Searching Databases with Keywords*

Shan Wang and Kun-Long Zhang

School of Information, Renmin University of China, Beijing, 100872, P.R. China

E-mail: swang@ruc.edu.cn

Abstract Traditionally, SQL query language is used to search the data in databases. However, it is inappropriate for end-users, since it is complex and hard to learn. It is the need of end-users that searching in databases with keywords, like in web search engines. This paper presents a survey of work on keyword search in databases. It also includes a brief introduction of the SEEKER system which we have developed.

Keywords Relational Databases, Keyword Search, Hidden Web, Information System Integration

1 Introduction

Although both database and information retrieval systems focus on searching data, methods to solve the problem are very different. Database systems[1] search structured data with complex query languages. Its results are sound and complete, and all results are equally good. Information retrieval systems[2] search unstructured data by keywords. Its results are usually imprecise and incomplete, and some results are more relevant than others. Fig.1 shows the difference between these two types of systems.

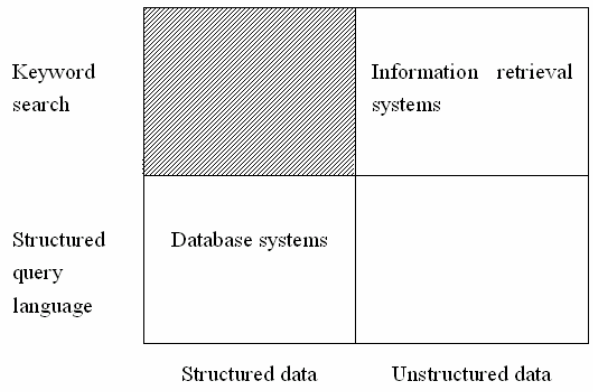

Fig.1. Database and information retrieval systems

Two questions are raised from Fig.1. Can we search databases with keywords? Is it necessary to searching databases with keywords? The answer to the first question is positive. In this paper, we'll present a survey of work on keyword search in databases. However, let's discuss the second question first.

Internet users usually search information with search engines. An internet user types some keywords as a query. The search engine returns a sorted list of relevant documents as the result. It is expected that database users would like to query databases in the same way. To search databases with keywords, it is not necessary to know the database schema and database query languages such as SQL.

In fact, only a few data on the Web can be found by search engines. Most of data on the Web are stored in databases and "hidden" to search engines because special search interfaces are needed to

* Survey

The work was supported by the National Natural Science Foundation of China under Grant Nos. 60473069, 60496325 and the National High Technology Development 863 Program of China under Grant No. 2003AA4Z3030.

find them. Data stored in databases form the Hidden Web[3]. Lawrence and Giles estimated in 1998 that 80% of all the data on the Web were stored in Hidden Web[4]. Bergman found in 2001 that the amount of information stored in Hidden Web is 400 or 550 times larger than the visible Web[5]. The Hidden Web problem is caused by the mismatch of search interfaces between search engines and databases. If database systems support keyword search, publishing and searching a database is expected to be more simple and easy.

Integrating different classes of information systems is a research focus in recent years[6]. Modern information systems should manage many kinds of data, such as structured relational data, semistructured XML documents and unstructured text documents. However, different query languages must be used for searching different kinds of data, such as SQL for relational databases, XQuery for XML documents and keyword search for text documents. Thus, to manage many kinds of data by one system, one of challenges is to provide a unified query language. Keyword search is expected to play this role.

The rest of the paper is organized as follows: Section 2 overviews and classifies related work. Section 3 identifies the key techniques of database retrieval systems which support searching databases with keywords. Section 4 outlines directions for future work. Section 5 concludes the paper and gives a brief introduction to the SEEKER system which we have developed.

2 Related Work

Hulgeri et al. surveyed the work related to keyword search in databases in 2001[7]. After several years, notable progress has made in this field. In this section, we first overview several database retrieval systems, then classify the related work

2.1 BANKS

A database can be modeled as a database graph which includes a scheme graph and a data graph. BANKS[7][8] uses directed graph to do keyword search. In BANKS' data graph, each tuple in the database is represented by a node, and each "foreign key → primary key" link between two tuples is represented by a directed edge.

