XML, JSON, and NoSQL

UBCO Master of Data Science – DATA 540

XML

Extensible Markup Language (***XML***) is a markup language that allows for the description of data semantics.

- XML can describe any type of data because the markup terms are user-defined.
- XML is **case-sensitive** unlike HTML. **
 or
**
- XML is a standard by the World Wide Web Consortium (W3C).

Advantages of XML:

- Simplicity, open standard, extensibility, interoperability, separation of data and presentation

XML Components

An XML document is a text document that contains markup in the form of ***tags***. An XML document consists of:

- An XML *declaration line* indicating the XML version.
- *Elements* (or tags) called *markup*. Each element may contain free-text, attributes, or other nested elements.
  - Every XML document has a single root element.
  - Tags, as in HTML, are matched pairs, as <item> … </item>.
  - Closing tags are not needed if the element contains no data: <item/>
  - Tags may be nested.
- An *attribute* is a name-value pair declared inside an element.
- Comments

XML data is ordered by nature.

XML Example

```
<?xml version = "1.0" encoding="UTF-8" ?>
<?xml-stylesheet type="text/xsl" href="dept.xsl"?>
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="https://myloc/schema.xsd">
<!-- Emp/Dept in XML -->
 <Dept dno = "D1">
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
 </Dept></root>
```

XML declaration

Stylesheet for presentation

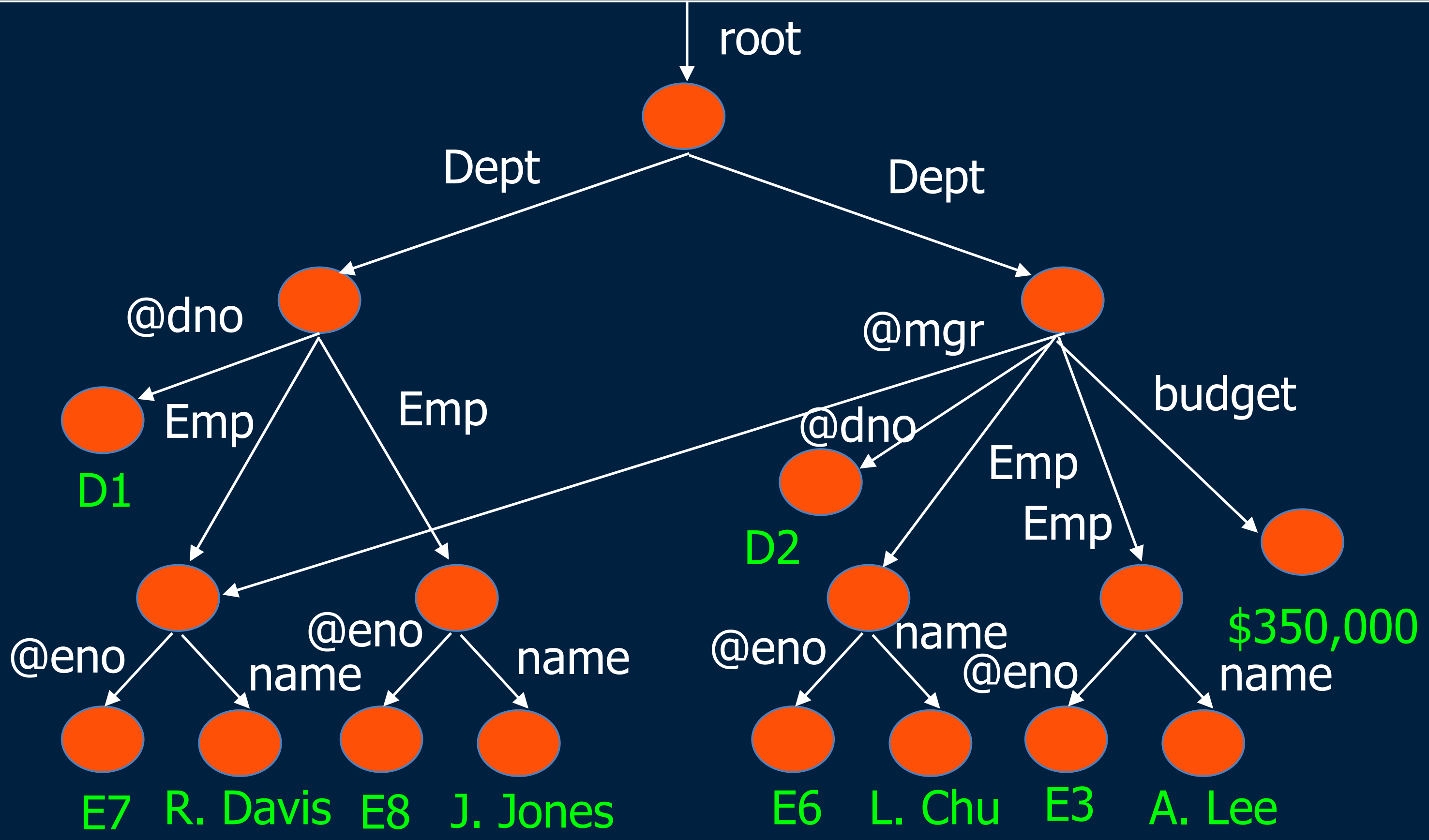
XML Schema for validation

Attribute

Comment

Element reference

XML (tree view)



