A Searching Strategy to Adopt Multi-Join Queries Based on Top-K Query Model

[1]M.Naveena, [2]S.Sangeetha,
[1]M.E-CSE, [2]AP-CSE
V.S.B. Engineering College,
Karur, Tamilnadu, India.
[1]naveenaskrn@gmail.com, [2]sangi.vs@gmail.com

***Abstract*—Asearch-as-you-type system determines answerson-the-fly as a user types in a keyword query, character by character. There arises a higher need to know the support search-as-you-type on data residing in a relational DBMS. The existing work on keyword query focuses on to support type of search using the native database SQL. The leverage existing database functionalities is to meet high performance requirement to achieve an interactive speed. It uses auxiliary indexes that are stored as tables to increase search performance. But the main drawbacks in existing work were that it handle search as you type for databases for single table at the same time multiple tables were not taken into consideration. The Proposed work presents a Fuzzy Multi-Join technique to support multiple tables for search as-you-type in relational databases. Further the proposal presents a Top-K Query Search model to support ranking queries for search as-you-type in relational databases Top-k join queries are generated in relational query processors.**

***Keywords*—**Search-as-you-type, Databases, SQL, Fuzzy Search.

I. INTRODUCTION

A 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.

Rank-aware query processing has become a vital need for many applications. In the context of the Web, the main applications include building meta-search engines, combining ranking functions and selecting documents based on multiple criteria. Efficient rank aggregation is the key to a useful search engine. In the context of multimedia and digital libraries, an important type of query is similarity matching. Users often specify multiple features to evaluate the similarity between the query media and the stored media.

Most of these applications have queries that involve joining multiple inputs, where users are usually interested in the top-k join results based on some score function. Since most of these applications are built on top of a commercial relational database system, our goal is to support top-k join queries in relational query processors. The answer to a top-k join query is an ordered set of join results according to some provided function that combines the orders on each input.

More precisely, consider a set of relations R1 to Rm. Each tuple in Ri is associated with some score that gives it a rank within Ri. The top-k join query joins R1 to Rm and produces the results ranked on a total score. The total score is computed according to some function, f, which combines individual scores. Note that the score attached with each relation can be the value of one attribute or a value computed using a predicate on a subset of its attributes.

To increase the search performance and to achieve an interactive speed it uses auxiliary indexes stored as tables. Support both single-keyword and multi-keyword queries using the database language SQL. A higher need to know the support of search-as-you-type on data residing in a relational DBMS it uses ranking and multi-join technique.

II. RELATED WORK

ESTER further supports are designed by H. Bast et al. (2007) [1] a natural blend of such semantic queries with ordinary full-text queries. Moreover, the prefix search operation allows for a fully interactive and proactive user interface, which after every keystroke suggests to the user possible semantic interpretations of his or her query, and speculatively executes the most likely of these interpretations.

Imagine a user of a search engine is developed by I. Weber et el. (2006) [2] typing a query. Then with every letter being typed, we would like an instant display of completions of the last query word which would lead to good hits. At the same time, the best hits for any of these completions should be displayed. Known indexing data structures that apply to this problem either incur large

ISSN (Online) 2347-3207

processing times for a substantial class of queries, or they use a lot of space. We present a new indexing data structure that uses no more space than a state-of-the-art compressed inverted index, but that yields an order of magnitude faster query processing times. Even on the large TREC Terabyte collection, which comprises over 25 million documents, we achieve, on a single machine and with the index on disk, average response times of one tenth of a second. We have built a full-edged, interactive search engine that realizes the proposed auto completion feature combined with support for proximity search, semi-structured (XML) text, sub word and phrase completion, and semantic tags.

Complete Search, an interactive search engine is developed by H. Bast (2007) [3] that offers the user a variety of complex features, which at first glance have little in common, yet are all provided via one and the same highly optimized core mechanism. This mechanism answers queries for what we call context-sensitive prefix search and completion: given a set of documents and a word range, compute all words from that range which are contained in one of the given documents, as well as those of the given documents which contain a word from the given range.

