A RELATIONAL APPROACH TO THE DESIGN OF
SEARCH ENGINES

BY

SWATHI KANURY

A Thesis Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirement for the Degree of
Master of Science

Southern Connecticut State University
New Haven, Connecticut
August 2005
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
A RELATIONAL APPROACH TO THE DESIGN OF SEARCH ENGINES

BY

SWATHI KANURY

This thesis was prepared under the direction of the candidate’s thesis advisor, Dr. Taraneh BaradaranSeyed, Department of Computer Science, and it has been approved by the members of the candidate’s thesis committee. It was submitted to the School of Graduate Studies and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

Sandra C. Holley, Ph.D.
Dean, School of Graduate Studies

August 4, 2005
Date

Taraneh BaradaranSeyed, Ph.D.
Thesis Advisor

Hrvoje Podnar, Ph.D.
Second Reader

Winnie Yu, MS.
Department Chairperson

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT

Author: Swathi Kanury
Title: A RELATIONAL APPROACH TO THE DESIGN OF SEARCH ENGINES
Thesis Advisor: Dr. Taranch BaradaranSeyed
Institution: Southern Connecticut State University
Year: 2005

The World Wide Web (the “Web” or “WWW” for short) is simply defined as the universe of global network-accessible information. It is an abstract space in which people can interact, and is currently mainly populated by interlinked pages of hypertext, images and animations, sounds, three dimensional worlds, and videos. It is a challenge to provide access to such a large volume of information. Search Engines are tools that solve the resource discovery problem in the specific context of the World-Wide Web. They provide a fast and convenient method for finding resources by maintaining indices of the Web that can be queried for documents about a specific topic. The goal of this thesis is to develop and implement a user-friendly web based search engine with almost all of the features of major search engines and use a relational database for storing and retrieving the information found in the web pages. The application is encoded in Java platform with connectivity to Oracle9i relational database that stores the relationship between web entities,
words, images, multimedia clips and other web pages and rank multiple results of a search query based on the relevancy of search words and web pages retrieved.
To my dear husband Ram.
None of this would be possible without your love and support.
TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION .................................................................................. 1

Search Engines .................................................................................................. 1

CHAPTER 2: BACKGROUND ............................................................................... 6

Background of Search Engines ......................................................................... 6

Shortcomings of popular search engines ......................................................... 8

Insight into Relational Search Engine (RSE) ................................................. 11

CHAPTER 3: RELATIONAL DATABASES ......................................................... 13

Introduction to Relation databases .................................................................. 13

Entity Relationship model ............................................................................. 14

Entity Objects and Relationships used in Search Engine .............................. 15

CHAPTER 4: RELATIONAL SEARCH ENGINE (RSE) .............................. 20

Introduction of RSE ........................................................................................ 20

CHAPTER 5: WEB CRAWLER .......................................................................... 25

Introduction to crawler .................................................................................. 25

Types of Crawlers .......................................................................................... 26

Algorithmic analysis of web crawler .............................................................. 27

Architecture of Web Crawler used in RSE .................................................... 28

CHAPTER 6: PARSERS AND INDEXING ......................................................... 33
Parsinig ................................................................. 33

Indexing .................................................................. 33

Analysis of parser and indexer used in RSE .......... 35

CHAPTER 7: RANKING .................................................. 38

Introduction to ranking .............................................. 38

Word Ranks .............................................................. 39

Page Rank ................................................................. 39

Algorithm used to calculate word and page ranks in RSE 41

  Word Rank Computation ......................................... 41

  Page Rank Computation ......................................... 43

CHAPTER 8: RELATIONAL SEARCH ENGINE (RSE) .... 47

Introduction to Search Engine Architecture ............... 47

  String matching ..................................................... 49

Software used .......................................................... 50

  Tomcat Servlet Engine ........................................... 50

  Velocity Template Engine ................................. 51

CHAPTER 9: CONCLUSIONS ...................................... 54

Implementation issues ............................................... 54

  Dynamic Environment ......................................... 54

  Spamming .......................................................... 54

Other Search engines ............................................... 55

  Scalability ......................................................... 56

Results ................................................................ 56
LIST OF TABLES

Table 1. Summary of weights used to calculate word ranks for RSE ........................................... 43
Table 2. List of queries used for RSE evaluation ............................................................................... 57
Table 3. Points assigned for evaluation marks ................................................................................ 59
Table 4. Average amount of grades scored by search engines for single word queries .............. 59
Table 5. Average amount of grades scored by search engines for double word queries .......... 62
Table 6. Average amount of grades scored by search engines for triple word queries .......... 64
LIST OF FIGURES

Figure 1. Growth of the number of Internet Hosts between January 1994 and January 2004........ 3
Figure 2. Growth of the number of the WWW websites since 1996....................................... 4
Figure 3. Number of users on Internet worldwide................................................................. 4
Figure 4. Meta Tag example .................................................................................................. 9
Figure 5. ER Model for the Search Software .......................................................................... 16
Figure 6. The Search Page that allows the user to search for a Keyword or Phrase.............. 23
Figure 7. The Search Results Page ......................................................................................... 24
Figure 8. Architecture of Web Crawler .................................................................................. 30
Figure 9. Crawler Algorithm ............................................................................................... 31
Figure 10. Example of Inverted List ...................................................................................... 35
Figure 11. Page Rank Example ............................................................................................. 45
Figure 12. Page Rank calculation for first iteration ............................................................... 45
Figure 13. Final ranks for 20 iterations .................................................................................. 46
Figure 14. Architecture of Search engine Software ............................................................... 48
Figure 15. Algorithm used to retrieve relevant documents for a given query ...................... 50
Figure 16. Servlet Container functionality ........................................................................... 51
Figure 17. Tomcat Servlet flow ............................................................................................ 52
Figure 18. Graph representation of document search results from RSE and existing southernct.edu search engine (Google) ........................................................................................................... 60

Figure 19. Graph representation of image search results from RSE and existing southernct.edu search engine (Google) ........................................................................................................... 61

Figure 20. Graph representation of document search results for two word queries .................. 62

Figure 21. Graph representation of image search results for two word queries ..................... 63

Figure 22. Graph representation of document search results for three word queries .............. 64

Figure 23. RSE image search results for “Seyed” ................................................................... 215

Figure 24. Google image search results for “Seyed” ............................................................... 216

Figure 25. RSE document search results for “Extended Learning” ....................................... 217

Figure 26. Google document search results for “Extended Learning” ................................... 218

Figure 27. RSE document search results for “International Studies” ..................................... 219

Figure 28. Google document search results for “International Studies” ............................... 220

Figure 29. RSE image search results for “Computer Science” ............................................. 221

Figure 30. Google image search results for “Computer Science” ......................................... 222

Figure 31. RSE document search results for “Football match schedule” ............................... 223

Figure 32. Google document search results for “Football match schedule” ........................... 224

Figure 33. RSE document search results for “Commencement Weekend Activities” ............ 225

Figure 34. Google document search results for “Commencement Weekend Activities” ......... 226

Figure 35. RSE media search results for “Public Health” ..................................................... 227

Figure 36. RSE e-mail search results for “Seyed” ................................................................. 228

Figure 37. Southernct.edu’s e-mail search results for “Seyed” .............................................. 228
CHAPTER 1: INTRODUCTION

Search Engines

The World Wide Web (the “Web” or “WWW” for short) is simply defined as the universe of global network-accessible information. It is an abstract space that provides people with interactivity, and is chiefly populated by interlinked pages of hypertext, images, animations, sounds, three dimensional worlds, and videos. Before the invention of the web, coexistence of any two computer systems situated globally apart was not easy. Lack of information sharing was still in its infancy before WWW. With the advent of WWW, the gap is filled as the computers that are globally apart can communicate with each other via web with great speed and ease. Thus the birth of WWW marks the end of an era of inability to communicate due to frustrating and debilitating incompatibilities between computer systems. The Web is based on a hypertext system, a system for storing images, text, and other computer files that allow direct links to related text, images, sound, and other data over the Internet.

The World Wide Web has become one of the fastest growing resources on the Internet today. Its popularity can be attributed mainly to its uniform access method to various network information services (Internic, n.d.) and its hypermedia support (Beeferman, n.d.) which link a wide range of multimedia data physically distributed all around the world into a single enormous virtual data source. WWW also provides a powerful and easy to setup medium for user-friendly information
dissemination. More and more information has become available through WWW, from personal data, to scientific reports, to up-to-the minute satellite images. In order to find specific WWW pages a user has to navigate and find his/her way through billions of WWW sites. This is an extremely time consuming process. Methods to relieve the users from this information overflow have been explored by others, from creating a special Usenet (Usenet, n.d.) newsgroup for announcing new WWW sites, to sharing personal hot lists (Hotlists, n.d.), compiled lists, and directories, to searchable full text indexed databases.

The Web grew out of a project at Conseil Européen pour la Recherche Nucléaire (called CERN for short) (World wide web and CERN, n.d.), beginning around 1989, where Tim Berners-Lee and Robert Cailliau (Cailliau, n.d.) built the prototype system that became the core of what is now the World Wide Web. The original intent of the system was to make it easier to share research papers among researchers. The original name of the first prototype was “Enquire Within Upon Everything” (Connolly, 2000). On August 6, 1991 (Connolly), Berners-Lee released files describing his idea for the “World Wide Web” to the Internet.

Since its inception in the early 1990’s, the World-Wide Web has undergone explosive, exponential growth. Figure 1 represents the growth of the number of hosts from January 1994 to January 2004 (Zakon, n.d.). A host is a computer system with registered IP address. As represented in the Figure 1, number of hosts in the world shows an increase of five fold in ten years.
Figure 1. Growth of the number of Internet Hosts between January 1994 and January 2004

Figure 2 represents the growth of the number of websites since 1996 (Zakon, n.d.). A website is hosted on a web server that holds all the files and data that is used by the website. One host may serve multiple web sites by using different domains or port numbers. As represented in Figure 2, there is one website in 1990 and by December 2003, the number grew to nearly 50 million web sites around the world.

The number of users of the WWW has grown tremendously (CyberAtlas, n.d.); Figure 3 represents the size of the growth during the period of 1994 to 2002.
Figure 2. Growth of the number of the WWW websites since 1996

Figure 3. Number of users on Internet worldwide.

Users increasingly find themselves unable to browse the ever-changing, distributed hyperlink structure of the web. They are subjected to information overload. Centralized web search indexes
have been developed to solve this problem with search engines as the preferred technology. An increasing number of services are acting as “Yellow-pages” (e.g. Bigbook (*Bigbook yellow pages service*, n.d.), Yahoo (*Yahoo web site*, n.d.), the Internet Business Pages (*Internet business pages*, n.d.), trying to organize information about WWW sites into neat, descriptive categories. All this organizational effort achieves, at best, is reduction in size of the space that a user needs to consider. The space may still be dauntingly large and it only achieves the intended benefit if the pre-determined categorization used in the Yellow-page service matches some category that the user recognizes as relevant.

The other popular technique for finding information on the WWW is to use one of the content-based search tools (Open Text (*Open text*, n.d.), Lycos (*Lycos search engine*, n.d.), Excite (*Excite search engine*, n.d.), Alta Vista (*Northern Webs*, n.d.), etc.). These search tools; also called search engines, attempt to index the entire web via its content, where they define content to be the words in a web page. This goal can never be completely achieved due to constant changes in data on web. The search engines provide users with the ability to search the indices, returning pointers (URLs) to pages that best match the query words. Chapter 2 describes the different types of search engines and their characteristics.
CHAPTER 2: BACKGROUND

Background of Search Engines

The Search Engines are tools that provide a fast channel to locate resources on WWW by maintaining an index that can be queried for documents about a specific topic. The importance of search engines as mediators of everyday use of the Internet cannot be underestimated – the leading search engine sites (Sullivan, 2003) are consistently among the top ranked sites based on frequency of user access. Pages that are not found in indices are often considered invisible.

Search engines can be classified into three categories (Sullivan, 2002).

- Crawler-based search engines.
- Human-powered search engines also called directories.
- Hybrid search engines (Introduction to hybrid search engines, n.d.), a combination of crawler based and human-powered approach.

Crawler-based search engines (Sullivan, 2002), known, as crawlers, robots, or spiders, are programs designed to “crawl” web pages periodically. Crawlers retrieve and read the content of the visited web pages. Different search engines use their own algorithms (Cho, Garcia-Molina, & Page, 1999) and rules in determining what information is stored and used in the search engine’s database. If a web page is changed, crawler-based search engines eventually will detect these
changes, and that can affect the way the webpage is listed in the search results. Page titles, body text and other elements in a webpage play a role in how it is categorized or ranked.

Directories (Open directory, n.d.) are created by human editors rather than robots or crawlers. The website authors either submit a short description of the entire website or submit the site to the directories without any description. After submitting a Web site to a directory, a human editor categorizes the web site based on its content and writes a short review of the site. The search engine looks for matches only in the descriptions submitted. Meta tags and content of pages are not taken into account. Changing web page content will have no effect on its listing. Factors that are useful for improving ranking of a webpage in a search engine have no relevance in improving the rank of a listing in a directory.

Hybrid search engines (Introduction to hybrid search engines, n.d.) combine a directory with a crawler based search engine to give their visitors the most relevant and complete results. Today, the top ten search engines are hybrids. For example, Yahoo (Open directory, n.d.) that started out as a directory, supplements its manually compiled listings with search results from Google (Google, n.d.), another search engine. Google uses Open Directory Project's (Google directory, n.d.) directory to enrich its search results. Hybrid search engines have a crawler-based search engine that crawls the web and builds listings, and a directory based search engine, which contains human submitted reviews for different websites. The search results are combined when presented to the end user.
Crawler-based search engines (Sullivan, 2002) have three major elements. First is the spider (Cho et al., 1999), also called the crawler. The spider visits a web page, processes the page, and then follows links to other pages within the site. The spider returns to the site on a regular basis, to search for changes. Everything the spider finds goes into the second part of the search engine, the index. The index, sometimes called the catalog, is like an enormous book containing a copy of every web page that the spider finds. Once the spider finds a change in a previously crawled page, this book is updated. It takes some time from the moment the spider visits the page to the moment the page is indexed. In other words, a web page may have been "spidered" but not yet "indexed." Until a page is added to the index, it is not available to the users of the search engine.

The search engine software is the third part of a search engine. It sifts through the millions of pages recorded in the index to find matches to a search, and ranks them in the order of relevancy. In general, page relevance (Web page relevance, n.d.) is calculated by noting the position of the search term within the text and assigning its position, a "weight" or level of importance. Some search utilities also include a popularity element in calculating the relevance algorithm; that is, the more a site is linked to by other web pages, the higher the rating. Search terms occurring in the title, summary, in key positions within a paragraph, or appearing several times within a paragraph usually carry more "weight" because there is a higher probability that terms in these positions indicate significant material on the topic.

*Shortcomings of popular search engines*

There are at least 150,000 different search engines (W3 Search engines, n.d.) available, which cover both general and specific subjects such as: medical directory searches, research paper search, and educational searches. Search engines follow unique criteria for organizing information
(Northern Webs, n.d.). There are different methods to organize search results, and most search engines use a combination of several techniques.

Link popularity of a website is measured in terms of the number of links pointing to it from other sites. A growing number of search engines use link popularity in their ranking algorithms. Google (Google, n.d.) considers link popularity as the most important factor in ranking web pages.

Meta tag is one of the hypertext language tags that is inserted in the "<head>" tag of the hypertext page. Information in the head area of a webpage is not seen by the users viewing the webpage. The Meta tag information in this area is used to communicate information that a human visitor may not be concerned with. Meta tags, for example, can tell a browser what "character set" to use or whether a web page has self-rated itself in terms of adult content. Figure 4 shows an example of Meta tag. The Meta tag, “description” holds the description of the page while the Meta tag “keywords” lists important keywords or phrases used in the document.

```
<HEAD>
<TITLE>Title of the page</TITLE>
<META name="description" content="Everything you wanted to know about stamps, from prices to history">
<META name="keywords" content="stamps, stamp collecting, stamp history, prices, stamps for sale">
</HEAD>
```

Figure 4. Meta Tag example
Google (Google, n.d.) is one of the major search engines offering its own database of indexed Web pages to public. It is one of the largest search engines, indexing web pages together with PDF, DOC, PS, and many other file types. Google indexes and stores the webpage on the search engine’s file system and leaves the decision to the user whether to view the webpage from google’s cache or to actually load the web page from the website to view the search result page. However, Google does not support full Boolean search, and only indexes first 101 KB of a web page and about 120 KB of PDF files.

Alltheweb (AllTheWeb, n.d.) is also one of the major search engines. It maintains a variety of databases for indexing web pages including PDF and flash files, news, pictures, videos, MP3 and FTP files. AlltheWeb indexes the full web pages, PDF and the text in flash files and supports boolean search. Alltheweb however does not maintain cached copies of the web pages.

AltaVista (Nothern Webs, n.d.) was one of the three largest and most important search engines for many years, but it has lost its ground. It has some powerful search features like proximity searching and truncation. Proximity searching is a feature which focuses the search term more closely to retrieve documents that contain the search terms in close proximity to each other: that is, near each other, in the same sentence, or in the same paragraph. Truncation allows the search engine to search for multiple variations of the search terms. For example, searching for the term “music” will also return results for “musical,” “musician,” “musicality” etc. Altavista does not have a large database and it only indexes the first 110K of a Web page and 750K of a PDF file. It has no cached copies of pages and does index variety of file types except PDF files.
MSN search (*MSN Search*, n.d.) is one of the search engines developed for the MSN portal. It has one of the largest Inktomi databases (*Inktomi*, n.d.), but lacks a cached archive.

**Insight into Relational Search Engine (RSE)**

As far as the public information indicates, none of the search engines that are widely used use a 100% relational database for storing the results obtained from crawling the web pages.

The goal of RSE is to design and implement a user-friendly web based search engine with almost all the features of major search engines and use a relational database for storing and retrieving the information about the web pages.

The search engine consists of:

1) Crawler: Crawls the web. The detail design and description of the crawler is presented in Chapter 5.

2) Page Processor: Processes the retrieved web pages and stores the extracted information in a relational database. The detailed description and design of the database is presented in Chapter 3.

3) Rank Processor: Calculates web pages and indexed words rankings that are stored in the relational database. The detailed description of rank processor is presented in Chapter 7.

4) Extract Processor: Extracts summary for the indexed words. The detailed description is presented in Chapter 4.
5) Search Software: The search engine software accepts the user queries and presents the search results in a user readable format. The detailed design of the search engine is presented in Chapter 4.
CHAPTER 3: RELATIONAL DATABASES

Introduction to Relation databases

A database is a collection of files containing related data. The representation of relationships between files depends on the model used. Early database models included hierarchical structure (Hoffer, 2002, chap. 1-5) (files are related in a parent/child manner, with each child having at most one parent), and the network structure (files are related as owners and members, similar to the hierarchical model except that each member record can have more than one owner). The concept of relational databases was first introduced by Edgar Frank Codd (Philipson, 2003) in the IBM research report RJ599, dated August 19th, 1969.

The relational database model (Hoffer, 2002, chap. 1-5) was a major step forward from the early models, as it allowed files to be related by means of a logical pointer, a common field between two files. In order to relate any two files, one simply needs to have a common field, which makes the model extremely flexible. The relational model derives from a mathematical concept of relations defined over datasets.

A Relational Database Management System (RDBMS) (Date, 2001) enables users to store related pieces of data in two-dimensional data structures called tables. A table is an entity, which can represent a tangible or intangible object. Columns in a table define attributes of the object
while rows which are made up of a combination of columns, represent a single instance of the entity. The table elements may be represented by defined data types, such as integer numbers, floating-point numbers, character strings, and timestamps. Data inserted in the table can be categorized by a grid-like system of columns and rows. Oracle, MySQL, Postgres are some of the available and widely used RDBMS.

*Entity Relationship model*

The Entity-Relationship (E-R) model (Date, 2001) was originally proposed by P. Chen in 1976 (Chen, 1976, p. 9-36) as a method to unify the network and relational database views. Simply stated the ER model is a conceptual data model that views the real world as entities and relationships. A basic component of the model is the Entity-Relationship diagram (Date), which is used to visually represent data objects. The E-R model views the real world as a construct of entities and association between entities called relationships.

Entities are the principal data object about which information is to be collected. Entities are usually recognizable concepts, either concrete or abstract, such as person, places, things, or events, which have relevance to the database. Some specific examples of entities are EMPLOYEES, PROJECTS, and INVOICES. An entity is analogous to a table in the relational model.

A Relationship represents an association between two or more entities. An example of a relationship would be, employees are assigned to projects, projects have subtasks, and departments manage one or more projects. Relationships are classified in terms of degree, connectivity, cardinality, and existence.
Figure 5 shows the E-R model used to build the proposed Relational Search Engine (RSE). Entities are represented by labeled rectangles. The diamond shaped objects in Figure 5 represent relationships between two entities. Attributes of an entity are represented as labels enclosed in ellipses. Attributes, that are identifiers, are underlined. A line ending in a crow’s foot represents cardinality of many. If the crow's foot is omitted, the cardinality is one. Placing a circle or a perpendicular bar on the line represents existence of an entity instance. The perpendicular bar (looks like a |) next to the crow’s feet shows the mandatory existence for an instance. A circle next to the crow’s feet shows the optional existence of an entity.

Entity Objects and Relationships used in Search Engine

There are five entities used in the design of RSE:

1) URL entity is used to store URLs and the document type of the URL. URL entity has the following attributes:

- URL_ID holds the unique identifier for each URL crawled by the Crawler.
- URL_DESC holds the actual URL
- DOC_TYPE_ID is the document type of the URL.
- RANK holds the page rank for each URL. The rank is calculated and populated by Rank Processor after crawling and page processing is completed.
- ADD_BY_DATE holds the date on which the URL is crawled.
Figure 5. ER Model for the Search Software

2) DOC_TYPE entity is used to store various types of documents that can be available on the web like word documents, “PDF,” “PS,” images, media. It has the following attributes:

- DOC_TYPE_ID holds a unique identifier for each row.
- DOC_DESC holds a short description for each document type.

3) WORD entity is used to store the words that are found when the document pointed by URL is parsed. WORD entity has the following attributes:

- WORD_ID holds a unique identifier for each row.
- WORD_DESC holds the word.

4) MMEDIA entity is used to store all the media found on the site.
• CLIP_ID holds a unique identifier for each row

• MEDIA_TYPE holds the type of media like “MP3,” “real media” or “avi.”

• LABEL holds a short description of the media file that is presented in the web
document.

• FILE_SIZE holds the size of the media file in bytes

• URL_ID represent the URL of the media, this is a reference key to URL entity.

5) IMAGES entity is used to store images found while parsing the URLs.

• URL_ID holds a unique identifier for each row.

• IMAGE_TYPE holds the type of image like “jpg” or “gif” or “bmp.”

• LABEL holds a short description of the image file that is presented in the web
document.

• FILE_SIZE holds the size of the image file in bytes.

• DIMENSIONS hold the dimensions of the image in pixels.

There are two relations:

1) URL_XREF: this is a unary relation (Chen, 1976, p. 9-36) for URL entity, which
represents the hyperlink structure of web.

• URL_ID represents a reference to URL_ID in URL entity.

• URL_REF_ID is the URL that links out to the URL represented by URL_ID.

2) URL_WORD_XREF: this is a binary relation (Chen, 1976, p. 9-36) between URL and
WORDS entities to represent the relationship between a documents and words that are in
that documents. Pre-defined weights are assigned to properties of the WORD.

• URL_ID holds the reference to URL_ID in URL entity.
• WORD_ID holds the reference to WORD_ID in WORD entity.

• REL_POSITION is the relative position of the word in the document.

• CAPITAL holds a pre-defined weight (see Chapter 7), if the word starts with a capital letter.

• ANCHOR holds a pre-defined weight (see Chapter 7), if the word is found in an anchor tag of the document.

• TITLE holds a pre-defined weight (see Chapter 7), if the word is found in the title tag of the document.

• STYLE holds a pre-defined weight (see Chapter 7), based on the font size of the word used in the document.

• RANK holds the word rank. Word rank is calculated based on all the pre-defined weights that are obtained for the word.

• ADD_BY_DATE holds the system date on which the word information is updated.

• SUMMARY holds a short summary of the word used in the document. For example, summary of a search term could be the first sentence in the web page where the search term occurred. The Summary Updater Process populates this attribute after the crawling process is done.

• TITLE_TEXT holds a title for the word and the document. The title of a word could be the title of the page or the title of the section where the word occurred in the document. The Summary Updater process populated this attribute after the crawling process is done.
Chapter 4 describes the inner workings of the RSE and the way each module will process the retrieved web pages to populate the tables of the underlying relational database.
CHAPTER 4: RELATIONAL SEARCH ENGINE (RSE)

Introduction of RSE

RSE is built in five modules. Each module is detailed below:

1) Crawler: A Crawler has two sections; first section is an URLResolver that resolves the DNS name of a given URL. This is used to get the correct IP address associated with the DNS name. The URLResolver also checks for duplicate URL to avoid crawling the same page more than once. The second section is a Web Crawler that crawls the web pages in specific intervals and forwards the web pages to Page Processor for further processing. Chapter 5 details the design of RSE crawler.

2) Page Processor: Page processor parses the web page content that is returned from the crawler and extracts all the words, URLs, images, and multimedia files, that are referenced in the web page. The information that is collected is passed on to four sub processors for further information processing.

3) URL Processor: URL Processor checks the URL’s that are found in the webpage for validity and stores them in the URL entity of the search database along with the document type that is represented by the URL. The type of the document is determined by the file extension in the URL. URL Processor stores the reference URL information in the URL_XREF entity. Reference URL is the URL that has referenced the current
URL that is being processed by URL Processor. It can be the web page URL in which the current URL was found.

4) Word Processor: The processor extracts the words along with the details of the word's relative position in the document, and the placement of the word in the document like whether the word is placed in the title of the document, in the anchor text or whether the word is capitalized. The actual word is stored in the WORDS entity and the information about the word is stored in the relationship between the URL entity and the WORDS entity.

5) Image Processor: the image processor will capture the information on images that is available while processing the document. The information captured could be dimensions, description and size of the image. This information is stored in the IMAGES entity along with the document URL in which the image is found.

6) Multimedia processor: the multimedia processor captures the information on multimedia files that are available in the web documents and captures the file size, description and type and stores them in MMEDIA entity. Chapter 6 details the parsing methodologies used in RSE.

7) Rank Processor: Rank Processor consists of two sections:
   a) Calculating Word Ranks: word ranks are calculated based on the position of the word in the document. The word weight will be calculated as the sum of all predefined weights specified for different positions in the documents like the positioning of the word in the title of the text, the anchor text and the capitalization of the text. This calculated weight along with the relative position of the text in the document will give the final rank of the word.
b) Calculating Page Ranks: A web page is ranked (Brin & Page, 1998) based on the number of links, both the outgoing links and the incoming links. Typically the web can be represented as a directed graph, the nodes represent pages and directed edges represent links between pages. The reference links are populated in the unary relation for URL entity called URL_XREF. The process takes the snapshot of this relation and calculates the page rank (Brin & Page) for every URL and stores it in the URL entity.

Chapter 7 details the ranking algorithms and methodologies used by RSE.

8) Extract Processor: Extract processor collects the summary and the title for each word and URL document stored in the URL_WORD_XREF entity. The processor takes each URL and WORD, retrieves the document from the web, parses the document to search for the WORD and extracts a short summary and a title to represent the word position in the document.

9) Search Software: Search Software has four sections:

   a) Retriever: Retriever is a java Servlet that provides the user interface for the user's search. The user interface is shown in Figure 6. Once the user selects the type of search (like document, images, media or email), enters the query and hits the "submit" button, the query is retrieved and passed to the Parser for further processing.

   b) Parser: the parser parses the query to remove any stop words and breaks the query into meaningful set of words. These words are handed over to the Query engine for further processing.
c) Query Engine: the query engine queries the database relationship between WORDS and URL to find the matches for the user submitted query. The results found are returned back to the presenter.

![Diagram of relational search engine](image)

*Figure 6. The Search Page that allows the user to search for a Keyword or Phrase*

d) Presenter: The presenter sorts the results based on the page rank and send the output to the browser in a readable format, so that the user can select and click on the links to go to the resultant pages. The results screen is shown in Figure 7.

The application is built in Java platform with connectivity to Oracle9i relational database that stores the information about the web pages and the relationship between web entities. The search results returned by the application are ordered by the relevance ranks. The search is built and tested on Southern Connecticut State University website.

23
<table>
<thead>
<tr>
<th>International Studies Home Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>International Culture and Language</td>
</tr>
<tr>
<td>International Relations Club is an organization of international and American students devoted to promoting international friendships, improving intercultural relations, and creating stronger bonds of world unity.</td>
</tr>
<tr>
<td><a href="http://clubs.ccsu.edu/clubroot/International/intnl.htm">http://clubs.ccsu.edu/clubroot/International/intnl.htm</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>International Students Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>GOAL To assist international (Temporary Visa) and U</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Welcome to George R. Muirhead Center for International Education</th>
</tr>
</thead>
<tbody>
<tr>
<td>Muirhead Center for International Education</td>
</tr>
<tr>
<td><a href="http://www.ccsu.edu/cie">http://www.ccsu.edu/cie</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>International Students Contact Office of International Student Services Engleman Hall 31E</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 Crescent St</td>
</tr>
</tbody>
</table>

*Figure 7. The Search Results Page*
CHAPTER 5: WEB CRAWLER

Introduction to crawler

Crawlers (also known as robots, spiders, worms, walkers and wanderers) are automated data retrieval programs traveling the web to find information and links. When a crawler visits a webpage, it reads its content and follows the hyperlinks from page to page. Crawlers are an important tool in extracting information and indexing the pages on the Internet. Web Crawlers are almost as old as the web itself (Web robots page, n.d.). The first crawler, Matthew Gray’s Wanderer (Gray, 1996), was written in the spring of 1993, roughly coinciding with the first release of NCSA Mosaic (NCSA Mosaic, n.d.).

Web crawlers typically spread their visits to a particular site over a period of time, in order to not to overload the target web servers. For similar reasons, web crawlers are supposed to obey the robots.txt protocol (Robots exclusion, n.d.) where web site owners can indicate which pages should not be spidered.

An ideal Web Crawler has two primary goals:

• To seek out new web objects: crawling newly added pages or sites.
To observe changes in previously discovered web objects (web-event detection). Already crawled pages need to be re-crawled to check for any changes. This can be achieved by periodically crawling the same sites.

Types of Crawlers

There are four types of crawlers available:

- General crawlers
- Topic driven crawlers
- Individual/personal crawlers
- Commercial crawlers

General crawlers (Yiqiao, n.d.) can be commercial or non-commercial and are used by search engines. They are also called query driven crawlers as they try to answer all possible ad-hoc queries. They are link based i.e. they consider the count of links to reflect the credibility of the web page. A general-purpose web crawler normally tries to gather as many pages as it can, using the hyperlinks from a particular set of sites.

A topic driven crawler (Yang & Wang, 2000) (also called Focused crawler) selectively seeks out pages that are relevant to a pre-defined set of topics. Due to limited bandwidth, storage, computational resources and dynamic nature of the Web, search engines cannot index every Web page, and even the covered portion of the Web cannot be monitored continuously for changes. Therefore it is essential to develop effective crawling strategies to prioritize the pages to be indexed. Rather than collecting and indexing all accessible web documents to be able to answer all
possible ad-hoc queries, a topic-driven web crawler (Yang & Wang) analyzes its crawl boundary to find the links that are likely to be most relevant for the crawl, and avoids irrelevant regions of the web.

Individual/personal web crawlers (Miller & Bharat, 1998), have limited functional scope. An example of personal crawler is the work discussed in the SPHINX project (Miller & Bharat). These are non-commercial crawlers used to find the layout or link structure of a web site or used to mirror some web sites for personal use.

Commercial Crawlers (Yiqiao, n.d.) have become the basis for the many specialized services such as investment portals and competitive intelligence tools. An example of commercial crawler is IPHRASE (IPHRASE, n.d.). These crawlers are mainly used by organizations to crawl through organizational data and thereby making it fast and easy to retrieve the data within the organization.

*Algorithmic analysis of web crawler*

There are two basic crawler methods, Breadth first (Cho et al., 1999) and Depth first (Cho et al.).

Breadth First crawlers use the BFS (Breadth-First Search) algorithm (Najork & Wiener, 2001), while depth first crawlers use the DFS (Depth-First Search) algorithm (Miller & Bharat, 1998).
Web-crawler software doesn’t actually move around to different computers on the Internet, as viruses or intelligent agents do. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does. Crawler simply automates the process of following the located links.

*Architecture of Web Crawler used in RSE*

The basic algorithm (Cho et al., 1999) executed by any web crawler takes a list of seed URLs as its input and repeatedly executes the following steps.

1) Remove a URL from the URL list.

2) Determine the IP address of its host name.

3) Download the corresponding document.

4) Extract any links contained in it. For each of the extracted links,
   a) Translate it to an absolute URL (if necessary)
   b) Add it to the list of URLs to download, provided it has not been encountered before.

   If desired, process the downloaded document in other ways (e.g., index its content).

This basic algorithm (Heydon & Najork, 1999) requires a number of functional components. A crawler needs a component for:

1) Storing the list of URLs to download.

2) Resolving host names into IP addresses.

3) Downloading documents using a transfer protocol.

4) Extracting links from document types.

5) Determining whether a URL has been encountered.
This section describes how the crawler designed for this project refines the basic algorithm, and the particular implementations chosen for the various components.

Figure 8 shows the main components of the crawler. Crawling is performed by multiple instances of the crawler running as threads. A crowd controller facility is responsible for selecting and locking a crawler for processing the page. Locking is performed using an application-level lock to prevent other database activities from occurring.

The first step in this process (Step 1) is to select a URL from the URL queue for downloading and assigning it a crawler processes. The URL queue is pre-populated with the home page of the Southern Connecticut State University (www.southernct.edu) since the goal is to build the search engine for the local university website. It is possible for some Web pages from the Southern Connecticut State University domain to have links to pages outside the domain. As a result, the crawler may fetch Web pages that are not related to Southern Connecticut State University. This problem is solved easily by checking whether a new URL is in the correct domain (e.g., check if it contains “southernct.edu” in the URL and in some cases check if the IP address is from the Southern Connecticut State University’s IP addresses.). Crawler adds a new URL to the queue only if it satisfies the domain requirement.

All URLs are absolute - beginning with a scheme (e.g., “http” or “ftp”), which identifies the network protocol that should be used to download it. In Step 3, the crawler invokes the “URL Seen” test to determine if the crawler has already visited the URL. In Step 4, based on the URL’s
scheme, the Crawler selects the appropriate protocol module for downloading the document and invokes its "download" method. The document is then downloaded into memory.

Figure 8. Architecture of Web Crawler

Every downloaded document has an associated MIME (Multipurpose Internet Mail Extensions) type (Cailliau, n.d.), which defines the type of the content or application from which this document was borne. In addition to associating schemes with protocol modules, the Crawler configuration also associates MIME types with a content processing module. A content processing module is responsible for extracting index words and new URLs from a downloaded document. As
not all document types contain links (explicitly defined URLs), not all content processing modules produce new URLs.

