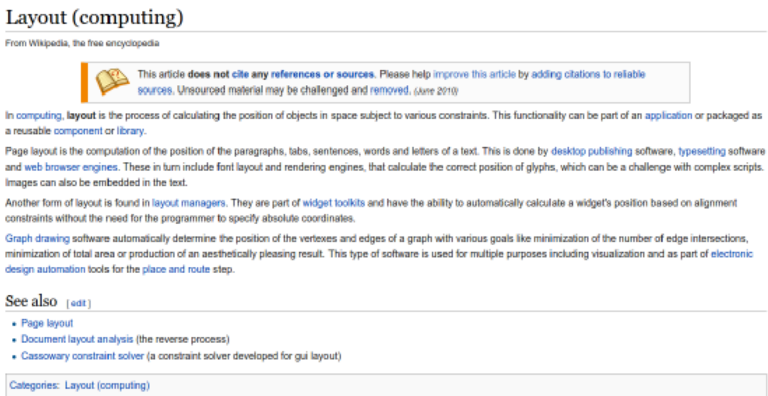
Database System Technology
Lab Project Report

Zhuoqun Chen, Wenbo Li, Linghao Zhu

1. Project description

In this course students are asked to implement a group-based mini research project. Among the three data sources, our group chose the Wikipedia database. Wikipedia offers free copies of all available content to interested users. After preprocessing, we can get the plain text of articles.



Layout (computing)

From Wikipedia, the free encyclopedia

This article **does not cite any references or sources**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. *(June 2010)*

In computing, **layout** is the process of calculating the position of objects in space subject to various constraints. This functionality can be part of an application or packaged as a reusable component or library.

Page layout is the computation of the position of the paragraphs, tabs, sentences, words and letters of a text. This is done by desktop publishing software, typesetting software and web browser engines. These in turn include font layout and rendering engines, that calculate the correct position of glyphs, which can be a challenge with complex scripts. Images can also be embedded in the text.

Another form of layout is found in layout managers. They are part of widget toolkits and have the ability to automatically calculate a widget's position based on alignment constraints without the need for the programmer to specify absolute coordinates.

Graph drawing software automatically determine the position of the vertexes and edges of a graph with various goals like minimization of the number of edge intersections, minimization of total area or production of an aesthetically pleasing result. This type of software is used for multiple purposes including visualization and as part of electronic design automation tools for the place and route step.

See also [edit]

- Page layout
- Document layout analysis (the reverse process)
- Cassowary constraint solver (a constraint solver developed for gui layout)

Categories: Layout (computing)

A wiki page is defined within a pair of html label <doc>, containing page information (page id, page url and wiki title). All the links are noted by a html hyper link destination is a wiki page title (wiki page linked to and content is the anchor text (the real words in page).

There are three required queries listed below.

1) Given an anchor, return all the link and their sentence-level co-occurrence count.
2) Given a wiki title, return all the anchor linked to it and their sentence-level co-occurrence count.
3) Given a link and word pair, return their sentence-level co-occurrence count.

2. Data description

```
<doc id="27469364" url="http://en.wikipedia.org/wiki?curid=27469364" title="Layo
ut (computing)">
Layout (computing)

In <a href="computing">computing</a>, layout is the process of calculating the p
osition of objects in space subject to various constraints. This functionality c
an be part of an <a href="Application software">application</a> or packaged as a
 reusable <a href="Component-based software engineering">component</a> or <a hre
f="Library (computing)">library</a>.
Page layout is the computation of the position of the paragraphs, tabs, sentence
s, words and letters of a text. This is done by <a href="desktop publishing">des
ktop publishing</a> software, <a href="typesetting">typesetting</a> software and
 <a href="web browser engine">web browser engines</a>. These in turn include fon
t layout and rendering engines, that calculate the correct position of glyphs, w
hich can be a challenge with complex scripts. Images can also be embedded in the
 text.
Another form of layout is found in <a href="layout manager">layout managers</a>.
 They are part of <a href="widget toolkit">widget toolkits</a> and have the abil
ity to automatically calculate a widget's position based on alignment constraint
s without the need for the programmer to specify absolute coordinates.
<a href="Graph drawing">Graph drawing</a> software automatically determine the p
osition of the vertexes and edges of a graph with various goals like minimizatio
n of the number of edge intersections, minimization of total area or production
of an aesthetically pleasing result. This type of software is used for multiple
purposes including visualization and as part of <a href="electronic design autom
ation">electronic design automation</a> tools for the <a href="place and route">
place and route</a> step.

</doc>
```

The plain text above is a sample wiki page. One wiki page is denoted by <doc> and </doc> in the xml file. An anchor and a link are coupled within a <a href> sign. And each paragraph is a single line. So for the three required queries, we need a relation between links, anchors and words. Title is not mentioned because in our preprocessing we regard titles as the same content as links.