We propose a simple algorithm based on novel is designed by Y. Ma et al. (2007) [4], indexing and optimization strategies that solve this problem without relying on approximation methods or extensive parameter tuning. We show the approach efficiently handles a variety of datasets across a wide setting of similarity thresholds, with large speedups over previous state-of-the-art approaches.

A system which enables keyword-based search was developed by G. Bhalotia et al. (2002) [5] on relational databases, together with data and schema browsing. BANKS enables users to extract information in a simple manner without any knowledge of the schema or any need for writing complex queries. A user can get information by typing a few keywords, following hyperlinks, and interacting with controls on the displayed results.

III. A SEARCHING STRATEGY TO ADOPT MULTI-JOIN QUERIES BASED ON TOP-K QUERY MODEL

In top-k selection queries, the goal is to apply a scoring function on multiple attributes of the same relation to select tuples ranked on their combined score.

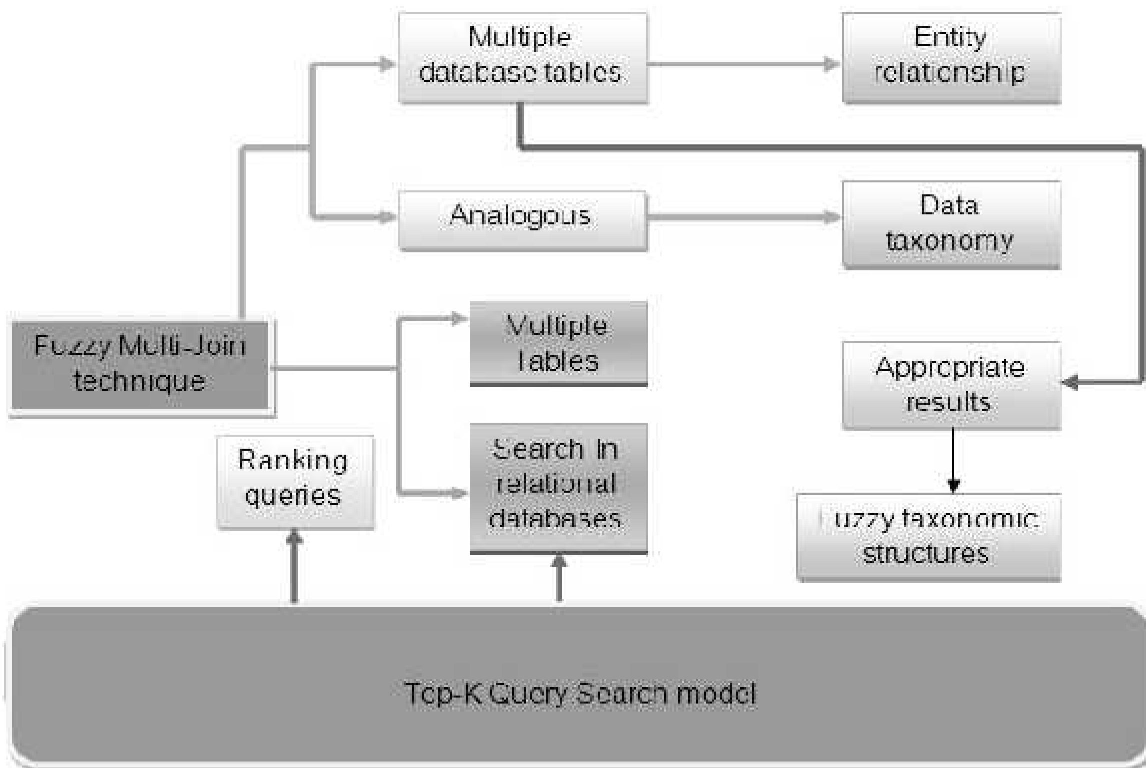

Fig. 1 Architecture Diagram of A Searching Strategy to Adopt Multi-Join Queries Based on Top-K Query Model

The problem is tackled in different contexts. In middleware environments, Fagin and Fagin et al. introduce the first efficient set of algorithms to answer ranking queries. Database objects with m attributes are viewed as m separate lists, each supports sorted and, possibly, random access to object scores. The TA algorithm assumes the availability of random access to object scores in any list besides the sorted access to each list.

