UNIVERSITÉ LUMIÈRE LYON 2

Département d'Informatique et de Statistique
DIS
Institut de la Communication

Semi-structured data & XML

M1 Informatique
Year 2015-2016
Jérôme Darmont

http://eric.univ-lyon2.fr/~jdarmont/

SSD & XML news

http://eric.univ-lyon2.fr/~jdarmont/?page_id=455

http://eric.univ-lyon2.fr/~jdarmont/?feed=rss2

https://twitter.com/darmont_lyon2 *hashtag* #ssdxml

Outline

- Semi-structured data
- XML language
- XQuery language

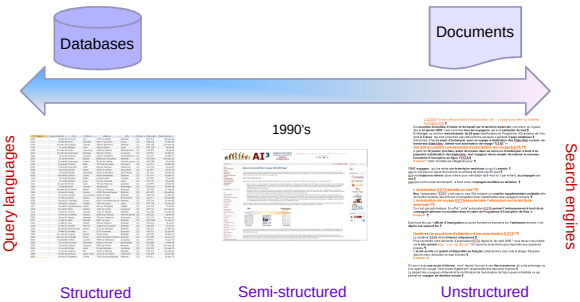
Structured data

- Data are organized in entities
- Similar entities form group (classes)
- Entities in same group have same descriptions (attributes)
- For all entities in a group
  - Same attribute format
  - Same predefined length
  - All attributes are present
  - Attributes follow a predefined order
- Structured data are described by a schema
  - They are usually stored in databases

Unstructured data

- Data can be of any type
- Data follow no predefined format nor sequence
- Data follow no rule
- Data are not predictable
- Unstructured data are freeform
  - Texts
  - Images
  - Videos
  - Sounds

Semi-structured data



Semi-structured data

- Data are organized in semantic entities
- Similar entities form groups

Structured

- Entities in same group may not have same attributes
- For all entities in a group
  - Types may differ
  - Sizes may differ
  - Attributes may be missing or duplicated
  - Attribute order is not necessarily important

Unstructured

- Semi-structured data are self-describing
  - Web pages, XML documents, emails…

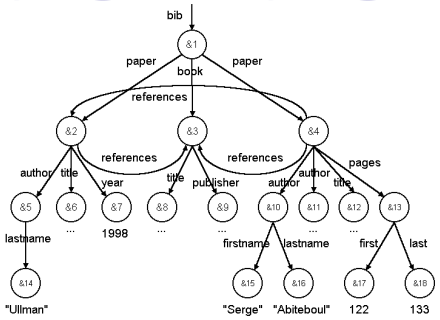
Example of semi-structured data

- Name Jérôme Darmont
- Email jerome.darmont@univ-lyon2.fr
  jdarmont@eric.univ-lyon2.fr
- Name
  - First name Stéphane
  - Last name Lallich
- Email stephane.lallich@univ-lyon2.fr
- Name Julien Velcin
- Affiliation Université Lumière Lyon 2

Semi-structured data model

- Pros
  - Can represent information from data sources that cannot be constrained by schema
  - Flexible format for data interoperability
  - Help view structured data as semi-structured (Web browsing)
  - Schema can evolve easily
- Cons
  - Query performance of wide-range data scans
- Standard representations
  - Electronic Data Interchange (EDI) – Financial domain
  - Object Exchange Model (OEM): graph-based model for SSD
  - SGML, HTML and XML

Example of OEM graph



Semi-structured data management

Challenges… and course objectives!

- How to model SSD?
  - Graphs (OEM) – Logical model
  - XML – Physical model
- How to query SSD?
  - XPath
  - XQuery
- How to store SSD?
  - Flat files
  - Relational, object-oriented or native database
- How to process SSD efficiently?
  - Many research issues!

References

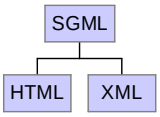
- Peter Wood, Birkbeck University of London
  Semi-Structured Data
  http://www.dcs.bbk.ac.uk/~ptw/teaching/ssd/toc.html
- Mike Bergman, Structured Dynamics LLC
  Semi-structured Data: Happy 10th Birthday!
  http://www.mkbergman.com/153/semi-structured-data-happy-10th-birthday/

Outline

- ✓ Semi-structured data
- XML language
- XQuery language

Generalities

- XML : *Extensible Markup Language*
  - Internet document and data structuring format issued from SGML
  - Document definition, management, creation, transmission and sharing
- XML is a W3C-promoted standard
  - *Draft* in 1996
  - XML 1.0 in 1997
  - XML 1.1 in 2004