Well-Formed and Valid XML Documents

An XML document is ***well-formed*** if it obeys the syntax of the XML standard. This includes:

- Having a single root element
- All elements must be properly closed and nested.

/ >

An XML document is ***valid*** if it is well-formed and it conforms to a Document Type Definition (DTD) or an XML Schema Definition (XSD).

- A document can be well-formed without being valid if it contains tags or nesting structures that are not allowed in its DTD/XSD.
- The DTD/XSD are schema definitions for an XML document.

XML Well-Formed Question

Question: How many of these two documents are well-formed?

1: `<x><a>Test</a><bT></bt></x>`

2: `<x>abc</x><y>def</y>`

A) 0

B) 1

C) 2

Namespaces

Namespaces allow tag names to be qualified to avoid naming conflicts. A naming conflict would occur when the same name is used by two different domains or vocabularies.

A namespace consists of two components:

- 1) A declaration of the namespace and its abbreviation.
- 2) Prefixing tag names with the namespace name to exactly define the tag's origin.

Namespaces Example

```
<?xml version = "1.0" encoding="UTF-8" standalone="no"?>
<root xmlns = "http://www.foo.com"
      xmlns:n1 = "http://www.abc.com">
 <Dept dno = "D1">
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <n1:budget>350000</n1:budget>
 </Dept>
</root>
```

Default namespace

n1 namespace

budget is a XML tag in the n1 namespace.

Schemas for XML

Although an unrestricted XML format is useful to some applications, database data normally has some structure, even though that structure may not be as rigid as relational schemas.

It is valuable to define schemas for XML documents that restrict the format of those documents.

Two ways of specifying a schema for XML:

- XML Schema
- Document Type Definition (DTD) (original, older)

Document Type Definitions (DTDs)

A ***Document Type Definition*** (***DTD***) defines the grammatical rules for the document. It is not required for an XML document but provides a mechanism for checking a document's validity. General DTD form:

```
<!DOCTYPE myroot [ <elements> ]>
```

name of root element in XML document

contents of DTD declares elements and attributes

A DTD is a set of document rules expressed using EBNF (Extended Backus-Naur Form) grammar. The rules limit:

- the set of allowable element names, how elements can be nested, and the attributes of an element among other things

DTD Example

```
<!DOCTYPE root [
 <!ELEMENT root(Dept+)>
 <!ELEMENT Dept(Emp*, budget?)>
        <!ATTLIST Dept dno ID #REQUIRED>
        <!ATTLIST Dept mgr IDREF #IMPLIED>
 <!ELEMENT budget (#PCDATA)>
 <!ELEMENT Emp (name)>
        <!ATTLIST Emp eno ID #REQUIRED>
 <!ELEMENT name (#PCDATA)>
]>
```

+ means 1 or more times

* means 0 or more times

? means 0 or 1 time

Element reference (like a foreign key)

ID is a unique value that identifies the element

Parsed Character Data (atomic value)

XML DTD Question

Question: How many of the following statements are **TRUE**?

1) Every XML document requires a DTD.
2) A document can be valid even if it does not have a DTD or XML Schema.
3) A + means 1 or more times.
4) A ? means 0 or more times.
5) A * means 0 or 1 times.
6) The order of elements listed in a DTD matters.

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5

XML Schema

XML Schema was defined by W3C to provide a standard XML schema language written in XML with better support for data modeling.

XML Schema Example

```
<?xml version = "1.0" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name = "root">
 <xsd:complexType>
   <xsd:sequence>
    <xsd:element name="Dept" minOccurs="1" maxOccurs="unbounded">
       <xsd:complexType>
         <xsd:sequence>
           <xsd:element name="Emp" minOccurs="0"
                                     maxOccurs="unbounded">
             <xsd:complexType>
               <xsd:sequence>
```

Root element is called `root`

Complex type contains other elements

Min and max number occurrences

XML Schema Example (2)

```
              <xsd:element name = "name" type = "xsd:string" />
            </xsd:sequence>
            <xsd:attribute name = "eno" type = "xsd:string" />
         </xsd:complexType>
       </xsd:element>
       <xsd:element name="budget" minOccurs="0"
                                        type ="xsd:decimal" />
     </xsd:sequence>
     <xsd:attribute name = "dno" type = "xsd:string" />
     <xsd:attribute name = "mgr" type = "xsd:string" />
   </xsd:complexType>
  </xsd:element>
 </xsd:sequence>
</xsd:complexType>
```