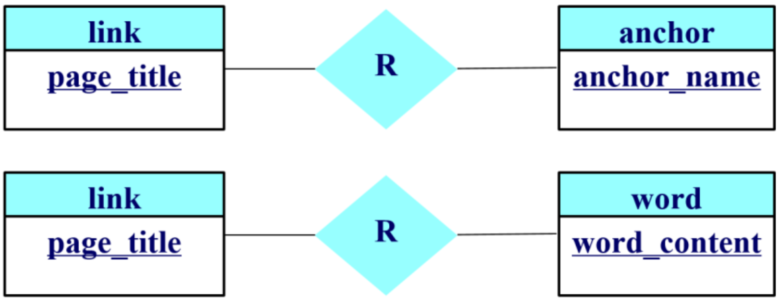
3. Implementation

3.1 System Environment

Our work are mainly divided into two parts, raw data preprocessing and database establishment. We use a 32G memory Think-Station to preprocessing the raw data and a personal PC with 8G memory and Windows 10 system to establish databases and run queries. We chose the personal PC in order to show the improved and optimized performance is due to the database design instead of strong hardware backup.

3.2 Model design

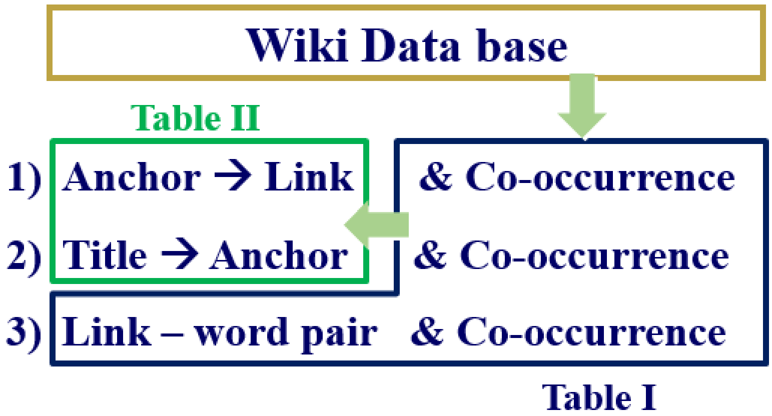
Since we only need to establish relations between links and anchors and links and words, we just followed the two simple ER models below. Both models represent a many to many relation between link and anchor or word. We use their content, i.e. the word(s) that they consist of, as the primary key. Although encoding those links, anchors and words into some binary or integer representation is an option, we didn't do the encoding since String type works fine and also has a good query performance.



3.3 Database preprocessing

3.3.1 Raw data decomposition

In this section we are going to introduce how we preprocess the raw data. Since the 12G raw Wiki database contains 4633003 wiki pages and millions of link-anchor pairs and link-word pairs, traversing the whole database for multiple times is unacceptable. So we designed a way to allow us get enough information in one traversal. As the following image shows, we can reorganize the three required queries into two progressive problems, resulting in two cascaded tables.

