World Applied Sciences Journal 3 (2): 330-335, 2008
ISSN 1818-4952

A New Approach of Query Optimization with Join of N Relations

[1]H.Shirgahi, [2]H. Motameni, [3]A.H. Gandomi and [4]P. Valipour

[1]Department of Computer, Islamic Azad University, Jouybar branch, Jouybar, Iran
[2]Department of Computer, Islamic Azad University, Sari branch, Sari 48164-194, Iran
[3]Department of Civil Engineering, Tafresh University, Tafresh, Iran
[4]Department of Engineering, Islamic Azad University, Jouybar branch, Jouybar, Iran

Abstract: An important subject in integration of information in the large scale is to select Topic with a view to ranking from multiple sources so that transfer cost is become minimum. For this purpose in relations join, the suitable size of relations inputs for getting Top K must be determined. We are presenting in this article, according to the quantity k that is determined in query, a dynamic algorithm for determining input size of N relations in rank aware Queries in the from of hierarchical description that in this case we can efficiently answer to the queries with join of N relations for getting Top K. we implemented suggested algorithm and it is observed according to the gotten results that the amount of sent information by pruned records extraordinarily will be decreased in comparison with traditional algorithm and also the time of query processing extraordinarily will be decreased.

Key words: Join of N relations . query optimization . relational database . rank aware query . Top K

INTRODUCTION

Appearing applied programs, which are connected with queries rank is asked efficient support of queries rank in Database management systems in the real world. Supports of queries rank make Database systems able to answer efficiently to the Information Retrieval systems advantages and Database. Database systems with powerful integrity and compatibility assurance provide data management. On the other hand, Information Retrieval systems provide mechanisms for efficient retrieval and fuzzy ranking that is desirable for the user. An important subject in this connection is determining the suitable size of inputs in relation N for answering to the rank aware query so that in this manner Top K is gotten. It is vital in information integration with large scale to select Top K rank aware from multiple sources and it has also basic role for minimizing transfer cost because as the size of relations become smaller, transfer cost become less. For answering to a query with Top K the traditional solution is to do the join on N relations firstly and then sort the answers according to ranking function and select Top K. this algorithm with a view to implementation is very simple but in fact for getting Top K most of the tuples are not important and they have not any role in the ultimate answer, these extra tuples must be pruned according to correct strategy. Efficient algorithms have been presented for answering to ranking queries in middleware environment [1], in [2] an efficient algorithm has been presented for processing queries with Top K on available database in web in company with increasing parallel making and minimizing the time of answer query. The other algorithm that is used for answering to the ranking queries is estimating input sizes by means of statistical relations, monotony hypothesis and random variables [3, 4]. The other new innovation is making ranking laws in relational databases [5]. The other solution in this connection is improving the join and using the ripple join that minimize the time in order to get estimation with relatively acceptable precision for query results. The main idea of ripple join is join algorithm aware the suggested rank for supporting join queries with Top K relational database [6]. But there are the other methods for answering to the queries with Top K, which are getting Top K by changing query optimisation theorem to the aware searches [7].

An algorithm also has been presented for pruning inputs for supporting Top K in queries with two relations join [8]. We in this article have expanded this algorithm and have presented an algorithm for supporting Top K in queries with N relation join.

MOTIVATIONS

When we have a rank aware query that it's aim is getting Top K, we don't need to all records of table, according to the value K some of the records in relations that have little score, have not any role in the ultimate result, They should be pruned and we should

Corresponding Author: Dr. Hossein Shirgahi, Department of Computer, Islamic Azad University, Jouybar Branch, P.O. Box: 47715-195, Jouybar, Iran

not waste time for computing and information transfer on these records.

Firstly we present a definition of rank aware query. In rank aware query, query define on M attribute A_1 'A2 '…'A_N and relation N in the form of R_1 'R '…'R_N that each A_i (i=1:M) belonge to one relation R_j (j=1:N). Each of the attributes have special domain in comparison with their kind. According to the query, a series of attributes of these relations are applied for projection, aseries of attributes of these relations are used for restriction and join. In the rank aware queries there is a part for ranking that some of relations attributes are presented in the form of a ranking relation which is called ranking function. Ranking function f is formed in the form of attribute M' that is M' <=M. a theory that we have for ranking function f is this: ranking function changes in comparison with all relations are monotonic. In addition to this, the number of suitable answers in rank aware queries is determined too that is just Top K. A sample of a rank aware query is presented in example 1.

