A FRAMEWORK FOR PROCESSING KEYWORD-BASED QUERIES IN RELATIONAL DATABASES

[1]EYAS EL-QAWASMEH, [1]OSSAMA ABU-EID, [2]ABDALLAH ALASHQUR
1 Jordan University of Science and Technology, Jordan
[2] Applied Science University, Jordan
E-mail: eyas@just.edu.jo , alashqur@asu.edu.jo

ABSTRACT

The large sizes of existing databases are rapidly increasing with time. In this environment, there is a need to help users, who are usually not familiar with SQL statements and the schema of a database, in getting the information sorted according to their relevancy. In response to this need, many researchers have introduced the capability of querying a database based on a list of keywords. A user does not have to state a full SQL query, but just provide the list of keywords that seem to be of interest. The system would then return the relevant records from different tables that appear to be close to what the user is looking for, based on the list of keywords that he/she provides. However, there is a need to improve the efficiency and effectiveness of existing systems. In this paper, we introduce a framework for processing keywords-based queries that improves the efficiency and effectiveness. In addition, the performance results are presented.

Keywords: *Relational Databases, Text Databases, Efficiency, Effectiveness, Relative_keyword_weight, Keyword_weight, Record Weight, Cosine Measure, Ranking.*

1. INTRODUCTION

Currently, databases are used in almost all business applications that handle a huge amount of data. Databases provide the ability to search for relevant information that the user is interested in.

A text database is "a collection of related documents assembled into a single searchable unit. The individual documents can be massive or minuscule, and should relate to each other" [16]. A text database is normally queried via a set of keywords provided by the user. The system searches for the documents relevant to the keywords provided by the user, and then returns those documents. The degree of correlation between each of retrieved documents and the list of keywords is computed to identify which documents are considered more relevant than others. After computing the degree of correlation, retrieved documents are ranked. A ranking technique is used whereas a score is assigned to each retrieved document. Documents that have higher score are considered more relevant to the query (i.e., list of keywords) and are placed at the top of the resulting set of documents.

A relational database is "a database constituting a set of relations (tables). A relation is a set of records. Records are a set of attribute values, and each attribute is identified by its name" [11]. A relational database contains multiple tables that are related to each other by relationships. The relevant results in a relational database can be found in multiple records. Moreover, there is a relationship among tables themselves via primary and foreign keys.

In many situations, the relevant information that is to be retrieved from a relational database requires writing sophisticated SQL statements [10]. Since the size of data stored in relational databases increases over time, the number and complexity of SQL queries that need to be written increases proportionally. To make it easier to query such databases, a keyword-based

approach is used, which alleviates the need to write complex query statements. This keyword-based approach is very similar to that used in text databases as described above. However, instead of searching through documents, the system searches records in relational tables. 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.

In this paper we present a framework for keyword-based queries, where users do not need to know the 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 based on their relevancy to the query.

There are two aspects, namely *efficiency* and *effectiveness*, that must be taken into account when retrieving the relevant records from a relational database. *Efficiency* measures "how fast a result is obtained from a database" [9]. Efficiency can be expressed in terms of response time (i.e., time needed to search and return the results from the database). *Effectiveness* is a measure used to find relevant records which are more relevant to the query than others [9]. Effectiveness describes how the system computes the degree of correlation between relevant records and the keywords-based query.

The framework which we will introduce takes into account efficiency and effectiveness. Efficiency is improved when the search operation relates different records to each other. On the other hand, effectiveness is improved through two steps. The first step is by finding the degree of correlation between relevant records and the query. The degree of correlation takes into account the appearance of keywords-based query inside relevant records. The next step is to improve effectiveness through ranking of relevant records and sorting them in a descending order according to matched keywords.

This paper is organized as follows: Section 2 summarizes current related work. Section 3 presents our framework for processing keyword-based queries. Section 4 presents an example to further describe our framework. Section 5 shows experimental results. Section 6 is the conclusion.

2. CURRENT RELATED WORK

Querying using keywords is the most common method that is used today. Querying of a database relies on query languages that are inappropriate for end-users who have little experience with databases. There are many models of keyword-based querying in relational databases. Among them are the uniform, statistical and graph models where the last one is the most common and will be the core of this section.

In the graph model of relational databases, there are many existing works such as BANKS [4], DBXplorer [13], and DISCOVER [17]. The graph model is used in keyword search as follows: Each record in the "database is modeled as a node in the directed graph and each foreign key-primary key link as an edge between the corresponding records" [2]. After that it "searches the hidden connections between those records that contain keywords specified in a user-given keyword-based query. Almost the previous works attempt to obtain the tree that are contains all the keywords in a database graph" [6].

BANKS [4] "works on graph representation of relational database. An answer to a query is considered to be a rooted directed tree containing a directed path from the root to each keyword node. The root node is called an 'information node' and the tree a connection tree".

