INFO-340: Database Management and Information Retrieval

Winter 2007
B.S. Informatics
Information School
University of Washington

Theories and models in system-centered approaches to information retrieval and database management. Information retrieval and database management systems include text and multimedia databases, web search engines and digital libraries. Issues in system design, development and evaluation, and tools for searching, retrieval, user interfaces, and usability. **Prerequisite:** CSE 373

Course website & Listserv

http://courses.washington.edu/info340/

info340a_wi07@u.washington.edu
(Archive: https://mailman1.u.washington.edu/mailman/listinfo/info340a_wi07)
Registered students are subscribed automatically using their UW mail account.

Credit Hours

5 (3 lecture hours; 2 lab hours; 10 outside hours)

Meeting times

<table>
<thead>
<tr>
<th>Lecture</th>
<th>Tuesday/Thursday 130 – 250, MEB 242</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab</td>
<td>Thursday 330 – 520, MGH 430</td>
</tr>
</tbody>
</table>

Instructor

David Hendry, Assistant Professor
330J Mary Gates Hall
dhendry@u.washington.edu | http://faculty.washington.edu/dhendry

Office hours: Tuesday, 330 – 430 or by appointment.

Teaching assistant

Beth Fournier, MLIS Student
bethf2@u.washington.edu

Office hours: Friday, 230 – 330 or by appointment (in TE Lab, MGH 440)

Student services

Dowell Eugenio, Student Services Administrator
470E Mary Gates Hall
deugen3@u.washington.edu

Please note: If you have any concerns about a course or the TA, please see the TA about these issues as soon as possible. If you are not comfortable talking with the TA or not satisfied with the response that you receive, you may contact the instructor of the course. If you are still not satisfied with the response that you receive, you may contact Joseph Janes, the Associate Dean for Academics in 370 Mary Gates Hall, by phone at: (206) 616-0987, or by e-mail at jwj@u.washington.edu. You may also contact the Graduate School at G-1 Communications Building, by phone at (206) 543-5900, or by e-mail at efeetham@u.washington.edu.
Overview
Information systems have an enormous impact on our personal and civic lives. We find information systems virtually everywhere we live, work, and play.

Information systems can be examined from many different perspectives. For example, we could study:

- The productivity benefits—or costs—of information systems to a person, a firm or a nation;
- The mathematics and engineering research base that underpins information systems;
- Why information systems often fail and methodologies that mitigate the risk;
- Specific types of information systems in domains such as urban planning, health care, environment science, popular culture, business, etc.

In this class, however, we will leave these important areas of study to the side. This class introduces the theory and practice of information system design. It focuses on two fundamental approaches: Relational Database systems and Information Retrieval (IR) systems. We shall see that these are distinct approaches for solving different problems.

Drawing on your experience in programming, in website development, and knowledge organization this course will enable you to develop working systems and prepare you for advanced courses in database and IR systems.

Textbooks and readings
The textbook for this course is


This textbook will be used for the first half of the course. It has been selected because of its breadth and depth of coverage of relational databases. It is well written, well designed, and contains many examples. You should find this book to be useful for several years to come.

For the second half of the course, we will draw upon readings from several sources including:


If you are particularly interested in Information Retrieval, consider purchasing the book by Belew. Readings for the second half of the course will be posted on the website.
Learning
Aims
The general aims of this course are to:
1. Develop a conceptual understanding for relational database and information retrieval systems
2. Develop skills in implementing information systems using these two approaches
3. Improve skills in collaboration in technical teams
4. Develop skills for integrating technologies to develop a small but complete application.

Objectives
On the successful completion of this course, you should be able to:
1. Describe the components of an information system and list risks for why information systems fail
2. Describe and practice techniques in conflict management and describe a developmental model for teams
3. Describe the functions and organization of database management systems
4. Describe the relational model, including the data structure and algebra
5. From problem statements, derive SQL statements for querying, updating and creating databases
6. Create Entity-Relationship and Enhanced-Entity Relationship models for small systems
7. Read an ER diagram as a specification and implement a database system for it
8. Describe the problems of data redundancy and update anomalies and be able to normalize a database to 3NF to avoid these problems
9. Describe a three-tier information system
10. Outline a methodology for designing database applications
11. Implement a small database-driven website using ODBC as the middle tier using the following open source tools: NetBeans 5.5, JSP, JDBC, and PostgreSQL
12. Describe the function and organization of an information retrieval (IR) system, including documents, document collections, terms, queries, matching, ranking, and results
13. Explain how an inverted file works and describe its basic space and time complexity
14. Describe the difference between Boolean and ranked retrieval
15. Know the formula for the Zipf distribution and recognize its curve in a data set
16. Explain inverse document frequency and other methods for weighting terms in documents
17. Describe PageRank
18. Given several weighting functions, explain their difference
19. Using existing components from the Lucene framework, implement a search interface for a website
20. Describe concepts for evaluating information systems, including system performance metrics (e.g., coverage, precision, recall, etc.) and usability metrics (task completion time, number of errors, etc.)
21. Describe methods for evaluating information systems (e.g., usability evaluations, log file analysis, etc.)
Academic accommodations

To request academic accommodations due to a disability, please contact Disabled Student Services: 448 Schmitz, 206-543-8924 (V/TTY). If you have a letter from DSS indicating that you have a disability which requires academic accommodations, please present the letter to me so we can discuss the accommodations you might need in the class.

Academic accommodations due to disability will not be made unless the student has a letter from DSS specifying the type and nature of accommodations needed.

For additional information, see Statements to Ensure Equal Opportunity and Reasonable Accommodation, downloaded March 5, 2003, http://www.washington.edu/admin/eoo/eoost.html

Academic honesty

The essence of academic life revolves around respect not only for the ideas of others, but also their rights to those ideas and their promulgation. It is therefore essential that all of us engaged in the life of the mind take the utmost care that the ideas and expressions of ideas of other people always be appropriately handled, and, where necessary, cited. For writing assignments, when ideas or materials of others are used, they must be cited. The format is not that important—as long as the source material can be located and the citation verified, it’s OK. What is important is that the material be cited. In any situation, if you have a question, please feel free to ask. Such attention to ideas and acknowledgment of their sources is central not only to academic life, but life in general.

Please acquaint yourself with the University of Washington's resources on academic honesty: http://depts.washington.edu/grading/issue1/honesty.htm

Students are encouraged to take drafts of their writing assignments to the Writing Center for assistance with using citations ethically and effectively. Information on scheduling an appointment can be found at: http://www.uwtc.washington.edu/resources/ewic/

Copyright

All of the expressions of ideas in this class that are fixed in any tangible medium such as digital and physical documents are protected by copyright law as embodied in title 17 of the United States Code. These expressions include the work product of both: (1) your student colleagues (e.g., any assignments published here in the course environment or statements committed to text in a discussion forum); and, (2) your instructors (e.g., the syllabus, assignments, reading lists, and lectures). Within the constraints of "fair use", you may copy these copyrighted expressions for your personal intellectual use in support of your education here in the iSchool. Such fair use by you does not include further distribution by any means of copying, performance or presentation beyond the circle of your close acquaintances, student colleagues in this class and your family. If you have any questions regarding whether a use to which you wish to put one of these expressions violates the creator's copyright interests, please feel free to ask the instructor for guidance.
Privacy
To support an academic environment of rigorous discussion and open expression of personal thoughts and feelings, we, as members of the academic community, must be committed to the inviolate right of privacy of our student and instructor colleagues. As a result, we must forego sharing personally identifiable information about any member of our community including information about the ideas they express, their families, life styles and their political and social affiliations. If you have any questions regarding whether a disclosure you wish to make regarding anyone in this course or in the iSchool community violates that person's privacy interests, please feel free to ask the instructor for guidance.

Knowing violations of these principles of academic conduct, privacy or copyright may result in University disciplinary action under the Student Code of Conduct.

Student Code of Conduct
Good student conduct is important for maintaining a healthy course environment. Please familiarize yourself with the University of Washington's Student Code of Conduct at: http://www.washington.edu/students/handbook/conduct.html

Assessment
The assignments strike a balance between theory and practice and between individual and group work.

<table>
<thead>
<tr>
<th>Assessment</th>
<th>% Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four individual assignments</td>
<td>25%</td>
</tr>
<tr>
<td>Group project</td>
<td>20%</td>
</tr>
<tr>
<td>Midterm exam</td>
<td>15%</td>
</tr>
<tr>
<td>Final exam</td>
<td>30%</td>
</tr>
<tr>
<td>Spirit and participation in classes/labs</td>
<td>10%</td>
</tr>
</tbody>
</table>

Individual assignments
You will complete four individual assignments:

<table>
<thead>
<tr>
<th>Assignment</th>
<th>Due</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1. Search Interface with the Google Base API</td>
<td>Jan 23</td>
<td>#4</td>
</tr>
<tr>
<td>A2. SQL Data Definition and Manipulation</td>
<td>Jan 30</td>
<td>#5</td>
</tr>
<tr>
<td>A3. Database Design</td>
<td>Feb 06</td>
<td>#6</td>
</tr>
<tr>
<td>A4. Information Retrieval: Matching &amp; Ranking</td>
<td>Mar 01</td>
<td>#9</td>
</tr>
</tbody>
</table>

A1 is worth 10%. A2, A3, and A4 are each worth 5%.

Please note: All assignments are due at the beginning of class.
Social Bookmarking Project
Working in groups of 2-3, you will develop an information system. The deliverables:

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1. System Design, Rough Draft</td>
<td>Jan 25</td>
<td>#4</td>
</tr>
<tr>
<td>P2. Design &amp; Code: Functions 1–2</td>
<td>Feb 22</td>
<td>#8</td>
</tr>
<tr>
<td>P3. Final System</td>
<td>Mar 08</td>
<td>#10</td>
</tr>
<tr>
<td>P4. Demonstration (to be arranged with Beth)</td>
<td>Mar 08-09</td>
<td>#10</td>
</tr>
</tbody>
</table>

Notes
1. P1 and P2 are worth 5% each and final system is worth 15%.
2. Deliverables are due at the beginning of class.
3. Groups will be selected by the instructor and teaching assistant on the basis of a skills profile.

Midterm and final exam
The mid term and final exam will assess your knowledge the conceptual foundations of relational database systems and information retrieval systems.

<table>
<thead>
<tr>
<th>Exam</th>
<th>Percentage</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Midterm exam</td>
<td>15%</td>
<td>Thr, Feb 8, in class</td>
</tr>
<tr>
<td>Final exam</td>
<td>30%</td>
<td>Friday, March 16, 2007, 230 – 430, MEB 242*</td>
</tr>
</tbody>
</table>

* Please note: We will seek to change this date to Tue March 13, 1:30 – 3:30 but this will require the unanimous consent of all students.

Spirit and participation in classes/labs
It is important to the instructor and teaching assistant that you help make INFO-340 fun, interesting, and challenging. With spirit and a professional manner, we can create a supportive and rewarding learning environment.

Among the things you can do are:
1. Treat all with respect – be constructive in all discussions
2. Come to class prepared – read carefully and be ready for discussion
3. Be an active listener – be attentive, be engaged, use in-class technology with discretion
4. Ask challenging questions, participate in discussion
5. Comment, build on, or clarify what others have done or said
6. Help your classmates use development tools and technologies
7. Post useful or interesting information to the class discussion list
8. Visit the instructor during office hours to chat, to ask questions, or to give feedback.

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Due</th>
<th>Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Personal Statement</td>
<td>Mar 8</td>
<td>#10</td>
</tr>
</tbody>
</table>

Please write a 2 or 3 paragraph personal statement on how you contributed to the class.
Attendance at labs will be taken and your lab website will be checked periodically. Participation
is graded because responsiveness and involvement are crucial elements of your development. Your participation is worth 10% of your final grade.