Example 1: a family wants to buy a house near a school and their aim is to decrease their general costs. Consider a simple ranking function that is estimating total price of house and five years tuition cost of the school. The search must be done in two relations of house and school in the database, according to the following query.

```
SELECT *
FROM House, School
WHERE (DISTANCE (House.Location, School.Location)<d)
ORDER BY (House.Price + 5*School.Tuition)
Top 10
```

The family only needs at the most to ten results in stead of all result so that the family among these ten results can make a decision. The old method is to do the join on two relations and then get all of the answers and at the end sort them according to the ranking function and select the first ten results, but the cost become too much when the relation are big and the number of answers are too many. But traditional method for big K is less costly, that we'll analyze methods for different K in the part of implementation and experiments. We'll explain the main idea for pruning inputs for supporting Top K on queries with join of relation N in part 3. Well present algorithm stages in the part 4. We'll explain implementation method and we'll present their experiments and results on same sample queries in the part 5. Conclusion and ultimate suggestions are presented in the part 6.

MAIN IDEA

Pruning inputs for supporting Top K in queries with join of N relations, input parameters in company with relation N is in the form of R1 'R2''…'N that each Ri (i=1:N) containing some attributes for joining with other relation that is presented in this form Ri.Join_Feilds[j] (j=1:N-1). Also there is also a ranking function f with monotonic increase that is conformable with equation 1.

$$f(R_1.\text{Rank\_Feilds}[l_1], R_2.\text{Rank\_Feilds}[l_2], \ldots, R_N.\text{Rank\_Feilds}[l_N]) \quad (1)$$

More over there is K for the number of suitable answers there is for relations, also attributes for projection and restriction in this manner: Ri.Projection_Feilds[j'], Ri.Restriction_Feilds[j''] which are applied in the general algorithm. Firstly we present the idea for two relations and then expand it for N relations. Suppose there is tow relations R1 and R2 that are arranged according to their ranking attributes in decreasing manner.

We get the first record of R1 at first and compare it with R2 records one by one from the outset to the finish from the join condition view point, then record the number of times and digit of last record of R2 that was join condition of two relations, we continue this until the counter value become equal with K, if the first record R1 has less number of join, we will continue this for the second record R1 with records R2, these operations will be continued until the counter value become equal with K, at the end we record the digit of least record of R2 that was join condition between two relations. If records of R1 ends but the number of records that was the join condition between two relations become less than K it means that the ultimate answer, which is gotten from join of two relations, is less than K. we also do these stages in the same way for R2 and record the results. Figure 1 shows these explanations.

Firstly we must consider a strategy for joining N relations to get input size for N relations. We can change each join of an N to binary join of N-1,

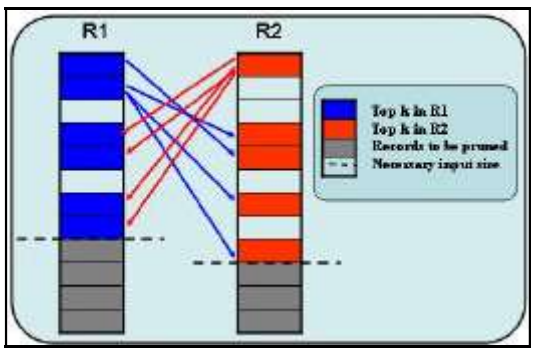

Fig. 1: Specify Input size for achieve top K

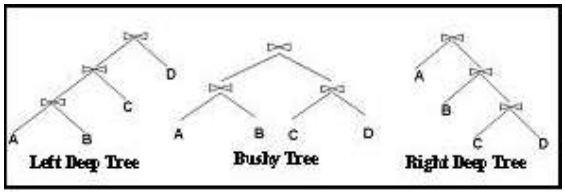

Fig. 2: Types join tree

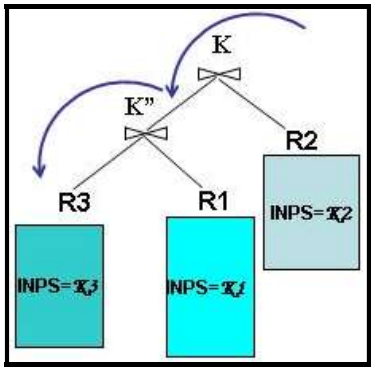

Fig. 3: Specify input sizes in top-down way

therefore join in a query with N relations will be done in the form of hierarchy. The important fact is

The order of join action between relations, it means that which relations must be joined firstly so that the number of operations become minimum, for this purpose join tree must be made. The must important point for determining suitable input size is determining suitable place for each relation in the join tree. Figure 2 shows structure kinds of join tree.

