ETH zürich

Ghislain Fourny

Big Data for Engineers Spring 2020

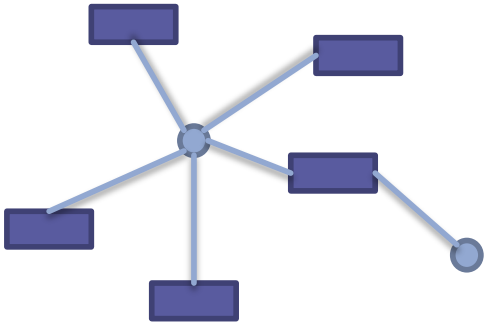
5. Syntax

Introduction

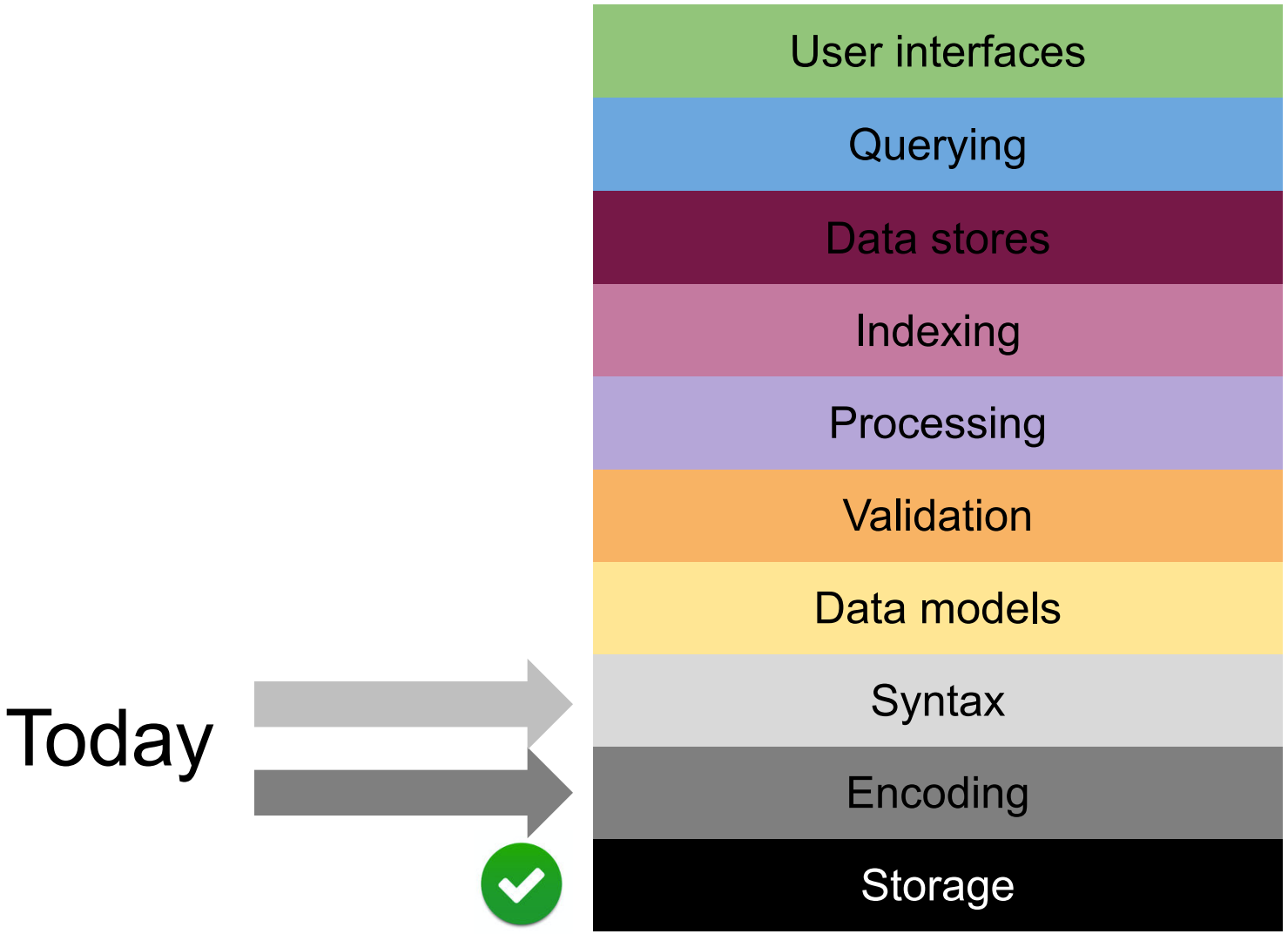
Data Shapes

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam vel erat nec dui aliquet vulputate sed quis nulla. Donec eget ultricies magna, eu dignissim elit. Nullam sed urna nec nisl rhoncus ullamcorper placerat et enim. Integer varius ornare libero quis consequat. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean eu efficitur orci. Aenean ac posuere tellus. Ut id commodo turpis.

Praesent nec libero metus. Praesent at turpis placerat, congue ipsum eget, scelerisque justo. Ut volutpat, massa ac lacinia cursus, nisl dui volutpat arcu, quis interdum sapien turpis in tellus. Suspendisse potenti. Vestibulum pharetra justo massa, ac venenatis mi condimentum nec. Proin viverra tortor non orci suscipit rutrum. Phasellus sit amet euismod diam. Nullam convallis nunc sit amet diam suscipit dapibus. Integer porta hendrerit nunc. Quisque pharetra congue porta. Suspendisse vestibulum sed mi in euismod. Etiam a purus suscipit, accumsan nibh vel, posuere ipsum. Nulla nec tempor nibh, id venenatis lectus. Duis lobortis id urna eget tincidunt.

