Database Systems

XML Databases

Last Lecture

- Query Optimization
- Any questions?

Outline

- Introduction to XML
  - XML Document, XML Schema
- XML data storage
  - Native XML Databases
    - XML data type
  - XML enable Databases
- XML data retrieve
  - XPath
  - XQuery
    - FLWOR

XML

- XML = eXtensible Markup Language.
- While HTML uses tags for *formatting* (e.g., “italic”), XML uses tags for *semantics* (e.g., “this is an address”).
- Key idea: create tag sets for a domain, and translate all data into properly tagged XML documents.

Example: XML Document

```
…
<catalog>
  <product dept="WMN">
    <number>557</number>
    <name language="en">Fleece Pullover</name>
    <colorChoices>navy black</colorChoices>
  </product>
  <product dept="ACC">
…
```


Well-Formed XML

- Start the document with a *declaration*, surrounded by <?xml … ?> .
- Normal declaration is:

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

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

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO> .
- Tags may be nested arbitrarily.
- XML tags are case sensitive.

7

7

Example: Well-Formed XML

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

A NAME subobject

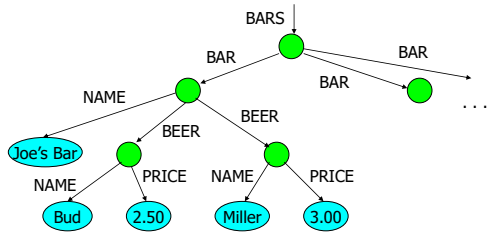
A BEER subobject

8

8

Example

- The <BARS> XML document is:

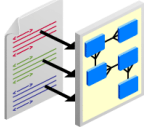


9

9

XML schema language

- Describe the structure and data content of an XML document
  - The description of an element consists of its name (tag), and a parenthesized description of any nested tags.
    - Includes order of sub tags and their multiplicity.
    - etc…
- Can be used to validate XML documents
- Type of schema languages
  - DTD (Document type Definition)
  - XML Schema
  - *RELAX NG*



10

10

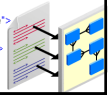
Example: XML Schema and XML Document

```
<?xml version="1.0"?>
<People  xmlns="http://www.sliit.lk"
         xmlns:xsi="http://www.w3.org/2001/XMLSch
         xsi:schemaLocation="http://www.sliit.lk peo
  <Student>
    <Pid>p123</Pid>
    <Name>Kamal</Name>
    <Sex>Male</Sex>
    <StudentNo>MSC/CS/2004/001</StudentNo>
  </Student>
  <Student>
    <Pid>p124</Pid>
    <Name>Aamal</Name>
    <Sex>Male</Sex>
    <StudentNo>MSC/CS/2004/002</StudentNo>
  </Student>
  <Employee>
    <Pid>p123</Pid>
```

Xml doc

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xm
http://www.sliit.lk" elementFormDefault="qualified">
  <xs:complexType name="PersonType" abstract="true">
    <xs:sequence>
      <xs:element name="Pid" type="xs:string"/>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Sex" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="StudentType">
    <xs:complexContent>
      <xs:extension base="PersonType">
        <xs:sequence>
          <xs:element name="StudentNo" type="xs:string"/>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
  <xs:complexType name="EmployeeType">
    <xs:complexContent>
      <xs:extension base="PersonType">
        <xs:sequence>
```

XML Schema



11

11

XML Data Storage

- XML is de-facto standard for exchanging data
- Storing and accessing large XML data stores is gaining in importance

12

12

The main approaches…

- Native XML databases (Eg: Timber, Tamino, Xindice)
  - stores and retrieves XML data in its native form
  - contain new techniques for storage and retrieval
- XML-enabled databases (Eg: Oracle, MS SQL Server,DB2)
  - Use existing database technology to store XML data
  - Database technology has matured over the last three decades
  - **Advantage:** Use more powerful existing database technologies
- Both approaches have merit!
  - Schema is static (structured) → XML enabled DB approach is advantageous
  - Schema is not static (unstructured/semi-structured) → Native XML DB approach is advantageous

13

XML enabled approach

XML storage options

- Mapping between XML and existing data models supported by DBMSs
- Large object storage (CLOB, BLOB)

14

Native storage as XML data type

- Native storage as XML data type
  - XMLType data type - Oracle
    R. Murthy and S. Banerjee, "*XML Schemas in Oracle XML DB*", *VLDB*, 2003.
  - Xml Type - SQL Server 2005
    S. Pal, I. Cseri, O. Seeliger, G. Schaller, L. Giakoumakis and V. Zolotov, "*Indexing XML Data Stored in a Relational Database*", *VLDB*, 2004.
- Internal representation of stored data is preserve the XML content of the data, such as containment hierarchy, document order, element and attribute values, and so on
- Difference between Native XML databases and XML type are blurred

15

Untyped and typed XML data type

- XML values can be stored natively in an XML data type column, which can be typed according to a collection of XML schemas, or can be left untyped
- Use untyped XML data type under the following conditions:
  - You do not have a schema for your XML data
  - You have schemas but you do not want the server to validate the data
- Use typed XML data type under the following conditions:
  - You have schemas for your XML data and you want the server to validate your XML data according on the XML schemas
  - You want to take advantage of storage and query optimizations based on type information

16

Untyped and typed XML data type

For MS SQL Server,

Untyped XML Column in Table:

