Applied Databases

Lecture 1
Introduction, Basics of XML

Sebastian Maneth

Univeristy of Edinburgh - January 16th, 2017

Applied Databases

→ Apply database technology (e.g. MySQL) in varying contexts

→ Together with other technologies:

- XML
- Lucene (full-text search)
- RDF

Applied Databases

→ Apply database technology (e.g. MySQL) in varying contexts

→ Together with other technologies:

- XML
- Lucene (full-text search)
- RDF

WARNING Course Catalogue mentions

→ Similarity Search
→ Data Analytics

Unfortunately, these will **NOT** be covered this year

Course Organization

Lectures	Monday 14:10–15:00 G.07, Medical School Thursday 14:10–15:00 Lecture Theatre 2, Appleton Tower

---

Lecturer	Sebastian Maneth (smaneth@inf.ed.ac.uk)
TA	Fabian Peternek

Assessment	Exam (60%) Assignment 1 (20%) due 17th February, 4:00pm Assignment 2 (20%) due 24th March, 4:00pm

Course Format

20 Lectures	All material covered in the lectures is examinable
Assignments	Lectures 1–12 cover material relevant to the Assignments

Assignments

→ taken, with consent and warm thanks,
from UCLA lecture “CS144: Web Applications”

Assignments 1 & 2

- Programming assignments, in Java & SQL

- Pair programming:
  you are allowed to program in pairs of two persons

Rules:
→ either alone or with partner
→ may change partner for 2nd assignment
→ submit **one** solution
→ same mark for both in the team

Assignment 1

1) design a relational schema for EBAY data
2) convert EBAY data from XML into relational tables (csv files)
3) import csv files into a MySQL database
4) execute some SQL queries over the database

Assignment 1

1) design a relational schema for EBAY data
2) convert EBAY data from XML into relational tables (csv files)
3) import csv files into a MySQL database
4) execute some SQL queries over the database

Requires

- XML parsing (DTDs, DOM, SAX)
- basic DB knowledge (schema design, basic SQL queries)

Assignment 1

1) design a relational schema for EBAY data
2) convert EBAY data from XML into relational tables (csv files)
3) import csv files into a MySQL database
4) execute some SQL queries over the database

Lectures 1 – 4

Requires

- XML parsing (DTDs, DOM, SAX)
- basic DB knowledge (schema design, basic SQL queries)

Lectures 5 – 8

Assignment 1

Pair programming

→ together design database schema

→ individually write load functions for different tables

Ideally together find abstractions that
make the code *small*, *elegant*, and *readable*

Assignment 2

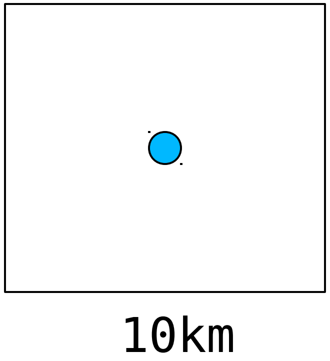
1) create a Lucene full-text Index (from Java)

2) implement a basic keyword search function

3) build a spatial index in MySQL

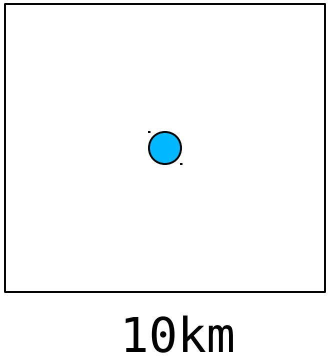
4) implement spatial search

5) create web interface for keyword & spatial search and for display of results



Assignment 2

1) create a Lucene full-text Index (from Java)
2) implement a basic keyword search function
3) build a spatial index in MySQL
4) implement spatial search
5) create web interface for keyword & spatial search and for display of results



Requires

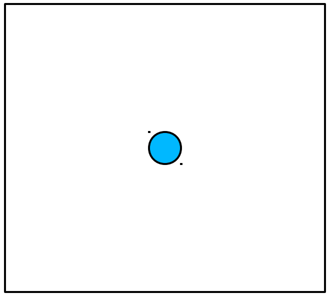
Lecture 9

- spatial search
- basic knowledge of Lucene / text-indexing

Lectures 10–12

Assignment 2

1) create a Lucene full-text Index (from Java)
2) implement a basic keyword search function
3) build a spatial index in MySQL
4) implement spatial search
5) create web interface for keyword & spatial search and for display of results



10km

Assignments 1 & 2

→ hands-on experience to implement a web store such as EBAY or similar!