DBXplorer [13] "returns all rows (either from single tables, or by joining tables connected by foreign-key joins) such that each row contains all keywords. Enabling such keyword search requires (a) a preprocessing step called Publish that enables databases for keyword search by

building the symbol table and associated structures, and (b) a Search step that gets matching rows from the published databases".

DISCOVER [17] "facilitates information discovery on them by allowing its user to issue keyword queries without any knowledge of the database schema or of SQL. DISCOVER returns qualified joining networks of records, that is, sets of records that are associated because they join on their primary and foreign keys and collectively contain all the keywords of the query".

Many researchers tried to improve the accuracy of the retrieval records. In their work, they tried to rank the retrieved records according to some criteria. Many of these ranking schemes applied to existing methods such as: 1) Fang Liu focused on the effectiveness when searching for keywords inside a database. The effectiveness technique computes the degree of correlation between relevant results and the query. After that, the relevant records will be ranked on a descending order according to their matching to the query [3]. 2) Vagelis Hristidis "presented a system for efficient IR-style keyword over relational databases. A query is simply a list of keywords, and does not need to specify any relation or attributes name. The system handles queries with both "AND" and "OR" semantics between keywords-based query. “AND” query operations mean, every keyword must appear in every relevant result. “OR” query operations mean, some keywords might be missing from relevant results" [18].

3. FRAMEWORK OF KEYWORD-BASED QUEIRIE

In this section, we will give an overall description of our framework and demonstrate how it works with an example. The framework consists of four steps. The details of each step can be seen as the following:

A. INVERTED INDEX FILE

An inverted index file is used to speedup the retrieval of records in response to certain search conditions [1]. Inverted index file provides a very efficient technique which allows access to records without affecting the physical placement of records on the disk.

The proposed framework use the inverted index file to find the records that includes the keywords of the query. Inverted index file consists of two fields: The first field is the keywords field, and the second field contains pointers to all records inside the database that contain these keywords [9]. The contents of keywords field in the inverted index file are sorted in an ascending order.

The inverted index file is built only once and will be re-used for all queries, whereas the remaining three steps are performed for each query.

B. RELEVANT ANSWER GENERATION

After we find the initial tables that include keywords, we must find how the keywords are related together inside a database. To do this, we need to build a relevant answer set from the database.

The process of relevant answer set aims to expand results for the query, such that, additional relevant related records could be added in response to the query. This can be achieved by relating separated records that include keywords to each others. The process of relevant answer set is achieved through a schema graph.

Each node represents a table and the arrows represent relationships among nodes. The primary and foreign keys are used to relate nodes among each other. We use threshold (T) value to find the maximum distance among nodes. In other words, if the sum of distance among nodes is greater than specific value of threshold then the generation process of nodes from the schema graph is stopped. The constructions of relevant related records are obtained from the schema graph. The relevant related records are used to improve the relevancy of the query. There is a set of assumptions that should be taken into account when constructing the relevant related records.

We explain the main idea of these assumptions by an example that is listed in section 4.

C. CONSTRUCTION OF WEIGHT_TABLES

After we find the relevant related records from the process of relevant answer set, we construct the **weight_tables**. The **weight_tables** are a set of tables that are used to compute the degree of correlation between relevant records and the keywords-based query. **Weight_tables** distinguish the correlation between relevant records and the keywords-based query. We assign a weight of each relevant record when computing degree of correlation between relevant records and the keywords-based query. The weight describes how the keywords-based query appears inside each relevant record. We use the cosine measure when computing the degree of correlation between relevant records and the keywords-based query [7]. Cosine measure assigns a score for each relevant record based on the correlation with the keywords-based query.

The **weight_tables** are used to compute the relative_keyword_weight, keyword_weight and the record weight for the set of relevant records that are considered relevant to the query. More details about these computations will be explained through an example in section 4.

D. RANKING

Ranking is an approach used to order relevant records [14, 15]. Relevant records that appear at the top of this order are considered to be more relevant. Ranking relevant records should be performed in a descending order according to the degree of correlation produced in the weight_tables and the size of relevant records. After that, the results will be displayed according to their rank.

4. FRAMWORK BY AN EXAMPLE

In this section, we give an example that illustrates how the proposed framework of keyword-based queries works and how it allows users to extract any information from a database just by providing the set of keywords. Our approach allows the user to search for any relevant information inside a database that satisfy the criteria for the query and rank the results based on the correlation with the query.

Consider a database which is called Customer-Notes database. This database consists of the following tables: **Customer**, **Supply**, and **Customer-Notes** as shown in Figure 1. The type of the relation between Customer table and Supply table is many to many relationship (M:N).

