Chapter 10 - XML

Forces Driving XML

- Document Processing
  - Goal: use document in various, evolving systems
  - structure - content - layout
  - grammar: markup vocabulary for mixed content

- Data Bases and Data Exchange
  - Goal: data independence
  - structured, typed data - schema-driven - integrity constraints

- Semi-structured Data and Information Integration
  - Goal: integrate autonomous data sources
  - data source schema not known in detail - schemata are dynamic
  - schema might be revealed through analysis only after data processing
XML and CM/DL

- XML can be used to represent documents and data
  - content and structure
- XML is a text-oriented language
  - not really suitable for multimedia content
- Multi-media content can be referenced in XML documents
  - URI, XLink, XPointer, XPath
  - Synchronized Multimedia Integration Language (SMIL)
- Multi-media content can be encoded in a text-based format
  - Scalable Vector Graphics (SVG)
- XML processing standards support flexible generation of different layout
  - XML Style Sheets Transformations
- XML for meta-data representation
  - RDF
  - Meta-data standards (e.g., Dublin Core)
    - Representation in XML

XML Language Specifications (W3C)

covered in foundation courses, see appendix!
XLink - XML Linking Language

- Hyperlinks, references in XML documents
  - separate specification
  - based on Uniform Resource Identifiers (URIs), XPath, XPointer as referencing mechanisms
  - more powerful than HTML hyperlinks (see chapter 7)
    - bi-directional, more than two resources (n-ary links)
    - powerful addressing of resources
      - direct reference to object components
    - link attributes provide metadata
    - storage of links independent of resources
  - but also provides support for "simple" links (comp. to HTML) through special abbreviated syntax

XLink Elements and Attributes

- XLink Element
  - has special attributes defined in XLink namespace
    - type (simple, extended, ...)
    - href (URI-reference or XPointer)
    - title, role (describes link semantics)
    - show (new, replace, embed, undefined) (activation behavior)
    - actuate (onLoad, onRequest, undefined) (link traversal)
    - from, to (definition of directed edges in link graph)

- Link types
  - outbound: local start resource, remote end resource
  - inbound: remote start resource, local end resource
  - third-party: remote start resource, remote end resource
XLink - Example

- in DTD:

```xml
<!ELEMENT Player ANY>
<!ATTLIST Player
    xlink:type (simple) #FIXED "simple"
    xlink:href CDATA #REQUIRED
    xlink:role NMTOKEN #FIXED "http://www.fck.com/links/spieler"
    xlink:title CDATA #IMPLIED
    xlink:show (new|embed|replace) "replace"
    xlink:actuate (onLoad|onRequest) "onRequest"
/>
```

- in document instance:

```xml
<Player xlink:href="http://www.fck.de/Spielerliste.xml"
    xlink:title="List of all FCK players"
    xlink:show="new">
    Here's a list of all FCK players.
</Player>
```

---

XSL - Transformation and Layout

- **XSL**: Extensible Stylesheet Language
  - formatting engine for XML
  - XML markup is presentation/layout-independent

- **XSLT**: XSL Transformation Language
  - stylesheet
  - transformation rules
    - consisting of a pattern and a template
    - usage of XPath
  - input tree
  - output tree

- **XSL-FO**: XSL Formatting Objects
  - vocabulary for the specification of formatting rules
  - reuse of complex formatting definitions
  - transformation into arbitrary formats (PDF, RTF, PostScript, ...)

---

© Prof.Dr.-Ing. Stefan Deßloch

Digital Libraries and Content Management
Principles of XSLT

- XSL processor
  - element-by-element processing of input tree, starting with the root
  - looks for applicable XSLT rule
- XSLT rule defines
  - template: for which element, in which relationship context does the rule apply
  - action:
    - what should be generated as output
    - reference to document content using 'select'-expressions
    - what elements should be processed next by the processor
    - `<xsl:apply-templates/>` - continue with child elements
    - extended syntax supports selection of specific elements, order restrictions, etc.
- Default rules (if no other rule is matched)
  - for all elements (incl. root), process the children
  - for all text nodes and attributes, use their value as output

