XML and Databases

Data Management for Big Data
2018-2019 (spring semester)

Dario Della Monica

These slides have been translated in English from slides in Italian by Angelo Montanari

What XML is

XML stands for eXtensible Markup Language

Main features:

(1) XML is a formal language. XML is defined through a set of formal rules (a grammar), that defines how an XML document is generated.

(2) XML allows for data description (markup). Data are included in XML documents as string of characters enclosed between markup tags, which describe structure and content type.

(3) XML is extensible. The language provides an extensible set of tags that can be adapted (unlike HTML, whose set of tags is fixed).

What XML is NOT - 1

An XML document is a text document that can be read and modified with any text editor.

Therefore, XML **is not**:

- a language for **data presentation**, like HTML. XML markups define data semantics (content), rather than presentation stlye. Style information can be included in a separate stylesheet. XSL (XML Stylesheet Language) stylesheet can be used to translate XML data into HTML. Resulting HTML pages can be displayed inside a browser.

What XML is NOT - 2

- a **programming language** (such as Java). An XML document does not perform computations.
- a **network transfer data protocol** (such as HTTP). XML does not transfer data over the network.
- a **DBMS** (such as Oracle). XML does not store or return data.

XML elements

An XML element is a text enclosed between text tags. Tags are enclosed between angle brackets.

Example—XML document:

$< person >$
$\quad Alan\ Turing$
$< /person >$

The document contains only 1 *element*, called *person*. Such an element is delimited by the beginning tag $< person >$ and the ending tag $< /person >$. Tags are used for text markup. Whatever is enclosed between a beginning and an ending tag is called **element content** (in the example, the string *Alan Turing*). An XML element can contain free text (called *character data*) or other XML elements.

Example: an XML document

Example.

$< person >$

$\quad < name >$

$\quad\quad < first > Alan < /first >$

$\quad\quad < last > Turing < /last >$

$\quad < /name >$

$\quad < profession > computer\ scientist < /profession >$

$\quad < profession > mathematician < /profession >$

$\quad < profession > cryptographer < /profession >$

$< /person >$

Element *person* contains a sub-element *name* and 3 sub-elements *profession*.

Element *name*, in turn, contains 2 sub-elements *first* and *last*.

Features of an XML document

- **Elements cannot overlap**: if the beginning tag of element B follows the beginning tag of element A, then ending tag of B must precede ending tag of A.
- There must exist a single element that encloses all of the other ones (**document element**).
- **Each element**, except for document element, **is enclosed** (in a direct fashion) **in exactly one other element**.

Consequently: an XML document can be represented as a **tree structure**: elements are the **nodes**, document element is the **root**, sub-elements are the **children**.

Another example

XML elements can **interleave free text** and **sub-elements** as in the following example:

$< person >$
$< first > Alan < /first >< last > Turing < /last > is\ mainly\ known$
$as\ a < profession > computer\ scientist < /profession > .\ However,\ he$
$was\ also\ an\ accomplished < profession > mathematician < /profession >$
$and\ a < profession > cryptographer < /profession > .$
$< /person >$

There might also be elements with no content at all (**empty elements**. An empty address element can be compactly represented as $< address/ >$).

Notice: XML is **case sensitive** (*address* and *Address* are different tags).

XML attributes

XML elements can have **attributes** describing their properties. An attribute has the following syntax: *name* = "*value*", where *name* is the attribute's name and *value* is a string. *value* can be enclosed between single or double quotes. An element can have an arbitrary number of attributes, which must have different names. Attribute order is irrelevant (unlike element order, which DOES matter).

```
< person born = "23/06/1912" died = "07/06/1954" >
    < name >
        < first > Alan < /first >
        < last > Turing < /last >
    < /name >
    < profession > computer scientist < /profession >
    < profession > mathematician < /profession >
    < profession > cryptographer < /profession >
< /person >
```

XML elements vs. XML attributes

Some pieces of information can be encoded **both** as **attribute** value and as **element** content.

$$
\begin{aligned}
&< person\ born = "23/06/1912"\ died = "07/06/1954" > \\
&\quad < name\ first = "Alan"\ last = "Turing" / > \\
&\quad < profession\ value = "computer\ scientist" / > \\
&\quad < profession\ value = "mathematician" / > \\
&\quad < profession\ value = "cryptographer" / > \\
&< /person >
\end{aligned}
$$

How to choose? Attributes for element metadata, elements for storing information.

XML references - 1

So far, XML documents have a **tree structure**. XML allows one to define and use references that make it possible to produce documents with a **graph structure**.