At this time, HTML is the only document type for which link extraction is supported. However, support for recognizing and extracting links in other document types such as word documents and PDF document is possible. Figure 9 represents the crawler algorithm.

```
Get the user’s input: the starting URL.
Add the URL to the currently empty list of URLs to search.
While the list of URLs to search is not empty,
{
    Get the first URL in the list.
    Move the URL to the list of URLs already searched.
    Check the URL to make sure it is the link within the
    Parent Site or the referrer of the link is the parent site.
    (If not, break out of the loop, back to “While”).
    Try to “open” the URL (retrieve that document From the Web).
    Parse the html file and retrieve the links.
    Send the parsed file to Page Processor for content processing and
    indexing.
    Step through the links found in the HTML page
    {
        If the URL isn’t present in either the to-search list or the
        already-searched list, add it to the to-search list.
    }
}
```

*Figure 9. Crawler Algorithm.*

Based on the downloaded document’s MIME type, the crawler process invokes the associated content processing module in Step 5. Index words extracted by the content processing module are stored in the relational database. New URLs extracted by the content processing
module are passed to the URL-seen test to be sure the URL has not already been downloaded or added to the queue.
CHAPTER 6: PARSERS AND INDEXING

Parsing

Parsing (Abney, 1996) is the process of extracting content from text. Text parsing technique has been receiving significant interest from web-developers. Much information on the Web and in desktop applications exists in semi-structured forms, which means structured data embedded in unstructured data. Information on the web is a conglomeration of heterogeneous data such as text, images and audio clips, which are often accessed through documents written according to the HTML specification (HTML, n.d.). HTML (HTML) documents are semi-structured in nature. To parse a semi-structured data, grammer based parsers are used. Grammar is a set of rules, each of which recognizes a class of text strings. These text strings can then be stored in the database using an inverted index to be explained in Chapter 5.

Indexing

An index is a mechanism allowing the search engine to locate the URLs, which match user requests. The web documents containing the query keywords are presented as a listing, which may include a brief summary of the site. A simple description of the function of index is to think of it as a computerized book index. To discover where a topic occurs in a book, we would look up the word in the index, which would indicate the page number(s) where the term occurs. Now imagine that every single word is included in the book index. A search engine uses its index to retrieve web
documents in which user's search terms occur. The index lists the term and its location(s) (the URL or address of the web page) much like a book index. Inverted file (Cacheda & Vina, n.d.) method is one of the principle indexing methods for large text data sources. The inverted list (Cacheda & Vina) consists of the relative position of the word in the document and some important characteristics of the word like, capitalization, anchoring (if word occurred in anchor tags), and title (if the word is in title). Figure 10 represents an example of inverted list. Table A represents a list of documents and Table B the represents inverted list for all the documents in Table A. For example, the word cold occurred in Document number 1, in position 6 and in document number 4 in position 8.

RSE uses java swing HTML parser (Java swing HTML parser, n.d.) to parse the semi-structured html documents. The crawler thread (Sun Microsystems, n.d.) hands off the downloaded document to the corresponding content processing module for further processing. The content processing module parses the document using java swing's html parser and builds an inverted list of all the words found in the document. This list is then passed to the indexer for further processing.
Table A

<table>
<thead>
<tr>
<th>Number</th>
<th>Text</th>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>cold</td>
<td>(1, 6), (4, 8)</td>
</tr>
<tr>
<td>2</td>
<td>days</td>
<td>(3, 2), (6, 2)</td>
</tr>
<tr>
<td>3</td>
<td>hot</td>
<td>(1, 3), (4, 4)</td>
</tr>
<tr>
<td>4</td>
<td>in</td>
<td>(2, 3), (3, 4)</td>
</tr>
<tr>
<td>5</td>
<td>it</td>
<td>(4, 3, 7), (5, 3)</td>
</tr>
<tr>
<td>6</td>
<td>like</td>
<td>(4, 2, 6), (5, 2)</td>
</tr>
<tr>
<td>7</td>
<td>nine</td>
<td>(3, 1), (6, 1)</td>
</tr>
<tr>
<td>8</td>
<td>old</td>
<td>(3, 3), (6, 3)</td>
</tr>
<tr>
<td>9</td>
<td>peas</td>
<td>(1, 1, 4), (2, 1)</td>
</tr>
<tr>
<td>10</td>
<td>porridge</td>
<td>(1, 2, 5), (2, 2)</td>
</tr>
<tr>
<td>11</td>
<td>pot</td>
<td>(2, 5), (5, 6)</td>
</tr>
<tr>
<td>12</td>
<td>some</td>
<td>(4, 1, 5), (5, 1)</td>
</tr>
<tr>
<td>13</td>
<td>the</td>
<td>(2, 4), (5, 5)</td>
</tr>
</tbody>
</table>

Table B

Figure 10. Example of Inverted List

Analysis of parser and indexer used in RSE

Web pages are usually formatted in a markup language, mostly in HTML (HTML, n.d.). Terms are parsed out of the document based on the HTML tags (HTML) used in the document. Tags usually appear in pairs with one indicating the start and the other indicating the end. For example, in HTML, the starting title tag is <title> and the ending title tag is </title>. A markup language specifies a document format through the use of a number of tags such as title, anchor, and list. The indexer identifies (Cutler, Shih & Meng, 1997) all possible HTML tags and tag attributes that can hold URLs. While most URLs appear in the anchor tag (e.g., <a href="URL">anchor text</a>), there are other tags in which URLs could appear. For example, a URL may appear in the option tag as in <option value="URL">options</option>, in the area tag (map) as in <area href="URL">area text</area>, or in the frame tag as in <frame src="URL"></frame>. However, a
URL link appearing in a Web page P might not contain the full path needed to locate the corresponding Web page. If, a partial path (relative path) is used, the full path can be derived from either the location of P or the base path provided in P.

Indexer creates an inverted list of words and document URLs. It organizes the information into an abbreviated list of terms, which then depending on the term, references a specific set of documents.

Currently, tag information is used in primarily two different ways to improve retrieval effectiveness. First, tag information can be used to select index terms, i.e., choosing terms to represent a document. Second, tag information can be used to determine the degrees of significance or weights of index terms. In traditional text retrieval systems, all terms except stop words in a document are used to index the document. Some “important” terms are given more weight than regular terms. An “important” term could be a term in the title, in one of the headers, in a location near the start or the end of the page, or a term that has a high frequency or is emphasized with an emphatic font such as boldface and large sized font. In contrast, terms that appear in the middle of the page and in reduced fonts such as small and tiny fonts can be considered less important. The practice of using only selected terms to index documents is called partial-text indexing, while the approach to use all terms is called full-text indexing. Additional tag information is very useful for identifying “important” terms.

Terms not in a document could be used to represent the document according to our needs. When a page A has a link to page B, a set of terms known as anchor terms is usually associated
with the link. The purpose of using the anchor terms is to provide information regarding the
contents of page B to facilitate navigation by human users. The anchor terms (HTML, n.d.) often
provide related terms or synonyms to the terms used to index page B. To utilize such valuable
information, several search engines like Google (Google, n.d.) and WWWW (World wide web
worm, n.d.) have suggested using anchor terms to index linked pages (e.g., page B). In general, a
Web page may be linked by many other Web pages and has many associated anchor terms. Anchor
terms can also be used as collateral text to index non-textual objects such as images.

Java HTML swing parser (Java swing HTML parser, n.d.) has an interface to handle the set
of tags in a HTML document. The content processing module implements the java’s HTML swing
parser interfaces to retrieve all the information from the page and the information is arranged in
inverted lists. The inverted lists generated in content processing module are stored in java
Hashtables and passed on to the indexer for further processing.
CHAPTER 7: RANKING

Introduction to ranking

At the first glance it might seem that all search engines have the same basic features, but one aspect that separates fully functional search engines from the rest, is the ability not only to list the search results in a meaningful way using ranking, but also to allow the user a chance to submit another closer-to-target query using those results. Search engines use their own algorithms for calculating relevance rankings for the search results.

AltaVista (Northern Webs, n.d.), one of the top search engines, ranks the search results based on the different criteria like: whether the words or phrases are found in the first few lines of the document (for example, in the title of a web page), the frequency of occurrence of a query word or phrase, whether all of the specified words or phrases appear in a document, and whether multiple query words or phrases are found close to each other in a document.

Google’s search engine (Google, n.d.) places great emphasis on content and link popularity. Factors that determine a web page’s rank on Google (Google) are: the number of links that point to that site, the popularity of the sites that link to that site, the text in and around the links that point to that site, and the pages that particular site links to. The META tags and keywords represented in Figure 7 have little influence on Google’s rankings.

38
Another popular search engine Teoma (Teoma search engine, n.d.) uses Subject-Specific (Yang & Wang, 2000) Popularity that ranks a site based on the number of same-subject pages in other ranked sites that link to it, not just general popularity. However, the exact algorithms used by search engines are not in public domain.

Word Ranks

Word rank (Brin & Page, 1998) specifies the relevance of the word in a particular document. If a word occurs in a document more than once, or if it occurs in the title of the document, the chances are that the document is more relevant to that word. The position of a word in a document can also play a role in deciding the word’s rank. Search engines follow different algorithms for calculating word ranks.

Page Rank

Web pages are extensively linked. The linkage information can be utilized in a number of ways to improve retrieval performance. Pagerank (Brin & Page, 1998), a widely used technique, introduced by Page et al, (Brin & Page), assigns a global importance measure to each page based on the importance of other pages pointing to it. The main advantage of the Pagerank measure is that it is independent of the query posed by a user; this means that it can be pre-computed and then used to optimize the layout of the inverted index structure (Cacheda & Vina, n.d.) accordingly. However, computing the Pagerank measure requires implementing an iterative process on a massive graph corresponding to billions of web pages and hyperlinks (Brin & Page). Given the large number of web pages on popular topics, one of the main challenges for a search engine is to provide a good ranking function that can identify the most useful results from among the many
relevant pages. Most existing search engines perform ranking through a combination of term-based (Berry & Browne, 1999) techniques, link-based techniques (Berry & Browne), and user feedback (Berry & Browne).

The basic idea of PageRank (Brin & Page, 1998) is that if page $U$ has a link to page $V$, then the author of $U$ is implicitly conferring some importance to page $V$. Intuitively, Yahoo (Yahoo website, n.d.) is an important page, reflected by the fact that many pages point to it. Likewise, pages prominently pointed to or from Yahoo are themselves probably important. To see how much importance does a page $U$ confer to its out-links, let $N_u$ be the out-degree of page $U$, and let $\text{Rank}(U)$ represent the importance (i.e., PageRank) of page $U$. Then the link $(U, V)$ confers $\text{Rank}(U)/N_u$ units of rank to $V$. If $N$ is the number of pages within the search domain, assign all pages the initial value $1/N$. Let $B_v$ represent the set of pages pointing to $V$. In each iteration, the rank is propagated as follows:

$$\text{Rank}_{i+1}(V) = \sum_{u \in B_v} \frac{\text{Rank}_i(U)}{N_u}$$  \hspace{1cm} (6.1)

The iterations are continued until Rank stabilizes to within some threshold. According to Google (Google, n.d.), convergence for over 322 million links took about 52 iterations (Brin & Page, 1998). The final vector Rank (Golub & VanLoan, 1932) contains the PageRank vector over the Web. This vector is computed only once after each crawl of the Web. The values can then be used to influence the ranking of search results (Brin & Page, 1998)).
Algorithm used to calculate word and page ranks in RSE

In RSE, both word ranks and page ranks are computed after each crawl.

Word Rank Computation

Several factors are used in calculating word ranks. For each factor, a pre-defined weight is added and word rank is calculated as the total of the weights for the word. Each property of the word is given a rank between 1 and 5, 5 being the highest rank. Factors used for word rank computation in RSE are:

1) If the word occurred in the title, then a weight of 5 is added to the total weight. The words used in the title are very important, because the title is the identifying factor in the document.

2) If the word occurs in anchor tag, a weight of 4 is added to the total weight of that word. The word occurring in an anchor tag is less important than the word occurring in the title tag of the page.

3) If the word starts with a capital letter, a weight of 1 is added to the total weight. The rank for capitalization is is rather low, because every new sentence might start with a capital letter.

4) The font properties of the word can be represented in 10 different styles:
   a) Bold: Bold fonts are used in the document to highlight important words. Bold words are given a weight of 4.
   b) Italic: Italic style font is used to indicate less important words than the bold font words, but they still hold some importance in the document. The weight of 3 is added to the total weight of the word if the word is in italics.
c) Big Font: A font can be either given a “bigfont” property or a size can be specified. If the font’s size is more than 4 points, then the weight of 5 is added to the total weight of the word. If the word is represented in big font, there is a greater chance that the content of the page is more relevant to the word.

d) Underline: Underlining a word in a document holds some importance of the word. A weight of 3 is added to the total weight of the word.

e) H1 header: H1 header is a pre-defined styled header in HTML. Words defined in H1 header are almost of the same importance as words with big fonts. A weight of 5 is added to the total weight of the word that has H1 header.

f) H2 header: H2 header is mostly used in subheading and is given a weight of 5 as the subheading in the documents has a better chance of representing relevance between word and document.

g) H3 header: H3 header weight is almost same as bold fonts. A weight of 4 is added to the total weight of the word that is specified in the H3 header.

h) H4 header: H4 header words have smaller font than H3 header, but they still carry some significance. A weight of 3 is added to the total weight of the word.

i) H5 header: A weight of 2 is added to the total weight of the word, as H5 header is less important than H4 header.

j) H6 header: A weight of 1 is added to the total weight of the word, as H6 has less importance than H5 header.
The words that are found in the META tag of the page describe the content of the page receive a weight of 5. Table 1 summarizes the word weights used to calculate the final word rank.

Table 1.

*Summary of weights used to calculate word ranks for RSE*

<table>
<thead>
<tr>
<th>Word Style</th>
<th>Assigned Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>4</td>
</tr>
<tr>
<td>Italic</td>
<td>3</td>
</tr>
<tr>
<td>Big Font</td>
<td>5</td>
</tr>
<tr>
<td>Underline</td>
<td>3</td>
</tr>
<tr>
<td>H1 header</td>
<td>5</td>
</tr>
<tr>
<td>H2 header</td>
<td>5</td>
</tr>
<tr>
<td>H3 header</td>
<td>4</td>
</tr>
<tr>
<td>H4 header</td>
<td>3</td>
</tr>
<tr>
<td>H5 header</td>
<td>2</td>
</tr>
<tr>
<td>H6 header</td>
<td>1</td>
</tr>
</tbody>
</table>

The relative position of a word in the document is also recorded and is used when formatting the query results for user display. The results are displayed in ascending order based on the word’s relative position in the document and the word ranks.

*Page Rank Computation*

The Web graph contains an enormous number of links and each link implies certain degree of importance over the pointed page. The rank of each page is computed based on the following three observations.
1) The larger the number of incoming links the page has, the more important the page is likely to be. More incoming links imply that more authors consider the page to be important. In other words, the rank of a page reflects the popularity of the page among all Web page authors.

2) A page is considered an important page if it has important incoming links even though there aren’t a large number of pages linking to it. Intuitively, important pages are likely to be published by important authors or organizations and their endorsement should have more weight in determining the importance of a page. For example, if a page is pointed to by Southern Connecticut State University website’s homepage, then the page is likely to be important.

3) The more the forward links a page has, the less the influence the page should have on the importance of a pointed page. If a page has multiple children pages, then these children pages should share the importance of the parent page. As a result, if a page has more children pages, it can only propagate a smaller fraction of its importance to each child page. The rank of a page can be considered as a weighted popularity measure of the page.

Formally, the page rank is defined as follows: for a given web page A, let T1..Tn be the web pages pointing to A. Let C(A) be the number of links going out of page A. The page rank of page A is represented as (Brin & Page, 1998):

\[ PR(A) = (1-d) + d \left( \frac{PR(T1)}{C(T1)} + \ldots + \frac{PR(Tn)}{C(Tn)} \right) \]  \hspace{1cm} (6.2)

\( PR(Tn)/C(Tn) \) represents the share of vote Page A will get from the total number of links in Page n. d is the damping factor which can be set between 0 and 1. Damping factor stops the pages from not being influenced by high ranks. The total vote is damped down by multiplying 0.85 by total
share of votes for page A. If a page has no links to it (no back links), the page rank will be 1-d.

Page Rank PR(A) is calculated using a simple iterative algorithm, and corresponds to the principle eigenvector of the normalized link matrix of the web (Brin & Page). The calculation is done recursively until the values stop changing. These final values are called limiting values.

![Diagram of Page Rank]

**Figure 11. Page Rank Example**

Figure 11 depicts four pages pointing to each other. Before the page rank calculation starts all pages receive an initial rank of 1. The calculation for the first iteration is shown in Figure 12.

The iterations are continued until the convergence of the ranks. Figure 13 represents the resultant rank after each iteration. The ranks are converged after 20 iterations.

<table>
<thead>
<tr>
<th>PR(A)</th>
<th>PR(B)</th>
<th>PR(C)</th>
<th>PR(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15 + 0.85 * (1/1) = 1.00</td>
<td>0.15 + 0.85 * (1/2) = 0.575</td>
<td>0.15 + 0.85 * (1/2 + 1/1 + 1/1) = 2.275</td>
<td>0.15 + 0.85 * (0) = 0.15</td>
</tr>
</tbody>
</table>

**Figure 12. Page Rank calculation for first iteration**

45
<table>
<thead>
<tr>
<th>Iteration</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
</tr>
<tr>
<td>1</td>
<td>1.000</td>
<td>0.575</td>
<td>2.275</td>
<td>0.150</td>
</tr>
<tr>
<td>2</td>
<td>2.084</td>
<td>0.575</td>
<td>1.191</td>
<td>0.150</td>
</tr>
<tr>
<td>3</td>
<td>1.163</td>
<td>1.036</td>
<td>1.652</td>
<td>0.150</td>
</tr>
<tr>
<td>4</td>
<td>1.554</td>
<td>0.644</td>
<td>1.652</td>
<td>0.150</td>
</tr>
<tr>
<td>5</td>
<td>1.554</td>
<td>0.810</td>
<td>1.485</td>
<td>0.150</td>
</tr>
<tr>
<td>6</td>
<td>1.413</td>
<td>0.810</td>
<td>1.627</td>
<td>0.150</td>
</tr>
<tr>
<td>7</td>
<td>1.533</td>
<td>0.750</td>
<td>1.567</td>
<td>0.150</td>
</tr>
<tr>
<td>8</td>
<td>1.482</td>
<td>0.801</td>
<td>1.567</td>
<td>0.150</td>
</tr>
<tr>
<td>9</td>
<td>1.482</td>
<td>0.780</td>
<td>1.588</td>
<td>0.150</td>
</tr>
<tr>
<td>10</td>
<td>1.500</td>
<td>0.780</td>
<td>1.570</td>
<td>0.150</td>
</tr>
<tr>
<td>11</td>
<td>1.485</td>
<td>0.788</td>
<td>1.578</td>
<td>0.150</td>
</tr>
<tr>
<td>12</td>
<td>1.491</td>
<td>0.781</td>
<td>1.578</td>
<td>0.150</td>
</tr>
<tr>
<td>13</td>
<td>1.491</td>
<td>0.784</td>
<td>1.575</td>
<td>0.150</td>
</tr>
<tr>
<td>14</td>
<td>1.489</td>
<td>0.784</td>
<td>1.577</td>
<td>0.150</td>
</tr>
<tr>
<td>15</td>
<td>1.491</td>
<td>0.783</td>
<td>1.576</td>
<td>0.150</td>
</tr>
<tr>
<td>16</td>
<td>1.490</td>
<td>0.784</td>
<td>1.576</td>
<td>0.150</td>
</tr>
<tr>
<td>17</td>
<td>1.490</td>
<td>0.783</td>
<td>1.577</td>
<td>0.150</td>
</tr>
<tr>
<td>18</td>
<td>1.490</td>
<td>0.783</td>
<td>1.576</td>
<td>0.150</td>
</tr>
<tr>
<td>19</td>
<td>1.490</td>
<td>0.783</td>
<td>1.577</td>
<td>0.150</td>
</tr>
<tr>
<td>20</td>
<td>1.490</td>
<td>0.783</td>
<td>1.577</td>
<td>0.150</td>
</tr>
</tbody>
</table>

*Figure 13. Final ranks for 20 iterations*
CHAPTER 8: RELATIONAL SEARCH ENGINE (RSE)

Introduction to Search Engine Architecture

The user interface for RSE is based on the HTML form (HTML, n.d.). It can be invoked by standard WWW clients such as Internet Explorer, Mosaic, or Netscape. Search engine is an end product of the three mechanisms (Brin & Page, 1998):

- A mechanism that identifies web pages to be included in the database.
- A mechanism that indexes the sites.
- A search mechanism with an interface, which scans, for keywords within the index.

Users search the index (and hence, the database or web documents) through a query box or a template. Documents in which the search terms occur are presented as “hits.”

Figure 14 represents the architecture of the search engine. A user enters the keywords and clicks on the “submit” button to send the query to the search engine. The request is processed in the following order.
Figure 14. Architecture of Search engine Software

1) The web server (Apache, n.d.) receives the request from user. RSE uses Apache (Apache) as the web server.

2) The request is forwarded to the servlet engine. RSE uses Tomcat (Jakarta, n.d.) as the servlet engine.

3) Servlet forwards the request to Search Engine Processor module. Search engine processor module parses the query into individual words. This word set is forwarded to the Query processor.

4) Query Processor queries the databases for relevant documents.

5) The database is searched for matched web pages and the results are returned to the Query Processor.

6) Query Processes returns the results to the Rank Processor.

48
7) Rank Processor computes the rank based on the query words and passes the results to Quesry Sorter. Query Sorter sorts the results and hands the final result set to the servlet engine.

8) Servlet engine passes the final query set to template engine (*Velocity template engine*, n.d.).

9) *Velocity template* (*Velocity template engine*, n.d.) generates HTML page with the results embedded and passes it back to the servlet engine.

10) Servlet engine forwards the response to the web server.

11) The web server forwards the response page to the users browser.

*String matching*

Search engines use different algorithms to calculate the “relevance” of the search results. Relevance is the probability that the “hit” or “match” is on-target with the user’s query. The creators of search engines change the way they calculate relevance and do not reveal the methodology to public; being high in the major search engines’ rankings means big business.

RSE implements its ranking algorithm in many steps. First, the query is tokenized into set of segments and is parsed to remove the stop words listed in APPENDIX B if any. The words are then converted into WordID’s. WordID’s are the words stored in the WORD entity of the search engine database. The search is performed against the inverted index to find matching documents for the query terms. Another set of results are generated by performing a wild card search for all the query terms and the results are merged to get a final results set. The purpose of the wild card search is to increase the quality of returned results. For example, if the search phrase entered by the
user is “SCSU Computers,” then the search software searches for exact terms “SCSU” and “Computers” and build the first result set. Another process does a wild card search for all terms containing words “SCSU” and “Computers” hidden in them. This second result set is merged with the first one and the results are sorted based on the ranks. Duplicate results are removed and ranks are calculated for the results based on relevance. Results are sorted in ascending order of the rank and relative position of the word in the document. For a query set q and a list of documents d, the algorithm follows the pseudo code in Figure 15.

```
initialize all sim(q, d_i) = 0;
for each term t_j in q
    { find I(t) using the database relations;
        for each (d_i, w_ij) in I(t)
            sim(q, d_i) += q_j * w_ij;
    }
for each document d_i
    sim(q, d_i) = sim(q, d_i) * (1/|q|) * nfi(I);
sort documents in descending similarities and display the top k to the user;
```

Figure 15. Algorithm used to retrieve relevant documents for a given query

Software used

**Tomcat Servlet Engine**

In RSE, tomcat is used as a servlet engine. Tomcat (Jaakarta, n.d.) is the Jakarta Project's open-source servlet container and is released under the Apache Software License. Tomcat provides support for Java Servlet (Java servlet technology, n.d.) and JSP technologies. A servlet engine is a servlet-aware web-server which allows dynamic loading of servlets to process client requests. Servlets interact with clients via a request-response model based on HTTP. Since servlet technology works on top of HTTP, a servlet container must support HTTP as the protocol for client
requests and server responses. However, a servlet container also can support similar protocols such as HTTPS (HTTP over SSL) for secure transactions. Figure 16 shows the flow of request and response from the servlet container.

![Diagram showing the flow of HTTP request and response through a browser, servlet container, servlet, and static content.]

*Figure 16. Servlet Container functionality.*

A servlet is loaded in the memory by the servlet container when the servlet is requested for the first time. The servlet receives the forwarded user request, processes it, and returns the response to the servlet container, which in turn sends the response back to the user. From there on, the servlet stays in the memory waiting for other requests. Each time the servlet is requested, however, the servlet container will compare the timestamp of the loaded servlet with the servlet class file. If the class file timestamp is more recent, the servlet is reloaded into memory. By using this technique, Tomcat servlet container does not need to be restarted every time the servlet is updated. Figure 17 represents the functional flow of a servlet container.

*Velocity Template Engine*

RSE uses velocity (*Velocity template engine*, n.d.) as the template engine. Velocity (*Velocity template engine*) is a Java-based template engine. It can be used as a standalone utility for generating source code, HTML files, reports, or can be combined with other systems to provide template services. It is one of Jakarta's open source projects.
Velocity allows web page designers to embed simple yet powerful script elements in their web pages. These script elements work in conjunction with a context object, which is defined in Java code. A Context object is a java-supported Hashtable that provides “get” and “set” methods for retrieving and setting objects by name within the Context. Context object provides a “hook” from the Java code to the Velocity script elements. These script elements allow a web designer to retrieve objects from the Context and insert these into a web page as text values. The web designer has some control, by using iterations and conditional statements. The Velocity Template Engine allows users to render data from within applications and servlets.

Figure 17. Tomcat Servlet flow.
RSE uses the velocity template engine to render HTML files. Servlet engine passes the final result set to velocity template. Velocity template reads the final list of results and builds the HTML page in a human presentable format. The results page rendered by the velocity template is sent back to Tomcat servlet engine for the user display.
CHAPTER 9: CONCLUSIONS

Implementation issues

Dynamic Environment

In the case of changes to non-electronic document, the document needs to wait until all the changes are grouped for republishing. In case of electronic document, changes can be made and be effective immediately. When a Web page is modified, either by new addition or deletion, the search engine is not notified. As a result, search engines often have index information based on obsolete documents.

One way to deal with this problem is to let the crawler re-visit web sites periodically for updated pages. As the Southern Connecticut State University web site is very large and growing rapidly, it takes a lot of resources to visit all the pages at once. Currently, a newly added page may be added to the index of the search engine within a week’s time.

Spamming

Spamming refers to techniques that are employed to increase the chance of a page being retrieved by search engines even when the content of the page is not related to a given query. Spamming can have different forms. For example, popular terms that do not reflect the real content of a page may be added to the page so that when these popular terms appear in a query the page will have a better chance to be retrieved. As an example, if there is a current hot topic, the
keywords related to that hot topic may be added to the page as spamming terms. Furthermore, the popular terms could be repeated many times in a page so that they will have high term frequencies. As term frequency weight is widely used by search engines and page rank is an increasing function of term frequency, this increases the likelihood for the page to be retrieved. In order not to affect the presentation of the main content of a page, spamming terms are de-emphasized or hidden from viewers. Common tricks are: using tiny font for spamming terms and place them as the end of the web page, place spamming terms in comments so that they would not show up for viewers and put spamming terms in the color which is identical to the background color of the page so that they would not be visible. Spamming may also appear in the form of links. For example, in order to artificially increase the popularity and/or authority of a page, the author may create many dummy pages with links to the page. A good search engine should employ retrieval techniques that can either eliminate or significantly curb the influence of spamming so that better quality documents can be returned to users.

In RSE, the indexer of the search engine chooses to ignore terms in comments and terms that have the same color as the background, and ignore or significantly reduce the weights of terms that are in very small fonts.

Other Search engines

There are many robot-based WWW index and search services on the Internet today. However, among the well-known robots, only a few employ full-text indexing, e.g., WebCrawler (Webcrawler search engine, n.d.), the Repository Based Software Engineering Project Spider (RBSE) (RBSC Spider, n.d.), and Lycos (Lycos search engine, n.d.). Other services index only
page titles and first level headings, (e.g., JumpStation *(JumpStation, n.d.)*) or titles headings and anchor hypertexts, (e.g., World Wide Web Worm or WWWW *(World wide web worm, n.d.)*) or only first 100 words in a text (e.g., Google *(Google, n.d.)*).

The Relational Search Engine’s (RSE) index builder considers other HTML tokens such as words in boldface or italics, titles, all headings and anchor hypertexts. RSE is a full-text search engine taking advantage of HTML meta-information as much as possible. RSE extracts the links that are hidden in HTML lists on web pages.

*Scalability*

The RSE is scalable to the World Wide Web. Currently the indices occupy around 10 Gigabytes of space and it can grow with more space provided. It can be configured either to crawl for a specific organization or for global web. The RSE is also portable to any relational database. The database connection properties are externalized so that the search engine can be adapted to work with any database without having to recompile the code.

*Results*

Results obtained from RSE are compared to the results obtained from the existing search engine used by the Southern Connecticut State University web site (Google.com) using the same search words.

*Evaluation Methodology*. A set of 30 queries is compiled based on the requirements of various uses and tested on both search engines. The query set is divided into three groups
consisting of ten single word queries, ten double word queries and ten triple word queries. Table 2 represents the list of queries used to evaluate RSE. First ten results that are returned by the search engines are ranked based on the following factors:

- Relevance and irrelevance of the content.
- Broken links
- Duplication of hits.

Table 2.

*List of queries used for RSE evaluation*

<table>
<thead>
<tr>
<th>Word ID</th>
<th>One word Queries</th>
<th>Two word Queries</th>
<th>Three word Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Seyed</td>
<td>Summer Courses</td>
<td>Master of Science</td>
</tr>
<tr>
<td>#2</td>
<td>Admissions</td>
<td>Computer Science</td>
<td>Hoot loot Card</td>
</tr>
<tr>
<td>#3</td>
<td>Summer</td>
<td>Extended Learning</td>
<td>Football match schedule</td>
</tr>
<tr>
<td>#4</td>
<td>Thesis</td>
<td>English Department</td>
<td>Off Campus housing</td>
</tr>
<tr>
<td>#5</td>
<td>Graduate</td>
<td>Undergraduate Admissions</td>
<td>International Student Application</td>
</tr>
<tr>
<td>#6</td>
<td>Tuition</td>
<td>International Studies</td>
<td>Dining Hall Location</td>
</tr>
<tr>
<td>#7</td>
<td>English</td>
<td>Library Science</td>
<td>How to get Parking Permit</td>
</tr>
<tr>
<td>#8</td>
<td>Research</td>
<td>Course Schedules</td>
<td>Graduation Ceremony Schedule</td>
</tr>
<tr>
<td>#9</td>
<td>Library</td>
<td>Registration Services</td>
<td>Homecoming Weekend activities</td>
</tr>
<tr>
<td>#10</td>
<td>Undergraduate</td>
<td>Parking Facilities</td>
<td>Commencement Weekend Activities</td>
</tr>
</tbody>
</table>

57
Evaluation Criteria. After a query is submitted to a search engine, results are extracted and the first ten results are used for the evaluation. A grade is assigned to each result produced by the search engine. The grades used are A, B, C, X and D. The criteria to assign each mark to a specific result is detailed below:

- A: corresponds to the best possible result, which is the result that best covers the query. This category involves web pages that thoroughly cover the requested query by subject. For example, if the actual query is “Graduate Admission Process,” an A category result could be a page containing details on Graduate Admission.

- B: corresponds to a fairly relevant result that partially covers the query theme, or a result that contains a link to an A-type page. For example, for the aforementioned query, a B category result could be a page containing information on the Registration and may contain a link to the admission procedures.

- C: corresponds to an irrelevant result with respect to the submitted query. That is a result that does not cover the particular query terms, but it did appear within the search results. For example, a page that belongs to “Department Of Communication Disorders,” where words “admission” and “process” are frequently matched.

- X: corresponds to errors of the servers or broken links, therefore indicates that no result was returned.

- D: corresponds to duplicate results.

Table 3 represents the points assigned to each mark detailed above.
Table 3.

Points assigned for evaluation marks

<table>
<thead>
<tr>
<th>Mark</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5 Points</td>
</tr>
<tr>
<td>B</td>
<td>3 Points</td>
</tr>
<tr>
<td>C</td>
<td>1 Point</td>
</tr>
<tr>
<td>X</td>
<td>0 Points</td>
</tr>
<tr>
<td>D</td>
<td>Points/number of duplicates</td>
</tr>
</tbody>
</table>

Results. The results returned from each search engines are collected and evaluated by assigning a mark to each of the first ten results returned. The total marks given by two different evaluators are averaged. Results and graph representations of results are detailed below:

1) Results for single word searches: Table 4 represents the average grades obtained by RSE and Google for single word queries.

Table 4.

Average amount of grades scored by search engines for single word queries

<table>
<thead>
<tr>
<th>Word ID</th>
<th>Average Doc Search</th>
<th>Average Image Search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average RSE Grade</td>
<td>Average Google Grade</td>
</tr>
<tr>
<td>#1</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#2</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#3</td>
<td>50</td>
<td>45</td>
</tr>
<tr>
<td>#4</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>#5</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#6</td>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td>#7</td>
<td>45</td>
<td>50</td>
</tr>
<tr>
<td>#8</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#9</td>
<td>46</td>
<td>43</td>
</tr>
<tr>
<td>#10</td>
<td>46</td>
<td>49</td>
</tr>
</tbody>
</table>

59
Figure 18 represents a chart of the average grade obtained for each single word searches listed in Table 2. The overall score of RSE in single word queries 1, 2, 3, 5, 6, 8 and 9 are almost the same or better than the Google search engine.

![Document Search Results For Single Word Queries](image)

*Figure 18. Graph representation of document search results from RSE and existing southernct.edu search engine (Google)*

Figure 19 represents a chart for image search results returned by RSE and Google for the 10 single word queries listed in Table 2. RSE gave better results for queries 1, 2, 4, and 6 when compared to the southernct.edu’s Google search engine. Results for queries 5, 7, 8, 9, and 10 are almost the same for both search engines. For search query 1, RSE returned more relevant result than Google. The original search result pages from both search engines are presented in APPENDIX E.
2) Results for two word searches: Table 5 presents the average grade obtained on document and image search results for all the ten two word search queries listed in Table 2.

Figure 20 is the chart representation of the document search results for the two word search queries. As represented in the graph, RSE returned almost the same or better results for double word queries 1, 2, 3, 5, 6, 7, 8 and 9. For search query “Extended Learning” (query 3), Google returns 50% of duplicate pages in the first 10 results. RSE returned more relevant results for query “International Studies” (query 6). The original results returned from RSE and Google for the same query are included in APPENDIX E.
Table 5.

*Average amount of grades scored by search engines for double word queries*

<table>
<thead>
<tr>
<th>Word ID</th>
<th>Average Doc Search</th>
<th>Average Image Search</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average RSE Grade</td>
<td>Average Google Grade</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#1</td>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>#2</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>#3</td>
<td>50</td>
<td>35</td>
</tr>
<tr>
<td>#4</td>
<td>30</td>
<td>48</td>
</tr>
<tr>
<td>#5</td>
<td>50</td>
<td>45.25</td>
</tr>
<tr>
<td>#6</td>
<td>49</td>
<td>40.5</td>
</tr>
<tr>
<td>#7</td>
<td>46</td>
<td>50</td>
</tr>
<tr>
<td>#8</td>
<td>48</td>
<td>46.25</td>
</tr>
<tr>
<td>#9</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#10</td>
<td>25.5</td>
<td>46</td>
</tr>
</tbody>
</table>

*Figure 20. Graph representation of document search results for two word queries*

Figure 21 represents the chart with image search results for the same set of ten double word queries listed in Table 2. RSE results for double word queries 1, 8 and 9 show better results than
the existing southernct.edu’s Google search engine. RSE image search results for query “Computer Science” (query 2) are more relevant than image search results returned by Google. The original results returned by both search engines are included in APPENDIX E.

**Figure 21.** Graph representation of image search results for two word queries

3) Results of three word searches: Table 6 represents the average grade obtained on document search results for all the ten three word search queries listed in Table 2.

Figure 22 represents document search results for the ten three word queries listed in Table 2. As represented in the chart, RSE returned almost the same or better results for triple word queries 1, 2, 3, 6, 8, 9 and 10. RSE returned the best relevant results for query “Football match schedule” (query 3) compared to irrelevant results returned by Google. For another search term “Commencement Weekend Activities” (query 10), RSE returned more relevant results that
reflect the search query than Google. Original results for both search queries returned by RSE and Google are included in APPENDIX E.