XSL - Example

- a fragment of an XML document:
  ```xml
  <Content>
    <Paragraph>
      XSLT <foreignTerm> (XSL Transformation Language) </foreignTerm> is a <Emphasis> phantastic </Emphasis> language to transform XML documents into XHTML <foreignTerm> (Extensible HTML) </foreignTerm>.
    </Paragraph>
  </Content>
  ```
- result document:
  ```xml
  <html>
    <head>
      <title>An XSLT example</title>
    </head>
    <body>
      XSLT <i>(XSL Transformation Language)</i> is a <b>phantastic</b> language to transform XML documents into XHTML <i>(Extensible HTML)</i>.
    </body>
  </html>
  ```
**XSL - Example**

XSLT-Stylesheet:
```
<xsl:stylesheet xmlns:xsl=http://www.w3.org/XSL/Transform/1.0
    xmlns=http://www.w3.org/TR/xhtml1
    indent-rules="yes">
    <!-- Rule 1 -->
    <xsl:template match="/">
    <html>
        <head>
            <title>An XSLT example</title>
        </head>
        <body>
            <xsl:apply-templates/>
        </body>
    </html>
    </xsl:template>
    <!-- Rule 2 -->
    <xsl:template match="Paragraph">
        <xsl:apply-templates/>
    </xsl:template>
    <!-- Rule 3 -->
    <xsl:template match="Emphasis">
        <b><xsl:apply-templates/></b>
    </xsl:template>
    <!-- Rule 4 -->
    <xsl:template match="foreignTerm">
        <i><xsl:apply-templates/></i>
    </xsl:template>
</xsl:stylesheet>
```

**XML Storage**

- **Requirements**
  - effective storage
  - efficient access of XML documents and fragments/elements
  - recreation of original document (or information from the document)

- **Possible approaches**
  - complete document storage, indexing (text-based native approach)
    - text index
    - text index + structure index
  - decomposition and generic storage (model-based native approach)
    - document graph storage
    - storage of DOM or XDM (XQuery Data Model) information
  - structured mapping to DB schema
    - relational DBMS
    - object-oriented and object-relational DBMS
    - support for user-defined mapping
    - mapping may be incomplete

- Approaches are often used in combination!
Complete XML Document Storage

- XML document is stored "as is" (e.g., in its textual form)
- content (including whitespace) is completely preserved
- in addition: text index (e.g., inverted word list)
- text retrieval cannot differentiate markup (e.g., element names) from text content
- no structural information
  - search for "hotel" AND "Saarbrücken" will return the above document!
  - result is always the full document

Document Storage With Structural Index

- XML structural index
- addtl. structural information

<table>
<thead>
<tr>
<th>Term</th>
<th>Reference</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kaiserslautern</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hammerweier</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saarbrücken</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema definition</td>
<td>not required</td>
</tr>
<tr>
<td>document reconstruction</td>
<td>not needed (original document is preserved)</td>
</tr>
<tr>
<td>queries</td>
<td>information retrieval exploit markup (structure) possible XML queries supported</td>
</tr>
<tr>
<td>additional comments</td>
<td>fulltext functionality (SQL-MM)</td>
</tr>
<tr>
<td>usage</td>
<td>document-centric or semi-structured</td>
</tr>
</tbody>
</table>

Complete Document Storage

- Complete Document
- Decomposed Generic Storage
- Structure Mapping to Database

- Text Index
- Text and XML Structure Index
- Information Retrieval Search Engines
  - OR-Text
    - Oracle Text
    - DB2 Text Extender
    - Informix Text Data Blades

- DB2 Text Extender
  - Oracle Text
  - Informix Verity Text Search Data Blade
  - XML Search Engines
Decomposition and Generic Storage

- Storage of document graph structure
  - nodes: elements and attributes
  - edges: child/attribute relationship
  - RDBMS can be used as infrastructure, based on generic schema
    - table ELEMENTS with columns DOCID, ELEMENTNAME, ID, PARENT, POSITION, VALUE
    - table ATTRIBUTE with columns DOCID, ATTRIBUTENAME, ELEMENTID, VALUE

