Keyword Search on Relational Databases

Wei Wang  Xuemin Lin  Yi Luo
School of Computer Science and Engineering
University of New South Wales
Australia
{weiw,lxue,luoyi}@cse.unsw.edu.au

1. Introduction

Recent years have seen increasing interest in the search technology. For example, search engines like Google have become the dominant avenue of obtaining information from the Internet. One of the factors that led to the success of search engines is that they adopt the keyword-based search paradigm: users submit keywords to search engines and a ranked list of relevant documents is returned. This enables even unsophisticated users to obtain information without detailed knowledge of the database schema or query languages.

While traditional relational database systems (RDBMSs) store the majority amount of the world’s enterprise data, they provide only limited support for keyword search. Specifically, they only search for single tuples that match all the query keywords. A few recent studies have recognized the need to dynamically grouping the individual matching tuples to form meaningful results. Consider a movie database for example. While the query peter king matches a tuple in the director relation, another meaningful result could be a join of two tuples, king kong from the movie relation and peter jackson from the director relation, through the foreign key to primary key relationship.

Adopting this keyword search on relational databases brings immediate benefits. First, more meaningful results can be returned by enabling searching for keyword matches across relation boundaries. This is particularly meaningful in relational databases as normalization is commonly used that decompose input data into several individual tuples and stored separately. Second, it lowers the access barrier for average users. For example, users can still query the database without having to know the SQL query language, the database schema, or data distribution. Third, it allows advanced types of queries, such as finding implicit relationship between two objects in the database.

1.1. Challenges and Related Work

In the following, we briefly review existing work according to several perspectives and discuss some challenges and proposed techniques.

**Data Model** According to the data models employed, existing research can be classified into two categories: graph-based approach and relational approach.

- The Graph-based approach materializes the entire content in the relational database into a graph, where each tuple is treated as a node and each foreign-key-to-primary-key relationship is treated as a link between corresponding nodes. Existing work include DataSpot [5], Proximity search [8], the BANKS I/II systems [1, 12]
- For the relation-based approach, the answer to the keyword query is a set of tuples that collectively contains all or part of the search keywords, and these tuples can be joined together through foreign-key-to-primary-key relationships. Existing work include DBXplorer [2], DISCOVER I/II systems [10, 9], and [15].

So far, there has been no formal comparison between the two approaches. Our impression is that graph-based approach works well for small to medium sized databases, and has fast query processing performance. Relational approach, on the other hand, can be easily integrated into the relational database systems, thus, avoiding issues such as scalability, data and index maintenance, etc. Its performance for complex query and large query results still needs improvement.

**Ranking** The effectiveness of the scoring and ranking functions is an important aspect of keyword search. As more than one result may match any keyword query, it is desirable to assign each result a score and rank the list of results according to their scores. Intuitively speaking, the top
results in the ranked list are more relevant to the query than those at the bottom. The IR community has developed theories and practice in ranking functions for documents [4]. A widely used IR ranking function [18] is:

\[
\sum_{Q,D} \frac{1 + \ln(1 + \ln(tf))}{(1 - s) + s \cdot \ln \frac{N + 1}{df}} \cdot qf \cdot \ln \frac{N}{df}
\]

Similar ranking functions have also been implemented in major relational databases systems.

However, the result of the keyword search in relational databases is in general a tree of tuples (e.g., a particular movie tuple joins with another director tuple), rather than a flat and homogeneous text document. Therefore, IR-based methods cannot be immediately applied, and other factors, especially those regarding the structure and semantics of the query results, need to be considered. Previous work has identified and used a few other metrics that contribute to more effective ranking functions for such search results, e.g., join tree size [2], PageRank node ranking [1, 11], normalized node prestige and edge weight [1], shortest distance between keywords [8, 12], etc. Of these, only [9, 15] considered combining some of the above factors with the IR-style ranking functions. However, the effectiveness of the proposed ranking functions are still questionable, as they have not been thoroughly tested against massive quantities of real data and queries.

**Query Processing** In terms of query processing, finding top-\(k\) results for keyword search is also technically challenging. Even with an unrealistically simplified ranking function, returning top-1 search result has already been proved to be NP-hard (by reducing it to the minimum group Steiner tree problem) [6]. Another theoretical result regarding the result enumeration complexity can be found in [14].

Several query processing and optimization techniques in the previous work are summarized below:

- **Indexing.** Just as the inverted index is the key to efficient IR queries, novel indexing methods have been designed for keyword search in relational databases. [8] employs a memory-sized hub index which substantially accelerates the computation of the shortest distance between any two nodes in a graph. [2] discussed relative strength and weakness of two alternative indexing methods at the column level (Pub-Col) and at the cell level (Pub-Cell).

