

TDDD43 Advanced Data Models and Databases

XML (Extensible Markup Language)

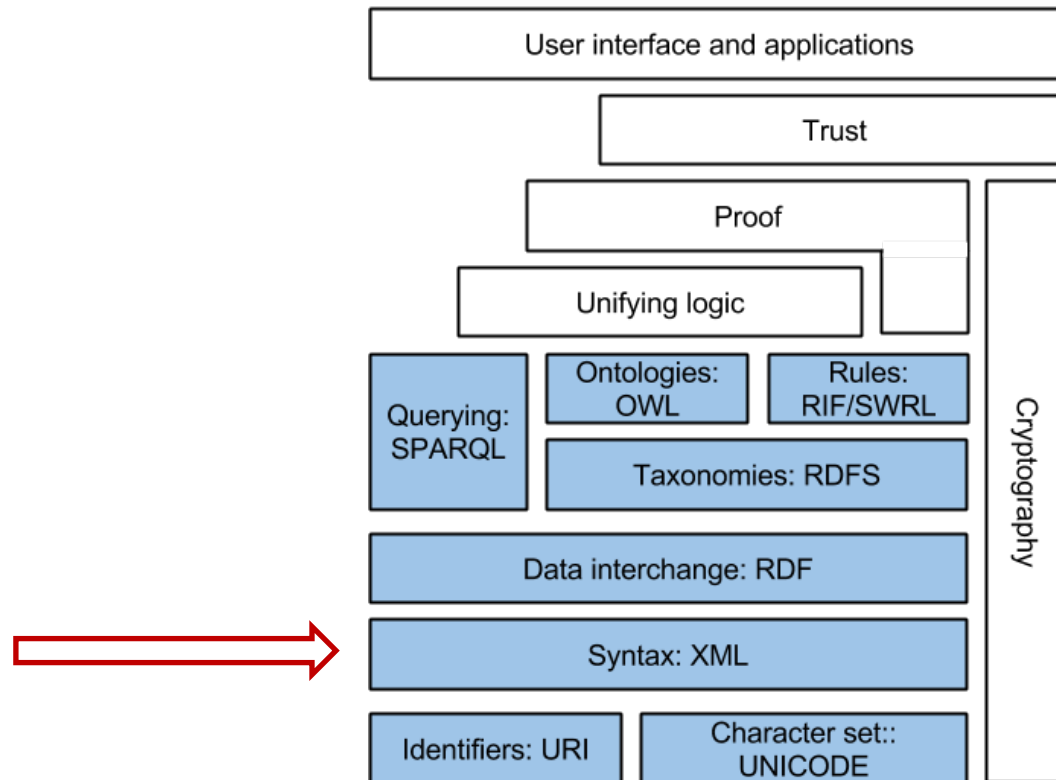
Huanyu Li
huanyu.li@liu.se

Recap

➤ Semantic Web

... to put machine-understandable data on the Web...

... data can be shared and processed by automated tools as well as by people...



Outline

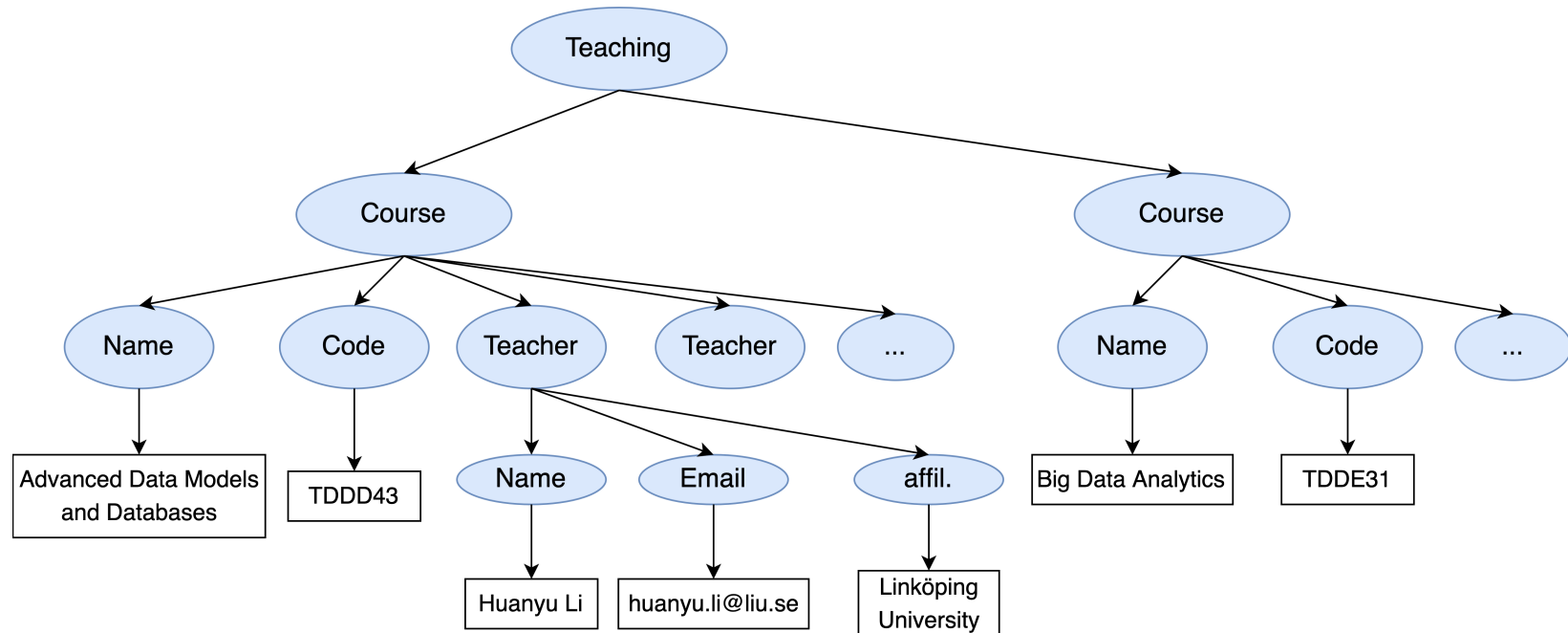
- XML
- Schema for XML
- Query languages for XML

