Database Application Development

CPS352: Database Systems

Simon Miner
Gordon College
Last Revised: 10/4/12

Agenda

- Check-in
- Application UI and the World Wide Web
- Database Access from Applications
- Design Project E-R Diagram Presentations
- Application Architecture
- Database Design Tips
- Exam 1

Check-in

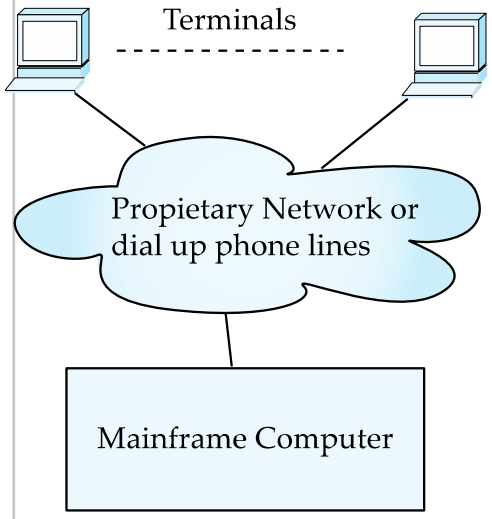
Application UI and the World Wide Web

Application Programs and User Interfaces (UI)

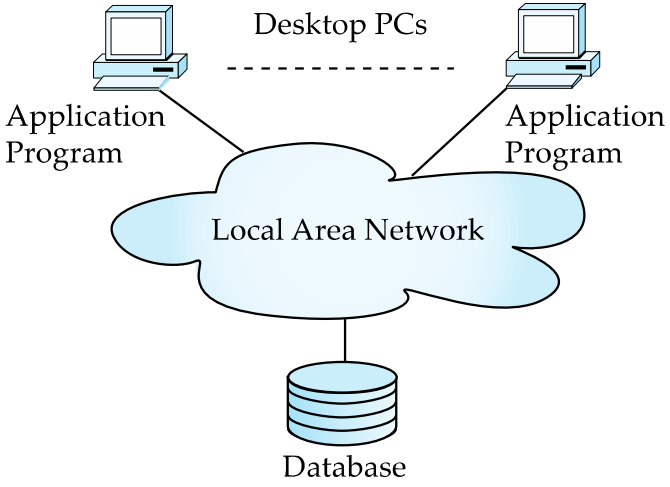
- Most database users do *not* use a query language like SQL
- An application program acts as the intermediary between users and the database
  - Applications split into
    - front-end
    - middle layer
    - backend
- Front-end: user interface
  - Forms
  - Graphical user interfaces
  - Many interfaces are Web-based

Application Architecture Evolution

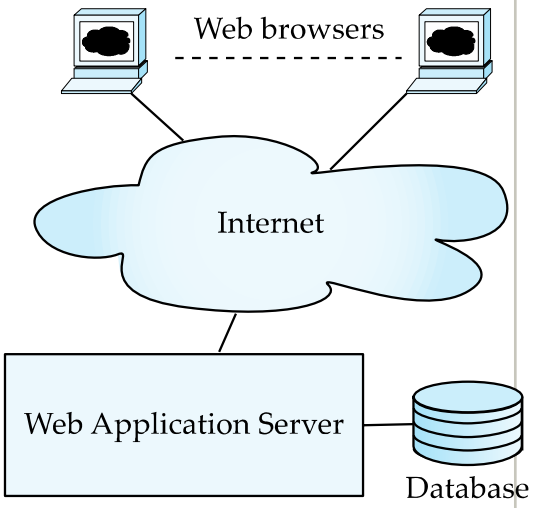
- Three distinct eras of application architecture
  - mainframe (1960's and 70's)
  - personal computer era (1980's)
  - Web era (1990's onwards)



(a) Mainframe Era



(b) Personal Computer Era



(c) Web era

Thick and Thin Clients

- Thick clients
  - Much of the work is done on the client, reducing server load
  - Requires (complex) business logic software to be downloaded and installed on the client machine
  - Trickier to update and secure business software
- Thin client
  - Most work is done on the server, minimizing the need for specialized client software
  - Updates and security are mostly handled on the server
  - Business logic can reside in:
    - Application programs or server-side scripts
    - Database server itself – stored procedures

Web Interface

- Web browsers have become the de-facto standard (thin client) user interface to databases
  - Enable large numbers of users to access databases from anywhere
  - Avoid the need for downloading/installing specialized code, while providing a good graphical user interface
    - Javascript, Flash and other scripting languages run in browser, but are downloaded transparently
  - Examples: banks, airline and rental car reservations, university course registration and grading, an so on.

The World Wide Web

- The Web is a distributed information system based on hypertext.
- Most Web documents are hypertext documents formatted via the HyperText Markup Language (HTML)
- HTML documents contain
  - text along with semantic and layout specifications
  - hypertext links to other documents, which can be associated with regions of the text.
  - forms, enabling users to enter data which can then be sent back to the Web server

Uniform Resource Locators (URLs)

- In the Web, functionality of pointers is provided by Uniform Resource Locators (URLs).
- URL example: http://www.acm.org/sigmod
  - The first part indicates how the document is to be accessed
    - "http" indicates that the document is to be accessed using the Hyper Text Transfer Protocol.
  - The second part gives the unique name of a machine on the Internet.
  - The rest of the URL identifies the document within the machine.
- The local identification can be:
  - The path name of a file on the machine, or
  - An identifier (path name) of a program, plus arguments to be passed to the program
    - e.g., http://www.google.com/search?q=silberschatz

HTML and HTTP

- HTML provides layout, semantic, hypertext linking, and image display features
  - including tables, stylesheets (to alter formatting), etc.
- HTML also provides input features
  - Select from a set of options
    - Pop-up menus, radio buttons, check boxes
  - Enter values
    - Text boxes and text areas
  - Filled in input sent back to the server, to be acted upon by an executable at the server
    - Buttons and images
- HyperText Transfer Protocol (HTTP) used for communication with the Web server

