Supporting Search-As-You-Type Using SQL in Databases

C. Bhagya Laxmi[1], Bhaludra Raveendranadh Singh [2], Moligi Sangeetha[3]

[1]pursuing M.Tech (CSE), [2]Principal, [3]Associate Professor & HOD (CSE)

[1,2,3]Visvesvaraya College of Engineering and Technology (VCET), M.P Patelguda, Ibrahimpatnam (M), Ranga Reddy (D)-501510, India

ABSTRACT : **A search-as-you-type system determines answers while a user types a keyword query character by character. We study the search-as-you-type on data which is residing in a relational DBMS. We focus on native database language, SQL as how to support this type of search. A main challenge is how to make full use of existing database functionalities to meet the high-performance to achieve an interactive speed. Further we study how to use auxiliary indexes stored in the tables to increase the search performance. We present solutions for both multi keyword queries and single-keyword queries, and develop novel techniques for the fuzzy search using SQL by allowing the mismatches between query keywords and answers. Since the volume of data increases day to day the real world, the searching process has become unvaried. The good search technique must be render the requested data in a stipulated time based on the user requested query. Using the native query language, i.e SQL to implement search-as-you-type reduces programming effort. Here we are providing user requested result based on the user previous search.**

Keywords: databases, Search-as-you-type , fuzzy search, SQL

I. INTRODUCTION

Data mining broaches the extraction of knowledge from the large amounts of data. By performing interesting knowledge,data mining, regularities, high-level information could be extracted from the databases and browsed or viewed from different angles. Conventional information systems return answers after the user submits a entire query. Users often feel that "left in the dark" when they have limited knowledge about underlying data, and have to use the try-and-see approach for finding the information. Currently many information systems improving user search experiences by providing the instant feedback as users formulate the search queries. Most of the search engines and online search forms support the auto completion, which showing suggested queries or even answers also "on the fly" as the user types in query search box character by character. By this feature of instant feedback user could understand the data in addition to formulating query Most of the information systems nowadays improving the user search experiences by providing feedback as users formulate search queries. Search engines and online search forms support the auto completion, which suggests queries or even answers "on the fly" as the user types in a search box character by character. For instance, consider Web search interface at Netflix, which allows the user to search for movie information. If a user types in a half of the query "mad," the system gives movies with a title matching with this keyword as a prefix, such as Mad Men: Season 1 and "Madagascar" . The instant feedback helps the user in formulating the query, and in understanding the underlying data. This type of search is called as search-as-you-type or type-ahead search. As many search systems store their information in the backend relational DBMS, and many companies storing their info in RDBMS here the question arises naturally: is how to support search-as-you-type on data which is residing in a DBMS? Some databases such as SQL and Oracle server already support prefix search, and we could use this feature to do the search-as-you-type. Nevertheless, not all databases provide this feature. To overcome this, we study new methods that could be used in all databases. One approach is developing a separate application layer on database to construct indexes, and implement algorithms for answering the queries. With this approach has the advantage of getting a high performance, but its main drawback is duplicating indexes and data, which results in additional hardware costs. Another approach for search-as-you-type is to use database extenders, such as Informix DataBlades ,DB2 Extenders, Oracle Cartridges and Microsoft SQL Server Common Language Runtime (CLR) integration, which allow developers to implement new functionalities to the DBMS. This approach is not feasible for databases that do not provide such an extender interface, such as MySQL. As it needs to utilize proprietary interfaces provided by the database vendors, a solution for one database might not be portable to others. In addition to, an extender-based solution is, especially those which are implemented in C/C++, could cause serious security and reliability problems to database engines. In this article we study how to support search-as-you-type on relational DBMS systems using the native query language i.e SQL. In other words, we want to use the SQL to find answers to the search query as a user types in keywords character by character. Our objective is to utilize the built-in query engine of database system as much as possible. By this way, we can reduce the programming efforts to support the search-as-you-type. Furtherly, the solution developed on one database using standard SQL technique is portable to other databases which supports the same standard. Similar observation are also made by the Gravano and Jestes which use SQL to support similarity join in databases.

LEARNING to rank is a kind of the learning based information retrieval techniques, specialized in learning the ranking model with some documents labeled with their

relevancies to some of the queries, where the model is hopefully capable of ranking the documents returned to the arbitrary new query automatically. Based on the various machine learning methods, ex., Ranking SVM RankBoost , ListNet , RankNet, LambdaRank etc., the learning to rank algorithms have already shown their challenging performances in the information retrieval,especially Web search.

A main question when adopting this attractive idea is: Is it feasible and scalable? In a particular scenario, can SQL meet the high performance requirement to implement interactive search interface? Studies have shown that ,such an interface requires each query to be answered within 100 milliseconds. Relational DBMS systems are not specially designed for the keyword queries, making it more challenging to the support search-as-you-type.