Simple type (has data type)

XML Schema Example (3)

```
<xsd:key name = "DeptKey">
      <xsd:selector xpath = "Dept" />
      <xsd:field xpath = "@dno" />
</xsd:key>
<xsd:key name = "EmpKey">
      <xsd:selector xpath = "Dept/Emp" />
      <xsd:field xpath = "@eno" />
</xsd:key>
<xsd:keyref name = "DeptMgrFK" refer = "EmpKey">
      <xsd:selector xpath = "Dept" />
      <xsd:field xpath = "@mgr" />
</xsd:keyref>
</xsd:element></xsd:schema>
```

Key constraints

Reference to another key (like a FK)

Other XML Technologies

An ***XML parser*** processes the XML document and determines if it is well-formed and valid (if a schema is provided).

Once a document is parsed, programs manipulate the document using one of two interfaces: DOM (tree-based) and SAX (event-based).

- Note: May process XML documents without a parser as document is a text file.

XSL (eXtenstible Stylesheet Language) defines how XML data is displayed.

- Similar to Cascading Stylesheet Specification (CSS) used with HTML.

XSLT (eXtenstible Stylesheet Language for Transformations) is a subset of XSL that provides a method for transforming XML (or other text documents) into other documents (XML, HTML).

Querying XML

XPath allows you to specify path expressions to navigate the tree-structured XML document.

XQuery is a full query language that uses XPath for path expressions (not studied).

Example XML Document

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

<Depts>
 <Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
 </Dept>
</Depts>
```

Path Descriptions in XPath

XPath provides the ability to navigate through a document using path descriptors.

Path descriptors are sequences of tags separated by slashes /.

- If the descriptor begins with `/`, then the path starts at the root.
- If the descriptor begins with `//`, the path can start anywhere.
- You may also start the path by giving the document name such as `doc(depts.xml)/`.

A path descriptor denotes a sequence of nodes. Examples:

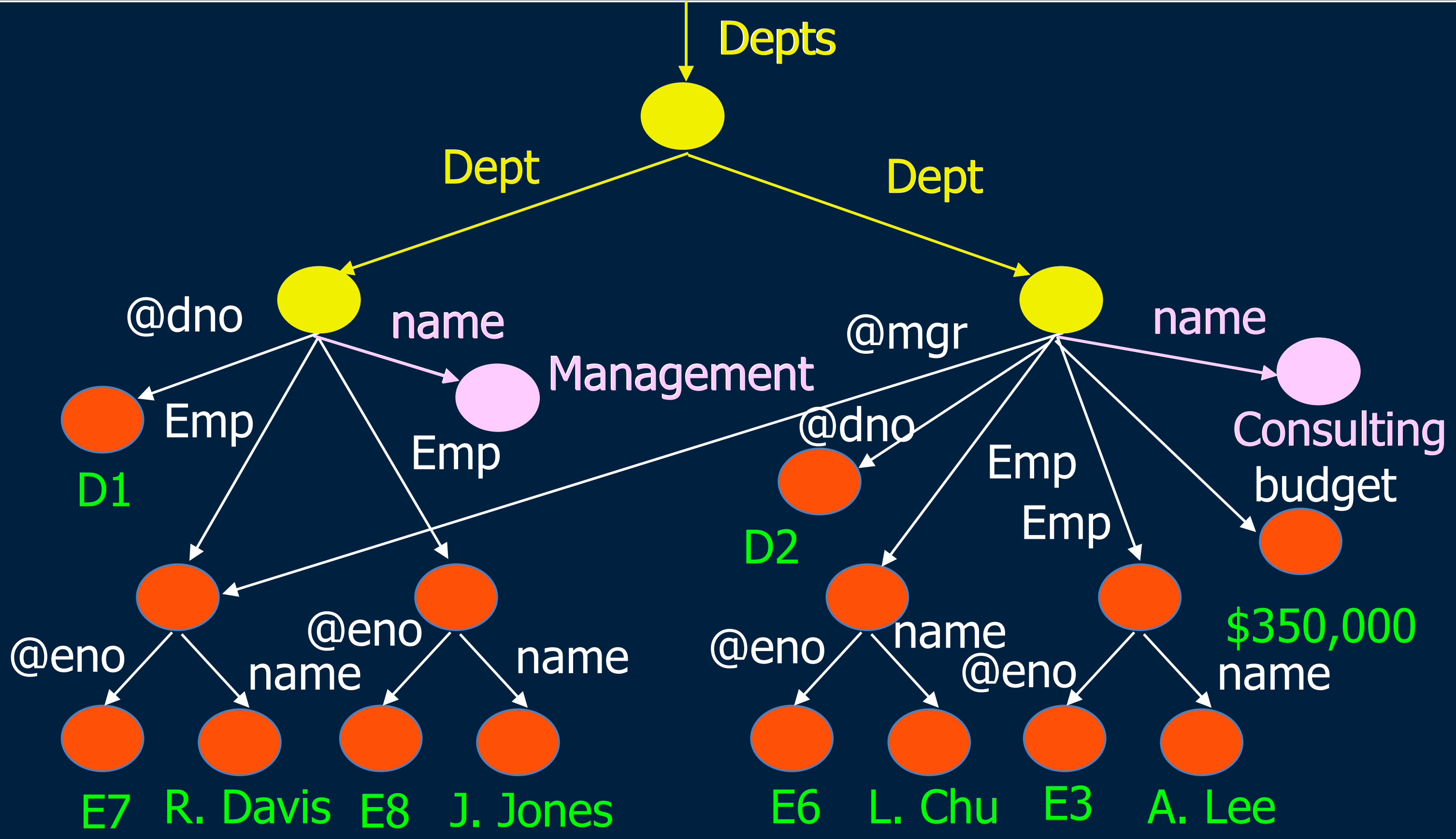
- `/Depts/Dept/name`
- `//Dept/name`
- `doc("depts.xml")/Depts/Dept/Emp/name`

Path: /Depts/Dept/name

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

<Depts>
 <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
 </Dept>
</Depts>
```