Sample HTML Source Text

```
<html>

<body>

  <table border>
     <tr> <th>ID</th> <th>Name</th> <th>Department</th> </tr>
     <tr> <td>00128</td> <td>Zhang</td> <td>Comp. Sci.</td> </tr>
     ….

  </table>

   <form action="PersonQuery" method=get>
     Search for:
        <select name="persontype">
            <option value="student" selected>Student </option>
            <option value="instructor"> Instructor </option>
        </select> <br>
     Name: <input type=text size=20 name="name">
     <input type=submit value="submit">

  </form>

</body> </html>
```

Display of Sample HTML Source

ID	Name	Department
00128	Zhang	Comp. Sci.
12345	Shankar	Comp. Sci.
19991	Brandt	History

Search for: Student

Name:

submit

Web Servers

- A Web server can serve as an intermediary to provide access to a variety of information services
  - i.e. files, databases, other web servers (via APIs), etc.
- The document name (path) in a URL may identify an executable program, that, when run, generates a HTML document.
  - When an HTTP server receives a request for such a document, it executes the program, and sends back the HTML document that is generated.
  - The Web client can pass extra arguments with the name of the document.
- To install a new service on the web server, one needs to create and install an executable that provides that service.
  - The web browser provides a graphical user interface to the information service.
- Common Gateway Interface (CGI): a standard interface between web and application server

HTTP and Sessions

- The HTTP protocol is **stateless**
  - That is, once the server replies to a request, the server closes the connection with the client, and forgets all about the request
  - In contrast, Unix logins, and database connections stay connected until the client disconnects
    - retaining user authentication and other information
  - Motivation: reduces load on server
    - operating systems have tight limits on number of open connections on a machine
- Information services need session information
  - e.g., user authentication should be done only once per session
- Solution: use a **cookie**
  - Or some other state-preserving mechanism (i.e. embedding state in the URL)

Sessions and Cookies

- A **cookie** is a small piece of text containing identifying information
  - Sent by server to browser
    - Sent on first interaction, to identify session
  - Sent by browser to the server that created the cookie on further interactions
    - part of the HTTP protocol
  - Server saves information about cookies it issued, and can use it when serving a request
    - e.g., authentication information, and user preferences
- Cookies can be stored permanently or for a limited time on the browser

Java Servlets

- Java Servlet specification defines an API for communication between the Web/application server and application program running in the server
  - e.g., methods to get parameter values from Web forms, and to send HTML text back to client
- Application program (also called a servlet) is loaded into the server
  - Each request spawns a new thread in the server
    - thread is closed once the request is serviced

Example Servlet Code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class PersonQueryServlet extends HttpServlet {

   public void doGet (HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<HEAD><TITLE> Query Result</TITLE></HEAD>");
      out.println("<BODY>");

   ….. BODY OF SERVLET (next slide) …

      out.println("</BODY>");
      out.close();
   }

}
```

Example Servlet Code (Continued)

```
String persontype = request.getParameter("persontype");
String number = request.getParameter("name");

if(persontype.equals("student")) {

     .. code to find students with the specified name …
    ... using JDBC to communicate with the database ..

    out.println("<table BORDER COLS=3>");
    out.println(" <tr> <td>ID</td> <td>Name: </td>" + " <td>Department</td> </tr>");

    for(... each result ...){

        ... retrieve ID, name and dept name
        ... into variables ID, name and deptname

        out.println("<tr> <td>" + ID + "</td>" + "<td>" + name + "</td>" + "<td>" +
deptname
            + "</td></tr>");
    }
    out.println("</table>");
} else {

    ... as above, but for instructors …

}
```

Server-Side Scripting

- Server-side scripting simplifies the task of connecting a database to the Web
  - Define an HTML document with embedded executable code/SQL queries.
  - Input values from HTML forms can be used directly in the embedded code/SQL queries.
  - When the document is requested, the Web server executes the embedded code/SQL queries to generate the actual HTML document.
- Numerous server-side scripting languages
  - JSP, PHP
  - General purpose scripting languages: VBScript, Perl, Python

Java Server Pages (JSP)

- A JSP page with embedded Java code

```
<html>
<head> <title> Hello </title> </head>
<body>
```

-

```
<% if (request.getParameter( “name” ) == null)
{ out.println( “Hello World” ); }
else { out.println( “Hello, “ + request.getParameter( “name” )); }
%>

</body>
</html>
```

- JSP is compiled into Java + Servlets
- JSP allows new tags to be defined, in tag libraries
  - such tags are like library functions, can are used for example to build rich user interfaces such as paginated display of large datasets

PHP

- PHP is widely used for Web server scripting
- Extensive libraries including for database access using ODBC

```
<html>
    <head> <title> Hello </title> </head>
    <body>

    <?php if (!isset($_REQUEST[ ‘name’ ]))
    { echo “Hello World”; }
    else { echo “Hello, ” + $_REQUEST[ ‘name’ ]; }
    ?>

    </body>
</html>
```

Client Side Scripting

- Browsers can fetch certain scripts (client-side scripts) or programs along with documents, and execute them in "safe mode" at the client site
  - Javascript
  - Adobe Flash
  - Java Applets
- Client-side scripts/programs allow documents to be active
  - e.g., animation by executing programs at the local site
  - e.g., ensure that values entered by users satisfy some correctness checks
  - Permit flexible interaction with the user.
    - Executing programs at the client site speeds up interaction by avoiding many round trips to server

Javascript

- Javascript very widely used
  - forms basis of new generation of Web applications (called Web 2.0 applications) offering rich user interfaces
- Javascript functions can
  - check input for validity
  - modify the displayed Web page, by altering the underling **document object model (DOM)** tree representation of the displayed HTML text
  - communicate with a Web server to fetch data and modify the current page using fetched data, without needing to reload/refresh the page
    - forms basis of AJAX technology used widely in Web 2.0 applications
    - e.g. on selecting a country in a drop-down menu, the list of states in that country is automatically populated in a linked drop-down menu

Javascript Example

- Example of Javascript used to validate form input