<table>
<thead>
<tr>
<th>docID</th>
<th>Elementname</th>
<th>ID</th>
<th>Parent</th>
<th>Position</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>H0001</td>
<td>hotel</td>
<td>001</td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>H0001</td>
<td>hotelname</td>
<td>002</td>
<td>001</td>
<td>1</td>
<td>Hotel Blechhammer</td>
</tr>
<tr>
<td>H0002</td>
<td>address</td>
<td>003</td>
<td>001</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- Storage of Document Object Model (DOM) or XQuery Data Model (XDM) information
  - generic storage schema follows node types
    - Document, Element, Attribute, ....
  - RDBMS can be used as infrastructure as well

DOM

- DOM structure is a tree structure with different node types

<table>
<thead>
<tr>
<th>Node type</th>
<th>Contains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>Element (maximum of one), Processinginstruction, Comment, DocumentType</td>
</tr>
<tr>
<td>DocumentFragment</td>
<td>Element, Processinginstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>DocumentType</td>
<td>no children</td>
</tr>
<tr>
<td>EntityReference</td>
<td>Element, Processinginstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Comment</td>
<td>no children</td>
</tr>
<tr>
<td>Text</td>
<td>no children</td>
</tr>
<tr>
<td>CDATASection</td>
<td>no children</td>
</tr>
<tr>
<td>Entity</td>
<td>Element, Processinginstruction, Comment, Text, CDATASection, EntityReference</td>
</tr>
<tr>
<td>Notation</td>
<td>no children</td>
</tr>
</tbody>
</table>
### Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema definition</td>
<td>not required</td>
</tr>
<tr>
<td>document reconstruction</td>
<td>possible (&quot;equivalent&quot; document), but expensive</td>
</tr>
<tr>
<td>queries</td>
<td>XML queries, DB-queries (need to be adjusted to the storage schema used)</td>
</tr>
<tr>
<td>additional comments</td>
<td>queries over many elements/attributes become expensive</td>
</tr>
<tr>
<td>usage</td>
<td>data-, document-centric, or semi-structured</td>
</tr>
</tbody>
</table>

### Generic Storage

- Complete Document
- Decomposed Generic Storage
  - (simple) representation of graph structure
  - storage of DOM, XDM information
- Structure Mapping to Database

**Implementations**
- Neil Bradley
- Florescu/Kossmann
- Shimura/Yoshikawa/Uemura
- Richard Edwards
- inforyte
- eXcelon XIS (POET)
Systems

InfonYTE-DB
- uses a persistent Document Object Model (PDOM) for XML storage
- doesn't use existing DBMS, but provides its own components for physical storage, optimized for XML documents
- query language: XQuery

Tamino
- model-based storage of XML documents
- query language: XPath/XQuery

eXcelon
- uses Document Object Model as a basis for generic storage
- all information supported for DOM nodes is stored in the OODBMS ObjectStore
- query language: OQL

Structured Storage Mapping to Databases

- XML document structure is reflected at the schema level
  - prerequisite: explicit XML schema exists
- Mapping to DB schema
  - default mapping by system
  - user-defined (application-specific) mapping
- Mapping definition for
  - transformation of query results
  - DB schema generation

Example (for object-relational DBMS):

<table>
<thead>
<tr>
<th>Hotel</th>
<th>id</th>
<th>hotelname</th>
<th>address</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>67659</td>
<td>Hotel Blechhammer</td>
<td>67659 Kaiserslautern Am ...</td>
<td></td>
</tr>
</tbody>
</table>
Pros and Cons

Advantages: when storing structured data
- Queries, data types, aggregate functions, views, ...
- Integration into/with existing databases

Disadvantages: when storing semi- and unstructured data
- Large schema, sparsely populated, many null values
- No/restricted data type flexibility, problems with storing alternatives
- No/weakly integrated information/text retrieval support

Properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>schema definition</td>
<td>required</td>
</tr>
<tr>
<td>document reconstruction</td>
<td>usually not possible (unless mapping is complete and complete mapping process is &quot;logged&quot;)</td>
</tr>
<tr>
<td>queries</td>
<td>- database queries</td>
</tr>
<tr>
<td></td>
<td>- XML queries possible (translation)</td>
</tr>
<tr>
<td>additional comments</td>
<td>- supports integration with existing data sources</td>
</tr>
<tr>
<td></td>
<td>- XML documents and DB &quot;independent from each other&quot; (keeping the original document)</td>
</tr>
<tr>
<td>usage</td>
<td>data-centric applications</td>
</tr>
</tbody>
</table>
Structural Mapping to Databases

- Complete Document
- Decomposed Generic Storage
- Structural Mapping to Database
  - Complete (automatic) Mapping
  - User-defined Mapping

Shanmugasundaram et al. (Ronald Bourret)
(Deutsch/Fernandez/Suciu: Stored)
Klettke/Meyer: POET

Deutsch/Fernandez/Suciu: Stored
Ronald Bourret
DB2 XML Extender

Systems: System-defined Mapping

POET
- for each element of DTD/XML-Schema specification, a corresponding Java class is generated
- stored in the system by generating a corresponding DB-relation
- for a document collection that has no schema, a fixed DB-schema is used, which is based on the DOM information model

Oracle
- starting with Oracle 8i, tools for XML storage are offered
- part of Oracle XML Developer's Kit (XDK)
- support for XML-Parser and XSL-Transformer
System: User-defined Mapping

IBM DB2 XML-Extender
- storage of XML documents in DB2 DBMS
- based on user-defined mapping definition file (DAD - Data Access Definition)

Oracle
- object-relational storage of XMLType
- storage mapping using XMLSchema annotations
- XML document access using XPath or SQL

Microsoft SQL-Server
- annotated XDR Schema for user-defined mapping
  - sql:relation, sql:field, ...

SQL and XML

- Use existing (object-)relational technology?
  - Large Objects: granularity understood by DBMS may be too coarse!
    - search/retrieval of subsets, update of documents
  - Decompose into tables: often complex, inefficient
    - mapping complexity, especially for highly "denormalized" documents
  - Useful, but not sufficient
    - should be standardized as part of SQL
    - but needs further enhancement to support "native" XML support in SQL

- Enable "hybrid" XML/relational data management
  - supports both relational and XML data
    - storage, access
    - query language
    - programming interfaces
  - ability to view/access relational as XML, and XML as relational
  - all major relational DBMS vendors are moving into this direction
SQL/XML Big Picture

- XML, XQuery client
- Enhanced SQL client
- SQL client

SQL/XML

Storage

Client view

SQL/XML:2003 Parts and Packages

- Two major goals:
  - "Publish" SQL query results as XML documents
  - Ability to store and retrieve XML documents
- Rules for mapping SQL types, SQL identifiers and SQL data values to and from corresponding XML concepts
- A new built-in type XML
- A number of built-in operators that produce values of type XML

Recent additions for SQL200n:
- Integration of the XQuery Data Model
- Additional XML Constructor Functions
- Querying XML values

Core SQL
XML Publishing Functions - Example

```sql
CREATE VIEW XMLDept (DeptDoc XML) AS (
    SELECT XMLELEMENT ( NAME "Department",
        XMLATTRIBUTES ( e.dept AS "name" ),
        XMLATTRIBUTES ( COUNT(*) AS "count",
        XMLAGG (XMLELEMENT (NAME "emp",
        XMLATTRIBUTES(NAME "name", e.lname)
        XMLATTRIBUTES(NAME "hire", e.hire))
        ) AS "dept_doc"
    FROM employees e GROUP BY dept ;

==>

<Department name="Accounting" count="2">
    <emp><name>Yates</name><hire>2005-11-01</hire></emp>
    <emp><name>Smith</name><hire>2005-01-01</hire></emp>
</Department>

<Department name="Shipping" count="2">
    <emp><name>Oppenheimer</name><hire>2002-10-01</hire></emp>
    <emp><name>Martin</name><hire>2005-05-01</hire></emp>
</Department>

