# Metadata-aware Keyword Search in Relational Databases

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 is the most widely used information retrieval approach which powers the successful web search engines. In recent years because of its user-friendly way it has been applied to RDBMS and several approaches have been proposed. According to a query with a set of keyword terms these approaches can retrieve results from multiple tuples in different relations. However, they only consider keyword terms in tuple instances but ignore the metadata part such as names of relations or attributes. As a matter of fact ordinary users have requirements to raise keywords which may be contained in the metadata part. In this paper, we propose metadata-aware keyword search approach. We define a tuple with annotation as an extension concept of a conventional tuple. Based on the WordNet ontology we calculate the similarity between the query and the metadata part. Furthermore, we propose the weight function which also cares about metadata information. Finally, we implement the query processing scheme in RDBMS in order to prove our proposed approach.

**Keyword** Relational Database, Keyword Search, Weight, Similarity

## 1. Introduction

Keyword search is the most widely used information retrieval approach which powers the successful web search engines because of its user-friendly way. However, according to the major RDBMS (Relational DataBase Management System) the general search requires users to have a good knowledge of the query language and the scheme of databases. It is the bottleneck of RDBMS's wide spreading to normal users. Therefore free-form keyword search has been applied to RDBMS and 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 RDB (Relational DataBase) the basic unit of information is a tuple which consists of several attributes. The major difference between keyword search in document and RDB is that for the former the result is just one document, for the latter the result is not limited to a single tuple, but several tuples from multiple relations based on some relationship.

In recent years various approaches to keyword search in RDB have been proposed such as [5, 6, 7, 9, 10, 11, 12, 15, 16]. According to a query with a set of keyword terms these approaches can retrieve results from multiple tuples in different relations. However, they only consider keyword terms in tuple instances but ignore the metadata part such as names of relations or attributes. As a matter of fact normal users have requirements to raise keywords which may be contained in the metadata part.

In our previous work [20], we defined a tuple with annotation as an extension concept of a conventional tuple and worked on the exact match for the terms of query and metadata part. In this paper, we propose metadata-aware keyword search approach considering similarity. Based on the WordNet ontology we calculate the similarity between the query and the metadata part. Furthermore, we propose the weight function which also cares about metadata information. The results containing all the keywords are ranked according to the proposed weight function.

The remainder of this paper is structured as follows: Section 2 introduces related work and previous work briefly. Section 3 gives the motivating example. Section 4 explains the proposed approach for keyword search in detail. 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 [6]. Discover exploits the RDBMS's 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, after users input a query, it firstly finds tuple instances in each relation that contain at least one keyword by using inverted index. Then it traverses database schema graph to enumerate all minimal joining networks of tuple sets called candidate networks. A candidate network CN is a minimal joining network of tuples because all end nodes of a CN should contain at least one distinct keyword. Therefore, a CN can be considered as a joining expression to produce joining networks of tuples which satisfy the keyword query.

A tuple set consists of the tuples which just contain a sub-set of the query keywords set in exactly one relation. A CN contains all the keywords and involves non-free tuple sets and free tuple sets. Free tuple sets in a CN 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. [7] 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. [7] also proposed some efficient query-processing algorithms to obtain Top-K results.

In this paper, we use [6, 7] as the basic approach to implement keyword search in RDB.

None of the above approaches consider keyword search including metadata. The weight function in them does not care about metadata information either.

In our previous work, we defined a tuple with annotation as an extension concept of a conventional tuple which consists of a set of attribute-value pairs:

$\{(A_1: v_{i1}), (A_2: v_{i2})... (A_n: v_{in}): (N_1: A_1), (N_2: A_2)... (N_n: A_n), (N_{n+1}: R)\}$

where i = 1, 2 … m, where m is the number of tuples in the relation and n is the number of attributes. $N_1 \sim N_{n+1}$ are new attributes for metadata and R is a relation name.

In the tuple with annotation the metadata information is added. Based on it we can extend the existing keyword search system to the metadata-aware keyword search system.

Our previous work is limited to the exact match for terms of the query and the metadata part. In fact, the queries can be more flexible. Therefore we propose metadata-aware keyword search considering similarity in this paper and design the special weight function which also cares about metadata part.