Extensible Markup Language

- *A language and a file format for storing data, in an exchangeable (sharable, interoperable) way...*
- *One of many standardized formats recommended by W3C (World Wide Web Consortium)*
- *<https://www.w3schools.com/xml/default.asp>*

XML Building Blocks - Elements

- Organizes data as a tree
- Special markups to indicate different elements of the data
- Tags to annotate elements and hierarchy, with one root element



XML – Tags and Content

<Teaching>

Start tag

```
<Course>
  <Name>Advanced Data Models and Databases</Name>
  <Code>TDDD43</Code>
  <Teacher>
    <Name>Huanyu Li</Name>
    <Email>huanyu.li@liu.se</Email>
    <Affiliation>Linköping University</Affiliation>
  </Teacher>
  <Teacher>
    <Name>Patrick Lambrix</Name>
    <Email>patrick.lambrix@liu.se</Email>
    <Affiliation>Linköping University</Affiliation>
  </Teacher>
</Course>
<Course>
  <Name>Big Data Analytics</Name>
  <Code>TDDE31</Code>
</Course>
```

Content

</Teaching>

End tag

XML – Tags and Content

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher>
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
    <Teacher>
      <Name>Patrick Lambrix</Name>
      <Email>patrick.lambrix@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
  </Course>
  <Course>
    <Name>Big Data Analytics</Name>
    <Code>TDDE31</Code>
  </Course>
</Teaching>
```

Start tag


Content

End tag

XML – Attributes

- Elements can have attributes
- Attributes contain data that relates to an element
 - Listed in element's start tag
 - Value must be quoted

Attribute id with value 1



```
<Course id="1">  
  <Name>Advanced Data Models and Databases</Name>  
  <Code>TDDD43</Code>  
  <Teacher>  
    <Name>Huanyu Li</Name>  
    <Email>huanyu.li@liu.se</Email>  
    <Affiliation>Linköping University</Affiliation>  
  </Teacher>  
</Course>
```


XML – Empty Elements

- XML allows empty elements that may not have content


```
<Course id="1">  
  <Name>Advanced Data Models and Databases</Name>  
  <Code>TDDD43</Code>  
  <Teacher/>  
</Course>
```

- But an empty element can have attributes


```
<Course id="1">  
  <Name>Advanced Data Models and Databases</Name>  
  <Code>TDDD43</Code>  
  <Teacher id="t1"/>  
</Course>
```

XML – Well-formed Documents

- An XML document must meet some rules to be valid
 - Correct nesting
 - Each start tag has its corresponding end tag
 - Elements don't have an attribute multiple times
 - Etc...



