International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

Search-as-You-Type by Using SQL in Bulk Databases

D.Revathi[1]& A. Malla Reddy[2]

[1]PG Scholar, Dept of CSE, Sri Indu Institute of Engineering & Technology, Sheriguda, Ibrahimpatnam, Telangana,501510

[2]Professor & HOD , Dept of CSE, Sri Indu Institute of Engineering & Technology, Sheriguda, Ibrahimpatnam, Telangana,501510

ABSTRACT:
A search-as-you-type system computes answers on-the-fly as a user types in a keyword query character by character. We study how to support search-as-you-type on data residing in a relational DBMS. We focus on how to support this type of search using the native database language, SQL. A main challenge is how to leverage existing database functionalities to meet the high performance requirement to achieve an interactive speed. We study how to use auxiliary indexes stored as tables to increase search performance. We present solutions for both single-keyword queries and multi keyword queries, and develop novel techniques for fuzzy search using SQL by allowing mismatches between query keywords and answers. We present techniques to answer first-N queries and discuss how to support updates efficiently. Experiments on large, real data sets show that our techniques enable DBMS systems on a commodity computer to support search-as-you-type on tables with millions of records.

Index Terms—Search-as-you-type; databases; SQL; fuzzy search

INTRODUCTION: MANY information systems nowadays improve user search experiences by providing instant feedback as users formulate search queries. Most search engines and online search forms support auto completion, which shows suggested queries or even answers "on the fly" 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 understanding the underlying data. This type of search is generally called search-as-you-type or type-ahead search. Since many search systems store their information in a backend relational DBMS, a question arises naturally: how to support search-as-you-type on the data residing in a DBMS? Some databases such as Oracle and SQL server already support prefix search, and we could use this feature to do search-as-you-type. However, not all databases provide this feature. For this reason, we study new methods that can be used in all databases. One approach is to develop a separate application layer on the database to construct indexes, and implement algorithms for answering queries. While this approach has the advantage of achieving a high performance, its main drawback is duplicating data and indexes, resulting in additional hardware costs. Another approach is to use database extenders, such as DB2 Extenders, Informix Data Blades, Microsoft SQL Server Common Language Runtime (CLR) integration, and Oracle Cartridges, which allow developers to implement new functionalities to a DBMS. This approach is not feasible for databases that do not provide such an extender interface, such as MySQL. Since it needs to utilize proprietary interfaces provided by database vendors, a solution for one database may not be portable to others. In addition, an extender-based

International Journal of Research
International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

solution, especially those implemented in C/C++, could cause serious reliability and security problems to database engines. In this paper we study how to support search-as-you-type on DBMS systems using the native query language (SQL). In other words, we want to use SQL to find answers to a search query as a user types in keywords character by character. 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. Similar observation are also made by Gravano et al. and Jestes et al. which use SQL to support similarity join in databases.

A main question when adopting this attractive idea is: Is it feasible and scalable? In particular, can SQL meet the high performance requirement to implement an interactive search interface? Studies have shown that such an interface requires each query be answered within 100 milliseconds [38]. DBMS systems are not specially designed for keyword queries, making it more challenging to support search-as-you-type. As we will see later in this paper, some important functionalities to support search-as-you-type require join operations, which could be rather expensive to execute by the query engine.

PRELIMINARIES: We first formulate the problem of search-as-you-type in DBMS and then discuss different ways to support search-as-you-type.

Problem Formulation: Let T be a relational table with attributes A1;A2; . . .;A‘. Let R ¼ fr1; r2; . . . ; rng be the collection of records in T, and ri½Aj_ denote the content of record ri in attribute Aj. Let W be the set of tokenized keywords in R.

Search-as-You-Type for Single-keyword Queries

Exact Search: As a user types in a single partial (prefix) keyword w character by character, search-as-you-type on the- fly finds the records that contain keywords with a prefix w.We call this search paradigm prefix search. Without loss of generality, each tokenized keyword in the data set and queries is assumed to use lower case characters. For example, consider the data in Table 1, A1 ¼ title, A2 ¼ authors, A3 ¼ booktitle, and A4 ¼ year. R ¼ fr1; . . . ; r10g. r3[booktitle] ¼ ‘‘sigmod’’.

EXACT SEARCH FOR SINGLE KEYWORD: This section proposes two types of methods to use SQL to support search-as-you-type for single-keyword queries. In , we discuss no-index methods. We build auxiliary tables as index structures to answer a query.