Path: /Depts/Dept/name (tree view)



Path: //Dept/name

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

<Depts>
 <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
 </Dept>
</Depts>
```

Path query returns same answer as previous one.

Path: //name

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

<Depts>
 <Dept dno = "D1">
      <name>Management</name>
      <Emp eno="E7"><name>R. Davis</name></Emp>
      <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
      <name>Consulting</name>
      <Emp eno="E6"><name>L. Chu</name></Emp>
      <Emp eno="E3"><name>A. Lee</name></Emp>
      <budget>350000</budget>
 </Dept>
</Depts>
```

Matches any name tag starting from anywhere in the document.

Path: /Depts/Dept

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

<Depts>
<Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
</Dept>
<Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
</Dept>
</Depts>
```

Wild Card Operator

The "*" wild card operator can be used to denote any ***single*** tag.

Examples:

- `/*/*/name` - Match any name that is nested 3 levels deep
- `//*` - Match anything

Question: What is /*/*/* ?

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

<Depts>
 <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
 </Dept>
</Depts>
```

Attributes

Attributes are referenced by putting a "@" in front of their name.

Attributes of a tag may appear in paths as if they were nested within that tag.

Examples:

- `/Depts/Dept/@dno` - `dno` attribute of `Dept` element
- `//Emp/@eno` - `eno` attribute of `Emp` element

Path: /Depts/Dept/@dno

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

<Depts>
 <Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
 </Dept>
</Depts>
```

Question: What is /*/*/@eno ?

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

<Depts>
 <Dept dno = "D1">
        <name>Management</name>
        <Emp eno="E7"><name>R. Davis</name></Emp>
        <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
        <name>Consulting</name>
        <Emp eno="E6"><name>L. Chu</name></Emp>
        <Emp eno="E3"><name>A. Lee</name></Emp>
        <budget>350000</budget>
 </Dept>
</Depts>
```

Predicate Expressions

The set of objects returned can be filtered by putting selection conditions on the path.

A ***predicate expression*** may be specified inside square brackets **[..]** following a tag. Only paths that have that tag and also satisfy the condition are included in the result of a path expression.

Examples:

- `/Depts/Dept/name[.="Management"]`
- `//Depts/Dept[budget>250000]`
- `//Emp[@eno="E5"]`

//Depts/Dept/budget[.>250000]

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

<Depts>
 <Dept dno = "D1">
       <name>Management</name>
       <Emp eno="E7"><name>R. Davis</name></Emp>
       <Emp eno="E8"><name>J. Jones</name></Emp>
 </Dept>
 <Dept dno = "D2" mgr = "E7">
       <name>Consulting</name>
       <Emp eno="E6"><name>L. Chu</name></Emp>
       <Emp eno="E3"><name>A. Lee</name></Emp>
       <budget>350000</budget>
 </Dept>
</Depts>
```

Note no budget element in first Dept so does not match path.

Axes and Abbreviations

XPath defines ***axes*** that allow us to go from the current node to other nodes. An axis to traverse is specified by putting the axis name before the tag name to be matched such as `child::Dept`.

Common axes have abbreviations:

- The default axis is `child::` which contains all children. Since it is the default, the child axis does not have to be explicitly specified.
  - `/Depts/Dept` is shorthand for `/Depts/child::Dept`
- **@** is a shorthand for the `attribute::` axis.
  - `/Depts/Dept/@dno` is short for `/Depts/Dept/attribute::dno`
- `..` is short for the `parent::` axis.
- `.` is short for the `self::` axis (current node).
- `//` is short for `descendant-or-self::` axis
  - // matches any node or any of its descendants

Summary of XPath Constructs

Symbol	Usage
/	Root element or separator between path steps
*	Match any single element name
@X	Match attribute X of current element
//	Match any descendant (or self) of current element
[C]	Evaluate condition on current element
[N]	Picks the N^{th} matching element (indexed from 1)
..	Matches parent element
.	Matches current element

DTD for Questions