```
<html> <head>
   <script type="text/javascript">
      function validate() {
         var credits=document.getElementById("credits").value;
         if (isNaN(credits)|| credits<=0 || credits>=16) {
            alert("Credits must be a number greater than 0 and less than 16");
            return false
         }
      }
   </script>
</head> <body>
   <form action="createCourse" onsubmit="return validate()">
      Title: <input type="text" id="title" size="20"><br />
      Credits: <input type="text" id="credits" size="2"><br />
      <Input type="submit" value="Submit">
   </form>
</body> </html>
```

Web Interfaces to Databases

- Dynamic generation of documents
  - Limitations of static HTML documents
    - Cannot customize fixed Web documents for individual users.
    - Problematic to update Web documents, especially if multiple Web documents replicate data.
  - Solution: Generate Web documents dynamically from data stored in a database.
    - Can tailor the display based on user information stored in the database.
      - e.g., customized ads, local news weather, …
    - Displayed information is up-to-date, unlike the static Web pages
      - e.g., Lane menus, stock market information, ..

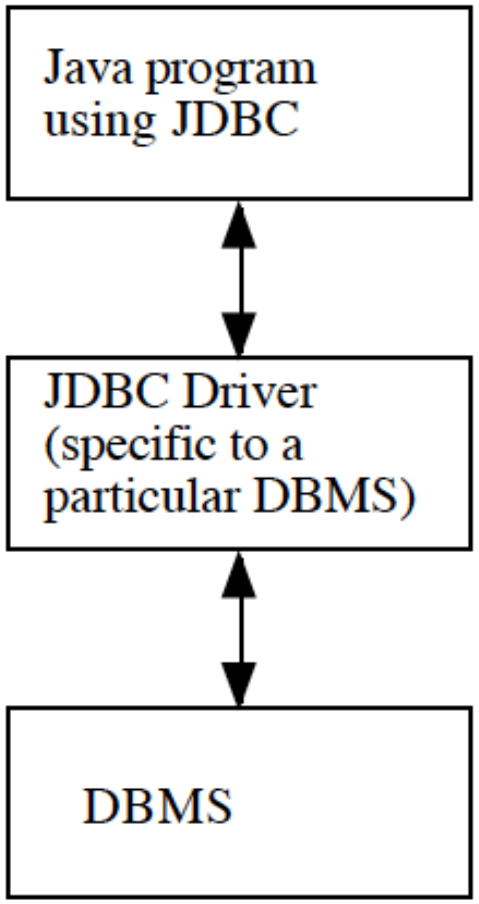
Database Access from Applications

Database Access Methods

- JDBC Style (a.k.a. “dynamic” SQL)
- Embedded (“static”) SQL
- Object-relational mapping (ORM)

JDBC Style

- JDBC – Java Database Connectivity
  - Communicates with various databases (i.e. Oracle, MySQL, DB2) via vendor-specific drivers
  - Provides common API to execute SQL commands and process their output
- Other languages have similar features
  - ODBC
  - Perl DBI
  - PEAR DB in PHP



JDBC Example

```
/**  Drop a specified student from a specified course
  *
  *  @param courseID id of the course
  *  @param studentName the name of the student
  *  @exception IllegalArgumentException if the specified student cannot
  *   be dropped from the specified course - message explains why
  */
 public void doDrop(String courseID, String studentName)
         throws IllegalArgumentException
 {
     try
     {
         int rows = statement.executeUpdate(
             "DELETE FROM ENROLLED_IN " +
               "WHERE ID = '" + courseID + "' AND NAME = '" + studentName + "'");
         if (rows == 0)
             throw new IllegalArgumentException(
                 "Student is not enrolled in course");
     }
     catch (SQLException e)
     {
         System.err.println(e.getMessage());
         System.err.println("SQL Error " + e.getSQLState());
         e.printStackTrace();
     }
 }
```

Parameterized Queries

- A better way to pass variables to SQL
  - More efficient – only compiles SQL statement once
  - More accurate – no need to worry about special database characters
    - i.e. Literal string delimiter (‘) – student_name = O’Reilly’
  - More secure – prevent SQL injection
- Also referred to as bind variables
  - Use “?” or other placeholder for variables in SQL
  - Statement is compiled before it is executed – can be resused
  - Pass actual variable values to SQL statement

JDBC Example with Parameterized Query