Table 6.

*Average amount of grades scored by search engines for triple word queries*

<table>
<thead>
<tr>
<th>Word ID</th>
<th>Average RSE Grade</th>
<th>Average Google Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>#2</td>
<td>46</td>
<td>50</td>
</tr>
<tr>
<td>#3</td>
<td>22</td>
<td>17.5</td>
</tr>
<tr>
<td>#4</td>
<td>40</td>
<td>49</td>
</tr>
<tr>
<td>#5</td>
<td>34</td>
<td>50</td>
</tr>
<tr>
<td>#6</td>
<td>26</td>
<td>20</td>
</tr>
<tr>
<td>#7</td>
<td>46</td>
<td>50</td>
</tr>
<tr>
<td>#8</td>
<td>14</td>
<td>6</td>
</tr>
<tr>
<td>#9</td>
<td>28</td>
<td>30</td>
</tr>
<tr>
<td>#10</td>
<td>48</td>
<td>12</td>
</tr>
</tbody>
</table>

*Figure 22. Graph representation of document search results for three word queries*
RSE also provides media search, which allows users to search for any multimedia files available within southernct.edu domain. For example, a search for "Public Health" on Google only returns relevant document and images. The media that matches to "Public Health" query is not recognized in Google results unless the user is aware of the media filename and searches specifically for its name. RSE on the other hand returns all the media files that match "Public Health." The original results are included in APPENDIX E.

RSE provides e-mail search that allows users to search for person’s e-mail address. For example to search for the e-mail address of a person, the user just needs to know the name of the person. RSE does not restrict users by requiring complete last name of the searched person. The user can also search e-mail address using the most general terms like “graduate office” for instance, to get the e-mail addresses of the contacts in the graduate office. Google does not provide search for person’s e-mail address. Southernct.edu uses its own “MySCSU E-mail search” which mandates the user to know the last name of the searched person. The results are included in APPENDIX E.
CHAPTER 10: FUTURE WORK

More dynamic systems

Web is an ever-changing arena. New pages are added and existing pages are updated. One important area, which requires much research, is discovering new or updated web pages. There is a need to devise intelligent algorithms to decide on pages that need to be re-crawled and the new pages that should be crawled.

Natural Language parsing

Although some facilities incorporate natural language searching (searching by asking a question “Where are the doughnuts?”), most search tools retrieve “hits” or “matches” by seeking occurrences of the search terms within its database and by attempting to match the terms against its index. Because the terms are converted to a string, the search engine must somehow be instructed to include plurals and alternate forms of a term. The goal of applying the Natural Language Processing to search engine is to analyze, understand, and generate languages that humans use naturally, so that more relevant results can be returned.

Latent Semantic Indexing

Regular parsers search a document collection with an accountant mentality: a document contains a given word or it doesn’t, without any middle ground. We create a result set by looking
through each document for certain keywords and phrases, tossing aside any documents that do not contain them, and ordering the rest based on some ranking system. Each document stands alone in judgment before the search algorithm - there is no interdependence of any kind between documents, which are evaluated solely on their contents.

Latent semantic indexing (LSI) (Deerwester & Dumais, 1998) adds an important step to the document indexing process. In addition to recording which keywords a document contains, the method examines the document collection as a whole, to see which other documents contain some of those same words. LSI considers documents that have many words in common to be semantically close, and ones with few words in common to be semantically distant. This simple method correlates surprisingly well with how a human being, looking at content, might classify a document collection. Although the LSI algorithm has no knowledge of the meaning of the words, the patterns it discovers can make it look astonishingly intelligent. The main reasons for missing relevant information is that there are surprisingly many different ways to describe the same idea or concept. If a document author uses one word and a searcher another, relevant materials will be missed. A query about “laptop” computers, for example, will fail to find articles about “portable” or “lightweight” or “notebook” or “palmtop” or “ThinkPad” computers. Searchers and authors alike find it very difficult to anticipate the many ways in which the same idea might be described. By automatically constructing a semantic or concept space, LSI enables users to find relevant information even when it shares no words with their queries. It requires no additional work by either the searcher to painstakingly describe their needs or by the content provider to carefully handcraft a thesaurus or knowledge base. LSI uses a powerful and fully automatic statistical method (singular value decomposition) to uncover the associations among terms in a large
collection of texts, to create a semantic or concept space, and to exploit this to improve retrieval. Given the gigantic size of web, LSI is a solution in conceptualizing search engines.

Relevance Feedbacks and Clustering

Relevance feedback is an effective method for iteratively improving a query without increasing the computational requirements to perform the query. Relevance feedback uses the terms contained in relevant documents to supplement and enrich the user’s initial keyword query, allowing greater retrieval performance (in terms of precision and recall) (Huang & Zhou, 2001). The goal is also to analyze and group web pages as well as keywords into conceptual clusters. By identifying keyword clusters, we can expand a query by adding more keywords belonging to the same cluster as that of the keywords in the original query. Page clusters can be used to identify other pages sharing the same cluster with the pages judged as relevant by the user. Other uses of page clustering like automatic catalog generation; index database reduction/compression and multi-server indexing can be explored as well.
APPENDIX A

Source Code

1. com.scsu.searchengine.controller.CrowdController.java
package com.scsu.searchengine.controller;
/*

Class CrowdController (C) Swathi K. email: swathikanury@yahoo.com
-----------------------------
This re-usable class enforces a restricted number of clients simultaneously
using some unspecified resource.

Say if you had an open-ended (unlimited) recursive mechanism spawning
threads, you would want to ensure that only so many threads in total are
active at any one time. Class CrowdController can enforce such a limitation
by handing out "tickets" (or licenses) to clients, so that they can "proceed"
with whatever they wish. Entities applying for a ticket when no more tickets
are available will block until a ticket is released by another entity.
*/

import com.scsu.searchengine.logging.Logger;

public class CrowdController {

static Logger LOGGER = Logger.instance(CrowdController.class.getName());

static int ticketDatabase[]; // array of valid issueable tickets
static int crowdSize; // current crowd size
static int crowdWaitSize; // current wait crowd size
static int maxCrowdSize; // max crowd size to enforce
static int maxWaitSize; // max number os threads that can wait.

public static CrowdController instance = null;

//----------------------------------------------------------------------------
// Constructor.
// Given the maximum size of a crowd to control, initialize the crowd to
// empty and mark all tickets as "not issued".
//----------------------------------------------------------------------------
private CrowdController(int maxCrowdSize, int maxWaitSize) {
    this.maxCrowdSize = maxCrowdSize; // crowd size to enforce
    this.maxWaitSize = maxWaitSize;

70

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LOGGER.trace("Max Crowd Size = "+this.maxCrowdSize);
LOGGER.trace("Max Crowd Wait Size = "+this.maxWaitSize);

crowdSize = 0;       // current size of crowd
crowdWaitSize = 0;

ticketDatabase = new int[maxCrowdSize];
for(int i=0; i < maxCrowdSize; i++) {
    ticketDatabase[i] = -1;     // mark all ticket IDs as available
}
}

// This Synchronized instance takes care of maintaining exactly one
// instance of CrowdController.

public static synchronized CrowdController getInstance(int size, int wSize) {
    if(instance == null)
        instance = new CrowdController(size, wSize);
    return instance;
}

// Apply for a ticket.
// If tickets are still available, give client a ticket and record its
// issue.
// If all tickets are currently issued, block (put client thread to sleep)
// until a ticket becomes available again (and gets recycled).
// Tickets can only be returned via the returnTicket() method.

public synchronized int getTicket() {

    if(crowdSize >= maxCrowdSize) {
        crowdWaitSize++;
        LOGGER.trace("Increasing the CrowdWaitSize to :") + crowdWaitSize);
    }

    // if crowd is already at full capacity: wait until crowd shrinks
    while (crowdSize == maxCrowdSize) {

        LOGGER.trace("Crowd too big.... you have to wait (" +
            Thread.currentThread().getName() + ")" + "tCrowdWaitSize is ":+crowdWaitSize);

        try {
            // wait for a ticket
        }
    }
wait();
} catch (InterruptedException leaveUsAlonePlease) {
}

// OK. crowd has shrunk a bit, so grab a ticket

int ticket = findFreeTicket();
if (ticket == -1) {
    LOGGER.trace("Bug: could not find free ticket when we should be able to.");
}
ticketDatabase[ticket] = ticket; // record ticket issue
crowdSize++; // track # of tickets out there

return ticket;

// A client has done whatever it needed to do and wants to give up its
// ticket. Take ticket back and recycle it for future clients. Wake up
// a random thread waiting to get a ticket of its own.
public synchronized void returnTicket(int ticket)
    throws IllegalArgumentException {

    String illegalTicket = "Alert: illegal ticket ID seized! (id= " + ticket + ")";

    if (crowdSize == 0) {
        throw new IllegalArgumentException(illegalTicket + " because crowdSize was 0");
    }

    if (ticketDatabase[ticket] != ticket) {
        throw new IllegalArgumentException(illegalTicket + " because ticket ID is not issued!");
    }

ticketDatabase[ticket] = -1; // mark ticket as available again.
crowdSize--;

crowdWaitSize--;
    if(crowdWaitSize > 0) { // reduce the wait size
        LOGGER.trace("Decrementing the crowdWaitSize to 1 :");
    }

    notifyAll(); // wake up a thread needing a ticket
}
protected int findFreeTicket() {

    for(int i=0; i < maxCrowdSize; i++) {
        if (ticketDatabase[i] == -1) {
            return i;
        }
    }
    return -1;
}
} // End of Class CrowdController

2. com.scsu.searchengine.controller.WebExplorer.java
package com.scsu.searchengine.controller;

import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Locale;
import java.util.StringTokenizer;

import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.util.SearchProperties;

/*

Program: WebExplorer (C) 2003 Swathi Kanury

This Java application will, given an initial Web page URL, visit this page and recursively visit all* pages referenced by it. The names of the pages are listed on the console.

* = this implementation only hops to absolutely referenced pages using anchor tags like <A HREF="http://...">.
Relative references (usually to documents on the same site) are not followed at all. The prime goal of this program was to visit as many sites as possible, and not to comprehensively chart every possible page.
public class WebExplorer {  // is an application, not an applet!

    final static String title = "WebExplorer v1.0 (c) 2003 K. Swathi (swathikanury@yahoo.com)";
    final static String title2 = "-----------------";

    static Logger LOGGER = null;

    public static void main(String[] args) {
        LoggerInit.initLogger();
        LOGGER = Logger.instance(WebExplorer.class.getName());
        LOGGER.trace(title);
        LOGGER.trace(title2);

        LOGGER.trace("Ready.. ");

        new WebExplorer().doIt(args);
    }

    public void doIt(String[] args) {
        /*
         * if (args.length == 0) {
         *     System.out.println("Usage: WebExplorer <URL>");
         *     System.exit(0);
         * }*/

        // create the root instance of a PageVisitor thread. This thread
        // will load the initial page and spawn more threads for every link
        // found in that page.
        //LOGGER.trace("pageDBFile = " + (String)SearchProperties.getInstance().getProperty("levelcrawler"));
        File file = new File((String)SearchProperties.getInstance().getProperty("levelcrawler"));
        boolean done = false;
        Vector list = new Vector();
        if(file.exists()) {
            done = true;
            Vector incomplete = new Vector();
            Hashtable complete = new Hashtable();
            String str = null;
            try {
                FileReader fr = new FileReader(file);
            } catch (FileNotFoundException e) {
                System.out.println("FileNotFoundException: ");
            }
        }
    }
}
BufferedReader br = new BufferedReader(fr);
do {
    str = br.readLine();
    list.add(str.trim());
    LOGGER.trace(str);
} while(str != null);
fr.close();
br.close();
} catch(Exception e) {
}

if(list.size() == 0) {
    String home = (String) SearchProperties.getInstance().getProperty("homepage");
    //list.add(home);
    StringTokenizer st = new StringTokenizer(home, ",");
    while(st.hasMoreTokens()) {
        String token = st.nextToken();
        list.add(token);
    }
    new PageVisitor(list);
}
} // End of Class WebExplorer

package com.scisu.searchengine.controller;

import java.util.*;
import java.io.*;
import java.net.*;
import java.text.*;
import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;

import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLDocument
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.parser.ParserDelegator;

import oracle.jdbc.driver.OracleTypes;

import org.apache.oro.text.perl.Perl5Util;

import com.scisu.searchengine.processors.*;
import com.scsu.searchengine.database.FileStorage;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.dbobjects.UrlDB;
import com.scsu.searchengine.dbobjects.WordDB;
import com.scsu.searchengine.dbobjects.ImageDB;
import com.scsu.searchengine.dbobjects.MediaDB;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.util.HTTP;
import com.scsu.searchengine.util.SearchProperties;
import com.scsu.searchengine.util.DocTypes;
import com.scsu.searchengine.util.LogException;

public class PageVisitor extends Thread {

    static Logger LOGGER = Logger.instance(PageProcessor.class.getName());

    private final static int MAX_THREADS =
        Integer.parseInt((String)SearchProperties.getInstance().getProperty("spider_threads"));

    private final static int MAX_WAIT_THREADS =
        Integer.parseInt((String)SearchProperties.getInstance().getProperty("spider_wait_threads"));

    // The following CrowdController object is critical to the operation
    // of the entire program: it enforces a maximum number of entities (here:
    // threads). The object is declared static so that only one instance is ever
    // created. All PageVisitor threads therefore have one single "cop" which
    // limits the number of threads (without this cop, the recursive algorithm
    // and hypertext structure of the Web would lead to an exponential explosion
    // in Threads.. a situation which would quickly lead to program failure.

    static CrowdController threadLimiter = CrowdController.getInstance( MAX_THREADS, MAX_WAIT_THREADS);

    // The following global hashtable is declared static so that only one
    // instance is ever created. All PageVisitor threads therefore access this
    // same hashtable which acts as a database of page URLs already encountered.

    /*Writing the urls to a file instead of the hashtable to avoid out of memory exceptions*/
    //static LinkedHashMap pageDatabase = new LinkedHashMap(); // empty to begin with

    public static FileWriter pdwriter = null;

    static Vector pageLinks = new Vector(); // bag to accumulate found URLs in
URL pageToFetch;  // communication var between go(..) and run()

static String homeIP = null;
static Vector homeIPs = new Vector();
static boolean singleIP = false;
static URL _home = null;
static final String GETRELATED_URL = "{call search_pkg.get_related_url(??.?)}";

//----------------------------------------------------------------------
// Constructor
// Given a String address of a Web page, transform address to URL object
// and launch the thread for this page URL. If the URL could not be built,
// don't even bother creating the thread (silently return).
//----------------------------------------------------------------------

public PageVisitor (String pageAddress) {

try {
    pageToFetch = new URL(pageAddress);
    setName(pageAddress);  // label this thread with the page name
    start();  // start the thread at run()
} catch (MalformedURLException badURL) {
    LogException.handleError(pageAddress + " is bad URL.. not starting thread for this one!",badURL);
} catch (OutOfMemoryError ofm) {
    LOGGER.error("Out of Memory for url = "+pageAddress,ofm);
    //System.gc();
    try {
        pdwriter.write(pageAddress+"invalid");
    } catch(Exception e1) {
        LogException.handleError("Error writing invalid url = "+pageAddress+" to the pagedatabase"",e1);
    }
    System.exit(0);
}

}  

public PageVisitor(Vector incomplete) {
    pageLinks.addAll(incomplete);
    if(pageLinks.size() > 0)
        new PageVisitor((String)pageLinks.get(0));
}

//----------------------------------------------------------------------
// The body of the thread (and also the heart of this program)

77
// in pseudo-code:
//
// request running permission (by obtaining a ticket)
// load pageToFetch
// extract hypertext links from page
// for all found links do
//   if new link
//     record link in database (so that we avoid it next time)
//     recurse for new page (launch new thread running same algorithm)
// release ticket
// die (thread terminates)
//
public void run() {

int ticket;       // thread can only run for real if it gets a ticket
String webPage;   // an entire Web page cached in a String

// before processing the page (and possibly spawning new threads to
// process referenced pages), obtain a ticket (i.e. a license) to do
// so, (getTicket()) will block if max number of threads are
// already running. When another thread releases its ticket, only then
// will a blocked thread be able to run() for real.

    ticket = threadLimiter.getTicket();
    LOGGER.trace("Thread " + getName() + " got ticket "+ ticket);

    if(homeRelatedPage(pageToFetch.toString())) {
        webPage = loadPage(pageToFetch);

        Hashtable hash = parseHTML(webPage, pageToFetch);
        Vector plinks = (Vector) hash.get("links");
        Vector words = (Vector) hash.get("words");
        Vector images = (Vector) hash.get("images");
        Vector media = new Vector();

        LOGGER.trace(getName() + "Size of plinks = " + plinks.size());
        LOGGER.trace(getName() + "Size of words = " + words.size());
        LOGGER.trace(getName() + "Size of images = " + images.size());

        int j=0;
        for(int i=0;i<plinks.size();i++) {
            URLProcessor up = (URLProcessor) plinks.get(i);
            if(up.getDocTypeID() == 1 && !up.getIsBase()) {
                j++;
            }
pageLinks.add(up.getUrlDesc());
}
if(up.getDocTypeID() == DocTypes.MEDIA)
    media.add(up);

//FileStorage.storeData(plinks,words, images);
UrlDB.getInstance().processUrls(plinks);
ImageDB.getInstance().processImages(images);
MediaDB.getInstance().processMedia(media);
WordDB.getInstance().processWords(words);
LOGGER.trace(getName() + " has " + j + " links." + " Total size of unvisited links is = " +
    pageLinks.size());
}
try {
// Now process all found URLs
for(int i=0;i<pageLinks.size();i++) {
    String page = (String) pageLinks.get(i);
    if (!alreadyVisited(page)) {

        if(homeRelatedPage(page)) {
            // print hypertext link relationship to console
            LOGGER.trace(getName() + " --> " + page);

            // and recursively start up a new thread to handle new page
            if(threadLimiter.crowdWaitSize < MAX_WAIT_THREADS) {
                markAsVisited(page,i);
                new PageVisitor(page);
            } else {
                // The wait threads are full so we should push this back into the pageLinks list.
                LOGGER.trace("Wait threads are full. Adding back to Queue :"+page);
            }
        }
    pageLinks.remove(i);
    pageLinks.add(page);
}
} else {
    markAsVisited(page,i);
    // remove from pageLinks as it is visited once
    // pageLinks.remove(i);
    LOGGER.trace(getName() + "--Not Home Related: " + page);
}
} else {
    // remove from pageLinks as it is visited once
    pageLinks.remove(i);
    LOGGER.trace(getName() + "--Already visited: " + page);
}

} finally {
    // We're done with our page, so release ticket and let another thread
    // process a page (returning our ticket will un-block some other
    // waiting thread.
    LOGGER.trace("Thread " + getName() + " releases ticket " + ticket);
    threadLimiter.returnTicket(ticket);
}

// insert some brief, random delay here before running off the end of the
// world (of run()). This randomizes the scheduling of threads waiting to
// obtain a ticket.

try {
    Thread.sleep( (int) (Math.random()*200) );
} catch (Exception e) {
    LogException.handleError("Exception in PageVisitor.run()",e);
}

// pretended code

// Given a valid WWW page URL, fetch and return the page as a big String.
// pretended code

protected String loadPage(URL page) {
    try {
        HTTP http = new HTTP();
        String str = http.downloadWWWPage(page.toString());
        return str;
    } catch (Exception e) {
        LogException.handleError("Problem reading page from url = "+page.toString()+" \nthis will be revalidated again... check out...",e);
        try {
            pwriter.write(page.toString()+"[invalid]");
        } catch(Exception e1) {
            LogException.handleError("Error writing invalid url = "+page.toString()+" to the pagedatabase",e1);
        }
    }

}
private Hashtable parseHTML (String str, URL url) {
    HTMLEditorKit htmlKit = new HTMLEditorKit();
    HTMLEDitorKit.htmlDoc = (HTMLEDitorKit.htmlKit.createDefaultDocument());
    HTMLEDitorKit.Parser parser = new ParserDelegator();
    Hashtable hash = new Hashtable();

    /*
    * Split html into 2 pieces
    * First one is the right nav
    * Second one is the center portion
    * If the page cannot be split into 2 then pass it as one html to
    * PageProcessor for regular parsing
    */
    Vector v = new Vector();
    Perl5Util putil = new Perl5Util();
    putil.split(v,"/<!--right col-->","",str,Perl5Util.SPLIT_ALL);
    if(v.size() > 1) {
        /*which means the split is successful*/
        StringReader leftnav = new StringReader((String) v.get(0));
        LeftNavProcessor ln = new LeftNavProcessor(url);
        try {
            parser.parse(leftnav, ln, true);
        } catch(Exception ioe) {
            ioe.printStackTrace();
        }
    }

    Vector _ilinks = ln.getLinks();
    Vector _iwords = ln.getWordList();
    Vector _iimages = ln.getImageList();

    //strip out the footer
    String part2 = (String) v.get(1);
    Vector v2 = new Vector();
    putil.split(v2,"/<!--FOOTER TABLE -->","",part2,Perl5Util.SPLIT_ALL);

    StringReader ctr_content = new StringReader((String) v2.get(0));
    CtrContentProcessor ccp = new CtrContentProcessor(url);
    try {
        parser.parse(ctr_content, ccp, true);
    } catch(Exception ioe) {

    }}
ioe.printStackTrace();
}

_ilinks.addAll ccp.getPageLinks();
_iwords.addAll ccp.getWordList();
_iimages.addAll ccp.getImageList();

hash.put("links", _ilinks);
hash.put("words", _iwords);
hash.put("images", _iimages);

} else {
    /* which means the page is from a department rather than from SCSU directly */
    PageProcessor pp = new PageProcessor(url);

    // strip out the footer
    Vector v2 = new Vector();
    putil.split(v2, "<!--FOOTER TABLE -->", str, Perl5Util.SPLIT_ALL);
    java.io.StringReader br = null;
    if(v2.size() > 0)
        br = new java.io.StringReader((String)v2.get(0));
    else
        br = new java.io.StringReader(str);
    try {
        parser.parse(br, pp, true);
    } catch(Exception ioe) {
        ioe.printStackTrace();
    }
    hash.put("links", pp.getPageLinks());
    hash.put("words", pp.getWordList());
    hash.put("images", pp.getImageList());
}
return hash;
}

// Find out whether a page has already been encountered in the past.
protected boolean alreadyVisited(String pageAddr) {
    //LOGGER.trace("Searching for "+pageAddr.toLowerCase(Locale.ENGLISH));

    if(UUrlDB.getInstance().isPageCrawled(pageAddr))
        return true;
    return false;
}
protected void markAsVisited(String pageAddr, int i) {
    //LOGGER.trace("Inserting " + pageAddr);
    try {
        pwriter.write(pageAddr + "\n");
    } catch(Exception e) {
        LogException.handleError("Exception while writing into the pagedatabase file", e);
    }
    pageLinks.remove(i);
}

private boolean homeRelatedPage(String page) {
    URL url = null;
    try {
        url = new URL(page);
    } catch (Exception e) {
        LogException.handleError("Malformed URL " + page, e);
    }
    String host = url.getHost();
    Vector v = SearchProperties.getInstance().getHosts();
    for (int i = 0; i < v.size(); i++) {
        String host2 = (String) v.get(i);
        if (host.endsWithWith(host2))
            return true;
    }
    return false;
}

static {
    try {
        //load pagedatabase
        pwriter = new FileWriter((String) SearchProperties.getInstance().getProperty("pageDatabaseFile"), true);
        pwriter.write("--------------" + new Date().toString() + "--------------\n\n\n\n")
        //This starts the pageDatabase file - that holds all the urls visited by the crawler.
    }
}
public static void main(String args[]) {
    Vector v = new Vector();
    v.add("http://www.futurefocus.net/scsu/index.htm");
    PageVisitor p = new PageVisitor(v);
}

// End of Class PageVisitor

package com.scsu.searchengine.database;
import java.util.Vector;
import java.io.FileWriter;
import com.scsu.searchengine.processors.URLProcessor;
import com.scsu.searchengine.processors.WordProcessor;
import com.scsu.searchengine.processors.ImageProcessor;
import com.scsu.searchengine.util.LogException;
/*
File Storage class stores all the urls, words, images, media found
In the crawling process. This documentation can be turned off in PageVisitor.
*/
public class FileStorage {
    private static Vector links = new Vector();
    private static Vector words = new Vector();
    private static Vector images = new Vector();
    public FileStorage() {
        }
    public static void storeData(Vector plinks, Vector words, Vector images) {
        try {
            //mergeLinkVectors(plinks);
            FileWriter fwriter = new FileWriter("/root/swathi/src/href.txt",true);
            fwriter.write("URL_ID | URL_DESC | URL_REF_ID | DOC_TYPE_ID");
            fwriter.write("n");
            for(int i=0;i<plinks.size();i++) {
                URLProcessor up = (URLProcessor) plinks.get(i);
            }
        }
}

84

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
fwriter.write(up.getUrlID()+" | "+up.getUrlDesc()+" | "+up.getUrlRefID()+" | 
+up.getUrlTypeID());
fwriter.write("n");
}
fwriter.close();
fwriter = new FileWriter("/root/swathi/src/words.txt", true);
fwriter.write("WORD_ID | URL_ID | WORD_DESC | TITLE | REL_POS | CAPITAL | ANCHOR | STYLE");
fwriter.write("n");
for(int i=0;i<words.size();i++) {
    WordProcessor wp = (WordProcessor) words.get(i);
    fwriter.write(wp.getUrlID()+" | "+wp.getUrlDesc()+" | "+wp.getUrlRefID()+" | "
+wp.getTitle()+ "+"+wp.getRelPosition()+" | "+wp.getCapital()+" | "+wp.getAnchor()+ " | " +
wp.getStyle());
    fwriter.write("n");
}
fwriter.close();
fwriter = new FileWriter("/root/swathi/src/images.txt", true);
fwriter.write("URL_ID | URL_DESC | REF_ID | SIZE | DIM | LABEL");
fwriter.write("n");
for(int i=0;i<images.size();i++) {
    ImageProcessor ip = (ImageProcessor) images.get(i);
    fwriter.write(ip.getUrlID()+" | "+ip.getUrlDesc()+" | "+ip.getUrlRefID()+" | "
+ip.getSize() + " | " + ip.getDimensions()+ " | " +ip.getLabel());
    fwriter.write("n");
}
fwriter.close();
} catch (Exception e) {
    IOException.handleError("Exception from storing data to file",e);
}

private void mergeLinkVectors(Vector plinks) {
    for(int i=0;i<plinks.size();i++) {
        URLProcessor up = (URLProcessor) plinks.get(i);
        if(checkDuplicate(up))
            plinks.remove(i);
    }
}

private boolean checkDuplicate(URLProcessor up) {
    String desc = up.getUrlDesc();
    for(int i=0;i<links.size();i++) {
        URLProcessor_up = (URLProcessor)links.get(i);
        if(_up.getUrlDesc().equalsIgnoreCase(desc))
            return true;
    }
    return false;
}
return true;
}
return false;
}

5. com.scu.searchengine.database.ConnectionManager.java
package com.scu.searchengine.database;
import java.util.*;
import java.sql.*;
import java.io.*;
import oracle.jdbc.*;
import com.scu.searchengine.logging.Logger;
import com.scu.searchengine.util.LogException;
/**
 * a simple wrapper class that encapsulates a DataSource.
 * this one is oracle specific. really the only useful generic methods this
 * exposes are getConnection() and getConnection(String).
 * the implementation is backed by an OracleConnectionCache created out of an
 * OraclePooledConnectionDataSource.
 * the OracleConnectionCache in turn is implemented by Oracle's own
 * OracleConnectionCacheImpl which so far seems to be good enough.
 * note that we provide for multiple caches, and getConnection(String cacheName)
 * allows for the caller to specify which one to get a connection from.
 * however, for the sake of simplicity, the no-arg getConnection() returns a
 * connection from a cache named "default". if no such cache exists,
 * getConnection() will throw a runtime exception.
 *
 * @author Swathi Kanury
 */
public class ConnectionManager extends Object {
    public final static String DEFAULT_PROPERTIES = "ConnectionManager.properties";
    private static Hashtable _caches = null;
    private final static String _defaultDSName = "defaultConn";
    private static Logger LOGGER = Logger.instance(ConnectionManager.class.getName());
    static {
        init();
        LOGGER.trace("Search Engine DataSource static initializer.");
    }
    /**
     * Constructor
     */
    private ConnectionManager() { // no constructing going on here.
    }
}
/**
 * get a connection from a cache named "default."
 *
 * @throws java.sql.SQLException if no such cache exists.
 */
public static Connection getConnection() throws java.sql.SQLException {
    return getConnection(_defaultDSName);
}
public static Connection getConnection(String poolName) throws java.sql.SQLException {
    LOGGER.info("-------------from the oracle connection----------------");
    OracleConnectionCacheImpl cache = (OracleConnectionCacheImpl) _caches.get(poolName);
    Connection c = cache.getConnection();
    while(!isValid(c))
        c = cache.getConnection();
    return c;
}
/**
 * Adds a cache to the cache hashtable.
 * at the minimum, the properties argument must contain the following keys:
 * effidatasource-name, url, user, password.
 * however, an attempt is made to set whatever is found.
 * interpretation of conflicting properties (e.g., url AND drivertype) is 
 * delegated to the underlying OraclePooledConnectionDataSource.
 */
private static void initCache(Properties props) {
    Enumeration enum = props.keys();
    while(enum.hasMoreElements()) {
        String key = (String) enum.nextElement();
        if(key.startsWith("url")) {
            StringTokenizer st = new StringTokenizer(key,".");
            Vector v = new Vector();
            while(st.hasMoreTokens()) {
                v.add(st.nextToken());
            }
            String url = (String) props.get(key);
            LOGGER.info("###url = "+url);
            String dsname = _defaultDSName;
            if(v.size() > 1)
                dsname = (String) v.get(1);
            addMiscParams(url,dsname,props);
        }
    }
}
private static void addMiscParams(String url, String dsname, Properties props) {
    try {
        OracleConnectionPoolDataSource oraPool = new OracleConnectionPoolDataSource();
        String maxConn = (String)props.get("lcc.database.JDBCPool.maxsize");
        String minConn = (String)props.get("lcc.database.JDBCPool.minsize");
        oraPool.setURL(url);
        if (isNullOrEmpty(maxConn)) maxConn = "30";
        if (isNullOrEmpty(minConn)) minConn = "20";
        OracleConnectionCacheImpl cache = new OracleConnectionCacheImpl(oraPool);
        cache.setMaxLimit(Integer.parseInt(maxConn));
        cache.setMinLimit(Integer.parseInt(minConn));
        cache.setCacheScheme(OracleConnectionCacheImplANTED_WAIT_SCHEME);
        _caches.put(dsname, cache);
    } catch (Exception e) {
        LogException.handleError("Exception while adding Misc params", e);
    }
}

private static boolean isNullOrEmpty(String s) {
    return (s == null || s.equals(""));
}

protected static void init() {
    _caches = new Hashtable();
    try {
        InputStream in = ConnectionManager.class.getResourceAsStream( DEFAULT_PROPERTIES
                )
        if (in == null)
            throw new FileNotFoundException( "y no properties file?
            dc/database/ConnectionManager.properties" );
        Properties prop = new Properties();
        prop.load(in);
        initCache(prop);
    } catch (Exception ex) {
        LogException.handleError("Exception while initializing OracleConnectionManager", ex);
    }
}

public static void restart() {
    init();
}

public static final void cleanup(Connection conn, Statement stmt, ResultSet rset) {
    LOGGER.trace("cleaning up ..");
    try {
        if (rset != null) rset.close();
        if (stmt != null) stmt.close();
        if (conn != null) conn.close();
    }
}

88
} catch (Exception ex) {
    LogException.handleError("Error while cleaning up the OracleConnectionManager", ex);
} finally {
    rset = null;
    stmt = null;
    conn = null;
}

private static boolean isValid(Connection conn) {
    LOGGER.trace("Before validating the conn from isValid: "+conn);
    boolean valid = true;
    Statement stmt = null;
    ResultSet rs = null;
    long t1 = System.currentTimeMillis();
    try {
        stmt = conn.createStatement();
        rs = stmt.executeQuery("Select sysdate from dual");
        if (rs.next())
            LOGGER.info("Connection is valid this time result = "+rs.getString(1));
    } catch (SQLException sqle) {
        conn = null;
        //LOGGER.error("Closing invalid connection.", sqle);
        return false;
    } finally {
        try {
            if (stmt != null)
                stmt.close();
            if (rs != null)
                rs.close();
        } catch (Exception sqle) {
            return false;
        }
        long t2 = System.currentTimeMillis();
        LOGGER.info("took "+(t2-t1)+" milliseconds to complete the request");
    }
    LOGGER.info("Validating connection from new isValid method result = "+valid);
    return valid;
}

6. com.scsu.searchengine.dobjects.UrlDB.java
package com.scsu.searchengine.dobjects;
import java.util.Vector;
import java.util.Hashtable;
import java.sql.Connection;
import java.sql.CallableStatement;
import oracle.jdbc.driver.OracleTypes;
import com.cscu.searchengine.database.ConnectionManager;
import com.cscu.searchengine.processors.ImageProcessor;
import com.cscu.searchengine.processors.URLProcessor;
import com.cscu.searchengine.util.DocTypes;
import com.cscu.searchengine.util.LogException;
import com.cscu.searchengine.logging.Logger;

public class UrlDB {
    private static Logger LOGGER = Logger.instance(UrlDB.class.getName());
    private static UrlDB _instance = null;
    private Vector urls;
    private final static String INSERT_URL = "\{call search_pkg.insert_url(?),?}\";
    private final static String INSERT_BASE_URL = "\{call search_pkg.insert_url(?),?}\";
    private final static String INSERT_URL_REF = "\{call search_pkg.insert_url_ref(?)}\";
    private final static String GET_REFERENCE_ID = "\{call search_pkg.get_reference_id(?)}\";
    private final static String CRAWL_CHECK = "\{call search_pkg.already_visited(?)}\";

    public static UrlDB getInstance() {
        if(_instance == null)
            _instance = new UrlDB();
        return _instance;
    }

    public void processesUrls(Vector plinks) {
        urls = new Vector();
        urls = plinks;
        for(int i=0;i<plinks.size();i++) {
            URLProcessor up = (URLProcessor) plinks.get(i);
            if(up.getDocTypeID() == DocTypes.MEDIA) {
            } else {
                if(up.getIsBase())
                    insertURL(up.getUrlDesc(), up.getDocTypeID(), up.getIsBase(), up.getReferenceURL());
                else
                    insertURL(up.getUrlDesc(), up.getDocTypeID(), up.getIsBase(),"\");
            }
        }
    }

    private long insertURL(String desc, int doc_type, boolean isBase, String reference_url) {
        Connection conn = null;
        CallableStatement cstmt = null;
        long url_id = 0;
        try {
            conn = ConnectionManager.getConnection();
        }
if(isBase)
    stmt = conn.prepareCall(INsert_BASE_URL);
else
    stmt = conn.prepareCall(INsert_URL);
stmt.setString(1, desc);
stmt.setString(2, doc_type);
stmt.registerOutParameter(3, OracleTypes.NUMBER);
if(isBase)
    stmt.setInt(4, 1); //this updates the crawled flag
stmt.executeUpdate();
url_id = stmt.getLong(3);
if(isBase)
    insertUrlReference(url_id, getReferenceID(reference_url));
} catch(Exception e) {
    LOGException.handleError("Exception desc="+desc+"&doc_type="+doc_type,e);
} finally {
    ConnectionManager.cleanup(conn,stmt,null);
}
return url_id;

private void insertUrlReference(long uid, long ref_id) {
    //LOGGER.trace("inserting image");
    Connection conn = null;
    CallableStatement stmt = null;
    try {
        conn = ConnectionManager.getConnection();
        stmt = conn.prepareCall(INsert_URL_REF);
        stmt.setLong(1, uid);
        stmt.setLong(2, ref_id);
        stmt.executeUpdate();
    } catch(Exception e) {
        LOGException.handleError("Exception url_id="+uid+"&ref_id="+ref_id,e);
    } finally {
        ConnectionManager.cleanup(conn,stmt,null);
    }
}

private long getReferenceID(String reference_url) {
    //LOGGER.trace("inserting image");
    Connection conn = null;
    CallableStatement stmt = null;
    try {
        conn = ConnectionManager.getConnection();
        stmt =
cstmt = conn.prepareCall(GET_REFERENCE_ID);
cstmt.setString(1, reference_url);
cstmt.registerOutParameter(2, OracleTypes.NUMBER);
cstmt.execute();
long url_id = cstmt.getLong(2);
if(url_id <= 0)
    url_id = CheckAndInsertToURL(reference_url);
return url_id;
} catch(Exception e) {
    LogException.handleError("Exception while getting the reference id for url = "+reference_url,e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}
return -1;
}

protected long insertImgUrl(ImageProcessor ip) {
    //LOGGER.trace("Inserting image here now");
    long uid = insertURL(ip.getUrlDesc(),DocTypes.IMAGE,false,ip.getReferenceUrl());
    return uid;
}

protected long insertMediaUrl(URLProcessor mp) {
    //LOGGER.trace("Inserting media here now");
    long uid = insertURL(mp.getUrlDesc(),DocTypes.MEDIA,mp.getIsBase(),mp.getReferenceURL());
    return uid;
}

private long CheckAndInsertToURL(String reference_url) {
    URLProcessor _up = getURLProcessor(reference_url);
    Connection conn = null;
    CallableStatement cstmt = null;
    long url_id = 0;
    if(_up != null) {
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(INSERT_URL);
            cstmt.setString(1, _up.getUrlDesc());
            cstmt.setInt(2, _up.getDocTypeID());
            cstmt.registerOutParameter(3, OracleTypes.NUMBER);
            cstmt.execute();
            url_id = cstmt.getLong(3);
        } catch(Exception e) {

