Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 2 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A Survey of Techniques for Answering Top-k Queries

By Neethu C V & Rejimol Robinson R R
CT College of Engineering Trivandrum, India

Abstract - Top-k queries are useful in retrieving top-k records from a given set of records depending on the value of a function F on their attributes. Many techniques have been proposed in database literature for answering top-k queries. These are mainly categorized into three: Sorted-list based, layer based and View based. In first category, records are sorted along each dimension and then assigned a rank to each of the records using parallel scanning method. Threshold Algorithm (TA) and Fagin's Algorithm (FA) are the examples of sorted-list based category. Second category is layer based category, in which all the records are organized into layers such as in onion technique and robust indexing technique. Third category includes methods such as PREFER and LPTA (Linear Programming Adaptation of Threshold Algorithm) and processing is based on the materialized views.

Keywords : monotone functions, prefer, linearly optimally ordered set, convex hull.

GJCST-C Classification : H.2.3

A Survey of Techniques for Answering Top-k Queries

Neethu C V [α] & Rejimol Robinson R R [σ]

Abstract - Top-k queries are useful in retrieving top-k records from a given set of records depending on the value of a function F on their attributes. Many techniques have been proposed in database literature for answering top-k queries. These are mainly categorized into three: Sorted-list based, layer based and View based. In first category, records are sorted along each dimension and then assigned a rank to each of the records using parallel scanning method. Threshold Algorithm (TA) and Fagin's Algorithm (FA) are the examples of sorted-list based category. Second category is layer based category, in which all the records are organized into layers such as in onion technique and robust indexing technique. Third category includes methods such as PREFER and LPTA (Linear Programming Adaptation of Threshold Algorithm) and processing is based on the materialized views.

Keywords : monotone functions, prefer, linearly optimally ordered set, convex hull.

I. Introduction

Top-k queries are intended for retrieving top-k records from the database which are subjected to minimization or maximization of the function F on the attributes of the relation. This kind of queries appears frequently in many applications such as college ranking, job ranking etc. Due to the popularity of top-k queries, many techniques have been proposed which are mainly includes sorted-list based, layer based and view based techniques.

a) Sorted-list based

Methods in this category sorts all records along each dimension and then assigned an overall grade to each of the records based on the sorted lists. For example, consider the example of college ranking. A student wants to join a college for doing graduation and he has some preferences based on the attributes like distance to the college, tuition fee and university under which college is working, performance of the college for previous four years etc. He then assigns grades to each of the attributes and sorted lists are created based on this assignment corresponding to each of the attributes. Then a list of colleges has retrieved based on their value for the query function. Here, the query function is a linear function in terms of the attributes of the records. FA and TA [1], [2], [3] are the two techniques included in this category.

Author α : Dept. of Computer Science & Engineering SCT College of Engineering Trivandrum, India.
E-mail : neethusureshbabu@gmail.com

b) Layer Based Category

The algorithms in this category organize all records into consecutive layers, such as Onion [4] and Robust Indexing Techniques [5]. The organization strategy is based on the common property among the records, such as the same convex hull layer in Onion [4]. Any top-k query can be answered by up to k layers of records. The Onion indexing is based on a geometric property of convex hull, which guarantees that the optimal value can always be found at one or more of its vertices.

The Onion indexing makes use of this property to construct convex hulls in layers with outer layers enclosing inner layers geometrically. A data record is indexed by its layer number or equivalently its depth in the layered convex hull. Queries with linear weightings issued at run time are evaluated from the outmost layer inwards. Onion indexing achieves orders of magnitude speedup against sequential linear scan when N is small compared to the cardinality of the set. The Onion technique also enables progressive retrieval, which processes and returns ranked results in a progressive manner. Furthermore, the proposed indexing can be extended into a hierarchical organization of data to accommodate both global and local queries.