ISSN (Online) 2347-3207

- Search as you type Multiple Tables
- Fuzzy Multi-join Search
- Top K-Query Ranking
- Exact Search for Single Keywords
- Fuzzy Search for Single Keywords
- Multi-Keyword Search Updates

A. Search as you type multiple tables

Search-as-you-Type multiple tables allow to add dynamic real-time search to requested queries. Dynamically present suggestions and auto-complete queries, before user is even done typing. Use Search-as-you-Type on any text input field; integrate it with multiple databases as server interface.

SQL to find answers to a search query as user types in keywords character by character in multiple tables. Utilize built-in query engine of databases system as much as possible. Reduce programming efforts to support search-as-you-type, solution developed on one database using standard SQL techniques is portable to multiple tables as well.

B. Fuzzy multi-join search

Search as you type queries look for results from multiple tables across the databases. Fuzzy multi-join search for multiple tables uses hierarchical taxonomy (concept hierarchy) of search data to generate different search criteria at different levels in the taxonomy for databases containing multiple tables.

Fuzzy multi-join search enable to discover crisp search as you type results fuzzy generalized search rules. Search queries are mapped to form similarity queries. Similarity queries are made into multi-join search in the databases.

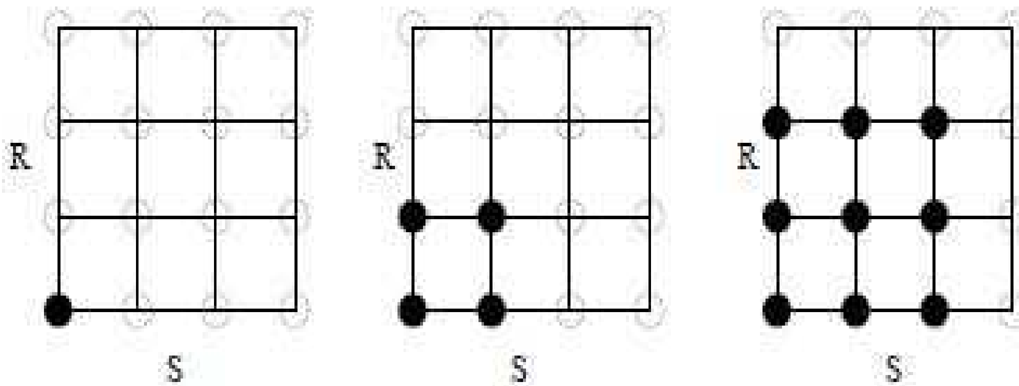

Fig. 2 Three steps in Ripple Join

Ripple join is a family of join algorithms introduced in the context of online processing of aggregation queries in a relational DBMS. Traditional join algorithms are designed to minimize the time till completion estimate of the query result is available. Ripple joins can be viewed as a generalization of nested-loops join and hash join.

In the simplest version of a two-table ripple join, one previously-unseen random tuple is retrieved from each table (e.g., R and S) at each sampling step. These new tuples are joined with the previously-seen tuples and with each other. Thus the Cartesian product R _ S is swept out as depicted in Fig 2.

The square version of ripple join draws samples from Rand S at the same rate. However, in order to provide the shortest possible condense intervals; it is often necessary to sample one relation at a higher rate. This requirement leads to the general rectangular version of the ripple join where more samples are drawn from one relation than from the other.

