DYNAMIC MATERIALIZED VIEW SELECTION BASED ON TWO-PHASE OPTIMIZATION

Boontita Suchyukorn

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Computer Science)
School of Applied Statistics
National Institute of Development Administration
2013
DYNAMIC MATERIALIZED VIEW SELECTION BASED ON
TWO-PHASE OPTIMIZATION
Boontita Suchyukorn
School of Applied Statistics

Associate Professor .................................................. Major Advisor
(Raweewan Auepanwiriyaakul, Ph.D.)

The Examining Committee Approved This Dissertation Submitted in Partial
Fulfillment of the Requirements for the Degree of Doctor of Philosophy (Computer
Science).

Associate Professor .............................................. Committee Chairperson
(Surapong Auwatanamongkol, Ph.D.)

Associate Professor .............................................. Committee
(Raweewan Auepanwiriyaakul, Ph.D.)

Assistant Professor ................................................ Committee
(Ohm Sornil, Ph.D.)

Assistant Professor ................................................ Committee
(Charun Sanrach, Ph.D.)

Instructor ............................................................ Dean
(Siwiga Dusadenoaod, Ph.D.)

April 2014
ABSTRACT

Title of Dissertation  Dynamic Materialized View Selection Based on Two-Phase Optimization
Author  Ms. Boontita Suchyukorn
Degree  Doctor of Philosophy (Computer Science)
Year  2013

A Data Warehouse is a repository of information integrated from a distributed data source. Information stored in a data warehouse is the form of the materialized view. Materializing view is a technique to improve query response time in a data warehouse. Deciding which of the appropriated views should be materialized views is one of the significant problems in data warehouse design. In order to solve this problem, constructing a search space close to optimal is a necessary task. It provides effective results for the selection of views to be materialized. The Multiple View Processing Plan (MVPP) is one of the several approaches to construct the optimal search space for the view selection problem. However, some merged queries in MVPP provide the query processing cost not close to optimal. Therefore, we proposed the re-optimized MVPP algorithm to improve the query processing cost of those queries by rewriting them using global common subexpression.

In the real situation, the requirements are frequently changed by the stakeholder. Therefore, the existing materialized views and virtual views derived from static materialized view selection should be considered whether they are still suitable to support all requirements, the existing and new requirements. In this research, we propose an approach for dynamic materialized view selection based on proposed re-optimized MVPP algorithm. We propose the algorithm to determine the existing materialized views and virtual views that are affected by changing the requirement rather than all existing resource in the search space.
The experiment shows that our approach, the re-optimized MVPP, improves the total query processing cost of MVPP. Also the summation of query processing costs and materialized view maintenance costs are reduced after the set of views are selected to be materialized by using the Two-Phase Optimization algorithm. For our dynamic materialized view selection approach, the experiment shows that our approach can specify the member of a set of views to be selected rather than all existing views in the search space. It provides optimal total cost without recalculating all requirements from scratch.
ACKNOWLEDGEMENTS

The author would like to express sincere thanks and really appreciation to my advisor, Associate Professor Dr. Raweewan Auepanwiriyakul, for her kindness in providing valuable advice, support, encouragement and guidance throughout the time of formulating the dissertation. I have been learned more than study by her coaching with kindness and very nice attitude. I cannot express my gratitude enough.

I would also to thanks and appreciation to all of the committee members, Associate Professor Dr. Surapong Auwatanamongkol, Assistant Professor Dr. Ohm Sornil and Assistant Professor Dr. Charun Sanrach, for their thoughtful comments and suggestions.

I also wish to express my thanks to all lecturers and staffs at the School of Applied Statistics, National Institute of Development Administration, for their kind support. I am also thankful to the NIDA Library and Information Center for their help in checking the format of the dissertation.

I would like to voice my special thanks to Mr. Sakchai Hanprasertporn, my former senior manager, for his kind support and encourage me the chance to have the work and study together. Also, I wish to extend thanks to my colleagues, classmates and all kanlayanamitr for providing me the encouragement.

Finally, I would like to thanks to my beloved family for giving me their love and support, and to present the accomplishment of this dissertation to my mother, and passed away father and sister.

Boontita Suchyukorn

April 2014
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter/Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>iii</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>v</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>ix</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xii</td>
</tr>
<tr>
<td><strong>CHAPTER 1 INTRODUCTION</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Problem Statements</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Dissertation Objectives</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Organization of Dissertation</td>
<td>5</td>
</tr>
<tr>
<td><strong>CHAPTER 2 LITERATURE REVIEW</strong></td>
<td>6</td>
</tr>
<tr>
<td>2.1 Query and View Lineage Models for Generating the Search Space</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Materialized View Selection Algorithms</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Dynamic Materialized View Selection Methodologies</td>
<td>14</td>
</tr>
<tr>
<td>2.4 Cost Model for Materialized View Selection</td>
<td>18</td>
</tr>
<tr>
<td>2.5 Common Subexpression</td>
<td>20</td>
</tr>
<tr>
<td>2.6 Weight of Node in MVPP</td>
<td>23</td>
</tr>
<tr>
<td><strong>CHAPTER 3 METHODOLOGY</strong></td>
<td>25</td>
</tr>
<tr>
<td>3.1 Proposed Methodologies</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Multiple View Processing Plan (MVPP)</td>
<td>28</td>
</tr>
<tr>
<td>3.3 Common Subexpression</td>
<td>29</td>
</tr>
<tr>
<td>3.4 Proposed Approach to Improve MVPP</td>
<td>31</td>
</tr>
<tr>
<td>3.5 Static Materialized View Selection</td>
<td>34</td>
</tr>
<tr>
<td>3.6 Selection Algorithm: Two-Phase Optimization (2PO)</td>
<td>35</td>
</tr>
</tbody>
</table>
3.7 Dynamic Materialized View Selection Approach 38
   3.7.1 MVPP Structure Analysis and Merging New Requirements Approach 38
   3.7.2 An Approach to Identify the Affected Nodes 41
3.8 Two-Phase Optimization (2PO) for Dynamic Materialized View Selection 43
3.9 Cost Model for Dynamic Materialized View Selection 44

CHAPTER 4 DESIGN OF EXPERIMENTS AND ANALYSIS OF RESULTS 46

4.1 TPC Benchmark™H (TPC-H) 46
4.2 Query Set for Static Materialized View Selection 48
4.3 The Cheapest MVPP Implementation 56
4.4 The Re-Optimized MVPP Implementation 68
4.5 Evaluation of the MVPP Re-Optimization Algorithm 72
4.6 Dynamic Materialized View Implementation 80
   4.6.1 Query Sets and Implementation of Scenarios 81
   4.6.2 Merging New Requirements into the Existing MVPP Implementation 114
   4.6.3 The Affected Node Identification Algorithm Implementation 129
4.7 Two-Phase Optimization for Dynamic Materialized View Selection 133
4.8 Result and Analysis of Dynamic Materialized View Selection 137
4.9 The Second Experiment for Dynamic Materialized View Selection 139
4.10 Analysis of the Affected Node Identification Algorithm 153

CHAPTER 5 CONCLUSION AND FUTURE WORK 170

BIBLIOGRAPHY 173

APPENDICES

Appendix A Result of Merging Queries to Construct MVPPs 180
Appendix B Result of Using Common Subexpression 197
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Tables</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1 The TPC-H Schema Table Size</td>
<td>47</td>
</tr>
<tr>
<td>4.2 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost</td>
<td>57</td>
</tr>
<tr>
<td>4.3 The Query Processing Cost of the First MVPP</td>
<td>66</td>
</tr>
<tr>
<td>4.4 The Query Processing Cost of All MVPPs</td>
<td>67</td>
</tr>
<tr>
<td>4.5 The Query Processing Cost of the Cheapest MVPP and the Re-Optimized MVPP</td>
<td>72</td>
</tr>
<tr>
<td>4.6 The Maintenance Cost of the Cheapest MVPP</td>
<td>75</td>
</tr>
<tr>
<td>4.7 The Query Processing Cost of the Cheapest MVPP</td>
<td>75</td>
</tr>
<tr>
<td>4.8 The Maintenance Cost of the Re-Optimized MVPP</td>
<td>77</td>
</tr>
<tr>
<td>4.9 The Query Processing Cost of the Re-Optimized MVPP</td>
<td>77</td>
</tr>
<tr>
<td>4.10 The Query Processing Cost, Maintenance Cost and Total Cost of the Cheapest MVPP</td>
<td>78</td>
</tr>
<tr>
<td>4.11 The Query Processing Cost, Maintenance Cost and Total Cost of the Re-Optimized MVPP</td>
<td>78</td>
</tr>
<tr>
<td>4.12 The Comparison of Total Costs of the Cheapest MVPP and the Re-Optimized MVPP</td>
<td>79</td>
</tr>
<tr>
<td>4.13 The Comparison of the Result from the Static Approach and the Dynamic Approach for the Nothing in Common Data Set</td>
<td>91</td>
</tr>
<tr>
<td>4.14 The Weight of the Existing Nodes for Construct Q9 and Q10</td>
<td>94</td>
</tr>
<tr>
<td>4.15 The Weight of Ancestor Nodes of Directly Affected Node of Q9, Q10</td>
<td>95</td>
</tr>
<tr>
<td>4.16 The Comparison of the Result from the Static Approach and the Dynamic Approach for the Subsumption Data Set</td>
<td>97</td>
</tr>
<tr>
<td>4.17 The Weight of the Existing Nodes for Construct Q11 and Q12</td>
<td>99</td>
</tr>
</tbody>
</table>
4.18 The Weight of Ancestor Nodes of Directly Affected Node of Q11, Q12
4.19 The Comparison of the Result from the Static Approach and the Dynamic Approach for Partially Overlapping Data Set
4.20 The Weight of the Existing Nodes for Q3
4.21 The Weight of Ancestor Nodes of Directly Affected Node of Q3
4.22 The Comparison of the Result from the Static and the Dynamic Approach for Deleting Query Q3
4.23 The Weight of the Existing Nodes for Constructing Q13
4.24 The Weight of Ancestor Nodes for the Directly Affected Node of Q13
4.25 The Comparison of the Result for the Static Approach and the Dynamic Approach for Query Q1-Q7 and Q13
4.26 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost of Q8 to Q13
4.27 The Possible Execution Plans for Q13 in the First Dynamic MVPP
4.28 The Query Processing Cost of All Dynamic MVPPs
4.29 The Weight of the Existing Nodes for Constructing All New Queries
4.30 The Weight of Ancestor Node of Directly Affected Node of New Queries
4.31 The Query Processing Cost of Dynamic Approach for All Queries
4.32 The Maintenance Cost of Dynamic Approach for All Queries
4.33 The Comparison of the Result from the Static Approach and the Dynamic Approach for All Queries
4.34 The Query Access Frequency, Query Cost, and Query Access Frequency Multiply Query Cost
4.35 The Query Processing Cost of All MVPPs
4.36 The Query Processing Cost, Maintenance Cost and Total Cost of the Optimal MVPP 144
4.37 The Comparison of the Results from the Static Approach and the Dynamic Approach for the Second Experiment 149
4.38 The Query Processing Cost of the Dynamic Approach 150
4.39 The Maintenance Cost of the Dynamic Approach 150
4.40 The Query Processing Cost of the Static Approach 151
4.41 The Maintenance Cost of the Static Approach 152
4.42 The Cost of All Queries for the Negative Weight Property 160
4.43 The Minimum Total Cost of the First Experiment 161
4.44 The Weight of Ancestor Nodes of Directly Affected Node of New Queries 162
4.45 The States Generated by 2PO of Dynamic Phase for All Nodes 166
4.46 The States Generated by 2PO of Dynamic Phase for Affected Nodes 168
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figures</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Lattice Framework of the Time Dimension</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Sample Data Cube Lattice Framework for Eight Views</td>
<td>8</td>
</tr>
<tr>
<td>2.3 (a) An Expression AND DAG (b) An Expression AND-OR DAG</td>
<td>9</td>
</tr>
<tr>
<td>2.4 (a) An AND DAG (b) An AND-OR DAG</td>
<td>10</td>
</tr>
<tr>
<td>2.5 Multiple View Processing Plan (MVPP)</td>
<td>10</td>
</tr>
<tr>
<td>2.6 Example of the Affected Transference</td>
<td>16</td>
</tr>
<tr>
<td>3.1 The Diagram for Dynamic Materialized View Selection Approach</td>
<td>26</td>
</tr>
<tr>
<td>3.2 The Simple MVPP of Three Queries Q1, Q2 and Q3</td>
<td>28</td>
</tr>
<tr>
<td>3.3 The Algorithm for Implementing the MVPP</td>
<td>29</td>
</tr>
<tr>
<td>3.4 The Categories of Subexpression Commonalities</td>
<td>30</td>
</tr>
<tr>
<td>3.5 The Proposed Algorithm: The MVPP Re-Optimization Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>3.6 The Plan of Q1 in MVPP of Q1,Q2 and Q3</td>
<td>33</td>
</tr>
<tr>
<td>3.7 The Rewriting Query Steps</td>
<td>34</td>
</tr>
<tr>
<td>3.8 The Static Materialized View Selection for the Static Phase</td>
<td>35</td>
</tr>
<tr>
<td>3.9 The Iterative Improvement (II) Algorithm</td>
<td>36</td>
</tr>
<tr>
<td>3.10 The Simulated Annealing (SA) Algorithm</td>
<td>36</td>
</tr>
<tr>
<td>3.11 The Materialized View Selection with 2PO</td>
<td>37</td>
</tr>
<tr>
<td>3.12 The Subgraph of the Existing MVPP is the Subsumption of the Subtree of New Query</td>
<td>39</td>
</tr>
<tr>
<td>3.13 The Methodology to Merge New Requirements into the Exiting MVPP</td>
<td>40</td>
</tr>
<tr>
<td>3.14 The Affected Node Identification Algorithm</td>
<td>42</td>
</tr>
</tbody>
</table>
3.15 The Materialized View Selection with 2PO for the Dynamic Phase

4.1 The TPC-H Schema Revision 2.14.2
4.2 Relational Algebra Query Tree of Query Q1
4.3 Relational Algebra Query Tree of Query Q2
4.4 Relational Algebra Query Tree of Query Q3
4.5 Relational Algebra Query Tree of Query Q4
4.6 Relational Algebra Query Tree of Query Q5
4.7 Relational Algebra Query Tree of Query Q6
4.8 Relational Algebra Query Tree of Query Q7
4.9 The Result of the First Step to Construct MVPP
4.10 The Result of Merging Steps for the First MVPP
4.11 The Result of Merging Steps for the Second MVPP
4.12 The First MVPP, the Queries in the List: {Q4, Q7, Q3, Q2, Q6, Q1, and Q5}
4.13 The Second MVPP, the Query in the List: {Q7, Q3, Q2, Q6, Q1, Q5, and Q4}
4.14 The Third MVPP (the Cheapest MVPP), the Query in the List: {Q3, Q2, Q6, Q1, Q5, Q4, and Q7}
4.15 The Fourth MVPP, the Query in the List: {Q2, Q6, Q1, Q5, Q4, Q7, and Q3}
4.16 The Fifth MVPP, the Query in the List: {Q6, Q1, Q5, Q4, Q7, Q3, and Q2}
4.17 The Sixth MVPP, the Query in the List: {Q1, Q5, Q4, Q7, Q3, Q2, and Q6}
4.18 The Seventh MVPP, the Query in the List: {Q5, Q4, Q7, Q3, Q2, Q6, and Q1}
4.19 The Execution Plan of Q1 Before and After Rewriting
4.20 The Cheapest MVPP after Applying the MVPP Re-Optimization Algorithm
4.21 The Materialized View Selection with 2PO
4.22 The Cheapest MVPP with Materialized Views 73
4.23 The Re-Optimized MVPP with Materialized View Nodes Selected by 2PO 76
4.24 Relational Algebra Query Tree of Query Q8 82
4.25 Relational Algebra Query Tree of Query Q9 83
4.26 Relational Algebra Query Tree of Query Q10 84
4.27 Relational Algebra Query Tree of Query Q11 85
4.28 Relational Algebra Query Tree of Query Q12 86
4.29 Relational Algebra Query Tree of Query Q13 88
4.30 The Existing Re-Optimized MVPP with Q8 by Dynamic Approach 90
4.31 The Re-Optimized MVPP for Q1-Q7, and Q8 by Static Approach 90
4.32 The Existing Re-Optimized MVPP with Q9 and Q10 by Dynamic Approach 92
4.33 The Re-Optimized MVPP for Q1-Q7, and Q9-Q10 by Static Approach 96
4.34 The Existing Re-Optimized MVPP with Q11 and Q12 by Dynamic Approach 98
4.35 The Existing Re-Optimized MVPP with Q11 and Q12 by Dynamic Approach after Applying 2PO 101
4.36 The Cheapest MVPP for Q1-Q7, and Q11-Q12 by Static Approach 102
4.37 The Re-Optimized MVPP for Q1-Q7, and Q11-Q12 by Static Approach 103
4.38 The Re-Optimized MVPP Structure with Deleting Q3 for Dynamic and Static Approach 104
4.39 The Existing Re-Optimized MVPP with Q13 by Dynamic Approach 108
4.40 The Existing Re-Optimized MVPP with Q13 by Dynamic Approach after Applying 2PO 112
4.41 The Cheapest MVPP for Query Q1 to Q7, and Q13 by Static Approach
4.42 The Re-Optimized Cheapest MVPP for Query Q1 to Q7, and Q13 by Static Approach
4.43 The Result of Merging Steps for the First Dynamic MVPP
4.44 The First Dynamic MVPP after Optimized, Queries in the List {Q10, Q8, Q11, Q12, Q9 and Q13}
4.45 The Result of Merging Steps for the Second Dynamic MVPP
4.46 The Second Dynamic MVPP: the Queries in the List {Q8, Q11, Q12, Q9, Q13 and Q10}
4.47 The Third Dynamic MVPP: the Queries in the List {Q11, Q12, Q9, Q13, Q10 and Q8}
4.48 The Fourth Dynamic MVPP: the Queries in the List {Q12, Q9, Q13,Q10, Q8 and Q11}
4.49 The Fifth Dynamic MVPP: the Queries in the List {Q9, Q13, Q10, Q8, Q11 and Q12}
4.50 The Sixth Dynamic MVPP: the Queries in the List {Q13,Q10, Q8, Q11, Q12 and Q9}
4.51 The Dynamic MVPP for Q8-Q13 after Applying 2PO
4.52 The Cheapest MVPP of Q1-Q13 by Static Approach
4.53 The Re-Optimized MVPP of Q1-Q13 by Static Approach
4.54 The Relational Algebra Query Tree for the Initial Queries: Q4, Q15, Q22, Q33, Q40, Q43 and Q50
4.55 The Cheapest MVPP for the Second Experiment: the Queries in the List {Q43, Q40, Q50 Q4, Q33, Q22 and Q15}
4.56 The Optimal MVPP with Materialized Views for Initial Requirements
4.57 The Relational Algebra Query Tree for the New Queries: Q3, Q6, Q28, Q30, Q32 and Q47
4.58 The Optimal Dynamic MVPP Constructed by Merging Steps
4.59 The Dynamic MVPP after Applying 2PO
4.60 The Re-Optimized MVPP by Static Approach for All Queries 148
4.61 The Dynamic MVPP for Q8-Q13 after Applying 2PO 161
CHAPTER 1

INTRODUCTION

Data warehousing is an approach to integrate data from heterogeneous data sources typically from multiple online transaction processing (OLTP) databases. A data warehouse is defined as a subject-oriented, integrated, time-varying, nonvolatile collection of data that is used primarily for querying and analysis to support management decisions for entire organization (Inmon, 2002: 389). The data warehouse is often used to support the decision support system. The queries used in data warehouse are more complex than those used in the traditional OLTP database. These queries are mostly complex of operations that are joins and aggregations of large volume of historical data. The query response time is important for analytical processing. Therefore, increasing the efficiency of query processing is necessary. In order to increase the performance of executing the query, we can use an approach which is to store some intermediate results of the queries called materialized views. The materialized view is a common technique to reduce query response time by pre-calculating expensive operations in the data warehouse (Bello, Dias, Downing, Feenan, Finnerty, Norcott, Sun, Witkowski, and Ziauddin, 1998: 659).

Normally, the views or virtual views are the derived relations defined by a query in terms of base relations and/or other views. The virtual views are stored in database system in form of the query definition in contrast to base relations whose tuples are always physically stored in the database system (Garcia-Molina, Ullman and Widom, 2009: 341; Elmasri and Navathe, 2010: 133). The view defines a function from a set of base relations to a derived relation. This function is typically recomputed every time when the view is referred to; benefit being that data provided is always up to date. In contrast, materialized view is a view whose contents are computed and stored. Materialized view is technique to reduce the response time of complex queries by pre-computation the complex queries and physically store in data-
base system. The main reason for defining and storing materialized view is to avoid accessing the original data source and to increase the efficiency of querying performance. Therefore, query accessing performance by using materialized views is speed up dramatically from hours or days to seconds or minutes. However, materialized views have to be in synchronization with the source data in order to maintain the consistency and integrity of the source data. The changing of the source data should be reflected to the materialized views. The process of refreshing a materialized view in response to the changes in the base relation is called view maintenance that incurs a view maintenance cost. Therefore, not only the query processing cost but materialized view maintenance cost also has to be considered. As materializing view technique has maintenance cost when base tables are changed, it is not possible to materialize all views so the tradeoff between performance and view maintenance is taken into consideration (Silberschatz, Korth and Sudarshan, 2010: 597).

The materialized view selection problem is one of the important problems in data warehouse design. It is defined as how to select an appropriate set of views to be materialized over a global processing plan by merging optimized individual processing plans of queries subjected to the minimum of summation of query processing and maintenance cost (Gupta and Mumick, 2005: 24; Zhang, Yao, and Yang, 2001: 287). There are three different strategies for materialized view selection to facilitate query processing (Yang, Karlapalem and Li, 1997: 139). First, to materialize all of the views in the data warehouse can achieve the best performance but it may take high maintenance cost, memory space and time constraints. Second, to have all the virtual views will have the low view maintenance cost but may take a lot of time to answer the queries. Third, some of views will be materialized and others will be left to be virtual views. The third strategy provides the tradeoff between maintenance cost and query processing cost. Therefore, it is necessary for selecting an appropriate set of views to be materialized which minimizes the summation of query processing cost and view maintenance cost. There are two concerning majority tasks in order to solve the materialized view selection problem. First, is to generate a search space, and second is to design the optimization algorithm for selecting the appropriate set of views to be materialized. The appropriated search space 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 for example Lattice Framework (Harinarayan, Rajaraman and Ullman, 1996: 205; Kalnis, Mamoulis and Papadias, 2002: 89), AND-OR DAG (Theodoratos and Sellis, 1999: 1; Gupta and Mumick, 2005:24) and Multiple View Processing Plan (MVPP) (Yang et al., 1997: 136; Yang, Zhang, and Yao, 2001: 282; Phuboon-ob and Auepanwiriyakul, 2007: 166; Derakhshan, Dehne, Korn and Stantic, 2006: 92; 2008:125). The second task can be classified into four categories i.e. deterministic, randomized, evolutionary and hybrid algorithm (Yang, Zhang and Yao, 2001: 282; Zhou, Geng and Xu, 2011: 131).

1.1 Problem Statements

In order to generate the search space, which is the first task to solve materialized view selection problem, the common subexpressions of queries have to be detected and exploited. The concept of common subexpressions has been applied to several areas of query processing (Jarke, 1984: 192; Chen and Dunham, 1998: 493; Lehner, Cochrane, Pirahesh and Zahatioudakis, 2001: 391; Zhou, Larson, Freytag and Lehner, 2007: 534; Silva, Larson and Zhou, 2012: 1339), and materialized view selection problem (Yang et al., 1997: 136; Mistry, Roy, Sudarshan and Ramamrithan, 2001: 311; Theodoratos and Xu, 2006: 75). However, it is practically impossible to consider all common subexpressions between all queries. The MVPP is one of the several approaches to construct the optimal search space for view selection problem proposed by Yang et al. (1997:138). It is generated by using the Multiple Queries Processing (MQP) technique. The reason we choosing the MVPP is that MVPP presents the realistic structured query language (SQL) queries and supports a large number of queries that reflects the real data warehouse environment. However, as the generating of MVPP is constructed by the merging of individual optimal plan in order of query weight so merging of incoming query has to use the common subexpressions of the previous merging. The benefit of MVPP approach is to avoid a huge search space due to the fact that some combination would not be considered. However, it will
lose the global optimization, so some queries should be rewritten by using common subexpression among queries to gain more optimal query processing cost.

The second task to solve the materialized view selection is to design the selection approach to select the appropriate set of views to be materialized. Normally, the materialized view selection approach, that is based on a fixed set of queries, frequency of executing the query and frequency of updating the base relation assumed to be fixed and time invariant, and a set of views to be materialized selected and computed from scratch, is known as Static Materialized View Selection. The objective of the static approach is selecting the set of views to be materialized from scratch with the minimum of query processing cost, or view maintenance cost, or summation of query processing and view maintenance cost.

However data warehouse is dynamic environment because the requirements specified by the various stakeholders are frequently changed and some queries are not known in advance to serve the business of the organization. Therefore, the existing materialized views derived from the static approach might be changed, and some new incoming queries could not be answered by the existing materialized views. Then, the new materialized views have to be added for answering the new incoming queries. On the other hand, the existing materialized views would be deleted if the query frequency of existing query is significantly decreased, or the existing queries are deleted due to no longer required.

In case if we use the static approach to resolve new requirement, we need to rerun the static materialized view selection method for all requirements that are existing and new requirements. From this context, there are some disadvantages to recalculate the new materialized views from scratch. First, re-computing from scratch is a waste of the existing resources because not the whole existing resources such as queries, materialized views are affected. Second, processing time for selecting views from entirely view sets is dissatisfied. Thus, our objective of dynamic materialized view selection is to minimize query processing and view maintenance cost based on the existing resources and without recalibrating from scratch. This is known as the Dynamic Materialized View Selection (Theodoratos and Sellis, 1999: 2; Zhang, Yang and Karlapalem, 2003: 451; Lawrence and Rau-Chaplin, 2008:48). Therefore, the main problems of dynamic materialized view selection are; i.e. how to identify which
existing resources are affected due to changed requirements, how to identify the new set of views to be materialized and the existing materialized view to be un-materialized.

1.2 Dissertation Objectives

The dissertation objective is developing the alternative methodology of dynamic materialized views selection to support the changing of requirements based on the existing resources and avoid re-computing from scratch. The MVPP is the structure for search space. The MVPP re-optimization algorithm is proposed to improve the total query processing cost of the cheapest MVPP by rewriting the query using global common subexpressions as supposing that the more optimal query processing cost the less total cost. For dynamic materialized view approach, after new requirements are merged into the existing resources, we apply our algorithm to identify the affected nodes aiming to reduce the search space in the selection set of views to be materialized step. We use Two-phase optimization (2PO) algorithm, the combination of Iterative Improvement (II) and Simulated Annealing (SA), to select the set of views to be materialized because 2PO provides minimum total cost (Phuboon-ob and Auepanwiriyakul, 2007: 166) that is the summation of query processing cost and materialized view maintenance cost. The assumption is that the application requirements of the designed data warehouse are changed frequently.

1.3 Organization of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 provides review the background and related works mainly focusing on lineage model to construct the search space, common subexpression and dynamic materialized view selection approach. Chapter 3 clarifies our proposed methodology to optimize MVPP and dynamic materialized selection approach in details. Chapter 4 provides the design of experiments, result and analysis. Finally, chapter 5 discusses the conclusion of our experiments and future works.
CHAPTER 2

LITERATURE REVIEW

One of data warehouse designing problems is the view selection problem. The problem is how to select a set of views to be materialized aiming to optimize the summation of materialized view maintenance cost and query processing cost for all queries. As these two costs are in conflict, an approach to ensure a balance between materialized view maintenance and query processing costs is taken into account. Materializing an appropriate set of views and answering queries using these materialized views can significantly speed up the processing of query as processing the query using materialized views will be faster than re-computing the virtual views. Therefore, materializing all the input queries can achieve the lowest query processing cost but the highest view maintenance cost. As materialized views have to be maintained in order to keep them consistent with the base relations. The objective to resolve this problem is to select the set of views to be materialized which minimizes one or more constraint i.e. space constraint (Harinarayan, Rajaraman and Ullman, 1996: 205; Lawrence and Rau-Chaplin, 2008: 49), query processing cost, view maintenance cost, or sum of query processing and view maintenance cost (Yang et al., 1997: 140; 2003:454; Theodorators and Sellis, 1999: 4; 2000: 12; Phuboon-ob and Auepanwiriyakul, 2007: 168).

There are two major concerning tasks to resolve the materialized view selection problem. The first task is to design the lineage model for generating a search space. The second task is to design the optimization algorithm for selecting the appropriate set of views to be materialized from the search space. Moreover, for the dynamic selection problem, the other tasks which are the relevance among the views, new views and existing views, and which existing resources are affected have to be considered.
In this chapter, we review the background and related works on lineage models to construct search space, materialized view selection algorithms, dynamic materialized view selection approaches, cost model, common subexpression and weight of intermediate node in MVPP.

2.1 Query and View Lineage Models for Generating the Search Space

The appropriated data structure is considered for view selection in order to optimize the query cost, view maintenance cost, or both. The various well known frameworks have been proposed such as Lattice Framework (Harinarayan, Rajaraman and Ullman, 1996: 205; Kalnis, Mamoulis and Papadias, 2002: 89; Lawrence and Rau-Chaplin, 2008: 54), AND-OR DAG (Theodoratos and Sellis, 1999: 1; Gupta and Mumick, 2005:24; Sun and Wang, 2009: 1; Zhang, Sun and Wang, 2009: 316; Theodoratos and Sellis, 2000: 7; Theodoratos, Dalamagas, Simitsis and Stavropoulos, 2001: 327) and Multiple View Processing Plan (Yang, Karlapalem and Li, 1997: 136; Yang, Zhang, and Yao, 2001: 282; Zhang, Yang and Karlapalem, 2003: 270; Derakhshan, Dehne, Korn and Stantic, 2006: 89; 2008:121; Phuboon-ob and Auepanwiriyakul, 2007: 166).

2.1.1 Lattice Framework

Data warehouse typically have schemes that are designed for querying and analysis. To speed up the response times, data will be modeled and viewed in multidimensional data cubes (Harinarayan et al., 1996: 205). Each cell of the data cube is a view consisting of an aggregation of the interesting information. The dependencies among the views can be expressed by a lattice framework. In lattice, each node denotes a view or a query and edges represent the dependencies among the views or queries. Given two nodes view \( u \) and view \( v \), there is a path from \( u \) to \( v \) if queries on \( v \) can be answered by using only the result from \( u \) denoted by \( u \subseteq v \). The \( \subseteq \) operator imposes a partial ordering on the queries. The simple example of lattice framework in the time dimension: day, month, and year is shown in Figure 2.1.
The Data Cube Lattice is built from the queries involved in the data warehouse application. The Data Cube Lattice is a graph whose nodes represent queries or views which are characterized by the attributes of the ‘group by’ clause. The edges denote the derivability relation between views. The node labeled ‘none’ corresponds to an empty set of ‘group by’ attributes. For example in Figure 2.2, if there is a path from view $V_i$ to a view $V_j$, then grouping attributes on $V_j$ can be calculated from grouping attributes on $V_i$. The benefit of this representation is that a query can be used to answer other queries.

**Figure 2.2** Sample Data Cube Lattice Framework for Eight Views  
Source: Harinarayan et al., 1996:207.

### 2.1.2. **AND-OR DAG**

An expression AND DAG for a query or view $V$ is a directed acyclic graph with $V$ as a source node, no incoming edge, and base relations as sink nodes, no
outgoing edge (Gupta and Mumick, 2005: 24). Each node has a unique evaluation that mean if node $u$ has outgoing edge to nodes $v_1, v_2, ..., v_i$ then all of node $v_1, v_2, ..., v_i$ are required to compute node $u$. This dependency is indicated by drawing a semicircle, called AND arc. For example, Figure 2.3 (a) shows an expression AND-DAG. The node view $a$ is computed from set of views $\{b, c, d\}$.

An expression AND-OR DAG for a query or view $V$ is a directed acyclic graph with $V$ as a source node and base relations as sink nodes. Each node has associated with one or more AND arcs. More than one AND arc at a node determines the multiple ways of computing that node. Figure 2.3 (b) shows the example of an expression AND-OR DAG. The node view $a$ can be computed either from the set of views $\{b, c, d\}$ or $\{d, e, f\}$.

![Figure 2.3](image)

**Figure 2.3** (a) An Expression AND DAG  (b) An Expression AND-OR DAG

**Source:** Gupta and Mumick, 2005: 25.

An AND-OR DAG is constructed by integrating the expression of the previous AND-OR DAG. Let $G_{i-1}$ be the AND-OR DAG. $G_{i-1}$ is formed by merging the expression AND-OR DAGs represented by $D_1, D_2, ..., D_k$. Then $G_i$ be an AND-OR DAG is formed by merging $D_1, D_2, ..., D_k$ with $G_{i-1}$. The merging process involves: (1) matching node in $D_i$ with $G_{i-1}$ that represent same relational expressions, (2) for the unmatched node in $D_i$, identify whether it can be derived from a set of node in $G_{i-1}$, (3) if node in (2) can be identified, then merge $D_i$ to a set of node used to derive $D_j$. 
Figure 2.4 shows an example of a global plan, for the queries (R ⋈ S ⋈ T) and (R ⋈ S ⋈ U), Figure 2.4 (a) is AND DAG and Figure 2.4 (b) is AND-OR DAG.

**Figure 2.4** (a) An AND DAG (b) An AND-OR DAG

**Source:** Gupta and Mumick, 2005: 30.

2.1.3. **Multiple View Processing Plan (MVPP)**

The MVPP is a directed acyclic graph of the relational algebra for a set of queries. The MVPP is constructed by merging the individual optimal query plan with the shared common subexpressions between the queries (Yang et al., 1997: 138). The simple MVPP is shown in Figure 2.5.

