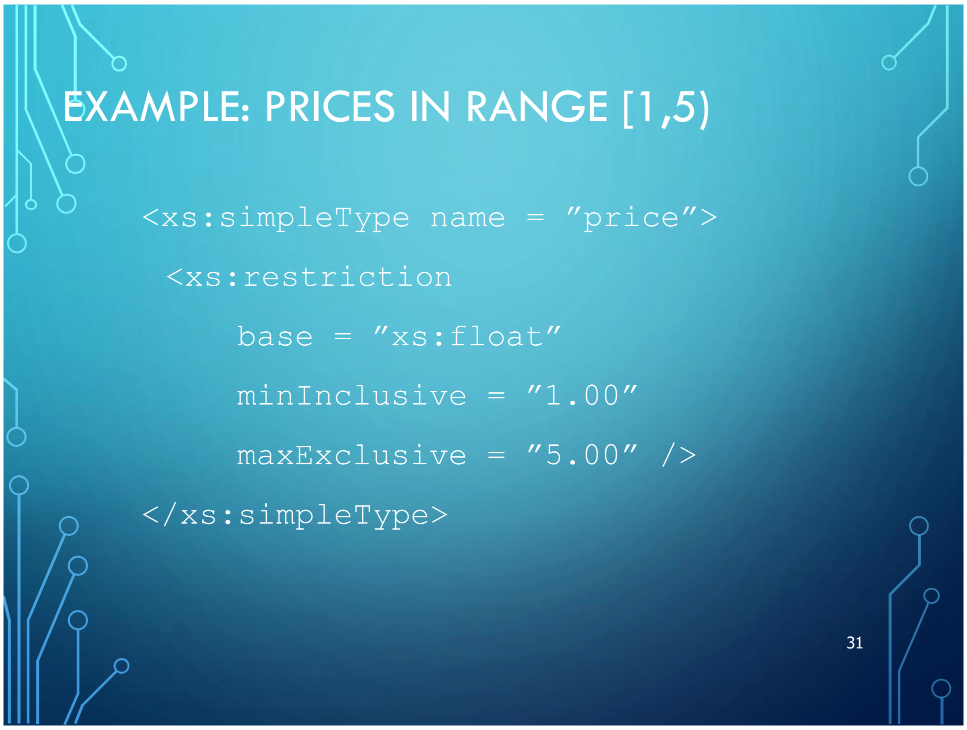


EXAMPLE: LICENSE ATTRIBUTE FOR BAR

```
<xs:simpleType name = ”license”>
 <xs:restriction base = ”xs:string”>
   <xs:enumeration value = ”Full” />
   <xs:enumeration value = ”Beer only”
 />
   <xs:enumeration value = ”Sushi” />
 </xs:restriction>
</xs:simpleType>
```

30

30

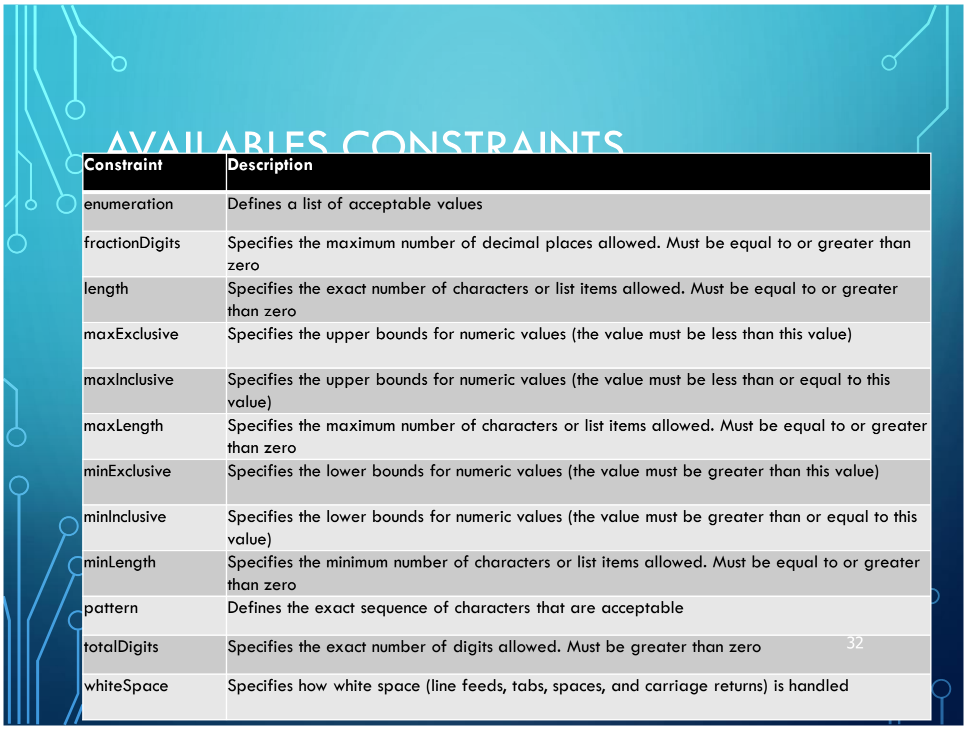


EXAMPLE: PRICES IN RANGE [1,5)

```
<xs:simpleType name = ”price”>
 <xs:restriction
     base = ”xs:float”
     minInclusive = ”1.00”
     maxExclusive = ”5.00” />
</xs:simpleType>
```

