International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 3 Issue: 2 641 - 645

Type Ahead Search in Database using SQL

Salunke Shrikant Dadasaheb
Dattakala Group of Institutions, Faculty of Engineering
University of Pune
shrikantsalunke25@gmail.com

Prof. Bere Sachin Sukhadeo
Dattakala Group of Institutions, Faculty of Engineering
University of Pune
sachinbere@gmail.com

Abstract— A type ahead search system computes answers on the fly as a user types in a keyword query character by character. We are going to study how to support type ahead search on data in a relational DBMS. We focus on how to help this type of search using the SQL. A prominent task that tests is how to influence existing database functionalities to meet the high performance to achieve an interactive speed. We extended the efficient way to the case of fuzzy queries, and suggested various techniques to improve query performance. We suggested incremental computation method to answer multi keyword queries, and calculated how to support first N queries and incremental updates. Our experimental results on large and real data sets showed that the proposed techniques can enables DBMS systems to support search as you type on large tables.

Keywords— *Fuzzy search, DBMS, SQL, Keyword query, incremental computation.*

*****

I. INTRODUCTION

Any data system today improves user search experiences by providing direct feedback as users formulate search queries. Most search engines and on-line search forms support motor vehicle completion, that shows urged queries or maybe answers "on the fly" as a user sorts in a very keyword question character by character. For instance, take under consideration the net search interface at Netflix, one that permits a user to go looking for moving picture info. If user sorts in a very partial question "mad," the system shows movies with a title matching this keyword as a prefix, like "Madagascar" and "Mad Men." the moment feedback helps the user not solely in formulating the question, however conjointly in understanding the underlying knowledge. This sort of search is usually referred to as type ahead search or type-ahead search.

Since several search systems store their info in very backend relative software, an issue arises naturally: a way to support type ahead search on the info residing in a very DBMS? Some databases like Oracle and SQL server already support prefix search, and that we may use this feature to try and do type ahead search. However, not all databases give this feature. For this reason, we have a tendency to study new ways which will be employed in all databases. One approach is to develop a separate application layer on the info to construct indexes, and implement algorithms for respondent queries. Whereas this approach has the advantage of achieving a high performance, its main disadvantage is duplicating knowledge and indexes, leading to extra hardware prices. Another approach is to use info extenders, like DB2 Extenders, Informix Data Blades, Microsoft SQL Server Common Language Runtime (CLR) integration, and Oracle Cartridges, which permit developers to implement new functionalities to software. This approach isn't possible for databases that don't give such Associate in nursing extender interface, like MySQL. Since it has to utilize proprietary interfaces provided by info vendors, an answer for signal info is might not be transportable to others. Additionally, Associate in nursing extender-based answer, particularly those enforced in C/C++, may cause serious irresponsibleness and security issues to info engines.

A main question once adopting this enticing plan is: Is it possible and scalable? Specifically, will SQL meet the high performance demand to implement Associate in nursing interactive search interface? Studies have shown that such Associate in Nursing interface needs every question be answered among a hundred milliseconds [38]. Software systems don't seem to be specially designed for keyword queries, creating it more difficult to support type ahead search. As we are going to see later during this paper, some vital practicality to support type ahead search needs be part of operations that might be rather high-priced to execute by the question engine.

The quantifiability becomes even additional unclear if we wish to support 2 helpful options in type ahead search, particularly multi keyword search and fuzzy search. In multi keyword search, we have a tendency to permit a question string to possess multiple keywords, and notice records that match these keywords, even though the keywords seem at completely different places. As an example, we have a tendency to permit a user UN agency sorts in a very question "privacy mining rack" to search out a publication by "Rakesh Agrawal" with a title as well as the keywords "privacy" and "mining," even supposing these keywords are at completely different places within the record. In fuzzy search, we wish to permit minor mismatches between question keywords and answers. As an example, a partial question "aggraw" ought to notice a record with a keyword "Agrawal" despite the erratum within the question. Whereas these options will additional improve user search experiences, supporting them makes it even more difficult to try and do type ahead search within software systems.