Robust indexing [5] method is a kind layered technique for answering ranked queries. The layered indexing methods are less sensitive to the query weights. A key observation is that it may be beneficial to push a tuple as deeply as possible so that it has less chance to be touched in query execution. Motivated by this, a new criterion for sequentially layered indexing had been proposed: for any k, the number of tuples in top k layers is minimal in comparison with all the other layered alternatives. Since any top-k query can be answered by at most k layers, this proposal aims at minimizing the worst case performance on any top-k queries. Hence the proposed index is robust. While Onion and other layered techniques are sensitive to the query weights, this method, even though not optimal in some cases, has the best expected performance. Another appealing advantage of our proposal is that the top-k query processing can be seamlessly integrated into current commercial databases. Both Onion and other layered methods require the advanced query execution algorithms, which are not supported by many database query engines so far.

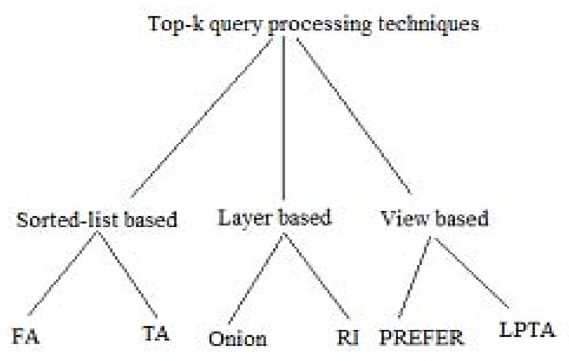
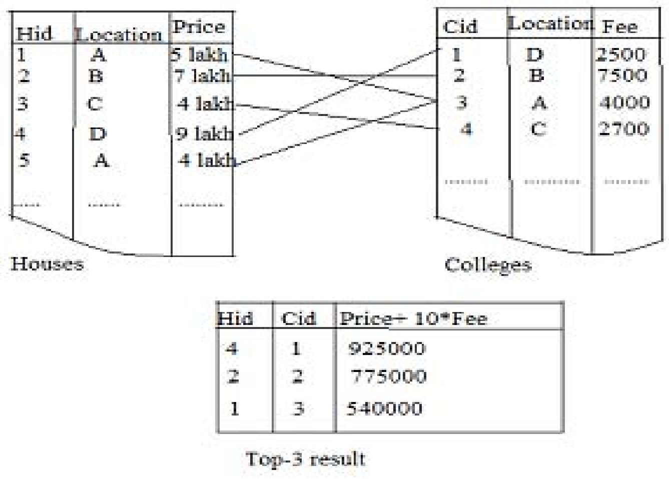


Figure 1 : Classification of Top-k query evaluation techniques

c) View based category

In view based techniques, the materialized views created from the relation can be used to answer top-k queries. PREFER [6] answers preference queries efficiently by using materialized views that have been preprocessed and stored. Queries with different weights will be first mapped to the pre-computed order and then answered by determining the lower bound value on that order. When the query weights are close to the pre-computed weights, the query can be answered extremely fast. Unfortunately, this method is very sensitive to weighting parameters. A reasonable derivation of the query weights (from the pre-computed weights) may severely deteriorate the query performance. PREFER is a layer on top of commercial relational databases and allows the efficient evaluation of multi parametric ranked queries. LPTA [7] is a linear programming adaptation of the classical TA algorithm to solve top-k query problem.



Hid	Location	Price
1	A	5 lakh
2	B	7 lakh
3	C	4 lakh
4	D	9 lakh
5	A	4 lakh
.....	......	....

Houses

Cid	Location	Fee
1	D	2500
2	B	7500
3	A	4000
4	C	2700
....	....	....

Colleges

Hid	Cid	Price+ 10*Fee
4	1	925000
2	2	775000
1	3	540000

Top-3 result

Figure 2 : Example of top-k query processing

II. Taxonomy of Processing Top-K Queries

Due to the high popularity of the top-k queries, various techniques have been proposed for solving such situations. Supporting efficient top-k query processing in database system is relatively recent and active line of research. In the following subsection, all the important techniques included in above explained categoris have been explored in detail.

a) Naïve Algorithm

To determine the top k objects, that is, k objects with the highest overall grades, the naive algorithm must access every object in the database, to find its grade under each attribute.