31

31

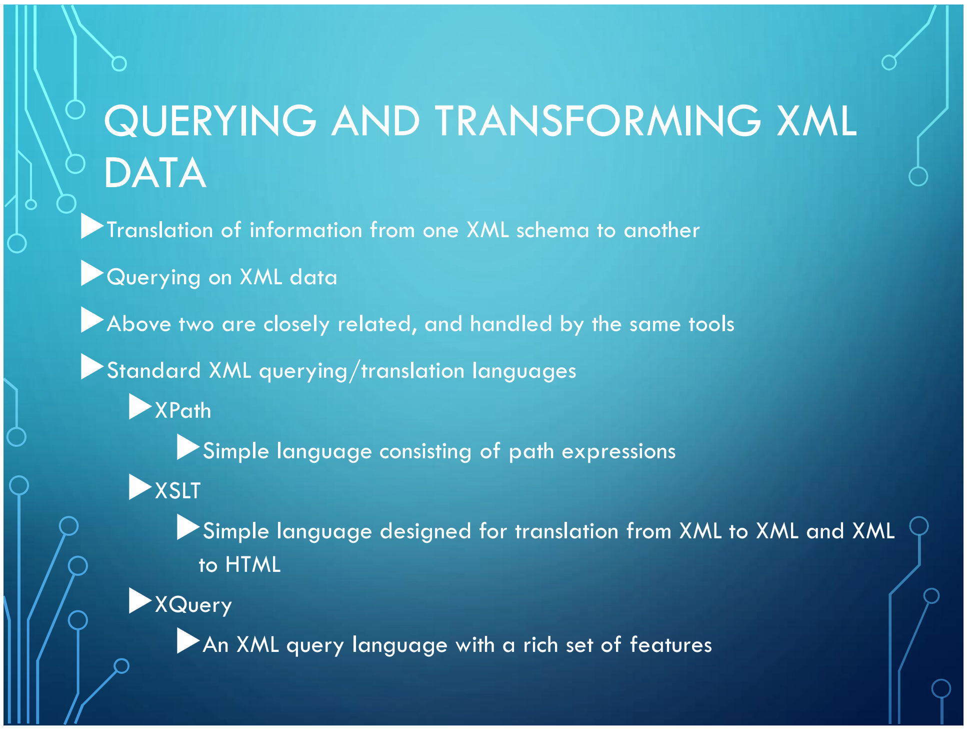


AVAILABLES CONSTRAINTS

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

32

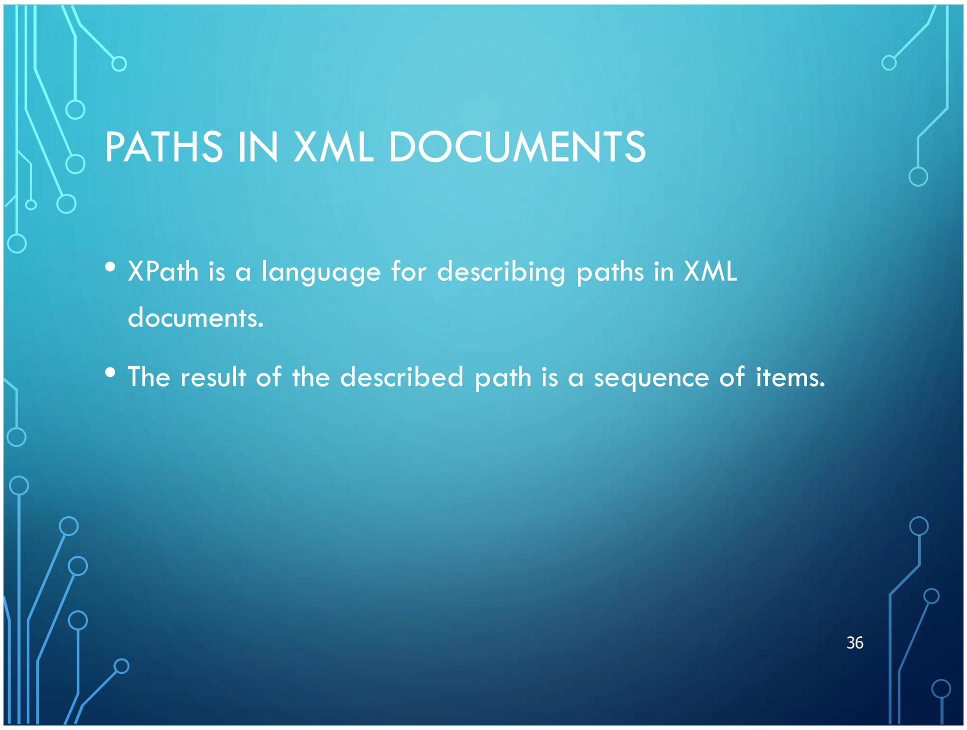
32



QUERYING AND TRANSFORMING XML DATA

- Translation of information from one XML schema to another
- Querying on XML data
- Above two are closely related, and handled by the same tools
- Standard XML querying/translation languages
  - XPath
    - Simple language consisting of path expressions
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XQuery
    - An XML query language with a rich set of features

35

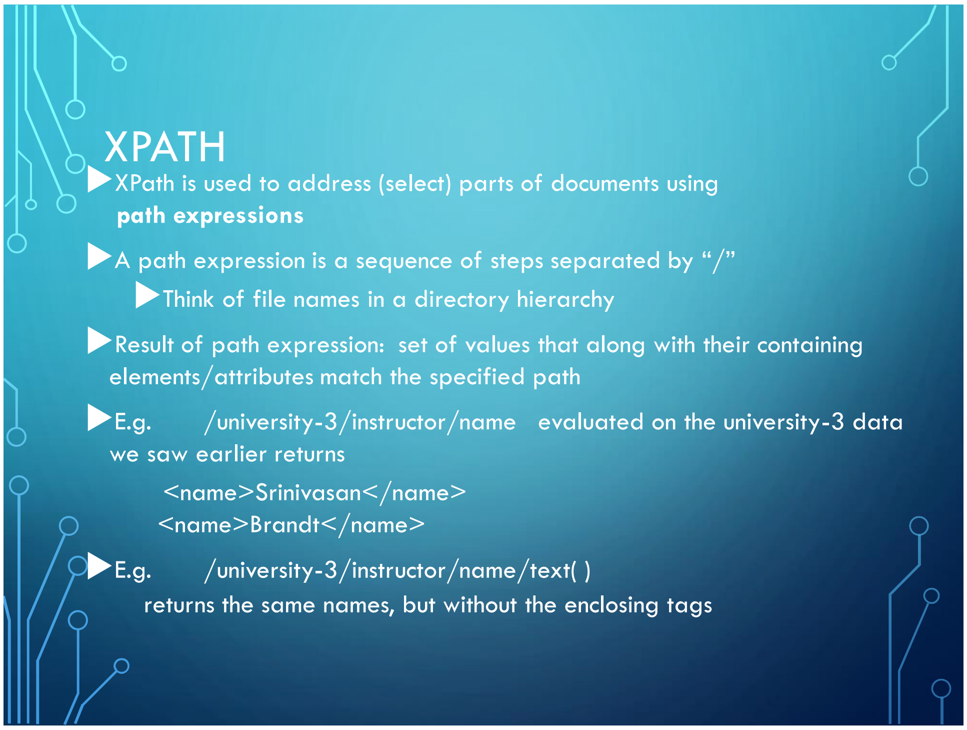


PATHS IN XML DOCUMENTS

- XPath is a language for describing paths in XML documents.
- The result of the described path is a sequence of items.

36

36



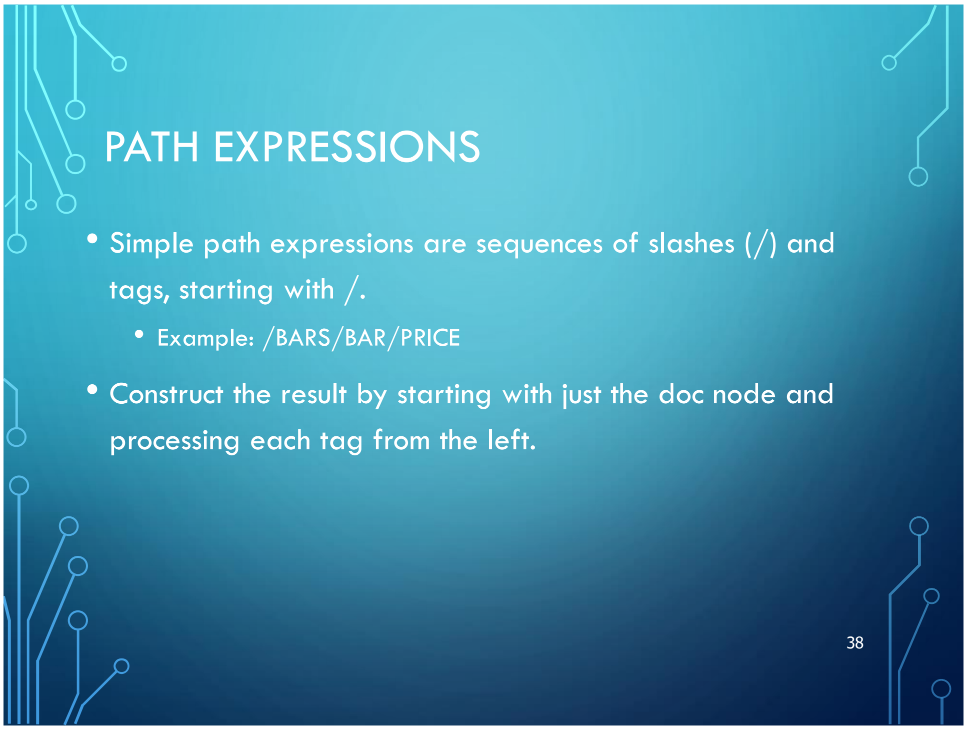
XPATH

- XPath is used to address (select) parts of documents using **path expressions**
- A path expression is a sequence of steps separated by "/"
  - Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
- E.g. /university-3/instructor/name evaluated on the university-3 data we saw earlier returns

```
<name>Srinivasan</name>
<name>Brandt</name>
```

- E.g. /university-3/instructor/name/text( )
  returns the same names, but without the enclosing tags

