2015-12-1

Wikipedia

Database Project Final Report

李晓波 沈嘉明 王玫

5120309721 5120309726 5120309691

SECTION I: PROJECT DESCRIPTIONS & GENERAL IDEAS

In this project, we are required to first study the given data source. Then, we are asked to design and optimize a relational database for this specific dataset so that we can let this database to support three types of required queries. Finally, we should implement a simple Graphical User Interface to perform the queries and analyze the performance of our design.

In high level, we divide this project into three major parts – data preprocessing, database construction & optimization, and information retrieval using GUI.

- In the data preprocessing part, we first analyze three types of required queries. We find none of them asks us to return the exact textual content of original wiki entry. To be more concrete, the first two queries focus on the title-anchor pairs. Given the anchor word, we should return the titles with counts and vice versa. The third query is about getting the count of each word-title pair. Given the word-title pair, we should return its count. Therefore, in the data preprocessing part, we first use Mathematica Script to extract all links and delete the sentences without a link. Then we use Spark to pre-calculate the counts of all title-anchor pairs as well as word-title pairs.

- In the database construction & optimization part, we first design two schemas corresponding to two pre-cleaned datasets. After checking this set of schemas satisfying BCNF, we create a table for each schema in MySQL, and use Python to input all pre-cleaned data into two tables. Then, we optimize our design by implementing indices so that the database can return results within reasonable time.

- Finally, we write a simple Graphical User Interface based on CodeIgniter, a popular PHP framework that utilized the MVC architecture. Using this GUI, we can check the performance of our database design and demonstrate the importance of previous database optimizations.

All the technical details and experiment results are discussed in Section III.

SECTION II: DATABASE DESIGN

The given data set contains 4633303 articles in Wikipedia. So a full scan on the whole text file for each query is impossible.

Since all of the three queries deal with a pair: In the first two queries, the pair is the title and its anchor, and in the third query, the pair is a title and a word. Therefore, we do not need to store the original information in the database. Instead, we just store the pairs and their co-occurrence counts.

(2.1) THE TABLE ANCHOR_COUNT

The first two queries are based on the title and anchor pair and find the count of their co-occurrences, so we designe a table to store those pairs and the count of their co-occurrences. The schema of the relation is:

```
anchor_count(title, anchor, count)
```

where the attributes `title` and `anchor` combines the primary key of the relation. As for the implementation, we use the SQL:

```
create table anchor_count(
    title varchar(1000) binary,
    anchor varchar(1000),
    count int);
```

Note that the word “binary” here is used to specify that the attribute `title` is case-sensitive. In addition, we found that if we declare the combination of the two attributes as the primary key, it will be very slow to insert tuples into the database. So we take the strategy that we guarantee the primary key constraint when preprocessing the data, which means that the data after preprocessing do not have two tuples whose attributes title and anchor are the same respectively.

Clearly, the schema of the relation is in BCNF since the canonical cover of functional dependencies is (title, anchor) → count, and (title, anchor) is the primary key of the relation.

(2.2) THE TABLE TITLE_WORD

The third query finds the sentence-level co-occurrence count of (title, word) pairs. Similar to the strategy above, we designe a table title_word to store the pairs and their co-occurrences. The schema is:

```
title_word(title, word, count)
```

The SQL code:

```
create table title_word(

title varchar(1000) binary,

word varchar(1000) binary,

count int);
```

Similarly as above, the word “binary” here specifies that title is case-sensitive. (Note that the word here is case-sensitive, but in the actual query in GUI, both the words with the first letter in upper case and lower case will be queried if the given word is in lower case.) And the primary key constraint is also guaranteed in the preprocessing. And the relation is also in BCNF.

(2.3) INDICES

Although we can avoid a full scan on the text file by the database, the efficiency is still very low because we need to sequentially scan the whole table to execute a query. Especially in the third query, the table title_word would be even larger than the original data set. So we have to optimize the query process via indexing.

As for table anchor_count, the queries are to find titles given an anchor or find anchors given a title. So we only need to index attributes title and anchor respectively.

As for table title_word, the queries are to find the count given both a word and a title. So here we have two ideas. First, we can index attributes title and word respectively as what we do for table anchor_table. And we can also make a joint index which combines both the attribute title and word. The first idea will result in a relatively smaller index, but it would cost more time to execute a query. Oppositely, the second idea helps form a quick query but the index will be large. We will find the better by experiments.