**Grading criteria**

Work in this course will be graded to criteria. In other words, you won't be graded on a curve. Each deliverable is designed to test your achievement against one or more of the learning objectives. Different assignments emphasize different learning objectives. The meanings of grades are described below.

General grading information for the University of Washington is available at:

- [http://www.washington.edu/students/gencat/front/Grading_Sys.html](http://www.washington.edu/students/gencat/front/Grading_Sys.html)

The iSchool has adopted its own criteria for grading graduate courses. The grading criteria used by the iSchool are available at:

- [http://depts.washington.edu/grading/practices/guidelin.htm](http://depts.washington.edu/grading/practices/guidelin.htm)

<table>
<thead>
<tr>
<th>Grade</th>
<th>Performance Quality*</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.9 - 4.0</td>
<td>Superior performance in all aspects of the course with work exemplifying the highest quality. Unquestionably prepared for subsequent courses in field.</td>
</tr>
<tr>
<td>3.5 - 3.8</td>
<td>Superior performance in most aspects of the course; high quality work in the remainder. Unquestionably prepared for subsequent courses in field.</td>
</tr>
<tr>
<td>3.2 - 3.4</td>
<td>High quality performance in all or most aspects of the course. Very good chance of success in subsequent courses in field.</td>
</tr>
<tr>
<td>2.9 - 3.1</td>
<td>High quality performance in some of the course; satisfactory performance in the remainder. Good chance of success in subsequent courses in field.</td>
</tr>
<tr>
<td>2.5 - 2.8</td>
<td>Satisfactory performance in the course. Evidence of sufficient learning to succeed in subsequent courses in field.</td>
</tr>
<tr>
<td>2.2 - 2.4</td>
<td>Satisfactory performance in most of the course, with the remainder being somewhat substandard. Evidence of sufficient learning to succeed in subsequent courses in field with effort.</td>
</tr>
<tr>
<td>1.9 - 2.1</td>
<td>Evidence of some learning but generally marginal performance. Marginal chance of success in subsequent courses in field.</td>
</tr>
</tbody>
</table>

*Taken from Faculty Resource on Grading, downloaded March 5, 2003, http://depts.washington.edu/grading/practices/guidelin.htm

**Standard cover sheet**

To protect your privacy when exercises are returned and to facilitate communication, submitted work must have a cover sheet. The cover sheet must include the following information and be formatted nicely:

- Course name,
- Quarter, program, department, and university
- Assignment name
- Your name and e-mail address
- A date
- A web site address (if relevant).

Staple the exercise pages to the cover sheet.
Late policy
1. If you will miss the deadline, you should inform the instructor as soon as you can, indicating when you will submit the work. The instructor will try to accommodate your needs. You should use this clause only for extraordinary personal reasons.
2. It is at the instructor's discretion to accept late work or assign late penalties (see 1 above). For any late assignment, 10% will be taken off your work per day. After five days, your work will not be accepted.
3. Late work must be handed to the instructor or teaching assistant in person. You may also be able to hand work in at the front desk of the Information School and at student services but this cannot be guaranteed.

Work that is handed in late is penalized for two reasons. First, to be fair, all students should be given the same time limits. Second, if you spend too much time on one assignment, it is quite likely that you will have insufficient time to spend on subsequent assignments.

Right to revise
The instructor reserves the right to revise this syllabus.

Re-grading policy
To have work re-graded, you must submit a Re-grade Request within five days of when your work was returned. The request must be a single page, printed on paper or sent by e-mail. It should contain the following information:

- Re-grade Request
- The information contained on the standard cover sheet
- An explanation for why you believe you deserve a higher grade.

The instructor, possibly in collaboration with the teaching assistant, will consider your request. If the instructor is convinced by your argument, your work will be re-graded. If not, the instructor will send you e-mail explaining why. No re-grades will be considered for late work.

Guidelines on using e-mail
When communicating with the instructor or teaching assistant, please follow these guidelines:

- You are welcome to give feedback to the instructor and teaching assistant about the course, to ask a question about an assignment, to share an interesting article or resource, to report that you will be absent from a class/lab, to request additional time for an assignment (because of significant health, personal, or educational matter), or similar communication;
- Whenever appropriate, please copy the class listserv with your question or comment;
- E-mail concerning assignments might not be replied to if it is sent within 36hr of the assignment due date;
- If your e-mail concerns your grade, please follow the re-grading policy (see above);
- E-mail that is sent on Friday afternoon or over the weekend is not replied to until Monday or Tuesday of the following week;
- If you don’t receive a reply within 2 days or so, please resend your e-mail or ask about it during class or lab.
Class Schedule

Week 1: Overview
Read  C & B, Chap. 1-2
L1  Greetings;
L2  Introduction to Relational Database Systems
Lab  Development Tools, I

Week 2: Relational model
Read  C & B, Chap. 3-4 (4.2 optional)
L1  Relational Data Structure & Relational Integrity
L2  Relational Algebra
Lab  Development Tools, II

Week 3: SQL
Read  C & B, Chap. 5-6, Appendix C
L1  Introduction to Storage and Indexing
L2  SQL Query Language
Lab  SQL

Week 4: ER-Modeling
Read  C & B, Chap. 9, 11-12,
L1  Entity Relationship Modeling, I
L2  Entity Relationship Modeling, II
Lab  Introduction to JDBC, I

Week 5: Normalization
Read  C & B, Chap. 13
L1  Database Normalization
L2  Review of Database Systems
Lab  Introduction to JDBC, II

Week 6: Introduction to IR Systems
L1  Introduction to Information Retrieval (IR) systems
L2  Midterm
Lab  Teamwork and Project Work

Week 7: Documents and indexing
Read  • Belew, Chap. 1-2
L1  Documents, Metadata & Document Surrogates
L2  Indexing
Lab  Introduction to Lucene, I

Week 8: Queries and matching
Read  Belew, Chap. 3
L1  Inverted File Structure
L2  Weighting and Matching
Lab  Introduction to Lucene, II

Week 9: Evaluation
Read  Baeza-Yates & Ribeiro-Neto, Chap. 3
L1  Precision/Recall experimentation
L2  Usability Evaluation
Lab  Project Work

Week 10: Review
L1  DB Review
L2  Review & Future Directions
Lab  DEMO lab (optional)
Learning objectives
- Develop knowledge for using a web service for search, especially query and result abstractions;
- Implement a small data-driven website in JSP, demonstrating the ability to:
  - Separate application logic (Java package) from data presentation (JSP files);
  - Process HTML form controls;
  - Dynamically generate HTML pages with JSP;
- Develop basic skills for the following tools and techniques:
  - Basic JSP coding;
  - Calling Java APIs from JSP;
  - Using NetBeans 5.5 IDE for Java and JSP development;
  - Use of WAR files.

Background
Google provides a variety of APIs that enable 3rd parties to develop interesting applications. In this assignment you will implement a simple search interface that relies upon the Google Base API.

For an introduction to the Google Base, see:  
http://code.google.com/apis/base/

The problem
The problem is to build a specialized search application for finding cars that have been added to Google Base. Google Base provides an API that can be used to store and search for any kind of structured entity, including recipes, books, jobs, events, and so on. In this assignment, we focus on cars only because it’s a relatively simple example.

The solution sketch
Your solution should follow this structure:

A. Search dialog The user enters a query and presses the "search" button;

B1. Results page If there are results, they are listed and the user can page forward and backward through them;

B2. No results found page If there are no results, tell the user.
A1: Developing a Search Interface with the Google Base (Version: 03 Jan 2007)
Worth: 10% (individual work)

User interface: Functional requirements

A. Search Dialog
The search dialog should enable users to do the following:
1. Enter free text (e.g., "hardtop convertible");
2. Select one make of car from a selection box (e.g., Audi, Volkswagen, etc.);
3. Select one or more colors using checkboxes;
4. Enter low and high price points.

Appropriate “default” values should be provided so that users are not required to enter complete information to do a search.

B1. Results Page
The Results Page should do the following:
1. Show the query and total number of results;
2. Show the first 10 results on one page;
3. Allow the user to page forward and backward through the results and to return to the search dialog page;
4. Each result should contain the following attributes:
   a. A title that is linked to a URL;
   b. The year of the car;
   c. Make and Model;
   d. Color, mileage, and price.

B2. No Results Found Page
If a user’s query produces zero results, the No Results Found Page should be shown. Show the user’s query and allow the user to return to the search dialog.

You are welcome to go beyond these minimal requirements but please follow this basic dialog structure.

Implementation
Your application should consist of two layers:

- **User-Interface layer.** Consists of one or more JSP pages. The code in these pages should concern presentation matters only. This layer should call upon the application logic layer.

- **Application logic layer.** Consists of one or more Java classes. Your classes will call the Google Base Application Programming Interface (API). The code in this class should concern application logic only. Your classes should not return HTML code.

To implement your solution, you should follow these steps:

2. Become familiar with JSP coding in NetBeans (lab #1);
3. Learn the basics of HTTP requests and responses and form processing in JSP (lab #1-2);

4. Become familiar with Google Base Query Language
   http://code.google.com/apis/base/attrs-queries.html

5. Obtain a Google Base developer key at:
   http://code.google.com/apis/base/signup.html

6. Study the example program for using the Google Base API (available on website);

7. Setup a NetBeans project and get the example program to work;

8. Study the skeleton Java class for how to structure the Application Logic Layer (available on Website) and then extend this class to implement the necessary application logic;

9. Develop JSP pages to handling the form processing and application flow;

10. Iterate over steps 8 and 9;

11. Prepare the deliverable (see below).

**Please note:** Labs #1-2 are designed to give you the technical background for moving through these steps.

**Deliverable**

The deliverable for this assignment consists of **two** parts:

1. An URL to your website;
2. A printed report handed in at the beginning of class.

Please see syllabus for due dates.

**1. Website URL**

To submit your website, please send an e-mail message to the teaching assistant (see syllabus for e-mail address). To help the teaching assistant, please use this subject line:

INFO340 <last name> A1: URL

The body of the e-mail should include the URL to your working website.

Your website should include a cover page. The cover page should concisely introduce and summarize this assignment and include a link to the Search Dialog Page.

Please assume that the reader of your site is a **prospective employer** who wishes to assess your skills. Thus, conciseness and clarity of expression are valued.
2. Printed Report
The report should have the following sections:
   a. A coversheet with the usual information (please see syllabus) plus the URL to
      your web application;
   b. A 2-3 page description of your system and how it works. You should:
      Part I. Draw a diagram showing the components that you used in the
      system;
      Part II. Write a brief description, explaining how you separated the
      application logic and presentation layers;
      Part III. Highlight any noteworthy features or limitations of your application.
   c. Appendix A. The JSP code.
   d. Appendix B. The Java application code.

Please comment your code and present it neatly in the appendices.

Grading Guidelines
1. A JSP application is written that processes an HTTP request and generates response pages as
   described above (10 points);

2. A Java package is written to send a queries to the Google Base API and to process the
   results (10 points);

3. The web application is precisely described and the code is well documented (15 points);

4. The website cover page is thoughtfully designed – it is concise, clear, and complete (5
   points);

5. BONUS: You go beyond the minimal requirements by providing additional search
   functionality OR you do an outstanding job of describing how your system works (5 points).

Please note: You must demonstrate that you’ve achieved an effective separation of application
logic from presentation.
Learning objectives

1. From simple problem statements, derive SQL statements for querying, updating and creating databases;
2. Describe the relational model, including the data structure and algebra.

Introduction

To complete this assignment, you will need to load your database with three tables: Pop, Zip, and State. The data and a script for creating these tables are available on the course website.

Tip: study these materials before beginning this assignment.

Deliverables

Label the tasks, 1—20, and format your SQL statements neatly. For example,

0. Show the state population figures.
   
   select * from Pop;

Please attach a coversheet to your work as described in the syllabus.