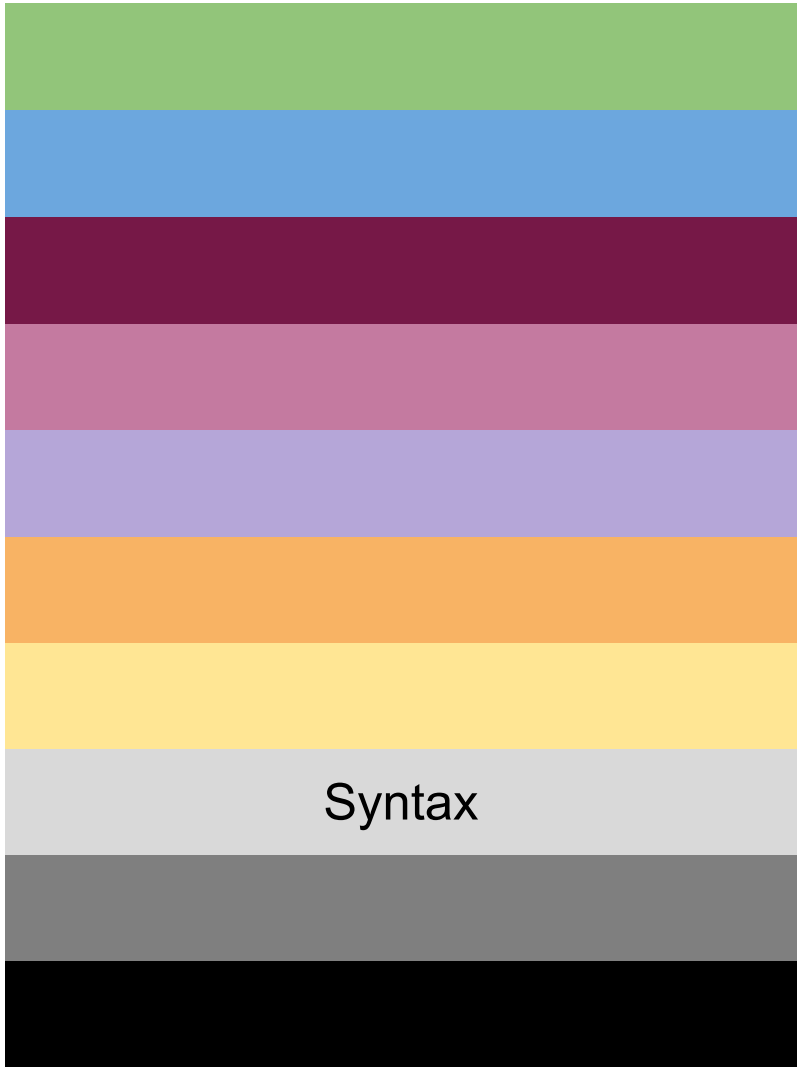


The stack


User interfaces
Querying
Data stores
Indexing
Processing
Validation
Data models
Syntax
Encoding
Storage
Today

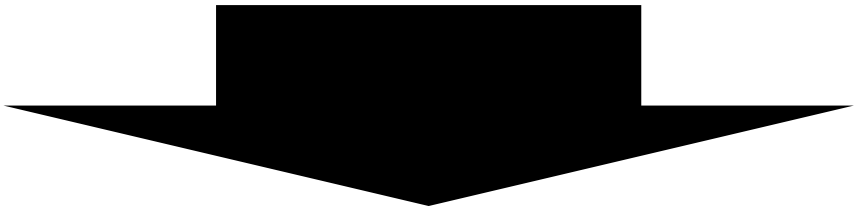
The stack: Syntax

Text
CSV
XML
JSON
RDF/XML
Turtle
XBRL



CSV (Comma separated values)

```
ID,Last name,First name
1,Einstein,Albert
2,Gödel,Kurt
```

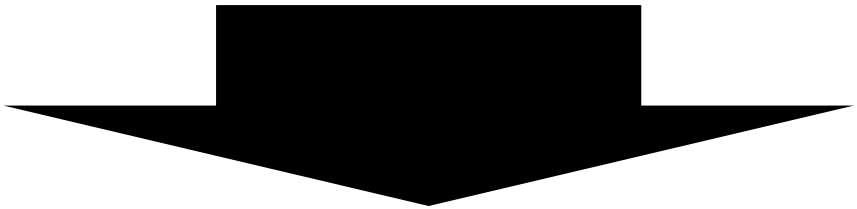


ID	Last name	First name
1	Einstein	Albert
2	Gödel	Kurt

CSV (Comma separated values)

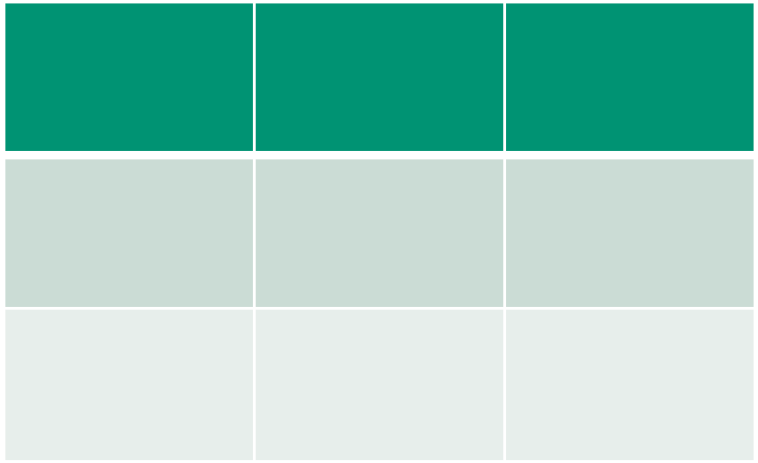
```
ID,Last name,First name,Theory,
1,Einstein,Albert,"General, Special Relativity"
2,Gödel,Kurt,"""Incompleteness"" Theorem"
```