**Figure 2.5** Multiple View Processing Plan (MVPP)

**Source:** Zhang, Yang and Karlapalem, 2003: 453.
The leaf node (the node that does not have edges coming into the node) represents the base relations. The root node (the node that does not have edges going out of the node) represents the queries. Each intermediate node is marked by a relational algebra operation and defined as a view. An edge exists between two nodes if the operator in the upper level is applied to the view derived by the operator in the lower level. The cost for each operation node is labeled at the right side of the node. The number of rows produced by this operation is labeled at the left side of the node. The frequency to access the queries are labeled on the top of the query node. For example, in Figure 2.5, the cost for obtaining tmp3 by using tmp1 and tmp2 is 36m. In Figure 2.5, k and m stand for one thousand and one million respectively.

2.2 Materialized View Selection Algorithms

The second task to solve the materialized selection problem is to design the optimization algorithms for selecting the appropriate set of views from the search space to be materialized. The algorithms for materialized view selection are classified into four majority categories: deterministic algorithms, randomized algorithms, evolutionary algorithms and hybrid algorithms (Yang, Zhang and Yao, 2001: 282; Zhou, Geng and Xu, 2011: 131).

**Deterministic Algorithms** is an algorithm which behaves predictably from the input. For a given certain input, the deterministic algorithm will always compute and give the same results. The deterministic algorithm searches a solution in a deterministic manner, and usually applies either by heuristics or by exhaustive search to construct a solution step by step (Zhang et al., 2001: 282). The following researchers apply the deterministic algorithm to the materialized view selection. Harinarayan et al. (1996: 205-216) proposed a greedy algorithm for static materialized view selection which focused on the conflict between the space and the average time to answer a query based on the concept of lattice framework. Shukla et al. (1998: 492) proposed PBS (Pick By Size) algorithms based on lattice frame work. Gupta (1997: 98); Gupta and Mumick (2005: 24) presented several heuristic algorithms; greedy algorithm, Inner-level greedy algorithm, AO-Greedy algorithm, $r$-level Greedy, Inverted-tree Greedy algorithm, and A* heuristic algorithm to select the set of views.
Randomized Algorithms perform a random walk through a search space via a series of moves (Ioannidis and Kang, 1990: 312). Each solution in the randomized algorithm is a state, which has a cost attached to it, in a search space i.e. a node in the graph. The states are connected by edges that are defined by a set of transitions from one state to another or neighbor state after applying a transformation. The algorithms perform a random walk along the edges according to certain rules, and terminate as soon as no more applicable moves exist or a time limit is exceeded. The goal of applying randomized algorithm is to find the state with the global minimum cost. Sometimes the heuristic algorithm is not general, so the randomized algorithm is an effective and general solution. The randomized algorithm may find a reasonable and approximate optimization solution in a relative short time. The general algorithms of randomized algorithm are Iterative Improvement (II) (Nahar, Sahni and Shragowitz, 1986:293), Simulated Annealing (SA) (Kirkpatrick, Gelatt and Vecchi, 1983: 671), Two-Phase Optimization (2PO) (Ioannidis and Kang, 1990: 312), Toured Simulated Annealing (TSA) (Lanzelotte, Valduriez and Zaït, 1993: 493) and Random Sampling (Galindo-Lengaria, Pellenkofdt and Kersten, 1994: 85). The following example, researches exploit the randomized algorithm for materialized view selection. Kalnis, Mamoulis and Papadias explored the application of four randomized algorithms: II, SA, RA and 2PO to the view selection problem in data warehouses under the space constraint and the maintenance cost constraint. They found that 2PO gives the best performance because it converges quickly to a good local minimum. Derakhshan et al. (2006: 89) proposed a materialized view selection using SA with MVPP. Their method provides a further significant improvement in the quality of the set of materialized views compared to the deterministic algorithm and genetic algorithm. They extended their work by consider to speed up the computation a parallelization of SA (Derakhshan, Stantic, Korn and Dehne, 2008: 126). They proposed Parallel Simulated Annealing (PSA) that provided a significant improvement in the quality of the obtained set of materialized views over sequential simulated annealing algorithms. Theodoratos, Dalamagas, Simitsis, and Stavropoulos (2001: 325) applied SA for the
multiple query optimizations based on AND-OR DAG over the space and maintenance cost constraint.

**Evolutionary algorithms** use a randomized search method similar to natural biological evolution i.e. mutation, crossover and natural selection in their search for near-optimum solutions. Although an evolutionary algorithm resembles randomized algorithms in this aspect, the approach shows enough differences to warrant a consideration of its own. The basic idea is to start with a random initial population and generate offspring by random variations e.g. crossover and mutation. The “fittest” members of the population survive the subsequent selection; the next generation is based on these members. The algorithm terminates as soon as there is no further improvement over a period of time or after a predetermined number of generations. The first evolutionary algorithm developed for materialized view selection problem is Genetic Algorithms (GA) based on MVPP (Zhang and Yang, 1999: 116). Zhang and Yang proposed a complete approach, GA, to choose materialized views and demonstrate that it is practical and effective compared with heuristic approaches. Yu, Yao, Choi and Gou (2003: 458) apply GA to the materialized view selection problem under maintenance cost constrain. Besides GA, other evolutionary algorithms i.e. Ant Colony (Song and Gao, 2010: 534), Particle Swarm (Sun and Wang, 2009:1), Memetic (Zhang, Sun and Wang, 2009:315) and Shuffeled Frog Leaping (Li, Qian, Jiang and Wang, 2010: 7) were also proposed for the materialized view selection problem.

**Hybrid algorithms** combine the evolutionary and deterministic algorithms in various ways. Zhang et al. (2001:281) proposed several hybrid evolutionary and heuristic algorithms for optimizing global processing plans and materialized view selection. Their experiment shows that the hybrid algorithm delivers better performance than either the evolutionary algorithm or deterministic used alone in terms of the minimal query and maintenance cost. Zhou, Geng and Xu (2011: 134) proposed the hybrid algorithm that combination of GA and SA, GA and ant colony to achieve the queries response time and maintain cost without space constraint.

The researches mentioned above are focused on the static materialized view selection. Static materialized view selection processes all queries together once at the beginning. However, if new queries are needed or existing queries are modified, all
queries have to be processed again, not simply those parts that have been modified. In the next section, we review the dynamic materialized selection approach which solves the limitations of static approach.

2.3 Dynamic Materialized View Selection Methodologies

There are some approaches that have been proposed to solve the dynamic materialized view selection problem. The variant appropriated lineage models are selected to be the search space such as Lattice Framework (Lawrence and Rau-Chaplin, 2008: 54), AND-OR DAG (Theodorators and Sellis, 1999: 1; 2000: 7; 2001: 325), and MVPP (Zhang, Yang and Karlapalem, 2003: 451). The selection algorithms are applied for selecting the set of views to be materialized such as Genetic Algorithm (Zhang, Yang and Karlapalem, 2003: 451), Randomized Algorithm (SA, II, 2PO) (Theodoratos, Dalamagas, Simitsis and Stavropoulos, 2001: 325; Lawrence and Rau-Chaplin, 2008: 47) and Deterministic Algorithm (Theodorators and Sellis, 1999:1; 2000: 7; Lawrence and Rau-Chaplin, 2008: 47). There are various situations for changing requirements taken into account such as materializing only new virtual views once the new queries are added, un-materializing the existing materialized views when they are no longer used, or materializing new views and un-materializing existing materialized views simultaneously. In this section, we review the researches for the dynamic materialized view selection.

Kotidis and Roussopoulos (1999: 372-376) proposed a system called DynaMat. DynaMat consolidated the view selection and materialized view maintenance problem under a single framework in order to minimize the space availability and available downtime of the system. They modeled this system by caching fragments of queries or views. The fragment is a portion of the query that results from a range selection on its dimension. DynaMat supports the following three types of the multidimensional range queries. First is a full range; a full range means the value of dimension \( d \) is between the minimum value and maximum value. For instance of dimension \( d \), its dimension value is between minimum value i.e. 1 and maximum values i.e. 50. Second is a single value \( d \), such as dimension \( d=20 \). Third is an empty range which means the dimension \( d \) is not presented in the query. DynaMat
includes two operational phases. The first phase is the on-line query answering in which DynaMat answers the incoming queries using fragment locator. Fragment locator determines whether the materialized results can be efficiently used to answer the query or not. DynaMat constantly monitors incoming queries and materializes the best set of views. The second phase is the updating phase, DynaMat reconciles the current materialized view selection and refreshes the most beneficially subset of it within a given maintenance window. The concept of Dynamat is to partitions dimensional data into fragments thus DynaMat supports some types of queries not wide ranges of queries. Therefore, the total cost of DynaMat will be high because of the cost to find a set of fragments to answer the query is high.

Theodorators and Sellis (1999: 1; 2000: 7) proposed the incremental designing of data warehouse that the new set of views has been selected to answer the new queries and restricted within the allocated extra space. They used AND-OR DAG as the search space structure. The approach subjects to minimize the total cost of evaluation cost of the new queries and view maintenance. This approach considers only the new incoming queries. All new queries can be answered by using the existing and the new materialized views. New queries will be rewritten using the existing materialized if new queries can be totally answered by the existing materialized views. Selecting a new set of views to be materialized views occurs when the new incoming queries cannot totally be answered by the existing materialized views. Then the AND-OR DAG for incoming queries is constructed from base relations and existing materialized views. The r-greedy algorithm (Theodorators and Sellis, 2000: 23) and Simulated Annealing (Theodoratos, Dalamagas, Simitsis and Stavropoulos, 2001: 326) are used for the selection algorithm.

Zhang, Yang and Karlapalem (1999: 247; 2003: 451) proposed the dynamic materialized view selection framework based on MVPP developed by Yang et al.(1997: 136). This method considers which existing queries are affected when the environment changes (Zhang and Yang, 1999: 252). The environment changes consist of three scenarios. (1) Adding new queries or deleting the existing queries, (2) the definition of existing query is changed, (3) the query frequency or updating frequency is changed. Once changes mentioned above occur, some existing queries would be identified as either directly or indirectly affected. The directly affected queries can be
identified by considering the existing queries whose intermediate node overlap with the changed queries. The indirectly affected queries are the existing queries whose intermediate nodes overlap with the directly affected queries.

**Figure 2.6** Example of the Affected Transference

**Source:** Zhang and Yang 1999: 253.

For example MVPP in Figure 2.6 shows the effect to existing queries when the new query Q1 is added. The filled triangle is denoted the existing materialized view node. The node tmp1 is used to construct the incoming query Q1 and the existing query Q2. So, tmp1 is the overlapping node between new incoming query and the existing query. Therefore, query Q2 is identified as the directly affected query. The indirectly affected queries are considered as following. After new query Q1 is added, the weight of tmp1 will increase then tmp1 may be chosen to be materialized rather than tmp3 because weight of tmp1 is greater than that of tmp3. The changing of tmp3 to be un-materialized affects to tmp2 because if weight of tmp3 is less than that of tmp2 then tmp2 will be changed from virtual view to materialized view. So, tmp3 is the overlapping part with the query Q2 and Q3 then Q3 would be determined as indirectly affected query. Therefore, for this example, all nodes used for constructing the affected queries either directly or indirectly are the set of candidate nodes to be selected to be either materialized or un-materialized.
Lawrence and Rau-Chaplin (2008: 47) proposed method for dynamic materialized view selection for OLAP under space constraint and available time for computing new views. They considered both adding new materialized views and discarding the existing materialized views. The objectives of this approach were to minimize the sum of query processing cost and materialized view maintenance cost under a space constraint. They developed a classical BPUS (Benefit Per Unit Space) to derive a greedy solution and three Randomized algorithm (SA, II, 2PO) based on Lattice framework. Although their result shows that BPUS overcome the three Randomized algorithms, the computation becomes impractically large when the number of dimensions grows so the Randomized algorithm offers an attractive alternative approach. They implemented BPUS and Randomize algorithm for both startup and online phase. The startup phase is a static selection of views to be materialized. The online phase is selecting a new set of views by discarding some views from startup phase, and adding new ones to be materialized because of the space constraint. Supposing $M$ is an existing set of materialized views selected by the startup phase and $M'$ is a new set of materialized views selected by the online phase. In the online phase, once the situation violates the search space constraint during add new materialized view and delete the existing materialized view, their proposed method to solve this situation was (1) the previously removed view is added back to the solution and continue only removing views in $M'-M$ until the space constraint is satisfied (2) the removed view is re-added, and randomly selected view in $M'-M$ is removed instead.

Xu, Theodoratos, Zuzarte, Wu, and Oria (2007: 55) presented the dynamic selection problem as the shortest path problem on DAG. The input of the problem is a sequence of queries and updated statement. Their approach constructed candidate views dynamically by considering common subexpressions of queries and/or views. They used a heuristic algorithm to determine a candidate set of views and to decide when the materialized views are created or dropped during the execution of statements. The cost model has been applied to find the minimum total cost which is the sum of processing cost and maintenance cost. This approach works properly for the applications where the workloads, the queries and update statements, are executed in sequence. For example, some routine queries are given during the day time of every
weekday for daily reports; some analytical queries are given during the weekend for weekly reports; during the night the data warehouse is updated in response to update statements collected during the day.

Gong and Zhao (2008: 391) proposed the clustering method for view selecting and dynamic materialized view adjustment when adding new queries. Normally, a materialized view relates to a SQL statement so materialized views are corresponded to the result of SQL statement execution. Therefore, the materialized view can be classified as a class. The classifying method is to calculate the similarity of incoming SQL statement. The similarity value has to be higher than the decided similarity threshold. They determined that query sentence belong to the select, project and join structure without subquery statement. They defined the criteria and similarity function to judge the similarity between two queries. They used PBS (Pick By Size) algorithm for selecting the initial set of materialized views which are the input for generating the initial clusters. The number of clustering is not determined. It will be produced increasingly in the clustering process. Another given parameter is the similarity threshold value. If the similarity value is higher than the similarity threshold then the materialized view will be in that cluster. For dynamic phase, when the new queries are added, each query will be classified into the cluster according to the similarity function. After the query is identified into the cluster, the cost function which includes query processing cost and view maintenance cost is calculated. If the space constraint is reached, the existing materialized views, less frequently accessed, are replaced by the new materialized views.

2.4 Cost Model for Materialized View Selection

The minimization of a cost function is the main objective for both static and dynamic materialized view selection problem. Normally, there are two parts in cost function, the query processing cost and the materialized view maintenance cost. The query processing cost and materialized view maintenance cost are in conflict. To achieve low view maintenance cost by accessing the source base relation directly many times for multiple queries that have sharable subexpression, the query processing cost is high. To achieve low query processing cost by materializing all the
input queries, the materialized view maintenance cost is high. So, the combination of
the query processing cost and materialized view maintenance cost is the optimal
strategy. The well known cost model used for this research was introduced by Yang et
al. (1997: 139-140), and it was exploited by some researches (Zhang and Yang, 1999:
140; Gupta and Mumick, 2005: 26; Phuboon-ob and Auepanwiriyakul, 2007: 167;
Derakhshan et al., 2006: 91, 2008: 126) for evaluation of their approach. The cost
model is described in following sections.

2.4.1 Query Processing Cost

For each query, the cost of query processing is query frequency multiplied by
the cost of query access from the materialized nodes. Cost of query access is the
number of rows in the table to answer q.

Let $M$ is the set of materialized views,

$Q$ is the set of queries,

$f_q$ is the frequency of executing queries,

$C_q(M)$ is the cost to compute $q$ from the set of materialized views $M$.

Then the total query processing cost is:

$$C_{query\text{processing}} = \sum_{q \in Q} f_q C_q(M)$$ (1)

For example, consider query Q4 in Figure 2.5 and suppose that node tmp3 is
materialized view. The frequency of executing the query Q4 is 10. If node tmp3 is not
materialized, this query accesses the nodes named tmp1, tmp2, tmp3, and result1. The
cost of each node is 1k, 12m, 36m, and 36k respectively. So the query processing cost
for Query Q4 is 10*(1k+12m+36m+36k). If tmp3 is materialized, the query
processing cost for Query Q4 is 10*36k. It would be beneficial to materialize them,
reducing the processing cost from 10*(1k+12m+36m+36k) to 10*36k.
2.4.2 View Maintenance Cost

The maintenance cost for the materialized view is the cost for the process of updating a materialized view in response to the changes in the base relation.

Let $M$ be the set of materialized views,

$f_u$ is the frequency of updating base relations,

$C_m(v)$ is the cost of maintenance when $v$ is materialized.

Then the total maintenance cost is:

$$C_{maintenance} = \sum_{v \in M} f_u C_m(v) \quad (2)$$

For example, consider query Q4 in Figure 2.5 when tmp3 is materialized and suppose that the frequency of updating base relation equal to 1. This materialized view has maintenance cost whenever the updating of involved base relation occurs. The maintenance cost is the number of base relations multiplied by the cost of each node. There are two base relations: Item and Sale, and nodes: tmp1, tmp2, and tmp3 itself are related. Then the view maintenance cost is $2 \times (1k+12m+36m)$.

Our goal is the minimum total cost all feasible sets of materialized views. Therefore the total cost of materialized views $M$ is:

$$C_{total} = \sum_{v \in M} f_u C_m(v)$$

2.5 Common Subexpression

In order to generate the search space for multiple queries, common subexpressions among the queries have to be detected and exploited. Thus, the original queries will be rewritten using the global common subexpressions. Given a query, the optimizer will find an appropriate order for performing the relational algebra operations such that the query will be evaluated efficiently. The order in which the operations are performed prescribes the order in which the subexpressions
of the query are evaluated, and then their results are used to evaluate other subexpressions until the whole query is evaluated. A common subexpression is a subexpression that appears in more than one query.

The following SQL statements illustrate the examples of possibility sharable subexpression among three queries.

Q1:  
```
select p_brand, min(ps_availqty)
from part, partsupp, supplier
where s_suppkey = ps_suppkey
    and p_partkey = ps_partkey
    and p_type like 'BRASS%'
group by p_brand;
```

Q2:  
```
select s_nationkey, max(ps_supplycost)
from part, partsupp, supplier
where s_suppkey = ps_suppkey
    and p_partkey = ps_partkey
    and p_type like 'BRASS%'
group by s_nationkey;
```

Q3:  
```
select n_name, variance(ps_availqty)
from part, partsupp, supplier, nation
where s_suppkey = ps_suppkey
    and p_partkey = ps_partkey
    and p_type like 'BRASS%'
    and s_nationkey = n_nationkey
group by n_name;
```

Considering the query Q1 and Q2, both queries have the same join base relations i.e. PART, PARTSUPP, SUPPLIER, the difference of Q1 and Q2 is group by on difference attribute: Q1 group by on p_brand whereas Q2 group by on s_nationkey. The third query, Q3, looks similar to the first two queries except that it has an additional join with relation NATION and group by on attribute n_name of NATION. All three queries have the same selection predicates which is (p_type like ‘BRASS%’). A traditional query optimizer would optimize the three queries separately and generate an execution plan for each of the queries. It is obvious that execution plan of those three queries have sharable subexpressions. Therefore, execution times could be reduced by using sharable intermediate results instead of recomputing conjunctively join of those three base relations three times. For this
example (s_suppkey = ps_suppkey) and (p_partkey = ps_partkey) and (p_type like ‘%BRASS%’) is one of possibly common subexpression of among queries.

The concept of common subexpression of queries was introduced by Finkelstien (1982: 235). They used concept of common subexpression to solve the multiple queries optimization problem. Their approach was based on the idea of building the multiple query optimizations on top of the current single query optimizers. In this approach, a single query optimizer generates one optimal plan for each query. A plan merger, another component in the system, will examine all the plans and merges them to generate a global execution plan. This global plan is derived from the shared temporary results of the common parts of the queries. This approach, however, may not guarantee the optimal global cost because it may miss some other plans, which are not necessary optimal for each query, that contain more common subexpression with other queries. The common subexpression, firstly, was focused on the subexpression identification, and later included subsumption and overlapping of selection condition (Chen and Dunham, 1998: 493). Chen and Dunham used multi-graph to represent the select-project-join (SPJ) operations and used the heuristic method for selecting common subexpression to be the global execution plan. Their approach covered the case for identical, subsumption and overlap of SPJ operation. The general term of common subexpression was described (Lehne, Cochrane, Pirahesh and Zahatioudakis, 2001: 391) as the sharable subexpression between the queries that can be used for rewriting the queries either completely or partially. Thus their original queries could be rewritten by using the given common subexpressions after the common subexpression between a pair of queries was constructed. Zhou, Larson, Freytag and Lehner (2007: 533) proposed the algorithm that exploited common subexpression for multi-query optimization and materialized view selection in a conventional database. They presented a comprehensive mechanism for detecting sharable subexpression and constructing candidates covering subexpression that cover a set of similar subexpressions. Theodoratos and Xu (2006: 75) proposed the technique called closest common subexpression derivator for constructing candidate views to be materialized. Once closest common subexpression derivators between the queries were determined, the queries were rewritten by using the closest common subexpression.
2.6 Weight of Node in MVPP

For MVPP structure, the positive weight of node defines the possibility of intermediate node to be materialized (Zhang et al., 2003: 454). The weight of node is represented by following formula.

\[
w(v) = \sum_{q \in O_v} \left\{ f_q(q) \ast \left( C^q_a(v) \right) \right\} - \sum_{r \in I_v} \left\{ f_u(r) \ast \left( C^r_m(v) \right) \right\}
\]

\(O_v\) denotes the queries which use view \(v\).

\(I_v\) denotes the base relations which are used to produce view \(v\).

\(C^q_a\) denotes the accessing cost \(a\) for query \(q\) using view \(v\). The cost of answering query \(q\) is the number of rows presented in the relation used to construct \(q\).

\(C^r_m\) denotes the maintenance cost \(m\) for materialized view \(v\) based on relation \(r\), which is occasionally updated.

\(f_q\) denotes the frequency of executing a query.

\(f_u\) denotes the frequency of updating a base relation.

\(w(v)\) denotes the weight of a node, the higher of weight the more likely the node will be materialized. For example Tmp3 in Figure 2.5, Tmp3 is constructed on two base relations Item and Sales, The frequency of updating each base relation is 1. Tmp3 is derived by node named Tmp1, Tmp2 and Tmp4 itself. The cost of each node is 1k, 12m and 36m, respectively. Tmp3 is accessed by query Q2, Q3 and Q4. The frequency of executing query for each query is 2, 1 and 10, respectively. Therefore, the weight of Tmp3 is calculated as; the first part \(2\)(36m) + \(1\)(36m) + \(10\)(36m), the second part \((1)(1k+12m+36m) + (1)(1k+12m+36m)\) that is equal to \((2+1+10)(36m) + (2)(1k+12m+36m)\).
The weight of node is exploited to determine how nodes are affected by each other in the process of materialization as following rules (Zhang, C. and Yang, J. 1999: 251).

Rule 1: when \( v_1 \) is a descendant of \( v_2 \), and the static weight \( w(v_1) > w(v_2) \), if \( v_1 \) cannot be materialized, then \( v_2 \) will not be materialized.

Rule 2: if \( v_1 \) is a descendant of \( v_2 \), and the static weight \( w(v_1) > w(v_2) \), \( v_1 \) and \( v_2 \) are supporting the same queries, and \( v_1 \) is materialized, then there is no gain to materialize \( v_2 \).

Rule 3: If \( v_1 \) is an ancestor of \( v_2 \), and the static weight \( w(v_1) > w(v_2) \), \( v_1 \) and \( v_2 \) are supporting the same queries, if \( v_1 \) is materialized, then \( v_2 \) shall not be materialized.

According to the previous works in this chapter and to the best of our survey, all of the related works have not implemented the dynamic materialized view selection using 2PO on MVPP structure, and not mentioned the methods to optimize the MVPP. In the next chapter, we focus on the methodology employed in our dissertation.
3.1 Proposed Methodologies

In this chapter, we discuss our proposed approach to solve the dynamic materialized view selection problem based on MVPP. The Iterative Improvement combined with Simulated Annealing called Two-Phase Optimization (2PO) is the optimization algorithm for selecting a set of views to be materialized. As the MVPP generated by Yang et al. (1997: 138) will lose the global optimization then we also propose the algorithm to optimize MVPP aiming to have more optimal MVPP. For the dynamic materialized view selection, we propose the technique to identify which existing resources are affected due to changing requirements, and to determine the new set of views to be materialized and existing materialized view to be un-materialized. Our goal for dynamic materialized view selection is to minimize the total cost which is summation of query processing cost and materialized view maintenance cost based on the existing resources without recalibrating from scratch.

Therefore, our proposed methodology to solve the dynamic materialized view selection problem includes two parts:

1. The optimization task to improve the MVPP which is a lineage models for search space for the materialized view selection.
2. The approach for Dynamic Materialized View Selection.

We use MVPP as the lineage model to generate the search space because MVPP can present the realistic SQL queries and large number of queries. We propose an algorithm to optimize the cheapest MVPP by rewriting the query using common subexpression. The details for building the cheapest MVPP are described in section 3.2. The details of our MVPP re-optimization algorithm are described in section 3.4, which uses the concept of a common subexpression that is described in section 3.3.
The proposed approach for dynamic materialized view selection is presented by diagram in Figure 3.1. The approach consists of two phases, the static phase and dynamic phase.

**Figure 3.1** The Diagram for Dynamic Materialized View Selection Approach

Static phase, the first step in diagram, is Static Materialized View Selection approach described in section 3.5. The static phase is to generate the re-optimized MVPP of the initial requirements that will be the initial search space for the dynamic phase once the new requirement occurs.

Dynamic phase, the second step to the fourth step in the diagram, is the Dynamic Materialized View Selection. The second step is to merge new requirements into existing MVPP. When the new requirements are added, the characteristic of new requirements might impact to the existing MVPP structure in a different situation i.e. adding the new queries, deleting the existing queries, changing the definition of existing queries, changing the frequency of executing query or frequency of updating base relations. The scope of our new requirements includes adding new queries and
deleting existing queries. For changing the definition of existing queries, we implement by deleting the existing query and adding query with a new definition. Changing the frequency of the executing query or frequency of updating base relation is not our scope because it does not impact to the MVPP structure. We will provide the analysis of various situations that impact to the existing MVPP structure and the merging new requirement into the existing MVPP in section 3.7.1.

The third step is to determine the existing resources in search space that are impacted by the new requirements. The objective of this step is to reduce the search space. The details of our algorithm to identify the affected nodes are described in section 3.7.2. The affected resource in our research is the intermediated nodes that are affected rather than the queries that are affected proposed by Zhang, Yang and Karlapalem (2003: 455). In section 4.6.1.6, we validate our assumption that to identify the query that is affected will provide too much number of nodes as a member of a set of views for the selection step.

Finally, the fourth step is to select set of views to be materialized. The input of the third step will be mapped into a binary string before being input to 2PO. We use 2PO as the selection algorithm because 2PO provides a minimal total cost comparing to the Deterministic algorithm, Simulated Annealing, Genetic Algorithm, and Hybrid algorithm (Phuboon-ob and Auepanwiriyakul, 2007: 169). The result from 2PO would be an appropriate new set of views to be materialized. Some existing materialized views identified as the affected nodes might be un-materialized to the virtual view if they are not frequently used by the stakeholders. The aiming of the selection step is the minimal total cost which is the summation of query processing cost and materialized view maintenance cost. The details of our 2PO and cost model are described in section 3.6 and 3.9 respectively.

After the set of views are materialized, the existing search space will be the new search space to support the next round of changing requirements. The “Existing MVPP” in diagram illustrates that the existing search space will be adjusted once the requirements are changed. Therefore, for each round of dynamic phase, the existing MVPP structure will be changed by the previous round.
3.2 Multiple View Processing Plan (MVPP)

The MVPP defined by Yang et al. (1997: 138) is a directed acyclic graph that presents the query processing plan of a set of queries. A simple MVPP is shown in Figure 3.2.

![Figure 3.2 The Simple MVPP of Three Queries Q1, Q2 and Q3](image)

Suppose that the root node is the node that does not have edges going out of the node representing the query, the leaf node is the node that does not have edges coming into the node representing the base relation, and the intermediate node is the representing operation. A link exists between two nodes, if the operator in the upper level is applied to the result derived by the operator in the lower level. Each intermediate node in MVPP is marked by a relational algebra operation and the cost for processing the operation. Two numbers are associated with each intermediate node. The number of rows needed to be read by the operation is labeled on the right side and the number of rows generated by the operation is labeled on the left side. The frequency of executing the query is labeled on the top of the query. Because of the 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

---

**Figure 3.2** The Simple MVPP of Three Queries Q1, Q2 and Q3
environment. The algorithm which is used to construct the MVPP is described in Figure 3.3. The query cost, in step 2, is the cost of accessing the query node, for example Q1 in Figure 3.2 the query cost is 910519. The total query processing cost, in step 7, is the summation of query processing cost of all queries that is mentioned in section 3.9.

begin
1. For every optimal query processing plan, if there is a join operation involved, push all the select, project operations 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 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 select, project and aggregate functions as deep as possible.
7. Calculate the total query processing cost of each MVPP, and select the one which gives the lowest cost.
end;

Figure 3.3 The Algorithm for Implementing the MVPP

3.3 Common Subexpression

Normally, the search space for a view selection problem is constructed by using all common or similar subexpressions among the queries. The concept of a common subexpression is initially referred to as an identical or equivalent expression, and later the term included expression subsumption. Thereafter, commonality between queries has included the possibility for overlapping the select condition.
There are four categories of commonality between the queries (Chen and Dunham, 1998: 3; Lehner, Cochrane, Pirahesh and Zahatioudakis, 2001: 393); (1) there is no sharable subexpression shown in Figure 3.4 (a), (2) identical, (3) totally overlapping is called subsumption that is a query $i$ is the intermediated query result for another query $j$ shown in Figure 3.4 (b), (4) partially overlapping that is a subtree of a query $i$ is also the subtree of query $j$ shown in Figure 3.4 (c). One query would have partially overlapping with many queries shown in Figure 3.4 (d-e). In Figure 3.4, (a) Q1 is constructed on base relations R1 and R2, while Q7 is constructed on R3 and R4, these two queries do not have sharable subexpression; (b) Q5 is the intermediate query result for Q6; (c) Q2 and Q3 has a subtree that is overlapping sharable subexpression; (d) and (e) shows that Q1 has more than one equivalent plan. Q1 has overlapping portion with Q4, meanwhile Q1 has alternative equivalent plan that has overlapping portion with Q6. After common subexpressions are detected, they are exploited to construct the global optimal equivalent plan for multiple queries processing plans. We use this concept of common subexpression to optimize the cheapest MVPP that is generated by the algorithm in section 3.2.

Figure 3.4 The Categories of Subexpression Commonalities
3.4 Proposed Approach to Improve MVPP

In general, to construct 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 (Yang et al., 1997: 142) can be adjusted to reduce the total query processing cost, as the method of merging described in section 3.2 does not consider the common subexpressions of among queries. In our approach, the queries in the cheapest MVPP whose query processing cost is more than the $n^{th}$ MVPP are taken into consideration. They would be rewritten by using concept of common subexpression. We match these queries with the existing sharable subexpressions in a bottom-up way. The MVPP re-optimization algorithm is described in the following sections.

3.4.1 The MVPP Re-Optimization Algorithm

The algorithm to optimize the cheapest MVPP is described in Figure 3.5. This approach is for rewriting the certain queries in the cheapest MVPP. It is used to optimize the MVPP for the static phase and the dynamic phase.

begin
1. Input = the cheapest MVPP.
2. Initial list $LV = \phi$.
3. $k = $ number of queries
4. for $i = 1$ to $k$
   
   Compare $Cq(i)$ of cheapest MVPP with $Cq_j(i)$ of other MVPPs.
   
   If $Cq_j(i)$ is less than $Cq(i)$ then
   
   insert $q(i)$ into $LV$.
5. For queries in $LV$, consider the possible commonalities with exists global equivalent plan as following:
   
   5.1 If there is nothing in common with global equivalent plan
   
   skip to the next query.
   
   5.2 If there is one or more overlapping portions,
   
   rewrite this query using exists common subexpression in
   
   MVPP in bottom-up way described in section 3.4.2
end;

Figure 3.5 The Proposed Algorithm: The MVPP Re-Optimization Algorithm
\( C_q(i) \) denotes the query processing cost of query \( i \) in the cheapest MVPP

\( C_q(j')(i) \) denotes the query processing cost of query \( i \) in the \( n^{th} \) MVPP.

If \( C_q(j')(i) \) less than \( C_q(i) \) imply that there is another optimal execution plan for query \( i \).

### 3.4.2 Rewriting the Query Using Common Subexpression

If a view \( V \) is defined as a common subexpression of a set of queries. Query \( Q \) is called a parent of the view \( V \) if it can be answered by using view \( V \). For example, Tmp4 in Figure 3.2 is a common subexpression of Q1, Q2 and Q3. Tmp4 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 (Halevy, 2001: 276). Suppose there is a set of views, \( V_m \), in MVPP, and a given query \( Q \) has the execution plan that doest not use \( V_m \). However, \( V_m \) can be used to answer the query \( Q \), then we can produce other execution plans of query \( Q \) by using the set of views \( V_m \) and/or base relations.

For example, Figure 3.6 illustrates the possible individual plans of query Q1 and MVPP of Q1, Q2 and Q3. Figure 3.6 (a) and (b) are the possible execution plans for Q1. The MVPP of Q1, Q2 and Q3 is shown in Figure 3.6 (c). In MVPP, Q1 is constructed by using Plan A as shown in Figure 3.6(a) that is \{Tmp10 \( \bowtie \) (PARTSUPP \( \bowtie \) PART)\}. The other possible plan for Q1 is shown in Figure 3.6 (b) that is \{((Tmp10 \( \bowtie \) PARTSUPP) \( \bowtie \) PART)\}.

