Chinese Keyword Search by Indexing in Relational Databases

Liang Zhu¹, Lijuan Pan¹, Qin Ma²

¹Key Laboratory of Machine Learning and Computational Intelligence, School of Mathematics and Computer Science, Hebei University, Baoding, Hebei 071002, China; ²Department of Foreign Language Teaching and Research, Hebei University, Baoding, Hebei 071002, China.

Email: zhu@hbu.edu.cn (L. Zhu), katharine162@126.com(L. Pan), maqin@hbu.edu.cn(Q. Ma)

Received October 15, 2012.

ABSTRACT

In this paper, we propose a new method based on index to realize IR-style Chinese keyword search with ranking strategies in relational databases. This method creates an index by using the related information of tuple words and presents a ranking strategy in terms of the nature of Chinese words. For a Chinese keyword query, the index is used to match query search words and the tuple words in index quickly, and to compute similarities between the query and tuples by the ranking strategy, and then the set of identifiers of candidate tuples is generated. Thus, we retrieve top-N results of the query using SQL selection statements and output the ranked answers according to the similarities. The experimental results show that our method is efficient and effective.

Keywords: Relational Database; Chinese Keyword Search; Index; Ranking Strategy

1. Introduction

For a database system, keyword search with the general-purpose query engine uses user-supplied data to query the contents of string properties that store keywords, and then requires users to have the knowledge of database schema and a query language (say, SQL). Inspired by the success of free-form keyword search on information retrieval (IR) and Web search engines, i.e., it is popular to users who need not know query languages and the structure of underlying data. Researches of English keyword search with IR-style free-form in relational databases have been extensively studied since 2002[1-7]. [1] and [2] join tuples from multiple relations in the database to identify tuple trees with all the query keywords, for each enumerated join tree, both of them simply rank join sequences according to the number of joins. ObjectRank system [3] applies authority-based ranking to keyword search in database modeled as labeled graph. [4] proposed a method (G-KS) for selecting the top-N candidates based on their potential to contain results for a given query. [5] succeeded in putting the model of computing similarities in IR into computing similarities between a candidate answers and tuples in relational databases, the methods pay more attention on effectiveness of keyword search. [6] proposed a middleware free approach to compute such m-keyword queries on RDBMSs using SQL only. In this paper, we will discuss Chinese keyword search in a relational database based on an index.

Chinese is totally different from English. For instance, (1) Chinese words are tighter in writing, unlike English; there is no space between words. (2) Chinese words are coded by using GB2312-80 and each Chinese word is stored in two consecutive bytes in memory, while an English word is composed of letter(s) and each letter takes one byte, say, the Chinese word "人" (means "person") is 0xC8CB with two bytes in memory; however, English word “person” takes 6 bytes. (3) Abbreviations in English are acronyms, such as “WWW” is short for “World Wide Web”; however, abbreviations in Chinese are extracted words from a phrase, say, “高代” means “高等代数” (Advanced Algebra).

Example 1. As shown in Figure 1, database BOOKS has three relations/tables: Titles(tid, title, Faid, Fpid, ...), Authors(aid, name,...), and Publishers(pid, pname,...). Without loss of generality, we suppose the relationship between Authors and Titles is “one to many”, not “many to many”. We regard all authors of a book together as one author, since the keyword search will be processed by each Chinese word and the index will be created by using the text attributes of Entities (say, title, name,
not using the id’s in relationships. Moreover, the 1-to-n type relationship may be more feasible in practice in our scenarios (say, a string of Chinese words will be concatenation of names of two or more authors if there is an error of typing). For query $Q = \{\text{高代; 高教社}\}$, a user want to get the answers of “title = 高等代数, author = 高代(如果 any), publisher = 高等教育出版社 (Higher Education Press)”. Traditional DBMS cannot obtain such results, and it is difficult for us to use directly the existing methods of English keyword search models in Chinese keyword query.

**Figure 1. Part of BOOKS database**

Inspired by the technique of creating index of tuple words in [8], we present a new method to perform Chinese keyword search by indexing. This work is a continuation of [9], which studied Chinese keyword search with only one relation; however, this paper discusses a relational database with multiple relations, and it is more challenging than the work in [9].

We will build an index based on the information of tuple words and improve the classic ranking strategy in IR. For a Chinese keyword query, its top-N answers will be obtained by the index and the improved ranking strategy.

### 2. Data Model and Query Model

Consider a database with $n$ relations $R_1, \ldots, R_n$. Each relation $R_i$ has $m_i$ text attributes $A'_{i1}, A'_{i2}, \ldots, A'_{im_i}$, a primary key and possibly foreign key(s) referencing other relation(s).

