Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de

Technische Universität Kaiserslautern

Chapter 13 - XML

XML Origin and Usages

- 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
  - For example:
    - <title> XML </title>
    - <slide> XML Origin and Usages </slide>
  - Meta-language: used to define arbitrary XML languages/vocabularies (e.g. XHTML)
- Derived from SGML (Standard Generalized Markup Language)
  - standard for document description
    - enables document interchange in publishing, office, engineering, ...
  - main idea: separate form from structure
- XML is simpler to use than SGML
  - roughly 20% complexity achieves 80% functionality

XML Origin and Usages (cont.)

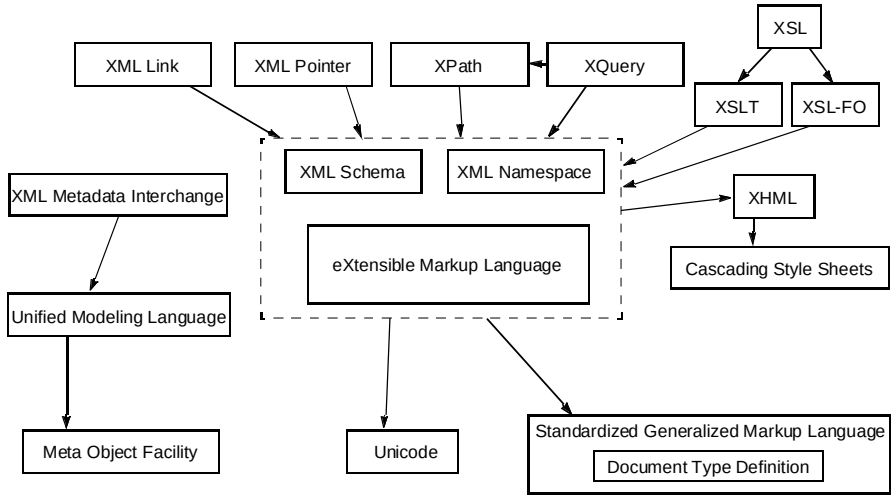
- XML documents are to some extent self-describing
  - Tags represent metadata
  - Metadata and data are combined in the same document
    - semi-structured data modeling
  - Example

```
<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>
```


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 Language Specifications



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
    - (optional) prolog (XML version, ...)
    - (optional) schema
    - root element (possibly nested)
    - comments, ...
  - correct sequence of start/end tags (nesting)
  - uniqueness of attribute names
  - ...

XML Documents: Elements

- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly **nested**
  - 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 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

XML Documents: Attributes

- **Attributes:** can be used to describe elements
- Attributes are specified by *name=value* pairs inside the starting tag of an element
- Example

```
<account acct-type = "checking" >
        <account-number> A-102 </account-number>
        <branch-name> Perryridge </branch-name>
        <balance> 400 </balance>
  </account>
```

- Attribute names must be unique within the element

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

XML Documents: 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
  ➔ 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
- IDs and IDREFs are untyped, unfortunately
  - Example below: 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 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>
```

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)**
    - contained in the document, or
    - stored separately, referenced in the document
  - **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) >

Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Structure is defined using regular expressions
  - sequence (*subel*, *subel*, ...), alternative (*subel* | *subel* | ...)
  - number of occurences
    - "?" - 0 or 1 occurrence
    - "+" - 1 or more occurrences
    - "*" - 0 or more occurrences
- Example

```
<!ELEMENT depositor (customer-name  account-number)>
<!ELEMENT customer-name(#PCDATA)>
<!ELEMENT account-number (#PCDATA)>
<!ELEMENT bank ( ( account | customer | depositor)+)>
```


Example: Bank DTD

```
<!DOCTYPE bank-2[
    <!ELEMENT account (branch-name, 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