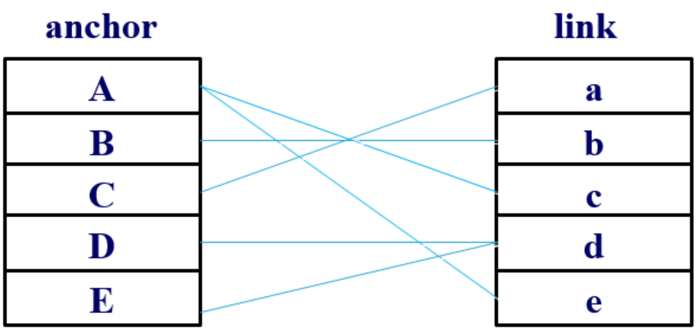


We simply traverse the raw wiki database and use link as our primary focus. During the traversal we record every new link-word pair and count their co-occurrence page by page, either adding a new link-word pair, or refreshing an existing link-word pair's co-occurrence. Since we do the recording page by page, this method, if time allows, can be further developed into a map-reduce problem, which can definitely boost the preprocessing performance. After one traversal we get our Table I, containing four columns, link, a/w, type, count. Link is the primary key here. 'a/w' is the word or the anchor we record. Noted that a word can simultaneously be a regular word and an anchor. So when we recorded them we recorded them separately, with the type column to clarify what kind of pair are they. The count column denotes the co-occurrence count between a link and an anchor or a regular word. When doing queries, if a word is also an anchor, we can simply add the L-A type' and L-W type's count. An example for Table I is shown below.

e.g. Computer centers have computers and computer tools.
Tool USA Science

Link	A/W	type	Cnt
tool	computer	L-A	2
tool	centers	L-A	0
tool	Computer	L-W	1
…	…	…	…
usa	computer	L-A	2
usa	centers	L-A	1
usa	computer	L-W	0
…	…	…	…

After the Table I is created, the third required queries is than automatically solved. Then we need to move to the other two required queries. Actually those two queries are already been down, since we can run a query to check the co-occurrence and the relation between a link and an anchor in Table I. But this would simply cost too much. So from Table I we need to establish another table focused only on the relation between anchors and links, Table II. It is a many-to-many relation and the table can be obtained by traversing the Table I, which costs far less than traversing the raw database again. The brief relation of Table II is shown below.



During our one-time traversal of the raw database, we also used some technics to avoid complexity and improve the performance, on both accuracy and efficiency front.

3.3.2 Linguistic normalization

In order to avoid complexity and additional disk space waste, and to be more like a search engine, we performed a linguistic normalization on the words before they are recorded into the Table I. We used the Porter2 Stemmer to filter our words.

word		stem	word		stem
consign		consign	knack		knack
consigned		consign	knackeries		knackeri
consigning		consign	knacks		knack
consignment		consign	knag		knag
consist		consist	knave		knave
consisted		consist	knaves		knave
consistency		consist	knavish		knavish
consistent		consist	kneaded		knead
consistently		consist	kneading		knead
consisting		consist	knee		knee
consists		consist	kneel		kneel
consolation		consol	kneeled		kneel
consolations		consol	kneeling		kneel
consolatory		consolatori	kneels		kneel
console		consol	knees		knee
consoled		consol	knell		knell
consoles		consol	knelt		knelt
consolidate		consolid	knew		knew
consolidated		consolid	knick		knick
consolidating	=>	consolid	knif	=>	knif
consoling		consol	knife		knife

Here is an example how a Porter2 Stemmer works. Although there could be some small collateral damage, but linguistic normalization can save tons of disk space and also improve the accuracy of the co-occurrence counting.

Link	Word	Cnt
wiki	computer	8
wiki	computers	4
wiki	Computer	5
wiki	Computers	7

Link	Word	Cnt
wiki	computer	24

For example during preprocessing, there is a very likely scenario as shown in the above image. The word ‘computer’ can be denoted as ‘computer’, ‘computers’, ‘Computer’ or ‘Computers’. They all mean the same and yet they takes 4 times disk space. Also, without linguistic normalization, when a user type different expression of ‘computer’, he can get different results like 8, 4, 5 or 7, while the accurate co-occurrence is 24. The linguistic normalization and avoid these problems and make our system more like a real life search engine.

But after evaluation we can see that there can be some collateral damage. For example ‘consign’ and ‘consignment’ are different words, but they could be stemmed into the same category. It’s a small price to pay but far less than what we’ve gained here.

Note that we only perform the linguistic normalization on words, not links or anchors.

Because even two anchors, ‘computer’ and ‘computers’, they still could mean different things and has different links in the database. So we decided to only run linguistic normalization on the regular words.

