Improving Keyword Searches Through Multistage Processing

Phyo Thu Thu Khine, Htwe Pa Pa Win, Khin Nwe Ni Tun
University of Computer Studies, Yangon
Myanmar
phyothuthukhine,hppwucsy,knntun@gmail.com

ABSTRACT: *End users resort to varied ways of text searching in text; however most of them rely on key word approach and hence in the Information Retrieval research, it is often provided priority. Data is being stored in relational databases where keyword based search has difficulties. Thus there is a need to introduce a keyword searching compatibility in relational database search. We in this paper introduce the way of indexing the records, followed by key word matching and finally the distilling result key words. Our system is found to score than the existing systems in the retrieval efficiency in the experimentation.*

Keywords: Text Mining, Keyword Indexing, Data Processing, Word Processing

Received: 15 February 2011, Revised 1 March 2011, Accepted 4 March 2011

1. Introduction

Users in the information retrieval systems found to use different keys and parameters, but key word search is prominent in terms of numbers and popularity. As key words express concepts and the volume of informaton pieces is very high, the key word search is gaining momentum. Database query systems search structured data with structured query language. The users need to know the data schema and query language. But Information retrieval systems have popularized for searching unstructured data by keywords as a query and results are sorted relevant documents. That is simply and user-friendly. If database users could query databases in the same way, database query 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. In databases, the information is viewed as data tables and their relationships, and query results may be a single tuple or joining tuples. Accordingly, the challenge is how to apply keyword-based search to find sorted relevant results in databases.

In this paper, we propose keyword-based searching and browsing system where the user doesn't need the knowledge of database schema or SQL (Structured Query Language).Instead of that, they submit a list of keywords. The system then searches for the relevant records, and ranks them on their relevancy to the query. The system enables data browsing together with keyword searching.

Adopting this keyword search on relational databases gives immediate advantages. First, more meaningful results can be returned by enabling searching for keyword matches across relation boundaries. This is particularly meaningful in relational databases as normalization is commonly used that decompose input data into several individual tuples and stored separately.

Second, it lowers the access barrier for average users.

This paper is organized as follows: Section 2 summarizes current related work. Section 3 shows the proposed model. Section 4 presents the system architecture for our proposed system and explains in details. Section 5 shows experimental results and section 6 is the conclusion.

2. Related Work

Keyword-based search over relational database gets much attention recently. The early three systems DISCOVER [7, 8], BANKS [1], and DBXplorer [6], 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, index construction time is longer as well, but query time is faster.

ITREKS (Indexing Tuple Relationship for Efficient Keyword Search) [12] 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. Saint (Structure-Aware INdexing for finding and ranking Tuple units) [17] 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. These two systems create the index and also relationships of the tuples in advance. Our proposed creates only the tuples index but not find relationship. The tuple relationship is found in the browsing step.

The above three systems [1, 6, 7, 8] 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 answers tree is constructed for the ranked and filtered records. The consumption of querying time can be reduced in the proposed system.

The IR community 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., normalized node prestige and edge weight [1], PageRank node ranking [1], shortest distance between keywords [2], join tree size [6], etc. Of these, only [7, 8] considered combining some of the above factors with the IR-style ranking functions. In Mragyati [3], the ranking function can be based on user-specified criteria but the default ranking is based on the number of foreign-key constraints and [4] used this function but they do not handle queries with more than 2 solution paths. 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. Proposed System Model

3.1 Database Model

Our system models the database as an undirected graph.The schema graph G_s (V, E), is an undirected graph that captures the primary key to foreign key relationships in the database schema. G_s has a node in V for each relation R_i of the database and an edge $R_i \rightarrow R_j$ in E for each primary key in R_i to foreign key in R_j relationships. Figure 5 shows the schema graph of DBLP (Digital Bibliography & Library Project) used in this paper. The sample database instances from DBLP are shown in Figure 1. In the next session, the detail processing for searching from DBLP relational database tables are described.

3.2 System Architecture

The proposed system architecture for keyword search over relational databases is shown in Figure 2. Given a set of query keywords, the proposed system returns all the results (the set of joinable tuples between relations connected by primary-key, foreign-key relationships). It involves five phases: (a) indexing relational database, (b) keyword-based searching (c) scoring the result, (d) filtering the relevant record and (e) browsing the resulted records in tree view.

