International Journal of Engineering, Mathematical and Physical Sciences
ISSN: 2517-9934
Vol:7, No:7, 2013

Re-Optimization MVPP Using Common Subexpression for Materialized View Selection

Boontita Suchyukorn and Raweewan Auepanwiriyakul

***Abstract*—**A Data Warehouses is a repository of information integrated from source data. Information stored in data warehouse is the form of materialized in order to provide the better performance for answering the queries. Deciding which appropriated views to be materialized is one of important problem. In order to achieve this requirement, the constructing search space close to optimal is a necessary task. It will provide effective result for selecting view to be materialized. In this paper we have proposed an approach to re-optimize Multiple View Processing Plan (MVPP) by using global common subexpressions. The merged queries which have query processing cost not close to optimal would be rewritten. The experiment shows that our approach can help to improve the total query processing cost of MVPP and sum of query processing cost and materialized view maintenance cost is reduced as well after views are selected to be materialized.

***Keywords*—**Data Warehouse, materialized views, query rewriting, common subexpressions.

I. Introduction

A data warehouse (DW) is a repository of subject-oriented, integrated, time-variant, and non-volatile data collected from multiple, possibly very large, distributed, heterogeneous sources and makes information readily available for querying and analysis. The main reason for defining and storing the materialized views is to avoid accessing the original data source and to increase the efficiency of query processing. The design of data warehouse is one of the most important problems called materialized view selection problem. It is defined as how to select an appropriate set of views to be materialized [1]. There are two concerning majority tasks to solve the materialized view selection problem. First is generating a search space and second is designing the optimization algorithm for selecting the appropriate set of views to be materialized. The appropriated data structure and view selection methodologies have been considered in order to optimize the query cost, view maintenance cost, or both. For the first task the various well known frameworks have been proposed i.e. Lattice Framework [2], [3], AND-OR dag [1], [4] and Multiple View Processing Plan (MVPP) [5]-[7]. The second task can be classified into four categories i.e. deterministic, randomized, evolution and hybrid algorithm [6]. In this paper we focus on search space construction.

Boontita Suchyukorn and Raweewan Auepanwiriyakul are with the School of Applied Statistics, National Institute of Development Administration (NIDA), Bangkok, Thailand (e-mail: boontiitas@hotmail.com raweewan@as.nida.ac.th).

To generate the search space, common subexpressions for among the queries have to be detected and exploited. Thus the original queries will be rewritten using the global common subexpressions. The concept of common subexpressions has been applied to several areas of query processing and optimization [8], [10], [11], [13], and materialized view selection problem [9], [11], [12]. In [10], the general concept of common subexpression was introduced. They described the generally term of common subexpression between the queries and it can be used for rewriting the queries either completely or partially. As the common subexpression between a pair of queries was constructed then original queries were rewritten by using the given common subexpressions. In [11], authors presented the algorithm that exploited common subexpression for multi-query optimization and materialized view selection in conventional database. They presented a comprehensive mechanism for detecting sharable subexpression and constructing candidate covering subexpression that cover a set of similar subexpressions. In [12], authors proposed the technique called closest common subexpression derivator for constructing candidate views to be materialized. Once closest common subexpression derivators between the queries were defined, they exploited them to rewrite the queries.

In order to generate the search space, it is practically impossible to consider all common subexpressions between among queries because of the numerous numbers of possible common subexpressions. The MVPP is one of the several approaches to construct the optimal search space for view selection problem proposed by Yang in [5]. It was generated by using the multiple query processing (MQP) technique. Based on our observation, as the generating of MVPP is constructed by the merging of individual plan in ordering of query weight thus merging of incoming query has to use the global common subexpressions of the previous merging. The benefit of this approach is to avoid a huge search space which some combination would not be considered. However it will lose the global optimization. Therefore some queries should be rewritten by using common subexpression among queries to gain more optimal query processing cost. In this paper, our proposed approach is re-optimization task which is the improvement of query processing cost of cheapest MVPP [5] by rewriting the query using common subexpressions.

II. Multiple View Processing Plan

The MVPP defined by Yang [5] is a directed acyclic graph that presents the query processing plan of a set of queries. A simple MVPP is shown in Fig. 1.

