ISSN:0975-9646

"Improving Performance in Keyword Search by Supporting Search as-you-type"

Mr. Datta M. Ningole ME (CSE), Prof. Vijay B. Patil

Department of Computer Science and Engineering
MIT College of Engineering, Aurangabad Maharashtra

Abstract: Now a day, data become an important factor in our day to day life. As volume of data increases, searching data has become a tedious searching process. Search is one of the most basic and important tool which are used in most of application. Most of search engines provide a feature known as "search as-you-type". This feature allows you to get answer on fly as user types a query character by character. In our paper, we studied exact and fuzzy search on single and multi-keyword. We focus on how "search as-you-type" feature are work on backend relational database. The implementation of "search as-you-type" includes many challenges that include security issue, application compatibility in all platforms and response time of application. To increase overall performance of searching we will uses indexes in tables. Lastly we have tested our application on large and real time data with millions of records that shows far better good results.

Keywords— Exact Search, Fuzzy Search, Like and UDF methods, Gram based method, Incremental Computation method, Neighborhood Generation method, Inverted table method.

I. INTRODUCTION

More information systems currently improved the user search experiences by providing instant feedback as the users verbalize search query. Frequently search engine, online search forms support sedan completion which are shows recommended queries or even answers on fly as the user types in the keyword query character by the character. Since instance consider Web search interface at the Netflix which tolerates the user to search for the movie information. Whether the user types in the partial query mad system shows movies with the title matching this keyword as the prefix such as Madagascar and Mad Men The instant feedback helps the user not only in the formulating the query then also in understanding underlying data. This is type of the search generally called search as you type or type onward search. Therefore additional search systems store their information in the Backend interpersonal DBMS question arises naturally how to the support search as you type on data residing in the DBMS. Some databases such as the Oracle and SQL server support prefix search .We study new method that can be used in all databases. Once the methodology is to the developed the separate application layer on to the database to construct indexes and the implement algorithm is for the answering queries. However this approach has the advantage of the achieving the high performance it is main drawbacks are duplicating data and the indexes resulting in the additional hardware cost. The alternative methodologies are to the use database extenders such as the DB2 Extenders Informix Data Blades, Microsoft SQL Servers and Oracle Cartridges which is allow developers to the implement novel functionalities to DBMS. In this type of approach isn't feasible for databases that don't provide such extender interface such as MySQL database. Another approach is to use standard SQL techniques which are also portable to other databases. We compare this "Standard SQL" technique with our proposed technique for exact and fuzzy search.

Organization of the Paper:

The rest of the paper is organized as follows. Section II represents the related work. Section III represents the system model with terminology and recalls the some background concepts. Section IV represents the proposed work for exact and fuzzy search related to single and multi-keyword query. Section V illustrates the performance evaluation of the proposed algorithm. Section VI represents the performance analysis of the proposed work. Section VII states our conclusion and possible extensions for a future work.

II. RELATED WORK

In particular, there are two types of search which is mostly observed, namely multikeyword search and fuzzy search. In multi-keyword search techniques, a user types in query containing multiple keywords, and find tuples that are similar to these keywords and the location of keywords. For example, if a user types in "Database System" to find out a book by "Mr. S.B.Navathe" with a title including "Database" and "System" with irrespective of the locations. In fuzzy search, minor differences may be present between query keyword and actual results. For example, if a user types in "Navthe" despite the word "Navathe", then this type of search techniques are useful. Depending on these search techniques, multiple methods have been discussed later in the paper.

In related work we studied about previous approach used to support "Search as-you-type'. This includes application layer based approach[1], Database extender, "Using standard SQL" and "BANKS".

2.1 Application Layer

Many search engines and online search forms are support to auto completion. It shows suggested queries or even answers "search as you type" as a user types in a keyword query. In an existing systems are not specially designed for keyword queries, it become more difficult to support search-as-you-type. SQL meet the high performance

