XML

Norbert Fuhr
University of Duisburg-Essen
Germany
Content

1. XML standards
   - plain XML
   - XML namespaces
   - DTDs and XML schema

2. XML Query Languages
   - Requirements
   - Development
   - XPath and XQuery
   - XML databases
Part I

XML Standards
Content

- Introduction

- XML namespaces

- Document Type Definitions (DTDs)

- XML Schema

- XML Stylesheet Language (XSL)

- Other standards

(For details of the XML standards, see http://www/w3c.org)
Introduction: Example XML document

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE book SYSTEM "/services/dtds/book.dtd">
<book class="H.3.3">
  <author>John Smith</author>
  <title>XML Retrieval</title>
  <chapter>
    <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>
    <heading>XML Query Language XQL</heading>
    <section>
      <heading>Examples</heading>
    </section>
    <section>
      <heading>Syntax</heading>
      Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
XML Retrieval

This...

XML Query Language XQL

Examples

Syntax

We describe syntax of XQL
XML properties

- **hierarchical structure**: nesting of elements
- **element**: start-tag – content – end tag
  
  `<tag-name> content </tag-name>`

- **tag-name**: logical name of element

- **content**: data or other elements
  (nesting of elements)
  
  `<author><first>John</first><last>Smith</last></author>`

- **attributes**: assigned to elements
  (specified in start tag)
  
  pair of *(attribute name, attribute value)*,
  e.g. `<date format="ISO">2000-05-01</date>`
XML: Basic ideas

- **markup of logical structure of documents**
  ≈ explicit logical structure, can be exploited by appropriate IR methods

- **separation of logical structure and layout**
  ≈ different presentations of one document, depending on output media, user group (language, ...)

- **support interoperability** of Web services and XML-based applications
  ≈ standard document format for IR systems
Basic XML standard does not deal with ...

- standardization of element names
  → XML namespaces

- structure of element content
  → XML DTDs

- data types of element content
  → XML schema
I.1 XML namespaces

allow for combination of element names defined independently (in different resources)

<?xml version="1.0"?>

<bk:book xmlns:bk='urn:loc.gov:books'
  <bk:title>Cheaper by the Dozen</bk:title>
  <isbn:number>1568491379</isbn:number>
</bk:book>
Example: Dublin Core namespace

  <dc:title>Generic Algebras with Involution of Degree 8m</dc:title>
  <dc:subject>Orthogonal group, Symplectic group,
  invariant field, rational
</dc:subject>
  <dc:date>2001-02-27</dc:date>
  <dc:format>application/postscript</dc:format>
  <dc:source>ESI preprints</dc:source>
  <dc:language>en</dc:language>
</oai_dc:dc>
I.2 Document Type Definitions

- **well-formed XML**: proper nesting of elements
  (e.g. `<a><b></a></b>` is forbidden)

- **valid XML**: document is well-formed and conforms to document type definition

**Declaration of DTD**
in the document header:

```xml
<!DOCTYPE name PUBLIC publicid systemid>
```

```xml
<!DOCTYPE name SYSTEM filename>
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>My Home Page</title>
  </head>
  <body>
    <p>Hello! This is my home page.</p>
  </body>
</html>
Example XML document with system DTD

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE book SYSTEM "/services/dtds/book.dtd">

<book class="H.3.3">
  <author>John Smith</author>
  <title>XML Retrieval</title>
  <chapter>
    <heading>Introduction</heading>
    This text explains all about XML and IR.
  </chapter>
  <chapter>
    <heading>XML Query Language XQL</heading>
    <section>
      <heading>Examples</heading>
    </section>
    <section>
      <heading>Syntax</heading>
      Now we describe the XQL syntax.
    </section>
  </chapter>
</book>
```
DTD for example document

<!ELEMENT book (author, title, chapter+)>  
<!ELEMENT author (#PCDATA)>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT chapter (heading,#PCDATA?,section*)>  
<!ELEMENT section (heading,#PCDATA?)>  
<!ELEMENT heading (#PCDATA)>  
<!ATTLIST book   
    class CDATA #REQUIRED  
    crdate CDATA #IMPLIED  
    type (monograph|collection|proceedings) "monograph">
DTD Specification