Notes:

While these tasks may appear simple, they are typically not. Think carefully about the task and be sure to test your solutions.

If you discover an ambiguity in a question, explain what it is, take a stance, and write the SQL statement from your stance. (Note: the tasks are not meant to contain ambiguities but you may find them.)

If you are asked to explain or fix something please examine the problem carefully and be sure that you fully understand the issue before answering the question.

Tasks

Answer the following questions. For those that require writing SQL statements, your statements should run in your database environment against the three tables (Pop, Zip, and State) and produce the correct answer.

1. List all the zip codes and cities for the state code WA and list your results alphabetically by city name.
2. List all details for all cities named Boston.
3. List the zip code, state code, and city name of all cities that contain ‘west’ in them in the states of Massachusetts, Washington, and Maine.
4. List the states (state codes only) with more than 1000 zip codes and list states alphabetically.
5. List the four states (state codes only) that fall within the lines 117 and 124 longitude.
6. For each state show the populations in 1990 and 2000 and the percent change (pop2000-pop1990)/pop1990.
7. List all states (state names only) with populations that exceed the average state population.
8. PostgreSQL can store three kinds of integers. What are they and what is the range of values that each can hold?
9. Find the number of zip codes in states with populations between 2.5-5.0 million in 2000. List the state name, state code, population in 2000, and number of zip codes. Order the results by state code.

10. Show the statement that makes the above query a view. Show an example query for your new view. Why might views be useful?

11. List the population and state code for all states found in the State table. Include state codes for which no population figures are available.

12. Write a query to add the record “Ontario, OT” into the State table. Write a query to modify the record “Ontario, OT” to “Ontario, ON” in the State table.

13. Write a query to remove the record “Northern Mariana Islands” from the State table.

14. The Zip table has no primary key defined. Write the SQL statement to make the column zcode the primary key. Why does this not work? How would you clean the data so you can apply a reasonable primary key? Please explain your answer.

15. Now imagine that you want to add to the Zip, Pop and State data to track where a person has traveled. You will do this with two additional tables, named Person and Visit. Create a table called Person which consists of a name, birthdate, and id which is auto-incremented. Create a table called Visit which consists of a personid, date, and zcode.

16. List two data integrity constraints that should be enforced on the table visit. (No SQL required – use English to write down your answers.)

17. Add a column address to the Person table. Remove the Person table.

18. Write one or more relational algebra expressions that lists the state name for the zip code 63675.

19. Explain the difference between the clauses HAVING and WHERE.

20. On one page, using blank paper and pencil, explain the relational model. Assume that the reader of your answer knows what the relational model is and wants to assess your understanding (e.g., in a job interview).

Marking guidelines
- Full marks will be given for solutions that are syntactically and logically correct and as simple as possible.
- Please do not include example output in your answers.
- Partial marks will be given for tasks that are partially correct.
Worth: 5% (individual work)

Learning Objectives
This assignment exercises these learning objectives:
1. Draw entity-relationship diagrams for small systems;
2. Read an entity-relationship diagram as a specification and implement a database.

Introduction
This assignment is divided into four parts. For each part, you should draw an ER or EER diagram. Please include a caption that rigorously describes what your diagram models. The caption should describe the entities, the relationships, and the multiplicities of the relationships. You should also explicitly state any assumptions that you make and justify those assumptions. Please use the UML notation that is employed in the textbook.

You may use pencil and blank paper to complete this assignment. Please be neat.

In each part, you should draw an ER/ERR diagram and include a caption that rigorously describes what your diagram models. The caption should describe the entities, the relationships, and the multiplicities of the relationships.

Part I
Assume that mechanics are certified to repair certain car models.
1. Draw an ER model of this situation that consists of one M-M relationship.
2. Decompose the model into a new one that consists of two 1-M relationships.

Part II
Imagine a company that runs projects where a mixture of skills is required. Projects are made up of programmers, engineers, and managers. An engineer can be qualified in applications- or system-development (and sometimes both). A manager can be either a technical manager or a business manager (but never both). Draw an enhanced entity relationship (EER) that models this scenario.

Part III
Imagine an information system for scheduling swim lessons for toddlers. A swim lesson can be defined by a ternary relationship between a group of toddlers (and adult guardian), an instructor, and a swimming pool.
1. Using EER, draw a conceptual model of this ternary relationship.
2. Using ER, simplify the conceptual model into a logical model, consisting of entities and 1-1 and 1-M relationships.

(Turn page over)
Worth: 5% (individual work)

**Part IV**

Consider a chain of hotels with different hotels in different cities. Each hotel has a different name. Hotels obviously have rooms of different types and prices. And, of course, guests check into hotels and stay for holidays and business. Draw an ER diagram that models this situation.

**Notes:**
1) Please include 4-7 entities in your model (a realistic design would consist of many more);
2) Please include several attributes for each entity in your model;
3) Please do not include any many-to-many relationships in your model;
4) Please indicate all primary and foreign keys; and
5) Please note the significant assumptions that you make.

**Grading**
Your work will be graded on the following:
1. An ER model that plausibly represents the scenario in a complete, useful fashion;
2. Rigorous and correct use of ER symbols;
3. A caption that accurately explains the model and states any assumptions;
4. A drawing that is neat and carefully presented.
Learning Objectives

After this assignment you should be able to:

1. Describe Zipf's law and recognize it when a frequency distribution is plotted;
2. Discuss the impact of different word normalizing approach on retrieval;
3. Discuss the how TF-IDF weighting algorithm.

Preliminaries

Note: To complete this assignment you will need a spreadsheet, which is posted on the course website.

This assignment consists of four parts, with several questions in each part.

Part I: Zipf's law

Zipf's law says that the frequency of occurrence of some event is related to its rank and that the relation can be defined by this relation:

\[ F(r) = \frac{C}{r^a} \quad a \approx 1 \quad C \approx 0.1 \]

\( r \) is the rank; \( F(r) \) is the frequency of the rank ; and \( a \) and \( C \) are parameters (that is, they are numbers that vary depending on the text that is analyzed).

This relation holds for many natural phenomena, including website popularity. See http://www.useit.com/alertbox/zipf.html

One way to explore Zipf's law is to follow these three steps:

1. Rank order events from the most frequent to the least frequent, giving a table like this

<table>
<thead>
<tr>
<th>Rank</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>f1</td>
</tr>
<tr>
<td>2</td>
<td>f2</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>n</td>
<td>fn</td>
</tr>
</tbody>
</table>

where \( f1 > f2 > ... fn \)

2. Compute the log10 of the Rank and the log10 of the Frequency

3. Plot a graph with log10(Rank) on the X-axis and log10(Frequency) on the Y-axis.

Questions

A: The spreadsheet contains the populations for 150 countries around the world. Following the above procedure, create a scatter plot for the country data. Now answer these questions:

1. What form does the data seem to take?
2. Does Zipf’s law apply to this data? Why or why not?
3. Why is it useful to plot the data on a log-log scale?
B: The spreadsheet contains two inverted files. Draw a scatter plot of the frequency distribution of these two inverted files. Does Zipf's law apply? Why or why not?

Please label your graphs carefully, with axis titles, plot title, and a caption that clearly describes the graph.

Part II: Terms

The Excel file contains two inverted files which were created with different indexing approaches but from a common corpus. Examine these inverted files carefully.

Questions:
A: For each inverted file, which term occurs in the most documents?
B: For each inverted file, which term occurs more frequently?
C: How many terms occur in each inverted file?
D: Carefully inspect the terms and hypothesize what indexing policies were applied in each case.
E: Discuss how using one inverted file or the other will change the performance of the IR system.

Part III: Search

Suppose that you searched for the term 'County' on INVERTED-FILE-1 and INVERTED-FILE-2. Now, suppose you search on the term 'county' on INVERTED-FILE-1 and INVERTED-FILE-2.

Discuss how and why the results from these queries might differ in their results. Will the number of documents retrieved be the same or different? Will the highest ranked document change? Do you expect the capitalization of 'county' to impact the number of document hits in the results?

Hint: The stem of 'county' is 'counti'

Part IV: Inverse Document Frequency

The TF-IDF weighting algorithm is represented by the following formula:

\[ w_{kd} = f_{kd} * \left( \log \frac{N_{rm}}{D_{k}} + 1 \right) \]

Answer the following questions:
A: In your own words, summarize the purpose of this algorithm and how it works;
B: Define each variable in this formula;
C: Draw a sketch of an inverted file, that is, the internal data structure that is needed to implement retrieval algorithms and show where values for each of these variables can be obtained.

Hint: Read Belew, 3.3 and especially 3.3.5 – 3.3.7
THE SOCIAL BOOKMARKING SYSTEM –
PROJECT BRIEF

Learning Aims
In this integrative project you will apply principles of database design and information retrieval to create a small but relatively complex information system. You will learn to use professional development tools and technologies (e.g., NetBeans 5.0, JSP, PostgreSQL, JDBC, Lucene search API). Most importantly, you will learn how to design and implement an information system that satisfies a relatively vague, open-end problem statement. Through this process you will gain confidence for tackling complex technical problems, develop your skills for design and implementation, and come to appreciate the beauty that underlies information systems.

Objective
The objective of this project is to develop a working prototype calling SBS for Social Bookmarking System. To guide this 8 week process, you will prepare a set of deliverables.

Root concept
SBS is an information system that enables people to collect items. People can describe items and tag them with keywords. Then, using the keywords people can browse and search their collections. In addition, your search should allow people to issue ‘best-match’ searches to retrieve a ranked list of the best matching items in your system.

Various kinds of social bookmarking systems are in wide use. One example is: [http://del.icio.us](http://del.icio.us)

Short scenarios
Consider these short scenarios that reify the root concept:
- A high school teacher wants to collect a set of websites on environmental education, especially in the Pacific Northwest.
- A professor wants her class to develop a collection of academic papers on the topic of “Why Information System Development Project often fail?”

Before reading further, please identify a similar scenario that might benefit from this approach. What features of your scenario make it like these examples?

User Tasks in SBS
Consider the following user tasks that people might perform with SBS. In these tasks, SMALL-CAPS-TEXT represents an entity/noun and underlined-text represents an operation/verb.

1. A USER can create a PROJECT.
2. A USER can submit ITEMS to his or her PROJECT.
3. A USER can assign zero, one, or more KEYWORDS to an ITEM.
4. A USER can organize the KEYWORDS into a HIERARCHY.
5. A USER can browse and search ITEMS by KEYWORD through a HIERARCHY.
6. A USER can search ITEMS by a free-text, best-match query (i.e., by words within items).
7. A USER can assign editing privileges to a group of USERS.
8. A USER’S BEHAVIOR is tracked and recorded for future analysis.
Before reading further, select a scenario from above or one of your own choosing and consider which tasks you think are most relevant. Does your scenario raise new tasks? Do some tasks for your scenario need to be modified? Do simpler tasks underlie the above tasks?

*Please note:* You will not need to implement solutions for all these tasks!

**General Requirements**

Your solution should satisfy the following requirements:

1. Develop an operational definition for the concept of keyword so that a solution can be implemented. You should be able to do useful things with keywords and it should be possible to represent a hierarchy of keywords of arbitrary depth. For example:
   
   ```
   keyword-A (sports)
   keyword-1 (skiing)
   keyword-a (downhill)
   keyword-b (x-skiing
   keyword-c (telemark)
   keyword-2 (cycling
   ... ...........................................
   ...
   ...
   keyword-B (eating)
   ...
   ```

2. Develop an abstraction that allows you to represent various kinds of items (e.g., webpages, music, movies, journal articles, and books). Assume that all item types have a title and a http link to the source.

3. Choose **ONE** item for your implementation but develop a general design so that **MANY** different kinds of items can potentially be represented in your system.

4. Develop an abstraction that allows you do track what users do to your system (e.g., track the keywords added, items added, items visited, etc.).