We can get the best state by means to bushy tree but its state space is too wide and its making is time consuming. In this article we have used left deep tree because it has less state space for making. Suggested idea for N relations is as explains below: we determine the size of input in return for both of relations Ri and Rj for them, we put those two relations that have the least input size as two relations on the top level, the theory of join Ri result with Rj lead to least input size and their Input size become orderly K' and K". Then among them we put that relation which has more input size on the right hand on the top level and the other relation on left hand on the top level and make the size of new K equal to min (K', K"), now we compare the other relation N-2 with left hand relation on the top level and output size of new K. for other next stages, each stage select a element which has the least input size as left hand element of level i and output size is equal to input size of this relation for lower level, this operation will be continued to the lowest level. Figure 3 shows a sample of this operation. Algorithm structure and stages will be explained in the next part.

Algorithm stages

1. At first we do the Restriction according to Restriction_feilds each of relations R_1, R_2,..., R_N on them and relations volume optimize greatly.
2. If information and queries are in Distributed systems or computer networks, It is better to optimize them in the manner of volume for transferring information and It seem necessary to prune the extra attributes which have not any role in query by means of projection. We must select all of the requisite attributes for each of the relations and prune the rest of attributes for projection implementation. Requisite attributes after stage1 operation are projection_feilds, join_feilds, Rank_feilds which are single for each of the relations that they are considered and the rest fields are pruned.
3. Relations have been optimized approximately from the line and column viewpoint, in this stage each of the relations become ordered in decreasing manner according to Rank_Feilds and a part of Ranking function that in single manner relates to them.
4. We in return for each R_i and R_j (i=1:N, j=1:N, i<>j) get their input size by means of function prepare_Input_size(R_i,R_j,K) and put in two dimensional array. Details of Function prepare_Input_size are presented in Fig. 4.
5. We get least value in arrays INPS [i,j] by function Min_Item(INPS,RN,CN)and then put relation R_{RN} on the top level on left hand and K selects as its output size, then put relation R_{CN} on the top level on right hand and K selects as its output size and INPS[j,i] selects as its input size. Output size for other stages of other relationN-2 changes to INPS [i,j]. Details of function Min_Item are presented in Fig. 5.
6. For preparing left Deep Tree we operate to form top-down, in this manner that we get the relation of left hand of considered level by procedure prepare_Left_Deep_Tree(R, K, TN) and put the relation of left hand of topper stage as relation of right hand of considered level and put the result of join left hand relation and right hand relation of considered level as relation of left hand of topper stage. We continue these to the lowest level in the form of hierarchical description. Details of procedure prepare_Left_Deep_Tree are presented in Fig. 6.
7. After ending the stage 6, we determine input and output sizes for all of the relations. In this stage we prune extra records according to relations input size.

```
Function Prepare_Input_Size(R1 , R2 :Relation ; K:Integer):Integer;
Var
 Co1,Co2,Accept_Count,Index:Integer;
Begin
 Index:=0; // down most of tuple in R1 that has a join condition in R2
 Accept_Count:=0; // Number of tuples in R1 that has a join condition in R2
 Co2:=0; // Row Number of the current tuple in R2
 R2 . FirstRecord;
 While ( Accept_Count < K ) and ( Co2 < R2 . Recordcount ) do
  Begin
    Co2:=Co2+1;
    Co1:=0; // Row Number of the current tuple in R1
    R1 . FirstRecord;
    While ( Accept_Count < K ) and ( Co1 < R1 . Recordcount ) do
     Begin
       Co1:=Co1+1;
       If Satisfy_Join_Condition ( R1.CurrentRecord , R2.CurrentRecord) Then
        Begin
         Accept_Count:= Accept_Count+1;
         If Co1 > Index Then
           Index:=Co1;
        End;
       R1 . NextRecord;
     End;
    R2 . NextRecord;
  End;
 Prepare_Input_Size:=Index;
End;
```

Fig. 4: Details of function prepare_Input_size

```
Type
 MIN = 10; // Max Tables Number
 Input_Size_For_Relation = array[1... MIN,1... MIN] of Integer ;
 Cell = Record
   Row: Byte;
   Column:Byte;
   Value: Integer;
 End;

Function Min_Item(INPS: Input_Size_For_Relation; Row_Num , Column_Num: Byte) : Cell ;
Var
 Co1,Co2:Byte;
 Min:Cell;
Begin
 Min.Row:=1;
 Min.Column:=2;
 Min.Value:= INPS[1,2];
 For Co1:=1 to Row_Num do
   For Co2:=1 to Column_Num do
     If ( Co1 <> Co2 ) and (INPS[Co1,Co2] < Min.Value ) Then
      Begin
       Min.Row:=Co1;
       Min.Column:=Co2;
       Min.Value:= INPS[Co1,Co2];
      End;
 Min_Item :=Min;
End;
```