## 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 the information about movies which were performed by Leonardo and Winslet. In reality, this kind of queries is often raised by normal 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. More generally, because normal users have no knowledge of the scheme of database, the keyword term they raised cannot exactly match with the terms in metadata part. For example, they may raise keyword such as “Film”, “Show” and “Performer”. These keywords have similar meanings to their corresponding metadata term.

In existing approaches, users cannot get any results from our example. Even if fortunately users can get some, they may not be the relevant results users expect.

In the next section we try to solve the above problem by using our proposed approach.

## 4. Proposed approach

### 4.1. Query model

We consider a database with a set of relations {$R_1$, $R_2$... $R_n$}. 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, $R_i \rightarrow R_j$ 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.

### 4.2. Similarity

As discussed in the motivating example part, it is often the case that the keyword doses not exactly match with the term in the metadata part. To cope with this problem, we employ WordNet ontology to find the similarity between the term of the query and the term of the metadata part.

WordNet [19] is a large lexical database of English language, developed in Princeton University. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms (synsets). Each synset expresses a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations. Different types of word have different relations between synsets. For nouns as an example, there are hypernymy, hyponymy, holonymy and metonymy conceptual-semantic relations.

Hypernymy/hyponymy is also called subset/superset or the is-a relation. For example, the relation for computer and machine is hypernymy/hyponymy relation because the computer is a machine. Holonymy/metonymy is the is-a-part-of relation. For example, the relation for computer and monitor is a holonymy/metonymy relation because the monitor is a part of a computer. Also synsets can be interlinked through lexical relations such as antonymy. For example, the relation for beauty and ugliness is antonymy relation because the meaning of beauty is opposite to the meaning of ugliness.

Additionally, in WordNet database, singular form and plural form of a word are considered in the same synset.

In our paper, we consider hypernymy/hyponymy relation for similarity. Figure 2 is an example of WordNet ontology while a rectangle represents a synset and a link represents hypernymy/hyponymy relation.

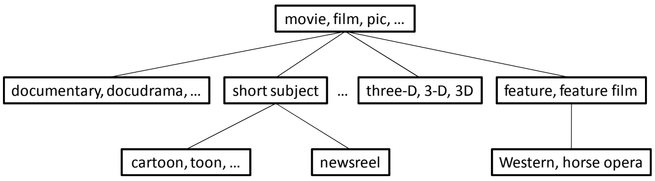

Figure 2: An example of WordNet ontology

For similarity calculation, we consider the inverse of (the shortest path length + 1) between synsets as the similarity for two terms. For example, according to Figure 2 the path length between “movie” and “cartoon” is 2, so the similarity for them is 0.3333. If two terms appear in the same synset such as “movie” and “film”, the similarity for them is 1. And if either of two terms cannot be found in any synsets, the similarity for them is 0. As a result the similarity for any two terms is between 0 and 1.

For implementation of similarity calculation in computers, we employ WordNet::Similarity [19] which implements a variety of semantic similarity and relatedness measures based on information found in the lexical database WordNet. Measures of similarity use information found in the is-a relation of synsets, and show how much synset A is similar to synset B. WordNet::Similarity cannot compare the similarity between noun pairs and verb pairs because the is-a relation in WordNet does not cross part of speech boundaries. WordNet::Similarity cannot be used for adjectives and adverbs because they are not organized into the is-a relation. In our experiments, the terms in the metadata part and executed queries are all nouns, so WordNet::Similarity can be applied to our research.

WordNet::Similarity not only supports the measure of path length between terms but also some other famous measures. More detail information can be found in [19].

### 4.3. Ranking

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

In order to distinguish the different weight for value in the tuple instance part and metadata part, the weight function should be divided into two parts.

For the tuple instance part, as in [7], we use single-attribute IR-style relevance scores for each textual attribute.

$$\mathrm{Score\_I}(t_k.A_{ij}, Q) = \sum_{w \in (Q \cap t_k.A_{ij})} \frac{1 + \ln(1 + \ln(\mathrm{tf}(w)))}{(1 - s) + s\frac{\mathrm{length}(t_k.A_{ij})}{\mathrm{avg}(A_{ij})}} * \ln\frac{N + 1}{\mathrm{df}(w)}$$

where tf(w) is the term frequency of word w in $t_k.A_{ij}$, df(w) is the number of tuples in $R_j$ with word w in attribute $A_{ij}$, length($t_k.A_{ij}$) is the size of $t_k.A_{ij}$ in characters, avg($A_{ij}$) is the average attribute-value size of $A_{ij}$, 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 choose maximum similarity between terms in the query and the metadata part.