XML base document

- Example: profs.xml

```
<?xml version="1.0"  encoding="utf-8" ?> <!-- Prolog (mandatory) -->
<professor_directory> <!-- Root element -->
   <!-- Embedded elements -->
   <professor>
      <name>Jérôme Darmont</name>
      <email>jerome.darmont@univ-lyon2.fr</email>
      <course>Database programming</course>
      <course>Semi-structured data, XML</course>
   </professor>
   <professor>
      <name>Julien Velcin</name>
      <email>julien.velcin@univ-lyon2.fr</email>
      <course>Artificial intelligence</course>
   </professor>
   <!-- Etc. -->
</professor_directory> <!-- Closing tag -->
```


Writing rules

- An XML document has only one root element.
- Elements must be properly embedded (tags must not overlap).
- Every element must have an opening and a closing tag.
- An element's name must be strictly identical in its opening and closing tags.
- Element names are case-sensitive.
- Element names must start by a letter or an underscore (_), and may be followed by letters, numbers, dots (.), minuses (-) and/or underscores (_).

Writing rules continued

- Element names starting by XML (any combination of small and/or large caps) are reserved for standardization purposes.
- An XML document respecting theses rules is said well-formed.
- An XML document must be well-formed!

Element contents

- Unauthorized characters: < &]]>
- Embedding elements: unlimited depth
  - Ex.

```
<professor_directory>
    <professor>
        <name>
            <last_name>Zighed</last_name>
            <first_name>Djamel</first_name>
            <first_name>Abdelkader</first_name>
        </name>
    </professor>
</professor_directory>
```

Element contents

- CDATA section: Bloc of free text where only the string]]> is forbidden
  - Ex.
```
<name>
    <![CDATA[<Darmont> & <Lallich>]]>
<name>
```
- Empty element: no content
  - Ex. <email> </email>
  - Equivalent formulation <email />

Element attributes

- Attributes : data associated with a given element
- Definition: name-value couple in element's opening tag
  - Ex. <office campus="Bron" building="K">061</office>

attribute value attribute value

- Attributes may be present in empty elements.
  - Ex. <picture source="my_face.png" />

Element contents vs. attribute values

- What to choose?
  -
```
<professor>
    <name>Darmont</name>
</professor>
```
  - <professor name="Darmont" />
- 4 principles to decide
  - Uche Ogbuji, Fourthought, Inc.
  - Source: http://www.ibm.com/developerworks/xml/library/x-eleatt/

Element contents vs. attribute values

Principle	Element	Attribute
Core content	Essential information	Peripheral information
Structured information	Structured information	Atomic information
Readability	Information for humans	Information for machines
Element/attribute binding	Information modified by another attribute	

Example of element/attribute binding

```
<supply-stock>
    <product quantity="1500">
        <name>Computer</name>
    </product>
    <product quantity="500">
        <name>Printer</name>
    </product>
</supply-stock>
```


Valid XML documents

- Valid XML document:
  - Well-formed document
  - + *Document Type Definition* (DTD) that defines its structure
  - + The root element conforms to the structure defined in the DTD.
- DTD: standard document prototype
  - Helps a validator check whether an XML document is valid
  - Enforces the uniformity of a group of similar XML documents

DTD specification

- A DTD may be directly included into an XML document (internal DTD).
- It is preferable to place it in a separate file (external DTD) ⇨ the same DTD can be reused for several XML documents.
- Inclusion of an external DTD (document prolog)

  <!DOCTYPE *name_of_root_element* SYSTEM "*URI*">
  - Ex. <!DOCTYPE professor_directory SYSTEM "profs.dtd">

DTD specification

- Element type declaration

  <!ELEMENT *name contents*>

  - Ex. <!ELEMENT name (#PCDATA)>

- Possible content types
  - Text data: (#PCDATA)
  - Empty element: EMPTY
  - Any (legal) content: ANY
  - Ordered sequence: (elt_1, elt_2, …, elt_n)

  Ex. <!ELEMENT professor (name, email, course)>

  Note: Subelements must be defined in turn.

DTD specification

- Choice: (elt_1 | elt_2 | … | elt_n)

  Ex. <!ELEMENT uri (http | ftp | mailto | telnet)>

  Note: Subelements must be defined in turn.

- Element multiplicity
  - 0 or 1 occurrence: ?
  - 1 to N occurrences: +
  - 0 to N occurrences: *
  - Ex. 1: Element *mountain* with one or several *names*, optional *height* and mandatory *country*.