It is possible to associate a unique identifier to elements as value of a suitable attribute (e.g., the *id* attribute).

$< state\ id = "s1" >$

$\quad < scode > CA < /scode >$

$\quad < sname > California < /sname >$

$< /state >$

XML references - 2

To refer to *state* element previously defined, it is possible to use *idref* attribute.

$< city\ id = "c1" >$
$\quad < ccode > LA < /ccode >$
$\quad < cname > Los\ Angeles < /cname >$
$\quad < state\text{-}of\ idref = "s1" / >$
$< /city >$

Notice that *state-of* is an empty element, whose only purpose is referring to another element through the value of *idref* attribute. Through references, it is possible to represent **looping/recursive data structures**. In the previous example, we can add a sub-element *capital* within *state* element, featuring attribute *idref* referring to element *city*.

XML parser

An XML parser is a software that reads an XML document and establishes whether or not it is well-formed. A well-formed document fulfills XML grammar rules:

1. every beginning tag has a corresponding ending tag;
2. elements do not overlap;
3. there must be exactly one *document* element;
4. attribute values must be enclosed between quotes;
5. attributes within the same element must have different names.

The simplest way to parse an XML document is to load it inside a web browser that *recognizes* XML (i.e., browser includes an implementation of an XML parser). There are also stand-alone XML parsers like *xmllint command line tool*.

XML main applications

XML main applications are the following:

- **Data exchange**. In presence of information from different data source (relational DB, object DB, text document), that needs to be exchanged/integrated, XML is a suitable common language to facilitate the inter-change/integration.
- **Semi-structured databases**. Semi-structured data are devoid of a regular schema; therefore, relational DB can be inadequate to deal with them. XML has been proposed as data model for semi-structured data. Such a data model is close to the **hierarchical data model**.

DTD (Document Type Definition)

It is possible to specify markup that is allowed inside an XML application through the definition of a **schema**. The most common schema definition language (the only one that is present in XML 1.0 standard) is **DTD** (Document Type Definition). An alternative is **XSchema**.

A DTD allows one to force **structural constraints** over an XML document. It lists elements, attributes and entities (as we will see later on, an XML entity is a name for a portion of text; some entity come by default, other can be defined by the user) used inside the document and specify the context within which they are used.

DTD **do not** carry any semantic information about element **contents** or attribute **values**.

DTD for the "Alan Turing" document - 1

A DTD is precisely a context-free grammar for the document.

DTD for the "Alan Turing" document:

$<!ELEMENT\ person\ (name, profession*)>$
$<!ATTLIST\ person\ born\ CDATA\ \#REQUIRED$
$\quad died\ CDATA\ \#IMPLIED>$
$<!ELEMENT\ name\ (first, last)>$
$<!ELEMENT\ first\ (\#PCDATA)>$
$<!ELEMENT\ last\ (\#PCDATA)>$
$<!ELEMENT\ profession\ (\#PCDATA)>$

First line states that element *person* has exactly one child element *name* and zero or more child elements *profession*, in this order.

DTD for the "Alan Turing" document - 2

Second line states that element *person* has an attribute *born* and an attribute *died* (order does not matter).

Sections CDATA (character data) are blocks of character data treated as pure textual data (markups occurring inside a CDATA section are not recognized; they are treated as character data).

Third line states that element *name* has 2 children called, respectively, *first* and *last* (order matters).

Last 3 lines specify that elements *first*, *last*, and *profession* must contain parsed character data (PCDATA), that is, pure text containing entity and references, but not tags.

Document type declaration - 1

An XML document can be associated with a DTD through a **Document type declaration**. Such a declaration should follow the XML declaration inside the XML document. Its format is as follows:

$<!DOCTYPE\ root\ [DTD]>$

where DOCTYPE is the keyword for the document type declaration, root is the name of the element of the XML document, and DTD is the set of rules defining the DTD (previous slides).

Document type declaration - 2

An XML document might contain a **DTD reference**, rather than including it. This allows for DTD sharing among more XML documents. In this case, the declaration is as follows:

$<!DOCTYPE\ root\ KEYWORD\ "URI" >$

where KEYWORD can be SYSTEM o PUBLIC. In the former case, the DTD is specified through a Uniform Resource Locator (URL); in the latter case, the DTD is specified through a Uniform Resource Name (URN), where the URN is a name that unambiguously identify the XML application. (a backup URL can be added in case local DTD becomes unavailable).