**Definition 1 (Schema Graph) [2, 4]**: A directed graph captures the primary key-foreign key relationships in the schema of the database. It has a node for each relation $R_i$ of the database and an edge $R_i \rightarrow R_j$ for each foreign key to primary key relationship from a set of attributes $(A'_{i1}, \ldots, A'_{ik})$ of $R_i$ to a set of attributes $(A'_{j1}, \ldots, A'_{jk})$ of $R_j$, where $A'_{ik} \equiv A'_{jk}$ for $k=1, \ldots, t$.

**Definition 2 (Tuple Tree) [2, 5]** A tuple tree $T$ is a joining tree of tuples. Each node $t_i$ in $T$ is a tuple in the base relation $R_i$. For each pair of adjacent tuples $t_i, t_j \in T$, where $t_i \in R_i, t_j \in R_j$, there is an edge $R_i \rightarrow R_j$ and $t_i \bowtie t_j \in R_i \bowtie R_j$.

**Definition 3 (Tuple Word):** For each tuple $t$ belongs to $R_i$ and $A \in \{A'_{i1}, A'_{i2}, \ldots, A'_{im_i}\}$, $t[A]$ is the attribute value on attribute $A$ of relation $R_i$, it contains single or multiple Chinese words. We define every single Chinese word as a Chinese tuple word.

**Definition 4 (Index Table):** An Index Table is composed of tuple words and their related information extracted from the database, its schema is $\text{TupleWordTable}$ (wordid, word, size, $DBValue$), where wordid is the primary key, word is the tuple word, size is the number of text attributes that contain the corresponding tuple word, $DBValue$ is a text attribute with form “$cid, df, tid, tf, dl; cid, df, tid, tf, dl; \ldots$”. In $DBValue$, $cid$ is the identifier of the attribute (or column) containing the tuple word, $df$ is the number of cells containing the tuple word in certain attribute (or column), $tid$ is the identifier of the tuple containing the tuple word, $tf$ is the number of tuple word appears in the cell determined by $tid$ and $cid$, and $dl$ is the total number of words (counting duplicate words) in the cell determined by tuple identifier $tid$ and attribute/column identifier $cid$.

**Definition 5 (Keyword Query):** A query $Q = (k_1, k_2, \ldots, k_p)$ is a set of Chinese tuple words. The results of the keyword query are the tuple trees joined by relations. The results are ranked by a ranking strategy, and then the top-N ones will be the desired answers of a user.

### 3. Construction of Index Table

In this section, we will describe how to create an index table with information of tuple words, including the design and implementation of the index.

#### 3.1. Design of Index Table

An index table which is designed as relation with schema $\text{TupleWordTable}$ (wordid, word, size, $DBValue$) is constructed to store the information of each single tuple word. Tuple words in tuples of relation $R_i (1 \leq i \leq n)$ are extracted from $R_i$ and are stored into the index table. For a query, we invoke an index to implement the search and display the top-N answers ranked by a ranking strategy. Therefore, it is important to decide information granularity of tuple words stored in index table. In this paper, we consider column level of granularity as well as cell level. The answers for a query are tuple trees joined by different attributes, and the nature of different attributes will affect the effectiveness of the keyword query, thus,
attributes information in database need to be recorded individually. Cell granularity is more detailed which will create more accurate similarities between query and results. In relation TupleTableWord, DBValue is used to store column level and cell level information.

3.2. Implementation of Index

The process of creating index table includes three steps: (1) Normalize tuples in \( R_i \), \( i=1, 2, ..., n \), remove useless characters and Chinese punctuation which will obstruct the processing of extracting Chinese words. (2) For each tuple word belongs to \( t[A] \), extract related information of tuple word. (3) Use the information to create index table.

\[
\text{IndexTableCreationAlgorithm} \ (R_1, R_2, ..., R_n) \ 
\begin{align*}
0. \ & \text{For each relation } R_i \ in \ \{R_1, R_2, ..., R_n\} \\
1. \ & \text{For each attribute } A_j \ in \ \{A_1, A_2, ..., A_m\} \ of \ R_i \\
2. \ & \text{For the value of each tuple } t \ on \ attribute \ A_j, \ (t[A_j]) \\
3. \ & \text{For each tuple word } z \ in \ t[A_j] = \{z_1, z_2, ..., z_i\} \\
4. \ & \{ \text{If } z \ has \ not \ been \ saved \ in \ word-list} \\
\quad \quad \{ \text{add a new node } W_i \ in \ word-list \ to \ save \ z, \ and \}
\end{align*}
\]

sort the nodes in word-list by the code of GB212-80; create a new Col-list pointed by \( W_i \) and add a new node \( D_z \) into the Col-list, to save column identifier (cid) and document frequency (df); create a new tl-list pointed by \( D_z \), add a new node \( T_z \) into tl-list to save tuple identifier(tid), tuple frequency(tf) and data length(dl);

