Browsing in Relational Databases using Keyword Search

Phyo Thu Thu Khine, Khin Nwe Ni Tun
University of Compute Studies, Yangon
phyothuthukhine@gmail.com, knntun@gmail.com

Abstract

With the great success of web search engines, keyword search has become the most well-liked search model for users. Information retrieval systems have popularized for searching unstructured data by keywords as a query and results are sorted relevant documents. It has become highly desirable to offer users with flexible ways to query/search information over databases as simple as keyword search like Google search. Thus, there is a need to introduce a keyword searching compatibility in relational database search. We, in this paper introduce the mechanism of scoring results, followed by filtering/ranking the records, and finally browsing the joinable tuples contain all keywords in the query. The proposed system is found to score than the existing systems in the retrieval efficiency in the experimentation.

1. Introduction

In the recent years, there have been great deals of research and development activities on extending keyword search capabilities to handle relational data, the dominant form in which business data are stored. The database research community has recently recognized the benefits of keyword search and has been introducing keyword search capability into relational databases [2, 4, 14], XML databases [10, 18], graph databases [13, 17], and heterogeneous data sources [3].

It has become highly desirable to provide flexible ways for users to search information by integrating database (DB) and information retrieval (IR) techniques in the same platform. The sophisticated DB facilities provided by a database management system assist users to query well-structured information using a query language based on database schemas. Such systems include conventional RDBMSs (such as DB2, ORACLE, SQL-Server), which use SQL (Structured Query Language) to query relational databases (RDBs). But IR techniques allow users to search unstructured information using keywords, and they do not need users to understand any database schemas. That is simply and user-friendly [8].

If database users could search databases in the same way, database queries would be simple without knowing database schema and query languages. However, keyword search techniques on the Web cannot directly be used on data stored in databases. Applying keyword search techniques in text databases (IR) to relational databases (DB) is a challenging task because these two types of databases are different. In text databases, the basic information units searched by users are documents. For a given keyword query, IR systems compute a numeric score for each document and rank the documents by this score. The top ranked documents are returned as answers. In relational databases, due to database normalization, logical units of information may be fragmented and scattered across several physical tables. Given a set of keywords, a matching result may need to be obtained by joining several tables [1].

In this paper, we focus on search capability on relational databases. The proposed system

searches the relevant records, and ranks them on their score to the query. It enables data browsing together with keyword searching. The contributions of this paper are summarized as follows:

- The scoring model is presented to rank the retrieved tuples by assigning each tuple with a score.
- The record filtering mechanism is proposed to filter the most relevant records and rank these with their score.
- In order to find the joinable tuples that contain all keywords in the query, the algorithm of Joinable Relation Matrix (JRM) is proposed by traversing the database schema graph.

The rest of this paper is organized as follows: Section 2 summarizes the related works. In Section 3, we briefly introduce the overview of the system. Section 4 explains how we analyze the challenges for browsing in relational databases and proposes a scoring, ranking strategy. In Section 5, we present experimental results to demonstrate the efficiency of our proposed system. Section 6 draws the conclusion.

2. Related Work

There are different approaches for keyword search in RDBs. Here we just introduce DISCOVER [15, 16], BANKS [2], and DBXplorer [14], share a similar approach. At query time, given a set of keywords, first find tuples in each relation that contain at least one of the keywords, usually using database system auxiliary full text indexes. Instead of these, the index table is created for all relations as a preprocessing stage in the proposed system. In this way, it takes a little index construction time as well, but query time is faster. The above three systems [2, 14, 15, 16] use graph-based approaches to find tuples among those from the previous step that can be joined together, such that the joined tuple contains all keywords in the query. All three systems use foreign-key relationships as edges in the graph, and point out that their approach could be extended to more general join conditions. A main shortage of the three systems is that they spend a plenty of time to find the candidate tuples that can be joined together. But in our system, only the candidate answer sets are constructed for the ranked and filtered records. The consumption of querying time can be reduced in the proposed system.