]>
```

Describing XML Data: XML Schema

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

XML Schema Structures

- **Datatypes (Part 2)**
  Describes Types of scalar (leaf) values
- **Structures (Part 1)**
  Describes types of complex values (attributes, elements)
  - Regular tree grammars
    repetition, optionality, choice recursion
- **Integrity constraints**
  Functional (keys) & inclusion dependencies (foreign keys)
- **Subtyping (similar to OO models)**
  Describes inheritance relationships between types
- **Supports schema reuse**

XML Schema Structures (cont.)

- Elements : tag name & simple or complex type

```
<xs:element name="sponsor" type="xsd:string"/>
<xs:element name="action" type="Action"/>
```

- Attributes : tag name & simple type

```
<xs:attribute name="date" type="xsd:date"/>
```

- Complex types

```
<xs:complexType name="Action">
    <xs:sequence>
        <xs:elemref name ="action-date"/>
        <xs:elemref name ="action-desc"/>
    </xs:sequence>
</xs:complexType>
```


XML Schema Structures (cont.)

- Sequence

```
<xs:sequence>
    <xs:element name="congress" type=xsd:string"/>
    <xs:element name="session" type=xsd:string"/>
</xs:sequence>
```

- Choice

```
<xs:choice>
    <xs:element name="author" type="PersonName"/>
    <xs:element name="editor" type="PersonName"/>
</xs:choice>
```

- Repetition

```
<xs:element   name ="section"
              type="Section"
              minOccurs="1"
              maxOccurs="unbounded"/>
```

Namespaces

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

XML Schema Version of Bank DTD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.banks.org"
            xmlns ="http://www.banks.org" >
<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:sequence>
    </xsd:complexType>
</xsd:element>                  ..... definitions of customer and depositor ....
<xsd:complexType name="BankType">
    <xsd:choice minOccurs="1" maxOccurs="unbounded">
        <xsd:element ref="account"/>
        <xsd:element ref="customer"/>
        <xsd:element ref="depositor"/>
    </xsd:choice>
</xsd:complexType>
</xsd:schema>
```

XML Document Using Bank Schema

```
<bank   xmlns="http://www.banks.org"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.banks.org Bank.xsd">
   <account>
        <account-number> ... </account-number>
        <branch-name> ... </branch-name>
        <balance> ... </balance>
   </account>
   ...
</bank>
```


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)

Transforming and Querying XML Data

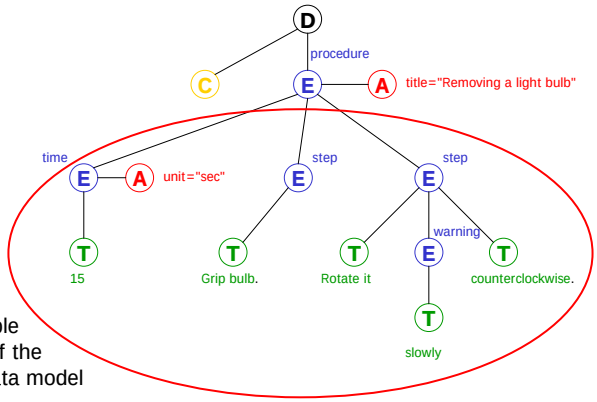
- XPath
  - path expressions for selecting document parts
  - not originally designed as a stand-alone language
- XSLT
  - transformations from XML to XML and XML to HTML
  - primarily designed for style transformations
  - recursive pattern-matching paradigm
    - difficult to optimize in a DBMS context
- XQuery
  - XML query language with a rich set of features
  - XQuery builds on experience with existing query languages: XPath, Quilt, XQL, XML-QL, Lorel, YATL, SQL, OQL, ...