Variants of ripple join are:

1. Block Ripple Join, where the sample units are blocks of tuples of size b (In classic ripple join, b = 1)
2. Hash Ripple Join, where all the sampled tuples are kept in hash tables in memory. In this case, calculating the join condition of a new sampled tuple with previously sampled tuples is very fast (saving I/O).

C Ttop k-query ranking

Multi-join queries search in multiple tables are improved by ranking top k queries in search as you type multiple databases. Introduced rank-join algorithm makes use of individual orders of its inputs to produce join results ordered on a user-specified scoring function. Rank the join results progressively during the join operation. The operators are non-blocking integrated into pipelined

execution plans. Arrive optimization heuristics to integrate new join operators in practical query.

Ripple joins are designed to minimize time acceptably precise estimate of query result is available. Ripple joins viewed as generalization of nested-loops join and hash join. In a two-table ripple join one previously-unseen random tuple is retrieved from each table (e.g., R and S) at each sampling step, new tuples are joined with previously seen tuples and with each other Cartesian product RS is swept out.

More precisely, consider a set of relations R1 to Rm. Each tuple in Ri is associated with some score that gives it a rank within Ri. The top-k join query joins R1 to Rm and produces the results ranked on a total score. The total score is computed according to some function, f, that combines individual scores. Note that the score attached with each relation can be the value of one attribute or a value computed using a predicate on a subset of its attributes.

A possible SQL-like notation for expressing a top-k join query is as follows:

SELECT *FROM R1,R2,….,Rm WHERE join condition(R1,R2,….,Rm)ORDERBY f(R1:score,R2:score, …..,Rm:score)STOP AFTER k;

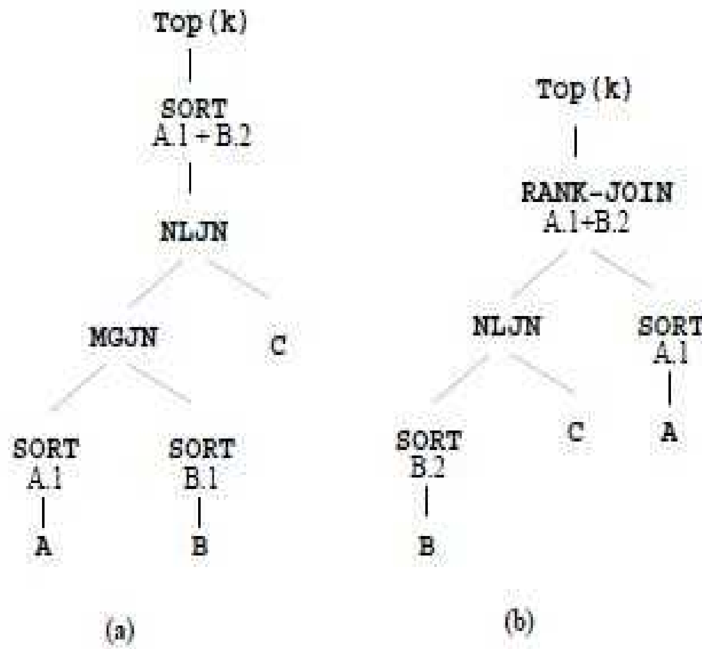

Fig.3 Alternative plans for Query Q1

Propose a new rank-join algorithm, with the desired properties, along with its correctness proof. Implement the proposed algorithm in practical pipelined rank-join operators based on ripple join, with better capabilities of preserving orders of their inputs. The new operators can be integrated in query plans as ordinary join operators and hence give the optimizer the chance to produce better execution plans. Fig 3(b) gives an example execution plan for Q1, using the proposed rank-join operator (RANK-JOIN). The plan avoids the unnecessary sort of the join results by utilizing the base table access plans that preserve interesting orders. Moreover, the plan produces the top-k results incrementally.

Propose a novel score-guided join strategy that minimizes the range of the Cartesian space that needs to be evaluated to produce the top-k ranked join results. We introduce an adaptive join strategy for joining ranked inputs from external sources, an important characteristic of the applications that use ranking.