Customer

RNo	ID	Name	Address
C_1	1	John	California
C_2	2	Allis	Texas
C_3	3	Robeson	Washington

Supply

RNo	ID	Provider	Model
S_1	1	**HP**	Compaq
S_2	2	Vaio	**Sony**
S_3	3	Dell	D2500

Customer-Notes

RNo	CID	SID	Date	Notes
CN_1	1	1	22-6-2007	**Sony** very power full
CN_2	2	2	25-6-2007	Low efficient on **Sony** laptop
CN_3	3	2	27-6-2007	**Sony** high performance with **HP**
CN_4	3	3	29-8-2007	Dell low performance

Figure 1: Instant Customer-Notes database system

Suppose that the user has typed the following query 'HP SONY'. We use the "OR" operation between keywords when retrieving the relevant records. The main steps to retrieve the relevant records for the mentioned query are as the following:

A. Generate Inverted Index File

The list of the distinct keywords from the query will be identified first. After that, we use inverted index file. The entries for an inverted index file in the left hand side include distinct keyword and in the right hand side include record number, column title, and the frequency for each distinct keyword in the retrieved record only.

Note that we use a symbol for each table to distinguish it from others (i.e., S for **Supply** table, C for **Customer** table, and CN for **Customer_Notes** table).

The portion of the Inverted Index File relevant to the query 'HP' or 'SONY' can be seen in Figure 2.

HP —› (S_1, Provider, 1) (CN_3, Notes, 1)
SONY —› (S_2, Model, 1) (CN_1, Notes, 1) (CN_2, Notes, 1) (CN_3, Notes, 1)

Figure 2: Inverted Index File for the query 'HP SONY'

From Figure 2 the record (CN_3) includes all keywords ('HP SONY'). We can determine records that include all keywords by taking the intersection among all entries.

B. Relevant Answer Generation

The answer that is presented is called relevant answer. The relevant answer process step is used to expand relevant results for the query. Relevant answer is constructed by using two steps: a schema graph established and finding relevant related records.

For the first step, 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 [4, 13, 17]. We consider the default distance between two related tables to be one. We use the threshold (T) value when we define the maximum distance among tables. The maximum value of the threshold in the previous example is equal to two. A threshold value can be provided from the user but in this case it can not exceed the maximum distance among tables.

Figure 3 shows the schema graph of the Customer-Notes 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. We use the abbreviation (K) for the tables that include keywords. The tables that include keywords-based query are identified from the inverted index file (Figure 2). For clarification, table S and table CN that includes keywords are written with abbreviation S^K and CN^K. Also we use the abbreviation (N) for the tables that do not include any keyword. For example, table C written as C^N. The distance between C^N and CN^K tables is one while between C^N and S^K tables is two.

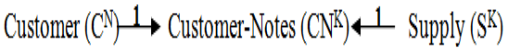

Figure 3: Schema graph for Customer-Notes database

We use the schema graph to define the relevant answer sub-tables. The relevant answer sub-tables are a set of tables that include the relevant related records for the query. The relevant answer sub-tables can be generated by using database depicted in Figure 3, and using the following three rules:

- Remove repeated relevant answer sub-tables.

- Remove relevant answer sub-tables if any leaf node does not include any keywords. Leaf node is any node that has one arrow. For example, $C^N \rightarrow CN^K$ is removed because the customer table (C^N) does not include any keywords.
- Stop generation of relevant answer sub-tables if the distances among tables are greater than the value of the threshold.

Table 1 shows the relevant answer sub-tables for keywords 'HP SONY' according to the previous three rules. The first three relevant answer sub-tables (S^K, CN^K, and $S^K \rightarrow CN^K$) are included while the last two sub-tables ($C^N \rightarrow CN^K$, and $C^N \rightarrow CN^K \leftarrow S^K$) are excluded because the leaf node C^N does not include any keywords.

Table 1: Relevant Answer Sub-Tables for the query 'HP SONY'

Relevant Answer	Status	Distance
S^K	Yes	0
CN^K	Yes	0
$S^K \rightarrow CN^K$	Yes	1
$C^N \rightarrow CN^K$	Removed	1
$C^N \rightarrow CN^K \leftarrow S^K$	Removed	2

The second step is to construct relevant related records by relating separate records from different tables via primary key and foreign key relationships. This expands the query results and gives more meaning for the relevant records. These relevant records are used to compute relevancy to the user query.

From Table 1 there are two separated relevant records with distance zero and two relevant related records with distance one. We use the inverted index file to retrieve the separated relevant records with distance zero and SQL statements to retrieve the relevant related records with distance one. For example, the relevant answer sub-tables $S^K \rightarrow CN^K$ is transferred to the following SQL statement.

```
Select  S.RNO, CN.RNO
From SUPPLY S, CUSTOMER_NOTES CN
Where (S.ID = CN.SID) and (upper(S.provider) like upper('%HP%') or upper(S.provider) like upper('%SONY%') or upper(S.model) like upper('%HP%') or upper(S.model) like upper('%SONY%')) and (upper(CN.notes) like upper('%HP%') or upper(CN.notes) like upper('%SONY%'))
```

Table 2 shows all relevant records for all relevant answer sub-tables on both distances zero and one according the previous three rules. In the next step we show the appearance of keywords inside relevant records affects on the degree of correlation.

