Keyword Search over Relational Databases

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

Abstract

A relational database is often operated by means of a structured query language (SQL). When composing SQL-queries one must have an understanding of the SQL syntax to be able to produce a query the database can execute. Additionally, one must be familiar with the attributes and relations in the database to be able to retrieve the data desired. When one wants to search the available data stored in the relational database, these requirements can be discouraging. In this case keyword-based search functionality could improve the accessibility of the data. In recent years, much research on keyword search on relational database has been done, and many prototypes. However, there are still critical problems on the efficiency and effectiveness of Keyword search systems over relational database. In this paper, a system that enables keyword-based search in relational database (KQRD) is presented. We propose Best-first (A) algorithm and Structured Pivoted Normalization Weighting method to be able to search and retrieve the document from relational database effectively.*

1. Introduction

Keyword search provides a simple yet effective way for the users to query and explore the underlying documents. The last decade has seen ever expanding adoption of the keyword search technology, and it has become a de facto standard for user interaction on the World Wide Web (WWW). In the recent years, there has been a great deal of research and development activities on extending keyword search capabilities to handle relational data, the dominant form in which business data are stored.

While traditional relational database systems (RDBMSs) store the majority amount of the world's enterprise data, they provide only limited support for keyword search. Specifically, they only search for single tuples that match all the query keywords. A few recent studies have recognized the need to dynamically grouping the individual matching tuples to form meaningful results.

Adopting this keyword search on relational databases brings immediate benefits. 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. For example, users can still query the database without having to know the SQL query language, the database schema, or data distribution.

In this paper, we propose a system for keyword-based queries 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. Our paper presents an algorithm based on a best-first A* graph traversal to improve the efficiency of the system when the number of query keyword increases or the database schema becomes complicated and to generate smaller joining tuples and to execution time. We propose structured pivoted normalized ranking strategy for effective keyword search.

This paper is organized as follows: Section 2 summarizes current related work. Section 3 presents the system overview for processing keyword-based queries. Section 4 shows

experimental results and section 5 is the conclusion.

2. Related Work

With the advent of the World Wide Web, search engines allow users to perform keyword searches. However, a great amount of information is stored in databases and cannot be indexed by search engines. Therefore, the need for supporting keyword searching over databases as well is growing. Recent approaches extended the idea of tokens to values that may be part of attribute values. Several systems have been proposed, including BANKS [1, 2], DISCOVER [7, 8], DBXplorer [6] and ObjectRank [5]. A different line of research focuses on search effectiveness rather than efficiency [10].

According to the data models employed, existing research can be classified into two categories: graph-based approach and relational approach.

Graph-based approach materializes the entire content in the relational database into a graph, where each tuple is treated as a node and each foreign-key to-primary-key relationship is treated as a link between corresponding nodes. Existing work include the BANKS I/II systems [1, 2], Progressive Keyword Search [10]. For the relation-based approach, the answer to the keyword query is a set of tuples that collectively contains all or part of the search keywords, and these tuples can be joined together through foreign-key-to-primary-key relationships. Existing work include DBXplorer [6], DISCOVER I/II systems [7, 8], and [9].

So far, there has been no formal comparison between the two approaches. Our impression is that graph-based approach works well for small to medium sized databases, and has fast query processing performance. Relational approach, on the other hand, can be easily integrated into the relational database systems, thus, avoiding issues such as scalability, data and index maintenance, etc. Its performance for complex query and large query results still needs improvement.

The effectiveness of the scoring and ranking functions is an important aspect of keyword search. As more than one result may match any keyword query, it is desirable to assign each result a score and rank the list of results according to their scores. Intuitively speaking, the top results in the ranked list are more relevant to the query than those at the bottom. The IR community has developed theories and practice in ranking functions for documents [11].

Similar ranking functions have also been implemented in major relational databases systems. However, the result of the keyword search in relational databases is in general a tree of tuples (e.g., a particular movie tuple joins with another director tuple), rather than a flat and homogeneous text document. Therefore, IRbased methods cannot be immediately applied, and other factors, especially those regarding the structure and semantics of the query results, need to be considered.