```
CREATE TABLE AdminDocs (
    id int  primary key,
    xDoc XML not null
    )
```

Typed XML Column in Table:

```
CREATE TABLE AdminDocs (
    id int primary key,
    xDoc XML (CONTENT myCollection)
    )
```

Schema collection

17

Untyped and typed XML data type

Inserting Data: Example

```
INSERT INTO AdminDocs VALUES (1,
'<book >
  <title>Writing Secure Code</title>
  <author>
    <first-name>Michael</first-name>
    <last-name>Howard</last-name>
  </author>
  <author>
    <first-name>David</first-name>
    <last-name>LeBlanc</last-name>
  </author>
  <price>39.99</price>
</book>')
```


18

How to Query XML Data?

- Main two ways to extract data
- Path expressions
  - great if you just want to copy certain elements and attributes as is
- XQuery expressions
  - XQuery extends XPath to a query language that has power similar to SQL.

19

19

Path Expressions

- *XPath* is a language for describing paths in XML documents.
  - i.e. Xpath 1.0 and Xpath 2.0
- Really think of the semistructured data graph and *its* paths.

20

20

Path Descriptors

- Simple path descriptors are sequences of tags separated by slashes (/).
- If the descriptor begins with /, then the path starts at the root and has those tags, in order.
- If the descriptor begins with //, then the path can start anywhere.

21

21

Value of a Path Descriptor

- Each path descriptor, applied to a document, has a value that is a sequence of elements.
- An *element* is an atomic value or a node.
- A *node* is matching tags and everything in between.
  - i.e., a node of the semistructured graph.

22

22

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>
```

/BARS/BAR/PRICE describes the set with these two PRICE elements as well as the PRICE elements for any other bars.

23

23

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>
```

//PRICE describes the PRICE Elements to appear within the document.

24

24

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/*

<BARS>
<BAR name = “JoesBar”>
<PRICE theBeer = “Bud”>2.50</PRICE>
<PRICE theBeer = “Miller”>3.00</PRICE>
</BAR> …
<BEER name = “Bud” soldBy = “JoesBar SuesBar …”/> …
</BARS>

/BARS/* captures all BAR and BEER elements, such as these.

Attributes

- In XPath, we refer to attributes by prepending @ to their name.
- Attributes of a tag may appear in paths as if they were nested within that tag.

Example: /BARS/*/@name

<BARS>
<BAR name = “JoesBar”>
<PRICE theBeer = “Bud”>2.50</PRICE>
<PRICE theBeer = “Miller”>3.00</PRICE>
</BAR> …
<BEER name = “Bud” soldBy = “JoesBar SuesBar …”/> …
</BARS>

/BARS/*/@name selects all name attributes of immediate subelements of the BARS element.

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*.

Axes

/child::BAR[@name=“JoesBar”]

Axis | Node test | Predicate

1. The axis (Optional)
   - Direction to navigate
2. The node test
   - The node of interest by name
3. Predicate
   - The criteria used to filter nodes

Example: Axes

- /BARS/BEER is really shorter way for /BARS/child::BEER .
- @ is really shorthand for the attribute:: axis.
  - Thus, /BARS/BEER[@name = “Bud”] is shorthand for
    /BARS/BEER[attribute::name = “Bud”]

31

31

More Axes

- Some other useful axes are:
  - parent:: = parent(s) of the current node(s).
  - descendant-or-self:: = the current node(s) and all descendants.
    - Note: // is really shorthand for this axis.
  - ancestor::, ancestor-or-self, etc.
  - the default axis is child:: --- go to all the children of the current set of nodes.

32

32

Predicates

- 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.

33

33

Example: Selection Condition

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

<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 satisfy the path descriptor.

34

34

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.

35

35