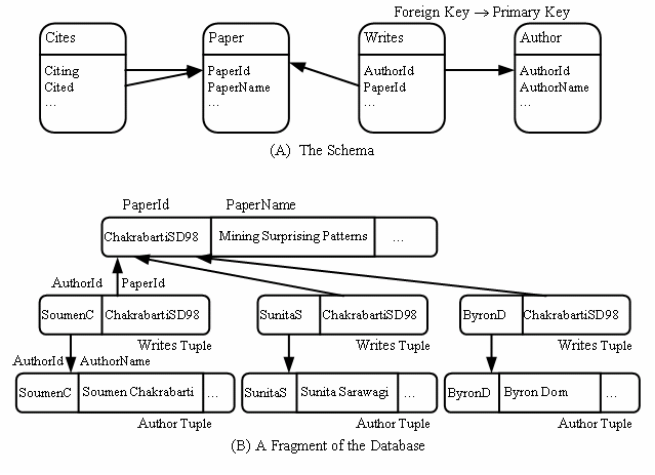

Fig.2. The DBLP bibliography database

Fig.2(A) shows the schema graph of the DBLP bibliography database. In the schema graph, nodes represent tables and edges represent "foreign key → primary key" dependencies between tables. Fig.2(B) shows a fragment of the data graph of the DBLP bibliography database. It means that Soumen Chakrabarti, Sunita Sarawagi and Byron Dom coauthor a paper. BANKS creates a back edge for each "foreign key → primary key" edge and the back edges are not shown in Fig.2(B).

Generally, a query consists of $n \geq 1$ search terms $t_1, t_2, \ldots, t_n$. For each search term t_i a relevant

nodes set S_i can be generated. A node is relevant to a search term if it contains the search term as part of an attribute value or metadata (such as column, or table names). Then the relevant nodes sets corresponding to t_1, t_2, …, t_n is S_1, S_2, …, S_n. BANKS defines an answer as a connection tree. The connection tree is a directed tree which includes at least one node from each S_i. The root of a connection tree is called information node.

BANKS employs a heuristic algorithm to search for all information nodes. In the example from Fig.2(B), let the search terms be Soumen, Sunita and Byron, which are leaves of the tree. The heuristic algorithm will traverse the data graph by using Dijkstra's single source shortest path algorithm starting from each leaf. Each copy of the single source shortest path algorithm will visit the root node. Then, the tree in Fig.2(B) is an answer tree and its root node is an information node which represents a paper written by three authors.

In BANKS, each answer tree has to be assigned a relevance score and answers have to be presented in decreasing order of that score. To find the relevance score of an answer tree, every edge in the tree is assigned a weight. The edge weights are normalized. Let $W(e)$ be the weight of edge e, the normalized edge score $E_{score}(e)$ of an edge e is computed as follows:

$$E_{score}(e) = W(e) / W_{min}$$

where W_{min} is the minimum edge weight in the data graph. The overall edge score E_{score} is

$$E_{score} = 1 / (1 + \Sigma_e E_{score}(e)) .$$

Further, every node in the tree is also assigned a weight. The node weights are normalized. Let $N(v)$ be the weight of node v, the normalized node score $N_{score}(v)$ of a node v is computed as follows:

$$N_{score}(v) = N(v) / N_{max}$$

where N_{max} is the maximum node weight in the data graph. The overall node score N_{score} is the average of all node scores. Finally the score function Score(T) for an answer tree T is

$$Score(T) = (1 - \lambda) E_{score} + \lambda N_{score}$$

where λ is a constant. Thus, if all edges and nodes are assigned with the same weight, it is easy to know that the relevance score of an answer tree is inversely proportional to the size of the tree.

The last step of a keyword query is to present the search result. BANKS uses a nested table to present an answer tree. For example, the answer tree in Fig.2(B) is presented as such a nested table: three tables representing intermediate nodes are nested in the table representing root node, and each of them contains a table representing a leaf node. BANKS also provides a rich interface to browse data stored in a relational database.

2.2 DBXplorer

Unlike BANKS, DBXplorer[9] does not use directed data graph. Instead, it uses undirected schema graph to do keyword search. DBXplorer defines a search result as a join of tuples, which contains all keywords.

Given a query consisting of a set of keywords, it is answered by DBXplorer as follows. 1) The symbol table is searched to find the tables, and columns/rows of the database that contain the query keywords. The symbol table functions as an inverted list and it is built by preprocessing the database before the search; 2) Enumerating join trees according to schema graph. A join tree is a sub-tree

of schema graph. It satisfies two conditions: one is that the table corresponding to a leaf node contains a query keyword at least; another is that every query keyword is contained by a table corresponding to a leaf node. If all tables in a join tree are joined, the results might contain rows having all keywords. 3) For each enumerated join tree, an SQL statement is constructed to join the tables in the tree and select those rows that contain all keywords. The final rows are ranked and presented to the user.

