Repetition: Distributed Databases

A distributed database consists of 2+ backend servers (possibly in different physical locations)

Two common reasons for such distribution:

Fault-tolerance (avoiding a **single point of failure**)

Scalability (if you have to manage enough data, even the largest server can't fit all at once anymore)

These are two **orthogonal issues** and should not be mixed up

Replication Models

Master/slave replication

1 master (authoritive copy), 1..N slaves (backups, get activated if the master is unavailable)

Election procedure is used to promote one slave to master if master becomes unavailable

Master/master replication

Multiple masters, voting is used to resolve inconsistencies

Sharding

Basic mechanism for dealing with data at scale: **sharding**

- Fancy name for splitting up data between multiple nodes

Like in replication, you have multiple database nodes

- Unlike in replication, the goal is **not** that they keep the same data

CAP Theorem

Well-known theoretical result for distributed databases:

CAP Theorem

You can have max. two of the following three properties at the same time in a distributed database

Consistency (all nodes have same copy of data)

Availability for Updates (you can at all times **write** to the database)

Partition tolerance (the system can deal graciously with a nodes being unable to talk for some time)

BASE

Tongue-in-cheek alternative to ACID properties for NoSQL:

BASE

(**b**asically **a**vailable, **s**oft state, **e**ventually consistent)

Classes of NoSQL Database Systems

Key/Value Stores

E.g., Redis, DynamoDB, memcached

Document Stores

E.g., MongoDB, CouchDB

Column-Based Databases

E.g., HBase

Graph Databases

E.g., Neo4J

Short Kahoot Quiz

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Data Formats and Representation

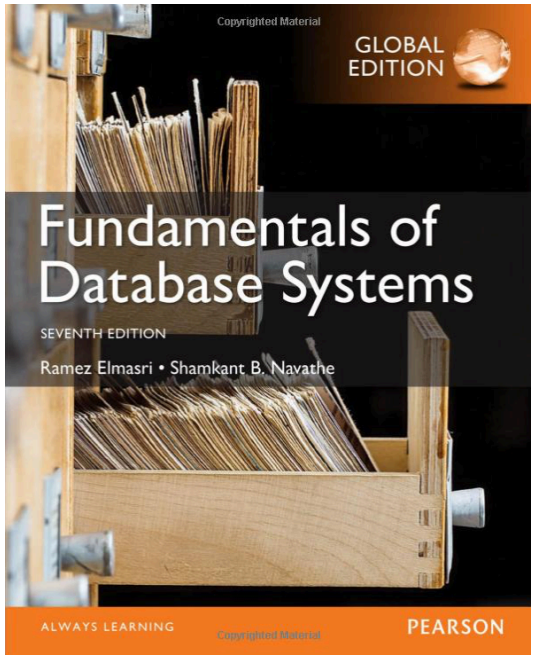
LECTURE 13

Dr. Philipp Leitner

philipp.leitner@chalmers.se

@xLeitix

LECTURE 13



Covers …

Parts of Chapter 13

Intro to semi-structured data and XML

CSV, JSON, YAML not covered in the book

What we will be covering

Fundamentals of data representation

(Some) common data formats:

Plain Text, CSV, XML, JSON, YAML

Example JSON

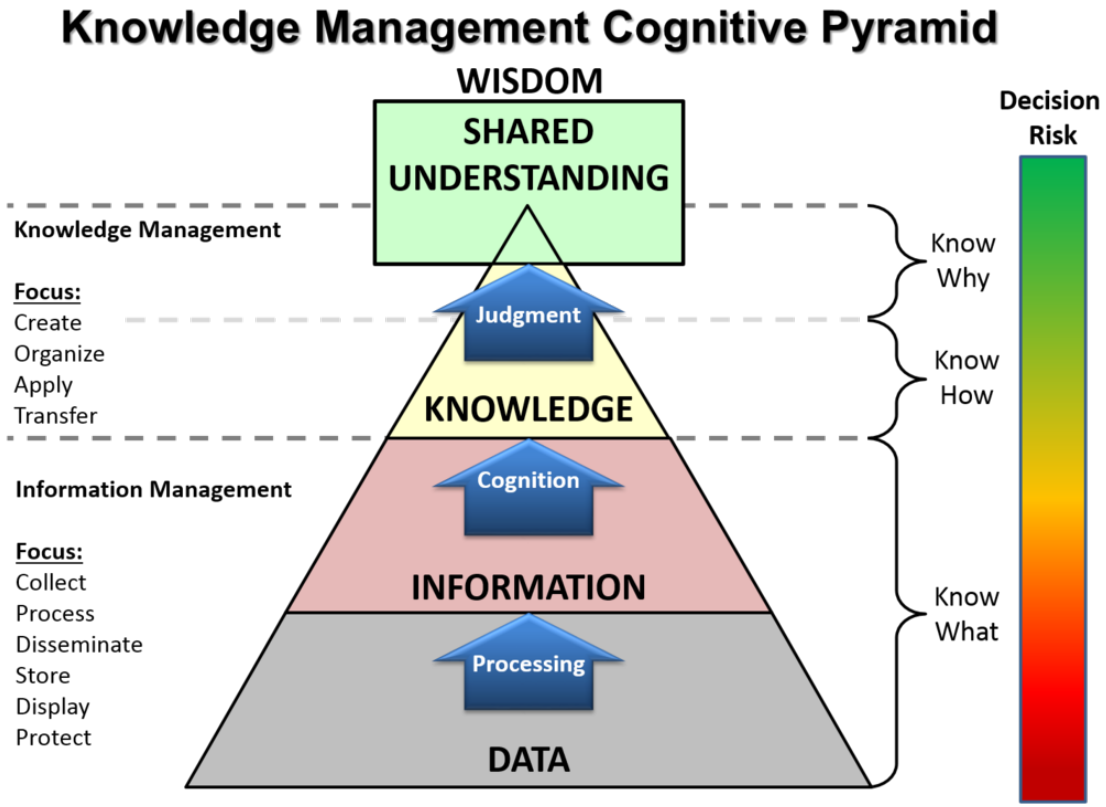
(a) project document with an array of embedded workers:

```
{
    _id:            "P1",
    Pname:          "ProductX",
    Plocation:      "Bellaire",
    Workers: [
                { Ename: "John Smith",
                  Hours: 32.5
                },
                { Ename: "Joyce English",
                  Hours: 20.0
                }
            ]
);
```

(b) project document with an embedded array of worker ids:

```
{
    _id:            "P1",
    Pname:          "ProductX",
    Plocation:      "Bellaire",
    WorkerIds:      [ "W1", "W2" ]
}
    { _id:          "W1",
    Ename:          "John Smith",
    Hours:          32.5
}
    { _id:          "W2",
    Ename:          "Joyce English",
    Hours:          20.0
}
```

Reminder: DIKW Pyramid



Data vs. Representation

Besides the DIKW Pyramid there is another way to think about data:

Data / Information:

The concepts that don't "exist" except as ideas

Representation:

How these concepts are saved on a computer

Different Levels of Representational Abstraction

In practice not all representations are equal

Some ways to represent data is **closer to the actual idea** than others

For instance:

10010000 (binary)

144 (textual)