```
<catalog>
  <product dept="WMN">
    <number>557</number>
    <name language="en">Linen Shirt</name>
    <colorChoices>beige sage</colorChoices>
  </product>
  <product dept="ACC">
    <number>563</number>
    <name language="en">Ten-Gallon Hat</name>
  </product>
  <product dept="ACC">
    <number>443</number>
    <name language="en">Golf Umbrella</name>
  </product>
  <product dept="MEN">
    <number>784</number>
    <name language="en">Rugby Shirt</name>
    <colorChoices>blue/white blue/red</colorChoices>
    <desc>Our <i>best-selling</i> shirt!</desc>
  </product>
</catalog>
```

36

Example: Predicates and Returned Elements

- Using number in a predicate does not mean that number elements are returned:

All product elements whose number child is less than 500:

```
product[number < 500]
```

All number elements whose value is less than 500:

```
product/number[. lt 500]
```

A period (".") is used to indicate the context item itself

37

37

Using Position in Predicates

- Can use a number in the predicate to indicate the position of the child

The 4th product child of catalog:

```
catalog/product[4]
```

The 4th child of catalog (regardless of its name):

```
catalog/*[4]
```

38

38

Using Position in Predicates

- More than one predicate can be used to specify multiple constraints in a step
  - evaluated left to right

Products whose number is less than 500 *and* whose department is ACC:

```
product[number < 500][@dept = "ACC"]
```

Of the products whose department is 'ACC', select the 2nd one:

```
product[@dept = "ACC"][2]
```

Take the 2nd product, if it's in the ACC department, select it:

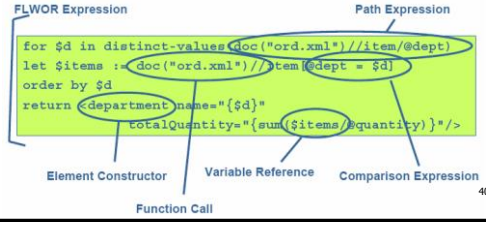
```
product[2][@dept = "ACC"]
```

9

39

XQuery

- XQuery extends XPath to a query language that has power similar to SQL.
- Basic building blocks,



40

40

```
<order num="00299432" date="2004-09-15" cust="0221A">
  <item dept="WMN" num="557" quantity="1" color="beige"/>
  <item dept="ACC" num="563" quantity="1"/>
  <item dept="ACC" num="443" quantity="2"/>
  <item dept="MEN" num="784" quantity="1" color="blue/white"/>
  <item dept="MEN" num="784" quantity="1" color="blue/red"/>
  <item dept="WMN" num="557" quantity="1" color="sage"/>
</order>
```

41

41

XQuery

- **Variables** are identified by a name preceded by a $

```
for $prod in
  (doc("cat.xml")//product)
return $prod/number
```

- Literal values can be expressed as:
  - strings (in single or double quotes)

```
doc("cat.xml")//product/@dept = "WMN"
```

  - numbers

```
doc("ord.xml")//item/@quantity > 1
```

42

42

Comments

- XQuery comments
  - Delimited by `(:` and `:)`
  - Anywhere insignificant whitespace is allowed
  - Do not appear in the results

```
(: This query... :)
```

- XML comments
  - May appear in the results
  - XML-like syntax `<!-- This element... -->`

Clauses of a FLWOR Expression

- `for` clause
  - iteratively binds the `$prod` variable to each item returned by a path expression.
- `let` clause
  - binds the `$prodDept` variable to the value of the `dept` attribute
- `where` clause
  - selects nodes whose `dept` attribute is equal to "`WMN`" or "`ACC`"
- `return` clause
  - returns the `name` children of the selected nodes