Table 2: Relevant records for the query 'HP SONY'

Number	Relevant records	Distance
1	CN_3	0
2	S_1, CN_1	1
3	S_2, CN_2	1
4	S_2, CN_3	1

C. Weight_tables

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 weight_tables which attempt to establish the degree of correlation between the relevant records and the keywords-based query. We construct a set of four tables when we compute the degree of correlation between relevant records and the keywords-based query. The four tables are (A) keyword frequency table,

(B) relative_keyword_weight table,

(C) keyword_weight table, and

(D) record weight table.

The details of weight_tables can be seen in the following steps:

1. Keyword Frequency (KF) Table

The value of keyword frequency (KF) represents the frequency for each keyword inside each relevant record which it appears. For this purpose, we construct a table which includes relevant records, keywords-based query, and the frequency for each keyword inside each relevant record. After that, we will count the frequency for each keyword inside each relevant record.

Table 3.A shows the keyword frequency value for the keywords 'HP' or 'SONY. We can find the frequency for each keyword inside relevant records from the inverted index file. We will use the keyword frequency to compute relative_keyword_weight (RKW) for each keyword-based query.

2. Relative Keywords Weight (RKW) Table

The relative_keyword_weight value illustrates the appearance of keywords inside each relevant record respectively. We compute the relative_keyword_weight (RKW) value for each keyword-based query inside each relevant record respectively. In [3, 18] the relative_keyword_weight of each keyword inside each relevant record is not taken into account. Relative_keyword_weight each keyword inside relevant record is computed using the following formula:

$$\mathbf{RKW = KF / N}$$

Where N represents the total number of occurrences of keywords inside each relevant record.

Table 3.B shows the relative_keyword_weight for keywords 'HP' and 'SONY'. For example, RKW for 'HP' inside relevant record CN3 is 1/2 = 0.5 (1 is the frequency of 'HP' inside record CN3) and (2 is the total number of the occurrences for keywords inside relevant record CN3).

3. Keyword_weight (KW) Table

The keyword_weight (KW) value indicates the appearance of keywords inside all relevant records. The value of keyword_weight value gives a weight for each keyword that depends on the appearance inside all relevant records. We use the keyword_weight to determine which keywords appear more inside all relevant records. The keyword_weight value depends on the frequency for the keyword inside all relevant records. In other words, the keywords with more frequency inside relevant records will take higher weight value. The keyword_weight value takes range from zero to one [0, 1]. We used this range to illustrate which keywords appear more than others.

The keyword_weight value is computed using the following formula:

$$\mathbf{KW} = \sum_{1}^{N} \mathbf{RKW / N}$$

Where N represents the number of relevant records of each keyword.

Table 3.C shows the values of keyword_weight for the keywords 'HP SONY'. For example, the keyword_weight for 'HP' inside all relevant records = (0.5+0.5+0+0.33) / 4 = 0.3325 and for 'SONY' = (0.5+0.5+1+0.67) / 4 = 0.6675.

4. Record Weight Table

The record weight value measure the correlation between the relative_keyword_weight and the keywords weight in order to determine which relevant records are more relevanant to the query. The record weight value indicates the correlation between the partial weight for each keyword inside each relevant record, and the keyword_weight inside all relevant records. Record weight value takes range from zero to one [0, 1]. A higher value of weight means the record is more relevant.

The cosine measure to compute the record weight between the keywords-based query and each relevant record. That main reason for using the cosine measure is to indicate the correlation between the relative_keyword_weight and the keyword_weight. **Cosine_measure** sim (Rj, Qi) with range [0,1] measures the correlation between relevant record (Rj) and keyword-based query(Qi)in a database D [7].

We compute the record weight between each relevant record (R_j) and the keywords-based query (Q_i) by using the cosine measure, as can be seen in the following formula:

$$RW(Rj,Qi) = \frac{\sum(RKW)(KW)}{\sqrt{\sum(RKW)^2 \sum(KW)^2}}$$

Table 3.D shows the record weight (RW) for all the records. For example, RW between CN_3 and 'HP SONY' is:

$$\mathbf{RW\ (CN_3, 'HP\ SONY')} = \frac{(0.5*0.3)+(0.5*0.7)}{\sqrt{(0.5^2+0.5^2)*(0.3^2+0.7^2)}} = 0.926$$

Table 3: Weight_tables for the query 'HP SONY'

(A) Keywords-Frequency Table

Relevant Records	HP	SONY	Total
CN_3	1	1	2
S_1, CN_1	1	1	2
S_2, CN_2	0	2	2
S_2, CN_3	1	2	3

(B) Partial-Keywords-Weight Table

Relevant Records	HP	SONY	Total
CN_3	1/2	1/2	2
S_1, CN_1	1/2	1/2	2
S_2, CN_2	0	2/2	2
S_2, CN_3	1/3	2/3	3

(C) Keywords-Weight Table

Relevant Records	HP	SONY
CN_3	1/2	1/2
S_1, CN_1	1/2	1/2
S_2, CN_2	0	2/2
S_2, CN_3	1/3	2/3
K-W	0.3	0.7