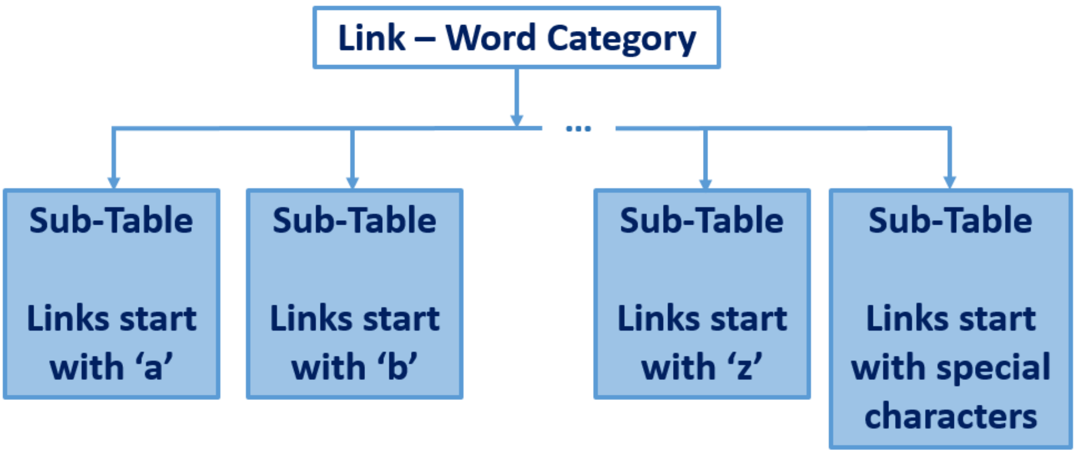
3.3.3 Stop words

Also, to avoid huge computational cost and unnecessary word, we make a list of stop words, containing words like ‘a’, ‘the’, ‘of’, etc. Thus we can avoid huge computational and spatial cost on meaningless words.

3.3.4 First order indexing

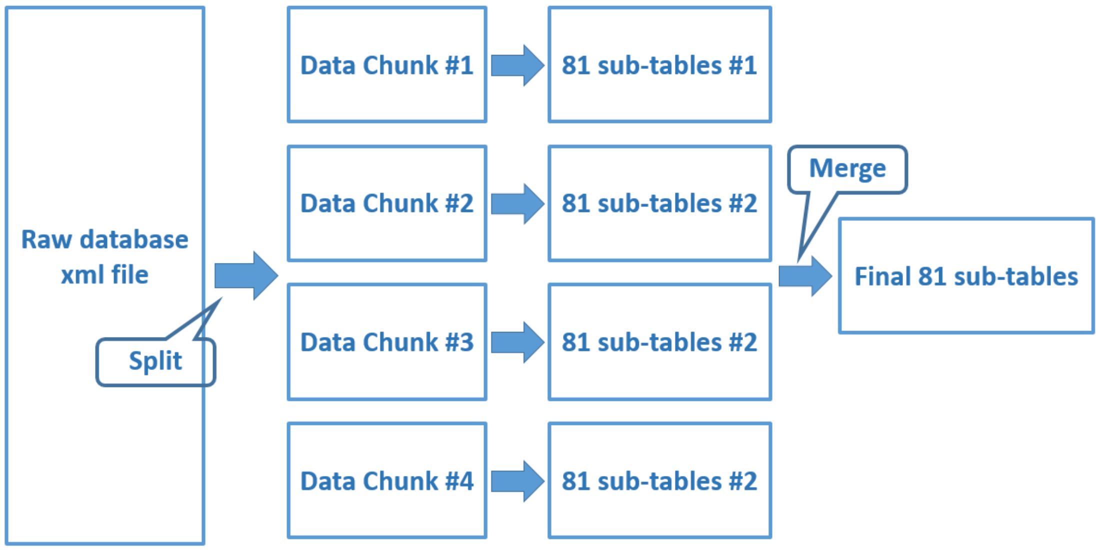
Although Table I has everything we need and only traversed the raw database once, it doesn’t mean this is the table that can be put in use. In fact, when we run the preprocessing code on the workstation, it took days to record word pairs after one million pages. That is because it’s too busy traversing Table I. But the raw database contains more than four million pages. So we need our first order indexing during the preprocessing work.

