

XML in Databases

Albrecht Schmidt

al@cs.auc.dk

`http://www.cs.auc.dk/~al`

What is XML? (1)

Where is the Life we have lost in living?

Where is the wisdom we have lost in knowledge?

Where is the knowledge we have lost in information?

-- T. S. Elliot

Where is all the information we have lost in data structures?

-- NOT T. S. Elliot

This session is about the problem printed in red and about a way to tackle it.

- We take a look at self-describing data -- data stored in a syntax called XML.
- We also look at how we can process these data.

Overview of Session

This session will cover the following topics:

- What is XML?
- Why XML? Data integration, data exchange
- XML syntax -- angle brackets & more
- XML semantics compared (?) to the relational model
- Viewing relational data as XML
- Native XML storage
- Path expressions as the basis of XML query languages
- XQuery and XPath by example
- Summary and Conclusion

I will provide you with a couple of questions (hand-in) at the end of the session.

What is XML? (2)

Many database products provide some support for XML

- <http://www.ibm.com/xml>
- <http://www.microsoft.com/xml>
- <http://www.oracle.com/xml>
- <http://www.sleepycat.com/products/xml.shtml>
- . . . and many more . . .

Is XML just another data source?

XML enables people to share data on the Web by providing uniform access and well-defined semantics.

XML is also some kind of social engineering.

<http://www.w3.org/XML/> for more information on XML standards

What is XML? (3)

XML stands for eXtensible Markup Language

History:

- A simpler SGML -- a flexible language widely used by publishers . . .
- . . . but one which is technically hard to understand and handle.
- Since then: a format for sharing data between machines (and humans).
- Now widely used for data exchange on the Web and between applications

Application areas:

- Sharing data between different components of an application.
- Archive data in text files.
- EDI (Electronic Data Interchange): transactions between banks, producer-supplier chains, weather data, . . .
- Drive towards standards: building relationships between companies and institutions.
- Scientists share experimental data.

What does XML look like? (1)

```
<bibliography>
  <book ISBN='0-13-031995-3' rating='excellent'>
    <title> Database Systems -- The Complete Book </title>
    <author> Garcia-Molina </author>
    <author> Ullman </author>
    <author> Widom </author>
  </book>
  <book>
    <title> Foundations of Databases </title>
    <author> Abiteboul </author>
    <author> Hull </author>
    <author> Vianu </author>
  </book>
</bibliography>
```

What does XML look like? (2)

Tags: bibliography, book, title, author

Start tags (`<book>`) and end tags (`</book>`) must correspond.

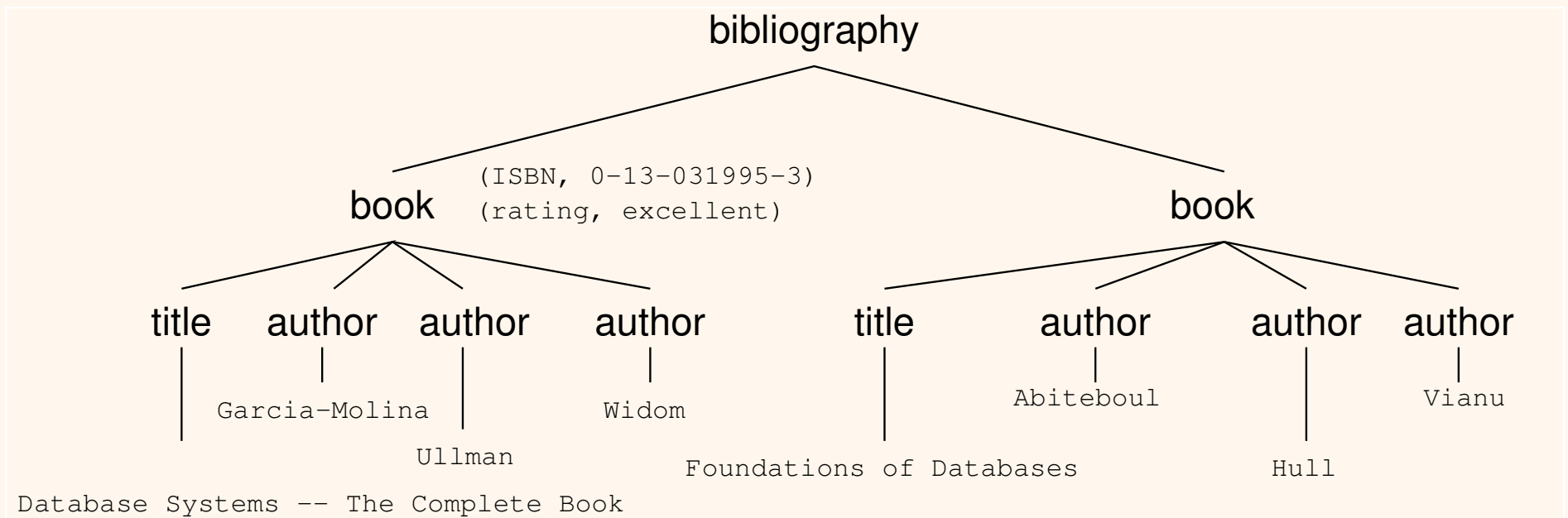
Attributes (`ISBN='0-13-031995-3' rating='excellent'`) are association lists local to elements.