5. Develop an abstraction for users that includes a profile, including such information as age, gender, interests, favorite places, and so on.

6. Assume that image, audio and other such files are stored in an external storage infrastructure and that these files have unique identifiers that can be accessed by hypertext links. Thus, a file upload function and general storage infrastructure is not required.

7. Do not implement a secure user authentication scheme. You may encode unencrypted user identifiers on HTTP requests.

8. A high quality visual design is not required. The emphasis of this project should be on interesting and powerful functions. The user-interface needs only to reveal these functions and be neat and clear.

As you proceed through the project, you should repeatedly return to this list of requirements and explore how they can help you reduce the complexity of your project.

**Team Organization**

Teams will consist of three people and will be selected by the instructor and teaching assistant based on a skills profile. If necessary, you will have an opportunity to change teams.

As outlined below some deliverables are the joint responsibility of the team. Other deliverables are the responsibility of individual team members. For the individual deliverables, you should collaborate closely
and help each other, but the design and coding must be done individually by a designated lead developer. This person must be clearly identified in the submitted deliverables.

The objective of this project is to develop a detailed specification and a working prototype. To do this, you will be asked to prepare a set of deliverables (see below).

**DELIVERABLES**

As it stands, the Social Bookmarking Systems is a large ill-defined project. It is up to you to create a design and to implement a solution.

To help you manage its complexity a set of deliverables and a timeline is proposed below.

**Preliminaries**

You will submit three reports:

- Report #1: Working vision report (week #4)
- Report #2: Draft report (week #8)
- Report #3: Final report (week #10)

The audience for the final report is a technical project manager; that is, someone with a background in information system design who plans and manages projects. This person will appreciate concise, clear, interesting writing and seeks to understand what you did and why you did it. In addition, this person will appreciate innovative ideas, critical discussion about their merits, and realistic plans for evaluating them.

You should create a simple website to post your project reports and any other necessary materials. Your website, of course, should link to the Social Bookmarking System prototype.

The front door of your website should include the team name, team members, the course name, course date, and so on. You should invent an appropriate name for your prototype.

For a degree of privacy, you may select an obscure URL for your website. You will give this URL to the teaching assistant and instructor so they can grade your work.

**Report #1: Working vision report**

The aim of report #1 is to start thinking about the project! The first report should consist of the following sections:

1. **Title Page**  
   Name of project, list of team members, etc.

2. **Scenario** (< 1 page)  
   A brief description of who the users of SBS are and what tasks they will complete (< 1 page)

3. **User Interface Sketch**  
   A sketch the major screens and interaction flow for your system. You should sketch the user interface with pencil on blank paper sheets OR sketch a design on a whiteboard and take images of your design OR sketch a design on Table PC OR sketch a design with a simple drawing package. The goal is quickly (and easily) specify your design clearly but not in great detail. High fidelity drawings are not desirable at this stage – clear thinking is.
4. **Conceptual model** (< 3 pages)
   A list of entities, attributes and relationships that your system will need to represent in order to allow you to implement the user interface. You are welcome to use an Entity-Relationship Diagram for this but it is not necessary. **Please note:** This is a draft and you are free to change at anytime during the quarter.

5. **Questions**
   A list of questions that you have about this project – we will discuss these questions in class.

### Structure of Report (Report #2 is a revision of Report #1)

- **Title Page** (~ 1 page)
  Project name ● Team members ● System URL ● etc.

- **Executive Summary** (~ 1 page)
  Concise summary of the project, indicating the vision, key technical components, and current status

- **Table of Contents** (~ 1 page)
  Present a table of contents for the final report

1. **Introduction** (~2 pages)
   Please present a brief introduction to the Social Bookmarking System. The introduction should orient the reader to the report and describe what the SBS is, why you are working on it, and so on.

   1.1 **Vision**
   The vision should briefly describe who the users are and what they will do with the system. You can select one of the above scenarios and elaborate it with your own ideas or you can propose a new scenario that is consistent with the root concept of social bookmarking.

   1.2 **Functional requirements**
   Include a list of functional requirements for your system. Functional requirements are terse statements of what the system will do but not how.

   They take the following form: *The system will have the ability to ...*

2. **Conceptual Model** (1 page for a diagram PLUS 1 page for description)
   Please present the conceptual model. The conceptual model describes the entities, attributes, and relationships of the system independent of the target implementation platform.

   You **must** use Enhanced Entity-Relationship modeling and UML notation. In addition to the diagram, this section must include a brief description of the key features of the model.

3. **Logical Model** (1 page for a diagram PLUS 1 page for description)
   Please present the logical model, where the conceptual model (section 2) is transformed into a set of entities, attributes, and relationships.

   In addition, your model (including description) must specify the following:
   a) The primary and foreign keys;
   b) The integrity constraints for the model;
   c) The parts of the overall model that you will implement.

   You **must** use UML for your logical model. Your logical model should consist of entities, and 1-1 and 1-M relationships only. It **must not** contain M-M relationships, complex relationships, and so on.

   In appendix A, you **must** include the SQL script used to define your database. The script should be neat and well commented.
4. **Information Retrieval Model** (~ 2 pages; final report only)
   Please present the IR model. This model consists of a description of the best-match searching that your system will offer. Referring back to the conceptual model, you should include a description of how you represent documents, what metadata they contain, the indexing policies used, and how data from the Information Retrieval module is kept consistent with the database module.

5. **System Architecture** (1 page for a diagram PLUS 1 page for description)
   Please clearly describe the system architecture used in your implementation. It should describe what these technologies are and how they fit together: JSP pages, HTTP requests and responses, Java application classes, PostgreSQL, Lucene, Tomcat web server, JDBC, web browsers, WAR files, and so on. You must use a three-tier model to present your system architecture.

6. **Implementation notes** (be concise and, please, no busy work)
   This section should include a description of your implementation. You should provide an overview of how you’ve implemented the various functions of your system. You should include screen shots of your system, describe strengths and weaknesses of your implementation, etc. After reading this section the reader should have a good understanding of the merits of your implementation. It is suggested that you organize your implementation into the following sections (although you may include different functional components):

   6.1 **Functions for phase #1**
      6.1.A Create user (<developer name>)
      6.1.B Create keyword (<developer name>)
      6.1.C Create item (<developer name>)

   6.2 **Functions for phase #2 (final report only)**
      6.2.A Browse by keyword (<developer name>)
      6.2.B Search by keyword (<developer name>)
      6.2.C Create project (<developer name>)

   6.3 **Functions #3 (final report only)**
      6.3.A Best-match search (<all>)

7. **Status and Next Steps** (~1 page; final report only)
   Summarize the state of the SBS by answering questions such questions as: 1) How effective is the design; 2) How much of the implementation is complete; 3) What needs to be revised; and 4) What would you do next if you were to continue with the project?

8. **SQL Build Script**
   In Appendix A, please include the SQL script that is used to build the initial version of the database.

9. **Team Reflection** (~1 page; final report only)
   In Appendix B, please include a brief statement of what you learned, how you helped each other, what worked, what you would do differently next time, and so on. You may write this a single group statement or three individual statements.

10. **Design tools and Process** (~1 page)
    In Appendix C, please describe how your team is working together. You should indicate: 1) How often you meet and where; 2) What communication tools you used; 3) What tools you used to coordinate and plan your work; 4) What documents you use to make progress; and 5) The challenges you experience when working together and how you overcome them.
GRADING GUIDELINES

General mechanics
- Please try to keep with the suggested page limits – conciseness and clarity are valued. If you need to go beyond the page limits, do so but have a good reason;
- If necessary, include other appendices (e.g., references, sketches of the user interface, etc.);
- All figures and tables must have captions;
- On the title page, please include an URL to the website;
- The code for the system should be readily available at the website;
- All sketches can be done with pencil and blank paper or whiteboard and then scanned into the document but they must be done neatly;
- If you find yourself doing busy work STOP; then, please talk to the teaching assistant or instructor. Everything in the three reports should help you make progress on your project. And, of course, if you have a better idea for the project or report format, please speak up.
- If you want to go beyond the general requirements do so but please weigh the benefits and manage the risks (and talk with the teaching assistant or instructor).

Teamwork
- All members of the team will receive the same grades for the first three components of the report;
- The forth component of the report may be graded individually;
- Teams are assigned by the instructor and teaching assistant – you will have an opportunity to suggest revisions to the team assignment (if necessary);
- If you experience difficulties working as a team, talk with the instructor or teaching assistant as soon as possible – we can help.

Values
Note: Please use Report #1 and Report #2 as an opportunity to get feedback. For the sections where you are uncertain, do your best and ask questions in writing. We will answer them! The key objective of the first two reports is to receive useful feedback for the final report.

General grading criteria:
- The design is elegant and simple;
- Decisions are made explicit and justified;
- UML notation is fully and rigorously used for the EER and ER diagrams;
- The writing is clear, concise, and interesting;
- All diagrams and sketches are neatly presented, with captions.

1. Overall 10%
- All report sections present;
- Absence of spelling and grammatical errors;
- Captions and legends explain diagrams and charts.

2. Front and Back 10%
- The project is introduced and concluded in an effective manner.

3. Information System Specification 20%
- Data models show entities, relationships, attributes, types, and so on;
- The overall architecture is presented in an effective manner.

4. Implementation 60%
1. The modules that make up your implementation are clearly outlined;
2. The modules are described and their merits – strengths and weaknesses – are discussed;
3. The user interface is included in the report and annotated.
About The Labs: FAQ

Why do we have a lab?
1. To learn and practice the skills needed to develop database applications;
2. To provide a weekly place for discussing the class project;
3. To provide a supportive learning environment where we can help each other – often, coding problems and solutions are best discussed in a group.

Are labs graded?
No – individual labs are not graded. However, your work in the labs contributes to your participation grade for the class. More to the point, you will not be able to complete the project without completing the labs.

We will take attendance at labs so we have a record of who is participating. And periodically throughout the quarter we will check your lab webpage (approximately week #3, #6 and #9).

These reviews and your attendance at the labs will contribute to your participation grade (which is worth 10%).

What's the lab webpage?
Each of the labs requires that you complete some work – often, a coding problem; sometimes, a brief writing assignment. You will post your solutions at your lab webpage.

Your lab webpage is located here:
http://linux.ischool.washington.edu:8180/~your_user_name/index.jsp

Use any design you want for your lab webpage. Here is a suggestion:

LABS for MARY SMITH
INFO-340: Database Management and Information Retrieval
Winter 2007

<brief introduction>

Lab 01
Lab 02

Clicking on the link Lab 01 will take you to the solutions for the first lab, etc.

You should assume that the audience for your lab webpage is a prospective employer who would like to judge the quality of your work – neatness, clarity, and conciseness are the thus important.
Learning objectives

- To learn how to edit, compile, debug, and deploy a JSP web application
- To learn how to embed JSP code in HTML
- To learn the basics of HTTP requests, responses, and headers
- To learn how to inspect the HTTP query string to build dynamic web page

Introduction

At the completion of this lab you will have created a JSP Web Application project that will contain all of your lab webpage. The JSP application will be deployed on this server:

```
linux.ischool.washington.edu
```

which is a Linux server running Tomcat and PostgreSQL. Tomcat is an open source web server for running JSP applications. The official home page for Tomcat:

```
http://tomcat.apache.org
```

Once you complete this lab, you will be able to run your JSP application at the following URL:

```
http://linux.ischool.washington.edu:8180/~username/
```

where `username` is your iSchool User Name. Tomcat usually runs on port 8180, but may use a different port on other servers. For INFO-340, it will always run on port 8180.

This lab consists of two milestones:

1. Setting up a NetBeans project
2. Passing parameters with JSP.

Lab 02 is a continuation of this lab. If you finish lab 01, move on to lab 02. Then, you are ready to start implementing Assignment 1.

