ADVANCED DBMS

Unit - 4

JSON & XML Databases

JSON & XML Databases

- JSON – syntax, datatypes, stringify, objects, schema
- XML Databases: XML Data Model – DTD – XML Schema – XML Querying
- Comparison of JSON with XML
- Comparing the usage of JSON and XML; Use JSON with PHP/ Python

JSON

- JSON stands for **JavaScript Object Notation**
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

JSON

JSON Syntax Rules

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

E.G.

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
}
```

JSON

- The JSON format is syntactically identical to the code for creating JavaScript objects.
- Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.
- The JSON syntax is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language.

JSON

JSON Objects

- JSON objects are written inside curly braces.
- Just like in JavaScript, objects can contain multiple name/value pairs:
- {"firstName":"John", "lastName":"Doe"}

JSON

JSON Arrays

- JSON arrays are written inside square brackets.
- Just like in JavaScript, an array can contain objects:
- "employees":[
  {"firstName":"John", "lastName":"Doe"},
  {"firstName":"Anna", "lastName":"Smith"},
  {"firstName":"Peter", "lastName":"Jones"}
  ]

JSON

Converting a JSON Text to a JavaScript Object

- A common use of JSON is to read data from a web server, and display the data in a web page.
- For simplicity, this can be demonstrated using a string as input.
- First, create a JavaScript string containing JSON syntax:
-
```
let text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
```

JSON

Then, use the JavaScript built-in function JSON.parse()
to convert the string into a JavaScript object:

```
const obj = JSON.parse(text);
```

Finally, use the new JavaScript object in your page:

```
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
</script>
```

XML Introduction

- XML stands for eXtensible Markup Language.
- XML was designed to store and transport data.
- XML plays an important role in many different IT systems.
- XML is often used for distributing data over the Internet.
- It is important (for all types of software developers!) to have a good understanding of XML.

XML example

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget about the
meeting</body>
</note>
```

XML Introduction

- The ability to specify new tags, and to create nested tag structures make XML a great way to exchange **data**, not just documents.
  - Much of the use of XML has been in data exchange applications, not as a replacement for HTML
- Tags make data (relatively) self-documenting
  - E.g.,

```
<university>
    <department>
        <dept_name> Comp. Sci. </dept_name>
        <building> Taylor </building>
        <budget> 100000 </budget>
    </department>
    <course>
        <course_id> CS-101 </course_id>
        <title> Intro. to Computer Science </title>
        <dept_name> Comp. Sci </dept_name>
        <credits> 4 </credits>
    </course>
</university>
```

XML: Motivation

- Data interchange is critical in today's networked world
  - Examples:
    - Banking: funds transfer
    - Order processing (especially inter-company orders)
    - Scientific data
      - Chemistry: ChemML, …
      - Genetics: BSML (Bio-Sequence Markup Language), …
  - Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats

XML Motivation (Cont.)

- Earlier generation formats were based on plain text with line headers indicating the meaning of fields
  - Similar in concept to email headers
  - Does not allow for nested structures, no standard “type” language
  - Tied too closely to low level document structure (lines, spaces, etc)
- Each XML based standard defines what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Descriptors)
    - XML Schema
  - Plus textual descriptions of the semantics
- XML allows new tags to be defined as required
  - However, this may be constrained by DTDs
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Comparison with Relational Data

- Inefficient: tags, which in effect represent schema information, are repeated
- Better than relational tuples as a data-exchange format
  - Unlike relational tuples, XML data is self-documenting due to presence of tags
  - Non-rigid format: tags can be added
  - Allows nested structures
  - Wide acceptance, not only in database systems, but also in browsers, tools, and applications

Structure of XML Data

- **Tag**: label for a section of data
- **Element**: section of data beginning with <*tagname*> and ending with matching </*tagname*>
- Elements must be properly nested
  - Proper nesting
    - <course> … <title> …. </title> </course>
  - Improper nesting
    - <course> … <title> …. </course> </title>
  - Formally: every start tag must have a unique matching end tag, that is in the context of the same parent element.
- Every document must have a single top-level element