    }
LogException.handleError("Exception
desc="+_up.getUrlDesc()+"&doc_type="+_up.getTypeID(),e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}

return url_id;

private URLProcessor getURLProcessor(String reference_url) {
    for(int i=0;i<urls.size();i++) {
        URLProcessor up = (URLProcessor) urls.get(i);
        if(up.getUrlDesc().equals(reference_url))
            return up;
    }
    return null;
}

public boolean isPageCrawled(String page) {
    Connection conn = null;
    CallableStatement cstmt = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(CRAWL_CHECK);
        cstmt.setString(1, page);
        cstmt.registerOutParameter(2, OracleTypes.NUMBER);
        cstmt.execute();
        if(cstmt.getInt(2) == 1)
            return true;
    } catch(Exception e) {
        LogException.handleError("Exception while checking if the page is crawled. page = "+page,e);
    } finally {
        ConnectionManager.cleanup(conn,cstmt,null);
    }
    return false;
}

7. com.scsu.searchengine.dbobjects.WordDB.java
package com.scsu.searchengine.dbobjects;
import java.util.Vector;
import java.util.Hashtable;
import java.sql.Connection;
import java.sql.CallableStatement;
import oracle.jdbc.driver.OracleTypes;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.processors.WordProcessor;
import com.scsu.searchengine.processors.URLProcessor;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.logging.Logger;

public class WordDB {
    private static Logger LOGGER = Logger.instance(WordDB.class.getName());
    private static WordDB _instance = null;
    private final static String INSERT_WORD = "call search_pkg.insert_word(?,?)";
    private final static String INSERT_URL_WORD_REF = "call search_pkg.insert_url_word_ref(?,?,?,?,?,?)";
    private final static String GET_URL_ID = "call search_pkg.get_url_id(?,?)";
    public static WordDB getInstance() {
        if(_instance == null)
            _instance = new WordDB();
        return _instance;
    }
    public void processesWords(Vector words) {
        for(int i=0;i<words.size();i++) {
            WordProcessor wp = (WordProcessor) words.get(i);
            if(wp.getWordDesc() != null && wp.getWordDesc().length() > 0)
                insertWord(wp);
        }
    }
    private void insertWord(WordProcessor wp) {
        Connection conn = null;
        CallableStatement cstmt = null;
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(INSERT_WORD);
            cstmt.setString(1, wp.getWordDesc());
            cstmt.registerOutParameter(2, OracleTypes.NUMBER );
            cstmt.execute();
            long word_id = cstmt.getLong(2);
            insertWordReference(wp, word_id);
        } catch(Exception e) {
            LogException.handleError("word desc = "+wp.getWordDesc()+
            "&url_desc="+wp.getUrlDesc(), e);
        } finally {
            ConnectionManager.cleanup(conn,cstmt,null);
        }
    }
    private void insertWordReference(WordProcessor wp, long word_id) {
        Connection conn = null;
        CallableStatement cstmt = null;
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(INSERT_URL_WORD_REF);
            cstmt.setString(1, wp.getWordDesc());
            cstmt.registerOutParameter(2, OracleTypes.NUMBER );
            cstmt.registerOutParameter(3, OracleTypes.NUMBER );
            cstmt.execute();
            long word_id2 = cstmt.getLong(2);
            long url_id2 = cstmt.getLong(3);
            insertWordReference(wp, word_id, word_id2, url_id2);
        } catch(Exception e) {
            LogException.handleError("word desc = "+wp.getWordDesc()+
            "&url_desc="+wp.getUrlDesc(), e);
        } finally {
            ConnectionManager.cleanup(conn,cstmt,null);
        }
    }
}

94

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
try {
    conn = ConnectionManager.getConnection();
    cstmt = conn.prepareCall("INSERT_URL_WORD_REF");
    cstmt.setLong(1, word_id);
    cstmt.setLong(2, getUridFromDB(wp.getUrlDesc()));
    cstmt.setInt(3, wp.getRelPosition());
    cstmt.setInt(4, wp.getCapital());
    cstmt.setInt(5, wp.getAnchor());
    cstmt.setInt(6, wp.getTitle());
    cstmt.setInt(7, wp.getStyle());
    cstmt.execute();
    /*LOGGER.trace("Inserting...Word_id=","+word_id+
       ",wp.getUrlDesc()+
       ",wp.getRelPosition()","+wp.getCapital()+
       ",wp.getAnchor()+","+wp.getTitle()+"+wp.getStyle();
     */ } catch(Exception e) {
    LogException.handleError("Exception in Word_id=","+word_id+
       ",wp.getUrlDesc()+
       ",wp.getRelPosition()","+wp.getCapital()+
       ",wp.getAnchor()+","+wp.getTitle()+"+wp.getStyle(),e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}

private long getUridFromDB(String url_desc) {
    Connection conn = null;
    CallableStatement cstmt = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall("GET_URL_ID");
        cstmt.setString(2, url_desc);
        cstmt.registerOutParameter(1, OracleTypes.NUMBER);
        cstmt.execute();
        long id = cstmt.getLong(1);
        return id;
    } catch(Exception e) {
        LogException.handleError("Error getting the url_id for url_desc="+url_desc,e);
    } finally {
        ConnectionManager.cleanup(conn,cstmt,null);
    }
    return -1;
}
package com.scsu.searchengine.dbobjects;
import java.util.Vector;
import java.util.Hashtable;
import java.sql.Connection;
import java.sql.CallableStatement;
import oracle.jdbc.driver.OracleTypes;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.processors.ImageProcessor;
import com.scsu.searchengine.util.LogException;
public class ImageDB {
    private static ImageDB _instance = null;
    private final static String INSERT_IMG = "{call search_pkg.insert_image(?,?,?,?)}";
    public static ImageDB getInstance() {
        if (_instance == null)
            _instance = new ImageDB();
        return _instance;
    }
    public void processesImages(Vector imgs) {
        for (int i = 0; i < imgs.size(); i++) {
            ImageProcessor ip = (ImageProcessor) imgs.get(i);
            insertImage(ip);
        }
    }
    private void insertImage(ImageProcessor ip) {
        Connection conn = null;
        CallableStatement cstmt = null;
        long url_id = UrlDB.getInstance().insertImgUrl(ip);
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(INSERT_IMG);
            cstmt.setLong(1, url_id);
            cstmt.setString(2, ip.getLabel());
            cstmt.setInt(3, ip.getSize());
            cstmt.setString(4, ip.getDimensions());
            cstmt.execute();
        } catch (Exception e) {
            LogException.handleError("Exception url_id="+url_id+"&label="+ip.getLabel()+"&size="+ip.getSize()+"&dim="+ip.getDimensions()+”,e);
        }
    }
}

96
package com.scsu.searchengine.dbobjects;
import java.util.Vector;
import java.util.Hashtable;
import java.sql.Connection;
import java.sql.CallableStatement;
import oracle.jdbc.driver.OracleTypes;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.processors.URLProcessor;
import com.scsu.searchengine.util.LogException;

public class MediaDB {
    private static MediaDB_instance = null;
    private final static String_INSERTMEDIA = "{call search_pkg.insert_media(?,?,?,?)}";
    public static MediaDB getInstance() {
        if (_instance == null)
            _instance = new MediaDB();
        return _instance;
    }
    public void processesMedia(Vector media) {
        for (int i = 0; i < media.size(); i++) {
            URLProcessor mp = (URLProcessor) media.get(i);
            insertMedia(mp);
        }
    }
    private void insertMedia(URLProcessor mp) {
        Connection conn = null;
        CallableStatement cstmt = null;
        long url_id = UrlDB.getInstance().insertMediaUrl(mp);
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(INSTANCE_MEDIA);
            cstmt.setLong(1, url_id);
            cstmt.setString(2, """);
            cstmt.setInt(3, 0);
            cstmt.setString(4, "MP3");
            cstmt.execute();
        } catch (Exception e) {
            LogException.handleError("Exception from insertMedia", e);
        } finally {
            ConnectionManager.cleanup(conn, cstmt, null);
        }
    }
}
10. com.scsu.searchengine.logging.LoggerInit.java
package com.scsu.searchengine.logging;
import java.util.*;
import java.io.*;
public class LoggerInit {
    public static void initLogger() {
        Properties defaultProperties = new Properties();
        java.io.InputStream in = Logger.class.getResourceAsStream("Logger.props");
        try {
            defaultProperties.load(in);
            in.close();
        } catch (Exception e) {
            System.out.println("logger initialization problem, can not find log4j property file : " + e);
        }
        String fileLocation = defaultProperties.getProperty("log4j-init-file");
        boolean loggerEnabled =
            Boolean.valueOf(defaultProperties.getProperty("enabled")).booleanValue();
        //System.out.println("Initializing Log4j");
        Logger.initLogger(fileLocation, loggerEnabled, true);
        //testing
        /*Logger LOGGER = Logger.instance(LoggerInit.class.getName());
        LOGGER.debug("This is a debug test");
        LOGGER.info("this is an info test");
        LOGGER.trace("This is a trace test");*/
    }
}

11. com.scsu.searchengine.processors.URLProcessor.java
package com.scsu.searchengine.processors;
/*
URLProcessor is a java bean implementation for URL object
*/
public class URLProcessor {
    private int url_id;
    private String url_desc;
    private int url_ref_id;
    private int doc_type_id;
    private boolean isBase;
    private String reference_url;
    public URLProcessor(int id, String desc, int ref_id, int doc_id, boolean isBase, String ref_url_desc) {
        this.setId(id);
        this.setDesc(desc);
        this.setRefId(ref_id);
    }
}
this.setDocTypeID(doc_id);
this.setIsBase(isBase);
this.setReferenceURL(ref_url_desc);
}

public URLProcessor(String desc, int doc_id, boolean isBase, String ref_url_desc) {
    this.setUrlDesc(desc);
    this.setDocTypeID(doc_id);
    this.setIsBase(isBase);
    this.setReferenceURL(ref_url_desc);
}

private void setUrlID(int id) {
    this.url_id = id;
}

private void setReferenceURL(String desc) {
    this.reference_url = desc;
}

private void setUrlDesc(String desc) {
    this.url_desc = desc;
}

private void setUrlRefID(int id) {
    this.url_ref_id = id;
}

private void setDocTypeID(int id) {
    this.doc_type_id = id;
}

private void setIsBase(boolean base) {
    this.isBase = base;
}

public int getUrlID() {
    return url_id;
}

public String getUrlDesc() {
    return url_desc;
}

public String getReferenceURL() {
    return reference_url;
}

public int getUrlRefID() {
    return url_ref_id;
}

public int getDocTypeID() {
    return doc_type_id;
}

public boolean getIsBase() {
    return isBase;
}
package com.scsu.searchengine.processors;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.HTMLParserCallback;
import java.net.URL;
import java.net.URLEncoder;
import java.net.InetAddress;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.Hashtable;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.InputStream;
import com.scsu.searchengine.util.DocTypes;
import com.scsu.searchengine.util.States;
import com.scsu.searchengine.util.Properties;
import com.scsu.searchengine.util.StopWords;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.util.Extensions;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.util.HTTP;
import java.util.Date;
public class PageProcessor extends ParserCallback {
    protected URL urlBase;
    int rel_posn = 0;
    int title = 0;
    int hdlBold = 0;
    int script = 0;
    int select = 0;
    String urlHdl = "zz";
    String mailHdl = "mm";
    Vector unvisitedLinks;
    Vector words;
    Vector images;
    String hStyle = null;
    static Logger LOGGER = Logger.instance(PageProcessor.class.getName());
    static FileWriter pdewriter = null;
    public PageProcessor(URL base) {
        urlBase = base;
rel_posn = 0;
title = 0;
hdlBold = 0;
unvisitedLinks = new Vector();
words = new Vector();
images = new Vector();
unvisitedLinks.add(Utility.checkinURL(base.toString(),true, "", Utility.getDocType(base.toString())));
}
public void handleEndTag(HTML.Tag t, int pos) {
  if (t == HTML.Tag.SCRIPT)
    script = 0;
  else if (t == HTML.Tag.STYLE)
    script = 0;
  else if (t == HTML.Tag.SELECT)
    select = 0;
}
public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
  hStyle = null;
  if (t == HTML.Tag.A) {
    String href = (String) a.getAttribute(HTML.Attribute.HREF);
    if (href != null) {
      if (href.indexOf("mailto:")) != -1) {
        mailHdl = href;
        return;
      } else if (href.indexOf("#") != -1) { return;
      } else if (href.indexOf("javascript:")) != -1) {
        if (Utility.getUrlFromJavaScript(urlBase, href) != null) {
          urlHdl = Utility.getUrlFromJavaScript(urlBase, href);
          return;
        } else if (href.indexOf("MM_openBrWindow") != -1) return;
      } else if (Utility.homeRelatedPage(Utility.getAbsoluteURL(urlBase, href)))
        return;
      urlHdl = Utility.getAbsoluteURL(urlBase, href);
    }
  } else if (t == HTML.Tag.META) {
    String name = (String) a.getAttribute(HTML.Attribute.NAME);
    // System.out.println("META....................." + name);
    if (name != null && (name.equalsIgnoreCase("keyword") ||
    name.equalsIgnoreCase("keywords"))) {

101
String keywords = (String)a.getAttribute(HTML.Attribute.CONTENT);
handleMetaWords(keywords);
}
} else if( t == HTML.Tag.FRAME) {
String href = a.getAttribute(HTML.Attribute.SRC).toString();
if(href != null) {
    if(href.indexOf("mailto:" ) != -1) {
        mailHdl = href;
        return;
    } else if(href.indexOf("#") != -1){
        return;
    } else if(href.indexOf("javascript:" ) != -1) {
        if(Utility.getUrlFromJavaScript(urlBase, href) != null) {
            urlHdl = Utility.getUrlFromJavaScrip(urlBase, href);
            return;
        }
    } else if(href.indexOf("MM_openBrWindow") != -1)
            return;
    else if(href.indexOf("theme=print") != -1)
            return;
    else if(href.indexOf("download") != -1)
            return;
    else if(Utility.homeRelatedPage(Utility.getAbsoluteURL(urlBase,href)))
            return;
    urlHdl = Utility.getAbsoluteURL(urlBase, href);
}
} else if( t == HTML.Tag.TITLE) {
title = 1;
} else if (t == HTML.Tag.FONT) {
    String str = a.getAttribute(HTML.Attribute.SIZE) == null ? "0": a.getAttribute(HTML.Attribute.SIZE).toString();
} else if (t == HTML.Tag.B) {
    bBold = 1;
} else if (t == HTML.Tag.IMG) {
    String src = (String)a.getAttribute(HTML.Attribute.SRC);
    String alt = (String)a.getAttribute(HTML.Attribute.ALT);
    String width = (String)a.getAttribute(HTML.Attribute.WIDTH);
    String height = (String)a.getAttribute(HTML.Attribute.HEIGHT);
    if(src != null)
        handleImage(src.trim(), alt, width, height);
    Enumeration enum = a.getAttributeNames();
    while(enum.hasMoreElements()) {
        Object atname = (Object) enum.nextElement();
        if(atname.toString().equalsIgnoreCase("onclick")) {
            String val = (String) a.getAttribute(atname);
        }
        break; // only handle one option
    }
}
String esword = (String) SearchProperties.getInstance().getProperty("jsEscapeWords");
if (val.startsWith(esword)) {
    String temp = val.substring(val.indexOf(esword) + esword.length());
    temp = temp.substring(0, temp.indexOf("\""));
    urlHdl = Utility.getAbsolutePath(urlBase, temp);
} else {
    String temp = val.substring(1, val.indexOf("\""));
    urlHdl = Utility.getAbsolutePath(urlBase, temp);
}

} else if (t == HTML.Tag.AREA) {
    String href = (String) a.getAttribute(HTML.Attribute.HREF);
    if (href != null) {
        if (href.indexOf("mailto:" ) != -1) {
            mailHdl = href;
            return;
        } else if (href.indexOf("#") != -1) {
            return;
        } else if (href.indexOf("javascript:" ) != -1) {
            if (Utility.getUrlFromJavaScript(urlBase, href) != null) {
                urlHdl = Utility.getUrlFromJavaScript(urlBase, href);
                return;
            }
        } else if (href.indexOf("MM_openBrWindow") != -1)
            return;
        else if (href.indexOf("theme=print") != -1)
            return;
        else if (href.indexOf("download") != -1)
            return;
        else if (!Utility.homeRelatedPage(Utility.getAbsolutePath(urlBase, href)))
            return;
    }
    String url = handleLink(urlBase, href);
    String alt = (String) a.getAttribute(HTML.Attribute.ALT);
    String title = (String) a.getAttribute(HTML.Attribute.TITLE);
    handleArea(url, title, alt);
}
else if (t.toString().equalsIgnoreCase("embed")) {
    Enumeration enum = a.getAttributeNames();
    while (enum.hasMoreElements()) {
        Object atname = (Object) enum.nextElement();
        if (atname.toString().equalsIgnoreCase("src")) {
            String val = (String) a.getAttribute(atname);

403
urlHdl = Utility.getAbsoluteURL(urlBase, val);
unvisitedLinks.add(Utility.checkinURL(urlHdl, false, urlBase, toString(), Utility.getDocType(urlHdl)));
}
}
urlHdl = "zz";
} else if(t == HTML.Tag.H1) {
hStyle = Styles.H1;
} else if(t == HTML.Tag.H2) {
hStyle = Styles.H2;
} else if(t == HTML.Tag.H3) {
hStyle = Styles.H3;
} else if(t == HTML.Tag.H4) {
hStyle = Styles.H4;
} else if(t == HTML.Tag.H5) {
hStyle = Styles.H5;
} else if(t == HTML.Tag.H6) {
hStyle = Styles.H6;
} else if(t == HTML.Tag.SCRIPT) {
script = 1;
} else if(t == HTML.Tag.STYLE) {
script = 1;
} else if(t == HTML.Tag.SELECT) {
select = 1;
} else if(t == HTML.Tag.OPTION) {
String href= (String) a.getAttribute(HTML.Attribute.VALUE);
String aburl = Utility.getAbsoluteURL(urlBase, href);
if(aburl != null & new HTTP().isValidURL(aburl)) {
  if(aburl.indexOf("mailto:" ) != -1 ) {
    mailHdl = aburl;
    return;
  } else if(aburl.indexOf("#") != -1) {
    return;
  } else if(aburl.indexOf("javascript:" ) != -1) {
    if(Utility.getUrlFromJavaScript(urlBase, href) != null) {
      urlHdl = Utility.getUrlFromJavaScript(urlBase, href);
      return;
    }
  }
} else if(aburl.indexOf("MM_openBrWindow") != -1) { 
  return;
} else if(aburl.indexOf("theme=print") != -1) { 
  return;
} else if(aburl.indexOf("download") != -1) { 
  return;
} else if(!Utility.homeRelatedPage(aburl))

104
return;
urlHdl = aburl;
}
}

public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    handleSimpleTag(t,a,pos);
}

public void handleText(char[] data, int pos) {
    String text = new String(data);
    if(text.trim().startsWith(">") || text.trim().startsWith(" "))
        text = text.substring(text.indexOf(" >")+1);
    if(title == 1) {
        title = 0;
        WordProcessor wp = new WordProcessor(urlBase.toString(),text,1,0,1,0,
SearchProperties.getInstance().getWordStyleRank(Styles.TITLE));
        words.add(wp);
    } else if(mailHdl != null && !mailHdl.equals("mm")) {
        unvisitedLinks.add(Utility.checkinURL(mailHdl, false, urlBase.toString(),
DocTypes.EMAIL));
        rel_posn++;  
        Vector v = Utility.checkinEmail(text, mailHdl, 0, rel_posn, 0);
        words.addAll(v);
        mailHdl = "mm";
    } else if (urlHdl != null && !urlHdl.equals("zz")) {
        // new link found
        if(!isDuplicate(urlHdl)) {
            unvisitedLinks.add(Utility.checkinURL(urlHdl, false,
urlBase.toString(),Utility.getDocType(urlHdl)));
            rel_posn++;  
            Vector v = Utility.checkinWord(text, urlHdl, 0, rel_posn,
SearchProperties.getInstance().getWordStyleRank(Styles.ANCHOR));
            words.addAll(v);
        }
        urlHdl = "zz";
    } else if(hdlBold == 1) {
        hdlBold = 0;
        Vector v = Utility.checkinWords(text, urlBase.toString(), 0, rel_posn, 0,
SearchProperties.getInstance().getStyleRank(Styles.BOLD),true);
        words.addAll(v);
        rel_posn += v.size();
    } else if (script == 1) {
        // script = 0;
        // do nothing
    } else {
Vector v = new Vector();
if(hStyle == null)
    v = Utility.checkInWords(text,urlBase.toString(), 0, rel_posn, 0, 0, true);
else
    v = Utility.checkInWords(text,urlBase.toString(), 0, rel_posn, 0,
SearchProperties.getInstance().getStyleRank(hStyle),true);
    words.addAll(v);
    rel_posn += v.size();
    hStyle = null;
}

private void handleImage(String src, String alt, String width, String height) {
    String abssrc = Utility.getAbsoluteURL(urlBase,src);

    if(!isDuplicate(abssrc)) {
        ImageProcessor ip = new ImageProcessor(width+"X"+height,alt,abssrc,urlBase.toString());
        images.add(ip);
    }
}

private boolean isDuplicate(String url) {
    for(int i=0;i<unvisitedLinks.size();i++) {
        URLProcessor up = (URLProcessor) unvisitedLinks.get(i);
        String t_url = up.getUrlDesc();
        if(t_url.equalsIgnoreCase(url))
            return true;
    }
    for(int i=0;i<images.size();i++) {
        ImageProcessor ip = (ImageProcessor) images.get(i);
        String t_url = ip.getUrlDesc();
        if(t_url.equalsIgnoreCase(url))
            return true;
    }
    return false;
}

private boolean stopWord(String word) {
    return StopWords.getInstance().stopWord(word);
}

public Vector getPageLinks() {
    return unvisitedLinks;
}
public Vector getWordList() {
    return words;
}

public Vector getImageList() {
    return images;
}

private void handleArea(String url, String title, String alt) {
    // LOGGER.trace("url retrieved from handleLink = "+url);
    if(url != null) {
        if(!isDuplicate(url)) {
            Utility.checkinURL(url, false, urlBase.toString(), Utility.getDocType(url));
            Vector v = new Vector();
            if(title != null) {
                v = Utility.checkinWord(title, url, 0, 0,
                        SearchProperties.getInstance().getWordStyleRank(Styles.ANCHOR));
            } else if(alt != null) {
                v = Utility.checkinWord(alt, url, 0, 0,
                        SearchProperties.getInstance().getWordStyleRank(Styles.ANCHOR));
            }
            words.addAll(v);
        }
    }
}

private void handleMetaWords(String keywords) {
    StringTokenizer st = new StringTokenizer(keywords, ",");
    while(st.hasMoreTokens()) {
        String token = st.nextToken().trim();
        if(token.length() > 0) {
            Vector v = Utility.checkinWord(token, urlBase.toString(), 0, 0,
                        SearchProperties.getInstance().getStyleRank(Styles.META));
            words.addAll(v);
        }
    }
}

public static void main(String args[]) throws Exception {
    /* System.out.println(PageProcessor.handleLink(
        new URL("http://www.southernct.edu/departments/fsrc/gradfellows03/index.htm"),"#test");
    */
    String page = "http://www.southernct.edu/departments/fsrc/gradfellows03//02fainstein.htm";
    try {
        HTTP http = new HTTP();
        107
    } finally {
    }
}
java.io.StringReader str = new java.io.StringReader(http.downloadWWWPage(page));
javax.swing.text.html.HTMLEditorKit htmlKit = new javax.swing.text.html.HTMLEditorKit();
javax.swing.text.html.HTMLDocument htmlDoc =
(javax.swing.text.html.HTMLDocument)htmlKit.createDefaultDocument();
javax.swing.text.html.HTMLEditorKit.Parser parser = new
javax.swing.text.html.parser.ParserDelegator();
URL url = new URL(page);
PageProcessor pp = new PageProcessor(url);
parser.parse(str, pp, true);
Vector list = pp.getPageLinks();
for(int i=0;i<list.size()&&++i) {
    URLProcessor up = (URLProcessor) list.get(i);
    System.out.println(up.getUrlDesc());
}
} catch(Exception e) {
    e.printStackTrace();
}
}

package com.scsu.searchengine.processors;
/*
WordProcessor is a java bean implementation of Word objects
*/
public class WordProcessor {
    String url_desc;
    int word_id;
    String word_desc;
    int title;
    int rel_posn;
    int capital;
    int anchor;
    int style;
    public WordProcessor(String url_desc,
            int word_id,
            String desc,
            int title,
            int rel_posn,
            int capital,
            int anchor,
            int style /*bold = 1, big font=2, italics = 3*/)
    {
        this.setUrlDesc(url_desc);
        this.setWordID(word_id);
        this.setWordDesc(desc);

        108

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
public void setTitle(int title) {
    this.title = title;
}
public void setRelPosition(int posn) {
    this.rel_posn = posn;
}
public void setCapital(int caps) {
    this.capital = caps;
}
public void setAnchor(int anchor) {
    this.anchor = anchor;
}
public void setStyle(int style) {
    this.style = style;
}
public String getUrlDesc() {
    return url_desc;
}
public int getWordID() {
    return word_id;
}
public String getWordDesc() {
    return word_desc;
}
public int getTitle() {
    return title;
}
public int getRelPosition() {
    return rel_posn;
}
public int getCapital() {
    return capital;
}
public int getAnchor() {
    return anchor;
}
public int getStyle() {
    return style;
}
}

package com.scsu.searchengine.processors;
import com.scsu.searchengine.util.HTTP;
import com.scsu.searchengine.util.LogException;

public class ImageProcessor {
    private int url_id;
    private int size;
    private String dim;
    private String label;
    private String url_desc;
    private int url_ref_id;
    private String reference_url;
    public ImageProcessor (int url_id,
        String dim,
        String label,
        int ref_id,
String url_desc,
String ref_url_desc) {
    this.setUrlID(url_id);
    this.setSize(url_desc);
    this.setDimensions(dim);
    this.setLabel(label);
    this.setUrlRefID(ref_id);
    this.setUrlDesc(url_desc);
    this.setReferenceUrl(ref_url_desc);
}

private void setUrlID(int id) {
    this.url_id = id;
}

private void setUrlDesc(String desc) {
    this.url_desc = desc;
}

private void setReferenceUrl(String desc) {
    this.reference_url = desc;
}

private void setUrlRefID(int id) {
    this.url_ref_id = id;
}

private void setSize(String url) {
    size = new HTTP().getWwwPageStreamSize(url);
}

private void setDimensions(String id) {
    this.dim = id;
}

private void setLabel(String label) {
    this.label = label;
}

public int getUrlID() {
    return url_id;
}

public String getUrlDesc() {
    return url_desc;
}

public String getReferenceUrl() {
    return reference_url;
}

public int getUrlRefID() {
    return url_ref_id;
}

public int getSize() {
return size;
}
public String getDimensions() {
    return dim;
}
public String getLabel() {
    return label;
}

package com.scru.searchengine.processors;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLDocumentParserCallback;
import java.net.URL;
import java.util.Vector;
import java.util.Enumeration;
import com.scru.searchengine.util.*;
public class CtrContentProcessor extends ParserCallback {
    URL baseurl;
    Vector ilinks = new Vector();
    Vector iwords = new Vector();
    Vector iimages = new Vector();
    boolean no_processing = false;
    int title = 0;
    String href = "none";
    String mail = "none";
    String hStyle = "none";
    int select = 0;
    int rel_posn = 0;

    public CtrContentProcessor(URL base) {
        this.baseurl = base;
        //add the base url to the list
        URLProcessor up = new
        URLProcessor(base.toString(), Utility.getDocType(base.toString()),true,");
        ilinks.add(up);
    }
    public void handleEndElement(HTML.Tag t, int pos) {
        if (t == HTML.Tag.SCRIPT)
            no_processing = false;
        else if (t == HTML.Tag.STYLE)
            no_processing = false;
else if(t == HTML.Tag.SELECT)  
    no_processing = false;
}

public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    if(t == HTML.Tag.STYLE) {
        no_processing = true;
    } else if (t == HTML.Tag.SELECT) {
    } else if(t == HTML.Tag.SCRIPT) {
        no_processing = true;
    } else if (t == HTML.Tag.TITLE) {
        title = 1;
    } else if (t == HTML.Tag.IMG) {
        String src = (String) a.getAttribute(HTML.Attribute.SRC);
        String alt = (String) a.getAttribute(HTML.Attribute.ALT);
        String absurl = Utility.getAbsoluteURL(baseurl, src);
        if(href.equals("none") == false) {
            /* This is to handle a situation where href
             * is made to an image rather than text
             */
            handleLink(href, ")
            href = "none";
        }
        if(absurl != null)  
            handleImage(absurl, alt);
    } else if (t == HTML.Tag.A || t == HTML.Tag.AREA) {
        String url = (String) a.getAttribute(HTML.Attribute.HREF);
        String absurl = Utility.getAbsoluteURL(baseurl, url);
        if(url != null) {
            if(url.indexOf("mailto:" ) != -1) {
                mail = url;
                return;
            } else if(url.indexOf("#") != -1) {
                return;
            } else if(url.indexOf("javascript:" ) != -1)
                if(Utility.getUrlFromJavaScript(baseurl, url) != null) {
                    href = Utility.getUrlFromJavaScript(baseurl, url);
                    return;
                }
            else if(url.indexOf("MM_openBrWindow") != -1)
                return;
            else if(url.indexOf("theme=print") != -1)
                return;
            else if(url.indexOf("download") != -1)
                return;
            else if(!Utility.homeRelatedPage(absurl))
        }
    }
}
return;
href = Utility.getAbsoluteURL(baseurl, url);
if(t == HTML.Tag.AREA) {
    URLProcessor up = new
    URLProcessor(href, Utility.getDocType(href), false, baseurl.toString());
    ilinks.add(up);
}
else if(t == HTML.Tag.OPTION) {
    String url = (String) t.getAttribute(HTML.Attribute.VALUE);
    String aburl = Utility.getAbsoluteURL(baseurl, url);
    if(aburl != null && new HTTP().isValidURL(aburl)) {
        if(aburl.indexOf("mailto:" ) != -1) {
            mail = aburl;
            return;
        } else if(aburl.indexOf("#" ) != -1){
            return;
        } else if(url.indexOf("javascript:" ) != -1)
        if(Utility.getUrlFromJavaScript(baseurl, url) != null) {
            href = Utility.getUrlFromJavaScript(baseurl, url);
            return;
        }
        else if(aburl.indexOf("MM_openBrWindow") != -1)
        return;
        else if(aburl.indexOf("theme=print") != -1)
        return;
        else if(aburl.indexOf("download") != -1)
        return;
        else if(!Utility.homeRelatedPage(aburl))
        return;
        href = aburl;
    }
} else if(t == HTML.Tag.H1) {
    hStyle = Styles.H1;
} else if(t == HTML.Tag.H2) {
    hStyle = Styles.H2;
} else if(t == HTML.Tag.H3) {
    hStyle = Styles.H3;
} else if(t == HTML.Tag.H4) {
    hStyle = Styles.H4;
} else if(t == HTML.Tag.H5) {
    hStyle = Styles.H5;
} else if(t == HTML.Tag.H6) {
    hStyle = Styles.H6;
} else if(t.toString().equalsIgnoreCase("embed")) {

114
Enumeration enum = a.getAttributeNames();
while(enum.hasMoreElements()) {
   Object atname = (Object) enum.nextElement();
   if(atname.toString().equalsIgnoreCase("src")) {
      String val = (String) a.getAttribute(atname);
      String url = Utility.getAbsoluteURL(baseurl,val);
      if(url != null) {
         URLProcessor up = new URLProcessor(url, Utility.getDocType(url),false, baseurl.toString());
         ilinks.add(up);
      }
   }
}

public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
   handleSimpleTag(t,a,pos);
}
public void handleText(char[] data,int pos) {
   if(no_processing == false) {
      if(title == 1) {
         /* This means this text belongs to title*/
         title = 0;
         //process title tag;
         Vector v = Utility.checkinWord(new String(data), baseurl.toString(), 1, 0, 1);
         iwords.addAll(v);
      } else if(href.equals("none") == false) {
         handleLink(href, new String(data));
         href = "none";
      } else if(mail.equals("none") == false) {
         //handle the email info
         URLProcessor up = new URLProcessor(mail, DocTypes.EMAIL,false, baseurl.toString());
         ilinks.add(up);
         Vector words = Utility.checkinEmail(new String(data),mail,0,0,SearchProperties.getInstance().getWordStyleRank(Styles.TITLE));
         iwords.addAll(words);
         mail = "none";
      } else if(hStyle.equals("none") == false) {
         Vector words = Utility.checkinWords(new String(data),baseurl.toString(),0,rel_posn,0,SearchProperties.getInstance().getWordStyleRank(hStyle),true);
         rel_posn += words.size();
         iwords.addAll(words);
      } else {

// So this is the regular text.
Vector words = Utility.checkinWords(new String(data), baseUrl.toString(), 0, rel_posn, 0, true);
    rel_posn = rel_posn + words.size();
iwords.addAll(words);
}
}
private void handleImage(String url, String alt_text) {
    ImageProcessor ip = new ImageProcessor(alt_text, url, baseUrl.toString());
iimages.add(ip);
}
private void handleLink(String url, String text) {
    // build object for url
    URLProcessor up = new URLProcessor(url, Utility.getDocType(url), false, baseUrl.toString());
ilinks.add(up);

    if (text != null && text.trim().length() > 0) {
        rel_posn = rel_posn + 1;
        Vector words = Utility.checkinWord(text, url, title, rel_posn, SearchProperties.getInstance().getWordStyleRank(Style.s.ANCHOR));
iwords.addAll(words);
    }
}
public Vector getPageLinks() {
    return ilinks;
}

public Vector getWordList() {
    return iwords;
}
public Vector getImageList() {
    return iimages;
}