Step of the Naïve algorithm [1] are given below.

- If (x1, x2,…,xm) are the grades of object R under the m attributes, then compute T(x1,x2,…,xm) overall grade of object R.
- Sort the list of computed values.
- Return top k rows corresponding to the sorted list.

The main disadvantage of the Naïve algorithm is the large processing time when dealing with large databases.

b) Fagin's Algorithm

Fagin introduced an algorithm (''Fagin's Algorithm [1]'', or FA), which often does much better than the naive algorithm. In the case where the orderings in the sorted lists are probabilistically independent, FA finds the top k answers, over a database with N objects with arbitrarily high probability. This algorithm is implemented in Garlic, an experimental IBM middleware system.

- Do sorted access in parallel to each of the m sorted lists Li: Wait until there are at least k ''matches'', that is, wait until there is a set of at least k objects such that each of these objects has been seen in each of the m lists.
- For each object R that has been seen, do random access as needed to each of the lists Li to find the ith field xi of R:
- Compute the grade t(R)= t(x1,x2,….xm) for each object R that has been seen. Let Y be a set containing the k objects that have been seen with the highest grades (ties are broken arbitrarily). The output is then the graded set {(R, t(R)) | R€Y}.

Fagin shows that his algorithm is optimal with high probability in the worst case if the aggregation function is strict (so that, intuitively, we are dealing with a notion of conjunction), and if the orderings in the sorted lists are probabilistically independent. In fact, the access pattern of FA is oblivious to the choice of aggregation function, and so for each fixed database, the middleware cost of FA is exactly the same no matter what the aggregation function is. This is true even for a constant aggregation function; in this case, of course, there is a trivial algorithm that gives us the top k answers (any k objects will do) with O(1) middleware cost. So FA is not optimal in any sense for some monotone aggregation functions t: As a more interesting example, when the aggregation function is max (which is not strict), it is shown in that there is a simple algorithm that makes at most m*k sorted accesses and no random accesses that finds the top k answers. By contrast, the algorithm TA is instance optimal for every monotone aggregation function, under very weak assumptions.

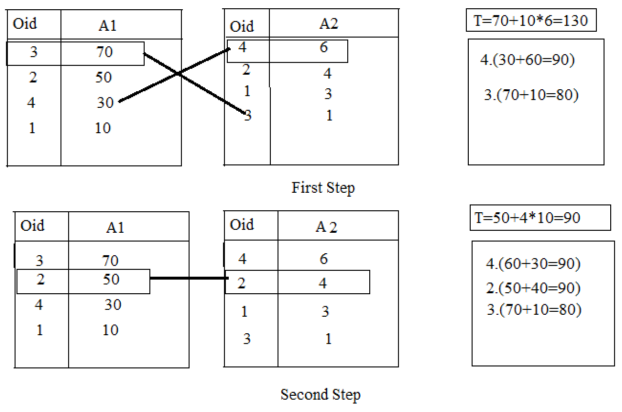
III. Threshold Algorithm

Even in the cases where FA is optimal, this optimality holds only in the worst case, with high probability. This leaves open the possibility that there are some algorithms that have much better middleware cost than FA over certain databases. The algorithm TA, which we now discuss, is such an algorithm.

- Do sorted access in parallel to each of the m sorted lists Li: As an object R is seen under sorted access in some list, do random accesses to the other lists to find the grade xi of object R in every list Li.
- Then compute the grade t(R) =t(x1,x2,…xm) of object R: If this grade is one of the k highest we have seen, then remember object R and its grade t(R).
- For each list Li, let xi be the grade of the last object seen under sorted access. Define the threshold value ψ to be t(x1, x2,….,xm). As soon as at least k objects have been seen whose grade is at least equal to ψ then halt.
- Let Y be a set containing the k objects that have been seen with the highest grades. The output is then the graded set {(R, t(R)) | R€Y}.