ID	Title	Authors	BookTitle	Year
r1	K-Auto morphism: A General Framework for Privacy Preserving Network Publication	Lei Zou, Lei Chen, M. Tamer O¨ zsu	PVLDB	2009
r2	Privacy-Preserving Singular Value Decomposition	Shuguo Han, Wee Keong Ng, Philip S. Yu	ICDE	2009
r3	Privacy Preservation of Aggregates in Hidden Databases:	Arjun Dasgupta, Nan zhang, Gautam Das, Surajit	SIGMOD	2009
r4	Privacy-preserving Indexing of Documents on the Network	Mayank Bawa, Roberto J. Bayardo, Rakesh Agrawal	VLDBJ	2009
r5	On Anti-Corruption Privacy Preserving Publication	Yufei Tao, Xiaokui Xiao, Jiexing Li, Donghui Zhang	ICDE	2008
r6	Preservation of Proximity Privacy in Publishing Numerical Sensitive	Jiexing Li, Yufei Tao, Xiaokui Xiao	SIGMOD	2008
r7	Hiding in the Crowd: Privacy Preservation on Evolving Streams	Feifei Li, Jimeng Sun, Spiros Papadimitriou,	SIGIR	2007
r8	The boundary between privacy and utility in data publishing	Vibhor Rastogi,sungho	VLDB	2007
r9	Privacy protection in personalized Search	Xuehua Shen, Bin Tan	SIGIR	2007

Table1 .DBLP in "Publication" relational database

requirement to get a "search as you type " search interface[1]. To support search-as-you-type requires multiple join operations, which could be rather expensive to execute by the query engine.

2.2 Database Extender

Another approach is by using database extenders, such as Informix, DB2 Extenders, Microsoft SQL Server (CLR) integration, and Oracle Cartridges. It allows developers to add new functionalities to a DBMS. This approach is not feasible for databases which do not provide such an extender interface. MySQL is not providing database extender so this it is not useful for MySQL. Since it needs to utilize proprietary interfaces provided by database vendors, a solution for one database is not be portable to others. In addition, an extender-based solution, especially those developed in C/C++, could cause reliability and security problems to database engines.

2.3 Use "Standard SQL"

The third method is to use SQL[2]. The SQL-based method is more compatible since it is using the standard SQL. Even if DBMS systems do not provide the search-as-you-type extension feature, the SQL-based method can also be used. Thus, the SQL-based method is more portable to a different platform than the first two methods.

2.4"BANKS"

BANKS is Keyword Searching and Browsing in Database using BANKS [3]. BANKS allow user with no knowledge of database system or schema to query and browse relational database with ease. It reduces the effort involved in publishing relational data on web and making it searchable. BANKS model the database as directed graph and table as nodes in the graph .Relationship between these tables are used as edges between this nodes. BANK is not feasible for large database.

III. SYSTEM MODEL

We will first plan the problem of search-as-you-type in database management system and then we will discuss different ways to support search-as-you-type.

3.1 Problem Formulation

Let T be a relational table with attributes A1;A2; . . .;An. Let {r1; r2; . . . ; rn} be the collection of records in T, and ri[Aj] denote the content of record ri in attribute Aj. Let W be the set of tokenized keywords in R.

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

Exact Search: As a user types in a single partial keyword w character by character as prefix, search-as-you-type finds the records that contain keywords with a prefix w. For example, consider the data in Table 1, A1 = title, A2 = authors, A3 = booktitle, and A4 = year. R ={r1; . . . ; r10}. r3[booktitle] = ''sigmod''. W ={privacy; sigmod; sigir; . . .}If a user types in a query "sig" it get records of rows having id r3, r6, and r9. In particular, r3 contains a keyword "sigmod" with a prefix "sig". "sig" is prefix of keyword "sigmod".

Fuzzy Search: As a user types in a single partial keyword w, fuzzy search finds records with keywords similar to the query keyword. In Table 1, assuming a user types in a query "corel," It returns record r7 because it contains a keyword "correlation" with a prefix "correl" similar to the query keyword "corel." Edit distance method are used to measure the similarity between strings ed(s1, s2) is known as the edit distance between two strings s1 and s2. It is the minimum number of single-character edit operations such as insertion, deletion, and substitution needed to transform s1 to s2.

3.1.2)Search-as-You-Type for Multikeyword Queries

Exact Search: Consider a multi-keyword query Q with m keywords w1,w2, . . . , Wm, as the user is completing the last keyword Wm, it consider Wm as a partial keyword and other keywords as complete keywords. As a user types in query Q, search-as-you-type it finds the records that contain the complete keywords and a keyword with a prefix Wm. if a user types in a query "privacysig" search-as-you-type it get records having row id r3, r6, and r9. Particularly r3 contains the complete keyword "privacy" and a keyword "sigmod" with a prefix "sig".

Fuzzy Search: Fuzzy search the records that contain keywords similar to the complete keywords and a keyword with a prefix similar to partial keyword Wm. Suppose edit-distance threshold t= 1 as a user types in a query "privicycorel" fuzzy search it returns record r7 since it contains a keyword "privacy" similar to the complete keyword "privicy" and prefix "correl" contain in keyword "correlation" which look similar to "corel" as the partial keyword.