    <!ELEMENT mountain (name+, height?, country)>

DTD specification

- Ex. 2: Element *mountain* with multiple-occurrence subelements

  <!ELEMENT mountain (name, height, country)*>

- Subelement embedding
  - Ex. <!ELEMENT mountain (name, height, (state | country | region))>

- Full, simple example

```
<!ELEMENT professor_directory (professor)*>
<!ELEMENT professor (name, email, course+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT course (#PCDATA)>
```


DTD specification

- Attribute list declaration: <!ATTLIST *elt_name att_definitions*>
  - Ex. <!ATTLIST office campus CDATA "Porte des Alpes"
    building CDATA #REQUIRED>

- Attribute definition
  - Attribute name
  - Type
  - Default value (#REQUIRED = none)

DTD specification

- Attribute types
  - String: CDATA
  - Tokenized:
    - ID: The attribute must have a unique value in each element.
    - IDREF/IDREFS: The attribute value must correspond to the value(s) of one (several) other attribute(s) of type ID. Multiple values are separated by spaces.
  - Enumerated type: list of possible values

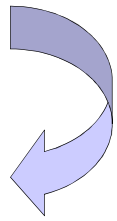
  Ex. <!ATTLIST office campus (Quai | Bron) "Bron">

Entities

- Entity: constant defined in a DTD and that can be referenced anywhere in an XML document conforming to this DTD
- Definition (DTD): <!ENTITY *name value*>
  - Ex. <!ENTITY lyon2-domain "univ-lyon2.fr">
- Reference (XML document): &*name*;
  - Ex. <email>jerome.darmont@&lyon2-domain;</email>
    <server>dis.&lyon2-domain;</server>

XML Schema

- Limits of DTDs
  - Specific syntax (not XML)
  - Weak typing
  - No partial modeling
  - But still readable by people
- Characteristics of XML Schema
  - Expressed as an XML document
  - Strong typing
  - Partial modeling is possible
  - Mostly for machine processing



XML Schemas should replace DTDs…

XML Schema document

- Document structure

```
<?xml version="1.0" encoding="utf-8" ?>
 <!-- my_schema.xsd -->
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <!-- Element definitions -->
</xsd:schema>
```

- Reference to schema in XML document

```
<root-elt xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="my_schema.xsd">
```


Element specifications

- Simple type

```
<xsd:element name="name" type="xsd:string" />
```

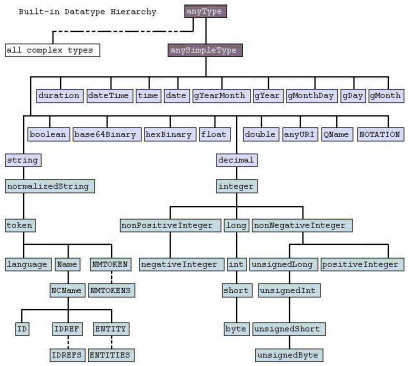
- Number of occurrences

```
<xsd:element name="course" type="xsd:string"
             minOccurs="1" maxOccurs="unbounded" />
<!-- minOccurs and maxOccurs can be any integers
such that maxOccurs ≥ minOccurs -->
```

- Complex type

```
<xsd:element name="professor">
    <xsd:complexType>
        <!-- Complex type specification -->
    </xsd:complexType>
</xsd:element>
```


XML Schema simple data types



Source: http://www.w3.org/TR/xmlschema-2/#built-in-datatypes.jpg

XML Schema complex types

- Sequence

```
<xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="course" type="xsd:string" />
    <xsd:element name="email" type="xsd:string" />
</xsd:sequence>
```

- Choice

```
<xsd:choice>
    <xsd:element name="http" type="xsd:string" />
    <xsd:element name="ftp" type="xsd:string" />
    <xsd:element name="mailto" type="xsd:string" />
    <xsd:element name="telnet" type="xsd:string" />
</xsd:choice>
```


XML Schema complex types

- All

```
<-- each element appears at most once, but in any order -->
<xsd:all>
    <xsd:element name="address" type="xsd:string" />
    <xsd:element name="email" type="xsd:string" />
    <xsd:element name="phone-number" type="xsd:string" />
</xsd:all>
```

- Reference to complex element

```
<xsd:element ref="professor" />
 <-- … -->
<xsd:element name="professor">
    <-- Complex type definition -->
</xsd:element>
```

Attribute specifications

- Definition

```
<xsd:attribute name= "update" type="xsd:date" default="2003-10-11" />
<xsd:attribute name= "amount" type="xsd:integer" use= "required" />
```

- Type restriction

```
<xsd:attribute name="office" use="required">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="K061"/>
            <xsd:enumeration value="K062"/>
            <xsd:enumeration value="K063"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
```