You will find example JSP applications that will help you through labs 01 and 02 here:

Click on this link Programming Examples
Click on this link Java Server Pages
**Milestone #1: Setting up a NetBeans project**

To complete this milestone, please go to the *NetBeans Introduction* and go through the steps. The NetBeans Introduction is located here:

- Click on this link: Programming Examples
- Click on this link: NetBeans Introduction

The NetBeans Introduction will take you through all the steps that are needed to create and deploy a simple JSP application.

**Milestone #2: Passing parameters with JSP**

It is often necessary to pass parameters to web pages and to process the values that are associated with the parameters. For example, we might need to pass the ID of an Item and have different items displayed. In example below the JSP page, showItem, will show different items depending on the ID of the item:

```
http://linux.ischool.washington.edu:8180/~username/showItem.jsp?Item=1
```

Read about HTTP requests and responses:

- Click on this link: Programming Examples
- Click on this link: HyperText Transfer Protocol

Create a JSP page called showItem.jsp that takes one parameter, called `Item`, and displays the value of the parameter on the page.

```
~ Continue to Lab 02 ~
```

**What to submit?**

Post solutions to Milestone #1 and Milestone #2 on your lab webpage.
NetBeans is an Open Source development environment that is ideally suited to web development applications. It is written in Java and is backed by Sun.

NetBeans provides capabilities similar to MS Visual Studio and Eclipse. Some major features:

- A full-featured source editor and debugger
- Tomcat server built into the IDE
- Runtime support for accessing JDBC compliant databases.

In short, NetBeans is a professional development environment for creating dynamic, data-driven web applications.

Quick Start

Please Note: It is important that you login under a LABS domain rather than an ISCHOOL Domain.

The following is a tour of some of the Major features of NetBeans. Follow these steps to learn how to code, compile, debug, and deploy a web application.

1. Start NetBeans IDE
   1. Find NetBeans IDE in start > All Programs > Programming and Development > NetBeans 5.5 > NetBeans IDE

2. Create a Web Application Project

   You can use NetBeans to create different kinds of projects. In INFO-340 we will be concerned with Web Application Projects only.

   A Web Application Project contains all the files and directories that are needed by the project. A NetBeans project organizes your development work so that you can code, test, debug, and deploy web applications efficiently.

   NetBeans may at first appear very complex. In time, however, you will find it to be a very powerful tool for coding Web Applications.

   To create a Web Application Project, follow these steps:

   1. Select: File > New Project
   2. In Categories column, select: Web
   3. In Projects column, select Web Application
4. Click Next > (at bottom of dialog)
5. For Project Name, type HelloWorld
6. Change Project Location to your home directory (located on the H: drive)
7. Unclick the checkbox "Set Source Level 1.4"
8. Check that the checkbox "Set as Main Project" is selected
9. Check Finish (at bottom of dialog)

3. **Create a JSP program**

Java Server Pages (JSP) is programming system for making HTML pages dynamic. With JSP, you embed Java within the HTML code for a page. To create a JSP, follow these steps:

1. In the top-left, you should see three tabs: Projects, Files and Runtime
2. Click on the Files tab
3. Click on the HelloWorld project to expand this directory. You should see the following directories:
   1. nproject
   2. src (.java source files are located in src/java)
   3. test
   4. web (.html and .jsp pages are located here)
4. Click on the folder nameedd "web" -- jsp files are located in this folder and its subfolders
5. Click on index.jsp
6. Change the file so that it contains the following HTML code:

   ```html
   <%@page contentType="text/html"%>
   <%@page pageEncoding="UTF-8"%>
   <html>
   <head>
     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
     <title>INFO-340 Labs for _your_name_</title>
   </head>
   <body>
   <h1>INFO-340 Labs</h1>
   <ol>
     <li><a href="./Lab01/mylab01.jsp">Lab #1</a></li>
   </ol>
   </body>
   </html>
   
   7. To see this HTML file a browser:
      1. Right click on index.html
      2. Select Run File

   This will: 1) Start the Tomcat webserver; and 2) Bring up a webbrowser on the page just created.

8. If you click on the link, you will notice that it is broken. To fix it, you must add a directory to the web application. To add a directory:
   1. Right click on the folder web
   2. Select New > Folder
   3. Type "Lab01" for the folder name
   4. Click the button Finish

9. Now, you must add a file to the directory:
1. Select the new folder Lab01
2. Right click the folder
3. Select New > JSP file
4. Type "mylab01" for the name of the file (The file extension should be .jsp)
5. Click the button Finish
6. Type in some HTML, such as:

```html
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Lab 01</title>
</head>
<body>
<h1>Hello this is my Lab 01</h1>
</body>
</html>
```

10. Now, test that the link works by right clicking on web/index.jsp and selecting Run File. Using just these features you can create complex websites.

11. Let's turn to a JSP example. Type the following code into the jsp file in the Lab01 folder, web/Lab01/mylab01.jsp
(Please note: this program has a small error, which will be corrected in the next step):

```html
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page language="java" import="java.util.*"%>
<html>
<head>
    <title>Hello</title>
</head>
<body>
<p>Here's a message for you: <font size="+3" color="red"> <%= msg %> </font> !!</p>
</body>
</html>
```

4. **Compile your file and build your project**
   1. Select the file mylab01.jsp
   2. Select Build > Compile "mylab01.jsp"
   3. You will see a syntax error -- fix the error
   4. Try to recompile your program and view it in the browser (Run)
   5. Once your program compiles correctly, go to the main menu and select Build > Clean and Build Main Project

5. **Run your program**
1. Select Run > Run Main Project
2. The Tomcat web server will start, and the output of the JSP page will appear in a browser window. Congratulations -- You've written, compiled, and run a JSP program!

6. Deploying a JSP Application

Once you have a working Web Application on your client development machine, the next natural step is to deploy it to a server machine. In order to complete this step, you must have a linux.ischool.washington.edu server account. These accounts will be assigned in the first lab.

Here are the steps for deploying a web application:

1. Select the Projects pane
2. Select your project "HelloWorld"
3. Select Build > Clean and Build Main Project
4. Notice that a file called HelloWorld.war is created in the folder: "dist", which stands for "distribution". The file "HelloWorld.war" is a Web Archive file that can be easily deployed on servers. To see how, follow these steps:
   1. Open ssh and login to your server account. To login, you need:
      1. The Host Name is: linux.ischool.washington.edu
      2. Your User Name is your iSchool User ID
      3. Your password is your iSchool password
   
To open SSH:

   1. Click on SSH Secure File Transfer
   2. Click on Profiles
   3. Click on Edit Profile
   4. General > Defaults
   5. Click on Add and select Keyboard-interactive
   6. Now click on Quick Connect
   7. Type the Host Name and User Name
   (NOTE: You should only have to edit the profile once.)

2. Find the WAR file -- it should be located in H:\\HelloWorld\dist\HelloWorld.war
3. Copy the WAR file to a working directory on the linux machine, say /homes/username/working
4. Copy the file from the working directory to the webapps directory, which is /homes/username/public_webapps
5. Open a new terminal window at this location: /homes/username/public_webapps
6. Extract the WAR file by typing the following command:
   - jar -xvf filename.war
7. To try out the application, browse to http://linux.ischool.washington.edu:8180/~username/index.jsp
8. Be sure to click the refresh button

Congratulations! You've successfully deployed an application

5. Testing your application

Even though you have tested your project locally, it's important to test the project thoroughly once it's been uploaded to the server, as a different environment can introduce new problems.

   1. To view the application you uploaded, browse to
Congratulations! You've successfully deployed an application.

7. Adding external libraries from Jar files

Often, it is necessary to add a jar file to your project. For example, when you use the Google API you will need to add a jar file. Here's how:

1. Right mouse click on your project name & click on Properties or click on File -> "Project Name" Properties
2. Select "Libraries"
3. Click "Add JAR/Folder"
4. Navigate to the location of googleapi.jar in the file system and select it
5. This will add the jar file to the list of resources that this project can access. Now, to use it inside your java source files (i.e. SomethingOrOther.java) you can add an import line (e.g. import com.google.soap.search.GoogleSearch or import com.google.soap.search.*) and in your code write: GoogleSearch s = new GoogleSearch();

To learn more about NetBeans, inspect the help information in the menu Help.
Learning objectives

- To learn how to edit, compile, debug, and deploy a JSP web application
- To learn how to embed JSP code in HTML
- To learn the basics of HTTP requests, responses, and headers
- To learn how to inspect the HTTP query string to build dynamic web pages

This lab is a continuation of lab 01.

Milestone #3: Conditionally display pages

Create a JSP page called showPage.jsp that displays four different pages depending on the value of a parameter called id:

http://linux.ischool.washington.edu:8180/~username/showPage.jsp?id=1

Allowable values for id are 0-3. Any value outside of this range should generate an error page. Your JSP application should allow you to navigate this little network of pages:

- Front door (id = 0) ➔ About Page (id = 1)
  ➔ Owners (id = 2)
  ➔ Introduction (id = 3)

Add the following links to each page: “top”, “next”, and “back”.

Milestone #4: Add a search box

Add a search box to the front door page so that people can type “about”, “owner”, or “introduction” and go directly to those pages.

What to submit?

Please post solutions for milestone #3 and milestone #4 to your lab webpage.
Learning Objectives

- To review the software infrastructure used in this class (tomcat, jsp, etc.) so that you can sketch a diagram of its major components and describe them
- To learn how to submit SQL queries to the PostgreSQL server from the command line, from NetBeans, and from pgAdmin
- To learn how to run PostgreSQL scripts to load and query databases
- To learn some basic SQL commands.

Resources

These links will be helpful for learning SQL and for learning about the PostgreSQL:

- Full documentation for PostgreSQL:
  [http://www.postgresql.org/docs/7.4/static/index.html](http://www.postgresql.org/docs/7.4/static/index.html)
- Tutorial for getting started with PostgreSQL:
  [http://www.postgresql.org/docs/7.4/static/tutorial.html](http://www.postgresql.org/docs/7.4/static/tutorial.html)
- The language:
  [http://www.postgresql.org/docs/7.4/static/sql.html](http://www.postgresql.org/docs/7.4/static/sql.html)
- Command reference:
  [http://www.postgresql.org/docs/7.4/static/reference.html](http://www.postgresql.org/docs/7.4/static/reference.html)

Getting Started with SQL and PostgreSQL

1) **psql command shell**: The psql command shell allows you to submit SQL queries for querying and updating databases. To use the shell, follow these steps:

   a. Use SSH to log into the iSchool Linux server;
   b. At the command line type `psql DB_NAME`, where DB_NAME is the name of the database that you will connect to (see server info handout);
   c. `psql` will prompt you for a password (see server info handout);
   d. You should now be in the psql command shell. You should change your PostgreSQL password now (see back of server info handout). In short, the SQL command is:

   ```sql
   alter user <USER_NAME> with password '<NEW_PASSWORD>';
   (don't forget the semi-colon; change labels between angle brackets; don't forget single quotes)
   ```
2) **Practice with the psql shell.** Using the psql shell, try out the following:
   a. Create a table for storing the *names* and *ages* for *Sailors*
   b. Make up some sailor names and ages, and insert three records into the table
   c. Write and execute an SQL query to select all sailors that are older than 30 years old

**Notes:**
1. The information you need for these tasks is available here:
   INFO-340 Course Page → Programming Examples → PostgreSQL → Quick Guide
   (Also attached at the end of this lab.)
2. The name attribute can be represented with the data type `varchar(64)` and the age attribute can be represented with the data type `integer`
3. For a complete list of data types available in PostgreSQL, see [http://www.postgresql.org/docs/7.4/static/datatype.html](http://www.postgresql.org/docs/7.4/static/datatype.html)

3) **Load data into tables with a psql script.**
   a. Examine the psql script that is located here:
      (Also attached to this lab.)
   b. Take this script and the data files and get it to work. Follow these steps:
      i. At the top of the script set the database name to your database
      ii. Create a directory in your server account
      iii. Copy the script to that directory
      iv. Copy the three data files (that is, `.txt` files and note that `tabs` are used to separate the data columns)
      v. Type the following command at the Linux command prompt
         `psql <DATABASENAME>  < mk_boat_club.psql`
      vi. Note: you will be prompted to enter your `database` password
   c. Investigate the CREATE VIEW command and create a view that contains only the sailors with a rating of 7 or above. Put this command at the bottom of your script.
      Hint: [http://www.postgresql.org/docs/7.4/static/reference.html](http://www.postgresql.org/docs/7.4/static/reference.html)
   d. Investigate the CREATE TABLE AS command and determine how it differs from the CREATE VIEW command.

4) **Connect to the database with NetBeans.**
   a. Install the PostgreSQL driver into NetBeans. The driver is located here:
      [http://jdbc.postgresql.org/download.html](http://jdbc.postgresql.org/download.html)
      **Download this version: pg74.216.jdbc3.jar**
      To install it, navigate to: Runtime tab, Databases, Drivers and then right click on Drivers. Then add the driver.
   b. Navigate to the PostgreSQL driver by clicking on the Runtime tab, then Databases, then Drivers
c. Now, right click on the PostgreSQL driver you just installed and select Connect using...
Configure the database connecting as follows:

```java
jdbc:postgresql://<HOST>:<PORT>/<DB>
HOST == linux.ischool.washington.edu
PORT == 5432
DB == your login name
```

The string should look something like:
```
jdbc:postgresql:// linux.ischool.washington.edu:5432/username
```

**Note:** NetBeans cannot directly connect to the database from off-campus. If you wish to connect to the database from outside the network, you must first log into the VPN. For instructions on connecting to the VPN, see the iSchool FAQ: [http://help.ischool.washington.edu/faqs/4_9_en.html](http://help.ischool.washington.edu/faqs/4_9_en.html).

d. Enter your `psql` user name (your login name) and password
e. Select the “public” schema and press OK
f. Expand the new connection and the folder called Tables. You should see the Sailors table that you created earlier.
g. Right click the folder Tables and select **Execute Command**. Type in some SQL commands such as select * from Sailor. Click the Run button to execute your queries.
h. You can use this GUI to execute SQL commands to databases.
i. You can navigate the folder to see tables and their attributes

Spend a few minutes viewing and editing your tables.

5) **Using pgAdmin**

Start pgAdmin which is available from the all programs menu of the lab machines. Connect to your database – how does it compare to the NetBeans GUI for issuing SQL queries.

Find pgAdmin at [www.pgadmin.org](http://www.pgadmin.org) (and it’s available on the lab computers’ start menu).

6) **Exporting your database**

Sometimes we might want to transfer our database from one server to another. We can do this by exporting our database file from PostgreSQL. The **pg_dump** utility allows us to convert an existing database into a sequence of SQL commands that can be used to recreate the database.

a. Return to the linux command prompt (not the psql prompt)
b. Enter the following command:

```
pg_dump DATABASE_NAME > lab2db.sql
```

the > character copies the output of pg_dump into a file named `lab2db.sql`. The file now contains the SQL needed to recreate the database.
c. You can now use the exported file to recreate your databases, as you did in Step 2. For instructions on how to export different types of data, type `man pg_dump` at the linux prompt.
What to submit?
On a lab 03 webpage, please answer these two questions:

1. Please post the script that you created in task #3 above.
2. Briefly explain the difference between the CREATE VIEW and CREATE TABLE AS command.
3. NetBeans SQL GUI vs. pgAdmin – which application is better? Why do you think so?
Quick Guide

You can interact with PostgreSQL through a very powerful command shell called psql. Often, you will find this to be a helpful way to debug SQL statements and to check up on your databases. psql is a client application, where you can run queries, inspect databases, and lots of other things.

To use psql, you must first secure shell to your server machine. Then, to learn about psql, try these commands:

To check the version of postgresSQL, type

```
# psql -V
```

(Note: Type this at the Linux command prompt.)

(If this command does not work, then postgresSQL is probably not available. Please contact the teaching assistant for help.)

To create a database use this command

```
# createdb <DATA_BASE_NAME>
```

(Note: Type this at the Linux command prompt.)

(This gives you a space for creating tables. Note: Replace <DATA_BASE_NAME> with your own name. If this command does not work, then something is might be wrong with the permissions. Please contact the teaching assistant for help.)

To remove a database use this command

```
# dropdb <DATA_BASE_NAME>
```

To log into PostgreSQL

```
# psql <DATA_BASE_NAME>
```

At the psql prompt

Note: =# is the command prompt

To leave the PostgreSQL command shell, type

```
=\q
```

For help with SQL commands, type

```
=\h
```

For help with psql shell commands (MANY), type

```
=\?
```

To see/edit your previous command, use 'up' arrow

Some key commands

To list all databases, type

```
=\l
```

To connect to a database, type

```
=\c <DATA_BASE_NAME>
```

To list the tables (relations) in a database, type

```
=\d /* you must be connected to a database */
```

To describe the properties of a table (relation), type

```
=\d <TABLE_NAME>
```
Some simple commands

To create a table, type

```sql
# create table Building(name varchar(20));
```

This command creates a table named Building. The name has a single attribute, named 'name'. The attribute 'name' is given the type `varchar(20)` which means that name can hold a variable length string up to 20 characters. Another common data type is `integer`.

To delete a table, type

```sql
# drop table <TABLE_NAME>
```

To add rows to the table, type

```sql
# insert into Building (name) values ('Mary Gates Hall');
# insert into Building (name) values ('Suzzallo Library');
```

To determine how many rows are in the table, type

```sql
# select count(*) from Building;
```

To select all rows from a table, type

```sql
# select * from Building;
```

To change your password, type

```sql
# alter user <USER_NAME> with password '<NEW_PASSWORD>';
```
/* filename: mk_boat_club.psql */

/* Set some key variables */
\set DB team12

/* Connect to your database */
\c :DB

/* Remove tables if they already exist */
DROP TABLE reservation;
DROP TABLE sailor;
DROP TABLE boat;
DROP DOMAIN SailorRating;
DROP DOMAIN BoatColor;

create domain SailorRating as integer
    check (value between 1 and 15);

create domain BoatColor as varchar(6)
    check (value in ('green', 'red', 'blue', 'white'));

/* Create the three tables */
create table sailor(
    id integer PRIMARY KEY,
    sname varchar(40),
    rating SailorRating,
    age real
);

create table boat(
    id integer PRIMARY KEY,
    name varchar(40),
    color BoatColor
);

create table reservation(
    sid integer references sailor(id),
    bid integer references boat(id),
    rdate date,
    PRIMARY KEY(sid, bid, rdate)
);

/* Now, load the tables with data */
\copy sailor from 'sailor.txt'
\copy boat from 'boat.txt'
\copy reservation from 'reservation.txt'

/* Now, run a couple of queries */
select * from sailor;
select * from boat;
select * from reservation;

/* Now, describe the tables */
\d sailor
\d boat
\d reservation
Learning Objectives

- To introduce JDBC, including connections, statements, and result sets.
- Create, debug and deploy a simple JSP application that uses JDBC.

Background

JDBC is a Java Application Programming Interface that enables applications programs to send SQL statements to a relational database and process results. JDBC is a common standard many relational database systems (e.g., Access, MySQL, Postgresql) provide support for JDBC.

JDBC can be used in a two-tier and three-tier models. Your Social Bookmarking System will be structured in a three-tier model as outlined here:

Client application  ←  JSP/Java Application  ←  PostgreSql
[Front-end]  http  [Middle Tier]  JDBC  [Database]

Resources

See chapter 29 in Connolly & Begg, 4th Edition for a brief introduction to JDBC.

For an overview

API documentation
http://java.sun.com/j2se/1.3/docs/api/java/sql/package-summary.html

Using JDBC

JDBC allows you to do three major things:

1. Establish a connection with the database;
2. Submit SQL statements (SELECT, CREATE TABLE, INSERT, UPDATE, etc.) to the database;
3. Receive and process the results (e.g., return results of a SELECT statement). that you can send

Activities

Please Note: To complete this lab you must have 1) finished lab 03 and 2) loaded the data for Assignment #2 into your database.

1) Navigate to the sample code for lab 04 on the course website. Download the code into your IDE. (The code is also included at the end of this assignment sheet.)

   a. Study the Java and JSP code for this example and answer these questions:
      i. What does this application do?
      ii. What do you think the classes Statement and ResultSet are for?
      iii. What do the methods, next() and getString(1), do?
2) Create a new Web Application by following these general steps:
   a. Create a new Web application
   b. Create a package the Java file (uw.ischool.info340.studentname.) and copy the Java file into
      that directory
   c. Edit the file so that it uses your database on your server
   d. Copy the JSP file into your web application and update the package path and any other
      information
   e. Compile, run and debug your application

3) Examine the Java Documentation for the following two classes:
   a. Statement
   b. ResultSet
   And determine what these classes do.

4) Modify your Web Application so that it prompts people to enter a string, either a state code (e.g., WA)
   or a state name (e.g., Washington). If a state code is entered, then you should return the state
   name. If a state name is entered, you should return the state code; otherwise, you should return an
   error.

What to submit?
On a lab 03 webpage, please:
   1. Answer the questions from activity #3;
   2. Please include a link to your Web Application from activity #4.
C:\Documents and Settings\dhendry\My
Documents\_Code\java_apps\_working\INFO340\src\uw\ischool\info340\labs\DbConnect.java

1 /*
2 * DbConnect.java
3 *
4 * Created on December 7, 2004, 7:52 AM
5 */
6 package uw.ischool.info340.labs;
7
8 import javax.naming.*;
9 import javax.sql.*;
10 import java.sql.*;
11
12 /*
13 * The program demonstrates how to connect to a postgresql database
14 * using JDBC.
15 *
16 * The following things must already be in place
17 *
18 * (1)
19 * You must know the name of the machine that is running postgresql
20 * (e.g., serv55.ischool.washington.edu)
21 * The machine must up and postgres must be running.
22 *
23 * (2)
24 * You must have a postgresql user ID and password and have been
25 * assigned a database.
26 *
27 * (3)
28 * You must have the following relation in the database
29 *
30 * Table "public.state"
31 * Column |         Type          | Modifiers
32 *  --------+-----------------------+-----------
33  * sname  | character varying(40) | not null
34  * scode  | character(2)          | not null
35  *
36 */
37
38 public class DbConnect
39 {
40 /*
41 * This is the location/URL of the database. It conforms to this connection string:
42 * jdbc:postgresql://<HOST>:<PORT>/<DB>
43 *
44 * Notes:
45 * a) team12 == is the name of the database
46 * b) 5432 is the port number -- it is optional
47 * c) serv55.ischool.washington.edu -- is the name of the host
48 */
49 private static String cDB_URL_STRING = "jdbc:postgresql://linux.ischool.washington.edu:5432/dhendry";
50
51 /*
52 * NOTE: Check that the user ID and password are correct
53 *
54 */
55 private static String cUserID = "<YOUR USER ID>";
56 private static String cUserPassword = "<YOUR DATABASE PASSWORD>";
57
58 /* This is the name of the JDBC driver for using postgresql */
59 private static String cDRIVER_STRING = "org.postgresql.Driver";
60
61 /* Holds the connection to database -- created by init() */
/* By default, transactions are auto-committed */
private Connection conn = null;

/* A variable for holding a state code -- normally set by client */
private String stateCode = "WA";

/* A variable for holding a state name -- set by query to database */
private String stateName = "unknown";

/* A variable for holding the answer to an SQL query*/
private String answer = "no answer";

/* A variable for holding the query*/
private String query = "no query";

private String debugString = "";

/* Returns information about the database connection */
public String getConnInfo()
{
    String t = "[Driver: " + cDRIVER_STRING + ";";
    t += "DB URL: " + cDB_URL_STRING + ";";
    t += "UserID: " + cUserID + ";";
    t += "Password: " + cUserPassword + ";";
    t += "]<p>" + debugString;
    return t;
}

/* Returns information about the connection */
public String connectInfo()
{
    if (conn != null)
        return "Connection up! " + getConnInfo();
    else
        return "Connection DOWN " + getConnInfo();
}

public boolean isConnected()
{
    return (conn == null) ? false : true;
}

/* Return the answer and query strings */
public String getAnswer() { return answer; }
public String getQuery() { return query; }

/* Setting and returning the value of state */
public String getStateCode()
{ return stateCode; }
public void setStateCode(String t)
{ stateCode = t; }

public String getStateName()
{ return stateName; }

/* Make a simple query based on parameter which should be a state code */
private String mkQuery(String stateCode)
{ String whereClause = "WHERE scode = " + stateCode + ";";
    return "SELECT sname FROM state " + whereClause;
}

/* Initializes the connection */
public void init()
{ try {
    /* This statement implicitly loads the driver */
    Class.forName(cDRIVER_STRING);
    /* Now, attempt to create a connection */
    conn = DriverManager.getConnection(cDB_URL_STRING, cUserID, cUserPassword);
}
if (conn == null) 
    throw new Exception("Could not connect to " + getConnInfo());
} 

} 

try { 
    if(conn != null) { 
        /* Create the SQL query that we would like to run */ 
        query = mkQuery(stateCode); 
        /* The class Statement is Used to send SQL statements to the database */ 
        Statement stmt = conn.createStatement(); 
        /* The class ResultSet is to hold the results of queries */ 
        ResultSet results; 
        /* We execute a statement with the following method */ 
        results = stmt.executeQuery(query); 
        /* Advance to the first record and extract value */ 
        if(results.next()) { 
            /* Retrieve the state name from the record set (a string in field #1) */ 
            stateName = results.getString(1); 
        } else { 
            stateName = "Not found"; 
        } 
    } 
    catch(Exception e) { 
        debugString = e.toString(); 
        e.printStackTrace(); 
    } 
}
<%@ page contentType="text/html"%>
<html>
<head><title>Simple JDBC Connection</title></head>
<body>

>> <a href="..">Up</a>
<h1>Simple JDBC Connection Example</h1>
<p>

As usual, import the classes in util (e.g., Date, String, etc.) --%>
<%@ page import="java.util.*"%>

Import the Java class that handles the database access logic --%>
<%@ page import="uw.ischool.info340.labs.DbConnect"%>

String sCode = "MA";

/* Initialize the class that manages the database */
DbConnect db = new DbConnect();
db.init();

/* Check if the connection is up */
if (db.isConnected()) {
    out.print("The database is connected ... nice!\n");
}
else {
    out.print("The database is NOT connected\n");
}

String info = db.getConnInfo();

Connection information
<pre>
<%=
info%
</pre>

/* Now execute the query */
db.setStateCode(sCode);
db.runQuery();

The query is: <%= sCode %>.

The result is: <%= db.getStateName() %>
</html>
Learning Objectives

- Implement a JDBC application that inserts records into a table, queries a table, and presents multi-line results.

Functional Requirements

Please Note: To complete this lab you should first finish lab 04 and Assignment #2.

The goal is write a JDBC application that allows a person to:

1. List all the state codes, longitudes, and latitudes for a particular city
2. Insert information for a new location, including:
   - zip, state code, city name, longitude, and latitude

To accomplish this, you will need to query the table zip.

Approach

Take the web application that you developed in lab04 and add a new class called ZipCode. It is suggested that you add the following methods to this class:

```java
boolean addNewLocation(String zip, String stateCode, String cityName, double longitude, double latitude)

// This function should add a record to the table zip. If an error occurs, you should detect it
// and return false; otherwise, return true.

What should you do if the zip code already exists?
```

```java
int getZipCodeInfo(String cityName);

// This function runs the query and returns the number of items found.
```

```java
String getDataField(int recordID, String fieldName);

// This function returns the data associated with the record ID and the field name for the
// previously run query.
```

Getting started

1. Study the following class files on the course website:
   - DbBaseConnect.java
   - ZipCode.java
2. Provide implementations for the above methods.
3. Create a jsp page to test your methods.

What to submit?

On a lab 05 webpage, please:

1. Post a link to a jsp page that demonstrates the above capabilities.
<?xml version="1.0" encoding="UTF-8"?>
<%@ page contentType="text/html"%>
<%@ page pageEncoding="UTF-8"%>
<%-- zipcode.jsp --%>
<%-- Generates an example document representing the results of an SQL query --%>
<%-- As usual, import the classes in util (e.g., Date, String, etc.) --%>
<%@ page import="java.util.*"%>
<%-- This class creates a db connection --%>
<%@ page import="uw.ischool.info340.labs.DbBaseConnect"%>
<%-- This class does the work -- it extends the class DbBaseConnect --%>
<%@ page import="uw.ischool.info340.labs.ZipCode"%>
<xml comment="An example document generated by jsp">
/* Initialize the class that manages the database */
ZipCode db = new ZipCode();
db.init();
/* Get the results in XML and put it into the output stream */
String t = db.getZipCodesForCityAsXML("Boston");
%</xml>
<xml version="1.0" encoding="UTF-8"/>
<xml comment="An example document generated by jsp">
<results num='17' city='Boston'>
    <item>
        <zip>2210</zip>
        <scode>MA</scode>
        <city>BOSTON</city>
        <longitude>71.0465</longitude>
        <latitude>42.3489</latitude>
    </item>
    <item>
        <zip>22713</zip>
        <scode>VA</scode>
        <city>BOSTON</city>
        <longitude>78.1423</longitude>
        <latitude>38.5382</latitude>
    </item>
    <item>
        <zip>14025</zip>
        <scode>NY</scode>
        <city>BOSTON</city>
        <longitude>78.7391</longitude>
        <latitude>42.6314</latitude>
    </item>
    <item>
        <zip>2199</zip>
        <scode>MA</scode>
        <city>BOSTON</city>
        <longitude>71.0825</longitude>
        <latitude>42.3479</latitude>
    </item>
</results>
</xml>
public class DbBaseConnect {

   /* CHECK that this connection string is right
   * a) dhendry == is the name of the database
   * b) 5432 is the port number -- it is optional
   * c) linux.ischool.washington.edu -- is the name of the host
   */
   private static String cDB_URL_STRING =
         "jdbc:postgresql://linux.ischool.washington.edu:5432/dhendry";

   /* NOTE: Check that the user ID and password are correct */
   private static String cUserID = "dhendry";
   private static String cUserPassword = "bluemonster";

   /* This is the name of the JDBC driver for using postgresql */
   private static String cDRIVER_STRING = "org.postgresql.Driver";

   /* Holds the connection to database -- created by init() */
   /* By default, transactions are auto-committed */
   /* NOTE: Subclasses will need to use this BUT not clients */
   protected Connection conn = null;

   /* Used to hold error messages */
   protected String debugString;

   /* Returns of a string containing basic information about the connection */
   private String _getConnInfo() {
      String t = "["
         + "Driver:" + cDRIVER_STRING + ";"
         + "DB URL:" + cDB_URL_STRING + ";"
         + "UserID:" + cUserID + ";"
         + "Password:" + cUserPassword + ";"
         + "]"><p>" + debugString;
      return t;
   }

   /* Returns information about the connection */
   public String getConnectInfo() {
      if (conn != null)
         return "Connection up! " + _getConnInfo();
      else
         return "Connection DOWN " + _getConnInfo();
   }
}
/**
 * Return true if the connection has been initialized
 */

public boolean isConnected() {
    return (conn == null) ? false : true;
}

/*
 * Subclasses MUST call this to initialize the connection
 */

public void init() {
    // Already initialized
    if (conn != null) {
        return;
    }

    try {
        /* This statement implicitly loads the driver */
        Class.forName(cDRIVER_STRING);

        /* Now, attempt to create a connection */
        conn = DriverManager.getConnection(cDB_URL_STRING, cUserID, cUserPassword);

        if (conn == null)
            throw new Exception("Could not connect to " + getConnectInfo());
    } catch (Exception e) {
        debugString = e.toString();
        e.printStackTrace();
    }
}
package uw.ischool.info340.labs;

import javax.naming.*;
import javax.sql.*;
import java.sql.*;

public class ZipCode extends DbBaseConnect {

    public ZipCode() {
        /* Call init() in the superclass to initialize the connection */
        init();
    }

    /* Adds a new location of the zip table */
    public boolean addNewLocation(String zip, String stateCode, String cityName, double longitude, double latitude) {
        // To DO
        return false;
    }

    /* Queries the zip table for all rows that match the cityName */
    public int queryZipCodeInfo(String cityName) {
        // TO DO
        return 0;
    }

    /* Gets the value of a particular row # and field name */
    public String getDataField(int recordID, String fieldName) {
        // TO DO
        return "";
    }

    /* Returns an XML document containing all state codes, longitudes and latitudes for a particular city */
    public String getZipCodesForCityAsXML(String cityName) {
        String theQuery = "";
        try {
            /* Build up query */
            String p1 = "select zcode, scode, city, longitude, latitude from zip ";
            String p2 = "where city=upper('" + cityName + ")\n"");
            theQuery = p1 + p2;

            /* Execute query -- the results are in r */
            Statement s = conn.createStatement();
        }
ResultSet r = s.executeQuery(theQuery);

/*
* We will keep track of the number of items processed
*/
int count=0;

/*
* We will put the XML source into this variable
*/
String t="";

/*
* A 'cursor' is used to move through results -- initially,
* the cursor is set BEFORE the first row in the result set.
* While next() returns TRUE we have a row to process
*/
while(r.next()) {
    /*
     * Query generated a result -- now extract data by Name & Index
     */
    String zcode = r.getString("zcode");
    String scode = r.getString("scode"); // by column name
    String city = r.getString("city");
    double longn = r.getDouble(4); // by index -- 4th column
    double latit = r.getDouble("latitude");

    /*
     * Create XML fragment and concatenate it to XML string
     */
    t += "  <item>
    <zip>" + zcode + "</zip>
    <scode>" + scode + "</scode>
    <city>" + city + "</city>
    <longitude>" + longn + "</longitude>
    <latitude>" + latit + "</latitude>
   </item>
    count++;
}

/*
* Check for results/no-results and build xml string
*/
if (count > 0) {
    t = "<results num='${count}' city='${cityName}'>" + t;
    t += "</results>";
} else {
    t = "<noresults city='${cityName}'>";
    t += "  <query>" + theQuery + "</query>";
    t += "  <reason>Unknown city</reason>";
    t += "</noresults>";
}

/*
* Return the XML document of results
*/
return t;

} catch(Exception e) {
    debugString = e.toString() + "Query: ||" + theQuery + "||";
    e.printStackTrace();
}

/*
* Return an ERROR tag -- if there's an exception send this back
*/
return "<ERROR function='getZipCodesForCityAsXML'>" + debugString
+ "</ERROR>";
}
Learning Objectives

- Describe the basic Lucene API including the following Lucene classes: Field, Document, IndexWriter, IndexReader, Analyzer, Query, and Hits
- Implement a simple best-match search system using Lucene for indexing and search
- Use the Luke tool to inspect an index file created by Lucene.

Overview

Lucene is a full featured search engine. It's written in Java and is an Open Source project. You will use Lucene to implement a best-match search capability for your Social Bookmarking Project.

The big question you will encounter is: How can best-match search and exact-match search be combined to enable effective user interactions?

This lab focuses on the basics of the Lucene API.

Resources

- The Lucene home page
  http://lucene.apache.org/java/docs/
- The Lucene API documentation:
  http://lucene.apache.org/java/docs/api/index.html
- Luke: Lucene Index Toolbox
  http://www.getopt.org/luke/
- Lucene query syntax
  http://lucene.apache.org/java/docs/queryparsersyntax.html

Step 1: Getting oriented

Take a look at the Lucene API. The API has many classes. Fortunately, we only need to interact with a small subset of these.

Take a look at the documentation for the following classes: Document, IndexWriter, IndexReader, StandardAnalyzer, Hits, Query, QueryParser. Where applicable, look at the superclasses and subclasses of these classes.

Step 2: Setting up the workspace

You’ll need to set up a few items before you can work with Lucene.

1. Lucene library file. On the course home page you’ll find the file lucene-core-2.0.0.jar. Download it and add it to a NetBeans project. This JAR file and other source files can be found at the Lucene Home page.

Step 3: Download the source

From the course home page, download the source for the lab. You’ll find several java and jsp files. Add them to your project and get the application working.

Step 4: Using Luke

Open Luke by double-clicking the JAR file. Luke will prompt you to choose a Lucene index file. A Lucene index is simply a folder containing a number of flat files. Choose the folder that was created in Step 3.

Use Luke to examine the Lucene index. What information can you uncover about the index using Luke?

Step 5: Your Social Bookmarking System

Working in project teams, discuss the following two questions:

1. For your project, what search tasks could be supported with Lucene?
2. How might your system be architected to use Lucene?

What to submit?

Please submit the following on your lab page:

1. Post a link to a working version of the code?
2. Write a brief description of what Luke can be used for?
3. Based on discussions in step #5, how do you anticipate using Lucene in your project?
C:\Documents and Settings\dhendry\My
Documents\_Teaching\_INFO-340\_Winter2007\labs\lab07\LuceneSearch\web\create_index.jsp