- element definitions
- definition of element attributes
- entity definitions (macros)
Element Definitions

elementdecl ::= '<!ELEMENT' Name contentspec '>
contentspec ::= 'EMPTY' | 'ANY' | Mixed | children
Mixed ::= '('#PCDATA' (' | ' Name)* ')'*
        | '('#PCDATA' ')
children ::= (choice | seq) ('?' | '*' | '+')?
cp ::= (Name | choice | seq) ('?' | '*' | '+')?
choice ::= '('# cp ( '|' cp )* ')'
seq ::= (' cp ( ',' cp )* ')"
Element Definition: Examples

<!ELEMENT br EMPTY>
<!ELEMENT p (#PCDATA|emph)* >
<!ELEMENT %name.para; %content.para; >
<!ELEMENT container ANY>
<!ELEMENT book (author, title, chapter+)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT chapter (heading, #PCDATA?, section*)>
<!ELEMENT section (heading, #PCDATA)>
Definition of element attributes

\[
\text{AttlistDecl} ::= ' <!ATTLIST' Name AttDef* ' >'
\]

\[
\text{AttDef} ::= \text{Name \ AttType \ DefaultDecl}
\]

\[
\text{AttType} ::= \text{StringType} \mid \text{TokenizedType} \mid \text{EnumeratedType}
\]

\[
\text{StringType} ::= 'CDATA'
\]

\[
\text{TokenizedType} ::= 'ID' \mid 'IDREF' \mid 'IDREFS' \mid 'ENTITY' \mid 'ENTITIES' \mid 'NMTOKEN' \mid 'NMTOKENS'
\]

\[
\text{EnumeratedType} ::= \text{NotationType} \mid \text{Enumeration}
\]

\[
\text{Enumeration} ::= '(' Nmtoken ('|' Nmtoken)* ')'
\]

\[
\text{DefaultDecl} ::= '#REQUIRED' \mid '#IMPLIED' \mid (('#FIXED')? \text{AttValue})
\]
I.3 XML Schema

types of XML applications:

1. structured documents
   text documents with markup of logical structure
   $\leadsto$ document-centric

2. formatted data
   (e.g. spreadsheets, business forms, databases, ...)
   XML as exchange format
   $\leadsto$ data-centric
XML Schema

resolves weaknesses of DTDs wrt. formatted data:

- **support for data types**
  (DTDs: only child elements, PCDATA, mixed content and EMPTY)

- **specification of structured data**
  (e.g. arrays with lower/upper bounds)

- **reuse of data-types**
  (explicit support for elements only, but not for attributes)

- **extensible type system**
  (user-defined types / subtypes)

- **support for namespaces**

- **support for reference mechanism**
  (ID, IDREF(S)) supports local references only)

- **DTD syntax in XML**
Classification of XSD types

- **atomic vs. aggregated:**
  - atomic: atomic values, content not further interpreted
  - aggregated: lists, union (of different types)

- **primitive vs. derived:**
  - primitive: independent of other types
  - derived: subtype of other type

- **predefined vs. user-defined:**
  - XML Schema Part 2 defines 44 types
  - users may define additional types by aggregation or as subtypes of existing types
Subtyping

- restriction
  - value restriction (string length, range, string pattern)
  - cardinality restriction (min, max bounds) of arrays

- extension
  (adding elements to a type, like type attributes in object-oriented programming)

- redefinition
  (redefine types of a given schema definition)
Example type definitions

<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
<xsd:complexType name="PurchaseOrderType">
  <xsd:sequence>
    <xsd:element name="shipTo" type="USAddress"/>
    <xsd:element name="billTo" type="USAddress"/>
    <xsd:element ref="comment" minOccurs="0"/>
    <xsd:element name="items" type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
</xsd:complexType>
Subtyping

