Intro to DB

# CHAPTER 10
# XML

# Chapter 10: XML

- Structure of XML Data
- XML Document Schema
- Querying and Transformation
- Storage of XML Data
- Application Program Interfaces to XML
- XML Applications

# eXtensible Markup Language

- Defined by the WWW Consortium (W3C)
- Originally intended as a document markup language not a database language
  - Documents have tags giving extra information about sections of the document
    - E.g. <title> XML </title> <slide> Introduction …</slide>
- Derived from SGML (Standard Generalized Markup Language), but simpler to use
  - SGML is too complex => nobody uses it!
- Goal was (is?) to replace HTML as the language for publishing documents on the Web

# HTML vs XML

- *Extensible*
  - unlike HTML
  - users can add new tags, and
  - *separately* specify how the tag should be handled for display
- HTML is not fit for data exchange
  - HTML tags are predefined
  - mainly deal with how to handle the display of texts and images
  - HTML tags do not annotate the meaning of the data
    - Example: <H1> aaa </H1>

      specifies that 'aaa' should appear as header in the display

# HTML vs XML – Example

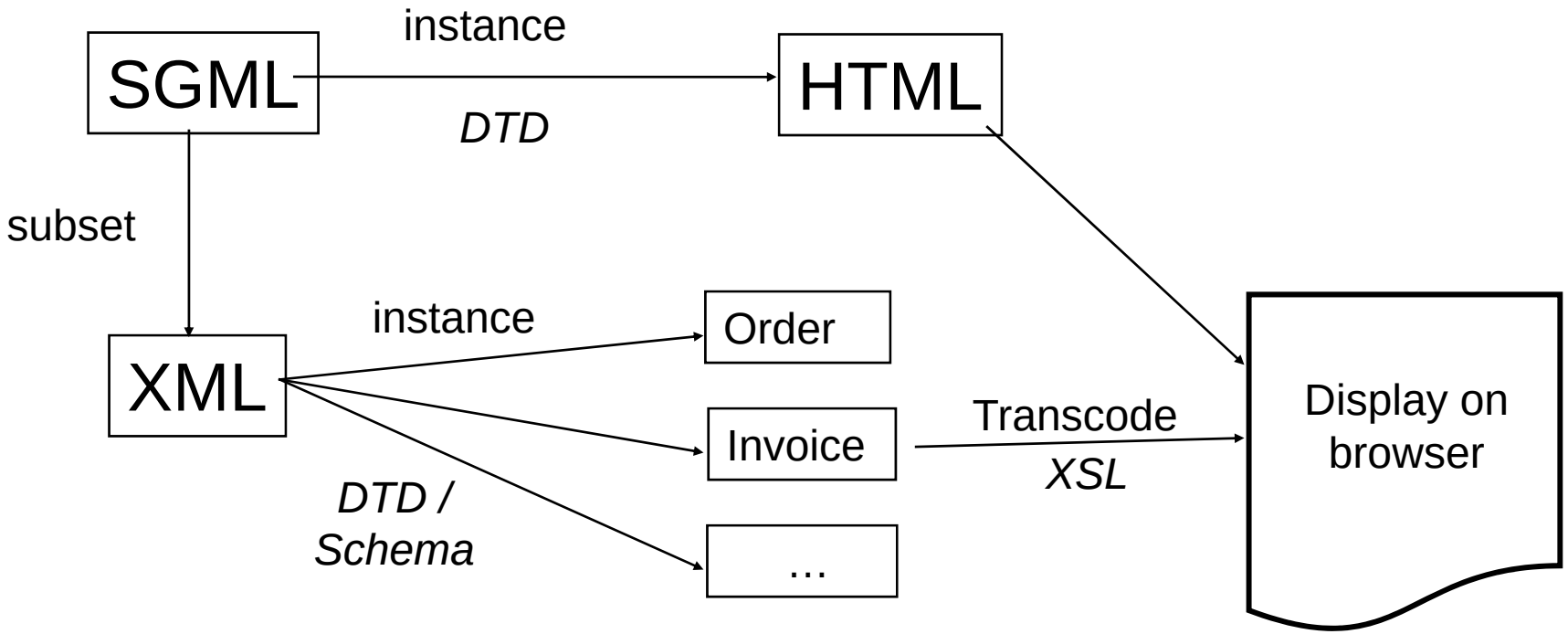
***HTML***

```
<H1>Invoice</H1>
<B>From: PC World </B><P>
To: SGLee <P>
Date: 2001/11/16<P>
Amount: $100<P>
```