In this paper we have a tendency to study a way to support type ahead search on software systems victimization the native command language (SQL). In different words, we wish to use SQL to search out answers to a pursuit question as a user sorts in keywords character by character. Our goal is to utilize the inherent question engine of the info system the maximum amount as attainable. During this method, we will scale back the programming efforts to support type

ahead search. Additionally, the answer developed on one info victimization commonplace SQL techniques is transportable to different databases that support constant commonplace. Additionally to the present to support prefix matching, we have a tendency to planned solutions that use auxiliary tables as index structures and SQL queries to support type ahead search. We have a tendency to extend the techniques to the case of fuzzy queries, and planned varied techniques to boost question performance. We have a tendency to planned incremental-computation techniques to answer multi-keyword queries, and studied a way to support first-N queries and progressive updates. Our experimental results on giant, real knowledge sets showed that the planned techniques will modify software systems to support type ahead search on giant tables.

II. Proposed Approach Framework and Design

Problem Definition

Most search engines and on-line search forms support motor vehicle completion, which provides instructed queries or perhaps answers "on the fly" as user varieties during a keyword question character by character. Since several search systems store their data during a backend relative package, a matter arises naturally: the way to support type ahead search on the information residing during a DBMS? Some databases like Oracle and SQL server already support prefix search, and that we may use this feature to try and do type ahead search. However, not all databases give this feature. For this reason, we have a tendency to study new ways that may be utilized in all databases. One approach is to develop a separate application layer on the information to construct indexes, and implement algorithms for respondent queries.

In associate degree existing systems don't seem to be specially designed for keyword queries, creating it tougher to support type ahead search. SQL meet the high performance demand to implement associate degree interactive search interface. Some necessary practicality to support type ahead search needs be a part of operations that can be rather valuable to execute by the question engine.

Proposed System Architecture

In this we are presenting the new architecture which is based on incremental based search with fuzzy methodology. Based on these terminologies below is proposed architecture in figure 1.

In this paper, we develop various methods to address changes. We propose two types of methods to support type ahead search for single-keyword queries, based on whether they require additional index structures stored as auxiliary tables.

III. No-Index Methods:

A clear-cut thanks to support type ahead search is to issue associate degree SQL question that scans every record and verifies whether or not the record is a solution to the question. There are a unit 2 ways that to try and do the checking: 1) vocation User-Defined Functions (UDFs). We will add functions into databases to verify whether or not a record contains the question keyword; and 2) victimization kind predicate. Databases give a LIKE predicate to permit users to perform string matching. We will use kind predicate to examine whether or not a record contains the question keyword. This technique might introduce false positives, e.g., keyword "publication" contains the question string "ic," however the keyword doesn't have the question string "ic" as a prefix. We will take away these false positives by vocation UDFs. the 2 no-index ways would like no extra house, however they will not scale since they have to scan all records within the table.