Proceeding Id	Proceeding Title	EditorId	Publisher Id	SeriesId	Year	ISBN
1	Recent advances in Development and Use of B Method	1	1	1	1998	3-540-64405-9
2	Machine Learning and Its Application	42	1	1	2001	3-540-42490-3
3	Mathematical Models for the Semantics of Parallelism	44	1	1	1987	3-540-18419-8

(a) Proceeding

InProceedingId	InProceedingTitle	Page	ProceedingId
1	Graphical Design of Reactive Systems	197	1
2	Optimization Networks	156	262
3	Layering Distributed Algorithms with B method	260	1
4	From Object-Oriented to Aspect-Oriented Databases	134	971

(b) InProceeding

SeriesId	SeriesTitle
1	Lecture Notes in Computer
2	IFIP Transactions
3	IFIP Conference Proceedings
4	LNI

(c) Series

PublisherId	PublisherName
1	Springer
2	Elsevier
3	North Holland
4	ACM
5	IEEE Computer Society

(d) Publisher

PersonId	Name
1	Didier Bert
2	Emil Sekerinski
3	Jean-Marc Meynadier

(e) Person

PersonId	InProceedingId
1	34854
1	34888
2	3910

(f) RelationPersonInproceeding

Figure 1. Sample Database Instances

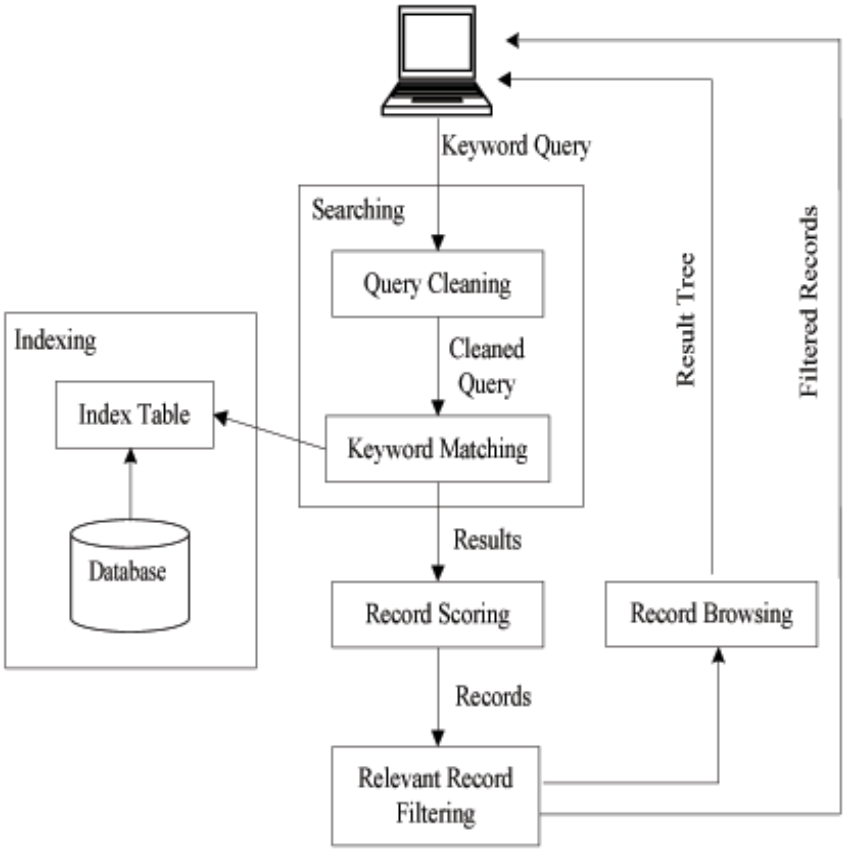

Figure 2. Architecture of proposed system

3.2.1 Indexing Relational Database

Indexing is used to speedup the retrieval of records. This approach can be useful when the database has large number of Text (Varchar) fields. Each value in such a field can be considered as a small text document that can be used for keyword-based search. To gain the speed benefits of indexing at retrieval time, the indexing algorithm is proposed and an index table is built by using this algorithm in advance. The records in the relational tables is found to be indexed and mapped them with postings (relation, tuple id) based on inverted indexing technique. A fragment of Index Table is generated by using indexing algorithm shown in Figure 3.