SECTION III: EXPERIMENTS

(3.1) EXPERIMENTS ENVIRONMENTS:

- For data preprocessing step 1:

  MacBook Pro Retina (Memory: 8GB, CPU: 2.4GHz)

  Mac OS X, Mathematica 10.2

- For data preprocessing step 2:

  A Cluster of eight machines, each of which has memory 30GB, running on Ubuntu 14.04.1 LTS

- Current Server:

  Operating System: Windows Server 2012, 64bit

  Database: MySQL 5.7 Community

  Database Engine: InnoDB

  Index: B+ Tree

  Web Server: Windows Internet Information Service (IIS) 8.5

- Other tools involved in the project:

  PHP 5.6.0, Python 2.7, Coder-Integrity 3.0, phpMyAdmin 4.5.1

(3.2) DATA PREPROCESSING

The first part is data preprocessing. This part actually consists of two sequential steps – text filters and counts calculation. First, we use Mathematica to extract all links and delete all sentences without a link. We save all links into one file named “href.txt” in which each line represents one occurrence of title-anchor pair, and save all sentences with a link into another file named “sentences.txt” in which each line represents one sentence. The

original file is of size 12.55GB, and after this step, the “href.txt” is of size 3.29GB and “sentences.txt” is of size 7.12GB. Two sample results are shown below. After this step, we can get a much cleaner version of text data that facilitate our further processing. This step is done within 2 hours.

hrefs_sample.txt

File Path : ~/Desktop/Senior_Course/DBproject/Project_Upload_Materials/hrefs_sample.txt

```
<a href="Indonesian language">Indonesian</a>
<a href="Manoj Punjabi">Manoj Punjabi</a>
<a href="Dhamoo Punjabi">Dhamoo Punjabi</a>
<a href="SCTV (Indonesia)">SCTV</a>
<a href="Indosiar">Indosiar</a>
<a href="RCTI">RCTI</a>
<a href="MNC International">MNC International</a>
<a href="MNCTV">MNCTV</a>
<a href="Global TV (Indonesia)">Global TV</a>
<a href="SINDOtv">SINDOtv</a>
<a href="Media Nusantara Citra">MNC Media</a>
<a href="Media Nusantara Citra">Media Nusantara Citra</a>
<a href="MediaCorp TV12 Suria">MediaCorp TV12 Suria</a>
<a href="MediaCorp TV12">MediaCorp TV12</a>
<a href="MediaCorp TV Channel 5">MediaCorp TV Channel 5</a>
<a href="MediaCorp TV HD5">MediaCorp TV HD5</a>
<a href="MediaCorp TV12 Suria">MediaCorp TV12 Suria</a>
<a href="Astro Aruna">Astro Aruna</a>
<a href="Sensasi">Sensasi</a>
<a href="MediaCorp TV">MediaCorp TV</a>
```

sentences_sample.txt

File Path : ~/Desktop/Senior_Course/DBproject/Project_Upload_Materials/sentences_sample.txt

```
Cinta Fitri ("Fitri's Love") is an <a href="Indonesian language">Indonesian</a> soap opera with 7 seasons and 1002 episc
It was produced by headed by <a href="Manoj Punjabi">Manoj Punjabi</a> and <a href="Dhamoo Punjabi">Dhamoo Punjabi</a>
It aired at 20:30 on <a href="SCTV (Indonesia)">SCTV</a> between 2 April 2007 until 28 November 2010
The show moved to <a href="Indosiar">Indosiar</a> on 11 January 2011 until 8 May 2011 between durated 4 years how much a
<a href="MediaCorp TV12 Suria">MediaCorp TV12 Suria</a> from <a href="MediaCorp TV12">MediaCorp TV12</a> was launched te
This series was somewhat criticized when it was first aired in <a href="Indonesia">Indonesia</a>, commissioner meeting h
<a href="MediaCorp TV">MediaCorp TV</a> was programmes produced by <a href="RCTI">RCTI</a> productions were exported to
She came from a place called <a href="Wonogiri">Wonogiri</a> and lived with her aunt, as her parents had long died
Fortunately, Fitri gets a job at a <a href="Soto (food)">Soto</a> and <a href="Rice">Rice</a> food stall owned by Maya,
Lia is trapped under the debris of the hotel after an <a href="earthquake">earthquake</a> rocks <a href="Yogyakarta">Yog
Farrel is conscious but suffers <a href="amnesia">amnesia</a>
Cinta Fitri sinetron end thanks for watching is a looker collabate celebrations of anniversary is an oldest year of 4-ol
"Emerge" is an <a href="electroclash">electroclash</a> song and the first single from debut album "<a href="1 (Fischersp
The song was originally released in 2001 through <a href="International DeeJay Gigolo Records">International DeeJay Gigo
In 2003, the song was released again by Fischerspooner's new label <a href="Capitol Records">Capitol</a>.
<a href="Pitchfork Media">Pitchfork Media</a> placed "Emerge" at #100 on "The Top 100 Singles of 2000-04" and at #243 or
<a href="Resident Advisor">Resident Advisor</a> placed "Emerge" at #24 on the "Top 100 Tracks of the '00s".
In 2002, Fischerspooner performed "Emerge" on the <a href="United Kingdom">British</a> <a href="UK Singles Chart">music
The song was also featured the American TV series "<a href="Nip/Tuck">Nip/Tuck</a>" (Episode 7
Cliff Mantegna), the <a href="Canada">Canadian</a> TV series "<a href="JPod (TV series)">jPod</a>" (Episode 13
```