Attribute declaration

- In simple-typed elements

```
<xsd:element name="blog-post">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="post-date" type="xsd:date" />
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```


Attribute declaration

- In complex-typed elements

```
<xsd:element name="professor">
  <xsd:complexType>
    <xsd:sequence>
      <!-- Some subelement definitions -->
    </xsd:sequence>
    <xsd:attribute name="office" type="xsd:string" />
  </xsd:complexType>
</xsd:element>
```

- By reference

```
<!-- As above, but… -->
<xsd:attribute ref="office" use="required" />
```


Sample XML Schema

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="professor_directory">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="professor" minOccurs="0"
                                     maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```


Sample XML Schema

```
  <xsd:element name="professor">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" />
        <xsd:element name="course" type="xsd:string"
                     minOccurs="1" maxOccurs="unbounded" />
        <xsd:element name="email" type="xsd:string" />
      </xsd:sequence>
      <xsd:attribute name="office" type="xsd:string" />
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```


Outline

- ✓ Semi-structured data
- ✓ XML language
- XQuery language

Generalities

- Query language defined over XML data
- Similarities to SQL
- Designed by the W3C
- Built upon XPath expressions (same data model, functions, operators)
- Version history
  - XQuery 1.0 (2007) ⊇ XPath 2.0
  - XQuery 3.0 (2014) ⊇ XPath 3.0
- Standardization in process
- Supported by DBMS publishers (Oracle, Microsoft, IBM…)

Sample XML document

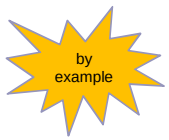
```
<?xml version="1.0" encoding="utf-8" ?>
<catalog>
        <dvd zone="1">
                <title>Blade runner</title>
                <director>Ridley Scott</director>
                <year>1982</year>
                <duration>117</duration>
                <language>English</language>
                <price>14.79</price>
        </dvd>
        <dvd zone="2">
                <title>La grande vadrouille</title>
                <director>Gérard Oury</director>
                <year>1966</year>
                <duration>122</duration>
                <language>French</language>
                <price>19.82</price>
        </dvd>                                  <!-- (…) -->
```


Sample XML document continued

```
        <dvd zone="2">
                <title>Le fabuleux destin d'Amélie Poulain</title>
                <director>Jean-Pierre Jeunet</director>
                <year>2001</year>
                <duration>120</duration>
                <language>French</language>
                <price>14.99</price>
        </dvd>
        <dvd zone="2">
                <title>The big Lebowski</title>
                <director>Ethan Coen</director>
                <director>Joel Coen</director>
                <year>1997</year>
                <duration>112</duration>
                <language>French</language>
                <language>English</language>
                <price>19.82</price>
        </dvd>
</catalog>
```


Simple path expressions

- Whole XML document
  doc("dvd.xml")/catalog
  Result
  The whole document



- A given element
  doc("dvd.xml")/catalog/dvd
  doc("dvd.xml")/catalog/dvd/title
  Result
  <title>Blade runner</title>
  <title>La grande vadrouille</title>
  <title>Le fabuleux destin d'Amélie Poulain</title>
  <title>The big Lebowski</title>

Simple path expressions continued

- A given attribute
  doc("dvd.xml")/catalog/dvd/data(@zone)
  Result
  1 2 2 2
- A given element, whatever its hierarchy level
  doc("dvd.xml")/catalog//title
  //title
  Result
  <title>Blade runner</title>
  <title>La grande vadrouille</title>
  <title>Le fabuleux destin d'Amélie Poulain</title>
  <title>The big Lebowski</title>

Simple path expressions continued

- All subelements of an element
  doc("dvd.xml")/catalog/dvd/*
  Result
  <title>Blade runner</title>
  <director>Ridley Scott</director>
  <year>1982</year>
  <duration>117</duration>
  <language>English</language>
  <price>14.79</price>
  <title>La grande vadrouille</title>
  <director>Gérard Oury</director>
  <year>1966</year>
  <duration>122</duration>
  <language>French</language>
  <price>19.82</price>
  Etc.