Evaluate our proposed join operators and compare them with other approaches to join ranked inputs. The experiments validate approach and show a superior performance of algorithm over other approaches.

The algorithm takes m ranked inputs, a join condition, a monotone combining ranking function f and the number of desired ranked join results k. The algorithm reports the top k ranked join results in descending order of their combined score. **The rank-join algorithm works as follows:**

Retrieve objects from the input relations in a descending order of their individual scores. For each new retrieved tuple:

1. Generate new valid join combinations with all tuples seen so far from other relations, using some join strategy.

2. For each resulting join combination, J, compute the score J:score as

f(O1:score;O2:score; : : : ;Om:score), where Oi:score is the score of the object from the ith input in this join combination.

3. Let the object O(di) i be the last object seen from input i, where di is number of objects retrieved from that input, O(1) i be the _rst object retrieved from input i, and T be the maximum of the following m values:

   f(O(d1)1 :score;O(1)2 :score; : : : ;O(1) m :score),
   f(O(1) 1 :score;O(d2)2 :score; : : : ;O(1) m :score), : : :,
   f(O(1) 1 :score;O(1) 2 :score; : : : ;O(dm) m :score).

4. Let Lk be a list of the k join results with the maximum combined score seen so far and let scorek be the lowest score in Lk, halt when scorek>_ T.

D. Exact search for single keywords

Exact search for single keyword comprises of two methods,

1. No-Index method
2. Index method

1. No-Index method

No-Index Method support search-as-you-type to issue an SQL query that scans each record and verifies whether record is an answer to the query.

Using the LIKE predicate databases provide a LIKE predicate to allow users to perform string matching, use LIKE predicate to check whether a record contains the query, keyword introduce false positives remove these false positives by calling UDFs.

Two no-index methods need no additional space, but they are not scalable since they need to scan all records in the table.

2. Index method

Index-Based Methods build auxiliary tables as index structures to facilitate prefix search, develop a new method used in all databases, performs prefix search more efficiently.

Inverted-index table

Given a table T, assign unique ids to the keywords in table T, following their alphabetical order. Create an inverted-index table IT with records in the form (kid; rid) 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.

(a) Keywords

kid	keyword
k_1	icde
k_2	icdt
k_3	preserving
k_4	privacy
k_5	publishing
k_6	pvldb
k_7	sigir
k_8	sigmod
k_9	vldb
k_{10}	vldbj
...	...

(b) Inverted-index Table

kid	*rid*
k_2	r_{10}
k_5	r_6
k_5	r_8
k_5	r_{10}
k_6	r_1
k_7	r_9
k_8	r_3
k_8	r_6
k_9	r_8
k_{10}	r_4
...	...

(c) Prefix Table

prefix	*lkid*	*ukid*
ic	k_1	k_2
p	k_3	k_6
pr	k_3	k_4
pri	k_4	k_4
pu	k_5	k_5
pv	k_6	k_6
pvl	k_6	k_6
sig	k_7	k_8
v	k_9	k_{10}
vl	k_9	k_{10}
...	...	...

Table.1 Inverted-index Table and Prefix Table

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 (p; lkid; ukid) 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.

Use the following SQL to answer the prefix-searchquery w:

SELECT T* FROM PT; IT; T WHERE
PT.prefix = "w" AND PT .ukid >= IT .kid AND PT.lkid>=IT.kid AND IT .rid =T.rid.

Thus, given a prefix keyword w, use the prefix table to find the range of keywords with the prefix.

E. Fuzzy search for single keywords

In fuzzy search for single keyword comprises of two methods,

1. No-Index method
2. Index method

1. No-Index method

In fuzzy search for single keyword UDFs support fuzzy search. Improve performance by doing early termination in dynamic-programming computation using edit-distance threshold, devise a new UDF. If there is a keyword in string having prefixes with an edit distance within returns true. Issue an SQL query that scans each record and calls UDF to verify the record.