```
// Assume there is an instance variable declared as follows:
PreparedStatement dropStatement;
// The following code needs to be executed just once
dropStatement = connection.prepareStatement(
     "DELETE FROM ENROLLED_IN WHERE ID = ? AND NAME = ?");
// The doDrop() procedure now becomes as follows:
public void doDrop(String courseID, String studentName)
          throws IllegalArgumentException
  {
      try
      {
         dropStatement.setString(1, courseID);
         dropStatement.setString(2, studentName);
         int rows = dropStatement.executeUpdate();
         if (rows == 0)
              throw new IllegalArgumentException(
                  "Student is not enrolled in course");
      }
      catch (SQLException e)
      {
          System.err.println(e.getMessage());
          System.err.println("SQL Error " + e.getSQLState());
          e.printStackTrace();
      }
  }
```

SQL Injection

- What would happen if a user specified the following values to the initial (non-parameterized) version of the query?

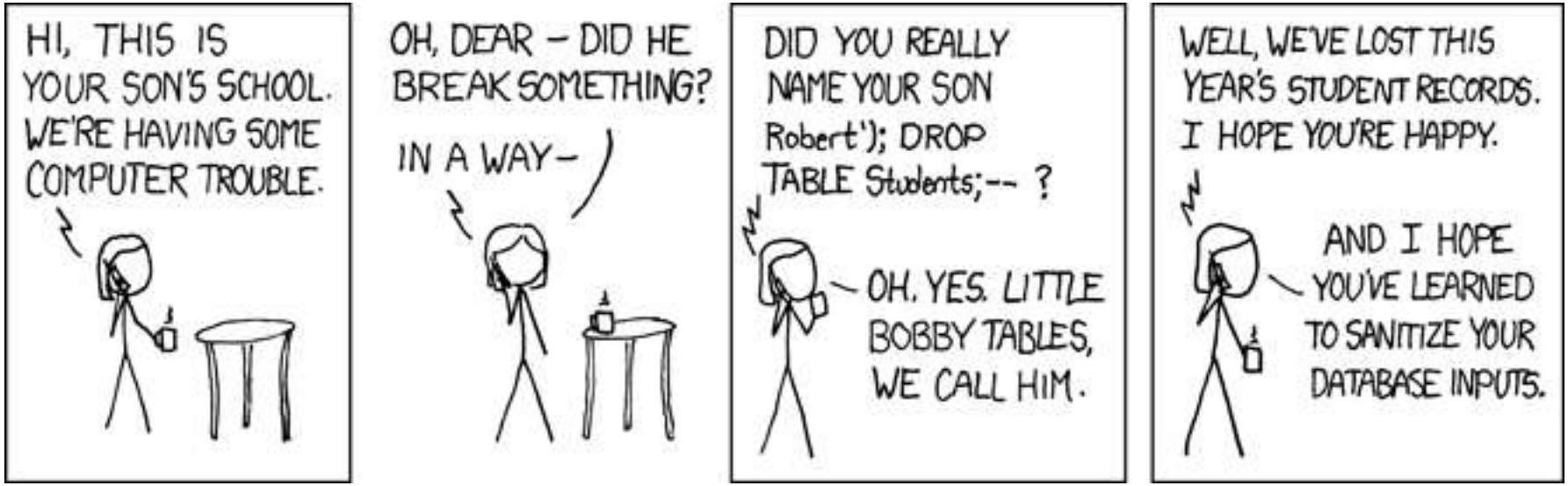
  courseID = “CPS352”
  studentName = “Aardvark’ OR ‘a’ = ‘a”

- Changes the scope of the statement

  ```
  DELETE FROM ENROLLED_IN
     WHERE ID = ‘CPS352’
     AND NAME = ‘Aardvark’ OR ‘a’ = ‘a’;
  ```

  - Can be used to steal or destroy data

Bobby Tables



Embedded SQL

- SQL(ish) statements are placed directly in the code of a host language
  - DB2 supports this for Java (SQLJ), C/C++, Perl ,etc.
  - SQL bracketed in code (i.e. #sql{ … } for SQLJ)
  - Host variables allow programs to pass variables to the database via the SQL statement, or vice versa
    - Typically preceded by a colon in the SQL (i.e. :categoryName)
    - SQL statement processed when it is encountered in program (even in conditionals or loops)
- Pre-processor program
  - Takes a file containing a mixture of source code and SQL (.sqlj file)
  - Emits (at least) two things)
    - A program in the host language which can be compiled (.java)
    - A SQL module (compiled) which gets bound to the underlying DBMS

SQLJ Example

```
/** Get information on an existing category about to be edited or deleted
 *
 *  @param categoryName the name of the category
 *  @return values recorded in the database for this category - an array
 *          of strings.
 *
 *  @exception an ErrorMessage is thrown with an appropriate message if
 *          the category does not exist
 */
public String[] getCategoryInformation(String categoryName) throws ErrorMessage
{
   String [] values = new String[3];
   values[0] = categoryName;
   int checkoutPeriod, maxBooksOut;
   try
   {
      #sql { select checkout_period, max_books_out
                 into :checkoutPeriod, :maxBooksOut
                 from category
                 where category_name = :categoryName
          };
      values[1] = "" + checkoutPeriod;
      values[2] = "" + maxBooksOut;
      return values;
   }
}
```