- restriction
  - value restriction
    (string length, range, string pattern)
  - cardinality restriction
    (min, max bounds) of arrays

- extension
  (adding elements to a type, like type attributes in OODBs)

- redefinition
  (redefine types of a given schema definition)
numeric range restriction:

```xml
<xsd:simpleType name="myInteger">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="10000"/>
      <xsd:maxInclusive value="99999"/>
   </xsd:restriction>
</xsd:simpleType>
```

pattern restriction

```xml
<xsd:simpleType name="SKU">
   <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{3}-[A-Z]{2}"/>
   </xsd:restriction>
</xsd:simpleType>
```
restriction by enumeration

<xsd:simpleType name="USState">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="AK"/>
        <xsd:enumeration value="AL"/>
        <xsd:enumeration value="AR"/>
        <!-- and so on ... -->
    </xsd:restriction>
</xsd:simpleType>
List type

simple example

<xsd:simpleType name="listOfMyIntType">
   <xsd:list itemType="myInteger"/>
</xsd:simpleType>

<xlistOfMyInt>20003 15037 95977 95945</xlistOfMyInt>
list of fixed length

<xsd:simpleType name="USStateList">
  <xsd:list itemType="USState"/>
</xsd:simpleType>

<xsd:simpleType name="SixUSStates">
  <xsd:restriction base="USStateList">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>
I.4 XSLT

transformation of XML documents
(e.g. from XML into HTML)

XSLT-Stylesheet =
frame + set of transformation rules

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns="http://www.w3.org/TR/REC-html40">
    <xsl:output method="html"/>