The algorithm scans multiple lists, representing different rankings of the same set of objects. An upper bound T is maintained for the overall score of unseen objects. The upper bound is computed by applying the scoring function to the partial scores of the last seen objects in different lists. Notice that the last seen objects in different lists could be different. The upper bound is updated every time a new object appears in one of the lists. The overall score of some seen object is computed by applying the scoring function to object's partial scores, obtained from different lists. To obtain such partial scores, each newly seen object in one of the lists is looked up in all other lists, and its scores are aggregated using the scoring function to obtain the overall score. All objects with total scores that are greater than or equal to T can be reported. The algorithm terminates after returning the kth output. Example 1 given below illustrates the processing of TA.

Example 1 (10): Consider two data sources containing same set of objects. Let A1 and A2 are the attributes in two data sources respectively. The Query function, F is defined as F=A1+10*A2. The working of TA is depicted in the following figure.



In the first step, retrieving the top object from each list, and probing the value of its other attribute value in the other list, results in revealing the exact scores for the top objects. The seen objects are buffered in the order of their scores. A threshold value, T, for the scores of unseen objects is computed by applying F to the last seen scores in both lists, which results in 70+6*10=130. Since both seen objects have scores less than T, no results can be reported. In the second step, T drops to 90, and objects 4 and 2 can be safely reported since its score is above T. The algorithm continues until k objects are reported, or sources are exhausted.

IV. Onion Technique

This technique comes under the layer based category and uses a special indexing structure for answering top-k queries. The Onion indexing is based on a geometric property of convex hull, which guarantees that the optimal value can always be found at one or more of its vertices. The Onion indexing makes use of this property to construct convex hulls in layers with outer layers enclosing inner layers geometrically. A data record is indexed by its layer number or equivalently its depth in the layered convex hull. Queries with linear weightings issued at run time are evaluated from the outmost layer inwards.

Basic idea of the onion technique is that partition the collection of d-dimensional data points into sets that are optimally linearly ordered. This property is used to construct convex hulls in layers with outer layers enclosing inner layers geometrically.

Definition 1: Optimally Linearly Ordered Set: A collection of sets{s1,s2,…,sn}are optimally linearly ordered sets if and only if a d-dimensional vector ā,

$$\exists\, \bar{o} \in s_i \text{ such that}$$

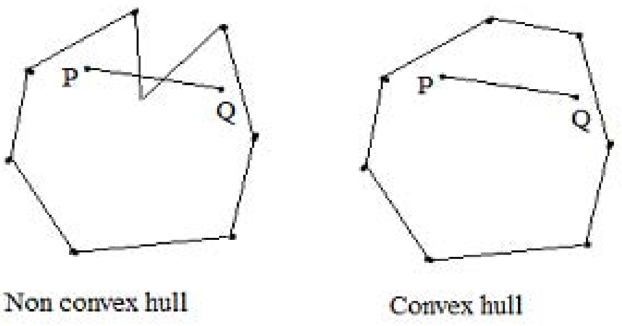
for every $\hat{c} \in s_{i+j}$, $j>0$, $\bar{a}^t\bar{o} > \bar{a}^t\hat{c}$ where $\bar{a}^t\bar{o}$ represents the inner product of two vectors.

Partitioning a set of data points into optimally linearly ordered sets is based on the following theorem.

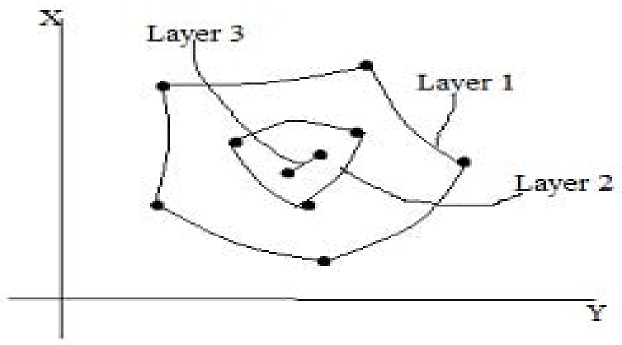
Global Journal of Computer Science and Technology (C) Volume XIII Issue II Version I Year 2013

Theorem 1: Given a set of records R mapped to a d-dimensional space, and a linear maximization criterion, the maximum objective value is achieved at one or more vertices of the convex hull of R.