3.2.2 Keyword-based Searching

After indexing, the system is ready for searching process. Given a user's query consisting of a set of keywords, the input query is cleaned and matched with IndexTable.

Query Cleaning: This phase takes a user entered query as an input, and produces a "clean" query output. This is achieved by filtering the stopwords from the query. These words may appear many times in the tuple, but they are meaningless. If they are not filtered, the answers with them maybe returned with great priority and this will not satisfy the users.

Keyword Matching: Once the cleaned query is produced, this system can match the keywords. This process finds a set of tuples TS $\{t_i\}$ (hits) which matches all the keywords in the user query. To do this, keyword matching algorithm is presented. The IndexTable is used to retrieve which tuples contain keywords and to construct tuple subsets.

For example, when user enters query ("ACM Springer"), the following tuple sets are retrieved for each keyword. The resulted tuples that match the user's query are found from only Publisher and Person table. Figure 4 shows a sample result of a query containing the keywords ACM and Springer executed on the bibliographic database.

Indexing Relational Database Algorithm

Input: A database D, database relations R_1, R_2…R_n
Output: An IndexTable IT
Begin

1. Scan database D and read the names of relations R_1,R_2...R_n.
2. For each relation R_i (i =1 to n) in D do
3. List S_i ← Read structure of relation R_i
4. Prefix ← R_i (substring (3))
5. For j = 0 to (size of S_i) -1
6. fn_j ← field name of S_i[]
7. dt_j ← data type of S_i[]
8. If (dt_j == “text”)
9. For each row r_k (k =1 to (length of R_i)) do
10. keyword ← values of fn_j in r_k
11. mapId ← Prefix + “R” + rowId of r_k
12. keyArray[k-1] ← keyword
13. mapArray[k-1] ← mapId
14. tempkey[k-1] ← keyword
15. End If
16. If (j >1 && dt_j == (“text”))
17. keyword ← keyword + tempkey[k]
18. End If
19. End for
20. End for
21. For m = 0 to (length of keyArray) -1
22. Insert into IndexTable values (keyArray[m], mapArray[m])
23. End for

End

Keyword	MapId
Graphical Design of Reactive Systems. 182-197	inpR1
Well Defined B. 29-45	inpR2
Didier Bert	perR1
B$198: Recent Advances in the Development and Use of the B Method, Second International B Conference, Montpellier, France, April 22-24, 1998, Proceedings 1998 3-540-64405-9 http://dblp.unitrier. de/db/conf/b/b1998.html	proR1
Springer	pubR1
Lecture Notes in Computer Science http://dblp.unitrier. de/db/journals/lncs.html	serR1

Figure 3. IndexTable for keywords

Keyword Matching Algorithm

```
Input: Cleaned keyword CK[ ], IndexTable IT, database D
Output: All database rows matching all keywords
Begin
1.  Separate by space to CK.
2.  mq: keyword matching query
3.  Generate mq.
4.  mq ← "SELECT keyword, mapId FROM  IndexTable
    WHERE MATCH (keyword) AGAINST (' "
5.  If (length of CK == 1) then
6.          mq ← mq + CK[0] + " ')"
7.  Else
8.      For i = 0 to (length of CK) -1
9.          tokenCK[i]
10.         If (i == length of CK -1)
11.             mq ← mq + token + " ')"
12.         Else
13.             mq ← mq + token + " ') " + " OR MATCH
                (keyword) AGAINST (' "
14.         End if
15.     End for
16.  Create related database connection.
17.  Create conditional query using mq.
18.  Assign mq to database connection.
19.  Execute mq on IT.
20.  Retrieve matched keywords.
21. End
```

Keyword	MapId
Springer	pubR1
Springer and British Computer Society	pubR26
ACM/Springer	pubR65
Thomas Springer	perR13597
George Springer	perR21066
Springer W. Cox	perR30916
Johannes Springer	perR39588
A. Springer	perR42805
Clayton Springer	perR50902
Stephen Springer	perR82706
Paul L. Springer	perR98750
Gordon K. Springer	perR106836
D. L. Springer	perR131731
Joseph F. Springer	perR132914

Figure 4. Search result of query "ACM and springer"

3.2.3 Record Scoring

When the query results are produced, calculation of the score for each result is needed. 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 record scoring algorithm is proposed to calculate the score of the results.