(D) Records-Weight Table

Relevant Records	Records-Weight	Distance
CN_3	0.926	0
S_1, CN_1	0.926	1
S_2, CN_2	0.918	1
S_2, CN_3	1.001	1

From Table 3.D we note the following observations: On distance zero, the different relevant record (CN_3) includes both keywords 'HP' and 'SONY'. On distance one the record weight for (S_2, CN_3) is greater than (S_1, CN_1) and (S_2, CN_2) because (S_2, CN_3) includes many different keywords. The record weight for (S_1, CN_1) is greater than (S_2, CN_2). The main reason for that (RW (S_1, CN_1) > RW (S_2, CN_2)) is the (S_1, CN_1) includes both keywords 'HP' and 'SONY' while (S_2, CN_2) includes single keywords 'SONY' in both records. Of course, the appearance of different keywords inside relevant record affects the record weight.

In general, relevant records that include many different keywords are considered more relevant than those with only a single keyword, or keywords with low weight values. The results of **weight_tables** are used to indicate the correlation between relevant records and the keywords.

D. Ranking

Ranking relevant records is the final step of the framework. To rank relevant records, we assign a score for each relevant record as an estimation of relevancy to the given query. The record weight and the size of relevant record should be taken into account when computing (A) Keywords-Frequency Table the score of relevant record.

The score of each relevant record is computed by using the following formula:

$$\mathbf{Score} = \frac{RW}{S}$$

Where S represents the size of relevant record. Table 4.A shows the score of each relevant record. After computing the score, the relevant records are sorted in a descending order as can be seen in Table 4.B.

Table 4: Ranking Relevant Records for the query 'HP SONY'

(A) Score for Relevant Records

Relevant	Record-Weight	Size	Score
CN_3	0.926	1	(0.926 / 1) = 0.926
S_1, CN_1	0.926	2	(0.926 / 2) = 0.463
S_2, CN_2	0.918	2	(0.918 / 2) = 0.459
S_2, CN_3	1.001	2	(1 / 2) = 0.5

(B) Sorting Relevant Records in a descending order

Relevant Records	Rank	Distance
CN_3	0.926	0
S_2, CN_3	0.5	1
S_1, CN_1	0.463	1
S_2, CN_2	0.459	1

From Table 4.B we note the following observations:

- On distance zero, there is one separated relevant record retrieved (CN_3). The relevant record CN_3 takes greatest rank than others because all keywords appear in one record.
- On distance one, there are three relevant related records ((S_2, CN_3), (S_1, CN_1), and (S_2, CN_2)). The relevant related record (S_2, CN_3) take rank greater than (S_1, CN_1), and (S_2, CN_2) because (S_2, CN_3) includes many keywords that are splitted over two records. Relevant related record (S_1, CN_1) take rank greater than (S_2, CN_2) because (S_1, CN_1) includes a single keyword 'Sony' while (S_1, CN_1) includes two different keywords 'HP' and 'SONY' that are distributed on two records. In other words, answers needed by users are not limited to individual relevant separated records, but results assembled from joining separate records together. For example, when related separated relevant records (S_2), and (CN_3) on distance zero together we get relevant related records on distance one to becomes (S_2, CN_3).

5. EXPERIMENTAL RESULTS

We used a real data set of DBLP [5] database, and applied a set of queries on this data to evaluate the performance of the enhancement approach.

The DBLP database includes the following tables with their corresponding structure:
Publisher (Publisher_ID, Name)
Proceeding (Proceeding_ID, Title, Year, Series_ID, Publisher_ID)
Series (Series_ID, Title)
InProceeding (InProceeding_ID, Title, Pages, Proceeding_ID)
RelationPersonInProceeding (InProceeding_ID, Person_ID)
Person (Person_ID, Name)

The experimental results were run on a PC machine with 3000 MHZ, 512 KB RAM, and using windows XP. We used Oracle 10g and Developer suite two to implement the enhancement approach.

To test the performance, we wrote twenty different keywords that are listed in Table 5. For each query we used a list of different keywords.

Table 5: Query used for the test

Query Number	Keywords Query
Q_1	Mobile Radio System.Markus Anja Klein
Q_2	Dynamic Cell Planning for Data Transmission Gfeller Weiss
Q_3	Cryptographic Primitives for Information Authentication
Q_4	Real Time Protocols Lann
Q_5	An Algebraic Specification of Process Algebra Sjouke Mauw

Q_6	Broadband Communications Services Zhili Sun
Q_7	Database Horlait principle
Q_8	Nelson networks notes Pires Weber
Q_9	Daniel Thalmann algorithm science
Q_{10}	Laurent Dairaine Elsevier notes algorithm
Q_{11}	Multimedia Traffic Control Cleevely North-Holland
Q_{12}	query optimization Cornelius Frankenfeld Elsevier Lecture
Q_{13}	Computer Science paul Yoon Springer
Q_{14}	Multimedia Mail Service Prototype
Q_{15}	Robust Reconstruction Daniel Thalmann Elsevier
Q_{16}	Linear Algebraic Greene North science
Q_{17}	Lecture Notes Anna Stefani Elsevier Traffic Control
Q_{18}	Multimedia Document Architecture
Q_{19}	introduction network Hewson Springer lecture
Q_{20}	Nelson Multimedia Thalmann Services