Saint (Structure-Aware INdexing for finding and ranking Tuple units) [5] proposes a structure-aware index based method to integrate multiple related tuple units to effectively answer keyword queries. The structural relationships between different tuple units are discovered and stored them into structure-aware indices, and progressively found the top-k answers using such indices.

ITREKS (Indexing Tuple Relationship for Efficient Keyword Search) [9] supports efficient keyword-based search over relational database by indexing tuple relationship: A basic database tuple relationship, FDJT, is established in advance and then a FDJTTuple-Index table is created, which records relationships between each tuple and FDJT. At query time, for each of keywords, system first finds tuples in every relation that contain it, using full text indexes offered by database management system. Then FDJT-Tuple-Index table is used to find the joinable tuples contain all keywords in the query. These two systems create the index and also relationships of the tuples in advance. The proposed system creates the tuples index as preprocessing step and the tuple relationship is discovered in the browsing step.

The IR systems proposed theories and practice in ranking methods for documents. The previous work has presented and used the metrics that contribute to more effective ranking method for the search results, e.g., join tree size [7], normalized node prestige and edge weight

[2], PageRank node ranking [2], shortest distance between keywords [17], etc. Of these, only [15, 16] considered combining some of the above factors with the IR-style ranking functions. [7] used this function but they do not handle queries with more than 2 solution paths. In Mragyati [11], the ranking function can be based on user-specified criteria but the default ranking is based on the number of foreign-key constraints and. In contrast, we compute scores to the resulted tuples which contain all keywords. The results are filtered and ranked according to their scores to get the most relevant results.

3. Overview of System

Given a set of query keywords, the system returns the rows (either from single tables, or by joining tables connected by foreign-key) such that the each row contains all keywords. This system involves (a) preprocessing step called Indexing that facilitates databases by building the Index Table and (b) answer generation step that retrieves the ranked tuples to the user.

Id	First_name	Last_name	Gender
1	Michael	babeepower Viera	M
2	Eloy	Chincheta	M
3	Dieguito	El Cigala	M
4	Antonio	El de Chipiona	M
5	Marcial	El Jalisco	M

(a) Actors

Id	First_name	Last_name
1	Todd	1
2	Lejaren	aHiller
3	Nian	A
4	Khairiya	A-Mansour
5	Ricardo	A. Solla

(b) Directors

Id	Name	Year	Genre
1	At First Sight	2000	Short
2	Angel on My Shoulder	2005	Drama
3	Broadway: The Next Generation	2005	Documentary
4	Diamond Dead	2005	Musical
5	Flash Gordon	2006	Action

(c) Movies

Director_id	Movie_id
1	30621
2	60570
3	63525
4	118137
5	39392

(d) Movies_Directors

Actor_id	Movie_id
1	135644
2	12083
3	101866
4	12148
5	80189

(e) Roles

Figure 1. Sample IMDB Database Instances

The sample database instances of IMDB (Internet Movie Database) are shown in figure 1.

3.1. Overview of Indexing and Answer Generation

Indexing: It is done through the following steps. A particular database is identified to generate the index to speed up the retrieval of records. It finds the text attributes in tuple (t) in each table. After finding this, the set of indices is defined. A set of attributes are found in table that can be joined with these indices and create Map Table. Map Table is then used at search time to efficiently determine the locations of query keywords in the database.

Generating Answers: Given a query consisting of a set of keywords, it is answered as follows.

- The map table is looked up to identify the tables, and columns/rows of the database that contain the query keywords.
- Each query result is assigned a relevance score.
- In order to get the most relevant record, the resulted records are filtered with the threshold value.
- If the resulted tuples have the related information (connected tuples in the schema), all the connected tuples are enumerated. For each enumerated

candidate answer sets, a SQL statement is constructed (and executed) that joins the tables in the answer sets. Then the final results are presented to the user.