Applied Databases

Main Topics

→ XML	**Lectures 1 – 4**
→ DB schema design, SQL	**Lectures 5 – 8**
→ Lucene	**Lectures 9 – 12**
→ String Matching	**Lectures 13 – 16**
→ XPath, XSLT, RDF, SPARQL	**Lectures 17 – 19**

Lecture 1

Basics of XML

Outline

1. Motivations for XML
2. Well-formed XML
3. Parsing / DTD Validation: Introduction

XML

- → Similar to HTML (Berners-Lee, CERN → W3C)
  use your own tags

- → XML is the de-facto standard for data exchange on the web

1. XML

Motivation

to have one language to speak about data

1. XML Motivation

→ XML is a **Data Exchange Format**

1974 SGML Standardized Generalized Markup Language (Charles Goldfarb at IBM Research)

1989 HTML (Tim Berners-Lee at CERN/Geneva)

1994 Berners-Lee founds Web Consortium (W3C)

1996 **XML** (W3C draft, v1.0 in 1998)

→ **http://www.w3.org/TR/REC-xml/**

XML = data

```
Philip Wadler
U. of Edinburgh
wadler@inf.ed.ac.uk

…


Helmut Seidl
TU Munich
seidl@inf.tum.de
```

Text file

XML = data **+ structure (mark-up)**

```
Philip Wadler
U. of Edinburgh
wadler@inf.ed.ac.uk

…

Helmut Seidl
TU Munich
seidl@inf.tum.de
```

“mark it up!” →

```
<Related>
<colleague>
<name>Philip Wadler</name>
<affil>U. of Edinburgh</affil>
<email>wadler@inf.ed.ac.uk
</email>
</colleague>
…
<friend>
<name>Helmut Seidl</name>
<affil>TU Munich</affil>
<email>seidl@inf.tum.de
</email>
</friend>
</Related>
```

Text file

XML document

XML = data + structure (mark-up)

```
Philip Wadler
U. of Edinburgh
wadler@inf.ed.ac.uk

…


Helmut Seidl
TU Munich
seidl@inf.tum.de
```

“mark it up!” →

```
<Related>
<colleague>
<name>Philip Wadler</name>
<affil>U. of Edinburgh</affil>
<email>wadler@inf.ed.ac.uk
</email>
</colleague>
…
<friend>
<name>Helmut Seidl</name>
<affil>TU Munich</affil>
<email>seidl@inf.tum.de
</email>
</friend>
</Related>
```

Text file

XML document

Is this a “good” structure?

XML Documents

- → Ordinary text files (UTF-8, UTF-16, US-ASCII …)
- → Originates from typesetting/DocProcessing community
- → Idea of labeled brackets (“mark up”) for structure is not new!
  (already used by Chomsky in the 1960’s)
- → Brackets describe a tree structure
- → Allows applications from different vendors to exchange data!

→ standardized, extremely widely accepted!

XML Documents

- → Ordinary text files (UTF-8, UTF-16, US-ASCII …)
- → Originates from typesetting/DocProcessing community
- → Idea of labeled brackets ("mark up") for structure is not new!
  (already used by Chomsky in the 1960's)
- → Brackets describe a tree structure
- → Allows applications from different vendors to exchange data!

