© 2015 IJSRSET | Volume 1 | Issue 2 | Print ISSN : 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Database Traversal to Support Search Enhance Technique using SQL

Sivakumar K, Sriram U, Yasar Arafath S, Bhanu Priya
Computer Science and Engineering, Dhanalakshmi College of Engineering, Chennai, Tamilnadu, India

ABSTRACT

A search-as-you-type system computes answers on-the-fly as a user sorts during a keyword question character by character. We have a tendency to support search-as-you-type on information residing during relative software. We have a tendency to target the way to support this kind of search mistreatment the native information language, SQL. A main challenge is the way to leverage existing information functionalities to satisfy the high performance demand to realize associate interactive speed. We have a tendency to use auxiliary indexes hold on as tables to extend search performance. We have a tendency to gift solutions for each single keyword queries and multi keyword queries, and develop novel techniques for fuzzy search mistreatment SQL by permitting mismatches between question keywords and answers. We gift techniques to answer first-N queries and discuss the way to support updates expeditiously. Experiments on massive, real information sets show that our techniques modify software systems on a trade goods pc to support search-as-you-type on tables with a lot of records.

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

I. INTRODUCTION

Many info systems today improve user search experiences by providing instant feedback as users formulate search queries. Most search engines and online search forms support auto completion that shows recommended queries or perhaps answers "on the fly" as a user sorts during a keyword question character by character. As an example, think about the net search interface at Netflix, 1 that permits a user to go looking for motion-picture show info. If a user sorts during a partial question "mad," the system shows movies with a title matching this keyword as a prefix, like "Madagascar" and "Mad Men: Season one." The instant feedback helps the user not solely in formulating the question, however additionally in understanding the underlying knowledge. This sort of search is usually known as search-as-you-type or type ahead search. Since several search systems store their info during a backend relative software system, a matter arises naturally: a way to support search-as-you type on the info residing during a DBMS? Some databases like Oracle and SQL server already support prefix search, and that we might use this feature to try to search-as-you-type. However, not all databases give this feature. For this reason, we have a tendency to study new strategies which will be employed in all databases. One approach is to develop a separate application layer on the information 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 further hardware prices. Another approach is to use information 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 a DBMS. This approach isn't possible for databases that don't give such associate extender interface, like MySQL. Since it must utilize proprietary interfaces provided by information vendors, an answer for one information might not be moveable to others. Additionally, associate extender-based resolution, particularly those enforced in C/C++, might cause serious reliableness and security issues to information engines. During this paper we have a tendency to study a way to support search-as-you-type on software system systems exploitation the native command language (SQL). In alternative words, we would like to use SQL to seek out answers to an exploration question as a user sorts in keywords character by character. Our goal is to utilize the integral question engine of the information system the maximum amount as attainable. During this manner, we are able to scale back the programming efforts to support search-as-you-type. Additionally, the answer developed on one information exploitation normal SQL techniques is moveable to alternative databases that support a similar normal. Similar observation are created by Gravano et al. [17] and Jestes et al. [23] that use SQL to support similarity take part databases.

Existing System

Most search engines and on-line search forms support automobile completion. That shows instructed queries or perhaps answers "on the fly" as user sorts in an exceedingly question char by char. Some information bases like Oracle and SQL server already support prefix search, we may use this feature to try and do search as you sort. But not all the databases support this feature, for this reason we have a tendency to study a way which will be utilized in all databases. In AN existing system, systems don't seem to be specially designed for keyword queries creating is more difficult to support search as you sort are disadvantage. Some vital practicality to support search as you sort needs be part of operations that might be rather valuable to execute by the question engine.

Disadvantages of Existing System

- Very Time Consuming Process.
- More challenging to support search

Proposed System

We develop numerous techniques to handle these challenges. We have a tendency to propose 2 varieties of ways to support Search-As-You sort for Single Keyword Queries. We have a tendency to use SQL to scan a table
And verify every record by vocation a User outlined operate (UDF) or victimization sort predicate. we have a tendency to discuss gram primarily based} ways and a UDF based methodology, because the 2 ways have an occasional performance we have a tendency to propose a brand new neighbourhood generation primarily based methodology. We have a tendency to use comma (,) as symbol to go looking over a quest that consumes less time and helps to retrieve search knowledge additional simply. We have a tendency to extend the technique to support multi keyword queries; we develop a word level progressive methodology to expeditiously answer multi-keyword. Since the results of earlier queries square measure hold on within the info and shared by future queries

Advantages of Proposed System

- Less Time Consuming Process, because of the multi-search concept.
- We may able to mine large amount of data simultaneously.