Previous work has identified and used a few other metrics that contribute to more effective ranking functions for such search results, e.g., join tree size [6], PageRank node ranking [1], normalized node prestige and edge weight [1], shortest distance between keywords [2], etc. Of these, only [8, 9] considered combining some of the above factors with the IR-style ranking functions. However, the effectiveness of the proposed ranking functions are still questionable, as they have not been thoroughly tested against massive quantities of real data and queries.

3. Proposed System

3.1. Data Model and Keyword Queries

Data Model

We consider a database with n relations $R_1,...R_n$. Each relation R_i has m_i attributes $a_{i1},\ldots,a_{imi}$, a primary key and possibly foreign keys into other relations. The schema graph G_s (V, E), is a directed 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 2 shows the schema graph of DBLP used in this paper.

Keyword Queries

We are given a set of keywords S = {w_1, w_2,…,w_m} and a database DB with a searchable interface of I(A_1, A_2,…, A_k), which relies on relation R(A_1, A_2, … A_k). Our goal is to retrieve all the records containing all the keywords, more specifically, an answer set Ans(S), such that

$$Ans(S) = \left\{ r \mid r \in R \wedge S \bigcup_{i=1}^{k} r.Ai \right\}$$

InPro Id	InProceedingTitle	Page	Pro Id
1	Graphical Design of Reactive Systems	197	1
2	Optimization Networks	156	262
3	Layering Distributed Algorithms with B method	260	1

(a) InProceeding

Pro Id	Proceeding Title	EId	Pub Id	Series Id	Year	ISBN
1	Recent Advances in Development	1	1	1	1998	3-540-64405-9
2	Machine Learning and Its pplication	42	1	1	2001	3-540-42490-3
3	Mathematical Models for the Semantics of Parallelism	44	1	1	1987	3-540-18419-8

(b) Proceeding

Pub Id	PublisherName
1	Springer
2	IEEE Computer Society
3	ACM

(c) Publisher

Series Id	SeriesTitle
1	Lecture Notes in Computer
2	IFIP Transactions
3	IFIP Confere

(d) Series

PId	InProId
1	34854
1	34888
2	3910
3	4153

(e) RelationPersonInProceeding

PId	Name
1	Didier Bert
2	Emil Sekerinski
3	Jean-Marc Meynadier
4	Patrick Behm

(f) Person

Figure 1. Sample Database Instances

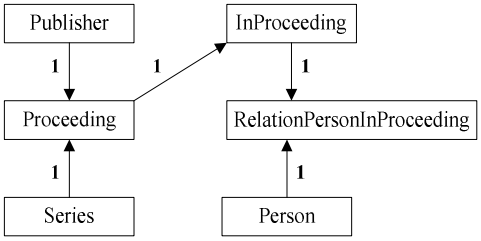

Figure 2. Schema Graph for Database

3.2. System Architecture

An overview of the system architecture of is presented in Figure 3. The proposed system works in three phases: Indexing the keyword, searching the query and ranking the result.

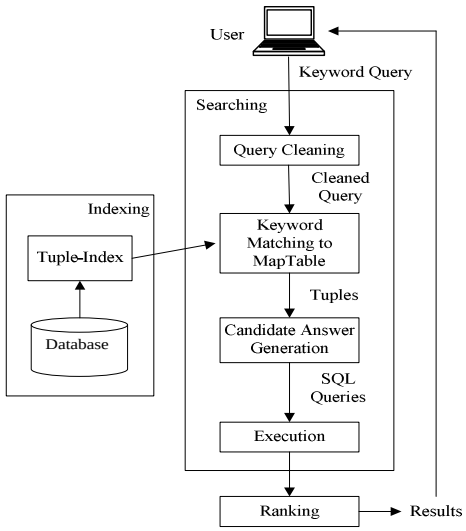

Figure 3. Architecture of proposed system

We have to build the index in advance to gain the speed benefits of indexing at retrieval time. When a user keyword query comes (Figure 3), the keyword query is cleaned by removing stopwords before searching the tuples. The cleaned keyword is then fetched to the query processing phase to retrieve the tuples which together match all the keywords in keyword query. The mapping table is used to retrieve which tuples contain keywords. After we find the tuple sets, we must search how the

keywords are related together inside a database. To do this, we need to build a candidate answer set from the database. The candidate answer sub-tables are then transferred to SQL queries, and execute them to get the results for the query. After execution process, multiple consequence result tuples are produced. Finally the tuples that are the most likely answer for the end users are presented to the user by proposing the result ranking method.