Record Scoring Algorithm

```
Input :( Matched keywords, mapId) ArrayListMK,
        cleaned keywords query CK
Output: Scored Records SR
Begin
1.    tkw: total keyword count
2.    ckw:cleaned keywords
3.    qkw : length of CK
4.     For i = 0 to (length of MK) - 1
5.          searchRecords SR[i] ← MK[i]
6.          For j = 0 to (length of CK)-1
7.               ckw ← CK[j]
8.               If (SR[j] has ckw)
9.                    tkw ++
10.              End if
11.       End for
12.       If (tkw!= 0)
13.            Score ← tkw/length of CK
14.           SR ← MK[i], Score
15.       End If
16. End for
End
```

After computing the score, the relevant records are found in Table 1.

3.2.4 Relevance Record 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 order to get the most relevant records, the record filtering algorithm is proposed. In our system, the threshold value is considered to filter the resulted records. The results which scores are less than the threshold value are filtered. The top results in the ranked list are more relevant to the query than those at the bottom.

3.2.5 Record Browsing

This system also provides a facility to browse the records stored in a relational database. When the system browses the related information with the resulted tuple set in order to find the relationship of input keywords, the candidate answer sets need to be constructed through the schema graph. A schema graph consists of all the tables inside a database and the relationship among these tables, and the distance among tables. The default distance between two related tables is considered to be one. In the schema graph, the arrows represent a relation between database tables and each node represents the table name.

Figure 6 shows the schema graph for each relation (table) of sample database schema shown in Figure 5. The relationships for each node are reordered with depth-first order. The root node for each schema tree has to pass at every time for browsing the table relations.

Relevant Records	Total keywordin each record (tkw)	Cleankeyword from user query (qkw)	Score
Springer	1	2	0.5
Springer and British Computer Society	1	2	0.5
ACM Springer	2	2	1
Thomas Springer	1	2	0.5
George Springer	1	2	0.5
Springer W.Cox	1	2	0.5
Johannes Springer	1	2	0.5
A.Springer	1	2	0.5
Clayton Springer	1	2	0.5
Stephen Springer	1	2	0.5
Paul L. Springer	1	2	0.5
Gordon K. Springer	1	2	0.5
D. L. Springer	1	2	0.5
Joseph F. Springer	1	2	0.5

Table 1. Scored Records for the query ("ACM Springer")

Relevant Record Filtering Algorithm

Input : Scored Records SR, Score S

Output: Filtered Records FR, Filtered Sort Records FSR, distance d

Begin

1. MaxRank : maximum score of S
2. MinRank : minimum score of S
3. RangeVal : average value of S
4. RangeVal ← (MaxRank + MinRank)/2
5. For i = 0 to (length of SR)-1
6. If (S_i >=RangeVal)
7. FR ← SR[i]
8. End if
9. End for
10. For j = 0 to (length of FR) -1
11. distance ← Score of FR[j] / length of FR[j]
12. End for
13. FSR ← sort (FR, distance)

End

For example, for the query "ACM Springer", firstly, the candidate answers are constructed for each record in the filtered and ranked record sets by using the forward and backward traversal of schema graph. The Publisher and Person tables are only included in the filtered records (as shown in section 4.2). The primary key PublisherId from Publisher exists as foreign key in Proceeding Table. The PersonId also situated as foreign key in RelationPersonInProceeding table. This backward traversal for the query "ACM Springer" is :

Candidate Answer Sets
Publisher → Proceeding → InProceeding → RelationPersonInProceeding → Person
Person → RelationPersonInProceeding → InProceeding → Proceeding → Series
Person → RelationPersonInProceeding → InProceeding → Proceeding → Publisher

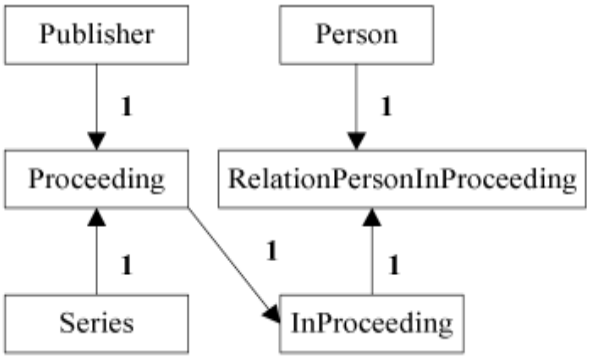

Figure 5. Schema Graph for Database

SELECT Proceeding.Title, Proceeding.Url
FROM Proceeding, Publisher
WHERE Proceeding.PublisherId = Publisher.PublisherId
AND Publisher.PublisherId= 1; and etc.