XML elements: everything between two corresponding tags.

- `<title> Database Systems -- The Complete Book </title>`
- Question: How many elements does the following fragment contain?
`<book ISBN='0-13-031995-3' rating='excellent'>`
 `<title> Database Systems -- The Complete Book </title>`
 `<author> Garcia-Molina </author>`
 `<author> Ullman </author>`
 `<author> Widom </author>`
`</book>`

What does XML look like? (3)

Often XML documents are viewed as trees:



What does XML look like? (3)

Elements may be nested (→ recursive structure).

Empty element: `<ISBN></ISBN>` → abbreviated as `<ISBN/>`

An XML document has one unique root element (and, thus, is a tree).

Attributes are sometimes alternatives to non-structured Elements:

- The book record from the example could also look like this:

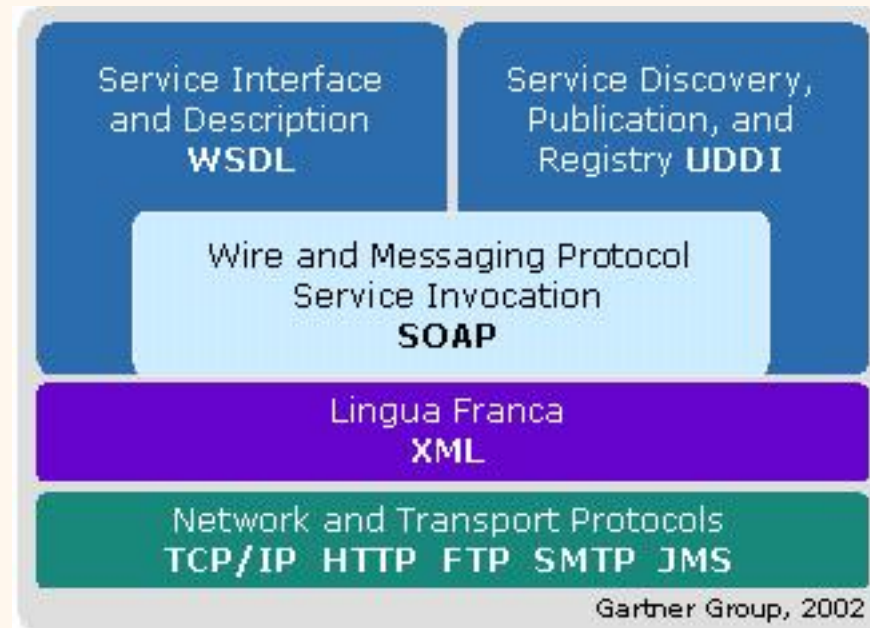
```
<book>
  <ISBN> 0-13-031995-3 </ISBN>
  <rating> excellent </rating>
  <title> Database Systems -- The Complete Book </title>
  <author> Garcia-Molina </author>
  <author> Ullman </author>
  <author> Widom </author>
</book>
```

What does XML look like? (4)

Many people distinguish between two kinds of XML documents:

- data-oriented like the bibliography example
 - ▷ *very structured*
 - ▷ *similar to relational tables*
- document-oriented
 - ▷ *more like natural language with markup interspersed*
 - ▷ `<document>`
 - `<chapter>`
 - `<emphasize>` This chapter `</emphasize>` introduces the notion of
 - `<keyword>` document-centric `</keyword>` XML documents.
 - We do not like to see document-centric documents in relational databases `<emphasize>` for a number of reasons `</emphasize>`:
 - `<list>`
 - `<listitem>` They are `<emphasize>` irregular `</emphasize>` `</listitem>`
 - `<listitem>` They are `<important>` irregular `</important>` `</listitem>`
 - `<listitem>` They are `<keyword>` irregular `</keyword>` `</listitem>`
 - `</list>`
 - `</chapter>` . . .
 - `</document>`

What Is XML Used for? Why Share Data (1)



XML is **not** a wire format like TCP/IP.

