ISSN 2229-5518

# Indexing Relational Databases for Efficient Keyword Search

Phyo Thu Thu Khine, Htwe Pa Pa Win, Khin New Ni Tun

**Abstract**— Keyword search is a widely accepted mechanism for querying in Information Retrieval (IR) systems and Internet search engines on the Web. They offer convenient keyword-based search interfaces. But searching in relational database systems the user needs to learn SQL and to know the schema of the underlying data even to pose simple searches. A system that can eliminate these requirements is needed. Therefore we proposed an efficient keyword-based search system for relational databases. The proposed system provides online search and offline indexing method to do efficient keyword based search. Firstly, a relational database is indexed in advance using the proposed indexing algorithm. At searching time, the index supports keyword-based searches with interactive response. The index size is manageable and database updates do not significantly hinder query performance. As long as the database table records can be extended, this system can be easily extendable for further searching records from tables. Experimental results show that the proposed algorithm provides less storage space and short offline indexing time and also reduces query processing time significantly compared to previous approaches.

**Index Terms**—search, relational database, information retrieval, indexing, searching, ranking.

## 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. One important advantage of keyword search is that it enables users to search for information without having to know complex query languages such as SQL and XQuery or prior knowledge about the structure of the underlying data. Keyword search provides an alternative means of querying relational databases, which is simple to most internet users. It lowers the access barrier for average users.

The alternative approaches of keyword search on relational databases are classified into candidate network based methods [8, 9, 10, 11], graph based algorithms [6, 12], and tuple unit based approaches [1, 2, 4, 5]. Li et al. [2] (Su et al. [1]) proposed the concept of tuple units (text objects) to efficiently answer keyword queries, which are composed of the most relevant tuples. The tuple units can be generated and indexed to improve search efficiency. Most previous approaches to keyword-based search in structured databases (reviewed in Section 2) perform a significant amount of database computation at search time. Our offline indexing approach does all significant work in advance so it can provide interactive answers.

This paper is organized as follows: Section 2 summarizes the literature reviews. Section 3 shows the overview of proposed system. Section 4 shows experimental results and section 6 is the conclusion and future work.


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


## 2 Literature Review

Verity [15] crawls the content of relational databases and builds an external text index for keyword searches, as well as external auxiliary indexes to enable parametric searches. DataSpot [16] extracts database content and builds an external, graph-based representation called a hyperbase to support keyword search. Graph nodes represent data objects such as relations, tuples, and attribute values. Query answers are connected subgraphs of the hyperbase whose nodes contain all of the query keywords.

DbSurfer [17] indexes the textual content of each relational tuple as a virtual web page. For querying and navigation, the database foreign-key constraints between tuples are treated as hyperlinks between virtual web pages. Given a keyword query, the system computes a ranked set of virtual web pages that match at least one keyword. Then a best-first expansion algorithm finds a ranked set of navigation paths originating from the starting web pages.

Three systems, DBXplorer [8], BANKS [6], and DISCOVER [9], 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 auxiliary indexes. Then 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.

Su et al [1] proposed a technique of indexing relational databases to improve the search efficiency. They indexes interconnected textual content in relational databases, and do keyword search over this content. A relational database is crawled in advance, text-indexing virtual documents that correspond to interconnected database content. At query time, the text index supports keyword-based searches with interactive response, identifying database objects corresponding to the virtual documents matching the query.

Li et al. [2] proposes the concept of tuple units (text objects) to efficiently answer keyword queries, which are composed of the most relevant tuples. The tuple units can be generated and indexed to improve search efficiency. However, existing methods identify a single tuple unit to answer keyword queries. They neglect the fact that in many cases a single tuple unit cannot answer a keyword query. It is promising to integrate several related tuple units to effectively answer keyword queries.

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) [4] 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.