The previous step is actually done locally, i.e., we manually split the raw dataset into six small parts, each of which consists of 10 million lines and is of size approximately 2GB. We run the data cleaning script on each small split, and finally combine them to generate the whole “href.txt” and “sentences.txt”. However, this strategy is not appropriate for the second step of data preprocessing, i.e., counts calculation. To complete the job, we turn to Apache Spark. We write some codes in Scala and use the computing resources in the laboratory. It takes approximately 10 minutes to calculate all title-anchor co-occurrences and 100 minutes to calculate all word-title co-occurrences. The results are shown below.

res1_sample.txt

File Path : ~/Desktop/Senior_Course/DBproject/Project_Upload_Materials/res1_sample.txt

```
I Told You So (Ocean Colour Scene song)::I Told You So  1
Axwell Tiberius::Axwell Tiberius    2
Lago di Comabbio::Lago di Comabbio  1
Clientelism::Patronâ   client relations 1
cubic feet per minute::cubic feet per minute    7
artichoke::globe artichokes 2
Michael of Trebizond::Michael Komnenos  3
Battle of Northern and Eastern Henan::Northern and Eastern Honan    1
Nabunturan, Compostela Valley::Nabunturan, Compostela Valley    1
Texas–Mexican Railway::Texas–Mexican Railway    1
Bessemer Securities::Bessemer Securities    2
Stephen Huss (musician)::Stephen Huss   2
radiance::intensity 3
Black-headed Gull::black-headed 1
microbes::microbial life forms  4
Roll centre::Roll centre    1
electronic rumors::electronic rumors    3
representational state transfer::representational state transfers   1
Cyril of Turaw::Cyril of Turaw  5
Lower Sloane Street::Lower Sloane Street    3
```

res2_sample.txt

File Path : ~/Desktop/Senior_Course/DBproject/Project_Upload_Materials/res2_sample.txt

```
Jewell Building:::Theater   1
Turn state's evidence:::there   2
Bill Griffiths:::Forum  1
New York Giants:::1969 season   1
sporangium:::protected  1
ophthalmologist:::train 1
Community college#Terminology:::s   1
Super Junior-H:::half   1
Basotho:::spoke 1
Rochester Red Wings:::Pacific Coast League  2
Marseille Open:::in 15
Federal Bureau of Investigation:::1971  11
futurist:::been 6
Historical Crisis:::novella 1
High Court of Galicia:::Galician Government 1
Charles Gibson:::assist 1
levitate:::she  3
TomÃ¡Å¡ Hladlovskáµ:::Michal DobroÅ    1
Fox Lake, Wisconsin:::strip 1
Value investing:::distinctive   1
```

(3.3) DATABASE CONSTRUCTION & OPTIMIZATION

A) INSERTING DATA INTO THE DATABASE

After the preprocessing, we now have two files containing the co-occurrences which we need in the queries. We designed two Python scripts based on package MySQLdb to insert those tuples into the database. The time used to insert and the space of the tables can be seen below:

Table	Data Size after preprocessing (MB)	Time Used to Insert (Seconds)	Size of the Table (MB)	Number of Tuples
anchor_count	404	3800	701	9607877
title_word	17465	122507	33792	604968409