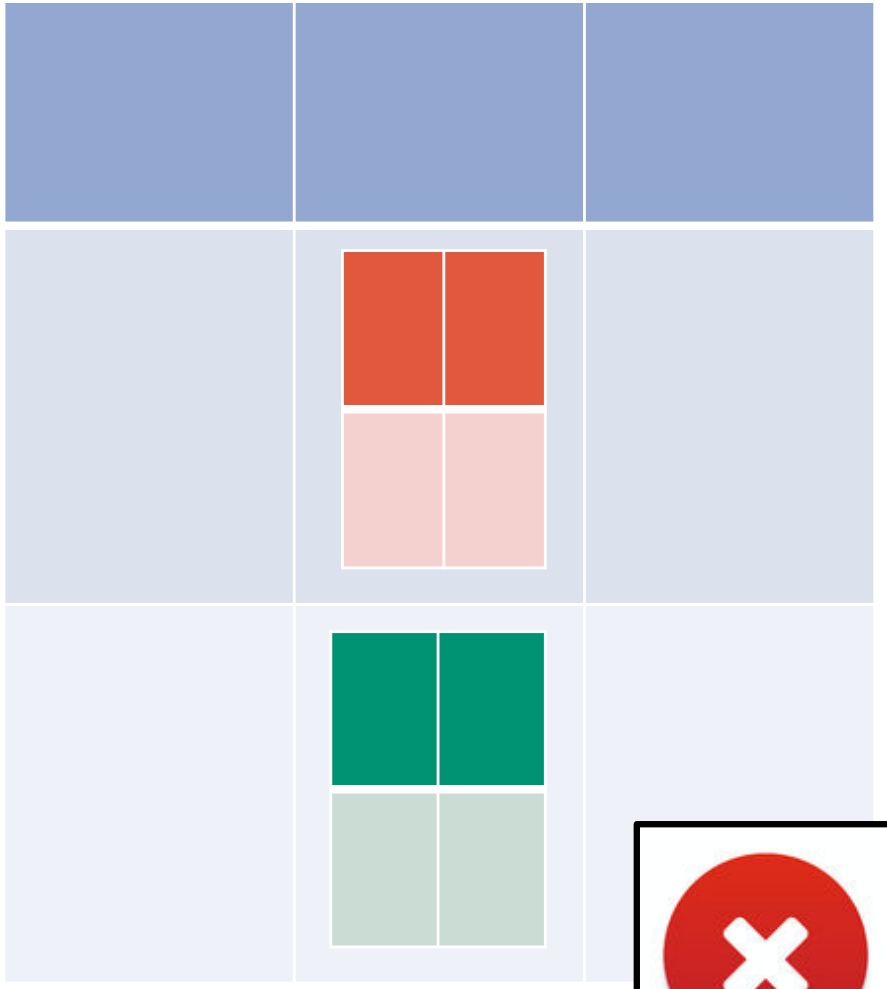
RFC 4180
(but lose in practice)



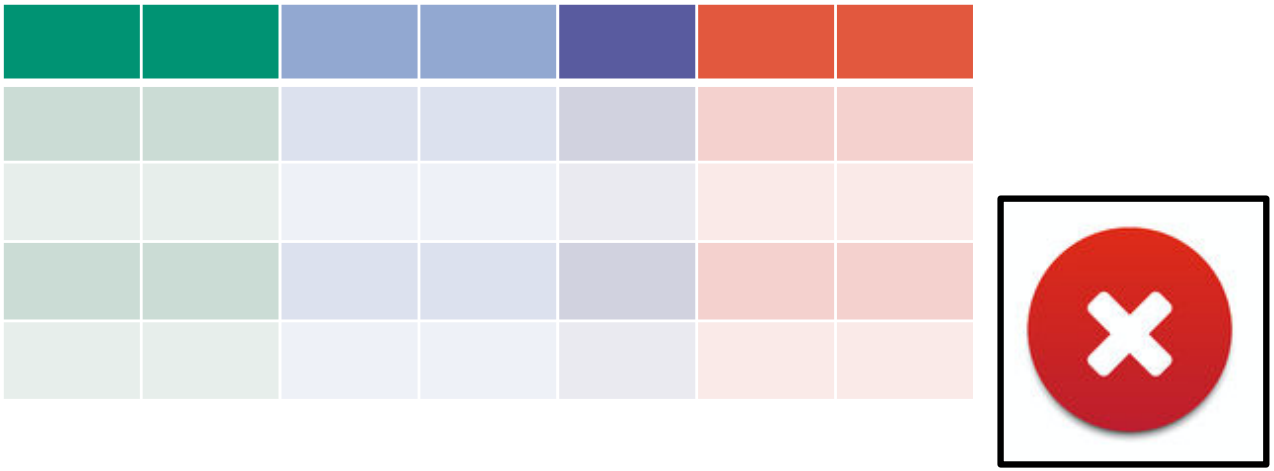
ID	Last name	First name	Theory
1	Einstein	Albert	General, Special Relativity
2	Gödel	Kurt	"Incompleteness" Theorem

1st Normal Form (Atomic integrity)





Boyce-Codd Normal Form



Data Denormalization (NoSQL)

1st normal form

2nd, 3rd, Boyce-Codd normal form

Normalization

Highly normalized

Avoid update anomalies!