$$\mathrm{Score\_A}(t_k.A_{ij}, Q) = \sum_{n \in \mathrm{Name}(A_{ij})} \max(\mathrm{Sim}(w, n))$$

$$\mathrm{Score\_R}(t_k.A_{ij}, Q) = \sum_{n \in \mathrm{Name}(R_j)} \max(\mathrm{Sim}(w, n))$$

where for a word w of the query, $\mathrm{Score\_A}(t_k.A_{ij}, Q)$ means the similarity for the query and $\mathrm{Name}(A_{ij})$, $\mathrm{Score\_R}(t_k.A_{ij}, Q)$ means the similarity for the query and $\mathrm{Name}(R_j)$, $\mathrm{Score\_A}(t_k.A_{ij}, Q)$ is for attribute level and $\mathrm{Score\_R}(t_k.A_{ij}, Q)$ is for relation level. $\mathrm{Sim}(w, n)$ means the similarity for term w and term n.

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

$$\mathrm{Weight}(t_k) = (\sum_{\mathrm{Score}(t_k.A_{ij}) \neq 0} \mathrm{Score\_I}(t_k.A_{ij}, Q) * (1 + \alpha * \mathrm{Score\_A}(t_k.A_{ij}, Q))) * (1 + \beta * \mathrm{Score\_R}(t_k.A_{ij}, Q))$$

where α and β are constants.

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

$$\mathrm{Combine}(T) = \frac{\sum_{t_k \in T} \mathrm{Weight}(t_k)}{\mathrm{size}(T)}$$

where size(T) is the number of tuples in T.

## 4.4. Query processing scheme

Figure 3 describes the architecture of query processing system.

Firstly, we have to construct an inverted index for terms 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 (tf(w)). For the latter, we do it in almost the same way. The difference is that in tuple ID part, we replace it with "ALL". For relation metadata, we also replace the attribute name with "ALL". We store the average attribute-value size of $A_{ij}$ (avg($A_{ij}$)) beforehand in the database. We also build an another index to store the size of $t_k.A_{ij}$ in characters (length($t_k.A_{ij}$)) in advance.

Through similarity calculation step, we find the similarity for every combination of each term in the query and the metadata part by using WordNet::Similarity.

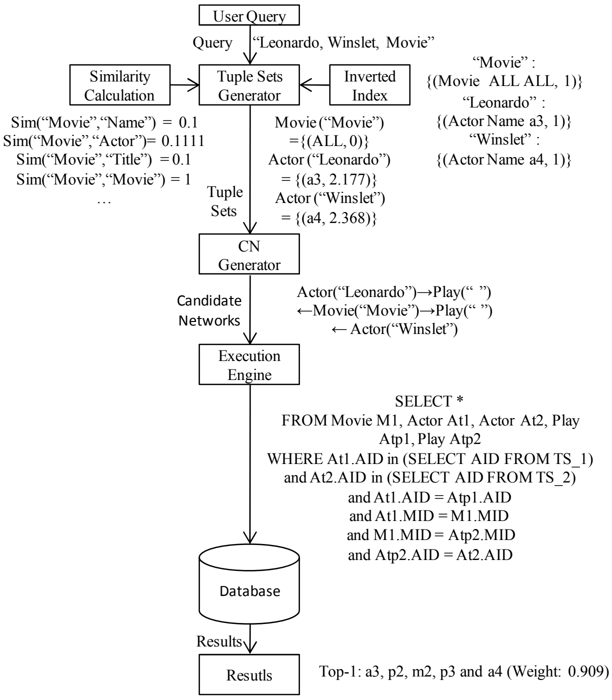
User Query
Query
"Leonardo, Winslet, Movie"
Similarity Calculation
Tuple Sets Generator
Inverted Index
"Movie" : {(Movie ALL ALL, 1)}
"Leonardo" : {(Actor Name a3, 1)}
"Winslet" : {(Actor Name a4, 1)}
Sim("Movie","Name") = 0.1
Sim("Movie","Actor")= 0.1111
Sim("Movie","Title") = 0.1
Sim("Movie","Movie") = 1
…
Movie ("Movie") ={(ALL, 0)}
Actor ("Leonardo") = {(a3, 2.177)}
Actor ("Winslet") = {(a4, 2.368)}
Tuple Sets
CN Generator
Candidate Networks
Actor("Leonardo")→Play(" ") ←Movie("Movie")→Play(" ") ← Actor("Winslet")
Execution Engine
SELECT *
FROM Movie M1, Actor At1, Actor At2, Play Atp1, Play Atp2
WHERE At1.AID in (SELECT AID FROM TS_1)
and At2.AID in (SELECT AID FROM TS_2)
and At1.AID = Atp1.AID
and At1.MID = M1.MID
and M1.MID = Atp2.MID
and Atp2.AID = At2.AID
Database
Results
Resutls
Top-1: a3, p2, m2, p3 and a4 (Weight: 0.909)