It provides abstraction by separating data from programs.

What Is XML Used for? (2)

XML standards exist for

- Parsing (World Wide Web Consortium, W3C)
- Schemas
- APIs
- . . . numerous other standards by third parties (geography, biology, business, . . .)

Benefits of using XML in B2B data integration (after a survey by Gartner Group):

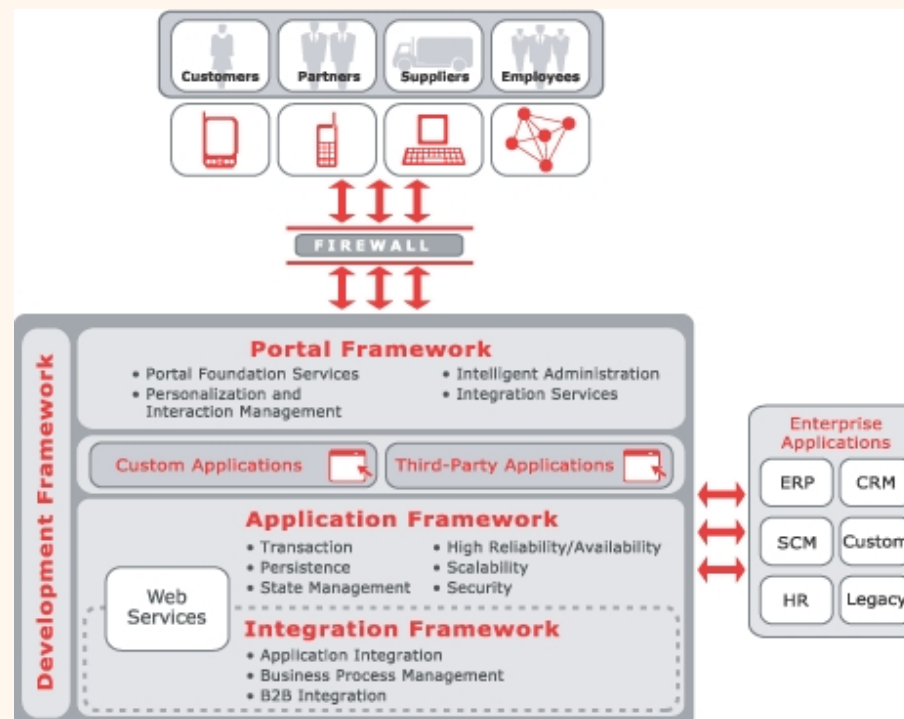
- Increase business agility and flexibility
- Enhance customer relationships
- Improve business efficiency and decision-making
- Reduce integration cost and complexity

What Is XML Used for? (3)

Example: B2B Data Integration, Web Services

Example architecture

- BEA WebLogic Enterprise Platform



What Is XML Used for? (4)

XML is one technology for information integration.

Does it help to build a paradigm for distributed computing?

- 'Independent' system components communicate via messages
- Publishing of information
- Definition of APIs

Example: Customer-Relationship Management (CRM)

Microsoft's .NET platform

J2EE, Enterprise Java Beans for businesses

Many other standards exist as well.

Constraints on XML Documents (1)

Some obvious constraints:

- Syntactical constraints.
- Element tags must match, Attribute names be unique, . . .

To achieve interoperability, constraints are useful:

- provide semantics and context
- enable queries
- integrity checks

Two categories of constraint specification languages:

- DTD -- Document Type Definition
 - ▷ *define a (mostly) context-free grammar for XML documents*
 - ▷ *often used as substitute for types or schemas*
- XSchema
 - much more sophisticated: more datatypes, constraints, extensible

Constraints on XML Documents (2)

A DTD for the example document:

- Elements:
 - <!ELEMENT bibliography (book*)>
 - <!ELEMENT book (title,author*)>
 - <!ELEMENT title (#PCDATA)>
 - <!ELEMENT author (#PCDATA)>
- Attributes:
 - <!ATTLIST book
 - ISBN CDATA #IMPLIED
 - rating CDATA #IMPLIED>
- Attributes are one of:
 - ▷ *#REQUIRED -- must be present in the document*
 - ▷ *#IMPLIED -- the application should know*
 - ▷ *#FIXED -- for example, unit must be metric*
- Attribute type: CDATA (string), ID (key), IDREF (foreign key) and some more
- Attributes for invisible content?