```
<Course id="1">  
  <Name>Advanced Data Models and Databases</Name>  
  <Code>TDDD43</Course>  
</Code>
```



```
<Course id="1">  
  <Name>Advanced Data Models and Databases</CourseName>  
</Course>
```



```
<Course id="1">  
  <Code>TDDD43</Code>  
  <Teacher id="huali12" id="huali123"/>  
</Course>
```

Outline

✓ XML

➤ Schema for XML

➤ Query languages for XML

Document Type Definition (DTD)

- A DTD defines the document structure with a list of legal elements and attributes of an XML document
 - Elements – XML elements
 - Attributes – XML attributes
 - Entities – shortcuts to special characters
 - PCDATA – parsed character data
 - CDATA – character data
- https://www.w3schools.com/xml/xml_dtd_intro.asp

Document Type Definition (DTD)

- **!DOCTYPE** is used to define what is the root element of the XML data
- Syntax:

```
<!DOCTYPE root-element [element-declarations]>
```

```
<!DOCTYPE Teaching [  
    ...  
>  
  
    <Teaching>  
        <Course>  
            <Name>Advanced Data Models and Databases</Name>  
            <Code>TDDD43</Code>  
            <Teacher>  
                <Name>Huanyu Li</Name>  
                <Email>huanyu.li@liu.se</Email>  
                <Affiliation>Linköping University</Affiliation>  
            </Teacher>  
            <Teacher>  
                <Name>Patrick Lambrix</Name>  
                <Email>patrick.lambrix@liu.se</Email>  
                <Affiliation>Linköping University</Affiliation>  
            </Teacher>  
        </Course>  
    </Teaching>
```

Document Type Definition (DTD)

- **!ELEMENT** is used to define what are the composing elements of a nested element, or the type of a simple element
- Syntax: (declaring elements)

```
<!ELEMENT element-name CATEGORY>  
<!ELEMENT element-name (element-content)>
```

Examples:

```
<!ELEMENT element-name EMPTY>  
<!ELEMENT element-name (child1, child2)>  
<!ELEMENT element-name (#PCDATA)>  
<!ELEMENT element-name ANY>
```

Document Type Definition (DTD)

```
<!ELEMENT Teaching (Course+)>
<!ELEMENT Course (Name, Code, Teacher+)>
<!ELEMENT Teacher (Name, Email?, Affiliation*)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Code (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Affiliation (#PCDATA)>
```

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher>
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping
        University</Affiliation>
    </Teacher>
  </Course>
</Teaching>
```

Document Type Definition (DTD)

- The content of the Teaching element contains 1 or more Course elements (+)
- The content of the Teacher element can contain 0 or more Affiliation elements (*), 0 or 1 Email elements (?)

```
<!ELEMENT Teaching (Course+)>  
<!ELEMENT Course (Name, Code, Teacher+)>  
<!ELEMENT Teacher (Name, Email?, Affiliation*)>
```


Document Type Definition (DTD)

- The content of the Name, Code, Email, Affiliation elements are defined as Parsed Character Data

```
<!ELEMENT Name (#PCDATA)>  
<!ELEMENT Code (#PCDATA)>  
<!ELEMENT Email (#PCDATA)>  
<!ELEMENT Affiliation (#PCDATA)>
```

Document Type Definition (DTD)

- The value of the id attribute of the Teacher element must be Character Data. And the definition of this attribute must be mandatory.
- Syntax: (declaring attributes)
 - Different options for attribute-type and attribute-value
 - https://www.w3schools.com/xml/xml_dtd_attributes.asp

`<!ATTLIST element-name attribute-name attribute-type attribute-value>`

`<!ATTLIST Teacher id CDATA #REQUIRED>`

Document Type Definition (DTD)

- External Declaration
 - Located in an external file, so that can be used by several XML files
 - Reference a dtd in the header of the XML file
 - Syntax:

```
<!DOCTYPE root-element SYSTEM "file_name">
```

```
<xml version="1.0"?>  
<!DOCTYPE Teaching "teaching.dtd">
```

external

```
<xml version="1.0"?>  
<!DOCTYPE Teaching [  
  <!ELEMENT Teaching (Course+)>  
  <!ELEMENT Course (Name, Code, Teacher+)>  
  <!ELEMENT Teacher (Name, Email*, Affiliation?)>  
  <!ATTLIST Teacher id CDATA #REQUIRED>  
  <!ELEMENT Name (#PCDATA)>  
  <!ELEMENT Code (#PCDATA)>  
  <!ELEMENT Email (#PCDATA)>  
  <!ELEMENT Affiliation (#PCDATA)>  
>  
<Teaching>...</Teaching>
```

internal

XML Schema Definition (XSD)

- An XML schema also defines the structure of an XML document, like DTD
- An XML schema also written in XML, with specific markups
- Aims:
 - To define allowed elements/attributes in a XML document
 - Number/order of child elements
 - Data types, default values, fixed values for elements/attributes
- https://www.w3schools.com/xml/schema_intro.asp

