Extending Keyword Search to Metadata in Relational Database

Jiajun GU† Hiroyuki KITAGAWA†‡

† Graduate School of Systems and Information Engineering
‡ Center for Computational Sciences
University of Tsukuba, Tennoudai 1-1-1, Tsukuba, Ibaraki, 305-8573, Japan
E-mail: gu@kde.cs.tsukuba.ac.jp, kitagawa@cs.tsukuba.ac.jp

Abstract Keyword search has been the most widely used kind of querying nowadays, especially for searching documents on the web because of its user-friendly way. In recent years various approaches for free-form keyword search over RDBMSs have been proposed. They can allow search for combinations of interesting terms and produce results across multiple tuples in different relations. However, they just consider keyword search for values in tuple instances. In fact users have requirements to search keywords which may be part of the metadata of the database such as names of relations or attributes. In this paper, we extend keyword search in relational database. We define a tuple with annotation as an extension concept of a conventional tuple. In addition we add proposed weight to tuples. The weight function also cares about metadata information. We implement the query processing scheme in RDBMS in order to prove the proposed approach.

Keyword Relational Database, Keyword Search, Weight

1. Introduction

Keyword search is an easy and user-friendly querying method, because users are not required to have knowledge of the query language and the underlying structure of data.

As the amount of available data in relational databases is growing rapidly, the need for general users to search such information is quickly increasing. Based on the major RDBMS(Relational DataBase Management System) the search requires users to know complex query languages and data schema. So free-form keyword search over RDBMSs has attracted recent research interests.

In conventional keyword search, each document constitutes one unit of information, and is included in a result, if it contains all or some of the keywords [10]. In RDBMS, the basic unit of information is a tuple. The key problem in applying search techniques to database is the information related to a single answer to a query may be split across multiple tuples in different relations [4]. In other words, answers are not limited to the individual tuples, but are assembled from joining tuples of multiple tables.

In recent years various approaches to keyword search in relational databases have been proposed such as [4, 5, 6, 8, 10, 11, 12, 14, 15]. They can allow search for combination of interesting terms without a-prior knowledge of the data schema and query language. However, they just consider keyword search of tuples. In fact, users have requirements for querying metadata information related with keywords.

In this paper we define a tuple with annotation as an extension concept of a conventional tuple and try to use it to extend keyword search to metadata in relational databases. In addition, we add weights to tuples which contain keywords. The weight function also cares about keywords in metadata information such as names of relations or attributes. The results containing all the keywords are ranked according to proposed weights.

The remainder of this paper is structured as follows: Section 2 introduces related work. Section 3 gives motivating example. Section 4 defines the extension concept of a tuple and describes the proposed approach for keyword search in relational databases with metadata in details. Section 5 shows the experiments results. Section 6 summarizes the paper and gives the future work.

2. Related Work

There are a handful of different approaches for keyword search in RDBs. Here we just introduce Discover [5]. Discover exploits the RDBMSs schema graph information to return qualified joining trees of tuples as results, that is, sets of tuples which are associated on their primary-foreign key relationships and contain all the keywords of the query.

At query time, it firstly finds tuple instances in each relation that contain at least one keyword by using inverted index and then use graph-based approach to traverse the database schema and enumerate all joining networks of tuple sets called candidate networks CNs. A tuple set consists of the tuples which just contain a sub-set of the query keywords set in exactly one relation. A CN is a joining expression that involves non-free tuple sets plus free tuple sets. Free tuple sets in a CN means they do not

contain any query keywords, but help to construct results by connecting the non-free tuple sets.

Finally, based on the enumerated joining networks of tuple sets, execution plans are translated into SQL statements. The results are ranked in ascending order of the number of the joins involved in the tuple trees.

After Discover was proposed, some papers were published to extend it and make it robust. [6] focused on effective keyword search. They proposed to use information retrieval (IR) ranking technologies for keyword search in relational databases to get more effective results. [6] also proposed some efficient query-processing algorithms to obtain Top-K results.

In this paper, we use [5, 6] as the basic approach to implement keyword search in RDB and extend it to keyword search including metadata.

3. Motivating Example

Figure 1 is a simple example of a part of movie information database. Figure 1(a) is a Movie relation consisting of three attributes MID, Title and Year while MID is a primary key. Figure 1(b) is a Play relation consisting of two attributes MID and AID while both are foreign keys. Figure 1(c) is an Actor relation consisting of two attributes AID and Name while AID is a primary key. For explanation, we assign ID to every tuple instance in each relation as shown in Figure 1.

