

# LSE: Local Search Engine 1.05

Karel Kubat, e-tunity

2002

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	Features of LSE . . . . .	2
1.2	Prerequisites . . . . .	3
<b>2</b>	<b>Installation and Configuration</b>	<b>4</b>
2.1	Creation of the indexing database . . . . .	4
2.2	Configuring the indexer . . . . .	5
2.3	Indexing the files . . . . .	6
2.4	Configuring the webservice . . . . .	6
<b>3</b>	<b>Tips, Tricks and Caveats</b>	<b>9</b>
<b>4</b>	<b>Sources</b>	<b>11</b>
4.1	LSE-index . . . . .	11
4.2	LSE.php . . . . .	18
4.3	search.php . . . . .	24

# Chapter 1

## Introduction

LSE is a small set of applications which together form a *local search engine* for websites. The idea of LSE is that files residing on a webserver are periodically scanned, and that an index of the words in these files is stored in a database. The index can then be queried via the web, when a visitor of the site wishes to search for the documents that contain certain words.

LSE is a *local* search engine in the sense that files on a webserver are scanned via the file system. Other search engines (e.g., `htdig`), read pages via the webserver itself; i.e., they *spider* the pages. Both approaches have advantages and disadvantages. The advantage of LSE is that the pages can be scanned without a working webserver, and without 'knowing' or storing the full URL to the site. A further advantage of LSE is that *all* local files under a given directory are scanned; not only the files that are linked together via `href`'s.

However, the biggest advantage of LSE would be its flexibility and configurability.

### 1.1 Features of LSE

The features of LSE are:

- LSE consists of several small tools. Each of these tools is customizable to suit your specific needs.
- LSE is 'open': its programs and data structures are clear and concise, you can see what's going on and you can build your own extra functionality when you need it.
- LSE uses standard Unix tools to do the job. If you have a Unix webserver, LSE is usable. If you have Windows or NT, LSE is still usable once you obtain the ports of the tools.
- LSE supports multiple index databases, and you can make as many search forms as you like to query the indexes. On one site, you might have several search engines and indexes to serve different roles of your visitors.

## 1.2 Prerequisites

If you want to run LSE, you will need:

- A webserver that is PHP-compliant. Apache is suggested.
- Perl, the script interpreter.
- MySQL, a lightweight and fast database.

Furthermore, the following tools or programs are **strongly** suggested:

- `lynx`, a text-oriented browser, which is used to read local HTML documents.
- `ps2ascii`, a tool to convert Postscript or PDF documents into text. The indexer uses this tool to see what's 'inside' Postscript or PDF documents.

## Chapter 2

# Installation and Configuration

The installation of LSE consists of several steps:

- Creation of the indexing database;
- Configuring the indexer;
- Indexing the files;
- Configuring the webservice.

### 2.1 Creation of the indexing database

Before anything else, you must choose a database name for the index, and create it. In the following description we're assuming a database named `intranet`, though you're free to choose your own.

The database will have three tables:

- `pages`, where URL's of the available documents are stored,
- `words`, where words of your site are kept,
- `hits`, stating which word occurs in which document and how many times.

The database is created using the following commands, usually as `root` of the system:

```
1  system> mysql
2  mysql> create database intranet;
3  mysql> grant all privileges on intranet.* to user@localhost;
4  mysql> quit;
```

Instead of `user`, a right username must be supplied of the user who will run the indexer and the webservices.

The tables in the database are created using the following MySQL statements:

```
# Table structure for table 'hits'
CREATE TABLE hits (
  wordid int(11) NOT NULL default '0',
  pageid int(11) NOT NULL default '0',
  hits int(11) default NULL,
  PRIMARY KEY (wordid,pageid),
  KEY i (wordid),
  KEY j (pageid)
) TYPE=MyISAM;
# Table structure for table 'pages'
CREATE TABLE pages (
  id int(11) NOT NULL auto_increment,
  url char(255) NOT NULL default '',
  stamp int(11) default NULL,
  PRIMARY KEY (id,url)
) TYPE=MyISAM;
# Table structure for table 'words'
CREATE TABLE words (
  id int(11) NOT NULL auto_increment,
  word char(255) NOT NULL default '',
  code int(11) default NULL,
  PRIMARY KEY (id,word),
  UNIQUE KEY i (word)
) TYPE=MyISAM;
```

## 2.2 Configuring the indexer

The indexer is a Perl program named `LSE-index`.

As for the configuration of the indexer, the following is relevant. At the top of the script, the *handlers* for different file types are defined. Here's an excerpt:

```
1 # Handlers for file types.
2 my %handlers = (
3   ".htm"    =>    "lynx -dump file://localhost/FULLPATH",
4   ".html"   =>    "lynx -dump file://localhost/FULLPATH",
5   ".txt"    =>    "cat FULLPATH",
6   ".pdf"    =>    "ps2ascii FULLPATH",
7 );
```

This code snippet defines `lynx` as the handler for `*.htm` and `*.html`, `cat` as the handler for `*.txt`, and `ps2ascii` as the handler for `*.pdf`. Handlers for other file types can be added here by following the example. The right part of the configuration is the command that `LSE-index` will run in order to get a list of 'words' in the file. The following rules apply:

- The string `FULLPATH` in the command will be replaced by the full path-name of the file in question.
- The command, once started, must send its output to `stdout`.