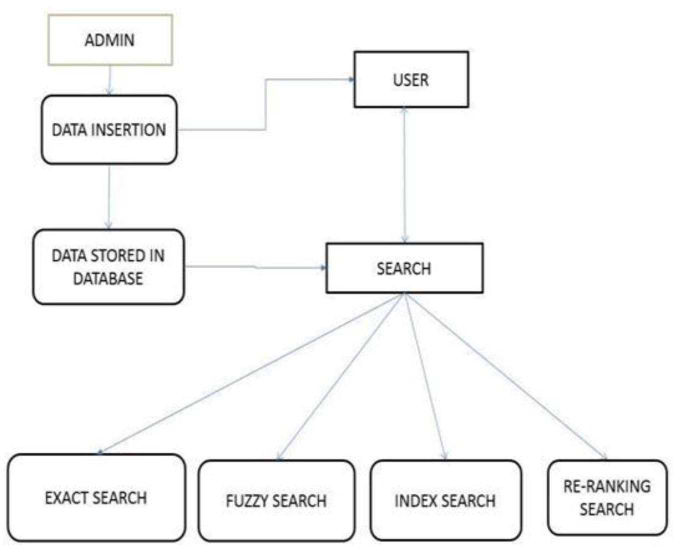


Figure 1: System Architecture

II. METHODS AND MATERIAL

SYSTEM MODULES:

1. Exact Search for Single Keyword
2. Fuzzy Search for Single Keyword.
3. Supporting Multi-keyword Queries.
4. Supporting First-N Query.
5. Supporting amalgamate search.

1. Exact Search for Single Keyword.

No-index-Based technique

In no-index based mostly technique the pointer moves or scans whole the information that consumes much time. There ar 2 ways that to try and do the checking. Job User-Defined functions (UDFs) and mistreatment LIKE predicate.

Index-Based methodology

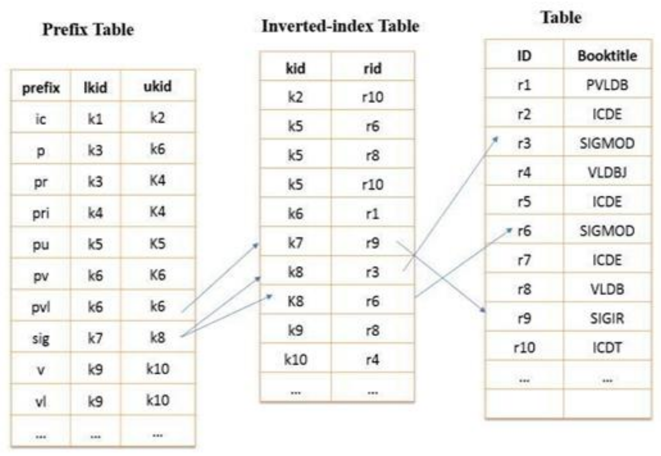
In Index-based methodology, it builds auxiliary tables as index structures to facilitate prefix search. In Index methodology, it invents 2 tables from given info.

- Inverted-Index Table
- Prefix Table

For example, assuming a user types in a partial query "sig" on table dblp (Table 1), we issue the following SQL:

SELECT dblp.* FROM Pdblp, Idblp, dblp WHERE Pdblp.prefix = "sig" AND Pdblp.ukid ≥ Idblp.kid AND Pdblp.lkid ≤ Idblp.kid AND Idblp.rid = dblp.rid

Table 1: Database Table - Result



Prefix Table

prefix	lkid	ukid
ic	k1	k2
p	k3	k6
pr	k3	K4
pri	k4	K4
pu	k5	K5
pv	k6	K6
pvl	k6	k6
sig	k7	k8
v	k9	k10
vl	k9	k10
...	...	...

Inverted-index Table

kid	rid
k2	r10
k5	r6
k5	r8
k5	r10
k6	r1
k7	r9
k8	r3
K8	r6
k9	r8
k10	r4
...	...

Table

ID	Booktitle
r1	PVLDB
r2	ICDE
r3	SIGMOD
r4	VLDBJ
r5	ICDE
r6	SIGMOD
r7	ICDE
r8	VLDB
r9	SIGIR
r10	ICDT
...	...