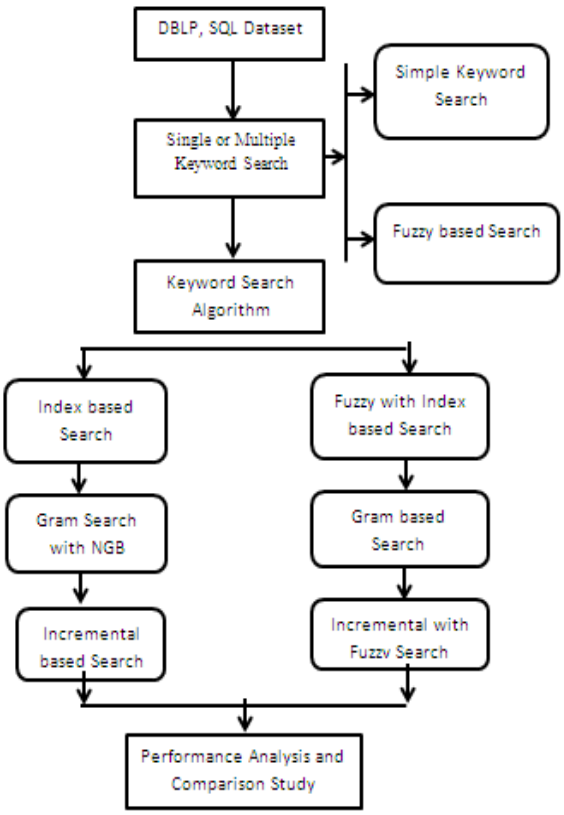

Fig 1: Proposed system architecture

- **Index-Based Methods:**

In this section, we have a tendency to propose to create auxiliary tables as index structures to facilitate prefix search. Some databases like Oracle and SQL server already support prefix search, and that we might use this feature to try to prefix search. However, not all databases give this feature. For this reason, we have a tendency to develop a replacement methodology that may be employed in all databases. Additionally, our experiments in Section eight.3 show that our methodology performs prefix search additional expeditiously.

- **Inverted-index table:**

Given a table T, we tend to assign distinctive ids to the keywords in table T, following their alphabetical order. We tend to produce AN inverted-index table IT with records within the type hkid; ridi, wherever child is that the id of a keyword and disembarrass is that the id of a record that contains the keyword. Given an entire keyword, we will use

the inverted-index table to search out records with the keyword.

- **Prefix table:**

Given a table T, for all prefixes of keywords in the table, we build a prefix table PT with records in the form hp; lkid; ukidi, where p is a prefix of a keyword, lkid is the smallest id of those keywords in the table T having p as a prefix, and ukid is the largest id of those keywords having p as a prefix. An interesting observation is that a complete word with p as a prefix must have an ID in the keyword range =lkid; ukid_, and each complete word in the table T with an ID in this keyword range must have a prefix p.

IV. WORK DONE

In this section we represent the input, result of practical work and environment used for implementation. Until now we implemented simple keyword search, fuzzy keyword search, index based search etc

Multi keyword queries: We implemented six methods for multi keyword queries:

1. using UDF;
2. using the LIKE predicate;
3. using full-text indexes and UDF (called "FI+UDF");
4. using full-text indexes and the LIKE predicate (called "FIþLIKE");
5. using the inverted-index table and prefix table (IPTables);
6. using the word-level incremental method (called "IPTablesþ")

4.1 Input Dataset:

For this implementation, we use the dataset DBLP.SQL which contains information related to keywords.

4.2 Hardware and Software Used

4.2.1 Hardware Configuration

- Processor - Pentium IV 2.6 GHz
- RAM - 512 mb dd ram
- Monitor - 15" color
- Hard Disk - 20 GB
- Key Board - Standard Windows Keyboard

4.2.2 Software Configuration

- Operating System - Windows XP/7
- Programming Language - Java
- Database - MySQL
- Tool - Net beans

4.3 Mathematical Model

T- Relational Table
R- Content of Record
W – Set of tokenized table
Aj – Attribute
$A_1,A_2,A_3,\ldots\ldots..,A_l$.
R={r1,r2,……..,rn}

Be the collection of records in T, and r_i. A_j denote the content of record r_i in attribute A_j. Let W be the set of tokenized keywords in R.

$R=\{r_1,r_2,\ldots\ldots r_{10}\}$.
A1=title,
A2=authors
A3=booktitle
A4=year
r3[booktitle]="sigmod".
Without loss of generality, each tokenized keyword in the data set and queries is assumed to use lower case characters.
W= {privacy, sigmod, sigir,….}.

T – Threshold

$$\mathrm{ed}(s_1, s_2),$$
$$\mathrm{ed}(p, w) \leq \tau.$$

$$\mathrm{ed}(p, w) \leq \tau.$$

The edit distance between two strings s1 and s2, denoted by Ed (s1, s2), is the minimum number of single-character edit operations (i.e., insertion, deletion, and substitution) needed to transform s1 to s2. For example, (correlation; correlation) 1 an n d (correlation, correlation) 2. Given an edit-distance threshold $\mathrm{ed}(p, w) \leq \tau.$, we say a prefix p of a keyword in W is similar to the partial keyword w .We say a keyword d inW is similar to the partial keyword w if d has a prefix p such that $\mathrm{ed}(p, w) \leq \tau.$. Fuzzy search finds the records with keywords similar to the query keywords.