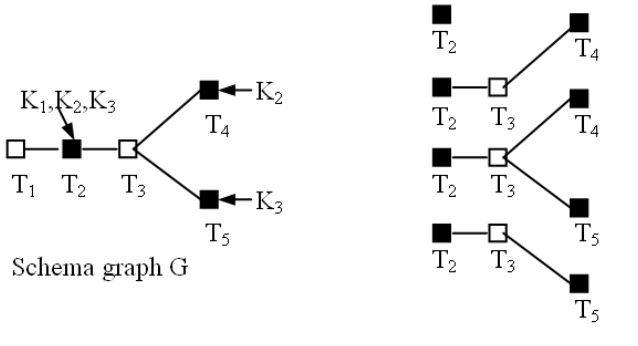

Fig.3. Join trees

The process of enumerating join trees includes three steps. First, a new schema graph G' is generated by removing the leaf nodes that do not contain any keyword from the schema graph G. Then a leaf node in G' is chosen by a heuristic method. Finally, A breadth-first traversal of G' is started from the chosen leaf node and it outputs all join trees. For example, consider the schema graph G in Fig.3 over five tables. Let the query keywords be K_1, K_2, and K_3, table T_2 contains all three keywords, table T_4 contains K_2, and table T_5 contains K_3. A black node represents a table contains at least one keyword. Then the enumeration algorithm starting from leaf node T_2 enumerates four join trees shown in the right part of Fig.3.

The score function that DBXplorer uses to rank results is very simple. The score of a result is the number of joins involved. Because joins involving many tables are harder to be understood, the score of a join tree is proportional to its size.

DBXplorer uses simple user interfaces to present results. It describes the join trees with text and displays the join results in a table.

2.3 DISCOVER

DISCOVER[10][11] can be regarded as an improvement of DBXplorer.

DISCOVER can answer the query example in section 2.1 correctly, but DBXplorer can not. DISCOVER enumerates candidate networks according to tuple set graph. Although the enumeration algorithm of DISCOVER is also based on breadth-first search algorithm, it does not prune the tuple set graph first. The concept of candidate networks is corresponding to the concept of join trees. The tuple set graph is built on the basis of schema graph and it includes tuple set related to keywords. Fig.4 shows the tuple set graph used by DISCOVER to answer the query example in section 2.1. Notice that the edge's direction is from primary key to foreign key and the Cites table in Fig.2 is omitted for simplification.

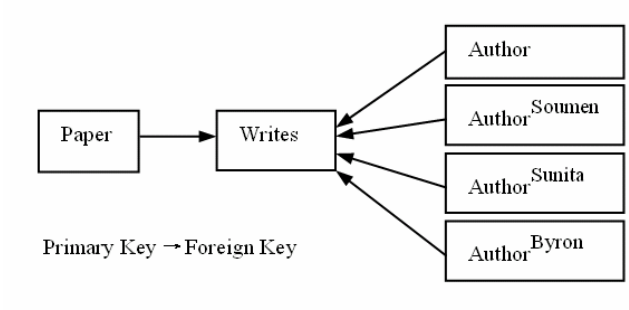

Fig.4. Tuple set graph

DISCOVER support both AND-semantics and

OR-semantics queries. AND-semantics queries require a query result contains all search terms, but OR-semantics queries require a query result contains one of search terms. BANKS and DBXplorer only support AND-semantics queries.

Modern database systems allow users to create full-text indexes on single attributes to perform keyword queries. For example, Oracle[12] and IBM DB2[13] use standard SQL to create full text indexes on text attributes of relations. Microsoft SQL Server[14] provides tools to generate full text indexes which are stored as files outside the database. DISCOVER not only exploits the full text search ability of modern database systems to find the relations and tuples that contain query keywords, but also uses the IR-style relevance ranking functionality of them to define its own ranking scheme. The score function used by DISCOVER is

$$Score(T,Q)=\frac{\sum_{a_i\in A} Score(a_i,Q)}{size(T)}$$

where Q is a keyword query, T is a result returned by Q, A is a set of all textual attribute values of T, $Score(a_i,Q)$ is the score of attribute a_i that is determined by database system, and $size(T)$ is the number of joins involved to generate the result T.

A keyword search is often a top-k query. In a top-k query, users are only interested in a small number of results *k* that best match the given condition. DISCOVER efficiently execute top-k queries by avoiding creating all query results.

2.4 ObjectRank

ObjectRank[15] differs largely from the systems we have discussed above. ObjectRank defines a result as a node of the data graph. It scores a result on the basis of PageRank algorithm[16] which is used by Google web search engine to rank web pages.