IV. PROPOSED WORK

System proposes two types of methods to use SQL to support search-as-you-type for single-keyword queries.

4.1 No-Index Methods

An appropriate 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. It consists of two ways to do the checking: 1) Calling User-Defined Functions. It can add functions into databases to verify whether a record contains the query keyword; and 2) Using the LIKE predicate. LIKE predicate performs string matching. LIKE predicate are used to check whether a record contains the query keyword. This method introduces false positivity, e.g., keyword "publication" contains the query string "ic," but the keyword does not have the query string "ic" as a prefix. It can remove these false positives by calling UDFs.

Proposed system based on auxiliary tables to build as index structures to provide a prefix search. Some databases like Oracle and SQL server already support prefix search, and it could use this feature to do prefix search. But not all databases provide prefix search. For this reason, it develops a new method that can be used in all databases.

Inverted-index table: Given a table T, assign unique ids to the keywords in table T, following their alphabetical order. It creates an inverted-index table IT with records in the form <kid; rid>. To find records with the keyword it can use the inverted-index table.

Prefix table:Given a table T, for all prefixes of keywords in the table, prefix table PT developed with records in the form <p; lkid; ukid> p- prefix of a keyword, lkid - smallest id of those keywords in the table T having p as a prefix, and ukid- the largest id of those keywords having p as a prefix. It is observed that a complete word with p as a prefix must have an ID in the keyword range {lkid; ukid} and every complete word in the table T with an ID in this keyword range must have a prefix p. Thus, given a prefix keyword w, system can use the prefix table to find the range of keywords with the prefix.

kid	Keyword
k1	Icde
K2	Icdt
K3	Preserving
K4	Privacy
K5	Publishing
K6	Sigmoid
k7	Sigir
k8	Sigmod

a) Keyword Table

Prefix	lkid	ukid
Ic	k1	k2
P	k3	k6
pr	k3	k4
pri	k4	K4
pu	k5	k5
pv	k6	K6
Sig	k6	K8

(b) Inverted Index Table

kid	Rid
K2	r10
K5	r6
K5	r8
K5	r10
k6	r1
K7	r9
k8	r3

(c) Prefix Table

Table 2. The Inverted-Index Table and Prefix Table

For example, as shown in table 2 this illustrates the inverted-index table and the prefix table for the records. The inverted index table has a tuple <k8; r3> keyword k8 ("sigmod") is present in record r3. The prefix table has a tuple <''sig'',k7; k8> keyword k7 ("sigir") is the minimal id of keywords with a prefix "sig," and keyword k8 ("sigmod") is the maximal id of keywords with a prefix "sig." the range of ids of keywords with a prefix "sig" is [k7; k8].We use the following SQL to answer the prefix-search query w:

SELECT $T.*$ FROM P_T, I_T, T
WHERE P_T.prefix = $"w"$ AND
$P_T.ukid \geq I_T.kid$ AND $P_T.lkid \leq I_T.kid$ AND
$I_T.rid = T.rid.$

For example, assuming a user types in a partial query "sig" on table (Table 1), it issue the followingSQL:

SELECT dblp.* FROM P_{dblp}, I_{dblp}, dblp
WHERE P_{dblp}.prefix = "sig" AND
$P_{dblp}.ukid \geq I_{dblp}.kid$ AND $P_{dblp}.lkid \leq I_{dblp}.kid$ AND
$I_{dblp}.rid$ = dblp.rid,

It returns records r3, r6, and r9. The SQL query first finds the keyword range [k7; k8] in prefix table. After that it finds the records containing a keyword with ID in [k7; k8] using the inverted-index table. To get the answer efficiently from the SQL query, we develop built-in indexes on attributes prefix, kid, and rid. The SQL first use the index to find the keyword range on prefix, and then compute the answers using the indexes on kid and rid.

FUZZY SEARCH FOR SINGLE KEYWORD

4.2.1 No-Index Methods

The LIKE predicate does not support fuzzy search, it cannot use the LIKE-based method. Proposed system can use UDFs to support fuzzy search. it use a UDF 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. PED(''pvb''; r10[title])= PED(''pvb''; ''privacy in database publishing'') = 1 as r10 contains a prefix "pub" with edit distance of 1 to the query.

Index-Based Methods

Proposed system approaches to use the inverted-index table and prefix table to support fuzzy search-as-you-type. Given a partial keyword w, compute its answers in two steps. First compute its similar prefixes from the prefix table PT, and get the keyword ranges of these similar prefixes. Then it computes the answers based on these ranges using the inverted-index table IT.