    <xsl:template match="...">
        ...
    </xsl:template>

</xsl:stylesheet>
```
Some XSLT elements

<xsl:template>
specifies a template rule
match attribute identifies source node(s) to which rule applies

<xsl:if>
test attribute specifies an expression:
if true, content template is instantiated

<xsl:choose>
selects one among a number of possible alternative child elements
<xsl:when> and <xsl:otherwise>

<xsl:when>
if expression specified by test attribute is true, content template is instantiated

<xsl:text>
contains literal data to be included in the output
A small example

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE brief SYSTEM "brief.dtd">

<brief>

<anrede geschlecht="f" sozial="du">Nora</anrede>

<text>
habe gerade den Ulysses beendet. Mal sehen, wann der in den USA gedruckt werden darf...
</text>

<gruss>J</gruss>

</brief>
Stylesheet

<xsl:template match="/">
<html>
 <body>
  <xsl:apply-templates/>
 </body>
</html>
</xsl:template>

<xsl:template match="anrede">
<p>
 <xsl:choose>
  <xsl:when test="@sozial='du'">
   <xsl:text>Liebe</xsl:text>
   <xsl:if test="@geschlecht='m'">
    <xsl:text>r</xsl:text>
   </xsl:if>
  </xsl:when>
  <xsl:when test="@sozial='Sie'">
   <xsl:text>Herr</xsl:text>
  </xsl:when>
  <xsl:otherwise>
   <xsl:text>Sehr geehrte Damen und Herren</xsl:text>
  </xsl:otherwise>
 </xsl:choose>
</p>
</xsl:template>
Sehr geehrter Herr
Sehr geehrte Frau
Sehr geehrte Frau
Sehr geehrte Frau

<p><xsl:apply-templates/></p>
Liebe Nora,

habe gerade den Ulysses beendet. Mal sehen, wann der in den USA gedruckt werden darf...

J
I.5 XLink: XML linking language

- linking possible in any XML-DTD
  → no special linking elements

- linking via special attribute (for arbitrary elements):
  xml:link

- out-of-line links:
  link stored neither in source or in target document

- multiway links
**XLink terminology**

**resource:** addressable service or unit of information that participates in a link

**link:** explicit relationship between two or more resources

**locator:** data, provided as part of a link, which identifies a resource (attribute `HREF`)

**inline link:** link which serves as one of its own resources
  
  *e.g.* A *in HTML*

**out-of-line link:** link whose content does not serve as one of the link’s resources
Simple links

- one-directional
- mostly inline

<mylink xml:link="simple" title="Citation"
    href="http://www.xyz.com/xml/foo.xml"
    show="new" content-role="Reference">
    as discussed in Smith(1997)
</mylink>

<!ELEMENT mylink (#PCDATA)>
<!ATTLIST mylink
    xml:link CDATA #FIXED "simple"
    href CDATA #REQUIRED
    content-role CDATA #IMPLIED>
Extended links

usually out-of-line links

capabilities:

- enable outgoing links in read-only documents
- create links to and from resources in other formats
- applying and filtering sets of relevant links on demand
- enable other advanced hypermedia capabilities (e.g. via attribute ROLE)
example out-of-line extended link:

<commentary xml:link="extended" inline="false">
  <locator href="smith2.1" role="Essay"/>
  <locator href="jones1.4" role="Rebuttal"/>
  <locator href="robin3.2" role="Comparison"/>
</commentary>
Link behaviour

**SHOW** attribute:
describes display behaviour on traversal of link

- *embed*: designated resource embedded in body of current resource
- *replace*: designated resource replaces current resource
- *new*: designated resource displayed in a new window

**ACTUATE** attribute:
when should traversal of link occur?

- *auto*: retrieve resource when current resource is encountered
- *user*: present resource only upon request from user

all combinations of **SHOW** and **ACTUATE** values are possible!
I.6 Other XML Standards

- **Linking**: XPointer
  (for specifying arbitrary parts of XML documents as XLink locators)

- **Stylesheets**: XSL-FO

- **Documents**: DOM (Document Object Model), MathML, SVG (Scalable Vector Graphics), etc.
Part II

XML Query Languages
The meeting (or clashing) place of databases and IR:

- Requirements
- Query languages history
- XPath locator language
- XQuery query language
- XML Databases
II.1 Requirements (1)

- From Semistructured Data
  - Selection: pattern + filter + constructor
  - Filtering
  - Reduction: pruned elements
  - Restructuring: grouping, sorting, etc.
  - Combine data: joins and semi-joins
  - Vague queries
  - Navigation
  - Aggregation
  - Existential and universal quantifiers
  - Data types
  - Insert, delete, and update operations
Requirements (2)

- **From Information Retrieval**
  - Keyword queries: Boolean, context, similarity, etc.
  - Pattern matching
  - Structural queries: inclusion, distance relations, etc.
  - Weighting query terms
  - Ranking

- **Others**
  - Use of metadata
  - DTD or Xschema awareness
  - Support for XLink and XPointer
  - Set operations on results
II.2 XPath locator language

restricted XML query language
retrieves complete elements (subtrees) of XML documents

used in

**XSLT** (Extensible Style Sheet Language Transformations)
for specifying argument of a transformation

**XPointer** (XML Pointer)
for defining sources / targets of links

**XQuery** (XML Query Language)
for selecting elements that are arguments of further operations (value joins, restructuring, aggregation)
Path Expressions

- search for single elements:
  heading

- parent-child:
  chapter/heading

- ancestor-descendant:
  chapter//heading

- document root:
  /book/*

- filter wrt. structure:
  //chapter[heading]

- filter wrt content:
  /document[@class="H.3.3" and author="John Smith"]
Axes

generalization of locator operators

child:: children of the context node

descendant:: descendants of the context node

parent:: parent of the context node

ancestor:: ancestors of the context node

following-sibling:: all the following siblings of the context node

preceding-sibling:: all the preceding siblings of the context node
following:: all nodes in the same document as the context node that are after the context node in document order

preceding:: all nodes in the same document as the context node that are before the context node in document order,

attribute:: attributes of the context node

namespace:: namespace nodes of the context node

self:: just the context node itself

descendant-or-self:: context node and the descendants of the context node

ancestor-or-self:: context node and the ancestors of the context node
Axes

Model: Ordered set of nodes with attributes
XPath axes examples

- child::para para element children of the context node

- child::* element children of the context node

- child::text() text node children of the context node

- child::node() children of the context node, whatever their node type

- attribute::name name attribute of the context node

- attribute::* the attributes of the context node
• descendant::para para element descendants of the context node

• ancestor::div div ancestors of the context node

• ancestor-or-self::div div ancestors of the context node and, if the context node is a div element, the context node as well

• descendant-or-self::para para element descendants of the context node and, if the context node is a para element, the context node as well

• self::para context node if it is a para element, and otherwise selects nothing

• child::chapter/descendant::para para element descendants of the chapter element children of the context node
- child::*/child::para para grandchildren of the context node

- / document root (which is always the parent of the document element)

- /descendant::para para elements in the same document as the context node

- /descendant::olist/child::item item elements that have an olist parent and that are in the same document as the context node
II.3 XQuery

Weak data model:

- Ordered, labelled forest, with node identity and data types
- Static semantics: type inference, structural hierarchy
- Dynamic semantics: value inference
- Same data model as XPath 2.0
- Pure functional language with impure syntax
  - A query is an expression
  - Expressions can be nested
  - Basic structure:
    FOR PathExpression
    WHERE AdditionalSelectionCriteria
    RETURN ResultConstruction
Advantages

- Expressive power
- Easy to learn
- Easy to implement (?)
- Optimizable in many environments
- Related to concepts people already know
- Several current implementations
- The accepted W3C XML Query Language
Expressions

- Element constructors
- Path expressions
- Restructuring
  - FLWOR expressions
  - Conditional expressions
  - Quantified expressions
- Operators and functions
- List constructors
- Expressions that test or modify data-types
Element Constructors

Element constructors look like the XML they construct