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
  <xs:element name="Teaching">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Course">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Name" type="xs:string"/>
              <xs:element name="Code" type="xs:string"/>
              <xs:element name="Teacher" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Name" type="xs:string"/>
                    <xs:element name="Email" type="xs:string"/>
                    <xs:element name="Affiliation" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

XML Schema (XSD) Elements and Attributes

- An XSD element can be of type simpleType, complexType or anyType
 - simpleType contains only text (no attributes or child elements)
 - complexType can contain text, elements or attributes
 - anyType specifies that any well-formed XML is allowed in its place in XML instance
- XSD attributes are always of type simpleType

XSD Elements, Attributes Definition

Element	Syntax	Explanation
element	<xs:element>	Defines an element
attribute	<xs:attribute>	Defines an attribute
complexType	<xs:complexType>	Defines a complex type element
all	<xs:all>	Specifies that child elements can appear in any order, each child element occur 0 or 1 time
sequence	<xs:sequence>	Specifies that child elements can appear in a sequence, each child element occur from 0 to any number of times
choice	<xs:choice>	Specifies only one of the elements contained in the declaration to be present within the containing element
restriction	<xs:restriction>	Defines restrictions on a SimpleType, simpleContent or a complexContent
any	<xs:any>	Enables the author to extend the XML document with elements not specified by the schema

https://www.w3schools.com/xml/schema_elements_ref.asp

XML Schema (XSD) – Simple Element definition

- A simple element only contain text. It cannot contain any other elements or attributes.

```
<xs:element name="Name" type="xs:string"/>  
<xs:element name="Birthday" type="xs:date"/>
```

- Fixed value or default value for simple elements

```
<xs:element name="Affiliation" type="xs:string" fixed="Linköping  
University"/>  
<xs:element name="Affiliation" type="xs:string" default="Linköping  
University"/>
```

- XML schema has built-in datatypes
 - xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time

XML Schema (XSD) – Complex definition

- A complex element contains other simple elements or attributes

```
<xs:element name="Teacher">
```

```
  <xs:complexType>
```

```
    <xs:all> —————> Any order of its nexting elements
```

```
      <xs:element name="Name" type="xs:string"/>
```

```
      <xs:element name="Email" type="xs:string"/>
```

```
    </xs:all>
```

```
  </xs:complexType>
```

```
</xs:element>
```

```
<xs:element name="Teacher">
```

```
  <xs:complexType>
```

```
    <xs:sequence> —————> The order must be Name, Email
```

```
      <xs:element name="Name" type="xs:string"/>
```

```
      <xs:element name="Email" type="xs:string"/>
```

```
    </xs:sequence>
```

```
  </xs:complexType>
```

```
</xs:element>
```


XML Schema (XSD) – Complex definition

- A complex element definition can be used by multiple elements in XML data

```
<xs:element name="Teacher" type="personinfo"/>
<xs:element name="Student" type="personinfo"/>

<xs:complexType name="personinfo">
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Email" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

XML Schema (XSD) – Attribute definition

- An attribute definition is similar to a simple element definition

```
<xs:element name="Teacher">
  <xs:complexType>
    <xs:all>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Email" type="xs:string"/>
      <xs:element name="Affiliation" type="xs:string"/>
    </xs:all>
    <xs:attribute name="employeeID" type="xs:string"/>
  </xs:complexType>
</xs:element>
```

XML Schema (XSD) – Restrictions

- Define acceptable values for XML elements or attributes
 - Value range, a set of values
 - Etc.

```
<xs:element name="faculty" type="xs:string"/>
```

```
<xs:element name="faculty">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Arts and Sciences"/>
      <xs:enumeration value="Medicine and Health Sciences"/>
      <xs:enumeration value="Science and Engineering"/>
      <xs:enumeration value="Educational Sciences"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema (XSD) – Indicators

- To indicate
 - Order indicator
 - Occurrence indicator
 - Etc.

```
<xs:complexType>
  <xs:all>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Email" type="xs:string"/>
  </xs:all>
</xs:complexType>
```

```
<xs:complexType>
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Email" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