## 3 Overview of Proposed System

Given a set of query keywords, the system returns the rows (either from single tables, or by joining tables connected by foreign-key joins) such that the each row contains all keywords. Enabling such keyword search requires (a) a preprocessing step called Indexing that enables databases for keyword search by building the Index Table and (b) a Search step that retrieves the ranked tuples to the user. Although there are two steps, we discuss only the indexing mechanism which is also the core part of the proposed system. The sample database instances from dblp are shown in Fig. 1. The DBLP schema is shown in Fig. 2.

| ProceedingId | Proceeding Title | EditorId | PublisherId | SerieId | Year | ISBN |
|---|---|---|---|---|---|---|
| 1 | Recent advances in Development and Use of B Method | 1 | 1 | 1 | 1988 | 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 | InProceeding Title | Page | ProceedingId |
|---|---|---|---|
| 1 | Graphical Design of Reactive Systems | 197 | 1 |
| 2 | Process Control Engineering: Contributions | 156 | 262 |
| 3 | Layering Distributed Algorithms with B Method | 260 | 1 |

(b) InProceeding

| SeriesId | SeriesTitle |
|---|---|
| 1 | Lecture Notes in Computer Science |
| 2 | IFIP Transactions |
| 3 | IFIP Conference Proceedings |

(c) Series

| PublisherId | Name |
|---|---|
| 1 | Springer |
| 2 | Elsevier |
| 3 | North Holland |

(d) Publisher

| PersonId | PersonName |
|---|---|
| 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

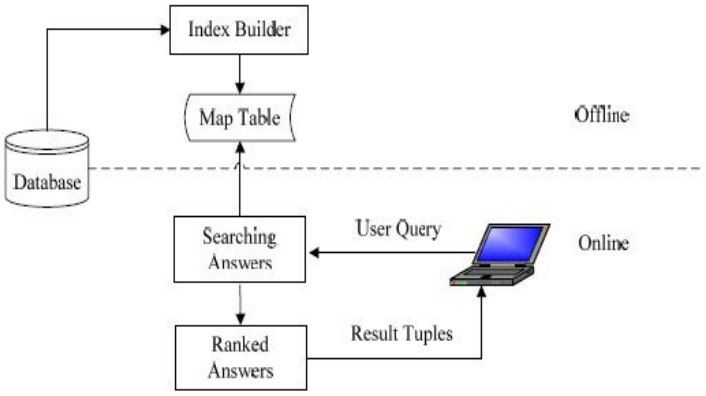



Figure 2. Framework of the System (System Overview)

### 3.1 Overview of Indexing and Searching Steps

**Index Builder**: A database is identified. The Index Builder finds the text attributes in tuple (t) in each table V. After finding, for the set of indices is defined as U, a set of attributes are found in table V that can be joined with these indices and create Index Table. Indexing Relational Database algorithm is proposed for this purpose. An index table is used at search time to efficiently determine the locations of query keywords in the database (i.e., the tables, columns, rows they occur in).

**Searching Answer**: Given a query consisting of a set of keywords, it is answered as follows.
Step 1: The index table is looked up to identify the tables, and columns/rows of the database that contain the query keywords.
Step 2: If the result tuples have the related information (connected tuples in the schema), all the connected tuples are

enumerated.
Step 3: For each enumerated candidate answer sets, a SQL statement is constructed (and executed) that joins the tables in the answer sets. Then the final rows are ranked and presented to the user.

### 3.2 Indexing a Relational Database

Indexing is also a major concept in information retrieval. Indexing (or) index construction is applied to gain the fast speed in the process of retrieval. This approach can be useful when the database has large number of fields of type Text or Varchar. A single relation (or table) is a set of tuples (or rows) each of which can be addressed by some primary key. To index these rows we extract the data from each row where the value of each column is text data type in turn and construct an index. 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. We propose an indexing algorithm to gain the speed of querying time. We find the records in the relational tables to be indexed and map them with postings (relation, tuple id). A fragment of IndexTable is shown in Fig. 2.

```
Algorithm Indexing Relational Database
Input: A database D, database relations R1, R2…Rn
Output: An IndexTable IT
Begin
1.  Scan database D and read the names of relations R1, R2...Rn.
2.  For each relation Ri (i=1 to n) in D do
3.      Structure Si ← Read structure of relation Ri
4.      Prefix ← Ri (substring (3))
5.      For each tuple t in Si to length of Si
6.          dt ← data type of t
7.          If (dt == "text")
8.          For each row ri (i=1 to length of Ri) in Ri do
9.              fieldname fn ← t
10.             keyword←values of fn (ri)
11.             mapId←Prefix + "R" + tupleid of ri
12.             Store keyword and mapId as curIndex.
13.             Store curIndex as tempIndex.
14.         End for
15.         End If
16. End for
17. If (t >1 && dt == ("text"))
18.     curIndex ← curIndex + tempIndex
19. End If
20. Insert curIndex into IndexTable
21. End for
End
```