Path predicates

- i^{th}, last, i first/last elements
  doc("dvd.xml")/catalog/dvd[1]
  doc("dvd.xml")/catalog/dvd[last()]
  doc("dvd.xml")/catalog/dvd[position() < 3]/title
  Result
  <title>Blade runner</title>
  <title>La grande vadrouille</title>

- Elements that possess a given element/attribute
  doc("dvd.xml")/catalog/dvd[duration]/title
  Result
  <title>La grande vadrouille</title>
  <title>Le fabuleux destin d'Amélie Poulain</title>
  <title>The big Lebowski</title>

  doc("dvd.xml")/catalog/dvd[@zone]

Path predicates continued

- Conditions over an element/attribute
  doc("dvd.xml")/catalog/dvd[price < 15]
  doc("dvd.xml")/catalog/dvd[@zone = "2" and price < 15]/title
  Result
  <title>Le fabuleux destin d'Amélie Poulain</title>

- Path combination
  doc("dvd.xml")//title | doc("dvd.xml")//price
  Result
  <title>Blade runner</title><price>14.79</price>
  <title>La grande vadrouille</title><price>19.82</price>
  <title>Le fabuleux destin d'Amélie Poulain</title><price>14.99</price>
  <title>The big Lebowski</title> <price>19.82</price>

FLWOR queries

- FLWOR ("flower"): For, Let, Where, Order by, Return

- For clause: binds a variable to each element return by an expression (iteration)

Example:

```
for $x in (1 to 3) <!-- By the way, this is a comment -->
return <res>{$x}</res>
```

Result:

```
<res>1</res>
<res>2</res>
<res>3</res>
```


For clause

Example:

```
for     $x in (1, 2),
        $y in (10, 20) (: This is also a comment :)
return <res>x = {$x} and y = {$y}</res>
```

Result:

```
<res>x = 1 and y = 10</res>
<res>x = 1 and y = 20</res>
<res>x = 2 and y = 10</res>
<res>x = 2 and y = 20</res>
```


For clause continued

Example:

```
for $x in doc("dvd.xml")/catalog/dvd
return $x/title
```

Result:

```
<title>Blade runner</title>
<title>La grande vadrouille</title>
<title>Le fabuleux destin d'Amélie Poulain</title>
<title>The big Lebowski</title>
```


For clause continued

Example:

```
for $x at $i in doc("dvd.xml")/catalog/dvd/title
return <dvd id="{$i}">{data($x)}</dvd>
```

Result:

```
<dvd id="1">Blade runner</dvd>
<dvd id="2">La grande vadrouille</dvd>
<dvd id="3">Le fabuleux destin d'Amélie Poulain</dvd>
<dvd id="4">The big Lebowski</dvd>
```

Let clause

- Let clause: Assigns value(s) to a variable (no iteration)

Example:

```
let $x := (1 to 5)
return <res>{$x}</res>
```

Result:

```
<res>1 2 3 4 5</res>
```


Where clause

- Where clause: Specifies condition(s) on the result

Example:

```
for $x in doc("dvd.xml")/catalog/dvd
where $x/price > 15
return $x/title
```

Example:

```
for $x in doc("dvd.xml")/catalog/dvd
where $x/@zone = "2" and $x/price < 10
return $x/title
```


Order by and Return clauses

- Order by clause: Sorts the result

Example:

```
for $x in doc("dvd.xml")/catalog/dvd
order by $x/title
return $x/title
```

Example :

```
for $x in doc("dvd.xml")/catalog/dvd
order by $x/@zone, $x/title descending
return $x/title
```

- Return clause: Specifies the result

Return clause

- Conditional expressions

Example:

```
for $x in doc("dvd.xml")/catalog/dvd
return if ($x/@zone="1")
        then <zoneUS>{data($x/title)}</zoneUS>
        else <zoneEU>{data($x/title)}</zoneEU>
```

Result:

```
<zoneUS>Blade runner</zoneUS>
<zoneEU>La grande vadrouille</zoneEU>
<zoneEU>Le fabuleux destin d'Amélie Poulain</zoneEU>
<zoneEU>The big Lebowski</zoneEU>
```


XQuery functions

- Access functions: data()…
- Numerical functions: abs(), floor(), ceiling(), round()…
- String functions: string-length(), upper-case(), lower-case(), normalize-space(), substring(), substring-after(), replace(), contains()…
- Temporal functions: day-from-date(), year-from-date()…
- Sequence functions: exists(), distinct-values(), reverse()…
- Aggregation functions: count(), avg(), max(), min(), sum()
- Context functions: last(), position()…
- Boolean functions: not()…

XQuery functions continued

Sample call:

```
for $x in doc("dvd.xml")/catalog/dvd/title
let $uppertitle := upper-case($x)
return <film>{$uppertitle}</film>
```

Result:

```
<film>BLADE RUNNER</film>
<film>LA GRANDE VADROUILLE</film>
<film>LE FABULEUX DESTIN D'AMÉLIE POULAIN</film>
<film>THE BIG LEBOWSKI</film>
```