Highly denormalized

Avoid joins!

Example

products	
product	price
varchar(30)	char(1)
Phone	800
Laptop	2000
HDTV	1000
USB Stick	10

sales		
product	customer	quantity
varchar(30)	text	date
Phone	John	1
Phone	Peter	2
Phone	Mary	1
Laptop	John	3
Laptop	Mary	1
HDTV	Mary	2

customers
customer
text
John
Peter
Mary
Bill

Boyce-Codd normal form

products	
product	price
varchar(30)	char(1)
Phone	800
Laptop	2000
HDTV	1000
USB Stick	10

sales		
product	customer	quantity
varchar(30)	text	date
Phone	John	1
Phone	Peter	2
Phone	Mary	1
Laptop	John	3
Laptop	Mary	1
HDTV	Mary	2

customers
customer
text
John
Peter
Mary
Bill

Denormalized to first normal form

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	date
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

Collection of tuples

Tuple: example

$$\begin{array}{lcl} \text{product} & \mapsto & \text{Phone} \\ \text{price} & \mapsto & 800 \\ \text{customer} & \mapsto & \text{John} \\ \text{quantity} & \mapsto & 1 \end{array}$$

$\mathbb{S}$ $\mathbb{V}$

Tuple: Syntax

```
{
  "product" : "Phone",
  "price" : 800,
  "customer" : "John",
  "quantity" : 1
}
```

Syntax of a collection of tuples

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	integer
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

```
{ "product" : "Phone", "price" : 800, "customer" : "John", "quantity" : 1 }
{ "product" : "Phone", "price" : 800, "customer" : "Peter", "quantity" : 2 }
{ "product" : "Phone", "price" : 800, "customer" : "Mary", "quantity" : 1 }
{ "product" : "Laptop", "price" : 2000, "customer" : "John", "quantity" : 1 }
{ "product" : "Laptop", "price" : 2000, "customer" : "Mary", "quantity" : 1 }
...
```

Nestedness

sales

product	orders
varchar(30)	text
Phone	customer (text) / quantity (integer): John 1; Peter 2; Mary 1
Laptop	customer (text) / quantity (integer): John 3; Mary 1
HDTV	customer (text) / quantity (date): Mary **1**

Syntax for nestedness

sales	
product	orders
varchar(30)	text
Phone	customer | quantity text | integer John | 1 Peter | 2 Mary | 1

```
{
  "product" : "Phone",
  "orders" : [
    { "customer" : John, "quantity" : 1 },
    { "customer" : Peter, "quantity" : 2 },
    { "customer" : Mary, "quantity" : 2 }
  ]
}
```

The denormalizing road from SQL to NoSQL

A	B	C	D
a	1	alpha	foo
a	2	NULL	foo
b	3	alpha	NULL
b	4	beta	NULL

```
{
  "A" : "a",
  "E" : [
    { "B" : 1, "C" : "alpha" },
    { "B" : 2 }
  ],
  "D" : "foo"
}
```

```
{
  "A" : "b",
  "E" : [
    { "B" : 3, "C" : "alpha" },
    { "B" : 4, "C" : "beta" }
  ]
}
```

Homogeneous collection of **flat** items.

Relational database

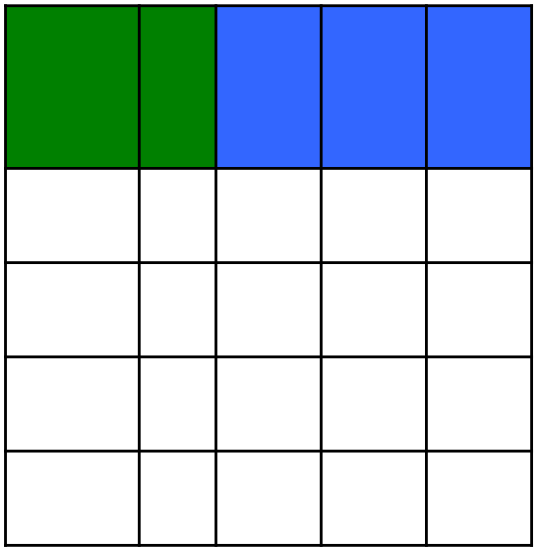
Heterogeneous collection of **arborescent** items.