Table 1 Inserting the Data into the Database

B) QUERY EXPERIMENTS AND INDEXING

1. Queries without Indices

   a) Given an anchor or a title, find titles or anchors and their co-occurrence count

   The two queries use the same table. We carried out some examples, and the results can be seen below:

Given Title	Number of Found Tuples	Time (Seconds)
big data	5	17.92
Gauss	5	17.65
China	102	18.14

Table 2 The Time Performance of Finding Anchors Given Titles Without Indices

Given Anchor	Number of Found Tuples	Time (Seconds)
big data	4	19.17
Gauss	20	19.65
China	451	18.84

Table 3 The Time Performance of Finding Titles Given Anchors Without Indices

Here we can find that the time cost of all six queries are similar. The reason behind it is that there is no index here, so the database system has to carry out a sequential scan on table anchor_count to find the results. So we can draw the conclusion that a sequential scan on table anchor_count needs around 20 seconds roughly.

b) Given a title and a word pair, find the sentence level co-occurrence count

This query uses the table title_word. We carried out three queries and the results can be seen below:

Title	Word	Time
hinomaru	aircraft	12min38.3s
medicine	person	11min53.74s
Nortwest Territories general election, 1888	the	12min4.13s

Table 4 The Time Performance of Finding the Sentence-level Co-occurrence Count Given a Title and a Word without Indices

Also, the time cost is similar since it need a full scan on the whole table. And we can see that the time cost is much more than that a person can tolerate.

2. Indexing

As the analysis above, we make indices on anchor and title respectively for table anchor_count. The time cost for the two indices is 10 min 57 s. The size of the indices is 333 MB.

When we are establishing the joint index, an error occurred, which reads that the specified key was too long. As a result, we have no choice but index using the first 500 characters of titles and the first 100 characters of words. In fact, the actual length of titles for most tuples is less than 500 and the actual length of words for most tuples is less than 100, so the prefix index is also efficient. The time cost for indexing the two attributes respectively is 8 h 11 min.

The establishing of the two separate indices for title and word used 8 h 39 min. And the index size is similar to the joint index.

3. Queries with Indices

a) Given an anchor or a title, find titles or anchors and their co-occurrence count

We carry out the same queries as those without indices, and the results can be seen below:

Given Title	Number of Found Tuples	Time (Seconds)
big data	5	0.06
Gauss	5	0.05
China	102	0.8

Table 5 The Time Performance of Finding Anchors Given Titles With Indices

Given Anchor	Number of Found Tuples	Time (Seconds)
big data	4	0.07
Gauss	20	0.19
China	451	3.22

Table 6 The Time Performance of Finding Titles Given Anchors With Indices

Here we can find that the time cost grows roughly linearly with the increment of results. The reason is that the time used for fetching data makes up the most time cost.

b) Given a title and a word pair, find the sentence level co-occurrence count

As mentioned above, here we have two types of indices. So the queries are carried out in two ways.

(1) With the Joint Index

Title	Word	Time
hinomaru	aircraft	0.11 s
medicine	person	0.14 s
Nortwest Territories general election, 1888	the	0.14 s
Salmonella	made	0.18 s

We can see that the fetch time is less than 1 second, and it is acceptable enough.