```

Product Support

- The “big three” support XML in SQL databases
  - IBM, Oracle implement (almost) complete support of SQL/XML
  - Microsoft supports similar capabilities using proprietary syntax
  - all three support XQuery inside SQL
  - differences in implementation of XML storage
- IBM DB2 upcoming release (SIGMOD2005, VLDB2005)
  - CLOB-based as well as native storage for XML values
  - efficient storage, indexing, processing techniques
  - allows to include SQL requests in XQuery expressions, too
- Oracle 10g (Oracle XML-DB technical whitepaper, VLDB2004)
  - storage based on CLOBs or object-relational tables
  - additional indexing capabilities, XML query rewrite
  - protocols (ftp, WebDAV, …) for supporting file-oriented XML storage/access
- Microsoft SQL Server 2005 (MSDN whitepaper, VLDB2005)
  - stored as BLOB in an internal format
  - primary (B+ tree) and secondary indexes, query processing based on mapping to RDM
Hybrid SQL/XML Databases

- Increasing importance of XML in combination with data management
  - flexible exchange of relational data using XML
  - managing XML data and documents
  - trend towards "hybrid" approaches for relational DBMS
- SQL/XML standard attempts to support the following
  - "Publish" SQL query results as XML documents
  - Ability to store and retrieve (parts of) XML documents with SQL databases
  - Rules and functionality for mapping SQL constructs to and from corresponding XML concepts
- Relies partly on XQuery standard
  - XML data model
  - queries over XML data
- Broad support by major SQL DBMS vendors
- Additional standards to further extend and complete the "big picture"!
  - XQJ: XML queries in Java
  - Grid Data Access Services (GGF): web/grid services to access DBs using SQL, XQuery

Resource Description Framework (RDF)

- Language for representing information (e.g., meta data) about resources on the web
  - identify something on the web using Uniform Resource Identifier (URI)
  - describe it using simple property/value pairs
- RDF statement can be represented using a graph
  - Example (from the RDF spec): "there is a Person identified by http://www.w3.org/People/EM/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr."
- RDF heritage: knowledge representation
  - semantic networks
RDF/XML

- XML-based syntax for encoding and exchanging RDF statements
  - Example
    ```xml
    <html version="1.0">
    <rdf:RDF xmlns:rdf="">
      <rdf:RDF
        xmlns:rdf="">
        <contact:Person
          rdf:about="http://www.w3.org/People/EM/contact#me">
          <contact:fullName>Eric Miller</contact:fullName>
          <contact:mailbox rdf:resource="mailto:em@w3.org"/>
          <contact:personalTitle>Dc.</contact:personalTitle>
        </contact:Person>
    </rdf:RDF>
    </rdf:RDF>
    ```
  - Could be used to provide semantic markup for XHTML documents
  - extension of the HTML META tag

Dublin Core

- Meta-data standard for describing networked resources
  - established by international, cross-disciplinary group of professionals from librarianship, computer science, text encoding, the museum community, and other related fields
  - major goal: help improve resource discovery
  - also used in closed environments, for other purposes (e.g., meta-data exchange)
- Meta-data description
  - uses a set of common meta data elements (nouns) and qualifiers (adjectives)
    - can be embedded in the resource (e.g., as HTML meta tags)
    - can be contained in a separate record/description of a resource (e.g., in a meta-data catalog file or database)
- Dublin Core defines
  - a vocabulary for meta data
    - simple to use, based on commonly used semantics, international, extensible,
  - best practices of how to use the language in various formats
    - HTML, XML, RDF
### Dublin Core Elements