Grouping XQueries

- No grouping construct in XQuery 1.0
  - Poor result readability (empty elements)
  - Poor efficiency (multiple passes over the document)
- New grouping clause in XQuery 3.0
  - Implementation in good progress in 2015

Grouping in XQuery 1

Example:

```
for $z in distinct-values(/catalog/dvd/@zone)
let $p := avg(/catalog/dvd[@zone = $z]/price)
order by $z
return      <zone value="{$z}">
                  <avgprice>{$p}</avgprice>
            </zone>
```

Result:

```
<zone value="1">
      <avgprice>14.79</avgprice>
</zone>
<zone value="2">
      <avgprice>18.21</avgprice>
</zone>
```


Multiple grouping in XQuery 1

```
for   $z in distinct-values(/catalog/dvd/@zone),
      $y in distinct-values(/catalog/dvd/year)

let $p := /catalogue/dvd[@zone = $z and year = $y]/price

order by $z, $y

return      <group zone="{$z}" year="{$y}">
                  <avgprice>{avg($p)}</avgprice>
            </group>
```


Grouping in XQuery 3

Single grouping:

```
for $d in /catalog/dvd
group by $z := $d/@zone
return      <zone value="{$z}">
                  <avgprice>{avg($d/price)}</avgprice>
            </zone>
```

Multiple grouping:

```
for $d in /catalog/dvd
group by $z := $d/@zone, $y := $d/year
return      <group zone="{$z}" year="{$y}">
                  <avgprice>{avg($d/price)}</avgprice>
            </group>
```


Join XQueries over multiple documents

```
<?xml version="1.0" encoding="utf-8" ?><!-- doc #1: customers.xml -->
<customers>
      <customer id="1">
            <last-name>Lallich</last-name>
            <first-name>Stéphane</first-name>
            <address>Bureau 04</address>
      </customer>
      <customer id="2">
            <last-name>Bentayeb</last-name>
            <first-name>Fadila</first-name>
            <address>Bureau 09B</address>
      </customer>
      <customer id="3">
            <last-name>Darmont</last-name>
            <first-name>Jérôme</first-name>
            <address>Bureau 09B</address>
      </customer>
</customers>
```


Join XQueries

```
<?xml version="1.0" encoding="utf-8" ?> <!-- doc #2: products.xml -->
<products>
      <product id="10">
            <name>Computer</name>
      </product>
      <product id="20">
            <name>21" monitor</name>
      </product>
      <product id="30">
            <name>Printer</name>
      </product>
</products>
```

Join XQueries continued

```
<?xml version="1.0" encoding="utf-8" ?> <!-- doc #3: orders.xml -->
<orders>
        <order cust-id="1" prod-id="10">
                <quantity>3</quantity>
        </order>
        <order cust-id="1" prod-id="20">
                <quantity>15</quantity>
        </order>
        <order cust-id="2" prod-id="10">
                <quantity>7</quantity>
        </order>
        <order cust-id="2" prod-id="30">
                <quantity>10</quantity>
        </order>
        <order cust-id="3" prod-id="30">
                <quantity>5</quantity>
        </order>
</orders>
```


Join XQueries continued

Example:

```
for   $c in doc("customers.xml")//customer,
      $o in doc("orders.xml")//order
where $c/@id = $o/@cust-id
return <res>{data($c/last-name)}, {data($c/first-name)}:
                                {data($o/quantity)}</res>
```

Result:

```
<res>Lallich, Stéphane: 3</res>
<res>Lallich, Stéphane: 15</res>
<res>Bentayeb, Fadila: 7</res>
<res>Bentayeb, Fadila: 10</res>
<res>Darmont, Jérôme: 5</res>
```


Join XQueries continued

Example:

```
for   $c in doc("customers.xml")//customer,
      $o in doc("orders.xml")//order,
      $p in doc("products.xml")//product
where $c/@id = $o/@cust-id
and $o/@prod-id = $p/@id
return <res>{data($c/last-name)}, {data($c/first-name)}:
                        {data($o/quantity)} x {data($p/name)}</res>
```

Result:

```
<res>Lallich, Stéphane: 3 x Computer</res>
<res>Lallich, Stéphane: 15 x 21" monitor</res>
<res>Bentayeb, Fadila: 7 x Computer</res>
<res>Bentayeb, Fadila: 10 x Printer</res>
<res>Darmont, Jérôme: 5 x Printer</res>
```


Join XQueries completed!