```
for $prod in doc("cat.xml")//product
let $prodDept := $prod/@dept
where $prodDept = "ACC" or $prodDept = "WMN"
return $prod/name
```


for Clauses

- Iteratively binds the variable to each item returned by the `in` expression
- The rest of the expression is evaluated once for each item returned
- Multiple `for` clauses are allowed in the same FLWOR

expression after `in` can evaluate to any sequence

```
for $prod in doc("cat.xml")//product
```


Range expressions

- Create sequences of consecutive integers
- Use `to` keyword
  - `(1 to 5)` evaluates to a sequence of 1, 2, 3, 4 and 5
- Useful in `for` clauses to iterate a specific number of times

```
for $i in (1 to 5)
...
```

```
for $i in (1 to $prodCount)
...
```


Multiple Variables in a *for* clause

- Use commas to separate multiple `in` expressions
  - or multiple `for` clauses
- Return clause evaluated for every combination of variable values

```
for $i in (1, 2), $j in (11, 12)
return <eval>i is {$i} and j is {$j}</eval>
```

```
<eval>i is 1 and j is 11</eval>
<eval>i is 1 and j is 12</eval>
<eval>i is 2 and j is 11</eval>
<eval>i is 2 and j is 12</eval>
```


Positional variables in *for* clause

- Positional variable keeps track of the iteration number
- Use `at` keyword

```
for $prod at $i in doc("cat.xml")//
  product[@dept = "ACC" or @dept = "WMN"]
return <eval>{$i}. {data($prod/name)}</eval>
```

```
<eval>1. Linen Shirt</eval>
<eval>2. Ten-Gallon Hat</eval>
<eval>3. Golf Umbrella</eval>
```

let clauses

- Convenient way to bind a variable
  - avoids repeating the same expression many times
- Does not result in iteration

```
for $i in (1 to 3)
return <eval>{$i}</eval>
```

```
<eval>1</eval>
<eval>2</eval>
<eval>3</eval>
```

```
let $i := (1 to 3)
return <eval>{$i}</eval>
```

```
<eval>1 2 3</eval>
```

49

49

Multiple *for* and *let* clauses

- `for` and `let` can be repeated and combined

```
let $catDoc := doc("cat.xml")
for $prod in $catDoc//product
let $prodDept := $prod/@dept
where $prodDept = "ACC" or $prodDept = "WMN"
return $prod/name
```

50

50

where clause

- Used to filter results
- Can contain many subexpressions
- Evaluates to a boolean value
  - effective boolean value is used
- If `true`, return clause is evaluated

```
where $prod/number > 100
  and starts-with($prod/name, "L")
  and exists($prod/colorChoices)
  and ($prodDept="ACC" or $prodDept="WMN")
```

51

51

return clause

- The value that is to be returned

```
for $prod in doc("cat.xml")//product
return $prod/name
```

- Single expression only
  - can combine multiple expressions into a sequence

```
return <a>{$i}</a>
       <b>{$j}</b>
```

```
return (<a>{$i}</a>,
        <b>{$i}</b>)
```

52

52

order by Clause

- Only way to sort results in XQuery
- Use `order by` before `return` clause
- Order by
  - atomic values, or
  - nodes that contain individual atomic values
- Can specify multiple values to sort on

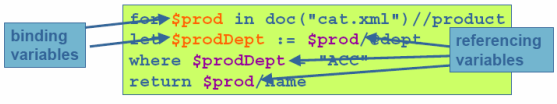
```
for $item in doc("ord.xml")//item
order by $item/@dept, $item/@num
return $item
```

53

53

Variable binding and referencing

- Variables are *bound* in the let/for clauses
- Once bound, variables can be *referenced* anywhere in the FLWOR



- Values cannot be changed once bound
  - e.g. no `let $count := $count + 1`

54

54

Comparisons

- Two kinds:
  - *value* comparisons
    - `eq, ne, lt, le, gt, ge`
    - used to compare individual values
    - each operand must be a single atomic value or a node containing a single atomic value
  - *general* comparisons
    - `=, !=, <, <=, >, >=`
    - can be used with sequences of multiple items

55

55

Comparisons

General Comparisons

Example	Value
doc("catalog.xml")/catalog/product[2]/name = 'Floppy Sun Hat'	true
doc("catalog.xml")/catalog/product[4]/number < 500	false
1 > 2	false
() = (1, 2)	false
(2, 5) > (1, 3)	true

56

56

Value vs. General Comparisons