→ **standardized, extremely widely accepted!**

↘ Social Implications!
All sciences (biology, geography, meteorology, astrology…)
have own XML "dialects" to exchange *their* data optimally

XML Documents

→ Ordinary text files (UTF-8, UTF-16, US-ASCII …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets (“mark up”) for structure is not new!
(already used by Chomsky in the 1960’s)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

→ **standardized, extremely widely accepted!**

Problem highly verbose, lots of repetitive markup

XML Documents

→ Ordinary text files (UTF-8, UTF-16, US-ASCII …)

→ Originates from typesetting/DocProcessing community

→ Idea of labeled brackets ("mark up") for structure is not new!
(already used by Chomsky in the 1960's)

→ Brackets describe a tree structure

→ Allows applications from different vendors to exchange data!

→ **standardized, extremely widely accepted!**

Contra.. highly verbose, lots of repetitive markup

Pro.. we have a standard! A STANDARD!
→ ☺ *You never need to write a parser again! Use XML!* ☺

XML: Validation & Parsing

… instead of writing a parser, you simply fix your own "XML dialect",

by describing all "admissible structures" (+ maybe even the specific

data types that may appear inside).

You do this, using an *XML Type definition language* such as DTD, XML Schema, or Relax NG.

→ *type definition languages* must be SIMPLE, because you want the parsers to be efficient!

They are similar to EBNF → context-free grammar with reg. expr's in the right-hand sides. ☺

XML Documents

Example **DTD** (Document Type Description)

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content

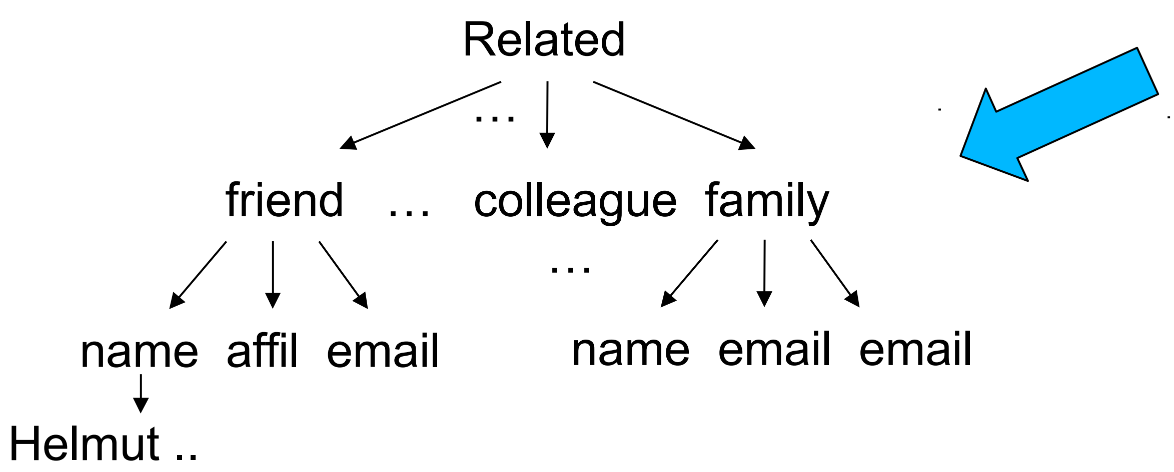
XML Documents

Example DTD (Document Type Description)