Example of Nested Elements

```
<purchase_order>
    <identifier> P-101 </identifier>
    <purchaser>  …. </purchaser>
    <itemlist>
        <item>
            <identifier> RS1 </identifier>
            <description> Atom powered rocket sled </description>
            <quantity> 2 </quantity>
            <price> 199.95 </price>
        </item>
        <item>
            <identifier> SG2 </identifier>
            <description> Superb glue </description>
            <quantity> 1 </quantity>
            <unit-of-measure> liter </unit-of-measure>
            <price> 29.95 </price>
        </item>
    </itemlist>
</purchase_order>
```

Motivation for Nesting

- Nesting of data is useful in data transfer
  - Example: elements representing *item* nested within an *itemlist* element
- Nesting is not supported, or discouraged, in relational databases
  - With multiple orders, customer name and address are stored redundantly
  - normalization replaces nested structures in each order by foreign key into table storing customer name and address information
  - Nesting is supported in object-relational databases
- But nesting is appropriate when transferring data
  - External application does not have direct access to data referenced by a foreign key

Structure of XML Data (Cont.)

- Mixture of text with sub-elements is legal in XML.
  - Example:

```
<course>
    This course is being offered for the first time in 2009.
    <course id> BIO-399 </course id>
    <title> Computational Biology </title>
    <dept name> Biology </dept name>
    <credits> 3 </credits>
</course>
```

  - Useful for document markup, but discouraged for data representation

Attributes

- Elements can have **attributes**

```
<course course_id= “CS-101”>
    <title> Intro. to Computer Science</title>
    <dept name> Comp. Sci. </dept name>
    <credits> 4 </credits>
</course>
```

- Attributes are specified by *name*=*value* pairs inside the starting tag of an element
- An element may have several attributes, but each attribute name can only occur once

```
<course  course_id = “CS-101”  credits=“4”>
```

Attributes vs. Subelements

- Distinction between subelement and attribute
  - In the context of documents, attributes are part of markup, while subelement contents are part of the basic document contents
  - In the context of data representation, the difference is unclear and may be confusing
    - Same information can be represented in two ways
      - <course course_id= “CS-101”> … </course>
      - <course>
        <course_id>CS-101</course_id> …
        </course>
  - Suggestion: use attributes for identifiers of elements, and use subelements for contents

Namespaces

- XML data has to be exchanged between organizations
- Same tag name may have different meaning in different organizations, causing confusion on exchanged documents
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name
- Avoid using long unique names all over document by using XML Namespaces

```
<university xmlns:yale="http://www.yale.edu">
        …
        <yale:course>
            <yale:course_id> CS-101 </yale:course_id>
            <yale:title> Intro. to Computer Science</yale:title>
            <yale:dept_name> Comp. Sci. </yale:dept_name>
            <yale:credits> 4 </yale:credits>
        </yale:course>
        …
</university>
```

More on XML Syntax

- Elements without subelements or text content can be abbreviated by ending the start tag with a /> and deleting the end tag
  - <course course_id="CS-101" Title="Intro. To Computer Science"
    dept_name = "Comp. Sci." credits="4" />
- To store string data that may contain tags, without the tags being interpreted as subelements, use CDATA as below
  - <![CDATA[<course> … </course>]]>

  Here, <course> and </course> are treated as just strings

  CDATA stands for "character data"

XML Document Schema

- Database schemas constrain what information can be stored, and the data types of stored values
- XML documents are not required to have an associated schema
- However, schemas are very important for XML data exchange
  - Otherwise, a site cannot automatically interpret data received from another site
- Two mechanisms for specifying XML schema
  - **Document Type Definition (DTD)**
    - Widely used
  - **XML Schema**
    - Newer, increasing use

Document Type Definition (DTD)

- The type of an XML document can be specified using a DTD
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Element Specification in DTD

- Subelements can be specified as
  - names of elements, or
  - #PCDATA (parsed character data), i.e., character strings
  - EMPTY (no subelements) or ANY (anything can be a subelement)
- Example

  <! ELEMENT department (dept_name building, budget)>
  <! ELEMENT dept_name (#PCDATA)>
  <! ELEMENT budget (#PCDATA)>
- Subelement specification may have regular expressions

  <!ELEMENT university ((department | course | instructor | teaches)+)>
  - Notation:
    - “|” - alternatives
    - “+” - 1 or more occurrences
    - “*” - 0 or more occurrences