```
<xs:element name="Teacher" minOccurs="1" maxOccurs="unbounded">
```

Why use DTD or XML Schema?

- Disadvantages:
 - Rules and structures can be very complicated and cumbersome for a specific modeling scenario
 - An overhead on validations
- Advantages:
 - Facilitate information exchange and improve the interoperability
 - Guards against errors and malformed data

Outline

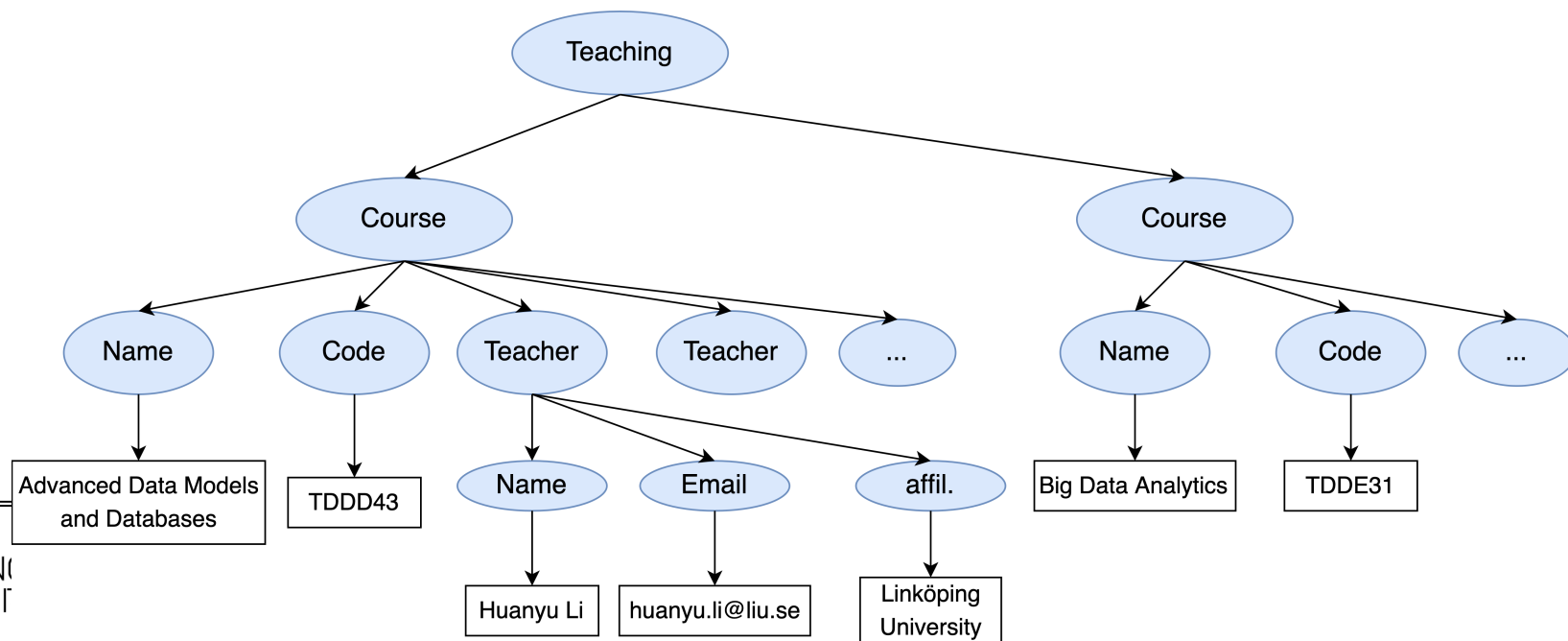
- ✓ XML
- ✓ Schema for XML
- Query languages for XML

Query Language for XML

- Option 1: XPath
 - https://www.w3schools.com/xml/xpath_intro.asp
- Option 2: XQuery
 - https://www.w3schools.com/xml/xquery_intro.asp

XPath (XML Path Language)

- XPath specifies path expressions to match XML data by navigating down (occasionally up and across) in XML documents
- W3C recommendation
- Looks very much like the path expressions in computer file systems
- Nodes: element, attribute, text, namespace, comment, root, etc.
- Relationship of Nodes
 - Parent, Children, Siblings, Ancestors, Descendants



XPath Constructors

Basic Expression and Operation – Selecting nodes

Expression	Operation
elementName	Selects all nodes with the tag named “elementName”
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes
*	Matches any element
@*	Matches any attribute
node()	Matches any node of any kind

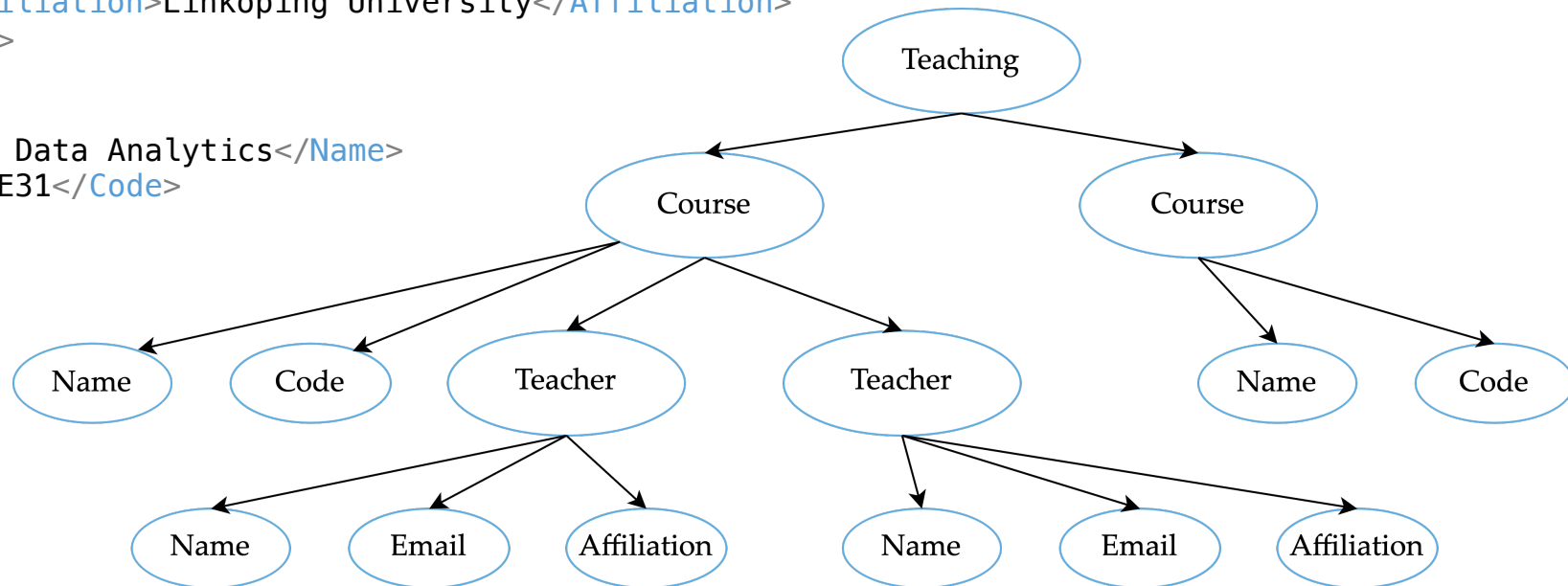
XPath - Constructors

- Predicates – finds a specific node or a node that contains a specific value, always embedded in square brackets
- [condition] matches the “current” element if condition evaluates to true on the current element

Predicate	Operation
[n]	Selects the nth child element
[last()]	Selects the last child element
[last()-1]	Selects the second-to-last child element
[position()<11]	Selects the first 10 children elements
[@attr]	Selects children elements with attribute attr
[@attr=val]	Selects children elements with value val in attribute attr
[@attr<n]	Selects children elements with a value for attribute attr less than n
[p or q]	Disjunction of predicates p or q
[p and q]	Conjunction of predicates p and q
[not (p)]	Negation of predicate p
[n][@attr]	Selects the nth child if it has attribute attr
[@attr][n]	Selects the nth child from those that have an attribute attr

XPath functions

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher id="t1">
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
    <Teacher id="t2">
      <Name>Patrick Lambrix</Name>
      <Email>patrick.lambrix@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
  </Course>
  <Course>
    <Name>Big Data Analytics</Name>
    <Code>TDDE31</Code>
  </Course>
</Teaching>
```