Although there are two steps, we discuss only the search step of the proposed system in this paper. The mechanism of indexing database and indexing algorithm is presented in [12].

4. Answer Generation

In this section, we discuss the generation of answers using keyword search. Let Q = {k_1, k_2,…, k_n} be the keywords in a query. Recall from Section 3.1 that the answer generation step has four steps. In the first step, the user query is cleaned and then the map table is looked to retrieve the tuples that contain at least one of the keywords in the query. This process finds a set of tuples {ti} (hits) which matches the keywords in the user query. To do this, keyword matching algorithm is presented. This step has been proposed in the paper [13]. We omit this step because of the lack of space. The next three steps are scoring the results, filtering/ranking these results and that of enumerating join tuples that are described in detail below.

4.1. Results Scoring

When the query results are produced, calculation of the score for each result is needed. In a keyword query, each query result is assigned a relevance score. This process is to determine which records are more relevant to the user query than the others. As more than one result may match any keyword query, each result is assigned with a score and ranked the list of results according to their scores. The effectiveness of the scoring and ranking functions is an important aspect of keyword search. The result scoring algorithm is proposed to calculate the score of the results. We compute the score for each result produced from the previous searching phase as the following equation.

$$score^k = \frac{count(f(k))}{n(k)} \quad (1)$$

n(k) = the number of keywords in user query

count(f(k)) = the number of keyword occurance in a result tuple

$score^k$ = the score of each relevant record

4.2. Results Filtering

In many application domains, end-users are more interested in the relevant answers in the potentially huge answer space. Different emerging applications warrant efficient support for record filtering. In our system, the threshold value is considered to filter the resulted records.

The threshold value is defined as follows:

$$T = \frac{max(score) + min(score)}{2} \quad (2)$$

For each keyword occurance, the ranked record is defined as following.

$$If\ score^k > T\ then\ R(k) \leftarrow score^k$$

$$otherwise\ remove\ from\ keyword\ list$$

The results which scores are less than the threshold value are filtered. Finally, these query results are ordered into a sorted list so that users can focus on the top ones, which are hopefully the most relevant ones. The top results in the ranked list are more relevant to the query than those at the bottom.

4.3. Browsing of Results

This system also provides a facility to browse the records stored in a relational database. Enabling keyword search on databases that does not require knowledge of the schema is a challenging task. One cannot apply techniques from the documents world to databases in a straightforward manner. Due to database normalization, logical units of information may

be fragmented and scattered across several physical tables. Given a set of keywords, a matching result may need to be obtained by joining several relations in the database. In order to find the joinable tuples that contain all keywords in the query, we have to generate the relation set (RS) which are composed of tuples connected by primary and foreign-key relationships through the schema graph. To do this, we proposed the algorithm of computing joinable matrix by traversing over the schema graph shown in figure 2.

Algorithm: Computing Joinable Relation Matrix

Input: Database Schema Graph SG, set of root relations RS (table names), set of Primary Key PRS from each relation in RS.
Output: Joinable Relation Matrix JRS[R][C]
R=Numbers of Primary Key in PRS
C =Numbers of Table Names in RS
JRS[R][C]← {0}

For each Primary Key (Pi∈ PRS) in PRS do
  For each relation (Rj∈ RS) in RS do
    Find the key relation Pi in Rj of Schema Graph
    If exist then JRS[i][j] = 1;
    Else JRS [i][j] = 0;
    End If
  End For
End For
  Return Joinable Relation Matrix JRS

We firstly discover the table R_j which have primary-foreign-key relationships with table R_i. If found the relation R_j, then we identify other tables which have primary-foreign key relationships with that intermediate relation R_j. Iteratively, for each relation $R_i \in R$ ($1 \le i \le m$), we can identify the relation set RS through joining the connected relations.