```xml
<book year="1994">
  <title>TCP/IP Illustrated</title>
  <author>
    <last>Stevens</last>
    <first>W.</first>
  </author>
  <publisher>Addison-Wesley</publisher>
  <price>65.95</price>
</book>
```
Element Constructors: Examples

Generate an `<emp>` element that has an "empid" attribute and nested `<name>` and `<job>` elements.

```xml
<emp empid = "12345">
    <name>John Smith</name>
    <job>Anthropologist</job>
</emp>
```

Generate an `<emp>` element that has an "empid" attribute. The value of the attribute and the content of the element are specified by variables that are bound in other parts of the query.

```xml
<emp empid = {$id}>
    {$name}
    {$job}
</emp>
```
Path Expressions

<br />

<-- XQuery uses the abbreviated syntax of XPath for path expressions -->

document("bib.xml")

/bib/book/author

/bib/book//*

//author[last="Stevens" and first="W."]

document("bib.xml")//author
Path Expressions: Extensions

--- precedes, follows ---
//book[ author[last="Stevens"] precedes author[last="Abiteboul"] ]

--- Namespaces ---
namespace rev = "www.reviews.com"
//rev:rating

--- Dereference ---
//publisher/web/@href->html
Path Expressions: Examples

In the second chapter of the document named "zoo.xml", find the figure(s) with caption "Tree Frogs".

document("zoo.xml")//chapter[2]//figure[caption = "Tree Frogs"]

Find all the figures in chapters 2 through 5 of the document named "zoo.xml."

document("zoo.xml")//chapter[2 TO 5]//figure

Find captions of figures that are referenced by <figref> elements in the chapter of "zoo.xml" with title "Frogs".

document("zoo.xml")//chapter[title = "Frogs"]//figref/@refid=>fig/caption
XQuery Examples (1)

List the names of the second-level managers of all employees whose rating is "Poor".

```
//emp[rating = "Poor"]/@mgr=>emp/@mgr=>emp/name
```

Find all captions of figures and tables in the chapter of "zoo.xml" with title "Monkeys".

```
document("zoo.xml")//chapter[title = "Monkeys"]
//(figure | table)/caption
```

From a document that contains employees and their monthly salaries, extract the annual salary of the employee named "Fred".