If a user gives a keyword query "Titanic, Kate", we can easily find the joining sequence of m2, p3 and a4 because the joining sequence contains two keywords in its end nodes. In this example, it is the only result because we do not consider answers containing only a part of query keywords.

	Movie		
	MID	**Title**	**Year**
m1	**01**	**The Year of the Yao**	**2004**
m2	**02**	**Titanic**	**1997**
m3	**03**	**Titanic**	**1953**
m4	**04**	**The Aviator**	**2004**
m5	**05**	**1953**	**2003**

(a)

	Play	
	MID	**AID**
p1	**01**	**002**
p2	**02**	**003**
p3	**02**	**004**
p4	**03**	**001**
p5	**04**	**003**

(b)

	Actor	
	AID	**Name**
a1	**001**	**Robert Wagner**
a2	**002**	**Ming Yao**
a3	**003**	**Leonardo DiCaprio**
a4	**004**	**Kate Winslet**

(c)

Figure 1: Database example

In some cases, however, users may give a query like "Leonardo, Winslet, Movie". It means that they want to know whether Leonardo and Winslet have performed in the same movies or not. In reality, this kind of queries is often raised by general users. Recalling the database example in Figure 1, "Movie" is not a value in any tuple instances, but a relation name, one kind of metadata. In existing approaches, users cannot get any results from our example. Even if fortunately users can get some, they may not be the exact results users expect.

Figure 2 is a data graph for Figure 1 while circle, rectangle and triangle represent tuple instance, its corresponding attribute and relation metadata respectively.

Current approaches can search keywords only in tuple instances but cannot solve keyword search with metadata such as the information in the dashed line circle in Figure 2, so our objective is to make it possible.

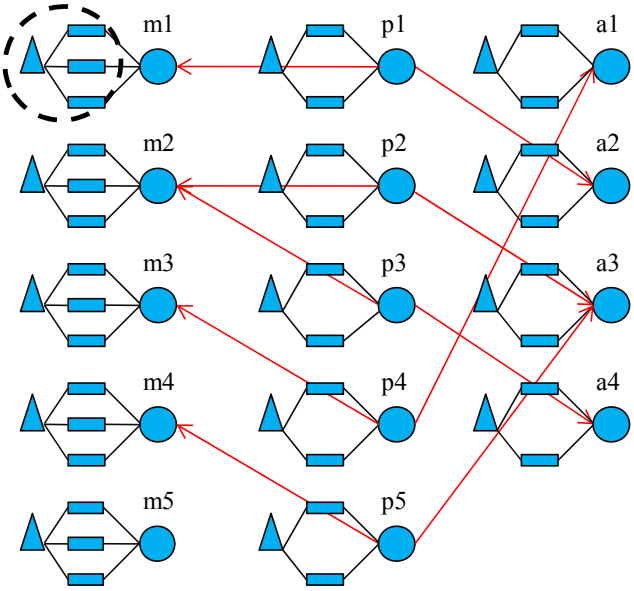

Figure 2: Data graph for Figure 1

4. Proposed Approach

We try to extend keyword search in relational databases to make it possible for keyword search including metadata.

4.1. Definition

We define a tuple with annotation as an extension concept of a conventional tuple [1]. Each conventional tuple consists of a set of attribute-value pairs:

$$\{(A_1: v_{i1}), (A_2: v_{i2}) \ldots (A_n: v_{in})\}$$

(i = 1, 2 … m, where m is the number of tuples in the relation and n is the number of attributes.)

We define a tuple with annotation consists of a set of attribute-value pairs:

$$\{(A_1: v_{i1}), (A_2: v_{i2}) \ldots (A_n: v_{in}): (N_1: A_1), (N_2: A_2) \ldots$$

$(N_n: A_n), (N_{n+1}: R)\}$

where $N_1 \sim N_{n+1}$ are new attributes for metadata and R is a relation name.

We just add the metadata information to the tuple instances as values.

Figure 3 is a new Movie relation consisting of tuples with annotation. We just keep a part of annotation for simple expression. We can see the tuples in this new Movie relation including a set of attribute-value pairs $\{(MID: v_{i1}), (Title: v_{i2}), (Year, v_{i3}): (N_4, Movie)\}$ while the original set of pairs is $\{(MID: v_{i1}), (Title: v_{i2}), (Year, v_{i3})\}$. The additional information for metadata is $\{(N_4, Movie)\}$. With this new Movie relation if given a query "Leonardo, Winslet, Movie", the result can be retrieved as the joining sequence of a3, p2, m2, p3 and a4. This answer satisfies users' requirement perfectly.