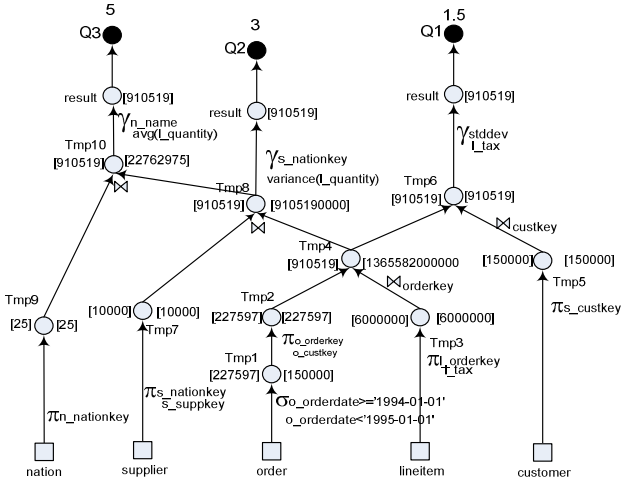


Fig. 1 The simple MVPP of three queries Q1, Q2 and Q3

The root node represents the query, the leaf nodes correspond to the base relations and all other internal nodes are selection, projection, join or aggregation function. A link exists between two node, if the operator in the upper level, is applied to the result derived by the operator in the lower level in some queries. Each internal node in MVPP is marked by relational operation and the cost of processing operation. Two numbers are associated with the node. The number of rows needed to be read is labeled on the right side and the number of rows generated by each operation is labeled on the left side of the node. The query access frequency is labeled on the top of query. Because of above work and its characteristic, MVPP can present the realistic SQL queries and can support the large number of queries that reflect the real data warehouse environment. The algorithm which is used to build an MVPP is listed below:

1. For every optimal query processing plan push all the select, project operation and aggregate function up along the tree.
2. Create a list of queries in descending order based on the result of their query access frequency multiplied by query processing cost.
3. Merge all optimal query processing plans in the list according to the following order:
   3.1. pick up the first optimal query processing plan from the list
   3.2. incorporate the second query into the first query if they share the same base relations
   3.3. incorporate the third query into previous merging, repeat this step until all optimal query processing plans are merged.
4. Move the first optimal query processing plan to the end of the list.
5. Repeat step 3 and 4 to generate all MVPPs.
6. Push down selection, projection and aggregation functions as deep as possible.
7. Calculate the total query processing cost of each MVPP, and select the one which gives the lowest cost.

III. Using Common Subexpression

Normally for view selection problem, the search space is constructed by using all common or similar subexpressions among the queries. The concept of common subexpression is initially referred to identical or equivalent expression, later the term included expression subsumption. Thereafter commonality between queries has included the possibility for overlapping selection condition. Fig. 2 shows the categorization of using common subexpression between the queries. First, nothing is common. Second, totally overlapping is called subsumption shown in Fig. 2 (a). In the figure shows that Q5 is the intermediate query result for Q6. Third is shown in Fig. 2 (b) which is the overlapping with another query. The last is shown in Figs. 2 (c) and (d). It shows that Q1 has more than one equivalent plan. Q1 has overlapping portion with Q4, meanwhile Q1 has alternative equivalent plan that has overlapping part with Q6. We use this concept of common subexpression to answer new incoming queries, after common subexpressions are detected then they are exploited to construct the global optimal equivalent plan for multiple query processing plans.

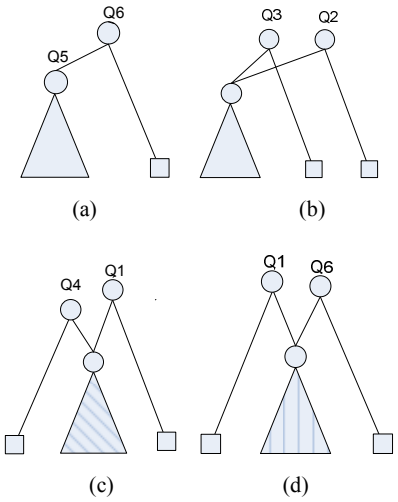


Fig. 2 The partially and totally overlapping of queries: (a) totally overlapping, (b) partially overlapping, and (c) and (d) overlapping more than one query

IV. Proposed Approach to Improve MVPP

In generally construction search space for a view selection problem by considering all possible equivalent plans for all queries is too huge. Constructing MVPP shows that it is the practically possible method to generate the search space.

However the cheapest MVPP [5] can be adjusted to reduce the total query processing cost, as the method of merging described in Section II not consider the common subexpressions of among queries. In our approach, we select the cheapest MVPP to adjust the query processing cost. We find out the alternative equivalent plan of considered queries by using concept of common subexpression. We match the adjusted queries with exists optimal global equivalent plan and those queries are matched in bottom-up way.

