Database Management Systems

Winter 20023

CMPUT 391: XML and Databases

Dr. Osmar R. Zaïane

University of Alberta

Chapter 27 of Textbook

Course Content

- Introduction
- Database Design Theory
- Query Processing and Optimisation
- Concurrency Control
- Data Base Recovery and Security
- Object-Oriented Databases
- Inverted Index for IR
- Spatial Data Management
- **XML and Databases**
- Data Warehousing
- Data Mining
- Parallel and Distributed Databases

Objectives of Lecture 8

XML and Databases

- Discuss semi-structured data and collections (databases) of semi-structured data.
- Introduce the Extensible Markup Language XML and discuss its use.
- Introduce query languages for querying and manipulating XML documents and XML document collections.

eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

The Structure of Data

- In the real world data can be of any type and not necessarily following any organized format or sequence.
- Such data is said to be unstructured. Unstructured data is chaotic because it doesn't follow any rule and is not predictable.
- Text data is usually unstructured. Many data on the Internet is unstructured (video streams, sound streams, images, etc).

Structured Data

- For applications manipulating data, the structure of data is very important to insure efficiency and effectiveness.
- The data is structured when:
  - Data is organized in semantic chunks (entities).
  - Similar entities are grouped together (relations or classes).
  - Entities in a same group have the same descriptions (attributes).
  - Entity descriptions for all entities in a group have the same defined format, a predefined length, are all present, and follow the same order (schema).
- This structure is sometimes too rigid for some applications.
- What is the alternative? Many data is neither completely unstructured nor completely structured.

Semi-Structured Data

- Structured data is rigidly organized & well defined ➔ predictable
- Unstructured data is disordered and unruly ➔unpredictable
- Semi-structured data is organized enough to be predictable
  - Data is organized in semantic entities
  - Similar entities are grouped together

  But
  - Entities in the same group may not have the same attributes
  - The order of the attributes is not necessarily important
  - The presence of some attributes may not always be required
  - The size of same attributes of entities in a same group may not be the same
  - The type of the same attributes of entities in a same group may not be of the same type.
- An HTML document is an example of semi-structured data

eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

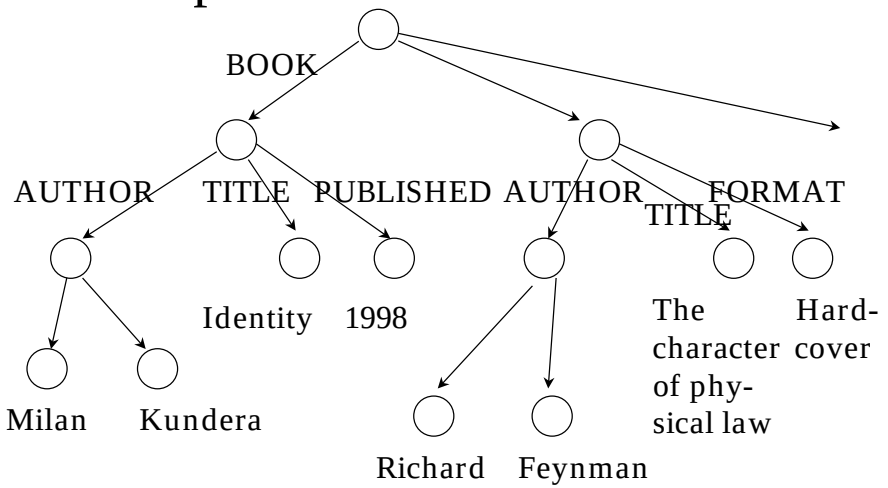
Data Model for Semi-Structured Data

- Semi-structured data doesn't have a schema since some data might be implicit, some might be hidden, unknown, or simply ignored (not entered).
- How do we query the data without knowledge of the schema?
- There are many data models proposed to represent semi-structured data. Most of them use the notion of labeled graphs.