```
    public String[] getCategoryInformation(String categoryName) throws ErrorMessage
    {
        String [] values = new String[3];
        values[0] = categoryName;
        int checkoutPeriod, maxBooksOut;
        try
        {
            /*@lineinfo:generated-code*//*@lineinfo:639^4*/
//  ************************************************************
//  #sql { select checkout_period, max_books_out
//                                 from category
//                                 where category_name = :categoryName
//                       };
//  ************************************************************
{
  sqlj.runtime.profile.RTResultSet __sJT_rtRs;
  sqlj.runtime.ConnectionContext __sJT_connCtx = sqlj.runtime.ref.DefaultContext.getDefaultContext();
  if (__sJT_connCtx == null) sqlj.runtime.error.RuntimeRefErrors.raise_NULL_DEFAULT_CONN_CTX();
  sqlj.runtime.ExecutionContext __sJT_execCtx = __sJT_connCtx.getExecutionContext();
  if (__sJT_execCtx == null) sqlj.runtime.error.RuntimeRefErrors.raise_NULL_EXEC_CTX();
 String __sJT_1 = categoryName;
 synchronized (__sJT_execCtx) {
    sqlj.runtime.profile.RTStatement __sJT_stmt = __sJT_execCtx.registerStatement(__sJT_connCtx, Database_SJProfileKeys.ge
   try
    { __sJT_stmt.setString(1, __sJT_1);
       sqlj.runtime.profile.RTResultSet __sJT_result = __sJT_execCtx.executeQuery();
      __sJT_rtRs = __sJT_result;
    }
    finally
    {
       __sJT_execCtx.releaseStatement();
    }
  }
 try
  { sqlj.runtime.ref.ResultSetIterImpl.checkColumns(__sJT_rtRs, 2);
   if (!__sJT_rtRs.next())
   {
       sqlj.runtime.error.RuntimeRefErrors.raise_NO_ROW_SELECT_INTO();
   }
    checkoutPeriod = __sJT_rtRs.getIntNoNull(1);
    maxBooksOut = __sJT_rtRs.getIntNoNull(2);
   if (__sJT_rtRs.next())
   {
       sqlj.runtime.error.RuntimeRefErrors.raise_MULTI_ROW_SELECT_INTO();
   }
  }
  finally
  {
```

TRANSLATION INTO "PURE JAVA" PRODUCED BY THE DB2 SQLJ COMPILER. (THE SQL IS ALSO TRANSLATED INTO A FORM THAT IS NOT HUMAN-READABLE)

Object Relational Mapping (ORM)

- Allows application code to be written on top of object-oriented data model, while storing data in a traditional relational database
  - alternative: implement object-oriented or object-relational database to store object model
    - has not been commercially successful
- Schema designer has to provide a mapping between object data and relational schema
  - e.g. Java class *Student* mapped to relation *student,* with corresponding mapping of attributes
  - An object can map to multiple tuples in multiple relations
- Application opens a session, which connects to the database
- Objects can be created and saved to the database using
  session.save(object)
  - mapping used to create appropriate tuples in the database
- Query can be run to retrieve objects satisfying specified predicates

Object-Relational Mapping and Hibernate

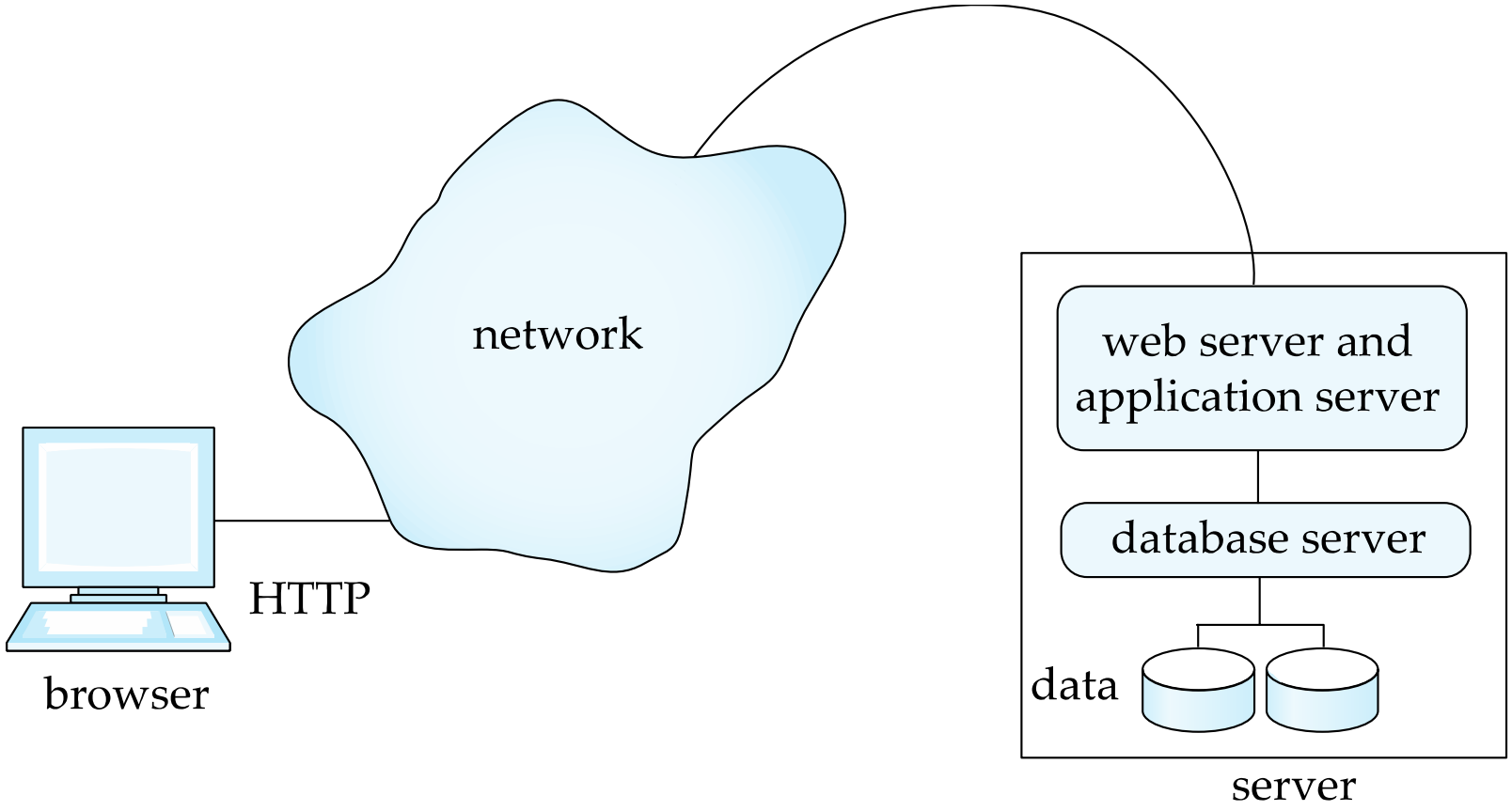