	Movie			
	MID	**Title**	**Year**	**N_4**
m1	**001**	**The Year of the Yao**	**2004**	**Movie**
m2	**002**	**Titanic**	**1997**	**Movie**
m3	**003**	**Titanic**	**1953**	**Movie**
m4	**004**	**The Aviator**	**2004**	**Movie**
m5	**005**	**1953**	**2003**	**Movie**

Figure 3: New Movie relation

(The terms with underline are metadata information)

4.2. Query Model

We consider a database with a set of relations $\{R_1, R_2 \ldots R_n\}$. The relations are connected through primary-foreign key relationship. The database schema graph is constructed with relations as nodes and relationships as edges. In notation, Ri→Rj represents primary key to foreign key relationship. For simple expression, we assume all primary and foreign key attributes are in one attribute and there is no more than one primary-foreign key relationship between two relations. We also assume there are no self-loops or parallel edges in the database schema graph.

A query Q is a set of distinct keywords and a result is a joining tree T of tuples with annotation. Each leaf node in T contains at least one unique keyword. It is not allowed that the same tuple appears more than once in one T. In this paper we assume Boolean-And semantics, so a result should contain all keywords of the given query.

4.3. Ranking

We now discuss how to rank the results for a given query Q.

There are some kinds of results which we have to distinguish by our weight.

For example, if given a query like "Year, 1953", both of m3 and m5 in Figure 1 contain keyword "1953". Because the term "1953" of m3 is in attribute "Year" which is also a keyword, we suppose that the weight of m3 will be larger than the weight of m5.

For another example, if given a query like "Movie, Yao", both of m1 and a2 in Figure 1 contain keyword "Yao". But we can see that m1 is in relation Movie which is associated with keyword "Movie", we also suppose that the weight of m1 will be larger than the weight of a2.

In order to distinguish the above examples, in our weight function we have to consider both values in a tuple instance and in metadata, so the weight should be designed in two parts.

For the tuple instance part, as in [6], we use single-attribute IR-style relevance scores for each textual attribute.

$$\text{Score\_I} = \sum_{w \in (Q \cap A_{ij})} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s\frac{dl}{avdl}} * \ln\frac{N + 1}{df}$$

where for a word w, tf is the term frequency of w in A_{ij}, A_{ij} is the i-th attribute in relation R_j, df is the number of tuples in R_j with word w in attribute A_{ij}, dl is the size of A_{ij} attribute-value in characters, avdl is the average attribute-value size, N is the total number of tuples in R_j, and s is a constant which usually equals to 0.2.

For the metadata part, we consider "Attribute" and "Relation" as two levels of metadata and just use term frequency for each keyword contained in two levels.

$$\text{Score\_A} = \sum_{(w \in (Q \cap A_{ij})} tf$$

$$\text{Score\_R} = \sum_{(w \in (Q \cap R_j)} tf$$

where for a word w, tf is the term frequency of w in its corresponding metadata level, Score_A is for attribute level and Score_R is for relation level.

The weight of a tuple instance is boosted by both attribute level and relation level metadata. So the weight function for a single-attribute of a tuple with annotation is as follows:

$$\text{Weight}(A_{ij}, Q) = \text{Score\_I} * (1 + \alpha * \text{Score\_A} + \beta * \text{Score\_R})$$

where α and ß are constants.

The final weight of the result which is a joining tree of tuples notated as T is as follows:

$$\text{Combine}(\text{Weight}(A_{ij}, Q), \text{size}(T)) = \frac{\sum_{Aij \in A} \text{Weight}(A_{ij}, Q)}{\text{size}(T)}$$

where A is the set of all textual attributes in T and size(T)

is the number of tuples in T.

For the query "Movie, Yao", by using the above weight function we can calculate the weight of a2 and m1 as follows:

$$\text{Weihgt(a2)} = \frac{1+\ln(1+\ln(1))}{(1-0.2)+0.2*\frac{7}{10.15}} * \ln\frac{4+1}{1}$$

$$\text{Weihgt(m1)} = \left(\frac{1+\ln(1+\ln(1))}{(1-0.2)+0.2*\frac{15}{8.6}} * \ln\frac{5+1}{1}\right) * (1+\alpha*0 + \beta*1)$$

4.4. Query Processing Scheme

Figure 4 describes the architecture of query processing system.