Labeled Graphs

- Nodes in the graph correspond to compound objects or atomic values.
- Edges in the graph correspond to attributes
- The graph is self describing (no need for a schema)
- Object Exchange Model (OEM): each object is described by a triplet <label, type, value>
- Complex objects are decomposed hierarchically into smaller objects

Example: Booklist Data in OEM



eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

Introduction to XML

- XML the eXtensible Markup Language is a standard of the World-Wide Web Consortium
- The official current version is 1.0 and was originally recommended in 1998
- The official specification from the W3C are: http://www.w3.org/TR/1998/REC-xml-19980210
- More info can be found at: http://www.w3.org/XML/
- Many working groups and advisory boards are currently enhancing XML

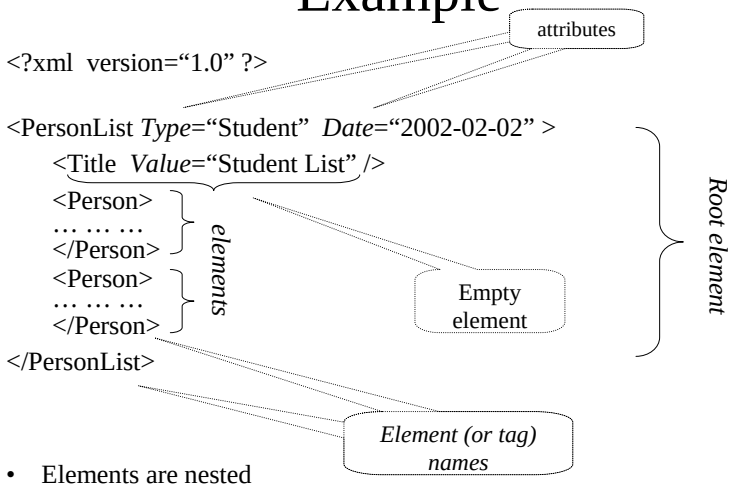
Introduction to XML (con't)

- XML: eXtensible Markup Language
- Suitable for semistructured data
  - Easy to describe object-like data
  - Selfdescribing
  - Doesn't require a schema (but can be provided optionally)
- All major database products have been extended to store and construct XML documents

What is Special with XML

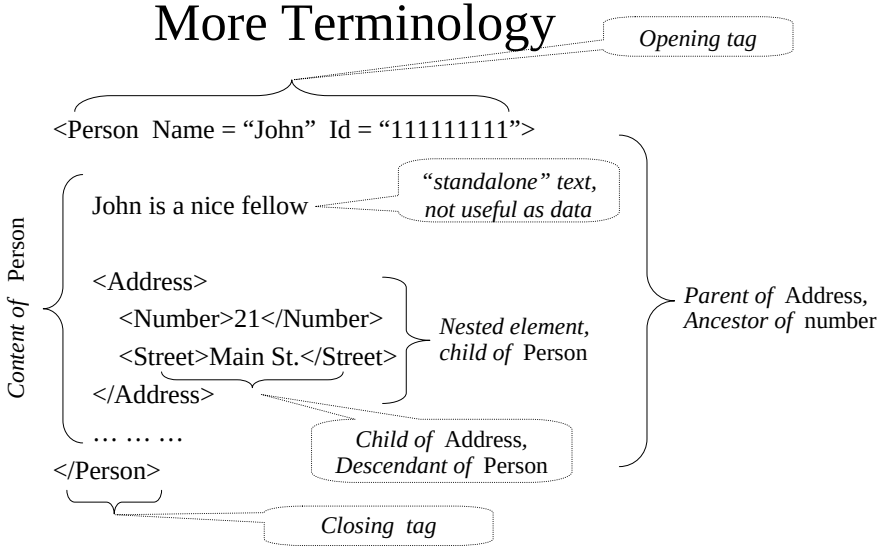
- It is a language to markup data
- There are no predefined tags like in HTML
- Extensible ➔ tags can be defined and extended based on applications and needs
  - Elements / tags
  - Attributes
  - Example: <BOOK page="453">…</BOOK>

Example



- Elements are nested
- Root element contains all others

More Terminology



Rules for Creating XML Documents

- **Rule 1**: All terminating tags shall be closed
  - Omitting a closing XML tag is an error. Example: **<FirstName>**Osmar**</FirstName>**
- **Rule 2**: All non-terminating tags shall be closed
  - Omitting a forward slash for non-terminating tags is an error. Example **<Available** answer="yes"**/>**
- **Rule 3**: XML shall be case sensitive
  - Using the wrong case is an error. Example: **<FirstName>**Osmar**</firstname>**
  - It is OK in HTML <H1>my header</h1>

More Rules for Creating XML

- **Rule 4**: An XML document shall have one root
  - Attempting to create more than one root element would generate a syntax error
- **Rule 5**: Terminating tags shall be properly nested
  - Closing a parent tag before closing a child's tag is an error. Example **<Author><name>**Osmar**</Author></name>**
  - It is OK in HTML <I>bold italic text</I>
- **Rule 6**: Attribute values shall be quoted
  - Omitting quotes, either single or double, around and XML attribute's value is an error. Example **<Product ID="123">**

What is needed?