Document store

gajus / 123RF Stock Photo

Semi-Structured Documents

Semi-Structured Documents

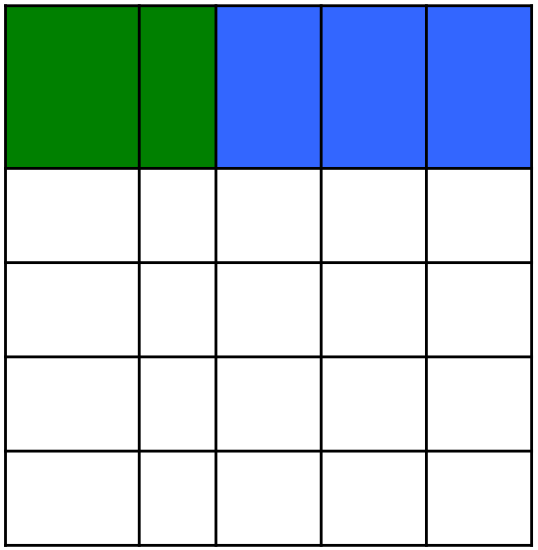


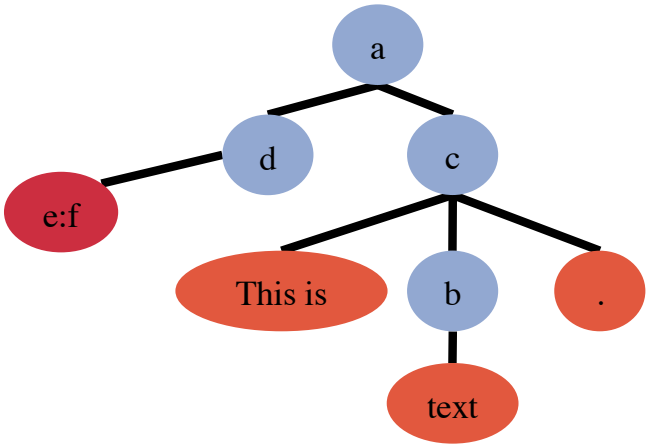
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Structured

Unstructured

Semi-Structured Documents





Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Structured Semi-structured Unstructured

Standards

For whom?