After that, we draw the database graph as can be seen in Figure 4

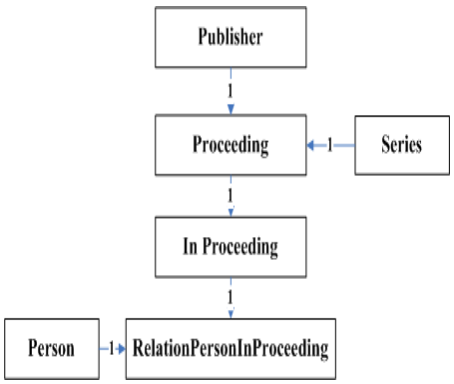

Figure 4: Database graph for DBLP database

We used the database graph to define the distance among tables, and to define the maximum value of the threshold (T) when we generate relevant answer sub-tables for all queries. From Figure 4 the maximum value of the threshold (T) is equal to four because the maximum distance among the nodes is equal to o four. We choose the three values of the threshold (T) (0, 2, and 4) when we generated relevant answer sub-tables for each query. The main reason for using these three values was to was to to show how the values of the threshold (T) affect on the number of relevant answer sub-tables, and relevant records.

We measured the execution time when retrieving the relevant answer sub-tables for each query. Execution time was calculated by taking average time after executing each query 500 runs.

Figure 5 shows all queries, and the number of relevant answer sub-tables for each value of the threshold (T).

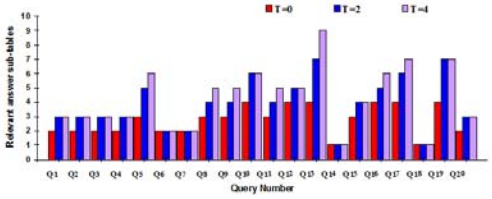

Figure 5: Relevant answer sub-tables for each query on three values of threshold (T)

For Figure 5, let use pickup any query for illustration. For example, Q13 shows the number of relevant answer sub-tables on three values of the threshold (T) as follows:

- Threshold zero: The total number of relevant answer sub-tables that include keywords is equal to four. In other words,

all the keywords appear in four different relevant answer sub-tables.

- Threshold two: The total number of relevant answer sub-tables is equal to seven. In other words, there are three additional relevant related answer sub-tables retrieved by using join constraints when the distance of threshold equals to two.
- Threshold four: The total number of relevant answer sub-tables is equal to nine In other words, there are two additional relevant related answer sub-tables retrieved when the distance of threshold equals to four.

From Figure 5 we note the following observations: The relevant answer sub-tables in some queries (Q_6, Q_7, Q_{14}, and Q_{18}) are not changed over the three values of the threshold (T). For example, in query Q6, the number of relevant answer sub-tables is equal to two over the three different values of threshold (T). The relevant answer sub-tables in the remaining queries (Q_1, Q_2, Q_3, Q_4, Q_5, Q_8, Q_9, Q_{10}, Q_{11}, Q_{12}, Q_{13}, Q_{15}, Q_{16}, Q_{17}, Q_{19}, and Q_{20}) are increased over the three values of threshold (T). For example, in query Q_8 the number of relevant answer sub-tables equal to three for threshold zero, four for threshold two, and five for threshold four. In other words, the number of relevant answer sub-tables increased over the three values of threshold (T). Of course, there are many relevant answer sub-tables that could be retrieved when the value of threshold (T) is changed.

Table 6 shows execution time for each query on different values of the threshold and the average time for the corresponding queries for the three values of the threshold.

Table 6: Execution time for each query on three values of threshold (T)

Query Number	Execution Time			Average Time
	T = 0	T = 2	T = 4	
Q_1	0.4	0.91	1	0.77
Q_2	0.34	0.81	0.99	0.71
Q_3	0.45	0.78	0.98	0.74
Q_4	0.43	0.98	1	0.80
Q_5	0.34	0.98	1.06	0.79
Q_6	0.4	1.02	1.04	0.82
Q_7	0.34	0.95	1.01	0.77
Q_8	0.4	1	1.1	0.83
Q_9	0.4	1.04	1.11	0.85
Q_{10}	0.5	0.96	1.02	0.83
Q_{11}	0.2	1.01	1.16	0.79
Q_{12}	0.35	1.02	1.08	0.82
Q_{13}	0.4	1.02	1.01	0.81
Q_{14}	0.42	0.76	0.87	0.68
Q_{15}	0.35	0.99	1.03	0.79
Q_{16}	0.3	1.01	1.07	0.79
Q_{17}	0.3	1.03	1.1	0.81
Q_{18}	0.4	0.8	0.98	0.73
Q_{19}	0.4	1.01	1.17	0.86
Q_{20}	0.3	1.01	1.11	0.81

From table 6 we note the following observations: Queries Q_{14}, Q_2, Q_{18}, and Q_3 take less average execution time, For the following reasons: (A) Q_{14}, Q_2, and Q_{18} the number of relevant answer sub-tables over the three values of the threshold are not changed. (B) Q_3 the number of relevant answer sub-tables is equal two on threshold zero, three on threshold two and four. Query Q_{19} takes greater average execution time than all queries because the keywords appeared in many tables during search and retrieve relevant answer sub-tables. Average execution time increased when there were many relevant answer sub-tables that were being retrieved, and when the values of threshold (T) were being changed.