4.2.1.1 Using UDF

Given a keyword w, system can use a UDF to find its similar prefixes from the prefix table PT .It issue an SQL query that scans each prefix in PT and calls the UDF to check if the prefix is similar to w. It issues the following SQL query to answer the prefix-search query w

SELECT $T.*$ FROM P_T, I_T, T
WHERE PEDTH$(w, P_T, \text{prefix}, \tau)$ AND
$P_T.ukid \geq I_T.kid$ AND $P_T.lkid \leq I_T.kid$ AND
$I_T.rid = T.rid$.

System can use length filtering to improve the performance, by adding the following clause to the where clause: "LENGTH (PT: prefix) <= LENGTH (w) +r AND LENGTH (PT: prefix) >= LENGTH (w)-r".

4.2.1.2 Gram-Based Method

There are many q-gram-based methods to support approximate string search[6]. Given a string s, its q-grams are its substrings with length q. Let Gq(s) denote the set of its q-grams and |Gq(s)j| denote the size of Gq(s). For example, for "pvldb" and "vldb," it has |G2(pvldb)= {pv, vl, ld, db} and G2(vldb)= {vl; ld; db}. Strings s1 and s2 have an edit distance within threshold t if where |s1| and |s2| are the lengths of string s1 and s2, respectively. This technique is called count filtering. This method may contain false positives to avoid it we use UDFs to verify the candidates to get the similar prefixes of w. Fig. 1 illustrate how to use the gram-based method to answer a query. It can further improve the query performance by using additional filtering techniques.

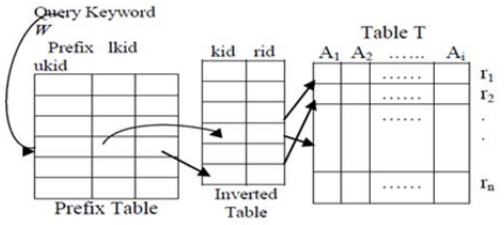

Fig 1 using the q gram table and the neighborhood generation table to support fuzzy search.

It could be expensive to use "GROUP BY" in databases, and the q-gram-based method is inefficient, especially for large q-gram tables. Moreover, this method is rather inefficient for short query keywords, as short keywords have smaller numbers of q-grams and the method has low pruning power.

4.2.1.3 Neighborhood-Generation-Based Method

Uk konen proposed a neighborhood-generation-based method to support approximate string search. Proposed systems 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 neighborhoods" of w. Let Di(w) denote the set of i-deletion neighborhoods of w and D Dt(w)=$U^{t}_{i=0}D_i$ (w). For example, given a string "pvldb," D0(pvldb) = {pvldb}, and D1{pvldb} ={vldb; pldb; pvdb, pvlb; pvld}. Suppose t= 1, eDt (pvldb) = {pvldb; vldb; pldb; pvdb; pvlb; pvld}. Moreover, there is a good property that given two strings s1 and s2, if ed(s1; s2) <=, eDt (s1) and eDt(s2)!=null as formalized in Lemma 1. This method is efficient for short strings. However, it is inefficient for long strings, especially for large edit-distance thresholds.

SUPPORTING MULTIKEYWORD QUERIES

We propose systematic techniques to support multikeyword queries.

1) Given a multikeyword query Q with m keywords w1 ; w2; . . . ; wm, there are two ways to answer it from scratch.1) Using the "INTERSECT" Operator: first compute the records for each keyword using the previous methods, and then use the "INTERSECT" operator to merge these records for different keywords to compute the answers.

2) Using Full-text Indexes: Here first use full-text indexes (e.g., CONTAINS command) to get records matching the first m 1 complete keywords, and then use proposed method to find records matching the last prefix keyword. Finally, we join the results. These two methods cannot use the pre computed results and may lead to low performance. To address this problem, we propose an incremental

computation method.The basic approach for keyword searching is as shown in following figure 2.