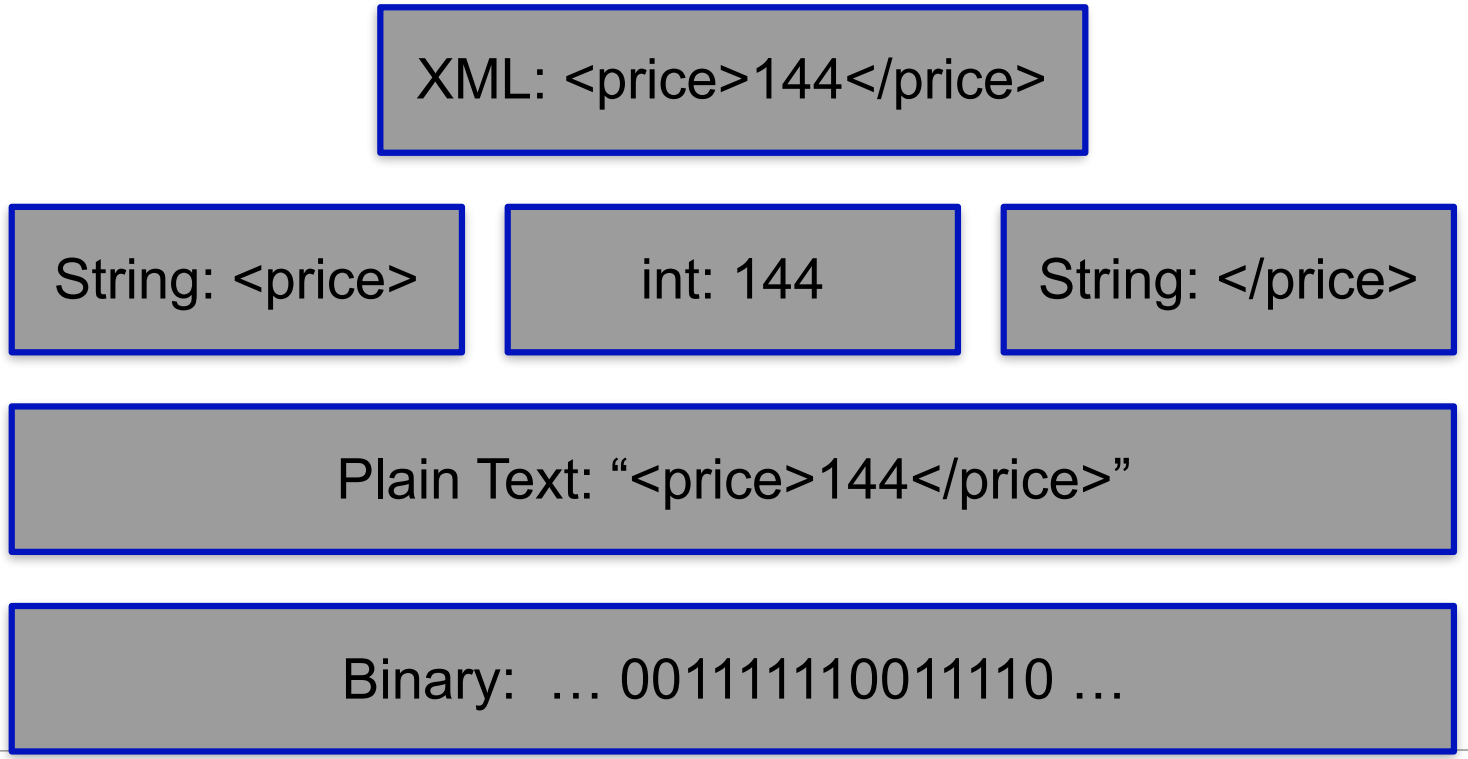
<price>144</price> (textual with metadata)

Lifting and Lowering

Different representations can be converted into each other:

We often use the terms **lifting** (bringing to higher level of abstraction) and **lowering** (bringing closer to machine level)