```html
<%@ page contentType="text/html" %>
<%@ page pageEncoding="UTF-8" %>
<html>
<%-- create_index.jsp --%>
<%-- Used to create a Lucene index --%>
<head><title>Create Lucene Index</title></head>
<body>
<h1>Simple JDBC Connection Example</h1>
<p>
/* Initialize the class that manages the database */
LabIndexer I = new LabIndexer();
String m = I.createLuceneIndex();
Message <br>
<pre>
<%=m %>
</pre>
</html>
```
<%@ page contentType="text/html"%>
<html>

<!-- create_index.jsp -->

<!-- Used to create a Lucene index -->
<head><title>Lucene Search</title></head>
<body>

<h1>Lucene Search</h1>
<p>

/* Initialize the class that manages the database */
LabSearcher S = new LabSearcher();
int num_hits = S.search("skiing beach");
if (num_hits == 0) {
    out.println("<p>No hits found!<p>");
} else {
    out.println("<p>Number of hits: "+ num_hits);
    out.println("<ol>");
    for (int k=0; k < num_hits; k++) {
        String n = S.getDataField(k,SearchBase.cUserNameField);
        String d = S.getDataField(k,SearchBase.cUserDescriptionField);
        out.println("<li> <b>"+n+"</b>: <i>"+d+"</i></li>");
    }
    out.println("</ol>");
    out.println("Where do the nulls come from? What is needed to fix this problem?");
}

</body>
</html>
/*
 * UserProfile.java
 */

package edu.uw.info340.lucene;

/**
 * We shall assume that we are to index a set of User Profiles -- A User Profile
 * is a kind of 'document' in that it has, as you can see, four fields:
 * a) ID to uniquely identify the profile
 * b) A name
 * c) A list of interests (just a bag of words)
 * d) An age.
 */

class UserProfile {
    int _id;
    String _uname;
    String _interests;
    int _age;

    /*
     * A constructor for assigning values to fields
     */
    UserProfile(int id, String n, int a, String i) {
        _id = id;
        _uname = n;
        _interests = i;
        _age = a;
    }

    /*
     * Selectors for access to the fields
     */
    int getId() { return _id; }
    String getName() { return _uname; }
    String getInterests() { return _interests; }
    int getAge() { return _age; }
}
package edu.uw.info340.lucene;

import java.util.*;
import java.io.*;

/**
 * Provides global needed by the indexing and search process
 */

public class SearchBase {

    private SearchBase() {
    }

    /**
     * Indexes that Lucene creates are stored in a directory. This is the directory name
     * that will be used -- CHANGE THIS TO AVOID COLLISIONS WITH OTHER STUDENTS
     */

    private static final String cDIR_NAME = "DGH_INDEX_DIR";

    /**
     * Returns the directory path for the index
     */

    public static final String getDirectoryPath() {
        return System.getProperty("java.io.tmpdir") + File.separator + cDIR_NAME;
    }

    public static final String cUserIDField           = "uid";
    public static final String cUserNameField         = "uname";
    public static final String cUserDescriptionField  = "descr";
}
LabIndexer.java

C:\Documents and Settings\dhendry\My
Documents\Teaching\INFO-340\Winter2007\labs\lab07\LuceneSearch\src\java\edu\uw\info340\lucene\LabIndexer.java

1 /*
2 * LabIndexer.java
3 *
4 * Demonstrates how to index a set of documents.
5 * D. Hendry (02/14/2007) based on code written by S. Kane (2006).
6 */
7
8 package edu.uw.info340.lucene;
9
10 import java.util.*;
11 import java.io.*;
13 import org.apache.lucene.document.Field;
14 import org.apache.lucene.analysis.standard.StandardAnalyzer;
15 import org.apache.lucene.index.IndexWriter;
16
17 /**
18 * This class shows you how to create a Lucene index
19 */
20 public class LabIndexer {
21
22     /*
23      * This variable is the full path of the directory that will hold the Lucene index
24      */
25     private String _indexLocation;
26
27     public LabIndexer() {
28         _indexLocation = SearchBase.getDirectoryPath();
29     }
30
31     /**
32      * This function returns a vector of UserProfiles. We will use
33      * this as the data source of 'documents' to be indexed. We will ask Lucene to
34      * index each of these profiles.
35      */
36     private Vector _makeListOfUsers() {
37         Vector v = new Vector();
38
39         v.addElement(new UserProfile(1, "Bill", 10, "cycling down hills candy land pancakes"));
40         v.addElement(new UserProfile(2, "Mary", 20, "x-country skiing tele skiing skating swiming"));
41         v.addElement(new UserProfile(3, "Charlie", 21, "beer beach sailing"));
42         v.addElement(new UserProfile(4, "Fred", 22, "apline skiing tele skiing tv beer"));
43         v.addElement(new UserProfile(5, "Jill", 24, "hiking reading coffee traveling"));
44         v.addElement(new UserProfile(6, "Paul", 21, "beer beach sand flies"));
45
46         return v;
47     }
48
49     /**
50      * This function demonstrates how to index a list of documents
51      *
52      * @param uses a Vector full of objects of type UserProfile, these will be each added to
53      * @param create If true, create or overwrite existing index; if false, add to the existi
54      * @exception IOException thrown by writer if you have a problem (i.e. trying to overwrit
55      */
56     private void _createIndex(Vector users, boolean create) throws IOException {
57         /*
58          * Create an IndexWriter
59          */
60         IndexWriter writer = new IndexWriter(_indexLocation, new StandardAnalyzer(), create);
61
62         /*
63          * Index each UserProfile by:
64          * 1) Creating a Lucene document;
65          * 2) Adding Lucene fields to it;
66          */
67     }
3) Adding the document to the IndexWriter
* Once all the documents have been added to the IndexWriter, call optimize(). This
* function creates the inverted index and saves it at the indexLocation.
*/
for (int i = 0; i < users.size(); i++) {
    /* Create a new Lucene document -- we will fill this up with data */
    Document doc = new Document();
    /* Get the UserProfile object, which contains the data we need to index */
    UserProfile u = (UserProfile) users.elementAt(i);
    /* Create instances of the Lucene Fields and add them to the document */
    doc.add(new Field(SearchBase.cUserIDField,
            String.valueOf(u.getId()), // The data to be stored
            Field.Store.YES, // Store this field in the index
            Field.Index.NO)); // Do NOT index this field
    doc.add(new Field(SearchBase.cUserNameField,
            u.getName(),
            Field.Store.YES, // Store this field in the index
            Field.Index.NO)); // Do NOT index this field
    doc.add(new Field(SearchBase.cUserDescriptionField,
            u.getIntersts(),
            Field.Store.NO, // Do NOT store this data
            Field.Index.TOKENIZED)); // Index this data
    /* At this point we've created a Lucene document. Now, add it to the IndexWriter */
    writer.addDocument(doc);
    /* Build the index -- it will be save to the disk */
    writer.optimize();
    writer.close();
}
/* Used to create a Lucene index */
public String createLuceneIndex () {
    String msg;
    try {
        Vector users = _makeListOfUsers();
        _createIndex(users, true);
        msg = "Wrote index to: " + _indexLocation;
    }
    catch (Exception e) {
        msg = "Error: " + e.toString();
    }
    return msg;
}
package edu.uw.info340.lucene;

import java.io.*;
import java.util.*;
import org.apache.lucene.search.Hits;
import org.apache.lucene.search.Searcher;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.search.Query;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.analysis.standard.StandardAnalyzer;
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.queryParser.ParseException;

public class LabSearcher {

    /*
     * This variable is the full path of the directory that will hold the Lucene index
     */
    private String _indexLocation;

    /*
     * Used to hold the results of a Lucene query
     */
    private Vector _UserProfileHits;

    public LabSearcher() {
        _indexLocation = SearchBase.getDirectoryPath();
        _UserProfileHits = new Vector();
    }

    /**
     * Demonstrates how to search a Lucene index
     */
    public int search(String qString) throws IOException, ParseException{
        /*
         * To use the index for searching or updating it, we must first open it
         */
        IndexReader reader = IndexReader.open(_indexLocation);

        /*
         * To search the index we must create an instance of the Searcher class
         */
        Searcher searcher = new IndexSearcher(reader);

        /*
         * To process queries, we must 'normalize' terms (stem, remove stopwords, etc.).
         * class Analyzer is used to this and there a many different kinds of analyzers.
         */
        Analyzer analyzer = new StandardAnalyzer();

        /*
         * Now, we create a QueryParser which specifies what field(s) to search and how to
         * should be analyzed during the search process
         */
    }
QueryParser parser = new QueryParser(SearchBase.cUserDescriptionField, analyzer);
/*
 * After all this setup, we are ready to create a query
 */
Query query = parser.parse(qString);
/*
 * Once we have a query, we can perform a search against the index
 */
Hits hits = searcher.search(query);
/*
 * Remove any previous results
 */
_userProfileHits.clear();
/*
 * Now loop through the hits and store the data in the vector
 */
int uid;
String t;
String uname;
String descr;
int age = 0;
int no_hits = hits.length();
for (int i = 0; i < no_hits; i++) {
    t = hits.doc(i).get(SearchBase.cUserIDField);
    uid = Integer.parseInt(t);
    uname = hits.doc(i).get(SearchBase.cUserNameField);
    descr = hits.doc(i).get(SearchBase.cUserDescriptionField);
    _UserProfileHits.add(new UserProfile(uid,uname,age,descr));
}
/*
 * Close the search object
 */
searcher.close();
return _UserProfileHits.size();
/**
 * Gives access to results
 */
public String getDataField(int record_no, String fieldName) {
    UserProfile u = (UserProfile) _UserProfileHits.get(record_no);
    String o = ""
    if (fieldName.compareToIgnoreCase(SearchBase.cUserIDField) == 0) {
        int t = u.getId();
        o = Integer.toString(t);
    }
    if (fieldName.compareToIgnoreCase(SearchBase.cUserNameField) == 0) {
        o = u.getName();
    }
    if (fieldName.compareToIgnoreCase(SearchBase.cUserDescriptionField) == 0) {
        o = u.getIntersts();
    }
    return o;