XML

Structure of XML Data

XML Document Schema

Querying and Transformation

Application Program Interfaces to XML

Storage of XML Data

XML Applications

Introduction

XML: Extensible Markup Language

Defined by the WWW Consortium (W3C)

Documents have tags(标签) 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

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.

```
<bank>
    <account>
        <account_number> A-101     </account_number>
        <branch_name>      Downtown </branch_name>
        <balance>            500        </balance>
    </account>
    <depositor>
        <account_number> A-101    </account_number>
        <customer_name> Johnson </customer_name>
    </depositor>
</bank>
```

XML: Motivation

Data interchange is critical in today's networked world

Examples:

Banking: funds transfer

Order processing (especially pany 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 e the basis for all new generation data interchange formats

XML Motivation (Cont.)

Each XML based standard defines what are valid elements, using

XML type specification languages to specify the syntax

- DTD (Document Type Descriptors)
- XML Schema

Plus textual descriptions of the semantics

XML allows new tags to be defined as required

However, this may be constrained by DTDs

A wide variety of tools is available for parsing, browsing and querying XML documents/data

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

Structure of XML Data

Tag(标签): 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

- <account> … <balance> …. </balance> </account>

Improper nesting

- <account> … <balance> …. </account> </balance>

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

Example of Nested Elements

```
<bank-1>
    <customer>
        <customer_name> Hayes </customer_name>
        <customer_street> Main </customer_street>
        <customer_city>     Harrison </customer_city>
        <account>
            <account_number> A-102 </account_number>
            <branch_name>      Perryridge </branch_name>
            <balance>          400 </balance>
        </account>
        <account>
            …
        </account>
    </customer>
      .
      .
</bank-1>
```

Motivation for Nesting

Nesting of data is useful in data transfer

- Example: elements representing *customer_id, customer_name*, and address nested within an *order* 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

Structure of XML Data (Cont.)

Mixture of text with sub-elements is legal in XML.

Example:

```
<account>
    This account is seldom used any more.
    <account_number> A-102</account_number>
    <branch_name> Perryridge</branch_name>
    <balance>400 </balance>
</account>
```

Useful for document markup, but discouraged for data representation

Attributes

Elements can have **attributes**

```
<account acct-type = “checking” >
    <account_number> A-102 </account_number>
    <branch_name> Perryridge </branch_name>
    <balance> 400 </balance>
</account>
```

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

```
<account  acct-type = “checking”  monthly-fee=“5”>
```

Attributes vs. Subelements

Distinction between subelement and attribute

In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents

In the context of data representation, the difference is unclear and may be confusing

- Same information can be represented in two ways
  - <account account_number = “A-101”> …. </account>
  - <account>
    <account_number>A-101</account_number> …
    </account>

Suggestion: use attributes for identifiers of elements, and use subelements for contents

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

```
<bank Xmlns:FB=‘ ’>
 …
    <FB:branch>
            <FB:branchname>Downtown</FB:branchname>
    <FB:branchcity>  Brooklyn  </FB:branchcity>
    </FB:branch>
…
    </bank>
```

More on XML Syntax

Elements without subelements or text content can be abbreviated by ending the start tag with a /> and deleting the end tag

```
<account  number="A-101" branch="Perryridge"  balance="200 />
```

To store string data that may contain tags, without the tags being interpreted as subelements, use CDATA as below

```
<![CDATA[<account> … </account>]]>
```

Here, <account> and </account> are treated as just strings

CDATA stands for "character data"

XML Document Schema

Database schemas constrain what information can be stored, and the data types of stored values

XML documents are not required to have an associated schema

However, schemas are very important for XML data exchange

Otherwise, a site cannot automatically interpret data received from another site

Two mechanisms for specifying XML schema

Document Type Definition (DTD)

- Widely used

XML Schema

- Newer, increasing use

Document Type Definition (DTD)

The type of an XML document can be specified using a DTD

DTD constraints structure of XML data

- What elements can occur
- What attributes can/must an element have
- What subelements can/must occur inside each element, and how many times.

DTD does not constrain data types

- All values represented as strings in XML

DTD syntax

- <!ELEMENT element (subelements-specification) >
- <!ATTLIST element (attributes) >

Element Specification in DTD

Subelements specification

name of element

type of element

- #PCDATA (parsed character data), i.e., character strings
- EMPTY (no subelements)
- ANY (anything can be a subelement)

Example

```
<! ELEMENT depositor (customer_name  account_number)>
<! ELEMENT customer_name (#PCDATA)>
<! ELEMENT account_number (#PCDATA)>
```

Subelement specification may have regular expressions

```
<!ELEMENT bank ( ( account | customer | depositor)+)>
```

- Notation:
  - “|” - alternatives
  - “+” - 1 or more occurrences
  - “*” - 0 or more occurrences

Bank DTD

```
<!DOCTYPE bank [
    <!ELEMENT bank ( ( account | customer | depositor)+)>
    <!ELEMENT account (account_number branch_name balance)>
    <! ELEMENT customer(customer_name customer_street
                                                customer_city)>
    <! ELEMENT depositor (customer_name account_number)>
    <! ELEMENT account_number (#PCDATA)>
    <! ELEMENT branch_name (#PCDATA)>
    <! ELEMENT balance(#PCDATA)>
    <! ELEMENT customer_name(#PCDATA)>
    <! ELEMENT customer_street(#PCDATA)>
    <! ELEMENT customer_city(#PCDATA)>
]>
```

Attribute Specification in DTD

Attribute specification : for each attribute

name

type declaration of attribute

- CDATA
- ID (identifier)
- IDREF (ID reference)
- IDREFS (multiple IDREFs)

default Declaration of the attribute

- mandatory (#REQUIRED)
- has a default value (value),
- or neither (#IMPLIED)