3.3. Indexing

A major concept in information retrieval is indexing. Indexing (or) index construction is applied to gain speed in the process of retrieval. An established indexing technique is to create what is called an inverted index. An inverted index is a mapping of words to their location in a set of documents. The basic idea of an inverted index is depicted in figure 4, where the numbers denote the document id-numbers in which the terms occur. The terms occurring in the document corpus are united in what is called a dictionary. Each term in this dictionary points to what is called a postings list consisting of separate postings.

For example, given a keyword query Brutus and Ceasar, the postings lists of the terms Brutus and Ceasar are retrieved and intersected to obtain the document id-numbers that occur in both postings lists.

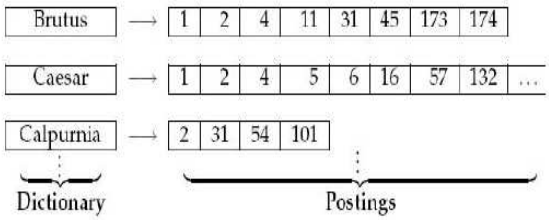

Figure 4. A fragment of inverted index

This approach can be useful when the database has large number of fields of type text (varchar). 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, we have to build the index in advance. The major steps in this are:

3.3.1. Keyword Finding

In the first step, we find the unique keyword to be indexed. To do this, we need to collect the relations' fields to be indexed, and tokenize the text, turning each tuple into a list of tokens. Then remove stopwords and do linguistic preprocessing, producing a list of normalized tokens, which are the indexing terms.

3.3.2. Keyword Indexing

After finding distinct keywords from each relation, the next step is keyword mapping process. Each distinct keyword is mapped with the tuples id that each keyword occurs in by creating a MapTable, consisting of a dictionary (keyword) and postings (mapId) based on inverted indexing technique.

Keyword	mapId (Relation Name Tuple Id)
springer	Pu (Publisher) R1 (Tuple Id)

keyword	mapId
springer	PuR1,PuR26,PuR65,
elsevier	PuR2,PuR27,PuR36,PuR69,
north	PuR3,PuR9,PuR27,PuR36,PuR42,PuR54,
holland	PuR3,PuR9,PuR27,PuR36,PuR42,PuR54,
national	PuR4,PuR53,PrR4,PrR13,PrR15,PrR23,P
institute	PuR4,PuR10,PuR30,PuR38,PuR53,PuR5
informatics	PuR4,PuR58,PuR64,
tokyo	PuR4,
chapman	PuR6,PuR81,
hall	PuR6,PuR81,
kluwer	PuR7,SR23,
gi	PuR8,PuR38,PuR45,PuR56,PuR58,PuR7
ieee	PuR9,PuR20,PuR37,
er	PuR1,PuR2,PuR7,PuR9,PuR10,PuR11,Pu
brandenburg	PuR11,
technical	PuR11,PuR30,PuR56,
university	PuR11,PuR12,PuR25,PuR30,PuR41,PuR
oxford	PuR12,
press	PuR12,PuR19,PuR21,PuR28,PuR29,PuR
measurement	PuR13,
group	PuR13,

Figure 5. Mapping Table generated using inverted indexing

3.4. Searching

When a user's keyword query comes, the input query is performed as the following steps:

3.4.1. Supporting Keyword Query Cleaning

The query cleaning phase takes a user entered keyword query as an input, and outputs a “clean” query. This is achieved by filtering the stopwords from the keyword.

