Searching For A Good Search

It often surprises those who have never looked at server logs (the detailed statistics about a website) that a tremendous percentage of site visitors come from searches. In the case of the Center for History and New Media, this is a staggering 400,000 unique visitors a month out of about one million. Furthermore, many of these visitors ignore a website's navigation and go right to the site search box to complete their quest for information. While I'm not a big fan of consultants that tell webmasters to sacrifice virtually everything for usability[1], I do feel that searching has been undervalued by digital humanities projects, in part because so much effort goes into digitization, markup, interpretation, and other time-consuming tasks. But there's another, technical reason too: it's actually very hard to create an effective search—one, for instance, that finds phrases as well as single words, that is able to rank matches well, and that is easy to maintain through software and server upgrades. In this installment of "Creating a Blog from Scratch" (for those who missed them, here are parts 1, 2, and 3) I'll take you behind the scenes to explain the pluses and minuses of the various options for adding a search feature to a blog, or any database-driven website for that matter.

There are basically four options for searching a website that is generated out of a database: 1) have the database do it for you, since it already has indexing and searching built in; 2) install another software package on your server that spiders your site, indices it, and powers your search; 3) use an application programming interface[2] (API) from Google, Yahoo, or MSN to power the search, taking search results from this external source and shoehorning them into your website's design; 4) outsourcing the search entirely by passing search queries to Google, Yahoo, or MSN's website, with a modifier that says "only search my site for these words."

Option #1 seems like the simplest. Just create an SQL statement (a line of code in database lingo) that sends the visitor's query to the database software—in the case of this blog, the popular MySQL—and have it

return a list of entries that match the query. Unfortunately, I've been using MySQL extensively for five years now and have found its ability to match such queries less than adequate. First of all, until the most recent version of the MySQL it would not handle phrase searching at all, so you would have to strip quotation marks out of queries and fool the user into believing your site could do something that it couldn't (that is, do a search like Google could). Secondly, I have found its indexing and ranking schemes to be far behind what you expect from a major search engine. Maybe this has changed in version 5, but for many years it seemed as if MySQL was using search principles from the early 1990s, where the number of times a word appeared on the page signified how well the page matched the query (rather than the importance of the place of each instance of the word on the page, or even better, how important the document was in the constellation of pages that contained that word). MySQL will return a fraction from 0 to 1 for the relevance of a match, but it's a crude measure. I'm still not convinced, even with the major upgrades in version 5, that MySQL's searching is acceptable for demanding users.

Option #2 is to install specialized search packages such as the open source ht://Dig[3] on your server, point it to your blog (or website) and let it spider the whole thing, just as Google or Yahoo does from the outside. These software packages can do a decent job indexing and swiftly finding documents that seem more relevant than the rankings in MySQL. But using them obviously requires installing and maintaining another complicated piece of software, and I've found that spiders have a way of wandering beyond the parameters you've set for them, or flaking out during server upgrades. (Over the last few days, for instance, I've had two spiders request hundreds of posts from this blog that don't exist. Maybe they can see into the future.) Anecdotally, I also think that the search results are better from commercial services such as Google or Yahoo.

I've become increasingly enamored of **Option #3**, which is to use APIs,

or direct server-to-server communications, with the indices maintained by Google, Yahoo, or Microsoft. The advantage of these APIs is that they provide you with very high quality search results and query handling (at least for Google and Yahoo; MSN is far behind). Ranking is done properly, with the most important documents (e.g., blog posts that many other bloggers link to or that you have referenced many times on your own site) coming up first if there are multiple hits in the search results. And these search giants have far more sophisticated ways of handling phrase searches (even long ones) and boolean searches than MySQL. The disadvantage of APIs is that for some reason the indices made available to software developers are only a fraction the size of the main indices for these search engines, and are only updated about once a month. So visitors may not find recent material, or some material that is ranked fairly low, through API searches. Another possibility for Option #3 is to use the API for a *blog* search engine, rather than a broad search engine. For instance, Technorati has a blog-specific search API[4]. Since Technorati automatically receives a ping from my Atom feed every time I post (via FeedBurner[5]), it's possible that this (or another blog search engine) will ultimately provide a solid API-based search.