Still, using link as our primary key, we built our first order index on link column. The index is simple. We divided the whole table into 27 sub-tables, as link’s firs letter ‘a’ to ‘z’ and an additional table for special characters. And for convenience sake, we established Table II the same time as Table I, just adding some simple codes for it to work. And during creating Table II, we also use anchor as the primary key to index its relation with links. This step won’t cost too much, since it just reverses the order of a link and an anchor to create a same table with different indexing. After the preprocessing we then have three categories, ‘link-word’, ‘link-anchor’, ‘anchor-link’, and 81 sub-tables. Each categories refers to one required query. Also, as later it proved, this kind of indexing is good for uploading data to the database server, even on a regular personal PC. The structure and indexing of link-word category and sub-tables are shown below.



3.3.5 Splitting and merging

Although we used the first order indexing, the preprocessing still took too long. So we split the raw database (the xml file) into four pieces, each of which has a little more than one million pages. The four data chunks produced 4*81 sub-tables. We then merge those sub-tables to the final 81 sub-tables.



3.4 MySQL and second order indexing

We install MySQL Server and XAMPP on a personal computer with 8G memory. We didn't do the indexing and queries on the workstation because we want to see how well can our search engine perform on a regular PC. We have 81 sub-tables (approximately 20G), which contains about 380 million records. So we need to think of a smart way to upload those data to the MySQL server. Luckily MySQL has operations that allow us we do SQL command in a batch manner. For example in our project we upload 40000 records per batch, instead of insert those records one by one. This can save us much time since the system won't be needing a hand-shake like connection for every SQL operation.

After we upload the data to the MySQL server, we used XAMPP's phpMyAdmin to check the database. Also we added a second order indexing for some of the tables using phpMyAdmin interface.

3.5 UI Design

We designed our UI using XAMPP, so the UI is on a local website. There are several panels corresponding to the three required queries. Noted that for link-word category, the input word will be normalized or be identified if it is a stop word. We implemented it by calling a python function into the php file. The details of the UI will be shown later in Section 5.

4. Performance Evaluation

4.1Time cost

We ran our preprocessing on a 32G memory workstation. Before we used the first order indexing, the preprocessing took days. So we stop the process and apply the first order indexing. It only took 10 hours to get the 81 sub-tables.

For uploading data to MySQL server and run queries we used a regular PC with 8G memory. The uploading took only 2 hours to upload 20G, 380 million records in a batch processing manner. And as we will mention later, our search engine can return results for most queries within 0.05 second.

4.2 Database and tables

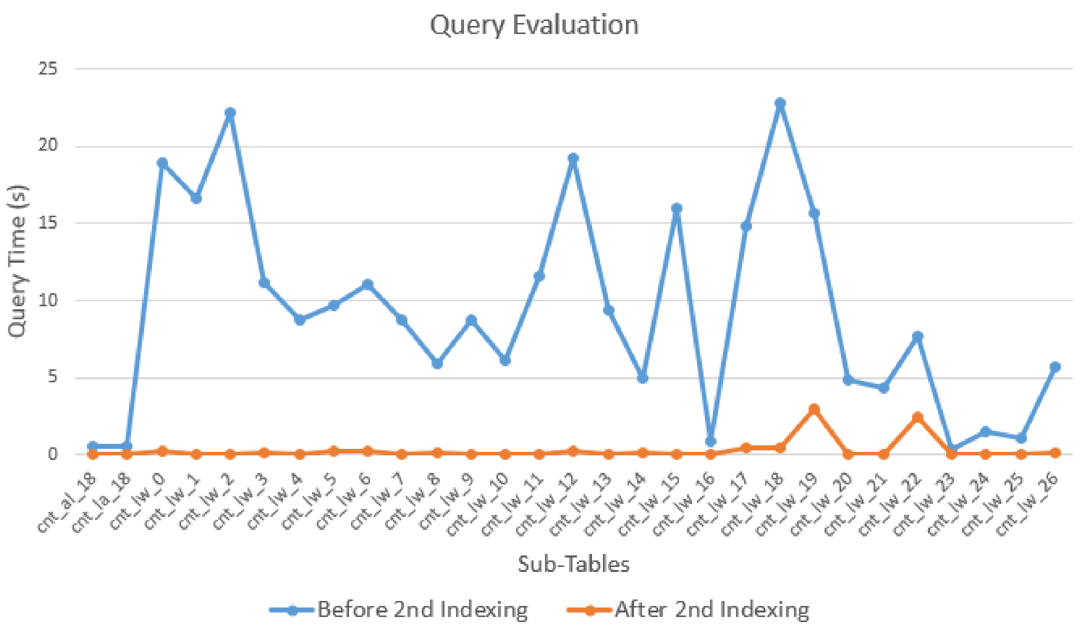
Here are details about our database and tables.