PageRank algorithm is a method of link analysis. Link analysis methods show that the interconnections between nodes have lots of valuable information. For example, the importance of a paper is not determined by the paper it cited, but by the paper cites it. To understand the PageRank algorithm, we can think that a node is a reservoir, the importance of the node is the amount of water stored in the reservoir, the link between the nodes is the channel between reservoirs, and the link semantics determine how water flows from one reservoir to another. At first every reservoir has an initial amount of water and finally the whole system will get to a steady consistent state. PageRank algorithm is used to calculate the importance of each node at this final steady consistent state.

In PageRank algorithm, a node may become important at the final state, though it is unimportant at the initial state. Because of this character of PageRank algorithm, ObjectRank can find relevant nodes that do not contain any query keyword.

Google only uses PageRank algorithm to determine the global importance of a web page which is not relevant to query keywords. However, for a node, ObjectRank calculates not only its global importance but also its keyword-specific relevance with PageRank algorithm. The score of a node *v* with respect to a keyword *w* is

$$r^{w,G}(v)=r^{w}(v)\,(r^{G}(v))^{g}$$

where $r^{w}(v)$ is the keyword-specific relevance, $r^{G}(v)$ is the global importance, and g is a constant defined by user. In ObjectRank, the score of a node with respect to a keyword is calculated by the prepos-

sessing module and is stored in a score index file. The calculation of scores is a computing intensive operation. ObjectRank proposes some efficient computation methods.

ObjectRank supports not only AND-semantics queries but also OR-semantics queries. For example, In ObjectRank, the score of a node v with respect to an AND-semantics keyword query $w_1, w_2, \ldots, w_m$ is

$$r_{AND}^{w_1,w_2,\ldots,w_m}(v) = \prod_{i=1,\ldots,m} r^{w_i}(v).$$

ObjectRank also supports top-k queries.

2.5 Classification of Related Work

Database retrieval systems have got attention of many researchers. The related work can be classified based on the characters of search algorithms and search results. Fig.5 shows our classification.

A search result is either one node or many nodes of the data graph. If a search result contains many nodes, these nodes should be connected together to let users understand the logic meaning of the result. For example, DataSpot and BANKS use information node to reveal the meaning of a search result. DBXplorer and DISCOVER define a result as a join of nodes. DbSufer finds a trail which is a browsing path from the first node to the last node.

Search the data graph	DataSpot[17], Proximity Search[18], BANKS[7][8], DbSurfer[19]	ObjectRank[15]
Not search the data graph	Mragyati[20], DBXplorer[9], DISCOVER[10][11]	Database full text search[12][13][14], SISQL[21][22]
	A search result contains many nodes	A search result contains one node

Fig.5. Classification of related work

A search algorithm may or may not search the data graph. If a search algorithm does not search the data graph, it may search the schema graph and generate SQL statements to get intermediate or final results.

It is the limitation of database full text search systems and SISQL that all query keywords must be contained in the same tuple.

3 Key Techniques

In this section, we summarize the key techniques of existing database retrieval systems from several aspects. We also discuss the problems that need to be solved in database retrieval systems.

3.1 Architecture

All database retrieval systems have two modules: preprocessing module and query module.

Preprocessing module is in charge of preprocessing databases before executing the first user query. It generates all kinds of data needed by query module to speed the query processing. The data generated by preprocessing module is stored either in main memory or on hard disk. For example, in BANKS, the data graph is stored in main memory. In DBXplorer, the symbol table is stored on hard disk. The complexity of preprocessing module depends on the database retrieval system. For example, in DISCOVER, the full text search ability of database system is exploited to do preprocess. In ObjectRank, most calculation is done by preprocessing module.

Query module is in charge of query processing. At the first step, the query module parses user's input to find query keywords and query semantics.

Then, the query module executes the search algorithm to generate results. The work to be done in this phase depends on the database retrieval system. For example, in ObjectRank, the task is to calculate the score of every node with respect to the query and generate the final results. In DISCOVER, the work include generating tuple set graph, enumerating candidate networks, and executing the top-k query.

Finally, the query module presents the query results to users. Section 3.5 discusses results presentation in detail.

As an example, the architecture of our SEEKER system is presented in section 5. The preprocessing module that generates the full text index is omitted. The query module is divided into five blocks as shown in Fig.7.

3.2 Data Model

Most database retrieval systems model a database as a database graph. A database graph is either directed or undirected. It consists of a schema graph and a data graph. For example, BANKS searches a directed data graph to find information nodes. DBXplorer searches an undirected schema graph to generate join trees. Typically, in a schema graph, nodes represent relations, and edges represent primary key-foreign key dependencies between two relations. In a data graph, nodes represent tuples, and edges represent primary key-foreign key dependencies between two tuples.