A. Re-Optimization to Improve MVPP

The algorithm of the proposed approach is described as Fig. 3.

Algorithm: Re-Optimization Improvement

1. Select cheapest MVPP.
2. Initial list LV = ϕ.
3. Compare Cq(i) of cheapest MVPP with $Cq_n'(i)$ of other MVPPs. If $Cq_n'(i)$ less than Cq(i) then insert query i into LV.

Suppose that Cq(i) presents the query processing cost of query i of cheapest MVPP, $Cq_n'(i)$ presents the query processing cost of query i of n^{th} MVPP.

$Cq_n'(i)$ less than Cq(i) imply that there is more optimal execution plan for query i.

4. For queries in LV, consider the possible commonalities with exists global equivalent plan as following:
   4.1. If there is nothing in common with global equivalent plan then skip to the next query.
   4.2. If there is the overlapping in the form of Figs. 2 (c)-(d) then

rewrite this query using exists common subexpression in MVPP in bottom-up way

else

skip to the next query.

5. Pushing down selection and projection operation as deep as possible.

Fig. 3 Re-Optimization algorithm for improving the cheapest MVPP

B. Rewriting Queries using Common Subexpression

If a view V is defined as a common subexpression of a set of queries. Each query Q in set of queries called a parent of the view V if it can be answered using V. For example node Tmp4 in Fig. 1 is a common subexpression of Q1, Q2 and Q3, it is defined as view V then Q1, Q2 and Q3 are called parent of Tmp4. The answering using view is known as query rewriting using view [14]. It is defined that there is a set of view V_1, V_2, ...,V_m and given a query Q, then a rewriting of query Q using views V is a query that reference V and/or base relations. For example Q1 in Fig. 1, there is more than one equivalent plans i.e. {(orders ⋈ lineitem) ⋈ customer}, {lineitem ⋈ (orders ⋈ customer)}. Its query processing cost is 7,531,979,700,000 if its equivalent plan is {lineitem ⋈ (orders ⋈ customer)}, whereas if we rewrite Q1 using tmp4, its query processing cost is 1,502,168,638,116. We use the cost model for finding query processing cost introduced by Yang [5]. After rewriting the query, the result shows that execution plan using Tmp4 is less than the previous one. We can conclusion that the execution plan providing minimal query processing cost should be chosen. In our approach we rewrite the query identified by our algorithm by comparing its individual plan with common subexpression in MVPP. The query rewriting will be processed in bottom-up way which is calculated from the base relations to the root of the equivalent plan. The process is listed as following:

1. Match individual plan of query with MVPP from base relation to the root node as following:
   1.1. Divide the individual plan into several disjointed subtrees.
   1.2. If there is subtree containing the set of leaf nodes that are already joined conjunctively in MVPP
   then
   this subtree has been selected.
   else
   the set of leaf nodes that are not joined in MVPP but joined in query has been selected.
2. Find the common ancestor node that provide the minimal query processing cost of elements of each subset either in MVPP or in query, create new node(s) to join these ancestor nodes, and associated edges in query.

C. Cost Model for Materialized View Selection Problem

According to [5], a linear cost model is used to calculate the processing cost of query Q. The cost of answering Q is the number of rows in the base relations used to construct Q. Denote *M* be a set of materialized views, $C_{q_i}(M)$ be the cost to compute q_i from the set of *M*, $C_m(v)$ be the cost of maintenance when *v* is materialized, and f_a, f_u are querying and updating frequency respectively. Then the total query processing cost is $\sum_{q_i \in Q} f_{q_i} C_{q_i}(M)$. The total maintenance cost is $\sum_{v \in M} f_u C_m(v)$

Therefore the total cost of materialized views *M* is

$$\sum_{q_i \in Q} f_{q_i} C_{q_i}(M) + \sum_{v \in M} f_u C_m(v)$$

Our goal is the value of total cost will be minimal among all feasible sets of materialized view.

V. Experiment Studies

In order to validate our approach, we have run tests on TPC-H database of size 1GB. This database consists of 8 tables i.e. region, nation, supplier, customer, orders, lineitem, part and partsupp. We use around 50 complex read-only queries as running example. Most of them are large, and perform different operations.

The example of our tested queries is shown as Fig. 4. The cost model proposed by [5] has been used to compute query

processing cost, materialized view maintenance cost and total cost. But no constraint has been considered for this evaluation. The queries are denoted as Q1, Q2, Q3, Q4, Q5, Q6 and Q7. Suppose that all base tables are updated once a time and the frequencies of Q1 to Q7 are 2,6,7,2,5,9,3 respectively. The individual accessing plans for each query are shown as Fig. 5. We use the algorithm proposed by [5], described in section II, to generate MVPPs for these 7 queries.