2. Index method

Use the inverted-index table and prefix table to support fuzzy search-as-you-type. Given a partial keyword compute its answers in two steps .First compute its similar prefixes from the prefix table get the keyword ranges of these similar prefixes, and then compute answers based on these ranges using the inverted-index table.

Gram-based Method

There are many q-gram-based methods to support approximate string search. Given a string s, its q-grams are its substrings with length q.

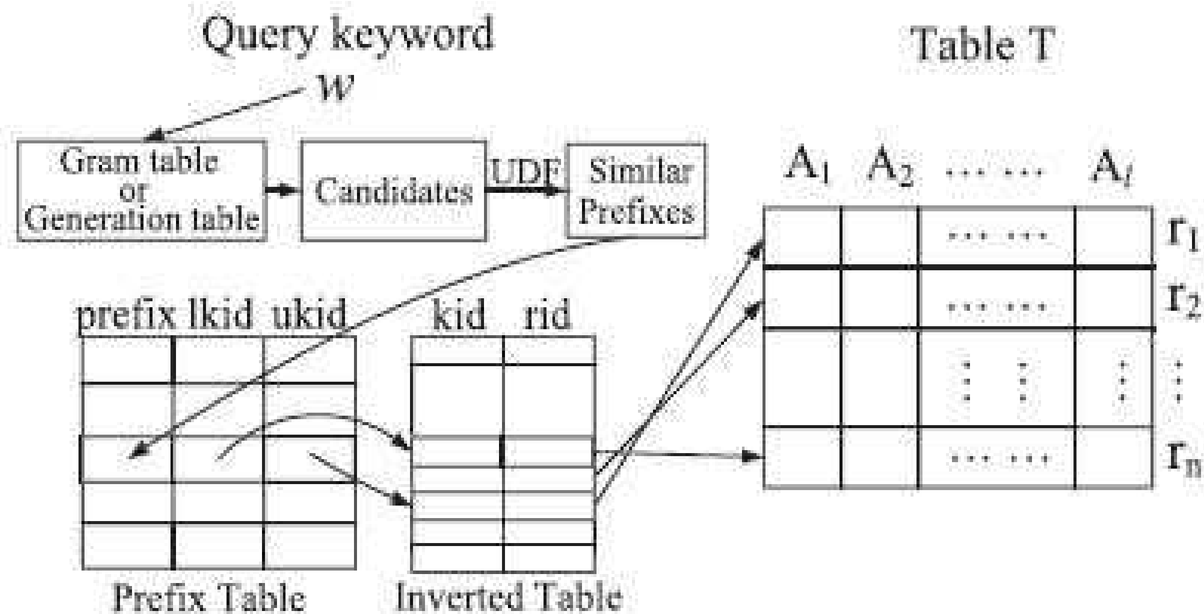

Fig.4 Use the q-gram table and the neighborhood generation table to Support fuzzy search.

To find similar prefixes of a query keyword, besides use the inverted-index table and the prefix table, also in need to create a q-gram table with records.

F. Multi-keyword search updates

Multi-keyword Search updates is given a multi-keyword query Q with m keywords, using the “INTERSECT” Operator first compute records for each keyword and then use INTERSECT operator to join these records for different keywords to compute answers. Using Full-text Indexes first use full-text indexes to find records matching the first complete keywords and then use proposed methods to find records matching the last prefix keyword. Two methods cannot use pre-computed results lead to low performance.

Word-Level Incremental Computation use previously computed results to incrementally answer a query. Assuming a user has typed in a query with keywords create a temporary table to cache the record ids of query. If the user types in a new keyword and submits a new query with keywords use temporary table to incrementally answer the new query. Exact search focus on the method that uses the prefix table and inverted-index table. Fuzzy search consider character level incremental method. Fuzzy search consider character level incremental method, the user arbitrarily modifies the query, can easily extend this method to answer new query.