(2) With the Indices for Single Attributes

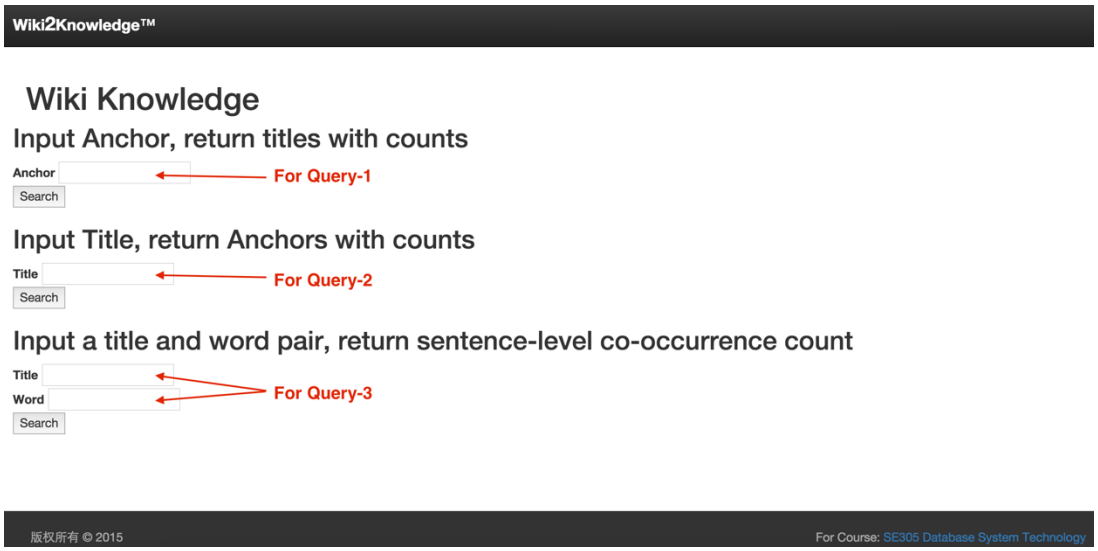
Title	Word	Time
hinomaru	aircraft	1.94 s
medicine	person	45.02 s
Nortwest Territories general election, 1888	the	1.82 s
Salmonella	made	16.84 s

Each of the queries are carried out twice where the two conditions in the where clause are switched and the time cost in the two tries is similar. We can find that the time varies and the variance is great. So this method is not quite acceptable.

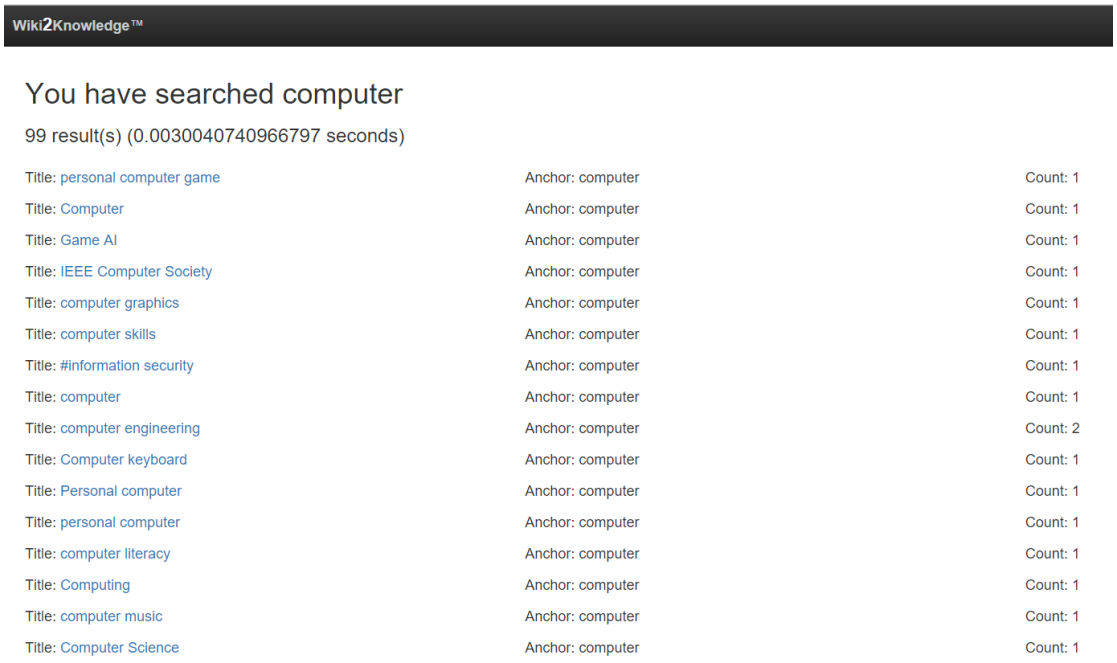
(3.4) INFORMATION RETRIEVAL FROM GUI

This graphical user interface is implemented based on CodeIgniter, a popular PHP framework adopting MVC architecture. Basically, for each type of query, we construct one model, one controller, and one view for displaying the returned results. Some screenshot of interfaces are shown below