The indexer is smart enough to "know" what a word is: the input stream coming from the handler is first split into chunks of letters and/or digits. Then, anything that has one or more letters in it, is considered a word. E.g., `this` is a word, `this2` too, but `1234` is not. A chunk like `a_12` isn't a word either, because it is seen by the indexer as two sequences, `a` and `12`.

## 2.3 Indexing the files

The collection of web documents can be indexed by starting `LSE-index`. The script expects at least three arguments:

- The name of the index database (`intranet` in the above example);
- The directory to start the indexing in. `LSE-index` will process all files in this directory, and will recurse into underlying directories;
- The relative URL to reach this directory. This is best explained in an example. At our intranet site, the document root is `/home/intranet/htdocs/`. Under this directory, `/documents` is located, the 'home' of the library. When the indexer prepares the information for the library, the invocation is:

```
1 system> LSE-index intranet /home/intranet/htdocs/documentation /documentation
```

Depending on your version of `LSE-index`, some flags may be supported. Start `LSE-index` without arguments to see the options. The full list of options is not further described in this section.

The first indexing process may take quite some time. However, the indexer is 'smart' enough to skip already indexed files during a second (or subsequent) run. Files will only be indexed if they're more recent than any existing index information.

## 2.4 Configuring the webservice

The access (API) to the search functionality is provided by a PHP class `LSE`, available after the inclusion of the file `LSE.php`.

The class `LSE` provides the following functionality:

- `$l = new LSE($db, $user, $words);` This instantiates an object of the class `LSE` and defines a search for `$words`, a space-separated list of items to search for. The first two arguments are the index database and the user who is authorized to connect to it. Optional other arguments are search flags, a result limit, and a log file.

The search flags are bitwise or-ed values of `LSE_OR`, `LSE_AND`, `LSE_VERBOSE`, `LSE_MATCHSTART` and `LSE_MATCHMID`.

`LSE_AND` and `LSE_OR` define whether the search should include all stated items, or any of the stated items. The default for the flags is `LSE_OR`.

LSE\_MATCHSTART, when present, specifies that search terms will match words when the beginning of the term matches the beginning of a word. E.g., the search term `lin` will match `linux`. When LSE\_MATCHMID is present, then the search terms will match words when any part of the word matches; e.g, the term `in` will match `linux`, `whithin` and so on. When neither LSE\_MATCHSTART nor LSE\_MATCHMID are present, then search terms will only match the words when they are exactly the same.

Exact matches are of course the fastest ones. However, for most users, LSE\_MATCHSTART will be a 'logical' choice and is therefore the default method. LSE\_MATCHMID is the most 'expensive' matching method.

The result limit, when given, specifies the maximum number of results that LSE should process. The default is 50.

The final argument of the constructor is a log file. When given, LSE will happily log whatever is going on. (The log file is of course for debugging purposes.)

- `list($url, $n, $title) = $l->nextpage($root);` This retrieves subsequent results of the search. The return value is an array of:
  - `$url`, the relative URL pointing to a page,
  - `$n`, the number of hits in that page,
  - `$title`, the page title (or an empty string).

The argument `$root` must reflect a directory, which will lead to a full filename when `$url` is appended to it. LSE needs this setting to access the local page on the server, so that the title may be determined.

The pages with the most hits are returned first; these pages reflect the 'best match'. The title is a string stating the HTML `<title>`. Pages lacking a title are obviously returned without this information.

- `$firstpart = $l->contentsof($filename);` The function `contentsof()` receives a file name as argument, and returns the first part of the file, without HTML tags. Normally this will be the first real text in an HTML document; i.e., without the `<head>`, `<title>` and so on.

This function accepts an optional second argument, `$maxlen`, the maximum length in characters of the first part of the file. The default is 200 characters.

The filename must point to an existing file in the filesystem of the web-server. Normally you'll construct this filename from the document root and the `$url` that was returned by the last call to `nextpage()`.

- `$l->limitexceeded()` returns `true` when `$l->nextpage()` returned more results than the stated limit. This boolean flag can be inspected to suggest that the user should limit the search by specifying more keywords, etc..
- `$hits = $l->foundhits();` This function returns the number of hits that were returned so far via `nextpage()`.
- `list($npages, $maxstamp, $words) = $l->info();` The function `$l->info()` returns an array with three numeric values: the number of pages in the index, the most recent index entry as a Unix timestamp, and the number of words in the index. This function can be used to obtain information about the state of the index database.

- `$printable = $l->stampotime($minstamp)`; This is an auxiliary function to convert a Unix timestamp into printable (and readable) format.

Using this API, PHP pages can be simply constructed to start searches, and to display results. As an example, `search.php` is provided in this document.