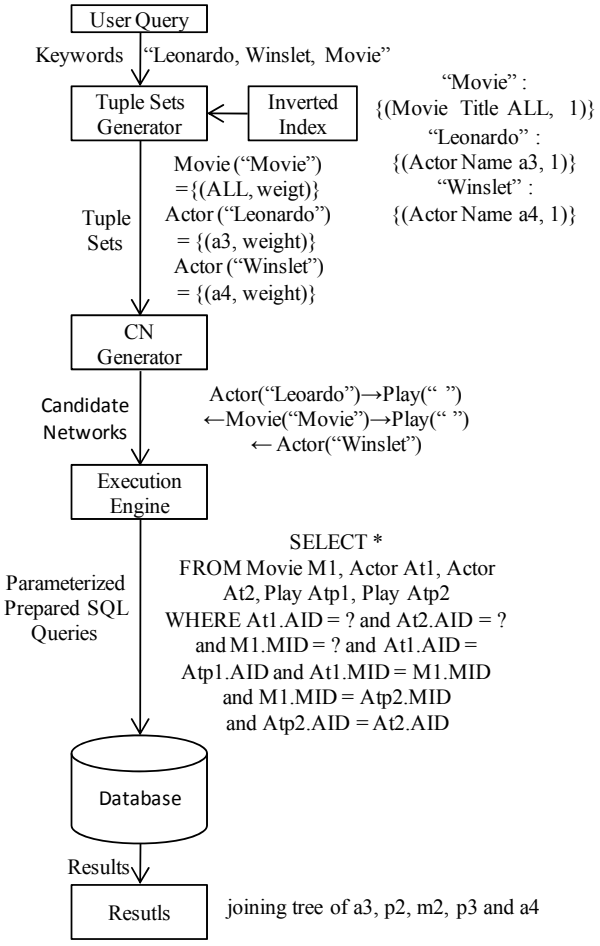

Figure 4: Architecture of query processing system

Firstly, we have to construct an inverted index for terms both in tuple instances and in metadata. For the former, we construct a list to keep the information consisting of two parts. One is location including the relation name, the attribute name and tuple ID and the other is term frequency. For the latter, we do it in almost the same way. The only difference is that in tuple ID part, we replace it with "ALL".

Secondly we use the inverted indexes to find which tuples contain keywords and to construct tuple sets. For example, a tuple set Actor ("Winslet") = {(a4, weight)}. If we get a keyword "Movie", we can obtain a tuple set as Movie ("Movie") = {(ALL, weight)} means all tuples in relation Movie contain keyword "Movie" with their weight.

Then we generate candidate networks which contain all query keywords by traversing the tuple sets graph. Each node in the tuple sets graph is a tuple set which is a non-free tuple set or free tuple set. Each edge represents primary-foreign key relationship based on the database schema. For example, for a given query "Leonardo, Winslet, Movie", we can obtain one CN as Actor ("Leonardo") → Play (" ") ← Movie ("Movie") → Play (" ") ← Actor ("Winslet") while Play (" ") means a free tuple set of Play.

Finally, by using each CN and its corresponding tuple IDs in returned tuple sets, we translate CNs into parameterized prepared SQL queries and execute them in RDBMSs to retrieve ranked results based on our proposed weights.

5. Experiments

In this section we implemented the proposed scheme described above in RDBMS. For our evaluation, we use the Internet Movie Database (IMDB) [16]. It is a real on line dataset of information about movies, actors, directors, etc.

We decomposed IMDB dataset into relations according to the database schema shown in Figure 5. We constructed nine relation tables by converting a subset of IMDB's raw files. The scheme of nine relations is shown in Figure 6.

We ran our experiments using the MySQL v5.0.22 with their default configurations. The system was run on a PC with a Xeon 2.13GHz CPU and 2G RAM. The database server and the client were run on the same PC. We implemented all query-processing algorithms in Java through JDBC connecting to the RDBMS.

To have a realistic query set where the results are not always empty, we picked up query keywords from a restricted subset of IMDB for queries with Boolean-And semantics. Specially, the query keywords are restricted in the nominators and winners of Oscar Awards [17] including Best Movie, Best Actor, Best Actress, Best Supporting Actor, Best Supporting Actress and Best

Directors. For the results, we just consider the joining tree T of tuples of which the size (T) is no more than 5. And we set both of α and ß as 1.