2. Fuzzy Search for Single Keyword.

No-Index-Based Methodology

In no-index primarily based methodology the pointer moves or scans whole the info that consumes abundantly time. By occupation User-Defined perform (UDFs) and it doesn't support LIKE predicate. We have a tendency to use a UDF pED (w,s) that takes a keyword w and a string s as 2 parameters and returns a tokenish edit distance between w and therefore the prefixes.2)

Index-Based method

It also uses two tables

- Inverted-Index Table
- Prefix Table

- **Using UDF**

It uses UDF to find its similar prefixes from the prefix table

SELECT T .* FROM PT , IT , T WHERE pEDTH(w,PT,prefix,T) AND PT .ukid ≥ IT .kid AND PT .lkid ≤ IT .kid AND IT .rid = T.rid.

- **Gram Based Method**

It uses q-gram based method to support approximate search. As this method involve false positive, we have to use UDF to verify the candidates. So that it is inefficient.

- **Neighborhood Generation Based Method**

It provides approximate string search by I-deletion neighborhood value.

D1(pvldb)={vldb,pldb,pvdb,pvlb,pvld}

3**. Supporting Multi-keyword Queries.**

The previous strategies have the subsequent limitations. First, they have to seek out similar prefixes of a keyword from scratch. Second, they'll have to be compelled to decision UDF persistently. during this Section, we have a tendency to propose a personality level progressive methodology to seek out similar prefixes of a keyword as a user varieties character by character. we have a tendency to develop effective index structures victimization auxiliary tables and devise pruning techniques to realize a high speed. We develop novel techniques on a way to use auxiliary tables, inbuilt indexes on key attributes and pruning technique. It provides data format, deletion, match, insertion, substitution.

Computing Answers from Scratch

Using the "INTERSECT" operator: it joins the records for different keywords.

- Using "Full text" indexes(CONTAINS command):it finds the records matching.
- The obove two methods cannot use precomputed results and may lead to low performance.

Word level Incremental Computation

- We can use previously computed results to answer a query
- Exact search: it uses prefix table and inverted index table
- Fuzzy search: here we consider a character by character incremental method, it allows mismatches.

4. Supporting First-N Query

It retrieves First-N elements from the data base based on the limit. It has two types:

- Exact First-N query: For exact search, we can use the "LIMIT N" syntax in databases to return the first-N results.
- Fuzzy First-N query: it is based on the edit distance; it will progressively increase the distance threshold and select the result.

5. Supporting amalgamate search

In this search, we use comma (,) as identifier to engage multiple search for retrieving more than a data simultaneously. Hence, this amalgamate search reduces the number of SQL commands used for mining data unlike other normal searches.

```
string[] split = words.Split(new Char[] { ',',' ' }, StringSplitOptions.RemoveEmptyEntries);

SqlDataAdapter adp = new SqlDataAdapter ("select filename from imge_upload where imgname like '%" + split[0] + "%' or imgname like '%" + split[1] + "%' or imgname like '%" + split[2] + "%'", con);
```

III. RESULTS AND DISCUSSION

From the above analysis, we derive to an conclusion that Exact search and fuzzy search uses more number of SQL commands and consumes lots of time to mine the required data. Whereas on the other hand, amalgamate search helps us to reduce the effort taken for using more number of SQL commands and time consumed for retrieving data one by one. Instead this search helps to retrieve data simultaneously.

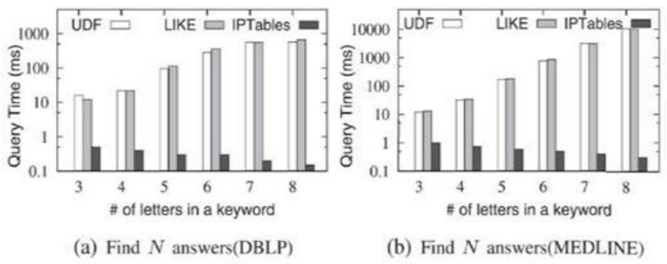


Figure 2: Exact-search performance for answering single-keyword queries (varying keyword length) [1]

IV. CONCLUSION

We studied the problem of using SQL to support search-as-you- type in databases. 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 reduced the time consumed for mining large amount of data and thus simultaneous search is being. Here, we use text word as keyword for mining data from large database. The processing time take for mining data one after the other by normal searches are overcome by this search method. It reduces the processing time taken for entering SQL queries every time and encourages simultaneous retrieval of data.

V. ACKNOWLEDGMENT

The authors would like to thank Asst Prof. Bhanu Priya of Dhanalakshmicollege of engineering at Chennai for her support and constructive comments on preliminary versions of this paper.

VI. 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 Keywordth-Based Search over Relational Data Bases," Proc. 18 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 DataBases Using Banks,” Proc. 18th Int’l Conf. Data Eng. (ICDE ’02), p p. 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.