A typical PHP query that produces output has the following stanza:

```
1
2 # First, we instantiate an LSE search object.
3 $l = new LSE ("mydatabase",
4             "username",
5             "whatever to search for"
6             # , [optionally supply LSE_* flags]
7             # , [optionally supply the max. nr. of hits]
8             );
9
10 # Now iterate through the results.
11 while (list ($url, $n, $title) =
12         $l->nextpage ("/usr/local/apache/htdocs")) {
13
14     # Show the results.
15     echo (" $n hits found in page
16         <a href=\"$url\">$url</a>
17         <br>\n");
18
19     # Display the title if we have it.
20     if ($title != "")
21         echo ("Page title: $title<br>\n");
22
23     # Display the first part of the file if you like.
24     echo ("Page contents: <i>" .
25         $l->contentsof ("/usr/local/apache/htdocs", $url) .
26         "</i><br>\n");
27 }
28
29 # See if we have matches, or maybe too many.
30 if ($l->limitexceeded())
31     echo ("There are more matches. Try more specific search terms!\n");
32 else if ($l->foundhits() == 0)
33     echo ("No matches found. Try other search terms!\n");
```

## Chapter 3

# Tips, Tricks and Caveats

This chapter points out some practical issues when using LSE.

**Re-initializing the index database:** If you want to re-initialize the indexer, simply remove all entries from the MySQL index:

```
1  system> mysql lse
2  mysql> delete from words;
3  mysql> delete from pages;
4  mysql> quit;
```

**Breaking up too long index runs:** If your index runs take way too long, then you might be better off by breaking the runs into smaller parts. E.g., say that you have the documentroot under `/usr/local/apache/htdocs/`, where two big directories exist `big1/` and `big2/`.

In that case, you can issue two indexing statements using:

```
1  LSE-index dbname /usr/local/apache/htdocs/big1 /big1
2  LSE-index dbname /usr/local/apache/htdocs/big2 /big2
```

Using this approach you could schedule runs with `cron` for odd and even days, or whichever way you like.

**Recovering from a broken LSE-index run:** If `LSE-index` should crash for some reason, then the risk exists that the last indexed file was only partially processed. If you think that this is the case, then follow these steps:

1. In MySQL, determine the last indexed file. This is the entry in `pages` with the highest timestamp:  

```
select max(stamp) from pages;
```
2. Using the result, obtain the ID of the page:  

```
select id from pages where stamp = maxvalue
```
3. Using the ID, reset the stamp to zero so that a next run will re-index the file:  

```
update pages set stamp = 0 where id = id
```

**Handling non-words and non-pages:** You might want the search engine to avoid finding certain non-words or non-pages. This is best done by removing the words or pages from the index database.

E.g., consider the following hypothetical situation. You have a directory `/usr/local/apache/htdocs/secure/` which is protected by an authentication method of the webserver. The search engine should not return hits pointing into this directory tree, because the search results of any non-authenticated visitor would be polluted by unreachable document links. In this case, you would remove the pages in MySQL:

```
delete from pages where uri like '/secure%';
```

Or consider the following. You want to take out 'non-words' from the index dictionary, because these words are meaningless in searches:

```
1  mysql> # Step 1: determine the word ID for 'the'
2  mysql> select id from words where word = 'the';
3  +-----+
4  | id   |
5  +-----+
6  | 284 |
7  +-----+
8  1 row in set (0.34 sec)
9  mysql> # Step 2: kill the entry in the words list
10 mysql> delete from words where id = 284;
11 mysql> # Step 3: kill the entries in the pages hitlist
12 mysql> delete from hits where wordid = 284;
```

**Note that** as of version 1.02, the script `LSE-index` supports a flag `-n`, where non-words can be specified on the commandline.

**Shortcut search forms:** Imagine that you have a searching form on your site, similar to the one shown in section 4.3. To create a 'shortcut' form in e.g. the banner of your site, add a mini-form that supplies most search values. When the users try this form, they will automatically go to the larger form that shows the first results, and provides tunable searching:

```
1  <form method="post" action="search.php">
2    <input type="text"   size="15"      name="words"><br>
3    <input type="hidden" name="logical"  value="or">
4    <input type="hidden" name="matchmode" value="matchexact">
5    <input type="submit" value="search">
6  </form>
```

Please see section 4.3 to understand why the form variables have the shown names and what their meanings are.

**Avoiding too high system load during indexing:** To avoid too high system load during indexing, try the flag `-m` of the indexer `LSE-index`. The working of the flag is quite rudimentary; `LSE-index` will simply wait until the CPU consumption of all `mysqld` processes drops below a given percentage. However, using this flag, you can start really long indexing jobs and let them run.

# Chapter 4

## Sources

This chapter lists the sources of LSE. To obtain these sources, please contact us at [info@e-tunity.com](mailto:info@e-tunity.com).