XML element definition in DTD - 1

Every element used in a valid document must be declared in the DTD through an **element definition**, in the following form:

$$<!ELEMENT\ name\ content>$$

where *name* is the name of the element and *content* specified the children that it must/can have, and their order.

content can be one of the following:

- **Parsed character data**

  $$<!ELEMENT\ email\ (\#PCDATA)>$$

- **Child element**. An element can have a child element of one type only

  $$<!ELEMENT\ contact\ (e\text{-}mail)>$$

XML element definition in DTD - 2

- **Choice**. An element can contain one or another type of child but cannot have children of both types

$$<!ELEMENT\ contact\ (e\text{-}mail|phone)>$$

- **Sequence**. An element can contain more children, in a given order.

$$<!ELEMENT\ name\ (first, last)>$$

- **Empty content**. An element cannot contain any content; however, it can have attributes.

$$<!ELEMENT\ image\ EMPTY>$$

- **Any content**. An element can have content of any kind (if it has children, they must be defined).

$$<!ELEMENT\ image\ ANY>$$

XML element definition in DTD - 3

- **Iteration**. Three different suffixes can be used to specify how many children element there can be:

  $*$ zero or more;

  $+$ one or more;

  ? zero or one.

  For instance, the following definition forces *name* to have zero or more *first* children, followed, possibly, by a child *middle* and by one or more *last* children.

$$<!ELEMENT\ name\ (first*, middle?, last+) >$$

Examples

According to previous definition, all of the following *name* elements are valid:

$< name >$
$\quad < first > Samuel < /first >$
$\quad < middle > Lee < /middle >$
$\quad < last > Jackson < /last >$
$< /name >$

$< name >$
$\quad < first > Samuel < /first >$
$\quad < first > Michael < /first >$
$\quad < last > Jackson < /last >$
$< /name >$

$< name >$
$\quad < last > Jackson < /last >$
$\quad < last > Keaton < /last >$
$< /name >$

More examples - 1

The following definition of *name* allows for arbitrary many occurrences of children *first*, *middle*, and *last* in any order:

$$<!ELEMENT\ name\ (first|middle|last)* >$$

The following definition allows for mixing text and markup: *name* can feature any number of occurrences of children *first*, *middle*, and *last* in any order, possibly interleaved with free text.

$$<!ELEMENT\ name\ (\#PCDATA|first|middle|last)* >$$

More examples - 2

According to the previous definition, the following *name* element is valid:

$< name >$

$First\ comes\ the\ first\ name : < first > Samuel < /first >$

$Then\ the\ middle\ one : < middle > Lee < /middle >$

$Last\ comes\ the\ last\ name : < last > Jackson < /last >$

$Not\ very\ surprising\ indeed!$

$< /name >$

Notice that this is the only way of characterizing mixed content: an element containing arbitrary many occurrences of some element from a given list, in any order, interleaved with free text (keyword $\#PCDATA$ must be the first component in the list).

Determinism and non-determinism

Consider the following alternative definition of *name*:

$$<!ELEMENT\ name\ (first|last|(first, last))>$$

Such a definition is not valid, because the content model of *name* is not deterministic.

The problem is about DTD (not XML): **content** model produced by **DTD** must be **deterministic**.

When a validator reads an element *first*, it can interpret it in 2 ways:

1. the definition of *name* is terminated (first disjoint in the definition);
2. the definition of *name* must be followed by an element *last* (last disjoint in the definition).

By reading the same symbol, the validator should proceed in 2 different ways (non-determinism).

Other components

A valid XML document must also define element **attributes**. Declaration is as follows:

$$<!ATTLIST\ element\ attribute\ TYPE\ DEFAULT>$$

ID and *IDREF* are important attributes. They can be thought of as the XML counterparts of **primary** and **foreign keys** in relational DB.

Besides attributes, it is possible to define **entities**. An entity is an abbreviation for a certain set of data, not necessarily in XML format. There are several kind of entity: internal or external, default or user-defined. Internal, user-defined entities are defined as follows:

$$<!ENTITY\ name\ "text">$$

Validity of an XML document

An XML document is said **valid** if it fulfills specifications contained in its DTD. In general, web browsers do not validate documents; they only verify they are well-formed

In order to validate, one can use parser API or online or stand-alone validators

XML Validation Form by Brown University Scholarly Technology Group is an online validator. *xmllint* is a command-line XML parser and validator; it is part of the XML library *libxml*, developed within the Gnome project; it can be used outside of Gnome as well

Validation modality

For **instance**, document Turing.xml can be validated wrt. Turing.dtd by executing the following command:

xmllint –dtdvalid Turing.dtd Turing.xml

If Turing.xml contains document type declaration, it is possible to use the following command:

xmllint –valid Turing.xml

To avoid showing XML document output, option –noout should be used.