- XML needs to be parsed to check whether the documents are well formed
- XML needs to be printed
- XML needs to be interpreted for information exchange or populating database
- XML needs to be queried efficiently

eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

Introduction to DTDs

- DTD stands for Document Type Definition
- A DTD is a set of rules that specify how to use an XML markup. It contains specifications for each element, the attributes of the elements, and the values the attributes can take.
- A DTD also specifies how elements are contained in each other
- A DTD ensures that XML documents created by different programs are consistent

Example1: Business Letter

```
<?xml version = "1.0"?>
<letter> <Urgency level="1"\>
 <contact type = "from">
        <name>John Doe</name>
        <address>123 Main St.</address>
        <city>Anytown</city>
        <province>Somewhere</province>
        <postalcode>A1B 2C3</postalcode>
</contact>
<contact type = "to">
        <name>Joe Schmoe</name>
        <address>123 Any Ave.</address>
        <city>Othertown</city>
        <province>Otherplace</province>
        <postalcode>Z9Y 8X7</postalcode>
</contact>
<paragraph>Dear Sir,</paragraph>
<paragraph>It is our privilege to inform you about our new database managed with XML.
This new system will allow you to reduce the load of your inventory list server by having the
client machine perform the work of sorting and filtering the data.</paragraph>
<paragraph>Sincerely, Mr. Doe</paragraph>
</letter>
```