Constraints on XML Documents (3)

A DTD as a context free grammar (well, not quite but close):

- Elements:
 - bibliography → book*
 - book → title author*
 - title → STRING
 - author → STRING
- What to do with Attributes?

Think of the XML tree as a parse/derivation tree.

Grammar is a pseudo EBNF grammar.

Grammar may contain recursive productions.

DTDs define Content Models.

Constraints on XML Documents (4)

XSchema has the following features:

- Generalizes DTDs.
- Uses XML syntax (unlike DTDs) -- hence can be queried.
- Large and complex standard.
- Defines constraints on document structure and datatypes.

Example snippet:

- XML Schema definition for a book in a bibliography:

```
<xsd:element name='book' type='booktype'>
  <xsd:complexType name='booktype'>
    <xsd:sequence>
      <xsd:element name='title' type='xsd:string' />
      <xsd:element name='author' type='xsd:string' minOccurs='1' />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Constraints on XML Documents (5)

More features of XML Schema:

- Simple Types: string, integer, long, short, time, date, ID, IDREF
 - ▷ *additionally: facets of Simple Types*
- Complex Types: user-defined
 - ▷ *scoping*
 - ▷ *derived types, restrictions*
 - ▷ *regular expressions (cf. DTDs)*
- Expressiveness
 - ▷ *Regular tree languages (over unranked trees)*
- Many other interesting things:
 - ▷ *inheritance*
 - ▷ *pre-defined data types*
 - ▷ *'NULL' values*
- Object-oriented feel
- Emphasis on data exchange

XML vs Semantic Data Models

XML is self-describing

- Schema information is part of the data
- As a consequence, XML offers more flexibility
- XML is an example of semistructured data.
- Parts of a document may be 'missing' or have complex structure.
- Still, documents do have a context.

Semi-structured data -- good or bad?

- ER models -- we are in control
- XML -- the data/community is in control
- Centralised processing vs distributed processing, data sharing
- Internet, World Wide Web (SOAP services)
- Politics play an important role.

XML does not fall into the same category as ER Models

Publishing of Relational Data as XML (1)

Suppose you are given a relation and want as XML.

- Person (CPR, Name, Address, Birthday)
- Example instance:

101075-1234	Happy Hacker	10 XML Avenue	10/10/75
111170-9876	Ben Bit	10 Lisp Road	11/11/70

- Possible markup:

```
<relation name='Person'>
```

```
  <tuple>
```

```
    <attribute name='CPR'> 101075-1234 </attribute>
```

```
    <attribute name='Name'> Happy Hacker </attribute>
```

```
    <attribute name='Address'> 10 XML Avenue </attribute>
```

```
    <attribute name='Birthday'> 10/10/75 </attribute>
```

```
  </tuple>
```

```
  . . .
```

```
</relation>
```

- Is this a nested data model? Or is it just text . . .

Publishing of Relational Data as XML (2)

Good or bad markup?

- What would an ER diagram for this look like?

Another possibility:

- What is the difference?