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content

ordered, unranked tree

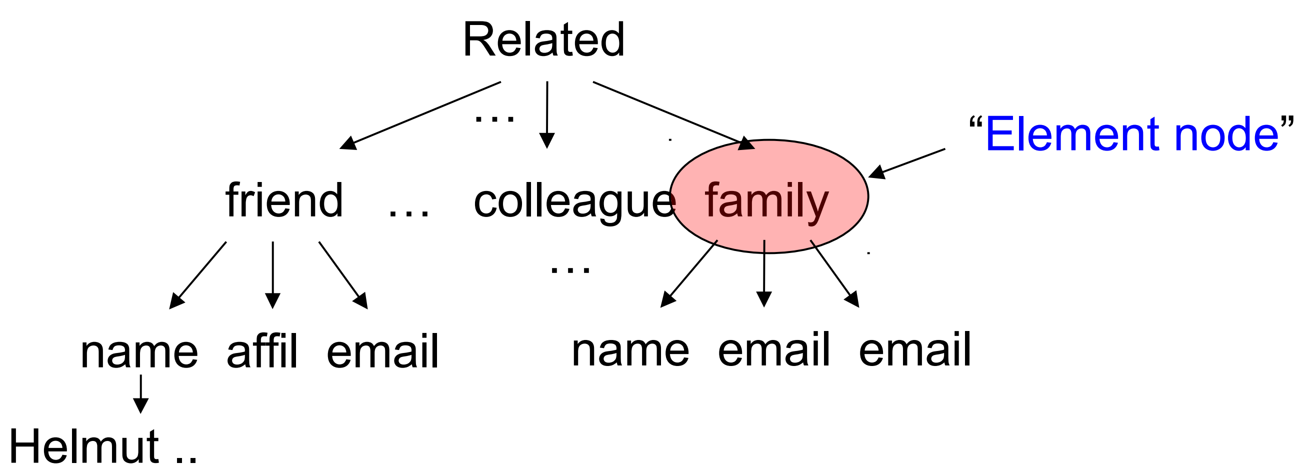


XML Documents

Example DTD

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content

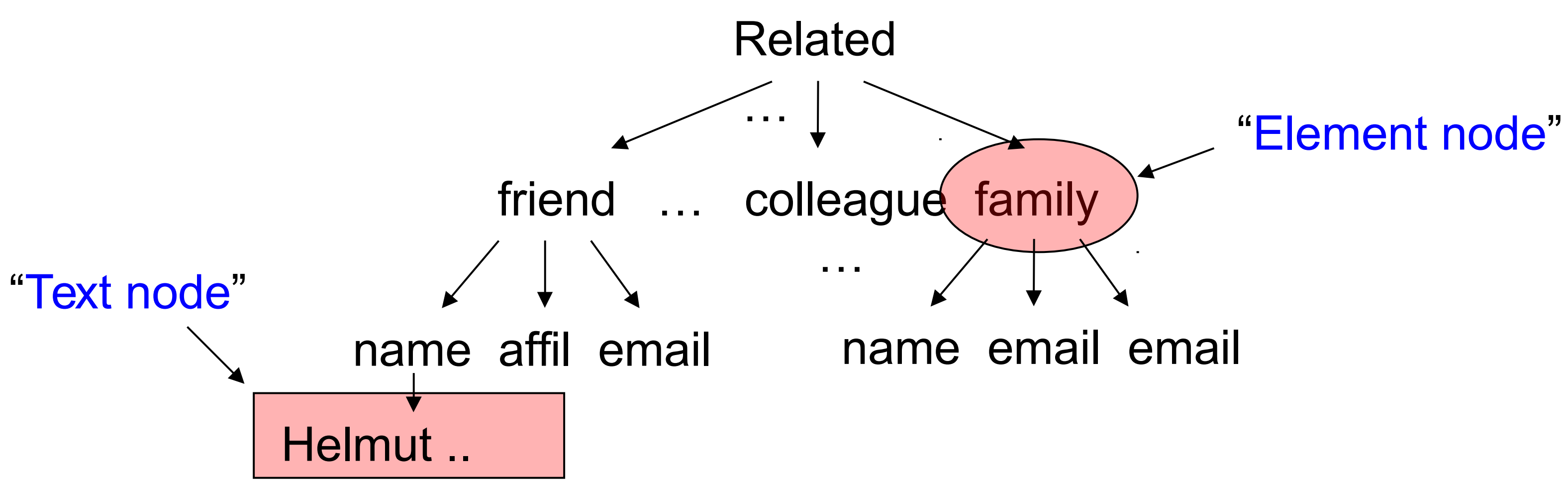


XML Documents

Example DTD

```
Related     → (colleague | friend | family)*
colleague   → (name,affil*,email*)
friend      → (name,affil*,email*)
family      → (name,affil*,email*)
name        → (#PCDATA)
…
```