Query1
```
select min(ps_supplycost)
from part, partsupp, supplier,
nation, region
where p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and s_nationkey =
n_nationkey
    and n_regionkey =
r_regionkey
    and r_name = 'ASIA';
```
Query2
```
select max(o_totalprice)
from customer, orders, lineitem,
nation, region
where c_custkey = o_custkey
   and o_orderkey = l_orderkey
   and c_nationkey =
   n_nationkey
   and n_regionkey =
   r_regionkey
   and r_name = 'ASIA'
   and o_orderdate >= '1994-01-
   01'
   and o_orderdate < '1995-01-
   01';
```
Query3
```
select n_name, sum(l_quantity)
from orders, lineitem, supplier,
nation, region
where o_orderkey = l_orderkey
   and l_suppkey = s_suppkey
   and s_nationkey =
   n_nationkey
   and n_regionkey =
   r_regionkey
   and r_name = 'ASIA'
   and o_orderdate >= '1994-01-
   01'
   and o_orderdate < '1995-01-
   01'
group by n_name;
```
Query4
```
select avg(c_accbal)
from partsupp, supplier, customer,
    nation, region
where ps_suppkey = s_suppkey
    and c_nationkey = n_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA';
```
Query5
```
select count(ps_suppkey)
from partsupp, part
where p_partkey = ps_partkey
    and p_brand <> 'Brand#45'
    and not p_type like
   '%BRASS%'
    and p_size in (9,19,49);
```
Query6
```
select variance(ps_supplycost)
from supplier, partsupp, part
where s_suppkey = ps_suppkey
    and p_partkey = ps_partkey
    and p_brand <> 'Brand#45'
    and not p_type like
   '%BRASS%'
    and p_size in (9,19,49);
```
Query7
```
select stddev(l_tax)
from customer, orders, lineitem
where c_custkey = o_custkey
    and o_orderkey = l_orderkey
    and o_orderdate >= '1994-01-
   01'
    and o_orderdate < '1995-01-
   01';
```

Fig. 4 The queries for experiment

Thus these 7 queries we can generate 7 MVPPs. Therefore ordering for merging of first MVPP is {Q4, Q7, Q3, Q2, Q6, Q1, and Q5} and the final MVPP is {Q5, Q4, Q7, Q3, Q2, Q6, and Q1}. Next selection, projection and aggregation function are pushed down as deep as possible for all MVPPs. Finally the total query processing costs of each MVPP will be calculated to find out the cheapest one. The ordering for merging cheapest MVPP is {Q3, Q2, Q6, Q1, Q5, Q4, and Q7}. The cheapest MVPP is shown as Fig. 6. In meanwhile query processing cost of all queries of each MVPP are calculated.

Next step our approach is applied to re-optimize the cheapest MVPP. First, we initial empty list LV. Next query processing cost Cq of each query in cheapest MVPP are compared with query processing cost Cq′ of other MVPP. The query will be put into LV, if its query processing Cq′ of n^{th} MVPP less than Cq of cheapest MVPP. For our experiment, the comparison of query processing cost of each MVPP shown as Table I. The details of the result are described as following. For Q1 its query processing cost of cheapest MVPP is 323,207,240,592 whereas its query processing cost of 1st and 6th MVPP is 67,303,240,592. For Q2 and Q4 their query processing cost of cheapest MVPP are less than or equal to other MVPPs. For Q3 its query processing cost of cheapest MVPP is less than other MVPPs. For Q5 its query processing cost of cheapest MVPP is 800,009,181,380 whereas its query processing cost of 1st and 6th MVPP is 36,282,181,380. For Q6 its query processing cost of cheapest MVPP is 1,443,281,130,000 whereas its query processing cost of 1st and 6th MVPP is 68,572,856,484. For Q7 its query processing cost of all MVPPs is the same. Therefore the result of LV contains {Q1, Q5, Q6}. Next we consider each query in LV using step 4 in Fig. 3. Considering optimal query processing plan of Q1 in Fig. 5, there are sharable base relations with Q3, Q4, Q5 and Q6. Comparing the individual plan of Q1 in Fig. 5 with cheapest MVPP in Fig. 6, individual plan Q1 in Fig. 5 has the partially overlapping with Q3 and Q4 at node Tmp6 and Tmp22 in Fig. 6 respectively. Tmp6 in Fig. 6 is Tmp6 of Q1 and Q3 in Fig. 5. Tmp22 in Fig. 6 is Tmp8 of Q1 and Q4 in Fig. 5. In Fig. 6, as Tmp22 is the upper level of Tmp6 then Tmp22 is chosen to be common subexpression for Q1 and Q4rather than Tmp6. Another node which is the overlapping portion for Q1 with Q5 and Q6 is Tmp18 in Fig. 6. Because Q1 is overlapping with other queries then Q1 is falling into the condition at line 4.2 in Fig. 3. Therefore Q1 would be rewritten as following. We match optimal query processing plan of Q1 from leaf node to the root, we can match the node in optimal query processing plan of Q1 with the nodes in cheapest MVPP at node Tmp4, Tmp6 and Tmp22 respectively. Thus leaf nodes of Q1 that are already joined in existing MVPP are region, nation, supplier, and partsupp. For the relation part, we create the node to join it with node Tmp22. We present the equivalent plan of Q1 before and after rewritten as Fig. 7. Suppose that R, N, S, PS and P represent the base relation region, nation, supplier, partsupp and part respectively. The query processing cost of Q1 after rewritten using sharable common subexpressions with Q4 is reduced from 323,207,240,592 to 67,303,240,592. For Q5 and Q6 they miss the condition at line 4.2 in Fig. 3 so there is no common subexpression for Q5, Q6 in cheapest MVPP. Therefore we skip the rewriting of Q5 and Q6. Finally step 5 we push down selection, operation and aggregation as deep as possible for all affected queries. The result of re-optimize cheapest MVPP is shown as Fig. 8.

