I J C T A, 9(24), 2016, pp. 385-391 ISSN: 0974-5572

Supporting Fuzzy Keyword Search in Databases

Jayavarthini C.* and Priya S.

ABSTRACT

An efficient keyword search system computes answers as a user types in a keyword query character by character. This can also be used to support search-as-you-type on data residing in a RDBMS. This paper deals with how to support this type of search using SQL and how to utilize the power of the SQL language to support efficient search. Use of auxiliary indexes to increase search performance is also discussed. Techniques for fuzzy search using SQL are developed by allowing mismatches between query keywords and answers. Solutions for both single-keyword queries and multi keyword queries are obtained. In addition to SQL queries with join operations, auxiliary tables, foreign key constraints, built-in indexes on key attributes, and incremental algorithms using cached results are carefully designed so that these SQL queries can be executed efficiently by the DBMS search engine to achieve a high speed.

Keywords: Keyword query, search-as-you-type, SQL, fuzzy search, index, auxiliary tables

1. INTRODUCTION

Data mining refers to the extraction of knowledge from huge data. By data processing, fascinating information, regularities, or high-level data will be extracted from databases and viewed or browsed from completely different angles. Ancient data systems retrieve answers once a user submits a whole question. Users typically feel troublesome if they have restricted knowledge regarding the underlying data, and ought to use several attempts-to-see approach for searching data. Several information systems today improve user search experiences by providing instant feedback as users formulate search queries. Most search engines and on-line search forms support auto-completion as a user types in a keyword query, character by character. For instance, consider the Web search interface at Netflix [1], which allows a user to search for movie information. If a user types in a partial query "mad," the system shows movies with a title matching this keyword as a prefix, such as "Madagascar" and "Mad Men: Season 1. " The instant feedback helps the user not only in formulating the query, but also in under-standing the underlying data. This type of search is generally called search-as-you-type or type-ahead search.

A search-as-you-type system computes answers as a user types in a keyword query character by character. Search-as-you-type is to be combined with SQL for better performance to retrieve the data from RDBMS. To make all the databases support search-as-you-type, many approaches were proposed. One approach is to develop an application layer with indexes followed by algorithms to support this so that, it can be used in all databases. Next is to use DB extenders. Third is to use SQL [2]. Third can be achieved, as SQL is highly portable and compatible.

Here is a challenge how to force current database functionalities to meet the high-performance requirement in terms of speed. One way to achieve this is to use auxiliary indexes. Solutions for all type of queries are to be presented. Fuzzy search is also included with search-as-you-type in SQL to allow mismatches

* Assistant Professor/Department of CSE, SRM University, Kattankulathur-603 203, Tamil Nadu, India, *Email: jayavarthini.c@ktr.srmuniv.ac.in*

** Assistant Professor, Department of CSE, SRM University, Kattankulathur-603 203, Tamil Nadu, India, *Email: priya.sn@ktr.srmuniv.ac.in*

between query keywords and answers. Our goal is to utilize the built-in query engine of the database system as much as possible. In this way, we can reduce the programming efforts to support search-as-you-type. In addition, the solution developed on one database using standard SQL techniques is portable to other databases which support the same standard. These techniques enable DBMS systems on a commodity system to support search-as-you-type.

The remainder of this paper is organized as follows. Section II describes about related works, whereas Section III describes the problem statement. Section IV describes the Fuzzy Keyword Search using SQL in Database, whereas Section V describes the exact and fuzzy search, then Section VI about Supporting Multikeyword Queries, then Section VII describes the results and discussion and Section VIII about the conclusion and future implementation.

2. RELATED WORKS

There was an efficient approach that leverages the inverted index on the document to identify the subset of documents relevant to the task and processes only those documents [3]. The key insight is to reduce the cost of query by exploiting the overlap of tokens among the set of entities. Another approach called BANKS, an integrated browsing and keyword querying system for relational databases was developed. BANKS allows users with no knowledge of database systems or schema to query and browse relational database with ease [4]. Traditional indexing data structures either incur large processing times for a substantial class of queries, or they use a lot of space. A new indexing data structure that uses no more space than a state-of-the-art compressed inverted index, but that yields an order of magnitude faster query processing times was developed [5]. A Tastier system brings instant gratification to users by supporting type-ahead search, which finds answers “on the fly” as the user types in query keywords [6]. The main challenge was how to achieve a high interactive speed for large amounts of data in multiple tables, so that a query can be answered efficiently within milliseconds. Efficient index structures and algorithms for incrementally computing answers to queries were implemented in order to achieve an interactive speed on large data sets.