DTD Example for business letter

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE LETTER [
<!ELEMENT LETTER (Urgency, contact+, paragraph+)>
<!ELEMENT Urgency (EMPTY)>
<!ATTLIST Urgency level CDATA #IMPLIED>
<!ELEMENT contact (name, address, city, province,  postalcode,
phone?, email?)>
<!ATTLIST contact type CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
…
]>
```

+ means one or more

Empty means no end tag

? means optional

CDATA means string #IMPLIED means that the attribute value is unspecified.

#PCDATA is parsed character data, it means that the element contains text

DTD Header

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

Version of the xml
Currently 1.0

Encoding specifies the character set used:
•UTF-8 Unicode Transformation 8 bits
•UTF-16 Unicode Transformation 16 bits
•Etc.

Enables use of different character sets ➔ Internationalization

DTD Rules

<!ELEMENT elementName (components or content type)>

Examples: <!ELEMENT name (#PCDATA)>
name is an element/tag for text data

<!ELEMENT Urgency (EMPTY)>
Urgency has no content

<!ELEMENT letter (Urgency, contact+, paragraph+)>
letter is an element that contains and Urgency element followed by one or more contact elements and one or more paragraph elements

Multiple Elements

<!ELEMENT letter (Urgency, contact+, paragraph+)>
<!ELEMENT contact (name, address, city, province, postalcode, phone?, email?)>

Are called multiple elements (lists of elements). They require the rule to specify their sequence and the number of times they can occur.

|	Any element may occur
,	Occur in specified sequence
?	Optional, may occur 0 or once
+	Occurs ate least once (1 or many)
*	Occurs many times (0 or many)

Attributes in DTD

<!ATTLIST elementName attributeName Type Specification>

- elementName and attributeName associate the attribute with the element
- The Type specifies if the attribute is free text (CDATA) or a list of predefined values (value1 | value2 | value3)
- Example:

<!ATTLIST Urgency level CDATA #IMPLIED>
<!ATTLIST contact type CDATA #REQUIRED>
<!ATTLIST P align (center | right | left) #IMPLIED>

- Specification could be:
  - #REQUIRED attribute must be specified
  - #IMPLIED attributes can be unspecified
  - #FIXED attribute is preset to a specific value
  - "defaultvalue" default value for the attribute

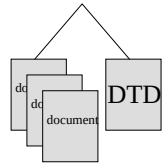
Calling an External DTD

- A DTD can be referenced from XML documents
  - <!DOCTYPE letter SYSTEM "letter.dtd">
  - Any element, attribute not explicitly defined in the DTD generates an error in the XML document.
  - An XML document that conforms to a DTD is called valid and well-formed.
  - There is a need to parse XML documents and validate them vis-à-vis a DTD.
- The keyword **SYSTEM** indicates that the DTD is intended for private use. **PUBLIC** references a public DTD.
- <!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN" "http://my.netscape.com/publish/formats/rss-0.91.dtd">

  Rich Site Summary (RSS) is a lightweight XML format designed for sharing headlines and other Web content. Originally developed by Netscape to fill channels for Netcenter.

Portability of XML

- Adherence to DTDs ensure consistency between XML documents
- Defining a DTD is equivalent to creating a customized markup language.
- There are many domain specific markup languages based on XML: MML (Mathematical Markup Language), CML (Chemical Markup Language),…many other XML-based languages
- This is one of the main reasons why XML is so successful for data exchange between applications



Beyond DTDs: XML Schema

- DTD are limited
  - very limited data types (just strings)
  - can't express strong consistency constraints
  - can't express unordered contents conveniently
  - all element names are global
    - can't have one Name type for people and another for companies:
      - <!ELEMENT Name (Last, First)>
      - <!ELEMENT Name (#PCDATA)>
    - both can't be in the same DTD
- XML Schema solves some of the problems with DTDs, but is much more complex than DTDs

eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

Why do we need to Query XML?

- Methods exist for efficient storage and retrieval of tree-structured objects, including XML documents
- Methods exist for mapping XML elements into relational or Object-Oriented databases.
- Methods exist for indexing semi-structured data.
- Many DBMS vendors are already providing tools for generating XML and even "importing" XML.
- **Very large collections of XML documents are prominent and inevitable.**

Querying XML Data

- Goal: High-level, declarative language that allows manipulation of XML documents.
- Manipulation means the retrieval of documents, sub-documents and elements and attribute values, as well as the generation of new XML documents.
- There are many languages proposed by researchers. One standard emerged recently (X-Query adopted by W3C)
- Lorel (1997), XML-QL (1999), XQL (1999), Quilt(2000), XQuery (2001), …
- Also XPath (lightweight XML query language) and XSLT (transformation language for XML)

Comparison of Languages

Some languages follow a database perspective other a document processing perspective.

In most XML query languages

- Typically, queries consist of 3 parts: a pattern clause, a filter clause and constructor clause. (also sorting, grouping…)
- Use of external functions such as aggregation functions, string related functions, etc.
- Use of constructs to impose nesting and order
- Use of join operator to combine data from different portions of documents.
- Use of tag variables or path expressions
- Use of constructs to test absence of data

Not all are available with all languages

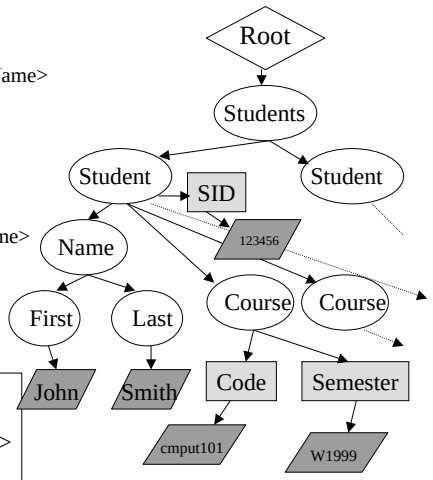
XPath: XML Pointer Language

- http://www.w3.org/TR/xpath or http://www.w3.org/TR/1999/REC-xpath-19991116
- A core query language used in X-Query and many other XML standards
- Simple selection operator for paths from XML-tree
- XML documents are modeled as trees
- In XPath document tree nodes are either elements, attributes, or text values (also comments). There is also an extra root.
- Xpath expressions take a document tree and return a set of nodes in the tree.

XPath Document Tree

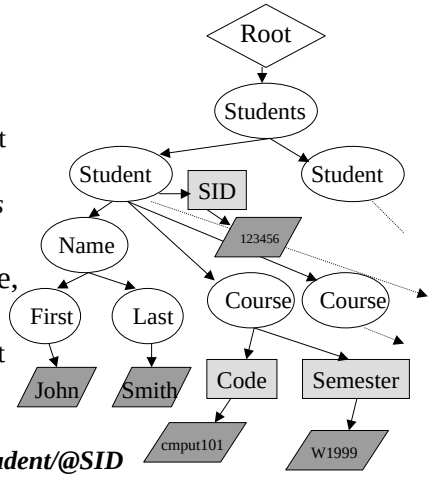
```
<? Xml version="1.0" ?>
<Students>
  <Student SID="123456">
    <Name><First>John</First><Last>Smith</Last></Name>
    <Status>Full-UndG</Status>
    <Course Code="cmput101"  Semester="W1999" />
    <Course Code="cmput291"  Semester="F1999" />
    <Course Code="cmput391"  Semester="F2000" />
  </Student>
 <Student SID="678123">
    <Name><First>Jane</First><Last>Doe</Last></Name>
    <Status>Full-UndG</Status>
    <Course Code="cmput114"  Semester="F1999" />
    <Course Code="cmput304"  Semester="F2000" />
</Student>
</Students>
```