```
//emp[name="Fred"]/salary * 12
```
FLWOR Expressions

FOR - LET - WHERE - ORDER BY - RETURN

Similar to SQL’s SELECT - FROM - WHERE

for $book in document("bib.xml")//book
where $book/publisher = "Addison-Wesley"
return
  <book>
    {
      $book/title,
      $book/author
    }
  </book>
FOR vs. LET

FOR iterates on a sequence, binds a variable to each node

LET binds a variable to a sequence as a whole

let $a := $book/author
where contains($book/publisher, "Addison-Wesley")
return
  <book>
    {
      $book/title,
      <count> Number of authors: { count($a) } </count>
    }
  </book>
Conditional Expressions

IF expr THEN expr ELSE expr
FOR $h$ IN //holding
RETURN

<holding>
{ $h/title,
  IF ($h/@type = "Journal")
    THEN $h/editor
    ELSE $h/author
  }
</holding>

Sorted Expressions:

expr SORTBY (expr ASCENDING , ... )

FOR $b$ IN //book
RETURN

$b$ SORTBY(title, author[1]/name)
Inner and Outer Joins

FOR $book IN document("www.bib.com/bib.xml")//book,
    $quote IN document("www.bookstore.com/quotes.xml")//listing
WHERE $book/isbn = $quote/isbn
RETURN
    <book>
        { $book/title }
        { $quote/price }
    </book>
SORTBY (title)

FOR $book IN document("bib.xml")//book
RETURN
    <book>
        { $book/title }
        {
            FOR $review IN document("reviews.xml")//review
                WHERE $book/isbn = $review/isbn
                RETURN $review/rating
         }
    </book>
SORTBY (title)
Quantifiers

EVERY var IN expr SATISFIES expr

SOME var IN expr SATISFIES expr

FOR $b$ IN //book
WHERE SOME $p$ IN $b$//para SATISFIES
contains($p$, "sailing")
 AND contains($p$, "windsurfing")
RETURN $b$/title

FOR $b$ IN //book
WHERE EVERY $p$ IN $b$//para SATISFIES
contains($p$, "sailing")
 AND contains($p$, "windsurfing")
RETURN $b$/title
FLWOR: Data for Examples

<book>
  <title>XML: An Introduction</title>
  <author>Smith</author>  <author>Miller</author>
  <publisher>Morgan Kaufmann</publisher>
  <year>1998</year>
  <price>50</price>
</book>

<book>
  <title>XSLT Course</title>
  <author>Jones</author>
  <publisher>Addison Wesley</publisher>
  <year>2000</year>
  <price>40</price>
</book>
XQuery Examples (2)

List the titles of books published by Morgan Kaufmann in 1998.

FOR $b$ IN document("bib.xml")//book
WHERE $b$/publisher = "Morgan Kaufmann"
AND $b$/year = "1998"
RETURN $b$/title

List each publisher and the average price of its books.

FOR $p$ IN distinct(document("bib.xml")//publisher)
LET $a$ := avg(document("bib.xml")//book[publisher = $p$]/price)
RETURN

  <publisher>
    <name> {$p$/text()} </name>
    <avgprice> {$a} </avgprice>
  </publisher>
XQuery Examples (3)

List the publishers who have published more than 100 books.

<big_publishers>
  {
    FOR $p$ IN distinct(document("bib.xml")//publisher)
    LET $b$ := document("bib.xml")//book[publisher = $p]
    WHERE count($b) > 100
    RETURN $p
  }
</big_publishers>
Invert the structure of the input document so that, instead of each book element containing a sequence of authors, each distinct author element contains a sequence of book-titles.