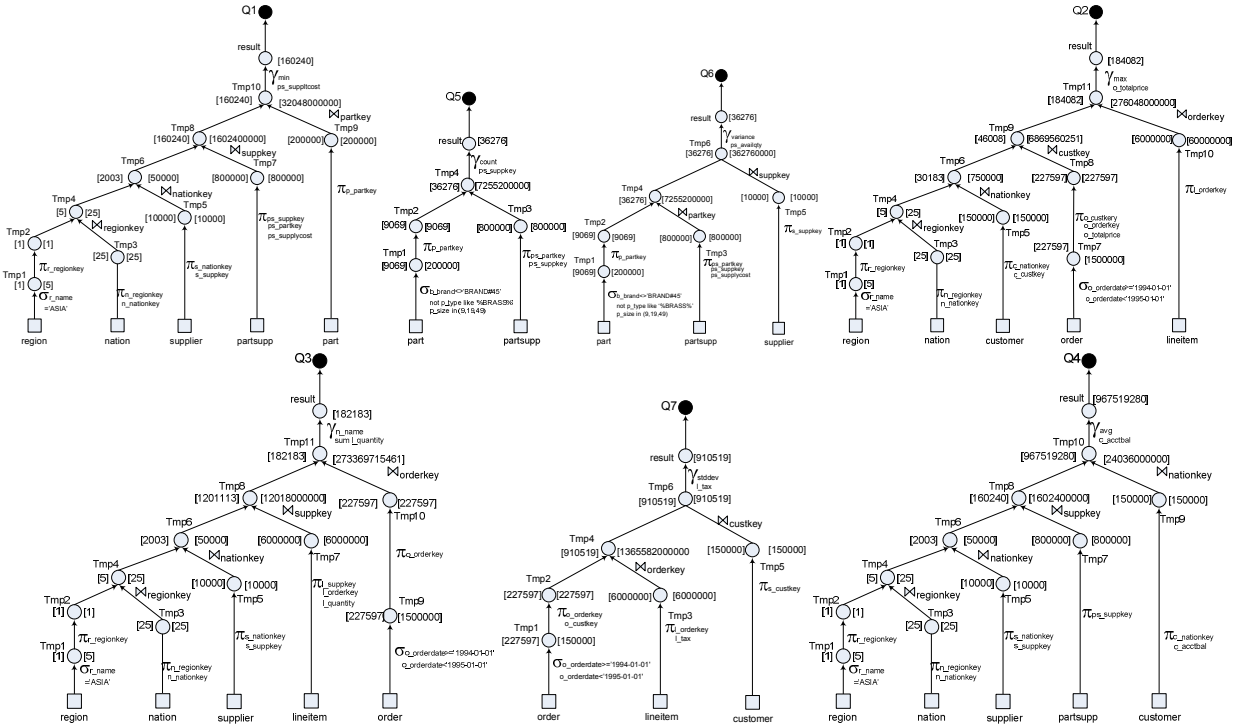


Fig. 5 Individual Optimal Query processing plan

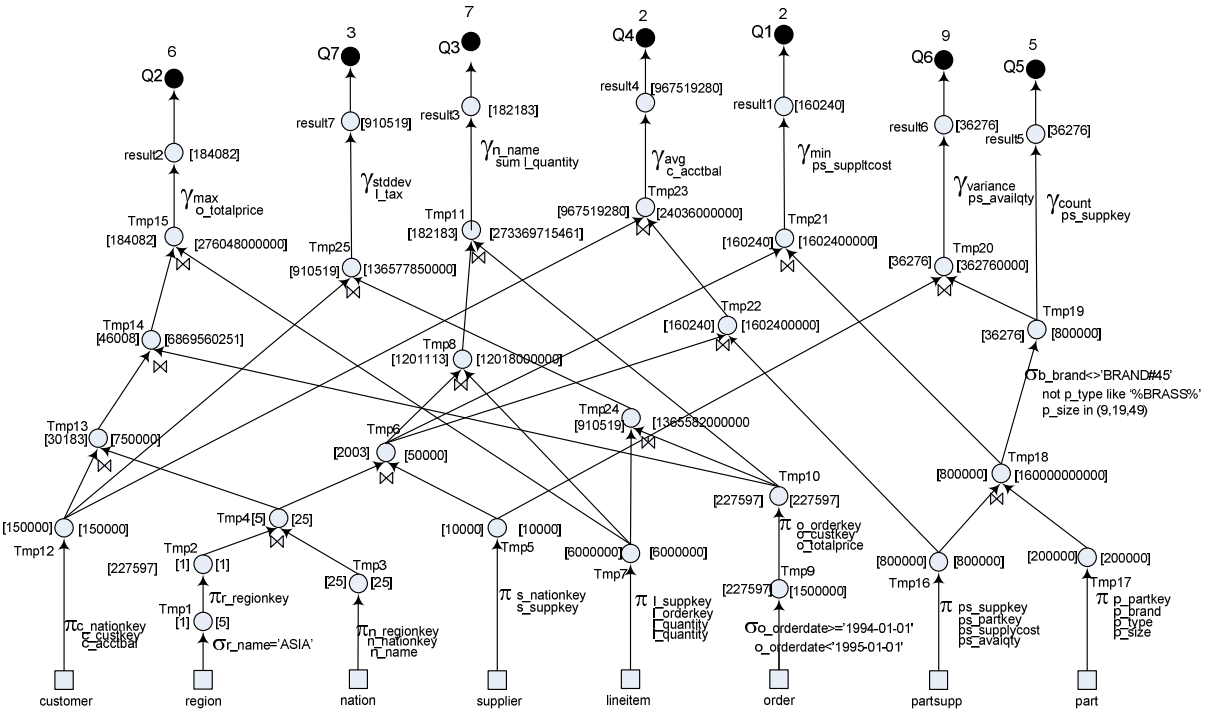
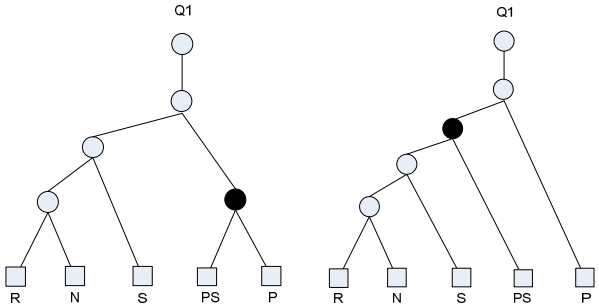


Fig. 6 The cheapest MVPP

TABLE I
QUERY PROCESSING COST FOR EACH QUERY OF EACH MVPP

Query	Cheapest MVPP (3rd MVPP)	1st MVPP	2nd MVPP	4th MVPP	5th MVPP	6th MVPP	7th MVPP
Q1	323,207,240,592	67,303,240,592	323,207,240,592	323,207,240,592	323,207,240,592	67,303,240,592	323,207,240,592
Q2	1,697,558,231,916	9,013,034,785,980	9,013,034,785,980	1,697,558,231,916	9,013,034,785,980	9,013,034,785,980	9,013,034,785,980
Q3	1,997,769,797,079	9,571,896,175,751	9,571,896,175,751	9,571,896,175,751	9,571,896,175,751	9,571,896,175,751	9,571,896,175,751
Q4	53,213,858,672	53,213,858,672	53,213,858,672	19,352,927,718,672	53,213,858,672	53,213,858,672	53,213,858,672
Q5	800,009,181,380	36,282,181,380	800,009,181,380	800,009,181,380	800,009,181,380	36,282,181,380	800,009,181,380
Q6	1,443,281,130,000	68,572,856,484	1,443,281,130,000	1,443,281,130,000	1,443,281,130,000	68,572,856,484	1,443,281,130,000
Q7	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348