Name	Records	Size	Name	Records	Size	Name	Records	Size
cnt_al_0	596430	42.6 MB	cnt_la_0	598973	42.6 MB	cnt_lw_0	24833174	1433 MB
cnt_al_1	499436	34.6 MB	cnt_la_1	511479	35.6 MB	cnt_lw_1	21443768	1228 MB
cnt_al_2	695960	50.6 MB	cnt_la_2	684834	49.6 MB	cnt_lw_2	29018298	1536 MB
cnt_al_3	370649	25.6 MB	cnt_la_3	356134	24.6 MB	cnt_lw_3	14981845	827 MB
cnt_al_4	301471	21.5 MB	cnt_la_4	301850	21.5 MB	cnt_lw_4	11885829	636 MB
cnt_al_5	330940	24.6 MB	cnt_la_5	321217	23.5 MB	cnt_lw_5	12535713	707 MB
cnt_al_6	339748	23.6 MB	cnt_la_6	352019	24.6 MB	cnt_lw_6	14336181	791 MB
cnt_al_7	333158	23.5 MB	cnt_la_7	350885	24.6 MB	cnt_lw_7	13546017	767 MB
cnt_al_8	236987	17.5 MB	cnt_la_8	235432	17.5 MB	cnt_lw_8	9149591	524 MB
cnt_al_9	270548	18.5 MB	cnt_la_9	288097	19.5 MB	cnt_lw_9	12688932	655 MB
cnt_al_10	230841	15.5 MB	cnt_la_10	231276	15.5 MB	cnt_lw_10	9004081	473 MB
cnt_al_11	370790	26.6 MB	cnt_la_11	451615	33.6 MB	cnt_lw_11	15660525	885 MB
cnt_al_12	604173	42.6 MB	cnt_la_12	609921	42.6 MB	cnt_lw_12	25570059	1433 MB
cnt_al_13	308305	23.5 MB	cnt_la_13	310915	23.5 MB	cnt_lw_13	12700152	699 MB
cnt_al_14	177097	12.5 MB	cnt_la_14	171529	12.5 MB	cnt_lw_14	6629748	378 MB
cnt_al_15	511465	36.6 MB	cnt_la_15	510354	36.6 MB	cnt_lw_15	21874973	1126 MB
cnt_al_16	33028	2.5 MB	cnt_la_16	31555	2.5 MB	cnt_lw_16	1308472	74 MB
cnt_al_17	390959	27.6 MB	cnt_la_17	389010	27.6 MB	cnt_lw_17	16173652	853 MB
cnt_al_18	830374	58.6 MB	cnt_la_18	789252	55.6 MB	cnt_lw_18	32511471	1741 MB
cnt_al_19	565765	40.6 MB	cnt_la_19	530264	37.6 MB	cnt_lw_19	22425716	1229 MB
cnt_al_20	129943	10.5 MB	cnt_la_20	155728	12.5 MB	cnt_lw_20	6155202	379 MB
cnt_al_21	140700	9.5 MB	cnt_la_21	141385	9.5 MB	cnt_lw_21	5750131	325 MB
cnt_al_22	261872	18.5 MB	cnt_la_22	260730	18.5 MB	cnt_lw_22	10319500	599 MB
cnt_al_23	14017	1.5 MB	cnt_la_23	13666	1.5 MB	cnt_lw_23	517663	28 MB
cnt_al_24	51365	3.5 MB	cnt_la_24	51594	3.5 MB	cnt_lw_24	2048584	111 MB
cnt_al_25	38314	2.5 MB	cnt_la_25	38420	2.5 MB	cnt_lw_25	1528981	78 MB
cnt_al_26	233852	16.5 MB	cnt_la_26	412420	33.6 MB	cnt_lw_26	7926336	521 MB
Total	8868187	631.7 MB	Total	9100554	652.8 MB	Total	362524594	20036 MB

4.3 Indexing

We ran some tests on our second order indexing. The second order indexing was built on each sub-table's primary key. The second order indexing is for the link-word category, since each sub-table in this category has tens of millions of records and it may take a while to show the search result. The other two categories, link-anchor and anchor-link, are relatively small and the query time in average is under 0.01s. So a second order indexing for those two categories is not necessary.