- **Top-\(k\) optimization.** It is often the case that users are only interested in the top \(k\) matches in the ranked result list, as the fuzzy nature of the keyword search paradigm tends to return a large number of results. Pushing this top-\(k\) constraints into query processing is a simple yet powerful idea that results in orders of magnitude speedup against the naïve method which retrieve the top-\(k\) results after computing all the answers. Fagin’s threshold algorithm (TA) lies at the heart of top-\(k\) optimization [7]. TA algorithm is provably instance optimal and thus has been applied to many top-\(k\) optimization problems in database research. Specifically, [9] developed the global pipeline algorithm based on the TA algorithm. It can terminate the computation as soon as the top-\(k\) keyword query results have been obtained.

**Other Related Work** Since keyword search originates with the information retrieval community, there is no doubt that much literature from the Information Retrieval field is related to this topic. Traditional information retrieval focuses mainly on unstructured plain text documents, and provides rich functionality and flexibility for text search. [4, 18] gives an overview of IR topics and techniques.

Another related area is full-text query for XML and semi-structured data. [3] is a latest tutorial that summarizes the existing research and open issues on full-text search for XML document collections.

2. SPARK

![Figure 1. Search “2001 hanks” on the SPARK System](image)

SPARK [16] is our recent work that presents answers to some of the aforementioned challenges. SPARK belongs to the relational approach, and is designed as a middle-ware that sits on top of most relational databases to provide keyword search capability. The user interface design of the system is inspired by Google, and the search results are displayed in a user-friendly manner (See Figure 1). Figure 2
shows a number of advanced functionalities supported by SPARK.

**Figure 2. Advanced Search**

SPARK uses its own sophisticated ranking function that considers the following three factors: IR relevance score, completeness of the results, and the size of the results. Unlike the previous ranking functions that calculate the IR relevance score separately for each tuple in the result, we take a holistic view of the tuples by merging them into a “virtual document” before applying the state-of-the-art IR ranking formula. We also found that, when the query is short, users prefer results that matches most, if not all, of the search keyword. Consequently, a completeness factor based on the extended Boolean model [17] is designed to capture this observation. Normalization based on the size of the results is needed as a larger result (i.e., one involving more tuples) is more likely to achieve higher IR relevance and completeness scores. These three factors are multiplied together to derive the final score for the results. We have conducted extensive experiments on the effectiveness of the proposed ranking function. We have shown in [16] that our new ranking method out-performs previous approaches substantially. The average reciprocal rank1 of SPARK is often close to the maximum value 1.0, while previous approaches are usually below 0.5.

The essence of the top-\(k\) query processing is to establish the early stopping criterion for the given scoring function. We have derived a non-trivial upper bounding function for our new ranking function. One important difference between our problem and traditional top-\(k\) query processing problem (especially in the middle-ware applications) is that every probe to the database is a potentially expensive parameterized join query. We proposed the skyline sweeping algorithm which minimizes such database probings to a minimum. The key idea is to check only the necessary can-

3. Future Research Issues

Despite years of research, there are still many open issues in this area. We list a few important issues that call for further research.

**Formal semantics of query results** Existing work differs in a number of aspects when defining the query results for keyword search over databases. The differences can be summarized along several dimensions: result is a tree or a general graph, the result tree or graph is directed or undirected, whether a result is allowed to have more than one match of a query keyword, etc. These differences render it impossible to compare different systems. With a unified definition of the query result, it is also possible to study other aspects, such as advanced query language.

**Alternative and advanced ranking methods** Given the wide applications scenarios of keyword search in relational databases, and the wide spectrum of structure and keyword distributions in the database, it is unlikely that a universal one-size-fits-all ranking function will work well. On the other hand, given the heterogeneity of the query results, other factors and ranking schemes should be considered and developed. For example, it is still an open question how a ranking function could take into consideration the detailed structure and semantics of the query results.

**Query processing and optimization methods** Query processing for various kinds of ranking functions will still remain an interesting issue. Approximate top-\(k\) query processing could also be considered as it is common to have prohibitive amount of results for a large database. An interesting open topic is how to apply query optimization techniques to this problem, apart from existing common-subquery sharing strategy. To this end, a cost model combining the database probing cost and selectivity estimation probably needs to be developed.
Benchmarks A carefully design benchmark will definitely be helpful to the further development of the topic. As a starting point, a few datasets, such as DBLP and IMDB, have been commonly used by existing work. However, queries are either picked up manually or totally at random, and, perhaps more importantly, there is no manually labeled correct results for the queries. The pooling strategy could be considered towards achieving this goal [19].

4. Conclusions

Keyword search in relational database is a central topic in integrating database and information technologies. We briefly introduce the problem, analyze the challenges, and survey the existing work. We also highlight our recent research progress of the SPARK project that partially addresses the challenges. A few future research issues are also discussed.

References