University DTD

```
<!DOCTYPE  university [
    <!ELEMENT university (
    (department|course|instructor|teaches)+)>
    <!ELEMENT department ( dept name, building, budget)>
    <!ELEMENT course ( course id, title, dept name, credits)>
    <!ELEMENT instructor (IID, name, dept name, salary)>
    <!ELEMENT teaches (IID, course id)>
    <!ELEMENT dept name( #PCDATA )>
    <!ELEMENT building( #PCDATA )>
    <!ELEMENT budget( #PCDATA )>
    <!ELEMENT course id ( #PCDATA )>
    <!ELEMENT title ( #PCDATA )>
    <!ELEMENT credits( #PCDATA )>
    <!ELEMENT IID( #PCDATA )>
    <!ELEMENT name( #PCDATA )>
    <!ELEMENT salary( #PCDATA )>
]>
```

Attribute Specification in DTD

- Attribute specification : for each attribute
  - Name
  - Type of attribute
    - CDATA
    - ID (identifier) or IDREF (ID reference) or IDREFS (multiple IDREFs)
      - more on this later
  - Whether
    - mandatory (#REQUIRED)
    - has a default value (value),
    - or neither (#IMPLIED)
- Examples
  - <!ATTLIST course course_id CDATA #REQUIRED>, or
  - <!ATTLIST course
    course_id ID #REQUIRED
    dept_name IDREF #REQUIRED
    instructors IDREFS #IMPLIED >

IDs and IDREFs

- An element can have at most one attribute of type ID
- The ID attribute value of each element in an XML document must be distinct
  - Thus the ID attribute value is an object identifier
- An attribute of type IDREF must contain the ID value of an element in the same document
- An attribute of type IDREFS contains a set of (0 or more) ID values. Each ID value must contain the ID value of an element in the same document

University DTD with Attributes

- University DTD with ID and IDREF attribute types.

```
<!DOCTYPE university-3 [
    <!ELEMENT university ( (department|course|instructor)+)>
    <!ELEMENT department ( building, budget )>
    <!ATTLIST department
        dept_name ID #REQUIRED >
    <!ELEMENT course (title, credits )>
    <!ATTLIST course
        course_id ID #REQUIRED
        dept_name IDREF #REQUIRED
        instructors IDREFS #IMPLIED >
    <!ELEMENT instructor ( name, salary )>
    <!ATTLIST instructor
        IID ID #REQUIRED
        dept_name IDREF #REQUIRED >
    . . . declarations for title, credits, building,
        budget, name and salary . . .
]>
```

XML data with ID and IDREF attributes

```
<university-3>
    <department dept name="Comp. Sci.">
        <building> Taylor </building>
        <budget> 100000 </budget>
    </department>
    <department dept name="Biology">
        <building> Watson </building>
        <budget> 90000 </budget>
    </department>
    <course course id="CS-101" dept name="Comp. Sci"
                instructors="10101 83821">
        <title> Intro. to Computer Science </title>
        <credits> 4 </credits>
    </course>
    ….
    <instructor IID="10101" dept name="Comp. Sci.">
        <name> Srinivasan </name>
        <salary> 65000 </salary>
    </instructor>
    ….
</university-3>
```

Limitations of DTDs

- No typing of text elements and attributes
  - All values are strings, no integers, reals, etc.
- Difficult to specify unordered sets of subelements
  - Order is usually irrelevant in databases (unlike in the document-layout environment from which XML evolved)
  - (A | B)* allows specification of an unordered set, but
    - Cannot ensure that each of A and B occurs only once
- IDs and IDREFs are untyped
  - The *instructors* attribute of an course may contain a reference to another course, which is meaningless
    - *instructors* attribute should ideally be constrained to refer to instructor elements

XML Schema

- XML Schema is a more sophisticated schema language which addresses the drawbacks of DTDs. Supports
  - Typing of values
    - E.g., integer, string, etc
    - Also, constraints on min/max values
  - User-defined, comlex types
  - Many more features, including
    - uniqueness and foreign key constraints, inheritance
- XML Schema is itself specified in XML syntax, unlike DTDs
  - More-standard representation, but verbose
- XML Scheme is integrated with namespaces
- BUT: XML Schema is significantly more complicated than DTDs.