The schema graph consists of all the relations inside a database, the relationship among these relations, and the distance among relations. The default distance between two related tables is considered to be one. In the schema graph shown in figure 2, an arrow (edge) represents the connection between the relations/tables and each node represents the relation/table name.

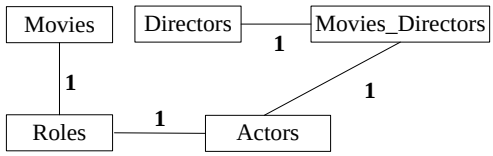


Figure 2. Schema Graph for IMDB Database

After computing according to the algorithm we get the Joinable Relation Matrix.

Joinable Relation Matrix

Relation / RelationId	Actors	Directors	Movies	Roles	Movies_ Directors
Actor_id				1	
Director_id					1
Movies_id				1	1

Table 1. Traversal for each relation in IMDb

Table Name	Depth-first order
Actors	Actors, Roles, Movies, Movies_Directors, Directors
Directors	Directors, Movies_Directors, Movies, Roles, Actors
Movies	Movies, Movies_Directors, Directors
	Movies, Roles, Actors

For example, for the query "The Next Generation and Bryan Rick", firstly, the candidate answers are constructed for each record in the filtered sets by using the relation schema graph. Movies table is only included in

the filtered records. The candidate answer sets for Movies table are:

Candidate Answer Sets for Movies
Movies → Movies_Directors → Directors
Movies → Roles → Actors

For Candidate Answer 1 and Movie_id = 3

```
SELECT   Movies.Name, Movies.Genre,
         Movies.Year,  Directors.First_name,
         Directors.Last_name
FROM     Movies, Directors, Movies_Directors
WHERE    Movies_Directors.Director_id=
         Directors.id
AND      Movies_Directors.movie_id=
         Movies.id
AND      Movies.Movie_id = 3;
```

For Candidate Answer 2 and Movie_id = 3

```
SELECT   Movies.Name, Movies.Genre,
         Movies.Year,Actors.First_name,
         Actors.Last_name
FROM     Movies, Actors, Roles
WHERE    Roles.Actor_id = Actors.id
AND      Roles.movie_id = Movies.id
AND      Movies.Movie_id = 3;
```

For example: user query is "**The Next Generation and Bryan Rick**"

Browsed results

Movie: Broadway: The Next Generation
Genre: Documentary
Year: 2005
Actor: Jason (I) Alexander
Director: Rick McKay and etc.

The proposed system demonstrates the browsed data from the relational databases with tree view.

5. Experiments

This system is implemented with Pentium Core 2 Dual 2.0GHz processor, 1GB of RAM. We evaluated the performance of the proposed system on two different datasets with different characteristics.

- DBLP[19]: Digital Bibliography and Library Project dataset, which has 881,867 tuples containing information on publications, authors, titles, and publishers.
- IMDB[20]: Internet Movie Database contains information on movies, actors, directors and so on. It has 763692 tuples.

Figure 3 and 4 shows that the comparison of query processing time based on the number of keywords between the proposed method, ITREKS and Saint (Structure-Aware INdexing for finding and ranking Tuple units) for DBLP and IMDB datasets.

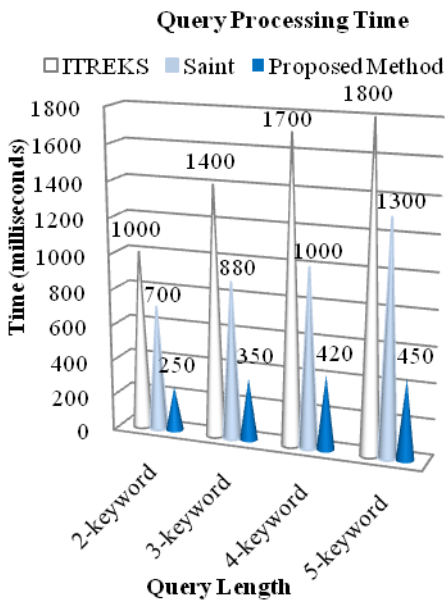


Figure 3. DBLP Query Performance

These two systems based on the indexing approach but they developed with the difference

trends compared to our approach. We use the number of keywords (query length) from 2 to 5 words. The keywords were selected randomly from the underlying database. We generated 50 queries for each query length.