### 4.1 LSE-index

```
1  #!/usr/bin/perl
2
3  use strict;
4  use Cwd;
5  use Getopt::Std;
6
7  # LSE-index:
8  # Local Site Search Engine indexer
9
10 # Config section
11 # =====
12
13 # Handlers for file types.
14 my %handlers = (
15     ".htm"    =>    "lynx -dump file://localhost/FULLPATH",
16     ".html"   =>    "lynx -dump file://localhost/FULLPATH",
17     ".txt"    =>    "cat FULLPATH",
18     ".pdf"    =>    "ps2ascii FULLPATH",
19 );
20
21 # Globals
22 # =====
23 my %opts;           # program options
24 my $dbname;        # database name
25
26 # Queries section
27 # =====
28
29 # Start a write-pipe for queries.
```

```

30 my $started = 0;
31 sub qry_open ($) {
32     my $qry = shift;
33
34     if (! $started) {
35         $started++;
36         open (OQRY, "|mysql $dbname") or die ("cannot start open qry");
37         my $oldh = select (OQRY);
38         $| = 1;
39         select ($oldh);
40
41         print OQRY ("set autocommit=1;\n");
42     }
43     print OQRY ("$qry;\n");
44 }
45
46 # Finalize an open query.
47 sub qry_finalize () {
48     close (OQRY);
49     $started = 0;
50 }
51
52 # Run a single read-query. Return the results.
53 sub qry_close ($) {
54     my $qry = shift;
55     my @ret;
56
57     open (QR, "echo \"\$qry\" | mysql $dbname |")
58         or die ("cannot run qry $qry\n");
59
60     my $nlines = 0;
61     LINE: while (<QR>) {
62         s/\n//;
63         next LINE unless ($nlines++);
64         push (@ret, split ("\t"));
65     }
66     close (QR);
67     return (@ret);
68 }
69
70 # Word handling section
71 # =====
72
73 my %wordinfo;
74
75 my $linelen = 0;
76 sub words_reset ($) {
77     my $pageid = shift;
78
79     %wordinfo = ();
80     qry_open ("delete from hits where pageid = $pageid");
81
82     $linelen = 0;
83 }

```

```

84
85 sub words_insert ($ $) {
86     my $word = shift;
87     my $pageid = shift;
88
89     # Check if this is a WORD or a NONWORD (specified by -n...)
90     my $skip = 0;
91     if ($opts{'n'}) {
92         my @nonwords = split(",", $opts{'n'});
93         my $nw;
94
95         foreach $nw (@nonwords) {
96             $skip++ if ($word eq $nw);
97         }
98     }
99
100    if (!$skip) {
101        my $key = "$word:$pageid";
102        $wordinfo{$key}++;
103    }
104
105    if ($opts{'w'}) {
106        if ($linelen + length($word) > 75) {
107            print ("\n");
108            $linelen = 0;
109        }
110
111        if ($skip) {
112            print ("-");
113        } else {
114            print ("+");
115        }
116        print (" $word ");
117        $linelen += length($word);
118        $linelen += 2;
119    }
120 }
121
122 sub words_flush () {
123     my $key;
124
125     print ("Inserting words and hits into database.\n");
126
127     # Check CPU usage of MySQL, if indicated by -m PERCENTAGE.
128     if ($opts{'m'}) {
129         my $toohigh = 1;
130
131         while ($toohigh) {
132             $toohigh = 0;
133             print ("Checking MySQL usage.\n");
134             open (TP, "top -b -n1 |") or die ("Cannot start top\n");
135             TPLINE: while (<TP>) {
136                 s/\n//;
137                 my ($pid, $usr, $pri, $ni, $sz, $rss, $share, $stat,
```

```

138             $cpu, $mem, $time, $cmd) = split (" ");
139             if ($cmd eq "mysqld") {
140                 print ("MySQL CPU consumption: $cpu\n");
141                 if ($cpu > $opts{'m'}) {
142                     print ("Consumption too high, waiting..\n");
143                     $toohigh = 1;
144                     last TPLINE;
145                 }
146             }
147         }
148         close (TP);
149         sleep (3) if ($toohigh);
150     }
151 }
152
153 foreach $key (keys (%wordinfo)) {
154     my ($word, $pageid) = split (":", $key);
155     my $hits = $wordinfo{$key};
156     my $wordid = word_id ($word);
157
158     qry_open ("insert into hits (wordid, pageid, hits) " .
159             "values ($wordid, $pageid, $hits)");
160 }
161
162 qry_finalize ();
163 }
164
165 # What index does a distinctive word have?
166 my %word_ids;
167 sub word_id ($) {
168     my $word = shift;
169
170     # Got it already?
171     return ($word_ids{$word}) if ($word_ids{$word} ne "");
172
173     # Already in the DB?
174     my @res = qry_close ("select id from words where word = '$word'");
175     if ($res[0] ne "") {
176         $word_ids{$word} = $res[0];
177         return ($res[0]);
178     }
179
180     # Nope.. put it there.
181     qry_close ("insert into words (word) values ('$word')");
182     my @res = qry_close ("select id from words where word = '$word'");
183     die ("Failed to insert word\n") if ($res[0] eq "");
184     $word_ids{$word} = $res[0];
185     return ($res[0]);
186 }
187
188 # File handling section
189 # =====
190
191 sub file_handle ($ $ $) {

```

```

192     my $fullpath = shift;
193     my $handler = shift;
194     my $pageid = shift;
195
196     $handler =~ s{FULLPATH}{$fullpath};
197
198     open (IF, "$handler |")
199         or die ("failed to start $handler");
200
201     # Reset the page hits.
202     words_reset ($pageid);
203
204     while (<IF>) {
205         s/\n//;
206         s/\r//;
207
208         my @parts = split (/\b/);
209         my $part;
210         foreach $part (@parts) {
211             $part = lc ($part);
212             words_insert ($part, $pageid) if ($part =~ /[a-z]/);
213         }
214     }
215     close (IF);
216
217     print ("\n") if ($opts{'w'});
218
219     # Now flush the hitinfo into the WORDS table.
220     words_flush ();
221 }
222
223 # Indexing section
224 # =====
225
226 # File indexer
227 sub index_file ($ $) {
228     my $file = shift;
229     my $uri = shift;
230
231     my $curdir = getcwd();
232     my $fullpath = "$curdir/$file";
233     print ("File: $fullpath\n");
234
235     my $modtime = (stat($fullpath))[9];
236
237     my $url;
238     $url = $fullpath;
239     $url =~ s/.*$uri/$uri/;
240
241     my ($id, $stamp) = pages_lookup ($url);
242     my @loc = localtime ($stamp);
243     my $readable = sprintf ("%4.4d-%2.2d-%2.2d",
244                             $loc[5] + 1900, $loc[4] + 1, $loc[3]);
245     print ("ID: $id, last indexed on $readable ($stamp)\n");