DTD and relational schemas - 1

DTD can be used **as schemas**. There are, however, some limitations:

- Sub-elements of an element (interpreted as attributes of a relation in a relational DB) are ordered. To allow for any order, it is necessary to explicitly list all possible orderings. Example: a relation $R(A, B)$ (from a relational DB) must correspond to a declaration $<!ELEMENT\ R\ ((A, B)|(B, A))>$.
- There is no notion of atomic entity. The only atomic type is PCDATA (string).
- It is not possible to specify constraints about sets of legal values for a given element (e.g., to force *age* to range between 0 and 125).

DTD and relational schemas - 2

- It is not possible to constrain the number of occurrences of elements. For instance, it is not possible to force at least 3 occurrences of a given element.
- The mechanism for managing ID / IDREF atributes is too simple. For instance, it is not possible to limit the uniqueness condition for ID attributes to only hold on a fragment of the document (rather than the whole document). Moreover, it is not possible to constrain the type for idref attributes (it is possible to do that with foreign keys in relational DB's). Finally, only single attributes can be used as keys.
- The type associated to an element tag is global. For instance, in an object DB a field *name* can have different structure depending on the class with which it is associated. This is not possible in XML. A possible solution would be to use different tags and namespaces.

DTD and relational schemas - 3

- DTD provides limited support for modularity, code reuse, and schema evolution. This makes it difficult to manage large schemas involving many associations.
- DTD are not described using XML notation; if that were the case, their management (well-formed check, validation, schema query, ...) could have been done using XML tools

On the bright side, DTD allows one to describe in a easy way optional and multi-valued attributes from E-R schemas.

Example.

$<!ELEMENT\ R\ (A, B?, C+))>$ states that A is mandatory, B is optional, and C must have one or more occurrences.

Beyond DTD: XML Schema

XML Schema has been proposed by W3C to overcome DTD limitations.

In particular, it provides:

1. a powerful type system: it allows definition of both simple and complex types, as well as definition of type inheritance mechanisms, along the lines of object-oriented programming languages,
2. types can be associated to both elements and attributes, thus increasing their semantic content.

On the dark side, such extensions make XML Schema more difficult to work with, especially for inexperienced users.

XML query languages

An XML DB is a collection of XML inter-related documents.

The different data model behind XML DB's (trees) wrt to the data model behind relational DB's (tables) requires the use of ad-hoc **query languages**.

The most common XML query languages are:

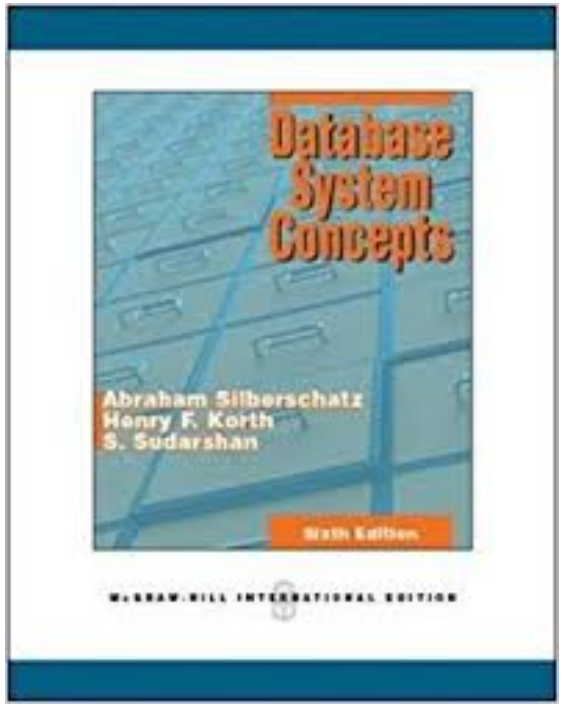
- XML Path Language (XPath). It makes it possible to get elements from a single XML document.

  Online XPath Tester/Evaluator:
  https://www.freeformatter.com/xpath-tester.html

- XML Query Language (XQuery). It is a full query language for XML DB's (it is the XML counterpart for SQL in relational DB's).

  Online XPath/XQuery Tester:
  http://www.xpathtester.com/xquery

XML references



Chapter 23

Database System Concepts
Silberschatz, Korth, Sudarshan,
6th ed., McGraw-Hill, 2011

`https://www.db-book.com/db6/index.html`