



## Lecture 22 of 42

### Servlets and JSP Discussion: Online DBs

Monday, 16 October 2006

William H. Hsu  
Department of Computing and Information Sciences, KSU

KSOL course page: <http://snipurl.com/va60>

Course web site: <http://www.kddresearch.org/Courses/Fall-2006/CIS560>

Instructor home page: <http://www.cis.ksu.edu/~bhsu>

#### Reading for Next Class:

Second half of Chapter 8, Silberschatz *et al.*, 5<sup>th</sup> edition



## Chapter 8: Application Design and Development

- User Interfaces and Tools
- Web Interfaces to Databases
- Web Fundamentals
- Servlets and JSP
- Building Large Web Applications
- Triggers
- Authorization in SQL
- Application Security





## Application Security

- Data may be *encrypted* when database authorization provisions do not offer sufficient protection.
- Properties of good encryption technique:
  - \* Relatively simple for authorized users to encrypt and decrypt data.
  - \* Encryption scheme depends not on the secrecy of the algorithm but on the secrecy of a parameter of the algorithm called the encryption key.
  - \* Extremely difficult for an intruder to determine the encryption key.



## Encryption (Cont.)

- *Data Encryption Standard* (DES) substitutes characters and rearranges their order on the basis of an encryption key which is provided to authorized users via a secure mechanism. Scheme is no more secure than the key transmission mechanism since the key has to be shared.
- Advanced Encryption Standard (AES) is a new standard replacing DES, and is based on the Rijndael algorithm, but is also dependent on shared secret keys
- *Public-key encryption* is based on each user having two keys:
  - \* *public key* – publicly published key used to encrypt data, but cannot be used to decrypt data
  - \* *private key* -- key known only to individual user, and used to decrypt data. Need not be transmitted to the site doing encryption.Encryption scheme is such that it is impossible or extremely hard to decrypt data given only the public key.
- The RSA public-key encryption scheme is based on the hardness of factoring a very large number (100's of digits) into its prime components.





## Authentication

- Password based authentication is widely used, but is susceptible to sniffing on a network
- **Challenge-response** systems avoid transmission of passwords
  - \* DB sends a (randomly generated) challenge string to user
  - \* User encrypts string and returns result.
  - \* DB verifies identity by decrypting result
  - \* Can use public-key encryption system by DB sending a message encrypted using user's public key, and user decrypting and sending the message back
- **Digital signatures** are used to verify authenticity of data
  - \* E.g. use private key (in reverse) to encrypt data, and anyone can verify authenticity by using public key (in reverse) to decrypt data. Only holder of private key could have created the encrypted data.
  - \* Digital signatures also help ensure **nonrepudiation**: sender cannot later claim to have not created the data



## Digital Certificates

- **Digital certificates** are used to verify authenticity of public keys.
- Problem: when you communicate with a web site, how do you know if you are talking with the genuine web site or an imposter?
  - \* Solution: use the public key of the web site
  - \* Problem: how to verify if the public key itself is genuine?
- Solution:
  - \* Every client (e.g. browser) has public keys of a few root-level **certification authorities**
  - \* A site can get its name/URL and public key signed by a certification authority: signed document is called a **certificate**
  - \* Client can use public key of certification authority to verify certificate
  - \* Multiple levels of certification authorities can exist. Each certification authority
    - ⇒ presents its own public-key certificate signed by a higher level authority, and
    - ⇒ Uses its private key to sign the certificate of other web sites/authorities





## XML

- Structure of XML Data
- XML Document Schema
- Querying and Transformation
- Application Program Interfaces to XML
- Storage of XML Data
- XML Applications



## Introduction

- XML: Extensible Markup Language
- Defined by the WWW Consortium (W3C)
- Derived from SGML (Standard Generalized Markup Language), but simpler to use than SGML
- Documents have tags giving extra information about sections of the document
  - \* E.g. `<title> XML </title>` `<slide> Introduction ...</slide>`
- **Extensible**, unlike HTML
  - \* Users can add new tags, and *separately* specify how the tag should be handled for display





## XML Introduction (Cont.)

- 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.

```
<bank>
  <account>
    <account_number> A-101 </account_number>
    <branch_name> Downtown </branch_name>
    <balance> 500 </balance>
  </account>
  <depositor>
    <account_number> A-101 </account_number>
    <customer_name> Johnson </customer_name>
  </depositor>
</bank>
```



## 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  
⇒ `<account> ... <balance> .... </balance> </account>`
  - \* Improper nesting  
⇒ `<account> ... <balance> .... </account> </balance>`
  - \* 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

```
<bank-1>
  <customer>
    <customer_name> Hayes </customer_name>
    <customer_street> Main </customer_street>
    <customer_city> Harrison </customer_city>
    <account>
      <account_number> A-102 </account_number>
      <branch_name> Perryridge </branch_name>
      <balance> 400 </balance>
    </account>
    <account>
      ...
    </account>
  </customer>
  .
</bank-1>
```





## Motivation for Nesting

- Nesting of data is useful in data transfer
  - \* Example: elements representing *customer\_id*, *customer\_name*, and address nested within an *order* 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:

```
<account>
  This account is seldom used any more.
  <account_number> A-102</account_number>
  <branch_name> Perryridge</branch_name>
  <balance>400 </balance>
</account>
```
  - \* Useful for document markup, but discouraged for data representation





## Attributes

- Elements can have **attributes**

```
<account acct-type = "checking" >
  <account_number> A-102 </account_number>
  <branch_name> Perryridge </branch_name>
  <balance> 400 </balance>
</account>
```
- 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

```
<account acct-type = "checking" monthly-fee="5">
```



## 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
      - ◆ `<account account_number = "A-101"> .... </account>`
      - ◆ `<account>`  
`<account_number>A-101</account_number> ...`  
`</account>`
  - \* 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

```
<bank xmlns:FB='http://www.FirstBank.com'>
  ...
  <FB:branch>
    <FB:branchname>Downtown</FB:branchname>
    <FB:branchcity> Brooklyn </FB:branchcity>
  </FB:branch>
  ...
</bank>
```



## 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
  - \* `<account number="A-101" branch="Perryridge" balance="200 />`
- To store string data that may contain tags, without the tags being interpreted as subelements, use CDATA as below

```
* <![CDATA[<account> ... </account>]]>
```

Here, `<account>` and `</account>` 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