Table 1. A Fragment of IndexTable

| Keywords | MapId |
|---|---|
| Graphical Design of Reactive Systems. 182-197 | inpR1 |
| Well Defined B. 29-45 | inpR2 |
| Process Control Engineering: Contribution to a Formal Structuring Framework with the B Method. 198-209 | inpR3 |
| Didier Bert | perR1 |
| Emil Sekerinski | perR2 |
| Jean-Marc Meynadier | perR3 |
| 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.uni-trier.de/db/conf/b/b1998.html | proR1 |
| Machine Learning and Its Applications, Advanced Lectures 2001 3-540-42490-3 http://dblp.uni-trier.de/db/conf/ac/ml2001.html | proR2 |
| Mathematical Models for the Semantics of Parallelism, Advanced School, Rome, Italy, September 24 - October 1, 1986, Proceedings 1987 3-540-18419-8 http://dblp.uni-trier.de/db/conf/ac/parallel1986.html | proR3 |
| Springer | pubR1 |
| Elsevier | pubR2 |
| North-Hollen | pubR3 |
| Lecture Notes in Computer Science http://dblp.uni-trier.de/db/journals/lncs.html | serR1 |
| IFIP Transactions http://dblp.uni-trier.de/db/series/ifip/transactions.html | serR2 |
| IFIP Conference Proceedings http://dblp.uni-trier.de/db/series/ifip/index.html | serR3 |

### 3.3 Keyword-based Searching

After the IndexTable is created, our system is ready for keyword search. Given a user's query consisting of a set of keywords, the input query is cleaned and matched with IndexTable.

The query cleaning phase takes a user entered keyword query as an input, and produces a “clean” query output. 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. 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.

The cleaned query is then fed into the query processing phase where the result is produced. This step finds a set of tuples TS {ti} (hits) which together match all the keywords in keyword query S. The MapTable is used to retrieve which tuples contain keywords.

Once the results are produced, each result is needed to calculate the score. A simple but effective ranking function is proposed to rank the result for a given query. The score of a result is assigned in the following way:

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

where n(k) = number of keywords in a user query,
count (f(k) = user keyword match keyword occurrence in m(U,V),
$score^k$ = score of each relevant record.