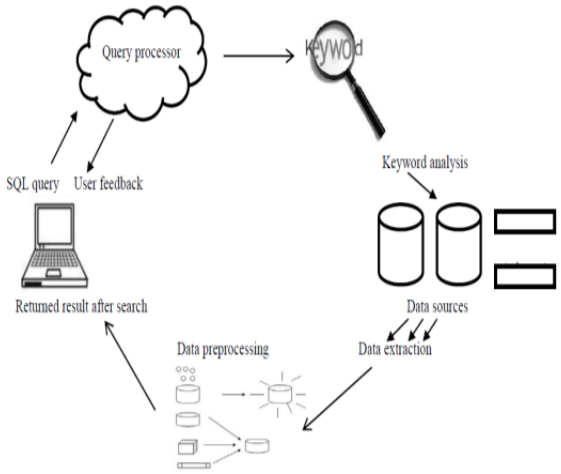

Fig 2 Searching keyword Using SQL

V. PERFORMANCE EVALUATION

This chapter evaluates performance of proposed system and performance of proposed system is analyzed with various exact and fuzzy search method on the parameters of Query time, memory used and number of record fetch. We implemented the proposed methods on real data sets i.e. "DBLP": It included 1.2 million computer sciences.The size of inverted-index table and prefix table is acceptable, compared with the data set size. As a keyword may have many deletion-based neighbors, the size of prefix-deletion table is rather large.

5.1 Experimental Setup

The system is implemented in Netbean IDE 8.0.2 environment, in which it will use the Netbean IDE 8.0.2 framework. While the front end is in Java the backend *i.e.* Database is in SQL Server 2008. This application can be used as a desktop or window application. The Results are successfully displayed on window. It examine the overall query time to return records when it apply exact and fuzzy search that has multiple methods on "paper" and "author" tables. System analysis is performed on real data sets. For now the impact of parameters are evaluated on the basis of Query Time, Heap memory used and number of record fetch.

Data Sets: System consists of two real DBLP dataset tables which provide both exact and fuzzy search result. The first table is "author" for author's details and "Paper" for paper details. "paper" table has column "title", "year", "conference" and "paper_key". In second table "author" table has column "paper_key" and "author_name".

5.2 Evaluation of Case Studies

In this chapter, the old method "Using Standard SQL" method which is based on no indexed method are going to compare with our proposed methods. The performance factors are query time, max heap memory used and number of record fetch.

Query time: Query time is the time in between from query submission to record or data retrieval from database. To get query time system note down the start time stamp at which query entered and getting results from query as end time. By calculating difference between start and end time system get query time. This query time is used to compare with "No-Indexed" method i.e. "Standard SQL".

Max heap memory: Max heap memory is used as one of the performance factor for proposed system. It calculate max heap memory used during data retrieval.

Number of record fetch: The number of record fetch from table is calculated for exact and fuzzy search in single and multi-keyword query. Proposed system consists of various methods which provide a feasible result. It uses this factor to compare the result.

Case Study 1:

In this case study user gives the query to the system, user get record from various methods. Here query keyword is "Public-Private" for single keyword search. First no indexed method it gives result which contain row id, title, Author, Book title and Year. Single Keyword:

Exact Search

i) It gives the following result. It is output for No-Index method.

r105	Enhancing Public-Private Partnerships Through SMS	Kristina Lugo	ANT/MobiWIS 2012

It requires the 2213 millisecond to fetch record from dataset. Proposed system provides an "Indexed-Based" which searches a query keyword "Public-Private" as prefix of word, firstly add into in keyword table. After it adds into Inverted Index Table .Next step is adding prefix for "Public-Private" query keyword as "Public" prefix in Prefix table. In proposed system search is totally depend upon overall search i.e. prefix search. It gives same result in 114 Millisecond when search as prefix "Public".

ii) Fuzzy Search

In fuzzy search consist of "No-Indexed" method which gives result of fuzzy search for query keyword "Public-Private". Proposed system consists of "Indexed Based" method which contain "UDF", "Grambased" and "NGB" i.e. neighborhood generation method. Input is "Public" as prefix. It gives following result in 1379 millisecond. In Proposed system consists of "Using UDF" and "gram Based". In "Using UDF" it need prefix" Public" to get result, this results get in only 25 millisecond. Then in "gram Based" it needs two inputs. First is keyword which have search "Public-Private" and substring "Public" if substring is valid then it get result, otherwise it does not give result. "Gram Based" method provides the results in 1580 millisecond.

2. Multi-keyword

Exact search: In multi keyword consists of exact search and fuzzy search for multi-keyword. So another keyword used with "Public-Private" is "Portuguese". Multi keyword exact search method search the records contain with "Public-Private" and "Portuguese" keyword. It fetches 13 records. 1 record for "Public-Private" and 12 results for "Portuguese". Query time required to fetch exact search result is 3931.

Fuzzy search: Multi keyword fuzzy search method search the records contain with "Pulic" and "Potu" keyword. It gives the 2 records which contain "Public" and "Portu" keywords. It gives result with respect to similar match keywords. It fetches result in 40 milliseconds which is far less as compared to exact search.