For Plan A in Figure 3.6 (a), Q1 accesses nodes named Tmp1, Tmp2, Tmp3, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp12 and result1. The processing cost of each node is 800000, 200000, 160000000000, 5, 1, 25, 25, 10000, 50000, 1602400000 and 160240, respectively. The frequency of executing the query Q1 is 2. Therefore, the query processing cost of Q1 is \((2)^*(800000 + 200000 + 160000000000 + 5 + 1 + 25 + 25 + 10000 + 50000 + 1602400000 + 160240)\) that is 161,603,620,296.

For Plan B in Figure 3.6 (b), Q1 accesses nodes named Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp1, Tmp11, Tmp2, Tmp12 and result1. The processing cost of each node is 5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240, respectively. Therefore, the query processing cost of Q1 is
(2)*(5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 200000 + 
320480000000 + 160240) that is 67,303,240,592.

In Figure 3.6 (c) the query processing cost of MVPP, the summation of query processing cost of Q1, Q2 and Q3, is 1,176,430,280,644. However if we rewrite Q1 using Plan B, that is Tmp11 in Figure 3.6 (c) join with PART, the query processing cost of MVPP is 156,799,280,644 which is less than the original MVPP. So we should rewrite Q1 in MVPP by using Plan B.

(a) Plan A of Q1  
(b) Plan B of Q1  
(c) The MVPP of Q1, Q2 and Q3

Figure 3.6 The Plan of Q1 in MVPP of Q1, Q2 and Q3
In our approach, we rewrite the query by comparing the individual plan of that query with the common subexpression in MVPP. The query rewriting will be processed in a bottom-up way which is calculated from the base relations to the root of the equivalent plan. The process to rewrite the queries is shown in Figure 3.7.

1. Push select, project operation and aggregation function of optimal individual plan up along the tree
2. Match optimal individual plan of query with MVPP from base relation to the root of query.
3. Merge the query into MVPP
   3.1 If there are sharable subexpression then merge to the subtree which provides the number of base relations that are joined conjunctively as much as possible.
   If merged query has select operation same as conjunctively joined node then push down select operation.
   3.2 If no sharable subexpression then create new conjunctively joined node for each subtree.
4. Push down select project operation and aggregation function as deep as possible.

Figure 3.7 The Rewriting Query Steps

3.5 Static Materialized View Selection

The purpose of the static materialized view selection is to generate the initial MVPP which is constructed from the initial requirements. The initial MVPP is generated by using the approach described as follows. First we use an algorithm proposed by Yang et al. mentioned in section 3.2 to generate the cheapest MVPP. Next, we apply our MVPP re-optimization algorithm described in section 3.4 to
optimize the cheapest MVPP. After the re-optimized MVPP is generated, the selection algorithm, which is the Two-Phase Optimization (2PO), is applied to select the set of views to be materialized. The final output of the static phase will be the initial MVPP structure with materialized view nodes. The steps of static phase is shown in Figure 3.8.

1. generate the cheapest MVPP introduced by Yang et al. described in section 3.2
2. optimize the cheapest MVPP by applying our MVPP re-optimization algorithm described in section 3.4
3. apply selection algorithms, Two-Phase Optimization, to select set of views to be materialized.

**Figure 3.8** The Static Materialized View Selection for the Static Phase

### 3.6 Selection Algorithm: Two-Phase Optimization (2PO)

In this section, we explain the Two-Phase Optimization which is the selection algorithm to select the set of views to be materialized. The Two-Phase Optimization (2PO) is the combination of Interactive Improvement (II) and Simulated Annealing (SA) (Ioannidis and Kang, 1990: 313).

**Iterative Improvement (II)**

The II algorithm, which is the randomized algorithm, starts with the initial randomly chosen state and performs random downhill moves until it reaches a local minimum. After a local minimum has been reached a new start state is generated randomly. This process is repeated until a stop criterion is reached, and then the lowest local minimum encountered is the result. The stop criterion is set to 10 local minimum calculated. The II algorithm is shown in Figure 3.9.
begin
  $S_{\text{min}} = S_\infty$; \{initial solution\}
  while not (stopping condition) do {
    $S$= random state
    while local minimum not reached do {
      $S'$ = random state in neighbor($S$)
      if $\text{cost}(S') < \text{cost}(S)$ then $S$ = $S'$
    }
    if $\text{cost}(S) < \text{cost}(S_{\text{min}})$ then $S_{\text{min}} = S$
  }
  return($S_{\text{min}}$)
end;

Figure 3.9 The Iterative Improvement (II) Algorithm

Simulated Annealing (SA)

The SA is a local search algorithm same as II but SA tries to leave from local minima by using random moves. It accepts the uphill moves to a neighbor. The initial state is randomly generated and moves to a neighbor with the lower cost, downhill move, similar to II, but it also accepts an uphill move with some probabilities. The SA algorithm is shown in Figure 3.10.

begin
  $S = S_0$; \{initial state\}
  $T = T_0$; \{initial value of time limit\}
  $S_{\text{min}} = S$;
  while not(time limit) do {
    while not(local minimum($S$)) do {
      $S'$ = random state in neighbor($S$)
      $\Delta C = \text{cost}(S') - \text{cost}(S)$
      if ($\Delta C \leq 0$) then $S$ = $S'$
      if ($\Delta C > 0$) then $S$ = $S'$ with probability $e^{\Delta C/T}$
      if $\text{cost}(S) < \text{cost}(S_{\text{min}})$ then $S_{\text{min}} = S$
    }
    $T$ = reduce($T$)
  }
  return($S_{\text{min}}$)
end;

Figure 3.10 The Simulated Annealing (SA) Algorithm
Once the neighbor cost is greater than the previous cost, the uphill move can be accepted with the probability that decreases exponentially with the ratio of the change in cost $C$ to a parameter time limit, $e^{-\Delta C/T}$, $\Delta C$ is the difference between the cost of the new state and the previous one. In the inner loop, the algorithm gradually decreases value $T$ for allowing SA to accept uphill moves as SA accepts the new solution which is that the new cost can be less than or equal to the previous cost with the non-zero probability. The probability decreases exponentially with the ratio of the change in cost to a parameter time limit, $e^{-\Delta C/time\_limit}$. For our experiment, the value of each parameter for SA is set similar to Phuboon-ob and Auepanwiriyakul (2007: 166) including the time limit which is set to 90 at the starting point, and the decrement factor for the exponential is set to 0.7.

Two-Phase Optimization Algorithm (2PO)

2PO combines both II and SA. It begins by running II to find a good local minimum, and then applies SA to search for the global minimum from the state found by II. Since the output from the MVPP is a DAG, so we map a DAG into a binary string. Given the MVPP, we first map a DAG into a binary string of 1s and 0s to represent views which will and will not be materialized, respectively. The initial binary string of each node is set to 0 indicated that all intermediate nodes are virtual view for the starting state. The algorithm is presented in Figure 3.11.

```
begin
  1. Input the MVPP represented by a DAG
  2. Use depth first search from root nodes to base relations to search through all of the nodes in the DAG.
  3. Produce the sequence of nodes into a binary string.
  4. Call Iterative Improvement
  5. Call Simulated Annealing algorithm
  6. Present set of views to materialized with minimum cost
end;
```

**Figure 3.11** The Materialized View Selection with 2PO
3.7 Dynamic Materialized View Selection Approach

3.7.1 MVPP Structure Analysis and Merging New Requirements Approach

There are several kinds of requirement due to the changing of application requirements. The new requirements might impact to the changing of existing MVPP structure. The requirement can be classified into four situations (Zhang, Yang and Karlapalem, 2003: 454)

1. The existing queries are deleted.
2. The new queries are added.
3. The definitions of existing query are changed.
4. The frequencies of executing query and/or the frequencies of updating base relation are changed.

Those above situations are the cause of the possible changing of MVPP structure as follows:

1) The MVPP will be changed for the first and second situation.
2) The third situation, MVPP structure may or may not be changed depending on the selection criteria. For instance in Figure 3.2, condition of Tmp1 is changed from year=’1995’ to year=’2000’ then MVPP structure is not changed whereas the query accessing cost of Q1, Q2 and Q3 maybe changed due to the number of rows returned filtered by the selection condition. In our research, changing query definition is implemented same as deleting the existing query and adding the new query with the new definition.

3) The fourth situation, changing the frequency of executing query or frequency of updating base relation, the query processing cost and maintenance cost always changed accordingly. Therefore, there is possibility for the existing materialized view to be un-materialized and the virtual view to be materialized. The existing MVPP structure is not changed as neither new node is created nor existing node is deleted. However, when we rerun the MVPP algorithm to generate the cheapest MVPP, the MVPP structure may not the same structure as previously generated. Because the order of merging queries may have changed depending on the frequency of executing the query multiply with query cost. So the available sharable
subexpressions for incoming merged query are changed due to an available sharable subexpression of previous merged queries that are changed. For our research, as we apply the MVPP re-optimization algorithm to the cheapest MVPP then we re-write the certain queries by using global sharable subexpressions that share for all queries. So, after applying our MVPP re-optimization algorithm, the re-optimized MVPP structure will be same as previously generated. However whenever the frequency of executing queries is changed, the query processing cost of the re-optimized MVPP also changed.

In case, the subtree of the existing query is the subsumption of the subtree of the new query then the existing query can use the subexpression that is shared with the new query. For example, the new query Qc shown in Figure 3.12 (a) is added into the existing MVPP shown in Figure 3.12 (b). Considering Tmp5 in Figure 3.12 (b) and given c represents the selection algebra (o_orderdate >= ‘1994-01-01’ and o_orderdate < ‘1995-01-01’); Tmp5 is the subgraph of existing MVPP and Tmp5 is subsumption of (σc ORDERS ⊢ LINEITEM) that is Tmp4 in Figure 3.12 (a). So, the conjunctively joined of (σc ORDERS ⊢ LINEITEM) is sharable subexpression for Qc and {Qa, Qb}. However, the existing MVPP does not have this conjunctively joined node because Tmp5 in the existing MVPP, Figure 3.12 (b), is the conjunctively joined of {σcORDERS ⊢ (σI_commitdate<l_receiptdate LINEITEM)}.

Figure 3.12 The Subgraph of the Existing MVPP is the Subsumption of the Subtree of New Query
Because the existing MVPP does not have this sharable subexpression for Qc then we have to create a new node, (σc.ORDERS ≡ LINEITEM) to support Qc. If Qa, Qb still use Tmp5 then (σc.ORDERS ≡ LINEITEM) will be created specific for Qc only. However, if we rewrite Qa, Qb by using (σc.ORDERS ≡ LINEITEM) then (σc.ORDERS ≡ LINEITEM) can be shared for all queries that the saving of the materialized view maintenance cost.

To serve this situation there are some approaches such as all select operation in existing MVPP are pushed up before we merge new queries into existing MVPP or rewriting the existing queries constructed by that node. For push up method, all exiting materialized view are affected then the queries use those materialized views

<table>
<thead>
<tr>
<th>Begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. For every optimal query processing plan for any query, if there is a join operation involved, push select, project operations and aggregate function up along the tree.</td>
</tr>
<tr>
<td>2. Create a list of the new queries in descending order based on the result of their query access frequency multiplied by query cost.</td>
</tr>
<tr>
<td>3. Merge the new query in the list into existing MVPP according to their order in the list by comparing as follows:</td>
</tr>
<tr>
<td>3.1 If there is sharable subexpression available for new query then</td>
</tr>
<tr>
<td>merge to that sharable subexpression</td>
</tr>
<tr>
<td>If sharable subexpression has select operation and new query also has the same select operation then</td>
</tr>
<tr>
<td>push down select operation of new query</td>
</tr>
<tr>
<td>3.2 If no sharable subexpression then create new node for new query</td>
</tr>
<tr>
<td>4. Repeat step 3 for other queries until all queries in the list are merged into existing MVPP.</td>
</tr>
<tr>
<td>5. Push down select, project and aggregate functions as deep as possible.</td>
</tr>
<tr>
<td>6. Move the first new query to the end of the list.</td>
</tr>
<tr>
<td>7. Repeat step 3 to 6 to generate all new MVPPs.</td>
</tr>
<tr>
<td>8. Calculate total query processing cost based on base relations and existing materialized views for each MVPP and select the cheapest MVPP.</td>
</tr>
<tr>
<td>9. Apply the MVPP re-optimization algorithm described in section 3.4 to the cheapest MVPP.</td>
</tr>
</tbody>
</table>

| end; |

**Figure 3.13** The Methodology to Merge New Requirements into the Exiting MVPP
also affected even though they are not affected by new queries. As our objective of
dynamic materialized selection approach is to avoid calculating all resources again.
Then we choose the rewriting method for only the affected existing queries. We
rewrite the affected existing query by using a common subexpression that can be
shared with the new query.

For adding new requirements into existing MVPP, we have to consider how to
merge the queries into and existing environment. The approach to add new
requirements into the existing MVPP is shown in Figure 3.13.

### 3.7.2 An Approach to Identify the Affected Nodes

According to our objective of the dynamic materialized view selection, that is
to avoid rerun static approach when new requirements are changed, the affected nodes
have to be determined as the member of the set of views to be materialized rather than
all nodes in the search space. We apply static weight of node and the relevance
between nodes introduced by Zhang et al. described in section 2.6 for our algorithm.
The detail of formula is presented below:

\[
    w(v) = \sum_{q \in O_v} \left( f_q(q) \cdot C_{aq}^q(v) \right) - \sum_{r \in I_v} f_u(r) \cdot C_{mr}^r(v)
\]

- \( w(v) \) denotes weight of node
- \( O_v \) denotes the queries which use view \( v \).
- \( C_{aq}^q \) denotes the accessing cost \( a \) for query \( q \) using view \( v \). The cost of
  answering query \( q \) is the number of rows presented in the relation used to construct \( q \).
- \( f_q \) denotes the frequency of executing a query.
- \( I_v \) denotes the base relations which are used to produce view \( v \).
- \( C_{mr}^r \) denotes the maintenance cost \( m \) for materialized view \( v \) based on base
  relation \( r \), which is occasionally updated.
- \( f_u \) denotes the frequency of updating base relation
We propose the affected node identification algorithm shown in Figure 3.14 to identify the affected nodes either directly or indirectly after the optimal dynamic MVPP is generated. The directly affected nodes are the nodes with positive weight $w(v)$ used to construct the new queries. The indirectly affected nodes are the ancestor of directly affected nodes with certain weight calculated by the above formula. The affected nodes are the member of the set of views to be materialized or unmaterialized, rather than all nodes in the search space. The existing materialized view nodes not identified as affected nodes means those nodes are not affected by new requirements, such they still have to be the materialized views for supporting the existing requirements.

\begin{algorithm}
\begin{enumerate}
  \item Initial list $M_{direct}$ and $M_{indirect} = \emptyset$
    \begin{align*}
    M_{direct} \quad & \text{is the set of directly affected node} \\
    M_{indirect} \quad & \text{is the set of indirectly affected node}
    \end{align*}
  \item For each new query
    \begin{enumerate}
      \item Depth first search from the root to base relations to determine the existing intermediate nodes, $v_i$, used to construct the new query.
      \item Calculate weight $w(v)$ of each node $v_i$.
        \begin{align*}
        v_i, \text{ that are conjunctively joined with positive weight, or project operation that is not the ancestor of base relation, or select operation, are inserted into the list } M_{direct}.
        \end{align*}
      \item For each node $v_i$ in list $M_{direct}$ search its ancestor node $u_j$, $u_j \notin M_{direct}$, up to the query node
        \begin{enumerate}
          \item Calculate weight of node $u_j$.
          \item If ( weight $v_i >$ weight $u_j$ ) and $u_j$ is existing materialized view then put $u_j$ into list $M_{indirect}$
          \item If ( weight $v_i <$ weight $u_j$ ) then traverse in bottom-up way to find the node that return maximum weight $u_j$ of each branch.
            put $u_j$ into list $M_{indirect}$
        \end{enumerate}
    \end{enumerate}
\end{enumerate}
\end{algorithm}

Figure 3.14 The Affected Node Identification Algorithm

The conditions to identify the indirectly affected node in step 3 of the affected node identification algorithm in Figure 3.14 are explained as follows:

Referring the rules to determine how nodes are affected by each other in MVPP described in section 2.6, we exploit the second and third rule for our algorithm
We apply the second rule for condition in line 3.2. For this condition, before adding the new requirements, $u_j$ is the materialized view. It implies that the weight of $u_j$ is greater than that of $v_j$. So, $u_j$ is possible to be selected to be materialized rather than $v_j$. After adding a new requirement, the weight of $u_j$ is less than that of $v_j$. It implies that $v_j$ possible to be selected to be materialized rather than $u_j$. When $v_j$, the descendent node of $u_j$ with weight greater than that of $u_j$, is materialized so no gain to materialize $u_j$. Before adding new requirements, $u_j$ is the existing materialized view and it is likely to be un-materialized when new requirements are added. Thus, $u_j$ have to be identified as the indirectly affected node.

We apply the third rule for condition in line 3.3. The node $u_j$ is the ancestor of $v_j$, and the weight of $u_j$ is greater than $v_j$ then $u_j$ is possible to be materialized. The same as $u_{j+1}$ if $u_{j+1} > u_j$ then $u_{j+1}$ is more possible to be materialized rather than $u_j$. Thus we have to traverse to look for the maximum weight of $u_j$. As $u_j$ would have many ancestors that means $u_j$ is supporting many queries so we have to traverse bottom-up way to find the node that returns the maximum weight $u_j$ of each branch.

### 3.8 Two-Phase Optimization (2PO) for Dynamic Materialized View Selection

The final step of dynamic materialized view selection is selecting the set of views to be materialized by 2PO. The 2PO algorithm is the combination of II and SA explained in section 3.6. We map all nodes in MVPP to binary string according to the types of node as follows.

The intermediate nodes, either existing materialized views or virtual views that are identified as affected nodes and new nodes, are mapped into binary strings. We initialize these nodes with 0.

The existing materialized view nodes which are not identified as affected nodes are fixed to 1. The reason to fix to 1 is that they always are the materialized view to support the existing requirement. We consider the existing materialized views not identified as affected nodes because we calculate the query processing cost and materialized view maintenance cost for the whole system. Our goal is the minimal total cost among all feasible sets of materialized views of all queries. Therefore, all
existing materialized views that are not identified as affected nodes have to be included for supporting the existing queries.

The other intermediate nodes neither identified as affected nodes nor existing materialized views to support the existing queries are fixed to 0. They are always the virtual views that have not been affected. Thus, they are not the member of a set of views to be selected.

The details of algorithm are presented in Figure 3.15.

begin
1. Input the MVPP represented by a DAG
2. Use depth first search from root node to base relation to search through all of the nodes in the DAG.
3. Map all intermediate nodes into binary string 1s or 0s as follows:
   3.1 initialized with 0 for all affected nodes identified by the affected node identification algorithm, and new created nodes.
   3.2 fixed to 1 for all existing materialized view nodes which are not identified as affected nodes.
   3.3 fixed to 0 for other nodes that are not the set of views to be selected.
4. Call Iterative Improvement
5. Call Simulated Annealing algorithm
6. Present set of views to materialized with minimum cost
end;

Figure 3.15 The Materialized View Selection with 2PO for the Dynamic Phase

3.9 Cost Model for Dynamic Materialized View Selection

According to (Yang et al., 1997: 140) a linear cost model is used to calculate the processing cost of query $Q$. The cost of answering query $Q$ is the number of rows in the base relations used to construct query $Q$. 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 query and updating frequency respectively. Then the total query processing cost is $\sum_{q \in Q} f_q C_{q_i}(M)$ and the total
maintenance cost is \( \sum_{v \in M} f_u C_m(v) \). The total cost, which is the summation of query processing cost and materialized view maintenance cost, on the set of materialized views \( M \) is illustrated as follow:

\[
C_{total} = \sum_{q \in Q} f_q C_q(M) + \sum_{v \in M} f_u C_m(v)
\]

Our goal is the total cost will be minimal for all feasible sets of materialized views.

This chapter details our methodologies i.e. the MVPP re-optimization algorithm and the dynamic materialized view selection approach. The goal of the MVPP re-optimization algorithm is to verify whether the cheapest MVPP is optimal and improve the query processing cost of the cheapest MVPP. The dynamic materialized view selection approach is to select a set of views, which are the specific member of the set of views rather than all nodes in search space, to be materialized or un-materialized by avoiding recalculation from scratch, and provides the minimal summation of query processing cost and maintenance cost on the set of materialized views selected by 2PO. In the next chapter, the design of experiments, result and analysis of our methodologies will be described.
CHAPTER 4

DESIGN OF EXPERIMENTS AND ANALYSIS OF RESULTS

In previous chapter, we described the methodologies employed in our dissertation that includes two parts; the optimization task to improve the cheapest MVPP, and the approach for dynamic materialized view selection. In this chapter, we discuss the design of the experiments used to evaluate our approach. Our testbed data is the TPC Benchmark™H (TPC-H) that is a decision support benchmark. Section 4.1 provides the details of TPC-H schema revision 2.14.2 and its data set. Section 4.2 provides a query set with relational algebra query tree which were introduced by Phuboon-ob (2009: 51) for static materialized view selection. Section 4.3 provides the implementation of the cheapest MVPP proposed by Yang et al.

For our experiments, it is separated into two parts according to our proposed methodologies. First part includes Section 4.4 and 4.5 described the implementation of our MVPP re-optimization algorithm. Second part includes Section 4.6 to 4.8 described the implementation of our dynamic materialized view selection approach, 2PO for our dynamic materialized view selection.

4.1 TPC Benchmark™H (TPC-H)

The component of TPC-H schema is defined to consist of eight tables (base relation) including REGION, NATION, CUSTOMER, SUPPLIER, PART, PARTSUPP, LINEITEM AND ORDERS. The relationships between these tables in TPC-H schema are illustrated in Figure 4.1. We run TPC-H schema by Oracle11gR2 with database size 1 GB that is the minimum required for a test database. The cardinalities of each base relation are presented in Table 4.1. The database size can scale up to 1 TB according to the scale factor multiple by minimum cardinality of all tables except NATION and REGION.
Figure 4.1 The TPC-H Schema Revision 2.14.2


Table 4.1 The TPC-H Schema Table Size

<table>
<thead>
<tr>
<th>Table name</th>
<th>Relation Size (in Tuples)</th>
<th>Record Size (in bytes)</th>
<th>Table Size (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGION</td>
<td>5</td>
<td>124</td>
<td>&lt;1</td>
</tr>
<tr>
<td>NATION</td>
<td>25</td>
<td>128</td>
<td>&lt;1</td>
</tr>
<tr>
<td>CUSTOMER</td>
<td>150,000</td>
<td>179</td>
<td>26</td>
</tr>
<tr>
<td>SUPPLIER</td>
<td>10,000</td>
<td>159</td>
<td>2</td>
</tr>
<tr>
<td>PART</td>
<td>200,000</td>
<td>155</td>
<td>30</td>
</tr>
<tr>
<td>PARTSUPP</td>
<td>800,000</td>
<td>144</td>
<td>110</td>
</tr>
<tr>
<td>LINEITEM</td>
<td>6,000,000</td>
<td>112</td>
<td>641</td>
</tr>
<tr>
<td>ORDERS</td>
<td>1,500,000</td>
<td>104</td>
<td>149</td>
</tr>
</tbody>
</table>
4.2 Query Set for Static Materialized View Selection

We separate the query set into two sets. The first set includes Query1 to Query7 for static materialized view selection approach. The second set includes Query8 to Query13 for dynamic materialized view selection approach. Query1 to Query7 were introduced by Phuboon-ob (2009: 51) for static materialized view selection problem using 2PO based on MVPP structure. We improve the query processing cost of the cheapest MVPP of Query1 to Query7 by our MVPP re-optimization algorithm and use 2PO to select set of views to be materialized. Thereafter, the result of static approach, which is derived in our static phase, is the initial search space for the dynamic materialized view selection. The details of Query1 to Query7 are discussed in this section. The details of Query8 to Query13 are explained in section 4.6. The Query1 to Query7 are denotes as Q1, Q2, Q3, Q4, Q5, Q6 and Q7. Suppose that all base relations are updated once and the frequencies of executing the query of Q1 to Q7 are 2, 6, 7, 2, 5, 9 and 3 respectively.

We first present the notation used in relational algebra query tree as follows:

$\sigma_a$ represents the select operation, where $a$ is a selection condition on one or more attributes of a relation.

$\pi_b$ represents the project operation, where $b$ is a list of one or more attributes of a relation.

$\Join$ represents the inner join operation.

$\gamma$ represents an aggregation function.

The details of queries and their relational algebra query trees are described as follows:
Query Q1 with the query frequency of 2 produces the minimum supply cost of each nation of suppliers in specific region, ASIA. Its relational algebra tree is shown in Figure 4.2.

**Query Q1**

```sql
SELECT N_NAME, 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'
GROUP BY N_NAME;
```

**Figure 4.2** Relational Algebra Query Tree of Query Q1
Query Q2 with the query frequency of 6 produces number of items that customers of each nation in specific region, ASIA, ordered in 1994. Its relational algebra tree is shown in Figure 4.3.

**Query Q2**

\[
\begin{align*}
\text{SELECT} & \quad \text{N\_NAME, COUNT(L\_ORDERKEY)} \\
\text{FROM} & \quad \text{CUSTOMER, ORDERS, LINEITEM, NATION, REGION} \\
\text{WHERE} & \quad \text{C\_CUSTKEY = O\_CUSTKEY} \\
& \quad \text{AND O\_ORDERKEY = L\_ORDERKEY} \\
& \quad \text{AND C\_NATIONKEY = N\_NATIONKEY} \\
& \quad \text{AND N\_REGIONKEY = R\_REGIONKEY} \\
& \quad \text{AND R\_NAME = 'ASIA'} \\
& \quad \text{AND O\_ORDERDATE >= '1994-01-01'} \\
& \quad \text{AND O\_ORDERDATE < '1995-01-01'}
\end{align*}
\]

\[
\text{GROUP BY N\_NAME;}
\]

**Figure 4.3** Relational Algebra Query Tree of Query Q2
Query Q3 with the query frequency of 7 produces summation of quantities of items that suppliers of each nation in specific region, ASIA, ordered in 1994. Its relational algebra tree is shown in Figure 4.4.

**Query Q3**

```
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;
```

**Figure 4.4** Relational Algebra Query Tree of Query Q3
Query Q4 with the query frequency of 2 produces summation of supply cost of each supplier that their nation same as that of customers in specific region, ASIA. Its relational algebra tree is shown in Figure 4.5.

**Query Q4**

```
SELECT S_NAME, SUM(PS_SUPPLYCOST)
FROM PARTSUPP, SUPPLIER, CUSTOMER, NATION, REGION
WHERE PS_SUPPKEY = S_SUPPKEY
    AND C_NATIONKEY = S_NATIONKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND R_NAME = 'ASIA'
GROUP BY S_NAME;
```

![Figure 4.5 Relational Algebra Query Tree of Query Q4](image-url)
Query Q5 with the query frequency of 5 produces number of suppliers’ part with specific brand, type and size. Its relational algebra tree is shown in Figure 4.6.

**Query Q5**

```sql
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);
```

![Relational Algebra Query Tree of Query Q5](image)

**Figure 4.6** Relational Algebra Query Tree of Query Q5
Query Q6 with the query frequency of 9 produces supply cost summation of parts with specific brand, type and size for each supplier. Its relational algebra tree is shown in Figure 4.7.

**Query Q6**

```sql
SELECT S_NAME, SUM(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)
GROUP BY S_NAME;
```

![Relational Algebra Query Tree of Query Q6](image)

**Figure 4.7** Relational Algebra Query Tree of Query Q6
Query Q7 with the query frequency of 3 produces the discount summation of items that customers ordered in 1994 for each market segment. Its relational algebra tree is shown in Figure 4.8.

**Query Q7**

```sql
SELECT C_MKTSEGMENT, SUM(L_DISCOUNT)
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'
GROUP BY C_MKTSEGMENT;
```

![Figure 4.8](image-url)  
*Figure 4.8 Relational Algebra Query Tree of Query Q7*
4.3 The Cheapest MVPP Implementation

In this section, we build the cheapest MVPP mentioned in section 3.2 for materialized view selection. The details are listed below:

1. For every optimal query processing plan, if there is a join operation involved, push all the select, project operations 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 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
   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 select, project and aggregate functions as deep as possible.

7. Calculate the total query processing cost of each MVPP, and select the one which gives the lowest cost.

In our experiment, all optimal query processing plans of Q1 to Q7, if there is a join operation involved, we first push all the select, project operations and aggregate function up along the tree. The result of this step is shown in Figure 4.9. Next in the second step, we multiply query access frequency with query cost shown in Table 4.2. Table 4.2 illustrates the frequency of executing the query, query cost, and frequency of executing the query multiplied with query cost for Q1 to Q7. Later, we create a list of these values in descending order. Therefore, the initial list is {Q4, Q7, Q3, Q2, Q6, Q1, and Q5}. 
Figure 4.9  The Result of the First Step to Construct MVPP

Table 4.2  The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Access Frequency($f_q$)</th>
<th>Query Cost</th>
<th>$f_q \times$ Query Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>160,240</td>
<td>320,480</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>184,042</td>
<td>1,104,252</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>182,183</td>
<td>1,275,281</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>967,519,280</td>
<td>1,935,038,560</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>36,276</td>
<td>181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>36,276</td>
<td>326,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>910,519</td>
<td>2,731,557</td>
</tr>
</tbody>
</table>

Third step, all queries in the list are merged to construct the first MVPP. The sequence to merge all queries of the first MVPP as follows: first is Q4 follow by Q7, then Q3 and so on until Q5 is merged. When Q7 is merged, as no sharable
subexpression between Q4 and Q7 then we first join the ORDERS with LINEITEM, and then join this result with the CUSTOMER. The result after Q7 is merged shown in Figure 4.10 (a). When Q3 is merged, Q3 is constructed on conjunctively joined of (REGION \Join NATION \Join SUPPLIER \Join LINEITEM \Join ORDERS). There are two subtrees available for Q3 that are (REGION \Join NATION \Join SUPPLIER) and (ORDERS \Join LINEITEM), so a new node is introduced as a join operation between those two subtrees. The result after Q3 is merged into the first MVPP shown in Figure 4.10 (b). When Q2 is merged, there already existing conjunctively join that are (REGION \Join NATION), the remaining base relations are CUSTOMER, ORDERS and LINEITEM conjunctively joined already for Q7.

(a) The result after Q7 is merged with Q4 of first MVPP

(b) The result after Q3 is merged of the first MVPP

**Figure 4.10** The Result of Merging Steps for the First MVPP
Therefore, the new node is introduced for Q2 as a join operation of those results: (REGION \bowtie NATION) and (CUSTOMER \bowtie ORDERS \bowtie LINEITEM). Figure 4.10 (c) shows the first MVPP after Q2 is merged. We repeat this step until all queries in the list are merged. The result after all queries are merged starting with Q4 is shown in Figure 4.10 (d).

Next the fourth step, after the first MVPP is generated, the first element of the list is moved to the end of the list. So Q4 is move to the end of the list, the list becomes \{Q7, Q3, Q2, Q6, Q1, Q5, and Q4\} that is the order of merging queries for
the second MVPP. We repeat the third step to construct the second MVPP. We start with second MVPP equal to Q7. Then merge Q3 into the second MVPP and follow by the next query in the list, Q2, Q6, Q1, Q5 and the last query is Q4. Figure 4.11 (a) shows the result after merged Q3 into the second MVPP. Figure 4.11 (b) shows the result after Q2 is merged into the second MVPP. The result of the second MVPP after all queries are merged starting with Q7 is shown in Figure 4.11 (c).

(a) The result after Q3 is merged with Q7 of the second MVPP

(b) The result after Q2 is merged of the second MVPP

**Figure 4.11** The Result of Merging Steps for the Second MVPP
The result of the second MVPP after queries Q1-Q7 are merged

Figure 4.11 (Continued)

We repeat the third and the fourth step until all MVPPs of seven queries are built. The list of queries of the last MVPP is \{Q5, Q4, Q7, Q3, Q2, Q6, and Q1\}. The results of all MVPPs generated in the merging steps are shown in Appendix A.

After all MVPPs are constructed, we optimize MVPP by push select, project and aggregate function down as deep as possible for all MVPPs. Figure 4.12 to 4.18 show the first MVPP to the last MVPP already optimized.

Finally the total query processing costs of MVPP, which is the summation of query processing cost of queries in the MVPP, are calculated to determine the cheapest one. The query processing cost is the frequency of executing the query multiplied with the cost of accessing the nodes to obtain the result of the query.

For example Q1 in the first MVPP as Figure 4.12, Q1 which has frequency of executing query is 2 accesses nodes named Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp18, Tmp22 and result1. The processing cost of each node is 5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240, respectively. Then, the query processing cost of Q1 is \((2) \times (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 200000 + 32048000000 + 160240)\) that is 67,303,240,592. The query processing cost of the first MVPP in Figure 4.12 are
shown in Table 4.3. The query processing cost of the others MVPPs are provided in Appendix A.

The query processing costs of all MVPPs are shown in Table 4.4. The cheapest MVPP is the third MVPP as it provides the minimal total query processing cost that is 10,821,545,680,471. The order of query in the list of the cheapest MVPP is \{Q3, Q2, Q6, Q1, Q5, Q4, and Q7\}.