```
<!DOCTYPE students [
   <!ELEMENT Students (Student*)>
   <!ELEMENT Student (Name, Status, Course*)>
   <!ELEMENT Name (First, Last)>
   <!ELEMENT First (#PCDATA)>
…]>
```



XPath Expressions

- Absolute path expressions:
  /Students/Student/Name
  refers to the composite *Name*
  //Name
  refers to *Name* descendent of the root
  /Students//First
  refers to descendent *First* of *Students*
- Relative path expressions:
  if current node corresponds to Name,
  ./First is first name of current
  ../Course a course of the current student
  ..//..//First is the first name of siblings,
  (// denotes arbitrary sub-path)
- @ is used for attributes: ***/Students/Student/@SID***
- ***/Students/Student[1]/Course[3]*** for specific nodes



XPath with Selection Conditions (Xpath Queries)

- Select all student who took a course in F2000

```
//Student[Course/@Semester="F2000"]
```

- Select undergrad students with last name starting with 'B'

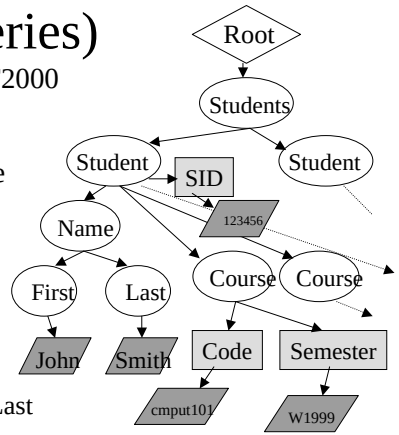
```
//Student[Status="UndG" AND
        starts-with(.//Last, "B")]
```

- Select last names of students who took cmput391

```
//Student[Course/@Code="cmput391"]/Name/Last
```

- Select all students who took cmput391 in F2001

```
//Student[Course/@Code="cmput391" AND   Course/@Semester="F2001"]
```



XML-QL as defined in Textbook (2nd ed)

- Takes advantage of the structure in XML
- Has a WHERE clause to define selectivity and a CONSTRUCT clause to define the results. Used also for restructuring XML documents.
- Find the last name of all book authors published by McGrawHill: (selections are expressed by placing text in content of elements.

```
WHERE
  <BOOK>
    <NAME><LAST>$1</LAST></NAME>
    <PUBLISHER>McGrawHill</PUBLISHER>
  </BOOK> in "www.booklist.com/books.xml
CONSTRUCT <RESULT> $1 </RESULT>
```

```
<RESULT> Smith</RESULT>
<RESULT>Amberger</RESULT>
```

XML-QL (Contd.)

Find the last names of authors of each book:

```
WHERE <BOOK>  $b  </BOOK> IN
   "www.booklist.com/books.xml",
  <AUTHOR> $n </AUTHOR>
  <TITLE> $t </TITLE> in $b
CONSTRUCT
  <RESULT>
    <TITLE> $t </TITLE>
      WHERE <LAST> $l </LAST> IN $n
      CONSTRUCT <LAST> $l </LAST>
  </RESULT>
```

```
<RESULT>
     <TITLE>my first book</TITLE>
       <LAST>Smith</LAST>
       <LAST>Jaup</LAST>
</RESULT>
<RESULT>
     <TITLE>DB How to program</TITLE>
     <LAST>Amberger</LAST>
</RESULT>
```


Comparing XML-QL and Lorel

- Based on paper: "XML Query Languages: Experiences and Examples" by Mary Fernandez, Jerome Simeon and Philip Wadler
- DTD example:

```
<!ELEMENT bib  (book* )>
 <!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>
 <!ATTLIST book  year CDATA  #REQUIRED >
 <!ELEMENT author  (last, first )>
 <!ELEMENT editor  (last, first, affiliation )>
 <!ELEMENT title  (#PCDATA )>
 <!ELEMENT last  (#PCDATA )>
 <!ELEMENT first  (#PCDATA )>
 <!ELEMENT affiliation  (#PCDATA )>
 <!ELEMENT publisher  (#PCDATA )>
 <!ELEMENT price  (#PCDATA )>
```

Assume it is in bib.xml

Simple Selection Example

- Select all titles of books published by Addison-Wesley after 1998.

In Lorel