A graph-partition-based method and query prediction techniques were proposed to improve search efficiency. Traditional information systems return answers after a user submits a complete query. An information-access paradigm, called “interactive, fuzzy search,” was developed in which the system searches the underlying data “on the fly” as the user types in query keywords. It extends autocomplete interfaces by allowing keywords to appear in multiple attributes of the underlying data then it finds relevant records that have keywords matching query keywords approximately [7]. It also explained how to efficiently find in a collection of strings those similar to a given string. The existing methodology used individual filters for approximate search. Existing filtering techniques was integrated with the filtering algorithms to show that they should be used together judiciously, since the way to do the integration can greatly affect the performance [8]. The complete search is an interactive search engine. Context-sensitive prefix search and completion mechanisms are used [9] to reduce the amount of data that has to be processed per query.

Features like automatic query completion, semi-structured (XML) retrieval, semantic search, DB-style joins and grouping and arbitrary combinations were also implemented [9]. The central completion mechanism of the Complete Search engine makes use of a novel kind of index data structure, called HYB in order to reduce the amount of data processed per query. Real-world applications require solving a similarity search problem where one is interested in all pairs of objects whose similarity is above a specified threshold. Inverted list based approach and All-Pairs algorithm that is easy to implement and does not require any parameter tuning was developed. This reduces the search space and All-Pairs algorithm is 1.3 to 15 times faster than an approximate algorithm [10]. It also handles a variety of datasets across a wide setting of similarity thresholds, with large speedups. A primitive operator SSJoin was implemented for performing similarity joins. The similarity joins based on a variety of textual and non-textual similarity functions can

be efficiently implemented using the SSJoin operator [11]. The SSJoin operator compares values based on "sets" associated with each one of them. The design and implementation of this logical operator leverages the existing set of relational operators, and helps in defining a rich space of alternatives for optimizing queries involving similarity joins. A system called DISCOVER was presented, which performs keyword search in relational databases. It proceeds in three steps. First it generates the smallest set of candidate networks that guarantee that all *MTJNT*'s will be produced [12].

Then the greedy algorithm creates a near-optimal execution plan to evaluate the set of candidate networks. Finally, the execution plan is executed by the DBMS. This keyword search enables information discovery without requiring from the user to know the schema of the database, SQL or some QBE-like interface, and the roles of the various entities and terms used in the query databases that do not require knowledge of the database schema or of a querying language.

3. PROBLEM STATEMENT

The challenges include: SQL meet the high performance requirement to implement an interactive search interface and some important functionality to support search-as-you-type requires join operations, which could be rather expensive to execute by the query engine. The previous approaches support search-as-you-type using the native SQL language.

To address these challenges we propose two types of methods to support search-as-you-type for single-keyword queries, based on whether they require additional index structures stored as auxiliary tables. The main consideration was to increase the speed by using auxiliary indexes stored as tables. We discuss the methods that use SQL to scan a table and verify each record by calling a user-defined function (UDF) or using the LIKE predicate. Exact search for single keyword queries are done using UDF, LIKE predicate and inverted-index table and the prefix table. Exact search for multi keyword queries are done using UDF, LIKE predicate, full-text indexes and UDF (called "FI+UDF"), full-text indexes and the LIKE predicate (called "FI+LIKE"), the inverted-index table with prefix table and word-level incremental method. We also support fuzzy search for single-keyword queries.

Fuzzy search for single keyword queries are implemented using UDF, gram-based method, neighbourhood-generation-based method, character-level incremental algorithms. As the gram-based method and a UDF-based method has a low performance, we propose a new neighbourhood-generation based method, using the idea that two strings are similar only if they have common neighbours obtained by deleting characters. We extend the techniques to support multi-keyword queries that when deployed in a Web application, the incremental-computation algorithms do not need to maintain session information, since the results of earlier queries are stored inside the database and shared by future queries.

Fuzzy search for multi keyword queries are implemented using word-level incremental algorithms, called NGB+ and Incre+. The approach using inverted index tables and prefix tables supports prefix, fuzzy search and achieve the best performance. The experimental results on large, real data sets showed that the proposed techniques can enable DBMS systems to support search-as-you-type on large tables.