There are some significant advantages to represent a database as a graph. Both the Web and an XML document can be represented as a graph. The similarity between a web graph and a database graph hints that the algorithms used by a search engine may be used by a database retrieval system. ObjectRank is such a success example. Because both XML documents and databases are modeled as graphs, the database retrieval systems are easily extended to retrieval XML documents. Actually, such an extension is applied to BANKS and DISCOVER.

Although searching on the data graph has better performance than searching on the schema graph, there are some disadvantages to search on the data graph. Scalability is a problem. Large amount of storage space is needed to store the data graph for a large database, but the capacity of main memory is limited. Database update causes another problem because any change in the underlying database implies a change in the derived data graph.

3.3 Query Language

A query is the formulation of a user information need. A database retrieval system should define the syntax and semantics of the query language it supports. The most elementary keyword-based query form is single-word query. Other basic keyword-based query forms include phrase query, proximity query, Boolean query, and pattern matching. Not all query forms are supported by current database retrieval systems. BANKS supports only AND-semantics queries. DBXplorer supports AND-semantics queries and sub-string matches. DISCOVER and ObjectRank support both AND-semantics and OR-semantics queries.

The default query scope includes all text attributes of relations. Although most systems declare their query scope can be easily expanded to include

metadata such as table names and column names, problems exist. For example, in BANKS, if a keyword matches a relation name, all tuples of the relation will be relevant to the keyword. Because BANKS initiates one thread for one keyword-matched node, if the relation contains lots of tuples, too many threads will be created. Another expansion of query scope is to include non-textual attributes, such as numerical attributes and date attributes. For example, a user wants to search the papers written by some author in 1992. Agrawal and Srikant[23] propose an approach to match numbers without attribute names. Their idea is a good start to do such a query scope expansion.

Sometimes it is useful to let user give schema information in query, though it violates the rule of keyword search in databases that users do not need to know the database schema. For example, a user can use "year:1992" to state an attribute name "year" and an attribute value "1992". Also the query language can be expanded to include non-textual numeric expressions. For example, query "year:>1992" is used to find those tuples whose year attribute value is greater than 1992.

An interesting problem is structured database retrieval which is analogous to structured text retrieval. For example, an end-user may search the DBLP bibliography database with specified keywords for a paper that is cited at least once by other papers. To support this kind of queries, the query language must be expanded elaborately.

3.4 Definition of Query Result

A query result can be defined as a tuple from a relation, or a combination of many tuples from many relations. For example, in ObjectRank, a result is defined as a single tuple, this tuple is relevant to the query and may not contain any query keyword. In DBXplorer, a result is defined as a join of many tuples and it must contain all query keywords.

Because query results are presented to end-users, it is important to define a query result as a meaningful information unit. For example, when a query result consists of many tuples, these tuples must be connected together to have a specific meaning.

3.5 Ranking of Results

Ranking is central to database retrieval systems. In a keyword query, each query result is assigned a relevance score and all results are presented in decreasing order of that score. There are three ranking factors considered by existing database retrieval systems.

The first ranking factor is the IR score of attribute values. The IR score of an attribute value is computed according to the number of keywords it contained. Traditional information retrieval weighting methods, such as *tf-idf* weighting, can be used to compute the IR score. In DISCOVER, the IR score of an attribute value is computed by database system. In SEEKER, the IR score of a tuple value is also computed by database system.

The second ranking factor is the structure or semantics of result trees. Result trees refer to connection trees used by BANKS, join trees used by DBXplorer, and candidate networks used by DISCOVER. A result tree is scored by its size, i.e. the number of nodes or edges, in BANKS, DBXplorer and DISCOVER. In addition, it is useful to score a

result tree by its semantics. For example, if it is more important to search a paper's author in DBLP bibliography database, the Author-Wirtes-Paper tree has to be assigned a higher score than the Paper-Cites-Paper tree, because they have the same size.

The third ranking factor is the semantics of links. The score of a node depends on the links between it and the other nodes. ObjectRank only uses this ranking factor. Geerts, Mannila and Terzi[24] generalize link analysis methods for analyzing relation databases. Their algorithms can be used to score nodes.

The score functions used by existing database retrieval systems are related to the definition of query result. If a query result contains many nodes, the IR score of attribute values and the structure of result trees are considered. The score function used by DISCOVER is such an example. If a query result contains only one node, the structure of result trees is not necessary to be considered. For example, ObjectRank only takes the semantics of links into account when scoring the results.

3.6 Efficient Execution

Different systems use different algorithms to generate query results. To execute these algorithms efficiently, different systems uses different optimization methods. For example, in DISCOVER, candidate networks are intermediate results. To generate the final results, a SQL statement is constructed for each candidate network. Then there are many SQL statements to be executed. DISCOVER exploits the common join expression to speed the execution of all SQL statements.