The result in Table 4.4 shows that the query processing cost of Q1, Q5 and Q6 of the third MVPP is higher than other MVPPs, although the third MVPP provides the minimal total query processing cost. It implies that there are other execution plans for these queries that have lower query processing cost. Then, we implement our MVPP re-optimization algorithm in section 4.4 to improve the query processing cost of problematic queries.

\[\text{Figure 4.12} \text{ The First MVPP, the Queries in the List: \{Q4, Q7, Q3, Q2, Q6, Q1, and Q5\}}\]
Figure 4.13 The Second MVPP, the Query in the List: \{Q7, Q3, Q2, Q6, Q1, Q5, and Q4\}

Figure 4.14 The Third MVPP (the Cheapest MVPP), the Query in the List: \{Q3, Q2, Q6, Q1, Q5, Q4, and Q7\}
Figure 4.15 The Fourth MVPP, the Query in the List: 
\{Q2, Q6, Q1, Q5, Q4, Q7, and Q3\}

Figure 4.16 The Fifth MVPP, the Query in the List: 
\{Q6, Q1, Q5, Q4, Q7, Q3, and Q2\}
Figure 4.17 The Sixth MVPP, the Query in the List: {Q1, Q5, Q4, Q7, Q3, Q2, and Q6}

Figure 4.18 The Seventh MVPP, the Query in the List: {Q5, Q4, Q7, Q3, Q2, Q6, and Q1}
Table 4.3 The Query Processing Cost of the First MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>$f_q$</th>
<th>Access from the Nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp7, Tmp8, Tmp18, Tmp22 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1024000000, 200000, 32048000000 and 160240</td>
<td>67,303,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15, Tmp17 and result2</td>
<td>5, 1, 25, 25, 6000000, 150000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp11, Tmp12, Tmp13, Tmp14, Tmp16 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 16024000000, 150000, 24036000000 and 967519280</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp7, Tmp18, Tmp19, Tmp20 and result5</td>
<td>800000, 200000, 200000, 7255200000 and 36276</td>
<td>36,282,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp7, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6</td>
<td>800000, 200000, 20000, 7255200000, 10000, 3627600000 and 36276</td>
<td>68,572,856,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15 and result7</td>
<td>6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the first MVPP 23,316,809,013,207
Table 4.4 The Query Processing Cost of All MVPPs

<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP (Cheapest MVPP)</th>
<th>4th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>9,013,034,785,980</td>
<td>9,013,034,785,980</td>
<td>1,697,558,231,916</td>
<td>1,697,558,231,916</td>
</tr>
<tr>
<td>Q3</td>
<td>9,571,896,175,751</td>
<td>9,571,896,175,751</td>
<td>1,997,769,797,079</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>36,282,181,380</td>
<td>800,009,181,380</td>
<td>800,009,181,380</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>68,572,856,484</td>
<td>1,443,281,456,484</td>
<td>1,443,281,456,484</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>23,316,809,013,207</strong></td>
<td><strong>25,711,148,613,207</strong></td>
<td><strong>10,821,545,680,471</strong></td>
<td><strong>18,395,672,059,143</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>5th MVPP</th>
<th>6th MVPP</th>
<th>7th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>9,013,034,785,980</td>
<td>9,013,034,785,980</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>9,571,896,175,751</td>
<td>9,571,896,175,751</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>800,009,181,380</td>
<td>36,282,181,380</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>1,443,281,456,484</td>
<td>68,572,856,484</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>25,711,148,613,207</strong></td>
<td><strong>23,316,809,013,207</strong></td>
<td><strong>25,711,148,613,207</strong></td>
</tr>
</tbody>
</table>

The 3rd MVPP is the cheapest MVPP because it provides the minimum query processing cost.
4.4 The Re-Optimized MVPP Implementation

The objective of the MVPP re-optimization algorithm, described in Figure 3.5 in section 3.4, is to reduce the total query processing cost of the cheapest MVPP. The details of algorithm are listed below:

1. Input = the cheapest MVPP
2. Initial list $LV = \emptyset$
3. $k =$ number of queries
4. for $i = 1$ to $k$
   
   Compare $Cq(i)$ of cheapest MVPP with $Cq_j(i)$ of other MVPPs.
   
   If $Cq_j(i)$ less than $Cq(i)$ then
   
   insert $q(i)$ into $LV$.
5. For queries in $LV$, consider the possible commonalities with exists
   
   global equivalent plan as following:
5.1 If there is nothing in common with global equivalent plan
   
   skip to the next query.
5.2 If there is more than one overlapping portion
   
   rewrite this query using exists common subexpression in
   
   MVPP in bottom-up way

The implementations of the MVPP re-optimization algorithm are explained as follows.

First, the cheapest MVPP is the input of MVPP re-optimization algorithm and
initial the list $LV=\emptyset$. Next, the query processing cost of each query in the cheapest
MVPP is compared with other MVPPs. Using the query processing cost in Table 4.4, the
details of comparison are described as follows:

Q1: its query processing cost of the cheapest MVPP is $323,207,240,592$
whereas its query processing cost of the first and the sixth MVPP is
$67,303,240,592$.

Q2: its query processing cost of the cheapest MVPP is less than other
MVPPs but is equal to the fourth MVPP.
Q3: its query processing cost of the cheapest MVPP is less than other MVPPs
Q4: its query processing cost is equal to all MVPPs.
Q5: its query processing cost of the cheapest MVPP is 800,009,181,380 whereas its query processing cost of the first and the sixth MVPP is 36,282,181,380.
Q6: its query processing cost of the cheapest MVPP is 1,443,281,456,484 whereas its query processing cost of the first and the sixth MVPP is 68,572,856,484.
Q7: its query processing cost is equal to all MVPPs.

So the result of the list $LV$ contains $\{Q1, Q5, Q6\}$ because their query processing cost of the cheapest MVPP are more than other MVPPs.

Thereafter, the queries in the list are considered to be rewritten if they can use alternative common subexpression available in the cheapest MVPP.

For Q1, query processing plan that is used in the cheapest MVPP is shown in Figure 4.19 (a). Considering optimal query processing plan of Q1 in Figure 4.2, there are possible common subexpression with the other queries in the cheapest MVPP in Figure 4.14 as follows:

- Tmp6 in Figure 4.2 is Tmp6 in Figure 4.14.
- Tmp8 in Figure 4.2 is Tmp17 in Figure 4.14. In Figure 4.14, as Tmp17 is the ancestor of Tmp6 then Tmp17 should be chosen to be common subexpression for Q1 rather than Tmp6.
- Tmp19 in Figure 4.14 is overlapping portion of Q1, Q5 and Q6. Tmp19 in Figure 4.14, the cheapest MVPP, is Tmp4 of Q5 in Figure 4.6 and Tmp4 of Q6 in Figure 4.7.

Because Q1 has overlapping with several queries and its query processing cost in the cheapest MVPP is higher than that of the first and the sixth MVPP so Q1 in the cheapest MVPP would be rewritten as follows:

- We match optimal query processing plan of Q1 in bottom-up way from leaf node up to the query node.
- We can match the node in optimal query processing plan of Q1 with the
nodes in the cheapest MVPP at node Tmp6 and Tmp17 in Figure 4.14. Tmp17 in Figure 4.14 is the conjunctively join of the base relations i.e. REGION, NATION, SUPPLIER, and PARTSUPP of Q1.

- The new node is introduced to join remain base relation, PART, with Tmp17 called Tmp19 in Figure 4.19 (b) and Figure 4.20.

We present the equivalent plan of Q1 in MVPP before and after rewritten as Figure 4.19. Suppose that R, N, S, PS and P represent the base relation REGION, NATION, SUPPLIER, PARTSUPP AND PART respectively. After rewrite Q1, the execution plan of Q1 in the MVPP, shown as Figure 4.19 (b), is same as the individual optimal query plan of Q1 in Figure 4.2. Tmp17 in Figure 4.19 (b) is Tmp8 of Figure 4.2. Tmp19 in Figure 4.19 (b) is the Tmp10 of Figure 4.2. The query processing cost of Q1 after rewritten is reduced from 323,207,240,592 to 67,303,240,592.

![Diagram of Q1 before and after rewriting](image)

(a) Q1 before rewriting, black node represents the node has to be rewritten

(b) Q1 after rewriting, black node represents the new node after rewriting

**Figure 4.19** The Execution Plan of Q1 Before and After Rewriting
For Q5 and Q6, they do not have sharable subexpression with other queries in the cheapest MVPP. Therefore, Q5 and Q6 are ignored to rewrite.

Finally we push select, project operation and aggregate function down as deep as possible for all affected queries. The result of re-optimized cheapest MVPP is shown in Figure 4.20. Considering the impact to Q5 and Q6 after select operation is push down. In the cheapest MVPP in Figure 4.14, select operation of Q5 and Q6, that is “P_BRAND <> ‘BRAND#45’ AND NOT P_TYPE LIKE ‘%BRASS%’ AND P_SIZE IN (9,19,49)”, cannot be pushed down beyond Tmp18 because Q1 is also derived from Tmp19 which uses the result from Tmp18 without these selection operation. After Q1 is rewritten, this operation can be pushed down beyond conjunctively joined node. The select operation node is Tmp20 and the conjunctively joined node is Tmp21 in Figure 4.20. So, the query processing cost of Q5 and Q6 in Figure 4.20 is less than the cheapest MVPP. The query processing cost after the cheapest MVPP is re-optimized is shown in Table 4.5. The result shows that the total query processing cost of MVPP after re-optimized is reduced from 10,821,545,680,471 to 8,427,206,080,471.

Figure 4.20 The Cheapest MVPP after Applying the MVPP Re-Optimization Algorithm
Table 4.5 The Query Processing Cost of the Cheapest MVPP and the Re-Optimized MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>The Cheapest MVPP</th>
<th>The Re-optimized MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number 1 (Q1)</td>
<td>323,207,240,592</td>
<td>67,303,240,592</td>
</tr>
<tr>
<td>Query number 2 (Q2)</td>
<td>1,697,558,231,916</td>
<td>1,697,558,231,916</td>
</tr>
<tr>
<td>Query number 3 (Q3)</td>
<td>1,997,769,797,079</td>
<td>1,997,769,797,079</td>
</tr>
<tr>
<td>Query number 4 (Q4)</td>
<td>53,213,858,672</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Query number 5 (Q5)</td>
<td>800,009,181,380</td>
<td>36,282,181,380</td>
</tr>
<tr>
<td>Query number 6 (Q6)</td>
<td>1,443,281,456,484</td>
<td>68,572,856,484</td>
</tr>
<tr>
<td>Query number 7 (Q7)</td>
<td>4,506,505,914,348</td>
<td>4,506,505,914,348</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>10,821,545,680,471</strong></td>
<td><strong>8,427,206,080,471</strong></td>
</tr>
</tbody>
</table>

The result from Table 4.5 shows that our MVPP re-optimization algorithm reduces the total query processing cost of the cheapest MVPP. Moreover, we further validate our re-optimized MVPP by implementing the selection a set of view to be materialized by 2PO in section 4.5. Our expectation is that the total cost which is the summation of query processing cost and materialized view maintenance cost of the re-optimized MVPP should less than that of the cheapest MVPP.

4.5 Evaluation of the MVPP Re-Optimization Algorithm

We evaluate our MVPP re-optimization algorithm by selecting the set of views to be materialized by 2PO. Our goal is that the total cost of the re-optimized MVPP should less than that of the cheapest MVPP for all costs which are all-virtual view, all-materialized views and materialized view.

The details of implementing 2PO for materialized view selection are described in Figure 4.21.
begin
7. Input the MVPP represented by a DAG
8. Use depth first search from root nodes to base relations to search through all of the nodes in the DAG.
9. Produce the sequence of nodes into a binary string.
10. Call Iterative Improvement
11. Call Simulated Annealing algorithm
12. Present set of views to materialized with minimum cost
end;

**Figure 4.21** The Materialized View Selection with 2PO

We apply the 2PO algorithm to the cheapest MVPP in Figure 4.14. We first search through the DAG for Figure 4.14 using depth first search, we obtain the mapping array as follow:

\[ [\text{Tmp22,0}], [\text{Tmp19,0}], [\text{Tmp18,0}], [\text{Tmp16,0}], [\text{Tmp6,0}], [\text{Tmp5,0}], [\text{Tmp4,0}], [\text{Tmp3,0}], [\text{Tmp2,0}], [\text{Tmp1,0}], [\text{Tmp15,0}], [\text{Tmp7,0}], [\text{Tmp14,0}], [\text{Tmp10,0}], [\text{Tmp9,0}], [\text{Tmp13,0}], [\text{Tmp12,0}], [\text{Tmp11,0}], [\text{Tmp8,0}], [\text{Tmp23,0}], [\text{Tmp17,0}], [\text{Tmp20,0}], [\text{Tmp21,0}], [\text{Tmp25,0}], [\text{Tmp24,0}] \]

**Figure 4.22** The Cheapest MVPP with Materialized Views
So initially, the binary string of above mapping \{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\} indicates that all intermediate node are virtual views. The result, after 2PO algorithm is applied to select the set of views to be materialized, \{0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1\} indicates that Tmp19, Tmp6, Tmp15, Tmp11 and Tmp24 are materialized views. Figure 4.22 represents the cheapest MVPP with materialized views after 2PO is applied.

Considering query Q1 in the cheapest MVPP in Figure 4.14, its frequency of executing the query is 2, before materializing the intermediate nodes, Q1 accesses the nodes named Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp18, Tmp19, Tmp22 and result1. The cost of each node is 5, 1, 25, 25, 10000, 800000, 200000, 160000000000, 1602400000 and 160240, respectively. So, the query processing cost of query Q1 is \((2^2)(5 + 1 + 25 + 25 + 10000 + 800000 + 200000 + 160000000000 + 1602400000 + 160240) = 323,207,240,592\).

After Tmp6 and Tmp19 are materialized, Q1 accesses the nodes named Tmp6, Tmp19, Tmp22 and result1. The cost of each node is 2003, 800000, 1602400000 and 160240, respectively. So, the query processing cost of Q1 using materialized views, Tmp6 and Tmp19, \(2^2(2003 + 800000 + 1602400000 + 160240)\) that is 3,206,724,486. It would be beneficial to materialize them, reducing the processing cost from 323,207,240,592 to 3,206,724,486.

However, there is materialized view maintenance cost whenever an update of involved base relations occurs. Tmp6 is constructed on three base relations, and accesses nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5 and the node itself. The cost of each node is 5, 1, 25, 25, 10000 and 50000, respectively. Then, the materialized view maintenance cost of Tmp6 is \(3^2(5 + 1 + 25 + 25 + 10000 + 50000)\) that is 180,168. The view maintenance cost of Tmp19, which is constructed on two base relations and accesses nodes Tmp18, Tmp19 and the node itself, is \(2^2(800000 + 200000 + 160000000000)\) that is 320,002,000,000.

After materialized views those five nodes, the total query processing cost is 469,452,527,576 and materialized view maintenance cost is 5,892,778,110,452. So the total cost of the cheapest MVPP is 6,362,230,638,028. The materialized view maintenance cost and query processing cost of the cheapest MVPP are shown Table 4.6 and 4.7 respectively.
Table 4.6 The Maintenance Cost of the Cheapest MVPP

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Number of Base Relations</th>
<th>Constructed from Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>3</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6</td>
<td>180,168</td>
</tr>
<tr>
<td>Tmp11</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11</td>
<td>1,426,977,515,570</td>
</tr>
<tr>
<td>Tmp15</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15</td>
<td>1,414,630,939,520</td>
</tr>
<tr>
<td>Tmp19</td>
<td>2</td>
<td>Tmp16, Tmp18, Tmp19</td>
<td>320,002,000,000</td>
</tr>
<tr>
<td>Tmp24</td>
<td>2</td>
<td>Tmp7, Tmp9, Tmp10, Tmp24</td>
<td>2,731,179,455,194</td>
</tr>
<tr>
<td><strong>Total materialized view maintenance cost</strong></td>
<td></td>
<td></td>
<td><strong>5,892,790,090,452</strong></td>
</tr>
</tbody>
</table>

Table 4.7 The Query Processing Cost of the Cheapest MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number 1 (Q1)</td>
<td>Tmp6, Tmp19, Tmp22, result1</td>
<td>3,206,724,486</td>
</tr>
<tr>
<td>Query number 2 (Q2)</td>
<td>Tmp15, result2</td>
<td>2,208,984</td>
</tr>
<tr>
<td>Query number 3 (Q3)</td>
<td>Tmp11, result3</td>
<td>2,550,562</td>
</tr>
<tr>
<td>Query number 4 (Q4)</td>
<td>Tmp6, Tmp16, Tmp17, Tmp12, Tmp23, result4</td>
<td>53,213,742,566</td>
</tr>
<tr>
<td>Query number 5 (Q5)</td>
<td>Tmp19, Tmp20, result5</td>
<td>8,181,380</td>
</tr>
<tr>
<td>Query number 6 (Q6)</td>
<td>Tmp19, Tmp20, Tmp21, Tmp5, result6</td>
<td>3,279,656,484</td>
</tr>
<tr>
<td>Query number 7 (Q7)</td>
<td>Tmp24, Tmp12, Tmp25, result7</td>
<td>409,739,463,114</td>
</tr>
<tr>
<td><strong>Total query processing cost</strong></td>
<td></td>
<td><strong>469,452,527,576</strong></td>
</tr>
</tbody>
</table>
We apply the 2PO algorithm to the re-optimized MVPP in Figure 4.20. The materialize views are Tmp6, Tmp11, Tmp15, Tmp21 and Tmp24. Figure 4.23 represents the re-optimized MVPP with materialized views after 2PO is applied. The results from 2PO for the re-optimized MVPP are provided in Appendix E.

**Figure 4.23** The Re-Optimized MVPP with Materialized View Nodes Selected by 2PO

After 2PO is applied to select the set of views to be materialized, the total query processing cost is 533,527,035,440 and the materialized view maintenance cost is 5,587,300,890,452. Therefore, the total cost of the re-optimized MVPP is 6,120,827,925,892. The materialized view maintenance cost of materialized views and the query processing cost of the re-optimized MVPP are shown in Table 4.8 and 4.9 respectively.
### Table 4.8 The Maintenance Cost of the Re-Optimized MVPP

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Number of Base Relations</th>
<th>Constructed from Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>3</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6</td>
<td>180,168</td>
</tr>
<tr>
<td>Tmp11</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11</td>
<td>1,426,977,515,570</td>
</tr>
<tr>
<td>Tmp15</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15</td>
<td>1,414,630,939,520</td>
</tr>
<tr>
<td>Tmp21</td>
<td>2</td>
<td>Tmp16, Tmp18, Tmp20, Tmp21</td>
<td>14,512,800,000</td>
</tr>
<tr>
<td>Tmp24</td>
<td>2</td>
<td>Tmp7, Tmp9, Tmp10, Tmp24</td>
<td>2,731,179,455,194</td>
</tr>
</tbody>
</table>

Total materialized view maintenance cost: 5,587,300,890,452

### Table 4.9 The Query Processing Cost of the Re-Optimized MVPP

<table>
<thead>
<tr>
<th>Query number 1 (Q1)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6, Tmp16, Tmp17, Tmp18, Tmp19, result1</td>
<td>67,303,124,486</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 2 (Q2)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp15, result2</td>
<td>2,208,984</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 3 (Q3)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp11, result3</td>
<td>2,550,562</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 4 (Q4)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6, Tmp16, Tmp17, Tmp12, Tmp23, result4</td>
<td>53,213,742,566</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 5 (Q5)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp21, result5</td>
<td>362,760</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 6 (Q6)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp21, Tmp5, Tmp22, result6</td>
<td>3,265,582,968</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 7 (Q7)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp24, Tmp12, Tmp25, result7</td>
<td>409,739,463,114</td>
<td></td>
</tr>
</tbody>
</table>

Total query processing cost: 533,527,035,440
We calculate the query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized views and materialized views on the set of materialized views selected by 2PO algorithm. All costs of the cheapest MVPP are shown in Table 4.10 and all costs of the re-optimized MVPP are shown in Table 4.11.

**Table 4.10** The Query Processing Cost, Maintenance Cost and Total Cost of the Cheapest MVPP

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>10,821,545,680,471</td>
<td>0</td>
<td>10,821,545,680,471</td>
</tr>
<tr>
<td>All-materialized views</td>
<td>1,940,978,234</td>
<td>9,090,266,440,303</td>
<td>9,092,207,418,537</td>
</tr>
<tr>
<td>2PO</td>
<td>469,452,527,576</td>
<td>5,892,790,090,452</td>
<td>6,362,242,618,028</td>
</tr>
</tbody>
</table>

**Table 4.11** The Query Processing Cost, Maintenance Cost and Total Cost of the Re-Optimized MVPP

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>8,427,206,080,471</td>
<td>0</td>
<td>8,427,206,080,471</td>
</tr>
<tr>
<td>All-materialized views</td>
<td>1,940,978,234</td>
<td>7,686,779,440,303</td>
<td>7,688,720,418,537</td>
</tr>
<tr>
<td>2PO</td>
<td>533,527,035,440</td>
<td>5,587,300,890,452</td>
<td>6,120,827,925,892</td>
</tr>
</tbody>
</table>

The comparisons of all costs for the cheapest MVPP and the re-optimized MVPP are shown in Table 4.12. The comparison result shows that all total costs of the re-optimized MVPP are less than that of the cheapest MVPP. The total cost of all-virtual view reduced from 10,821,545,680,471 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 selecting materialized views using 2PO algorithm is reduced from 6,362,242,618,028 to 6,120,827,925,892.

**Table 4.12** The Comparison of Total Costs of the Cheapest MVPP and the Re-Optimized MVPP

<table>
<thead>
<tr>
<th></th>
<th>The Cheapest MVPP</th>
<th>The Re-Optimized MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>10,821,545,680,471</td>
<td>8,427,206,080,471</td>
</tr>
<tr>
<td>All-materialized views</td>
<td>9,092,207,418,537</td>
<td>7,688,720,418,537</td>
</tr>
<tr>
<td>2PO</td>
<td>6,362,242,618,028</td>
<td>6,120,827,925,892</td>
</tr>
</tbody>
</table>

The above experiments are the static materialized view selection, and applying our MVPP re-optimization algorithm to improve the query processing cost of MVPP. However, the static approach has to be processed repeatedly all requirements, existing and new requirements, whenever the requirements are changed. In the next section, the experiments of dynamic materialized view selection to support the new requirements will be presented. The MVPP, shown in Figure 4.23, will be used as an initial search space for the dynamic phase.
4.6 Dynamic Materialized View Implementation

According to the changing requirement situations in section 3.7.1 include (1) the existing queries are deleted, (2) the new queries are added, (3) the definitions of existing query are changed and (4) the frequencies of executing queries and/or the frequencies of updating base relations are changed. For our new requirements that we are implementing includes the new queries are added and the existing queries are deleted. In case of the definition of existing queries is changed, we implement by deleting the existing query and re-adding the query with new definition. In case, when the frequency of executing query and/or frequency of updating base relations is changed, the query processing cost and maintenance cost always changed accordingly. The existing MVPP structure is not changed as neither new node is created nor the existing node is deleted. The nodes in existing MVPP, used to construct the queries might be affected. Therefore, it is possible that the existing materialized views will be un-materialized, and the virtual views will be materialized according to the cost function. Further investigation, for the case subtree of existing query is the subsumption of subtree of new query, so the existing query can be rewritten using the sharable subexpression with new query. We show our example experiment of this case as Example A in Appendix B.

We design the scenario for our experiments regarding to the types of commonality mentioned in section 3.3 to investigate how the existing MVPP is affected by adding each type of query. The scenarios cover three commonalities that are (1) nothing in common (2) subsumption (3) partially overlapping. Moreover, we also implement the query deletion and addition of the query that is constructed on all base relations.

The following scenarios are the design of our experiments to validate our dynamic materialized view approach.

(1) Adding new query identified as nothing in common with existing MVPP is described in section 4.6.1.2.

(2) Adding new query identified as subsumption of existing MVPP is described in section 4.6.1.3.
(3) Adding new query identified as partial overlapping with existing MVPP is described in section 4.6.1.4

(4) Deleting the existing query to see the effect of deletion the existing query to MVPP is described in section 4.6.1.5.

(5) Adding new query constructed on all base relations, described in section 4.6.1.6. We would like to verify that our affected node identification algorithm is valid even new query use all base relations. Using all base relations is possible and all existing queries are affected by the new query.

(6) As in real environment, several types of new query are added together then we add all types of query simultaneously into the existing MVPP for our experiment. The detail of this scenario is described in 4.6.2 and 4.6.3.

In our experiment, the initial MVPP structure generated in the static phase as shown in Figure 4.23 is the initial MVPP for the dynamic phase.

4.6.1 Query Sets and Implementation of Scenarios

4.6.1.1 Query Set for New Requirements

In this section, we explain the selected queries for dynamic phase covering our above scenarios. The queries are constructed on base relations and/or the existing materialized view(s) generated in the static phase. The query set for dynamic phase includes Query8 to Query13. Their SQL statements and relational algebra query trees are described as follows:
Query Q8 with the query frequency of 6 produces the maximum of item’s tax for each brand for specific part type and the committed date is before receipt date. Its relational algebra tree is shown in Figure 4.24.

**Query Q8**

```sql
SELECT P_BRAND, MAX(L_TAX)
FROM PART, LINEITEM
WHERE P_PARTKEY = L_PARTKEY
AND P_TYPE LIKE '%BRASS%'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

![Relational Algebra Query Tree of Query Q8](image)

**Figure 4.24** Relational Algebra Query Tree of Query Q8
Query Q9 with the query frequency of 4 produces the average total price of orders occurred in 1994 for each nation of customers in specific region, ASIA. Its relational algebra tree is shown in Figure 4.25.

**Query Q9**

SELECT N_NAME, AVG(O_TOTALPRICE)
FROM REGION, NATION, CUSTOMER, ORDERS
WHERE C_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND C_CUSTKEY = O_CUSTKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND R_NAME = 'ASIA'
GROUP BY N_NAME;

**Figure 4.25** Relational Algebra Query Tree of Query Q9
Query Q10 with the query frequency of 5 produces the minimum discount for each type of order’s priority occurred in 1994. Its relational algebra tree is shown in Figure 4.26.

**Query Q10**

SELECT O_ORDERPRIORITY, MIN(L_DISCOUNT)
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY O_ORDERPRIORITY;

![Relational Algebra Query Tree of Query Q10](image-url)

**Figure 4.26** Relational Algebra Query Tree of Query Q10
Query Q11 with the query frequency of 5 produces the summation of extended price for each status of order occurred in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure 4.27.

**Query Q11**

```
SELECT O_ORDERSTATUS, SUM(L_EXTENDEDPRICE)
FROM PARTSUPP, LINEITEM, ORDERS
WHERE O_ORDERKEY = L_ORDERKEY
    AND PS_PARTKEY = L_PARTKEY
    AND PS_SUPPKEY = L_SUPPKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
    AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERSTATUS;
```

*Figure 4.27* Relational Algebra Query Tree of Query Q11
Query Q12 with the query frequency of 2 produces the maximum of extended price for each supplier in specific region, ASIA, and specific part type that committed date is before receipt date. Its relational algebra tree is shown in Figure 4.28.

**Query Q12**

\[
\text{SELECT } \text{N\_NAME, MAX(L\_EXTENDEDPRICE)} \\
\text{FROM } \text{REGION, NATION, SUPPLIER, PARTSUPP, PART, LINEITEM} \\
\text{WHERE } \text{S\_SUPPKEY} = \text{PS\_SUPPKEY} \\
\text{AND S\_NATIONKEY} = \text{N\_NATIONKEY} \\
\text{AND N\_REGIONKEY} = \text{R\_REGIONKEY} \\
\text{AND PS\_PARTKEY} = \text{L\_PARTKEY} \\
\text{AND PS\_SUPPKEY} = \text{L\_SUPPKEY} \\
\text{AND PS\_PARTKEY} = \text{P\_PARTKEY} \\
\text{AND P\_TYPE LIKE '%BRASS%'} \\
\text{AND L\_COMMITDATE < L\_RECEIPTDATE} \\
\text{AND R\_NAME = 'ASIA'} \\
\text{GROUP BY N\_NAME;}
\]
Query Q13 with the query frequency of 5 produces the average of extended price for each brand that customer in region, ASIA, ordered in 1994 with specific part type that available quantity is more than 200, and the committed date is before receipt date. Its relational algebra tree is shown in Figure 4.29.

**Query Q13**