Representations are built hierarchically



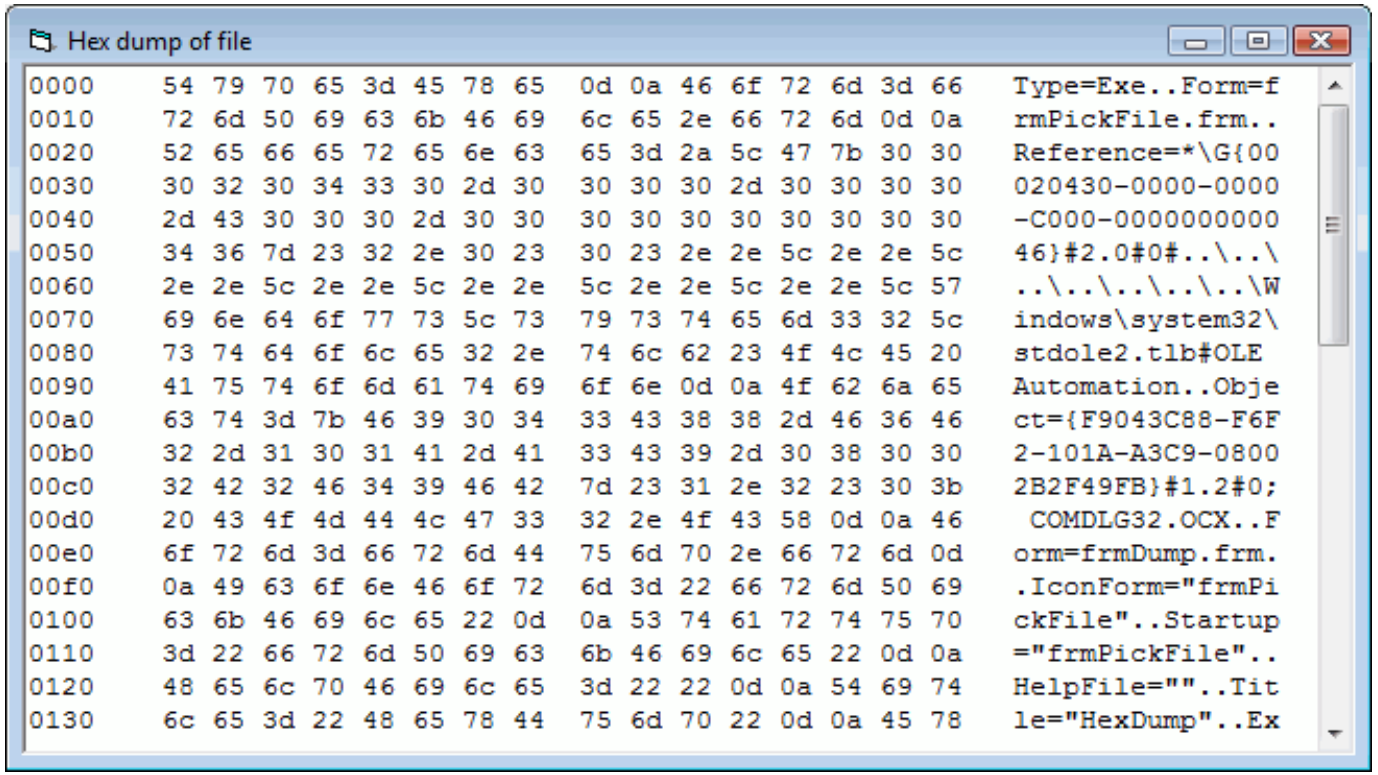
Binary vs. Textual

Two fundamental types of representations

- **Binary**
  - Low-level format
  - Machine-readable
  - Typically optimized for efficiency (disk space, …)
- **Textual**
  - Human-readable
  - Directly editable (e.g., text editor)

Example Binary Data

(as seen in a hex editor)



Binary vs. Textual

If you go down low enough, all representations are binary

Conversion conventions from text to binary:

- ASCII
- Unicode (UTF-8, UTF-16)
- Plus many, many more (Windows Latin 1, GB18030, …)

More info:

- http://kunststube.net/encoding/

Unstructured vs. Semi-Structured vs. Structured

(Highly) structured data:

There is a **predefined schema** that all data items adhere to. Deviations, if allowed at all, are highly regulated (e.g., NULL values).

Example: relational model

Unstructured data:

There are **no rules or assumptions** about what each data item looks like. Two data items can be very similar, or entirely different (a JPG image vs. a freeform text).

Example: file system, Key/Value Stores

Semi-structured data:

There is **no strict schema**, but there are rules that govern the structure of data (e.g., needs to be JSON, specific fields need to be available). Documents are often assumed to be **self-descriptive**.