Table 7 shows the total number of relevant records for each query on different values of the threshold (T), and average number of relevant records for the corresponding queries over the three values of the threshold (T). The number of relevant records for each query is defined from the relevant answer sub-tables that are found in the previous table (Table 5).

Table 7: Total number of relevant records for each query on three values of threshold (T)

Number	T = 0	T = 2	T = 4	Relevant Records
Q_1	11	12	12	11.7
Q_2	324	328	328	326.7
Q_3	276	278	278	277.3
Q_4	69	70	70	69.7
Q_5	367	378	381	375.3
Q_6	36	36	36	36.0
Q_7	12	12	12	12.0
Q_8	35	54	57	48.7
Q_9	61	104	112	92.3
Q_{10}	55	101	101	85.7
Q_{11}	62	85	86	77.7
Q_{12}	11	19	19	16.3
Q_{13}	40	95	125	86.7
Q_{14}	57	57	57	57.0
Q_{15}	13	14	14	13.7
Q_{16}	22	36	37	31.7
Q_{17}	38	61	68	55.7
Q_{18}	57	57	57	57.0
Q_{19}	73	192	192	152.3
Q_{20}	48	49	49	48.7

From Table 7 we note the following observations:

1) Queries (Q_5 Q_8, Q_9, Q_{11}, Q_{13}, Q_{16}, and Q_{17}) the number of relevant records is increased when the value of the threshold is changed. In other words, there are many relevant records retrieved when the value of threshold (T) was changed.

2) Queries (Q_6, Q_7, Q_{14}, and Q_{18}) the number of relevant records is not changed over the three values the threshold. In other words, there are no more relevant records retrieved when the value of the threshold is changed.

3) Queries (Q_1, Q_2, Q_3, Q_4, Q_{10}, Q_{12}, Q_{15}, Q_{19}, and Q_{20}) the number of relevant records is not changed when the values of the threshold equal two and four

Table 8 shows the appearance of each keyword inside the query (Q_{19}) when the value of threshold equal to zero.

Table 8: Keywords occurrences for the query Q_{19} on threshold zero

Keywords Query	Frequency
Introduction	24
Network	45
Hewson	1
Springer	1
Lecture	2

Figure 6 shows the number of relevant records for each query on the three values of the threshold.

Figure 6: Relevant records for each query on three values of threshold (T)

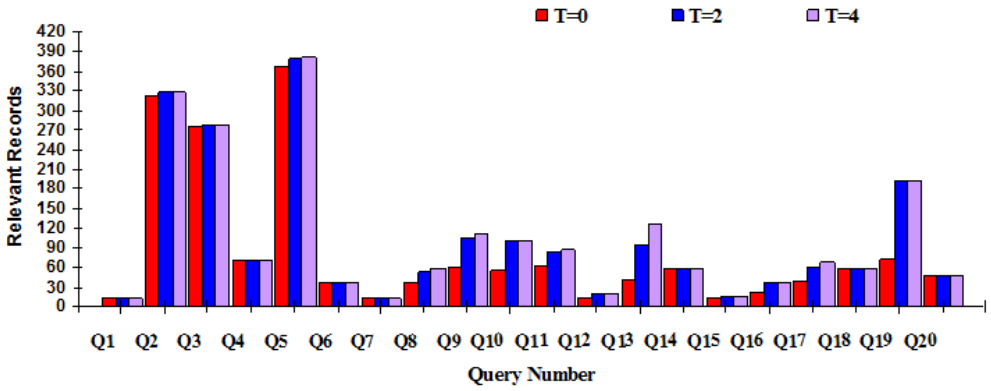

Figure 7 shows average relevant records for each query over the three values of the threshold (T).