4. FUZZY KEYWORD SEARCH USING SQL IN DATABASE

The keyword table, inverted index table, prefix table, gram table were already created for related data in RDBMS. When the user types a keyword query, the presence of keyword with tuple id is obtained using keyword, inverted and prefix table. The approximate matching of keywords is done with the help of gram table. Now the data related to user entered keywords are obtained using incremental computation and neighborhood generation method.

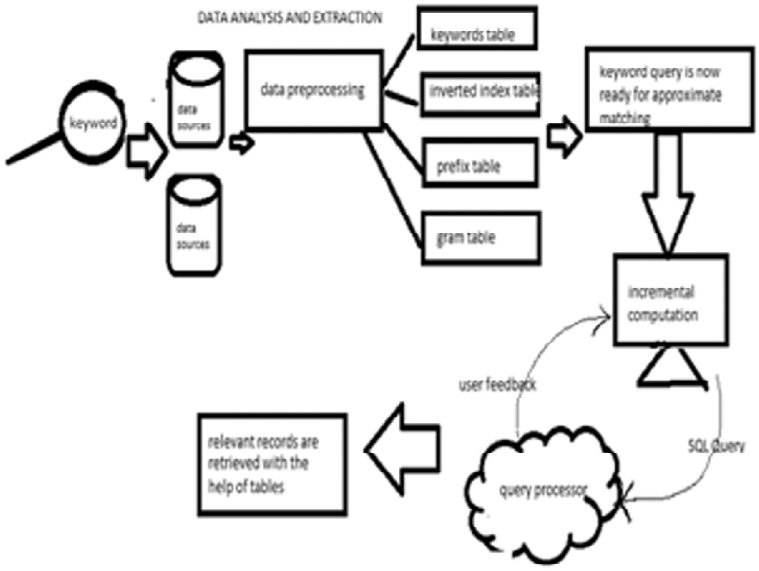

Figure 1: Architectural diagram for fuzzy keyword search using databases.

5. EXACT AND FUZZY SEARCH

5.1. Exact Search

During Exact search, a user types a single partial key word W character by character, search as you type system finds the record that contain keyword with a prefix W. It is also known as prefix search.

5.1.1. Index Search

Inverted index and prefix tables are created to facilitate prefix search. Inverted index table contains $\{k_{id}, t_{id}\}$ whereas, k_{id} is a unique id assigned in alphabetical order for the keywords and t_{id} is the id assigned for each tuple in a relation. Prefix table contains $\{p, lk_{id}, uk_{id}\}$ whereas p is the prefix, lk_{id} is least id of those keywords in table T (contains all the prefixes of keywords) and uk_{id} is greatest id of those keywords in table T. Given partial keyword w, keyword range is obtained using prefix table P_t and then find the records in the keyword range through inverted index.

5.1.2. No-Index Search

SQL query is given, which scans each record and verifies whether the record is an answer to the query. There are two ways to do this:

1) To Call User-Defined Functions (UDFs). We can add modules into databases to validate whether a record contains the query keyword
2) To use the LIKE predicate. Databases provide a LIKE predicate to perform string matching. It can be used to check whether a record contains the query keyword. But this method may introduce false positives.

5.2. Fuzzy Search

A fuzzy search is a search process that locates contents that are likely to be relevant to a keyword in an approximate manner so that even when the keyword does not exactly match the desired information.

5.2.1. No-Index Methods

In Exact search, two methods were used. But in fuzzy search, only UDF can be used and LIKE predicate cannot be used since it does approximate matching. Given a keyword w and string s, minimal edit distance

is calculated between keyword and prefix and returned as answer. Using the minimal edit distance, we can measure the relevance of keyword to the desired information.

5.2.2. Index- Based Methods

UDF

We can find similar prefixes of given keyword from prefix table using UDF. Similar prefix is identified using the edit distance. Then the k_{id} of inverted table is compared with both lk_{id} and uk_{id} of prefix table. The records in the table are retrieved if $lk_{id} <= k_{id} >= uk_{id}$ and also t_{id} is the same in both inverted table and table, where the records are stored.

GRAM based

There are many q-gram-based methods to support approximate string search . Given a string s, its q-grams are its substrings with length q. To find similar prefixes of a query keyword, besides use the inverted-index table and the prefix table, also in need to create a q-gram table with records.

Neighbourhood-generation method

A neighbourhood-generation-based method to support approximate string search was developed. We extend this method to use SQL to support fuzzy search-as-you-type. Given a keyword w, the substrings of w by deleting I characters are called “i-deletion neighbourhoods” of w.