37

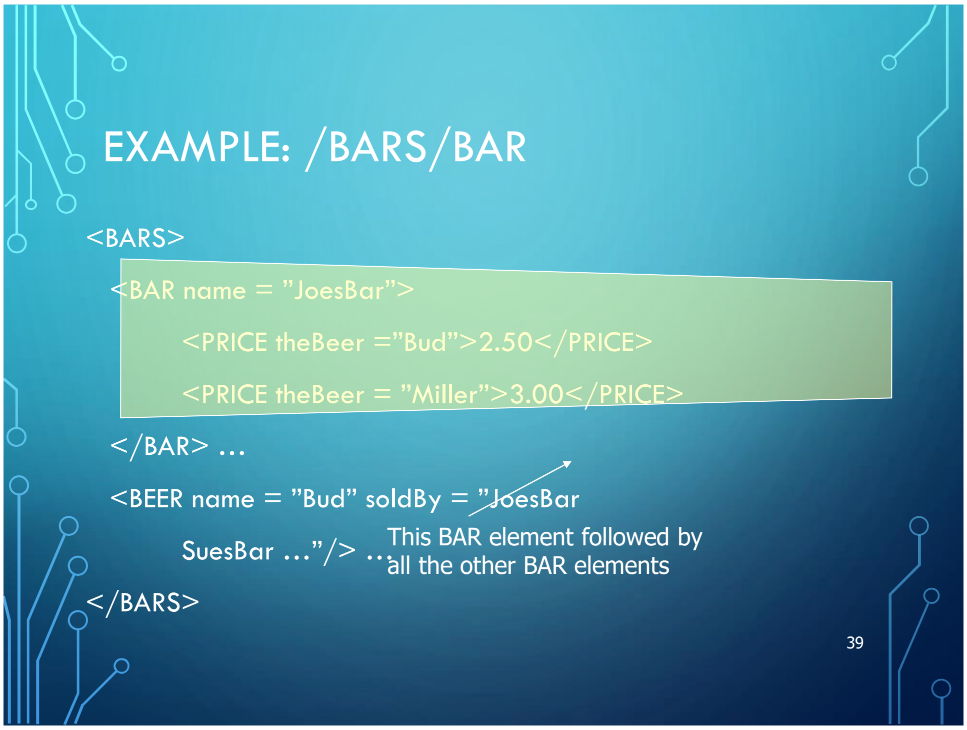


PATH EXPRESSIONS

- 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.

38

38



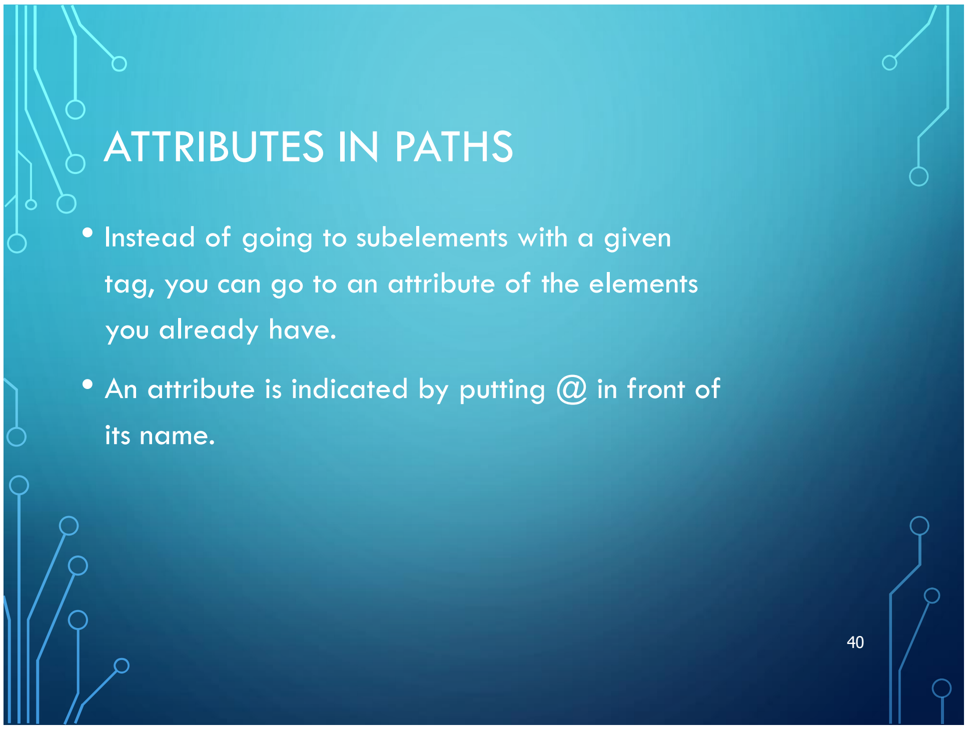
EXAMPLE: /BARS/BAR

```
<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

39

39

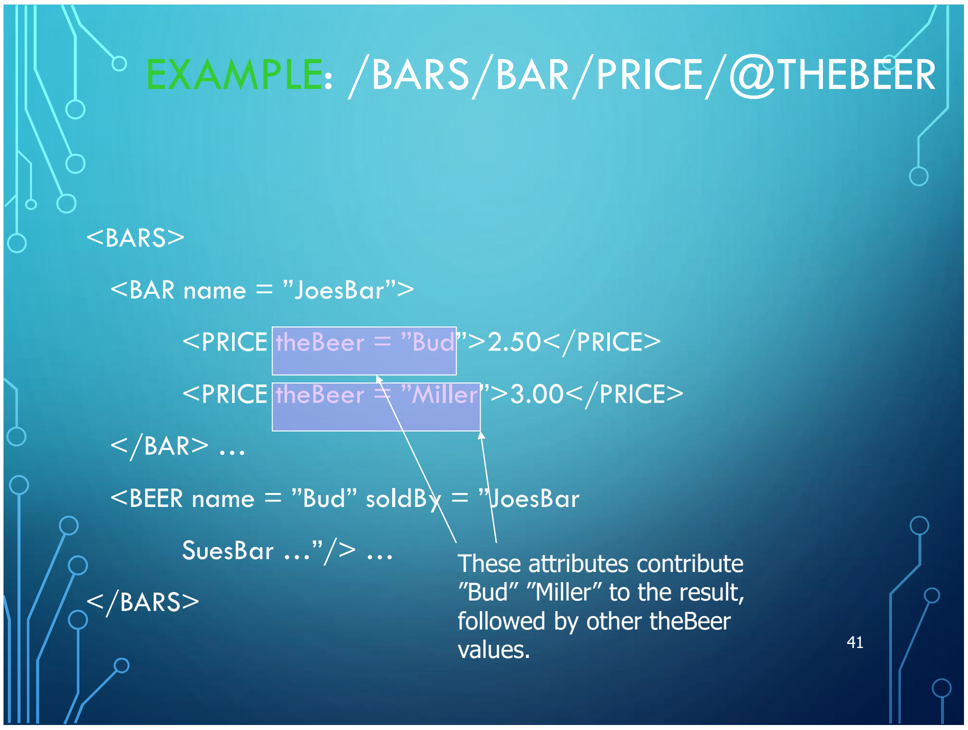


ATTRIBUTES IN PATHS

- 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.

40

40



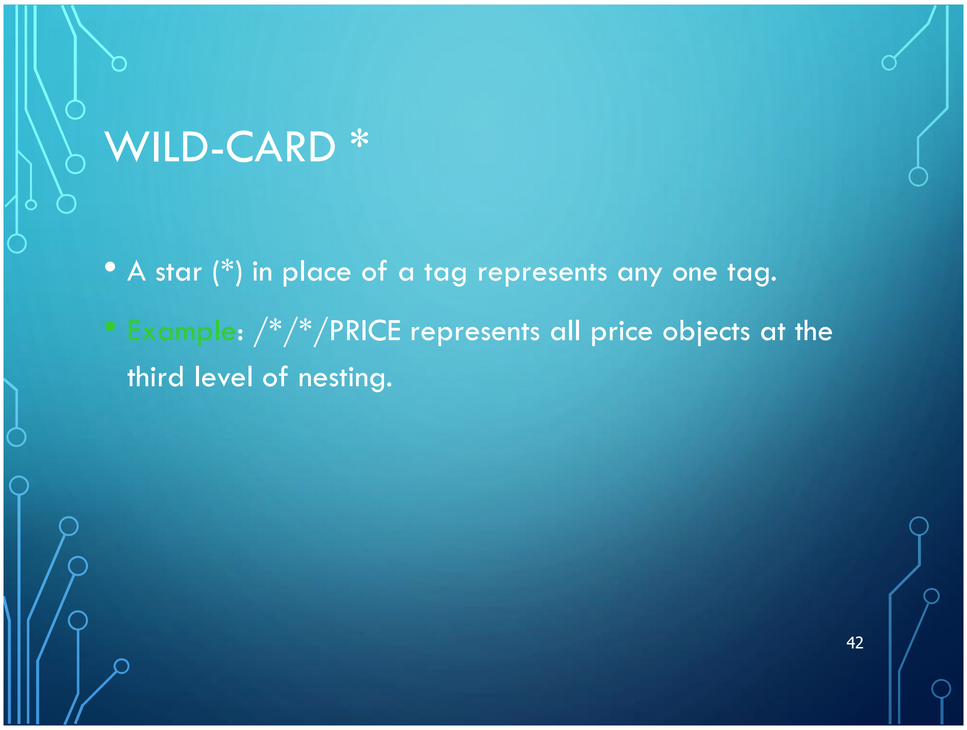
EXAMPLE: /BARS/BAR/PRICE/@THEBEER

```
<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.

