Introduction to Databases, Fall 2004

IT University of Copenhagen

Lecture 8, part II: XML for data exchange

October 22, 2004

Lecturer: Rasmus Pagh

Today's lecture, part II

XML for data exchange

- Semistructured data and XML.
- Defining XML formats using XML schemas.

Next: Semistructured data and XML.

Integration of databases

- In many businesses, data from a large number of heterogeneous databases need to be integrated
  - in connection with data warehousing, or
  - in connection with system integration in general
- This is no easy task, due to differences in formats, conventions, systems, etc.

This part of today's lecture gives an overview of (some of the things that go into) transferring data from one system to another.

The semistructured data model

The **semistructured data model** is a *flexible* way of describing data.

The flexibility makes it a good data model for data exchange:

- It is (mostly) easy to convert a given data set to a semistructured representation.
- It is (often) easy to perform transformation from one semistructured representation to another.

[Figure 4.19 shown on slide]

Semistructured data

Semistructured data can be represented as a **graph** with **nodes**, and **arcs** with labels between the nodes.

There are three kinds of nodes:

- A single **root node**, with no arcs entering, represents the entire database.
- Leaf nodes, with no arcs leaving, have associated data (e.g. strings).
- Interior nodes have arcs entering and leaving, but no data.

`[Figure 4.19 shown on slide]`

XML

XML is a *standardized textual notation for semistructured data*.

It is (primarily) aimed at semistructured data which is a **tree**, i.e., where all nodes (except the root) have exactly one arc entering.

- An arc in the tree with label `l` pointing at a node n in the semistructured data is represented in the XML document as a pair of **tags**:

  `<l>...</l>`

  Here `...` is the XML description of the part of the semistructured data for which n is the root.

- Leaf nodes are represented by the data they contain.

XML example

If we disregard the "crossing arcs" in [Figure 4.19], it is represented by the following XML document:

```
<?xml version="1.0">
<root>
  <star>
    <name>Carrie Fisher</name>
    <address><street>Maple</street><city>H'wood</city></address>
    <address><street>Locust</street><city>Malibu</city></address>
  </star>
  <star>
    <name>Mark Hamill</name>
    <address><street>Oak</street><city>B'wood</city></address>
  </star>
  <movie>
    <title>Star Wars</title>
    <year>1977</year>
  </movie>
</root>
```

XML and WWW

Besides data interchange, another use of XML in connection with databases is for sharing information via the World Wide Web.

- Newer web browsers have special facilities for viewing XML documents (e.g., containing the result of a database query).
- There are specialized languages such as XSLT that can be used to specify how XML data is to be presented in a browser (converting it to HTML).

Problem session

Suggest a way of representing the below relation in XML.

accountNo	*balance*	*type*
12345	1000.00	`savings`
67890	2846.92	`checking`
32178	-3210.00	`loan`

Does your approach work for arbitrary relations?

What about entire relational databases?

Next: Defining XML formats using XML schemas.

Schemas for XML

When doing data interchange it is necessary to have a *common description* of the data format, i.e., we need a specification of what data is allowable.

Several languages for writing such **schemas** for XML are used. The most widespread are **DTD** (old and well-established) and **XML Schema** (new, more powerful standard).

These schema languages work by specifying a **grammar** for the XML documents allowed, i.e., a set of rules that can be used to form any allowable XML document.

Essentially, for each `<l>...</l>` (called an XML **element**) it is specified what can occur between `<l>` and `</l>`.

DTDs

DTD by example: `[Figure 4.22 shown on slide]`

DTDs have these ways of stating what can be in an element:

- Text, written as `#PCDATA`.
- A sequence of elements, written `(ELEM1, ELEM2, ELEM3, ...)`
- Zero or more occurrences of the same element, written `ELEM*`.
- A choice between elements, written `ELEM1 | ELEM2 | ELEM3 | ...`
- An optional element, written `ELEM?`

It is possible to combine the above, and write expressions such as:

```
((A? | (B|C)*), D)
```

XML Schemas

"XML Schema" seems to be the upcoming standard for XML schemas.

Some main features, relative to DTDs:

- Sophisticated type system (in contrast to `#PCDATA`).
- Large schema definitions can be split into modules.
- Can specify "no content".
- Supports a mechanism for distinguishing different XML elements with the same name ("namespaces").

`[Murray, Figure 3, shown on slide]`

Why XML?

The reason for the success of XML and XML Schema is, in fact, *not* that it does something that could not be done before.

The good new thing is **standardization**:

- There is *widespread agreement* that the XML standards will form the basis of information interchange in the future.
- Consequently, the major players in the software industry, many country administrations, etc., support XML.
- There are many tools available for XML and related technologies.

Most important points in this lecture

As a minimum, you should after this week:

- Be able to recognize an XML document.
- Be able to understand a simple DTD or XML Schema.

In the course material, you will find many more details than what I have talked about. I do not expect you to learn these details.

Next lecture

Next time we will look at the basics of relational database efficiency:

- Indexes.
- Update versus query performance.
- Database tuning.