Examples

<!ATTLIST account acct-type CDATA “checking”>

<!ATTLIST customer
customer_id ID # REQUIRED
accounts IDREFS # REQUIRED >

IDs and IDREFs

An element can have at most one attribute of type ID

The ID attribute value of each element in an XML document must be distinct

Thus the ID attribute value is an object identifier

An attribute of type IDREF must contain the ID value of an element in the same document

An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document

Bank DTD with Attributes

Bank DTD with ID and IDREF attribute types.

```
<!DOCTYPE bank-2[
    <!ELEMENT account (branch, balance)>
    <!ATTLIST account
              account_number ID          # REQUIRED
              owners              IDREFS # REQUIRED>
     <!ELEMENT customer(customer_name, customer_street,
                                                customer_city)>
     <!ATTLIST customer
              customer_id       ID         # REQUIRED
              accounts          IDREFS # REQUIRED>

     … declarations for branch, balance, customer_name,
                          customer_street and customer_city
]>
```

XML data with ID and IDREF attributes

```
<bank-2>
    <account account_number="A-401" owners="C100 C102">
        <branch_name> Downtown </branch_name>
        <balance>        500 </balance>
    </account>
    <customer customer_id="C100" accounts="A-401">
        <customer_name>Joe        </customer_name>
        <customer_street> Monroe   </customer_street>
        <customer_city>     Madison</customer_city>
    </customer>
    <customer customer_id="C102" accounts="A-401 A-402">
        <customer_name> Mary     </customer_name>
        <customer_street> Erin       </customer_street>
        <customer_city>     Newark </customer_city>
    </customer>
</bank-2>
```

Limitations of DTDs

No typing of text elements and attributes

All values are strings, no integers, reals, etc.

Difficult to specify unordered sets of subelements

Order is usually irrelevant in databases (unlike in the document-layout environment from which XML evolved)

(A | B)* allows specification of an unordered set, but

- Cannot ensure that each of A and B occurs only once

IDs and IDREFs are untyped

The *owners* attribute of an account may contain a reference to another account, which is meaningless

- *owners* attribute should ideally be constrained to refer to customer elements

XML Schema

XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs. Supports

Typing of values

- E.g. integer, string, etc
- Also, constraints on min/max values

User-defined, comlex 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.

XML Schema Version of Bank DTD

```
<xs:schema  /2001/XMLSchema>
<xs:element name="bank" type="BankType"/>
<xs:element name="account">
    < plexType>                plexType, simpleType
        <xs:sequence>                  // order indicator: sequence, all, choice
            <xs:element name="account_number" type="xs:string"/>
            <xs:element name="branch_name"      type="xs:string"/>
            <xs:element name="balance"              type="xs:decimal"/>
        </xs:squence>
    < plexType>
</xs:element>
….. definitions of customer and depositor ….
< plexType name="BankType">
    <xs:squence>
<xs:element ref="account"    minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="customer" minOccurs="0" maxOccurs="unbounded"/>
<xs:element ref="depositor" minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
< plexType>
</xs:schema>
```

XML Schema Version of Bank DTD

Choice of “xs:” was ours -- any other namespace prefix could be chosen

Element “bank” has type “BankType”, which is defined separately

plexType is used later to create the named complex type “BankType”

Element “account” has its type defined in-line

XML Schema - complexType

complexType: either type that has either attributes or nested subelements must be specified to be a complex type.

order indicator: sequence, all, choice

XML Schema - simpleType

Simple types:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

```
<xs:element name="color“  type="xs:string" default ="red"/>
```

XML Schema - simpleType

Restriction of simple types:

“age” type restriction : age is between 0 and 120

```
<xs:element name="age">
    <xs:simpleType>
        <xs:restriction base="xs:integer">
            <xs:minInclusive value="0"/>
            <xs:maxInclusive value="120"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schema - simpleType

Restriction of simple types:

“car” type restriction

```
<xs:element name="car">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:enumeration value="Audi"/>
            <xs:enumeration value="Golf"/>
            <xs:enumeration value="BMW"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

XML Schema - simpleType

Restriction of simple types:

“initial” type restriction

```
<xs:element name="initials">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:pattern value="[A-Z][A-Z][A-Z]"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Restriction on simple types

限定	描述
enumeration	定义可接受值的一个列表
fractionDigits	定义所允许的最大的小数位数。必须大于等于0。
length	定义所允许的字符或者列表项目的精确数目。必须大于或等于0。
maxExclusive	定义数值的上限。所允许的值必须小于此值。
maxInclusive	定义数值的上限。所允许的值必须小于或等于此值。
maxLength	定义所允许的字符或者列表项目的最大数目。必须大于或等于0。
minExclusive	定义数值的下限。所允许的值必需大于此值。
minInclusive	定义数值的下限。所允许的值必需大于或等于此值。
minLength	定义所允许的字符或者列表项目的最小数目。必须大于或等于0。
pattern	定义可接受的字符的精确序列。
totalDigits	定义所允许的阿拉伯数字的精确位数。必须大于0。
whiteSpace	定义空白字符（换行、回车、空格以及制表符）的处理方式。

More features of XML Schema

Attributes specified by xs:attribute tag:

<xs:attribute name = “account_number”/>

adding the attribute use = “required” means value must be specified. Other wise the value is “optional”

Key constraint: “account numbers form a key for account elements under the root bank element:

```
<xs:key name = “accountKey”>
        <xs:selector xpath = “/bank/account”/>
        <xs:field xpath = “account_number”/>
<\xs:key>
```

selector is a path expression that defines the scope for the constraints

field declaration specifiey the elements or attributes that form the key.