```
<author_list>
  {
    FOR $a$ IN distinct(document("bib.xml")//author)
    RETURN
      <author>
        <name> {$a/text()} </name>
        {
          FOR $b$ IN document("bib.xml")//book[author = $a]
          RETURN $b/title
        }
      </author>
  }
</author_list>
```
XQuery Examples (5)

For each book whose price is greater than the average price, return the title of the book and the amount by which the book’s price exceeds the average price.

<result>
{
  LET $a := avg(document("bib.xml")//book/price)
  FOR $b IN document("bib.xml")//book
  WHERE $b/price > $a
  RETURN
    <expensive_book>
      {$b/title}
      <price_difference>
        {$b/price - $a}
      </price_difference>
    </expensive_book>
}
</result>
XQuery Examples (6)

Construct a new element having the same name as the element bound to $e$. Transform all the attributes of $e$ into subelements, and all the subelements of $e$ into attributes.

```xml
{name($e)}
  {
    FOR $c IN $e/*
    RETURN attribute(name($c), string($c))
  }
  {
    FOR $a IN $e/@*
    RETURN
      <{name($a)}>
        {string($a)}
      </>
  }
</>
```
Conditions on Text

Equality:

//section[title="Procedure"]

Full-text:

//section[contains(title, "Procedure")]

More Full-text support in the future

Last published requirements: June 2003
Full-text Requirements

- Full-Text predicates and SCORE functions independently
- Full-Text predicates use a language subset of SCORE functions
- Allow the user to return and sort-by SCORE (float in 0-1)
- SCORE must not require explicit global corpus statistics
- SCORE algorithm should be provided and can be disabled
- **Minimal operations:**
  - single-word and phrase search with stopwords
  - suffixes, prefix, infix
  - proximity searching (with order)
  - Boolean operations
  - Word normalization, diacritics
  - Ranking, relevance
- **Search over everything, including attributes**
- **Proximity across markup elements**
- **Extensible**
Filters

Filter( expression )

Result is an "ordered forest" that preserves sequence and hierarchy
Functions

- Built-in functions: `max()`, `min()`, `sum()`, `count()`, `avg()` `distinct()`, `empty()`, `contains()`
- User-defined functions
- Defined in XQuery syntax
- May be recursive
- May be typed
- Extensibility mechanisms planned

Example:

```xml
define function depth(element $e) returns integer {
    <!-- An empty element has depth 1
    Otherwise, add 1 to max depth of children -->
    if (empty($e/*))
        then 1
        else max(depth($e/*)) + 1
}

depth(document("partlist.xml"))
```
Possible Future Extensions

Insert-replace-delete:

INSERT
UPDATE
FOR $e IN /emp
INSERT <n_skills>{ count($e/skill) }</n_skills>
BEFORE $e/skill[1]

REPLACE
UPDATE
FOR $e IN /emp
WHERE $e/empno = "12345"
REPLACE $e/job
WITH <job> Broom Tester </job>

UPDATE
FOR $e IN /emp/[job = "Programmer"],
   $s IN $e/skill
WHERE $s/rating < 4
   OR $s/cert_date < date("1995-01-01")
DELETE $s

Further support for full-text retrieval
XQuery Implementations

- Software AG's QuiP
- Microsoft demo
- Lucent Galax
- FhG-IPSI
- X-Hive
- XML Global
- SourceForge XQuench
- Fatdog
- Qexo (GNU Kawa) - compiles to Java byte code
- Openlink, CL-XML (Common Lisp), Kweelt, ...
- Soda3, DB4XML and about ten more
II.4 XML Databases

- XML as exchange protocol in heterogeneous systems
- Moving and sharing data
- Data-centric vs. document-centric
- Publish XML: old data
- Store XML: native format or not
- Querying: just in XML!
- Hard, but solvable problems
Product Categories

- Middleware: data-centric
- XML-Enabled databases: mostly data-centric
- Native XML Databases: data and document-centric
  Examples: eXist, Lucid, Ozone, Tamino, Virtuoso, XIndice, XYZFind, ...
- XML Servers or Query Engines: both
- Content Management Systems: document-centric
- XML Data-binding: data-centric
- Benchmarks: XOO7, XMach-1, XMark, Michigan, XSLT Processor
- XML Generators: ToXgene, Niagara, IBM
- XML Validators, parsers, authoring, etc.