Fig. 5: Details of function Min_Item

8. Because relations optimized enough, we do the join between tables and gets the value of topper k according to the Ranking function f that hand been determined in query. It this state the number of last answers maybe more that K, in this state we select answer k with more Ranking volume and prune the rest. But in some queries maybe there is nit answer k that we at most number of possible answers. In this state suggested approach cost become more than traditional approach, even for big ks suggested approach computing cost sometimes become more than traditional approach. We should consider some plans for optimization approach.

If two relations are found in stage four that number of records of they which can join together become less thank don't continue this approach any more and do the traditional approach system that is more optimized from the cost viewpoint, but operations of stage one to three

```
Type
  Relations = array [1..TN] of Relation;//TN is Number of Tables in Query
  Input_Size_for_Relations=array[1..TN] of Integer;
Procedure Prepare_Left_Deep_Tree(R:Relations;var K:Integer;var TN:byte);
Type
 Min_Index=record
  Num:byte;
  Value:Integer;
 End;
Var
  INPS:Input_Size_for_Relations;
  MIN:Min_Index;
  co:byte;
  Temp:Relation; // for Transfer R with min Input Size to
Begin                    // Left Side In Current Level
  MIN.Num:=1;MIN.Value:=Max_value('Integer');//
  For co:=1 to TN-1 do
    Begin
     INPS[co]:=Prepare_Input_Size(R[co],R[TN],K);
     If INPS[co]< MIN.Value then
       Begin
        MIN.Num:=co;
        MIN.Value:=INPS[co];
       End;
    End;
  R[TN].INPS:=Prepare_Input_Size(R[TN],R[MIN.Num],K);
  R[TN].OUTPS:=K;
  if(TN=2) then // for last Level
   Begin
     R[TN-1].INPS:=Prepare_Input_Size(R[MIN.Num],R[TN],K);
     R[TN-1].OUTPS:=K;
   End;
  R[MIN.Num].OUTPS:=MIN.Value;
  K:=MIN.v; // for Next Step
  TN:=TN-1;
  Temp:=R[TN];
  R[TN]:=R[MIN.Num];
  R[MIN.Num]:=Temp;
  if TN >1 then
    Prepare_Left_Deep_Tree(R,K,TN);
End;
```

Fig. 6: Details of function Prepare_Left_Deep_Tree

are efficient for both of traditional and suggested approaches. When two tables haven't join condition together, we must multiply them together Cartesian that in this manner if we consider two relation output size K, their input size will consider K.

IMPLEMENTATIONS AND EXPERIMENTS

We have implemented that approach by means of software Delphi7, SQL server 2000. This system has been consist of these parts and facilities:

- A part for determining database to determine considered database
- You can add a query to the system.
- We design a decomposer that it can get the query and check it and identify and get the query different relations and parts which consist of restriction, projection, join and its conditions, ranking and Top K.
- Implementation of algorithm stages is done on input queries according to the part 4.
- We present the query ultimate results at the end.

We implement this system on some sample query and for different Ks and compare its results with traditional system. Database that is considered for the sample is a database that has been designed for

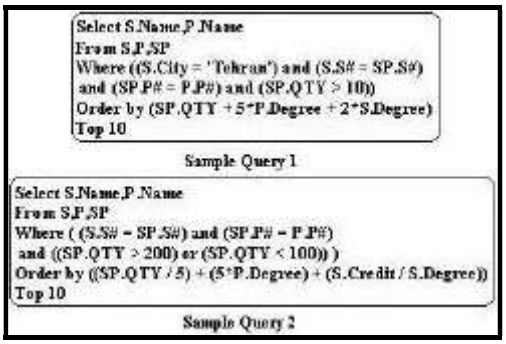

Fig. 7: Two sample queries

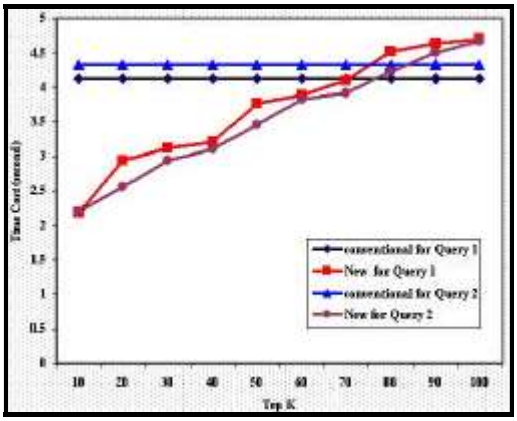

Fig. 8: Comparison of time cost for traditional and suggested systems

information maintenance of a generator system. This contains theses three relations:

- Suppliers relation(s):It contains information about potential and actually suppliers. Its attributes are: Supplier number (S#), Supplier name (Name), Supplier city(City), Supplier degree(Degree), Supplier credit(Credit).
- Products relation (p): It contains information about products which can produce. Its attributes are: Product number (P#), Product name (Name), Product keeping city(City), Product degree(Degree), Product color(Color), Product weight(Weight).
- Productions relation (SP): It contains information about productions. Its attributes are: Supplier number (S#), Product number (P#), quantity (QTY).

We prepare relations information by designing a random producer program that relation of P and S is nearly 4000 record and SP relation is approximately 10000 record. We consider two queries of Fig. 7 as

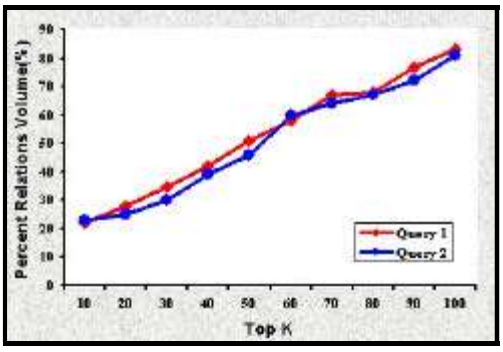

Fig. 9: Comparison relations volume after use algorithm with primary relations volume

sample and implement them for different Ks in traditional and suggested approaches and then analyze the results.

Figure 8 shows the comparison of time cost for traditional and suggested systems. We have used simulation approaches for getting instructions implementation cost, in this manner that we get the number of applied instructions in implementing the query in the program and multiply it to the approximate cost average of each instruction. Moreover we compute records retrieval cost by product of the number of retrieved records in average of retrieval time of a record that is gotten by stored procedures in SQL server 2000, time cost equals to sum of these two time. Figure 9 shows all the relations volume after algorithm implementation on inputs size in comparison with volume of relations primary state for different Ks. We also analyze the accuracy of information which is gotten by means of this approach, In all of the analyzed queries and for different Ks, the answers of this approach is100% equal to answers that is gotten by means of traditional approaches.

CONCLUSION AND SUGGESTIONS

The presented approach in big databases that has high information volume and their aim is implementing Rank aware Queries with small Ks amount is suitable and efficient. It is also efficient for systems that their aim is information integration from some systems proportionate to their requisite queries and relations because it optimizes information volume of relations proportionate to query and determination of requisite input size of relations. this approach makes time cost less from 40 to 50% for small Ks and also makes requisite volume of relation less for transferring from half of the primary volume. We can expand this approach for distributed databases and take advantages from this approach in those systems. The other

operation that we can implement for improving and optimizing the results is to implement the suggested approach as a complement for an approach that is presented in [7] and implement the search by means of changing optimization to aware searches, instead of join cost of N relations, on a graph that is gotten of relations.

REFERENCES

1. Fagin, R. and A. Lotem, 2001. Optimal Aggregation Algorithms for Middleware. In Proceedings of PODS 2001, Santa Barbara, USA.
2. Marian, A. and N. Bruno, 2004. Evaluating Top-K Queries over Web Accessible Databases. ACM Transactions on Database Systems, 29 (2): 319-362.
3. Ilyas, I.F. and W.G. Aref, 2006. Adaptive Rank aware Query Optimization in Relational Databases. ACM Transactions on Database Systems.
4. Ilyas, I.F. and R. Shah, 2004. Rank-aware query optimisation. SIGMOD 2004, June 13-18, Paris, France.
5. Ilyas, I.F. and C. Li, 2005. Ranksql: Query algebra and optimization for relational top-k queries. SIGMOD 2005, June 14-16, Baltimore, Maryland, USA.
6. Ilyas, I.F. and W.G. Aref, 2003. Supporting Top-k Join Queries in Relational Databases. Proceedings of the 29th VLDB Conference, Berlin, Germany.
7. Zhang, Z. and S. Hwang, 2006. Boolean+Ranking: Querying a Database by KConstrained Optimization. SIGMOD 2006, June 27.29, 2006, Chicago, Illinois, USA.
8. Liu Jie and Liang Feng, 2006. A Pruning-based Approach for Supporting Top-K Join Queries. ACM, Edinburgh, Scotland.