```sql
SELECT P_BRAND, AVG(L_EXTENDEDPRICE)
FROM REGION, NATION, SUPPLIER, CUSTOMER, PART, ORDERS,
PARTSUPP, LINEITEM
WHERE
  C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND O_ORDERKEY = L_ORDERKEY
AND S_SUPPKEY = PS_SUPPKEY
AND P_PARTKEY = PS_PARTKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
AND P_TYPE LIKE '%BRASS%'
AND PS_AVAILQTY < 2000
AND R_NAME = 'ASIA'
GROUP BY P_BRAND;
```
Suppose that all base relations are updated once and the frequencies of executing a query of Q8, Q9, Q10, Q11, Q12 and Q13 are 6, 4, 5, 5, 2 and 5 respectively.

The relevance of the new queries to existing MVPP generated in static phase shown in Figure 4.23 is explained as follows:

Q8 is classified as nothing in common because there is neither conjunctive joined nor select operation sharable subexpression with existing MVPP.

Q9 and Q10 are classified as totally overlapping or subsumption. Tmp9 in individual plan of Q9 in Figure 4.25 is subsumption of the existing query Q2 at node Tmp14 in the existing MVPP, Figure 4.23, and there is no existing materialized view in this subgraph. Tmp4 in individual plan of Q10 in Figure 4.26 is subsumption of the existing query Q7 at the materialized view node Tmp24 in Figure 4.23.
Q11 and Q12 are classified as partially overlapping. Q11 is partially overlapping with Q7, the materialized view Tmp24 in existing MVPP is the sharable subexpression for Q7 and Q11. The other nodes of Q11 are not the sharable subexpression with the existing MVPP. Q12 is partially overlapping with Q1 and Q4. Tmp17 is sharing subexpression for Q12, Q1 and Q4. Tmp19 is sharable subexpression for Q12 and Q1. The other nodes of Q12 are not sharable subexpression with the existing MVPP.

Q13 is constructed by using all base relations in TPC-H schema and it has sharable subexpressions with existing MVPP that will be explained in section 4.6.1.6.

4.6.1.2 Analysis Result of the Nothing in Common Data Set

We add only Q8 into existing MVPP, as Q8 does not have shareable subexpression with the existing MVPP then new nodes Tmp26, Tmp27 and Tmp28 are created to support Q8. Figure 4.30 shows MVPP after Q8 is merged.

Next, the affected node identification algorithm mentioned in section 3.7.2 is applied to identify the affected nodes. We first depth first search to find the existing nodes used to construct Q8 that are Tmp7 and Tmp18. Although Q8 is built on existing intermediate nodes, Tmp7 and Tmp18, both nodes are the project operation the ancestor of base relation that is not taken into consideration as the directly affected node. Thus, only new nodes {Tmp26, Tmp27 and Tmp28} are the set of views to be selected to be new materialized views.

Therefore, the number of nodes to be selected in the dynamic phase is 3 nodes. All nodes are new nodes to support new requirements, whereas the number of nodes to be selected by rerun static approach for all queries is 28 nodes described below. Thereafter, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that only Tmp28 is materialized.

To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes Q1 to Q7 and Q8. Figure 4.31 shows the re-optimized MVPP generated by static approach for Q1 to Q7 and Q8. The MVPP structure in Figure 4.31 provides the same structure as the dynamic approach in Figure 4.30. The comparison of the results from the static approach and the dynamic approach are shown in Table 4.13. The result shows that although total cost
of the static approach equal to the dynamic approach, the number of nodes to be selected by dynamic approach is 3 nodes, which all are new nodes, whereas the number of nodes to be selected by static approach that we have to rerun from the beginning is 28 nodes for all queries, Q1 to Q7 and Q8.

Figure 4.30 The Existing Re-Optimized MVPP with Q8 by Dynamic Approach

Figure 4.31 The Re-Optimized MVPP for Q1-Q7, and Q8 by Static Approach
Table 4.13  The Comparison of the Result from the Static Approach and the Dynamic Approach for the Nothing in Common Data Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes to be Selected</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>28</td>
<td>533,536,140,392</td>
<td>5,891,229,392,788</td>
<td>6,424,765,533,180</td>
</tr>
<tr>
<td>Dynamic</td>
<td>3</td>
<td>533,536,140,392</td>
<td>5,891,229,392,788</td>
<td>6,424,765,533,180</td>
</tr>
</tbody>
</table>

The conclusion for the query identified as nothing in common is that as new queries do not have sharable subexpression with the existing MVPP so new intermediate nodes have to be created to support new query. The existing nodes are not affected by new queries. The only new intermediate nodes are the member of nodes to be the set of views to select to be new materialize view. Therefore, we have not to recalibrate all queries, existing and new queries, again for the nothing in common data set.

4.6.1.3 Analysis Result of the Subsumption Data Set

We add Q9 and Q10, classified as subsumption or totally overlapping, into the existing MVPP. There is no new intermediate node generated because Q9 and Q10 are constructed on the existing intermediate nodes that would be either virtual views or materialized views. Q9 is subgraph of existing MVPP without materialized view in that subgraph whilst Q10 uses the existing materialized view. The MVPP after merging Q9 and Q10 is shown in Figure 4.32.
represents materialized view node selected in static phase for Q1-Q7
represents new materialized view node selected in dynamic phase

Figure 4.32 The Existing Re-Optimized MVPP with Q9 and Q10 by Dynamic Approach

Next, the affected node identification algorithm mentioned in Figure 3.14 in section 3.7.2 is applied to identify the affected nodes. The details of algorithm are list below:

1. For each new query
   1.1 Depth first search from the root to base relations to determine the existing intermediate nodes, $v_i$, used to construct the new query.
   1.2 Calculate weight $w(v)$ of each node $v_i$.

$v_i$, that are conjunctively joined with positive weight or project operation that is not the ancestor of base relation or select operation, are inserted into the list $M_{direct}$.

2. For each node $v_i$ in list $M_{direct}$ search its ancestor node $u_j$, $u_j \notin M_{direct}$, up to the query node
   2.1 calculate weight of node $u_j$
   2.2 if ( weight $v_i >$ weight $u_j$ ) and $u_j$ is existing materialized view then put $u_j$ into list indirectly affected node, $M_{indirect}$
   2.3 if ( weight $v_i <$ weight $u_j$ ) then traverse in bottom-up way to find the node that return maximum weight $u_j$ of each branch.

put $u_j$ into list $M_{indirect}$
We first depth first search to find the existing nodes used to construct Q9 and Q10. The existing nodes used to construct Q9 are {Tmp14, Tmp10, Tmp9, Tmp13, Tmp4, Tmp3, Tmp2, Tmp1 and Tmp12} and Q10 are {Tmp24, Tmp10, Tmp9 and Tmp7}.

The weight \( w(v) \) is calculated as:

\[
w(v) = \sum_{q \in O_v} \left\lfloor \frac{f_q(q)}{(C^q_a(v)} \right\rfloor - \sum_{r \in I_v} \left\lfloor \frac{f_q(r) * (C^r_m(v))}{(C^r_m(v))} \right\rfloor
\]

\( w(v) \) denotes weight of node

\( O_v \) denotes the queries which use view \( v \).

\( C^q_a \) denotes the accessing cost \( a \) for query \( q \) using view \( v \). The cost of answering query \( q \) is the number of rows presented in the relation used to construct \( q \).

\( f_q \) denotes the frequency of executing a query.

\( I_v \) denotes the base relations which are used to produce view \( v \).

\( C^r_m \) denotes the maintenance cost \( m \) for materialized view \( v \) based on base relation \( r \), which is occasionally updated.

\( f_u \) denotes the frequency of updating base relation

We calculate the weight of nodes used to construct Q9 and Q10 for example Tmp14. Tmp14 is derived by four base relations and accessed by Q2 and Q9. Q2 is the existing query which its frequency of executing the query is 6. Q9 is new query which its frequency of executing the query is 4. Then, weight of Tmp14 is \((6 + 4) * 6869560251\) – \(4 * (5 + 1 + 25 + 25 + 150000 + 750000 + 150000 + 227597 + 6869560251)\) = 41,206,850,894. As weight of Tmp14 is positive then Tmp14 is identified as the directly affected node. The weights of existing nodes used to construct Q9 and Q10 are shown in Table 4.14. The details of weight calculation are provided in Appendix D.
Table 4.14  The Weight of the Existing Nodes for Construct Q9 and Q10

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>100</td>
</tr>
<tr>
<td>Tmp2</td>
<td>15</td>
</tr>
<tr>
<td>Tmp3</td>
<td>500</td>
</tr>
<tr>
<td>Tmp4</td>
<td>413</td>
</tr>
<tr>
<td>Tmp7</td>
<td>120,000,000</td>
</tr>
<tr>
<td>Tmp9</td>
<td>36,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>3,962,328</td>
</tr>
<tr>
<td>Tmp12</td>
<td>2,100,000</td>
</tr>
<tr>
<td>Tmp13</td>
<td>4,799,832</td>
</tr>
<tr>
<td>Tmp14</td>
<td>41,206,850,894</td>
</tr>
<tr>
<td>Tmp24</td>
<td>8,193,476,544,806</td>
</tr>
</tbody>
</table>

Later the intermediate nodes, which are the conjunctively joined nodes with positive weight, or project operation that is not the ancestor of base relation or select operation, are the directly affected nodes. From Table 4.14, the directly affected nodes are \{Tmp1, Tmp2, Tmp4, Tmp9, Tmp10, Tmp13, Tmp14 and Tmp24\}. The weight of those nodes is increased because more queries build on them.

Thereafter, we identify the indirectly affected nodes that are the ancestor nodes, with certain weight \( w(v) \), of directly affected nodes described as follows.

Tmp1, its ancestor is Tmp2 that is the directly affected node, so skip to the next node in the list.

Tmp2, its ancestor is Tmp4 that is the directly affected node, so skip to the next node in the list.

Tmp4, its ancestor nodes are Tmp6 and Tmp13. We consider only Tmp6 because Tmp13 is already in the list of directly affected node that we have to identify the indirectly affected node of Tmp13 instead. The weight of ancestor nodes of Tmp4 and Tmp6 are shown in Table 4.15. As weight of Tmp6 which is 369,832 is higher than that of Tmp4 which is 413 shown in Table 4.14, and Tmp6 has the ancestor nodes. Then we continues move up to the query node to look for the ancestors that
provide the maximum weight of each branch. The branch, which includes Tmp6, Tmp8 and Tmp11, returns weight of Tmp11 as the maximum weight. The other ancestor nodes of Tmp6 are Tmp17, Tmp19 and Tmp23. Their weights are negative. Then, the indirectly affected node of Tmp4 is only Tmp11.

Next node in the directly affected node list is Tmp9, its ancestor is Tmp10 that is the directly affected node, so skip to the next node in the list.

Tmp10, its ancestor is Tmp24 that is the directly affected node, so skip to the next node in the list.

Tmp13, its ancestor is Tmp14 that is the directly affected node, so skip to the next node in the list.

Tmp14, its ancestor is only Tmp15. As weight of Tmp15, shown in Table 4.15, is higher than that of Tmp14 then Tmp15 is identified as the indirectly affected node. The last directly affected node is Tmp24, its ancestor is Tmp25. As weight of Tmp25, shown in Table 4.15, is negative then it is not identified as the indirectly affected node.

Table 4.15 The Weight of Ancestor Nodes of Directly Affected Node of Q9, Q10

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp6</td>
<td>369,832</td>
</tr>
<tr>
<td></td>
<td>Tmp8</td>
<td>36,029,759,776</td>
</tr>
<tr>
<td></td>
<td>Tmp11</td>
<td>486,610,492,657</td>
</tr>
<tr>
<td></td>
<td>Tmp17</td>
<td>-3,440,224</td>
</tr>
<tr>
<td></td>
<td>Tmp19</td>
<td>-104,160,300,280</td>
</tr>
<tr>
<td></td>
<td>Tmp23</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>-4,096,769,632,791</td>
</tr>
</tbody>
</table>

Then, the affected nodes are:

Directly affected nodes: Tmp14, Tmp13, Tmp10, Tmp9, Tmp4, Tmp2, Tmp1 and Tmp24

Indirectly affected nodes: Tmp11, Tmp15
Therefore, the number of nodes to be selected in the dynamic phase is 10 nodes, all nodes are the existing nodes, whereas the number of nodes to be selected by running the static approach is 25 nodes described below. Thereafter, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that Tmp13 and Tmp10 are changed from virtual views to materialized views because their weights are increased enough to be materialized to support new requirements.

To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes Q1 to Q7 and Q9, Q10. After we apply our MVPP re-optimization algorithm to the cheapest MVPP, the MVPP structure in Figure 4.33 provides the same structure as the dynamic approach in Figure 4.32. The comparison of result from the static approach and the dynamic approach are shown in Table 4.16. The result shows that although total cost of static approach equal to the dynamic approach, the number of nodes to be selected by the dynamic approach is less than that of static approach that we have to rerun from the beginning for all queries Q1 to Q7, Q9 and Q10.

---

**Figure 4.33** The Re-Optimized MVPP for Q1-Q7, and Q9-Q10 by Static Approach
Table 4.16 The Comparison of the Result from the Static Approach and the Dynamic Approach for the Subsumption Data Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes to be Selected</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>25</td>
<td>561,015,596,786</td>
<td>5,587,305,318,217</td>
<td>6,148,320,915,003</td>
</tr>
<tr>
<td>Dynamic</td>
<td>10</td>
<td>561,015,596,786</td>
<td>5,587,305,318,217</td>
<td>6,148,320,915,003</td>
</tr>
</tbody>
</table>

The conclusion for the query identified as subsumption of the existing MVPP is that the existing node might be changed from the virtual view to materialized view depending on the frequency of executing the queries. If the frequency is higher, it is more likely that the nodes will be materialized. There is no new node created as the new queries can be totally derived from the existing nodes. Therefore, we have not to rerun the materialized view selection using static approach for all queries again.

4.6.1.4 Analysis Result of the Partially Overlapping Data Set

We add Q11 and Q12, classified as partially overlapping, into the existing MVPP. Some new intermediate nodes have to be created as they are not sharable subexpression with the existing query. The subtree of Q11 is constructed on the existing materialized view, Tmp24. Q12 is constructed on the virtual views, Tmp19. Figure 4.34 shows MVPP after Q11 and Q12 are merged into existing MVPP.
represents materialized view node selected in static phase for Q1-Q7

Figure 4.34 The Existing Re-Optimized MVPP with Q11 and Q12 by Dynamic Approach

Next, the affected node identification algorithm is applied to identify the affected nodes as follows:

We first depth search to find the intermediate nodes used to construct Q11 and Q12. The intermediate nodes used to construct Q11 are {Tmp28, Tmp16, Tmp27, Tmp10, Tmp9 and Tmp7}. Tmp27 and Tmp28 are new intermediate nodes, the others are existing intermediate nodes, and Tmp24 is the existing materialized view. The intermediate nodes used to construct Q12 are {Tmp30, Tmp26, Tmp7, Tmp29, Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2 and Tmp1}. Tmp30, Tmp29, Tmp26 are new intermediate nodes and the others are the existing intermediate nodes.

Later, we calculate the weight $w(v)$ of the existing intermediate node used to construct Q11 and Q12. For example, the weight of Tmp17 is $(2 + 2 + 2) \times (16024000000 - 4) \times (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000) = 41,206,850,894$. As weight of Tmp17, conjunctive joined nodes, is positive then it is identified as the directly affected node. The weights of existing nodes used to construct Q11 and Q12 are shown in Table 4.17.
Table 4.17  The Weight of the Existing Nodes for Construct Q11 and Q12

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight (w(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>90</td>
</tr>
<tr>
<td>Tmp2</td>
<td>13</td>
</tr>
<tr>
<td>Tmp3</td>
<td>450</td>
</tr>
<tr>
<td>Tmp4</td>
<td>363</td>
</tr>
<tr>
<td>Tmp5</td>
<td>210,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>469,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>120,000,000</td>
</tr>
<tr>
<td>Tmp9</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>3,051,940</td>
</tr>
<tr>
<td>Tmp16</td>
<td>15,200,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>3,201,359,776</td>
</tr>
<tr>
<td>Tmp18</td>
<td>3,400,000</td>
</tr>
<tr>
<td>Tmp19</td>
<td>-40,065,300,280</td>
</tr>
<tr>
<td>Tmp24</td>
<td>8,193,476,544,806</td>
</tr>
</tbody>
</table>

Later intermediate nodes, which are the conjunctively joined nodes with positive weight or project operation that is not the ancestor of base relation or select operation, are the directly affected nodes. Therefore, from Table 4.17 the directly affected nodes are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp17 and Tmp24\}.

Thereafter, we identify the indirectly affected nodes that are the ancestor nodes, with certain weight \(w(v)\), of the directly affected nodes as follows.

Tmp1, its ancestor is Tmp2 that is the directly affected node, so skip to the next node in the list.

Tmp2, its ancestor is Tmp4 that is the directly affected node, so skip to the next node in the list.

Tmp4, its ancestors are Tmp6 and Tmp13. Tmp6 is the directly affected node then we will identify the indirectly affected node of Tmp6 instead. The weights of ancestor nodes of Tmp4 on branch Tmp13 are shown in Table 4.18. As weight of
Tmp15 is the maximum weight of this branch then Tmp15 is the indirectly affected node.

Tmp6, its ancestors are Tmp8 and Tmp17. Tmp17 is the directly affected node then we will identify the indirectly affected nodes of Tmp17 instead. The weights of ancestor nodes of Tmp6 on branch Tmp8 are shown in Table 4.18. As weight of Tmp11 is the maximum weight of this branch then Tmp11 is the indirectly affected node.

**Table 4.18** The Weight of Ancestor Nodes of Directly Affected Node of Q11, Q12

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>1,799,832</td>
</tr>
<tr>
<td></td>
<td>Tmp14</td>
<td>13,728,609,890</td>
</tr>
<tr>
<td></td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>36,029,759,776</td>
</tr>
<tr>
<td></td>
<td>Tmp11</td>
<td>486,610,492,657</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Tmp19</td>
<td>-40,065,300,280</td>
</tr>
<tr>
<td></td>
<td>Tmp23</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>-4,096,769,632,791</td>
</tr>
</tbody>
</table>

Tmp9, its ancestor is Tmp10 that is the directly affected node, so skip to the next node in the list.

Tmp10, its ancestor is Tmp24 that is the directly affected node, so skip to the next node in the list.

Tmp17, its ancestor nodes are Tmp19 and Tmp23. The weights of Tmp19 and Tmp23 are shown in Table 4.18. As weight of Tmp19 and Tmp23 is negative then Tmp19 and Tmp23 are not the indirectly affected nodes.

The last node in the directly affected node list is Tmp24, its ancestor node is only Tmp25. As weight of Tmp25 is negative as shown in table 4.18, then Tmp25 is not identified as the indirectly affected node.
Then, the affected nodes are:

Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp17 and Tmp24

Indirectly affected nodes: Tmp11, Tmp15

Therefore, the number of nodes to be selected in the dynamic phase is 15 nodes, 10 existing nodes and 5 new created nodes, whereas the number of nodes to be selected by running the static approach is 30 nodes described below. Thereafter, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that Tmp17 is materialized and Tmp6 is un-materialized. Tmp17 is the existing virtual view changed to materialize view.

![Diagram](image)

**Figure 4.35** The Existing Re-Optimized MVPP with Q11 and Q12 by Dynamic Approach after Applying 2PO

Considering in Figure 4.35, Tmp6 is created to support Q1, Q3, Q4 and Q12. When Tmp11 and Tmp17 are materialized because of adding Q11 and Q12, then Q1 and Q4 are rewritten by using Tmp17 instead of Tmp6, Q12 is derived by Tmp17, and Q3 is derived by Tmp11. So, those queries are not derived by Tmp6 anymore.
Therefore Tmp6 is un-materialized. If Tmp6 is materialized, it will provides higher materialized view maintenance cost but the query processing cost is not reduced, reflecting the higher total cost.

To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes Q1 to Q7, Q11 and Q12. We generate 9 MVPPs for 9 queries by Yang et al. algorithm and select the cheapest MVPP shown in Figure 4.36.

![Figure 4.36 The Cheapest MVPP for Q1-Q7, and Q11-Q12 by Static Approach](image)

Later, we apply our MVPP re-optimization algorithm to the cheapest MVPP. The MVPP structure shown in Figure 4.37 provides the same structure as our dynamic approach in Figure 4.35.

Finally, we apply 2PO algorithm to select the set of views to be materialized. Figure 4.37 shows the MVPP with materialized view node generated by rerun static approach for Q1 to Q7, Q11 and Q12. The comparison of results from the static approach and the dynamic are shown in Table 4.19. The result shows that although total cost of static approach equal to the dynamic approach, the number of nodes to be selected by the dynamic approach is less than that of the static approach.
Figure 4.37 The Re-Optimized MVPP for Q1-Q7, and Q11-Q12 by Static Approach

Table 4.19 The Comparison of the Result from the Static Approach and the Dynamic Approach for Partially Overlapping Data Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of nodes to be selected</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>30</td>
<td>3,352,056,780,640</td>
<td>5,593,713,750,508</td>
<td>8,945,770,531,148</td>
</tr>
<tr>
<td>Dynamic</td>
<td>15</td>
<td>3,352,056,780,640</td>
<td>5,593,713,750,508</td>
<td>8,945,770,531,148</td>
</tr>
</tbody>
</table>

The conclusion for the query identified as partially overlapping with the existing MVPP is that the existing node might be changed from virtual view to materialized view and virtual view might be materialized to support the new queries. New intermediate nodes are created to support new queries for the not the sharable subexpression. So, we have not to rerun materialized view selection using static approach for all queries again.
4.6.1.5 Analysis Result of Deleting the Query

Deleting the query from data warehouse is the possible situation when the existing query is no longer required. We implement the deleting scenario by deleting Q3 from the existing MVPP shown in Figure 4.38.

![Figure 4.38 The Re-Optimized MVPP Structure with Deleting Q3 for Dynamic and Static Approach](image)

After Q3 is deleted, we apply the affected node identification algorithm to identify the affected nodes described as follows:

We first depth first search to find the existing nodes used to construct Q3. The nodes used to build Q3 are \{Tmp11, Tmp10, Tmp9, Tmp8, Tmp7, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2 and Tmp1\}. After Q3 is deleted, the nodes used to construct only Q3 are deleted that are Tmp11 and Tmp8; the nodes used to construct Q3 and the other queries are remained and identified as the directly affected nodes.

Later, we calculate the weight \(w(v)\) of the existing node used to construct the query. For example, weight of Tmp6 is \((2+2)*(50000) - (3)*(5+1+25+25+5000)\) that is 184,832. The weights of existing nodes used to construct Q3 are shown in Table 4.20.
Table 4.20 The Weight of the Existing Nodes for Q3

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>45</td>
</tr>
<tr>
<td>Tmp2</td>
<td>4</td>
</tr>
<tr>
<td>Tmp3</td>
<td>225</td>
</tr>
<tr>
<td>Tmp4</td>
<td>138</td>
</tr>
<tr>
<td>Tmp5</td>
<td>120,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>19,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>48,000,000</td>
</tr>
<tr>
<td>Tmp9</td>
<td>12,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>320,776</td>
</tr>
</tbody>
</table>

Later intermediate nodes, which are the conjunctively joined nodes with positive weight, or project operation that is not the ancestor of base relation or select operation, are the directly affected nodes. Therefore, from Table 4.20 the directly affected nodes are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp9 and Tmp10\}.

Thereafter, we identify the indirectly affected nodes that are the ancestor nodes, with certain weight \( w(v) \), of the directly affected node described as follows:

Tmp1, its ancestor is Tmp2 that is the directly affected node, so skip to the next node in the list.

Tmp2, its ancestor is Tmp4 that is the directly affected node, so skip to the next node in the list.

Tmp4, its ancestors are Tmp6 and Tmp13. Tmp6 is the directly affected node, we will identify the indirectly affected node of Tmp6 instead. The weights of ancestor nodes of Tmp4 on branch Tmp13 are shown in Table 4.21. As weight of Tmp15 is the maximum of this branch then it is identified as the indirectly affected node.

Tmp6, its ancestor nodes are Tmp17, Tmp19 and Tmp23. The weights of ancestor nodes of Tmp6 are shown in Table 4.21. As their weight is negative then Tmp6 does not have ancestor node that is identified as the indirectly affected node.

Tmp9, its ancestor is Tmp10 that is the directly affected node, so skip to the next node in the list.
Tmp10, the last node in the directly affected node list, its ancestor nodes are Tmp24 and Tmp25. Their weights are shown in Table 4.21. As weight of Tmp24 is the maximum of this branch then it is identified as the indirectly affected node.

Table 4.21 The Weight of Ancestor Nodes of Directly Affected Node of Q3

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>1,799,832</td>
</tr>
<tr>
<td></td>
<td>Tmp14</td>
<td>13,728,609,890</td>
</tr>
<tr>
<td></td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Tmp17</td>
<td>-3,440,224</td>
</tr>
<tr>
<td></td>
<td>Tmp19</td>
<td>-104,160,300,280</td>
</tr>
<tr>
<td></td>
<td>Tmp23</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Tmp24</td>
<td>1,365,566,544,806</td>
</tr>
<tr>
<td></td>
<td>Tmp25</td>
<td>-4,096,769,632,791</td>
</tr>
</tbody>
</table>

Then, the affected nodes are:

Directly affected nodes: Tmp10, Tmp9, Tmp6, Tmp4, Tmp2 and Tmp1

Indirectly affected nodes: Tmp15 and Tmp24

Therefore, the number of nodes to be selected in the dynamic phase is 8 existing nodes, whereas the number of nodes to be selected by rerun static approach for all queries, Q1, Q2, Q4, Q5, Q6 and Q7, is 23.

Finally, the selection algorithm, 2PO, is applied to select the set of views to be materialized. Figure 4.38 show the re-optimized MVPP with materialized view nodes. In Figure 4.38, there is neither virtual view to be materialized nor materialized view to be un-materialized. The existing materialized view, Tmp6, is still the materialized view to support the other queries, Q1 and Q4. However its weight is reduced from 369,832 to 19,832. The comparison of results from the static approach and dynamic approach are shown in Table 4.22. The result shows that although total cost of the
static approach equal to the dynamic approach, the number of nodes to be selected by
dynamic approach is less than that of the static approach.

Table 4.22 The Comparison of the Result from the Static and the Dynamic Approach
for Deleting Query Q3

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes to be Selected</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>23</td>
<td>533,524,484,878</td>
<td>4,160,323,374,882</td>
<td>4,693,847,859,760</td>
</tr>
<tr>
<td>Dynamic</td>
<td>8</td>
<td>533,524,484,878</td>
<td>4,160,323,374,882</td>
<td>4,693,847,859,760</td>
</tr>
</tbody>
</table>

By the result of experiment and behavior of deleting the query, we classify
deleting the existing query into the subsumption data set because the existing query is
subsumption of MVPP. The weight of existing nodes that are affected is reduced
because the frequency of executing the queries is decreased then the first part of
weight \( w(v) \), \( \sum_{q \in \Omega} |q| \frac{C_q}{C_q(v)} \), is reduced. Therefore, the existing materialized node
that used to construct the deleted query may or may not be un-materialized.

4.6.1.6 Analysis Result of Adding the Query Constructed on All
Base Relations

Moreover, to validate our affected node identification algorithm that
aim to reduce the size of search space, we do further experiment by adding new
query, Q13, which is constructed on all base relations in TPC-H schema. The purpose
to implement this scenario is that the new query constructed on all base relations
would possible to affect to all queries in the existing MVPPs. So, if we identify the
query is affected rather than the intermediate node then all nodes used to derive those
queries are the set of affected nodes.

The dynamic MVPP after Q13 is merged into the existing re-optimized
MVPPP is shown in Figure 4.39.
Figure 4.39 The Existing Re-Optimized MVPP with Q13 by Dynamic Approach

From Figure 4.39, considering how all existing queries are affected by adding Q13 as follows:

**Q1**: Tmp19 and its descendent nodes are the sharable subexpression
**Q2**: Tmp4 and its descendent nodes are the sharable subexpression
**Q3**: Tmp6 and its descendent nodes are the sharable subexpression
**Q4**: Tmp17 and its descendent nodes are the sharable subexpression
**Q5**: Tmp16, Tmp18 are the projection node
**Q6**: Tmp16, Tmp18 are the projection node
**Q7**: Tmp25 and its descendent nodes are the sharable subexpression

Therefore, if we identify the query is affected by requirement changed rather than the intermediate node is affected then all intermediate nodes used to construct all above queries will be the set of affected nodes to be selected in the selection step.

After Q13 is merged into existing MVPP, we apply the affected node identification algorithm to identify the affected node described as follows:
We first depth first search to find the existing nodes used to construct Q13 that are \{Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1, Tmp25, Tmp24, Tmp10, Tmp9, Tmp7, and Tmp12\}.

Thereafter, we calculate the weight $w(v)$ of the existing nodes used to construct Q13. For example Tmp19 is $(2+5)*(32048000000) - (3)*(2003 + 800000 + 1602400000 + 32048000000) = 123,382,393,991$. So the directly affected nodes are \{Tmp19, Tmp17, Tmp6, Tmp4, Tmp2, Tmp1, Tmp24, Tmp10 and Tmp9\}. The weights of the existing node used to construct Q13 are shown in Table 4.23.

**Table 4.23** The Weight of the Existing Nodes for Construct Q13

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight ($w(v)$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>105</td>
</tr>
<tr>
<td>Tmp2</td>
<td>16</td>
</tr>
<tr>
<td>Tmp3</td>
<td>525</td>
</tr>
<tr>
<td>Tmp4</td>
<td>438</td>
</tr>
<tr>
<td>Tmp5</td>
<td>240,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>619,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>120,000,000</td>
</tr>
<tr>
<td>Tmp9</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>3,051,940</td>
</tr>
<tr>
<td>Tmp12</td>
<td>2,250,000</td>
</tr>
<tr>
<td>Tmp16</td>
<td>17,600,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>8,008,559,776</td>
</tr>
<tr>
<td>Tmp18</td>
<td>4,000,000</td>
</tr>
<tr>
<td>Tmp19</td>
<td>56,078,699,720</td>
</tr>
<tr>
<td>Tmp24</td>
<td>8,193,476,544,806</td>
</tr>
<tr>
<td>Tmp25</td>
<td>-3,413,880,382,791</td>
</tr>
</tbody>
</table>

Later the intermediate nodes, which are the conjunctively joined nodes with positive weight, or project operation that is not the ancestor of base relation or select
operation, are identified as the directly affected nodes. Therefore, the directly affected
nodes are \{Tmp19, Tmp17, Tmp6, Tmp4, Tmp2, Tmp1, Tmp24, Tmp10 and Tmp9\}.

Next step, we identify the indirectly affected nodes that are the ancestor nodes,
with certain weight \(w(v)\), of the directly affected nodes described as follows:

Tmp1, its ancestor is Tmp2 that is the directly affected node, so skip to the
next node in the list.

Tmp2, its ancestor is Tmp4 that is the directly affected node, so skip to the
next node in the list.

Tmp4, its ancestors are Tmp6 and Tmp13. Tmp6 is the directly affected node
then we will identify the indirectly affected node of Tmp6 instead. The weights of
ancestor nodes of Tmp4 on branch Tmp13 are shown in Table 4.24. As weight of
Tmp15 is the maximum of this branch then it is identified as the indirectly affected
node.

Tmp6, its ancestors are Tmp8 and Tmp17. Tmp17 is the directly affected node
then we will identify the indirectly affected node of Tmp17 instead. The weights of
ancestor nodes of Tmp6 on branch Tmp8 are shown in Table 4.24. As weight of
Tmp11 is the maximum of this branch then it is identified as the indirectly affected
node.

Tmp9, its ancestor is Tmp10 that is the directly affected node, so skip to the
next node in the list.

Tmp10, its ancestor is Tmp24 that is the directly affected node, so skip to the
next node in the list.

Tmp17, its ancestors are Tmp19 and Tmp23. As Tmp19 is identified as the
directly affected node then we consider Tmp23 only. From Table 4.24, the weight of
Tmp23 is negative then Tmp23 is not the indirectly affected node.

Tmp24, the last node in the directly affected node list, its ancestor is Tmp25.
From Table 4.24, the weight of Tmp25 is negative then Tmp25 is not identified as the
indirectly affected node.
Table 4.24 The Weight of Ancestor Nodes for the Directly Affected Node of Q13

<table>
<thead>
<tr>
<th>Directly Affected Nodes</th>
<th>Ancestor Nodes</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>1,799,832</td>
</tr>
<tr>
<td></td>
<td>Tmp14</td>
<td>13,728,609,890</td>
</tr>
<tr>
<td></td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>36,029,999,776</td>
</tr>
<tr>
<td></td>
<td>Tmp11</td>
<td>486,610,792,657</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>-3,413,880,382,791</td>
</tr>
</tbody>
</table>

Therefore, the affected nodes are:

Directly affected nodes: Tmp19, Tmp17, Tmp6, Tmp4, Tmp2, Tmp1, Tmp24, Tmp10 and Tmp9
Indirectly affected nodes: Tmp11 and Tmp15

Therefore, the number of nodes to be selected in the dynamic phase is 14 nodes, 11 existing nodes and 3 new created nodes, whereas the number of nodes to be selected by the static approach for all queries is 28 nodes described below. Thereafter, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that Tmp19 and Tmp17 are the existing virtual views changed to materialize views whilst Tmp6, the existing materialized view, is un-materialized to be virtual view. Tmp6 is un-materialized because it is not accessed by any queries anymore; Q4 derived by materialized view Tmp17 instead.
represents materialized view node selected in static phase for Q1-Q7
represents materialized view node selected in dynamic phase
represents un-materialized node in dynamic phase

**Figure 4.40** The Existing Re-Optimized MVPP with Q13 by Dynamic Approach after Applying 2PO

To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes Q1 to Q7 and Q13. We generate all MVPPs for those 8 queries by Yang et al. algorithm and select the cheapest MVPP shown in Figure 4.41. Later, we apply our MVPP re-optimization algorithm to the cheapest MVPP thus Q1, Q5, Q6 and Q13 have to be rewritten because their query processing cost is higher than that of the other MVPPs. The MVPP after applying our MVPP re-optimization algorithm in Figure 4.42 provides the same structure as our dynamic approach in Figure 4.40.