```

```
246     if ($stamp == 0) {
247         print ("Never handled, indexing.\n");
248     } elsif ($stamp < $modtime) {
249         print ("File modified since last run, will try to index.\n");
250     } else {
251         print ("Indexing was recent enough, not re-indexing\n");
252         return;
253     }
254
255     my $re;
256     my $found = 0;
257     my $now = time();
258     foreach $re (keys (%handlers)) {
259         if ($file =~ /$re$/) {
260             file_handle ($fullpath, $handlers{$re}, $id);
261             $found++;
262         }
263     }
264
265     if (! $found) {
266         print ("WARNING: no handler for file type\n");
267     } else {
268         qry_close ("update pages set stamp = $now where id = $id");
269     }
270 }
271
272 # Directory Indexer
273 sub index_dir ($ $) {
274     my $localpath = shift;
275     my $relativeuri = shift;
276     my $prevdir = getcwd();
277
278
279     my $i;
280     print ("\n");
281     for ($i = 0; $i < 78; $i++) {
282         print ("=");
283     }
284     print ("\nDirectory: $localpath\n");
285     for ($i = 0; $i < 78; $i++) {
286         print ("=");
287     }
288     print ("\n");
289
290     chdir ($localpath) or die ("cannot cd to $localpath: $!\n");
291     $localpath = getcwd();
292     print ("Indexing directory: $localpath\n");
293
294     my @dirs;
295     my $dirent;
296     foreach $dirent (<*>) {
297         index_file ($dirent, $relativeuri) if (-f $dirent);
298         push (@dirs, $dirent) if (-d $dirent);
299     }
```

```

300
301     foreach $direntry (@dirs) {
302         index_dir ($direntry, $relativeuri);
303     }
304     chdir($prevdir) or die ("cannot cd to $prevdir: $!\n");
305 }
306
307
308 # Pages section
309 # =====
310 my %pageinfo;
311
312 # Get the page info. Reads the PAGES table into core.
313 sub pages_getinfo () {
314
315     print ("Getting information on already processed pages.\n");
316     my @res = qry_close ("select id,url,stamp from pages");
317     my $i;
318
319     my $count = 0;
320     for ($i = 0; $i <= $#res; $i += 3) {
321         my $id    = $res[$i];
322         my $uri   = $res[$i + 1];
323         my $stamp = $res[$i + 2];
324
325         $pageinfo{$uri} = "$id:$stamp";
326         $count++;
327     }
328
329     print ("$$count pages were previously processed\n");
330 }
331
332 # Look up a page. Return the ID and the stamp as a list.
333 # If the page wasn't in core yet, then it is added (and to the db
334 # as well).
335 sub pages_lookup ($) {
336     my $uri = shift;
337
338     # Do we have it yet?
339     my $val = $pageinfo{$uri};
340     return (split (":", $val)) if ($val ne "");
341
342     # Nope.. add it.
343     qry_close ("insert into pages (url, stamp) " .
344             "values ('$uri', 0)");
345     my @ret = qry_close ("select id from pages where url = '$uri'");
346     return ($ret[0], 0);
347 }
348
349 # Show usage
350 # =====
351 sub usage () {
352     die ("\n",
353         "Usage: LSE-index [-flags] database localdir relativeurl\n",

```

```

354         "Flags may be:\n",
355         "   -m PERCENTAGE: wait for MySQL until its CPU usage drops\n",
356         "       below PERCENTAGE\n",
357         "   -n WORD,WORD,WORD: skip these non-words\n",
358         "   -w: show words as we go along\n",
359         "The database is the index db.\n",
360         "The local directory is where the indexing will start.\n",
361         "The relative URL is a URL-directory part to each file under\n",
362         "the local directory.\n",
363         "\n");
364     }
365
366     # Main starts here
367     # -----
368     usage() unless getopt ("wn:m:", \%opts);
369     usage() unless (-d $ARGV[1] and $ARGV[2] ne "");
370     $dbname = $ARGV[0];
371
372     $| = 1;
373     pages_getinfo ();
374     index_dir ($ARGV[1], $ARGV[2]);
375

```

## 4.2 LSE.php

```

1  <script language="php">
2  # Central include file for LSE, the Local Search Engine.
3  # -----
4
5  # Defines
6  # -----
7  define ("LSE_OR", 1);           # OR-search
8  define ("LSE_AND", 2);         # AND search
9  define ("LSE_VERBOSE", 4);     # Verbosity on
10 define ("LSE_MATCHSTART", 8);  # Match at start of term
11 define ("LSE_MATCHMID", 16);   # Match at middle of term
12 define ("LSE_DEFAULT", 1);     # Default
13
14 class LSE {
15
16     var $qry;
17     var $verbose;
18     var $limit;
19     var $nreturned;
20     var $beyondlimit;
21     var $logfile;
22
23     function LSE ($dbname, $user, $words, $flags = LSE_DEFAULT,
24                 $lim = 50, $logf = "") {
25
26         # Remember the parameters.