Syntax

Well-formedness

Well-formedness

One syntax = one language

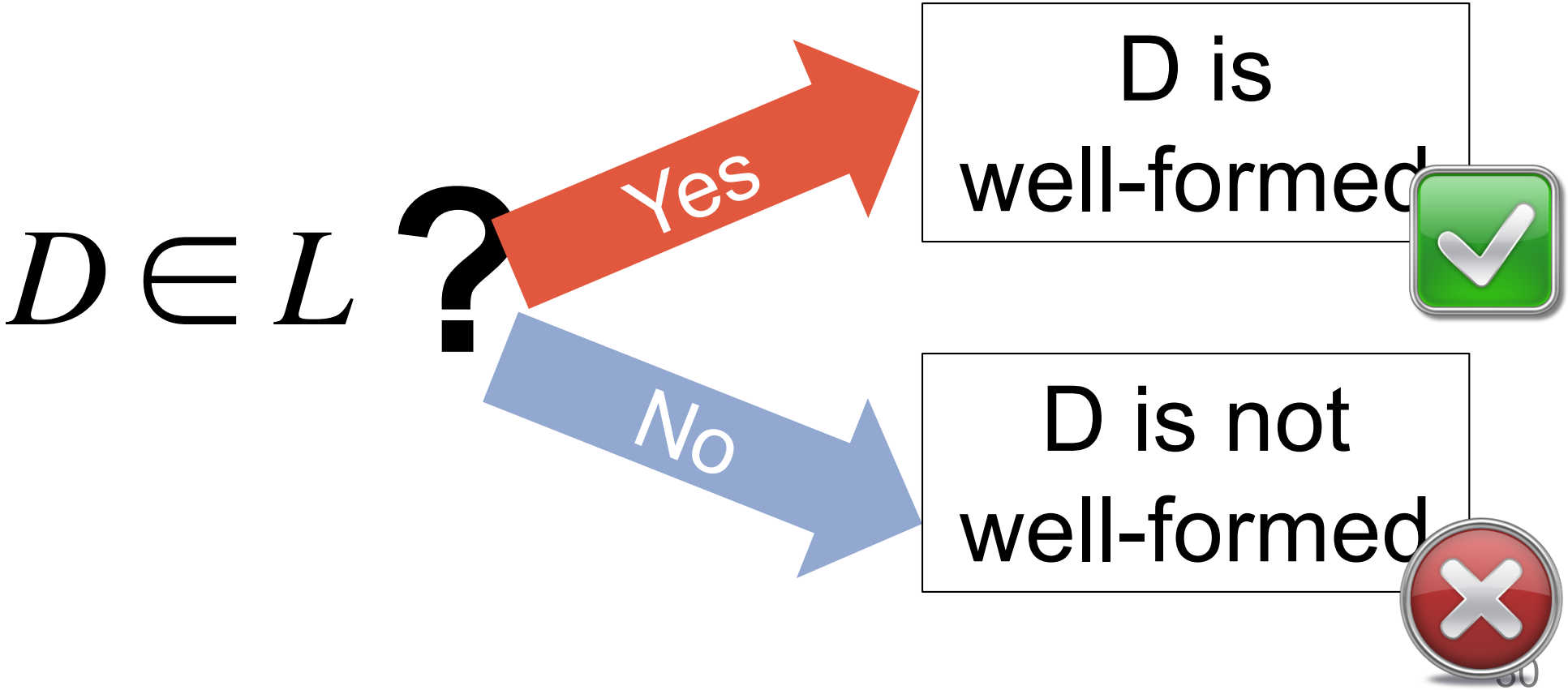
Well-formedness

One syntax = one language

$$D \in L \, ?$$

Well-formedness

One syntax = one language



HTML

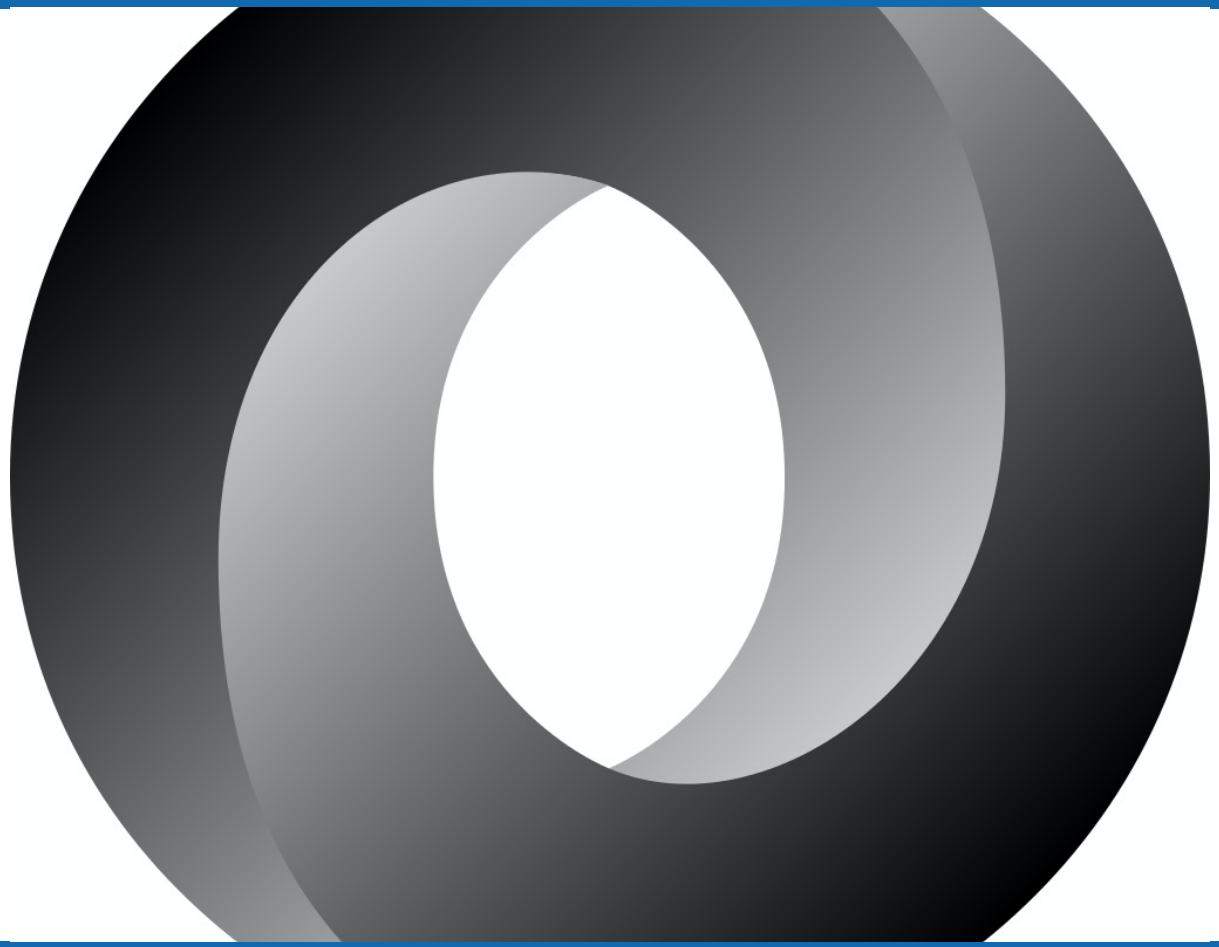
```
<!DOCTYPE html>
<html>
  <head>
    <title>Country</title>
  </head>
  <body>
    <h1 class="Title">Switzerland</h1>
    <div>Population: 8014000<br>
      Currency: Swiss Franc (CHF)</div>
    <h2>Cities</h2>
    <ul>
      <li>Zurich</li>
      <li>Geneva></li>
      <li>Bern <!-- the Federal City --></li>
    </ul>
  </body>
</html>
```

XML

```
<?xml version="1.0"?>
<country code="CH">
  <name>Switzerland</name>
  <population>8014000</population>
  <currency code="CHF">Swiss Franc</currency>
  <cities>
    <city>Zurich</city>
    <city>Geneva></city>
    <city>Bern <!-- the Federal City --></city>
  </cities>
  <description>
    We produce <b>very</b> good chocolate.
  </description>
</country>
```

JSON

```
{
  "code": "CH",
  "name": "Switzerland",
  "population": 8014000,
  "currency": {
    "name": "Swiss Franc",
    "code": "CHF"
  },
  "confederation": true,
  "president" : "Ueli Maurer",
  "capital": null,
  "cities": [ "Zurich", "Geneva", "Bern" ],
  "description": "We produce very good chocolate."
}
```



JSON

JSON: String

```
"foo"
```

```
"foo\nbar\u005f"
```

JSON: Number

```
3.1415
```

```
-1.2345E+5
```

JSON: Boolean

```
true
```

```
false
```

JSON: Null

```
null
```

JSON: Array

```
[
    3.14159265368979,
    true,
    "This is a string",
    { "foo" : false },
    null
]
```

JSON: Object

```
{
   "foo": 3.14159265368979,
   "bar": true,
   "str": "This is a string",
   "obj": { "school" : "ETH"},
   "Q": null
}
```

JSON: Well-formedness

```
{ "foo" : "bar", "foo" : "bar2" }
```

JSON: Well-formedness

```
{ "foo" : "bar", "foo" : "bar2" }
```

```
{ "foo" : "bar", "bar" : "foo" }
```

(SHOULD)

JSON: Well-formedness

```
{ [ 1 ] : "bar",  2 : "bar2" }
```

JSON: Well-formedness

```
{ [ 1 ] : "bar",  2 : "bar2" }
```

```
{ "1" : "bar", "2" : "foo" }
```

JSON: Well-formedness

```
{ foo: "bar",  bar: "bar2" }
```

JSON: Well-formedness

```
{ foo: "bar",  bar: "bar2" }
```

```
{ "foo" : "bar", "bar" : "foo" }
```