41

41

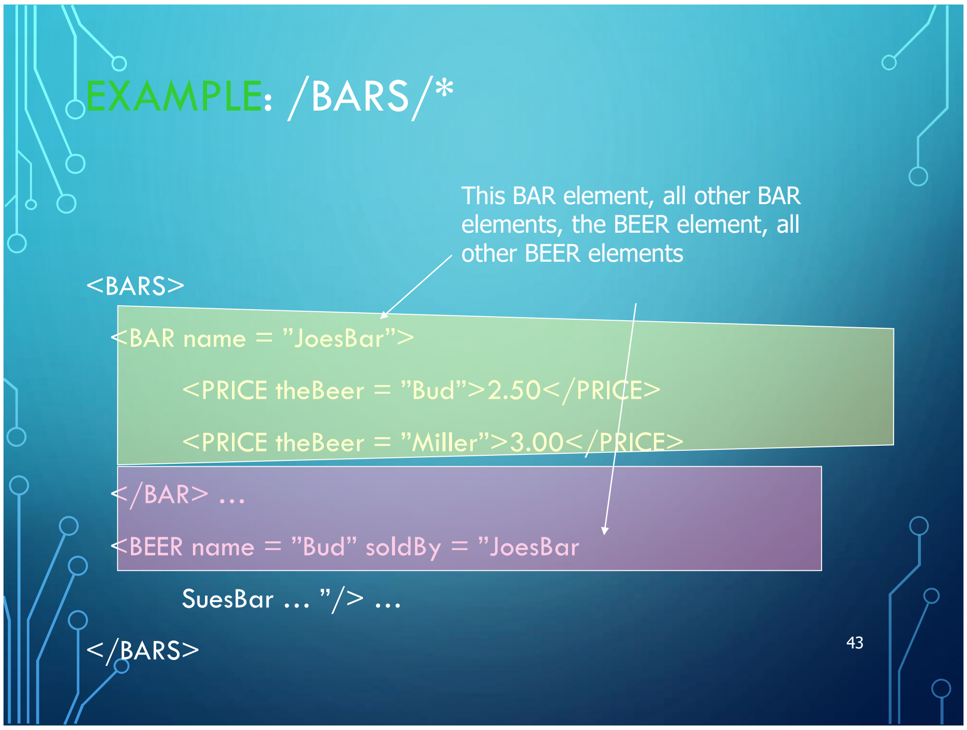


WILD-CARD *

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

42

42



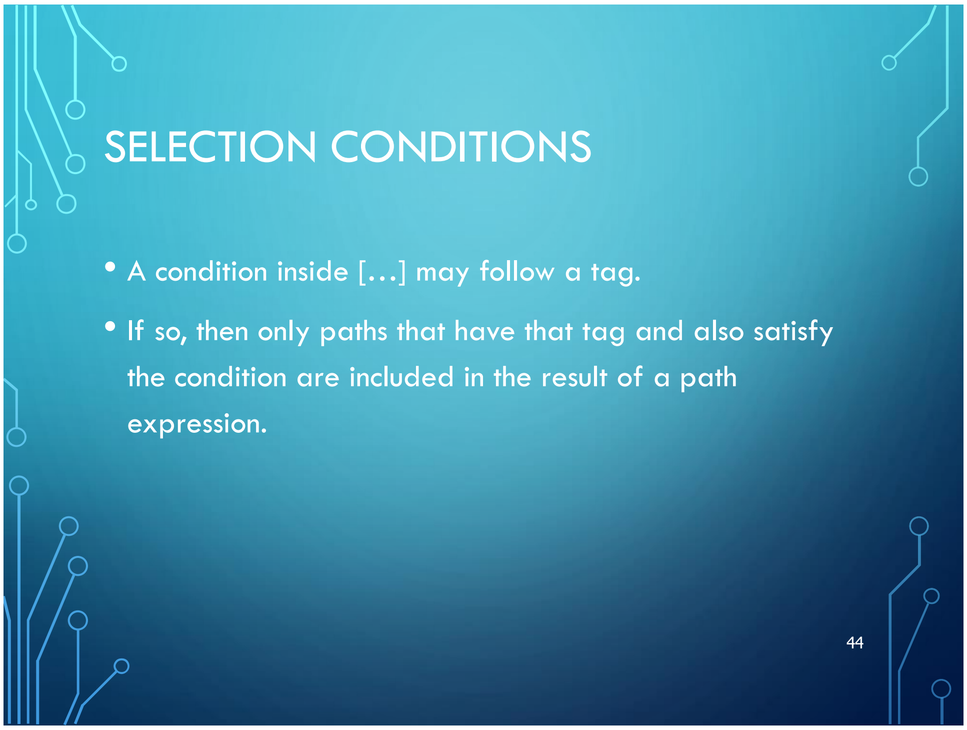
EXAMPLE: /BARS/*

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>
```

43

43

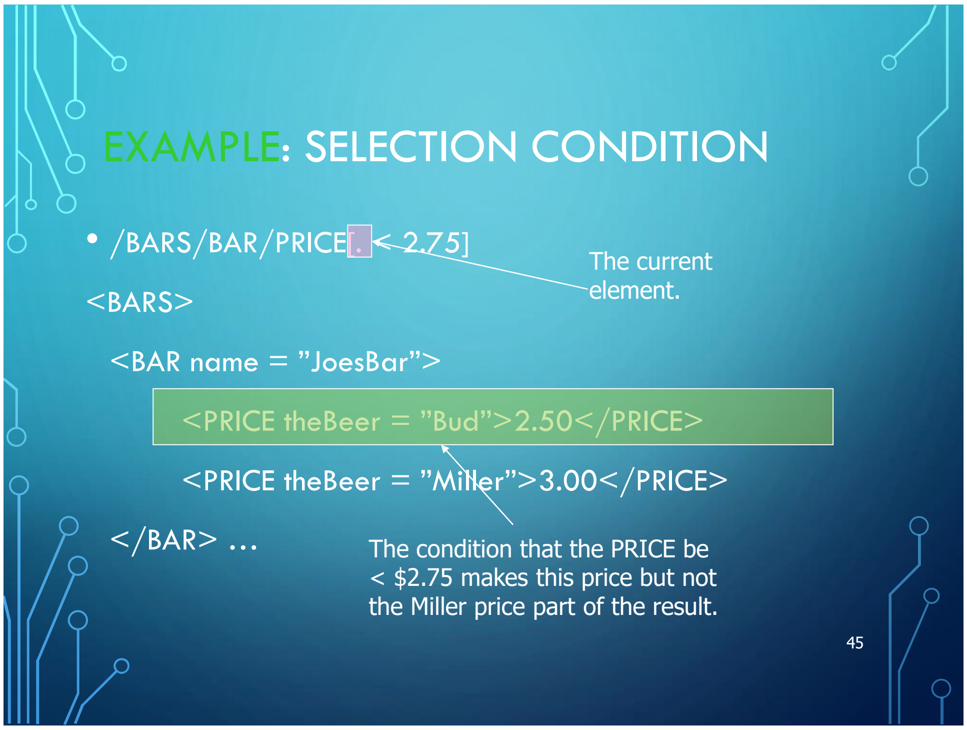


SELECTION CONDITIONS

- 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.

44

44



EXAMPLE: SELECTION CONDITION

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

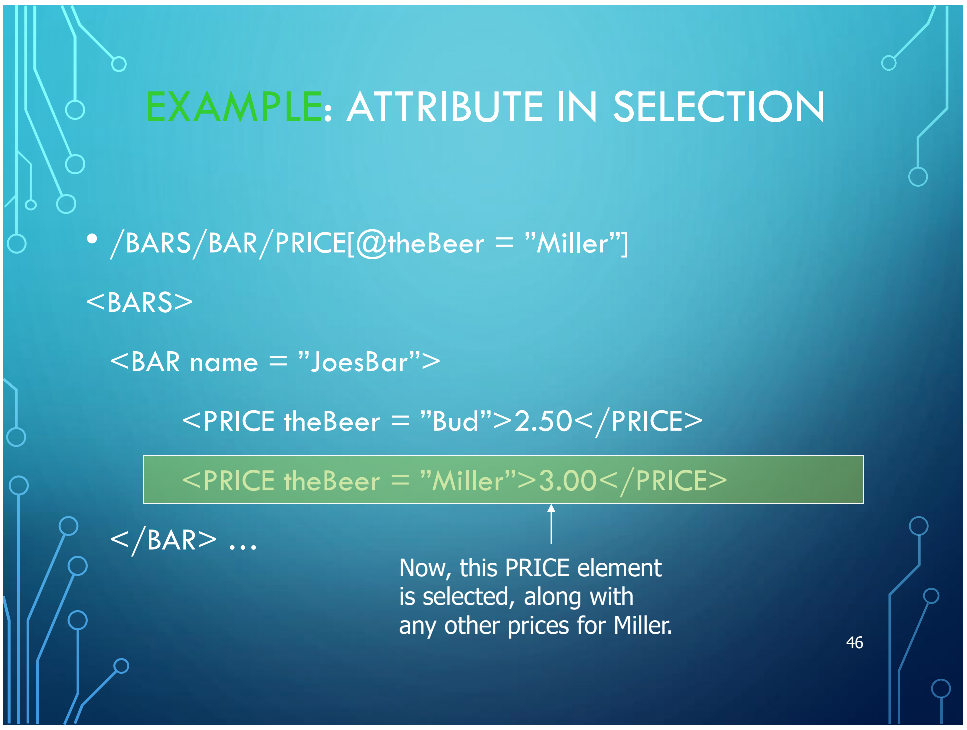
The current element.