XPath examples

- To get all courses:

`/Teaching/Course`

- alternative:

`//Course`

- Returns a set of elements with the tag `<Course>`, node-set

XPath examples

- To access an attribute:

//Teacher/@id

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher id="t1">
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
    <Teacher id="t2">
      <Name>Patrick Lambrix</Name>
      <Email>patrick.lambrix@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
    </Teacher>
  </Course>
  <Course>
    <Name>Big Data Analytics</Name>
    <Code>TDDE31</Code>
  </Course>
</Teaching>
```

XPath examples

- To get affiliation information of a specific teacher (using predicates)

`//Teacher[@id="t1"]/Affiliation[1]`

`//Teacher[@id="t1"]/Affiliation[last()]`

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher id="t1">
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
      <Affiliation>Swedish e-Science Research Centre</Affiliation>
    </Teacher>
    <Teacher id="t2">
      <Name>Patrick Lambrix</Name>
      <Email>patrick.lambrix@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
      <Affiliation>Swedish e-Science Research Centre</Affiliation>
    </Teacher>
  </Course>
</Teaching>
```

XPath - Functions

Function	Operation
<code>contains(x, y)</code>	Checks if string x contains string y
<code>count(node-set)</code>	Returns the number of elements in the node-set
<code>name()</code>	Returns the name of an element
<code>starts-with(string,x)</code>	Checks if string starts with the character sequence x
<code>ends-with(string,x)</code>	Checks if string ends with the character sequence x
<code>substring(string,start,length)</code>	Obtains a sub-sequence of string starting at the given position, optional length can be provided
<code>normalize-space()</code>	Removes trailing whitespaces
<code>string-length(string)</code>	Returns the length of string