Incremental Computing

Word-Level Incremental Computation use previously computed results to incrementally answer a query. Assuming a user has typed in a query with keywords create a temporary table to cache the record ids of query. If the user types in a new keyword and submits a new query with keywords use temporary table to incrementally answer the new query. Exact search focus on the method that uses the prefix table and inverted-index table. Fuzzy search consider character level incremental method. Fuzzy search consider character level incremental method, the user arbitrarily modifies the query, can easily extend this method to answer new query.

6. SUPPORTING MULTIKEYWORD QUERIES

Multi-keyword Search updates is given a multi-keyword query Q with m keywords, using the “INTERSECT” Operator first compute records for each keyword and then use INTERSECT operator to join these records for different keywords to compute answers. Using Full-text Indexes first use full-text indexes to find records matching the first complete keywords and then use proposed methods to find records matching the last prefix keyword. Two methods cannot use pre-computed results lead to low performance.

6.1. Supporting First N-Queries

The above methods cannot be easily extended to support fuzzy search, as they cannot distinguish the results of exact search and fuzzy search. Generally, we need to first return the “best results” with smaller edit distances. To address this issue, we propose to progressively compute the results.

As an example, we consider the character-level incremental method. For a single-keyword query w, we first get the results with edit distance 0. If we have gotten N answers, we terminate the execution; otherwise, we progressively increase the edit distance threshold and select the records with edit-distance thresholds 1; 2; . . .;, until we get N answers.

7. RESULTS AND DISCUSSION

We compared different methods to support search-as-you-type. The first graph shows the relationship between query processing time and number of keywords for various approaches in exact search. From the graph, we understand that IPT+ has low processing time even though the number of keywords increases. The next graph shows the relationship between query processing time and number of keywords with respect to Gram, UDF, Incremental methods of fuzzy search. Incremental Based approach show very low processing time even when number of keywords increase. Also the results are relevant to the keyword query.

8. CONCLUSION AND FUTURE IMPLEMENTATION

We targeted on the challenge of ways to leverage existing software database functionalities to satisfy the superior demand to realize an interactive speed. To support prefix matching, we have a tendency to projected solutions that use auxiliary tables as index structures and SQL queries to support search-as-you-type. We extended the techniques to the case of fuzzy queries, and projected numerous techniques to boost query performance. We projected incremental-computation techniques to answer multi-keyword queries, and studied ways to support first-N queries and progressive updates. Our experimental results on giant, real knowledge sets showed that the proposed techniques will modify software database systems to support search-as-you-type on huge tables. So as to realize a high speed, we propose index-based ways. The approach