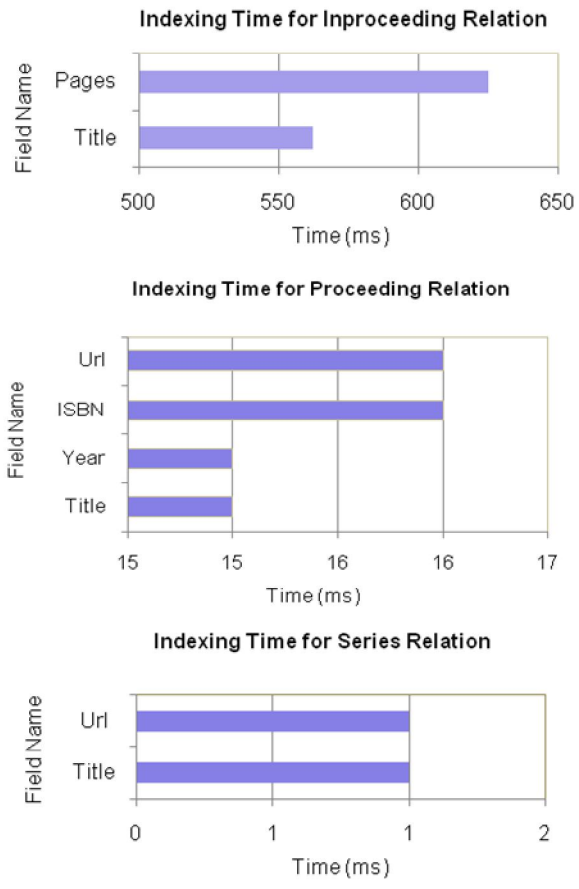
## 4 Experimental Results

We used 92.1 MB of DBLP dataset to evaluate the performance of the system. We implemented with Pentium Dual-Core 2.0GHz processor and 956 MB of RAM. The system decomposed the datasets into 6 relations according to the schema shown in Fig. 2. Table 2 summarizes the 6 DBLP relations.

Table 2. DBLP Dataset Characteristics

| Relations | Tuples |
|---|---|
| Inproceeding | 208,086 |
| Series | 24 |
| Publisher | 81 |
| Person | 162,907 |
| Proceeding | 2,749 |
| RelationPersonInProceeding | 491,777 |
| Total | 1,357,401 |

We evaluated the performance of the proposed indexing algorithm. Fig. 3 shows that the indexing time for each attibutes of each realation in DBLP.



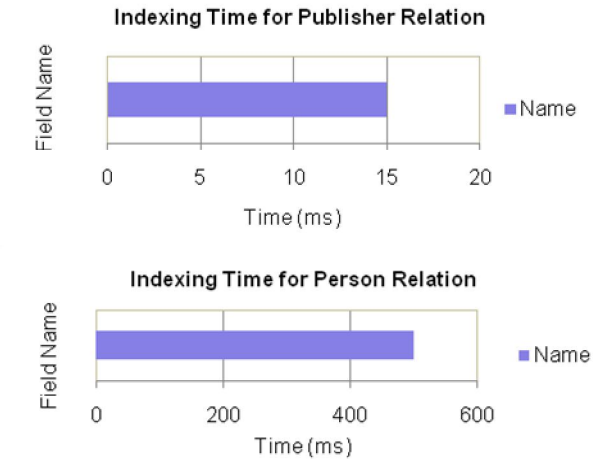




Figure 3. Indexing time for each attribute of DBLP database's relation

Table 1 shows that the IndexTable which is produced in preprocessing step. Indexing time includes IndexTable (keyword-posting) producing times and only consumes at index step.Testing case takes 373,847 rows from all tables. Fig. 4 shows the indexing time DBLP database. 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 only takes 1766 milliseconds (1.8 seconds). It shows that the proposed indexing mechanism can reduce the indexing time significantly.

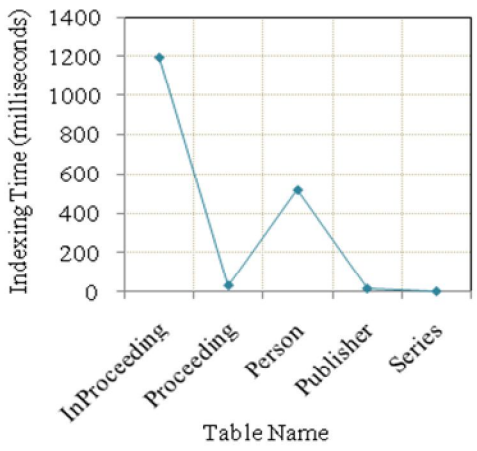



Figure 4. Indexing time for DBLP database

We evaluate search efficiency by submitting conjunctive keyword queries of length 2, 3, 4 and 5 words. Fig. 5 shows that the comparison of query processing time between the proposed method, ITREKS and Saint. We generated 50 queries for each query length. The figure shows that the proposed algorithm achieves much higher search efficiency than ITREKS and Saint. 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.