There is no forward search in Publisher and Person table hence no foreign keys are included. The outputs of this step, the answers are browsed with tree view.

4. Experimental Results

To evaluate the performance of the system, 92.1 MB of DBLP dataset is used. The system is implemented with Pentium Dual-Core 2.0GHz processor and 1GB of RAM. The system decomposed the datasets into 6 relations according to the schema shown in Figure 5. Table 3 summarizes the 6 DBLP relations

Relation Name	**Tuples**
Publisher	81
Inproceeding	208,086
Series	24
Proceeding	2,749
Person	162,907
RelationPersonInProceeding	491,777
Total	1,357,401

Table 2. DBLP Dataset Characteristics

It shows that if the record size of each relation is increased, the indexing time for this relation is increased. The total indexing time for all relation takes only 1766 milliseconds (1.8 seconds). It shows that the proposed indexing mechanism can reduce the

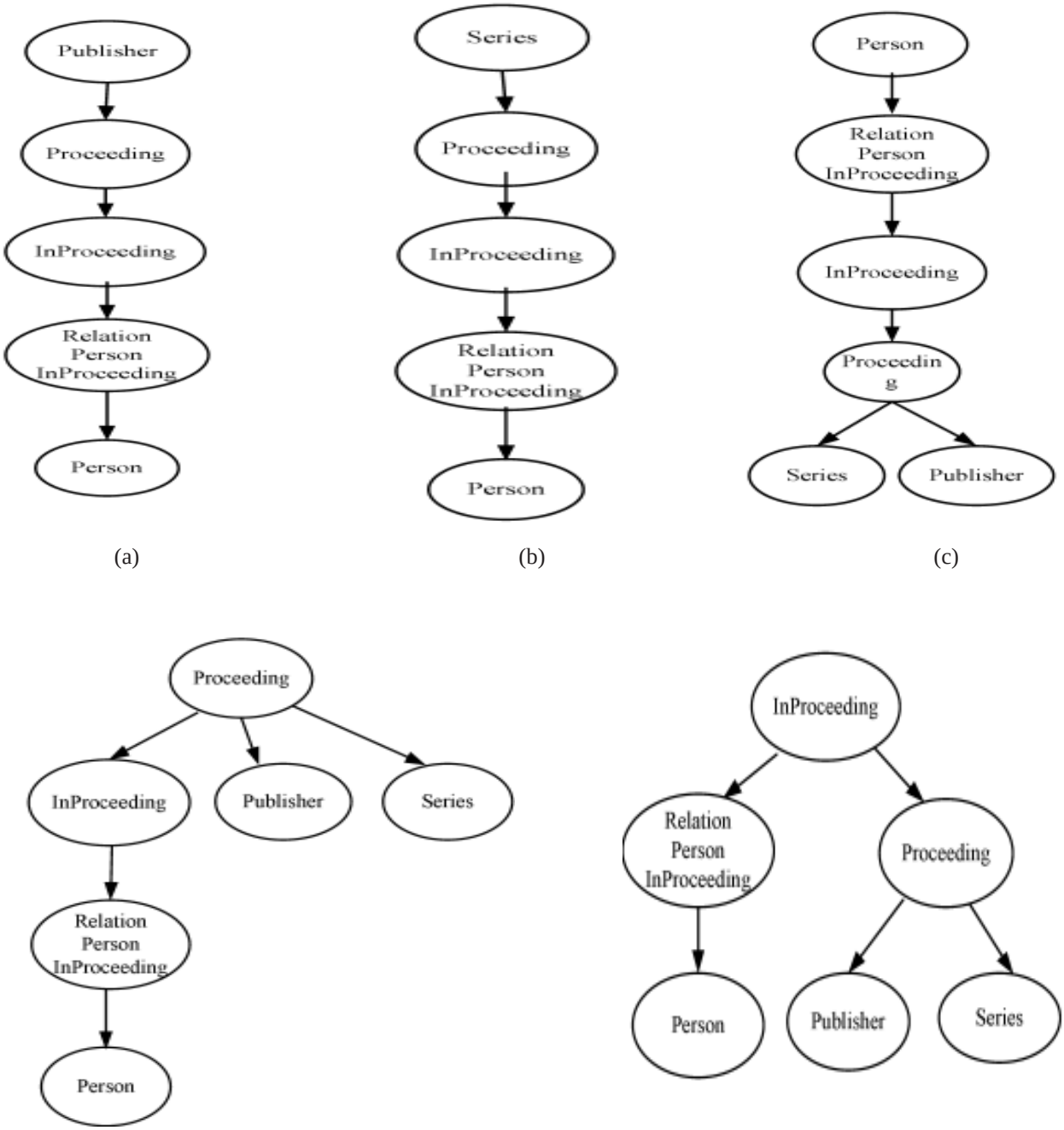

Figure 6. Schema Graph for (a) Publisher Table (b) Series Table
(c) Person Table (d) Proceeding Table (e) InProceeding Table