```

```

27     $this->verbose = ($flags & LSE_VERBOSE);
28     $this->limit = $lim;
29     $this->nreturned = 0;
30     $this->beyondlimit = false;
31     $this->logfile = $logf;
32
33     # Since the indexer stores words in lower case, we should
34     # use lower case too.
35     $words = strtolower($words);
36
37     # Initialize the DB connection.
38     $link = mysql_pconnect("localhost", $user)
39         or $this->error ("Failed to connect to MySQL server on " .
40             "localhost as user $user.");
41     $this->msg ("Connected to MySQL on localhost as user $user.");
42     mysql_select_db ($dbname, $link)
43         or $this->error ("Failed to select database $dbname: " .
44             mysql_error());
45     $this->msg ("Selected database: $dbname.");
46
47     # Show what we have in the DB, when verbose.
48     if ($this->verbose) {
49         list ($pages, $latest, $nwords) = $this->info ();
50         $this->msg ($pages . " indexed pages, latest at " .
51             $this->stamptotime ($earliest) . ", " .
52             $nwords . " words in the index.");
53         $this->msg ("Operating flags: $flags");
54         if ($flags & LSE_OR)
55             $this->msg ("OR matching is on.");
56         if ($flags & LSE_AND)
57             $this->msg ("AND matching is on.");
58         if ($flags & LSE_VERBOSE)
59             $this->msg ("VERBOSITY is on.");
60         if ($flags & LSE_MATCHSTART)
61             $this->msg ("STARTMATCH is on.");
62         if ($flags & LSE_MATCHMID)
63             $this->msg ("MIDMATCH is on.");
64     }
65
66     # Chop up the words.
67     $fields = explode (" ", $words);
68
69     # Build up the words selection.
70     $wordsel = "";
71     foreach ($fields as $field) {
72         if ($wordsel == "")
73             $wordsel = "(";
74         else
75             $wordsel .= " or ";
76
77         if ($flags & LSE_MATCHSTART)
78             $wordsel .= "w.word like '$field%'";
79         else if ($flags & LSE_MATCHMID)
80             $wordsel .= "w.word like '%$field%'";

```

```

81         else
82             $wordsel .= "w.word = '$field'";
83     }
84     $wordsel .= ")";
85     $this->msg ("Word selection: $wordsel");
86
87     # Build up the total select statement.
88     $stmt = "select count(*) as nr, w.word, sum(h.hits) as total, p.url " .
89           "from words w " .
90           "inner join hits h on (w.id = h.wordid) " .
91           "inner join pages p on (p.id = h.pageid) " .
92           "where $wordsel " .
93           "group by h.pageid ";
94     if ($flags & LSE_AND)
95         $stmt .= "having nr = " . count($fields) . " ";
96     $stmt .= "order by total desc limit $lim";
97
98     $this->msg ("SQL statement: $stmt.");
99
100    $this->qry = mysql_query ($stmt)
101        or $this->error ("Bad SQL statement: $stmt");
102
103    $this->msg ("Results set will be limited to " . $this->limit . ".");
104 }
105
106 function foundhits() {
107     return ($this->nreturned);
108 }
109
110 function nexturl () {
111     if ($this->nreturned == $this->limit) {
112         $this->beyondlimit = true;
113         mysql_free_result ($this->qry);
114         return (array ());
115     }
116
117     if (! (list ($a, $w, $n, $page) = mysql_fetch_row ($this->qry)) ) {
118         mysql_free_result ($this->qry);
119         return (array ());
120     }
121
122     $this->nreturned++;
123     return (array ($page, $n));
124 }
125
126 function nextpage ($base) {
127     if (! (list ($url, $n) = $this->nexturl ()) )
128         return (array ());
129
130     # Construct the local filename and try to read it.
131     $file = $base . $url;
132     $this->msg ("Next page: " . $file . ", " . $n . " hits.");
133
134     return (array ($url, $n, $this->titleof ($file)));