We established second order indexing for every sub-table in link-word category and the largest sub-table in the other two category. We chose to query for the last record in a sub-table, regarding it as the worst case scenario. We can do this because for each record in each sub-table, MySQL has to traverse every record before it before fetching the record. Before the second order indexing, the worst case scenario for sub-tables in link-word category is around 15s, above 20s for some large sub-tables. And the sub-tables in the other two categories have the worst case scenario of 0.5s, which is enough for a regular query, but we still want to improve it.



After the second indexing we can see almost all sub-tables has the worst case scenario close to zeros, an average of 0.05s actually. So now for most queries in our search engine, which runs on a regular PC, we can return the result within 0.05 second.

5. Demonstration

5.1 Front page

5.2 Anchor-Link search (Required queries No.1)

Anchor-Link Search

Anchor:

Search Reset

Search Result

94 links related to "andy" are found.

Anchor	andy	
	andy bernard	77
	andy dwyer	67
	andy sugden	15
	andy lee (korean singer)	14
	andy schleck	12
	andy gibb	12
	andy mcdonald (coronation street)	7
	andy biersack	6
	andy flower	4
	andy murray	4
	andy moore (footballer)	4
	andranik madadian	4
	andy trudeau	3
	andy creeggan	3
	andy (album)	3
	andy davidson (torchwood)	3
	andy dawson	3
	list of fatal fury characters#andy bogard	3
	andy williams (doves)	3
	andy davis (toy story)	2
	andy mccall (footballer born 1925)	2

5.3 Link-Anchor search (Required queries No.2)

- Further Search
- Sign Up

Link-Anchor Search

Link:

Search Reset

Search Result

17 anchors related to "wiki" are found.

Link	wiki	
Anchor/ Co-occurrence	wiki	579
	wikis	135
	wiki community	2
	wiki content	1
	greek	1
	wiki site	1
	wiki-based	1
	user edited content	1
	wiki system	1
	wiki-styled	1
	wikiwiki	1
	wiki process	1
	wiki-database	1
	wiki-	1
	collaboratively edited	1
	wiki webpages	1
	wikia	1

5.4 Link-Word search (Required queries No.3)

5.5 Stop words filtering

5.6 Linguistic normalization

Wikipedia Search

- Search Tools
  - Link-Anchor Search
  - Anchor-link Search
  - Link-word Search
  - Further Search
- Sign Up

Link-Word Search

Link:
Word:
Search Reset

stop word or not ? -- "computers" : False
stem ? -- "computers" : comput

Search Result

Link	wiki	Word	computers
Co-occurence count	13		

Wikipedia Search

- Search Tools
  - Link-Anchor Search
  - Anchor-link Search
  - Link-word Search
  - Further Search
- Sign Up