3. First-n- query

This proposed method limits the results which are fetched from tables. It provides filtering technique to exact and fuzzy search. It needs to provide a limit. The limit restricts the result to show the restricted or particular number of results.

Exact Search: In first n query consists of exact search and fuzzy search for multi-keyword. So another keyword used with "Public-Private" is "Portuguese". First n query supports multi keyword exact search method search the records contain with "Public-Private" and "Portuguese" keyword. It fetches the records which is less than or equal to that limit i.e. 3. It shows only top 3 results. It's required query time is 2887 millisecond. Heap memory uses up to 42 MB.

Fuzzy Search: Top n query supports multi keyword fuzzy search method that searches the records contain with "Pulic" and "Potu" keyword. It gives the 2 records which contain "Public" and "Portu" keywords. It gives result with respect to similar match keywords. It fetches result in 37 milliseconds which is far less as compared to exact search.

Case Study 2:

In case study second, query keyword as single word is "Satellites" and other word used with this keyword is "quasi-birth-and-death". "Satellites" and "quasi-birth-and-death" are used with in multi keyword search.

1. Single keyword

i) Exact Search: It return 4 record having same title "Highly Efficient Exploration of Large Design Spaces: Fractionated Satellites as an Example of Adaptable Systems" but with different author name. Paper may have multiple authors

Query time:" No-Index method" requires the 2259 millisecond to fetch record from dataset. Proposed system provides an "Indexed-Based" which searches a query keyword "Satellites" as prefix of word. "Satel" prefix in Prefix table. In proposed system search is totally depend upon overall search i.e. prefix search. It gives same result in 25 Millisecond when search as prefix "Satel".

Max Heap memory used:" No-Index method" fetches maximum number of record from dataset. So it uses the max heap memory are 64 MB. Proposed system provides an "Indexed-Based" which searches a query keyword "Satellites" as prefix of word. "Satel" prefix in Prefix table. It uses 40 MB. It uses minimum heap memory as compared to" No-Index method" .

Number of record:" No-Index method" fetches maximum number of record from dataset. Here it fetches 4 records. Proposed system provides an "Indexed-Based" provides one records.

i) Fuzzy Search

In fuzzy search consist of "No-Indexed" method which gives result of fuzzy search for query keyword "Satellites". "No-Indexed" method needs prefix "Satel" used for query keyword "Satellites".

Query time : " No-Index method" requires the 1421 millisecond to fetch record from dataset. Proposed system provides an "Indexed-Based" using "UDF" searches a query keyword "Satellites" as prefix of word. "Satel" prefix in Prefix table. In proposed system search is totally depend upon overall search i.e. prefix search. It gives same result in 25 Millisecond when search as prefix "Satel". Second proposed method "gram based" takes two inputs. First is query keyword "Satellites" and second is substring "Satel". If substring is valid only then it fetch record from dataset. It requires 2953 milliseconds.

Max Heap memory used: " No-Index method" fetches maximum number of record from dataset. So it uses the max heap memory are 42 MB. Proposed system provides an "Indexed-Based" consist "UDF" and "gram based" method. "UDF" which searches a query keyword "Satellites" as prefix of word "Satel" prefix in Prefix table. It uses 25 MB. "Gram based "method uses 41 MB.

Number of record: " No-Index method" fetches maximum number of record from dataset. Here it fetches 4 records. Proposed system provides an "Indexed-Based" methods such as "UDF" and "gram based " provides one record.

2. Multi-keyword

1. Exact search

In multi keyword consists of exact search and fuzzy search for multi-keyword. So another keyword used with "Satellites" is "quasi-birth-and-death".

Query time: It requires 4444 milliseconds. As compared to single keyword, multi-keyword exact search require more query time.

Max Heap memory used: Multi-keyword exact search uses 49 MB.

Number of record: It fetches 6 records. 4 record for "Satellites" and 2 results for "quasi-birth-and-death".

2. Fuzzy search

Multi keyword fuzzy search method search the records contain with "Sael" and "quasi-bith" keyword.

Query time: It requires 50 milliseconds. As compared to single keyword, multi-keyword fuzzy search require more query time.

Max Heap memory used: Multi-keyword exact search uses 35 MB.

Number of record: It fetches 2 records. One record for "Satellites" and one record for "quasi-birth-and-death".

3.First-n- query

Exact Search : In first n query consists of exact search and fuzzy search for multi-keyword. So another keyword used with "Satellites" is "quasi-birth-and-death". First n query supports multi keyword exact search method search the records contain with "Satellites" and "quasi-birth-and-death" keyword. It fetches the records which is less than or equal to that limit i.e. 3. It shows only top 3 results. Its required query is time 4678 millisecond. Heap memory uses up to 39 MB.