```
<BARS>
  <BAR name = ”JoesBar”>
      <PRICE theBeer = ”Bud”>2.50</PRICE>
      <PRICE theBeer = ”Miller”>3.00</PRICE>
  </BAR> …
```

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

45

45



EXAMPLE: ATTRIBUTE IN SELECTION

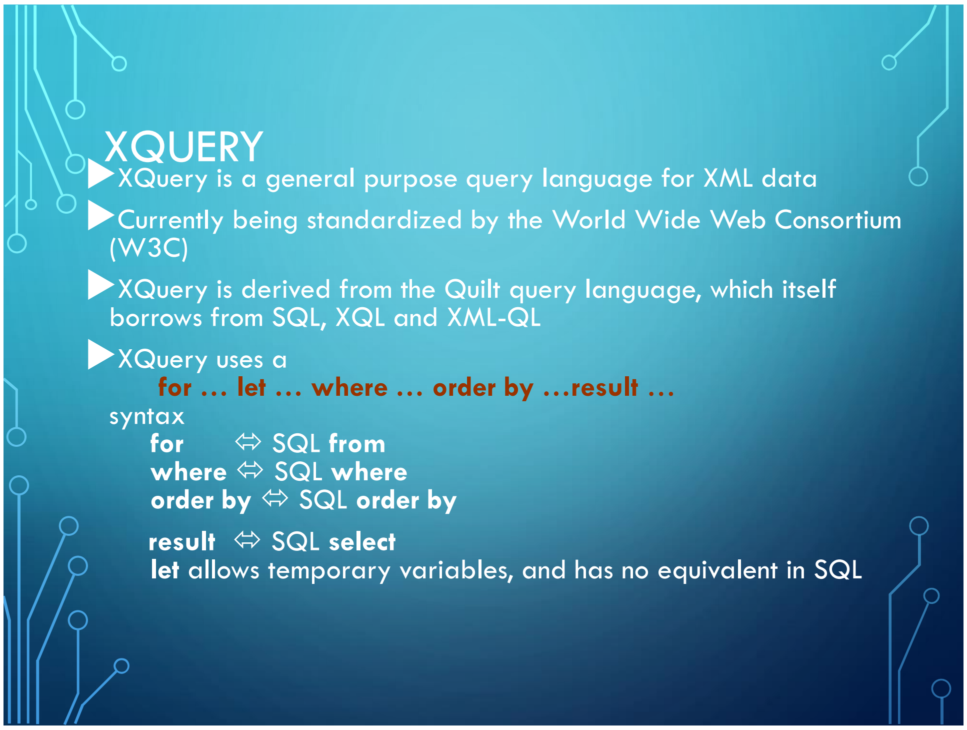
- /BARS/BAR/PRICE[@theBeer = ”Miller”]

```
<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.

46

46

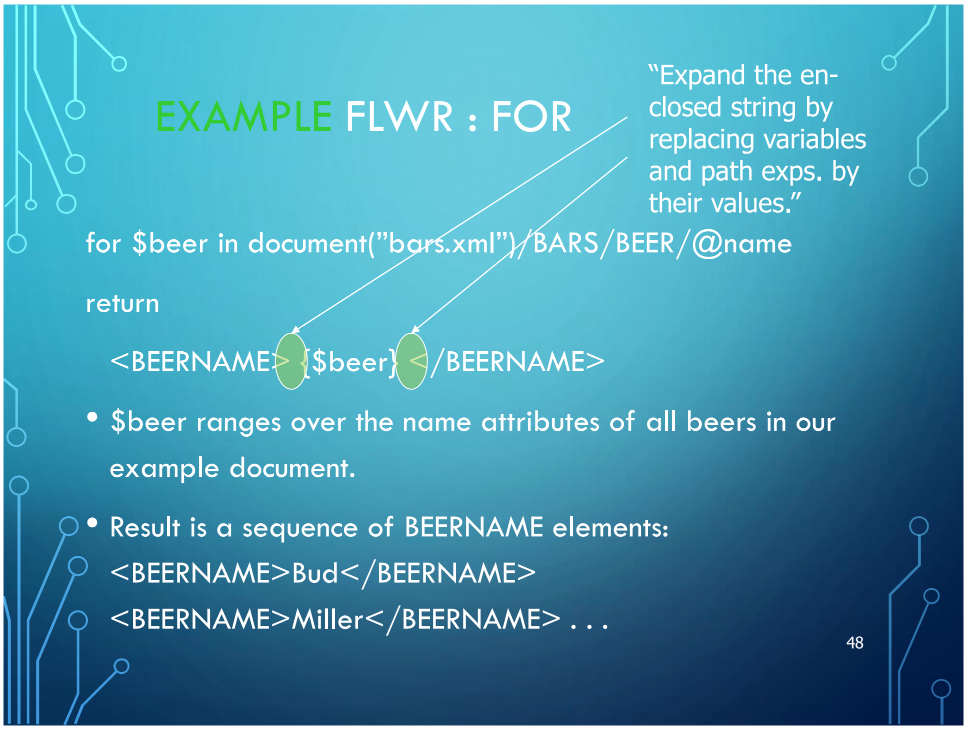


XQUERY

- XQuery is a general purpose query language for XML data
- Currently being standardized by the World Wide Web Consortium (W3C)
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- XQuery uses a
  for … let … where … order by …result …
  syntax
  for ⇔ SQL **from**
  where ⇔ SQL **where**
  order by ⇔ SQL **order by**

  result ⇔ SQL **select**
  let allows temporary variables, and has no equivalent in SQL