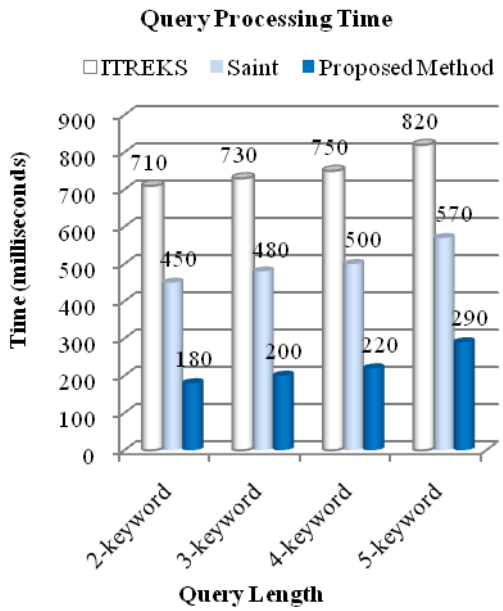


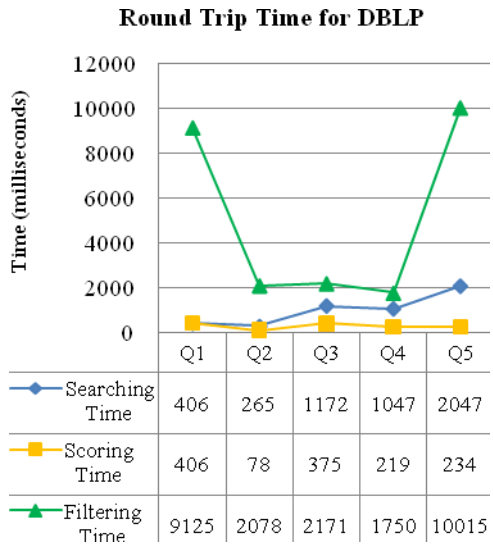
Figure 4. IMDB Query Performance

Figure 3 and 4 illustrate that the proposed algorithm achieves much higher search efficiency than ITREKS and Saint. Although the query length (number of keyword) increases, query execution time doesn't increase sharply between the individual queries. It shows that our algorithm can answer such queries efficiently. We use the tuple-aware indexing method to identify the answers through our proposed Map Table, and thus the proposed method can significantly improve the search efficiency.

To show the performance of round trip time, we used ten different queries that are listed in Table 2 for two dataset. For each query contains a list of different keywords. The keyword length (number of keyword in each query) is not the same.

Table 2. Queries employed in experiments

QueryID	Queries
Queries on DBLP	
Q1	Information retrieval database
Q2	IR database
Q3	DB Retrieval System
Q4	XML relational keyword search
Q5	Data mining algorithm 2006
Queries on IMDB	
Q6	Lethal Weapon 4 academic
Q7	Police Academy 3 customer
Q8	Halloween 5 college
Q9	Love 45 tradesman
Q10	Robocop 3 college



	Q1	Q2	Q3	Q4	Q5
Searching Time	406	265	1172	1047	2047
Scoring Time	406	78	375	219	234
Filtering Time	9125	2078	2171	1750	10015

Figure 5. Round Trip Time for DBLP

The round-trip time of the proposed system consists of three components: searching time, result score computing time, and relevant result filtering time. Different queries have different round trip time. Figure 5 is only the round trip time result of the proposed system tested on DBLP dataset and figure 6 is the round trip time for IMDB dataset.