Fuzzy Search : Top n query supports multi keyword fuzzy search method that searches the records contain with "atel" and "quasi-birh" keyword. It gives the 2 records which contain "Satel" and "quasi-birth" keyword. It gives result with respect to similar match keywords. It fetches result in

37 milliseconds which is far less as compared to exact search. It uses heap memory up to 40 MB.

VI. PERFORMANCE ANALYSIS:

Case Study	Method	Search		Query Time	Memory Used	No. of record
Case Study I	No Indexed Based	Exact		2213	72	5
		Fuzzy		1379	44	5
	Indexed based	Exact		114	43	1
		Fuzzy	gram	1580	42	1
			udf	24	44	1
Case Study II	No Indexed Based	Exact		2259	64	4
		Fuzzy		1421	42	4
	Indexed based	Exact		20	40	1
		Fuzzy	gram	2923	41	1
			udf	25	39	1

Table 3Comparative performance of methods

Overall performance of query time related case study 1 is shown below.

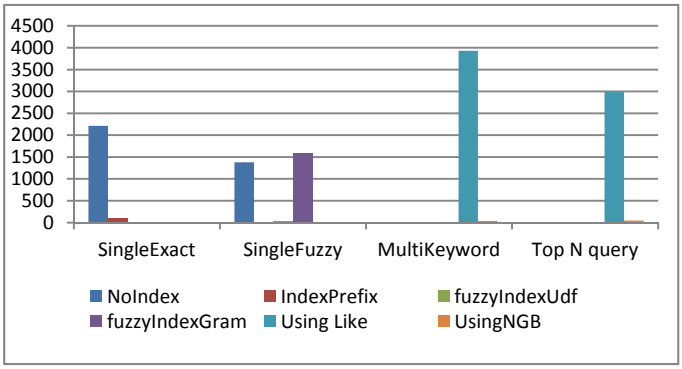

Fig 3 Query Time Analysis for case study 1

From fig 3 it is concluded that query time required to "No-Indexed Based" method is more than our proposed "Indexed Based" method. Overall performance of max heap memory related case study 1 is shown below.

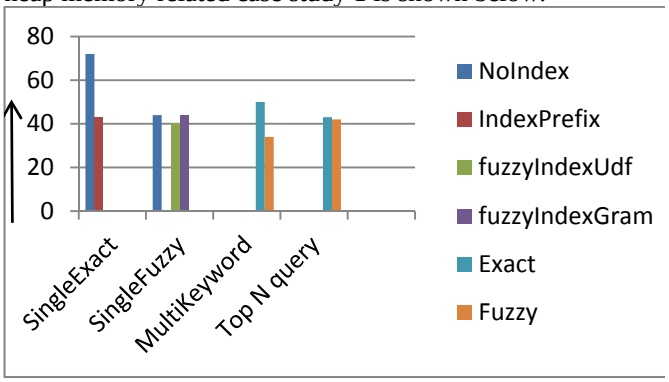

Fig 4 Heap Memory used for case study 1

From fig 4 it is concluded that max heap memory required to "No-Indexed Based" method is more than our proposed "Indexed Based" method. Single exact search in heap memory is approximately same as to the fuzzy search.

Overall performance of number of record related case study 1 is shown below.

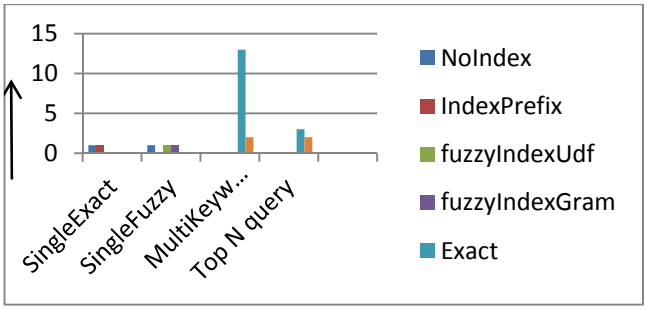

Fig 5 Number record fetches For Case Study 1

From fig 5 it is concluded that number of record fetch from "No-Indexed Based" method is more than our proposed "Indexed Based" method. For case study 2 , the overall performance factor "Query time " related fuzzy and exact search for single and Multi-Keyword are shown in following fig 6.

