

INLS 490-154: Information Retrieval Systems Design and Implementation. Spring 2009.

2. IR with MySQL and Text Files

Chirag Shah*
School of Information & Library Science (SILS)
UNC Chapel Hill NC 27599
chirag@unc.edu

1 Introduction

In this class we will continue (and finish) exploring pure structured data and move toward more unstructured domain of textual information. To help orient us for the rest of our journey, we will present a model of information seeking in the following section. While the focus of this course is information retrieval, it is important to understand how it fits in the overarching issue of information seeking.


In this class we will learn how we can enable MySQL databases for performing searching through fields with textual data. We will use SQL to search through MySQL tables and retrieve records matching the query. At this point we will start talking about concepts such as indexing and stop words. These concepts will stick with us for the rest of our explorations of unstructured textual information.

2 A model for Information Seeking

A general model of information access and organization is depicted in Figure 1 (Shah, 2008). On the left side, four layers are labeled, on the right side, examples are given for these layers, and in the middle, a typical scenario is presented. These four layers are described below in detail.

Layer-1: Information

This layer contains information in various sources and formats (structured, semi-structured, and unstructured). The sources include digital libraries, wikis, blogs, databases, and web-pages; formats include text, images, and videos.

*  These notes for INLS 490-154 Spring 2009 by Chirag Shah (<http://www.unc.edu/~chirags>) are licensed under a Creative Commons Attribution-NonCommercial-No Derivative Works 3.0 United States License.

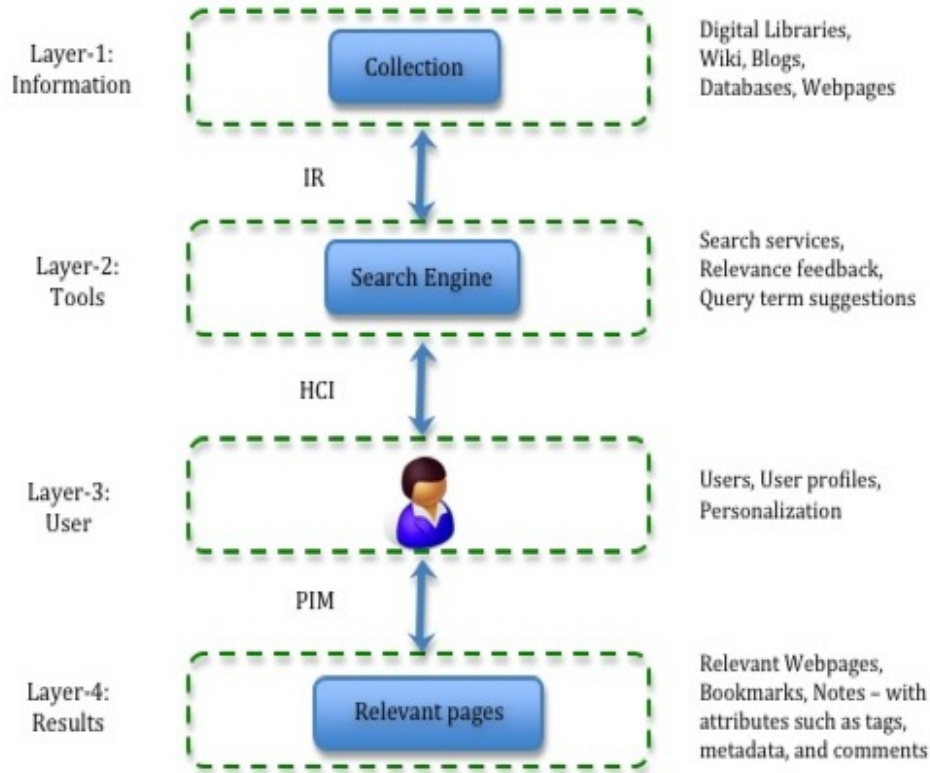


Figure 1: An IR-centric model of Information Seeking (IS)

Layer-2: Tools

This layer consists of tools and techniques a user can use to access the information of layer-1. They include search services, relevance feedback (Buckley, Salton, & Allen, 1994), and query term suggestions (Anick, 2003). In addition, since this layer also acts as a mediating layer between information and users, it includes a variety of user interfaces, starting from results as rank-lists to touch panels with mechanisms to visualize results. We can see that a large amount of research in IR is focused on the link between layer-1 and layer-2; that is, developing tools and services appropriate for retrieving information of various forms.