XML Data Model

- There is no uniform XML data model
  - different approaches with different goals
    - XML Information Set, DOM Structure Model, XPath 1.0 data model, XQuery data model
- Common denominator: an XML document is modeled as a **tree**, with nodes of different **node types**
  - Document, Element, Attribute, Text, Namespace, Comment, Processing Instruction
- XQuery data model builds on a tree-based model, but extends it to support
  - **sequences** of items
    - nodes of different types (see above) as well as atomic values
    - can contain heterogeneous values, are ordered, can be empty
  - typed values and type annotations
    - result of schema validation
    - type may be unknown
- Closure property
  - XQuery expressions operate on/produce instances of the XQuery Data Model

Example

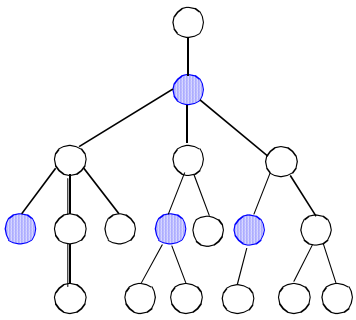
```
<?xml version = "1.0"?>
<!-- Requires one trained person -->
<procedure title = "Removing a light bulb">
  <time unit = "sec">15</time>
  <step>Grip bulb.</step>
  <step>
     Rotate it
     <warning>slowly</warning>
     counterclockwise.
  </step>
</procedure>
```



one possible instance of the XQuery data model

Processing XML Data: XPath

- XPath is used to address (select) parts of documents using path expressions
- A path expression consists of one or more steps separated by "/"
  - Each step in an XPath expression maps a node (the context node) into a set of nodes
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
  - E.g.: /bank-2/customer/customer-name evaluated on the bank-2 data returns
    - <customer-name> Joe </ customer-name>
    - < customer- name> Mary </ customer-name>
  - E.g.:/bank-2/customer/cust-name/text( ) returns the same names, but without the enclosing tags



XPath

- The initial "/" denotes root of the document (above the top-level tag)
- In general, a step has three parts:
  - The ***axis*** (direction of movement: child, descendant, parent, ancestor, following, preceding, attribute, … - 13 axes in all -)
  - A ***node test*** (type and/or name of qualifying nodes)
  - Some ***predicates*** (refine the set of qualifying nodes)
- 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-2/account[balance > 400]
    - returns account elements with a balance value greater than 400
    - /bank-2/account[balance] returns account elements containing a balance subelement
- Attributes are accessed using "@"
  - E.g. /bank-2/account[balance > 400]/@account-number
    - returns the account numbers of those accounts with balance > 400
  - IDREF attributes are not dereferenced automatically (more on this later)

XPath Summary

- Strengths:
  - Compact and powerful syntax for navigating a tree, but not as powerful as a regular-expression language
  - Recognized and accepted in XML community
  - Used in other XML processors/specifications such as XPointer, XSLT, XQuery
- Limitations:
  - Operates on one document (no joins)
  - No grouping or aggregation
  - No facility for generating new output structures

Transforming XML Data: 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

Understanding A Template

- Most templates have the following form:

```
<xsl:template match="emphasis">
  <i><xsl:apply-templates/></i>
</xsl:template>
```

- The whole <xsl:template> element is a **template**
- The **match pattern** determines where this template applies
  - XPath pattern
- **Literal result element(s)** come from non-XSL namespace(s)
- XSLT elements come from the XSL namespace

XQuery

- XQuery is a general purpose query language for XML data
- Standardized by the World Wide Web Consortium (W3C)
- 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) query language, which itself borrows from
  - **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

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:

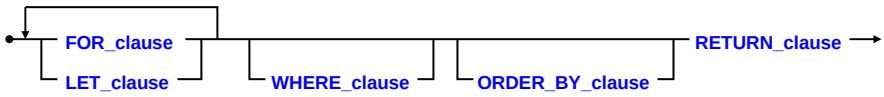
```
<book isbn = "12345">
   <title>Huckleberry Finn</title>
</book>
```

  - If the content of an element or attribute must be computed, use a nested expression enclosed in { }

```
<book isbn = "{$x}">
   {$b/title }
</book>
```

