Database Application Design and Development

- Virtually all real-world user interaction with databases is indirect — it is mediated through an application
- A database application effectively adds additional layers over the database
  - User interface — presentation of data, interaction
  - Business logic — rules, control flow, restrictions, validation/verification of data
- In the end, a database application is like any other application — it just has persistent storage underneath

What You Should Know by Now

- You know how to specify, document, and design a relational database (E-R/UML, object-relational mapping, functional dependencies, normal forms)
- You know how to add/modify/delete and, most of all, retrieve information from that database (relational algebra, SQL)
- You know how to communicate with that database programmatically (JDBC) using a recommended design pattern (DAO)
Database Application Issues

• Databases are most useful when shared — issues and additional knowledge requirements ensue
  ◊ Network structure and protocols
  ◊ Security concerns
  ◊ Concurrency and transaction management
  ◊ Resource management — databases, network bandwidth, and server capacity are all finite
• Very prevalent “special case” — Web applications

• So, in order to write a decent end-user application for your database, you also need to know:
  ◊ How to write a decent non-database end-user application to begin with
    • Standalone application: Java/Swing, others
    • Web application: XHTML, CSS, JavaScript, CGI/scripting, Web containers (JSP, ASP, etc.)
    • Other useful technologies: XML, computer graphics creation and programming (paint programs, Java2D, maybe OpenGL), application packaging/deployment (standalone downloads, Java WebStart)
  ◊ How different processes running potentially on different machines can interact and communicate
• Then, “just add database”
  ◊ What we’ve said before, plus additional tools that may make the database part easier (BeanUtils, Hibernate)
Thick Client, Plain Server

- No Web yet, just the user’s machine (“client”) and the database server
- Unit tests run in this environment — “client machine” is actually the “test machine”

+ Fairly simple, not so many pieces
  - Database server is exposed directly; needs to be able to scale to however many simultaneous clients are expected for the application
  - Each client has a copy of the database access code
  - Fading paradigm: doesn’t easily accommodate Web layer (you may as well write a whole new program)
Web-Only Application

- Follows Web paradigm exactly: http[s] protocol, Web pages for responses, PUTs/POSTs/URLs for requests
- “Web application container” ranges from no container at all — standalone CGI scripts — to full-blown environments (PHP, servlets, JSP)

+ The archetypal “thin client” — the user only needs a Web browser; no need to download anything else
+ Majority of code is in the Web application container
+/- Primary bottleneck is now the Web server
— Web browser metaphor is typically stateless; require ways to maintain state (cookies, JavaScript, Web application container features)
“Enterprise-Wide” Application Server

- Centralized application server abstracts out the database, encapsulates business logic
- Web and standalone applications that communicate with this server are distilled as pure presentation/interaction mechanisms
  - Provides maximum flexibility, very clear separation of tasks, potentially efficient distribution of work
  - Once setup, can lead to rapid application development due to clean separation of concerns
- Lots of parts to coordinate
  - Tools are emerging to help automate the development of these types of applications
- May vertically lock you into specific technology trains (J2EE, .NET, WebObjects)
Java-Centric Enterprise Architecture Arsenal

- **JBoss**: open source application server, based on J2EE/EJB standards; includes *Tomcat* Web application container (reference implementation for servlet/JSP standards)
- **Hibernate**: object-relational mapper, significantly decreases the need for direct JDBC/SQL; uses XML to define how relations map to classes and vice versa
- **xDoclet**: Java source code preprocessor; allows automated generation of source code (particularly useful for EJB and Hibernate)
- **Struts**: Web application framework; layer over JSP

- **Ant**: build automation system; for this type of application, build automation is an absolute necessity
- **CruiseControl**: continuous-integration tool; enables unattended, automated building and testing