I've been experimenting with ways of getting new material into the main Google index swiftly (i.e., within a day or two rather than a month or two), and have come up with a good enough solution that I have chosen **Option #4**: outsourcing the search entirely to Google, by using their free (though unfortunately ad-supported) site-specific search. With little fanfare, this year Google released Google Sitemaps[6], which provides an easy way for those who maintain websites, especially database-driven ones, to specify where all of their web pages are using an XML schema. (Spiders often miss web pages generated out of a database because there are often so many of them, and some of these pages may not be linked to.) While not guaranteeing that everything in your sitemap will be crawled and indexed, Google does say that it makes it easier for them to crawl your site more effectively. (By the way, Google's recent acquisition of 5 percent of AOL seems to have been, at least ostensibly, very much

about providing AOL with better crawls[7], thus upping the visibility of their millions of pages without messing with Google's ranking schemes.) And—here's the big news if you've made it this far—I've found that having a sitemap gets new blog posts into the main Google index extremely fast. Indeed, usually within 24 hours of submitting a new post Google downloads my updated sitemap (created automatically by a PHP script I've written), sees the new URL for the post, and adds it to its index. This means that I can very effectively use the Google's main search engine for this blog, although because I'm not using the API I can't format the results page to match the design of my site exactly.

One final note, and I think an important one for those looking to increase the visibility of their blog posts (or any web page created from a database) in Google's search results: have good URLs, i.e., ones with important keywords rather than meaningless numbers or letters. Database-driven sites often have such poor URLs featuring an ugly string of variables, which is a shame, since server technology (such as Apache's mod_rewrite[8]) allows webmasters to replace these variables with more memorable words. Moreover, Google, Yahoo, and other search engines clearly favor keywords in URLs (very apparent when you begin to work with Google's Web API), assigning them a high value when determining the relevance of a web page to a query. Some blog software automatically creates good URLs (like Blogger, owned by Google), while many other software packages do not—typically emphasizing the date of a post in the URL or the page number in the blog. For my own blogging software, I designed a special field in the database just for URLs, so I can craft a particularly relevant and keyword-laden string. Mod_rewrite takes care of the rest, translating this string into an ID number that's retrieved by the database to generate the page you're reading.

For many reasons, including making it accessible to alternative platforms such as audio browsers and cell phones, I wanted to generate this page in strict XHTML[9], unlike my old website, which had poor coding practices left over from the 1990s. Unfortunately, as the next post in this series details, I failed terribly in the pursuit of this goal, and this floundering

made me think twice about writing my own blogging software when existing packages like WordPress will generate XHTML for you, with no fuss.

Part 5: What is XHTML, and Why Should I Care?[10]

This entry was posted on Monday, December 26th, 2005 at 9:15 pm and is filed under APIs[11], Blogs[12], Google[13], Programming[14], Search[15], Software[16], Usability[17]. You can follow any responses to this entry through the RSS 2.0[18] feed. You can leave a response[19], or trackback[20] from your own site.

References

1. ^ I'm not a big fan of consultants that tell webmasters to sacrifice virtually everything for usability (chnm.gmu.edu)
2. ^ application programming interface (www.dancohen.org)
3. ^ ht://Dig (www.htdig.org)
4. ^ Technorati has a blog-specific search API (technorati.com)
5. ^ FeedBurner (www.feedburner.com)
6. ^ Google Sitemaps (www.google.com)
7. ^ very much about providing AOL with better crawls (googleblog.blogspot.com)
8. ^ Apache's mod_rewrite (httpd.apache.org)
9. ^ XHTML (www.w3.org)
10. ^ Part 5: What is XHTML, and Why Should I Care? (www.dancohen.org)
11. ^ View all posts in APIs (www.dancohen.org)
12. ^ View all posts in Blogs (www.dancohen.org)
13. ^ View all posts in Google (www.dancohen.org)
14. ^ View all posts in Programming (www.dancohen.org)
15. ^ View all posts in Search (www.dancohen.org)
16. ^ View all posts in Software (www.dancohen.org)
17. ^ View all posts in Usability (www.dancohen.org)

18. ^ RSS 2.0 (www.dancohen.org)
19. ^ leave a response (www.dancohen.org)
20. ^ trackback (www.dancohen.org)

Readability — An Arc90 Laboratory Experiment

http://lab.arc90.com/experiments/readability