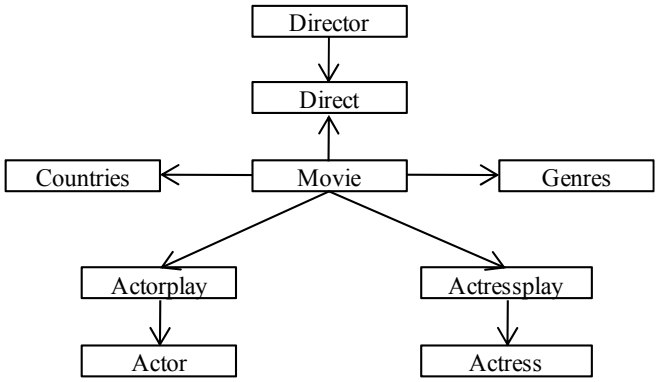

Figure 5: IMDB database schema
(→ represents primary-foreign key relationship)

Relation Scheme	Number of Tuples
Actor (*atID*, Name)	873786
Actorplay (atID, mID)	5766668
Actress (*asID*, Name)	524846
Actressplay (asID, mID)	3291573
Countries (mID, Country)	634924
Director (*dID*, Name)	140828
Direct (dID, mID)	745202
Genres (mID, Genre)	741437
Movie (*mID*, Title, Year)	1131831

Figure 6: Relation Scheme for IMDB
(Text attributes are with underline and primary keys are in italic type)

5.1. Evaluation of results size

(1) Queries without metadata information

We executed 10 queries in our processing system and the existing processing system respectively.

From the results, we confirmed that the two systems returned the same results with the same weight for each query. That means our proposed system does not influence the keyword search without metadata.

(2) Queries with metadata information

We executed 10 queries of three keywords and each query contains only one keyword from metadata information such as “Title”, “Director”, etc.

The number of results is shown in Figure 7. By using our proposed processing scheme considering metadata, from Figure 7 we can find that there are much more results retrieved.

Query	1	2	3	4	5
Non_metadata	6	3	16	3	8
Proposed	1626	1623	38	1295	107
Query	6	7	8	9	10
Non_metadata	8	6	9	3	2
Proposed	112	13	1604	1508	1507

Figure 7: Number of results for each query
(Non_metadata means the number of results without considering metadata. Proposed means the number of results considering metadata)

5.2. Evaluation of weight function

For simplicity's sake, we denote the set of results without considering metadata as A and the set of results with considering metadata as B. Thus we specially denote “the set of increased results” as B-A.

To evaluate the weight function, in the experiments, we compared the Top-K results for each query in two systems. We want to find out the influence of B-A to A in Top-K results.

Figure 8 and Figure 9 show the influence for Queries 1~5 and Queries 6~10 respectively. The horizontal axis represents the Top-K results and the vertical axis represents the number between 0 and 10. It means how many results of B-A are in the Top-K results.

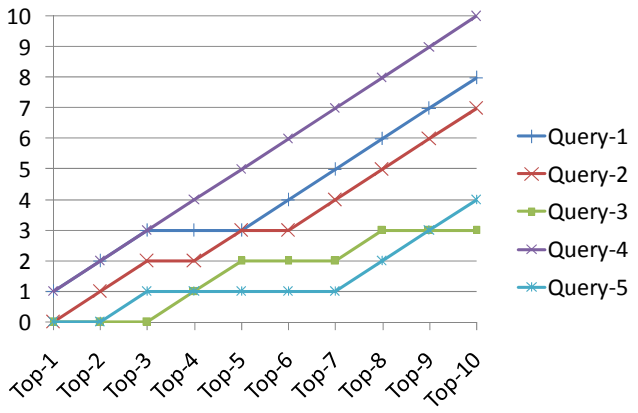

Figure 8: Graph about the influence for Queries 1~5

Based on Figure 8 and Figure 9, we summarized the average number of each Top-K point and plotted the Figure 10. The horizontal axis represents the Top-K results and the vertical axis represents the percentage. It means the percent of the average number of results from B-A in Top-K results. By using our proposed weight function, we can find that in general about 60% of Top-K results are from B-A part. The weight function influenced about 60% of the results in A part.