Definition 2 : A set S is convex if whenever two points P and Q are inside S, then the whole line segment PQ is also in S.



Procedure for index creation:
Step 1: Input a set of records R and iterate the following steps until size(R) become less than zero.
Step 2: Construct convex hull of the data records R.
Step 3: Store the records of hull vertices in set Vi.
Step 4: Assign records in set V to layer k.
Step 5: Set R=R-V and k=k+1.



This indexing structure can be used for query evaluation. Onion indexing achieves orders of magnitude speed up against sequential linear scan when N is small compared to the cardinality of the set. The Onion technique also enables progressive retrieval, which processes and returns ranked results in a progressive manner. Furthermore, the proposed indexing can be extended into a hierarchical organization of data to accommodate both global and local queries.

V. Robust Indexing Structure

This is an another layered indexing structure useful for the evaluation of top-k queries. The idea of multi-layer indexing has been also adopted by to provide robust indexing [5], [10] for top-k queries. Robustness is defined in terms of providing the best possible performance in worst case scenario, which is fully scanning the first k layers to find the top-k answers. The main idea is that if each object Oi is pushed to the deepest possible layer, its retrieval can be avoided if it is unnecessary. This is accomplished by searching for the minimum rank of each object oi in all linear scoring functions. Such rank represents the layer number, denoted l*(Oi), where object Oi is pushed to. For n objects having d scoring predicates, computing the exact layer numbers for all objects has a complexity of O(nd log n), which is an overkill when n or d are large. Approximation is used to reduce the computation cost. An approximate layer number, denoted l (Oi), is computed such that l (Oi) • l*(Oi), which ensures that no false positives are produced in the top-k query answer.

VI. Prefer

This is a view based evaluation of the top-k queries. Recent successful work in non-layered approaches includes the PREFER system [6],[10], where tuples are sorted by a pre-computed linear weighting configuration Users often need to optimize the selection of objects by appropriately weighting the importance of multiple object attributes. Such optimization problems appear often in operations research and applied mathematics as well as everyday life; e.g., a buyer may select a home as a weighted function of a number of attributes like its distance from office, its price, its area, etc.

The queries here use a weight function over a relation's attributes to derive a score for each tuple. Database systems cannot efficiently produce the top results of a preference query because they need to evaluate the weight function over all tuples of the relation. PREFER [6] answers preference queries efficiently by using materialized views that have been preprocessed and stored. Queries with different weights will be first mapped to the pre-computed order and then answered by determining the lower bound value on that order. When the query weights are close to the pre-computed weights, the query can be answered extremely fast. Unfortunately, this method is very sensitive to weighting parameters. A reasonable derivation of the query weights (from the pre-computed weights) may severely deteriorate the query performance. PREFER is a layer on top of commercial relational databases and allows the efficient evaluation of multi parametric ranked queries For example consider a database containing houses available for sale. The properties have attributes such as price, number of bedrooms, age, square feet, etc. For a user, the price of a property and the square feet area may be the most important issues, equally weighted in the final choice of a property, and the property's age may also be an important issue, but of lesser weight. The vast majority of e-commerce systems available for such applications do not help users in answering such queries, as they commonly order according to a single attribute. In these cases, preference queries have significant role and for PRFER system also.

VII. LPTA

Algorithm (LPTA)[7],[10] is another technique included in the view based category.It performs much better than
PREFER.

Problem 1: (Top-K Query Answer Using Views). Given a set U of views, and a query Q, obtain an answer to Q combining all the information conveyed by the views in U.