Element names and their content



XML Documents

Example DTD

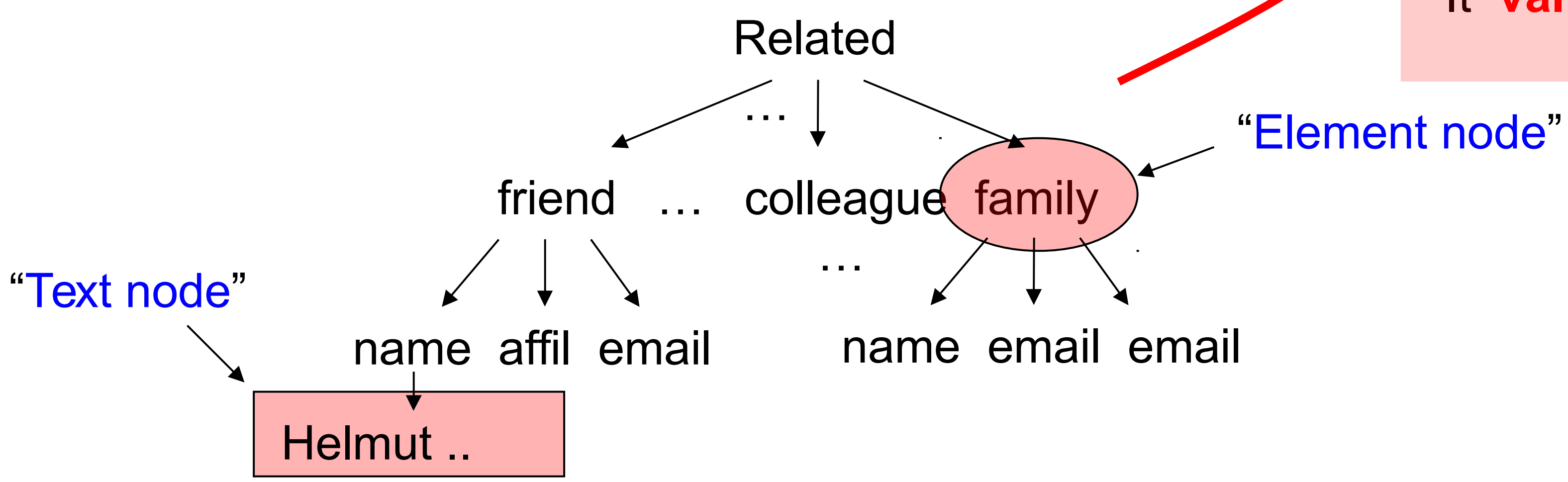
```
Related    → (colleague | friend | family)*
colleague  → (name,affil*,email*)
friend     → (name,affil*,email*)
family     → (name,affil*,email*)
name       → (#PCDATA)
…
```

Element names and their content

Terminology

document is **valid** wrt the DTD

"It **validates**"



XML Documents

What else: (besides *element* and *text* nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

XML Documents

What else: (besides *element* and *text* nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

```
<entry date="2017-01-16">
<name>
…
</entry>
```

XML Documents

What else: (besides *element* and *text* nodes)

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

```
<entry date="2017-01-16">
<name>
…
</entry>
```

→ at most one date-attribute
→ no substructure possible

versus:

```
<date>2017-01-16</date>
<date>
    <year>2017</year>
    <month>01</month>
    <day>16</day>
</date>
```

XML Documents

```
<?php sql ("SELECT * FROM …") …?>
```

intended to carry instructions to the application

What else:

→ attributes
→ processing instructions
→ comments
→ namespaces
→ entity references (two kinds)

```
<entry date="2017-01-16">
<name>
…
</entry>
```

XML Documents

```
<?php sql ("SELECT * FROM …") …?>
```

intended to carry instructions to the application

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity references (two kinds)

```
<entry date="2017-01-16">
<name>
…
</entry>
```

XML Documents

```
<?php sql ("SELECT * FROM …") …?>
```

intended to carry instructions to the application

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity references (two kinds)

```
<entry date="2017-01-16">
<name>
…
</entry>
```

```
<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>
```

Namespaces provide unique element and attribute names

XML Documents

```
<?php sql ("SELECT * FROM …") …?>
```

intended to carry instructions to the application

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ entity *references* (two kinds) → *character reference*