- Elements can be broadly grouped into three categories:
  - **Content**
    - **Title**: A name given to the resource.
    - **Subject**: The topic of the content of the resource.
    - **Description**: An account of the content of the resource.
    - **Type**: The nature or genre of the content of the resource.
    - **Source**: A reference to a resource from which the present resource is derived.
    - **Relation**: A reference to a related resource.
    - **Coverage**: The extent or scope of the content of the resource.
  - **Intellectual Property**
    - **Creator**: An entity primarily responsible for making the content of the resource.
    - **Contributor**: An entity responsible for making contributions to the resource content.
    - **Publisher**: An entity responsible for making the resource available.
    - **Rights**: Information about rights held in and over the resource.
  - **Instantiation**
    - **Date**: A date associated with an event in the life cycle of the resource.
    - **Format**: The physical or digital manifestation of the resource.
    - **Identifier**: An unambiguous reference to the resource within a given context.
    - **Language**: A language of the intellectual content of the resource.

### Qualifiers

- Each element is optional and repeatable.
- There is no defined order of elements.
- Definition of controlled vocabularies possible (i.e., permitted values for elements).
- Two broad classes:
  - Element Refinement: make the meaning of an element narrower or more specific.
  - Encoding Scheme: identify schemes that aid in the interpretation of an element value.

### Example: Element Date

- **Refinements**
  - Created, Valid, Available, Issued, Modified, Date Copyrighted, Date Submitted
- **Encoding Schemes**
  - DCMI Period, W3C-DTF

### Example: Element Relation

- **Refinements**
- **Encoding Scheme**
  - URI
DC XML Implementation Guidelines

- Each DC element is represented as a separate XML element
  - refinements become elements of their own
  - encoding schemes are represented using xsi:type
- Example

  ```xml
  <metadata xmlns=...>
  <dc:title> UKOLN </dc:title>
  <dc:subject> national centre, network information support </dc:subject>
  <dc:identifier xsi:type="dcterms:URI"> http://www.ukoln.ac.uk/ </dc:identifier>
  <dcterms:modified xsi:type="dcterms:W3CDTF"> 2001-07-18 </dcterms:modified>
  ... 
  </metadata>
  ```

DC RDF Implementation Guidelines

- Each resource is described in an RDF description element
  - most appropriate URI to be used for rdf:about attribute
- Example

  ```xml
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc = "http://purl.org/dc/elements/1.1/"> 
  <rdf:Description rdf:about="http://dublincore.org/"> 
  <dc:title>Dublin Core Metadata Initiative - Home Page</dc:title> 
  <dc:description> The Dublin Core Metadata Initiative Web site. </dc:description> 
  <dc:date>2001-01-16</dc:date> 
  <dc:format>text/html</dc:format> 
  <dc:language>en</dc:language> 
  <dc:contributor>The Dublin Core Metadata Initiative</dc:contributor> 
  </rdf:Description> 
  </rdf:RDF>
  ```
XML Documents

- XML documents are text (unicode)
  - markup (always starts with '<' or '&')
    - start/end tags
    - references (e.g., &lt, &amp, ...)
    - declarations, comments, processing instructions, ...
  - data (character data)
    - characters '<' and '&' need to be indicated using references (e.g., &lt) or using the character code
    - alternative syntax: <![CDATA[ (a<b)&(c<d) ]]>  
  - XML documents are well-formed
    - logical structure:
      - [declaration] [dtd] [comment-or-PI] element [comment-or-PI]
      - (optional) XML declaration (XML version, encoding, ...)
      - (optional) schema (DTD)
      - single root element (possibly nested)
      - comments
      - processing instructions
      - example: reference to a stylesheet, used by a browser
    - additional requirements on the structure and content of element
XML Documents: Elements

- **Tag**: label for a section of data
- **Element**:
  - start tag `<tagname>`
  - content: text and/or nested element(s)
    - may be empty; alternative syntax: `<tagname/>`
  - end tag `</tagname>`
- Elements must be properly **nested** for the document to be **well-formed**
- Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Mixture of text with sub-elements (mixed content) is legal in XML
  - Example:
    ```xml
    <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
  - Element content (i.e., text and nested elements) is ordered!