***XML***

```
<Invoice>
<From>PC World </From>
<To> SGLee </To>
<Date year="2001" month="11"
    day="16" />
<Amount unit="dollars">
    100</Amount>
</Invoice>
```

# XML, HTML, & SGML



# 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
    - <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

# XML Data – Example

```
<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>
```

# 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

# 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)***
    - Original spec. Simpler but limited
  - ***XML Schema***
    - Newer, more complicated but with more features

# 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) >

# 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)>
]>
```

# Bank Element

```
<bank>
 <account>
    <account_number> A-101 </account_number>
    <branch_name> Downtown </branch_name>
    <balance> 500 </balance>
 </account>
 …
 <customer>
    <customer_name> Johnson </customer_name>
    <customer_street> Alma </customer_street>
    <customer_city> Palo Alto </customer_city>
 </customer>
 …
 <depositor>
    <account_number> A101 </account_number>
    <customer_name> Johnson </customer_name>
 <depositor>
 …
</bank>
```

# 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 types
  - Is itself specified in XML syntax, unlike DTDs
    - More standard representation, but verbose
  - Is integrated with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance ..
- Significantly more complicated than DTDs but with more power

# XML Schema Version of Bank DTD

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
<xsd:element name="bank" type="BankType" />
<xsd:element name="account">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="account-number" type="xsd:string" />
            <xsd:element name="branch-name"    type="xsd:string" />
            <xsd:element name="balance"        type="xsd:decimal" />
        </xsd:squence>
    </xsd:complexType>
</xsd:element>
….. definitions of customer and depositor ….
<xsd:complexType name="BankType">
    <xsd:squence>
        <xsd:element ref="account" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="customer" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="depositor" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

# 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='http://www.FirstBank.com'>
    …
    <FB:branch>
        <FB:branchname>Downtown</FB:branchname>
      <FB:branchcity>    Brooklyn   </FB:branchcity>
    </FB:branch>
    …
</bank>
```

# 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
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XPath
    - Simple language consisting of path expressions
  - XQuery
    - An XML query language with a rich set of features

# Tree Model of XML Data

- Query and transformation languages are based on a **tree model** of XML data
- An XML document is modeled as a tree, with **nodes** corresponding to elements and attributes
  - Element nodes have child nodes, which can be attributes or subelements
  - Text in an element is modeled as a text node child of the element
  - Children of a node are ordered according to their order in the XML document
  - Element and attribute nodes (except for the root node) have a single parent, which is an element node
  - The root node has a single child, which is the root element of the document

# 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.
  - /bank/customer/customer_name

    evaluated on the data shown in figure 10.1 returns

    <customer_name> Johnson </customer_name>
    <customer_name> Hayes </customer_name>
  - /bank/customer/customer_name/text( )

    returns the same names, but without the enclosing tags

# XPath (Cont.)

- The initial "/" denotes root of the document (above the top-level tag)
- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in [ ]
  - E.g.