UDFs to support fuzzy search. We use a UDF $\mathrm{PED}(w, s)$ that takes a keyword w and a string s as two parameters, and returns the minimal edit distance between w and the prefixes of keywords in s.

$$\mathrm{PED}(\text{"pvb"}, r_{10}[\text{title}])$$
$$= \mathrm{PED}(\text{"pvb"}, \text{"privacy in database publishing "}) = 1$$

As r10 contains a prefix "pub" with edit distance of 1 to the query.

We can improve the performance by doing early termination in the dynamic programming computation using an edit-distance threshold (if prefixes of two strings are not similar enough, then the two substrings cannot be similar), and devise a new UDF $\mathrm{PEDTH}(w, s, \tau)$. If there is a keyword in string s having prefixes with an edit distance to w within T, PEDTH returns true. In this way, we issue an SQL query that scans each record and calls UDF PEDTH to verify the record.

$G^q(s)$
$|G^q(s)|$
$G^q(s).$

S1 - String
S2 – String
Q-

Substrings with length q. Let $G^q(s)$ denote the set4 of its q-grams and $|G^q(s)|$ denote the size of $G^q(s)$. For

example, for "pvldb" and "vldb," we have $G^2(\text{pvldb}) = \{\text{pv}, \text{vl}, \text{ld}, \text{db}\}$ $G^2(\text{vldb}) = \{\text{vl}, \text{ld}, \text{db}\}$. Strings s1 and s2 have an edit distance within threshold T if

$$|G^q(s_1) \cap G^q(s_2)| \geq \max(|s_1|, |s_2|) + 1 - q - \tau * q$$

Where js1j and js2j are the lengths of string s1 and s2, respectively. This technique is called count filtering.

4.4 Results of Practical Work

Here fig 2 shows framework of search as you type, in this initially user has to enter keywords to search related terms from database, if user not select any technique then system give results based on simple search.

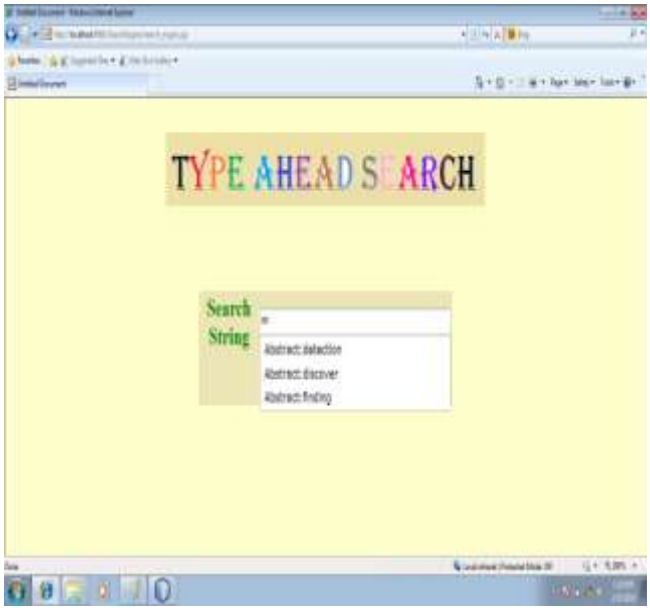

Fig 2: Framework of system

Fig 3: Result

Fig 4: Screenshot3

Fig 5 shows result of fuzzy index based search, here we combine fuzzy and index based methods to get better and accurate results from system.