47



EXAMPLE FLWR : FOR

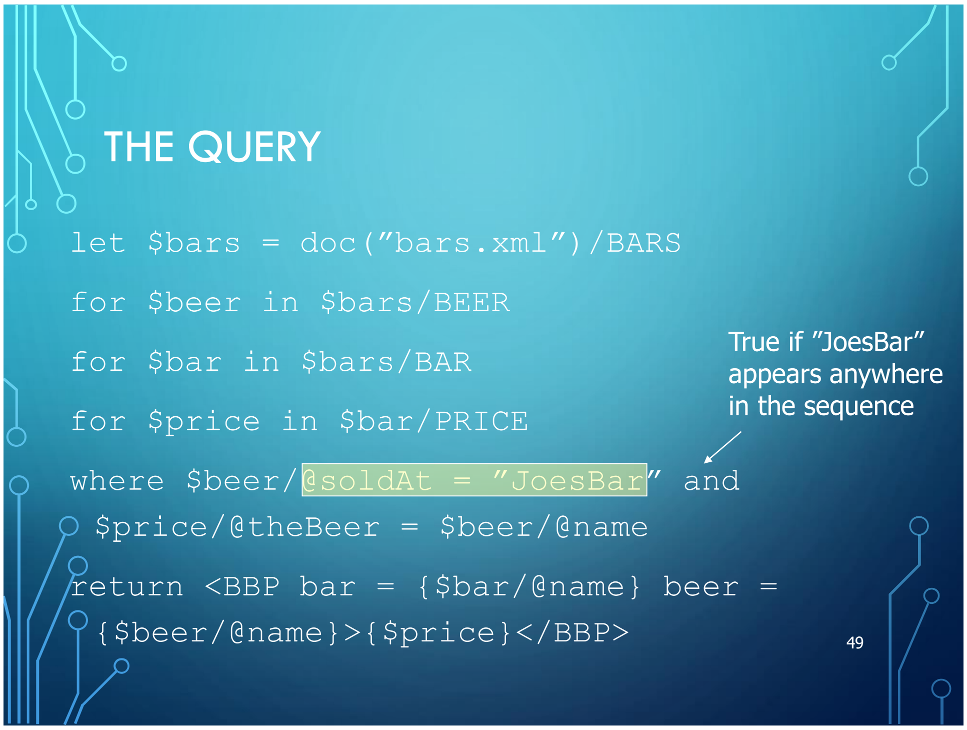
"Expand the en-
closed string by
replacing variables
and path exps. by
their values."

```
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> . . .

48

48



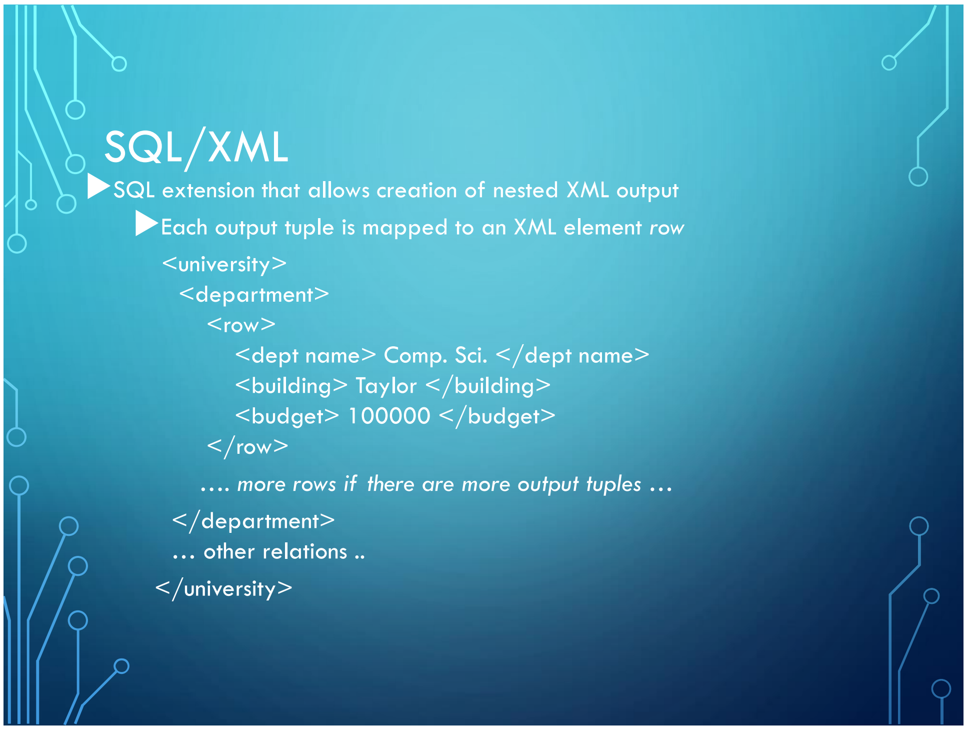
THE QUERY

```
let $bars = doc("bars.xml")/BARS
for $beer in $bars/BEER
for $bar in $bars/BAR
for $price in $bar/PRICE
where $beer/@soldAt = "JoesBar" and
 $price/@theBeer = $beer/@name
return <BBP bar = {$bar/@name} beer =
 {$beer/@name}>{$price}</BBP>
```

True if "JoesBar" appears anywhere in the sequence

49

49

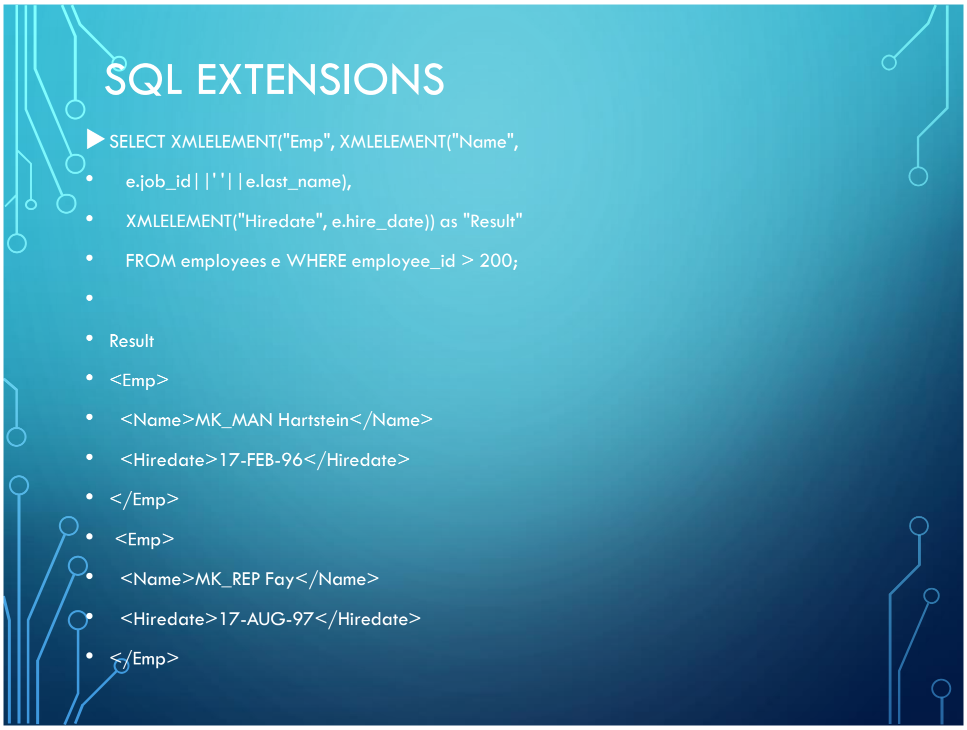


SQL/XML

- SQL extension that allows creation of nested XML output
  - Each output tuple is mapped to an XML element *row*