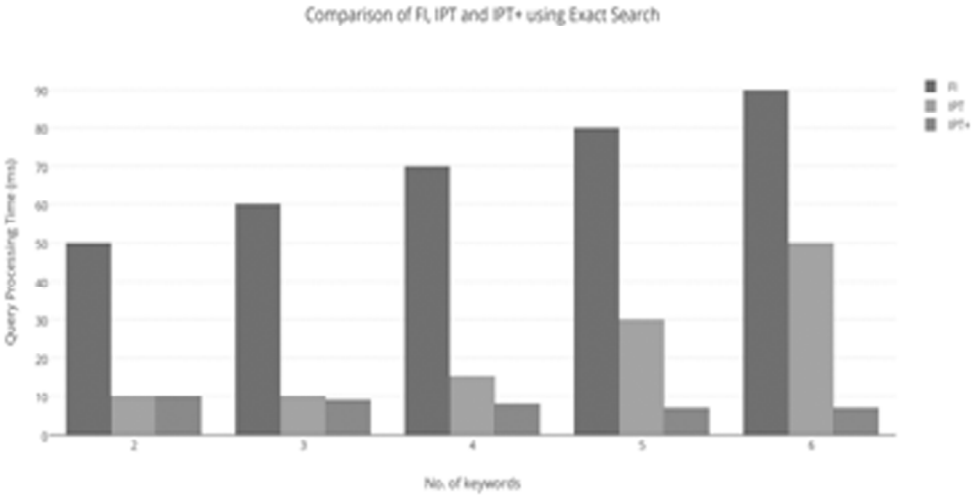

Figure 2: Comparison of FI, IPT and IPT+ - Exact Search

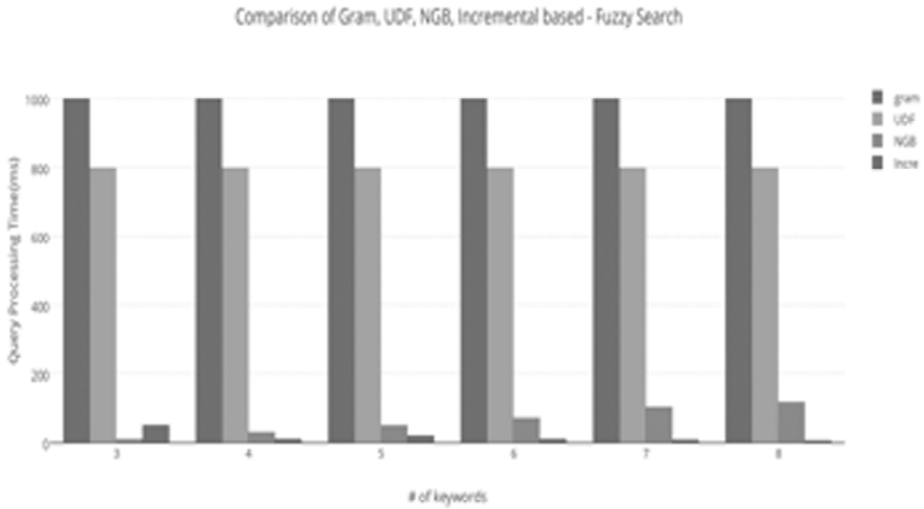

Figure 3: Comparison of Gram, UDF, NGB, Incremental Based – Fuzzy Search

victimisation inverted-index tables and also the prefix tables will support prefix, fuzzy search, and reach the most effective performance among of these ways and crush the constitutional ways in SQL Server and Oracle. Our SQL-based methodology can do a high interactive speed and scale well. There are various open problems to support search-as-you-type. It includes different ranking approaches to get relevant data efficiently.

REFERENCES

[1] Y. Koren, Tutorial on recent progress in collaborative filtering, in: Proceed-ings of the 2008 ACM Conference on Recommender Systems, ACM, Lausanne, Switzerland, 2008, pp. 333–334.

[2] Guoliang Li, Jianhua Feng, Chen Li,Supporting search-as-you-type using SQL in Databases, IEEE Transactions on knowledge and data engineering, vol. 25, no. 2, 2013

[3] S. Agrawal, K. Chakrabarti, S. Chaudhuri, and V. Ganti, "Scalable Ad-Hoc Entity Extraction from Text Collections," Proc. VLDB Endowment, vol. 1, no. 1, pp. 945-957, 2008.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword Searching and Browsing in Data Bases Using Banks," Proc. 18th Int'l Conf. Data Eng. (ICDE '02), pp. 431-440, 2002.

[5] H. Bast and I. Weber, "Type Less, Find More: Fast Autocompletion Search with a Succinct Index," Proc. 29th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '06), pp. 364-371, 2006.

[6] G. Li, S. Ji, C. Li, and J. Feng, "Efficient Type-Ahead Search on Relational Data: A Tastier Approach," Proc. 35th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09), pp. 695-706, 2009.

[7] S. Ji, G. Li, C. Li, and J. Feng, "Efficient Interactive Fuzzy Keyword Search," Proc. 18th ACM SIGMOD Int'l Conf. World Wide Web (WWW), pp. 371-380, 2009

[8] C. Li, J. Lu, and Y. Lu, "Efficient Merging and Filtering Algorithms for Approximate String Searches," Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08), pp. 257-266, 2008.

[9] H. Bast and I. Weber, "The Complete Search Engine: Interactive, Efficient, and Towards IR & DB Integration," Proc. Conf. Innovative Data Systems Research (CIDR), pp. 88-95, 2007.

[10] R.J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all Pairs Similarity Search," Proc. 16th Int'l Conf. World Wide Web (WWW '07), pp. 131-140, 2007.

[11] S. Chaudhuri, V. Ganti, and R. Kaushik, "A Primitive Operator for Similarity Joins in Data Cleaning," Proc. 22nd Int'l Conf. Data Eng. (ICDE '06), pp. 5-16, 2006.

[12] V. Hristidis and Y. Papakonstantinou, "Discover: Keyword Search in Relational Data Bases," Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02), pp. 670-681,2002.