A common problem for all database retrieval systems is how to execute top-k queries efficiently. The trivial method is to generate all results first, then sort them, and finally output the first *k* results. Obviously, the efficiency of this method is low. DISCOVER proposes several algorithms to execute a top-k join queries[25] efficiently. Top-k selection query[26] is a closely related problem to top-k join query. However, the efficient execution of top-k selection queries is not considered in DISCOVER.

3.7 Presentation of Results

It is not simple to present the query results. The reasons are as follows. Firstly, the users need meaningful results. Due to database normalization, logical units of information may be fragmented and scattered across several physical tables. Thus, a tuple may not be a logical unit of information, and the logical meaning of a join is not easy to be understood.

Secondly, there are lots of similar results. For example, as shown in Fig.6, author A_1's paper cites author A_2's paper. If the query contains keywords A_1 and A_2, six results will be generated. These results have the same structure. Too many similar results would hinder the users from discovering the most important result.

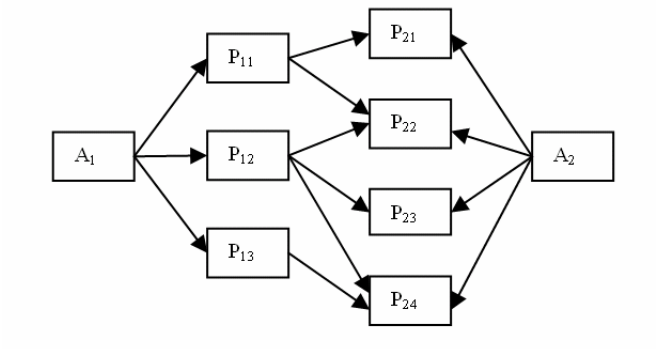

Fig.6. Author A_1's paper cites Author A_2's paper

Thirdly, it is necessary to support query reformulation. It has been observed that most users need to spend large amounts of time reformulating their queries to accomplish effective information retrieval. Relevance feedback is the most popular query reformulation strategy. In a relevance feedback cycle, the user is presented with a list of results and, after examining them, marks those which are relevant. Finally, the users need to browse the database. The presentation of query results should be a jumping-off point of database browsing.

Many ways are used by existing database retrieval systems to present query results. BANKS shows the query results in a nested table, and it supports browsing data stored in the database. DBXplorer describes the join trees with text and shows join results in a table. DbSurfer uses tree-like structure to display all trails. DISCOVER uses "information unit" and "presentation graph" to solve the problems mentioned above[27].

4 Future Directions

In Section 3, we have summarized some problems unresolved by existing database retrieval systems. For example, relevance feedback is rarely supported. Besides to solve these problems, the future work should pay attention to the following topics.

4.1 Performance Improvement

A database retrieval system would not be put into practice if its performance is low. The existing systems are tested on the database that has a simple schema and a small volume of data. Their performance is not assured when the database schema becomes complicated or the volume of data becomes huge. Thus scalability with respect to the case corresponding to a large volume of data is undoubtedly an important research issue.

Benchmark-based experimental results for all current database retrieval systems are needed. There are many reference collections used to evaluate information retrieval systems, such as the TREC collection[28]. A test collection of real world XML documents is also built for the evaluation of XML retrieval systems in recent year[29]. We believe that it is necessary to build a reference database collection used for evaluating the performance of database retrieval systems.

4.2 Searching the Hidden Web

A large volume of data is stored in the Hidden Web which can not be indexed by search engines. Because of using the same keyword search interface as search engines, database retrieval systems promise a good start for solving the Hidden Web problem. The next step is that a search engine must decide which keywords it should use to search in a database.

A trivial way is to search all words of a dictionary. This way causes lots of useless searches. Thus, it is necessary to find a better way to search a database with the least number of keywords. Another simple method is to let the search engine act as a meta-search engine. The search engine sends the search request to every database retrieval system, and integrates all results returned. This method is only suitable for integrating a small number of database retrieval systems. When the number of databases increases, the response time of this method decreases.

4.3 Integrating with XML Retrieval Systems

XML retrieval systems that support searching XML documents with keywords developed rapidly in recent years[27][29][30] [31][32] [33]. For example, many a good paper focuses on the index structure and query language for keyword search in XML documents. Because of page limit, this paper mainly discusses relational database retrieval. However some researchers believe that XML database retrieval is a more important research direction.