II RELATED WORK

In this related work we are going to discuss different possible methods that are supports our approach support search-as-you-type and give their limitations and advantages. Using a separate application layer is the first method which can get very high performance as it can use various complex data structures and programming languages. Nevertheless, it is isolated from the RDBMS systems. Database extenders is the second method. Nevertheless, this extension-based method is "not safe" for the query engine, which could cause security and reliability problems to the database engine. This method depends on the API of a specific DBMS being used, and the different DBMS systems have different APIs. Furthermore, this method does not work if a DBMS system has no extender feature, ex., MySQL.Using SQL is the third method. The SQL-based method is more compatible as it is using the standard SQL. Even if DBMS systems do not provide search-as-you-type extension feature (indeed no Database Management Systems provide such an extension) the SQL-based method can also used in this particular case. So, the SQL-based method is more portable to the different platform than the first two methods.
A simple way to support search-as-you-type is to issue a SQL query that scans every record and verifies whether record is an answer to the query or not. There are 2 ways to do the checking: one is Calling User-Defined Functions i.e UDFs. We could add functions into the databases to verify whether a record contains query keyword; and second is: Using LIKE predicate. Databases provide the LIKE predicate to enable users to perform string matching. We can use LIKE predicate to check if a record contains the query keyword or not. This method might introduce false positives, example, keyword "publication" contains the query the string "ic," but the keyword does not have query string "ic" as a prefix. We can remove these false positives by calling the UDFs. The two no-index methods needs no additional space, but they may not scale as they need to scan all the records in the table.In this section, we propose to keep auxiliary tables as index structures to facilitate the prefix search. Some databases such as SQL server and Oracle have already support prefix search, and we can use this feature to do the prefix search. Nevertheless, not all the databases provide this feature. For this particular reason, we are developing a new method that could be used in all databases. Furthermore, we are maintaing inverted table that contains each keyword with specific unique id. Based on this specific keyword we can give the result to the user who is giving the request on-the-fly.

On the other hand ranking adaptation is closely related to the classifier adaptation, which has shown its efficiency for many learning .Nevertheless, to the best of our knowledge, there are no prior works on adaptation for the ranking problem. Furthermore the general difficulties faced by classifier adaptation, such as the covariate shift (or namely sample selection bias) and the concept drifting, ranking adaptation is relatively more challenging. Unlike the classifier adaptation, which mainly deals with the binary targets,and ranking adaptation desires to adapt model which is used to predict rankings for a collection of documents. Though documents are normally labeled with the several relevance levels, which seems to be handled by multi-class regression or classification,it is still difficult to directly use the classifier adaption for ranking. The reason lies in two-fold: one: in ranking, the mainly concentration is about the preference of the two documents or ranking of a collection of documents, which is a difficult to be modeled by the regression or classification; two : the relevance levels in between different domains are sometimes varied and need to be aligned. In this paper, we are also focusing on the adaptation of ranking models, instead of utilizing labeled data from the auxiliary domains directly, which might be inaccessible due to the privacy issue and data missing. Furthermore, Model adaptation is more advisable than data adaptation, because ,learning complexity is now only correlated to size of the target domain training set, which should be more smaller than size of auxiliary dataset.

III RESULTS

Search-as-you-type for single keyword:
Exact Search: As a user types keyword w in the search box character by character, the system we are developing search-as-you-type on-the-fly finds bunch of records that contain keywords with a prefix w. We call this search paradigm as prefix search. Without loss of generality, every tokenized keyword in data set and queries is assumed to use the lower case characters. For example, consider the data in Table 1, A1 ≈ title, A2 ≈ authors, A3 ≈ book title, and A4 ≈ year. In this exact search the keyword entered by the user is undergone to the DBMS engine and finds appropriate query with which it is started. If it found any matching then it will gives the query as suggestion to the user. By seeing this suggested result user could do the search more easily. In a particular situation user might not have proper idea about the query. In that situation

our system helps more. It reduces the user burden by giving on-the-fly suggestion.

TABLE 1
Table dblp: A Sample Publication Table (about "Privacy")

ID	Title	Authors	Booktitle	Year
r_1	K-Automorphism: A General Framework for Privacy Preserving Network Publication	Lei Zou, Lei Chen, M. Tamer Özsu	PVLDB	2009
r_2	Privacy-Preserving Singular Value Decomposition	Shuguo Han, Wee Keong Ng, Philip S. Yu	ICDE	2009
r_3	Privacy Preservation of Aggregates in Hidden Databases: Why and How?	Arjun Dasgupta, Nan Zhang, Gautam Das, Surajit Chaudhuri	SIGMOD	2009
r_4	Privacy-preserving Indexing of Documents on the Network	Mayank Bawa, Roberto J. Bayardo, Rakesh Agrawal, Jaideep Vaidya	VLDBJ	2009
r_5	On Anti-Corruption Privacy Preserving Publication	Yufei Tao, Xiaokui Xiao, Jiexing Li, Donghui Zhang	ICDE	2008
r_6	Preservation of Proximity Privacy in Publishing Numerical Sensitive Data	Jiexing Li, Yufei Tao, Xiaokui Xiao	SIGMOD	2008
r_7	Hiding in the Crowd: Privacy Preservation on Evolving Streams through Correlation Tracking	Feifei Li, Jimeng Sun, Spiros Papadimitriou, George A. Mihaila, Ioana Stanoi	ICDE	2007
r_8	The Boundary Between Privacy and Utility in Data Publishing	Vibhor Rastogi, Sungho Hong, Dan Suciu	VLDB	2007
r_9	Privacy Protection in Personalized Search	Xuehua Shen, Bin Tan, ChengXiang Zhai	SIGIR	2007
r_{10}	Privacy in Database Publishing	Alin Deutsch, Yannis Papakonstantinou	ICDT	2005