XML Schema Version of Univ. DTD

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="university" type="universityType" />
<xs:element name="department">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="dept name" type="xs:string"/>
            <xs:element name="building" type="xs:string"/>
            <xs:element name="budget" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
....
<xs:element name="instructor">
   <xs:complexType>
       <xs:sequence>
           <xs:element name="IID" type="xs:string"/>
           <xs:element name="name" type="xs:string"/>
           <xs:element name="dept name" type="xs:string"/>
           <xs:element name="salary" type="xs:decimal"/>
       </xs:sequence>
   </xs:complexType>
</xs:element>
```

… Contd.

XML Schema Version of Univ. DTD (Cont.)

```
….
<xs:complexType name=“UniversityType”>
   <xs:sequence>
      <xs:element ref=“department” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“course” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“instructor” minOccurs=“0” maxOccurs=“unbounded”/>
      <xs:element ref=“teaches” minOccurs=“0” maxOccurs=“unbounded”/>
   </xs:sequence>
</xs:complexType>
</xs:schema>
```

- Choice of “xs:” was ours -- any other namespace prefix could be chosen
- Element “university” has type “universityType”, which is defined separately
  - xs:complexType is used later to create the named complex type “UniversityType”

More features of XML Schema

- Attributes specified by xs:attribute tag:
  - <xs:attribute name = “dept_name”/>
  - adding the attribute use = “required” means value must be specified
- Key constraint: “department names form a key for department elements under the root university element:

```
<xs:key name = “deptKey”>
        <xs:selector xpath = “/university/department”/>
        <xs:field xpath = “dept_name”/>
<\xs:key>
```

- Foreign key constraint from course to department:

```
<xs:keyref name = “courseDeptFKey” refer=“deptKey”>
        <xs:selector xpath = “/university/course”/>
        <xs:field xpath = “dept_name”/>
<\xs:keyref>
```

Querying and Transforming XML Data

- Translation of information from one XML schema to another
- Querying on XML data
- Above two are closely related, and handled by the same tools
- Standard XML querying/translation languages
  - XPath
    - Simple language consisting of path expressions
  - XSLT
    - Simple language designed for translation from XML to XML and XML to HTML
  - XQuery
    - An XML query language with a rich set of features

XPath

- XPath is used to address (select) parts of documents using **path expressions**
- A path expression is a sequence of steps separated by "/"
  - Think of file names in a directory hierarchy
- Result of path expression: set of values that along with their containing elements/attributes match the specified path
- E.g., /university-3/instructor/name evaluated on the university-3 data we saw earlier returns

  <name>Srinivasan</name>
  <name>Brandt</name>
- E.g., /university-3/instructor/name/text( )
  returns the same names, but without the enclosing tags

XPath (Cont.)

- The initial “/” denotes root of the document (above the top-level tag)
- Path expressions are evaluated left to right
  - Each step operates on the set of instances produced by the previous step
- Selection predicates may follow any step in a path, in [ ]
  - E.g., /university-3/course[credits >= 4]
    - returns account elements with a balance value greater than 400
    - /university-3/course[credits] returns account elements containing a credits subelement
- Attributes are accessed using “@”
  - E.g., /university-3/course[credits >= 4]/@course_id
    - returns the course identifiers of courses with credits >= 4

XQuery

- XQuery is a general purpose query language for XML data
- XQuery is derived from the Quilt query language, which itself borrows from SQL, XQL and XML-QL
- XQuery uses a
  for … let … where … order by …result …
  syntax
  for ⇔ SQL **from**
  where ⇔ SQL **where**
  order by ⇔ SQL **order by**
  result ⇔ SQL **select**
  let allows temporary variables, and has no equivalent in SQL

XSLT

- A **stylesheet** stores formatting options for a document, usually separately from document
  - E.g. an HTML style sheet may specify font colors and sizes for headings, etc.
- The **XML Stylesheet Language (XSL)** was originally designed for generating HTML from XML
- XSLT is a general-purpose transformation language
  - Can translate XML to XML, and XML to HTML
- XSLT transformations are expressed using rules called **templates**
  - Templates combine selection using XPath with construction of results