Type <key>less-than</key> (<) to save options.

```
<entry date="2017-01-16">
<name>
…
</entry>
```

Name	Character	Unicode code point (decimal)	Standard	Description
quot	"	U+0022 (34)	XML 1.0	double quotation mark
amp	&	U+0026 (38)	XML 1.0	ampersand
apos	'	U+0027 (39)	XML 1.0	apostrophe *(apostrophe-quote)*
lt	<	U+003C (60)	XML 1.0	less-than sign
gt	>	U+003E (62)	XML 1.0	greater-than sign

```
<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>
```

Namespaces provide unique <u>element</u> and <u>attribute names</u>

XML Documents

```
<?php sql ("SELECT * FROM …") …?>
```

intended to carry instructions to the application

What else:

→ attributes
→ processing instructions
→ comments `<!-- some comment -->`
→ namespaces
→ <u>entity *references*</u> (two kinds)

character reference
Type <key>less-than</key> (<) to save options.

```
<entry date="2017-01-16">
<name>
…
</entry>
```

This document was prepared on <u>&docdate;</u> and

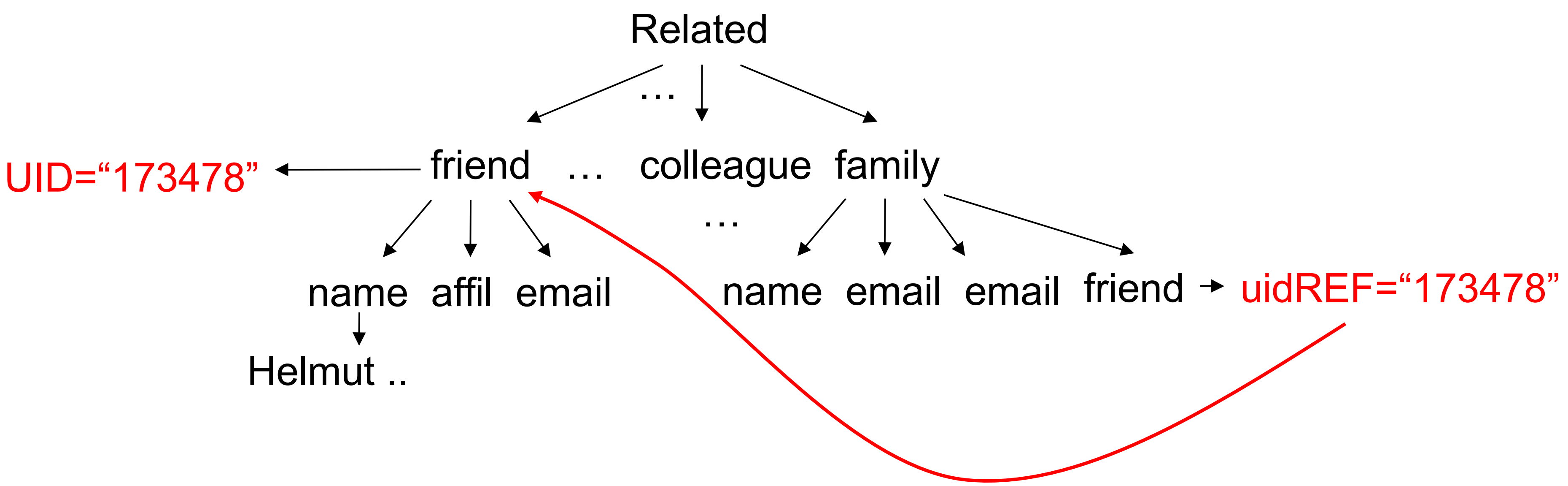
```
<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price xmlns:edi='http://ecommerce.org/schema' units='Euro'>32.18</edi:price>
```