IV. PERFORMANCE RESULTS AND DISCUSSION

We use a simple ranking query that joins four tables on the non-key attribute JC and retrieves the join results ordered on a simple function. The function combines individual scores which in this case a weighted sum of the scores (wi is the weight associated with input i). Only the top k results are retrieved by the query.

Performance measurement metrics are given below,

- ✓ Number of Tables
- ✓ Number of letters in keyword
- ✓ Query Time
- ✓ Number of answers
- ✓ Number of records

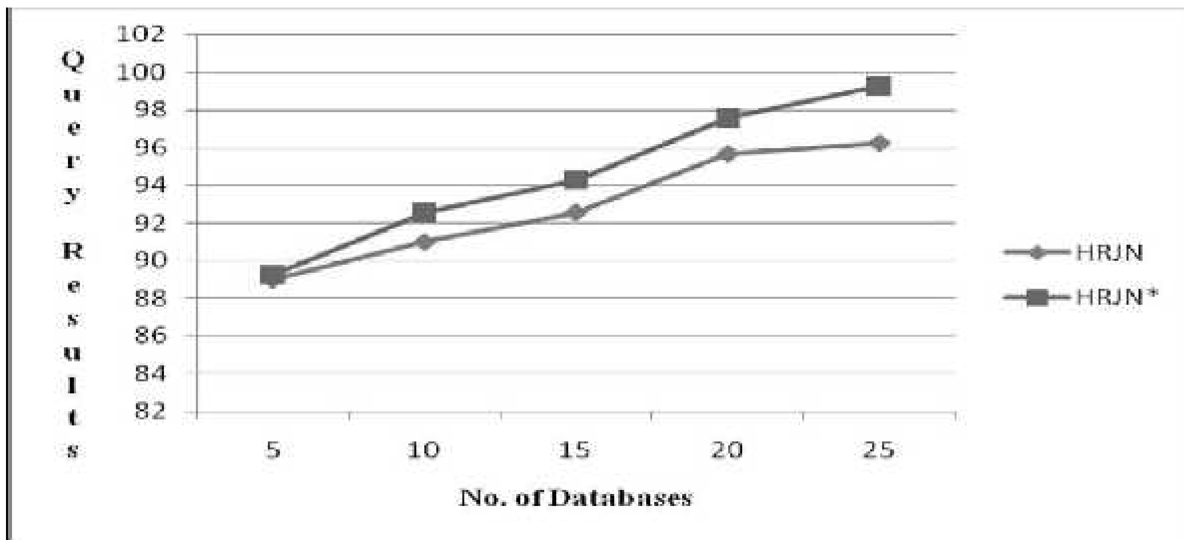

Fig. 5 A Searching Strategy to Adopt Multi-Join Queries Based on Top-K Query Model of Number of databases and Query time.

Fig. 5 shows compares the total time to report 50 ranked results, while compare the number of accessed disk pages and the extra space overhead, respectively.

For all selectivity values, HRJN shows the best performance. J has a better performance than HRJN for high selectivity values while HRJN performs better for low selectivity values.

The reason is that HRJN* combines the advantages of J* and HRJN. While HRJN* uses a score-guided strategy to navigate in the Cartesian space for a faster termination (similar to J*), it also uses the power of producing fast join results by using the symmetric hash join technique (similar to HRJN).

The CPU complexity of J* increases significantly increases. On the other hand, J* and HRJN* show better performance in terms of the number of accessed pages compare to HRJN (Fig 6), because of the score guided strategy they are using. HRJN_ is the most scalable in terms of the space overhead.