5. Else
\{ Return node \( W_i \) containing \( z \), search the Col-list pointed by \( W_i \); \}

6. If Col-list has the node \( D_z \) corresponding to \( z.cid \) \{ update df, search the tl-list pointed by \( D_z \); \}

7. If the tl-list has node \( T_z \) with \( z.tid \), update \( tf; \) Else add a new node of tl-list to save \( cid, tf, dl; \)

8. Else
\{ add a new node of Col-list to save \( cid \) and \( df; \) add a new node of tl-list to save \( tid, tf, dl; \}
\} // end else (8)\}

As an example, a part of index table is shown in Figure 3 for database BOOKS in our experiment. The index table contains 4506 tuple words including most level 1 and level 2 Chinese words.

![Figure 2. Structure of the index](image)

Figure 2. Structure of the index

The structure of the index is shown in Figure 2. It has a Hash-Table and three layers of linked lists. The first layer is the Word-list which stores tuple words. Node \( W_i \) \( i=1, 2, ..., d \) in Word-list corresponds to a single tuple word and \( wordid \) in the index table, it also has a pointer pointing to a Col-list which is the second layer linked list. Each node in a Col-list saves the information that correlates to the tuple word appeared in a certain attribute (or column). Different node in the Col-list of \( W_i \) indicates that \( W_i \) turns up in different attribute. Likewise, one node in the third layer tl-list stores the related information of \( W_i \) of a specific tuple. The algorithm of creating the index table is described below:

![Figure 3. A part of the index table for BOOKS database](image)

Figure 3. A part of the index table for BOOKS database

In order to implement the keyword search, we need to load the index table into memory, and match the keywords with tuple words. For a huge index table, however, it is hard to load the whole index table into the memory. Therefore, we need to compress and to improve the index. Firstly, we remove the stop words which are meaningless words such as “的” and “和”, and the words appeared in specific attribute of most tuples, say, “出版社”, “出版社”, and “出版社” in pname attribute of relation Publishers, more than 95% of tuples contain these words. Secondly, it is necessary for the index table to shrink its structure. We only load the Hash-Table and Word-list into memo-
4. Chinese Keyword Search

4.1. Generation of identifiers of candidate tuples

For the query \( Q = (k_1, k_2, ..., k_p) \), we match query words \( k_i, \) (\( i = 1, 2, ..., p \)) with tuple words by using the index in memory, and obtain the set of \( DBValue \) of tuple words. We divide \( DBValue \) into pieces \((cid, df, tid, tf, dl)\).

A tuple is solely identified by its \( tid \), so we collect \( tid \) from every piece and obtain the set of all identifiers of basic tuples matched with query words \( k_i \), denote by \( R^t_i = \{t \mid t = tid\} \), where \( tid \) means the identifier of tuple \( t \). Combining all sets \( R^t_i \) (\( i = 1, 2, ..., p \)), we get \( R^Q = R^t_1 \cup R^t_2 \cup ... \cup R^t_n \) (\( j=1, 2, ..., n \)), which are the sets of all identifiers of candidate tuples containing query words in relation \( R_t \). In the process of collecting \( tid \), we record the number of distinct query words of each tuple. For the tuple, the more distinct query words contained in a tuple, the closer of it gets to \( Q \). For example, a given query \( Q = \{k_1, k_2, k_3\} \), tuple \( t_1 \) contains one \( k_1 \) and one \( k_2 \), \( t_2 \) contains two \( k_2 \)’s, \( t_3 \) contains one \( k_1 \), one \( k_2 \) and one \( k_3 \). Thus, \( t_1 \) is the best match for \( Q \), and \( t_1 \) is much closer to \( Q \) than \( t_2 \) according to the similarities between \( Q \) and \( t_1, t_2, t_3 \), which can be obtained as follows.