    /bank/account[balance > 400]

    returns account elements with a balance value greater than 400

    /bank/account[balance]

    returns account elements containing a balance subelement

# XSLT

- A **stylesheet** stores formatting options for a document, usually separately from document
  - E.g. HTML style sheet may specify font colors and sizes for headings, etc.
- The **XML Stylesheet Language (XSL)** was originally designed for generating HTML from XML
- XSLT is a general-purpose transformation language
  - Can translate XML to XML, and XML to HTML
- XSLT transformations are expressed using rules called **templates**
  - Templates combine selection using XPath with construction of results

# XSLT Example

```
<xsl:template match="/bank/customer">
    <name>
    <xsl:value-of select="customer_name"/>
    </name>
</xsl:template>
<xsl:template match="*"/>
```

Result:

```
<name>
Johnson
</name>
<name>
Hayes
</name>
```

# Storage of XML Data

XML data can be stored in

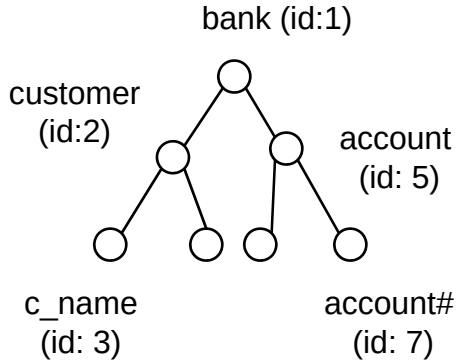
- Non-relational data stores
  - Flat files
    - Natural for storing XML
    - But has all problems discussed in Chap. 1 (no concurrency, no recovery, …)
  - XML database
    - Database built specifically for storing XML data
    - Supporting DOM model and declarative querying
    - Currently no dominant commercial-grade systems
- Relational databases
  - Translated data into relational form
  - Advantage: mature database systems
  - Disadvantages: overhead of translating data and queries

# Storing XML in Relational Databases

**Store as string**

- E.g. store each top level element as a string field of a tuple in a database
  - use a single relation to store all elements, or
  - use a separate relation for each top-level element type
    - E.g. *account*, *customer*, *depositor*, each with a string-valued attribute to store the element
- Benefits:
  - can store any XML data even without DTD
  - as long as there are many top-level elements in a document, strings are small compared to full document, allowing faster access to individual elements.
- Drawbacks:
  - need to parse strings to access values inside the elements
    - parsing is slow

# Storing XML in Relational DB (Cont.)




**Tree representation**

- Model XML data as tree and store using relations
  *nodes(id, type, label, value)*
  *child (child-id, parent-id)*
  - each element/attribute is given a unique identifier
  - ***type*** indicates element/attribute
  - ***label*** specifies the tag name of the element/name of attribute
  - ***value*** is the text value of the element/attribute
  - relation *child* holds the parent-child relationships in the tree
- Benefits:
  - flexible: can store any XML data, even without DTD
- Drawbacks:
  - data is broken up into too many pieces, increasing space overheads
  - even simple queries require a large number of joins, which can be slow

# Storing XML in Relational DB (Cont.)

**Map to relations**

- If DTD of document is known, can map data to relations
  - bottom-level elements and attributes are mapped to attributes of relations
- A relation is created for each element type
  - an *id* attribute to store a unique id for each element
  - all element attributes become relation attributes
  - subelements that can occur multiple times represented in a separate table
- Benefits:
  - efficient storage
  - can translate XML queries into SQL, execute efficiently, and then translate SQL results back to XML
- Drawbacks
  - need to know DTD
  - translation overheads still present

# Application Program Interface

- SAX (Simple API for XML)
  - Based on parser model, user provides event handlers for parsing events
    - E.g. start of element, end of element
  - Not suitable for database applications
- DOM (Document Object Model)
  - XML data is parsed into a tree representation
  - Variety of functions provided for traversing the DOM tree
  - E.g.: Java DOM API provides Node class with methods
    getParentNode( ), getFirstChild( ), getNextSibling( )
    getAttribute( ), getData( ) (for text node)
    getElementsByTagName( ), …
  - Also provides functions for updating DOM tree

# END OF CHAPTER 10