```
Select xml(bib:{
  (Select xml(book:{@year:y, title:t})
   From bib.book b, b.title t, b.year y
   Where b.publisher = "Addison-Wesley" and y > 1998)})
```

In XML-QL

```
CONSTRUCT <bib> {
WHERE
   <bib>
    <book year=$y>
      <title>$t</title>
      <publisher>Addison-Wesley</publisher>
    </book>
   </bib> IN "www.books.com/bib.xml",
   $y > 1998
CONSTRUCT <book year=$y><title>$t</title></book>
} </bib>
```


Sorting Example

- Select all titles of books published by Addison-Wesley after 1998 ordered alphabetically.

In Lorel

```
Select xml(bib:{
  (Select xml(book:{@year:y, title:t})
   From bib.book b, b.title t, b.year y
   Where b.publisher = "Addison-Wesley" and y > 1998
    Order by t)})
```

In XML-QL

```
CONSTRUCT <bib> {
WHERE
   <bib>
    <book year=$y>
      <title>$t</title>
      <publisher>Addison-Wesley</publisher>
    </book>
   </bib> IN "www.books.com/bib.xml",
   $y > 1998
   ORDER-BY $t
CONSTRUCT <book year=$y><title>$t</title></book>
} </bib>
```

Selection Preserving Structure

- Return all titles and their author list.

In Lorel

```
Select xml(results:
  Select xml(result:{b.title, b.author})
  From bib.book b)
```

In XML-QL

```
CONSTRUCT <results> {
 WHERE
  <bib><book> <title>$t</title></book>
       CONTENT_AS $b
  </bib> IN "www.books.com/bib.xml"
 CONSTRUCT
  <result> <title>$t</title>
   { WHERE <author>$a</author> IN $b
     CONSTRUCT <author>$a</author>}
  </result>
} </results>
```


Selection Flattening Structure

- Return all title-author pairs.

In Lorel

```
Select xml(results:
  (Select xml(result:{title: t,
                      author: a})
   From bib.book b, b.title t, b.author a))
```

In XML-QL

```
CONSTRUCT <results> {
 WHERE
  <bib> <book>
    <title>$t</title><author>$a</author>
   </book>
  </bib> IN "www.books.com/bib.xml"
 CONSTRUCT
  <result> <title>$t</title>
           <author>$a</author>
  </result>
} </results>
```


Selection Creating New Structure

- Return all titles by author.
- Creates a structure different from original XML document
- Requires a join on author's name

In Lorel

```
Select xml(results:
  (Select xml(result:{author: a,
                      (Select xml(title: t)
                       From bib.book b, b.title t
                       Where
                        b.author.first = a.first and
                        b.author.last = a.last)})
   From bib.book.author a))
```


Selection Creating New Structure

- Return all titles by author.

In XML-QL

```
CONSTRUCT <results> {
 WHERE
  <bib><book>
    <author><last>$l</last><first>$f</first></author>
   </book>  </bib> IN "www.books.com/bib.xml"
 CONSTRUCT
  <result> <author><last>$l</last><first>$f</first></author>
   {
    WHERE
     <bib> <book>
       <title>$t</title>  // join on $l and $f
       <author><last>$l</last><first>$f</first></author>
      </book> </bib> IN "www.books.com/bib.xml"
    CONSTRUCT <title>$t</title>
   }
  </result>
} </results>
```

Combining Data Sources

- How do we combine different fragments of documents from different XML documents?
- Suppose we have another XML document at www.anotherbook.com/reviews.xml with the following DTD:

```
<!ELEMENT reviews (entry*)>
 <!ELEMENT entry   (title, price, review)>
 <!ELEMENT title   (#PCDATA)>
 <!ELEMENT price   (#PCDATA)>
 <!ELEMENT review  (#PCDATA)>
```


Selection From Multiple Sources

- Return all book tiles with their prices in both sources.

In Lorel

```
Select xml(books-with-prices:
  (Select xml(book-with-prices: { title: ta,
                                  price-anotherbook: pa,
                                  price-books: pb }
   From bib.book b, b.title tb, b.price pb,
     reviews.entry e, e.title ta, e.price pa
   Where tb = ta)))
```


Selection From Multiple Sources

- Return all book tiles with their prices in both sources.

In XML-QL