4.2. Ranking Strategy

For each candidate tuple that contains query words, we extract the information from piece and calculate the similarity between the query \( Q \) and the tuple. In this paper, based on vector space model widely used in IR ranking strategy, we improve a classic method \([10, 11]\) to compute the similarities between the query \( Q \) and tuple trees. Query words and tuple trees are represented as a vector of terms, and each term may be an individual word or a multi-word phrase. The vocabulary of terms makes up a term space, and each term occupies a dimension in the space. Each element of vector is non-negative weight that measures the importance of the term in the text. The similarity is:

\[
Sim(Q, T) = n*c + \sum_{k \in Q} Sim(k, T) \times weight(k, Q) \tag{1}
\]

\[
Sim(k, T) = \sum_{j \in T} weight(k, A_j) \times w_i \tag{2}
\]

\[
weight(k, A_j) = \frac{1 + \ln(1 + \ln(tf))}{(1-s) + s \times \frac{dl}{avdl}} \times \frac{N + 1}{df} \tag{3}
\]

Equation (1) and (3) are derived from \([9, 10]\), In Equation (1), \( weight(k, Q) \) is the appearance frequency of word \( k \) in query \( Q \), tuple tree \( T \) is composed of attributes \( \{A_1, A_2, ..., A_n\} \), \( n*c \) is a new item for our Chinese key-

word query, \( c \) is constant (\( c = 10 \) by training in our experiments), and \( n \) is the number of different tuple words contained in \( T \). In Equation (3), \( s \) is a constant (usually set to 0.2), \( N \) is the number of tuples corresponding to the attribute, and \( avdl \) is the average number of words in tuples corresponding to the attribute. For a query, we define Equation (2) due to the weights (or important factors) of different attributes. For example, when buying a book, the title of a book is usually more important than its publisher; therefore, the weight of attribute Titles: title should be higher than that of attribute Publishers: name. Sorting the weights of different attributes, we get the set \( W = \{w_1, w_2, ..., w_n\} \) and \( w_i \geq w_j \) (\( i \neq j \)). According to Equation (1) and (3), if the values of \( tf, df \) and \( dl \) are equivalent, larger weights will lead to higher similarities, and the tuple trees with larger weights will rank higher.

Suppose \( M \) is the maximum value of the number of distinct query words in a candidate tuple. After obtaining \( R^Q_t \), we compute firstly the tuples that owns \( M \) distinct query words, then \( M-1, M-2 \), down to one query words. According to the number of the distinct query words (i.e., \( n \)), we achieve the subsets \( S_i \)'s of results for the query, and then the set \( S \) of all results is \( S = S_1 \cup S_2 \cup ... \cup S_k \), \( S_i \cap S_j = \emptyset (i \neq j) \), where \( S_i \) is the collection of tuple trees whose numbers of distinct query words are \( i \). In general, tuple trees in \( S_j \) have higher similarities than those in \( S_i \) if \( i < j \).

4.3. Answers for queries

We utilize the schema graph and \( R^Q_j \) (\( j = 1, 2, ..., n \)) to construct SQL selection statements, and then retrieve tuple tree from the database. Query conditions of SQL statements have the relationship of foreign key-primary key and the location of query words. For example, if the schema graph of database is \( R_1 \rightarrow R_2 \rightarrow R_3 \), for a query \( Q = \{k_1, k_2\} \), SQL statements are as below:

\[
Select * from R_1, R_2, R_3 where R_1.Pid = R_2.Fid and R_3.Pid = R_2.Fid and R_3.Pid in R_3^Q \tag{S1}
\]

\[
Select * from R_1, R_2, R_3 where R_1.Pid = R_2.Fid and R_3.Pid = R_3.Pid in R_3^Q \tag{S2}
\]

\[
Select * from R_1, R_2, R_3 where R_1.Pid = R_2.Fid and R_3.Pid = R_2.Fid and R_3.Pid in R_3^Q \tag{S3}
\]

Where \( Pid \) and \( Fid \) denote Primary key and Foreign key respectively. If multiple relations (or multiple attributes) contain the query words that are in the same returned tuple trees, the above SQL statements may lead to redundant search results. In order to avoid redundancy, it is necessary to reduce the repeat selection: Let \( R_1^Q = R_1^Q - R_1^Q \) after (S1) is executed, and \( R_2^Q = R_2^Q - R_2^Q - R_1^Q \) after (S1) and (S2) are executed, where \( R_1^Q \) is the identifier set
of tuples both appear in $R_1^Q$ and $R_2^Q$, and so as to $R_2^Q$ and $R_3^Q$.

Example 1 (cont.) For query $Q = \{\text{高代}; \text{高教社}\}$ submitted by the user, a part of answers with "title = 高等代数, publisher = 高等教育出版社" are shown in Figure 4 and are ranked by their similarities, where the three results with id’s 58734, 58709 and 58735 have the same similarities.