Figure 4.41 The Cheapest MVPP for Query Q1 to Q7, and Q13 by Static Approach

Figure 4.42 The Re-Optimized Cheapest MVPP for Query Q1 to Q7, and Q13 by Static Approach

- **Blue dot** represents materialized view node same as static approach for Q1-Q7
- **Green dot** represents additional materialized view node selected by static approach
Finally, we apply 2PO algorithm to select the set of views to be materialized. The comparison of the results from the static approach and the dynamic approach for Q1 to Q7 and Q13 are shown in Table 4.25. The result shows that although the total cost of dynamic approach equal to the static approach the number of nodes to be selected by the dynamic approach is less than that of the static approach.

**Table 4.25** The Comparison of the Result for the Static Approach and the Dynamic Approach for Query Q1-Q7 and Q13

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes to be Selected</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>28</td>
<td>1,164,589,162,793</td>
<td>5,593,713,750,508</td>
<td>6,758,302,913,301</td>
</tr>
<tr>
<td>Dynamic</td>
<td>14</td>
<td>1,164,589,162,793</td>
<td>5,593,713,750,508</td>
<td>6,758,302,913,301</td>
</tr>
</tbody>
</table>

The conclusion for adding the query constructed on all base relations is that we can identify the specific existing nodes that are affected to be the set of views for selection to be materialized. Considering the Figure 4.39, if we identify the query is affected rather than node is affected then all 7 existing queries includes 25 nodes that are affected.

The details of the affected node identification algorithm results for section 4.6.1.3 to 4.6.1.6 are provided in Appendix D.

### 4.6.2 Merging New Requirements into theExisting MVPP Implementation

In the real situation, however, there are several types of new query added simultaneously, so we add all queries Q8-Q13 together into the existing MVPP. The method for merging new requirements into the existing MVPP, mentioned in Figure 3.13 in section 3.7.1, is applied when many queries are added. The details are listed as follows:
1. For every optimal query processing plan for any query, if there is a join operation involved, push select, project operations and aggregate function up along the tree.
2. Create a list of the new queries in descending order based on the result of their query access frequency multiplied by query cost.
3. Merge the new query in the list into existing MVPP according to their order in the list by comparing as follows:
   3.1 If there is sharable subexpression available for new query
       then
       merge to that sharable subexpression
       If sharable subexpression has select operation and new query
       also has the same select operation
       then
       push down select operation of new query
   3.2 If no sharable subexpression then create new node for new query
4. Repeat step 3 for other queries until all queries in the list are merged into existing MVPP.
5. Push down select, project and aggregate functions as deep as possible.
6. Move the first new query to the end of the list.
7. Repeat step 3 to 6 to generate all new MVPPs.
8. Calculate total query processing cost based on base relations and existing materialized views for each MVPP and select the cheapest MVPP.
9. Apply the re-optimized algorithm described in section 3.4 to the cheapest MVPP.

First, for every optimal query plan of the new requirements, select, project operations and aggregate functions are pushed up along the tree, if there is a join operation involved. Thereafter, we calculate the weight which is the frequency of executing the query multiplied with the query cost of queries shown in Table 4.26. We create a list of queries and order them based on descending order of their weight. Then the initial list is \{Q10, Q8, Q11, Q12, Q9 and Q13\}. 
Table 4.26 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost of Q8 to Q13

<table>
<thead>
<tr>
<th>Query</th>
<th>Frequency of Executing Query ($f_q$)</th>
<th>Query Cost</th>
<th>$f_q$ * Query Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q8</td>
<td>6</td>
<td>758,746</td>
<td>4,552,476</td>
</tr>
<tr>
<td>Q9</td>
<td>4</td>
<td>46,008</td>
<td>184,032</td>
</tr>
<tr>
<td>Q10</td>
<td>5</td>
<td>910,519</td>
<td>4,552,595</td>
</tr>
<tr>
<td>Q11</td>
<td>5</td>
<td>575,169</td>
<td>2,875,845</td>
</tr>
<tr>
<td>Q12</td>
<td>2</td>
<td>152,424</td>
<td>304,848</td>
</tr>
<tr>
<td>Q13</td>
<td>5</td>
<td>215</td>
<td>1,075</td>
</tr>
</tbody>
</table>

Thereafter, all queries in the list are merged into the existing re-optimized MVPP, generated in the static phase, shown as Figure 4.23.

Starting with Q10 that has conjunctively join (ORDERS \(\bowtie\) LINEITEM) that already available in the existing MVPP that is Tmp24. However, Tmp24 includes the select operation \(\sigma_{o\_orderdate='1994-01-01' \land o\_orderdate<'1995-01-01'}\ ORDER) then this select operation of Q10 is push down. So, Q10 is merged into the existing MVPP at Tmp24 and no new intermediate node created for Q10. Figure 4.43 (a) show the dynamic MVPP when Q10 is merged into the existing MVPP. Next, when Q8 is merged, as there is no conjunctive joined node for PART and LINEITEM then new intermediate node is introduced. The first dynamic MVPP when Q8 is merged is shown in Figure 4.43 (b). Later, when Q11 is merged, the existing conjunctive joined node is available for subtree of Q11 that is Tmp24. The remaining base relation is PARTSUPP then the new node is introduced as a join operation between Tmp24 and PARTSUPP, \{Tmp24 \(\bowtie\) PARTSUPP\}. The first dynamic MVPP, after Q11 is merged, is shown in Figure 4.43 (c).
The Result of Merging Steps for the First Dynamic MVPP
(c) The first dynamic MVPP after Q11 is merged into the existing MVPP

Figure 4.43 (Continued)
(e) The first dynamic MVPP after Q9 is merged into the existing MVPP

Figure 4.43 (Continued)

Next, when Q12 is merged, the existing conjunctive joined node is available for subtree of Q12 that is Tmp19. The remaining base relation is LINEITEM then the new node is introduced as a join operation between Tmp19 and LINEITEM, \{Tmp19 \bowtie LINEITEM\}. The first dynamic MVPP when Q12 is merged is shown in Figure 4.43 (d). The next query in the list is Q9, when Q9 is merged the existing conjunctive joined node is available for Q9 that is Tmp14. As Q9 is the subsumption of the existing MVPP then no new node is introduced. The dynamic MVPP when Q9 is merged is shown in Figure 4.43(e).

The last query in list is Q13. When Q13 is merged into MVPP that Q12 is already in the MVPP, there are two possible plans shown in Table 4.27.
Table 4.27 The Possible Execution Plans for Q13 in the First Dynamic MVPP

<table>
<thead>
<tr>
<th>Plan</th>
<th>Execution Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plan 1</td>
<td>(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) that is Tmp19, (LINEITEM ⋈ ORDERS ⋈ CUSTOMER) that is Tmp25. Its query processing cost is 869,827,064,180</td>
</tr>
<tr>
<td>Plan 2</td>
<td>((REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) ⋈ LINEITEM) that is Tmp12 and then join with ORDERS ⋈ CUSTOMER. Its query processing cost is 817,446,813,290</td>
</tr>
</tbody>
</table>

As we match the optimal individual plan of query from the leaf node to the root node with the existing MVPP and merge it to the subgraph of MVPP which provides the number of base relations that are conjunctive joined as much as possible. Then, the second plan is chosen, also the second plan provide query processing cost less than that of the first plan. The details of query processing cost of these plans are provided in Appendix B. Figure 4.43 (f) shows the MVPP after Q8-Q13 are merged.

![Figure 4.43](image)

(f) The first dynamic MVPP after all new queries are merged

Figure 4.43 (Continued)
Next step of merging process, we push down the select, project and aggregate functions for the first dynamic MVPP as deep as possible to optimize MVPP. The first dynamic MVPP after optimized is shown in Figure 4.44

Figure 4.44 The First Dynamic MVPP after Optimized, Queries in the List \{ Q10, Q8, Q11, Q12, Q9 and Q13 \}

After the first dynamic MVPP is generated, the first element of the list is moved to the end of the list. So Q10 is moved to the end of list, the list becomes \{Q8, Q11, Q12, Q9, Q13 and Q10\}. We start the second dynamic MVPP with Q8, as there is no conjunctive joined node available in the existing MVPP for base relation PART and LINEITEM then new intermediate node introduced to support Q8. The dynamic MVPP when Q8 is merged is shown in Figure 4.45 (a).
Figure 4.45 The Result of Merging Steps for the Second Dynamic MVPP
(c) The second dynamic MVPP after all new queries are merged

Figure 4.45 (Continued)

Next, when Q11 is merged, the existing conjunctive joined node is available for the subtree of Q11 that is Tmp24. The remaining base relation is PARTSUPP then the new node is introduced as a join operation between Tmp24 and PARTSUPP. The second dynamic MVPP when Q11 is merged is shown in Figure 4.45 (b). We repeat this merging step until all new queries in the list are merged into the existing MVPP. Figure 4.45 (c) shows the second dynamic MVPP after all queries are merged.

Next step of merging process, we push down select, project and aggregate functions for first dynamic MVPP as deep as possible to optimize MVPP. The second dynamic MVPP after optimized is shown in Figure 4.46.

After the second dynamic MVPP is generated, the first element of the list is moved to the end of the list. So, Q8 is moved to the end of list, the list becomes {Q11, Q12, Q9, Q13, Q10 and Q8}. We repeat merging step to generate all new MVPPs. So the list of the last dynamic MVPP is {Q13, Q10, Q8, Q11, Q12 and Q9}. Figure 4.46 to Figure 4.50 show the second dynamic MVPP to the last dynamic MVPP already optimized. The details of merging queries to generate all dynamic MVPPs are provided in Appendix C.
Figure 4.46 The Second Dynamic MVPP: the Queries in the List \{Q8, Q11, Q12, Q9, Q13 and Q10\}

Figure 4.47 The Third Dynamic MVPP: the Queries in the List \{Q11, Q12, Q9, Q13, Q10 and Q8\}
Figure 4.48 The Fourth Dynamic MVPP: the Queries in the List {Q12, Q9, Q13, Q10, Q8 and Q11}

Figure 4.49 The Fifth Dynamic MVPP: the Queries in the List {Q9, Q13, Q10, Q8, Q11 and Q12}
The details of query processing cost of MVPP to plan of Q13 in these MVPP is Figure 4.50. The Sixth Dynamic MVPP: the Queries in the List {Q13, Q10, Q8, Q11, Q12 and Q9}

In Figure 4.49 and 4.50 the fifth and the sixth dynamic MVPP, the execution plan of Q13 in these MVPP is not equal to the first fourth MVPPs, because there is only one plan, Plan 1 in Table 4.27, available for Q13 in the existing MVPP. In Figure 4.44 to 4.48, because Q12 is merged into MVPP before Q13, then there are two possible plans for Q13 as described above.

After all dynamic MVPPs are generated, we calculate the weight of all queries and total query processing cost of MVPP to identify the cheapest one. For example query Q11 with the frequency of executing the query is 5 in Figure 4.44, Q11 accesses the nodes Tmp24 (existing materialized view), Tmp29, Tmp16, Tmp30 and result11. So, the query processing cost of Q11 is (5)*(910519 + 910519 + 800000 + 460,135,200,000 + 575169) that is 2,300,691,981,035. The query processing cost of all dynamic MVPPs are shown in Table 4.28. The details of query processing cost of all dynamic MVPPs are provided in Appendix C.

Therefore, from Table 4.28, we choose the first MVPP as the cheapest MVPP that provides the total query processing cost is 4,902,508,929,085.
Thereafter, we apply our MVPP re-optimization algorithm to the cheapest one. The result is that the query processing cost of Q1 to Q12 is equal to the other MVPPs, and the query processing cost of Q13 is equal to the second to the fourth MVPPs, and less than that of the fifth and the sixth MVPP. Therefore, there is no query have to be rewritten.

In the next section, we will identify the affected node as the member of set of views to be selected by implementing our affected node identification algorithm,
Table 4.28  The Query Processing Cost of All Dynamic MVPPs

<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP</th>
<th>4th MVPP</th>
<th>5th MVPP</th>
<th>6th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>67,303,124,486</td>
<td>67,303,124,486</td>
<td>67,303,124,486</td>
<td>67,303,124,486</td>
<td>67,303,124,486</td>
<td>67,303,124,486</td>
</tr>
<tr>
<td>Q2</td>
<td>2,208,984</td>
<td>2,208,984</td>
<td>2,208,984</td>
<td>2,208,984</td>
<td>2,208,984</td>
<td>2,208,984</td>
</tr>
<tr>
<td>Q4</td>
<td>53,213,742,566</td>
<td>53,213,742,566</td>
<td>53,213,742,566</td>
<td>53,213,742,566</td>
<td>53,213,742,566</td>
<td>53,213,742,566</td>
</tr>
<tr>
<td>Q5</td>
<td>362,760</td>
<td>362,760</td>
<td>362,760</td>
<td>362,760</td>
<td>362,760</td>
<td>362,760</td>
</tr>
<tr>
<td>Q6</td>
<td>3,265,582,968</td>
<td>3,265,582,968</td>
<td>3,265,582,968</td>
<td>3,265,582,968</td>
<td>3,265,582,968</td>
<td>3,265,582,968</td>
</tr>
<tr>
<td>Q7</td>
<td>409,739,463,114</td>
<td>409,739,463,114</td>
<td>409,739,463,114</td>
<td>409,739,463,114</td>
<td>409,739,463,114</td>
<td>409,739,463,114</td>
</tr>
<tr>
<td>Q8</td>
<td>911,790,059,484</td>
<td>911,790,059,484</td>
<td>911,790,059,484</td>
<td>911,790,059,484</td>
<td>911,790,059,484</td>
<td>911,790,059,484</td>
</tr>
<tr>
<td>Q10</td>
<td>9,105,190</td>
<td>9,105,190</td>
<td>9,105,190</td>
<td>9,105,190</td>
<td>9,105,190</td>
<td>9,105,190</td>
</tr>
<tr>
<td>Q11</td>
<td>2,300,691,981,035</td>
<td>2,300,691,981,035</td>
<td>2,300,691,981,035</td>
<td>2,300,691,981,035</td>
<td>2,300,691,981,035</td>
<td>2,300,691,981,035</td>
</tr>
<tr>
<td>Q12</td>
<td>311,554,998,998</td>
<td>311,554,998,998</td>
<td>311,554,998,998</td>
<td>311,554,998,998</td>
<td>311,554,998,998</td>
<td>311,554,998,998</td>
</tr>
<tr>
<td>Q13</td>
<td>817,446,813,290</td>
<td>817,446,813,290</td>
<td>817,446,813,290</td>
<td>817,446,813,290</td>
<td>869,827,064,180</td>
<td>869,827,064,180</td>
</tr>
<tr>
<td>Total</td>
<td>4,902,508,929,085</td>
<td>4,902,508,929,085</td>
<td>4,902,508,929,085</td>
<td>4,902,508,929,085</td>
<td>4,954,889,179,975</td>
<td>4,954,889,179,975</td>
</tr>
</tbody>
</table>
4.6.3 The Affected Node Identification Algorithm Implementation

In this section, we describe the implementation of our affected node identification algorithm mentioned in 3.7.2. The algorithm aims to reduce the size of search space of dynamic MVPP for selecting views to be materialized. It is applied to the dynamic re-optimized MVPP generated in the previous section as shown in Figure 4.44.

To identify the directly affected node

Initial list of the directly affected node $M_{direct} = \phi$

Start with Q8, we do depth first search to find the nodes that are used to build query Q8. The existing nodes used to build Q8 are {Tmp18, Tmp7}

Q9: the existing nodes used to build Q9 are {Tmp14, Tmp10, Tmp9, Tmp13, Tmp4, Tmp3, Tmp2, Tmp1 and Tmp12}.

Q10: the existing nodes used to build Q10 are {Tmp24, Tmp10, Tmp9 and Tmp7}.

Q11: the existing nodes used to build Q11 are {Tmp16, Tmp24, Tmp10, Tmp9 and Tmp7}.

Q12: the existing nodes used to build Q12 are {Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1 and Tmp7}.

Q13: the existing nodes used to build Q13 are {Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1, Tmp7, Tmp10, Tmp9 and Tmp12}.

So the existing nodes used to build new queries Q8-Q13 are {Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp9, Tmp10, Tmp12, Tmp13, Tmp14, Tmp16, Tmp17, Tmp18, Tmp19, Tmp24}.

Next we calculate weight $w(v)$ for the existing nodes used to construct new queries. For example Tmp19, it is accessed by Q1, Q12 and Q13 which their frequencies of executing the query are 2, 2 and 5, respectively. Tmp19 is constructed from nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17 and Tmp18.. The cost of each node and Tmp19 is 5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000, respectively. Therefore, its weight is calculated as $(2 + 2 + 5) * (32048000000) - (5) * (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 200000 + 32048000000)$ that is 120,174,699,720.
From the result, Tmp19 is possible to be materialized view as its weight is positive. In static phase, shown Figure 4.23, only Q1 is used Tmp19, then the weight of Tmp19 is negative that is (-3,206,804,006). The weight of Tmp19 is changed from negative in the static phase to positive in the dynamic phase because there are additional queries, Q12 and Q13, are constructed on Tmp19.

The weights of the existing nodes used to construct the new queries are shown in Table 4.29.

### Table 4.29 The Weight of the Existing Nodes for Constructing All New Queries

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight (w(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>135</td>
</tr>
<tr>
<td>Tmp2</td>
<td>22</td>
</tr>
<tr>
<td>Tmp3</td>
<td>675</td>
</tr>
<tr>
<td>Tmp4</td>
<td>588</td>
</tr>
<tr>
<td>Tmp5</td>
<td>260,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>719,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>228,000,000</td>
</tr>
<tr>
<td>Tmp9</td>
<td>51,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>6,238,298</td>
</tr>
<tr>
<td>Tmp12</td>
<td>2,850,000</td>
</tr>
<tr>
<td>Tmp13</td>
<td>4,799,832</td>
</tr>
<tr>
<td>Tmp14</td>
<td>41,206,850,894</td>
</tr>
<tr>
<td>Tmp16</td>
<td>23,200,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>11,213,359,776</td>
</tr>
<tr>
<td>Tmp18</td>
<td>5,600,000</td>
</tr>
<tr>
<td>Tmp19</td>
<td>120,174,699,720</td>
</tr>
<tr>
<td>Tmp24</td>
<td>15,021,386,544,806</td>
</tr>
</tbody>
</table>

Later intermediate nodes, which are the conjunctively joined nodes with positive weight, or project operation that is not the ancestor of base relation or select operation, are inserted into the list of directly affected node $M_{direct}$. Therefore, from
Table 4.29, the directly affected nodes are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp13, Tmp14, Tmp17, Tmp19 and Tmp24\}. Tmp3, Tmp5, Tmp7, Tmp12, Tmp16 and Tmp18 are the projection nodes and the ancestor of base relations, so they are not identified as the directly affected node.

**To identify the indirectly affected node**

Next step, we identify the indirectly affected nodes that are the ancestor nodes, with certain weight \(w(v)\), of the directly affected nodes described as follows:

Tmp1, its ancestor is Tmp2 that is already in \(M_{\text{direct}}\), so skip to the next node in the list.

Tmp2, its ancestor is Tmp4 that is already in \(M_{\text{direct}}\), so skip to the next node in the list.

Tmp4, its ancestor nodes are Tmp13 and Tmp6 that are already in \(M_{\text{direct}}\), so skip to the next node in the list.

Tmp6, its ancestor nodes are Tmp8 and Tmp17, as Tmp17 already in \(M_{\text{direct}}\) then we will consider the indirectly affected node of Tmp17 later. From Tmp6, we move up to Tmp8, the ancestor node of Tmp8 is Tmp11 only. We compare the weight of node Tmp6, Tmp8 and Tmp11 to identify the maximum weight of this branch. The weights of these nodes are shown in Table 4.30. As the weight of Tmp11 is the maximum weight of this branch then Tmp11 is identified as the indirectly affected node.

Tmp9, its ancestor is Tmp10 only that is already in the directly list, so we skip to the next node in the list.

Tmp10, its ancestors are Tmp24, Tmp11 and Tmp14 that all are already in the list, so we skip to the next node.

Tmp13, its ancestor is only Tmp14 that is already in the list.

Tmp14, its ancestor is Tmp15. As weight of Tmp15, shown in Table 4.30, is higher than that of Tmp14 then Tmp15 is the indirectly affected node.

Tmp17, its ancestors are Tmp19 and Tmp23. The weight of Tmp23 is negative, shown in Table 4.30, then Tmp23 is not taken into consideration. Tmp19 is already in the list of directly affected node then there is no the indirectly affected node of Tmp17.
Tmp19 does not have the existing ancestor node. Then we skip to the next node in the list. Tmp24, the last node in $M_{direct}$, its ancestor node is only Tmp25. From Table 4.30, as weight of Tmp25 is negative then it is not the indirectly affected node.

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>36,029,759,776</td>
</tr>
<tr>
<td></td>
<td>Tmp11</td>
<td>486,610,492,657</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Tmp23</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>-4,096,769,632,791</td>
</tr>
</tbody>
</table>

Therefore, the existing nodes identified as affected nodes includes

Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp13, Tmp14, Tmp17, Tmp19, and Tmp24

Indirectly affected nodes: Tmp11, Tmp15

The new intermediate nodes are created to support the new requirements are Tmp26, Tmp27, Tmp28, Tmp29, Tmp30, Tmp31, Tmp32, Tmp33, Tmp34 and Tmp35.

Therefore, the total number of nodes to be selected in the dynamic phase is 23, 13 existing node and 10 new nodes, whereas the number of nodes to select by the static approach, that is presenting in section 4.8, is 35 nodes. Those 23 nodes are the member of set of views to be materialized or un-materialized for the materialized view selection process described in the next section.

In the next section, we are presenting the implementation of selection algorithm, 2PO, to select the set of views to be materialized or un-materialized.
4.7 Two-Phase Optimization for Dynamic Materialized View Selection

We use the Two-Phase Optimization (2PO) algorithm to select the set of views to be materialized. The search space is the MVPP after new requirements are merged shown in Figure 4.44. We first map all nodes in DAG to the binary string using depth first search. There are three types of node to map to binary string regarding to the algorithm in Figure 3.15 in section 3.8. The binary string of 1s and 0s represent views which will and will not be materialized, respectively. The mapping rule is implemented as follows:

1) The affected node, both the directly and the indirectly affected node, and new created nodes are mapped into binary string.

   The list of affected node is \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp13, Tmp14, Tmp17, Tmp19, Tmp24, Tmp11, and Tmp15\}.

   The list of new created nodes is \{Tmp26, Tmp27, Tmp28, Tmp29, Tmp30, Tmp31, Tmp32, Tmp33, Tmp34 and Tmp35\}.

   These nodes are initially set to 0.

2) All existing materialized view node which is not identified as the affected node that is Tmp21 is fixed to 1. The reason to fix Tmp21 to 1 is that Tmp21 is always the materialized view to support the existing requirement.

3) The other nodes are fixed to 0. The nodes are \{Tmp3, Tmp5, Tmp7, Tmp8, Tmp12, Tmp16, Tmp18, Tmp20, Tmp22, Tmp23 and Tmp25\}. They are the virtual views that have not been affected by changing the requirements.

Therefore searching through the DAG in Figure 4.44 using depth first search, we obtain the mapping array as follow:

\[\text{[Tmp19,0], [Tmp18,0], [Tmp17,0], [Tmp16,0], [Tmp6,0], [Tmp5,0], [Tmp4,0], [Tmp3,0], [Tmp2,0], [Tmp1,0], [Tmp15,0], [Tmp14,0], [Tmp10,0], [Tmp9,0], [Tmp13,0], [Tmp12,0], [Tmp11,0], [Tmp8,0], [Tmp23,0], [Tmp21,1], [Tmp20,0], [Tmp22,0], [Tmp25,0], [Tmp24,0], [Tmp28,0], [Tmp26,0], [Tmp27,0], [Tmp30,0], [Tmp29,0], [Tmp32,0], [Tmp31,0], [Tmp35,0], [Tmp34,0], [Tmp33,0]}\]

So the binary string of above mapping is

\{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}
The result by 2PO algorithm is \{1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1\} indicates that Tmp10, Tmp11, Tmp13, Tmp15, Tmp17, Tmp19, Tmp21, Tmp24, Tmp27 and Tmp28 are materialized views. The meaning of each materialized view described as follows:

- Tmp11, Tmp15, Tmp21 and Tmp24 are the materialized view. They are the existing materialized views selected in the static phase and also still be the materialized views in dynamic phase.

- Tmp10, Tmp13, Tmp17, and Tmp19 are the virtual views in the static phase and changed to materialized views in the dynamic phase.

- Tmp27 and Tmp28 are the new materialized views to support the new queries.

- Tmp6 is the existing materialized view in the static phase and it is un-materialized in the dynamic phase because Tmp17 and Tmp19 are materialized then Tmp6 is not used by any query anymore.

<table>
<thead>
<tr>
<th>Query number 1 (Q1)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp19, result1</td>
<td></td>
<td>640,960</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 2 (Q2)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp15, result2</td>
<td></td>
<td>2,208,984</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 3 (Q3)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp11, result3</td>
<td></td>
<td>2,550,562</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 4 (Q4)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp17, Tmp12, Tmp23, result4</td>
<td></td>
<td>50,007,659,040</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 5 (Q5)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp21, result5</td>
<td></td>
<td>362,760</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 6 (Q6)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp21, Tmp5, Tmp22, result6</td>
<td></td>
<td>3,265,582,968</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 7 (Q7)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp24, Tmp12, Tmp25, result7</td>
<td></td>
<td>409,739,463,114</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 8 (Q8)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp28, result8</td>
<td></td>
<td>9,104,952</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 9 (Q9)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp13, result9</td>
<td></td>
<td>27,479,456,156</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 10 (Q10)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp24, result10</td>
<td></td>
<td>9,105,190</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 11 (Q11)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp24, Tmp29, Tmp16, Tmp30 and result11</td>
<td></td>
<td>2,300,691,981,035</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query number 12 (Q12)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp19, Tmp31, Tmp27, Tmp32 and result12</td>
<td></td>
<td>244,236,102,064</td>
</tr>
</tbody>
</table>
### Table 4.31 (Continued)

<table>
<thead>
<tr>
<th>Query number 13 (Q13)</th>
<th>Access from Node</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp19, Tmp31, Tmp27, Tmp32, Tmp33, Tmp10, Tmp34, Tmp35 and result13</td>
<td>649,149,570,955</td>
<td></td>
</tr>
</tbody>
</table>

**Total query processing cost**: 3,684,593,788,740

### Table 4.32  The Maintenance Cost of Dynamic Approach for All Queries

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Number of Base Relations</th>
<th>Constructed from Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp10</td>
<td>1</td>
<td>Tmp9, Tmp10</td>
<td>1,727,597</td>
</tr>
<tr>
<td>Tmp11</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11</td>
<td>1,426,977,515,570</td>
</tr>
<tr>
<td>Tmp13</td>
<td>3</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13</td>
<td>2,700,168</td>
</tr>
<tr>
<td>Tmp15</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15</td>
<td>1,414,630,939,520</td>
</tr>
<tr>
<td>Tmp17</td>
<td>4</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp6, Tmp16, Tmp17</td>
<td>6,413,040,224</td>
</tr>
<tr>
<td>Tmp19</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp6, Tmp16, Tmp17, Tmp18, Tmp19</td>
<td>168,257,300,280</td>
</tr>
<tr>
<td>Tmp21</td>
<td>2</td>
<td>Tmp16, Tmp18, Tmp20, Tmp21</td>
<td>14,512,800,000</td>
</tr>
<tr>
<td>Tmp24</td>
<td>2</td>
<td>Tmp7, Tmp9, Tmp10, Tmp24</td>
<td>2,731,179,455,194</td>
</tr>
<tr>
<td>Tmp27</td>
<td>1</td>
<td>Tmp7, Tmp27</td>
<td>12,000,000</td>
</tr>
<tr>
<td>Tmp28</td>
<td>2</td>
<td>Tmp7, Tmp27, Tmp18, Tmp28</td>
<td>303,928,502,336</td>
</tr>
</tbody>
</table>

**Total materialized view maintenance cost**: 6,065,915,980,889
The total query processing cost after the set of views are selected by 2PO is 3,684,593,788,740 as shown in Table 4.31. The total materialized view maintenance cost is 6,065,915,980,889 as shown in Table 4.32. The total cost which is summation of query processing cost and maintenance cost is 9,750,509,769,629. Figure 4.51 is represented the dynamic MVPP after 2PO is applied.

![Dynamic MVPP for Q8-Q13 after Applying 2PO](image)

**Figure 4.51** The Dynamic MVPP for Q8-Q13 after Applying 2PO

In the next section, we are presenting the evaluation of our dynamic approach by rerun static approach of all requirements, the existing queries and the new queries.
4.8 Result and Analysis of Dynamic Materialized View Selection

To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes the initial requirements, Q1-Q7, and new requirements, Q8-Q13. We build MVPP algorithm by Yang et al., the implementation step mentioned in section 4.3, to generate 13 MVPPs for 13 queries and select the cheapest MVPP. The cheapest MVPP for Q1-Q13 is shown in Figure 4.52.

![Figure 4.52 The Cheapest MVPP of Q1-Q13 by Static Approach](image)

Next, we apply our MVPP re-optimization algorithm to the cheapest MVPP, then Q1, Q5, Q6, Q12 and Q13 have to be rewritten as their query processing cost is higher than that of other MVPPs. After the MVPP re-optimization algorithm is applied to the cheapest MVPP, the result is shown in Figure 4.53. The MVPP structure after applying our re-optimized algorithm, shown in Figure 4.53, provides the same structure as our dynamic approach in Figure 4.51.
Figure 4.53 The Re-Optimized MVPP of Q1-Q13 by Static Approach

Table 4.33 shows the comparison of the result from the dynamic approach and the static approach. We compare the number of nodes to be selected to be materialized. The number of nodes infers to the size of search space. The number of node for dynamic approach is 23 nodes whereas for the static approach is 35 nodes, although the total cost of dynamic materialized view selection equal to the static approach.

Table 4.33 The Comparison of the Result from the Static Approach and the Dynamic Approach for All Queries

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>35</td>
<td>3,684,593,788,740</td>
<td>6,065,915,980,889</td>
<td>9,750,509,769,629</td>
</tr>
<tr>
<td>Dynamic</td>
<td>23</td>
<td>3,684,593,788,740</td>
<td>6,065,915,980,889</td>
<td>9,750,509,769,629</td>
</tr>
</tbody>
</table>
4.9 The Second Experiment for Dynamic Materialized View Selection

4.9.1 Static Materialized View Selection for the Initial Requirements

The query set of initial requirements include Q4, Q15, Q22, Q33, Q40, Q43 and Q50 shown in Figure 4.54. The details of their SQL statements are provided in Appendix F.

(a) Query Q4  
(b) Query Q15  
(c) Query Q22  
(d) Query Q33  
(e) Query Q40

Figure 4.54 The Relational Algebra Query Tree for the Initial Queries: Q4, Q15, Q22, Q33, Q40, Q43 and Q50
Figure 4.54 (Continued)

We order the queries in descending order of the query access frequency multiplied with query cost shown in Table 4.34. Then, the order of queries for the first MVPP is \{Q4, Q33, Q22, Q15, Q43, Q40 and Q50\} and the last order is \{Q50, Q4, Q33, Q22, Q15, Q43 and Q40\}.
Table 4.34 The Query Access Frequency, Query Cost, and Query Access Frequency

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Access Frequency ($f_q$)</th>
<th>Query Cost</th>
<th>$f_q$ * Query Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4</td>
<td>6</td>
<td>3,793,296</td>
<td>22,759,776</td>
</tr>
<tr>
<td>Q15</td>
<td>5</td>
<td>160,232</td>
<td>801,160</td>
</tr>
<tr>
<td>Q22</td>
<td>3</td>
<td>575,169</td>
<td>1,725,507</td>
</tr>
<tr>
<td>Q33</td>
<td>6</td>
<td>759,474</td>
<td>4,556,844</td>
</tr>
<tr>
<td>Q40</td>
<td>4</td>
<td>6,492</td>
<td>25,968</td>
</tr>
<tr>
<td>Q43</td>
<td>7</td>
<td>22,778</td>
<td>159,446</td>
</tr>
<tr>
<td>Q50</td>
<td>5</td>
<td>215</td>
<td>1,075</td>
</tr>
</tbody>
</table>

The query processing costs of all MVPPs for query set \{Q4, Q15, Q22, Q33, Q40, Q43 and Q50\} are computed and shown in Table 4.35. The cheapest MVPP is the fifth MVPP as shown in Figure 4.55.

Figure 4.55 The Cheapest MVPP for the Second Experiment: the Queries in the List \{Q43, Q40, Q50 Q4, Q33, Q22 and Q15\}
<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP</th>
<th>4th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q22</td>
<td>$12,039,079,457,322$</td>
<td>$12,039,079,457,322$</td>
<td>$4,315,574,657,322$</td>
<td>$12,039,079,457,322$</td>
</tr>
<tr>
<td>Q40</td>
<td>$134,600,384,632$</td>
<td>$11,570,291,056$</td>
<td>$134,600,384,632$</td>
<td>$672,007,903,472$</td>
</tr>
<tr>
<td>Q43</td>
<td>$28,735,371,933,681$</td>
<td>$28,735,371,933,681$</td>
<td>$10,713,860,733,681$</td>
<td>$29,065,873,503,681$</td>
</tr>
<tr>
<td>Q50</td>
<td>$20,525,276,724,865$</td>
<td>$20,525,276,724,865$</td>
<td>$20,525,276,724,865$</td>
<td>$20,761,338,241,345$</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$98,063,963,179,496$</td>
<td>$83,344,165,360,832$</td>
<td>$72,318,947,179,496$</td>
<td>$85,242,925,459,728$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Query</th>
<th>5th MVPP</th>
<th>6th MVPP</th>
<th>7th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(cheapest)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4</td>
<td>$18,207,907,119,552$</td>
<td>$18,207,907,119,552$</td>
<td>$18,207,907,119,552$</td>
</tr>
<tr>
<td>Q22</td>
<td>$4,315,574,657,322$</td>
<td>$12,039,079,457,322$</td>
<td>$12,039,079,457,322$</td>
</tr>
<tr>
<td>Q33</td>
<td>$3,656,709,383,196$</td>
<td>$3,656,709,383,196$</td>
<td>$3,656,709,383,196$</td>
</tr>
<tr>
<td>Q40</td>
<td>$11,570,291,056$</td>
<td>$11,570,291,056$</td>
<td>$11,570,291,056$</td>
</tr>
<tr>
<td>Q43</td>
<td>$28,735,371,933,681$</td>
<td>$28,735,371,933,681$</td>
<td>$28,735,371,933,681$</td>
</tr>
<tr>
<td>Q50</td>
<td>$870,692,174,234$</td>
<td>$176,201,501,795*$</td>
<td>$176,201,501,795*$</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>$37,944,564,810,201$</td>
<td>$62,995,090,137,762$</td>
<td>$62,995,090,137,762$</td>
</tr>
</tbody>
</table>

Note: * query processing cost of nth MVPP less than the cheapest MVPP
We apply the MVPP re-optimization algorithm to the cheapest MVPP. The query processing cost of queries of the cheapest MVPP are compared with the other MVPPs. The result shows that Q50 of the sixth and the seventh MVPP is less than that of the cheapest MVPP, so Q50 is possible to be rewritten.

When we match the individual plan of Q50 with the cheapest MVPP, Tmp12 is the common subexpression that consists of the most number of base relations that are (REGION ▷ NATION ▷ SUPPLIER ▷ PARTSUPP ▷ PART) the same as individual plan of Q50 from the base relations to the query node. The next base relations matched with individual plan are LINEITEM and ORDERS that they are already joined conjunctively, Tmp18. As there is no another plan for Q50 in the cheapest MVPP then the execution plan of Q50 in the cheapest MVPP is optimal when Q50 is sharing the subexpression with other queries. Therefore, the cheapest MVPP is the optimal MVPP.

Next, 2PO is applied to the optimal MVPP to select the set of views to be materialized. The result of 2PO is that \{Tmp9, Tmp11, Tmp15, Tmp18, Tmp19, Tmp26 and Tmp27\} are materialized shown in Figure 4.56. 