```
<!DOCTYPE Bookstore [
 <!ELEMENT Bookstore (Book | Magazine)*>
 <!ELEMENT Book (Title, Authors, Remark?)>
 <!ATTLIST Book ISBN CDATA #REQUIRED>
 <!ATTLIST Book Price CDATA #REQUIRED>
 <!ATTLIST Book Edition CDATA #IMPLIED>
 <!ELEMENT Magazine (Title)>
 <!ATTLIST Magazine Month CDATA #REQUIRED>
 <!ATTLIST Year CDATA #REQUIRED>
 <!ELEMENT Title (#PCDATA)>
 <!ELEMENT Authors (Author+)>
 <!ELEMENT Remark (#PCDATA)>
 <!ELEMENT Author (First_Name, Last_Name)>
 <!ELEMENT First_Name (#PCDATA)>
 <!ELEMENT Last_Name (#PCDATA)>]
```

Example XML Document for Questions

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

<Bookstore>
 <Book ISBN="ISBN-0-201-70857-4" Price="65" Edition="3rd">
 <Title>Database Systems</Title>
 <Authors>
   <Author><First_Name>Thomas</First_Name><Last_Name>Connolly</Last_Name> </Author>
   <Author><First_Name>Carolyn</First_Name><Last_Name>Begg</Last_Name></Author>
  </Authors>
 </Book>
 <Book ISBN="ISBN-0-13-031995-3" Price="75">
 <Title>Database Systems: The Complete Book</Title>
 <Authors>
 <Author><First_Name>H.</First_Name><Last_Name>Garcia-Molina</Last_Name></Author>
  <Author><First_Name>Jeffrey</First_Name><Last_Name>Ullman</Last_Name> </Author>
  <Author> <First_Name>Jennifer</First_Name> <Last_Name>Widom</Last_Name> </Author>
 </Authors>
 <Remark> Amazon.com says: Buy these books together for a great deal!</Remark>
 </Book> </Bookstore>
```

XPath Questions

What are the elements selected by these XPath queries:

- `/Bookstore/*/Title`
- `//First_Name[.="Thomas"]`
- `//Last_Name[.="Ullman"]/../../..[@Price < 60]`

Write XPath queries to retrieve:

- all book titles
- all books < $70
- all last names anywhere
- all books containing a remark
- all book titles where the book < $80 and Ullman is an author
- retrieve the second book

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) is a method for serializing data objects into text form.

Benefits:

- Human-readable
- Supports semi-structured data
- Supported by many languages (not just JavaScript)

Often used for data interchange especially with AJAX/REST from web server to client.

JSON Example

JSON constructs:

- **Values:** number, strings (double quoted), true, false, null
- **Objects:** enclosed in { } and consist of set of key-value pairs
- **Arrays:** enclosed in [ ] and are lists of values
- Objects and arrays can be nested.

```
{"Employees": [{"eno":"E1", "ename":"J. Doe", "title":"EE", "salary":30000, "WorksOn":["P1"]},
               {"eno":"E2", "ename":"M. Smith", "title":"SA", "salary":50000, "WorksOn":["P1","P2"]},
               {"eno":"E3", "ename":"A. Lee", "title":"ME", "salary":40000, "WorksOn":["P3"]}
],
"Projects": [{"pno":"P1", "pname":"Instruments", "budget":150000},
               {"pno":"P2", "pname":"DB Develop", "budget":135000},
               {"pno":"P3", "pname":"Budget", "budget":250000}
]
}
```

JSON versus Relations

	JSON	Relational
Structure	Nested objects + arrays	Tables
Schema	Variable (and not required)	Fixed
Queries	Limited	SQL, RA
Ordering	Arrays are sorted	No
Systems	Used with programming languages and NoSQL systems	Many commercial and open source systems
Case-sensitive?	Yes	No (mostly)

JSON Parsers

A **JSON parser** converts a JSON file (or string) into program objects assuming no syntactic errors.

- In JavaScript, can call `eval()` method on variable containing a JSON string.

A **JSON validator** validates according to a schema and then performs the parsing.