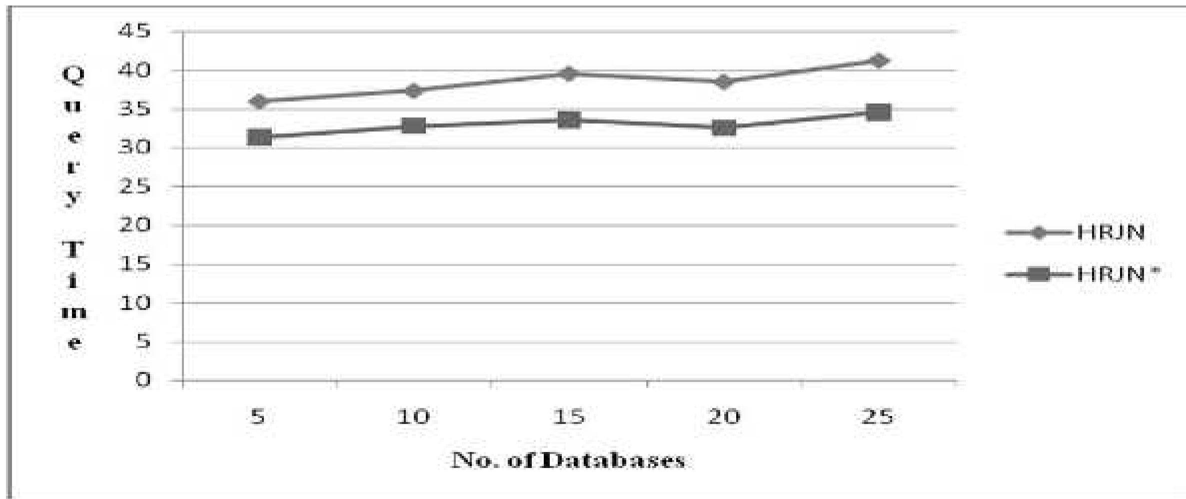

Fig. 6 A Searching Strategy to Adopt Multi-Join Queries Based on Top-K Query Model of Number of databases and Query Results.

The reason is that HRJN* combines the advantages of J* and HRJN. While HRJN* uses a score-guided strategy to navigate in the Cartesian space for a faster termination (similar to J*), it also uses the power of producing fast join results by using the symmetric hash join technique (similar to HRJN).

V. CONCLUSION

Top-k join queries in practical relational query processors. We introduce a new rank-join algorithm that is independent of the join strategy, along with its correctness proof. The proposed rank-join algorithm makes use of the ranking on the input relations to produce ranked join results on a combined score. The ranking is performed progressively during the join and hence, there is no need for a blocking sort operation after join. We present a physical query operator to implement rank-join based on ripple join; the hash rank join (HRJN).

We propose a new join strategy that is guided by the input score values. We apply the new strategy on the original HRJN algorithm and call the new operator HRJN*. We address exploiting available indexes on the join columns. We propose a general rank-join algorithm that utilizes these indexes for faster termination of the ranking process. We experimentally evaluate the proposed join operators and compare their performance with a recent algorithm to join ranked inputs. We conduct several experiments varying the number of required answers, the join selectivity, and the number of inputs in the pipeline.

REFERENCES

[1] H. Bast, A. Chitea, F.M. Suchanek, and I. Weber, "*ESTER:Efficient Search on Text, Entities, and Relations*," Proc. 30th nn.Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '07), pp. 671-678, 2007.
[2] H. Bast and I. Weber, "*Type Less, Find More: Fast Autocompletion Search with a Succinct Index*," Proc. 29th Ann. Int'l ACMSIGIR Conf. Research and Development in Information Retrieval (SIGIR '06), pp. 364-371, 2006.
[3] H. Bast and I. Weber, "*The Complete Search Engine: Interactive,Efficient, and Towards IR & DB Integration*," Proc. ConfInnovativeData Systems Research (CIDR), pp. 88-95, 2007.
[4] 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.
[5] K. Chakrabarti, S. Chaudhuri, V. Ganti, and D. Xin, "*An EfficientFilter for Approximate Membership Checking*," Proc. ACM SIGMODInt'l Conf. Management of Data (SIGMOD '08), pp. 805- 818, 2008.