```

```

135     }
136
137     function contentsof ($file, $max = 200) {
138         $this->msg ("CONTENTSOF starts.");
139
140         if (! file_exists ($file))
141             $this->error ("URL " . $url . " points to non-existing file " .
142                 $file);
143         if (! ($fp = fopen ($file, "r")) )
144             $this->error ("Cannot read " . $file);
145
146         # Extract up to XX chars from the file.
147         $total = "";
148
149         $intagblock = 0;
150         while (! feof ($fp)) {
151             # Append to total.
152             $line = rtrim (fgets ($fp, 4096));
153             $this->msg ("Got line: " . htmlspecialchars($line));
154
155             $addline = true;
156             if ($intagblock) {
157                 $this->msg ("I am inside a TAG BLOCK.");
158                 if (preg_match ("/\<script>/i", $line) or
159                     preg_match ("/\<style>/i", $line)
160                 ) {
161                     $intagblock = 0;
162                     $line = preg_replace ("/.*\<script>/i", "", $line);
163                     $line = preg_replace ("/.*\<style>/i", "", $line);
164                     $this->msg ("ENDTAGBLOCK removed, now " .
165                         htmlspecialchars($line));
166                 } else
167                     $addline = false;
168             }
169
170             if ($addline) {
171                 $lastchar = $total[strlen ($total) - 1];
172                 if ($lastchar != ">" and $lastchar != "<")
173                     $total .= " ";
174                 $total .= $line;
175                 $this->msg ("TOTAL is now: " . htmlspecialchars($total));
176             }
177
178             # Kill <script> ... </script> if we have that,
179             # or <style> ... </style>.
180             $total = preg_replace ("/<script.*>.*\</script>/i", "", $total);
181             $total = preg_replace ("/<style.*>.*\</style>/i", "", $total);
182
183             # Are we inside tag block that we should skip?
184             if (preg_match ("/<script/i", $total) or
185                 preg_match ("/<style/i", $total)
186             ) {
187                 $intagblock++;
188                 $this->msg ("Going into TAG BLOCK mode.");

```

```

189         $total = preg_replace ("/<script.*/i", "", $total);
190         $total = preg_replace ("/<style.*/i", "", $total);
191         $this->msg ("TOTAL now: " . htmlspecialchars($total));
192     }
193
194     # Reached max yet?
195     if (strlen (strip_tags($total)) > $max) {
196         fclose ($fp);
197         $this->msg ("CONTENTSOF ends.");
198         if (preg_match ("/<html>/", $total)) {
199             $this->msg ("CONTENTSOF: it is an HTML file.");
200             return (substr (strip_tags($total), 0, $max) . "...");
201         } else {
202             $this->msg ("CONTENTSOF: Not HTML!");
203             return ("");
204         }
205     }
206 }
207
208 # File was too short.
209 fclose ($fp);
210 $this->msg ("CONTENTSOF reached EOF before MAX.");
211
212 if (preg_match ("/<html>/", $total)) {
213     $this->msg ("CONTENTSOF: it is an HTML file.");
214     return (substr (strip_tags($total)));
215 } else {
216     $this->msg ("CONTENTSOF: Not HTML!");
217     return ("");
218 }
219 }
220
221 function titleof ($file) {
222     if (! file_exists ($file))
223         $this->error ("URL " . $url . " points to non-existing file " .
224             $file);
225     if (! ($fp = fopen ($file, "r")) )
226         $this->error ("Cannot read " . $file);
227
228     # Extract the title from the file.
229     $total = "";
230
231     $nlines = 0;
232     while (! feof ($fp) and $nlines++ < 50) {
233         # Append the current line to the total.
234         $line = rtrim (fgets ($fp, 256));
235         $lastchar = $total[strlen ($total) - 1];
236         if ($lastchar != ">" and $lastchar != "<")
237             $total .= " ";
238         $total .= $line;
239
240         # Do we have a title spec yet?
241         if (preg_match ("/<title>/i", $total) and
242             preg_match ("/<\/title>/i", $total)

```

```
243         ) {
244             # Got the title. Extract it!
245             $this->msg ("HTML string with title: " .
246                 htmlspecialchars($total));
247             $title = preg_replace ("/.*<title>(.*?)<\/title>.*\/i", "$1",
248                 $total);
249             $this->msg ("Extracted title: " .
250                 htmlspecialchars($title));
251             fclose ($fp);
252             return (strip_tags ($title));
253         }
254     }
255     fclose ($fp);
256
257     # We did not get a title...
258     return ("");
259 }
260
261 function limitexceeded () {
262     return ($this->beyondlimit);
263 }
264
265 function info () {
266     $q = mysql_query ("select count(*) from pages");
267     list ($npages) = mysql_fetch_row ($q);
268     mysql_free_result ($q);
269
270     $q = mysql_query ("select max(stamp) from pages");
271     list ($maxstamp) = mysql_fetch_row ($q);
272     mysql_free_result ($q);
273
274     $q = mysql_query ("select count(*) from words");
275     list ($nwords) = mysql_fetch_row ($q);
276     mysql_free_result ($q);
277
278     return (array ($npages, $maxstamp, $nwords));
279 }
280
281 function stampTOTYPE ($stamp) {
282     $parts = localtime ($stamp, true);
283     $txt = sprintf ("%4.4d-%2.2d-%2.2d@%2.2d:%2.2d:%2.2d",
284         $parts["tm_year"] + 1900,
285         $parts["tm_mon"] + 1,
286         $parts["tm_mday"],
287         $parts["tm_hour"],
288         $parts["tm_min"],
289         $parts["tm_sec"]);
290
291     $txt = preg_replace ("/ /", "0", $txt);
292     $txt = preg_replace ("/@/", " ", $txt);
293
294     return ($txt);
295 }
296
```

```

297     function error ($msg) {
298         die ("<p>Search engine error:<br>
299             <i>$msg</i>\n");
300     }
301
302     function msg ($msg) {
303         if ($this->logfile) {
304             if (! ($fp = fopen ($this->logfile, "a")) )
305                 $fp = fopen ($this->logfile, "w");
306             if ($fp) {
307                 fputs ($fp, "$msg\n");
308                 fclose ($fp);
309             }
310         }
311
312         if ($this->verbose) {
313             echo ("<font size=\"1\"> $msg </font><br>\n");
314             flush();
315         }
316     }
317 }
318
319 </script>
320