Online validation tool: http://jsonlint.com

Using JSON in Programs

Many programming languages have APIs to allow for the creation and manipulation of JSON.

One common usage is for the JSON data to be provided from a server (either from a relational or NoSQL database) and sent to a web client.

The web client then uses JavaScript to convert the JSON into objects and manipulate it as required.

JSON Question

Question: **True or False:** The following JSON is valid.

```
{"employee":1}
```

A) true

B) false

JSON Question (2)

Question: **True or False:** The following JSON is valid.

```
{"array":["a", 1, {"c":2}]}
```

A) true

B) false

JSON Question (3)

Question: **True or False:** The following JSON is valid.

```
{"array":["a", true, FALSE]}
```

A) true

B) false

JSON Question (4)

Question: **True or False:** The following JSON is valid.

```
{ }
```

A) true

B) false

JSON Question (5)

Question: **True or False:** The following JSON is valid.

```
{4, 5, "c":"a"}
```

A) true

B) false

Relational Databases

Relational databases are the dominant form of database and apply to many data management problems.

Relational databases are not the only way to represent data and not the best way for some problems.

Other models:

- Hierarchical model
- Object-oriented
- XML
- Graphs
- Key-value stores
- Document models

Relational Databases Challenges

Some features of relational databases make them "challenging" for certain problems:

- 1) Fixed schemas – The schemas must be defined ahead of time, changes are difficult, and lots of real-world data is "messy".
  - Solution: Get rid of the schemas! Who wants to do that design work anyways? Will you miss them?
- 2) Complicated queries – SQL is declarative and powerful but may be overkill.
  - Solution: Simple query mechanisms and do a lot of work in code.
- 3) Transaction overhead – Not all data and query answers need to be perfect. "Close enough is sometimes good enough".
- 4) Scalability – Relational databases may not scale sufficiently to handle high data and query loads or this scalability comes with a very high cost.

NoSQL

NoSQL databases are useful for several problems not well-suited for relational databases with some typical features:

- Variable data: semi-structured, evolving, or has no schema
- Massive data: terabytes or petabytes of data from new applications (web analysis, sensors, social graphs)
- Parallelism: large data requires architectures to handle massive parallelism, scalability, and reliability
- Simpler queries: may not need full SQL expressiveness
- Relaxed consistency: more tolerant of errors, delays, or inconsistent results ("eventual consistency")
- **Easier/cheaper:** less initial cost to get started

NoSQL is not really about SQL but instead developing data management systems that are not relational.

- NoSQL – "Not Only SQL"

NoSQL Systems

There are a variety of systems that are not relational:

- MapReduce – useful for large scale analysis
- Key-value stores – ideal for retrieving specific data records from a large set of data
- Document stores – similar to key-value stores except value is a document in some form (e.g. JSON)
- Graph databases – represent data as graphs

MapReduce

MapReduce was invented by Google and has an open source implementation called Hadoop.

Data is stored in files. Users provide functions:

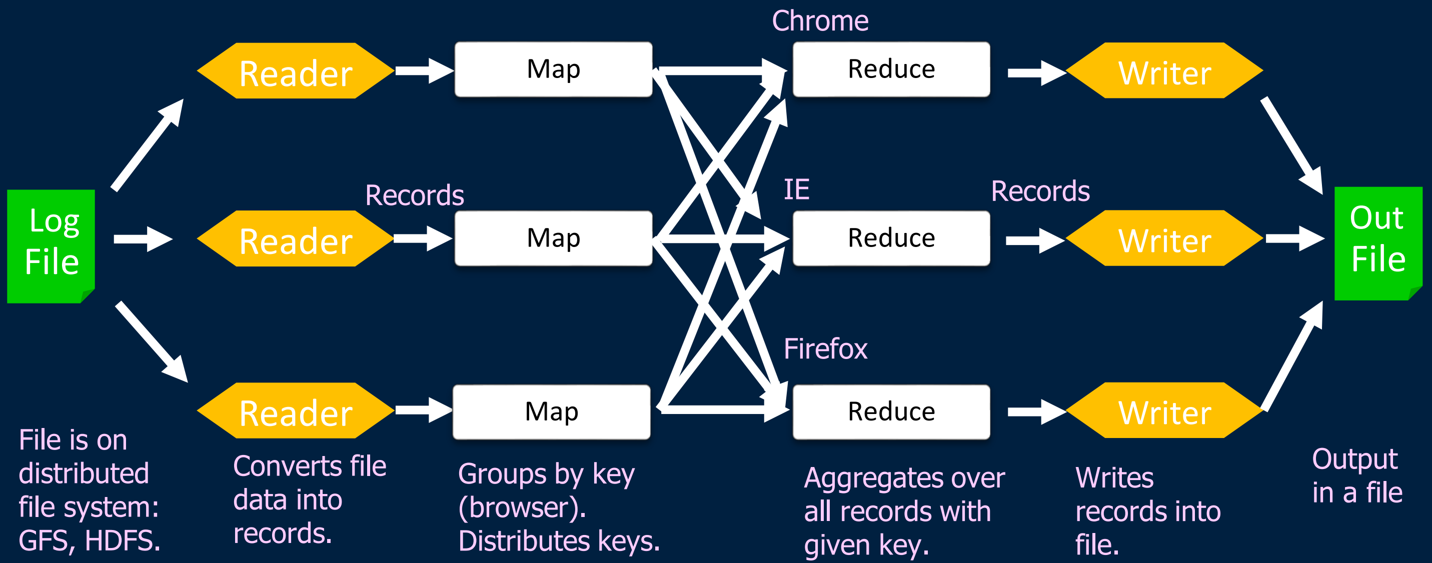
- reader(file) – converts file data into records
- map(records) – converts records into key-value pairs
- combine(key, list of values) – optional aggregation of pairs after map stage
- reduce(key, list of values) – summary on key values to produce output records
- write(file) – writes records to output file