Consider a single relation R with m numeric attributes X1, X2,….Xm, and n tuples t1, . . . , tn. Let Domi = [lbi, ubi] be the domain of the ith attribute. Refer to table R as a base table. Each tuple t may be viewed as a numeric vector t = (t[1], t[2], . . . , t[m]). Each tuple is associated with a tuple-id (tid).Here consider top-k ranking queries, which can be expressed in SQL-like syntax: SELECT TOP [k] FROM R WHERE RangeQ ORDER BY Score Q. More abstractly, a ranking query may be expressed as a triple Q = (ScoreQ, k, Range Q), where Score Q(t) is a function that assigns a numeric score to any tuple t (the function does not necessarily involve all attributes of the table), and Range Q(t) is a Boolean function that defines a selection condition for the tuples of R in the form of a conjunction of range restrictions on Domi, i 2 {1, . . . ,m}. Each range restriction is of the form li ≤ Xi ≤ ui, I ϵ {1, . . . ,m} and the interval [li, ui] Domi. The semantics requires that the system retrieve the k tuples with the top scores satisfying the selection condition.

LPTA [7] is a linear programming adaptation of the classical TA algorithm to solve Problem 1.1 for the special case when views and queries are of the form V 0 = (Score V 0, n, *) and Q = (Score Q, k, *) respectively. Consider a relation with attributes X1, X2 and X3 as shown in Figure 1.3.2.1. Let views V1 and V2 have scoring functions f1, f2 respectively as shown in Figure 1.3.2.1 and consider a query Q = (f3, k, *).

The algorithm initializes the top-k buffer to empty. It then starts retrieving the tids from the views V1, V2 in a lock-step fashion, in the order of decreasing score (w.r.t. the view's scoring functions). For each tid read, the algorithm retrieves the corresponding tuple by random access on R, computes its score according to the query's scoring function f3, updates the top-k buffer to contain the top-k largest scores (according to the query's scoring function), and checks for the stopping condition as follows: After the dth iteration, let the last tuple read from view V1 be (tidd1, sd1) and from view V2 be (tidd2, sd2). Let the minimum score in the top-k buffer be topkmin. At this stage, the unseen tuples in the view have to satisfy the following inequalities (the domain of each attribute of R of Figure is [1, 100]).

R	tid	X1	X2	X3
	1	82	1	59
	2	53	19	83
	3	29	1	2
	4	80	22	90
	5	28	8	87
	6	12	55	82
	7	16	99	42
	8	18	42	67
	9	42	1	23
	10	23	21	88

	f1 = 2x1+5x2	
V1	tid	score
	7	527
	6	299
	4	270
	8	246
	2	201

	f2 = x2+2x3	
V2	tid	score
	6	219
	4	202
	10	197

Figure 3 : Example of views

The following system of inequalities defines a Convex region in three dimensional space.

$$0 \leq X_1, X_2, X_3 \leq 100 \quad (1)$$
$$2X_1 + 5X_2 \leq s_d^1 \quad (2)$$
$$X2 + 2X3 \leq s_d^2 \quad (3)$$

This system of inequalities defines a convex region in three dimensional space. Let unseenmax be the solution to the linear program where we maximize the function f3 = 3X1 + 10X2 + 5X3 subject to these inequalities. It is easy to see that unseenmax represents the maximum possible score (with respect to the ranking query's scoring function) of any tuple not yet visited in the views. The algorithm terminates when the top-k buffer is full and unseenmax ≤ topkmin. Considering the example of given figure, the algorithm will proceed as follows;

First retrieve tid and conduct a random access to R to retrieve the full tuple and tid 6 from V2 accessing R again. The top-2 buffer contains the following pairs (tiddi, sdi) {(7, 1248), (6, 996)}. The solution to the linear program with s1q= 527 and s2d = 219 yields an unseenmax =1338 > topkmax = 1248 and the algorithm conducts one more iteration.This time we access tid 6 from V1 and tid 4 from V2. The top-2 buffer remains unchanged and the linear program is solved one more time using sd1 = 299 and sd 2 = 202. This time, unseen max= 953.5 < topkmax = 1248 and the algorithm terminates. Thus, in total LPTA conducts two sequential and two random accesses per view. In contrast, the TA algorithm executed on R of Figure 1 will identify the correct top-2 results after 12 sorted and 12 random accesses in total. The performance advantage of LPTA is evident.