```

### 4.3 search.php

```

1
2 <!--
3     Here's a sample form that uses the LSE search indexer.
4     The form "calls itself" to make re-searching possible.
5     Also, the form may be called from other site parts.
6 -->
7
8 <script language="php">
9 # First, let's strip possibly offending information from the
10 # form variables. This is always a good idea!
11 # -----
12 $logical = strip_tags ($logical);
13 $words   = strip_tags ($words);
14 $debug   = strip_tags ($debug);
15 </script>
16
17 <html>
18     <head>
19         <title>Search Results</title>
20         <link rel="stylesheet" type="text/css" href="/css/style.css">
21     </head>
22
23     <body>
24

```

```

25     <h1>Search results</h1>
26     <hr>
27
28     <!--
29         This form presents the searching or re-searching
30         functionality. Note that when the form calls itself,
31         it will substitute previously entered options at the
32         right fields.
33
34         The form asks for the following info:
35         - HOW do we search: must all words match (logical AND)
36           or may any of the words match (logical OR);
37         - WHAT we search for, the list of terms
38         - HOW we match each term with the index: must the term
39           match exactly, must it be the start of a word, or may
40           it occur anywhere in the word. If you know SQL: this
41           will lead to
42             word = 'TERM'    or
43             word = 'TERM%'  or
44             word = '%TERM%'
45
46         The checkbox for "debugging" is of course optional.
47         This checkbox is used during development (by me).
48     -->
49
50     <form method="post" action="search.html">
51         <table>
52             <tr>
53                 <td> Search terms: </td>
54                 <td>
55                     <select name="logical">
56                         <option value="or"> Any word may match </option>
57                         <option value="and"
58                             <? if ($logical == "and") echo ("selected"); ?> >
59                             All words must match </option>
60                     </select>
61                     of
62                     <input type="text" name="words" size="30"
63                         <? if ($words) echo ("value=\"\$words\""); ?> >
64                 </td>
65             </tr>
66             <tr>
67                 <td> Match mode: </td>
68                 <td>
69                     <select name="matchmode">
70                         <option value="matchexact"
71                             <? if ($matchmode == "matchexact") echo ("selected"); ?> >
72                             Match terms exactly as stated </option>
73                         <option value="matchstart"
74                             <? if ($matchmode == "matchstart") echo ("selected"); ?> >
75                             Match starts of terms </option>
76                         <option value="matchmid"
77                             <? if ($matchmode == "matchmid") echo ("selected"); ?> >
78                             Match any part of terms </option>

```



```

133     # - the words to search for, as one string, space-separated;
134     # - optional: the operation flags. When not given, LSE_OR is
135     #   the default;
136     # - the limit for the # of returned rows. When not given,
137     #   50 is the default.
138     # - A log file where stuff is appended. Default is an empty string,
139     #   meaning no logging.
140     $lse = new LSE ("intranet", "intranet", $words,
141                   $flags, 50, "/tmp/LSE.log");
142
143     # Start outputting the results as a table.
144     echo("<table>
145         <tr>
146             <td>
147                 <font size=\"1\">
148                     <b>Hits</b>
149                 </font>
150             </td>
151             <td>
152                 <font size=\"1\">
153                     <b>Page, URL and first contents</b>
154                 </font>
155             </td>
156         </tr>\n");
157
158     # Retrieve the results.
159     # The returned values are the (relative) URL to a page with
160     # a hit, the number of hits in that page, and the page title.
161     # The title may be an empty string, when the document doesn't have
162     # a title or when it's not an HTML document.
163     while (list ($url, $n, $title) =
164           $lse->nextpage ("/home/intranet/htdocs")) {
165         $contents = $lse->contentsof ("/home/intranet/htdocs/" . $url);
166         echo("<tr>
167             <td valign=\"top\" align=\"right\">
168                 <font size=\"1\" color=\"red\"> $n </font>
169             </td>
170             <td valign=\"top\">
171                 <font size=\"1\">
172                     <a href=\"$url\"><b>$title</b></a>
173                     (<a href=\"$url\">$url</a>) <br>
174                     <i>$contents</i>
175                 </font>
176             </td>
177         </tr>\n");
178     }
179     echo("</table>\n");
180
181     # All done, the results up to the limit are shown.
182     # If the limit was exceeded, say so.
183     if ($lse->limitexceeded ())
184         echo("<p> There are more results. Try limiting your
185             search by adding words and by matching documents that
186             contain all of the stated words.\n");

```

```
187     else {
188         $nhits = $lse->foundhits ();
189         if (! $nhits)
190             echo ("<p> Your search didn't produce any hits.
191                 Try adding words and searching for documents that
192                 contain any of the stated words; or try a different
193                 search term.\n");
194     }
195
196     # Some stats.
197     list($pages, $latest, $words) = $lse->info();
198     $l_str = $lse->stamptime($latest);
199     echo ("<p>
200         <font size=\"1\">
201             The index holds $words words in $pages pages,
202             last indexed on $l_str.
203         </font>\n");
204
205     # That's all folks!
206
207     </script>
208
209 </body>
210
211 </html>
212
```