Layer-3: User

This layer consists of a user, who uses the tools in layer-2 to access the information in layer-1 and accumulate the knowledge in layer-4. We can see that the focus of HCI research has been on the link between layer-2 and layer-3; that is, presenting the information and the information access tools in effective ways to the user. This layer-3 also includes elements relating to a user, such as user profiles, which can be used for personalization (Teevan, Dumais, & Horvitz, 2005).

Layer-4: Results

The user of layer-3 accumulates the information relevant to him in layer-4. In the most basic sense, this could be a set of webpages that the user found relevant from his searches on the Web. Extending this further, we can have bookmarks, notes, and other kinds of

results, sometimes stored with attributes such as tags, metadata, and comments. At a more conceptual level, this layer consists of the knowledge that the user gained by his information seeking process. The focus of research in PIM (Dumais et al., 2003) has been on the link between layers 3 and 4, addressing the issues of information storage and organization by users.

For this course, our focus will be on layer-2, but of course, we need to consider how what we do there fits in the big picture too. Later in the course, we will address some of the issues in the other layers as well.

3 Searching through MySQL databases

Even without doing anything extra, our MySQL database is ready to give us text search functionalities. Let us give it a spin with our ‘world‘ database. Try the queries given below and similar and see what you get.

```
SELECT * FROM Country WHERE HeadOfState LIKE '%bush%';  
SELECT * FROM Country WHERE HeadOfState LIKE '%elisa%';  
SELECT * FROM Country WHERE HeadOfState LIKE '%II%';
```

You would notice that in the above expressions, ‘%’ acts as a wildcard. Thus, looking for %elisa% gives all the records that have ‘elisa’ as a substring.

Now let us take this a step further and see how MySQL supports more sophisticated full-text searching. Add an index to ‘Country’ table by issuing the following command.

```
ALTER TABLE ‘Country‘ ADD FULLTEXT ‘HeadOfState‘ (‘HeadOfState‘);
```

This should create an index (an efficient representation of the field that has been indexed) inside ‘Country‘ table. Once this index is created, we can issue queries such as this:

```
SELECT * FROM Country WHERE MATCH(HeadOfState) AGAINST(‘elisa‘);
```

Since the above query does not contain any wildcards, you should get the results with records where head of state has ‘elisa’ as a full word. Can you obtain the same set of results using LIKE expression?

The real question is - why would we want to create an index if we could do searches using LIKE expression? A comparison between the above two approaches is given in Table 1.

Now to answer the above question, while usage of MATCH requires one to create an index and induces a slight overhead while writing a record, it helps significantly during searches. Without an index, MySQL goes record-by-record looking for an expression (serial scanning approach). This is inefficient, and impractical for large data-sets. Indexing allows MySQL to organize the information in a better data structure that can reduce the search time significantly. On top of that, MySQL also removes stop words from the text while indexing. Stop words are the words that are not useful for storage or matching. Typically, these words include the most frequent words used in a language (e.g., *a*, *an*, *the*, *is*, *are*, etc. for English). In addition to these words,¹ MySQL also discounts all the words that occur in more than 50% of the records or shorter than three characters.

¹MySQL stop words list is available at <http://dev.mysql.com/doc/refman/5.0/en/fulltext-stopwords.html>

Table 1: Comparison of LIKE and MATCH approaches

LIKE	MATCH
No need to create an index	Need to create an index
Serial scanning while searching	Efficient searching with sophisticated data structures
No change in write operation	Write operation becomes slightly costlier
No change in reading operation	Reading (searching) becomes significantly faster
Considers all the terms	Disregards stop words

4 Building a front-end for MySQL database searching

Let us now start building a simple interface for MySQL searching. Our objective here is to create an HTML form that lets one enter a query to be run on the ‘world’ database and return the results in HTML format. Instead of providing the actual code, we will list the steps to design such a system.

1. Create an HTML page with appropriate form fields such as a text box and a submit button. Make that form point to a PHP script.
2. Write a PHP script to handle the form request. Code 1 shows how to read HTML form values using PHP.
3. Establish a connection to MySQL database (Code 2).
4. Use the values read from the HTML form to formulate a query and execute it via PHP (Code 3).
5. Once the result set is returned from MySQL (Code 3), format it in HTML and send it back to the browser.