(a) Before rewriting (b) After rewriting

Fig. 7 The black node represents the node for rewriting

Finally to evaluate our experiment, Deterministic Algorithm introduced by [5] has been used for selecting view to be materialized. We calculate the query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized view and after select views to materializes by Deterministic algorithm. All cost of cheapest MVPP show in Table II and all cost of improved MVPP show in Table III. The result shows that total cost for all value of re-optimization MVPP are less than cheapest MVPP. The total cost of All-virtual view reduced from 9,353,211,451,044 to 8,427,206,080,471, the total cost of All-materialized views is reduced from 9,092,207,418,537 to 7,688,720,418,537, and the total cost of selected materialized view using Deterministic algorithm is reduced from 6,362,230,690,072 to 6,120,827,977,936.

VI. CONCLUSION

The MVPP is the practically possible search space structure for realistic SQL queries for view selection problem. It exploited the concept of MQP for constructing. We have shown that MVPP will lose the global optimization because of using common subexpression of the previous merging queries rather than among queries.

In this paper, the approach to re-optimize the search space MVPP for view selection problem has been proposed by using concept of commonality of global common subexpression. The evaluation shows that the total query processing cost of MVPP is reduced and after selected views to be materialized the sum of query processing cost and materialized view maintenance cost is reduced as well.

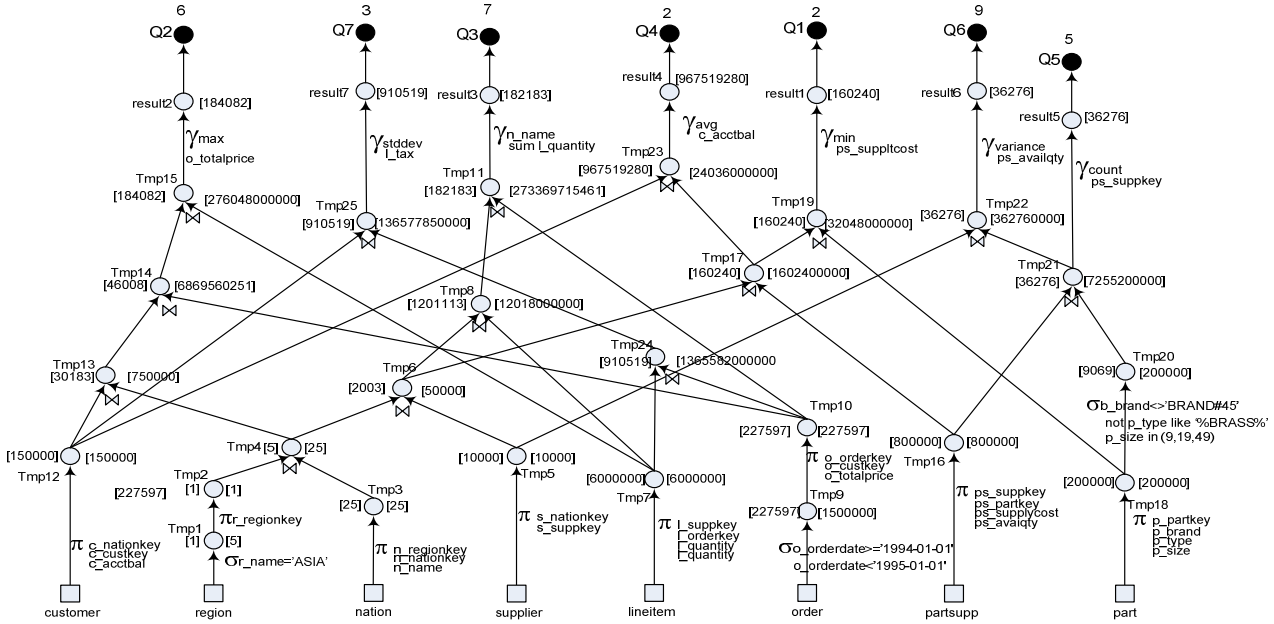


Fig. 8 MVPP after rewriting Q1 using common subexpressions with Q4

TABLE II
THE CHEAPEST MVPP, THE QUERY PROCESSING COST, THE MAINTENANCE AND TOTAL COST

	Cost of query processing	Cost of maintenance	Total Cost
All-virtual view	10,821,545,680,471	0	9,353,211,451,044
All-materialized views	1,940,978,234	9,090,266,440,303	9,092,207,418,537
Deterministic	469,452,759,788	5,892,777,930,284	6,362,230,690,072