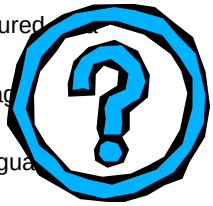
Variants with path predicates:

```
for   $c in doc("customers.xml")//customer,
      $o in doc("orders.xml")//order[@cust-id=$c/@id]
return <res>{data($c/last-name)}, {data($c/first-name)}:
                                {data($o/quantity)}</res>
```

```
for   $c in //customer,
      $p in //product,
      $o in //order[@cust-id=$c/@id and @prod-id=$p/@id]
return <res>{data($c/last-name)}, {data($c/first-name)} :
                        {data($o/quantity)} x {data($p/name)}</res>
```


Outline

- ✓ Semi-structured
- ✓ XML languag
- ✓ XQuery langua



Example #1: XML schemas

- Provide:
  - a DTD
  - an XML Schema
- so that the following trains.xml document is valid.

Example #1: Document

```
<!-- trains.xml -->
<?xml version="1.1" encoding="utf-8" ?>
<trains>
        <train id="T1" departure="C1" arrival="C3" otherstations="C2">
                <wagon id="W11">
                        <container vol="5">Beef meat</container>
                        <container vol="5">Horse meat</container>
                        <container vol="10">Pasta</container>
                </wagon>
                <wagon id="W12">
                        <container vol="40">Cars</container>
                </wagon>
                <wagon id="W13" /> <!-- Empty wagon -->
                <wagon id="W14" /> <!-- Empty wagon -->
        </train>
```


Example #1: Document

```
<!-- trains.xml continued -->
        <train id="T2" departure="C6" arrival="C4" otherstations="C5 C1">
                <wagon id="W21">
                        <container vol="20">Cassoulet</container>
                </wagon>
        </train>
        <train id="T3" departure="C1" arrival="C2" /> <!-- Just the loco -->
</trains>

        <!-- cities.xml (for reference only) -->
        <?xml version="1.1" encoding="utf-8" ?>
        <cities>
                <city id="C1">Paris</city>
                <city id="C2">Lyon</city>
                <city id="C3">Marseille</city>
                <city id="C4">Lille</city>
                <city id="C5">Bordeaux</city>
                <city id="C6">Toulouse</city>
        </cities>
```


Example #1: DTD

```
<!-- trains.dtd -->

<!ELEMENT trains (train)+>

<!ELEMENT train (wagon)*>
        <!ATTLIST train id ID #REQUIRED departure IDREF #REQUIRED
                arrival IDREF #REQUIRED otherstations IDREFS "">

<!ELEMENT wagon (container)*>
        <!ATTLIST wagon id ID #REQUIRED>

<!ELEMENT container (#PCDATA)>
        <!ATTLIST container vol CDATA #REQUIRED>
```


Example #1: XML Schema

```
<!-- trains.xsd -->
<?xml version="1.1" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="trains">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element ref="train" minOccurs="1"
                                        maxOccurs="unbounded" />
                        </xsd:sequence>
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="train">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element ref="wagon" minOccurs="0"
                                        maxOccurs="unbounded" />
                        </xsd:sequence>
```


Example #1: XML Schema

```
<!-- trains.xsd continued -->
                        <xsd:attribute name="id" type="xsd:ID" use= "required" />
                        <xsd:attribute name="departure" type="xsd:IDREF"
                                use="required" />
                        <xsd:attribute name="arrival" type="xsd:IDREF"
                                use="required" />
                        <xsd:attribute name="otherstations" type="xsd:IDREFS"
                                use="required" /> <!-- default="" is not allowed -->
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="wagon">
                <xsd:complexType>
                        <xsd:sequence>
                                <xsd:element ref="container" minOccurs="0"
                                        maxOccurs="unbounded" />
                        </xsd:sequence>
```


Example #1: XML Schema

```
<!-- trains.xsd continued -->

                        <xsd:attribute name="id" type="xsd:ID" use= "required" />
                </xsd:complexType>
        </xsd:element>
        <xsd:element name="container">
                <xsd:complexType>
                        <xsd:simpleContent>
                                <xsd:extension base="xsd:string">
                                        <xsd:attribute name="vol"
                                                type="xsd:float" />
                                </xsd:extension>
                        </xsd:simpleContent>
                </xsd:complexType>
        </xsd:element>
</xsd:schema>
```

Example #2: Complex XQueries

- Given the XML documents:
  - products.xml, stores.xml, cities.xml, times.xml, sales.xml
- Formulate the FLWOR queries (in XQuery 3):
  1. Number of sales per product ID
  2. Total sale amount and quantity per store
  3. Total sale amount and quantity per store and per product
  4. Total sale amount and quantity per city and per product
  5. Total sale amount and quantity per store, per product and per month