indexing time significantly.

Figure 8 shows the comparison of query processing time between the proposed method and ITREKS (Indexing Tuple Relationship for Efficient Keyword Search) by using 10 randomly generated different queries with query length 3 to 10. It shows that the proposed mechanism can reduce the query processing time compared with ITREKS.

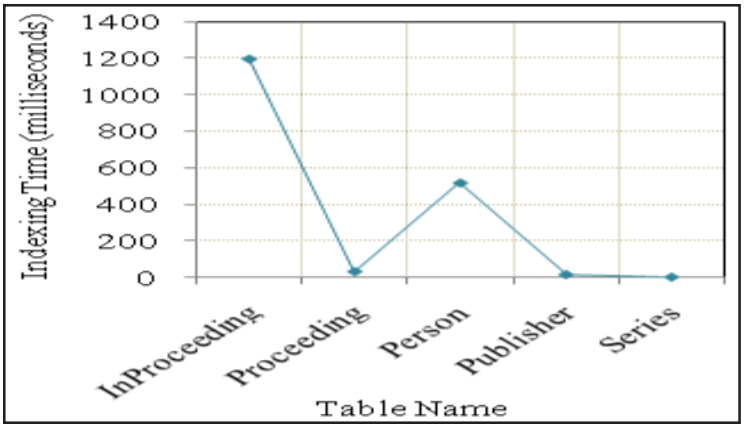

Figure 7. Indexing Time for DBLP database

Figure 9 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). These two systems used the indexing technique but they developed with the difference trend 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.

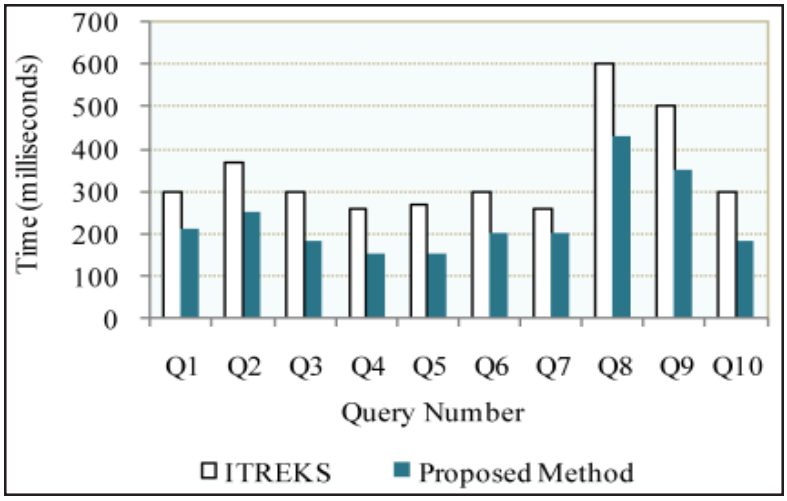

Figure 8. Comparison of Indexing Mechanism

Figure 9 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). These two systems used the indexing technique but they developed with the difference trend 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.

It shows 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 Index Table, and thus the proposed method can significantly improve the search efficiency.