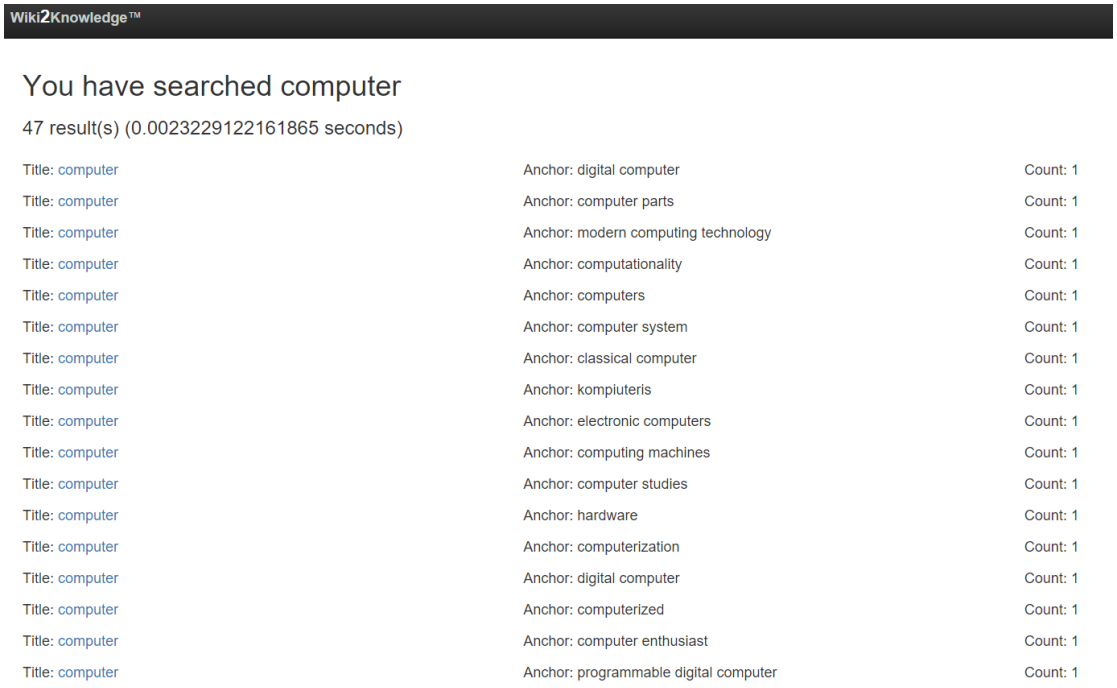
The index page:



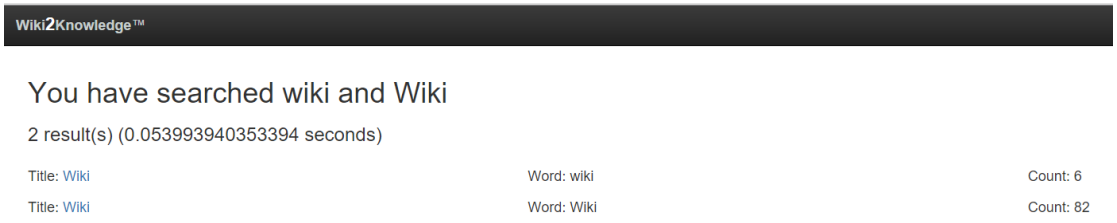
Query 1 page:



Query 2 page:



Query 3 page:



You can visit our system following the URL below:

URL: http://202.120.38.134/wiki/index.php/query1/search1

We will keep it until 15th, December.

SECTION IV：CONCLUSIONS & FUTURE WORK

(4.1) CONCLUSIONS

1. Indices and speed up queries greatly, which can be seen from the comparison of the query time with and without indices.
2. A joint index which indices more than one attribute is much more efficient for queries whose search keys are those involved in the index. This can be seen from the experiments of the third query.
3. We can decrease the size of an index by using the prefix index with proper key length according to the table. In our experiment, the size of the joint index is almost the same as the size of the two single indices for the two attributes respectively.
4. The updating operation in a database is expensive, so the proper preprocessing is better than using update operations when establishing the database. And a proper preprocessing needs a good analysis of the dataset and the queries.

(4.2) FUTURE WORK

1. The database is very big, and as a result, the index is also big. We will try to implement a multiple level index, which may speed up the query.
2. For a word in the data set, we just take it as it is in the text. However, a word may be just a form of another word. For example, in our system, “running” and “run” are thought different, however, the user may think them the same. In a future system, we may deal with the user’s input and find the derivatives of the word, then return the results.
3. The system is used for English Wikipedia. In fact, it supports other languages in database level since the character set used here is UTF-8. However, it does not support images, we may upgrade it to add support for images.