TABLE III
THE IMPROVED MVPP, THE QUERY PROCESSING, MAINTENANCE AND TOTAL COST

	Cost of query processing	Cost of maintenance	Total Cost
All-virtual view	8,427,206,080,471	0	8,427,206,080,471
All-materialized views	1,940,978,234	7,686,779,440,303	7,688,720,418,537
Deterministic	533,527,267,652	5,587,300,710,284	6,120,827,977,936

REFERENCES

[1] H. Gupta and I. S. Mumick, "Selection of Views to Materialize in a Data Warehouse," IEEE Transactions on Knowledge and Data Engineering, 2005, pp.24-43.

[2] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing Data Cubes Efficiently," In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 04-06, 1996 (SIGMOD '96). New York: ACM. pp.205-216.

[3] P. Kalnis, N. Mamoulis, and D. Papadias, "View Selection Using Randomized Search," Data & Knowledge Engineering, vol.42 n.1, 2002, pp.89-111.

[4] D. Theodorators and T. Sellis, "Dynamic Data Warehouse Design," In Data Warehousing and Knowledge Discovery(DaWaK'99) of LNCS, Springer-Verlag, 1999, vol.1676, pp.1–10.

[5] J. Yang, K. Karlapalem and Q Li, "Algorithms for Materialized View Design in Data Warehousing Environment," In Proceedings of the 23rd International Conference on Very Large Data Bases, August 25-29, 1997(VLDB'97), 1997, pp.136-145.

[6] J. Yang, C. Zhang, and X. Yao, "An Evolution Approach to Materialized Views Selection in a Data Warehouse Environment," IEEE, vol.31, 2001, pp.282-294.

[7] J. Phuboon-ob and R. Auepanwiriyakul, "Two-Phase Optimization for Selecting Materialized Views in a Data Warehouse," PWASET, vol.21, 2007, pp.277-281.

[8] F. F. Chen, M. H. Dunham, "Common Subexpression Processing in Multipler-Query Processing," IEEE Transaction Knowledge Data Engineering, vol.10 n.3, May 1998, pp.493-499.

[9] H. Mistry, P. Roy, S. Sudarshan, and K. Ramamrithan, "Materialized view selection and maintenance using multi-query optimization," In Proc of the ACM SIGMOD International Conference on Management of Data. Santa Barbara, California, United States, 2001, pp.307-318.

[10] W. Lehner, B. Cochrane, H. Pirahesh, and M. Zahatioudakis, "fAST Refresh Using Mass Query Optimization," In Proc. of the 17th International Conference on Data Engineer, Washington, DC, USA: IEEE Computer Society, 2001, pp.391-398.

[11] J. Zhou , P. Larson , J. Freytag , W. Lehner, "Efficient exploitation of similar subexpressions for query processing," Proceedings of the 2007 ACM SIGMOD international conference on Management of data, June,2007, 11-14, Beijing.

[12] D. Theodoratos, W. Xu, "Computing Closest Common Subexpressions for View Selection Problem," In Proc. of the ACM international workshop on Data warehousing and OLAP (DOLAP 2006), Washington, USA, 2006, pp.75-82.

[13] Y. N. Silva, P Larson, and J. Zhou, "Exploiting Common Subexpressions for Cloud Query Processing," The 28th International Conference on Data Engineering(ICDE), Washington DC. USA, 2012..

[14] A. Y. Halevy, "Answering queries using views: A survey," VLDB Journal, 10(4), 2001.

Boontita Suchyukorn received the B.Eng,(2nd Hons.) degree in Computer Engineering and M.Sc. in Eletronic business from King Mongkut's University of Technology Thonburi, Thailand.

She is currently studying Ph.D. in Computer Science at School of Applied Statistics, National Institute of Development Administration (NIDA), Thailand. Her research interests include database, data warehouses and software application development.

Raweewan Auepanwiriyakul received the B.Sc. degree on Radiological Technology from Mahidol University, Thailand, and the M.S. and Ph.D. degree in Computer Science from University of North Texas, U.S.A.

Currently she is an Associated Professor with School of Applied Statistics, National Institute of Development Administration (NIDA). Bangkok, Thailand. Her research interests include database, objected-oriented analysis and design, and data warehouse.