- The **Hibernate** object-relational mapping system is widely used
  - public domain system, runs on a variety of database systems
  - supports a query language (HQL) that can express complex queries involving joins
    - translates queries into SQL queries
  - allows relationships to be mapped to sets associated with objects
    - e.g. courses taken by a student can be a set in Student object
  - see page 394 of *Database System Concepts* for Hibernate code example
- The **Entity Data Model** developed by Microsoft
  - provides an entity-relationship model directly to application
  - maps data between entity data model and underlying storage, which can be relational
  - Entity SQL language operates directly on Entity Data Model
- DBIx::Class package for Perl and the Perl DBI

Design Project Presentations

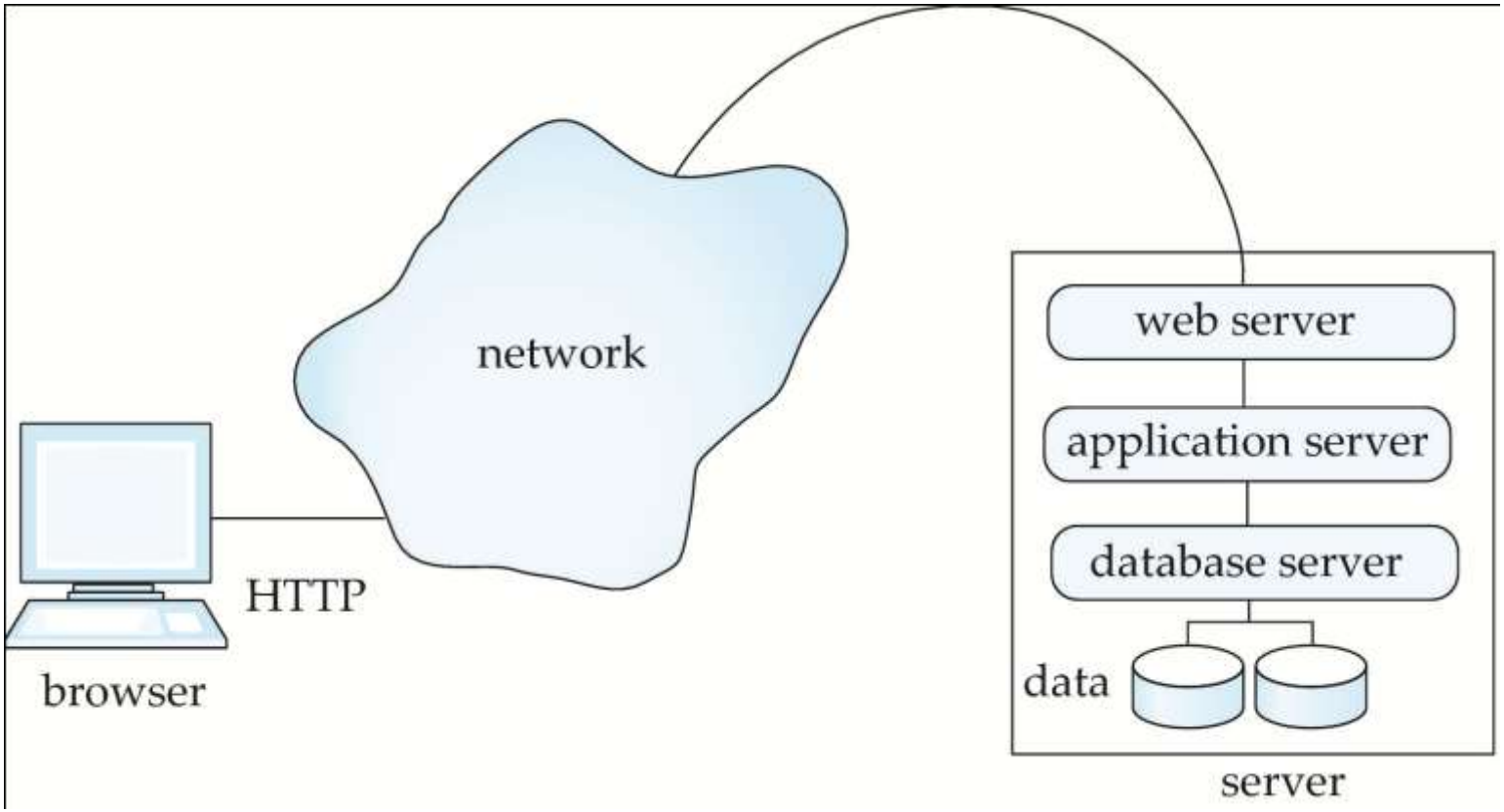
Part 2: E-R Diagrams

Web Application Architecture

Two-Tier Web Architecture



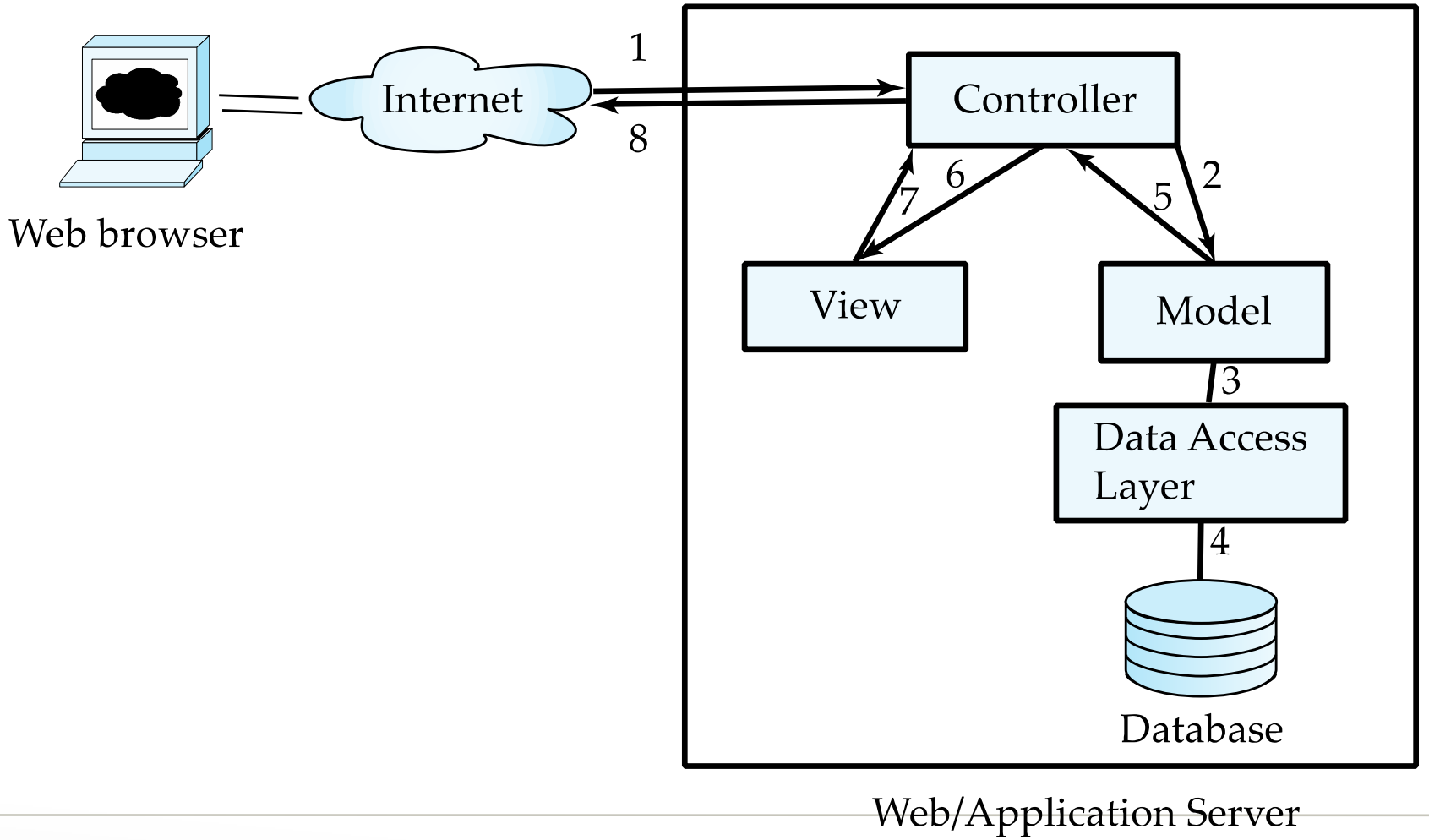
Three-Tier Web Architecture



Application Layers

- Presentation or user interface
  - **model-view-controller (MVC)** architecture
    - **model**: business logic
    - **view**: presentation of data, depends on display device
    - **controller**: receives events, executes actions, and returns a view to the user
- **business-logic** layer
  - provides high level view of data and actions on data
    - often using an object data model
  - hides details of data storage schema
- **data access** layer
  - interfaces between business logic layer and the underlying database
  - provides mapping from object model of business layer to relational model of database

Application Architecture Diagram



Business Logic Layer

- Provides abstractions of entities
  - e.g. students, instructors, courses, etc.
- Enforces **business rules** for carrying out actions
  - E.g. student can enroll in a class only if she has completed prerequisites, and has paid her tuition fees
- May support **workflows** which define how a task involving multiple participants is to be carried out
  - E.g. how to process application by a student applying to a university
  - Sequence of steps to carry out task
  - Error handling
    - e.g. what to do if recommendation letters not received on time

Web Services

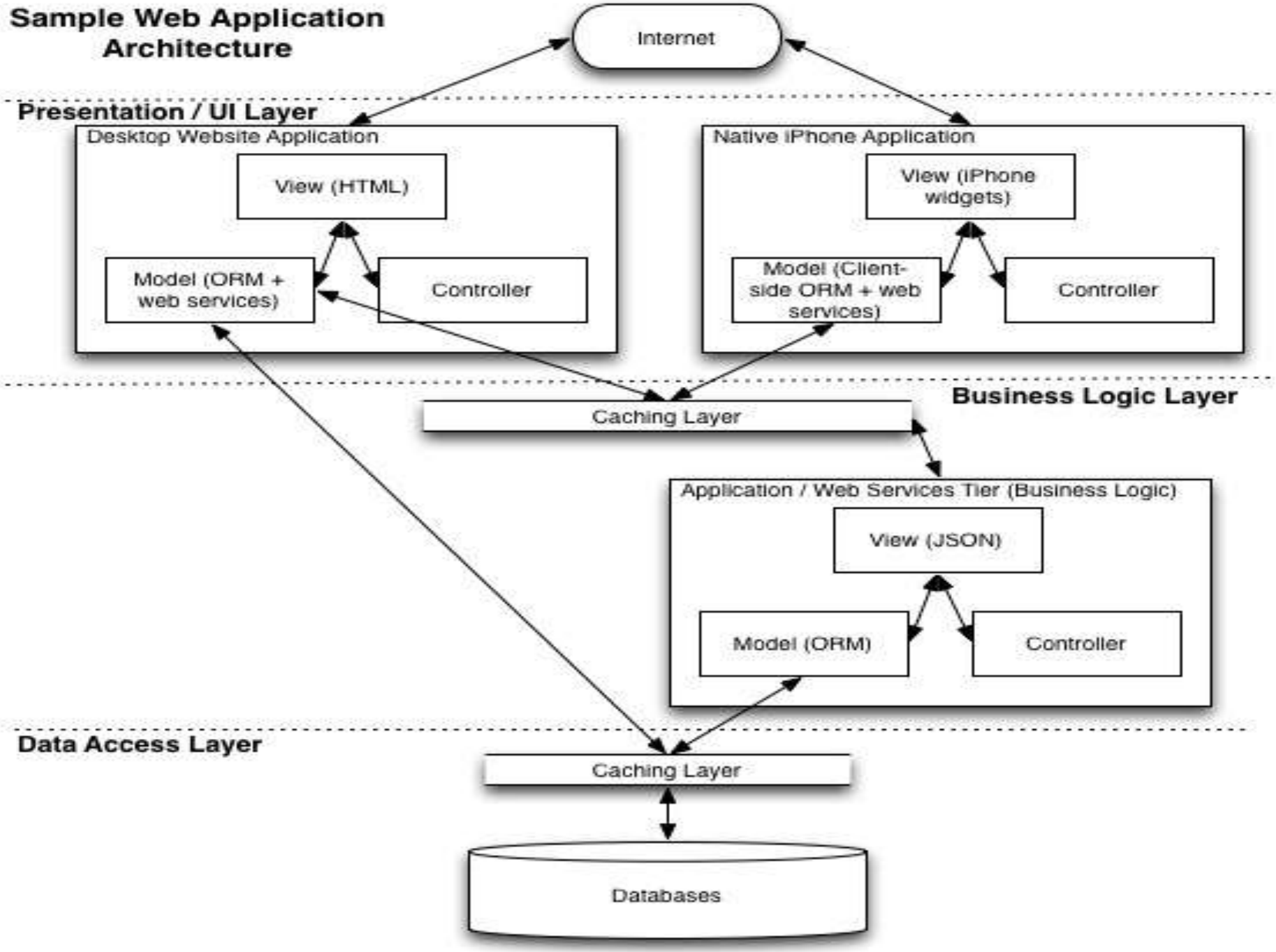
- Allow data on Web to be accessed using remote procedure call mechanism
- Two approaches are widely used
  - **Representation State Transfer (REST)**: allows use of standard HTTP request to a URL to execute a request and return data
    - returned data is encoded either in XML, or in **JavaScript Object Notation (JSON)**
      - JSON is lightweight and immediately usable in Javascript
  - **Big Web Services:**
    - uses XML representation for sending request data, as well as for returning results
    - standard protocol layer built on top of HTTP