```
CONSTRUCT <books-with-prices> {
  WHERE
    <bib> <book> <title>$t</title> <price>$pb</price> </book>
    </bib> IN "www.books.com/bib.xml“
    <reviews> <entry> <title>$t</title> <price>$pa</price> </entry>
    </reviews> IN "www.anotherbook.com/reviews.xml"
  CONSTRUCT
    <book-with-prices> <title>$t</title>
      <price-anotherbook>$pa</price-anotherbook>
      <price-books>$pb</price-books>
    </book-with-prices>
} </books-with-prices>
```


Example for X-Query

Consider a set of XML documents defined by the following DTD

```
<!DOCTYPE BOOKLIST> [
<!ELEMENT BOOLIST (BOOK)*>
   <!ELEMENT BOOK ( AUTHOR+, TITLE, PUBLISHED?)>
     <!ELEMENT AUTHOR ( FIRSTNAME,LASTNAME)>
           <!ELEMENT FIRSTNAME(#PCDATA)>
           <!ELEMENT LASTNAME(#PCDATA)>
     <!ELEMENT TITLE (#PCDATA)>
     <!ELEMENT PUBLISHED (#PCDATA) >
   <!ATTLIST BOOK GENRE(Science|Fiction) #REQURIED>
   <!ATTLIST BOOK FORMAT (Paperback|Hardcover) “Paperback”>
]>
```

Example (Cont.)

```
<?XML VERSION="1.0" ENCODING="UTF-8" STANDALONE="YES">
<BOOKLIST>
    <BOOK GENRE="Science" FORMAT= "Hardcover">
        <AUTHOR> <FIRSTNAME> Richard</FIRSTNAME>
                 <LASTNAME>Feynman</LASTNAME>
        </AUTHOR>
        <TITLE> The Character of Physical Law</TITLE>
        <PUBLISHED> 1980</PUBLISHE>
    </BOOK>
    <BOOK GENRE="Fiction">
        <AUTHOR> <FIRSTNAME>R.K.</FIRSTNAME>
                 <LASTNAME>Narayan</LASTNAME>
        </AUTHOR>
        <TITLE> Waiting for the Mahatma</TITLE>
        <PUBLISHED> 1981</PUBLISHE>
    </BOOK>
    <BOOK GENRE="Fiction">
        <AUTHOR> <FIRSTNAME>R.K.</FIRSTNAME>
                 <LASTNAME>Narayan</LASTNAME>
        </AUTHOR>
        <TITLE> The English Teacher</TITLE>
        <PUBLISHED> 1980</PUBLISHE>
    </BOOK>
</BOOKLIST>
```


XQuery Basics

- General structure:
  - [FOR | LET] variable declarations
  - WHERE condition
  - RETURN document
  - variable declaration:
    "$" VarName "in" Expression
- Variable binding
  - FOR binds a variable to each element satisfying the expression
  - LET binds a variable to the whole collection of elements that satisfy the expression

Example – FOR clause

- Assume the previous document is stored at www.outbookstore.com/books.xml
- QUERY: Find the last names of all authors

FOR
  $l **IN** doc(www.ourbookstore.com/books.xml)//AUTHOR/LASTNAME
RETURN
  <RESULT> $l </RESULT>

ANSWER
  <RESULT><LASTNAME> Feynman</LASTNAME></RESULT>
  <RESULT><LASTNAME> Narayan</LASTNAME></RESULT>

Example – LET clause

LET
  $l **IN** doc(www.ourbookstore.com/books.xml)//AUTHOR/LASTNAME
RETURN
  <RESULT> $l </RESULT>

ANSWER
  <RESULT>
    <LASTNAME> Feynman</LASTNAME>
    <LASTNAME> Narayan</LASTNAME>
  </RESULT>

Example: WHERE clause

FOR
  $1 **IN** doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK
WHERE $1/PUBLISHED = “1980”
RETURN
  <RESULT> $1/AUTHOUR/FIRSTNAME, $1/AUTHOR/LASTNAME
  </RESULT>

ANSWER