XPath examples

- To get courses with more than 1 teacher

`//Course[count(Teacher)>1]`

- To get courses whose names contain “Data”

`//Course[contains(Name, “Data”)]`

```
<Teaching>
  <Course>
    <Name>Advanced Data Models and Databases</Name>
    <Code>TDDD43</Code>
    <Teacher id="t1">
      <Name>Huanyu Li</Name>
      <Email>huanyu.li@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
      <Affiliation>Swedish e-Science Research Centre</Affiliation>
    </Teacher>
    <Teacher id="t2">
      <Name>Patrick Lambrix</Name>
      <Email>patrick.lambrix@liu.se</Email>
      <Affiliation>Linköping University</Affiliation>
      <Affiliation>Swedish e-Science Research Centre</Affiliation>
    </Teacher>
  </Course>
</Teaching>
```


XPath - Axes

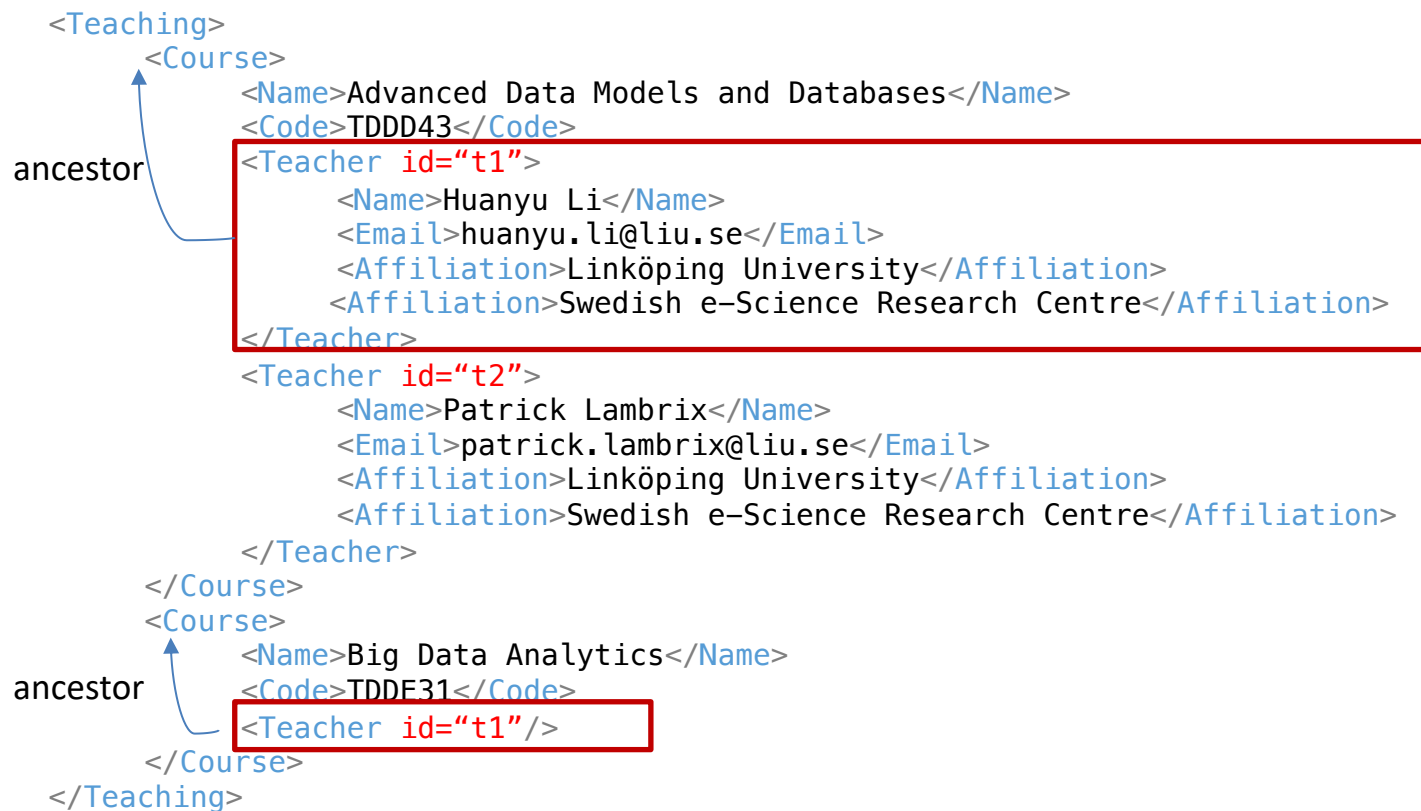
- An axis represents a relationship to the current node
- Used to locate nodes relative to that node on the tree
- Syntax: **axis-name::element-name or attribute-name**