      - e.g. SOAP, RPC
    - More overhead involved, but also more standardized

Rapid Application Development

- A lot of effort is required to develop Web application interfaces
  - more so, to support rich interaction functionality associated with Web 2.0 applications
- Several approaches to speed up application development
  - Function library to generate user-interface elements
  - Drag-and-drop features in an IDE to create user-interface elements
  - Automatically generate code for user interface from a declarative specification
- Above features have been in used as part of **rapid application development (RAD**) tools even before advent of Web
- Web application development frameworks
  - Java Server Faces (JSF) includes JSP tag library
  - Spring Roo (Java)
  - Ruby on Rails
    - Allows easy creation of simple **CRUD** (create, read, update and delete) interfaces by code generation from database schema or object model
  - Perl Catalyst framework

Web Application Performance Optimization

- Performance is an issue for popular Web sites
  - May be accessed by millions of users every day, thousands of requests per second at peak time
- Caching techniques used to reduce cost of serving pages by exploiting commonalities between requests
  - At the server site:
    - Caching of JDBC connections between servlet requests
      - a.k.a. **connection pooling**
    - Caching results of database queries
      - Cached results must be updated if underlying database changes
    - Caching of generated HTML and web service reponses
  - At the client's network
    - Caching of pages by Web proxy
    - Content delivery network (CDN)


Sample Web Application Architecture
Internet
Presentation / UI Layer
Desktop Website Application
View (HTML)
Model (ORM + web services)
Controller
Native iPhone Application
View (iPhone widgets)
Model (Client-side ORM + web services)
Controller
Business Logic Layer
Caching Layer
Application / Web Services Tier (Business Logic)
View (JSON)
Model (ORM)
Controller
Data Access Layer
Caching Layer
Databases

Database Design Tips

The Importance of Good Names

- Names chosen for database objects (i.e. tables, columns) will probably last a long time
- Naming guidelines
  - Appropriately descriptive (depending on context)
    - Consider a table named "transfer_student"
    - Potential candidates for primary key column name
      - "transfer_student_id", "student_id", "id"
      - "Best" choice may depend on how this column will be accessed (i.e. "select transfer_student.id …"
  - Succinct yet clear
    - Consider the name of a table to hold data on candidates for the US House of Representatives
      - "united_states_house_of_representatives_candidate" vs. "ushrc" – neither is good
      - "house_candidate" might be a good balance
    - Need to consider DBMS character limit restrictions (i.e. Oracle allows a max of 32 characters for names of database objects)