Code 1: Reading HTML form values with PHP

```
<?php
    // Reading text fields
    $value1 = $_GET['item1'];

    // Reading checkboxes
    $value2 = $_GET['item2'];
    if ($value2 == 'on') // Checkboxes can be on or off
    {
        // Do something
    }

    // Reading radio buttons
    $value3 = $_GET['item3'];
```

```

if ($value3) // Radio buttons have TRUE (1) or FALSE (0) values
{
    // Do something
}

// Reading list boxes
$value4 = $_GET['item4'];
switch ($value4) // List boxes are multiple choices
{
    case 'option1':
        // Do something
        break;
    case 'option2':
        // Do something
        break;
}
?>

```

Code 2: MySQL connection commands in PHP

```

<?php
    $host = "localhost";
    $username = "me";
    $password = "";
    $database = "example";
    $dbh = mysql_connect($host,$username,$password)
        or die("Cannot connect to the database: ". mysql_error());
    $db_selected = mysql_select_db($database)
        or die ('Cannot connect to the database: ' . mysql_error());
?>

```

Code 3: Create and execute a SQL query as well as read the records in PHP

```

<?php
    // Formulate the query
    $query = "SELECT * FROM table_name";
    // Execute the query, get the results
    $results = mysql_query($query) or die(" ". mysql_error());
    // Go record by record
    while ($line = mysql_fetch_array($results, MYSQL_ASSOC))
    {
        // Do something
    }
?>

```

5 Working with text files for IR

Let us now see how we could start dealing with unstructured data. To begin our exploration, which will continue for the rest of this course, we will take a very simple example. We have a text file and we want to look through it for a word. Of course, utilities such as `grep` in UNIX can do this easily. But let us write a simple program to achieve a similar effect.²

We have listed such a PHP script in Code 4. It reads the query (a word) from the input, looks for that query in a document (hardcoded here), and prints ‘Found’ if/when found that query.

Code 4: Searching through a text file with PHP

```
<?php
$query = trim(fgets(STDIN));
$fin = fopen("document.txt", "r");

while ($line = fgets($fin))
{
    $words = explode(" ", $line);
    $i = 0;
    $found = 0;
    while (($words[$i] && ($found==0))
    {
        if (strcmp($words[$i], $query) == 0)
        {
            echo "Found!\n";
            $found = 1;
        }
        $i++;
    }
}

fclose($fin);
?>
```

This process works, but has several issues that are not quite visible in the example given. For instance, it is case sensitive. It assumes *clean* text, without punctuation and other special symbols attached to the words. It does not allow matching phrases or look for more than one words. And above all, it is really an inefficient process, which makes it almost impractical for large-scale systems. In the next class we will see how we could take this simple process and start enhancing it so that we achieve something more *practical*.

²Note that `grep` is really powerful and allows one to use regular expressions in searching.

6 Summary

- Information retrieval can be seen as a part of the overarching domain of information seeking. We will focus on information retrieval, but also see how it fits with other components in an information seeking model.
- MySQL inherently supports searching with regular expressions. To achieve better performance while searching on textual information, we need to index that field.
- Indexing allows one to prepare a more sophisticated representation of the information that can help in better searching.
- Stop words are the words that we do not want to store or process since they are not very useful while searching.

References

- Anick, P. (2003). Using terminological feedback for web search refinement - a log-based study. In *Proceedings of ACM SIGIR* (p. 88-95).
- Buckley, C., Salton, G., & Allen, J. (1994). The effect of adding relevance information in a relevance feedback environment. In *Proceedings of ACM SIGIR* (p. 292-300). New York, NY: Springer-Verlag.
- Dumais, S. T., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., & Robbins, D. C. (2003, August). Stuff I've Seen: A System for Personal Information Retrieval and Re-Use. In *Proceedings of ACM SIGIR*. ACM Press.
- Shah, C. (2008, June 20). Toward Collaborative Information Seeking (CIS). In *Collaborative exploratory search workshop*. Pittsburgh, PA.
- Teevan, J., Dumais, S. T., & Horvitz, E. (2005). Personalizing search via automated analysis of interests and activities. In *Proceedings of ACM SIGIR* (p. 449-456).