Axis	Operation
child::	Selects all children of the current node
attribute::	Selects all attributes of the current node
descendant::	Selects all descendants of the current node
descendant-or-self::	Selects all descendants of the current node and the current node itself
parent::	Selects the parent of the current node
ancestor::	Selects all ancestors of the current node
ancestor-or-self::	Selects all ancestors of the current node and the current node itself
following-sibling::	Selects all siblings after the current node
preceding-sibling::	Selects all nodes appearing before the current node
self::	Selects the current node

XPath - Axes

- Examples: (select all course ancestors of the Teacher node whose id is "t1")

`//Teacher[@id="t1"]/ancestor::Course`



Query Language for XML

- Option 1: XPath
 - https://www.w3schools.com/xml/xpath_intro.asp
- Option 2: XQuery
 - https://www.w3schools.com/xml/xquery_intro.asp

XQuery

- XQuery is built on XPath expressions
 - Path matching
 - FLWOR expressions
 - FOR-LET-WHERE-ORDER BY-RETURN
 - Aggregation, functions
- W3C recommendation

XQuery – FLWOR Expressions

- For: selects a sequence of nodes/elements
- Let: binds a sequence to a variable
- Where: filters the nodes
- Order by: sorts the nodes
- Return: what to return

FLWOR Expressions

```
<result>{  
  for $c in doc("teaching_data_1.xml")//Course  
  let $t := $c/Teacher  
  where $t/Affiliation = "Linköping University"  
  return  
    <selected_course>  
      {$c/Name}  
      {$c/Code}  
      {$t}  
    </selected_course>  
}  
</result>
```

for/in – iterate over elements

let – value assignment

where – filter condition

return – construct a result

XQuery - Joins

```
<result>{  
  for $c1 in doc("teaching_data_1.xml")//Course  
  for $c2 in doc("teaching_data_2.xml")//Course  
  let $t1 := $c1/Teacher  
  let $t2 := $c2/Teacher  
  where $c1/Teacher/Name = $c2/Teacher/Name  
  return  
    <selected_course>  
      {$c1}  
      {$c2}  
    </selected_course>  
}  
</result>
```

XQuery - Joins

```
<result>{  
  for $c1 in doc("teaching_data_1.xml")//Course  
  for $c2 in doc("teaching_data_2.xml")//Course[Teacher/Name =  
    $c1/Teacher/Name]  
  return  
    <selected_course>  
      {$c1}  
      {$c2}  
    </selected_course>  
}  
</result>
```


XQuery – User-Defined functions

```
declare function prefix:function_name($parameter as datatype) as  
returnDatatype  
{  
    ...function code ...  
}
```

JSON – Javascript Object Notation

- Also a standardized format for exchanging data, based on key/value pairs
 - Keys must be strings, written with double quotes
 - Values can be in a string, a number, an object, an array, a Boolean, null

```
{  
  "Teachers":  
  [  
    {  
      "Name": "Huanyu Li",  
      "Email": "huanyu.li@liu.se",  
      "Affiliation": "Linköping University"  
    },  
    {  
      "Name": "Patrick Lambrix",  
      "Email": "patrick.lambrix@liu.se",  
      "Affiliation": "Linköping University"  
    }  
  ]  
}
```

GraphQL Schema

- GraphQL is a conceptual framework for building web APIs
- The framework introduces GraphQL schema, resolver function and a query language
- A GraphQL schema tells users how underlying data can be accessed

```
type Course {  
  Name: String  
  Code: String  
  hasTeachers: [Teacher]  
}  
type Teacher {  
  ID: String  
  Name: String  
  Email: String  
  Affiliation: String  
}  
type Query {  
  ListCourses: [Course]  
  ListTeachers: [Teacher]  
}
```

GraphQL schema

```
{  
  ListTeachers  
  {  
    Name  
    Email  
    Affiliation  
  }  
}
```

GraphQL query

```
{  
  "data": {  
    "ListTeachers":  
    [{  
      "Name": "Huanyu Li",  
      "Email": "huanyu.li@liu.se",  
      "Affiliation": "Linköping University"  
    },  
    {  
      "Name": "Patrick Lambrix",  
      "Email": "patrick.lambrix@liu.se",  
      "Affiliation": "Linköping University"  
    }  
  ]  
}
```

GraphQL query response

www.liu.se