JSON: Well-formedness

JSON as Data


```
{
  "target": "Italian",
  "sample": "af0e25c7637fb0dcdc56fac6d49aa55e",
  "choices": ["Chinese", "German", "Italian", "English"],
  "guess": "German",
  "date": "2018-11-07",
  "country": "CH"
}
```

Syntax of a collection of tuples

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	integer
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

```
{ "product" : "Phone", "price" : 800, "customer" : "John", "quantity" : 1 }
{ "product" : "Phone", "price" : 800, "customer" : "Peter", "quantity" : 2 }
{ "product" : "Phone", "price" : 800, "customer" : "Mary", "quantity" : 1 }
{ "product" : "Laptop", "price" : 2000, "customer" : "John", "quantity" : 1 }
{ "product" : "Laptop", "price" : 2000, "customer" : "Mary", "quantity" : 1 }
...
```

Syntax for nestedness

sales	
product	orders
varchar(30)	text
Phone	customer | quantity text | integer John | 1 Peter | 2 Mary | 1

```
{
  "product" : "Phone",
  "orders" : [
    { "customer" : "John", "quantity" : 1 },
    { "customer" : "Peter", "quantity" : 2 },
    { "customer" : "Mary", "quantity" : 2 }
  ]
}
```

Robert Eastman / 123 Stock Photo

XML

XML: Element

```
<foo>[more XML]</foo>
```

XML: Element

```
<foo>[more XML]</foo>
```

```
<bar/> = <bar></bar>
```

XML: Element

```
<foo>[more XML]</foo>
```

opening tag

closing tag

```
<bar/> = <bar></bar>
```

empty tag

XML: Attribute

```
<a attr="value"/>
```

XML: Text

```
<a>This is text</a>
```

Also (but out of scope)

Comments

```
<!-- This is a comment -->
```

Processing instructions

```
<?myapp do whatever ?>
```

Text declaration

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

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
<foo/>
```

XML: Well-formedness

```
<foo/>
```

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
<foo/>
<bar/>
```

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>
  <bar/>
</foo>
```

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
text
<foo>
  <bar/>
</foo>
text
```

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
<foo>
  text <bar/> text
</foo>
```

XML: Well-formedness

```
<?xml version="1.0" encoding="UTF-8"?>
<foo <element/>>
  <bar></bar>
</foo>
```

XML: Document Type

```
<?xml version="1.0"?>
<!DOCTYPE document>
<document>
Lorem ipsum dolor sit amet, consectetur adipiscing
elit, sed do eiusmod tempor incididunt ut labore et
dolore magna aliqua.
</document>
```

What Appears Where?

	Top-Level	Between Element Tags	Inside Opening Element Tag
Elements	once		
Attributes			
Text			

XML: Well-formedness

```
<a foo="bar" foo="bar2"/>
```

XML: Well-formedness

```
<a foo="bar" foo="bar2"/>
```

```
<a foo="bar" bar="foo"/>
```

XML: Well-formedness

```
<a><b></a></b>
```

XML: Well-formedness

```
<a><b></a></b>
```

```
<a><b></b></a>
```

XML: Well-formedness

```
<a>1 < 2</a>
```

XML: Well-formedness

```
<a>1 < 2</a>
```

```
<a>1 &lt; 2</a>
```

XML: Entity References

```
<?xml version "1.0"?>
<document>
2 &lt; 3
</document>
```

XML: Entity References

<  <

XML: Entity References

>  >

XML: Entity References

'  '

XML: Entity References

"  "

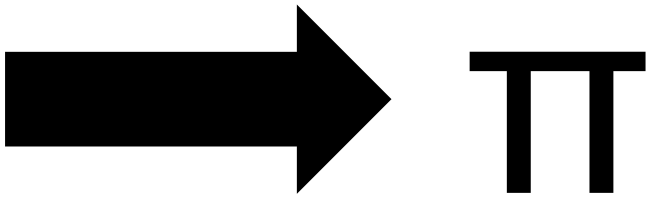
XML: Entity References

&  &

XML: Character References (dec)

<?xml version "1.0"?>
<document>
Lorem ipsum dolor sit amet,
consectetur adipiscing elit, sed
do

π



π

eiusmod tempor incididunt ut
labore et dolore magna aliqua.
</document>

XML: Well-formedness

```
<a attr="a "quote""/>
```

XML: Well-formedness

```
<a attr="a "quote""/>
```

```
<a attr="a &quot;quote&quot;"/>
```

XML: Well-formedness

```
<a attr="an <element/>"/>
```

XML: Well-formedness

```
<a attr="an <element/>"/>
```

```
<a attr="an &lt;element/>"/>
```

XML Names

```
<1234/>
<a<b/>
<xml/>
```

XML Names

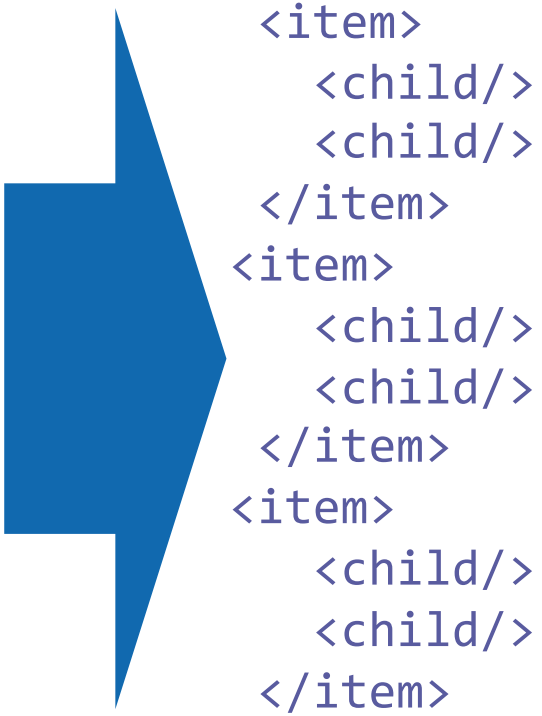
```
<1234/>
<a<b/>
<xml/>
```