Figure 3: Architecture of query processing system

Secondly we use the inverted index to find which tuples contain keywords and to construct tuple sets. We store the tuple sets in the database. From the inverted index, we can also get the number of tuples in $R_j$ with word w in attribute $A_{ij}$ (df(w)). In Figure 3, a tuple set Actor ("Winslet") is generated and stored as a new relation TS_1 in the database. Then we can calculate the weight of each tuple with word "Winslet" and add the weight to this temporal relation. Therefore, TS_1 = {(a4, 1.624)}. TS_2 = {(a4, 1.624)}. If we get a keyword "Movie", we can obtain a tuple set as Movie ("Movie") = {(ALL, 0)} which means all tuples in relation Movie contain keyword "Movie" and the weight for each tuple is 0.

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 SQL statements and execute them in RDBMS to retrieve ranked results based on our proposed weights. In this example, there is only one result being retrieved, so Top-1 result is the joining tree of a3, p2, m2, p3 and a4. The final weight for this result is 0.909.

## 5. Experiments

In this section we implemented the proposed scheme described above in RDBMS. For our evaluation, we use the Internet Movie Database (IMDB) [17]. 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 4. We constructed nine relation tables by converting a subset of IMDB’s raw files. The scheme of nine relations is shown in Figure 5.

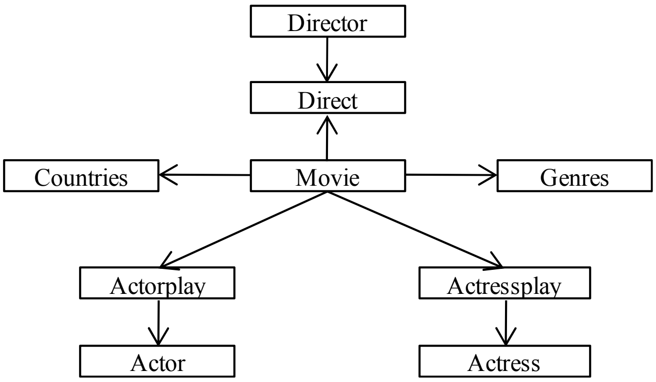

Figure 4: IMDB database schema
(→ represents primary-foreign key relationship)

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.

For the results, we just consider the joining tree T of tuples of which the size (T) is no more than 5. And we setboth of α and β as 1. We assume the Boolean-And semantics, so the result should contain all keywords of the given query. In our experiments, if the similarity for two terms is equal or larger than 0.5, we consider the two terms are the same.

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

Figure 5: Relation Scheme for IMDB
(Text attributes are with underline and primary keys are in italic type)

### 5.1. Evaluation of results size

We executed 10 queries of three keywords and each query contains only one keyword which can be considered as the same to the term from the metadata part such as “Show”, “Film”, etc. To have a query set where the results are not always empty, by analyzing inverted index we picked up keywords which have high term frequency. The detail of 10 queries is shown in Figure 6.

| **Number** | **Query** |
|---|---|
| Q1 | {Jerry, Tom ,Director} |
| Q2 | {City, Frank, Director} |
| Q3 | {Days, World, Actress} |
| Q4 | {American, Tom, Director} |
| Q5 | {Helen, Tom, Actor} |
| Q6 | {Gibson, Kate, Player} |
| Q7 | {Steven, Tom, Head} |
| Q8 | {City, Larry, Show} |
| Q9 | {Street, Victoria, Film} |
| Q10 | {Business, Kim, Head} |