Search-as-You-Type for Multi keyword Queries:

Take a multi keyword query Q with m number of keywords they are w1; w2; . . . ; wm, as user is completing the last keyword that is wm, we treat wm as the partial keyword and the other keywords as complete keywords. As a user types in query Q letter by letter, our system search-as-you-type on-the-fly finds records that contain the complete keywords and the keyword with a prefix wm. For an example scenario, if a user types in a query "privacysig," the system search-as-you-type returns records as r3,r6, and r9. In a particular, r3 contains the complete keyword "privacy" and another keyword "sigmod" with a prefix "sig ."As user types the word it searches in DBMS with same matching pattern. If found correct query related to the user wish then it comes in suggestion box. If user type any one of the similar word and types other word which is not related to the actual query in that case it won't give any suggestion. As each query is divided into some number of keywords. So unrelated word never found in the DBMS system so the system search-as-you type will not give any suggestion to the user.

Fuzzy Search:

In some of the cases the information is inserted into DBMS by special words. Here the special words means some of the information is stored with a name which is not related to that particular information. As a result user will never find the information as if he/she go normal search. By having this discussion it is worth full to have fuzzy search. Fuzzy means anonymous. The information is stored with anonymous name; For this in our system search-as-you-type we are developing fuzzy search also. In this type of search admin can upload the information of files with anonymous names. If user types the keyword related to the information file he/she won't get any kind of suggestions. If user enters the proper anonymous keyword only the files which reside in the database comes as suggestion for user.By making this we can provide little security and only limited persons are allowed to access. So this fuzzy search helps in giving security when compared to the normal search.

Inverted table UDF Search:

I this UDF search scenario the query is partitioned by different words in the correspond query with a specific index for each word. All these queries are uploaded by the admin into DBMS. For example a query contains 10 words like w1,w2,w3…w10 as words. For all these word a specific number or unique number is generated. And in other hand the entire query also will have specific or unique number to identify the query. Now if a user enters a number then it checks whether the number is available in the keywords list or not. If it founds in keyword list then it gives the suggestion as the main query which is connected to the keyword list. By this the inverted table helps in our system search-as-you-type on giving the results on-the-fly.

Rank based Suggestions:

Apart from all these search methodologies we are also giving rank based suggestions to the user. In this aspect we are taking the user click as feedback and based those feedback we are giving suggestions to the user. For example a user enter a query and for that query out system search-as-you-type gives many suggestions among those suggestion user might interested in any query. Now we are getting which query is further processed as feedback. Based on this feedback we are increasing the rank of that query. If any other user comes do search with the same keyword which is having higher rank then it will be visible on the first row. Therefore based on the ranks of the query our system is going to give response within fraction of seconds.

IV CONCLUSION

In this article, we studied the problem of using the SQL to support the system search-as-you-type in data bases. And implemented various kinds of search techniques. We mainly concentrated on the challenge of how to make full use of the existing DBMS functionalities to meet high-performance requirement to get an interactive speed. To support the prefix matching, we proposed a solutions that uses the auxiliary tables as index structures and SQL queries to support the search-as-you-type. We enhanced the techniques in the case of fuzzy queries, and proposed various techniques to improve the query performance. We proposed multi keyword queries search, and studied how to support first-N queries and the incremental updates. And we are getting the feedback of the user requested queries and based on that we are giving rank to those queries. This is very helpful in the rank based search.

V REFERENCES

[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.

AUTHOR PROFILE

C.Bhagya Laxmi is currently pursuing M.Tech in the Department of Computer Science & Engineering, Visvesvaraya College of Engineering and Technology, M.P Patelguda, Ibrahimpatnam (M), Ranga Reddy(D), India.

Sri Dr. Bhaludra Raveendranadh Singh working as Associate Professor & Principal in Visvesvaraya College of Engineering and Technology. He obtained M.Tech, Ph.D(CSE)., is a young, decent, dynamic Renowned Educationist and Eminent Academician, has overall 20 years of teaching experience in different capacities. He is a life member of CSI, ISTE and also a member of IEEE (USA).

Ms's. Sangeetha M working as Assoc. Professor & HOD (CSE). She has completed bachelor of technology from Swamy Ramananda Theertha Institute of Science & Technology and Post-graduation from Jawaharlal Nehru Technological University,Kakinada campus and is having 12 years of teaching experience.