MapReduce (Hadoop) provides infrastructure for tying everything together and distributing work across machines.

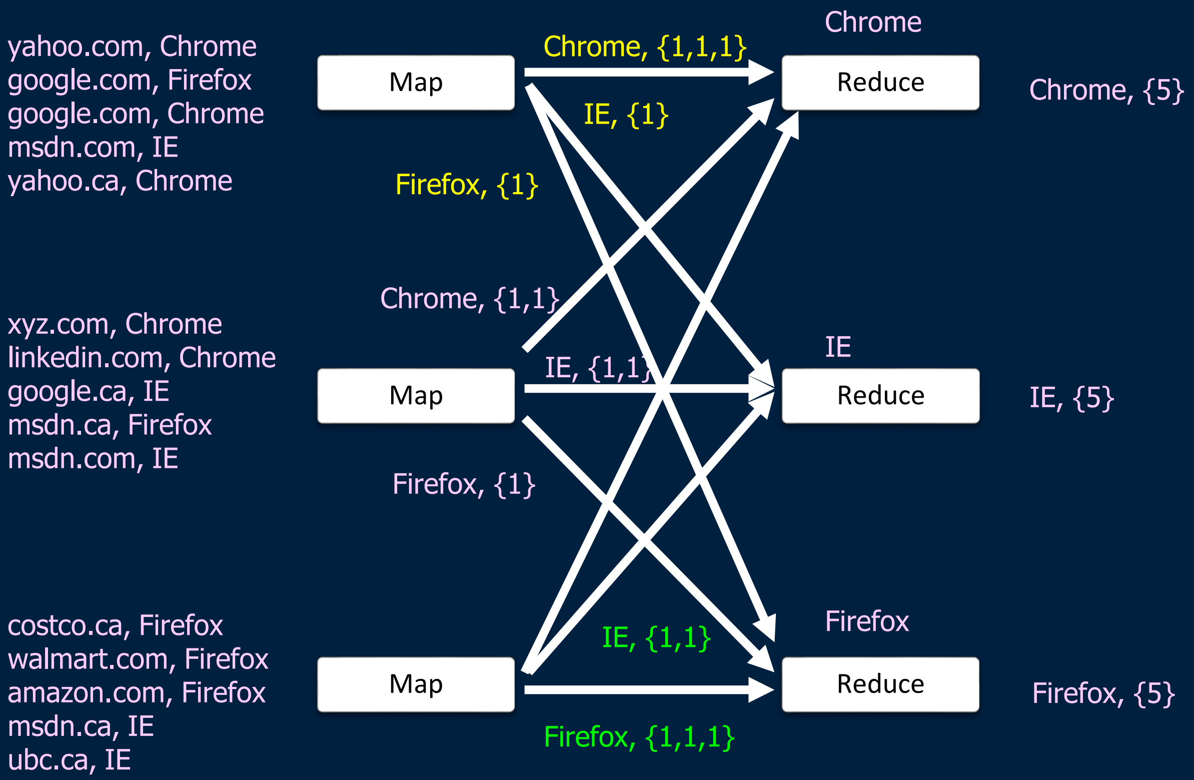
MapReduce Example
Web Data Analysis

Data file records: URL, timestamp, browser

Goal: Determine the most popular browser used.



MapReduce Example (2)
Web Data Analysis



MapReduce Extensions

The key benefit of MapReduce and Hadoop is their scalable performance, not that they do not support SQL. In fact, schemas and declarative SQL have many advantages.

Extensions to Hadoop combine the massive parallel processing with familiar SQL features:

- Hive – an SQL-like language variant
- Pig – similar to relational operators

Data manipulations expressed in these languages are then converted into a MapReduce workflow automatically.