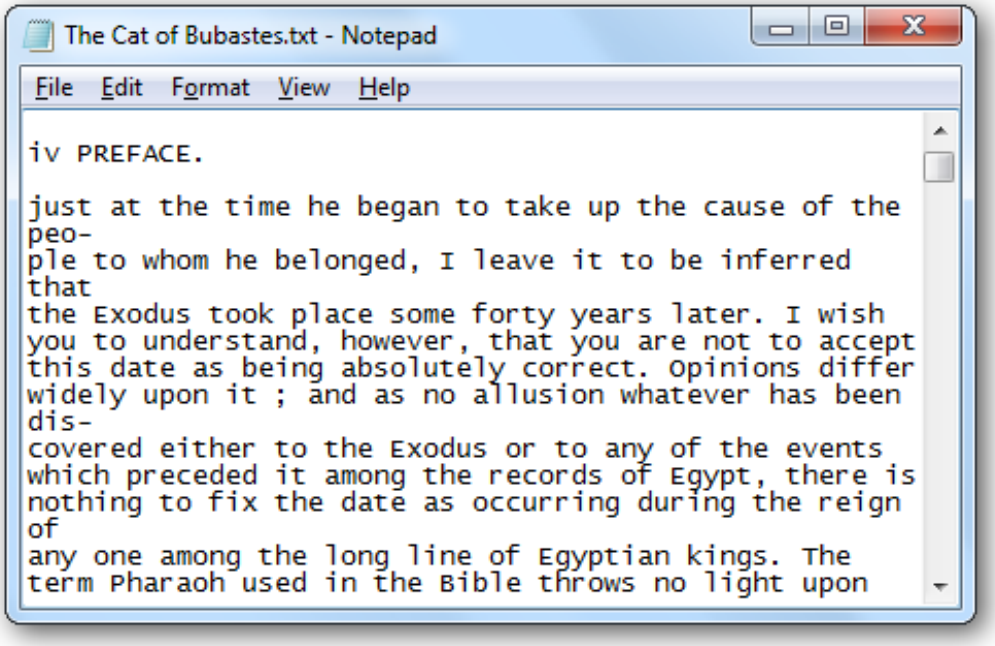
Example: document stores

Unstructured Data - Plain Text

No defined semantics on top of character encodings

All “meaning” is external

- In the head of a human
- In an external programme



Semi-Structured Data

"Middle ground" between unstructured text and structured info

No pre-defined rules, but data that is "self-explanatory"

Semi-structured documents are a mixture of **data and meta-data**

OSM xml.xml | Main Form.aspx.cs | Main Form.aspx

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenstreetServer">
  <node lat="30.866056900000004" lon="75.8824308">
    <tag k="amenity" v="Atm" />
    <tag k="Name" v="HDFC ATM" />
    <tag k="addr:StreetNo." v="2" />
    <tag k="addr:HouseNo." v="32" />
    <tag k="addr:City" v="jagraon" />
    <tag k="addr:state" v="Punjab" />
    <tag k="addr:country" v="Indian" />
    <tag k="PhoneNo." v="326" />
    <tag k="Website" v="www.example.com" />
    <tag k="comment" v="Adding POI" />
    <tag k="created_by" v="Er. Pawan verma" />
  </node>
</osm>
```

Examples of Semi-Structured Data Formats

Comma-**S**eparated **V**alues (CSV)

e**X**tensible **M**arkup **L**anguage (XML)

Java**S**cript **O**bject **N**otation (JSON)

YAML **A**in't **M**arkup **L**anguage (YAML)

(**R**esource **D**escription **F**ramework (RDF))

CSV Files

Comma-**S**eparated **V**alues (CSV)

Format:

```
[HEADER1,HEADER2, … HEADERn]
VALUE  1,VALUE 2, … VALUE n
VALUE  3,VALUE 4, … VALUE m
VALUE  5,VALUE 6, … VALUE o
```

(sometimes different separators are used, e.g., “;”)

CSV Files - Advantages and Disadvantages

Advantages:

- Very simple - baseline format for exchange of tabular data
  - E.g., MS Excel can also export / import CSV data
- Trivial to parse from our own programs
- Fairly low-overhead (basically no markup except a header)

Disadvantages:

- Low expressiveness
- Can only represent columnar data
- Problematic for sparse data

XML

The “original” semi-structured data representation format:

- Either the **basis** (RDF) or **spiritual predecessor** (JSON, YAML) of most other semi-structured representation formats

Itself a subset of a much older language (SGML)

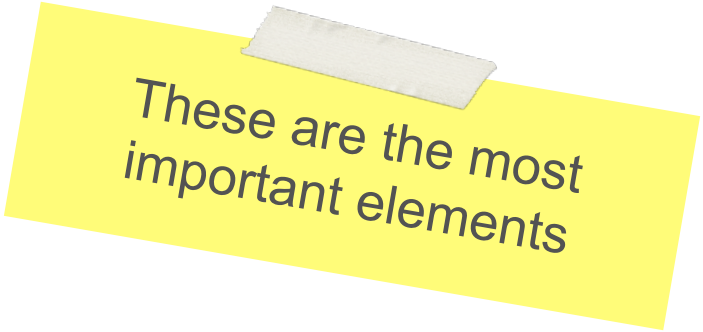
Basic concept:

- Markup language
- Data is interweaved with meta-data (which describes the meaning of the data)

XML Example