    - Separate words in names with underscores
      - Camel case will not work well because some DBMS's print names in ALL UPPERCASE

Tables

- Table names
  - Singular (or plural) name of stored entity (be consistent)
  - May include short (2-5 character) prefix to group related tables within a single schema
    - Example: "dlm" = Downloadable Media" (i.e. dlm_product, dlm_vendor)
- Columns every table definition should include
  - id – unique integer value used for primary key
    - Independent of all other data in the table that may change (including keys)
  - Date/time stamp columns
    - created and last_modified – for tables whose records might be updated
    - timestamp – for tables whose records will never be updated (i.e. page_view)
  - status – current state of each record in the table
    - i.e. Active, Inactive, Pending
    - Provides a way to "turn off" a record without actually deleting it (logic to check this must be coded in the database application)
- Break up "wide" tables with too many columns into smaller tables (decomposition)
  - Sets of related columns that could form their own table (relation)
  - Sparsely populated columns

Columns

- Column names
  - Prefer conciseness: i.e. “page_count” over “number_of_pages”
  - Phrase columns containing Boolean values as questions
    - Examples: is_checked_out, can_merge_into_superrobot
    - Value of column should answer the question
    - Foreign key columns – *foreign_table_name*_id
- Boolean vs. enumerated values
  - When creating a column to hold a Boolean value, consider if there could ever be a “third” answer beyond true and false
  - Example
    - “is_active” column set to true if the record is active and false otherwise
    - What happens when a record can be in a pending state
    - “status” would be a better name – allows for a short set of enumerated values (Active, Inactive, Pending)
- “Flags in the wind”
  - Scenario: want to store many similar pieces of data about a record
    - i.e. preference data: fiction, bibles, homeschool, pastor, music, etc.
    - Don’t create separate Boolean columns for each flag
    - Do create a separate table to store this information via a one- or many-to-many relationship with the original table

Application Design

- Keep business logic out of the presentation and data access layers
  - Ties you to a given platform or client and DBMS
    - If the web server or database ever changes, need to recode business logic
    - If additional clients or databases need to be supported, need to duplicate business logic
    - Avoid triggers and stored procedures – these store business logic in the data access layer
  - Where should business logic go?
    - In the model (MVC) – allows reuse throughout the application
    - In the application tier (as web services)
      - Allows access from multiple platforms / programming languages

Exam 1