It is the time to integrate database retrieval systems, information retrieval systems and XML retrieval systems now. Although there are many ways to integrate these different types of systems, keyword search will be one of future integrated system's user interfaces. It is also expected that a future database retrieval system is an integrated system because relational data, text data and XML data can all be stored in a database.

5 Conclusions

Keyword search is easy to learn by end-users. The "Hidden Web" problem will be alleviated by the employ of keyword search in databases. Also keyword search can be used for information system integration. Because of these advantages, database retrieval systems have attracted many researchers recently and some prototype systems have been developed.

In this paper, the existing database retrieval systems are classified into four types based on the characters of search algorithms and search results. The key techniques of the existing database retrieval systems are summarized from several aspects: the architecture of a database retrieval system; the definition of data model, query language and query result; the ranking of results; the efficient execution of a retrieval algorithm; and the presentation of results. The problems related to the existing database retrieval systems are also discussed.

There will be more and more research on database retrieval systems. We expect that system scalability and meaningfulness of results will be two important issues to be resolved in the next few years. We also think that keyword search in distributed heterogeneous databases will be a fruitful research topic.

We have developed a database retrieval system named SEEKER. The architecture of this system is showed in Fig.7.

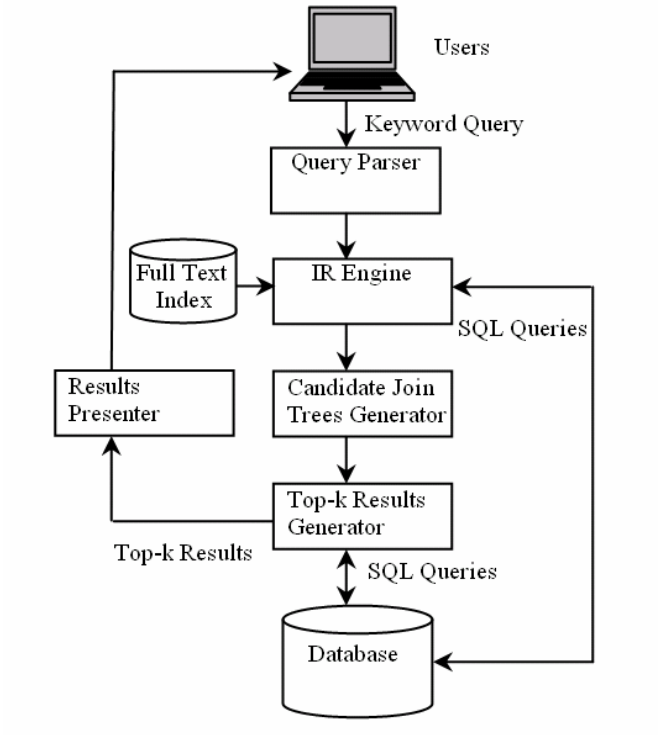

Fig.7. the architecture of SEEKER

Compared with existing systems, SEEKER has the following advantages: 1) SEEKER not only can search text attributes in relational databases, but also can search metadata and numeric attributes; 2)

SEEKER adopts a more reasonable scoring function to rank results, only top-k results will be returned to users. 3) SEEKER employs more efficient algorithms to speed keyword search process.

SEEKER is developed on Oracle 9i and a series of experiments are conducted to evaluate SEEKER's algorithms. Section 3 discussed some problems related to the design and implementation of SEEKER.

References

[1] Silberschatz A, Korth H, Sudarshan S. Database System Concepts, 4th Edition. *McGraw Hill*, 2001.

[2] Baeza-Yates R, Ribeiro-Neto B. Modern Information Retrieval. *ACM Press*, 1999.

[3] Florescu D, Levy A Y, Mendelzon A O. Database Techniques for World Wide Web: A Survey. *SIGMOD record*, 1998, 27(3):59-74.

[4] Lawrence S, Giles C L. Searching the World Wide Web. *Science*, 1998, 280(5360):98-100.

[5] Bergman M K. The Deep Web: Surfacing Hidden Value. White paper, Bright Planet, 2000.

[6] Raghavan S, Garcia-Molina H. Integrating Diverse Information Management Systems: A Brief Survey. *IEEE Data Engineering Bulletin*, 2001, 24(4):44-52.

[7] Hulgeri A, Bhalotia G, Nakhe C, Chakrabarti S, Sudarshan S. Keyword Search in Databases. *IEEE Data Engineering Bulletin*, 2001, 24(3):22-32.

[8] Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword Searching and Browsing in Databases using BANKS. In *Proceedings of 18th International Conference on Data Engineering*, San Jose, CA, Feb. 2002, pp.431-440.