16. com.scsu.searchengine.processors.LeftNavProcessor
package com.scsu.searchengine.processors;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.HTMLParserCallback;
import java.net.URL;
import java.util.Vector;

116
import java.util Enumeration;
import com.scsu.searchengine.util.*;
public class LeftNavProcessor extends ParserCallback {
    URL baseurl;
    Vector inlinks = new Vector();
    Vector iwords = new Vector();
    Vector iimages = new Vector();
    boolean no_processing = false;
    int title = 0;
    String href = "none";
    String mail = "none";
    /**
     * @param base
     */
    public LeftNavProcessor(URL base) {
        baseurl = base;
        //add the base url to the list
        URLProcessor up = new
        URLProcessor(base.toString(), Utility.getDocType(base.toString()), true, "")
        inlinks.add(up);
    }
    /**
     * @param t
     * @param pos
     */
    public void handleEndTag(HTML.Tag t, int pos) {
        if (t == HTML.Tag.SCRIPT)
            no_processing = false;
        else if (t == HTML.Tag.STYLE)
            no_processing = false;
        else if (t == HTML.Tag.SELECT)
            no_processing = false;
    }
    /**
     * @param t
     * @param a
     * @param pos
     */
    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        if (t == HTML.Tag.STYLE) {
            no_processing = true;
        } else if (t == HTML.Tag.SELECT) {
            no_processing = false;
        }
    }
no_processing = true;
} else if(t == HTML.Tag.SCRIPT) {
    no_processing = true;
} else if (t == HTML.Tag.TITLE) {
    title = t;
} else if (t == HTML.Tag.IMG) {
    String src = (String) a.getAttribute(HTML.Attribute.SRC);
    String alt = (String) a.getAttribute(HTML.Attribute.ALT);
    String absurl = Utility.getAbsolutePath(baseurl, src);
    if(href.equals("none") == false) {
        /* This is to handle a situation where href
         * is made to an image rather than text
         */
        handleLink(href, alt);
        href = "none";
    }
    if(absurl != null)
        handleImage(absurl, alt);
} else if (t == HTML.Tag.A || t == HTML.Tag.AREA) {
    String url = (String) a.getAttribute(HTML.Attribute.HREF);
    if(url != null) {
        if(url.indexOf("mailto:")) != -1) {
            mail = url;
            return;
        } else if(url.indexOf("#") != -1) {
            return;
        } else if(url.indexOf("javascript:")) != -1) {
            if(Utility.getUrlFromJavaScript(baseurl, url) != null) {
                href = Utility.getUrlFromJavaScript(baseurl, url);
            }
        } else if(url.indexOf("MM_openBrWindow") != -1)
            return;
    } else if(url.indexOf("theme=print") != -1)
        return;
    else if(url.indexOf("download") != -1)
        return;
    href = Utility.getAbsolutePath(baseurl, url);
    if(t == HTML.Tag.AREA) {
        URLProcessor up = new URLProcessor(href, Utility.getDocType(href), false, baseurl.toString());
        ilinks.add(up);
    }
} else if(t == HTML.Tag.OPTION) {

118

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
String url= (String) a.getAttribute(HTML.Attribute.VALUE);
String aburl = Utility.getAbsoluteURL(baseurl, url);
if(aburl != null && new HTTP().isValidURL(aburl)) {
    if(aburl.indexOf("mailto:" ) != -1) {
        mail = aburl;
        return;
    } else if(aburl.indexOf("#") != -1) {
        return;
    } else if(url.indexOf("javascript:" ) != -1) {
        if(Utility.getUrlFromJavaScript(baseurl, url) != null) {
            href = Utility.getUrlFromJavaScript(baseurl, url);
            return;
        }
    } else if(aburl.indexOf("MM_openBrWindow") != -1)
        return;
    else if(url.indexOf("theme=print") != -1)
        return;
    else if(url.indexOf("download") != -1)
        return;
    else if(!Utility.homeRelatedPage(aburl))
        return;
    href = aburl;
}
} else if(t.toString().equalsIgnoreCase("embed")) {
    Enumeration enum = a.getAttributeNames();
    while(enum.hasMoreElements()) {
        Object atname = (Object) enum.nextElement();
        if(atname.toString().equalsIgnoreCase("src")) {
            String val = (String) a.getAttribute(atname);
            String url = Utility.getAbsoluteURL(baseurl,val);
            if(url != null) {
                URLProcessor up = new URLProcessor(url,Utity.getDocType(url),false,baseurl.toString());
                ilinks.add(up);
            }
        }
    }
}
/**
 * @param t
 * @param a
 * @param pos
*/
public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    handleSimpleTag(t,a,pos);
}
/**
 * @param data
 * @param pos
 */
public void handleText(char[] data,int pos) {
    if(no_processing == false) {
        if(title == 1) {
            /* This means this text belongs to title*/
            title = 0;
            //process title tag;
            Vector v = Utility.checkinWord(new String(data),baseUrl.toString(),1,0,SearchProperties.getInstance().getWordStyleRank(Styles.TITLE));
            iwords.addAll(v);
        } else if(href.equals("none") == false) {
            handleLink(href, new String(data));
            href = "none";
        } else if(mail.equals("none") == false) {
            /*handle the email info*/
            URLProcessor up = new URLProcessor(mail, DocTypes.EMAIL,false,baseUrl.toString());
            ilinks.add(up);
            Vector words = Utility.checkinEmail(new String(data),mail,0,0,SearchProperties.getInstance().getWordStyleRank(Styles.TITLE));
            iwords.addAll(words);
            mail = "none";
        }
    }
}
/**
 * @return
 */
public Vector getPageLinks() {
    return ilinks;
}
/**
 * @return
 */
public Vector getWordList() {
    return iwords;
}
public Vector getImageList() {
    return images;
}

private void handleImage(String url, String alt_text) {
    ImageProcessor ip = new ImageProcessor(alt_text, url, baseurl.toString());
    images.add(ip);
}

private void handleLink(String url, String text) {
    //build object for url
    URLProcessor up = new URLProcessor(url, Utility.getDocType(url), false, baseurl.toString());
    iLinks.add(up);
    if (text != null && text.trim().length() > 0) {
        Vector words = Utility.checkInWord(text, url, title, 0, SearchProperties.getInstance().getWordStyleRank(Styles.ANC_HOR));
        iWords.addAll(words);
    }
}

public static void main(String[] args) {
    try {
        URL u1 = new URL("http://www.southernct.edu/departments/womenscenter//resources/events.htm");
        URL u2 = new URL(u1, "/.../safe/assault.htm");
        System.out.println(u2.toString());
    } catch (Exception e) { e.printStackTrace(); } */
    try {
        URL url = new URL("http://www.southernct.edu/faculty/paffairs/news");
        String str = new HTTP().downloadWWWPage(url.toString());
        java.io.StringReader br = new java.io.StringReader(str);
        HTMLEditorKit htmlKit = new HTMLEditorKit();
        javax.swing.text.html.HTMLDocument htmlDoc =
            javax.swing.text.html.HTMLDocument.htmlKit.createDefaultDocument();
        HTMLEditorKit.Parser parser = new javax.swing.text.html.parser.ParserDelegator();
        java.util.Hashtable hash = new java.util.Hashtable();
        /*
        * Split html into 2 pieces
        * First one is the right nav
        * Second one is the center portion
        * If the page cannot be split into 2 then pass it as one html to
        * PageProcessor for regular parsing
        */
    }
}
Vector v = new Vector();
putil.split(v, "\<!--right col-->\".str, org.apache.orotext.perl.Perl5Util.SPLIT_ALL);
System.out.println(v.size());
if(v.size() > 1) {
/* which means the split is successful */
java.io.StringReader leftnav = new java.io.StringReader((String) v.get(0));
/* System.out.println(" 
System.out.println((String) v.get(0));
System.out.println(" 
LeftNavProcessor lnp = new LeftNavProcessor(url);
try {
p = parser.parse(leftnav, lnp, true);
} catch(Exception ioe) {

ioe.printStackTrace();
}
Vector _links = lnp.getPageLinks();
Vector _words = lnp.getWordList();
Vector _images = lnp.getImageList();
String part2 = (String) v.get(1);
Vector v2 = new Vector();
putil.split(v2, "\<!--FOOTER TABLE -->\".str, org.apache.orotext.perl.Perl5Util.SPLIT_ALL);
java.io.StringReader ctr_content = new java.io.StringReader((String) v2.get(0));
CtrlContentProcessor ccp = new CtrlContentProcessor(url);
try {
p = parser.parse(ctr_content, ccp, true);
} catch(Exception ioe) {

ioe.printStackTrace();
}
_i_links.addAll(ccp.getPageLinks());
_i_words.addAll(ccp.getWordList());
_i_images.addAll(ccp.getImageList());
hash.put("links", _i_links);
hash.put("words", _i_words);
hash.put("images", _i_images);
} else {
/* which means the page is from a department rather then from SCSU directly */
PageProcessor pp = new PageProcessor(url);
Vector v2 = new Vector();
putil.split(v2, "\<!--FOOTER TABLE -->\".str, org.apache.orotext.perl.Perl5Util.SPLIT_ALL);
br = new java.io.StringReader((String)v2.get(0));
try {
p = parser.parse(br, pp, true);
*/
} catch(Exception ioe) {
    ioe.printStackTrace();
}

hash.put("links", pp.getPageLinks());
hash.put("words", pp.getWordList());
hash.put("images", pp.getImageList());

//print
Vector v1 = (Vector) hash.get("links");
System.out.println("-----------printing urls----------");
for(int i=0;i<v1.size(); i++) {
    URLProcessor up = (URLProcessor) v1.get(i);
    System.out.println(up.getUrlDesc());
}

Vector v2 = (Vector) hash.get("words");
System.out.println("-----------printing words----------");
for(int i=0;i<v2.size(); i++) {
    WordProcessor wp = (WordProcessor) v2.get(i);
    System.out.println(wp.getUrlDesc()+", word_desc = "+wp.getWordDesc() + ", @pos = "+wp.getRelPosition());
}

Vector v3 = (Vector) hash.get("images");
System.out.println("-----------printing images----------");
for(int i=0;i<v3.size(); i++) {
    ImageProcessor ip = (ImageProcessor) v3.get(i);
    System.out.println(ip.getUrlDesc()+", label="+ip.getLabel());
}
}

} catch (Exception e) {e.printStackTrace();}
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.logging.Logger;

public class PDFContentProcessor {
    private final static String GET_PDF_DOCS = "\{call search_pkg.getAllPDFDocs(?)}";
    private final static String INSERT_WORD = "\{call search_pkg.insert_word(?,?)\}";
    private final static String INSERT_URL_WORD_REF = "\{call search_pkg.insert_url_word_ref(?,?,?,?,?)\}";
    private final static String GET_URL_ID = "\{call search_pkg.get_url_id(?,?)\}";
    private static final String UPDATE_SUMMARIES = "\{call search_pkg.update_summary(?,?,?)\}";

    public static void main(String[] args) {
        Connection conn = null;
        CallableStatement cstmt = null;
        Vector v = new Vector();
        ResultSet rset = null;
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(GET_PDF_DOCS);
            cstmt.registerOutParameter(1, OracleTypes.CURSOR);
            cstmt.executeQuery();
            rset = (ResultSet) cstmt.getObject(1);
            while(rset.next()) {
                v.add(rset.getString("url_desc"));
            }
        } catch(Exception e) {
            LogException.handleError("", e);
        } finally {
            ConnectionManager.cleanup(conn, cstmt, null);
        }
        for(int i=0;i<v.size();i++) {
            String url = (String)v.get(i);
            PDFContentProcessor.process(url);
        }
    }

    public static void process(String url) {
        InputStream is = null;
        try {
            WordExtractor extractor = new WordExtractor();
            //String s = extractor.extractText(new FileInputStream(args[0]));
            //System.out.println(s);
            HttpClient client = new HttpClient();
            client.setTimeout(10000);
            GetMethod get = null;

            124
get = new GetMethod(URIUtil.encodePathQuery(url));
client.executeMethod(get);
is = get.getResponseBodyAsStream();
PDFExtractor wextractor = new PDFExtractor();
String str = wextractor.extractText(is);
System.out.println("---------
http://www.southernct.edu/faculty/facdev/files/SummerTech03.pdf--------");
System.out.println(str);
String summary = str.substring(0,1000);
System.out.println("\n\n\nsummary: "+summary);
StringTokenizer tokens = new StringTokenizer(str, "\n");
while(tokens.hasMoreTokens()) {
    String token = tokens.nextToken();
    System.out.println(token);
    if(token.length() > 2000)
        saveWord(url, token.substring(0, 1999), summary);
    else
        saveWord(url, token, summary);
}
try {
    is.close();
} catch (Exception e) {
    e.printStackTrace();
} finally {
    try {
        is.close();
    } catch (Exception e1) {}
}

public static void saveWord(String url, String token, String summary) {
    Connection conn = null;
    CallableStatement cstmt = null;
    long word_id = 0;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall("INSERT_WORD");
        cstmt.setString(1, token);
        cstmt.registerOutParameter(2, OracleTypes.NUMBER);
        cstmt.execute();
        word_id = cstmt.getLong(2);
    } catch (Exception e) {
        LogException.handleException("word desc = "+token+
                        
                        
                        
                        
                        
                        "&url_desc="+url, e);
    } finally {
        ConnectionManager.cleanup(conn, cstmt, null);
    }

125
if (word_id != 0) {
  // get url_id
  conn = null;
  cstnnt = null;
  long url_id = 0;
  try {
    conn = ConnectionManager.getConnection();
    cstnnt = conn.prepareCall(GET_URL_ID);
    cstnnt.setString(2, url);
    cstnnt.registerOutParameter(1, OracleTypes.NUMBER);
    cstnnt.executeUpdate();
    url_id = cstnnt.getLong(1);
  } catch (Exception e) {
    LogException.handleError("Error getting the url_id for url_desc="+url,e);
  }
  finally {
    ConnectionManager.cleanup(conn,cstnnt,null);
  }
  // insert word reference
  conn = null;
  cstnnt = null;
  try {
    conn = ConnectionManager.getConnection();
    cstnnt = conn.prepareCall(INSERT_URL_WORD_REF);
    cstnnt.setLong(1, word_id);
    cstnnt.setLong(2, url_id);
    cstnnt.setInt(3, 0);
    cstnnt.setInt(4, 0);
    cstnnt.setInt(5, 0);
    cstnnt.setInt(6, 0);
    cstnnt.setInt(7, 0);
    cstnnt.executeUpdate();
  } catch (Exception e) {
    LogException.handleError("Exception in Word_id="+word_id+
                           ", url_desc="+url,e);
  }
  finally {
    ConnectionManager.cleanup(conn,cstnnt,null);
  }

  // insert summary
  conn = null;
  cstnnt = null;
  String summ = summary;
  if (summary.length() > 4000)
    summ = summary.substring(0, 3900);
  try {

126

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
conn = ConnectionManager.getConnection();
cstmt = conn.prepareCall(UPDATE_SUMMARIES);
cstmt.setLong(1, word_id);
cstmt.setLong(2, url_id);
cstmt.setString(3, summ);
cstmt.setString(4, "");
cstmt.executeUpdate();
} catch(Exception e) {
    LogException.handleError("Error while update summaries " +
            "word_id = " + word_id +
            "url_id = " + url_id,e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}
}
}

18. com.scsu.searchengine.processors.WordContentProcessor
package com.scsu.searchengine.processors;
import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Vector;
import oracle.jdbc.driver.OracleTypes;
import java.util.StringTokenizer;
import java.io.*;
import java.net.*;
import org.textmining.text.extraction.*;
import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.methods.*;
import org.apache.commons.httpclient.util.*;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.logging.Logger;
public class WordContentProcessor {
    private final static String GET_WORD_DOCS = "{call search_pkg.getAllWordDocs(?)}";
    private final static String INSERT_WORD = "{call search_pkg.insert_word(?)}";
    private final static String INSERT_URL_WORD_REF = "{call
        search_pkg.insert_url_word_ref(?*,*)}";
    private final static String GET_URL_ID = "{call search_pkg.get_url_id(?)}";
    private static final String UPDATE_SUMMARIES = "{call
        search_pkg.update_summary(?*,*)}";
    public static void main(String[] args) {
        Connection conn = null;
CallableStatement cstmt = null;
Vector v = new Vector();
ResultSet rset = null;
try {
    conn = ConnectionManager.getConnection();
cstmt = conn.prepareCall(GET_WORD_DOCS);
cstmt.registerOutParameter(1, OracleTypes.CURSOR);
cstmt.execute();
rset = (ResultSet) cstmt.getObject(1);
while(rset.next()) {
    v.add(rset.getString("url_desc"));
}
} catch(Exception e) {
    LogException.handleError("", e);
} finally {
    ConnectionManager.cleanup(conn, cstmt, null);
}
for(int i=0;i<v.size();i++) {
    System.out.println("----url----"+v.get(i));
    String url = (String)v.get(i);
    WordContentProcessor.process(url);
}

public static void process(String url) {
    InputStream is = null;
    try {
        WordExtractor extractor = new WordExtractor();
        //String s = extractor.extractText(new FileInputStream(args[0]));
        //System.out.println(s);
        HttpClient client = new HttpClient();
        client.setHttpHM(10000);
        GetMethod get = null;
        get = new GetMethod(URIUtil.encodePathQuery(url));
        client.executeMethod(get);
        is = get.getResponseAsStream();
        WordExtractor wextractor = new WordExtractor();
        String str = wextractor.extractText(is);
        System.out.println("--------"+url+"--------");
        System.out.println(str);
        String summary = str.substring(0,1000);
        System.out.println("\n\n\nsummary: "+summary);
        StringTokenizer tokens = new StringTokenizer(str,\n"");
        while(tokens.hasMoreTokens()) {
            String token = tokens.nextToken();
            System.out.println(token);
        }
    } finally {
        if(is != null) is.close();
    }
}
saveWord(url, token, summary);
}
}
catch (Exception e) {
e.printStackTrace();
} finally {
try {
is.close();
} catch (Exception e1){}
}

public static void saveWord(String url, String token, String summary) {
    Connection conn = null;
    CallableStatement cstmt = null;
    long word_id = 0;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(INSERT_WORD);
        cstmt.setString(1, token);
        cstmt.registerOutParameter(2, OracleTypes.NUMBER);
        cstmt.execute();
        word_id = cstmt.getLong(2);
    } catch (Exception e) {
        LogException.handleError("word desc = "+token+
                
"&url_desc="+url, e);
    } finally {
        ConnectionManager.cleanup(conn, cstmt, null);
    }
    if(word_id != 0) {
        //get url_id
        conn = null;
        cstmt = null;
        long url_id = 0;
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(GET_URL_ID);
            cstmt.setString(2, url);
            cstmt.registerOutParameter(1, OracleTypes.NUMBER);
            cstmt.execute();
            url_id = cstmt.getLong(1);
        } catch (Exception e) {
            LogException.handleError("Error getting the url_id for url_desc="+url,e);
        } finally {
            ConnectionManager.cleanup(conn, cstmt, null);
        }
//insert word reference
conn = null;
cstmt = null;
try {
    conn = ConnectionManager.getConnection();
cstmt = conn.prepareCall(INSERT_URL WORD REF);
cstmt.setLong(1, word_id);
cstmt.setLong(2, url_id);
cstmt.setInt(3, 0);
cstmt.setInt(4, 0);
cstmt.setInt(5, 0);
cstmt.setInt(6, 0);
cstmt.setInt(7, 0);
cstmt.executeUpdate();
} catch (Exception e) {
    LogException.handleException("Exception in Word_id="+word_id+
"&url_desc="+url,e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}

//insert summary
conn = null;
cstmt = null;
try {
    conn = ConnectionManager.getConnection();
cstmt = conn.prepareCall(UPDATE_SUMMARIES);
cstmt.setLong(1, word_id);
cstmt.setLong(2, url_id);
cstmt.setString(3, summary);
cstmt.setString(4, "");
cstmt.executeUpdate();
} catch (Exception e) {
    LogException.handleException("Error while update summaries "+
"word_id = " + word_id +
"url_id = " + url_id,e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,null);
}

package com.scsu.searchengine.search;
import java.util.Hashtable;
import java.util.StringTokenizer;
import java.util Enumeration;
import java.util Vector;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Comparator;
import java.util.Arrays;
import java.sql.Connection;
import java.sql.ResultSet;
import oracle.jdbc.driver.OracleTypes;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.util.StopWords;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.util.SearchProperties;
import com.scsu.searchengine.database.ConnectionManager;
public class Search {
    private static Logger LOGGER = Logger.instance(Search.class.getName());
    private static Search _instance = null;
    private static final String GET_SEARCH_RESULTS = "{call search_pkg.get_search_results(? ,?,?) }",
    private static final String GET_IMG_SEARCH_RESULTS = "{call search_pkg.get_img_search_results(? ,?,?) }",
    private static final String GET_EMAIL_SEARCH_RESULTS = "{call search_pkg.get_email_results(? ,?) }",
    private static final String GET_MEDIA_SEARCH_RESULTS = "{call search_pkg.get_media_results(? ,?) }",
    public static Search getInstance() {
        if (_instance == null)
            _instance = new Search();
        return _instance;
    }
    public Vector getSearchResults(String phrase, int dtype) {
        Hashable words = parsePhrase(phrase, dtype);
        Vector results = new Vector();
        if (dtype == 1)
            results = reArrangeResults(getMatchableLinks(words, dtype, phrase));
        else if (dtype == 2)
            results = reArrangeImgResults(getMatchableImages(words, dtype, phrase));
        else if (dtype == 11) //email
            results = getMatchableEmails(phrase);
        else if (dtype == 3) //media
            results = getMatchableMedia(phrase);
        return results;
    }
}
return results;
}

private Hashtable parsePhrase(String phrase, int dtype) {
    Hashtable words = new Hashtable();
    StringTokenizer st = new StringTokenizer(phrase);
    int cnt = st.countTokens();
    int tokens_cnt = cnt + 1;
    while(st.hasMoreTokens()) {
        String token = st.nextToken();
        // System.out.println("*****token got from the phrase*****"+token);
        if(!StopWords.getInstance().stopWord(token)) {
            // System.out.println("*****token adding to the list*****"+token+"="+cnt);
            words.put(token,String.valueOf(cnt));
            cnt--;
        }
    }
    words.put(phrase, "10");
    return words;
}

private Vector getMatchableLinks(Hashtable words, int dtype, String phrase) {
    Vector v = new Vector();
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Hashtable urlHash = new Hashtable();
    Enumeration enum = words.keys();
    long t = System.currentTimeMillis();
    while(enum.hasMoreElements()) {
        String key = (String) enum.nextElement();
        int keyrank = Integer.parseInt((String)words.get(key));
        // System.out.println("Results for *****"+key);
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(GET_SEARCH_RESULTS);
            cstmt.setString(1, "%"+key+"%"+"%");
            cstmt.setInt(2, dtype);
            cstmt.registerOutParameter(3, OracleTypes.CURSOR);
            cstmt.execute();
            rset = (ResultSet) cstmt.getObject(3);
            while(rset.next()) {
                // System.out.println("=-----"+key+"=----------------"+rset.getString("url_desc"));
                boolean dupe = false;
                if(!urlHash.get(rset.getString("url_desc")) != null)
                    dupe = true;
            }
        }
    }
    return v;
}
double rank = rset.getDouble("rank");
if(rset.getString("title_Text") != null) {
    String ttext = rset.getString("title_Text");
    if(ttext.indexOf(sphrase) != -1)
        rank = rank + (keyrank * 700);
    if((ttext.toLowerCase()).indexOf(sphrase) != -1)
        rank = rank + (keyrank * 500);
    if((ttext.toLowerCase()).indexOf(sphrase.toLowerCase()) != -1)
        rank = rank * (keyrank * 300);
    StringTokenizer st = new StringTokenizer(ttext);
    int i = 0;
    while(st.hasMoreTokens()) {
        String test = st.nextToken();
        i++;
        if(test.equals(key))
            rank = rank + (keyrank * 999) - (i*100);
        else if(test.equalsIgnoreCase(key))
            rank = rank + (keyrank * 700) - (i*100);
    }
    //if title starts with the search phrase, then bump the rank a bit high
    if(rset.getString("title_Text") != null && rset.getString("title_Text").startsWith(sphrase))
        rank = rank + (keyrank * 9999);
    if(rset.getString("title_Text") != null &&
        rset.getString("title_Text").toLowerCase().startsWith(sphrase.toLowerCase())
        rank = rank + (keyrank * 8888);
    //if url_desc has the key init then bump the rank up
    if(rset.getString("url_desc").indexOf(key) != -1)
        rank = rank + (keyrank * 100);
    //testing
    String swords = (String) SearchProperties.getInstance().getProperty("word");
    StringTokenizer st = new StringTokenizer(swords, ",");
    while(st.hasMoreTokens()) {
        if(rset.getString("url_desc").indexOf(st.nextToken()) != -1)
            rank = rank + (keyrank * 100);
    }
    /*LOGGER.trace("before ="+rset.getString("url_desc")+"\n"+
        rset.getString("summary")+"\n"+
        rank);
    */
    if(!dupe) {
        String ttext = rset.getString("title_Text");
        if(ttext != null && ttext.length() > 90)
            ttext = ttext.substring(0,90)+"....";
        ResultsObject ro = new ResultsObject(rset.getString("url_desc"),

rset.getString("summary"),
    rank,
    ttext);
    urlHash.put(rset.getString("url_desc"),ro);
} else {
    ResultsObject ro = (ResultsObject) urlHash.get(rset.getString("url_desc"));
    ro.setRank(ro.getRank() + rank);
    urlHash.put(rset.getString("url_desc"),ro);
}

    long t1 = System.currentTimeMillis();
    System.out.println("Time took to return the results = " + (t1 - t) + " milliseconds");
} catch(Exception e) {
    LogException.handleError("Error while getting the urls from url_xref table",e);
} finally {
    ConnectionManager.cleanup(conn,csmt,rset);
}
}

    v.addAll(urlHash.values());
    return v;
}

private Vector getMatchableImages(Hashtable words, int dtype, String phrase) { 
    Vector v = new Vector();
    Connection conn = null;
    CallableStatement csmt = null;
    ResultSet rset = null;
    //Hash to remove dup
    Hashtable dup = new Hashtable();
    if(words.size() > 1) {
        Vector temp = getResultsVector(phrase, dtype, 125, dup);
        v.addAll(temp);
    }
    /*The rank needs to be reduced based on the term being queries*/
    int i=100;
    Enumeration enum = words.keys();
    while(enum.hasMoreElements()) {
        if(i >= 25)
            i = i - 25;
        else
            i = 0;
        String key = (String) enum.nextElement();
        Vector temp = getResultsVector(key, dtype, i, dup);
        v.addAll(temp);
    }
    return v;

134
private Vector getMatchableEmails(String phrase) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Vector v = new Vector();
    String word = phrase;
    HashTable hash = new HashTable();
    if (word.indexOf("@") != -1)
        word = phrase.substring(0, phrase.indexOf("@") + 1);
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_EMAIL_SEARCH_RESULTS);
        cstmt.setString(1, "%" + word + "%" + "");
        cstmt.registerOutParameter(2, OracleTypes.CURSOR);
        cstmt.execute();
        rset = (ResultSet) cstmt.getObject(2);
        if (rset.next()) {
            String wdesc = rset.getString("word_desc");
            if (wdesc != null)
                if (hash.get(wdesc) == null) {
                    ResultsObject ro = new ResultsObject(rset.getString("url_desc"),
                        wdesc);
                    v.add(ro);
                }
            hash.put(wdesc, "not-null");
        }
        catch (Exception e) {
            LogException.handleError("Error while getting the urls from url_xref table", e);
        }
    } finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
        hash = null;
    }
    return v;
}

private Vector getMatchableMedia(String phrase) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Vector v = new Vector();
    String word = phrase;
    if (word.indexOf("@") != -1)
        word = phrase.substring(0, phrase.indexOf("@") + 1);
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_MEDIA_SEARCH_RESULTS);
        cstmt.setString(1, "%" + word + "%" + "");
        cstmt.registerOutParameter(2, OracleTypes.CURSOR);
        cstmt.execute();
        rset = (ResultSet) cstmt.getObject(2);
        if (rset.next()) {
            String wdesc = rset.getString("word_desc");
            if (wdesc != null)
                if (hash.get(wdesc) == null) {
                    ResultsObject ro = new ResultsObject(rset.getString("url_desc"),
                        wdesc);
                    v.add(ro);
                }
            hash.put(wdesc, "not-null");
        }
        catch (Exception e) {
            LogException.handleError("Error while getting the urls from url_xref table", e);
        }
    } finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
        hash = null;
    }
    return v;
}
cstmt = conn.prepareStatement(GET_MEDIA_SEARCH_RESULTS);
cstmt.setString(1, "%"+word+"%");
cstmt.registerOutParameter(2, OracleTypes.CURSOR);
cstmt.executeQuery();
rset = (ResultSet) cstmt.getObject(2);
while(rset.next()) {
    ResultsObject ro = new ResultsObject(rset.getString("url_desc"),
                                          rset.getString("label"));
    v.add(ro);
}
} catch(Exception e) {
    LogException.handleError("Error while getting the urls from url_xref table",e);
} finally {
    ConnectionManager.cleanup(conn, cstmt, rset);
}
return v;

private Vector getResultSetVector(String sword, int dtype, int rank, Hashtable dups) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Vector v = new Vector();
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareStatement(GET_IMG_SEARCH_RESULTS);
        cstmt.setString(1, "%"+sword+"%");
        cstmt.setInt(2, dtype);
        cstmt.registerOutParameter(3, OracleTypes.CURSOR);
        cstmt.executeQuery();
        rset = (ResultSet) cstmt.getObject(3);
        while(rset.next()) {
            //testing
            /* LOGGER.trace("before ="+rset.getString("url_desc")+"\n"+
                            rset.getString("summary")+"\n"+
                            rset.getDouble("rank");* /
            //check for dups
            if(dups.get(rset.getString("url_desc")) == null) {
                ResultsObject ro = new ResultsObject(rset.getString("url_desc"),
                                                    rset.getString("label"),
                                                    rset.getString("dimensions"),
                                                    rset.getString("file_Size"),
                                                    rset.getString("image_type"),
                                                    rank);
                v.add(ro);
                dups.put(rset.getString("url_desc"), true);
            }
private Vector reArrangeResults(Vector results) {
    LinkedHashMap lm1 = new LinkedHashMap();
    LinkedHashMap lm2 = new LinkedHashMap();
    Vector finalResults = new Vector(100);
    for(int i=0;i<results.size();i++) {
        ResultsObject rs = (ResultsObject) results.get(i);
        String url = rs.getUrl();
        lm1.put(url,new Double(rs.getRank()));
        if(lm2.get(url) != null) {
            ResultsObject rs1 = (ResultsObject) lm2.get(url);
            if(rs1.getSummary() == null)
                lm2.put(url,rs);
        } else
            lm2.put(url,rs);
    }
    Vector v = sortMapByValuesDesc(lm1);
    for(int i=v.size()-1;i>=0;i--) {
        Object key = v.get(i);
        Object val = lm2.get(key);
        ResultsObject test = (ResultsObject)val;
        finalResults.add(test);
    }
    //testing
    /*for(int i=finalResults.size()-1;i>=0;i--) {
        ResultsObject test = (ResultsObject)finalResults.get(i);
        LOGGER.trace("url = "+test.getUrl()+"\nrank = "+test.getRank()+"\nsummary = "+test.getSummary());
    }*/
    return finalResults;
}

private Vector reArrangeImgResults(Vector results) {
    LinkedHashMap lm = new LinkedHashMap();
    Vector finalResults = new Vector();
    for(int i=0;i<results.size();i++) {
        ResultsObject rs = (ResultsObject) results.get(i);
        int rank = rs.getImgRank();
        lm.put(rs,new Integer(rank));
    }
Vector v = sortMapByIntValues(lm);
for(int i=v.size()-1;i>=0;i--) {
    Object key = v.get(i);
    finalResults.add(key);
}

//testing
/*@for(int i=finalResults.size()-1;i>=0;i--) {
    ResultsObject test = (ResultsObject)finalResults.get(i);
    LOGGER.trace("url = "+test.getUrl()+"\trank = "+test.getImgRank()+"\n"+test.getLabel());
}@*/
return finalResults;

public static Vector sortMapByValuesDesc(Map m) {
    Comparator c = new Comparator() {
        public int compare(Object o1, Object o2) {
            Double d1 = (Double)((Map.Entry)o1).getValue();
            Double d2 = (Double)((Map.Entry)o2).getValue();
            return d1.compareTo(d2);
        }
    };    
    Object[] o = m.entrySet().toArray();
    Arrays.sort(o, c);
    //LinkedList lm = new LinkedList(o.length);
    Vector v = new Vector();
    for (int i = 0; i < o.length; i++) {
        Map.Entry entry = (Map.Entry) o[i];
        //lm.put(entry.getKey(), entry.getValue());
        v.add(entry.getKey());
    }
    return lm;
    return v;
}

public static Vector sortMapByIntValues(Map m) {
    Comparator c = new Comparator() {
        public int compare(Object o1, Object o2) {
            Integer d1 = (Integer)((Map.Entry)o1).getValue();
            Integer d2 = (Integer)((Map.Entry)o2).getValue();
            return d1.compareTo(d2);
        }
    };    
    Object[] o = m.entrySet().toArray();
Arrays.sort(o, c);
//LinkedHashMap lh = new LinkedHashMap(o.length);
Vector v = new Vector();
for (int i = 0; i < o.length; i++) {
    Map.Entry entry = (Map.Entry) o[i];
    //lh.put(entry.getKey(), entry.getValue());
    v.add(entry.getKey());
}
//return lh;
return v;
}

public static void main(String[] args) {
    LoggerInit.initLogger();
    Search.getInstance().getSearchResults("SCSU", 1);
}

20. com.scsu.searchengine.search.ResultSetObject.java
package com.scsu.searchengine.search;
/*
 * ResultSetObject holds bean implementation for search results
 */

public class ResultSetObject {
    private String summary;
    private String url;
    private String word_desc;
    private double rank;
    private String title;
    private String label;
    private String dim;
    private String file_size;
    private String img_type;
    private int img_rank;
    public ResultSetObject(String url, String summary, double rank, String title) {
        this.url(url);
        this.summary(summary);
        this.rank(rank);
        this.title(title);
    }
    public ResultSetObject(String url, String label, String dimensions, String file_size, String image_type, int rank) {
        this.url(url);
        this.label(label);
        this.dimensions(dimensions);
        this.fileSize(file_size);
    }
this.setImgType(image_type);
this.setImgRank(rank);
}

public ResultsObject(String url, String word_Desc) {
    this.setUrl(url);
    this.setWordDesc(word_Desc);
}

private void setUrl(String url) {
    this.url = url;
}

private void setWordDesc(String word) {
    this.word_desc = word;
}

private void setSummary(String summary) {
    this.summary = summary;
}

private void setRank(double rank) {
    this.rank = rank;
}

private void setTitle(String title) {
    this.title = title;
}

public String getUrl() {
    return url;
}

public String getWordDesc() {
    return word_desc;
}

public String getSummary() {
    if(summary != null && summary.length() > 255)
        return summary.substring(0, 255);
    return summary;
}

public double getRank() {
    return rank;
}

public String getTitle() {
    return title;
}

private void setLabel(String label) {
    this.label = label;
}

private void setDimensions(String dimensions) {
    this.dim = dimensions;
}
private void setFileSize(String file_size) {
    this.file_size = file_size;
}

private void setImageType(String image_type) {
    this.img_type = image_type;
}

private void setImgRank(int rank) {
    this.img_rank = rank;
}

public String getLabel() {
    return label;
}

public String getDimensions() {
    return dim;
}

public String getFileSize() {
    return file_size;
}

public String getImageType() {
    return img_type;
}

public int getImgRank() {
    return img_rank;
}

package com.scsu.searchengine.search;