```
<order id=''X123''>
   <customer account="10">
      <name>Thomas Edison</name>
      <telephone>123456123</telephone>
   ...
   </customer>
   <items>
      <item id=''CD1''>
         <name>cd player</name>
         <unitprice>60.6</unitprice>
         <quantity>1</quantity>
      </item>
   ...
   </items>
</order>
```

Elements of an XML Document



Begin and End Tags:

```
<aaa> </aaa> or <aaa />
```

(the latter is a combined begin/end without subelements)

Attributes:

```
<aaa b="bbb" />
```

Text nodes:

```
<aaa> This is some text content </aaa>
```

Further XML Elements

Document Preamble:

```
<?xml version="1.0" encoding="utf-8"?>
```

Namespaces:

Tag prefix: `<myns:aaa />`

Comments:

```
<!-- This is an XML comment -->
```

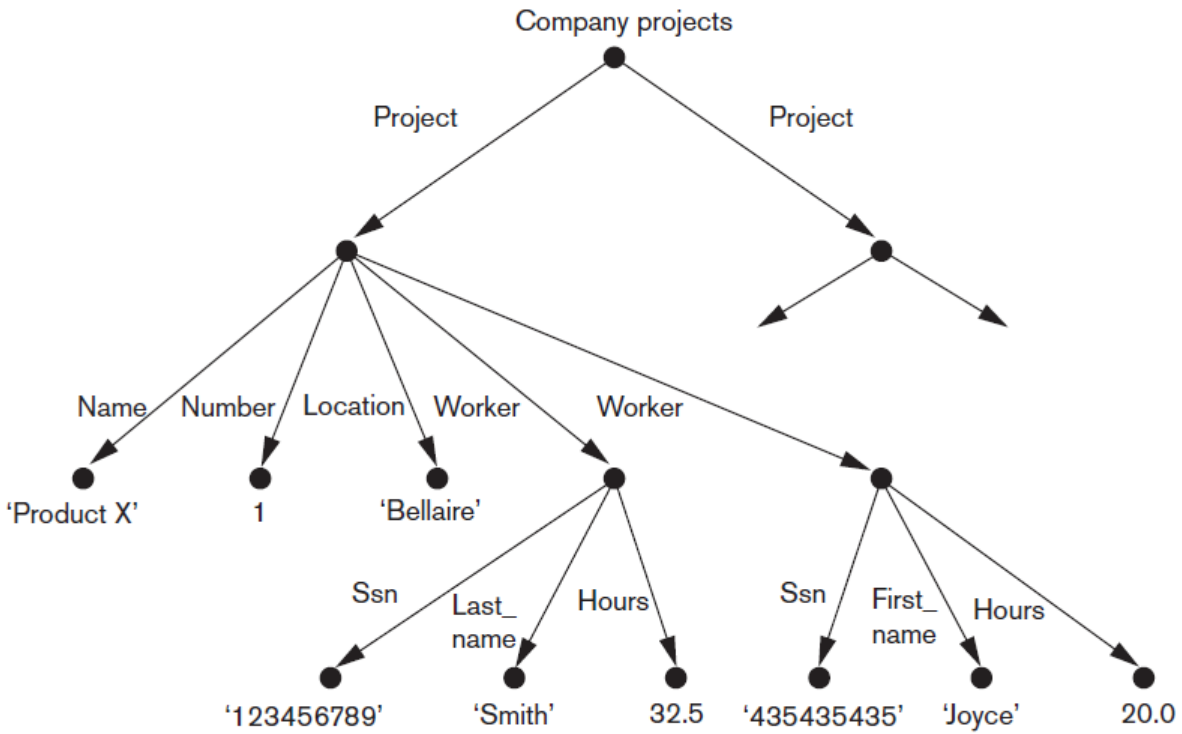
Processing Entities:

```
<aaa> A is &lt; B </aaa>
```

CDATA:

```
<![CDATA[  This is some block-escaped text data ]]>
```

Hierarchical Data Model



Well-Formedness and Validity

Two characteristics for XML documents:

Well-Formed

An XML document is said to be well-formed if it follows all rules of XML

Otherwise most parsers will fail parsing the document

Valid

An XML document is said to be valid if it has a schema definition and is true to it

Rules for Well-Formedness

- Document needs to have exactly one root tag
- Tags:
  - All tags are closed
  - The start and end tags are nested correctly

    ```
    <a><b>text</a></b>
    ```
  - Tags (actually: XML in general) are case-sensitive

    ```
    <a><b>text</A></B>
    ```
  - Tag and attribute names do not have whitespace in them

    ```
    <a bcd />
    ```
- Attributes:
  - Values need to be quoted: `<a att=12 />`
  - Attribute names need to be unique for each tag `<a a="1" a="2" />`
- *(plus a bunch of other less important syntax rules we are not covering)*

Short In-Class Exercise

Find the **10** errors in the given XML document that make this document not well-formed.

Document Definitions and XML Schema

There are a number of different languages for defining the **schema** of a given family of XML documents

- Essentially making these XML documents fully structured

The most important languages:

- **DTD** (oldest version, pretty much outdated by now)
- **XML Schema** (de-facto standard)
- **RelaxNG** (slightly simpler for some types of documents)

DTD Example

(a)
```
<!DOCTYPE Projects [
    <!ELEMENT Projects (Project+)>
    <!ELEMENT Project (Name, Number, Location, Dept_no?, Workers)>
        <!ATTLIST Project
            ProjId ID #REQUIRED>
    <!ELEMENT Name (#PCDATA)>
    <!ELEMENT Number (#PCDATA)
    <!ELEMENT Location (#PCDATA)>
    <!ELEMENT Dept_no (#PCDATA)>
    <!ELEMENT Workers (Worker*)>
    <!ELEMENT Worker (Ssn, Last_name?, First_name?, Hours)>
    <!ELEMENT Ssn (#PCDATA)>
    <!ELEMENT Last_name (#PCDATA)>
    <!ELEMENT First_name (#PCDATA)>
    <!ELEMENT Hours (#PCDATA)>
]>
```

XML Schema Example

```
<xsd:element name="root">
    <xsd:sequence>
    <xsd:element name="course" minOccurs="0" maxOccurs="unbounded">
        <xsd:sequence>
            <xsd:element name="cname" type="xsd:string" />
            <xsd:element name="cnumber" type="xsd:unsignedInt" />
            <xsd:element name="section" minOccurs="0" maxOccurs="unbounded">
                <xsd:sequence>
                    <xsd:element name="secnumber" type="xsd:unsignedInt" />
                    <xsd:element name="year" type="xsd:string" />
                    <xsd:element name="quarter" type="xsd:string" />
                    <xsd:element name="student" minOccurs="0" maxOccurs="unbounded">
                        <xsd:sequence>
                            <xsd:element name="ssn" type="xsd:string" />
                            <xsd:element name="sname" type="xsd:string" />
                            <xsd:element name="class" type="xsd:string" />
                            <xsd:element name="grade" type="xsd:string" />
                        </xsd:sequence>
                    </xsd:element>
                </xsd:sequence>
            </xsd:element>
        </xsd:sequence>
    </xsd:element>
    </xsd:sequence>
    </xsd:element>
```

XML Ecosystem

There is an entire zoo of XML languages associated with the validation, processing, and transformation of documents:

Most of those (but not all) are themselves defined in XML

Linking:

XLink

Schema definition:

DTD, XML Schema, RelaxNG, Schematron

Transformation / querying:

XPath, XSLT, XQuery

Displaying / formatting:

CSS

XML Ecosystem

The flexibility and expressiveness of XML has seen that it has become the implementation basis for **a lot** of other languages

Typically those are defined via XML Schema and use their own namespace

Wikipedia lists close to concrete 250 languages

https://en.wikipedia.org/wiki/List_of_XML_markup_languages

XML Languages - Example 1: XHTML