[9] Agrawal S, Chaudhuri S, Das G. DBXplorer: A System for Keyword-Based Search over Relational Databases. In *Proceedings of 18th International Conference on Data Engineering*, San Jose, CA, Feb. 2002, pp.5-16.

[10] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword Search in Relational Databases. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, Aug. 2002, pp.670-681.

[11] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-Style Keyword Search over Relational Databases. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, Sep. 2003, pp.850-861.

[12] Dixon P. Basics of Oracle Text Retrieval. *IEEE Data Engineering Bulletin*, 2001, 24(4):11-14.

[13] Maier A, Simmen D. DB2 Optimization in Support of Full Text Search. *IEEE Data Engineering Bulletin*, 2001, 24(4):3-6.

[14] Hamilton J, Nayak T. Microsoft SQL Server Full-text Search. *IEEE Data Engineering Bulletin*, 2001, 24(4):7-10.

[15] Balmin A, Hristidis V, Papakonstantinou Y. ObjectRank: Authority-Based Keyword Search in Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, Aug. 2004, pp.564-575.

[16] Brin S, Page L. The Anatomy of a Large-Scale Hyper-textual Web Search Engine. In *Proceedings of the 7th International World Wide Web Conference*, Brisbane, Australia, 1998, pp.107-117.

[17] Dar S, Entin G, Geva S, Palmon E. DTL's DataSpot: Database Exploration Using Plain Language. In *Proceedings of the 24th International Conference on Very Large Databases*, New York City, USA, Aug. 1998, pp.645-649.

[18] Goldman R, Shivajumar N, Venkatasubramanian S, Garcia-Molina H. Proximity Search in Databases. In *Proceedings of the 24th International Conference on Very Large Databases*, New York City, USA, Aug. 1998, pp.26-37.

[19] Wheeldon R, Levene M, Keenoy K. DbSurfer: A Search and Navigation Tool for Relational Databases. In *Proceedings of the 21st Annual British National Conference on Databases*, Edinburgh, UK, Jul. 2004, pp.144-149.

[20] Sarda N L, Jain A. Mragyati: A System for Keyword-Based Searching in Databases. Report No. cs.DB/011052 on CORR, 2001.

[21] Masermann U, Vossen G. Schema Independent Database Querying (on and off the Web). In *Proceedings of the 4th IDEAS*, Yokohoma, Japan, Sep. 2000, pp.55-64.

[22] Masermann U, Vossen G. Design and Implementation of a Novel Approach to Keyword Searching in Relational Databases. *ADBIS-DASFAA Symposium*, Prague, Czech Republic, Sep. 2000, pp.171-184.

[23] Agrawal R, Srikant R. Searching with Numbers. In *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May, 2002, pp.420-431.

[24] Geerts F, Mannila H, Terzi E. Relational Link-based Ranking. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, Aug. 2004, pp.552-563.

[25] Ilyas I, Aref W, Elmagarmid A. Supporting Top-k Join Queries in Relational Databases. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, Sep. 2003, pp.754-765.

[26] Fagin R, Lotem A, Naor M. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 2003, 66(4):614-656.

[27] Hristidis V, Papakonstantinou Y, Balmin A. Keyword Proximity Search on XML Graphs. In

Proceedings of 19th International Conference on Data Engineering, Bangalore, India, Mar. 2003, pp.367-378.

[28] Voorhees E M, Harman D K. Overview of the 6th Text REtrieval Conference (TREC-6). In *Proceedings of the 6th Text REtrieval Conference*, Gaithersburg, Maryland, 1997, pp.1-24.

[29] Gövert N, Kazai G. Overview of the INitiative for the Evaluation of XML Retrieval (INEX 2002). In *Proceedings of the First INEX Workshop*, Dagstuhl, Germany, Dec. 2002, pp.1-17.

[30] Florescu D, Kossmann D, Manolescu I. Integrating Keyword Search into XML Query Processing. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, NL, May, 2000, pp.119-135.

[31] Guo L, Shao F, Botev C, Shanmugasundaram J. XRANK: Ranked Keyword Search over XML Documents. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Diego, California, USA, Jun. 2003, pp.16-27.

[32] Cohen S et al. XSearch: A Semantic Search Engine for XML. In *Proceedings of the 29th International Conference on Very Large Data Bases*, Berlin, Germany, Sep. 2003, pp.45-56.

[33] Li Y, Yu C, Jagadish H V. Schema-Free XQuery. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, Aug. 2004, pp.72-83.

Shan Wang is a professor of School of Information, Renmin University of China. Her current research interests include database and knowledge systems, data warehousing technology, and Grid data management

Kun-Long Zhang is a Ph.D. candidate in School of Information, Renmin University of China. His current research interests include database systems and distributed systems.