No-Index Methods: A straightforward way to support search-as-you-type is to issue an SQL query that scans each record and verifies whether the record is an answer to the query. There are two ways to do the checking: 1) Calling User-Defined Functions (UDFs). We can add functions into databases to verify whether a record contains the query keyword; and 2) Using the LIKE predicate. Databases provide a LIKE predicate to allow users to perform string matching. We can use the LIKE predicate to check whether a record contains the query keyword. This method may introduce false positives, e.g., keyword “publication” contains the query string “ic,” but the keyword does not have the query string “ic” as a prefix. We can remove these false positives by calling UDFs. The two no-index methods need no additional space, but they may not scale since they need to scan all records in the table

Index-Based Methods: In this section, we propose to build auxiliary tables as index structures to facilitate prefix search. Some databases such as Oracle and SQL server already support prefix search, and we could use this feature to do prefix search. However, not all databases provide this feature. For this reason, we develop a new method that can be used in all databases. In addition, our experiments in Section

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

8.3 show that our method performs prefix search more efficiently. Inverted-index table. Given a table T, we assign unique ids to the keywords in table T, following their alphabetical order. We create an inverted-index table IT with records in **n**the form hkid; ridi, where kid is the id of a keyword and rid is the id of a record that contains the keyword. Given a complete keyword, we can use the inverted-index table to find 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 ½lkid; ukid_, and each complete word in the table T with an ID in this keyword range must have a prefix p. Thus, given a prefix keyword w, we can use the prefix table to find the range of keywords with the prefix.

FUZZY SEARCH FOR SINGLE KEYWORD

No-Index Methods

Recall the two no-index methods for exact search in Since the LIKE predicate does not support fuzzy search, we cannot use the LIKE-based method. We can use Index-Based Methods This section proposes to use the inverted-index table and prefix table to support fuzzy search-as you-type. Given a partial keyword w, we compute its answers in two steps. First we compute its similar prefixes from the prefix table PT , and get the keyword ranges of these similar prefixes.\ Then we compute the answers based on these ranges using the inverted-index table IT as discussed in. In this section, we focus on the first step: computing w's similar prefixes.

Using UDF

Given a keyword w, we can use a UDF to find its similar prefixes from the prefix table PT . We issue an SQL query that scans each prefix in PT and calls the UDF to check if the prefix is similar to w. We issue the following SQL query to answer the prefix-search query. Incrementally Computing Similar Prefixes The previous methods have the following limitations. First, they need to find similar prefixes of a keyword from scratch. Second, they may need to call UDFs many times. In this section, we propose a character-level incremental method to find similar prefixes of a keyword as a user types character by character.

EXISTING SYSTEM:

Most search engines and online search forms support auto completion, which shows suggested queries or even answers "on the fly" as a user types in a keyword query character by character.

Since many search systems store their information in a backend relational DBMS, a question arises naturally: how to support search-as-you-type on the data residing in a DBMS? Some databases such as Oracle and SQL server already support prefix search, and we could use this feature to do search-as-you-type. However, not all databases provide this feature. For this reason, we study new methods that can be used in all databases. One approach is to develop a separate application layer on the database to construct indexes, and implement algorithms for answering queries.

DISADVANTAGES OF EXISTING SYSTEM:

- In an existing systems are not specially designed for keyword queries, making it more challenging to support search-as-you-type.
- SQL meet the high performance requirement to implement an interactive search interface.
- Some important functionality to support search-as-you-type requires join operations, which could be rather expensive to execute by the query engine.

International Journal of Research (IJR)
e-ISSN: 2348-6848, p- ISSN: 2348-795X Volume 2, Issue 09, September 2015
Available at http://internationaljournalofresearch.org

PROPOSED SYSTEM:

In this paper, we develop various techniques 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.

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. We study how to support fuzzy search for single-keyword queries.

We discuss a gram-based method and a UDF-based method. As the two methods have a low performance, we propose a new neighborhood-generation based method, using the idea that two strings are similar only if they have common neighbors obtained by deleting characters.

We extend the techniques to support multi-keyword queries. We develop a word-level incremental method to efficiently answer multi-keyword queries. Notice 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.

ADVANTAGES OF PROPOSED SYSTEM:

- A main challenge is how to utilize the limited expressive power of the SQL language (compared with other languages such as C++ and Java) to support efficient search.
- We study how to use the available resources inside a DBMS, such as the capabilities to build auxiliary tables, to improve query performance.
- An interesting observation is that despite the fact we need SQL queries with join operations, using carefully designed auxiliary tables, built-in indexes on key attributes, foreign key constraints, and incremental algorithms using cached results, these SQL queries can be executed efficiently by the DBMS engine to achieve a high speed.

CONCLUSION AND FUTURE WORK

In this paper, we studied the problem of using SQL to support search-as-you-type 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 search-as-you-type. We extended the techniques to the case of fuzzy queries, and proposed various techniques to improve query performance. We proposed 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 search-as-you-type on large tables. There are several open problems to support search-as you- type using SQL. One is how to support ranking queries efficiently. Another one is how to support multiple tables.

REFERENCE:

[1.] Guoliang Li, Jianhua Feng,Member, IEEE, and Chen Li, Member, IEEE " Supporting Search-As-You-Type Using SQL in Databases"- **IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 25, NO. 2, FEBRUARY 2013.**