Unlike relational databases, MapReduce systems handle failures during execution and will complete a query even if a server fails.

Key-Value Stores

Key-value stores store and retrieve data using keys. The data values are arbitrary. Designed for "web sized" data sets.

Operations:

- insert(key, value)
- fetch(key)
- update(key)
- delete(key)

Benefits: high-scalability, availability, and performance

Limitations: single record transactions, eventual consistency, simple query interface

Systems: Cassandra, Amazon Dynamo, Google BigTable, HBase

Document Stores

Document stores are similar to key-value stores but the value stored is a structured document (e.g. JSON, XML).

Can store and query documents by key as well as retrieve and filter documents by their properties.

Benefits: high-scalability, availability, and performance

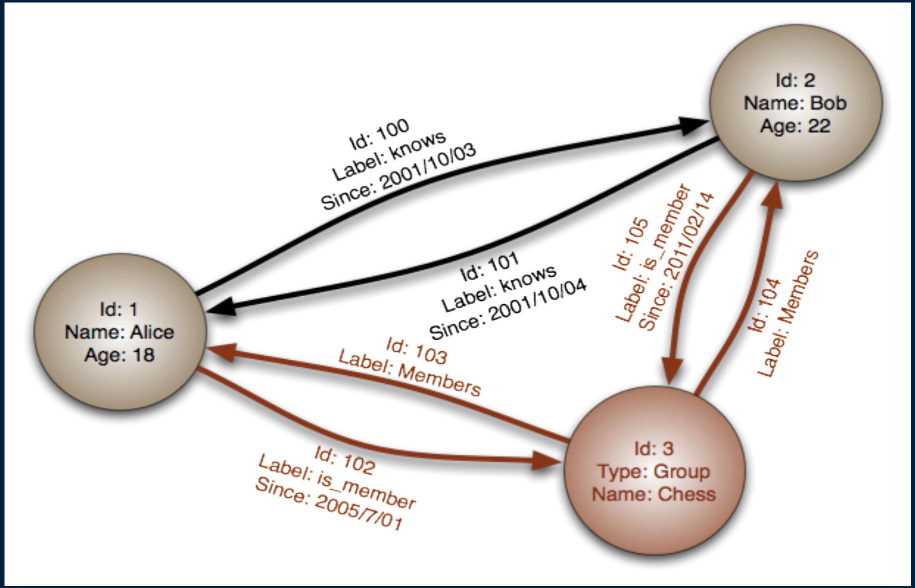
Limitations: same as key-value stores, may cause redundancy and more code to manipulate documents

Systems: MongoDB, CouchDB, SimpleDB

Graph Databases

Graph databases model data as nodes (with properties) and edges (with labels).

- Systems: Neo4J, FlockDB



NoSQL Question

Question: How many of the following statements are ***true***?

1) Neo4J is a document store.

2) Unlike Hadoop, a relational database is designed to restart any failed part of a query when a failure occurs.

3) Key-value stores are similar to a tree data structure.

4) SQL cannot be used to query MongoDB.

5) Relational systems typically scale better than NoSQL systems.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Conclusion

Extensible Markup Language (***XML***) is a markup language that allows for the description of data semantics.

- An XML document does not need a schema to be *well-formed*. An XML document is *valid* if it conforms to its schema (DTD or XML Schema).
- ***XPath*** is a language for specifying paths through XML documents.

JavaScript Object Notation (JSON) serializes data objects into text form.

- Benefits: human-readable, supports semi-structured data, supported by many languages (not just JavaScript)

NoSQL databases ("Not only SQL") is a category of data management systems that do not use the relational model.

- There is a variety of NoSQL systems including: MapReduce systems, Key-value stores, Document stores, and Graph databases.
- NoSQL databases are designed for high performance, availability, and scalability at the compromise of restricted query languages and weaker consistency guarantees.

Objectives

- List some advantages of XML.
- Given an XML document, determine if it is well-formed.
- Given an XML document and a DTD, determine if it is valid.
- Know the symbols (?,*,+) for cardinality constraints in DTDs.
- Compare and contrast ID/IDREFs in DTDs with keys and foreign keys in the relational model.
- List some advantages that XML Schema has over DTDs.
- Explain why and when namespaces are used.
- Given an XML document and query description, write an XPath query to retrieve the appropriate node sequence to answer the query.
- Given an XML document and an XPath expression, list the result of evaluating the expression.

Objectives (2)

- Understand the basic constructs used to encode JSON data.
- Compare JSON representation versus relational model.
- Understand alternative models for representing data besides the relational model.
- List some NoSQL databases and reason about their benefits and issues compared to the relational model for certain problems.
- Explain at a high-level how MapReduce works.


UBC
THE UNIVERSITY OF BRITISH COLUMBIA