```
<university>
  <department>
     <row>
        <dept name> Comp. Sci. </dept name>
        <building> Taylor </building>
        <budget> 100000 </budget>
     </row>
    .... more rows if there are more output tuples ...
  </department>
  ... other relations ..
</university>
```

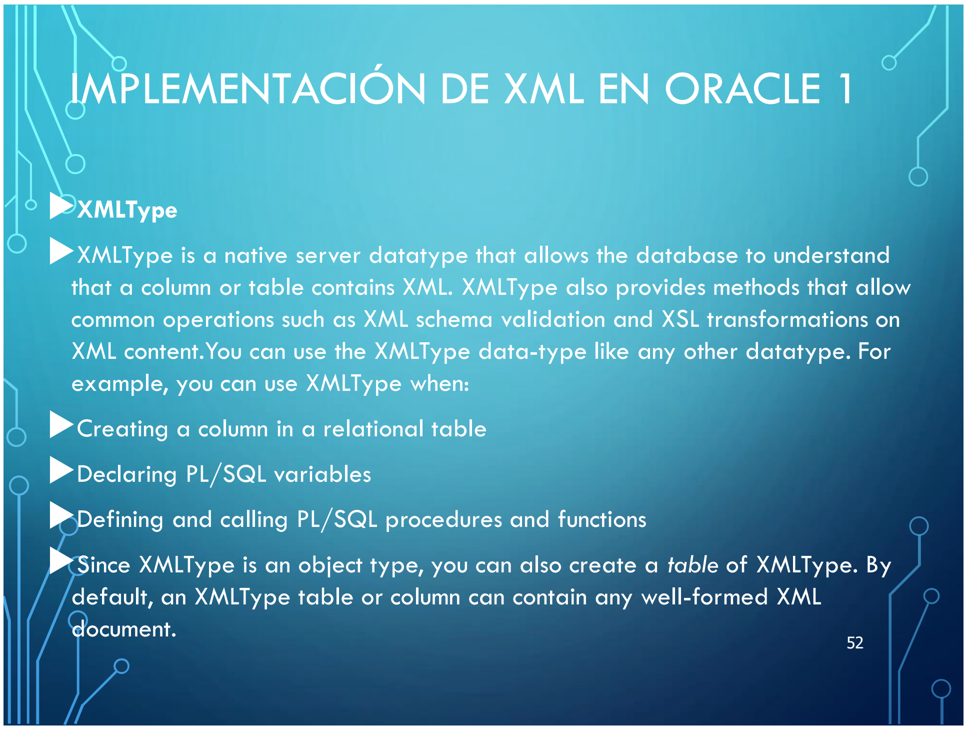
50



SQL EXTENSIONS

- SELECT XMLELEMENT("Emp", XMLELEMENT("Name",
-   e.job_id||' '||e.last_name),
-   XMLELEMENT("Hiredate", e.hire_date)) as "Result"
-   FROM employees e WHERE employee_id > 200;
-
- Result
- <Emp>
-   <Name>MK_MAN Hartstein</Name>
-   <Hiredate>17-FEB-96</Hiredate>
- </Emp>
- <Emp>
-   <Name>MK_REP Fay</Name>
-   <Hiredate>17-AUG-97</Hiredate>
- </Emp>

51

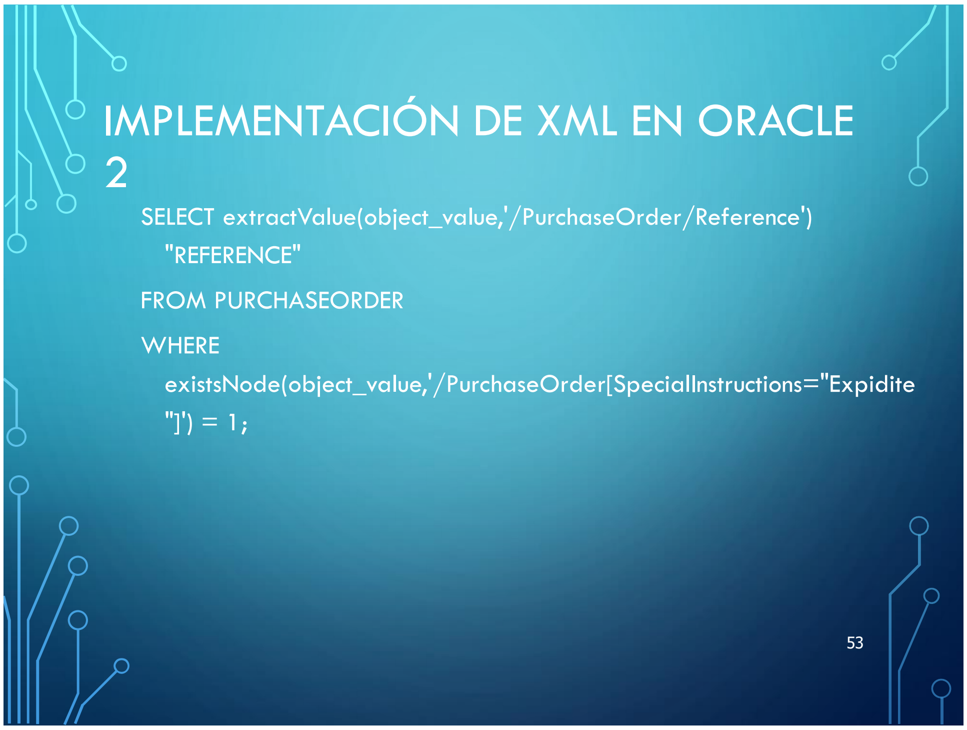


IMPLEMENTACIÓN DE XML EN ORACLE 1

- **XMLType**
- XMLType is a native server datatype that allows the database to understand that a column or table contains XML. XMLType also provides methods that allow common operations such as XML schema validation and XSL transformations on XML content.You can use the XMLType data-type like any other datatype. For example, you can use XMLType when:
- Creating a column in a relational table
- Declaring PL/SQL variables
- Defining and calling PL/SQL procedures and functions
- Since XMLType is an object type, you can also create a *table* of XMLType. By default, an XMLType table or column can contain any well-formed XML document.

52

52

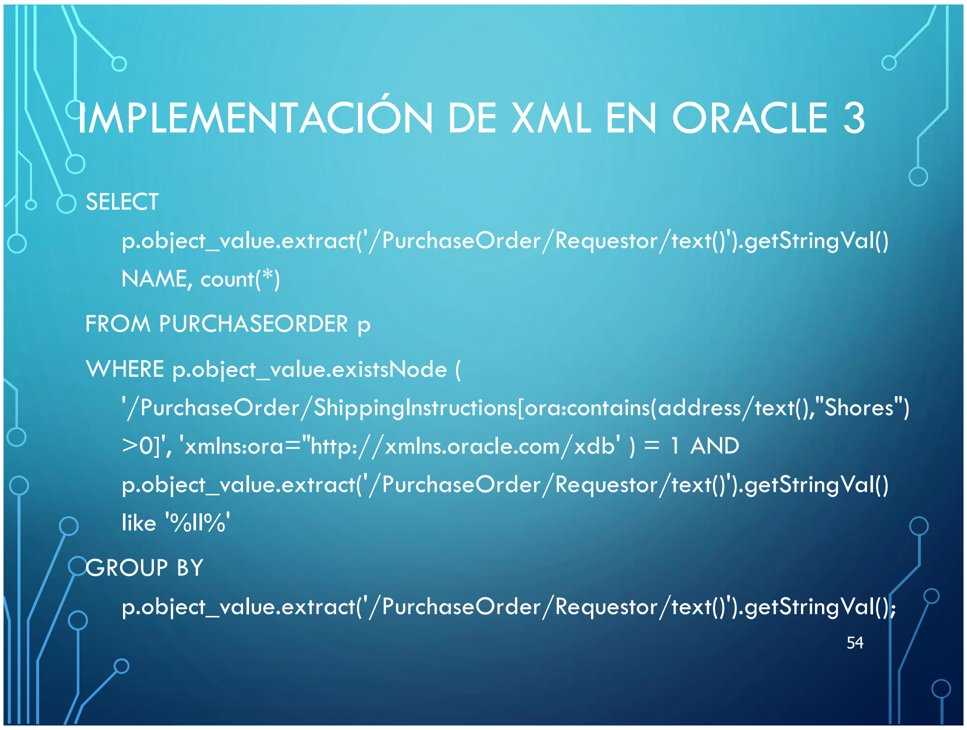


IMPLEMENTACIÓN DE XML EN ORACLE 2