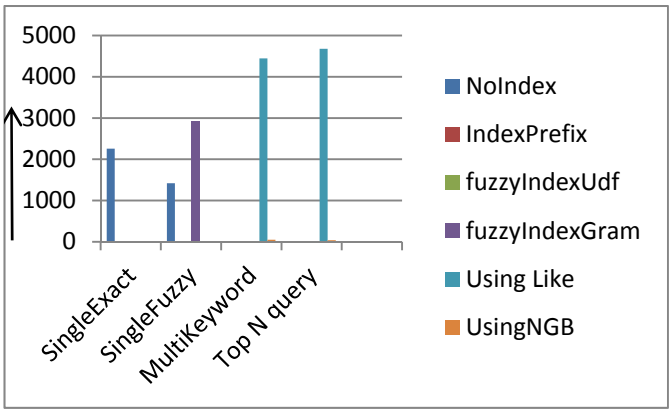

Fig.6 "Query time" of fuzzy and exact search for single and Multi-Keyword

The overall performance factor "Heap Memory Used" related fuzzy and exact search for single and Multi-Keyword are shown in following fig 7.

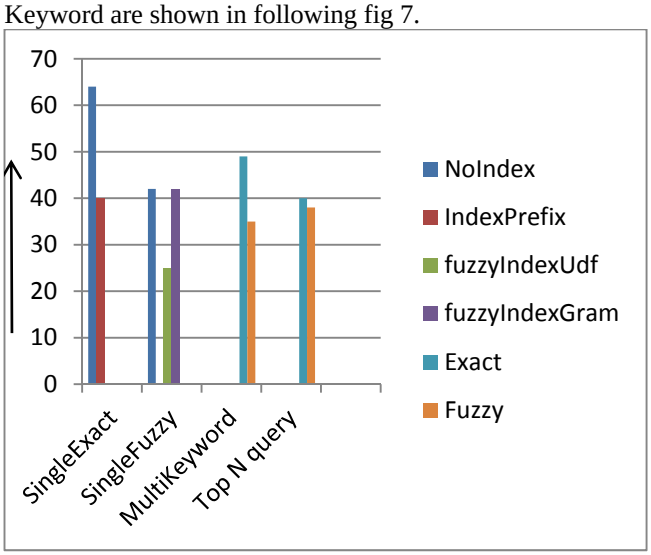

Fig 7 "Heap Memory Used" related fuzzy and exact search for single and Multi-Keyword.

The overall performance factor "Number of record " related fuzzy and exact search for single and Multi-Keyword are shown in following fig 8.

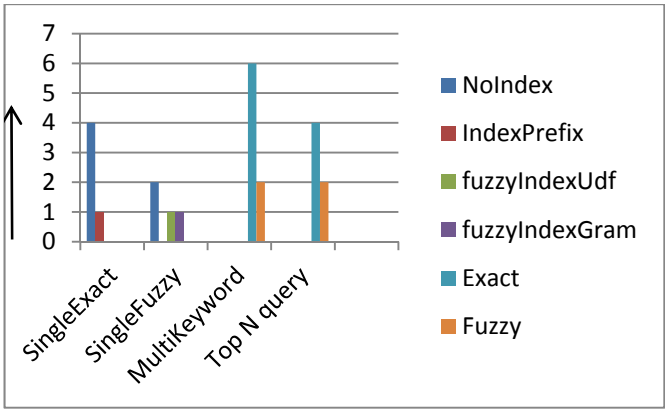

Fig 8 "Number of record" related fuzzy and exact search for single and Multi-Keyword.

VII CONCLUSION AND FUTURE SCOPE:

After evaluating overall the performance of the "Indexed based" method with different set of inputs, related outputs are studied. Then it starts the performance analysis of the "No-Indexed Based" methodology with the "Indexed Based" strategy. Through the two scenarios with every scenario being having different input we analyze the Performance of both strategies. The overall observation from the all scenarios proved that the "Indexed Based" strategy outperforms the "Non-Indexed" in every feature. Query time of "Indexed Based" is much more less than "Non-Indexed Based" method. "Number of record" fetched in "Non-Indexed" method is more as compared to "Indexed Based" method but it contain more percentage of false positivity. There are several open problems to support search-as you- type using SQL. One is how to support ranking queries efficiently.

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] J. Jestes, F. Li, Z. Yan, and K. Yi, "Probabilistic String Similarity Joins," Proc. Int'l Conf. Management of Data (SIGMOD '10), pp. 327- 338, 2010.

[6] R.B. Miller, "Response Time in Man-Computer Conversational Transactions," Proc. AFIPS '68: Fall Joint Computer Conf., Part I, pp. 267-277, 1968.

[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] 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.