/*
 * Cacher stores the result objects until tomcat is bounced
 */
import java.util.Hashtable;
import java.util.Vector;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.util.LogException;

public class Cacher {
    private static Logger LOGGER = Logger.instance(Cacher.class.getName());
    private static Cacher _instance = null;
    private static Hashtable _cache = new Hashtable();
    public static Cacher getInstance() {
        if(_instance == null)
            _instance = new Cacher();
        return _instance;
    }
    public void cacheIt(Vector results, String phrase, int type) {

    }
String name = phrase + "_" + type;
    _cache.put(name, results);
}
public boolean cached(String phrase, int type) {
    String name = phrase + "_" + type;
    if(_cache.get(name) != null)
        return true;
    return false;
}
public Object getCachedRowSet(String phrase, int type) {
    String name = phrase + "_" + type;
    return _cache.get(name);
}

22. com.scsu.searchengine.search.Servlet.java
package com.scsu.searchengine.search;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.Properties;
import java.util.Vector;
import javax.servlet.ServletConfig;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.velocity.Template;
import org.apache.velocity.context.Context;
import org.apache.velocity.servlet.VelocityEngine;
import org.apache.velocity.exception.ResourceNotFoundException;
import org.apache.velocity.exception.ParseErrorException;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.util.LogException;

public class SearchServlet extends VelocityServlet {
    private static final Logger LOGGER = Logger.getInstance(SearchServlet.class.getName());
    /**
     * Called by the VelocityServlet
     * init(). We want to set a set of properties
     * so that templates will be found in the webapp
     * root. This makes this easier to work with as
     * an example, so a new user doesn't have to worry
     * about config issues when first figuring things
     * out
     */

protected Properties loadConfiguration(ServletConfig config) 
    throws IOException, FileNotFoundException {
    LoggerInit.initLogger();
    Properties p = new Properties();
    String path = config.getServletContext().getRealPath("/");
    if (path == null) {
        LOGGER.trace("SearchServlet.loadConfiguration() : unable to "
            + "get the current webapp root. Using '/' Please fix.");
        path = "/";
    }
    p.setProperty(Velocity.FILE_RESOURCE_LOADER_PATH, path);
    p.setProperty("runtime.log", path + "velocity.log");
    return p;
}
/**
 * <p>
 * main routine to handle a request. Called by
 * VelocityServlet, your responsibility as programmer
 * is to simply return a valid Template
 * </p>
 * @param ctx a Velocity Context object to be filled with
 * data. Will be used for rendering this
 * template
 * @return Template to be used for request
 */
public Template handleRequest(HttpServletRequest request,
        HttpServletResponse response,
        Context ctx) {
        if (request.getParameter("getresults") != null &&
            request.getParameter("getresults").equals("true")) {
                processSearch(request, response, ctx);
                return serveTemplate("/results.html");
            }
        return serveTemplate("/search.html");
    }
private void processSearch(HttpServletRequest request,
        HttpServletResponse response,
        Context ctx) {
        String phrase = request.getParameter("search");
        int type =
            request.getParameter("searchtype")!=null?Integer.parseInt(request.getParameter("searchtype")):1;
            LOGGER.trace("Search phrase = "+phrase + "\ntype = "+type);
if(!Cacher.getInstance().cached(phrase, type)) {
    Vector results = Search.getInstance().getSearchResults(phrase, type);
    LOGGER.trace("Results Vector Size = "+results.size());
    Cacher.getInstance().cachelt(results, phrase, type);
    if(results.size() == 0)
        ctx.put("noresults","true");
    else
        processResults(request, response, ctx, results);
} else
    processResults(request, response, ctx, (Vector) Cacher.getInstance().getCachedRowSet(phrase, type));
    ctx.put("phrase",phrase);
    ctx.put("type", String.valueOf(type));
}
private Template serveTemplate(String templateUrl) {
    Template template = null;
    try {
        template = getTemplate(templateUrl);
    } catch(ResourceNotFoundException e1) {
        LogException.handleException("",e1);
    } catch(ParseErrorException e2) {
        LogException.handleException("",e2);
    } catch(Exception e3) {
        LogException.handleException("",e3);
    }
    return template;
}
public void processResults(HttpServletRequest request,
    HttpServletResponse response,
    Context ctx,
    Vector results) {
    Vector list = new Vector();
    if(results.size() > 0) {
        if(request.getParameter("direction")!= null)
            ctx.put("direction", request.getParameter("direction").toString());
        int pageNum = 1;
        if(request.getParameter("page") != null)
            pageNum = Integer.parseInt(request.getParameter("page"));
        int start = (pageNum - 1) * 10;
        int end = pageNum * 10;
        if(results.size() < end)
            end = results.size();
        for(int i=start; i<end; i++)
            list.add(results.get(i));
        int pages = results.size()/10;
    }
if(results.size() % 10 > 0)
    pages = pages + 1;
ctx.put("results", list);
ctx.put("pages", new Integer(pages));
ctx.put("selpage", new Integer(pageNum));
ctx.put("start", String.valueOf(start));
ctx.put("end", String.valueOf(end));
ctx.put("size", String.valueOf(results.size()));
}
else {
    ctx.put("noresults","true");
}

23. com.scsu.searchengine.util.SearchProperties.java
package com.scsu.searchengine.util;
import java.io.InputStream;
import java.util.Properties;
import java.util.Enumeration;
import java.util.StringTokenizer;
import java.util.Vector;
import java.util.Hashtable;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.util.LogException;
public class SearchProperties {
    private static Logger LOGGER = Logger.instance(SearchProperties.class.getName());
    private static SearchProperties _instance = null;
    private final static String _defaultProperties = "search.prop";
    private static Properties _props = new Properties();
    private static Hashtable _styles = new Hashtable();
    private static Hashtable _wstyles = new Hashtable();
    public static SearchProperties getInstance() {
        if(_instance == null)
            _instance = new SearchProperties();
        return _instance;
    }
    private SearchProperties() {
        init();
    }
    private void init() {
        try {
            InputStream in = SearchProperties.class.getResourceAsStream(_defaultProperties);
            _props.load(in);
            loadProps();
        }
    }
}
```java
} catch (Exception e) {
    LogException.handleError("Error while reading the Search.props file", e);
}

private void loadProps() {
    Enumeration enum = _props.keys();
    Hashtable tempDesc = new Hashtable();
    Hashtable tempRank = new Hashtable();
    while (enum.hasMoreElements()) {
        String key = (String) enum.nextElement();
        String value = (String) _props.get(key);
        StringTokenizer st = new StringTokenizer(key, ":");
        Vector v = new Vector();
        while (st.hasMoreTokens()) {
            v.add(st.nextToken());
        }
        if (v.size() == 3) {
            // Possibly this is where the style descriptions and
            // Style ranks will fall
            //LOGGER.info("Vector size == " + v.size() + "\n 0 =" + v.get(0) + "\n 1 =" + v.get(1) + "\n 2 =" + v.get(2));
            if (v.get(2).toString().equals("desc")) {
                String dval = (String) tempRank.get(v.get(1));
                //LOGGER.info("dval = " + dval);
                if (dval == null) {
                    //LOGGER.info("Writing to deschash, key = " + v.get(1) + " value = " + value);
                    tempDesc.put(v.get(1), value);
                }
                else {
                    _styles.put(value, dval);
                    //LOGGER.info("store rank to styles, key = " + value + " value = " + dval);
                    tempDesc.remove(v.get(1));
                }
            }
            else if (v.get(2).toString().equals("rank")) {
                String dval = (String) tempDesc.get(v.get(1));
                //LOGGER.info("dval = " + dval);
                if (dval == null) {
                    //LOGGER.info("Writing to rankhash, key = " + v.get(1) + " value = " + value);
                    tempRank.put(v.get(1), value);
                }
                else {
                    //LOGGER.info("store rank to styles, key = " + dval + " value = " + value);
                }
            }
        }
    }
}
```
_styles.put(dval,value);
    tempRank.remove(v.get(1));
}
}
} else if(v.size() == 2) {
    /*
    This is where the general word ranks will fall.
    */
    //LOGGER.info("Vector Size == "+v.size()+"nWritng to words style, key= "+v.get(0)+" 
    value= "+value);
    _wstyles.put(v.get(0),value);
}
}
}
public Hashable getStyles() {
    return _styles;
}

public Hashable getWordStyles() {
    return _wstyles;
}

public int getStyleRank(String sdesc) {
    int style = 0;
    try {
        style = Integer.parseInt((String) _styles.get(sdesc));
    } catch (NumberFormatException nfe) {
        LogException.handleError("Error while trying to get the Style rank for sdesc = "+sdesc,nfe);
    }
    return style;
}

public int getWordStyleRank(String sdesc) {
    int style = 0;
    try {
        style = Integer.parseInt((String) _wstyles.get(sdesc));
    } catch (NumberFormatException nfe) {
        LogException.handleError("Error while trying to get the Word Style rank",nfe);
    }
    return style;
}

public Object getProperty(String key) {
    return _props.get(key);
}

public static void main (String[] args) {
    LoggerInit.initLogger();
}
public class Extensions {
    private static Logger LOGGER = Logger.instance(Extensions.class.getName());
    private static Extensions_instance = null;
    String[] movie_ext = {"mp3", "MP3",
        ".ram", ".RAM",
        ".wmv", ".WMV",
        ".wma", ".WMA",
        ".wvx", ".WVX",
        ".wax", ".WAX",
        ".asf", ".ASF",
        ".asx", ".ASX",
        ".wms", ".WMS",
        ".wmz", ".WMZ",
        ".wmd", ".WMD",
        ".avi", ".AVI",
        ".dv", ".DV",
        ".FLV", ".f4v",
        ".MOV", ".mov",
        ".mp4", ".MP4",
        ".ra", ".RA",
        ".rm", ".RM",
        ".swf", ".SWF"};
    String[] img_ext = {"bmp", ".BMP",
        ".cgm", ".CGM",
        ".dib", ".DIB",
        ".eps", ".EPS",
        ".img", ".IMG",
        ".gif", ".GIF",
        ".jas", ".JAS",
        ".jpg", ".JPG",
        ".mac", ".MAC",
        ".msp", ".MSP",
        ".pcx", ".PCX",
        ".pic", ".PIC";
".ras", ".RAS",
".raw", ".RAW",
".rle", ".RLE",
".tga", ".TGA",
".tif", ".TIF",
".wpg", ".WPG"
};
public static Extensions getInstance() {
    if (_instance == null)
        _instance = new Extensions();
    return _instance;
}
public boolean isFileFormat(int type, String url) {
    boolean format = false;
    switch (type) {
        case DocTypes.IMAGE:
            for (int i = 0; i < img_ext.length; i++) {
                if (url.endsWith(img_ext[i])) {
                    format = true;
                    break;
                }
            }
            break;
        case DocTypes.MEDIA:
            for (int i = 0; i < movie_ext.length; i++) {
                if (url.endsWith(movie_ext[i])) {
                    format = true;
                    break;
                }
            }
            break;
        default:
            format = false;
            break;
    }
    return format;
}
}

25. com.scsu.searchengine.util.HTTP.java
package com.scsu.searchengine.util;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.HttpRecoverableException;
import org.apache.commons.httpclient.MultiThreadedHttpConnectionManager;
import org.apache.commons.httpclient.HttpStatus;
import com.scsu.searchengine.util.LogException;
import com.scsu.searchengine.controller.PageVisitor;
import com.scsu.searchengine.logging.Logger;

public class HTTP {
    private static Logger LOGGER = Logger.instance(HTTP.class.getName());
    private static MultiThreadedHttpClientConnectionManager connectionManager = null;

    public String downloadWWWPage(String pageURL) {
        String buffer = "";
        HttpClient client = new HttpClient(getCM());
        client.setResponseTimeout(10000);
        GetMethod get = new GetMethod(pageURL);
        try {
            client.executeMethod(get);
            buffer = get.getUserName();
        } catch (HttpException e) {
            LogException.handleError("This is a recoverable exception so trying again url = " + pageURL,e);
            try {
                client.executeMethod(get);
                buffer = get.getUserName();
            } catch (Exception e) {
                LogException.handleError("Exception while reading url = " + pageURL,e);
                try {
                    LOGGER.trace("Writing an invalid url to the LOG file");
                    PageVisitor.pw.println(pageURL + "invalid" + "n");
                } catch (Exception e1) {
                    LogException.handleError("Error writing invalid url = " + pageURL + " to the page database",e1);
                }
            }
        } catch (Exception e) {
            LogException.handleError("Exception while reading url = " + pageURL,e);
            try {
                LOGGER.trace("Writing an invalid url to the LOG file");
                PageVisitor.pw.println(pageURL + "invalid" + "n");
            } catch (Exception e1) {
                LogException.handleError("Error writing invalid url = " + pageURL + " to the page database",e1);
            }
        }
    }

    public static void main(String[] args) {
        String url = "http://example.com"
        String page = downloadWWWPage(url);
        System.out.println(page);
    }
}
return ""
;

} finally {
    get.releaseConnection();
}

return buffer;

}  

public int getWWWPageStreamSize(String url) {
    int size = 0;
    HttpClient client = new HttpClient(getCM());
    client.set_timeout(10000);
    GetMethod get = new GetMethod(url);
    try {
        client.executeMethod(get);
        byte[] buffer[] = get.getResponseBody();
        size = buffer.length;
    } catch(Exception e) {
        LogException.handleError("Exception while reading url = "+url,e);
        try {
            LOGGER.trace("Writing an invalid url to the pageDatabase");
            PageVisitor.pdwriter.write(url+"\invalid");
        } catch(Exception e) {
            LogException.handleError("Error writing invalid url = "+url+" to the pagedatabase",e);
        }
    } finally {
        get.releaseConnection();
    }

    return size;

}  

public boolean isValidURL(String url) {
    boolean valid = true;
    HttpClient client = new HttpClient(getCM());
    client.set_timeout(10000);
    GetMethod get = new GetMethod(url);
    try {
        int status = client.executeMethod(get);
        if(status == HttpStatus.SC_BAD_REQUEST)
            valid = false;
    } catch(Exception e) {
        valid = false;
        LogException.handleError("Exception while closing inpStreams = "+url,e);
        try {
            LOGGER.trace("Writing an invalid url to the pageDatabase");
            PageVisitor.pdwriter.write(url+"\invalid");
        } catch(Exception e) {
            LogException.handleError("Error writing invalid url = "+url+" to the pagedatabase",e);
        }
    }

    return valid;

}
private MultiThreadedHttpConnectionManager getCM() {
    if (connectionManager == null)
        connectionManager = new MultiThreadedHttpConnectionManager();
    return connectionManager;
}

26. com.scsu.searchengine.util.ImageInfo.java
package com.scsu.searchengine.util;
/*
* ImageInfo.java
*
* Version 1.1
*
* A Java class to determine image width, height and color depth for
* a number of image file formats.
* @author swathi
*/

import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Vector;
import java.util.Hashtable;
import java.util.Enumeration;
import java.net.URL;
import java.util.StringTokenizer;
import java.io.InputStream;
import java.awt.image.BufferedImage;
import java.net.URLEncoder;
import javax.imageio.ImageIO;
import javax.imageio.ImageReader;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.parser.ParserDelegator;
import javax.swing.text.html.HTML;
import javax.swing.text.html.MutableAttributeSet;
import javax.swing.text.html.HTMLEditorKit.ParserCallback;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleCallableStatement;
import org.apache.commons.httpclient.HttpMethod;
import org.apache.commons.httpclient.methods.GetMethod;
import org.apache.commons.httpclient.HttpClient;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerFactory;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;

public class ImageInfo {
    private static Logger LOGGER = LoggerFactory.getLogger(ImageInfo.class);
    private static final String GET_IMAGES_LIST = "{call search_pkg.get_image_urls(?)}";
    private static final String UPDATE_DIMS = "{call search_pkg.update_img_dimensions(?)}";
    public void getImages() {
        Connection conn = null;
        CallableStatement cstnt = null;
        ResultSet rset = null;
        Vector imgs = new Vector();
        try {
            conn = ConnectionManager.getConnection();
            cstnt = conn.prepareCall(GET_IMAGES_LIST);
            cstnt.registerOutParameter(1, OracleTypes.CURSOR);
            cstnt.execute();
            rset = (ResultSet) cstnt.getObject(1);
            while(rset.next()) {
                ImageDims dims = new ImageDims();
                dims.url_id = rset.getLong("url_id");
                dims.url_desc = rset.getString("url_desc");
                dims.dims = "";
                imgs.add(dims);
            }
        } catch(Exception e) {
            LogException.handleError("Error while getting the image list from images table", e);
        } finally {
            ConnectionManager.cleanup(conn, cstnt, rset);
        }
        processImages(imgs);
        updateDims(imgs);
    }

    private void processImages(Vector imgs) {
        for(int i=0; i<imgs.size(); i++) {
            ImageDims dim = (ImageDims) imgs.get(i);
            String url = dim.url_desc;
            String dsn = ""
            HttpClient client = new HttpClient();
            client.setTimeout(10000);
    }
GetMethod get = null;
InputStream in = null;
try {
    get = new GetMethod(URITool.encodePathQuery(url));
    client.executeMethod(get);
    in = get.getResponseBodyAsStream();
    BufferedImage bi = ImageIO.read(in);
    int width = bi.getWidth();
    int height = bi.getHeight();
    dsn = width + "X" + height;
    LOGGER.trace("Dims for "+url+" = "+dsn);
} catch(Exception e) {
    LogException.handleError("Error while getting the image dimensions from image url = "+url,e);
} finally {
    get.releaseConnection();
}
dim.dims = dsn;
}

private void updateDims(Vector imgs) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(UPDATE_DIMS);
        ((OracleCallableStatement)cstmt).setExecuteBatch(50);
        for(int i=0;i<imgs.size();i++) {
            ImageDims dim = (ImageDims) imgs.get(i);
            long id = dim.url_id;
            String dsn = dim.dims;
            cstmt.setLong(1, id);
            cstmt.setString(2, dsn);
            System.out.println("Updating : image_id = "+id+" dims = "+dsn);
            cstmt.executeUpdate();
        }
    } catch(Exception e) {
        LogException.handleError("Error while bulk update of image Dimensions",e);
    } finally {
        ConnectionManager.cleanup(conn,cstmt,rset);
    }
}

public static void main(String args[]) {

154

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LoggerInit.initLogger();
    ImageInfo ii = new ImageInfo();
    ii.getImages();
}

private class ImageDims {
    String dims;
    String url_desc;
    long url_id;
}

27. com.scsu.searchengine.util.ImageLabels.java
package com.scsu.searchengine.util;
import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Vector;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.HashMap;
import java.net.URL;
import java.util.StringTokenizer;
import java.io.StringReader;
import java.net.URLEncoder;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.parser.ParserDelegator;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLEditorKit.ParserCallback;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleCallableStatement;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;
/
* ImageLabels updates the labels for images. Page titles or subheading are also considered in determining the title of the image.
@Author: Swathi K
*/
public class ImageLabels {
    private static Logger LOGGER = Logger.instance(ImageLabels.class.getName());
    private static ImageLabels _instance = null;
    static Hashable summaries = new Hashable();
    private static final String GET_IMAGES_LIST = "{call search_pkg.get_img_urls(?)}";
private static final String GET_IMAGE_URLREFS = "\{'call search_pkg.get_imgref_urls(?,?)\'}";
private static final String UPDATE_IMAGE_LABELS = "\{'call search_pkg.update_img_labels(?,?)\'}";

public static ImageLabels getInstance() {
    if (_instance == null)
        _instance = new ImageLabels();
    return _instance;
}

public void getImageList() {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Hashtable imgs = new Hashtable();
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_IMAGES_LIST);
        cstmt.registerOutParameter(1, OracleTypes.CURSOR);
        cstmt.execute();
        rset = (ResultSet) cstmt.getObject(1);
        while (rset.next()) {
            String label = rset.getString("label") != null ? rset.getString("label").toString() : "";
            imgs.put(rset.getString("url_id"), label);
        }
    } catch (SQLException e) {
        LogException.handleException("Error while getting the image list from images table", e);
    }
    finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
    }
    processImages(imgs);
    updateLabels(imgs);
}

private void processImages(Hashtable imgs) {
    Enumeration enum = imgs.keys();
    while (enum.hasMoreElements()) {
        Object key = enum.nextElement();
        long id = Long.parseLong((String) key);
        Vector ref_urls = getReferenceUrls(id);
        String ldesc = null;
        for (int j = 0; j < ref_urls.size(); j++) {
            if (ldesc != null)
                j = ref_urls.size() + 10;
            else
                ldesc = getLabel((String) ref_urls.get(j));
        }
        //update it again
    }
}
String label = (String) imgs.get(key);
if(ldesc != null) {
    label = label + ": " + ldesc;
}
imgs.put(key, label);

private Vector getReferenceUrls(long url_id) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    Vector v = new Vector();
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_IMAGE_URLREFS);
        cstmt.setLong(1, url_id);
        cstmt.registerOutParameter(2, OracleTypes.CURSOR);
        cstmt.execute();
        rset = (ResultSet) cstmt.getObject(2);
        while(rset.next()) {
            v.add(rset.getString(1));
        }
    } catch(Exception e) {
        LogException.handleException("Error while getting the image list from images table",e);
    } finally {
        ConnectionManager.cleanup(conn,cstmt,rset);
    }
    return v;
}

private String getLabel(String url) {
    try {
        //url encoding
        String page1 = "";
        StringTokenizer st = new StringTokenizer(url);
        int count = st.countTokens();
        if(count > 1) {
            int i=0;
            StringBuffer sb = new StringBuffer();
            while(st.hasMoreTokens()) {
                i++;
                sb.append(st.nextToken());
                if(i < count)
                    sb.append("%20");
            }
            page1 = sb.toString();
    }
} else
    page1 = url;
    URL page = new URL(page1);
    String pageString = new HTTP().downloadWWWPage(page1);
    String label = CleanUpTags(pageString, page);
    return label;
} catch (Exception e) {
    LogException.handleError("Exception while getting summary for url = "+url,e);
}
return null;

private String CleanUpTags(String pageString, URL page) {
    Hashtable hash = new Hashtable();
    try {
        HTMLEditorKit htmlKit = new HTMLEditorKit();
        HTMLDocument htmlDoc = (HTMLDocument)htmlKit.createDefaultDocument();
        HTMLEditorKit.Parser parser = new ParserDelegator();
        LabelParser pp = new LabelParser();
        StringReader br = new StringReader(pageString);
        parser.parse(br, pp, true);
        return pp.getLabel();
    } catch (Exception e) {
        LogException.handleError("Error while getting clean text from html","e);
    }
    return null;
}

private void updateLabels(Hashtable imgs) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(UPDATE_IMAGE_LABELS);
        ((OracleCallableStatement)cstmt).setExecuteBatch(50);
        Enumeration enum = imgs.keys();
        while (enum.hasMoreElements()) {
            Object key = enum.nextElement();
            long id = Long.parseLong((String)key);
            String label = (String)imgs.get(key);
            System.out.println("Updating : image_id = "+id+
label = "+label);
            cstmt.executeUpdate();
    }
catch(Exception e) {
    LogException.handleError("Error while bulk update of image labels", e);
} finally {
    ConnectionManager.cleanup(conn, stmt, rs);
}

private class LabelParser extends ParserCallback {
    StringBuffer sb = new StringBuffer();
    int script = 0;
    int title = 0;
    int realTitle = 0;
    int h4 = 0;
    String _ttext = null, ttext = null;
    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        if (t == HTML.Tag.SCRIPT) {
            script = 1;
        } else if (t == HTML.Tag.TITLE) {
            realTitle = 1;
        } else if (t == HTML.Tag.H4) {
            h4 = 1;
        } else if (t == HTML.Tag.H2) {
            h4 = 1;
        } else if (t == HTML.Tag.STYLE) {
            script = 1;
        }
    }
    public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        handleSimpleTag(t, a, pos);
    }
    public void handleText(char[] data, int pos) {
        if (script == 1) {
            script = 0;
            // do nothing
            return;
        }
        String text = new String(data);
        if (title == 0) {
            title = 1;
            LOGGER.trace("Title = "+text);
            ttext = text;
        } else if (h4 == 1) {
            h4 = 0;
            ttext = text;
        }
public String getLabel() {
  if (text != null && text.length() > 4000)
    return text.substring(0, 3990);
  return text;
}

public static void main(String[] args) {
  LoggerInit.initLogger();
  ImageLabels.getInstance().getImageList();
}

package com.scsu.searchengine.util.LogException.java
import com.scsu.searchengine.util;
import com.scsu.searchengine.logging.Logger;

/*
 * The errors are logged in error.log
 */
public class LogException {
  private static Logger LOGGER = Logger.instance("error.log");
  public static void handleError(String reason, Exception e) {
    LOGGER.error(reason, e);
  }
}

package com.scsu.searchengine.util.PageRanks.java
import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Hashtable;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Comparator;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.TreeSet;
import java.util.TreeMap;
import java.util.TreeSet;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleCallableStatement;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;

Page Rank Calculation

/*
 @author Swathi K
 */

public class PageRanks {
  private static Logger LOGGER = Logger.instance(PageRanks.class.getName());
  private static PageRanks _instance = null;
  private static Hashtable urls = new Hashtable();
  private static LinkedHashMap initRanks = new LinkedHashMap();
  private static final String GET_URL_LINKS = "{call search_pkg.get_url_links(?)}";
  private static final String UPDATE_RANKS = "{call search_pkg.update_ranks(?)}";
  private static final String GET_HREF_CNT = "{call search_pkg.get_link_cnt(?)}";
  private static final String GET_LNK_CNT = "{call search_pkg.get_urls_cnt(?)}";

  public static PageRanks getInstance() {
    if (_instance == null)
      _instance = new PageRanks();
    return _instance;
  }

  protected void getUrlLinks() {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
      long lnk_cnt = getLinkCount();
      double fraction = 1 / lnk_cnt;
      conn = ConnectionManager.getConnection();
      cstmt = conn.prepareCall(GET_URL_LINKS);
      cstmt.registerOutParameter(1, OracleTypes.CURSOR);
      cstmt.execute();
      rset = (ResultSet) cstmt.getObject(1);
      while (rset.next()) {
        String val = (String) urls.get(String.valueOf(rset.getLong("url_id")));
        if (val == null)
          urls.put(rset.getString("url_id"), rset.getString("url_ref_id"));
        else {
          StringBuffer sb = new StringBuffer(val);
          sb.append(",");
          sb.append(rset.getString("url_ref_id"));
          urls.put(rset.getString("url_id"), sb.toString());
        }
        initRanks.put(new Integer(rset.getString("url_id")), new Double(fraction));
        initRanks.put(new Integer(rset.getString("url_ref_id")), new Double(fraction));
      }
    }
    finally {
      if (conn != null) conn.close();
      if (cstmt != null) cstmt.close();
      if (rset != null) rset.close();
    }
  }
}

161

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
} catch(Exception e) {
    LogException.handleError("Error while getting the urls from url_xref table",e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,rset);
}

private long getLinkCount() {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_LNK_CNT);
        cstmt.registerOutParameter(1, OracleTypes.LONGVARCHAR);
        cstmt.execute();
        long cnt = cstmt.getLong(1);
        return cnt;
    } catch(Exception e) {
        LogException.handleError("Error while getting the count of urls",e);
    } finally {
        ConnectionManager.cleanup(conn,cstmt,rset);
    }
    return 0;
}

protected void calculateRanks() {
    double damp_factor = 0.85;
    LinkedHashSet<Url>Urls = new LinkedHashSet<Url>();
    Enumeration enum = urls.keys();
    while(enum.hasMoreElements()) {
        double PRN = 0;
        String key = (String) enum.nextElement();
        String refs = (String) urls.get(key);
        StringTokenizer st = new StringTokenizer(refs, ",");
        while(st.hasMoreTokens()) {
            String pid = st.nextToken();
            //LOGGER.info("pid = "+pid + ",tval = "+((Double) initRanks.get(new Integer(pid))).doubleValue());
            double pr = ((Double) initRanks.get(new Integer(pid))).doubleValue();
            String plinks = (String) urls.get(pid);
            if(plinks == null) {
                /*This will happen only if the crawling is incomplete*/
                plinks = retrieveLnkCnt(pid);
            }
        }
    }
}
StringTokenizer st1 = new StringTokenizer(plinks, "\", "");
int cnt = st1.countTokens();
PRN += pr/cnt;
}
double fin = (1 - damp_factor) + damp_factor * PRN;
finRanks.put(new Integer(key), new Double(fin));
}
if(converged(initRanks, finiRanks))
i=51;
//initRanks.putAll(finiRanks);
LOGGER.info("*******************completed round************* = "+i);

//testing area
/*Iterator iter = initRanks.keySet().iterator();
while(iter.hasNext()) {
    Object key = iter.next();
    double val = ((Double) initRanks.get(key)).doubleValue();
    LOGGER.info("url_id = "+((Integer)key).longValue()+\"\t\rank = "+val);
}*/
i++;
}
}

protected void updateRanks() {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(UPDATE_RANKS);
        ((OracleCallableStatement)cstmt).setExecuteBatch(50);
        Iterator iter = initRanks.keySet().iterator();
        while(iter.hasNext()) {
            Object key = iter.next();
            double val = ((Double) initRanks.get(key)).doubleValue();
            cstmt.setLong(1, ((Integer)key).longValue());
            cstmt.setDouble(2, val);
            cstmt.executeUpdate();
        }
    } catch(Exception e) {
        LogException.handleError("Error while getting the urls from url_xref table", e);
    } finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
    }
}
private String retrieveLnkCnt(String uid) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    StringBuffer sb = new StringBuffer();
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(GET_HREF_CNT);
        cstmt.setLong(1, Long.parseLong(uid));
        cstmt.registerOutParameter(2, OracleTypes.CURSOR);
        cstmt.execute();
        rset = (ResultSet) cstmt.getObject(2);
        while(rset.next()) {
            sb.append(rset.getString(1));
            sb.append(",");
        }
        urls.put(uid, sb.toString());
    } catch(Exception e) {
       LogException.handleException("Error while getting the urls from url_xref table", e);
    } finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
    }
    return sb.toString();
}

public static LinkedHashMap sortMapByValues(Map m) {
    Comparator c = new Comparator() {
        public int compare(Object o1, Object o2) {
            Double d1 = (Double) ((Map.Entry) o1).getValue();
            Double d2 = (Double) ((Map.Entry) o2).getValue();
            return d1.compareTo(d2);
        }
    };
    Object[] o = m.entrySet().toArray();
    Arrays.sort(o, c);
    LinkedHashMap lhm = new LinkedHashMap(o.length);
    for (int i = 0; i < o.length; i++) {
        Map.Entry entry = (Map.Entry) o[i];
        lhm.put(entry.getKey(), entry.getValue());
    }
    return lhm;
}

private boolean converged(LinkedHashMap lm1, LinkedHashMap lm2) {
    LinkedHashMap _lm2 = sortMapByValues(lm2);
    boolean done = Arrays.equals(lm1.keySet().toArray(), _lm2.keySet().toArray());
    //testing area
}
/*Iterator iter1 = lm1.keySet().iterator();
Iterator iter2 = _lm2.keySet().iterator();
while(iter1.hasNext() && iter2.hasNext()) {
  Object key = iter1.next();
  double val = ((Double) lm1.get(key)).doubleValue();
  Object key1 = iter2.next();
  double val1 = ((Double) _lm2.get(key1)).doubleValue();
  System.out.println("url_id = "+(Integer)key).longValue()+
                  "trank = "+val+"url_id = "+(Integer)key1).longValue()+
                  "trank = "+val1);
}/*

//put up all sorted values
initRanks = _lm2;
return done;
}

public static void main(String[] args) {
  LoggerInit.initLogger();
  LOGGER.info("updating page ranks");
  long t1 = System.currentTimeMillis();
  PageRanks.getInstance().getUrlLinks();
  LOGGER.info("Got url link structure from DB in "+ (System.currentTimeMillis()-t1) + " milliseconds.");
  PageRanks.getInstance().calculateRanks();
  LOGGER.info("Ranks converged in "+ (System.currentTimeMillis()-t1) + " milliseconds.");
  PageRanks.getInstance().updateRanks();
  long t2 = System.currentTimeMillis();
  LOGGER.info("Took "+ (t2-t1) + " milliseconds to compute and update page ranks");
}

29. com.scsu.searchengine.util.DocTypes.java
package com.scsu.searchengine.util;

public interface DocTypes {
  public static final int DOCUMENT = 1;
  public static final int IMAGE = 2;
  public static final int MEDIA = 3;
  public static final int PDF = 4;
  public static final int PS = 5;
  public static final int PHP = 6;
  public static final int WORD = 7;
  public static final int TXT = 8;
  public static final int EXCEL = 9;
  public static final int PPT = 10;
  public static final int EMAIL = 11;

  165

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
30. com.scsu.searchengine.util.StopWords.java
package com.scsu.searchengine.util;
import java.io.FileReader;
import java.io.BufferedReader;
import java.util.Hashtable;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.util.SearchProperties;
public class StopWords {
    private static Logger LOGGER = Logger.instance(StopWords.class.getName());
    private static StopWords _instance = null;
    private static Hashtable stoplist = new Hashtable();
    public static StopWords getInstance() {
        if (_instance == null)
            _instance = new StopWords();
        return _instance;
    }
    protected StopWords() {
        init();
    }
    private void init() {
        try {
            FileReader fileReader = new
FileReader((String)SearchProperties.getInstance().getProperty("stopwords"));
            BufferedReader br = new BufferedReader(fileReader);
            String list = ";
            while((list = br.readLine()) != null) {
                //System.out.println("-------------------------stop word------ = "+list);
                stoplist.put(list.trim().toLowerCase(),"none");
            }
        } catch (Exception e) {
            LogException.handleError("Error getting the stopwords file location from Search.props file",e);
        }
    }
    public boolean stopWord(String word) {
        if(stoplist.size() == 0) {
            init();
        }
        //System.out.println("word ===== "+word.trim().toLowerCase()+"*********
*********="+stoplist.get(word.toLowerCase()));
        if(stoplist.get(word.trim().toLowerCase()) != null)
            return true;
        else if(word.trim().length() == 0) //takes care of spaces
return true;
else if(word.equals(" ")) //takes care of spaces that cannot be trimmed
return true;
else if(isRepetitive(word))
    return true;
return false;

/*This method captures words with same repeating chars like #### or ----*/
private boolean isRepetitive(String word) {
    String fchar = word.substring(0,1);
    if(stoplist.get(fchar.toLowerCase()) != null) {
        for(int i=1;i<word.trim().length();i++) {
            String schar = null;
            if(i+1 >= word.trim().length())
                schar = word.substring(i);
            else
                schar = word.substring(i,i+1);
            if(!schar.equalsIgnoreCase(fchar))
                return false;
        }
        return true;
    }
    return false;
}

package com.scsu.searchengine.util;
public interface Styles {
    public static final String BOLD = "bold";
    public static final String ITALIC = "italic";
    public static final String META = "meta";
    public static final String UNDERLINE = "underline";
    public static final String BIGFONT = "bigfont";
    public static final String H1 = "h1";
    public static final String H2 = "h2";
    public static final String H3 = "h3";
    public static final String H4 = "h4";
    public static final String H5 = "h5";
    public static final String H6 = "h6";
    //Word styles
    public static final String CAPITAL = "capital";
    public static final String ANCHOR = "anchor";
    public static final String TITLE = "title";
}
32. com.scsu.searchengine.util.Summary.java
package com.scsu.searchengine.util;

import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Vector;
import java.util.Hashtable;
import java.net.URL;
import java.util.StringTokenizer;
import java.io.StringReader;
import java.net.URLDecoder;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLEditorKit;
import javax.swing.text.html.parser.ParserDelegator;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLEditorKit.ParserCallback;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleCallableStatement;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;

public class Summary {

    private static Logger LOGGER = Logger.instance(Summary.class.getName());
    private static Summary _instance = null;
    //static Hashtable summaries = new Hashtable();

    private static final String GET_SUMMARY_LIST = "\{call search_pkg.get_summary_list(?)}";
    private static final String GET_SUMMARY_LIST1 = "\{call search_pkg.get_summary_list1(?)}";
    private static final String UPDATE_SUMMARIES = "\{call search_pkg.update_summary(?),?}\";

    public static Summary getInstance() {
        if (_instance == null)
            _instance = new Summary();
        return _instance;
    }