Figure 10 shows the round trip time for twenty queries listed in Table 3 on the DBLP dataset [15]. The round-trip time of the proposed system consists of three components: record searching time, record scoring time, and relevant record filtering time.

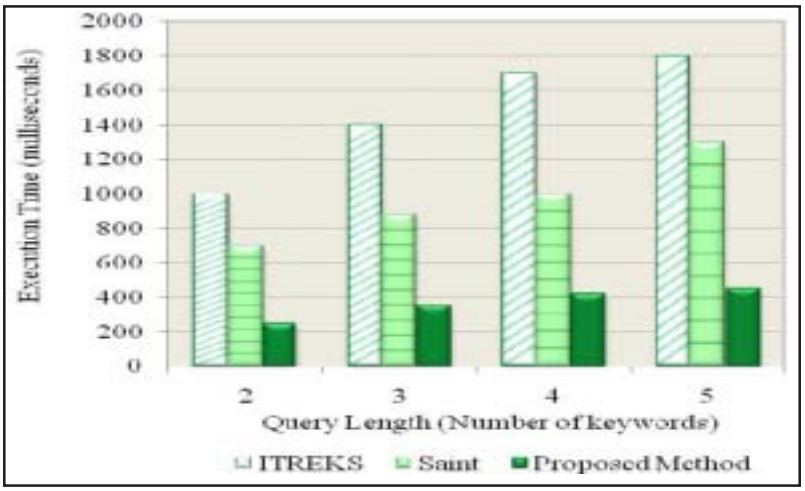

Figure 9. Scalability (DBLP)

Different queries have different round trip time. This figure only shows the round trip time of the proposed system.

Query No	Keyword Query
Q1	Mobile Radio System Markus Anja Klein
Q2	Dynamic Cell Planning for Data Transmission Gfeller Weiss
Q3	Cryptographic Primitives for Information Authentication
Q4	Real Time Protocols Lann
Q5	An Algebraic Specification of Process Algebra Sjouke Mauw
Q6	Broadband Communications Services Zhili Sun
Q7	Database Horlait principle
Q8	Nelson networks notes Pires Weber
Q9	Daniel Thalmann algorithm science
Q10	Laurent Dairaine Elsevier notes algorithm
Q11	Multimedia Traffic Control Cleevely North-Holland
Q12	query optimization Cornelius Frankenfeld Elsevier Lecture
Q13	Computer Science paul Yoon Springer
Q14	Multimedia Mail Service Prototype
Q15	Robust Reconstruction Daniel Thalmann Elsevier
Q16	Linear Algebraic Greene North science
Q17	Lecture Notes Anna Stefani Elsevier Traffic Control
Q18	Multimedia Document Architecture
Q19	introduction network Hewson Springer lecture
Q20	Nelson Multimedia Thalmann Services

Table 3. Query Used For The Test [15]

5. Conclusion

The proposed system allows users with no knowledge of database system or schema to query and browse relational database

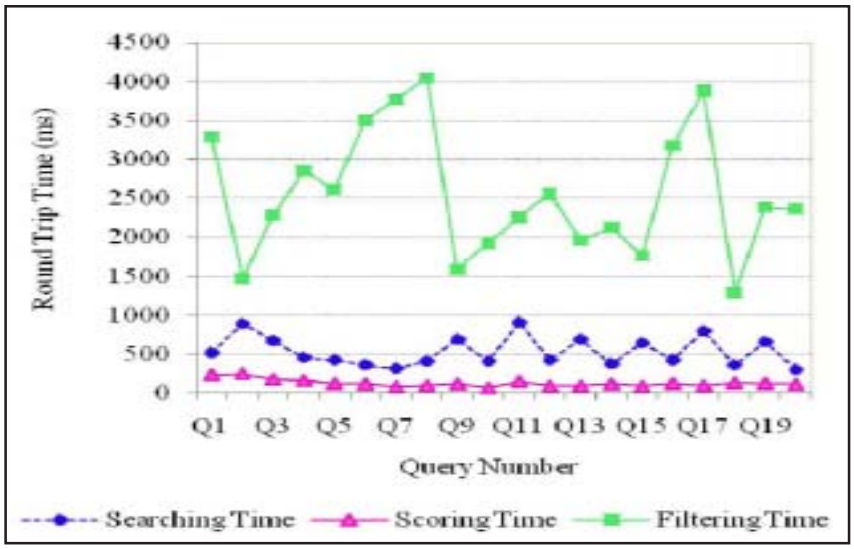

Figure 10. Round-trip time for different queries

with ease. Indexing and keyword matching algorithms are proposed to find queries efficiently. The proposed system can reduce a waste of time during searching records. The answer to such a query consists of the ranked tuples which potentially include tuples from multiple relations. To calculate the score of the resulted tuples, the record scoring algorithm is presented. Moreover, we proposed an algorithm to filter many results so that the system can provide the most relevant query results to the user.