```
doc("ord.xml")//item/@quantity > 1
```

- returns true if any `quantity` attributes have values greater than 1

```
doc("ord.xml")//item/@quantity gt 1
```

- returns true if there is only *one* `quantity` attribute returned by the expression, and its value is greater than 1
- if more than one `quantity` is returned, an error is raised

57

57

Conditional expressions

- if-then-else syntax

```
for $prod in (doc("cat.xml")/catalog/product)
return if ($prod/@dept = 'ACC')
       then <acc>{data($prod/number)}</acc>
       else <other>{data($prod/number)}</other>
```

- parentheses around `if` expression are required
- else is always required
  - but it can be just `else ()`

58

58

Effective boolean value

- "`if`" expression must be boolean
  - if it is not, its *effective boolean value* is found
- effective boolean value is false for:
  - the `xs:boolean` value `false`
  - the number `0` or `NaN`
  - a zero-length string
  - the empty sequence
- it's an error for a sequence of many atomic values

59

59

Nesting conditional expressions

- Conditional expressions can be nested
  - provides "else if" functionality

```
if ($prod/@dept = 'ACC')
then <accessory>{data($prod/number)}</accessory>
else if ($prod/@dept = 'WMN')
     then <womens>{data($prod/number)}</womens>
     else if ($prod/@dept = 'MEN')
          then <mens>{data($prod/number)}</mens>
          else <other>{data($prod/number)}</other>
```

60

60

Functions

- Built-in functions
  - over 100 functions built into XQuery
  - names do not need to be prefixed when called
- User-defined functions
  - defined in the query or in a separate module
  - names must be prefixed

61

61

Built in Functions: A Sample

- String-related
  - `substring, contains, matches, concat, normalize-space, tokenize`
- Date-related
  - `current-date, month-from-date, adjust-time-to-timezone`
- Number-related
  - `round, avg, sum, ceiling`
- Sequence-related
  - `index-of, insert-before, reverse, subsequence, distinct-values`

62

62

More built in functions

- Node-related
  - `data, empty, exists, id, idref`
- Name-related
  - `local-name, in-scope-prefixes, QName, resolve-QName`
- Error handling and trapping
  - `error, trace, exactly-one`
- Document- and URI-related
  - `collection, doc, root, base-uri`

63

63

MS SQL Server: Methods on XML Data Type

- **query() -** fragments of an XML document can be extracted using the query() method of XML data type. The query() method accepts an XQuery expression as an argument and returns an untyped XML instance.
- Eg.

```
SELECT xDoc.query('/doc[@id = 123]//section')
FROM   AdminDocs
```

64

64

MS SQL Server: Methods on XML Data Type

- **value()** - Scalar values can be extracted from an XML instance using the value() method by specifying an XQuery expression and the desired SQL type to be returned.

Eg.

```
SELECT xDoc.value(
  'data((/doc//section[@num = 3]/title)[1])', 'nvarchar(max)')
FROM   AdminDocs
```

65

65

MS SQL Server : Methods on XML Data Type

- **exist()** - method is useful for existential checks on an XML instance. It returns 1 if the XQuery expression evaluates to non-null node list; otherwise it returns 0.

```
SELECT xDoc.query('/doc[@id = 123]//section')
FROM   AdminDocs
WHERE  xDoc.exist ('/doc[@id = 123]') = 1
```

66

66

MS SQL Server: Methods on XML Data Type

- **modify()** - Data manipulation operations can be performed on an XML instance using the modify() method. Support for XML DML is provided through insert, delete, and update keywords added to XQuery. One or more nodes can be inserted, deleted, and updated using the insert, delete, and update keywords, respectively.
- Eg.

```
UPDATE AdminDocs SET xDoc.modify('
 insert
  <section num="2">
      <title>Background</title>
  </section>
 after (/doc//section[@num=1])[1]')
```


67