Keyword queries are often dirty, i.e., they may contain words that are not intended as part of the queries. In the context of keyword search for databases, dirty keywords can also be keywords that do not appear in the database. As an example, consider the following query issued by the user: “Performance Analysis of a Distributed Simulation Scheme and Devendra Kumar ACM Press”

In this query, the only keywords that are relevant to the database content are “Performance Analysis”, “Distributed Simulation Scheme” (InProceeding title), “Devendra Kumnar”(Person name) and “ACM Press” (Publisher name). The rest of the query (e.g., “and”, “of”, “a”) are part of the natural language. One possible approach to handle such a problem is to filter them out as stopwords. First of all, we get the keywords which user submitted, and then we filter out the stopwords such as "an", "a", "the" and so on. These words may appear many times in the tuple tree, but they are meaningless. If we do not filter them out, the answers with them maybe returned with great priority and this will not satisfy the users, then we access to the database in the query process.

3.4.2. Query Processing

In this phase, the cleaned query is then fed into the query processing phase where the result is produced. This phase comprises two steps:

(i) Keyword Matching

This step finds a set of tuples TS $\{t_i\}$ (hits) which together match all the keywords in keyword query S, and group them by keyword. The MapTable is used to retrieve which tuples contain keywords and to construct tuple subsets. For example, when user enters query ("ACM Press")

ACM = {PuR5, PuR34, PuR9, PuR65, PrR45, SR9, SR20}
Press = {PuR12, PuR19, PuR21, PuR28, PuR29, PuR34, PuR41, PuR44, PuR47, PuR48, PuR49, PuR60…}

(ii) Candidate answer generation

After we find the tuple sets that include keywords, we must find how the keywords are related together inside a database. To do this, we need to build a candidate answer set from the database. The matching tuples in the database to each keyword are identified, and the goal is to find ways to join tuples from (potentially) different tables. Since each keyword can have multiple hits in the database, different (tuple sets) subspaces can be generated, corresponding to different combinations of hits for each keyword. The system searches through possible subspaces, aiming to optimize for a certain quality measure. The process of candidate answer set is achieved through a schema graph. So we proposed a candidate answer generation algorithm based on A* search algorithm to traverse the schema graph.

We use a graph to model the database schema. A schema graph consists of all the tables inside a database and the relationship among these tables, and the distance among tables. We consider the default distance between two related tables to be one. Figure 2 shows the schema graph of the database. The schema graph is represented by using arrows and nodes. The arrows represent a relation between database tables. Each node represents the table name.

Algorithm Candidate Answer Generation
Input: schema graph G_s, start, goal
Output: set of joining network
Begin
  Q: a queue data structure
  Add initial node to Q
  while Q is not empty {

N: a node at the front of the queue
If Q is empty, then quit
/*a path could not be found*/
Else
If N is the goal node, then quit
/* a path has been found*/
Else expand node and append all its derived nodes to the end of Q}
End

We get the candidate answer sets by traversing the schema graph using this algorithm.

Table 1. Candidate Answer Sets

Candidate Answers
Publisher
Publisher - Proceeding - Inproceeding
Proceeding
Proceeding - Inproceeding
Series
Series - Proceeding - Inproceeding

The output of this step, the candidate answer sub-tables are then transferred to SQL queries, each responsible for generating a possible final result.

(iii) Query execution

This step executes the SQL statement chosen from the previous step, produces the final result, and presents it to the user.
For Candidate set (Publisher - Proceeding - Inproceeding)

```
SELECT * From Publisher Pu, Proceeding Pr, InProceeding I
Where (Pu.PubId = Pr.PubId)
AND (I.ProId = P.ProId)
AND Pu.Name LIKE "%ACM%" OR "%Press%"
AND I.Title LIKE "%ACM%" OR "%Press%"
AND Pr.Title LIKE "%ACM%" OR "%Press%";
```

(iv) Calculating the resulted keyword weight

After finding the relevant records, we determine which relevant records that have more correlation with the query than others. Such a decision usually depends on the weights which attempt to establish the degree of correlation between the relevant records and the keywords-based query. We construct a weight table to compute the the most relevant answer. The weight value is computed using the following formula:

$$\text{Weight} = \sum_{1}^{N} \left(\frac{\text{KF/N}}{\text{RN}} \right)$$

where KF = frequency of the keyword.
RN= the number of relevant records of each keyword.
N = the total number of occurrences of keywords inside each relevant record.

The weight value determines which relevant records are more relevanant to the query.