Namespaces provide unique <u>element</u> and <u>attribute names</u>

XML: not tree but Graph

attributes of type ID: must be unique, i.e., no duplicate values

may be referenced via attributes of type IDREF



→ ID-attributes are similar to keys in relational DBs

XML, typical usage scenario

Document structure
Def. of price, gst, …

DTD, XML Schema

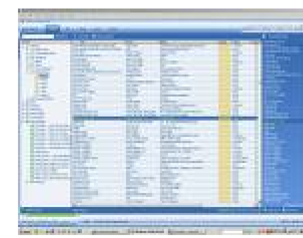
Presentation
Format info

XML Stylesheet

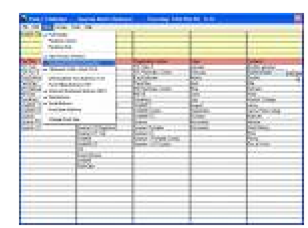
```
<Product>
  <product_id> m101 </product_id>
  <name> Sony walkman </name>
  <currency> AUD </currency>
  <price> 200.00 </price>
  <gst> 10% </gst>
</Product>
…
```

XML

XML Stylesheet



XML Stylesheet



One data source

→ several dynamically generated **views**

XML: has it succeded?

Yes and No:

has become *very* popular and adopted

technically it is still (!) challenging:

(*) standard too complex
(*) causes, e.g., slowness of XML parsers
(a “threat to databases”)

→ JSON

- invented in 2001 by Douglas Crockford
- took off since 2005/2006

JavaScript Object Notation

XML vs JSON

```
<Related>
<colleague>
<name>Philip Wadler</name>
<affil>U. of Edinburgh</affil>
<email>
wadler@inf.ed.ac.uk
</email>
</colleague>
…
<friend>
<name>Helmut Seidl</name>
<affil>TU Munich</affil>
<email>seidl@inf.tum.de
</email>
</friend>
</Related>
```

```
Related = {
“colleague”:{
“name”:”Philip Wadler”,
“affil”:”U. of Edinburgh”,
“email”:”wadler@inf.ed.ac.uk”
}
…
“friend”: {
“name”:”Helmut Seidl”,
“affil”:”TU Munich”,
“email”:”seidl@inf.tum.de”}
}
```

XML vs JSON

- 7 node types
- DTDs are built in

Very rich schema languages, e.g.,

- XML Schema

(e.g., XHTML schema: >2000 lines)

6 data types:

- number
- string
- boolean (`true` / `false`)
- array
- object (set of `name:value` pairs)
- empty value (`null`)

2. Well-Formed XML

From the W3C XML recommendation

`http://www.w3.org/TR/REC-xml/`

"A textual object is **well-formed XML** if,

(1) taken as a whole, it **matches the production labeled** *document*
(2) it meets all the **well-formedness constraints** given in this specification .."

---

document = start symbol of a context-free grammar ("XML grammar")

→ (1) contains the *contex-free properties* of well-formed XML
→ (2) contains the *context-dependent properties* of well-formed XML

There are 10 WFCs (well-formedness constraints).
E.g.: **Element Type Match** "The Name in an element's end tag must match the element name in the start tag."

→ Why is this *not* context-free?

XML Grammar - EBNF-style


```
[1]    document    ::=   prolog element Misc*
[2]        Char    ::=   a Unicode character
[3]           S    ::=   (‘ ’ | ‘\t’ | ‘\n’ | ‘\r’)+
[4]    NameChar    ::=   (Letter | Digit | ‘.’ | ‘-’ | ‘:’)
[5]        Name    ::=   (Letter | '_' | ':') (NameChar)*

[22]     prolog    ::=   XMLDecl? Misc* (doctypedecl Misc*)?
[23]    XMLDecl    ::=   '<?xml' VersionInfo EncodingDecl? SDDecl? S? '?>‘
[24]VersionInfo    ::=   S'version'Eq("'"VersionNum"'"|'"'VersionNum'"')
[25]         Eq    ::=   S? '=' S?
[26]VersionNum     ::=   '1.0‘

[39]    element    ::=   EmptyElemTag
                       | STag content Etag
[40]       STag    ::=   '<' Name (S Attribute)* S? '>'
[41] Attribute     ::=   Name Eq AttValue
[42]       ETag    ::=   '</' Name S? '>‘
[43]    content    ::=   (element | Reference | CharData?)*
[44]EmptyElemTag ::=     '<' Name (S Attribute)* S? '/>‘

[67] Reference     ::=   EntityRef | CharRef
[68] EntityRef     ::=   '&' Name ';‘
[84] Letter        ::=   [a-zA-Z]
[88] Digit         ::=   [0-9]
```