![Diagram](image_url)

Figure 4.56 The Optimal MVPP with Materialized Views for Initial Requirements
The MVPP in Figure 4.56 is the initial search space for dynamic phase. The query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized-views and materialized views selected by 2PO algorithm of the cheapest MVPP are shown in Table 4.36.

**Table 4.36** The Query Processing Cost, Maintenance Cost and Total Cost of the Optimal MVPP

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>37,944,564,810,201</td>
<td>0</td>
<td>37,944,564,810,201</td>
</tr>
<tr>
<td>All-materialized view</td>
<td>30,029,776</td>
<td>36,413,780,040,394</td>
<td>36,413,810,070,170</td>
</tr>
<tr>
<td>2PO</td>
<td>1,836,434,674,685</td>
<td>14,884,264,303,377</td>
<td>16,720,698,978,062</td>
</tr>
</tbody>
</table>

In conclusion, Q50 is possible to be rewritten as its query processing cost of the cheapest MVPP is more than that of the sixth and the seventh MVPP. After the MVPP re-optimization algorithm is applied, Q50 is forced to use the common subexpression that is available in the cheapest MVPP, and Q50 is still constructed with the same query plan of the cheapest MVPP. Therefore, the cheapest MVPP is the optimal MVPP.

### 4.9.2 Dynamic Materialized View Selection for the Second Experiment

The query set for the dynamic phase consists of Q3, Q6, Q28, Q30, Q31 and Q47. Their relation algebra query trees are shown in Figure 4.57. The details of their SQL statements are provided in Appendix F.
Figure 4.57  The Relational Algebra Query Tree for the New Queries: Q3, Q6, Q28, Q30, Q32 and Q47
The new queries are merged into the existing MVPP derived from the static phase, Figure 4.56, according to the merging steps mentioned in section 3.7.1. The result of the optimal dynamic MVPP is shown in Figure 4.58.

![Diagram](image-url)

*represents materialized view node selected in static phase for initial requirements

**Figure 4.58** The Optimal Dynamic MVPP Constructed by Merging Steps

After the dynamic MVPPs are generated and the optimal one is selected, the affected node identification algorithm is applied. The details of the algorithm are provided in Appendix D. The affected nodes are shown as follows:

**Directly affected nodes:** Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp9, Tmp11, Tmp14, Tmp15, Tmp16, Tmp17, Tmp18, Tmp27

**Indirectly affected nodes:** Tmp19, Tmp26

Therefore, the number of nodes to be selected by 2PO in the dynamic phase is 21 nodes, 15 existing nodes and 6 new created nodes.
Thereafter, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that the existing materialized views {Tmp9, Tmp11, Tmp15, Tmp18, Tmp19, Tmp26 and Tmp27}, the virtual views {Tmp6 and Tmp17}, and new virtual views {Tmp29, Tmp30} are materialized. Figure 4.59 shows the dynamic MVPP with materialized views selected by 2PO.

![Figure 4.59](image-url)  

Figure 4.59  The Dynamic MVPP after Applying 2PO
To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes the initial requirements \{Q4, Q15, Q22, Q33, Q40, Q43 and Q50\} and the new requirements \{Q3, Q6, Q28, Q30, Q31 and Q47\}. The results of the static approach, after applying our MVPP re-optimization algorithm and 2PO to select the set of views to materialized, is shown in Figure 4.60. According to the result, \{Tmp3, Tmp4, Tmp12, Tmp14, Tmp21, Tmp23, Tmp24, Tmp29, Tmp33, Tmp34 and Tmp35\} are the materialized views.

![Figure 4.60](image)

Figure 4.60 The Re-Optimized MVPP by Static Approach for All Queries

It can be seen that the MVPP structure of the static approach shown in Figure 4.60 is not the same as the structure of the dynamic MVPP shown in Figure 4.59. The execution plan of Q31 and Q47 in the MVPP are different between the two approaches. As when merging the new queries into the existing MVPP, the relevance of the new queries to the existing MVPP has to be considered. For instance, when Q31 and Q47 are merged into existing MVPP in the dynamic phase, Q31 and Q47 have to be constructed by using the common subexpressions that are available in the existing MVPP as well as their common subexpression. For the static approach, the...
execution plans of Q31 and Q47 have sharable subexpression, and both are constructed using optimal individual plan. For this reason, the static approach, before selecting the set of views to be materialized, the execution plans of Q31 and Q47 that provide the minimal query processing cost have to be chosen.

The query processing cost, materialized view maintenance cost and total cost of the static and the dynamic approach are shown in Table 4.37.

**Table 4.37**  The Comparison of the Results from the Static Approach and the Dynamic Approach for the Second Experiment

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>35</td>
<td>3,429,632,748,193</td>
<td>15,862,487,308,569</td>
<td>19,292,120,056,762</td>
</tr>
<tr>
<td>Dynamic</td>
<td>21</td>
<td>3,838,202,906,833</td>
<td>14,960,158,671,142</td>
<td>18,798,361,577,975</td>
</tr>
</tbody>
</table>

In conclusion, the static approach requires 35 intermediate nodes to be selected while our dynamic approach requires 21 intermediate nodes to be selected, 15 existing nodes and 6 new nodes.

The total cost after 2PO is applied is that the dynamic approach provides total cost less than the static approach. The static approach provide the lower query processing because the execution plan of Q31 and Q47 in MVPP are close to their optimal individual plan. However, dynamic approach provides the lower materialized view maintenance cost because the new queries are constructed on the available sharable common subexpression in the existing MVPP that is the benefit to reduce the maintenance cost.

The query processing cost and materialized view maintenance cost for the dynamic approach are shown in Table 4.38 and 4.39, respectively. The query processing cost and materialized view maintenance cost of the static approach are shown in Table 4.40 and 4.41, respectively.
### Table 4.38 The Query Processing Cost of the Dynamic Approach

<table>
<thead>
<tr>
<th>Query</th>
<th>( f_q )</th>
<th>Accessed nodes</th>
<th>Cost of Query Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp26(materialized view), result4</td>
<td>45,519,552</td>
</tr>
<tr>
<td>Q15</td>
<td>5</td>
<td>Tmp5, Tmp27(materialized view), Tmp28, result15</td>
<td>8,013,252,320</td>
</tr>
<tr>
<td>Q22</td>
<td>3</td>
<td>Tmp10, Tmp20, Tmp19(materialized view), result22</td>
<td>345,105,451,014</td>
</tr>
<tr>
<td>Q33</td>
<td>6</td>
<td>Tmp15(materialized view), result33</td>
<td>9,113,688</td>
</tr>
<tr>
<td>Q40</td>
<td>4</td>
<td>Tmp9(materialized view), Tmp11(materialized view), Tmp12, result40</td>
<td>5,155,298,792</td>
</tr>
<tr>
<td>Q43</td>
<td>7</td>
<td>Tmp5, Tmp19(materialized view), Tmp10, Tmp20, Tmp21, Tmp22, Tmp23 and result43</td>
<td>1,449,432,585,629</td>
</tr>
<tr>
<td>Q50</td>
<td>5</td>
<td>Tmp9(materialized view), Tmp11(materialized view), Tmp12, Tmp18(materialized view), Tmp22, Tmp24, Tmp25, result50</td>
<td>28,673,453,690</td>
</tr>
<tr>
<td>Q3</td>
<td>4</td>
<td>Tmp29(materialized view), result3</td>
<td>30,346,368</td>
</tr>
<tr>
<td>Q6</td>
<td>7</td>
<td>Tmp27(materialized view), result6</td>
<td>2,243,248</td>
</tr>
<tr>
<td>Q28</td>
<td>5</td>
<td>Tmp9(materialized view), result28</td>
<td>321,720</td>
</tr>
<tr>
<td>Q30</td>
<td>4</td>
<td>Tmp13, Tmp17(materialized view), Tmp31, Tmp30(materialized view), Tmp32 and result30</td>
<td>1,131,696,008,452</td>
</tr>
<tr>
<td>Q31</td>
<td>5</td>
<td>Tmp6(materialized view), Tmp18(materialized view), Tmp33, result31</td>
<td>5,763,780,255</td>
</tr>
<tr>
<td>Q47</td>
<td>5</td>
<td>Tmp15(materialized view), Tmp17(materialized view), Tmp34, result47</td>
<td>864,275,532,105</td>
</tr>
</tbody>
</table>

**Total query processing cost** 3,838,202,906,833

### Table 4.39 The Maintenance Cost of the Dynamic Approach

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Number of Base Relation</th>
<th>Derived by Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>3</td>
<td>Tmp1, 2, 3, 4, 5, 6</td>
<td>180,168</td>
</tr>
<tr>
<td>Tmp9</td>
<td>4</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td>6,413,681,184</td>
</tr>
<tr>
<td>Tmp11</td>
<td>1</td>
<td>Tmp10, 11</td>
<td>400,000</td>
</tr>
</tbody>
</table>
Table 4.39 (Continued)

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Number of Base Relation</th>
<th>Derived by Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp15</td>
<td>5</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 15</td>
<td>3,047,265,055,480</td>
</tr>
<tr>
<td>Tmp17</td>
<td>1</td>
<td>Tmp16, 17</td>
<td>1,727,597</td>
</tr>
<tr>
<td>Tmp18</td>
<td>2</td>
<td>Tmp13, 14, 16, 17, 18</td>
<td>1,726,713,034,618</td>
</tr>
<tr>
<td>Tmp19</td>
<td>3</td>
<td>Tmp7, 13, 14, 16, 17, 18, 19</td>
<td>3,970,477,551,927</td>
</tr>
<tr>
<td>Tmp26</td>
<td>2</td>
<td>Tmp7, 13, 14, 26</td>
<td>6,069,299,200,000</td>
</tr>
<tr>
<td>Tmp27</td>
<td>2</td>
<td>Tmp7, 10, 11, 27</td>
<td>64,095,200,000</td>
</tr>
<tr>
<td>Tmp29</td>
<td>2</td>
<td>Tmp5, 13, 14, 29</td>
<td>75,889,940,000</td>
</tr>
<tr>
<td>Tmp30</td>
<td>3</td>
<td>Tmp1, 2, 3, 4, 22, 30</td>
<td>2,700,168</td>
</tr>
</tbody>
</table>

Total materialized view maintenance cost 14,960,158,671,142

Table 4.40 The Query Processing Cost of the Static Approach

<table>
<thead>
<tr>
<th>Query</th>
<th>f_q</th>
<th>Accessed Nodes</th>
<th>Cost of Query Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp35(materialized view), result4</td>
<td>45,519,552</td>
</tr>
<tr>
<td>Q15</td>
<td>5</td>
<td>Tmp4(materialized view), Tmp5, Tmp6, result15</td>
<td>8,013,252,320</td>
</tr>
<tr>
<td>Q22</td>
<td>3</td>
<td>Tmp2, Tmp24(materialized view), Tmp25, result22</td>
<td>345,105,451,014</td>
</tr>
<tr>
<td>Q33</td>
<td>6</td>
<td>Tmp33(materialized view), result33</td>
<td>9,113,688</td>
</tr>
<tr>
<td>Q40</td>
<td>4</td>
<td>Tmp3(materialized view), Tmp29(materialized view), Tmp30, result40</td>
<td>5,155,298,792</td>
</tr>
<tr>
<td>Q43</td>
<td>7</td>
<td>Tmp2, Tmp5, Tmp11, Tmp24(materialized view), Tmp25, Tmp26, result43</td>
<td>1,449,432,585,629</td>
</tr>
<tr>
<td>Q50</td>
<td>5</td>
<td>Tmp3(materialized view), Tmp11, Tmp23(materialized view), Tmp29(materialized view), Tmp30, Tmp31, Tmp32, result50</td>
<td>28,673,453,690</td>
</tr>
<tr>
<td>Q3</td>
<td>4</td>
<td>Tmp34(materialized view), result3</td>
<td>30,346,368</td>
</tr>
<tr>
<td>Q6</td>
<td>7</td>
<td>Tmp4(materialized view), result6</td>
<td>2,243,248</td>
</tr>
<tr>
<td>Q28</td>
<td>5</td>
<td>Tmp29(materialized view), result28</td>
<td>321,720</td>
</tr>
</tbody>
</table>


**Table 4.40** (Continued)

<table>
<thead>
<tr>
<th>Query</th>
<th>Accessed Nodes</th>
<th>Cost of Query Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q30</td>
<td>Tmp12(materialized view), Tmp13, Tmp15, Tmp14(materialized view), Tmp16, Tmp17, result30</td>
<td>1,131,672,008,452</td>
</tr>
<tr>
<td>Q31</td>
<td>Tmp21(materialized view), result31</td>
<td>576,860</td>
</tr>
<tr>
<td>Q47</td>
<td>Tmp1, Tmp21(materialized view), Tmp22, result47</td>
<td>461,492,576,860</td>
</tr>
<tr>
<td></td>
<td><strong>Total query processing cost</strong></td>
<td><strong>3,429,632,748,193</strong></td>
</tr>
</tbody>
</table>

**Table 4.41** The Maintenance Cost of the Static Approach

<table>
<thead>
<tr>
<th>Materialized view</th>
<th>Number of Base Relation</th>
<th>Derived by Nodes</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp3</td>
<td>1</td>
<td>Tmp2, 3</td>
<td>400,000</td>
</tr>
<tr>
<td>Tmp4</td>
<td>2</td>
<td>Tmp1, 2, 3, 4</td>
<td>64,095,200,000</td>
</tr>
<tr>
<td>Tmp12</td>
<td>3</td>
<td>Tmp7, 8, 9, 10, 11, 12</td>
<td>2,700,168</td>
</tr>
<tr>
<td>Tmp14</td>
<td>1</td>
<td>Tmp13, 14</td>
<td>1,727,597</td>
</tr>
<tr>
<td>Tmp21</td>
<td>5</td>
<td>Tmp5, 7, 8, 9, 10, 13, 14, 16, 18, 19, 20, 21</td>
<td>902,328,817,595</td>
</tr>
<tr>
<td>Tmp23</td>
<td>2</td>
<td>Tmp13, 14, 16, 19, 23</td>
<td>1,726,713,034,618</td>
</tr>
<tr>
<td>Tmp24</td>
<td>3</td>
<td>Tmp1, 13, 14, 16, 19, 23, 24</td>
<td>3,970,477,551,927</td>
</tr>
<tr>
<td>Tmp29</td>
<td>4</td>
<td>Tmp1, 5, 7, 8, 9, 10, 18, 28, 29</td>
<td>6,413,681,184</td>
</tr>
<tr>
<td>Tmp33</td>
<td>5</td>
<td>Tmp1, 5, 7, 8, 9, 10, 16, 18, 19, 28, 33</td>
<td>3,047,265,055,480</td>
</tr>
<tr>
<td>Tmp34</td>
<td>2</td>
<td>Tmp5, 16, 19, 24</td>
<td>75,889,940,000</td>
</tr>
<tr>
<td>Tmp35</td>
<td>2</td>
<td>Tmp1, 16, 19, 35</td>
<td>6,069,299,200,000</td>
</tr>
<tr>
<td></td>
<td><strong>Total materialized view maintenance cost</strong></td>
<td><strong>15,862,487,308,569</strong></td>
<td></td>
</tr>
</tbody>
</table>
4.10 Analysis of the Affected Node Identification Algorithm

In this section, we provide the analysis and testing result of the characteristic of affected nodes derived by the affected node identification algorithm mentioned in section 3.7.2. According to the affected node identification algorithm, the properties of nodes that determine whether the affected nodes are (1) the conjunctive joined node with positive weight, or (2) project operation node is not the ancestor of base relation, or (3) the ancestor of directly affected node with the certain weight.

Conversely, the properties of nodes that are the unaffected nodes include:

1. They are the project operations that are the ancestor of base relation.
2. They are the conjunctive joined node with negative weights.
3. The property of the indirectly affected node is the node as in line 3.3 in algorithm that provides the maximum weight of each branch. Then, the ancestor of the directly affected nodes, that its weight is not maximum weight, is unaffected node.

The experiments in the following sections show that the total cost is not minimal when the unaffected nodes are materialized.

The first and the second cases are presented by the second experiment and the third case is presented by the first experiment.

For the second experiment that is shown in Figure 4.59, the existing nodes are neither indirectly nor directly affected nodes include:

Tmp3, Tmp5, Tmp7, Tmp10, Tmp12, Tmp13, Tmp20, Tmp21, Tmp22, Tmp23, Tmp24, Tmp25 and Tmp28

The project operation nodes, that are the ancestor of base relation, include Tmp3, Tmp5, Tmp7, Tmp10, Tmp13 and Tmp22. The conjunctive joined nodes with negative weight include Tmp12, Tmp20, Tmp21, Tmp23, Tmp24, Tmp25 and Tmp28.

4.10.1 Case I: Materializing the Project Operation which is the Ancestor of Base Relation

This property provides the total cost is not minimal because the number of tuples to be read is equal to the number of tuples produced from this operation. The
project operation, represented by algebra notation: \( \prod \), specifics the attributes from a base relation to be selected. The query processing cost and materialized view maintenance cost in cost model is formulated by the number of tuples not the attributes of the base relation.

**Example 1: Tmp5**

The materializing Tmp5 affect to Q15. The frequency of executing query Q15 is 5.

Considering the query processing cost of Q15 before and after Tmp5 is materialized.

**Before Tmp5 is materialized:**

Q15 accesses nodes Tmp5, Tmp28, Tmp27 (materialized view) and result15. The cost of each node is 10000, 1602320000, 160232 and 160232, respectively. The query processing cost of Q5 is \((5)*(10000 + 1602320000 + 160232 + 160232)\) that is 8,013,252,320.

The materialized view maintenance cost of Tmp5 is 0.

**After Tmp5 is materialized:**

Q15 accesses nodes Tmp5 (materialized view), Tmp28, Tmp27 (materialized view) and result15. The cost of each node is 10000, 1602320000, 160232 and 160232, respectively.

The query processing cost of Q5 is \((5)*(10000 + 1602320000 + 160232 + 160232)\) that is 8,013,252,320.

The materialized view maintenance cost of Tmp5 is \((1)*(10000)\).

Therefore, the query processing cost of Q5 is 8,013,252,320 for either Tmp5 be materialized view or virtual view. However, there is additional materialized view maintenance cost when Tmp5 is materialized. Therefore, total cost is increased by the materialized view maintenance cost of Tmp5.

**Example 2: Tmp5, Tmp10, Tmp22**

The materializing Tmp5, Tmp10 and Tmp22 together affect to Q43. The frequency of executing query Q43 is 7.
Considering the query processing cost of Q43 before and after these three nodes are materialized as Q43 derived on those nodes.

**Before these nodes are materialized:**

Q43 accesses nodes Tmp5, Tmp19 (materialized view), Tmp10, Tmp20, Tmp21, Tmp22, Tmp23 and result43. The query processing cost is \((7) \times (10000 + 575169 + 200000 + 115033800000 + 5751690000 + 150000 + 86275350000 + 22778)\) that is 1,449,432,585,629.

The materialized view maintenance cost of Tmp22, Tmp5, Tmp10 are 0.

**After these nodes are materialized:**

Q43 accesses nodes Tmp5 (materialized view), Tmp19 (materialized view), Tmp10 (materialized view), Tmp20, Tmp21, Tmp22 (materialized view), Tmp23 and result43. The query processing cost is \((7) \times (10000 + 575169 + 200000 + 115033800000 + 5751690000 + 150000 + 86275350000 + 22778)\) that is 1,449,432,585,629.

The materialized view maintenance cost of Tmp5, Tmp10, Tmp22 are \((1) \times (10000)\), \((1) \times (200000)\), \((1) \times (150000)\), respectively. The total materialized view maintenance cost for the three nodes is 360,000.

Therefore, the query processing cost of Q43 is 1,449,432,585,629 for either those nodes be materialized views or virtual views. However, there is additional materialized maintenance cost when Tmp5, Tmp10, Tmp22 are materialized. Therefore, total cost is increased by the summation of materialized view maintenance cost of these nodes.

**Example 3: Tmp10**

In Figure 4.59, only Q22 derived from Tmp10 directly, other queries derived from materialized views Tmp27 and Tmp11, the ancestor of Tmp10. The frequency of executing the query Q22 is 3

**Before Tmp10 is materialized:**

Q22 accesses nodes Tmp19 (materialized view), Tmp10, Tmp20 and result22. The query processing cost of Q22 is \((3) \times (575169 + 200000 + 115033800000 + 575169)\) that is 345,105,451,014.

The materialized view maintenance cost of Tmp10 is 0.
After **Tmp10** is materialized:

Q22 accesses nodes Tmp19 (materialized view), Tmp10 (materialized view), Tmp20 and result22. The query processing cost of Q22 is \((3)*(575169 + 200000 + 115033800000 + 575169)\) that is 345,105,451,014.

The materialized view maintenance cost of Tmp10 is \((1)*(200000) = 200000\).

Therefore, the query processing cost of Q22 is 345,105,451,014 for either those nodes be materialized views or virtual views. However, there is additional materialized view maintenance cost when those nodes are materialized. Therefore, total cost is increased by the summation of materialized view maintenance cost of those nodes.

### 4.10.2 Case 2: Materializing the Conjunctive Joined Node with Negative Weight

According to the weight formula mentioned in section 3.7.2, the first part presents the query processing cost and the second part presents the materialized view maintenance cost. The positive weight of node defines the possibility of node to be materialized. When the node with negative weight becomes a materialized view, the maintenance cost increases. In the second experiment, Tmp12, Tmp20, Tmp21, Tmp23, Tmp24, Tmp25, Tmp28 are the conjunctive joined nodes with negative weight. The following examples show that the total cost is not minimal when the conjunctive joined nodes with negative are materialized.

**Example 4: Tmp25**

Tmp25 is used to construct Q50. The frequency of executing query Q50 is 5. Tmp25 is constructed on 8 base relations. It is derived from the nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp13, Tmp14, Tmp16, Tmp17, Tmp18, Tmp22 and Tmp24. The cost of each node and Tmp25 is 5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 160240, 200000, 200000, 1288745976, 6000000, 6000000, 1500000, 227597, 863342789712, 150000, 3733997148 and 711150000, respectively.
So, the weight of Tmp25 is calculated as:

\[ w(Tmp25) = (5) \times (711150000) - (8) \times (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 160240 + 200000 + 200000 + 1288745976 + 6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 150000 + 3733997148 + 711150000) = 6,961,999,295,832. \]

To materialize Tmp25, the materialized view maintenance cost of Tmp25 is 6,965,555,045,832. So the total materialized view maintenance cost is increased by the materialized view maintenance cost of Tmp25.

Once Tmp25 is materialized, Q50 will be derived from materialized view Tmp25 instead, then the query processing cost of Q50 is reduced from 28,674,895,230 to 1,075 (5*215).

Then, the total cost is increased by 6,936,880,151,677 as the materialized view maintenance cost is increased by 6,965,555,045,832 whilst the query processing cost is reduced by 28,674,894,155 (28,674,895,230 - 1,075).

Although the query processing cost of Q50 is reduced from 28,674,895,230 to 1,075, the decrease of query processing cost is less than the increase of materialized view maintenance cost of Tmp25. So, the total cost increases.

**Example 5: Tmp12**

Tmp12 is accessed by Q40 and Q50 with the frequency of executing query is 4 and 5, respectively. Tmp12 is constructed on 5 base relations. Tmp12 is derived from nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and Tmp11. The cost of each node and Tmp12 is 5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 160240, 200000, 200000, and 1288745976, respectively.

So, the weight of Tmp12 is calculated as:

\[ w(Tmp12) = (4+5) \times (1288745976) - (5) \times (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 160240 + 200000 + 200000 + 1288745976) = -2,864,117,576. \]

To materialize Tmp12, Q40 and Q50 will derive from Tmp12 instead of Tmp9. However, Tmp9 still should be materialized to support Q28. Example 6 provides the result for the case Tmp9 is virtual view.

The materialized view maintenance cost of Tmp12 is 14,462,831,360.
For this case, Tmp11 should be un-materialized as there is no query derived from Tmp11. The materialized view maintenance cost of Tmp11 is 400,000. So, the maintenance cost is increased by 14,462,431,360 (14,462,831,360 - 400,000).

Q40 derived from the result of Tmp12 and result40. The query processing cost of Q40 derived from Tmp12 is \((4)*(6492 + 6492)\) that is 51,936.

Q50 derived from the result of Tmp12, the result of Tmp18, Tmp24, Tmp22, Tmp25 and result50. The query processing cost of Q50 derived from Tmp12 is \((5)*(6492 + 575169 + 3733997148 + 150000 + 711150000 + 215)\) that is 22,229,395,120.

So, query processing cost of Q40 when derived from Tmp12 decreases from 5,156,452,024 to 51,936. The query processing cost of Q50 when derived from Tmp12 decreased from 28,674,895,230 to 22,229,395,120. So, the total query processing cost is reduced by 11,601,900,198 \(((5,156,452,024 - 51,936) + (28,674,895,230 - 22,229,395,120))\).

Therefore, the total cost is increased by 2,860,531,162 \((14,462,431,360 - 11,601,900,198)\).

**Example 6: Tmp12 is materialized; Tmp9 and Tmp11 are virtual view**

For this example, there is one more query, Q28 that is affected when compared with Example 5. The materialized view maintenance cost is reduced due to Tmp9 is un-materialized. The query processing cost of Q40 and Q50 are the same as Example 5.

If Tmp9 are materialized, Q28 accesses nodes Tmp9 (materialized view) and result28. The query processing cost of Q28 is 321,720.

If Tmp9 is un-materialized, Q28 accesses nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9 and result28. The query processing cost of Q28 is increased by 8,017,262,340.

Comparing with Example 5, the total query processing cost of this example is increased by 8,016,940,620 \((8,017,262,340 - 321,720)\). The materialized view maintenance cost of Tmp9 is 6,413,681,184. Then, the total materialized maintenance cost is reduced by 6,413,081,184 compared with Example 5.
Therefore, the total cost is increased by 1,603,859,436 (8,016,940,620 - 6,413,081,184).

**Example 7: Tmp20 is materialized and Tmp19 is virtual view**

Tmp20 is accessed by Q22 and Q43, when Tmp20 is materialized; Tmp19 is un-materialized as Tmp20 is ancestor of Tmp19. Also there is no more query beside Q22 and Q43 accesses Tmp19.

The weight of Tmp20 is calculated as follows:

Tmp20 is constructed on 4 base relations. Tmp20 is accessed by Q22 and Q43 with the frequency of executing query is 3 and 7, respectively. Tmp20 is derived from nodes Tmp7, Tmp10, Tmp13, Tmp14, Tmp16, Tmp17, Tmp18 and Tmp19. The cost of each node and Tmp20 is 8000000, 200000, 6000000, 6000000, 1500000, 227597, 863342789712, 460135200000 and 115033800000, respectively.

So, the weight of Tmp20 is calculated as:

\[
w(Tmp20) = (3+7)*(115033800000) - (4)*(8000000 + 200000 + 6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 460135200000 + 115033800000) \]

that is (-4,603,796,869,236).

The materialized view maintenance cost of Tmp20 is 5,754,134,869,236 and Tmp19 is 3,970,499,151,927. Then, the total materialized view maintenance cost is increased by 1,783,635,717,309 (5,754,134,869,236 - 3,970,499,151,927).

When Tmp20 is materialized, Q22 accesses the result of Tmp20 and result22. The query processing cost of Q22 is 3,451,014 ((3)*(575169 + 575169)). The query processing cost of Q22 is reduced from 345,105,451,014 to 3,451,014.

Q43 accesses the result of Tmp20, Tmp21, Tmp22, Tmp23 and result43. The query processing cost of Q42 is 644,194,585,629 ((7)*(575169 + 10000 + 5751690000 + 150000 + 86275350000 + 22778)). The query processing cost of Q43 is reduced from 1,449,432,585,629 to 644,194,585,629.

Therefore, the total cost is increased by 633,295,717,309 as the maintenance cost is increased by 1,783,635,717,309 whilst the total query processing cost is reduced by 1,150,340,000,000.

The results of the above examples with negative weights are shown in Table 4.42.
Table 4.42  The Cost of All Queries for the Negative Weight Property

<table>
<thead>
<tr>
<th>Example</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>3,838,202,906,833</td>
<td>14,960,158,671,142</td>
<td>18,798,361,577,975</td>
</tr>
<tr>
<td>4</td>
<td>3,809,528,012,678</td>
<td>21,925,713,716,974</td>
<td>25,735,241,729,652</td>
</tr>
<tr>
<td>5</td>
<td>3,826,601,006,635</td>
<td>14,974,621,102,502</td>
<td>18,801,222,109,137</td>
</tr>
<tr>
<td>6</td>
<td>3,834,617,947,255</td>
<td>14,968,207,421,318</td>
<td>18,802,825,368,573</td>
</tr>
<tr>
<td>7</td>
<td>2,687,862,906,833</td>
<td>16,743,794,388,451</td>
<td>19,431,657,295,284</td>
</tr>
</tbody>
</table>

Note: * is the minimal total cost

The result of all above examples when nodes with negative weights are materialized show that the total query processing cost is reduced but the decreasing of query processing cost is less than the increasing of materialized view maintenance cost. Therefore, the total cost is increased.

4.10.3 Case 2: Materializing the Node without the Maximum Weight of the Branch

We explain this property by using the result of the first experiment. We represent the result of the first experiment again in Figure 4.61 and Table 4.43.
represents materialized view node selected in static phase for Q1-Q7
represents new materialized view node selected in dynamic phase
represents un-materialized node in dynamic phase

Figure 4.61  The Dynamic MVPP for Q8-Q13 after Applying 2PO

Table 4.43  The Minimum Total Cost of the First Experiment

<table>
<thead>
<tr>
<th>Materialized view</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp10, 11, 13, 15,</td>
<td>3,684,593,788,740</td>
<td>6,065,915,980,889</td>
<td>9,750,509,769,629</td>
</tr>
<tr>
<td>17, 19, 21, 24, 27, 28</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The directly affected nodes are Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp13, Tmp14, Tmp17, Tmp19, and Tmp24. The directly affected nodes that have ancestor nodes, not in the list of directly affected nodes, are shown in Table 4.44.

In Table 4.44, Tmp6 is the directly affected node used to construct new queries Q12 and Q13. It has 2 ancestor nodes, Tmp8 and Tmp11. Tmp11 provides the maximum weight of this branch, Tmp6 → Tmp8 → Tmp11, then Tmp11 is determined as indirectly affected node but not Tmp8.
Table 4.44 The Weight of Ancestor Nodes of Directly Affected Node of New Queries

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>36,029,759,776</td>
</tr>
<tr>
<td></td>
<td>Tmp11</td>
<td>486,610,492,657</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Tmp15</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Tmp23</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>-4,096,769,632,791</td>
</tr>
</tbody>
</table>

We provide the experiment to validate that only the node with the maximum weight should be considered as an indirectly affected node as follows.

Our assumption is that if Tmp8 is possible to be the affected node then the total cost has to be less than the minimum total cost in Table 4.43 when Tmp8 is materialized.

There are two scenarios for this experiment; (1) Tmp8 and Tmp11 are materialized view, (2) Tmp8 is materialized and Tmp11 is a virtual view. If one out of two scenarios provides the total cost less than minimum cost, then Tmp8 has to be determined as an affected node.

Example 8: The first scenario, Tmp8 and Tmp11 are materialized

Tmp8 and Tmp11 support Q3, then the query processing cost of Q3 might be affected when Tmp8, Tmp11 are either materialized or virtual views.

All costs in Table 4.43 already include materialized view Tmp11, so we consider additional cost due to Tmp8 be materialized.

Tmp8 is constructed on 4 base relations, and derived from the nodes Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6 and Tmp7. The cost of each node and Tmp8 is 5, 1, 25, 25, 10000, 50000, 6000000 and 12018000000, respectively. The materialized view maintenance cost for Tmp8 is 48,096,240,224 ((4)\*(5 + 1 + 25 + 25 + 10000 + 50000 + 6000000 + 12018000000)).

The total maintenance cost is 6,114,012,221,113 (6,065,915,980,889 + 48,096,240,224).
The total query processing cost is the same as query processing cost in Table 4.43 that is 3,684,593,788,740 even Tmp8 is materialized as there is only Q3 derived from Tmp8.

Therefore, the total cost is 9,798,606,009,853 (3,684,593,788,740 + 6,114,012,221,113). That is greater than that in Table 4.44.

For this assumption, it is obviously that the maintenance cost is increased without the benefit to speed up query process of Q3 as Q3 derived from Tmp11 (materialized view) directly.

**Example 9: The second scenario, Tmp8 is materialized view and Tmp11 is virtual view**

For this scenario, the query processing cost of Q3 is changed as Q3 has to derive from Tmp8 and Tmp11 instead Tmp11 only as in Example 8.

Q3 with the frequency of executing query is 7 and accesses the nodes Tmp8 (materialized view), Tmp10, Tmp11 and result8. The query processing cost of Q3 is 1,913,599,284,478 ((7)*(1201113 + 227597 + 273369715461 + 182183)). It is changed from 2,550,562 to 1,913,599,284,478. Therefore, the total query processing cost is increased by 1,913,596,733,916 (1,913,599,284,478 - 2,550,562).

The materialized view maintenance cost of Tmp11 and Tmp8 is 1,426,977,515,570 and 48,096,240,224, respectively. The total materialized view maintenance cost is reduced by 1,378,881,275,346 (1,426,977,515,570 - 48,096,240,224).

As the query processing is increased and maintenance cost is reduced, then the total cost is increased by 534,715,458,570 (1,913,596,733,916 - 1,378,881,275,346).

When we materialize Tmp8 and un-materialize Tmp11, the total cost is greater than the total cost in Table 4.43. According to our assumptions, Tmp8 should not be determined as affected node because it provides the higher total cost than minimal total cost in Table 4.43.

The results of all above examples show that when the unaffected nodes are materialized, the total cost always increases. Therefore, it is not necessary to include these nodes for materializing.
We further validate our affected node identification algorithm using the second experiment by setting all nodes in search space as the member of nodes to be selected for 2PO. Our assumption is that if any solutions provide the total cost, which unaffected nodes is included, less than the minimal total cost in Table 4.37, then our affected node determination is not efficient enough. The result of selecting all nodes is shown in Table 4.45. The result shows that the state providing the minimum total cost is materializing \{Tmp6, Tmp9, Tmp11, Tmp15, Tmp17, Tmp18, Tmp19, Tmp26, Tmp27, Tmp29 and Tmp30\}. The set of materialized view is same as our dynamic approach as shown in Table 4.46. The query processing cost is 3,838,202,906,833, the materialized view maintenance cost is 14,960,158,671,142 and the total cost which is summation of query processing cost and materialized view maintenance cost is 18,798,361,577,975, the same as Table 4.37.

We provide the explanation for some example states in following sections that include the unaffected node. The state including unaffected nodes do not provide the minimum total cost.

**Analysis of the Conjunctive Joined Node with Negative Weight Property**

In Table 4.45, the 14th state includes conjunctive joined node with negative weight that is Tmp12.

The total cost of the 14th state is greater than that of the 15th state. Tmp12 is not determined as affected node as its weight is negative. Although Tmp12 is materialized in the 14th state and total cost is less than the 11th state, the total cost of the 14th state is more than the 15th state. Tmp12 is built by Tmp9 and Tmp11. The weights of Tmp9 and Tmp11 are positive. So, materializing Tmp9 and Tmp11 together provides total cost less than materializing Tmp12. The other example states compared with the 16th state, the 18th state includes Tmp23 and the 23rd state includes Tmp25. The weights of Tmp23 and Tmp25 are negative. When materializing Tmp23 and Tmp25, the total cost of the 18th and the 23rd state are greater than the 16th state even though the query processing cost of both states are less than that of 16th state.
Analysis of the project operation node, the ancestor of base relation property

From Table 4.45, the following states are the example states that include project node that is the ancestor of base relations.

Initial state: Tmp15, 18, 19, 26, 27, \( C_{\text{maintence}}: 14,877,850,042,025 \)

4th state: Tmp5, 15, 18, 19, 26, 27, \( C_{\text{maintence}}: 14,877,850,052,025 \)

8th state: Tmp10, 15, 18, 19, 26, 27, \( C_{\text{maintence}}: 14,877,850,242,025 \)

The costs of query processing of all above states are same, 4,012,431,182,390, but the costs of maintenance are difference. In the 4th state, Tmp5 is the ancestor of base relation, SUPPLIER. Comparing the 4th state with the initial state, the maintenance cost is increased by the materialized view maintenance cost of Tmp5, 10,000. Considering the 8th state, Tmp10 is the ancestor of base relation, PART, the total cost of the 8th state is greater than the initial state by the materialized view maintenance cost of Tmp10, 200,000. Also, the 18th to the 25th state has the materialized views same as the 16th state plus additional nodes. The weights of additional nodes are negative. Therefore, the total cost of these states is greater than that of 16th state.