```
<Person>
  <tuple>
    <CPR> 101075-1234 </CPR>
    <Name> Happy Hacker </Name>
    <Address> 10 XML Avenue </Address>
    <Birthday> 10/10/75 </Birthday>
  </tuple>
  . . .
</Person>
```

- Can you think of even more ways?
- What knowledge is required to understand the data?

Storing XML Documents in an RDBMS (1)

Several approaches exist

- Is schema information available?
- Derive mapping from a DTD
- Use XSchema to derive relation schema (always possible?)
- Specify mapping by hand
- application specific vs general mapping

Approaches fall in two categories:

- adaptive mappings -- imitate semantic modelling
- static mappings -- one translation algorithm for all documents

General ideas:

- Associate XML Elements with 'entities' and try to map them to a table
- Inline XML Attributes and contents (sub-elements) when possible

Storing XML Documents in an RDBMS (2)

Some approaches for physical mappings:

- Use of big table to store everything:
 - ▷ *Node (ID, ParentID, type, value)*
 - ▷ *Very simple to implement.*
 - ▷ *Do we need a DTD or even more information for this?*
- Create one table for every Element/Attribute.
 - ▷ *Bibliography (DocID, Root)*
 - ▷ *Book (Parent, Child)*
 - ▷ *Title (Parent, Child)*
 - ▷ *Author (Parent, Child)*
 - ▷ *Strings (Parent, Value)*
 - ▷ *...*
- Inlining: Try to combine several tables into one.
 - ▷ *When is this possible?*
 - ▷ *What are the advantages and disadvantages?*
 - ▷ *Is this like deriving a schema from an ER diagram?*

Storing XML Documents in an RDBMS (3)

Some things to think about:

- What about textual order?
- What about updates?
- Our examples were very much SQL like . . .
 - ▷ *What if we have real documents (prose-like, document-centric)?*
 - ▷ *What about containment?*
 - ▷ *What about Information Retrieval?*
- How to decide which storage schema is best?
- How can we reconstruct the original document from the relations? Is this always possible?

In general:

- Is it a good idea to store XML in a RDBMS?
- What are the pros and cons?
- Think of an application scenario, transactions, system engineering, . . .

Querying XML Documents

It XML data are stored in an RDBMS:

- Use SQL with some additional user-defined functions.

To work directly on XML documents:

- XPath -- basic, re-usable functionality
- XQuery -- full blown query language ('includes' XPath)
- XSLT -- stylesheet language

Why query XML documents?

- One query language for everything
- Data extraction and integration
- Are there alternatives?

Querying XML Documents: Path Expressions (1)

Path expressions are an integral part of XML query languages

- Basic idea: parent-child sequence of tags can be used to identify interesting parts of the document tree.
- Allow wildcards as short cuts
- Define more sophisticated predicates

Examples:

- `bibliography/book/author`
 - ▷ `<author> Garcia-Molina </author>`
`<author> Ullman </author>`
`<author> Widom </author>`
`<author> Abiteboul </author>`
`<author> Hull </author>`
`<author> Vianu </author>`

Querying XML Documents: Path Expressions (2)

More examples:

- bibliography/book/title
 - ▷ `<title> Database Systems -- The Complete Book </title>`
`<title> Foundations of Databases </title>`
- bibliography/book
 - ▷ `<book ISBN='0-13-031995-3' rating='excellent'>`
`<title> Database Systems -- The Complete Book </title>`
`<author> Garcia-Molina </author>`
`<author> Ullman </author>`
`<author> Widom </author>`
`</book>`
`<book>`
`<title> Foundations of Databases </title>`
`<author> Abiteboul </author>`
`<author> Hull </author>`
`<author> Vianu </author>`
`</book>`

Querying XML Documents: Path Expressions (3)

Path expressions navigate along 13 axes relative to context nodes:

- child
- descendant
- descendant-or-self
- parent
- ancestor
- ancestor-or-self
- following-sibling
- preceding-sibling
- following
- preceding
- attribute
- namespace
- self

Querying XML Documents: Path Expressions (4)

Still more examples:

- `/bibliography/book/title/text()`
 - ▷ `/child::bibliography/child::book/child::title/child::text()`
 - ▷ *Database Systems -- The Complete Book*
Foundations of Databases
- `/bibliography/book/@rating`
 - ▷ `/child::bibliography/child::book/attribute::rating`
 - ▷ *excellent*
- `/bibliography/book | /bibliography/article`
 - ▷ *logical or*
- `/bibliography/*`
 - ▷ *matches any element that is a child of /bibliography*
- `//book`
 - ▷ *matches any book element in the same document as context node*

Querying XML Documents: Path Expressions (5)

More examples of path expressions:

path expression	matches
bibliography	
*	any element
/	
/bibliography	
bibliography/paper	
bibliography//paper	a paper under a bibliography
//paper	a paper at any depth
article book	an article or a book
@rating	a rating attribute
bibliography/book/@rating	
bibliography/book/[@rating='excellent']/author	

Mathematically, a path expression p is a function

- $p : Element \rightarrow \{Element\}$
- p maps a context node to an answer set

Querying XML Documents: XQuery (1)

XQuery is a (draft) W3C Standard supported by major database vendors. Think of it as XPath++.

Based on FLWR expressions (*flower* expressions)

- **F**or: create a stream (list) of elements
- **L**et: creates a local variable (why is this convenient?)
- **W**here: defines a filter predicate
- **R**eturn: constructs a value to be returned for every variable

Example query:

- Find the titles of all books (co-)authored by Ullman:

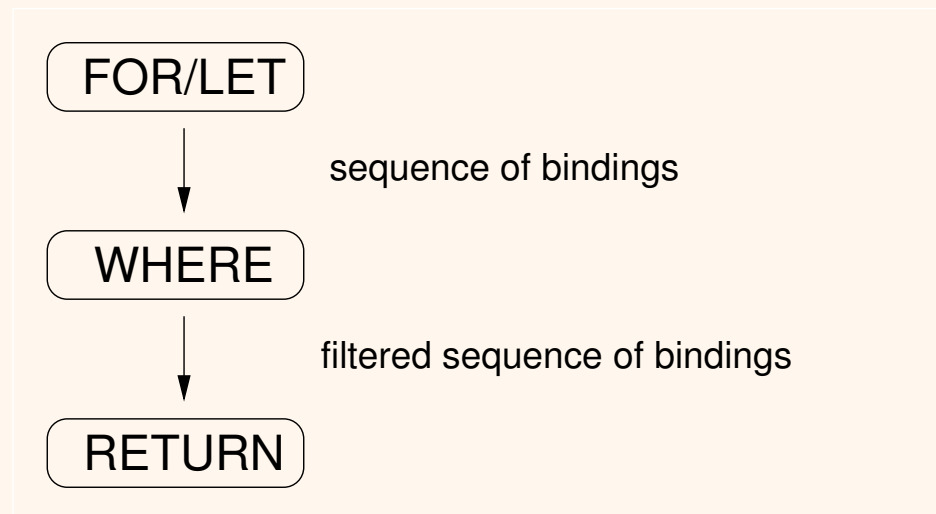
```
<result> {  
  FOR $b IN document("bibliography.xml")//book  
  WHERE $b/author/data() = 'Ullman'  
  RETURN $b/title  
} </result>
```

Querying XML Documents: XQuery (2)

Result of query on previous slide:

- `<result>`
 `<title> Database Systems -- The Complete Book </title>`
`<result>`

Data flow in XQuery:



Querying XML Documents: XQuery (3)

How does the data flow compare to the Relational Algebra/SQL?

More features of XQuery:

- Aggregations
- Collections (remember the result type of path expressions?)
- For \$e in DISTINCT (expr): ordered vs unordered collections
- Existential and universal quantification (SOME/EVERY)
- Sorting of results
- IF . . . THEN . . . ELSE . . .
- Recursive functions
- BEFORE and AFTER for querying order
- FILTER for removing edges from the input
- User-defined functions
- . . .

Summary

The eXtensible Markup Language

- encourages standardisation.
- is an important data exchange format.
- is a standard technology for structuring documents.
- encourages separation of programs and data.

Relational Databases Management Systems

- can be used to store XML . . .
- . . . but it depends on the scenario whether this is a good idea.

Path Expressions

- are the backbone of XML query languages.
- are based on the idea of navigating through the XML tree.

Literature and Links

<http://www.w3c.org/XML> for the standards

Garcia-Molina et. al., Database Systems -- The Complete Book

- Chapters 4.6 and 4.7
- Also look at links on book Web page.

<http://www.xml.com>

<http://www.rpbouret.com>

Abiteboul et. al, Data on the Web, Morgan Kaufmann

. . . much more to be found on the Web . . .

Thank You.

Some points to think about . . .

Can you remember the following terms?

- Element, tag, attribute, PCDATA, DTD, path expression, semi-structured data, FLWR expression, . . .

Why do people want to use XML?

- Structure your answer and mark it up in XML :-)
- `<answer> <list> <item> . . . </item> . . . </list> </answer>`
- Write some path expressions that extract interesting information from your answer.
- 4.7.1 -- 4.7.4 in Database Systems -- The Complete Book

Why is it harder to convert XML to relations than vice versa?

- For example, start out by considering ER diagrams . . .
- Why is XML **not** a datamodel like the ER model?

How does XQuery compare to SQL?

- What about types and values? What about paths/path expressions?