Link-Word Search

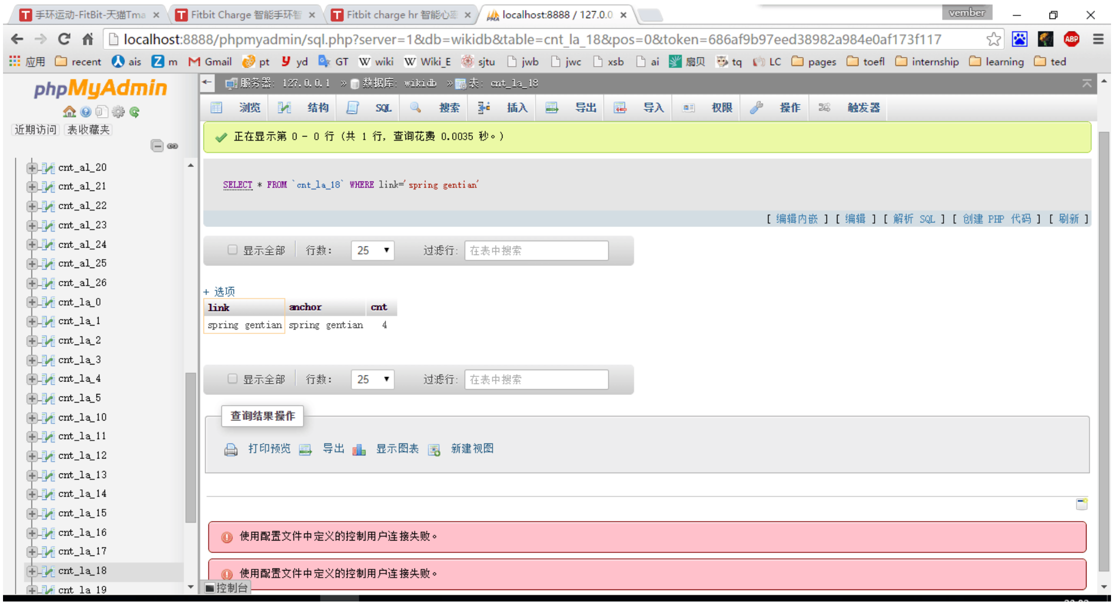
Link:
Word:
Search Reset

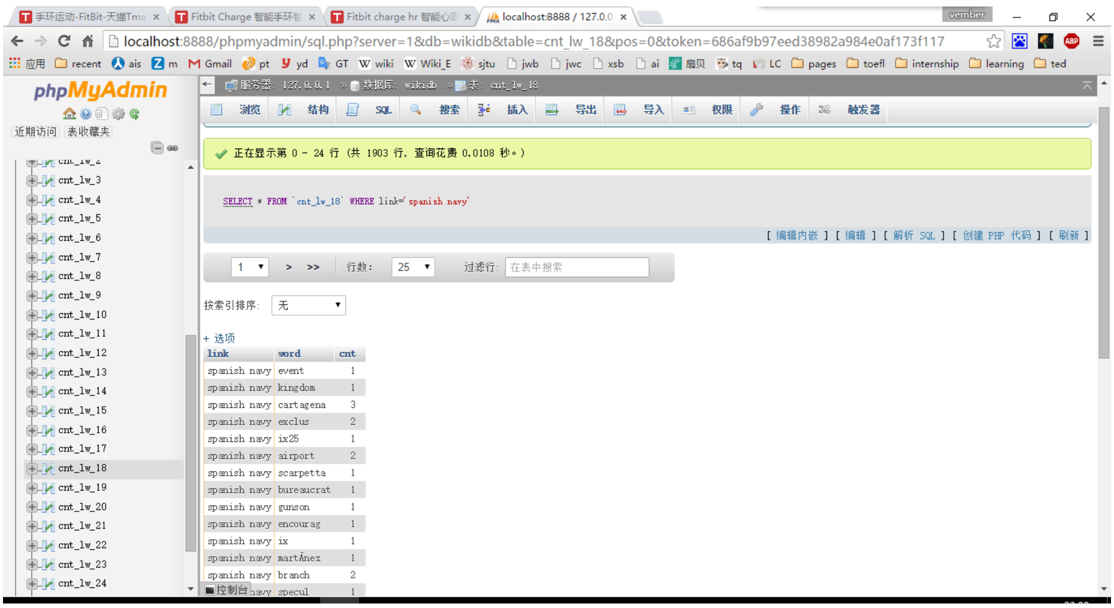
stop word or not ? -- "computing" : False
stem ? -- "computing" : comput

Search Result

Link	wiki	Word	computing
Co-occurence count	13		

5.7 Time counting (phpMyAdmin interface)





6. Conclusion

During this project we learned how to use MySQL to manage databases and how to establish a search engine. We came up with several ideas about indexing and managing the database and eventually achieved a pretty good result. Our search engine now can work really fast. It's worth mention that in this project, what we faced was a huge database, a database that can barely manage on regular PC. This gave us the big picture of how to handle a huge database. A database that a tradeoff between time and space doesn't even exist. We need to reduce both space and time cost greatly. Managing this kind of database let us know what this course is really about, and what we will encounter in the database business.