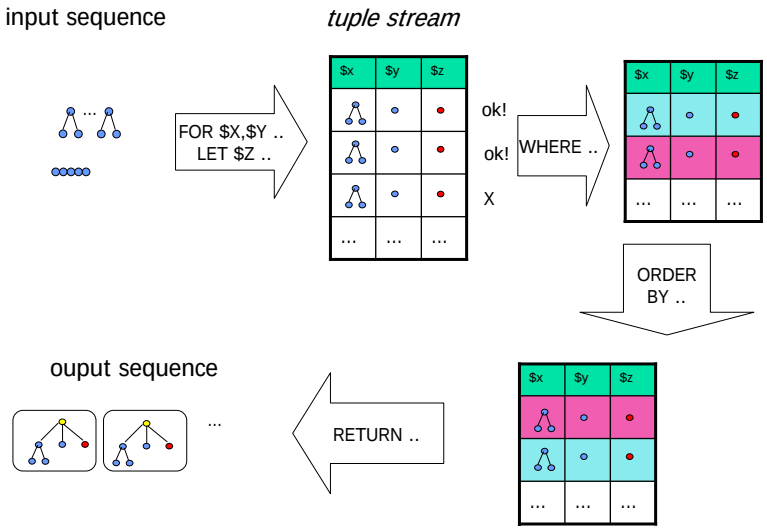
- FLWOR - Expressions

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 an XPath 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 ⇔ SQL from
  where ⇔ SQL where
  order by ⇔ SQL order by
  return ⇔ SQL select
  let allows temporary variables, and has no equivalent in SQL

Evaluating FLWOR Expressions



FLWOR - Examples

- Simple FLWR expression in XQuery
  - Find all accounts with balance > 400, with each result enclosed in an <account-number> .. </account-number> tag

```
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:

```
for       $x in /bank-2/account[balance>400]
return <account-number> {$x/@account-number}
                                  </account-number>
```


Nesting of Expressions

- Here: nesting inside the return clause
  - Example: inversion of a hierarchy

```
<book>
   <title>
   <author>
   <author>
</book>
<book>
   <title>
   <author>
   <author>
</book>
```

```
FOR $a IN fn:distinct-values(//author)
ORDER BY $a/name
RETURN
    <author>
        <name> { $a/text() } </name>
        { FOR $b IN //book[author = $a]
              RETURN $b/title }
    </author>
```

```
<author>
   <name>
   <title>
   <title>
</author>
<author>
   <name>
   <title>
   <title>
</author>
```

XQuery: Joins

- Joins are specified in a manner very similar to SQL

```
for   $a  in  /bank/account,
      $c  in  /bank/customer,
      $d  in  /bank/depositor
where $a/account-number = $d/account-number
   and $c/customer-name = $d/customer-name
return <cust-acct>{ $c $a }</cust-acct>
```

- The same query can be expressed with the selections specified as XPath selections:

```
for   $a in /bank/account
      $c in /bank/customer
      $d in /bank/depositor[
                  account-number =$a/account-number and
                  customer-name = $c/customer-name]
return <cust-acct>{ $c $a }</cust-acct>
```

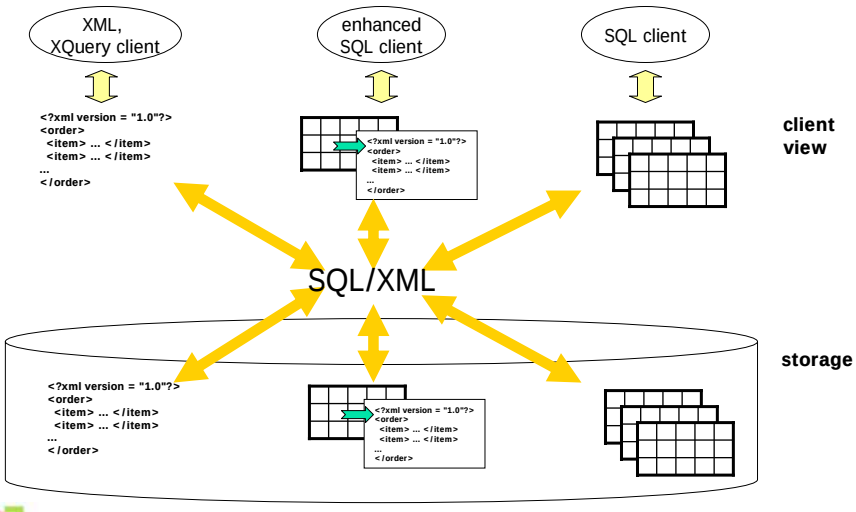

XQuery - Status

- Current status: w3c proposed recommendation
  - fairly close to becoming a w3c recommendation
- Ongoing and Future Work
  - Full-text support
  - Insert, Update, Delete
  - View definitions, DDL
  - Host language bindings, APIs
    - JSR 225: XQuery API for JavaTM (XQJ)
    - problem to overcome: traditional XML processing API is based on well-formed documents

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

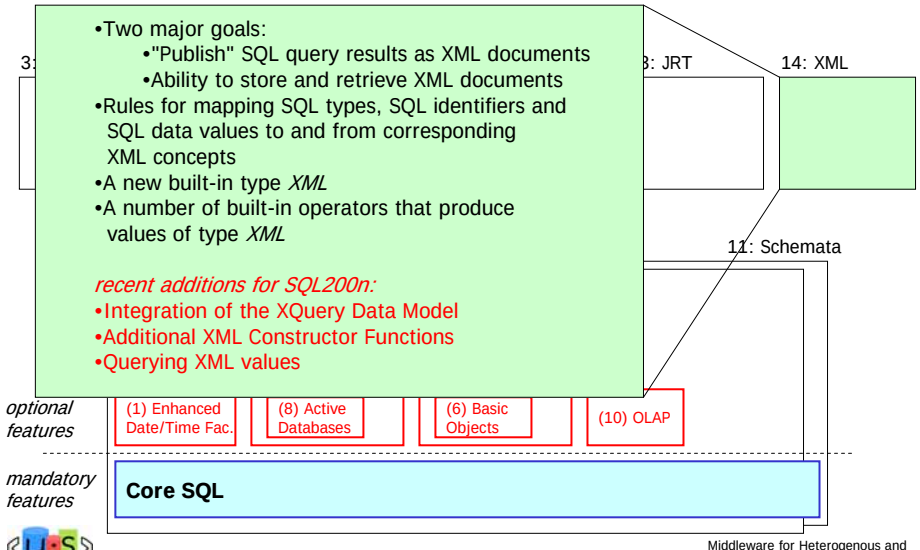


SQL:2003 Parts and Packages

- 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



XML Publishing Functions - Example

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

==>

```
dept_doc
```

```
<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>
```

Manipulating XML Data

- Constructor functions
  - focus on publishing SQL data as XML
  - no further manipulation of XML
- More requirements
  - how do we select or extract portions of XML data (e.g., from stored XML)?
  - how can we decompose XML into relational data?
    - XMLCAST is not sufficient
  - both require a language to identify, extract and possibly combine parts of XML values

SQL/XML utilizes the XQuery standard for this!

XMLQUERY

- Evaluates an XQuery or XPath expression
  - returns a sequence of XQuery nodes
- XMLQUERY – Example

```
SELECT XMLQUERY('for $e in $dept[@count > 1]/emp
                 where $e/hire > 2004-12-31 return $e/name'
          PASSING BY REF dept_doc AS "dept"
          RETURNING SEQUENCE) AS "Name_elements"
FROM XMLDept
```

=>

Name_elements
<name>Yates</name> <name>Smith</name
<name>Martin</name>

Summary: XML Advantages for Integration

- Integrates data and meta-data (tags)
  - Self-describing
- XMLSchema, Namespaces
  - Defining valid document structure
  - Integrating heterogenous terminology and structures
- XML can be validated against schema (xsd, dtd) outside the application
- Many technologies exist for processing, transforming, querying XML documents
  - DOM, SAX, XSLT, XPath, XQuery
- XML processing can help handle schema heterogenity, schema evolution
  - Focus on known element tags, attributes, namespaces ...
  - Powerful filter and transformation capabilities
- XML is independent of platforms, middleware, databases, applications ...

Summary: XML and Data Management

- 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
- 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