```
<?xml version="1.0"?>

<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
      <title>Title of document</title>
    </head>
    <body>
      some content
    </body>
</html>
```

XML Languages - Example 2: XML-RPC

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

XML Languages - Example 3: ATOM

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns=“http://www.w3.org/2005/Atom">
    <title>Example Feed</title>
    <subtitle>A subtitle.</subtitle>
    <link href="http://example.org/feed/" rel="self" />
    <link href="http://example.org/" />
    <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
    <updated>2003-12-13T18:30:02Z</updated>

    <entry>
          <title>Atom-Powered Robots Run Amok</title>
          <link href="http://example.org/2003/12/13/atom03" />
          <!-- more content -->
    </entry>
</feed>
```

XML Languages - Example 4: RDF

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:cd="http://www.recshop.fake/cd#">

      <rdf:Description
            rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
              <cd:artist>Bob Dylan</cd:artist>
              <cd:country>USA</cd:country>
              <cd:company>Columbia</cd:company>
              <cd:price>10.90</cd:price>
              <cd:year>1985</cd:year>
      </rdf:Description>

</rdf:RDF>
```

XML - Advantages and Disadvantages

XML is **very** powerful, but this comes with a price:

- Verbose (start and close tags, lots of metadata)
- Not particularly easy to read for humans
- Complexity
  - Of XML itself (namespaces, entities, …)
  - And of some support tech (namely XML Schema and XQuery)

Lightweight Semi-Structured Data

Given these problems with XML, some alternatives have been proposed in the last years

- Keep the core ideas (e.g., hierarchical structure)
- But remove most of the heavy-weight support technology

Examples:

- JSON
- YAML

JSON

JSON (JavaScript Object Notation)

Basically serialized JavaScript objects

Particularly easy to use from JavaScript, but nowadays libraries for most programming languages

JSON Syntax

Basic syntax:

```
{
    KEY1: VALUE1, KEY1: VALUE2, … KEYn: VALUEn,
}
```

`KEY`: string

`VALUE`: one of simple type, list (`[ ]`), complex (`{ }`), or `null`

JSON Syntax

Supported simple types:

- Number
- String
- Boolean

No support for attributes, namespaces, entities, …

JSON Example

```
{
        "firstName": "John",
        "lastName": "Smith",
        "isAlive": true,
        "age": 27,
        "address": {
          "streetAddress": "21 2nd Street",
          "city": "New York"
        },
        "phoneNumbers": [
          {
            "type": "home",
            "number": "212 555-1234"
          },
          {
            "type": "office",
            "number": "646 555-4567"
          }
        ],
        "children": [],
        "spouse": null
}
```

JSON - Advantages and Disadvantages

JSON is:

- Much simpler than XML
- (A little) easier to read

But still a big soup of brackets and special characters

YAML

YAML:

- "YAML ain't a markup language" *or*
- "Yet another markup language"

Superset of JSON

- All legal JSON documents are also legal YAML documents
- But: removes need for most brackets, significant whitespace

Core usage:

- Human-readable config files

YAML

Basic structure is still

```
key: value
```

But quotes are now optional, and whitespace is used to indicate nesting

Dash symbol (-) used to indicate list entries

```
firstName: John
lastName: Smith
age: 25
address:
  streetAddress: 21 2nd Street
  city: New York
  state: NY
  postalCode: '10021'
phoneNumber:
- type: home
  number: 212 555-1234
- type: fax
  number: 646 555-4567
gender:
  type: male
```

Key Takeaways (1)

Semi-structured data mixes data and meta-data in a document

Difference between textual and binary data

Different types of semi-structured data:

- XML
- JSON
- YAML

Key Takeaways (2)

XML well-formedness and validity

Core XML syntax rules:

- Tags, attributes, and text nodes
- Every start tag needs an end tag
- Correct nesting of tags

Examples of languages implemented on top of XML