Figure 6: The detail of 10 queries

The number of results is shown in Figure 7. By using our proposed processing scheme considering metadata, from Figure 7 we can find that most of the queries got more results.

| Query | Q1 | Q2 | Q3 | Q4 | Q5 |
|---|---|---|---|---|---|
| Non_metadata | 1 | 7 | 0 | 19 | 1 |
| Proposed | 231 | 57 | 10 | 34 | 1106 |
| Query | Q6 | Q7 | Q8 | Q9 | Q10 |
| Non_metadata | 0 | 3 | 84 | 0 | 1 |
| Proposed | 143 | 1981 | 48 | 73 | 15 |

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. Relevance of results

Figure 8 shows the the number of relevant results without considering metadata. We can observe that most of the queries cannot get relevant results.

Figure 9 shows the precision for each query with considering metadata. In Figure 9, the horizontal axis represents ten queries in Figure 7 and the vertical axis represents the precision for each query. The precision here is based on all returned results in “Proposed” row of Figure 7. Except Query 2 the precision of others queries is high. The reason for the outlier Query 2 is that a big part of results with considering metadata were also the results without considering metadata. And from Figure 8 we know that results of Query 2 have no relevant ones, so the precision for Query 2 was affected much.

| Query | Q1 | Q2 | Q3 | Q4 | Q5 |
|---|---|---|---|---|---|
| Non_metadata | 1 | 7 | 0 | 19 | 1 |
| Relevant | 0 | 0 | 0 | 0 | 0 |
| Query | Q6 | Q7 | Q8 | Q9 | Q10 |
| Non_metadata | 1 | 3 | 84 | 0 | 1 |
| Relevant | 0 | 2 | 0 | 0 | 0 |

Figure 8: Number of results and relevant results for each query (Non_metadata means the number of results without considering metadata. Relevant means the number of relevant results)

In Figure 10, we did another experiment for the average precision for Top-K results with considering metadata. The horizontal axis represents Top-K results (K is from 1 to 10) and the vertical axis represents the average precision for ten queries for each Top-K results. We can observe that the average precision is larger than 0.7. It has proved that the proposed weight function works well.

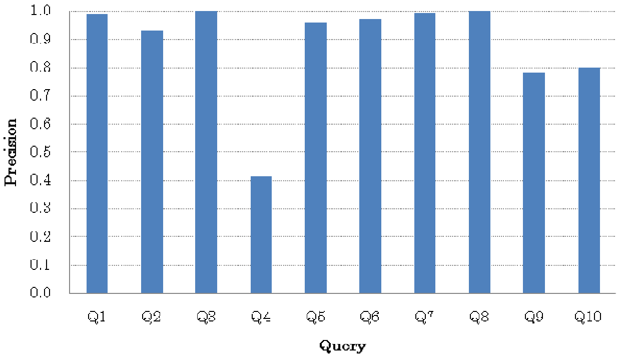

Figure 9: The precision for each query

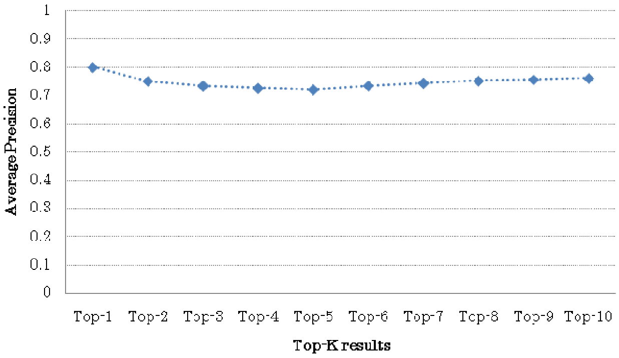

Figure 10: The average precision for Top-K

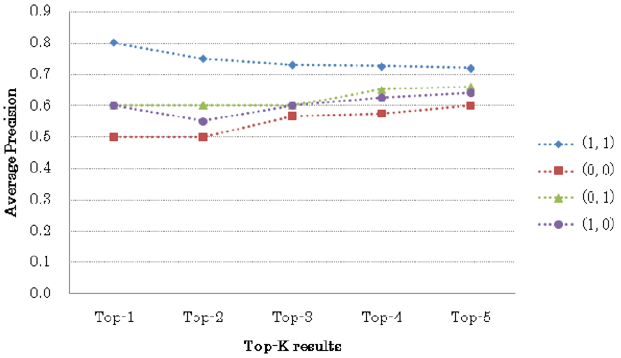

Figure 11: Parameters adjustment (For the value (1, 0) means α = 1 and β = 0, the same to the other kinds.)