XML Documents: Attributes

- **Attributes**: can be used to further describe elements
  - attributes are specified by `name="value"` pairs inside the starting tag of an element
  - value is a text string
    - no further structuring of attribute values
  - attributes are not ordered
  - Example:
    ```xml
    <account acct-type = "checking">
      <account-number> A-102 </account-number>
      <branch-name> Perryridge </branch-name>
      <balance> 400 </balance>
    </account>
    ```
  - Well-formed documents:
    - attribute names must be unique within the element
    - attribute values are enclosed in single or double quotation marks
Namespaces

- A single XML document may contain elements and attributes defined by different vocabularies
  - Motivated by modularization considerations, for example
- Name collisions have to be avoided
- Example:
  - A Book vocabulary contains a Title element for the title of a book
  - A Person vocabulary contains a Title element for an honorary title of a person
  - A BookOrder vocabulary uses both vocabularies
- Namespaces specifies how to construct universally unique names

Namespaces (cont.)

- Namespace is a collection of names identified by a URI
- Namespaces are declared via a set of special attributes
  - These attributes are prefixed by xmlns - Example:
    <BookOrder xmlns:Customer="http://mySite.com/Person"
    xmlns:Item="http://yourSite.com/Book">
    ...Customer:Title='Dr'...
    ...Item:Title='Introduction to XML'...
- Elements/attributes from a particular namespace are prefixed by the name assigned to the namespace in the corresponding declaration of the using XML document
- Default namespace declaration for fixing the namespace of unqualified names
  - Example:
    <BookOrder xmlns="http://mySite.com/Person"
    xmlns:Item="http://yourSite.com/Book">
XML Document Schema

- XML documents may optionally have a schema
  - standardized data exchange, ...
- Schema restricts the structures and data types allowed in a document
  - document is **valid**, if it follows the restrictions defined by the schema
- Two important mechanisms for specifying an XML schema
  - Document Type Definition (DTD)
  - XML Schema

Document Type Definition - DTD

- Original mechanism to specify type and structure of an XML document
  - 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
- Special DTD syntax
  - `<ELEMENT element (subelements-specification) >`
  - `<ATTLIST   element (attributes)  >`
- DTD is
  - contained in the document, or
  - stored separately, referenced in the document
- DTD clause in XML document specifies the root element type, supplies or references the DTD
  - `<DOCTYPE bank [ ... ]>`
Schema Definition with XML Schema

- XML Schema is closer to the general understanding of a (database) schema
- XML Schema (unlike DTD) supports
  - Typing of values
    - E.g., integer, string, etc
  - Constraints on min/max values
  - Typed references
  - User defined types
  - Schema specification in XML syntax
    - schema is a well-formed and valid XML document
  - Integration with namespaces
  - Many more features
    - List types, uniqueness and foreign key constraints, inheritance...
- BUT: significantly more complicated than DTDs

XQuery

- XQuery is a general purpose query language for XML data
- Standard developed by the World Wide Web Consortium (W3C)
  - W3C Recommendation since January 23rd, 2007
- XQuery is derived from
  - the Quilt "Quilt" refers both to the origin of the language and to its use in "knitting" together heterogeneous data sources
  - XPath: a concise language for navigating in trees
  - XML-QL: a powerful language for generating new structures
  - SQL: a database language based on a series of keyword-clauses: SELECT - FROM - WHERE
  - OQL: a functional language in which many kinds of expressions can be nested with full generality
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, attributes, text, etc.
- Example:

```xml
<?xml version = "1.0"?>
<!-- Requires one trained person -->
<procedure title = "Removing a light bulb">
  <time unit = "sec">15</time>
  <step>Grip bulb.</step>
  <step>Rotate it</step>
  <warning>slowly</warning> counterclockwise.
</procedure>
```

XQuery Data Model (XDM)

- Builds on a tree-based model, but extends it to support sequences of items
  - represent collections of documents and complex values
  - reflect (intermediate) results of query evaluation
  - closure property
    - XQuery queries and expressions operate on/produce instances of the XDM
- Based on XML Schema for precise type information
- XDM instance
  - ordered sequence of zero or more items
  - can contain heterogenous values
  - cannot be nested - all operations on sequences automatically "flatten" sequences
    - no distinction between an item and a sequence of length 1
    - may contain duplicate nodes (see below)
- An item is a node or an atomic value
- Atomic values are typed values
  - XML Schema simple types
  - important for representing results of intermediate expressions in the data model
XQuery – Main Constituents