References

[1] Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, A. (2002). Keyword Searching and Browsing in Databases using BANKS, ICDE, p. 431-440.

[2] Kacholia, V., Pandit, Chakrabarti, A., Sudarhan, S., Desai, R.,Karambelkar, H. (2005). Biderectional Expansion for Keyword Search on Graph Database, VLDB, VLDB Endowment, p. 505-516.

[3] Sarda, N.L., Jain, A. (2002). Mragyati: A System for Keyword-based Searching in Databases, TR CoRR cs.DB.

[4] Saelee, J., Boonjing, V. (2009). A Metadata Search Approach with Branch and Bound Algorithm to Keyword Search in Relational Databases, ICCIT, p. 571-576.

[5] Balmin, A., Hristidis, V., Papakonstantinou, Y. (2004).ObjectRank Authority-Based Keyword Search in Databases, VLDB, VLDB Endowment, p. 564 -575.

[6] Agrawal, S., Chaudhuri, S., Das, G. (2002). DBXplorer: A System for Keyword-Based Search over Relational Databases, ICDE, p.5-16.

[7] Hristidis, V., Papakonstantinou, Y. (2002). Discover: Keyword Search in Relational Databases, VLDB, VLDB Endowment, p. 670- 681.

[8] Hristidis, V., Gravano, L., Papakonstantinou, Y. (2003). Efficient IRStyle Keyword Search Over Relational Databases, VLDB, VLDB Endowment, p. 850-861.

[9] Liu, F., Yu, C.T., Meng, W., Chowdhury, A. (2006). Effective keyword search in relational databases. *In:* SIGMOD, p. 563–574.

[10] Manning, C.D., Raghavan, P., Schütze, H. (2009). An Introduction to Information Retrieval, Cambridge University Press, p. 569.

[11] Su, Q., Widom, J. (2005). Indexing Relational Database Content Offline for Efficient Keyword-Based Search, IDEAS, 25-27 July, p. 297-306.

[12] Zhan, J., Wang, S. (2007). ITREKS: Keyword Search over Relational Database by Indexing Tuple Relationship, 12th *International Conference on Database Systems for Advance Applications* (DASFAA).

[13] Xu, Y., Ishikawa, Y., Guan, J. (2009). Effective top-k keyword search in relational databases considering query semantics, *In:* Proc. APWeb-WAIM International Workshops, V. 5731/2009 of LNCS, p.172–184.

[14] Qin, L.,Yu, J, X., Chang, L. (2009). Keyword search in databases: The power of rdbms, *In*: Proc. 2009 ACM SIGMOD Int. Conf, On Management of Data, p. 681–694.

[15] Yu, J, X., Qin,L., Chang, L. (2010). Keyword Search in Databases: A Survey, IEEE Computer Society Technical Committee on Data Engineering.

[16] El-Qawasmeh, E., Abu-eid, I., Alashqur-Based, A. (2005). A Framework for Processing Keyword Queries in Relational Databases, *Journal of Theoretical and Applied Information Technology*, p. 149-163.

[17] Li, G., Feng, J.,Wang, J. (2009). Structure-Aware Indexing for Keyword Search in Databases, CIKM, p. 1453-1456.

[18] Moffat, A., Zobel, J. (1996). Self-Indexing Inverted Files for Fast Text Retrieval, *Journal of Association for Computing Machinery* (*ACM) Transactions on Information Systems*, 14 (4) 349-379.

[19] Hristidis, V., Papakonstantinou, V., Balmin, A. (2003). Keyword proximity search on XML graph. In ICDE.

[20] Khine, P, T, T., Tun, K, N,N. (2011). A System for Keyword-based Queries over Relational Database, UCSY, *In:* Proceeding of the Ninth International Conference on Computer Application.