```
SELECT extractValue(object_value,'/PurchaseOrder/Reference')
  "REFERENCE"
FROM PURCHASEORDER
WHERE
  existsNode(object_value,'/PurchaseOrder[SpecialInstructions="Expidite
  "]') = 1;
```

53

53

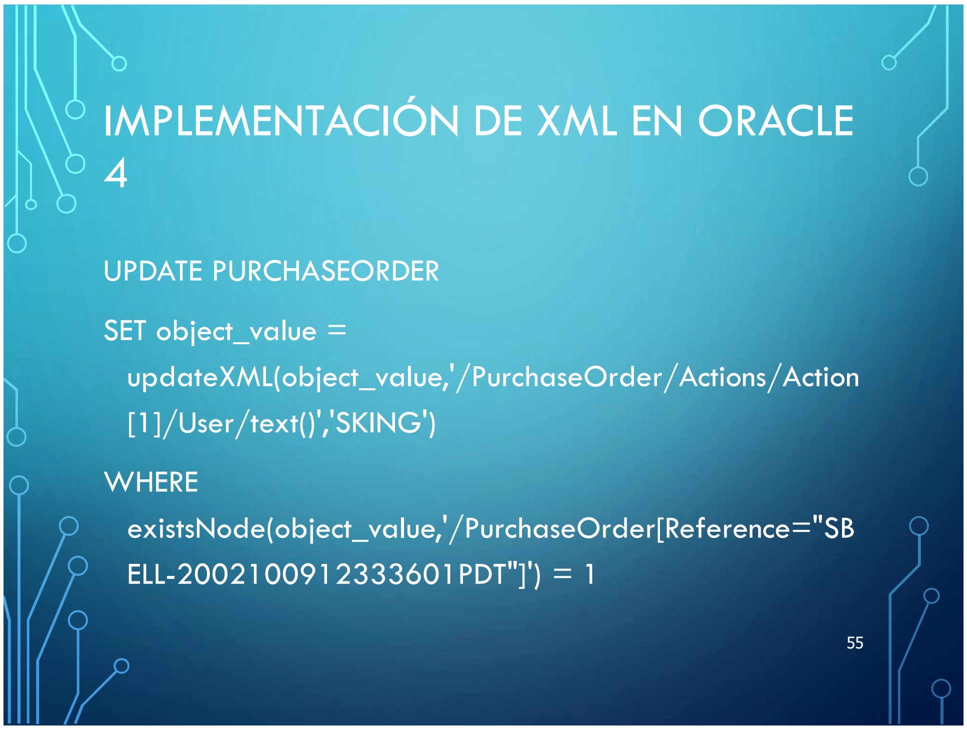


IMPLEMENTACIÓN DE XML EN ORACLE 3

```
SELECT
    p.object_value.extract('/PurchaseOrder/Requestor/text()').getStringVal()
    NAME, count(*)
FROM PURCHASEORDER p
WHERE p.object_value.existsNode (
    '/PurchaseOrder/ShippingInstructions[ora:contains(address/text(),"Shores")
    >0]', 'xmlns:ora="http://xmlns.oracle.com/xdb' ) = 1 AND
    p.object_value.extract('/PurchaseOrder/Requestor/text()').getStringVal()
    like '%ll%'
GROUP BY
    p.object_value.extract('/PurchaseOrder/Requestor/text()').getStringVal();
```

54

54

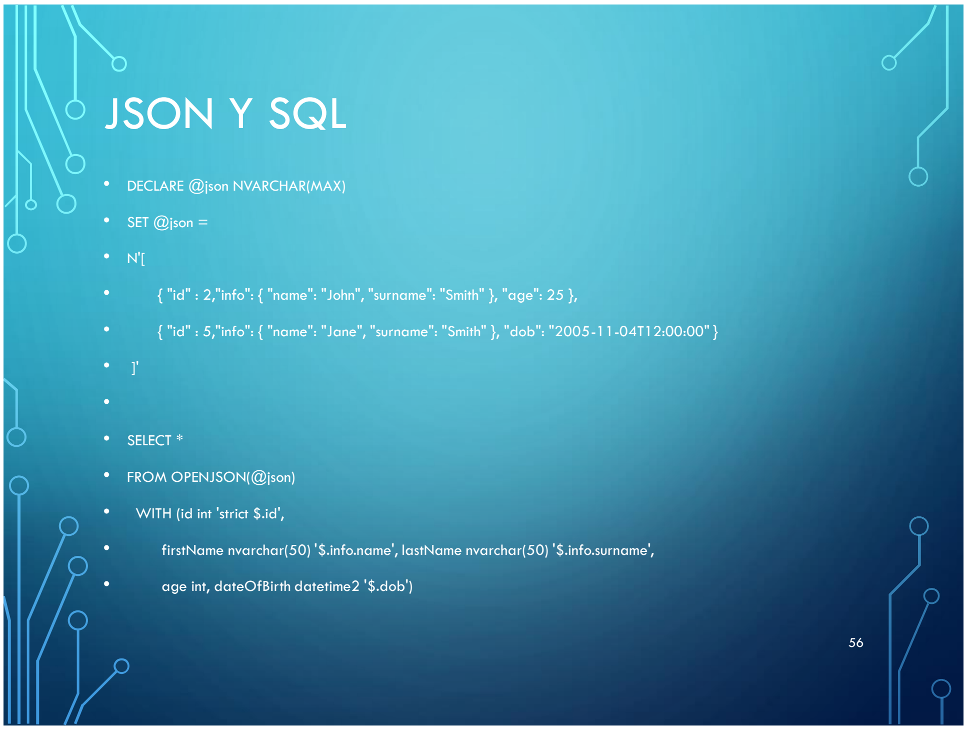


IMPLEMENTACIÓN DE XML EN ORACLE 4

```
UPDATE PURCHASEORDER
SET object_value =
  updateXML(object_value,'/PurchaseOrder/Actions/Action
  [1]/User/text()','SKING')
WHERE
  existsNode(object_value,'/PurchaseOrder[Reference="SB
  ELL-2002100912333601PDT"]') = 1
```

55

55

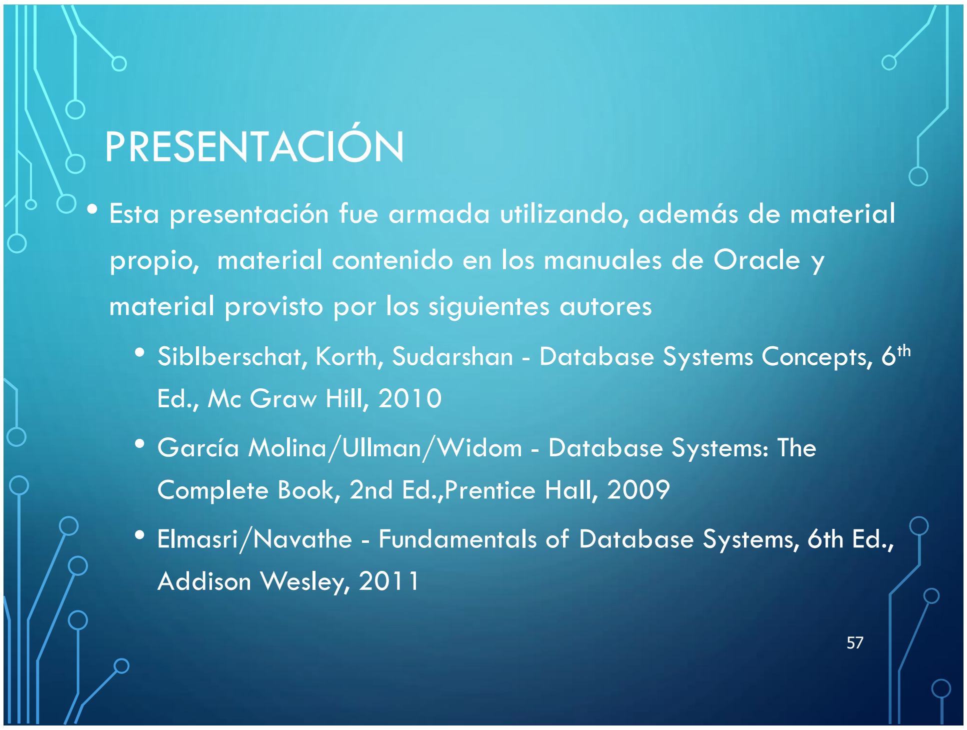


JSON Y SQL

```
• DECLARE @json NVARCHAR(MAX)
• SET @json =
• N'[
•       { "id" : 2,"info": { "name": "John", "surname": "Smith" }, "age": 25 },
•       { "id" : 5,"info": { "name": "Jane", "surname": "Smith" }, "dob": "2005-11-04T12:00:00" }
•  ]'
•
• SELECT *
• FROM OPENJSON(@json)
•   WITH (id int 'strict $.id',
•        firstName nvarchar(50) '$.info.name', lastName nvarchar(50) '$.info.surname',
•        age int, dateOfBirth datetime2 '$.dob')
```

56

56



PRESENTACIÓN

- Esta presentación fue armada utilizando, además de material propio, material contenido en los manuales de Oracle y material provisto por los siguientes autores
  - Siblberschat, Korth, Sudarshan - Database Systems Concepts, 6th Ed., Mc Graw Hill, 2010
  - García Molina/Ullman/Widom - Database Systems: The Complete Book, 2nd Ed.,Prentice Hall, 2009
  - Elmasri/Navathe - Fundamentals of Database Systems, 6th Ed., Addison Wesley, 2011

57

57