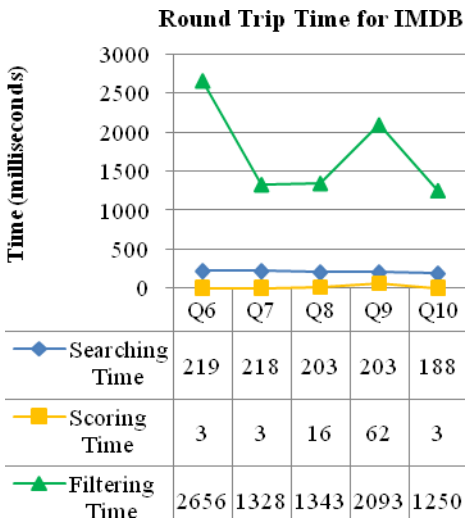


Figure 6. Round Trip Time for IMDB

5. Conclusion

The mechanism for keyword search system over relational databases is proposed. Experimental results show that the proposed methodology is efficient for searching of the user queries. The proposed algorithms can retrieve the most relevant results in a short time that match the user's needs efficiently.

References

[1] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective Keyword Search in Relational Databases", SIGMOD, 2006, pp. 563-574.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti, "Keyword Searching and Browsing in Databases using BANKS", ICDE, 2002, pp. 431-440.

[3] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou, "EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data", ACM SIGMOD, 2008, pp. 903–914.

[4] G. Li, J. Feng, and L. Zhou, "Progressive ranking for efficient keyword search over relational databases", BNCOD, 2008, pp. 193-197.

[5] G. Li, J. Feng, and J. Wang, "Structure-Aware Indexing for Keyword Search in Databases", CIKM, 2009, pp. 1453-1456.

[6] H. He, H. Wang, J. Yang, and P. S. Yu, "BLINKS: ranked keyword searches on graphs", ACM SIGMOD, 2007, pp. 305–316.

[7] J. Saelee, and V. Boonjing, "A Metadata Search Approach with Branch and Bound Algorithm to Keyword Search in Relational Databases", ICCIT, 2009, pp. 571-576.

[8] J. X. Yu, L. Qin, and L. Chang, *Keyword Search in Databases*, Synthesis Lectures on Data Management, Morgan & Claypool, 2010.

[9] J. Zhan, and S. Wang, "ITREKS: Keyword Search over Relational Database by Indexing Tuple Relationship", DASFAA, 2007, pp.67-78.

[10] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram, "XRANK: ranked keyword search over XML documents", ACM SIGMOD, 2003, pp. 16–27.

[11] N. L. Sarda, and A. Jain, "Mragyati: A System for Keyword-based Searching in Databases", TR CoRR cs.DB, 2001.

[12] P. T. T. Khine, and K. N. N. Tun, "Indexing Relational Databases for Efficient Keyword Search", International Journal of Scientific & Engineering Research (IJSER), Volume 2, Issue 10, October, 2011.

[13] P. T. T. Khine, and K. N. N. Tun, "Keyword Searching and Browsing System over Relational Databases", IEEE ICDIM, 2011, pp. 121-126.

[14] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Databases", ICDE, 2002, pp. 5-16.

[15] V. Hristidis, and Y. Papakonstantinou, "Discover: Keyword Search in Relational Databases", VLDB, 2002, pp. 670-681.

[16] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-Style Keyword Search Over Relational Databases", VLDB, 2003, pp. 850-861.

[17] V. Kacholia, S. Pandit, A. Chakrabarti, S. Sudarhan, R. Desai, and H. Karambelkar, "Biderectional Expansion for Keyword Search on Graph Databases", VLDB, 2005, pp. 505-516.

[18] Y. Xu, and Y. Papakonstantinou, "Efficient LCA based Keyword Search in XML Data", EDBT, 2008, pp. 535-546.

[19] http://dblp.uni-trier.de/xml

[20] http://www.imdb.com