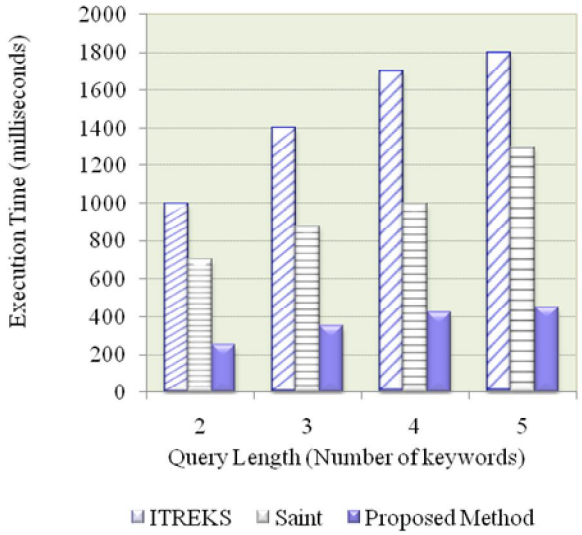



Figure 5. Search efficiency

## 5 CONCLUSION AND FUTURE WORK

We presented a general architecture for supporting keyword-based search over relational database by indexing contents from the relational database, providing intuitive and efficient keyword search capabilities over these contents. Our system trades online search and offline indexing method to do efficient keyword based search over relational database. The proposed system provides storage space and offline indexing time to significantly reduce query time computation compared to previous approaches. Experimental results show that index size is manageable and keyword query response time is interactive for typical queries. In the future, we will extend our method to semi-structured data like XML and implement our system over more databases.

## REFERENCES

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

[2] Li, G., Feng, J., and Zhou, L.(2008): "Retrieving and Materializing Tuple Units for Effective Keyword Search over Relational Databases." In ER.

[3] DBLP bibliography. http://www.informatik.uni-trier.de/~ley/db/index.html

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

[5] Li, G., Feng, J., and Wang, J. (2009): "Structrued Aware Indexing for Keyword Search in Databases". CIKM'09, pp 1453-1456.

[6] Aditya, B., Bhalotia, G., Chakrabarti, S., Hulgeri, A., Nakhe, C., Parag, Sudarshan, S. (2002): "BANKS: Browsing and keyword searching in relational databases." VLDB, pp. 1083–1086.

[7] Balmin, A., Hristidis, V., Papakonstantinou, Y. (2004): "ObjectRank: Authority-Based Keyword Search in Databases." In: Nascimento, M.A., et al. (eds.) Proc. of the 30th Int'l. Conf. on Very Large Data Bases, Morgan Kaufmann Publishers, San Francisco, pp. 564–575.

[8] Agrawal, S., Chaudhuri, S., Das, G. (2002): "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, IEEE Press, Los Alamitos, pp. 5–16.

[9] Hristidis, V., Papakonstantinou, Y. (2002). "DISCOVER: Keyword search in relational databases." In: Bernstein, P.A., et al. (eds.) Proc. of the 28th Int'l. Conf. on Very Large Data Bases, Morgan Kaufmann Publishers, San Francisco, pp. 670–681.

[10] Hristidis, V., Gravano, L., Papakonstantinou, Y. (2003). "Efficient IR-style keyword search over relational databases." In: Freytag, J.C., et al. (eds.) Proc. of the 29th Int'l. Conf. on Very Large Data Bases, Morgan Kaufmann Publishers, San Francisco, pp. 850–861.

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

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

[13] Manning,D. Raghavan, P. and Schütze, H, (2009), "An Introduction to Information Retrieval." Cambridge University Press.

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

[15] Raghavan, P. (2001): "Structured and unstructured search in enterprises." IEEE Data Engineering Bulletin, 24(4).

[16] Dar, S., Entin, G., Geva, S. and Palmon, E.(1998): "Dtl's dataspot: Database exploration using plain languages." In Proc. of VLDB.

[17] Wheeldon, R., Levene, M. and Keenoy, K. Search and navigation in relational databases. http://arxiv.org/abs/cs.DB/0307073.