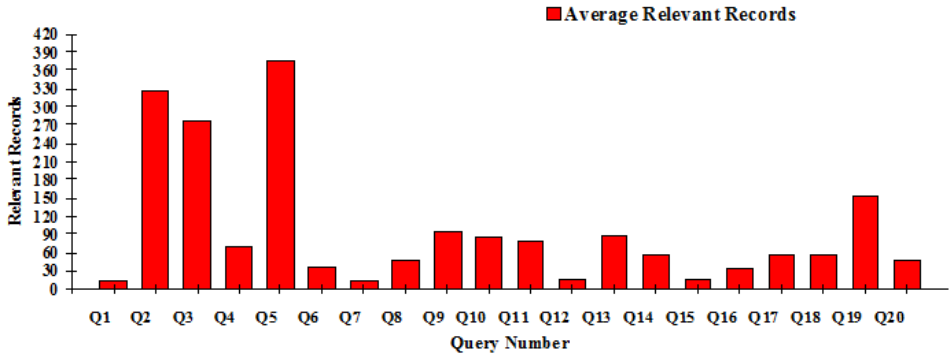

Figure 7: Average relevant records for each query

6. CONCLUSION

We introduced a framework for processing keyword-based queries in relational databases that takes into account two aspects: efficiency and effectiveness. Efficiency is improved through two steps. First step is by using inverted index file. The second step is through relevant answer generation. Relevant answer process is used to relate separate records to each other to expand results for a given query.

We used a schema graph to build relevant answer sub-tables, and to relate separated relevant records to each other. Constructions of relevant records are obtained from the schema graph. A threshold (T) is used to define the maximum distance among tables, and to define the termination process of relevant answer generation. We choose the three different values of the threshold (T) to show how these values affect on the number of relevant records.

Effectiveness is improved through the construction of weight_tables, and ranks the relevant records. Construction of **weight_tables** is used to assign weight values for both keywords-based query, and the relevant records that include keywords. The results of weight_tables give an indication of the correlation between keywords-based query, and the relevant records. Ranking of relevant records is used to order the relevant records in a descending order. When ranking the relevant

records, the size of relevant records and the record weights should be taken into account. The relevant records that include many keywords take higher rank than other relevant records that include a single keywords. There are two case scenarios when we generate relevant records. Best case scenario occurs when the number of relevant records is changed over the three values of threshold (T). Worst case scenario occurs when the number of relevant records is not changed over the three values of threshold (T). The value of keyword_weight in the best case scenario is changed because the keyword_weight depends on the occurrences of keywords inside relevant records.

REFERENCES :

[1] Alistair Moffat, and Justin 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.

[2] Bhavana Dalvi, "Keyword Search on Relational and Semi-structured Databases," MTech Seminar Report.

[3] Fang Liu, Clement Yu, Weiyi Meng, Abdur Chowdhury "Effective Keyword Search in Relational Databases," In Proceedings of Association for Computing Machinery (ACM), and Special Interest Group on Management Of Data (SIGMOD) International Conference on Management of Data, pp. 563-574, 2006.

[4] Gaurav Bhalotia, Charuta Nakhe, Arvind Hulgeri, Soumen Chakrabarti, and Shashank Sudarshan, "Keyword Searching and Browsing in Databases Using BANKS," In Proceedings of International Conference on Data Engineering (ICDE), pp. 431–440, 2002.

[5] Http://www4.wiwiss.fu-berlin.de/bizer/D2RQ/benchmarks/index01.html.

[6] Lu Qin, Jeffrey Xu Yu, Lijun Chang, and Yufei Tao, "Querying Communities in Relational Databases," In Proceedings of International Conference on Data Engineering (ICDE), pp. 724-735, 2009.

[7] Margaret Dunham, "Data Mining Introductory and Advanced Topics," Prentice Hall, 2003.

[8] Nandlal Sarda, and Ankur Jain, "Mragyati: A System for Keyword-based Searching in Databases," Journal of Computing Research Repository (CoRR), Vol. cs.DB/0110052, 2001.

[9] Ophir Frieder, David Grossman, Abdur Chowdhury and Gideon Frieder, **"**Efficiency Considerations for Scalable Information Retrieval Servers**,"** Journal of Digital Information (JDI), Vol. 1, No. 5, 2000.

[10] Oracle database website, Http://www.oracle.com.

[11] Relational Database website, www.c2.com/cgi/wiki?RelationalDatabase.

[12] Ricardo Baeza-Yates, and Berthier Riberio-Neto, "Modern Information Retrieval," Addison Wesley, 1999.

[13] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das, "DBXplorer: A system for Keyword-Based Search Over Relational Databases," In Proceedings of International Conference on Data Engineering (ICDE), pp. 5–16, 2002.

[14] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis, "Automated Ranking of Database Query Results," In Proceedings of Conference

on Innovative Data Systems Research (CIDR), 2003.

[15] Surajit Chaudhuri, Gautam Das, Vagelis Hristidis, and Gerhard Weikum, “Probabilistic Ranking of Database Query Results,” In Proceedings of Very Large Data Bases (VLDB), pp. 888-899, 2004.

[16] Text database website, www.psychcrawler.com.

[17] Vagelis Hristidis, and Yannis Papakonstantinou, "DISCOVER: Keyword Search in Relational Databases," In Proceedings of Very Large Data Bases (VLDB), pp. 670-681, 2002.

[18] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou, "Efficient IR-Style Keyword Search Over Relational Databases," In Proceedings of Very Large Data Bases (VLDB), pp. 850-861, 2003.