XML Parsing: A Threat to Database Performance

Matthias Nicola
IBM Silicon Valley Lab
555 Bailey Avenue
San Jose, CA 95123, USA
mnicola@us.ibm.com

Jasmi John
IBM Toronto Lab
8200 Warden Ave
Markham, ON L6G 1C7, Canada
jasmij@ca.ibm.com

ABSTRACT
XML parsing is generally known to have poor performance characteristics relative to transactional database processing. Yet, its potentially fatal impact on overall database performance is being underestimated. We report real-word database applications where XML parsing performance is a key obstacle to a successful XML deployment. There is a considerable share of XML database applications which are prone to fail at an early and simple road block: XML parsing. We analyze XML parsing performance and quantify the extra overhead of DTD and schema validation. Comparison with relational da[...]
response times and tr[...]
achieved without maj[...]
ogy. Thus, we identif[...]
for XML parser perfor[...]

in: Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management,
New Orleans, Louisiana, USA, November 2-8, 2003

Categories and Subject Descriptors
H.2.4 [**Database Management**]: Systems–*transaction processing.*

General Terms: Algorithms, Measurement, Performance, Design.

Keywords: XML, Parser, Database, Performance, SAX, DOM, Validation.

1. INTRODUCTION
XML has become much more than just a data format for information exchange. Enterprises are keeping large amounts of business critical data permanently in XML format. Data centric as well as document and content centric businesses in virtually every industry are embracing XML for their data management and B2B needs [8]. E.g. the world's leading financial companies have been working on over a dozen major XML vocabularies to standardize their industry's data processing [9].

All major relational database vendors offer XML capabilities in their products and numerous "native" XML database systems have emerged [2]. However, neither the XML-enabled relational sys-

tially because processing of XML requires *parsing* of XML documents which is very CPU intensive.

The performance of many XML operations is often determined by the performance of the XML parser. Examples are converting XML into a relational format, evaluating XPath expressions, or XSLT processing. Our experiences from working with companies, which have introduced or are prototyping XML database applications, show that XML parsing recurs as a major bottleneck and is often the single biggest performance concern seriously threatening the overall success of the project. This observation is general to

(2 to 5 times the size of the XML document, hence unsuitable for large documents). Lazy DOM parsers materialize only those parts of the document tree which are actually accessed, but if most the document is accessed lazy DOM is slower than regular DOM. SAX parsers report parsing events (e.g. start and end of elements) to the application through callbacks. They deliver an event stream which the application processes in event handlers. The memory consumption does not grow with the size of the document. In general, applications requiring random access to the document nodes use a DOM parser while for serial access a SAX parser is better.

XML parsing allows for optional validation of an XML document against a DTD or XML schema. Schema validation not only checks a document's compliance with the schema but also determines type information for every node of the document (aka *type annotation*). This is a critical observation because database systems and the Xquery language are sensitive to data types. Hence most processing of documents in a data management context not only requires parsing but also "validation".

Depending on an XML database system's implementation, there are various operations which require XML parsing and possibly

How expensive is XML Parsing?

→ DTD is part of XML

→ DTDs may contain (deterministic) regular expressions

→ How expensive is it to match a text of size n against a regular expression of size m?

→ DTDs allow recursive definitions

→ DTDs allow ID and IDREF attributes (ID: check uniqueness, IDREF: check existence)

How expensive is XML Parsing?

→ DTD is part of XML

→ DTDs may contain (deterministic) regular expressions

→ How expensive is it to match a text of size n
against a regular expression of size m?

→ DTDs allow recursive definitions

→ DTDs allow ID and IDREF attributes
(ID: check uniqueness, IDREF: check existence)

Compare this to parsing complexity of

→ JSON
→ csv files (csv = "comma-separated values") [IBM Fortran, 1967]

END

Lecture 1