}
public void getUrlWordList() {
    Connection conn = null;
    CallableStatement stmt = null;
    ResultSet rset = null;
    Vector v = new Vector();

    try {
        conn = ConnectionManager.getConnection();
        stmt = conn.prepareCall(GET_SUMMARY_LIST);
        stmt.registerOutParameter(1, OracleTypes.CURSOR);
        stmt.execute();
        rset = (ResultSet) stmt.getObject(1);
        while(rset.next()) {
            SummaryList sl = new SummaryList();
            sl.word_id = rset.getLong("word_id");
            sl.url_id = rset.getLong("url_id");
            sl.word_desc = rset.getString("word_desc");
            sl.url_desc = rset.getString("url_desc");
            sl.summary = "";
            sl.title = "";
            v.add(sl);
            LOGGER.trace("Adding url element to vector : "+rset.getLong("url_id"));
        }
    } catch(Exception e) {
        LogException.handleError("Error while getting the urls from url_xref table",e);
    } finally {
        ConnectionManager.cleanup(conn,stmt,rset);
    }
    updateSummaries(v);
}

public void getUrlWordList() {
    Connection conn = null;
    CallableStatement stmt = null;
    ResultSet rset = null;
    boolean notdone = true;
    while(notdone) {
        SummaryList sl = new SummaryList();
        conn = null;
        stmt = null;
        rset = null;
        try {

conn = ConnectionManager.getConnection();
cstmt = conn.prepareCall(GET_SUMMARY_LIST1);
cstmt.registerOutParameter(1, OracleTypes.CURSOR);
cstmt.executeQuery();
rs = (ResultSet)cstmt.getObject(1);
if(!rs.next()) {
    sl.word_id = rs.getLong("word_id");
    sl.url_id = rs.getLong("url_id");
    sl.word_desc = rs.getString("word_desc");
    sl.url_desc = rs.getString("url_desc");
    sl.summary = "";
    sl.title = ""
} else {
    notdone = false;
}
catch(Exception e) {
    LogException handleError("Error while getting the urls from url_xref table", e);
} finally {
    ConnectionManager.cleanup(conn, cstmt, rs);
}
updateSummaries(sl);

/* private void updateSummaries(Vector v) {*/
    Vector newSet = new Vector();
    for(int i=0;i<v.size();i++) {
        SummaryList sl = (SummaryList)v.get(i);
        long word_id = sl.word_id;
        long url_id = sl.url_id;
        String url = sl.url_desc;
        String word = sl.word_desc;
        LOGGER.trace("word_id = "+word_id+
          "\url_id = "+url_id+
          "\url_desc = "+url+
          "\tword_desc = "+word);
        /if(summaries.get(url) == null) {
          Hashable hash = getSummary(url, word);
          sl.summary = (String) hash.get("summary");
          sl.title = (String) hash.get("title");
          /}*/
        else {
            sl.summary = getSummary1((Hashable)summaries.get(url), word);
            sl.title = (String) ((Hashable)summaries.get(url)).get("title");
        }*/
    /* updateImmidiate(sl);*/
}
LOGGER.trace("Summary updated..................for \n"+ i);
    // newList.addAll(sl);
    }

    // uploadSummariesToDB(newList);
}

public void updateSummaries(SummaryList sl) {
    long word_id = sl.word_id;
    long url_id = sl.url_id;
    String url = sl.url_desc;
    String word = sl.word_desc;
    LOGGER.trace("word_id = "+word_id+
    "url_id = "+url_id+
    "url_desc = "+url+
    "word_desc = "+word);
    Hashable hash = getSummary(url, word);
    sl.summary = (String) hash.get("summary");
    sl.title = (String) hash.get("title");
    updateImmidiate(sl);
}

public Hashable getSummary(String url, String word) {
    try {
        //testing
        String page1 = "";
        StringTokenizer st = new StringTokenizer(url);
        int count = st.countTokens();
        if(count > 1) {
            int i=0;
            StringBuffer sb = new StringBuffer();
            while(st.hasMoreTokens()) {
                i++;
                sb.append(st.nextToken());
                if(i < count)
                    sb.append("%20");
            }
            page1 = sb.toString();
        } else
            page1 = url;
        URL page = new URL(page1);
        String pageString = new HTTP().downloadWWWPage(page1);
        Hashable hash = new Hashable();
        if(pageString != null & & pageString.trim().length() > 0) {
            Hashable cleanPage = CleanUpTags(pageString, page);
            //summarizes.put(url, cleanPage);
        }
    }
    catch(Exception e) {
    }
//System.out.println(cleanPage);
String title = (String) cleanPage.get("title");
if(title == null) title="";
if(title.length() > 1000)
    title = title.substring(0, 990);
    hash.put("summary",getSummary1(cleanPage,word));
    hash.put("title",title);
} else {
    hash.put("summary","");
    hash.put("title","");
}
return hash;
} catch (Exception e) {
    LogException.handleError("Exception while getting summary for url = "+url,e);
}
return null;

private String getSummary1(Hashtable cleanpage, String word) {
    String page = (String) cleanpage.get("cleanText");
    StringTokenizer st = new StringTokenizer(page,".");
    while(st.hasMoreTokens()) {
        String token = st.nextToken();
        if(token.indexOf(word.toLowerCase()) != -1 || token.indexOf(word) != -1) {
            if(token.length() > 4000)
                token = token.substring(0, 3950);
            return token;
        }
    }
    st = new StringTokenizer(page,".");
    if(st.countTokens() > 0) {
        String token = st.nextToken();
        if(token.length() > 4000)
            token = token.substring(0, 3950);
        return token;
    }
    return "";
}

private Hashtable CleanUpTags(String pageString, URL page) {
    Hashtable hash = new Hashtable();
    try {
        HTMLEditorKit htmlKit = new HTMLEditorKit();
        HTMLDocument htmlDoc = (HTMLDocument)htmlKit.createDefaultDocument();
        HTMLEditorKit.Parser parser = new ParserDelegator();
MyParser pp = new MyParser(page.toString());
StringReader br = new StringReader(pageString);
parser.parse(br, pp, true);
hash.put("cleanText", pp.getCleanText());
// System.out.println("clean summary = "+pp.getCleanText());
hash.put("title", pp.getTitle());
} catch (Exception e) {
    LogException.handleError("Error while getting clean text from html", e);
}
return hash;
}

private void uploadSummariesToDB(Vector newList) {
    Connection conn = null;
    CallableStatement csstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        csstmt = conn.prepareCall("UPDATE SUMMARIES (SELECT word_id, url_id, tolower(summary), tolower(title) FROM " + table + " WHERE id = ?) WHERE id = ?;");
        csstmt.setInt(1, newList.size());
        System.out.println("Uploading summaries to db...");
        for(int i = 0; i < newList.size(); i++) {
            SummaryList sl = (SummaryList) newList.get(i);
            csstmt.setLong(1, sl.word_id);
            csstmt.setLong(2, sl.url_id);
            csstmt.setString(3, sl.summary);
            csstmt.setString(4, sl.title);
            System.out.println("Updating: word_id = " + sl.word_id +
                              " url_id = " + sl.url_id +
                              " summary = " + sl.summary +
                              " title = " + sl.title +
                              " iteration = " + i);
            csstmt.executeUpdate();
        }
    } catch (Exception e) {
        LogException.handleError("Error while bulk update of summaries", e);
    } finally {
        ConnectionManager.cleanup(conn, csstmt, rset);
    }
}

private void updateImmediate(SummaryList slist) {
    Connection conn = null;
    CallableStatement csstmt = null;
    ResultSet rset = null;

    try {
        conn = ConnectionManager.getConnection();
        csstmt = conn.prepareCall("UPDATE SUMMARIES (SELECT word_id, url_id, tolower(summary), tolower(title) FROM " + table + " WHERE id = ?) WHERE id = ?;");
        csstmt.setInt(1, slist.size());
        System.out.println("Uploading summaries to db...");
        for(int i = 0; i < slist.size(); i++) {
            SummaryList sl = (SummaryList) slist.get(i);
            csstmt.setLong(1, sl.word_id);
            csstmt.setLong(2, sl.url_id);
            csstmt.setString(3, sl.summary);
            csstmt.setString(4, sl.title);
            System.out.println("Updating: word_id = " + sl.word_id +
                              " url_id = " + sl.url_id +
                              " summary = " + sl.summary +
                              " title = " + sl.title +
                              " iteration = " + i);
            csstmt.executeUpdate();
        }
    } catch (Exception e) {
        LogException.handleError("Error while bulk update of summaries", e);
    } finally {
        ConnectionManager.cleanup(conn, csstmt, rset);
    }
}
try {
    conn = ConnectionManager.getConnection();
    cstmt = conn.prepareCall(UPDATE_SUMMARIES);
    System.out.println("------------------------summary----------------
"+slist.summary+"\tlength"+slist.summary.length());
    System.out.println("------------------------title----------------
"+slist.title+"\tlength"+slist.title.length());
    cstmt.setLong(1, slist.word_id);
    cstmt.setLong(2, slist.url_id);
    cstmt.setString(3, slist.summary);
    cstmt.setString(4, slist.title);
    cstmt.executeUpdate();
} catch(Exception e) {
    LogException.handleError("Error while update summaries "+
    "word_id = " + slist.word_id +
    "url_id = " + slist.url_id,e);
}
} finally {
    ConnectionManager.cleanup(conn,cstmt,rset);
}

private class SummaryList {
    long word_id;
    long url_id;
    String summary;
    String url_desc;
    String word_desc;
    String title;
}

private class MyParser extends ParserCallback {
    StringBuffer sb = new StringBuffer();
    int script = 0;
    int title = 0;
    int realTitle = 0;
    int h4 = 0;
    String _text = null, ttext = null;
    String currUrl = null;

    MyParser(String url) {
        if(url.indexOf("#") != -1)
            this.currUrl = url.substring(url.indexOf("#")+1);
    }

    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {

// System.out.println(((HTML.Tag)t).toString());
if (t == HTML.Tag.SCRIPT) {
    script = 1;
} else if(t == HTML.Tag.TITLE) {
    realTitle = 1;
} else if(t == HTML.Tag.H4) {
    h4 = 1;
} else if(t == HTML.Tag.H2) {
    h4 = 1;
} else if(t == HTML.Tag.STYLE) {
    script = 1;
} else if(t == HTML.Tag.SELECT) {
    script = 1;
} else if(t == HTML.Tag.A) {
    if(currUrl != null) {
        String name = (String) a.getAttribute(HTML.Attribute.NAME);
        if(name != null && name.equalsIgnoreCase(currUrl))
            sb = new StringBuffer();
    } else if (t == HTML.Tag.FONT) {
        String size = (String) a.getAttribute(HTML.Attribute.SIZE);
        System.out.println("size = " + size);
        if(size != null && size.equalsIgnoreCase("+1")) {
            h4 = 1;
        }
    }
}

public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    handleSimpleTag(t,a,pos);
}

public void handleEndTag(HTML.Tag t, int pos) {
    if (t == HTML.Tag.SCRIPT)
        script = 0;
    else if(t == HTML.Tag.SELECT)
        script = 0;
}

public void handleText(char[] data,int pos) {
    if(script == 1) {
        //script = 0;
        //do nothing
        return;
    }

    175

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
```java
public String getCleanText() {
    return sb.toString();
}

private void setTitle(String title) {
    _text = title;
}

public String getTitle() {
    if (_text != null && _text.length() > 1000)
        return _text.substring(0, 990);
    if (_text == null)
        return "";
    return _text;
}
```
public static void main(String[] args) {
    LoggerInit.initLogger();
    Summary.getInstance().getUrlWordList();
    /*String url = "http://www.southernct.edu/departments/graduatestudies/thesis.php3";
     Hashtable hash = Summary.getInstance().getSummary(url, "Thesis");
     System.out.println("Title : "+ hash.get("title");
     System.out.println("Summary : "+ hash.get("summary");
     */
}

33. com.scsu.searchengine.util.WordRanks.java
package com.scsu.searchengine.util;
import java.sql.Connection;
import java.sql.CallableStatement;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;
public class WordRanks {
    private static Logger LOGGER = Logger.instance(WordRanks.class.getName());
    private static WordRanks_instance = null;
    private static final String UPDATE_RANKS = "{call search_pkg.update_word_ranks(?,?,?)";
    public static WordRanks getInstance() {
        if(_instance == null)
            _instance = new WordRanks();
        return _instance;
    }
    public void runProc() {
        Connection conn = null;
        CallableStatement cstmt = null;
        SearchProperties sp = SearchProperties.getInstance();
        LOGGER.info("Starting to run the search_pkg.update_word_ranks proc....");
        try {
            conn = ConnectionManager.getConnection();
            cstmt = conn.prepareCall(UPDATE_RANKS);
            cstmt.setInt(1, sp.getWordStyleRank(Styles.CAPITAL));
            cstmt.setInt(2, sp.getWordStyleRank(Styles.ANCHOR));
            cstmt.setInt(3, sp.getWordStyleRank(Styles.TITLE));
            cstmt.execute();
        } catch(Exception e) {
            LogException.handleError("Error while running the search_pkg.update_word_ranks proc\n", e);
        } finally {

public static void main(String[] args) 
{
    LOGGER.info("Calling runProc()");
    long t1 = System.currentTimeMillis();
    WordRanks.getInstance().runProc();
    long t2 = System.currentTimeMillis();
    LOGGER.info("Took "+ (t2-t1) + " milliseconds to run the proc");
}
}

34. com.scsu.searchengine.util.LevelCrawler
package com.scsu.searchengine.util;

import java.net.URL;
import java.util.Hashtable;
import java.utilEnumeration;
import java.util.StringTokenizer;
import java.io.StringReader;
import java.io.FileWriter;
import java.util.Vector;

import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLDocumentFactory;
import javax.swing.text.html.htmlwriter.ParserDelegator;
import javax.swing.text.html.HTML;
import javax.swing.text.html.MutableAttributeSet;
import javax.swing.text.html.HTMLEditorKit.ParserCallback;

import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.util.LogException;

public class LevelCrawler {

    private static Logger LOGGER = Logger.getInstance(LevelCrawler.class.getName());

    public static void main(String args[]) 
    {
        LOGGER.info("Calling runProc()");
        LevelCrawler lc = new LevelCrawler();
        if(args.length != 2) 
        { 
            System.out.println("Usage: <url> <level>");
            System.exit(0);
        }
String url = args[0];
int level = Integer.parseInt(args[1]);
Hashtable urlList = new Hashtable();
urlList.put(url,"true");
for(int i=0;i<level;i++) {
    urlList = lc.crawlList(urlList);
    LOGGER.trace("="+LEVEL "+i+"=");
    Enumeration enum = urlList.keys();
    while(enum.hasMoreElements())
        LOGGER.trace((String)enum.nextElement());
}

FileWriter lcfile = null;
try {
    lcfile = new FileWriter((String)SearchProperties.getInstance().getProperty("levelcrawler"),true);
    Enumeration enum = urlList.keys();
    while(enum.hasMoreElements())
        lcfile.write((String) enum.nextElement()+"n");
} catch(Exception e) {
    LogException.handleError("Exception while writing the final list",e);
} finally {
    if(lcfile != null)
        try {
            lcfile.close();
        } catch (Exception e1) {
        }
}

public Hashtable crawlList(Hashtable list) {
    Hashtable newList = new Hashtable();
    Enumeration enum = list.keys();
    while(enum.hasMoreElements()) {
        String page = (String) enum.nextElement();
        if(homeRelated(page)) {
            try {
                URL url = new URL(page);
                StringReader sr = new StringReader(new HTTP().downloadWwwPage(page));
                HTMLEDitorKit htmlKit = new HTMLEDitorKit();
                HTMLDocument htmlDoc = (HTMLDocument)htmlKit.createDefaultDocument();
                HTMLEDitorKit.Parser parser = new ParserDelegator();
                LinkParser pp = new LinkParser(url);
                } catch (Exception e) {
                    LogException.handleError("Exception while crawling",e);
                }
            }
parser.parse(sr, pp, true);
newList.putAll(pp.getUrlList());
} catch (Exception e) {
    LogException.handleError("Exception while Level crawling", e);
}
}
return newList;

private boolean homeRelated(String page) {
    URL url = null;
    try {
        url = new URL(page);
    } catch (Exception e) {
        LogException.handleError("Malformed URL " + page, e);
    }
    String host = url.getHost();
    Vector v = SearchProperties.getInstance().getHosts();
    for (int i = 0; i < v.size(); i++) {
        String host = (String) v.get(i);
        if (host.endsWith(host))
            return true;
    }
    return false;
}

private class LinkParser extends ParserCallback {

    Hashtable list = new Hashtable();
    URL base;

    public LinkParser (URL base) {
        this.base = base;
    }

    public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
        if (t == HTML.Tag.A) {
            String href = (String) a.getAttribute(HTML.Attribute.HREF);
            if (href == null)
                return;
            else if (href.indexOf("javascript:"!) != -1)
                return;
            else if (href.toLowerCase().startsWith("mailto:"))
                return;

    }
String url = handleLink(base, href);
    list.put(url, "true");
}

public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    handleSimpleTag(t, a, pos);
}

public void handleEndElement(HTML.Tag t, int pos) {
}

public void handleText(char[] data, int pos) {
}

public HasTable getUrlList() {
    return list;
}

public String handleLink(URL base, String str) {
    String absURL = null;
    URL test = null;
    if (str.toLowerCase().startsWith("http")) {
        // it is an absolute url
        try {
            test = new URL(str);
            absURL = str;
        } catch (Exception e) {
            LogException.handleError("(((bad absolute url from " + base.toString() + "tfailed url = " + str, e);
            absURL = null;
        }
    } else {
        // not an abs url so make it absolute
        // handle directories
        StringBuffer sb = new StringBuffer();
        // handle parameters issue
        String tempbase = base.toString();
        if (base.toString().indexOf("?") != -1)
            tempbase = tempbase.substring(0, base.toString().indexOf("?") - 1);
        if (tempbase.indexOf("#") != -1)
            tempbase = tempbase.substring(0, tempbase.indexOf("#") - 1);
        if (base.isFile().indexOf(".") == -1 && !str.startsWith("/")) {

181
sb.append(tempbase);
sb.append("/\n";
sb.append(str);
} else {
    sb.append(base.getProtocol());
    sb.append("://\n";
    sb.append(base.getHost());
    if(!str.startsWith("/\n"))
        sb.append("/\n";
    sb.append(str);
}

boolean goodone = true;
String temp = sb.toString();
absURL = str;
if(temp.indexOf(" ") != -1) {
    StringTokenizer st = new StringTokenizer(absURL, " ");
    StringBuffer sb1 = new StringBuffer();
    int cnt = st.countTokens();
    int i=0;
    while(st.hasMoreTokens()) {
        i++;
        sb1.append(st.nextToken());
        if(i < cnt)
            sb1.append("%20");
    }
    temp = sb1.toString();
}
absURL = temp;  //may or may not be an encoded url;
}
return absURL;

}

35. com.scsu.searchengine.util.MediaLabels
package com.scsu.searchengine.util;

import java.sql.Connection;
import java.sql.CallableStatement;
import java.sql.ResultSet;
import java.util.Vector;
import java.util.Hashtable;
import java.util Enumeration;
import java.net.URL;
import java.util.StringTokenizer;
import java.io.StringReader;
import java.net.URLDecoder;
import javax.swing.text.html.HTMLDocument;
import javax.swing.text.html.HTMLDocumentEditorKit;
import javax.swing.text.html.parser.ParserDelegator;
import javax.swing.text.html.HTML;
import javax.swing.text.MutableAttributeSet;
import javax.swing.text.html.HTMLDocumentEditorKit.ParserCallback;
import oracle.jdbc.driver.OracleTypes;
import oracle.jdbc.driver.OracleCallableStatement;
import com.scsu.searchengine.logging.Logger;
import com.scsu.searchengine.logging.LoggerInit;
import com.scsu.searchengine.database.ConnectionManager;
import com.scsu.searchengine.util.LogException;

public class MediaLabels {

    private static Logger LOGGER = Logger.instance(MediaLabels.class.getName());
    private static MediaLabels_instance = null;
    static Hashtable summaries = new Hashtable();

    private static final String GET_IMAGES_LIST = "\{call search_pkg.get_media_urls(?)\}";
    private static final String GET_IMAGE_URLREFS = "\{call search_pkg.get_imgref_urls(?,?)\}"
    private static final String UPDATE_IMAGE_LABELS = "\{call search_pkg.update_media_labels(?,?)\}"

    public static MediaLabels getInstance() {
        if (instance == null)
            instance = new MediaLabels();
        return instance;
    }

    public void getMediaList() {
        Connection conn = null;
        CallableStatement cstmt = null;
        ResultSet rset = null;
        Hashtable imgsls = new Hashtable();

        try {
            conn = ConnectionManager.getConnection();

        } finally {
            if (cstmt != null) cstmt.close();
            if (rset != null) rset.close();
            if (conn != null) conn.close();
        }
    }
}
cstmt = conn.prepareCall(GET_IMAGES_LIST);
cstmt.registerOutParameter(1, OracleTypes.CURSOR);
cstmt.execute();
rs = (ResultSet) cstmt.getObject(1);
while(rs.next()) {
    String label = rs.getString("label")!=null?rs.getString("label"):"";
    imgs.put(rs.getString("url_id"), label);
}
} catch(Exception e) {
    LOG.error("Error while getting the image list from images table",e);
} finally {
    ConnectionManager.cleanup(conn,cstmt,rs);
}
processMedia(imgs);
updateLabels(imgs);

private void processMedia(Hashtable imgs) {
    Enumeration enum = imgs.keys();
    while(enum.hasMoreElements()) {
        Object key = enum.nextElement();
        long id = Long.parseLong((String)key);
        Vector ref_urls = getReferenceUrls(id);
        String ldesc = null;
        for(int j=0; j<ref_urls.size(); j++) {
            if(ldesc != null)
                j = ref_urls.size() + 10;
            else
                ldesc = getLabel((String)ref_urls.get(j));
        }
        //update it again
        String label = (String) imgs.get(key);
        if(ldesc != null) {
            label = label + " : " + ldesc;
        }
        imgs.put(key, label);
    }
}

private Vector getReferenceUrls(long url_id) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rs = null;
    Vector v = new Vector();

    private void processMedia(Hashtable imgs) {
        Enumeration enum = imgs.keys();
        while(enum.hasMoreElements()) {
            Object key = enum.nextElement();
            long id = Long.parseLong((String)key);
            Vector ref_urls = getReferenceUrls(id);
            String ldesc = null;
            for(int j=0; j<ref_urls.size(); j++) {
                if(ldesc != null)
                    j = ref_urls.size() + 10;
                else
                    ldesc = getLabel((String)ref_urls.get(j));
            }
            //update it again
            String label = (String) imgs.get(key);
            if(ldesc != null) {
                label = label + " : " + ldesc;
            }
            imgs.put(key, label);
        }
    }
    private Vector getReferenceUrls(long url_id) {
        Connection conn = null;
        CallableStatement cstmt = null;
        ResultSet rs = null;
        Vector v = new Vector();
try {
    conn = ConnectionManager.getConnection();
    cstmt = conn.prepareCall(GET_IMAGE_URLREFS);
    cstmt.setLong(1, url_id);
    cstmt.registerOutParameter(2, OracleTypes.CURSOR);
    cstmt.execute();
    rset = (ResultSet) cstmt.getObject(2);
    while(rset.next()) {
        v.add(rset.getString(1));
    }
} catch(Exception e) {
    LogException.handleError("Error while getting the image list from images table", e);
} finally {
    ConnectionManager.cleanup(conn, cstmt, rset);
}
return v;

private String getLabel(String url) {
try {
    //url encoding
    String page1 = "",
    StringTokenizer st = new StringTokenizer(url);
    int count = st.countTokens();
    if(count > 1) {
        int i=0;
        StringBuffer sb = new StringBuffer();
        while(st.hasMoreTokens()) {
            i++;
            sb.append(st.nextToken());
            if(i < count)
                sb.append("%20");
        }
        page1 = sb.toString();
    } else
        page1 = url;
    URL page = new URL(page1);
    String pageString = new HTTP().downloadWWWPage(page1);
    String label = CleanUpTags(pageString, page);
    return label;
} catch (Exception e) {
    LogException.handleError("Exception while getting summary for url = "+url, e);
}
return null;
}
private String CleanUpTags(String pageString, URL page) {
    Hashtable hash = new Hashtable();
    try {
        HTMLEditorKit htmlKit = new HTMLEditorKit();
        HTMLDocument htmlDoc = (HTMLDocument)htmlKit.createDefaultDocument();
        HTMLEditorKit.Parser parser = new ParserDelegator();
        LabelParser pp = new LabelParser();
        StringReader br = new StringReader(pageString);
        parser.parse(br, pp, true);
        return pp.getLabel();
    } catch(Exception e) {
        LogException.handleError("Error while getting clean text from html", e);
    }
    return null;
}

private void updateLabels(Hashtable imgs) {
    Connection conn = null;
    CallableStatement cstmt = null;
    ResultSet rset = null;
    try {
        conn = ConnectionManager.getConnection();
        cstmt = conn.prepareCall(UPDATE_IMAGE_LABELS);
        ((OracleCallableStatement)cstmt).setExecuteBatch(50);
        Enumeration enum = imgs.keys();
        while(enum.hasMoreElements()) {
            Object key = enum.nextElement();
            long id = Long.parseLong((String)key);
            String label = (String) imgs.get(key);
            cstmt.setLong(1, id);
            cstmt.setString(2, label);
            System.out.println("Updating : image_id = "+id+
            label = "+label);
            cstmt.executeUpdate();
        }
    } catch(Exception e) {
        LogException.handleError("Error while bulk update of image labels", e);
    } finally {
        ConnectionManager.cleanup(conn, cstmt, rset);
    }
}

private class LabelParser extends ParserCallback {
    186

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
StringBuffer sb = new StringBuffer();
int script = 0;
int title = 0;
int realTitle = 0;
int h4 = 0;
String _text = null, ttext = null;
public void handleSimpleTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    if (t == HTML.Tag.SCRIPT) {
        script = 1;
    } else if (t == HTML.Tag.TITLE) {
        realTitle = 1;
    } else if (t == HTML.Tag.H4) {
        h4 = 1;
    } else if (t == HTML.Tag.H2) {
        h4 = 1;
    } else if (t == HTML.Tag.STYLE) {
        script = 1;
    }
}

public void handleStartTag(HTML.Tag t, MutableAttributeSet a, int pos) {
    handleSimpleTag(t, a, pos);
}

public void handleText(char[] data, int pos) {
    if (script == 1) {
        script = 0;
        //do nothing
        return;
    }
    String text = new String(data);
    if (title == 0) {
        title = 1;
        LOGGER.trace("Title = "+text);
        ttext = text;
    }
    if (h4 == 1) {
        h4 = 0;
        ttext = text;
    }
}

public String getLabel() {
    if (ttext != null && ttext.length() > 4000)
        return ttext.substring(0, 3990);
public static void main(String[] args) {
    LoggerInit.initLogger();
    MediaLabels.getInstance().getMediaList();
}

36. com.scsu.searchengine.util.Utility
package com.scsu.searchengine.util;

import java.net.URL;
import java.util.Vector;
import java.util.StringTokenizer;

import com.scsu.searchengine.processors.*;

public class Utility {

    public static String getAbsoluteURL(URL base, String str) {
        if(str != null) {
            try {
                URL u2 = new URL(base, str.trim());
                return u2.toString();
            } catch (Exception e) {
                LogException.handleError("Exception getting absurl, base= "+base.toString()+" str= "+str,e);
                return null;
            }
        }
        return null;
    }

    public static int getDocType(String url) {
        int doc_type = DocTypes.DOCUMENT;
        if(Extensions.getInstance().isFileFormat(DocTypes.MEDIA,url))
            doc_type = DocTypes.MEDIA; // multi media
        else if(Extensions.getInstance().isFileFormat(DocTypes.IMAGE,url))
            doc_type = DocTypes.IMAGE; // image file
        else if(url.endsWith(".PDF") || url.endsWith(".pdf"))
            doc_type = DocTypes.PDF;
        else if(url.endsWith(".PS") || url.endsWith(".ps"))
            doc_type = DocTypes.PS;
        // else if....
    }

    private static MediaLabels labels = MediaLabels.getInstance();

    // Other methods...

    public void getMediaList() {
        for(int i = 0; i < labels.size(); i++) {
            MediaLabel label = labels.get(i);
            System.out.println(label.getName());
        }
    }
}

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
else if(url.endsWith(".TXT") || url.endsWith(".txt"))
    doc_type = DocTypes.TXT;
else if(url.endsWith(".DOC") || url.endsWith(".doc"))
    doc_type = DocTypes.WORD;
else if(url.endsWith(".XLS") || url.endsWith(".xls"))
    doc_type = DocTypes.EXCEL;
else if(url.endsWith(".PPT") || url.endsWith(".ppt"))
    doc_type = DocTypes.PPT;
else if(url.startsWith("mailto") || url.startsWith("MAILTO"))
    doc_type = DocTypes.EMAIL;
return doc_type;
}

public static Vector checkinWords(String data, String url, int title, int rel_posn, int anchor, int style, boolean posn_cnt) {
    Vector twords = new Vector();
    //build object for words

    String text = new String(data);
    if(text.trim().startsWith(">") || text.trim().startsWith("/>
        text = text.substring(text.indexOf("">")+1);
    StringTokenizer st = new StringTokenizer(text);
    while(st.hasMoreTokens()) {
        WordProcessor wp = null;
        String token = st.nextToken();
        if(token.trim().length() > 0) {
            if(!StopWords.getInstance().stopWord(token)) {
                String stoken = token;
                if(!stoken.startsWith("http") && !stoken.startsWith("www"))
                    stoken = StopWords.getInstance().stripSpaces(token);
                int caps = 0;
                char[] arr = stoken.toCharArray();
                if(Character.isUpperCase(arr[0]))
                    caps = 1;
                if(posn_cnt)
                    wp = new WordProcessor(url, stoken.trim(), title, rel_posn++, caps, anchor, style);
                else
                    wp = new WordProcessor(url, stoken.trim(), title, rel_posn, caps, anchor, style);
                twords.add(wp);
            }
        }
    }
    return twords;
}
public static Vector checkinWord(String data, String url, int title, int rel_posn, int rel_style) {
    Vector words = new Vector();
    if(data != null && data.trim().length() > 0) {
        String word = new String(data);
        if(!word.startsWith("http") && !word.startsWith("www"))
            word = StopWords.getInstance().stripSpaces(new String(data));
        WordProcessor wp = new WordProcessor(url.toString(), word, 1, 0, 1, 0, style);
        words.add(wp);
    }
    return words;
}

public static Vector checkinEmail(String data, String url, int title, int rel_posn, int rel_style) {
    WordProcessor wp = new WordProcessor(url.toString(), data, 1, 0, 1, 0, style);
    Vector words = new Vector();
    words.add(wp);
    return words;
}

public static URLProcessor checkinURL(String href, boolean baseurl, String ref_url, int doc_type_id) {
    URLProcessor up = new URLProcessor(href, doc_type_id, baseurl, ref_url);
    return up;
}

public static boolean homeRelatedPage(String page) {
    URL url = null;
    try {
        url = new URL(page);
    } catch (Exception e) {
        LogException.handleError("Malformed URL "+page,e);
    }
    String host = url.getHost();
    Vector v = SearchProperties.getInstance().getHosts();
    for(int i=0;i<v.size();i++) {
        String host = (String) v.get(i);
        if(host.endsWith(host))
            return true;
    }
    return false;
}

public static String getUrlFromJavaScript(URL base, String url) {
    String part1 = url.substring(url.indexOf("/")+1,url.indexOf("")-1);
    if(part1.startsWith("" || part1.startsWith(""))
        return part1;
}
part1 = part1.substring(1);
//Strip the last quots
if (part1.indexOf("\") != -1)
    part1 = part1.substring(0, part1.indexOf("\")-1);

//Now validate the url
part1 = getAbsoluteURL(base, part1);
if (part1 != null && new HTTP().isValidURL(part1))
    return part1;
return null;
}

public static void main(String args[]) throws Exception {
    System.out.println(Utility.getUrlFromJavaScript(new URL("http://www.southernct.edu"),

    }
}

37. ConnectionManager.properties
driver = oracle.jdbc.driver.OracleDriver
#this will be the default url
url=jdbc:oracle:thin:test/test@localhost:1521:thesis
# the minimum and maximum size
# of the pool. -1 indicates that there
# is no value for this constraint
scsu.database.jdbcPool.minsize=10
scsu.database.jdbcPool.maxsize=30
# the amount of time in milliseconds before an
# inactive connection is closed. The time reflects
# the last time the connection was used
scsu.database.jdbcPool.expiration=1200000
# the frequency that the pool checks for expired
# connections
scsu.database.jdbcPool.frequency=180000
#user name and password for the database
screenName=test
password=test

38. Logger.props
#the purpose of this file is to document the logger.xml location
log4j-init-
file=C:/Swathi/Assignments/Thesis/src/classes/com/scsu/searchengine/logging/logger.xml
#specify if logging needs to be enabled
enabled=true

191
39. Logger.xml

<!-- This file holds the properties of logger. -->
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd" [
<!ENTITY MyCategory "com.scsu.searchengine.logging.MyCategory">
]>

<log4j:configuration debug="false">
<!-- Indicate what factory is being used by PPCSCategory -->
categoryFactory name="MyCategory" class="com.scsu.searchengine.logging.MyCategory$MyFactory.class" />

<!-- Console Logger -->
<appender name="CONSOLE" class="org.apache.log4j.ConsoleAppender">
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d [%t] %-5p %20.30c - %m%n"/>
  </layout>
</appender>

<!-- Daily Rolling File Logger -->
<appender name="ROLL_FILE_1" class="org.apache.log4j.DailyRollingFileAppender">
  <param name="File" value="daily.log"/>
  <param name="DatePattern" value="."yyyy-MM-dd" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d [%t] %-5p %20.30c - %m%n"/>
  </layout>
</appender>

<appender name="ERROR.Roll_FILE"
  class="org.apache.log4j.DailyRollingFileAppender">
  <param name="File" value="error.log"/>
  <param name="DatePattern" value="."yyyy-MM-dd" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%d [%t] %-5p %20.30c - %m%n"/>
  </layout>
</appender>

<!-- specific category specifications -->
category name="error.log" class="com.scsu.searchengine.logging.MyCategory">
  <priority value="INFO" class="com.scsu.searchengine.logging.MyPriority"/>
  <appender-ref ref="ERROR.Roll_FILE" />
  <appender-ref ref="CONSOLE" />
</category>