Algorithm LPTA(U,Q)

```
U={V1,…,V$_r$}//set of views
Q=(Score$_Q$,k,*)//Query
Topk-Buffer={}
topk$_{min}$ =0
for d=1 to n do
    for all views Vi(1≤i≤r) in block-step do
        Let(tid$^i_d$, s$^i_d$) be the d-th item in Vi
            //Update top-k buffer
        Let t$^i_d$ =RandomAccess(tid$^i_d$)
        if Score$_Q$(t$^i_d$)>topk$_{min}$ then
            if(|topk-Buffer|=k) then
                Remove min score tuple from
                                topk-Buffer
            end if
            Add(tid$^i_d$,Score$_Q$(tid$^i_d$)) to topk-buffer
            Topk$_{min}$=min score of topk-Buffer
        end if
    // Checking stopping conditions by solving LP
    Let Unseen$_{max}$ =convex region defined by
    lb$_j$≤X$_j$≤ub$_j$ for every 1≤j≤m
    Score$_{vj}$ ≤s$^j_d$ for every 1≤j≤r
    Compute Unseen$_{max}$ =max$_{tcunseen}${Score$_Q$(t)}
    If(|topk-Buffer|=k) and (Unseen$_{max}$≤topk$_{min}$)then
        Return topk-Buffer
    end if
    end for
end for
```

VIII. Comparison of Different Techniques

This section includes comparison of different techniques employed in the top-k query evaluation. The comparison is performed based on the three important criteria which are ranking function, ranking model and data access operation involved in the different techniques. The ranking function can be generic or monotone. Most of the current top processing techniques assume monotone ranking functions since they fit in many practical scenarios, and have appealing properties allowing for efficient top-k processing. But Few recent techniques address top-k queries in the context of constrained function optimization. The ranking function in this case is allowed to take a generic form.

	Ranking Function		Ranking Model		Data Access	
	Generic	Monotone	Top-k join	Top-k selection	Sorted	Random
PREFER		•	•	•	N/A	N/A
Onion Technique		•		•	N/A	N/A
TA		•		•	•	•
FA		•		•	•	•
Naive	•			•	•	

Another criteria is ranking model. It can be top-k join or top-k selection. In top-k selection model, the scores are assumed to be attached to base tuples. A top-k selection query is required to report the k tuples with the highest scores. Scores might not be readily available since they could be the outcome of some user-de Consider a set of relations R1 ,….,Rn. A top-k join query joins R1,…,Rn, and returns the k join results with the largest combined scores. The combined score of each join result is computed according to some function F(p1,…., pm), where p1,….,pm are scoring predicates defined over the join results. Fined scoring function that aggregates information coming from different tuple attributes. Third criteria is data access which can be sorted access or random access. In sorted access, Object R has the lth highest grade in the ith list, then l sorted accesses to the ith list are required to see the grade under sorted access and in random access, grade of object R in the ith list obtains it in one random access.

IX. Conclusion

A surevey of top-k query processing techniques based on the different criterias have done. For this purpose, a detailed analysis of different techniques included in three important categories like sorted-list based category, layer based category and view based category have explored.