```
<RESULT>
  <FIRSTNAME> Richard <.FIRSTNAME>
  <LASTNAME>Feynman</LASTNAME>
</RESULT>
<RESULT>
  <FIRSTNAME> R.K.<.FIRSTNAME>
  <LASTNAME>Narayan</LASTNAME>
</RESULT>
```


Example: Nested Queries & Grouping

FOR $l **IN DISTINCT**
  doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK/PUBLISHED
RETURN
<RESULT>
  $1,
  FOR $a IN DISTINCT doc(www.ourbookstore.com/books.cml)
    /BOOKLIST/BOOK[PUBLISHED=$1]/AUTHOUR/LASTNAME
  RETURN $a
</RESULT>

ANSWER

```
<RESULT>
  <PUBLISHED> 1980 </PUBLISHED>
  <LASTTNAME> Feynman<LASTNAME>
  <LASTNAME>Narayan</LASTNAME>
</RESULT>
<RESULT>
  <PUBLISHED> 1981<PUBLISHED>
  <LASTNAME>Narayan</LASTNAME>
</RESULT>
```


Example: Join & Aggregation

FOR $a **IN DISTINCT**
  doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK/AUTHOR
LET $t **IN**
  doc(www.ourbookstore.com/books.xml)//BOOK/[AUTHOR=$a]/TITLE
RETURN
<RESULT>
    $a/LASTNAME, <TotalBooks> count(distinct($t)) </TotalBooks>
</RESULT>
SORT BY (LASTNAME descending)

ANSWER (e.g.)

```
<RESULT>
  <LASTTNAME>Narayan<LASTNAME>
  <TotalBooks> 5 </TotalBooks>
</RESULT>
<RESULT>
  <LASTNAME>Feynman</LASTNAME>
  <TotalBooks> 2 </TotalBooks>
</RESULT>
```


How to store and retrieve XML Data?

- Storing XML data in the file system
- Storing XML in BLOB/CLOB
- Native XML databases
- XML enabled databases

Native XML databases

- Database designed especially to store XML documents
  - data model is based on XML
  - XML query languages
- Why ?
  - large collections of semi-structured data
  - retrieval speed
- Why not?
  - development cost
  - market
  - not all data are XML data

XML Enabled Databases

- Store and query XML data in a relational database
  - convert XML data into a set of tuples and store them into tables
  - a query to XML will be converted to SQL queries to the database
- Problems with this approach:
  - the translation process requires a schema for the data
  - the query evaluation is very inefficient

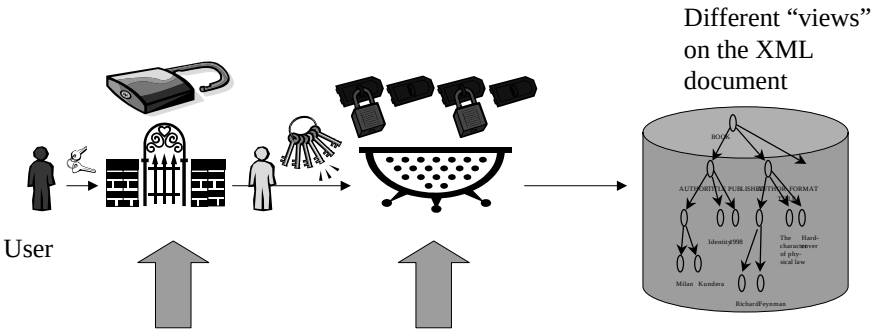
Open Research Questions

- Query Optimization
- Security and Access Control for XML documents
- Indexing XML Data
  - Value Index (e.g. B^+-tree)
  - Structure Index (Path indexing)

eXtensible Markup Language

- Semi-Structured Data
- Data Model for XML
- Introduction to XML
- Syntax and Document Type Definition
- Querying XML Documents
- XML and Security Access

Access Control and XML



Subjects and Objects

- State authorizations on elements/attributes or sub-trees.
- Subjects and objects are defined against these authorizations.
- Subject is a pair <user, address> where the user is either a user ID or a group ID, address is an IP address or a symbolic Internet address.
- Objects are XML elements defined using XPath.
- A hierarchy on the addresses can be created using wild cards.
- Authorization are read/write/update, + or -

Access Control and Views

- When a user requests access to an XML document the XML tree is labelled with the authorizations allowed to the user accessing from the given host, and only the trees with label "+" are seen ➔ view
- Example of authorizations:

<<Admin, *.lab.org>, LabReports.xml:project[./@type="internal"],read,+>
<<Public,*>,LabReports.xml:/laboratory//reviews[.@category="private"],read,->
<<Osmar,*.ualberta.ca>,LabReports.xml:/Projects,read,+>