```
<foo1234/>
<_bar/>
```

Whitespaces

This kind of whitespace
can usually safely
be ignored

(this is very convenient)



```
<items>
  <item>
    <child/>
    <child/>
  </item>
 <item>
    <child/>
    <child/>
  </item>
 <item>
    <child/>
    <child/>
  </item>
</items>
```

ASCII characters allowed in XML names

Control characters															
Control characters															
SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Allowed anywhere in name

Allowed but not at start

not allowed

(Not) Well-Formed XML

```
<?xml version="1.0" encoding="utf-16"?>
<movies>
  <movie id=”56225”>
    <title>Love Story</title>
    <title></title>
    <year>1980</year>
    <_director name='Coppola'></_director>
    <comment text=”Five start” text=”Average”/>
    <xml>Introduce XML content</xml>
    <newcomment text="An <important>
text">Oscar</newcomment>
    <comment lang=de>&copy; 1980 Warner Bros.</comment>
    <!-- Famous movie of the --80s -->
  </Movie>
</movies>
```

(Not) Well-Formed XML

```
<?xml version="1.0" encoding="utf-16"?>
<movies>
  <movie id=”6225”>
    <title>Love Story</title>
    <title></title>
    <year>1980</year>
    <_director name='Coppola'></_director>
    <comment text=”Five start” text=”Average”/>
    <xml>Introduce XML content</xml>
    <newcomment text="An <important>
text">Oscar</newcomment>
    <comment lang=de>&copy; 1980 Warner Bros.</comment>
    <!-- Famous movie of the --80s -->
  </Movie>
</movies>
```

Well-Formedness: How To Tell?

An editor (oXygen, ...) will tell you.

Well Formed XML

```
<?xml version="1.0" encoding="utf-16"?>
<!DOCTYPE movies [
<!ENTITY copy "&#169;">
]>
<movies>
    <Movie id="56225">
        <title>Love Story</title>
        <title></title>
        <year>1980</year>
        <_director name='Coppola'></_director>
        <comment text="Five start"/>
        <comment text="Average"/>
        <newcomment text="An &lt;important&gt;
                          text">Oscar</newcomment>
        <comment lang="de">&copy; 1980 Warner Bros.</comment>
        <!-- Famous movie of the 80s -->
    </Movie>
</movies>
```

Syntax of a collection of tuples

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	date
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

```
<sale>
  <product>Phone</product>
  <price>800</price>
  <customer>John</customer>
  <quantity>1</quantity>
</sale>
```

Syntax of a collection of tuples

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	date
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

```
<sales>
<sale><product>Phone</product><price>800</price><customer>John</customer><quantity>1</quantity></sale>
<sale><product>Phone</product><price>800</price><customer>Peter</customer><quantity>2</quantity></sale>
<sale><product>Phone</product><price>800</price><customer>Mary</customer><quantity>1</quantity></sale>
<sale><product>Laptop</product><price>200</price><customer>John</customer><quantity>3</quantity></sale>
...
</sales>
```

Syntax of a collection of tuples

sales			
product	price	customer	quantity
varchar(30)	char(1)	text	date
Phone	800	John	1
Phone	800	Peter	2
Phone	800	Mary	1
Laptop	2000	John	3
Laptop	2000	Mary	1
HDTV	1000	Mary	2

```
<?xml version "1.0"?>
<!DOCTYPE sales>
<sales>
  <sale>
    <product>Phone</product>
    <price>800</price>
    <customer>John</customer>
    <quantity>1</quantity>
  </sale>
  <sale>
    <product>Phone</product>
    <price>800</price>
    <customer>Peter</customer>
    <quantity>2</quantity>
  </sale>
  <sale>
    <product>Phone</product>
    <price>800</price>
    <customer>Mary</customer>
    <quantity>1</quantity>
  </sale>
...
</sales>
```

Nested syntax

sales	
product	orders
varchar(30)	text
Phone	customer / quantity text / date John / 1 Peter / 2 Mary / 1

```
<?xml version "1.0"?>
<!DOCTYPE sales>
<sales>
 <sale>
  <product>Phone</product>
  <price>800</price>
  <orders>
   <order>
    <customer>John</customer>
    <quantity>1</quantity>
   </order>
   <order>
    <customer>Peter</customer>
    <quantity>2</quantity>
   </order>
   <order>
    <customer>Mary</customer>
    <quantity>1</quantity>
   </order>
  </orders>
 <sale>
...
</sales>
```

Which syntax?

Other syntaxes

YAML (the "Python of JSON")

```yaml
%YAML 1.2
---
Country:
  code: 'CH'
  name: 'Switzerland'
  population: 8014000
  currency:
    name: 'Swiss Franc'
    code: 'CHF'
  confederation: true
  president : Ueli Maurer'
  capital: null
  cities:
    - 'Zurich'
    - 'Geneva'
    - 'Bern'
  description: 'We produce very good chocolate.'
```