<!-- Configure logging settings -->
<root class="com.scsu.searchengine.logging.MyCategory">
  <priority value="TRACE" class="com.scsu.searchengine.logging.MyPriority"/>
  <appender-ref ref="ROLL_FILE_1" />
</root>
</log4j:configuration>
40. search.prop
#styles
style.1.desc=bold
style.2.desc=italic
style.3.desc=bigfont
style.4.desc=underline
style.5.desc=meta
style.6.desc=h1
style.7.desc=h2
style.8.desc=h3
style.9.desc=h4
style.10.desc=h5
style.11.desc=h6
#style ranks
style.1.rank=4
style.2.rank=3
style.3.rank=5
style.4.rank=3
style.5.rank=5
style.6.rank=5
style.7.rank=5
style.8.rank=4
style.9.rank=3
style.10.rank=2
style.11.rank=1
#ranks for word properties
capital.rank=1
anchor.rank=4
title.rank=5
meta.rank=5
#Stop words file name
stopwords=C:/Swathi/Assignments/Thesis/src/classes/stoplist.properties
#home page
homepage=http://www.southernct.edu/index.html
#search words
word=southernct,scsu
#pageDatabase
pageDatabaseErrorFile=E:/Swathi/Assignments/Thesis/src/pageDatabase_errors.txt
#pageDatabaseFile
pageDatabaseFile=E:/Swathi/Assignments/Thesis/src/pageDatabase.txt
APPENDIX B

Stop Words

A AS ABLE ABOUT ABOVE ACCORDING ACCORDINGLY ACROSS ACTUALLY AFTER Afterwards AGAIN AGAINST AIN'T ALL ALLOW ALLOWS ALMOST ALONE ALONG ALREADY ALSO ALTHOUGH ALWAYS AM AMONG AMONGST AN AND ANOTHER ANY ANYBODY ANYHOW ANYONE ANYTHING ANYWAYS ANYWHERE APART APPEAR APPRECIATE APPROPRIATE ARE AREN'T AROUND AS ASIDE ASK ASKING ASSOCIATED AT AVAILABLE AWAY AWFULLY BE BECAME BECAUSE BECOME BECOMES BECOMING BEEN BEFORE BEFOREHAND BEHIND BEING BELIEVE BELOW BESIDE BESIDES BEST BETTER BETWEEN BEYOND BOTH BRIEF BUT BY C'MON C'S CAME CAN CAN'T CANNOT CANT CAUSE CAUSES CERTAIN CERTAINLY CHANGES CLEARLY CO COM COME COMES CONCERNING CONSEQUENTLY CONSIDER CONSIDERING CONTAIN CONTAINING CONTAINS CORRESPONDING COULD COULDN'T COURSE CURRENTLY D DEFINITELY DESCRIBED DESPITE DID DIDN'T DIFFERENT DO DOES DOESN'T DOING DON'T DONE DOWN DOWNWARDS DURING E EACH EDU EG EIGHT EITHER ELSE ELSEWHERE ENOUGH ENTIRELY ESPECIALLY ET ETC EVEN EVERY EVERYBODY EVERYONE EVERYTHING EVERYWHERE EX EXACTLY EXAMPLE EXCEPT F FAR FEW FIFTH FIRST FIVE FOLLOWED FOLLOWING FOLLOWS FOR FORMER FORMERLY FORTH FOUR FROM FURTHER FURTHERMORE G GET GETS GETTING GIVEN GIVES GO GOES GOING GONE GOT GOTTEN GREETINGS H HAD HADN'T HAPPENS HARDLY HAS HASN'T HAVE HAVEN'T HAVING HE HE'S HELLO HELP HENCE HER HERE HERE'S HERE AFTER HEREBY HEREIN HEREUPON HERS HERSELF HI HIM HIMSELF HIS HITHER HOPEFULLY HOW HOWBEIT HOWEVER I ID I'LL I'M I'VE IE IF IGNORED IMMEDIATE IN INASMUCH INC INDEED INDICATE INDICATED INDICATES INNER INsofar INSTEAD INTO INWARD IS ISN'T IT IT'D IT'LL IT'S ITS ITSELF J JUST K KEEP KEEPS KEPT KNOW KNOWS KNOWN L LAST LATELY LATER LATTER LATTERLY LEAST LESS LEST LET LET'S LIKE LIKED LITTLE LOOK LOOKING LOOKS LTD M MAINLY MANY MAY MAYBE ME MEAN MEANWHILE MERELY MIGHT MORE MOREOVER MOST MOSTLY MUCH MUST MY MYSELF N NAME NAMELY ND NEAR NEARLY NECESSARY NEED NEEDS NEITHER NEVER NEVERE THELESS NEW NEXT NINE NO NOBODY NON NONE NOONE NOR NORMALY NOT NOTHING NOVEL NOW NOWHERE O OBVIOUSLY OF OFF OFTEN OH OK OKAY OLD ON ONCE ONE ONES ONLY ONTO OR OTHER OTHERWISE OUGHT OUR OURS OURSELVES OUT OUTSIDE OVER OVERALL OWN P PARTICULAR PARTICULARLY PER PERHAPS PLACED PLEASE PLUS POSSIBLE PRESUMABLY PROBABLY PROVIDES Q QUE QUITE QV R RATHER RD RE REALLY REASONABLY REGARDING REGARDLESS REGARDS RELATIVELY RESPECTIVELY RIGHT S SAID SAME SAW SAY SAYING SAYS SECOND SECONDLY SEE SEEING SEEM SEEMED SEEMING SEEMS SEEN SELF SELVES SENSIBLE SENT SERIOUS SERIOUSLY SEVEN SEVERAL SHALL SHE SHOULD SHOULDN'T SINCE SIX SO SOME SOMEBODY SOMEHOW SOMEONE SOMETHING SOMETIME SOMETIMES SOMEWHAT SOMEWHERE SOON SORRY SPECIFIED SPECIFY SPECIFYING STILL SUB

194

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
APPENDIX C

SQL Scripts

--CLEAN UP BEFORE RUNNING SCRIPT
DROP TABLE URL_WORD_XREF CASCADE CONSTRAINTS;
DROP TABLE WORD CASCADE CONSTRAINTS;
DROP TABLE IMAGES CASCADE CONSTRAINTS;
DROP TABLE MMEDIA CASCADE CONSTRAINTS;
DROP TABLE URL_XREF CASCADE CONSTRAINTS;
DROP TABLE URL CASCADE CONSTRAINTS;
DROP TABLE DOC_TYPE CASCADE CONSTRAINTS;
DROP SEQUENCE URL_SEQ;
DROP SEQUENCE MMEDIA_SEQ;
DROP SEQUENCE WORD_SEQ;
DROP PACKAGE search_pkg;

CREATE TABLE DOC_TYPE

    DOC_TYPE_ID    NUMBER(20,0) NOT NULL,
    DOC_DESC      VARCHAR2(50) NOT NULL,

CONSTRAINT DOC_TYPE_PK PRIMARY KEY(DOC_TYPE_ID)
);

CREATE TABLE URL

    URL_ID       NUMBER(20,0) NOT NULL,
    URL_DESC     VARCHAR2(250) NOT NULL unique,
    DOC_TYPE_ID  NUMBER(20,0) NOT NULL,
    RANK         NUMBER(20,10),
    ADD_BY_DATE  DATE default sysdate,

CONSTRAINT URL_PK PRIMARY KEY(URL_ID),
CONSTRAINT URL_FK1 FOREIGN KEY(DOC_TYPE_ID) REFERENCES DOC_TYPE(DOC_TYPE_ID)
);

--CREATE SEQUENCE FOR URL_ID
CREATE SEQUENCE URL_SEQ INCREMENT BY 1 START WITH 1;

CREATE TABLE URL_XREF

    URL_ID       NUMBER(20,0) NOT NULL,
    URL_REF_ID   NUMBER(20,0) NOT NULL,

CONSTRAINT URL_XREF_PK1 PRIMARY KEY(URL_ID,URL_REF_ID),
CONSTRAINT URL_XREF_FK1 FOREIGN KEY(URL_ID) REFERENCES URL(URL_ID),
CONSTRAINT URL_XREF_FK2 FOREIGN KEY(URL_REF_ID) REFERENCES URL(URL_ID)
);
CREATE TABLE MMEDIA ( 
  CLIP_ID NUMBER(20,0) NOT NULL,
  MEDIA_TYPE VARCHAR2(50),
  LABEL VARCHAR2(4000),
  FILE_SIZE NUMBER(20,0),
  URL_ID NUMBER(20,0) NOT NULL,
CONSTRANT MMEDIA_PK1 PRIMARY KEY(CLIP_ID),
CONSTRANT MMEDIA_FK1 FOREIGN KEY(URL_ID) REFERENCES URL(URL_ID) 
);

--CREATE SEQUENCE FOR CLIP_ID
CREATE SEQUENCE MMEDIA_SEQ INCREMENT BY 1 START WITH 1;

--CREATE TRIGGER FOR SEQUENCE KEY ON MMEDIA
CREATE OR REPLACE TRIGGER MMEDIA BEFORE INS ROW
ON MMEDIA
REFERENCING OLD AS OLD NEW AS NEW
FOR EACH ROW
DECLARE
  INTID number;
BEGIN
  select MMEDIA_SEQ.nextval into INTID from dual;
  :new.CLIP_ID := INTID;
END;
/

CREATE TABLE IMAGES ( 
  URL_ID NUMBER(20,0) NOT NULL,
  IMAGE_TYPE VARCHAR2(50),
  LABEL VARCHAR2(4000),
  FILE_SIZE NUMBER(20,0),
  DIMENSIONS VARCHAR2(50),
CONSTRANT IMAGES_PK1 PRIMARY KEY(URL_ID),
CONSTRANT IMAGES_FK1 FOREIGN KEY(URL_ID) REFERENCES URL(URL_ID) 
);

CREATE TABLE WORD ( 
  WORD_ID NUMBER(20,0) NOT NULL,
  WORD_DESC VARCHAR2(2000) NOT NULL unique,
CONSTRANT WORDS_PK PRIMARY KEY(WORD_ID) 
);

--CREATE SEQUENCE FOR WORD_ID
CREATE SEQUENCE WORD_SEQ INCREMENT BY 1 START WITH 1;

CREATE TABLE URL_WORD_XREF (
    URL_ID NUMBER(20,0) NOT NULL,
    WORD_ID NUMBER(20,0) NOT NULL,
    REL_POSITION NUMBER(10,0) NOT NULL,
    CAPITAL NUMBER(1),
    ANCHOR NUMBER(1),
    TITLE NUMBER(1),
    STYLE NUMBER(1),
    RANK NUMBER(10),
    ADD_BY_DATE DATE NOT NULL,
    SUMMARY VARCHAR2(4000),
    TITLE_TEXT VARCHAR2(1000),
    PROCESSED NUMBER(1),
    CONSTRAINT URL_WORD_XREF_PK1 PRIMARY KEY(URL_ID,WORD_ID,REL_POSITION),
    CONSTRAINT URL_WORD_XREF_FK1 FOREIGN KEY(URL_ID) REFERENCES URL(URL_ID),
    CONSTRAINT URL_WORD_XREF_FK2 FOREIGN KEY(WORD_ID) REFERENCES WORD(WORD_ID)
);

--INSERT INTO DOC_TYPE
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(1, 'DOCUMENT');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(2, 'IMAGE');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(3, 'MEDIA');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(4, 'PDF');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(5, 'PS');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(6, 'PHP');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(7, 'WORD');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(8, 'TXT');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(9, 'EXCEL');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(10, 'PPT');
INSERT INTO DOC_TYPE (DOC_TYPE_ID,DOC_DESC) VALUES(11, 'EMAIL');

COMMIT;

ALTER table images add (processed number(1));

--PROCEDURES
CREATE OR REPLACE PACKAGE search_pkg IS
    TYPE t_recset_cur IS REF CURSOR;
    PROCEDURE INSERT_URL (  
        OP_URL_DESC IN URL.URL_DESC%TYPE,  
        198

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
OP_DOC_TYPE IN URL.DOC_TYPE_ID%TYPE,
OP_URL_ID OUT URL.URL_ID%TYPE,
p_crawled_flag IN number DEFAULT 0
);
PROCEDURE insert_url_ref(
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_URL_REF_ID IN URL_XREF.URL_REF_ID%TYPE
);
PROCEDURE insert_word(
    OP_WORD_DESC IN WORD.WORD_DESC%TYPE,
    OP_WORD_ID OUT WORD.WORD_ID%TYPE
);
PROCEDURE insert_url_word_ref(
    OP_WORD_ID IN WORD.WORD_ID%TYPE,
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_REL_POS IN URL_WORD_XREF.REL_POSITION%TYPE,
    OP_CAPITAL IN URL_WORD_XREF.CAPITAL%TYPE,
    OP_ANCHOR IN URL_WORD_XREF.ANCHOR%TYPE,
    OP_TITLE IN URL_WORD_XREF.TITLE%TYPE,
    OP_STYLE IN URL_WORD_XREF.STYLE%TYPE
);
PROCEDURE insert_image(
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_LABEL IN IMAGES.LABEL%TYPE,
    OP_FILE_SIZE IN IMAGES.FILE_SIZE%TYPE,
    OP_DIMENSIONS IN IMAGES.DIMENSIONS%TYPE
);
PROCEDURE insert_media(
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_LABEL IN MMEDIA.LABEL%TYPE,
    OP_FILE_SIZE IN MMEDIA.FILE_SIZE%TYPE,
    OP_MEDIA_TYPE IN MMEDIA.MEDIA_TYPE%TYPE
);
PROCEDURE update_word_ranks(
    op_capital_rank IN number,
    op_anchor_rank IN number,
    op_title_rank IN number
);
PROCEDURE update_ranks(
    OP_URL_ID IN URL.URL_ID%TYPE,
    op_rank IN URL.RANK%TYPE
);
PROCEDURE get_url_links(v_list OUT t_recset_cur);
PROCEDURE get_link_cnt(
    OP_URL_ID IN URL.URL_ID%TYPE,
v_list OUT t_recset_cur

PROCEDURE get_search_results(
  OP_phrase IN varchar2,
  op_doc_type_id IN DOC_TYPE.DOC_TYPE_ID%TYPE,
  v_list OUT t_recset_cur
);

PROCEDURE get_img_search_results(
  OP_phrase IN varchar2,
  op_doc_type_id IN DOC_TYPE.DOC_TYPE_ID%TYPE,
  v_list OUT t_recset_cur
);

PROCEDURE get_related_url(
  OP_url IN URL.URL_DESC%TYPE,
  v_list OUT t_recset_cur
);

PROCEDURE get_summary_list(
  v_list OUT t_recset_cur
);

PROCEDURE get_summary_list1(
  v_list OUT t_recset_cur
);

PROCEDURE update_summary(
  OP_WORD_ID IN URL_WORD_XREF.WORD_ID%TYPE,
  OP_URL_ID IN URL_WORD_XREF.URL_ID%TYPE,
  OP_SUMMARY IN URL_WORD_XREF.SUMMARY%TYPE,
  OP_TITLE IN URL_WORD_XREF.TITLE_TEXT%TYPE
);

PROCEDURE get_url_id(
  OP_URL_ID OUT URL.URL_ID%TYPE,
  OP_URL_DESC IN URL.URL_DESC%TYPE
);

PROCEDURE get_img_urls(
  v_list OUT t_recset_cur
);

PROCEDURE get_media_urls(
  v_list OUT t_recset_cur
);

PROCEDURE get_imgref_urls(
  OP_URL_ID IN URL.URL_ID%TYPE,
  v_list OUT t_recset_cur
);

PROCEDURE update_img_labels(
  OP_url_ID IN IMAGES.url_ID%TYPE,
  OP_label IN IMAGES.LABEL%TYPE

200
PROCEDURE update_media_labels(
    OP_url_ID IN mmmedia.url_ID%TYPE,
    OP_label IN mmmedia.LABEL%TYPE
);

PROCEDURE get_email_results(
    OP_phrase IN varchar2,
    v_list OUT t_recset_cur
);

PROCEDURE get_media_results(
    OP_phrase IN varchar2,
    v_list OUT t_recset_cur
);

PROCEDURE get_image_urls(
    v_list OUT t_recset_cur
);

PROCEDURE update_img_dimensions(
    OP_url_ID IN IMAGES.url_ID%TYPE,
    OP_dim IN IMAGES.DIMENSIONS%TYPE
);

PROCEDURE get_urls_cnt(
    OP_URL_CNT OUT NUMBER
);

PROCEDURE get_reference_id(
    op_ref_url IN URL.URL_DESC%TYPE,
    OP_URL_ID OUT NUMBER
);

PROCEDURE already_visited(
    op_url IN URL.URL_DESC%TYPE,
    crawl_flag OUT NUMBER
);

PROCEDURE getAllWordDocs(
    v_list OUT t_recset_cur
);

PROCEDURE getAllPDFDocs(
    v_list OUT t_recset_cur
);

END search_pkg;
/
SHOW ERRORS;

--CREATE PACKAGE BODY
CREATE OR REPLACE PACKAGE BODY search_pkg AS

PROCEDURE INSERT_URL (}
OP_URL_DESC IN URL.URL_DESC%TYPE,
   OP_DOC_TYPE IN URL.DOC_TYPE_ID%TYPE,
   OP_URL_ID OUT URL.URL_ID%TYPE,
   p_crawled_flag IN number DEFAULT 0
)
IS
   u_id URL.URL_ID%TYPE;
BEGIN
   select url_id into u_id from url
      where upper(url_desc) = upper(op_url_Desc);
   EXCEPTION
      WHEN NO_DATA_FOUND THEN NULL;
   END;
   if u_id IS NULL then
      if p_crawled_flag = 1 then
         select url_seq.nextval into u_id from dual;
         insert into url (URL_ID,URL_DESC,DOC_TYPE_ID, crawled)
            values(u_id, op_url_desc, op_doc_type, 1);
      else
         select url_seq.nextval into u_id from dual;
         insert into url (URL_ID,URL_DESC,DOC_TYPE_ID)
            values(u_id, op_url_desc, op_doc_type);
      end if;
   else
      if p_crawled_flag = 1 then
         update url set crawled = 1 where url_id = u_id;
      end if;
   end if;
   OP_URL_ID := u_id;
end INSERT_URL;

PROCEDURE insert_url_ref(
   OP_URL_ID IN URL.URL_ID%TYPE,
   OP_URL_REF_ID IN URL_XREF.URL_REF_ID%TYPE
)
IS
   begin
      insert into URL_XREF(url_id, url_ref_id)
         values (OP_URL_ID, OP_URL_REF_ID);
      EXCEPTION WHEN DUP_VAL_ON_INDEX THEN null;
   end insert_url_ref;

PROCEDURE insert_word(
   OP_WORD_DESC IN WORD.WORD_DESC%TYPE,
   OP_WORD_ID OUT WORD.WORD_ID%TYPE
)
Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
and word_id = op_word_id
and rel_position = op_rel_pos;
ELSE
insert into URL_WORD_XREF(URL_ID,WORD_ID,REL_POSITION,
CAPITAL,ANCHOR,TITLE,STYLE,ADD_BY_DATE)
values(OP_URL_ID,OP_WORD_ID,OP_REL_POS,
OP_CAPITAL,OP_ANCHOR,OP_TITLE,OP_STYLE,SYSDATE);
END IF;
end insert_url_word_ref;

PROCEDURE insert_image ( 
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_LABEL IN IMAGES.LABEL%TYPE,
    OP_FILE_SIZE IN IMAGES.FILE_SIZE%TYPE,
    OP_DIMENSIONS IN IMAGES.DIMENSIONS%TYPE
) IS
    u_id URL.URL_ID%TYPE;
BEGIN
    BEGIN
        SELECT url_id INTO u_id FROM images
        WHERE url_id = op_url_id;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN NULL;
    END;
    IF u_id IS NULL THEN
        INSERT INTO IMAGES(URL_ID,LABEL,FILE_SIZE,DIMENSIONS)
        VALUES(OP_URL_ID,OP_LABEL,OP_FILE_SIZE,OP_DIMENSIONS);
    END IF;
end insert_image;

PROCEDURE insert_media ( 
    OP_URL_ID IN URL.URL_ID%TYPE,
    OP_LABEL IN MMEDIA.LABEL%TYPE,
    OP_FILE_SIZE IN MMEDIA.FILE_SIZE%TYPE,
    OP_MEDIA_TYPE IN MMEDIA.MEDIA_TYPE%TYPE
) IS
    BEGIN
        INSERT INTO MMEDIA(MEDIA_TYPE,LABEL,FILE_SIZE,URL_ID)
        VALUES(OP_MEDIA_TYPE,OP_LABEL,OP_FILE_SIZE,OP_URL_ID);
    end insert_media;

PROCEDURE update_word_ranks ( 
    op_capital_rank IN number,
    op_anchor_rank IN number,
op_title_rank IN number

) IS
CURSOR update_rows_cur IS
SELECT * FROM url_word_xref WHERE add_by_date > (sysdate - 1);
begin
FOR update_rows_rec in update_rows_cur LOOP
BEGIN
    update url_word_xref
    set rank = (update_rows_rec.capital * op_capital_rank) +
        (update_rows_rec.anchor * op_anchor_rank) +
        (update_rows_rec.title * op_title_rank) +
        (update_rows_rec.style)
    where url_id = update_rows_rec.url_id
    and word_id = update_rows_rec.word_id
    and rel_position = update_rows_rec.rel_position;
commit;
END;
END LOOP;
end update_word_ranks;

PROCEDURE get_url_links (v_list OUT t_recset_cur)
IS
begin
    open v_list for
        select u.xref.url_id, u.xref.url_ref_id
        from url_xref u.xref,
        url u
        where u.xref.url_id = u.url_id
        and u.DOC_TYPE_ID = 1
        order by u.xref.url_id;
end get_url_links;

PROCEDURE update_ranks (  
    OP_URL_ID IN URL.URL_ID%TYPE,
    v_list OUT t_recset_cur
) is
begin
    update url set rank=op_rank where url_id=op_url_id;
end update_ranks;

PROCEDURE get_link_cnt(  
    OP_URL_ID IN URL.URL_ID%TYPE,
    v_list OUT t_recset_cur
) IS
begin
205
open v_list for
    select uxfref.url_id
    from url_xref uxfref,
        url u
    where uxfref.url_ref_id = op_url_id
    and uxfref.url_id = u.url_id
    and u.DOCTYPE_ID = 1
    order by uxfref.url_id;
end get_link_cnt;

PROCEDURE get_search_results(
    OP_phrase      IN  varchar2,
    op_doc_type_id IN  DOC_TYPE.DOC_TYPE_ID%TYPE,
    v_list        OUT  t_recset_cur
) is
/
    begin
        open v_list for
            select my2.*,
                u.rank
            from url u,
                ( select my.url_id, u.url_desc, min(my.rel_position) pos, my.summary,
                    my.title_text
                    from url u,
                    ( select w.word_id, w.word_desc, uwx.rank,
                        uwx.url_id, uwx.rel_position, uwx.summary, uwx.title_text
                        from word w, url_word_xref uwx
                        where uwx.word_id = w.word_id
                        and upper(w.word_desc) like upper(op_phrase)
                        group by w.word_id, w.word_desc, uwx.rank, uwx.url_id,
                            uwx.rel_position, uwx.summary, uwx.title_text
                    ) my
                    where my.url_id = u.url_id
                    group by my.url_id, u.url_desc, my.summary, my.title_text
                ) my2
            where my2.url_id = u.url_id
            and u.doc_type_id in (1,4,7)
            --and my2.pos < 100
            order by u.rank desc,
                my2.pos asc;
        end get_search_results;

PROCEDURE get_img_search_results(
    OP_phrase      IN  varchar2,
op_doc_type_id IN DOC_TYPE.DOC_TYPE_ID%TYPE,
    v_list OUT t_recset_cur
)
    is
begin
    open v_list for
    select distinct i.label, i.dimensions, i.file_size, i.image_type, u.url_desc
          from images i, url u
          where i.url_id = u.url_id
          and upper(i.label) like upper(op_phrase);
end get_img_search_results;

PROCEDURE get_related_url(
    OP_url IN URL.URL_DESC%TYPE,
    v_list OUT t_recset_cur
)
    is
begin
    open v_list for
    select u.url_desc
          from url u,
          url_xref ux
          where u.url_id = ux.url_ref_id
          and ux.url_id in ( select url_id from url where url_desc like op_url);
    exception
        when no_data_found then null;
end get_related_url;

PROCEDURE get_summary_list(
    v_list OUT t_recset_cur
)
    is
begin
    open v_list for
    select my.url_id, my.word_id, u.url_desc, w.word_desc
          from url u, word w,
          ( select distinct url_id, word_id
            from url_word_xref
            where add_by_date > sysdate - 1
            and url_id in ( select url_id
                             from url
                             where doc_type_id = 1
                             )
            and processed = 0
            order by url_id
          ) my
          ordered by u.url_id
          )
          my
          )
          .
          )
          .
          )
where u.url_id = my.url_id
and w.word_id = my.word_id
order by my.url_id;
end get_summary_list;

PROCEDURE get_summary_list1
(v_list OUT t_recset_cur)
) is
begin
open v_list for
select my.url_id, my.word_id, u.url_desc, w.word_desc
from url u, word w,
(
select distinct url_id, word_id
from url_word_xref
where add_by_date > sysdate - 1
and url_id in (select url_id
from url
where doc_type_id = 1
and processed = 0
and rownum = 1)
) my
where u.url_id = my.url_id
and w.word_id = my.word_id;
end get_summary_list1;

PROCEDURE update_summary
(OP_WORD_ID IN URL_WORD_XREF.WORD_ID%TYPE,
OP_URL_ID IN URL_WORD_XREF.URL_ID%TYPE,
OP_SUMMARY IN URL_WORD_XREF.SUMMARY%TYPE,
OP_TITLE IN URL_WORD_XREF.TITLE_TEXT%TYPE)
) is
begin
update url_word_xref
set summary = op_summary,
TITLE_TEXT = op_title,
processed = 1
where url_id = op_url_id
and word_id = op_word_id;
end update_summary;

PROCEDURE get_url_id
(OP_URL_ID OUT URL.URL_ID%TYPE,
OP_URL_DESC IN URL.URL_DESC%TYPE

    ) is
begin
    select url_id into op_url_id from url
    where upper(url_desc) = upper(op_url_desc);
end get_url_id;

PROCEDURE get_img_urls(
    v_list OUT t_recset_cur
) is
begin
    open v_list for
    select label, url_id
    from images
    where processed = 0;
end get_img_urls;

PROCEDURE get_media_urls(
    v_list OUT t_recset_cur
) is
begin
    open v_list for
    select label, url_id
    from mmedia;
end get_media_urls;

PROCEDURE get_imgref_urls(
    OP_URL_ID IN URL.URL_ID%TYPE,
    v_list OUT t_recset_cur
) is
begin
    open v_list for
    select url_desc
    from url
    where url_id in ( select url_ref_id
                        from url_xref
                        where url_id = op_url_id
                      );
end get_imgref_urls;

PROCEDURE update_img_labels(
    OP_url_ID IN IMAGES.url_ID%TYPE,
    OP_label IN IMAGES.LABEL%TYPE
) is begin
update images
  set label = op_label,
  processed = 1
  where url_id = op_url_id;
end update_img_labels;

PROCEDURE update_media_labels(
  OP_url_ID IN mmedia.url_ID%TYPE,
  OP_label IN mmedia.LABEL%TYPE
) is begin
  update mmedia
  set label = op_label
  where url_id = op_url_id;
end update_media_labels;

PROCEDURE get_email_results(
  OP_phrase IN varchar2,
  v_list OUT t_recset_cur
) is begin
  open v_list for
    select distinct u.url_desc, myl.word_desc
    from url u, url_word_Xref uwx,
    ( select word_id, word_desc
      from word
      where upper(word_desc) like upper(op_phrase)
    ) myl
    where myl.word_id = uwx.word_id
    and uwx.url_id = u.url_id
    and u.doc_type_id = 11;
end get_email_results;

PROCEDURE get_media_results(
  OP_phrase IN varchar2,
  v_list OUT t_recset_cur
) is begin
  open v_list for
    select distinct m.label,
               u.url_desc
    from mmedia m,
         url u
    where upper(m.label) like upper(op_phrase)
    and m.url_id = u.url_id;
end get_media_results;

PROCEDURE get_image_urls(
    v_list OUT t_recset_cur
) is
begin
    open v_list for
    select i.url_id, u.url_desc
    from images i, url u
    where i.url_id = u.url_id;
end get_image_urls;

PROCEDURE update_img_dimensions(
    OP_url_ID IN IMAGES.url_ID%TYPE,
    OP_dim IN IMAGES.DIMENSIONS%TYPE
) is
begin
    update images
    set DIMENSIONS = op_dim
    where url_id = op_url_id;
end update_img_dimensions;

PROCEDURE get_urls_cnt(
    OP_URL_CNT OUT NUMBER
) is
begin
    select count(*) into op_url_cnt
    from url where doc_type_id = 1;
end get_urls_cnt;

PROCEDURE get_reference_id(
    op_ref_url IN URL.URL_DESC%TYPE,
    OP_URL_ID OUT NUMBER
) is
begin
    BEGIN
        select url_id into op_url_id
        from url where url_desc = op_ref_url;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN NULL;
    END;
end get_reference_id;

PROCEDURE already_visited(
    op_url IN URL.URL_DESC%TYPE,
crawl_flag OUT NUMBER

) is
begin
BEGIN
  select crawled into crawl_flag
  from url where url_desc = op_url;
EXCEPTION
  WHEN NO_DATA_FOUND THEN CRAWL_FLAG := 0;
  WHEN OTHERS THEN CRAWL_FLAG := 0;
END;
end already_visited;

PROCEDURE getAllWordDocs(
  v_list OUT t_recset_cur
) is
begin
  open v_list for
    select url_id, url_desc
    from url
    where doc_type_id = 7;
end getAllWordDocs;

PROCEDURE getAllPDFDocs(
  v_list OUT t_recset_cur
) is
begin
  open v_list for
    select url_id, url_desc
    from url
    where doc_type_id = 4;
end getAllPDFDocs;
END search_pkg;
/
SHOW ERRORS;
APPENDIX D

User Manual

Relational Search Engine. (RSE) is a java application that crawls, indexes, ranks and builds a database for searching. With RSE, any web site can be crawled without making changes to the java source code.

RSE Features. RSE software has the following features:
1) Crawls any specified web site.
2) Parses the crawled pages and index the words found in the web documents.
3) Finds and indexes images used in the web site.
4) Finds and indexes the multi media files used in the web site.
5) Calculates word ranks and page ranks to prioritize the search results.
6) Portable search software that returns search results based on the keyword or phrase.
7) Database connection pooling to manage and re-use the database connections.
8) Caching of search results for faster response time in searching.

System Requirements. RSE software is successfully tested on Windows 2000/NT, RedHat, Unix and Solaris operating systems and Oracle 8i/9i database.

Database. With RSE, any database can be used without changing the java source code. The file “ConnectionManager.properties” holds the properties for a database connection. The following changes to the properties file ensure the configuration of the database for RSE.

1) JDBC Driver: Specify the driver that should be used by java to interface with the database.
2) JDBC URL: The URL used to connect to the database consists of:
   a. Address of the machine on which database is installed
   b. Port number in which the database is accessible.
   c. Username to connect to the database
   d. Password to connect to the database.
   e. The database system identification (SID).
3) “scsu.database.JDBCPool.minSize” property consists of the minimum number of connections that are allowed to the database.
4) “scsu.database.JDBCPool.maxSize” property represents the maximum database connections maintained by the connection pool at any given instance.
5) “scsu.database.JDBCPool.expiration” property represents the maximum time a database connection can be active. After this time, the connection expires and a new connection has to be made to the database.
6) “scsu.database.JDBCPool.frequency” property represents the frequency used to check for expired connections in the connection pool.

The script in APPENDIX D represents all the tables and SQL Packages used in RSE. This script is successfully tested on Oracle 8i and Oracle 9i. Relevant changes need to be made to the SQL script depending on the Database being used.
Crawling. com.scsu.searchengine.controller.WebExplorer.java is the program to run inorder to start crawling process. The Crawling can be limited to one website or a set of websites. The configuration for crawling is done in “search.props” file. The property named “homepage” holds the website address that should be crawled. If there are more than one website, the list should be separated by comma. For example, to crawl two websites the “homepage” property is configured as “homepage = http://www.southernct.edu,http://www.yahoo.com”.

Ranking. After the crawling is completed, procedures to calculate the ranks should be run. The following classes needs to be run in order to calculate word ranks and page ranks.

1. com.scsu.searchengine.util.WordRanks: This class calculates word ranks based on various factors detailed in Chapter 6.
2. com.scsu.searchengine.util.PageRanks: This class calculates page ranks as detailed in Chapter 6.

Title and Summary. Title and summary for webpages and images found should be updated after ranking is done. The following classes should be run in order to update summary and title for webpages and images.

1. com.scsu.searchengine.util.Summary: This class retrieves the list of webpages from the RSE database and extracts the summary and title for the webpage.
2. com.scsu.searchengine.util.ImageLabels: This class retrieves the list of images stored in the RSE database and updates appropriate description for the image and file size of the image.
APPENDIX E

Results

1) Image search results for single word query “Seyed”
   1. Results returned by RSE

![RSE image search results for “Seyed”]

*Figure 23. RSE image search results for “Seyed”*
2. Results returned by Google

Figure 24. Google image search results for “Seyed”
2) Document search results for “Extended Learning”

1. Results returned by RSE

![Image showing RSE document search results for “Extended Learning”]

*Figure 25. RSE document search results for “Extended Learning”*
2. Results returned by Google

---

Figure 26. Google document search results for “Extended Learning”
3) Document search results for “International Studies”
   1. Results returned by RSE

   ![Screenshot of search results for International Studies](image)

   **Figure 27.** RSE document search results for “International Studies”

   219
2. Results returned by Google

![Google search results for International Studies](image)

**Figure 28.** Google document search results for “International Studies”
4) Image search results for “Computer Science”
   1. Results returned by RSE

   ![Image search results for “Computer Science”]

   **Figure 29.** RSE image search results for “Computer Science”
2. Results returned by Google

![Google Image Search Results for "Computer Science"](image)

*Figure 30. Google image search results for “Computer Science”*
Figure 31. RSE document search results for “Football match schedule”
2. Results returned by Google

Figure 32. Google document search results for “Football match schedule”
6) Document search results for “Commencement Weekend Activities”
   1. Results returned by RSE

**Figure 33.** RSE document search results for “Commencement Weekend Activities”

225
2. Results returned by Google

Figure 34. Google document search results for “Commencement Weekend Activities”
7) Media search results for “Public Health”
   1. Results returned by RSE

Figure 35. RSE media search results for “Public Health”
8) Email search results for “Seyed”
   1. Results returned by RSE

   ![RSE e-mail search results for “Seyed”](image1)

   **Figure 36.** RSE e-mail search results for “Seyed”

   2. Results returned from southerncst.edu’s email search engine

   ![southerncst.edu’s e-mail search results for “Seyed”](image2)

   **Figure 37.** Southerncst.edu’s e-mail search results for “Seyed”

228
9) Email search results for “graduate office”
   1. Results returned by RSE

   Figure 38. RSE e-mail search results for “graduate office”

   2. Results returned from southernc.edu’s email search engine

   Figure 39. Southernc.edu’s e-mail search results for “graduate office”
REFERENCES


Date, C. J. (February 2001). *There's only one relational model*. Retrieved December 2, 2003, from [http://www.pgro.uk7.net/cjd6a.htm](http://www.pgro.uk7.net/cjd6a.htm)


Java servlet technology (n. d.). Retrieved December 2, 2003, from
http://java.sun.com/products/servlet/


JumpStation. (n. d.). Retrieved December 2, 2003, from
http://www.theheartofgold.org/jumpstation/


Brisbane, Australia.


Proc. of the 10th Intl. WWW Conf.

http://www.ncsa.uiuc.edu/Divisions/PublicAffairs/MosaicHistory/history.html

http://www.northernwebs.com/set/alta_vista.html

http://www.northernwebs.com/set/

http://dir.yahoo.com


234
World wide web and CERN. (n. d.) Retrieved December 1, 2003, from

World wide web worm. (n.d.) Retrieved June 16, 2003 from


from http://www.cs.ualberta.ca/~yiqiao/courses/crawl.ppt


235