### 5.3. Parameters adjustment

We assigned two different values to parameters α and β, so there are four kinds of results showed in Figure 11. The horizontal axis represents Top-K results (K is from 1 to 5) and the vertical axis represents the average precision for ten queries for each Top-K results. We compared the average precision for Top-K among four kinds of results. We can observe that if we do not assign weight to the metadata part, the average precision goes down. It proved the effectiveness of the proposed weight function for the metadata part. It also proved the necessity of considering both levels of relation and attribute metadata.

## 5.4. Comparison

In our previous work, we just consider the exact match. In this chapter, we consider the similarity match. Here we also did a comparison experiment between these two proposals. For the exact match, we consider that if the keyword term exactly matches with the term of the metadata part, the similarity for these two terms is 1. If not, we consider the similarity for them is 0.

Figure 12 and Figure 13 show the detail of relevance of retrieved Top-10 results. Figure 12 represents the similarity-based situation and Figure 13 represents the exact match situation. The head column of these two tables means 10 queries and the head row of these two tables means the n-th result. "○" and "×" represent relevant and non-relevant respectively. "−" means no results. The red marks in Figure 13 mean changes compared to Figure 12.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| Q1 | × | × | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q2 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q3 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q4 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q5 | ○ | × | × | × | × | × | × | × | × | × |
| Q6 | × | × | × | × | ○ | ○ | ○ | ○ | ○ | ○ |
| Q7 | ○ | ○ | × | × | × | × | × | × | × | × |
| Q8 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q9 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q10 | ○ | ○ | ○ | ○ | × | ○ | ○ | ○ | ○ | ○ |

Figure 12: Top-10 results in similarity-based match for 10 queries

From Figure 12 and Figure 13, we can observe that the precision of five queries goes down. These five queries are Q6, Q7, Q8, Q9 and Q10. The same point of these queries is that all of them contain a metadata-similar keyword which does not exactly match with the term of metadata part. Especially for Q6 and Q9, without considering similarity, there are no results retrieved from the dataset. Therefore, compared with Figure 12 and Figure 13, it has proved the necessity of considering similarity-based match.

We can also observe that the precision of Q5 goes up. The point of this query is that it contains a keyword "Actor". From WordNet::Similarity, we found that the similarity for "Actor" and "Actress" is 0.5, so these two terms will be considered as the similar terms in our experiments and some results which do not contain the information about "Actor" returned. But in our setting of the database schema in the experiments, we do not consider "Actor" and "Actress" are the same. Therefore, for the exact match, this query can retrieve better results and the precision of it goes up.

| | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| Q1 | × | × | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q2 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q3 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q4 | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ |
| Q5 | ○ | × | × | × | × | × | × | × | ○ | ○ |
| Q6 | − | − | − | − | − | − | − | − | − | − |
| Q7 | ○ | ○ | × | − | − | − | − | − | − | − |
| Q8 | × | × | × | × | × | × | × | × | × | × |
| Q9 | − | − | − | − | − | − | − | − | − | − |
| Q10 | × | − | − | − | − | − | − | − | − | − |

Figure 13: Top-10 results in exact match for 10 queries

## 6. Conclusion and future work

In this paper, we proposed a metadata-aware keyword search approach with considering similarity. In addition, we proposed a special weight function which considers tuple instance and metadata both. We implemented the scheme with real data in RDBMS and from the experiments our proposed approach has been proved.

In future work, we are going to do more extensive experiments on different real databases to evaluate the weight function. Furthermore, we will plan to consider the similarity in the level of tuple instances. 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] J. M. Smith, D. C. P. Smith. Database Abstractions: Aggregation and Generalization. In ACM TODS, Vol. 2, No. 2, June 1977, Pages 105-133.

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

[3] 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.

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

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

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

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

[8] 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.

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

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

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

[12] 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.

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

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

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

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

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

[18] http://wordnetweb.princeton.edu

[19] http://wn-similarity.sourceforge.net/

[20] J.Gu, H.Kitagawa. Extending Keyword Search to Metadata on Relational Databases. In INGS, 2008.