3.5. Ranking the result

Ranking relevant record is the final step of the system. After execution process, multiple consequence result tuples are produced. Result ranking process is used to solve which tuples that are the most likely answer for the end users.

Result quality is one of the most important factors for keyword search systems. Retrieval effectiveness of the scoring and ranking functions is an important aspect of keyword search. As more than one result may match any keyword query, it is desirable to assign each result a score and rank the list of results according to their scores. The top results in the ranked list are more relevant to the query than those at the bottom. We propose a pivoted normalization ranking method for structured (relational) data.

$$Score(T_i, Q) = \sum_{k \in Q \cap T} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s.\left(\frac{dl}{avgdl} \right)} . \ln\left(\frac{N + 1}{df} \right)$$

where Ti = a result tuple that relevant to keyword query Q,
tf = the frequency of a keyword k appears in tuple Ti,
df = the number of tuples that k appears,
dl = the size of Ti,
avgdl = the average size of T,

N = the total number of T, and
s = a constant usually be 0.2.

4. Performance Evaluation

We used a data set of DBLP database, and applied a set of queries on this data to evaluate the performance of this approach. We implement our keyword search system in Java Environment using open source tool Eclipse and MySql Database. To test the performance, we wrote ten different keywords queries that are listed in Table 2. Figure 6 shows the index execution time of our proposed indexing mechanism for the queries.

Table 2. Query used for the test

Query Number	Keyword Query
Q1	IEEE Computer Society
Q2	Electrical Engineering and Computer Science
Q3	Institute of Cybernetics
Q4	Computer Simulation
Q5	World Scientific
Q6	Pennsylvania State University
Q7	ACM Press
Q8	Institute of Informatics, Faculty of Electrical Engineering
Q9	The Society for Computer Simulation
Q10	Cambridge University Press

The data retrieving time from the index file is also the core part of the keyword search system. We test the efficiency between our proposed indexing method and native method by submitting 10 queries to both methods. As the experiment result in Figure 6 shows that our proposed mechanism improved compared with the later ones.

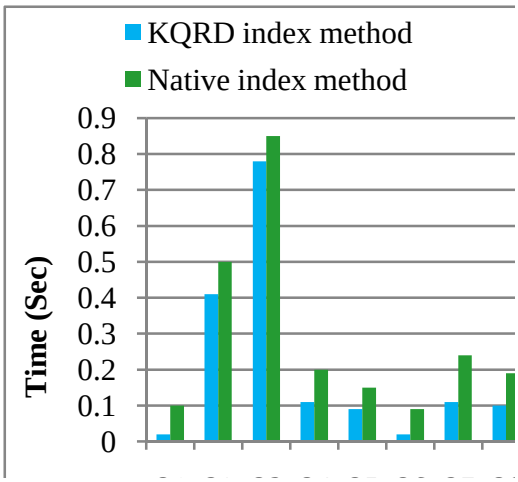

Figure 6. Efficiency of indexing mechanism comparison between two methods

5. Conclusion

As the amount of information stored in databases is growing, all humanity such as students, scientists, researchers, reporters may want to find the documents relevant to their need. Thus keyword search systems over relational databases have become more and more important. In this paper we propose a system that enables querying relational database using keywords which takes into account two measures: efficiency and effectiveness.

Efficiency is improved during two steps. One step is by constructing the MapTable. The second step is through combining relevant records from candidate answer sets by using A* based answer generation algorithm. In both cases we can reduce response time of the query.

Effectiveness is used to improve the accuracy of the result when the weight for both keyword and relevant records are assigned. The pivoted normalized weighting function gives the indication about the similarity between user keywords query and the set of relevant records.

Without efficiency there is a waste of time during searching for the relevant records. Also if a system takes to long time during retrieved result the user does not use a system again. Without effectiveness the user feel that the requirement for the information does note take

into count. By using the proposed processing scheme, the system will retrieve more relevant results that match the user needs.

References

[1] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, Parag, and S. Sudarshan, "BANKS: Browsing and keyword searching in relational databases". In VLDB, pages1083–1086, 2002.

[2] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional Expansion for Keyword Search on Graph Databases". In: Böhm, K., et al. (eds.) Proc. of the 31st Int'l. Conf. on Very Large Data Bases, pp. 505–516. ACM, New York (2005).