- Path expressions
  - Inherited from XPath
  - An XPath expression maps a node (the context node) into a set of nodes

- Element constructors
  - To construct an element with a known name and content, use XML-like syntax:
    \[
    \text{<book isbn = "12345">}
    \text{<title>Huckleberry Finn</title>}
    \text{</book>}
    \]
  - If the content of an element or attribute must be computed, use a nested expression enclosed in \{ \}:
    \[
    \text{<book isbn = "\{x\}"} \text{<title>}
    \text{\{b/title \}}\text{</title>}
    \text{</book>}
    \]

- FLWOR - Expressions

Path Expressions in XQuery

- An XPath expression maps a node (the context node) into a sequence of nodes
  - consists of one or more steps separated by "/"
  - e.g.: return the names of all customers in bank:
    \[
    \text{/child::bank/child::customer/child::name}
    \]

- Evaluation of path expression
  - step by step, from left to right
  - starting from an externally provided context node, or from document root
  - each step works on a sequence of nodes
    - for each node in the sequence, look up other nodes based on step expression
    - eliminate duplicates from result sequence
    - sort nodes in document order
  - empty result sequence does not result in an error
**XQuery: The General Syntax Expression FLWOR**

- **FOR clause**, **LET clause** generate list of tuples of bound variables (order preserving) by
  - iterating over a set of nodes (possibly specified by a path expression), or
  - binding a variable to the result of an expression
- **WHERE clause** applies a predicate to filter the tuples produced by FOR/LET
- **ORDER BY clause** imposes order on the surviving tuples
- **RETURN clause** is executed for each surviving tuple, generates ordered list of outputs

**Associations to SQL query expressions**
- for \( \Rightarrow \) SQL from
- where \( \Rightarrow \) SQL where
- order by \( \Rightarrow \) SQL order by
- return \( \Rightarrow \) SQL select

Let allows temporary variables, and has no equivalent in SQL

---

**FLWOR - Examples**

- Simple FLWR expression in XQuery
  - Find all accounts with balance > 400, with each result enclosed in an `<account-number>` tag
    ```xml
    for $x in /bank-2/account
    let $acctno := $x/@account-number
    where $x/balance > 400
    return <account-number> {$acctno} </account-number>
    ```
  - Let and Where clause not really needed in this query, and selection can be done in XPath.
    - Query can be written as:
      ```xml
      for $x in /bank-2/account[balance>400]
      return <account-number> {$x/@account-number} </account-number>
      ```
Evaluating FLWOR Expressions

input sequence

```
      *...
      ...
      ...
```

```
FOR $X, $Y ..
LET $Z ..
WHERE ..
ORDER
```

```
RETURN ..
```

output sequence

```
      *...
      ...
      ...
```

```
ok!
```

```
tuple stream
```

```
ok!
```

```
WHERE ..
```

```
X
```

```
```

Application Programming with XML

- Application needs to work with XML data/document
- Parsing XML to extract relevant information
- Produce XML
  - Write character data
  - Build internal XML document representation and Serialize it
- Generic XML Parsing
  - Simple API for XML (SAX)
    - “Push” parsing (event-based parsing)
      - Parser sends notifications to application about the type of document pieces it encounters
      - Notifications are sent in “reading order” as they appear in the document
      - Preferred for large documents (high memory efficiency)
  - Document Object Model (DOM) – w3c recommendation
    - “One-step” parsing
      - Generates in-memory representation of the document (parse tree)
      - DOM specifies the types of parse tree objects, their properties and operations
        - Independent of programming language (uses IDL)
        - Bindings available to specific programming languages (e.g., Java)
    - Parsing includes
      - checking for well-formedness
      - optionally checking for validity (often used for debugging only)