Example #2: Documents

```
<!-- products.xml -->
<products>
  <product id="p1" name="Computer" />
  <product id="p2" name="Mouse" />
  <product id="p3" name="Screen" />
  <product id="p4" name="Snack" />
</products>
```

```
<!-- stores.xml -->
<stores>
  <store id="s1" name="Store 1" city-id="c1" />
  <store id="s2" name="Store 2" city-id="c1" />
  <store id="s3" name="Store 3" city-id="c2" />
</stores>
```

```
<!-- times.xml -->
<times>
  <time date-time="2012-02-08 15:55:00" month-year="2012-02" />
  <time date-time="2012-02-01 14:00:00" month-year="2012-02" />
  <time date-time="2012-01-15 12:00:00" month-year="2012-01" />
</times>
```


Example #2: Documents

```
<!-- sales.xml -->
<sales>
  <sale prod-id="p1" store-id="s1" date-time="2012-02-08 15:55:00">
    <amount>100</amount>
    <qty>5</qty>
  </sale>
  <sale prod-id="p1" store-id="s1" date-time="2012-02-01 14:00:00">
    <amount>200</amount>
    <qty>1</qty>
  </sale>
  <sale prod-id="p1" store-id="s2" date-time="2012-02-08 15:55:00">
    <amount>10</amount>
    <qty>10</qty>
  </sale>
  <sale prod-id="p2" store-id="s2" date-time="2012-02-01 14:00:00">
    <amount>1000</amount>
    <qty>1</qty>
  </sale>
  <sale prod-id="p3" store-id="s2" date-time="2012-01-15 12:00:00">
    <amount>500</amount>
    <qty>2</qty>
  </sale>
  <sale prod-id="p4" store-id="s1" date-time="2012-02-01 14:00:00">
    <amount>150</amount>
    <qty>3</qty>
  </sale>
  <sale prod-id="p3" store-id="s2" date-time="2012-01-01 12:00:00">
    <amount>750</amount>
    <qty>2</qty>
  </sale>
</sales>
```


Example #2: Query #1

```
(: Number of sales per product ID :)

for $f in doc("sales.xml")//sale

group by $pid := $f/@prod-id

return <group prod-id="{$pid}">
          <num_sales>{count($f)}</num_sales>
       </group>
```


Example #2: Query #2

```
(: Total sale amount and quantity per store :)

for     $s in doc("stores.xml")//store,
        $f in doc("sales.xml")//sale[@store-id = $s/@id]

group by $st := $s/@name

return <group store="{$st}">
          <tot_amount>{sum($f/amount)}</tot_amount>
          <tot_qty>{sum($f/qty)}</tot_qty>
       </group>
```


Example #2: Query #3

```
(: Total sale amount and quantity per store and per product :)

for     $s in doc("stores.xml")//store,
        $p in doc("products.xml")//product,
        $f in doc("sales.xml")//sale[@store-id = $s/@id
                                     and @prod-id = $p/@id]

group by $st := $s/@name, $pr := $p/@name

return  <group store="{$st}" product="{$pr}">
          <tot_amount>{sum($f/amount)}</tot_amount>
          <tot_qty>{sum($f/qty)}</tot_qty>
        </group>
```

Example #2: Query #4

```
(: Total sale amount and quantity per city and per product
   (rollup from store to city) :)

for     $c in doc("cities.xml")//city,
        $s in doc("stores.xml")//store[@city-id = $c/@id],
        $p in doc("products.xml")//product,
        $f in doc("sales.xml")//sale[@store-id = $s/@id
                                     and @prod-id = $p/@id]

group by $ct := data($c), $pr := $p/@name

return  <group city="{$ct}" product="{$pr}">
            <tot_amount>{sum($f/amount)}</tot_amount>
            <tot_qty>{sum($f/qty)}</tot_qty>
        </group>
```


Example #2: Query #5

```
(: Total sale amount and quantity per store, per product
   and per month :)

for     $s in doc("stores.xml")//store,
        $p in doc("products.xml")//product,
        $m in distinct-values(doc("times.xml")//time/@month-year),
        $f in doc("sales.xml")//sale[@store-id = $s/@id
            and @prod-id = $p/@id and substring(@date-time, 1, 7) = $m]

group by $st := $s/@name, $pr := $p/@name, $my := $m

return  <group store="{$st}" product="{$pr}" month="{$my}">
            <tot_amount>{sum($f/amount)}</tot_amount>
            <tot_qty>{sum($f/qty)}</tot_qty>
        </group>
```