[3] Wang, S., Peng, Z.H., Zhang, J., Qin, L., Wang, S., Yu, J., Ding, B.L.: "NUITS: A Novel User Interface for Efficient Keyword Search over Databases". In: Dayal, U., et al. (eds.) Proc. of the 32nd Int'l. Conf. on Very Large Data Bases, pp. 1143–1146. ACM, New York (2006).

[4] Ding, B.L., Yu, J., Wang, S., Qin, L., Zhang, X., Lin, X.M.: Finding Top-k Min-Cost Connected Trees in Databases. In: Proc. of the 23rd Int'l. Conf. on Data Engineering, pp. 836–845. IEEE Press, Los Alamitos (2007).

[5] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "ObjectRank: Authority-Based Keyword Search in Databases", VLDB Endowment, 2004, pp. 564 -575.

[6] A. Konar, Artificial Intelligence and Soft Computing, CRC Press, 2000.

[7] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A system for keyword-based search over relational databases". In: Agrawal, R., et al. (eds.) Proc. of the 18th Int'l. Conf. on Data Engineering, pp. 5–16. IEEE Press, Los Alamitos (2002).

[8] V. Hristidis, Y. Papakonstantinou: "DISCOVER: Keyword Search in Relational Databases". VLDB, 2002:670-681.

[9] V. Hristidis, L. Gravano, Y. Papakonstantinou, "Efficient IR-Style Keyword Search over Relational Databases", VLDB, 2003:850-861.

[10] F. Liu, C. T. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases". In SIGMOD, pages 563–574, 2006.

[11] G. Li, X. Zhou, J. Feng, J. Wang. "Progressive Keyword Search in Relational Databases". ICDE, pp 1183-1186, 2009.

[12] C. D. Manning, P. Raghavan and H.Schütze, An Introduction to Information Retrieval, Cambridge University Press, pages 569, 2009.

[13] Graupmann, J., Schenkel, R., Weikum, G.: The SphereSearch engine for unified ranked retrieval of heterogeneous xml and web documents. In: VLDB, pp. 529–540 (2005)

[14] Guo, L., Shao, F., Botev, C., Shanmugasundaram, J.: XRANK: Ranked keyword search over xml documents. In: SIGMOD Conference, pp. 16–27 (2003)

[15] Hristidis, V., Koudas, N., Papakonstantinou, Y., Srivastava, D.: Keyword proximity search in XML trees. IEEE Trans. Knowl. Data Eng. 18(4), 525–539 (2006)

[16] Hristidis, V., Papakonstantinou, Y., Balmin, A.: Keyword proximity search on XML graphs. In: ICDE, pp. 367–378 (2003).

[17] S. Qi and W. Jennifer, "Indexing Relational Database Content Offline for Efficient Keyword-Based Search", Proceeding of IDEAS, 2005:297-306.

[18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. "XRANK: Ranked keyword search over XML documents". In ACM SIGMOD, 2003.

[19] V. Hristidis, Y. Papakonstantinou, and A. Balmin. "Keyword proximity search on XML graphs". In ICDE, 2003.

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

[21] A.Moffat and J. Zobel, "Self-Indexing Inverted Files for Fast Text Retrieval," Journal of Association for Computing Machinery (ACM) Transactions on Information Systems, Vol. 14, No. 4, pp. 349-379, 1996.

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

[23] Xu, Y. Ishikawa, and J. Guan. "Effective top-k keyword search in relational databases considering query semantics". In Proc. APWeb-WAIM 2009 International Workshops, volume 5731/2009 of LNCS, pages 172–184, Suzhou, China, Apr. 2009.

[24] L. Qin, J. X. Yu, and L. Chang. "Keyword search in databases: The power of rdbms". In Proc. 2009 ACM SIGMOD Int. Conf. On Management of Data, pages 681–694, 2009.

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

[26] P. T. T. Khine and K. N. N. Tun, "A Framework for Querying Relational Database using Keyword Search", The fifth Conerence on Parallel & Soft Computing, University of Computer Studies, Yangon, Myanmar, December 16, 2010.