References Références Referencias

1. R. Fagin, A. Lotem, and M. Naor, "Optimal Aggregation Algorithms for Middleware," Proc. Symp. Principles of Database Systems (PODS), 2001.
2. S. Nepal and M.V. Ramakrishna, "Query Processing Issues in Image (Multimedia) Databases," Proc. 15th Int'l Conf. Data Eng. (ICDE), 1999.
3. U. Guntzer, W.T. Balke, and W. Kiebling, "Optimizing Multi-Feature Queries for Image Databases," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2000.
4. Y.-C. Chang, L.D. Bergman, V. Castelli, C.S. Li, M.L. Lo, and J.R. Smith, "The Onion Technique: Indexing for Linear Optimization Queries," Proc. ACM SIGMOD, 2000.
5. D. Xin, C. Chen, and J. Han, "Towards Robust Indexing for Ranked Queries," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2006.
6. Hristidis, N. Koudas, and Y. Papakonstantinou, "Prefer: A System for the Efficient Execution of Multi -Parametric Ranked Queries," Proc. ACM SIGMOD, 2001.
7. G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, "Answering Top-K Queries Using Views," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2006.
8. S. Bo¨ rzso¨ nyi, D. Kossmann, and K. Stocker,

"The Skyline Operator," Proc. 17th Int'l Conf. Data Eng. (ICDE), 2001.
9. D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An Optimal and Progressive Algorithm for Skyline Queries," Proc. ACM SIGMOD, 2003.
10. Ihab F. Ilyas, George Beskales And Mohamed A. Soliman, "A survey of top-k query processing technique in relational database systems" University of Waterloo, Support was provided in part by the Natural Sciences and Engineering Research Council of Canada 2011.
11. D. Kossmann, F. Ramsak, and S. Rost, "Shooting Stars in the Sky: An Online Algorithm for Skyline Queries," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2002.
12. T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms. MIT Press, 2001.
13. D. Campbell and R. Nagahisa, "A Foundation for Pareto Aggregation," J. Economical Theory, vol. 64, pp. 277-285, 1994.
14. M. Voorneveld, "Characterization of Pareto Dominance," Operations Research Letters, vol. 32, no. 3, pp. 7-11, 2003.
15. C. Li, B.C. Ooi, A.K.H. Tung, and S. Wang, "DADA: A Data Cube for Dominant Relationship Analysis," Proc. ACM SIGMOD, 2006.
16. Y. Tao, V. Hristidis, D. Papadias, and Y. Papakonstantinou, "Branch-and-Bound Processing of Ranked Queries," Information Systems, vol. 32, no. 3, pp. 424-445, 2007.
17. R.J. Lipton, J.F. Naughton, and D.A. Schneider, "Practical Selectivity Estimation through Adaptive Sampling," Proc. ACM SIGMOD, 1990.
18. R.J. Lipton and J.F. Naughton, "Query Size Estimation by Adaptive Sampling," Proc. Symp. Principles of Database Systems (PODS), 1990.
19. S. Chaudhuri, N.N. Dalvi, and R. Kaushik, "Robust Cardinality and Cost Estimation for Skyline Operator," Proc. 22nd Int'l Conf. Data Eng. (ICDE), 2006.
20. C.B. Barber, D.P. Dobkin, and H. Huhdanpaa, "The QuickhullAlgorithm for Convex Hulls," ACM Trans. Math. Software, vol. 22, pp. 469-483, 1996.
21. D. Xin, J. Han, H. Cheng, and X. Li, "Answering Top-k Queries with Multi-Dimensional Selections: The Ranking Cube Approach," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2006.
22. S. Nepal and M. Ramakrishna, "Query Processing Issues in Image (Multimedia) Databases," Proc. 15th Int'l Conf. Data Eng. (ICDE), 1999.
23. M. Li and Y. Liu, "Iso-Map: Energy-Efficient Contour Mapping in Wireless Sensor Networks," IEEE Trans. Knowledge and Data Eng., vol. 22, no. 5, pp. 699-710, May 2010.
24. H. Bast, D. Majumdar, R. Schenkel, M. Theobald, and G. Weikum, "IO-Top-k: Index-Access Optimized Top-k Query Processing," Proc. Int'l Conf. Very Large Data Bases (VLDB), 2006.
25. N. Mamoulis, K.H. Cheng, M.L. Yiu, and D.W. Cheung, "Efficient Aggregation of Ranked Inputs," Proc. 22nd Int'l Conf. Data Eng. (ICDE), 2006.

This page is intentionally left blank