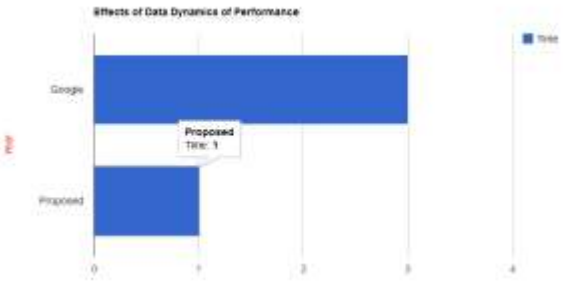

Fig 5 Graph Result

V. Conclusion and Future Work

In this paper, we studied the problem of using SQL to support type ahead search in data bases. We focused on the challenge of how to leverage existing DBMS functionalities to meet the high-performance requirement to achieve an interactive speed. To support prefix matching, we proposed solutions that use auxiliary tables as index structures and SQL queries to support type ahead search. We extended the techniques to the case of fuzzy queries, and proposed various techniques to improve query performance.

We suggested incremental-computation techniques to answer multi keyword queries, and studied how to support first-N queries and incremental updates. Our experimental results on large, real data sets showed that the proposed techniques can enable DBMS systems to support type ahead search on large tables.

References

Main Reference paper:

[1] 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.

[2] S. Agrawal, S. Chaudhuri, and G. Das, "DBXplorer: A System for Keyword-Based Search over Relational Data Bases," Proc. 18th Int'l Conf. Data Eng. (ICDE '02), pp. 5-16, 2002.

[3] A. Arasu, V. Ganti, and R. Kaushik, "Efficient Exact Set-Similarity Joins," Proc. 32nd Int'l Conf. Very Large Data Bases (VLDB '06), pp. 918-929, 2006.

[4] H. Bast, A. Chitea, F.M. Suchanek, and I. Weber, "ESTER: Efficient Search on Text, Entities, and Relations," Proc. 30th Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '07), pp. 671-678, 2007.

[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] 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.

[7] 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.

[8] 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.

[9] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, "An Efficient Filter for Approximate Membership Checking," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '08), pp. 805- 818, 2008.

[10] S. Chaudhuri, K. Ganjam, V. Ganti, R. Kapoor, V. Narasayya, and T. Vassilakis, "Data Cleaning in Microsoft SQL Server 2005," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '05), pp. 918-920, 2005.

[11] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and Efficient Fuzzy Match for Online Data Cleaning," Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '03), pp. 313- 324, 2003.

[12] 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.

[13] S. Chaudhuri, V. Ganti, and R. Motwani, "Robust Identification of Fuzzy Duplicates," Proc. 21st Int'l Conf. Data Eng. (ICDE), pp. 865- 876, 2005.

[14] S. Chaudhuri and R. Kaushik, "Extending Autocompletion to Tolerate Errors," Proc. 35th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09), pp. 433-439, 2009.

[15] B.B. Dalvi, M. Kshirsagar, and S. Sudarshan, "Keyword Search on External Memory Data Graphs," Proc. VLDB Endowment, vol. 1, no. 1, pp. 1189-1204, 2008.

[16] B. Ding, J.X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding Top-K Min-Cost Connected Trees in Data Bases," Proc. IEEE 23rd Int'l Conf. Data Eng. (ICDE '07), pp. 836-845, 2007.

[17] L. Gravano, P.G. Ipeirotis, H.V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate String Joins in a Data Base (Almost) for Free," Proc. 27th Int'l Conf. Very Large Data Bases (VLDB '01), pp. 491-500, 2001.

[18] M. Hadjieleftheriou, A. Chandel, N. Koudas, and D. Srivastava, "Fast Indexes and Algorithms for Set Similarity Selection Queries," Proc. IEEE 24th Int'l Conf. Data Eng. (ICDE '08), pp. 267-276, 2008.

[19] M. Hadjieleftheriou, N. Koudas, and D. Srivastava, "Incremental Maintenance of Length Normalized Indexes for Approximate String Matching," Proc. 35th ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '09), pp. 429-440, 2009.

[20] M. Hadjieleftheriou, X. Yu, N. Koudas, and D. Srivastava, "Hashed Samples: Selectivity Estimators for Set Similarity Selection Queries," Proc. VLDB Endowment, vol. 1, no. 1, pp. 201-212, 2008.