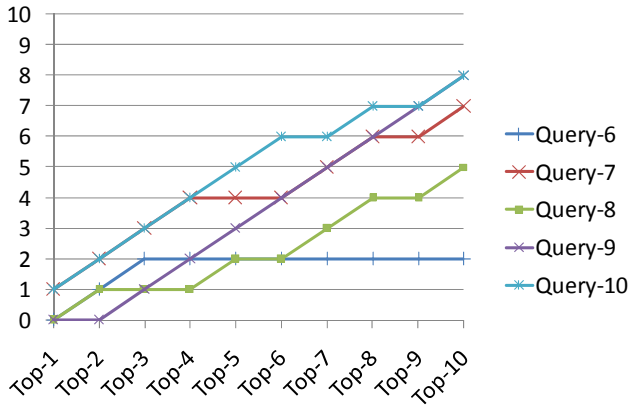

Figure 9: Graph about the influence for Queries 6~10

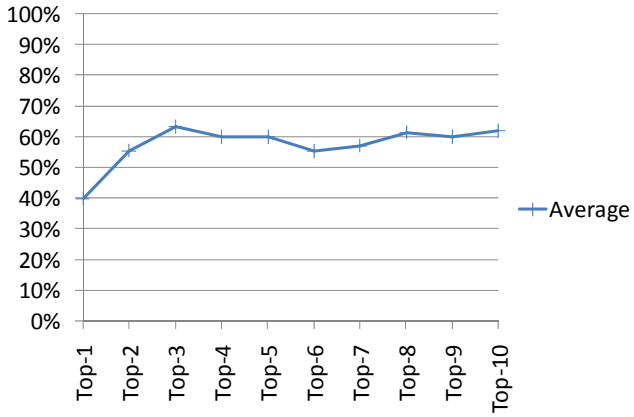

Figure 10: The graph for average number in Top-K

5.3. Discussion

The main conclusion of our experiments is that with considering metadata information we can get much more results from the database and based on the proposed weight function in average about 60% of Top-K results will be influenced by the increased results.

Here we just consider one keyword from metadata and did not experiment the adjustment of constants α and β. In the future, we will plan to do more experiments on this part.

6. Conclusion and Future Work

In this paper, we proposed a scheme to realize keyword search for relational databases including metadata. In addition, we proposed a special weight function which considers tuple instance and metadata both. And we implemented the proposed approach in real RDBMS.

In future work, we are going to do more extensive experiments on different real databases. We will also consider more complex situation such as keyword search in multi-database with semantically related information but different structure.

7. Acknowledgements

This research has been supported in part by Grant-in-Aid for Scientific Research from MEXT (#19024006).

References

[1] C.J. Date. An introduction to Database Systems VOLUME I, Fifth Edition, Addison-Wesley, 1990.

[2] W. Kim, I. Choi, S. Gala, M. Scheevel. On resolving Schematic Heterogeneity in Multidatabase Systems. In Distributed and Paralled Databases 1 (1993), pp.251-279.

[3] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina. Proximity Search in Databases. In VLDB, 1998.

[4] G. Bhalotia, A. Hulgeri, C.Nakhe, S. Chakrabarti. Keyword Search in Databases. In IEEE Data Engineering Bulletin, 2001

[5] V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In VLDB, 2002.

[6] V. Hristidis, L. Gravano, Y. Papakonstantinou. Efficient IR-Style Keyword Search over Relational Databases. In VLDB, 2003.

[7] C.M. Myss, E.L. Rovertson. Relational Languages for Metadata Integration. In ACM Transactions on Database Systems, Vol.30, No.2, June 2005, pp.624-660.

[8] F. Liu, C. Yu, W. Meng, A. Chowdhury. Effective Keyword Search in Relational Databases. In SIGMOD, 2006.

[9] A. Markwetz, Y. Yang, D. Papadias. Keyword Search on Relational Data Streams. In SIGMOD, 2007.

[10] Y. Luo, X. Lin, W. Wang. SPARKS: Top-k Keyword Query in Relational Databases. In SIGMOD, 2007.

[11] J. Zhang, Z. Peng, S. Wang, H. Nie. CLASCN: Candidate Network Selection for Efficient Top-k Keyword Queries over Databases. In J. Comput. Sci. & Technol, Mar. 2007, Vol.22, No.2, pp.197-207.

[12] C.M. Wyss, F.I. Wyss. Extending Relational Query Optimization to Dynamic Schemas for Information Integration in Multidatabases. In SIGMOD, 2007.

[13] G. Koutrika, A. Simitsis, Y. Ioannidis. Precis: The Essence of a Query Answer. In ICDE, 2006.

[14] M. Sayyadian, H. LeKhac, A. Doan, L. Gravano. Efficient Keyword Search Across Heterogeneous Relational Databases. In ICDE, 2007.

[15] B. Yu, G. Li, K. Sollins. Effective Keyword-based Selection of Relational Databases. In SIGMOD, 2007.

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

[17] http://www.oscar.com