<table>
<thead>
<tr>
<th>id</th>
<th>title</th>
<th>author</th>
<th>publisher</th>
</tr>
</thead>
<tbody>
<tr>
<td>49339</td>
<td>高等代数</td>
<td>北京大学数学系</td>
<td>高等教育出版社</td>
</tr>
<tr>
<td>58760</td>
<td>高等代数简明教程</td>
<td>徐彩晨</td>
<td>高等教育出版社</td>
</tr>
<tr>
<td>58709</td>
<td>高等代数教程</td>
<td>刘亚平</td>
<td>高等教育出版社</td>
</tr>
<tr>
<td>58735</td>
<td>高等代数.下册</td>
<td>高等代数出版社</td>
<td></td>
</tr>
<tr>
<td>58734</td>
<td>高等代数.上册</td>
<td>高等代数出版社</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4. Part results of query “高代; 高教社”.

5. Experiments

Our experiments are carried out using Microsoft’s SQL Server 2000 and VC++ 6.0 on a PC with Windows XP, Intel(R) Core2 Duo 2.0 GHz CPU, and 2.0GB memory.

The real dataset comes from the library of Hebei University, which is a fragment of the data of Chinese books. As shown in Figure 1, our database BOOKS has three relations: Titles(tid, title, Faid, Fpid), Authors(aid, aname), Publishers(pid, pname), tid, aid, pid are the primary keys for three relations respectively, and Faid, Fpid are foreign keys of Titles referencing Authors.aid and Publishers.pid respectively. The relation Titles contains 87762 tuples, Authors contains 62120 tuples and Publishers contains 2995 tuples.

The parameters that we vary in the experiments are the number of query words and the number of results $N$ requested in Top-$N$ queries. We consider 10 groups queries, where keywords are randomly chosen from the attribute word of TupleWordTable. We denote 10 groups by $G_i(i=1, 2, ..., 10)$, each $G_i$ contains 10 queries and the number of keywords of each query in $G_i$ was $i$. The 100 queries are used to measure the time and accuracy of our method.

(1) Time. For the 10 groups of keyword queries, we record the running time of matching the index table (denoted by Index-Time) and the time of returning results (denoted by Result-Time) respectively. As shown in Figure 5, Result-time is between 50 and 400 milliseconds, Index-time are between 0 and 100 milliseconds. Generally, Result-time costs more than Index-time, for it requires more time to rank similarities and I/O costs frequently.

When the number of query words comes to 4 and 9 in Figure 3, Result-time turns up to peak values, it is obvious that Result-time is larger than Index-time, the reason is as follows: some query words are contained in a large amount of tuples, like query words “中 迷 清 坚” in group $G_4$, the number of tuples which contain any single word is 17637, while the general number of tuples is about several thousands. We have to calculate the similarity of every single tuple and then rank these tuples by our ranking strategy.

Figure 5. Elapsed time for keyword queries

(2) Recall and precision. For $N=3, 10, 20, 50, 80$ and 100, Figure 6 and Figure 7 show recalls and precisions of Top-$N$ results of keyword search respectively.

Figure 6. Recalls of Top-$N$ results

Figure 7. Precisions of Top-$N$ results

In Figure 6, for each $G_i$, with the increase of $N$, recall will become larger. The reason is that the total number of desired results in the database is constant, while the
number of matching tuple in the Top-\(N\) results will increase as \(N\) becomes larger. For a fixed \(N\), the overall rate of recall will decrease as the number of query words increases. In Figure 7, for each \(G_i\), with the increase \(N\), precision will decrease. When \(N \geq 80\), the average rates of recalls and precisions are higher than 50% and 60% respectively. In addition, according to requirements, all ranked results may be displayed for a query.

6. Conclusions
In this paper, we proposed a new method to realize IR-style free-form Chinese keyword search over relational databases. The basic idea of this method is to create an index by extracting information from relations in a database. For a given query, we use the index to obtain the candidate tuples and calculate the similarity of between the query and each candidate tuple through improved ranking strategy. The Top-\(N\) results are retrieved by SQL selection statements for the natural join of relations in the database. Extensive experiments were carried out to measure the performance of our method based on a real dataset. Experimental results show that the average elapsed time including Index-time and Result-time is less than 500 milliseconds for queries with 1 to 10 query words. When \(N \geq 80\), the average recalls and precisions are higher than 50% and 60% respectively.

7. Acknowledgements
This work is supported in part by NSFC (61170039) and the NSF of Hebei Province (F2012201006).

REFERENCES