The 2PO result for setting all nodes are the members of search space

The initial state for 2PO is that all nodes are set to 0. The output generated by II is that \{Tmp_{15}, Tmp_{18}, Tmp_{19}, Tmp_{26} and Tmp_{17}\} are materialized that is the initial state of SA of the 2PO.

**Table 4.45** The States Generated by 2PO of Dynamic Phase for All Nodes

<table>
<thead>
<tr>
<th>State</th>
<th>Materialized View</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tmp_{15}, 18, 19, 26, 27 (the initial state for SA)</td>
<td>4,012,431,182,390</td>
<td>14,877,850,042,025</td>
<td>18,890,281,224,415</td>
</tr>
<tr>
<td>1</td>
<td>Tmp_{7}, 14, 15, 18, 19, 27</td>
<td>22,220,256,782,390</td>
<td>8,808,563,642,025</td>
<td>31,028,820,424,415</td>
</tr>
<tr>
<td>2</td>
<td>Tmp_{7}, 13, 15, 18, 19, 27</td>
<td>22,220,256,782,390</td>
<td>8,808,557,642,025</td>
<td>31,028,814,424,415</td>
</tr>
<tr>
<td>3</td>
<td>Tmp_{15}, 18, 19, 26, 28</td>
<td>4,228,751,610,766</td>
<td>14,914,704,632,025</td>
<td>19,143,456,242,791</td>
</tr>
<tr>
<td>4</td>
<td>Tmp_{5}, 15, 18, 19, 26, 27</td>
<td>4,012,431,182,390</td>
<td>14,877,850,052,025</td>
<td>18,890,281,424,415</td>
</tr>
<tr>
<td>5</td>
<td>Tmp_{7}, 11, 15, 18, 19, 26</td>
<td>4,396,994,700,534</td>
<td>14,813,756,042,025</td>
<td>19,210,750,742,559</td>
</tr>
<tr>
<td>6</td>
<td>Tmp_{7}, 10, 15, 18, 19, 26</td>
<td>4,397,000,459,606</td>
<td>14,813,755,842,025</td>
<td>19,210,756,301,631</td>
</tr>
<tr>
<td>7</td>
<td>Tmp_{15}, 18, 20, 26, 27</td>
<td>2,862,091,182,390</td>
<td>16,661,478,559,334</td>
<td>19,523,569,741,724</td>
</tr>
<tr>
<td>8</td>
<td>Tmp_{10}, 15, 18, 19, 26, 27</td>
<td>4,012,431,182,390</td>
<td>14,877,850,242,025</td>
<td>18,890,281,424,415</td>
</tr>
<tr>
<td>9</td>
<td>Tmp_{8}, 14, 18, 19, 26, 27</td>
<td>7,637,008,803,006</td>
<td>11,837,010,026,769</td>
<td>19,474,018,829,775</td>
</tr>
<tr>
<td>10</td>
<td>Tmp_{8}, 13, 18, 19, 26, 27</td>
<td>7,637,082,663,342</td>
<td>11,837,004,026,769</td>
<td>19,474,086,690,111</td>
</tr>
<tr>
<td>11</td>
<td>Tmp_{6}, 15, 18, 19, 26, 27</td>
<td>4,012,430,079,383</td>
<td>14,877,850,222,193</td>
<td>18,890,280,301,576</td>
</tr>
<tr>
<td>12</td>
<td>Tmp_{4}, 5, 15, 18, 19, 26, 27</td>
<td>4,012,431,181,217</td>
<td>14,877,850,042,137</td>
<td>18,890,281,223,354</td>
</tr>
<tr>
<td>13</td>
<td>Tmp_{4}, 15, 18, 19, 26, 27</td>
<td>4,012,431,181,217</td>
<td>14,877,850,042,137</td>
<td>18,890,281,223,354</td>
</tr>
<tr>
<td>14</td>
<td>Tmp_{6}, 12, 15, 18, 19, 26, 27</td>
<td>3,986,400,439,685</td>
<td>14,892,313,053,553</td>
<td>18,878,713,493,238</td>
</tr>
<tr>
<td>15</td>
<td>Tmp_{6}, 9, 15, 18, 19, 26, 27</td>
<td>3,989,983,818,331</td>
<td>14,884,263,903,377</td>
<td>18,874,247,721,708</td>
</tr>
<tr>
<td>State</td>
<td>Materialized View</td>
<td>Cost of Query Processing</td>
<td>Cost of Maintenance</td>
<td>Total Cost</td>
</tr>
<tr>
<td>-------</td>
<td>-------------------------------------------------------</td>
<td>--------------------------</td>
<td>---------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>16</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27</td>
<td>3,989,980,218,911</td>
<td>14,884,264,303,377</td>
<td>18,874,244,522,288</td>
</tr>
<tr>
<td>17</td>
<td>Tmp6, 8, 11, 15, 18, 19, 26, 27</td>
<td>3,989,983,101,991</td>
<td>14,884,263,662,417</td>
<td>18,874,246,764,408</td>
</tr>
<tr>
<td>18</td>
<td>Tmp6, 9, 11, 15, 18, 19, 23, 26, 27</td>
<td>2,540,547,952,174</td>
<td>24,067,588,334,828</td>
<td>26,608,139,170,082</td>
</tr>
<tr>
<td>19</td>
<td>Tmp6, 9, 11, 15, 18, 19, 22, 21, 26, 27</td>
<td>3,144,480,388,911</td>
<td>22,105,657,267,519</td>
<td>25,250,137,506,430</td>
</tr>
<tr>
<td>20</td>
<td>Tmp6, 9, 11, 15, 18, 19, 21, 26, 27</td>
<td>3,144,480,388,911</td>
<td>22,105,657,110,210</td>
<td>25,250,137,506,430</td>
</tr>
<tr>
<td>21</td>
<td>Tmp5, 6, 9, 11, 15, 18, 19, 20, 26, 27</td>
<td>3,184,742,218,911</td>
<td>20,638,372,110,210</td>
<td>23,823,114,319,121</td>
</tr>
<tr>
<td>22</td>
<td>Tmp6, 9, 11, 15, 18, 20, 26, 27</td>
<td>3,184,742,218,911</td>
<td>20,638,372,100,210</td>
<td>23,823,114,319,121</td>
</tr>
<tr>
<td>23</td>
<td>Tmp5, 6, 9, 11, 15, 18, 19, 25, 26, 27</td>
<td>3,961,306,767,371</td>
<td>21,849,821,076,806</td>
<td>25,811,127,844,177</td>
</tr>
<tr>
<td>24</td>
<td>Tmp5, 6, 9, 11, 15, 18, 19, 24, 26, 27</td>
<td>3,961,663,290,001</td>
<td>20,104,164,516,348</td>
<td>24,065,827,815,349</td>
</tr>
<tr>
<td>25</td>
<td>Tmp6, 9, 11, 15, 18, 19, 24, 26, 27</td>
<td>3,961,663,290,001</td>
<td>20,104,164,515,348</td>
<td>24,065,827,805,349</td>
</tr>
<tr>
<td>26</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29</td>
<td>3,838,215,512,095</td>
<td>14,960,155,707,974</td>
<td>18,798,371,483,069</td>
</tr>
<tr>
<td>27</td>
<td>Tmp5, 6, 9, 11, 14, 15, 18, 19, 26, 27</td>
<td>3,989,947,392,095</td>
<td>14,884,278,040,974</td>
<td>18,874,225,433,069</td>
</tr>
<tr>
<td>28</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 32</td>
<td>2,722,547,177,757</td>
<td>16,374,786,910,494</td>
<td>19,097,334,088,251</td>
</tr>
<tr>
<td>29</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 31</td>
<td>3,826,762,625,461</td>
<td>14,987,650,722,590</td>
<td>18,814,413,348,051</td>
</tr>
<tr>
<td>30</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 31</td>
<td>3,826,762,625,461</td>
<td>14,987,644,722,590</td>
<td>18,814,407,348,051</td>
</tr>
<tr>
<td>31</td>
<td>Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 30 (the minimal state)</td>
<td>3,838,202,906,833</td>
<td>14,960,158,671,142</td>
<td>18,798,361,577,975</td>
</tr>
<tr>
<td>32</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 30</td>
<td>3,838,208,906,833</td>
<td>14,960,156,943,545</td>
<td>18,798,365,850,378</td>
</tr>
<tr>
<td>33</td>
<td>Tmp4, 6, 9, 11, 15, 17, 18, 19, 22, 26, 27, 29</td>
<td>3,838,212,386,121</td>
<td>14,960,156,121,086</td>
<td>18,798,368,507,207</td>
</tr>
<tr>
<td>34</td>
<td>Tmp9, 11, 15, 17, 18, 19, 26, 27, 29, 30, 33</td>
<td>3,832,440,280,298</td>
<td>16,691,479,770,464</td>
<td>20,523,920,050,762</td>
</tr>
<tr>
<td>35</td>
<td>Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 34</td>
<td>2,973,928,528,448</td>
<td>19,654,011,127,168</td>
<td>22,627,939,655,616</td>
</tr>
<tr>
<td>36</td>
<td>Tmp6, 8, 11, 14, 15, 17, 18, 19, 26, 27, 29</td>
<td>6,620,974,199,060</td>
<td>16,606,746,071,688</td>
<td>23,227,720,270,748</td>
</tr>
</tbody>
</table>
The 2PO result for only affected node and new nodes are the members of search space

Initial State for II is all nodes set to 0. The output generated by II is that \{Tmp15, Tmp18, Tmp19, Tmp26 and Tmp27\} are materialized that is the initial state for SA of the 2PO.

**Table 4.46** The States Generated by 2PO of Dynamic Phase for Affected Nodes

<table>
<thead>
<tr>
<th>State</th>
<th>Materialized View</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Tmp15, 18, 19, 26, 27 (the initial state for SA)</td>
<td>4,012,431,182,390</td>
<td>14,877,850,042,025</td>
<td>18,890,281,224,415</td>
</tr>
<tr>
<td>1</td>
<td>Tmp14, 15, 18, 19, 27</td>
<td>22,220,256,782,390</td>
<td>8,808,562,842,025</td>
<td>31,028,819,624,415</td>
</tr>
<tr>
<td>2</td>
<td>Tmp11, 15, 18, 19, 26</td>
<td>4,396,994,700,534</td>
<td>14,813,755,242,025</td>
<td>19,210,749,942,559</td>
</tr>
<tr>
<td>3</td>
<td>Tmp8, 14, 15, 18, 19, 26, 27</td>
<td>7,637,008,803,006</td>
<td>11,837,010,026,769</td>
<td>19,474,018,829,775</td>
</tr>
<tr>
<td>4</td>
<td>Tmp6, 15, 18, 19, 26, 27</td>
<td>4,012,430,079,383</td>
<td>14,877,850,222,193</td>
<td>18,890,280,301,576</td>
</tr>
<tr>
<td>5</td>
<td>Tmp4, 15, 18, 19, 26, 27</td>
<td>4,012,431,181,217</td>
<td>14,877,850,042,137</td>
<td>18,890,281,223,354</td>
</tr>
<tr>
<td>6</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27</td>
<td>3,989,980,218,911</td>
<td>14,884,264,303,377</td>
<td>18,874,244,522,288</td>
</tr>
<tr>
<td>7</td>
<td>Tmp6, 8, 11, 15, 18, 19, 26, 27</td>
<td>3,989,983,101,991</td>
<td>14,884,263,662,417</td>
<td>18,874,246,764,408</td>
</tr>
<tr>
<td>8</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29</td>
<td>3,838,215,512,095</td>
<td>14,960,155,970,974</td>
<td>18,798,371,483,069</td>
</tr>
<tr>
<td>9</td>
<td>Tmp6, 9, 11, 14, 15, 18, 19, 26, 27</td>
<td>3,989,947,392,095</td>
<td>14,884,278,030,974</td>
<td>18,874,225,423,069</td>
</tr>
<tr>
<td>10</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 32</td>
<td>2,722,547,177,757</td>
<td>16,374,786,910,494</td>
<td>19,097,334,088,251</td>
</tr>
<tr>
<td>11</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 31</td>
<td>3,826,762,625,461</td>
<td>14,987,644,722,590</td>
<td>18,814,407,348,051</td>
</tr>
<tr>
<td>12</td>
<td>Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 30 (the minimal state)</td>
<td>3,838,202,906,833</td>
<td>14,960,158,671,142</td>
<td>18,798,361,577,975</td>
</tr>
<tr>
<td>13</td>
<td>Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 30</td>
<td>3,838,208,906,833</td>
<td>14,960,156,943,545</td>
<td>18,798,365,850,378</td>
</tr>
<tr>
<td>14</td>
<td>Tmp9, 11, 15, 17, 18, 19, 26, 27, 29, 30, 33</td>
<td>3,832,440,280,298</td>
<td>16,691,479,770,464</td>
<td>20,523,920,050,762</td>
</tr>
<tr>
<td>15</td>
<td>Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 34</td>
<td>2,973,928,528,448</td>
<td>19,654,011,127,168</td>
<td>22,627,939,655,616</td>
</tr>
<tr>
<td>16</td>
<td>Tmp6, 8, 11, 14, 15, 17, 18, 19, 26, 27, 29</td>
<td>6,620,974,199,060</td>
<td>16,606,746,071,688</td>
<td>23,227,720,270,748</td>
</tr>
</tbody>
</table>
In this chapter, we present the experiments designed to evaluate the effectiveness of our proposed methodologies that are the MVPP re-optimization algorithm and the dynamic materialized view selection approach. The experiment results do show that our MVPP re-optimization algorithm improve the query processing cost of the search space. The dynamic approach also help support new requirements that can avoid the repeatedly run all requirements, the existing and the new requirements, and the affected node identification algorithm can reduce the size of search space. The conclusion of our methodologies to solve the dynamic materialized view selection problem will be presented in the next chapter.
CHAPTER 5

CONCLUSION AND FUTURE WORK

Materializing view is a technique to improve the query performance in a data warehouse. However, materialize views have maintenance cost, so materialization of all views is not possible. Deciding which of the appropriated views are to be materialized is one of the most important problems in data warehouse design. In order to solve this problem, constructing a search space to identify a set of views to be materialized is a necessary task. To generate the search space, it is practically impossible to consider all common subexpressions among queries because of the numerous numbers of possible common subexpressions. The MVPP is one of the several approaches to constructing the optimal search space for the view selection problem. As the generating of MVPP is constructed by merging the individual optimal plans in order of the frequency of executing query multiplied with query cost. Thus, merging of incoming query has to use the common subexpressions of the previous merging. Therefore, it will lose the global optimization.

We propose the MVPP re-optimization algorithm to verify whether the cheapest MVPP is optimal, and to improve the query processing cost of the cheapest MVPP by rewriting the query using the concept of commonality of common subexpression for all queries. Our goal is to preserve global optimization by reducing the query processing cost of the cheapest MVPP. The result shows that the total query processing cost of MVPP is reduced if the query can be rewritten. After materialized views are selected by selection algorithm, 2PO, the total cost, which is the summation of query processing cost and materialized view maintenance cost, is reduced as well.

Moreover, in the real situation, the requirements specified by the various stakeholders are frequently changed and such changes will cause the existing resources to be changed. We start our dynamic materialized view selection with the static phase by implementing the static materialized view selection for the initial
requirements. In static phase, we apply our MVPP re-optimization algorithm to improve the cheapest MVPP and apply 2PO algorithm to select a set of views to be materialized. Two experiments have been conducted to evaluate the performance of the proposed algorithms. The experiments are performed on our testbed 50 queries with the combination of base relations. For the first experiment, when the MVPP re-optimization algorithm is applied, the cheapest MVPP is not the optimal one as some queries can be rewritten using an alternative execution plan in the cheapest MVPP. After the problematic queries are rewritten, the re-optimized MVPP provides the total query processing cost less than that of the cheapest MVPP. Later, we apply 2PO to select the set of views to be materialized, the total cost, 6,120,827,925,892, is less than the cheapest MVPP, 6,362,230,638,028. For the second experiment, some queries in the cheapest MVPP do not provide the minimum query processing cost compared with the other MVPPs. However, there is no alternative plan in the cheapest MVPP for that problematic query then it cannot be rewritten. Therefore, our MVPP re-optimization algorithm can help to verify that the cheapest MVPP is the optimal MVPP.

For the dynamic phase, there are new queries added into the existing MVPP generated in the static phase. After the new queries are merged into the existing MVPP, the MVPP re-optimization algorithm is applied to preserve the global optimization of MVPP. Later, we apply our affected node identification algorithm aiming to reduce the search space for selection algorithm, 2PO. The results show that our affected node identification algorithm can identify the necessary nodes to be selected to be materialized or un-materialized. The number of nodes to be selected by our dynamic approach is less than that of the static approach because the static approach has to recalibrate from scratch for all requirements, existing and new requirements. The number of nodes in the search space for the first experiment is reduced from 35 to 23 nodes and the second experiment is reduced from 35 to 21 nodes. Therefore, the proposed approach achieves our objective of the dynamic materialized view selection problem, that is not all existing resources need to be considered for materializing.

Finally, we use 2PO algorithm to select the set of views to be materialized. The first experiment of our dynamic approach, after new queries are merged, provides
the MVPP structure the same as that of the static approach. The total cost of our
dynamic approach is equal to the total cost of the static approach despite the fact that
the number of nodes to be selected by 2PO of the dynamic approach is less than that
of the static approach. For the second experiment, our dynamic approach provides the
MVPP structure different from that of the static approach. The result is that the total
cost of dynamic approach is less than that of static approach, and the number of nodes
to be selected by 2PO of dynamic approach is less than that of the static approach. We
also select complex SQL standard queries to cover all categories of subexpression
commonality to reflect the real situation in which all kinds of relevance with existing
resource would be added into the existing environment simultaneously.

In a real application system, there are other constraints that the system should
consider beside the number of tuples, frequency of executing a query and frequency
of updating base relations used in cost model such as indexes, system storage
constraints or time constraints. Also, further study should be done on how to better
calculate the maintenance cost of a materialized view, such as calculating the cost
based on maintenance cost of its descendent materialized views.
BIBLIOGRAPHY


APPENDICES
APPENDIX A

Result of Merging Queries to Construct MVPPs

In this appendix, the details to merge queries based on order of queries’ weight for query set Q1 to Q7 are presented. The first order list of the first MVPPs of this query set is \{Q4, Q7, Q3, Q2, Q6, Q1, and Q5\}.

A.1 The First MVPP

The details of constructing the first MVPP already have been described as Figure 4.10 in section 4.3. We represent the first MVPP again in Figure A.1.

![Figure A.1 The Result of the First MVPP](image)

The details for other MVPPs are described in the follow sections.

A.2 The Second MVPP

The following Figure A.2 shows the result of the second MVPP based on the order list \{Q7, Q3, Q2, Q6, Q1, Q5 and Q4\}. As Q7 is the first query of the second
MVPP then we start the second MVPP with Q7. Next when Q3 is merged, we first join REGION with NATION and then join this result with SUPPLIER. There is the subtree joined conjunctively of LINEITEM and ORDERS so the new node is introduced to join those results i.e. (REGION \bowtie NATION \bowtie SUPPLIER) and (LINEITEM \bowtie ORDERS). Figure A.2 (a) shows the result after Q3 is merged into MVPP. Next query in the list is query Q2, there are two conjunctively joined nodes available in MVPP that are (REGION \bowtie NATION) and (LINEITEM \bowtie ORDERS \bowtie
CUSTOMER). Therefore, new node is introduced to join those results. Figure A.2 (b) shows the result after Q2 is merged into MVPP. Next query in the list is Q6, LINEITEM and ORDERS conjunctively joined node is available in the MVPP then new node is introduced to join that result with SUPPLIER. Figure A.2 (c) shows the result after Q6 is merged into MVPP. Next query is Q1, there are two sharable conjunctively joined available for Q1, therefore new node is introduced to join result of (REGION ▷ NATION ▷ SUPPLIER) and (LINEITEM ▷ ORDERS). Figure A.2 (d) shows the result after Q1 is merged into MVPP. Next query is Q5, as Q5 is subsumption of Q6 that is already available in the MVPP then no new node is generated for Q5. Figure A.2 (e) shows the result after Q5 is merged into MVPP. The last query in the list is Q4, the available conjunctively join is (REGION ▷ NATION ▷ SUPPLIER) then new nodes are introduced to join that result with PARTSUPP, and then join the result with CUSTOMER. Figure A.2 (f) shows the result of the second MVPP after all queries are merged.

A.3 The Third MVPP

After the second MVPP is built, the first element, Q7, of the list is moved to the end of the list. Therefore, the new list becomes {Q3, Q2, Q6, Q1, Q5, Q4 and Q7}. Figure A.3 shows the result of the third MVPP. As Q3 is the first query of the list then we start with the third MVPP equal to Q3. Next query in the list is Q2, when we merge Q2 into MVPP only REGION and NATION are already conjunctively joined then new nodes are introduced as join operation between that conjunctively joined node with CUSTOMER, then join with ORDERS and then join with LINEITEM respectively as shown in Figure A.3 (a). Next when we merge Q6 into MVPP, as the MVPP does not have possibly sharable conjunctively joined node with Q6 then new nodes are introduced as join operation for Q6 as shown in Figure A.3 (b). Next query in the list is Q1, when we merge Q1 into MVPP, there are sharable conjunctively join for Q1 so only new node is introduced as join operation node for those results (REGION ▷ NATION ▷ SUPPLIER) and (LINEITEM ▷ ORDERS). Figure A.3 (c) shows the result after Q1 is merged into MVPP. Next, Q5 is subsumption of Q6 that is already available in MVPP then no new node is
generated for Q5 as shown in Figure A.3 (d). Next, when we merge Q4 into MVPP, new nodes are introduced as join

\[ \text{Figure A.3 The Result of the Third MVPP Based on the Order List \{Q3, Q2, Q6, Q1, Q5, Q4 and Q7\}} \]

operation for existing sharable subexpression and base relation that are the result of \((\text{REGION} \bowtie \text{NATION})\) joined with SUPPLIER and then the result is joined with PARTSUPP and next join with CUSTOMER respectively. Figure A.3 (e) shows the result after Q4 is merged into MVPP. The last query is Q7, as there is no sharable conjunctively join in MVPP that Q7 can use, then new nodes are introduced as join operation node of \((\text{LINEITEM} \bowtie \text{ORDERS})\) and its result is joined with
CUSTOMER. Figure A.3 (f) shows the result after all queries of the third MVPP are merged.

**A.4 The Fourth MVPP**

![Diagram of the Fourth MVPP](image)

**Figure A.4** The Result of the Fourth MVPP Based on the Order List \{Q2, Q6, Q1, Q5, Q4, Q7 and Q3\}

After the third MVPP is built, the first element of the list, Q3, is moved to the end of the list then the new list becomes \{Q2, Q6, Q1, Q5, Q4, Q7 and Q3\}. Figure A.4 shows the result of the fourth MVPP. We start the fourth MVPP equal to Q2 as
Q2 is the first query in the list. When Q6 is merged into MVPP, there is no sharable between Q2 and Q6 then new nodes are introduced to construct Q6 as shown in Figure A.4 (a). Next, when we merge Q1 into MVPP, we first join SUPPLIER with joined result of (REGION \( \Join \) NATION). The remaining base relations are PARTSUPP and PART already joined for Q6. Then, new node is introduced as join operation for that result as shown in Figure A.4 (b). Next Q5 is subsumption of Q6 that is already available in MVPP then no new node is generated for Q5 as shown in Figure A.4 (c). Next query is Q4, Q4 is merged into MVPP by using sharable join operator of (REGION \( \Join \) NATION \( \Join \) SUPPLIER). For PARTSUPP and CUSTOMER, the new nodes are introduced as join operation respectively. Figure A.4 (d) shows the result after Q4 is merged into MVPP. Next query in the list is Q7, there is no sharable in MVPP for Q7 then new nodes are introduced as shown in Figure A.4 (e). The last query in the list is Q3, there are two sharable subexpression available in MVPP, therefore the new node is introduced to join those results that are (REGION \( \Join \) NATION \( \Join \) SUPPLIER) and (LINEITEM \( \Join \) ORDERS). Figure A.4 (f) shows the result after all queries are merged for the fourth MVPP.

A.5 The Fifth to Seventh MVPPs

We repeat these steps of MVPP algorithm to construct the fifth to seventh MVPP based on following order list of query

The fifth MVPP : \{Q6, Q1, Q5, Q4, Q7, Q3 and Q2\}
The sixth MVPP : \{Q1, Q5, Q4, Q7, Q3, Q2 and Q6\}
The seventh MVPP : \{Q5, Q4, Q7, Q3, Q2, Q6 and Q1\}

The pictorial views of merging each query into MVPP show as Figure A.5 to A.7 for the fifth to seventh MVPP respectively.
Figure A.5  The Result of the Fifth MVPP Based on the Query List \{Q_6, Q_1, Q_5, Q_4, Q_7, Q_3 \text{ and } Q_2\}
Figure A.6 The Result of the Sixth MVPP Based on the Query List \{Q1, Q5, Q4, Q7, Q3, Q2 and Q6\}
After all MVPPs are constructed by above steps, we optimize MVPP by push select, project and aggregate function down as deep as possible for all MVPPs. Figure A.8 to Figure A.14 show the first MVPP to the seventh MVPP already optimized. Finally the total query processing costs of MVPP, which is the summation of query processing cost of queries in the MVPP, are calculated. The query processing cost is the frequency of executing the query multiplied with the cost of accessing the nodes to obtain the result of the query. Table A.1 to A.7 show the query processing of queries for the first MVPP to the seventh MVPP respectively.

Figure A.7 The Result of the Seventh MVPP Based on the Query list \{Q5, Q4, Q7, Q3, Q2, Q6 and Q1\}
The First MVPP, Queries in the List: {Q4, Q7, Q3, Q2, Q6, Q1, and Q5}

Table A.1  The Query Processing Cost of the First MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>Access from the nodes</th>
<th>Cost of Each Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp7, Tmp8, Tmp18, Tmp22 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240</td>
<td>67,303,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15, Tmp17 and result2</td>
<td>5, 1, 25, 25, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp11, Tmp12, Tmp13, Tmp14, Tmp16 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1823769557 and 182183</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>Tmp7, Tmp18, Tmp19, Tmp20 and result5</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>36,282,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>Tmp7, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
<tr>
<td>Q7</td>
<td>Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15 and result7</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000 and 910519</td>
<td>23,316,809,013,207</td>
</tr>
</tbody>
</table>

The total query processing cost of the first MVPP
Figure A.9 The Second MVPP, Query in the List: {Q7, Q3, Q2, Q6, Q1, Q5 and Q4}

Table A.2 The Query Processing Cost of the Second MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp16, Tmp17, Tmp20 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 20000, 160000000000, 160240000 and 160240</td>
<td>323,207,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp16, Tmp17, Tmp20 and result2</td>
<td>5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp16, Tmp17, Tmp20 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1500000, 227597, 1365582000000, 1500000, 227597, 1823769557</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp16, Tmp17, Tmp20 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 160240000000, 1500000, 24036000000, 967519280 and 184082</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>Tmp15, Tmp16, Tmp17, Tmp18 and result5</td>
<td>800000, 200000, 160000000000, 800000 and 36276</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>Tmp15, Tmp16, Tmp17, Tmp18, Tmp19 and result6</td>
<td>800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6 and result7</td>
<td>6000000, 1500000, 227597, 1365582000000, 1500000, 136577850000 and 91051</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the second MVPP 25,711,148,613,207
Figure A.10 The Third MVPP (the Cheapest MVPP), Query in the List: \{Q3, Q2, Q6, Q1, Q5, Q4 and Q7\}

Table A.3 The Query Processing Cost of the Third MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>fq</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp18, Tmp19, Tmp22 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240</td>
<td>323,207,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2</td>
<td>5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251, 600000, 276048000000 and 184082</td>
<td>1,697,558,231,916</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 12018000000, 1500000, 227597, 600000, 276048000000 and 184082</td>
<td>1,997,769,797,079</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 12018000000, 1500000, 227597, 600000, 276048000000 and 184082</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp16, Tmp18, Tmp19, Tmp20 and result5</td>
<td>8000000, 2000000, 160000000000, 8000000 and 36276</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp16, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6</td>
<td>8000000, 2000000, 160000000000, 8000000, 10000, 3627600000 and 36276</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp7, Tmp9, Tmp10, Tmp12, Tmp24, Tmp25 and result7</td>
<td>6000000, 1500000, 227597, 1365582000000, 1500000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the third MVPP 10,821,545,680,471
Figure A.11  The Fourth MVPP, Query in the List: {Q2, Q6, Q1, Q5, Q4, Q7 and Q3}

Table A.4  The Query Processing Cost of the Fourth MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>( f_i )</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp6, Tmp18, Tmp12, Tmp13, Tmp14 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 80000, 20000, 160000000000, 16024000000 and 160240</td>
<td>323,207,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2</td>
<td>5, 1, 25, 25, 150000, 750000, 6689560251, 600000, 276048000000 and 184082</td>
<td>1,697,558,231,196</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp6, Tmp18, Tmp10, Tmp7, Tmp8, Tmp22 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 1365582000000, 1823769557 and 182183</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp16, Tmp18, Tmp12, Tmp20, Tmp5, Tmp21 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 16024000000, 150000, 240360000000 and 967519280</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp12, Tmp13, Tmp14, Tmp15 and result5</td>
<td>800000, 200000, 600000000000, 800000 and 36276</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp12, Tmp13, Tmp14, Tmp15, Tmp16 and result6</td>
<td>800000, 200000, 160000000000, 800000, 10000, 362760000000 and 36276</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp10, Tmp7, Tmp8, Tmp22, Tmp5, Tmp23 and result7</td>
<td>600000, 1500000, 227597, 136558200000, 150000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the fourth MVPP 18,395,672,059,143
Figure A.12 The Fifth MVPP, Query in the List: \{Q6, Q1, Q5, Q4, Q7, Q3 and Q2\}

Table A.5 The Query Processing Cost of the Fifth MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>f_q</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp1, Tmp2, Tmp3, Tmp12 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 16024000000 and 160240</td>
<td>323,207,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp7, Tmp8, Tmp9, Tmp16, Tmp17, Tmp18, Tmp19, Tmp14, Tmp20, Tmp22 and result2</td>
<td>5, 1, 25, 25, 6000000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp21, Tmp16, Tmp17, Tmp18, Tmp19, result3 and and result2</td>
<td>5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 136558200000, 18237695575 and 182183</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp13, Tmp14, Tmp15 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 16024000000, 1500000, 24036000000, and 967519280</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4 result5</td>
<td>800000, 200000, 160000000000, 800000 and 36276</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6 and result6</td>
<td>800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp16, Tmp17, Tmp18, Tmp19, Tmp14, Tmp20, result7</td>
<td>6000000, 1500000, 227597, 136558200000, 1500000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the fifth MVPP: 25,711,148,613,207
Figure A.13 The Sixth MVPP, Query in the List: \{Q1, Q5, Q4, Q7, Q3, Q2, and Q6\}

Table A.6 The Query Processing Cost of the Sixth MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>f_i</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240</td>
<td>67,303,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result1</td>
<td>5, 1, 25, 25, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp7, Tmp9, Tmp11, Tmp12 and result5</td>
<td>800000, 200000, 200000, 72552000000 and 36276</td>
<td>36,282,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp7, Tmp9, Tmp11, Tmp12 and result6</td>
<td>800000, 200000, 200000, 72552000000 and 36276</td>
<td>68,572,856,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp15, Tmp16, Tmp17, Tmp18, Tmp19 and result7</td>
<td>600000, 1500000, 227597, 136558200000, 1500000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the sixth MVPP: 23,316,809,013,207
Figure A.14  The Seventh MVPP, Query in the List: \{Q5, Q4, Q7, Q3, Q2, Q6, and Q1\}

Table A.7  The Query Processing Cost of the Seventh MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>(f_q)</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp1, Tmp2, Tmp3, Tmp22 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240</td>
<td>323,207,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp14, Tmp15, Tmp16, Tmp17, Tmp12, Tmp18, Tmp20 and result2</td>
<td>5, 1, 25, 25, 600000, 1500000, 227597, 136558200000, 1500000, 136577850000, 4552595 and 184082</td>
<td>9,013,034,785,980</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp14, Tmp15, Tmp16, Tmp17, Tmp19 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 600000, 1500000, 227597, 136558200000, 1823769557 and 182183</td>
<td>9,571,896,175,751</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp13 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 15000, 24036000000, and 967519280</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4 and result5</td>
<td>800000, 200000, 160000000000, 800000 and 36276</td>
<td>800,009,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp9, Tmp21 and result6</td>
<td>800000, 200000, 160000000000, 800000, 10000, 3627600000 and 36276</td>
<td>1,443,281,456,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp14, Tmp15, Tmp16, Tmp17, Tmp12, Tmp18 and result7</td>
<td>6000000, 1500000, 227597, 136558200000, 150000, 136577850000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the seventh MVPP  25,711,148,613,207
Figure A.15 The Cheapest MVPP after Re-Optimized.

Table A.8 The Query Processing Cost of the Re-Optimized MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>$f_q$</th>
<th>Access from the nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp18, Tmp19 and result1</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 160240000, 200000, 32048000000 and 1602400</td>
<td>67,303,240,592</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp10, Tmp14, Tmp15 and result2</td>
<td>5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251, 6000000, 276048000000 and 184082</td>
<td>1,697,558,231,916</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp10, Tmp11 and result3</td>
<td>5, 1, 25, 25, 10000, 50000, 6000000, 12018000000, 1500000, 227597, 233369715461 and 182180</td>
<td>1,997,769,797,079</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp12, Tmp23 and result4</td>
<td>5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 9675192800</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp16, Tmp18, Tmp20, Tmp21 and result5</td>
<td>8000000, 2000000, 1000000, 72552000000 and 362704</td>
<td>36,282,181,380</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp16, Tmp18, Tmp20, Tmp21, Tmp5, Tmp22 and result6</td>
<td>8000000, 2000000, 1000000, 72552000000, 1000000, 36276000000 and 3627600000</td>
<td>68,572,856,484</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp7, Tmp9, Tmp10, Tmp24, Tmp12, Tmp25 and result7</td>
<td>6000000, 1500000, 227597, 1365582000000, 1500000, 1365778500000 and 910519</td>
<td>4,506,505,914,348</td>
</tr>
</tbody>
</table>

The total query processing cost of the re-optimized MVPP 8,427,206,080,471

Figure A.15 is the cheapest MVPP after applying our MVPP re-optimization algorithm. Table A.8 shows the query processing cost of re-optimized MVPP.
APPENDIX B

Result of Using Common Subexpression

This appendix provides the analysis result for using sharable subexpression of new queries when they are merged into existing MVPP. Example A shows the details of merging new query in which the subgraph of existing MVPP is the subtree of new query. This situation is mentioned in section 4.6 that is the existing query can use sharable subexpression with new query. Therefore, the existing query is rewritten to use sharable subexpression with new query rather than creating the new node to support only new query.

B.1 Example A

Suppose that the queries for the static phase are Q19, Q22 and Q31, the initial requirements, and the new requirement added in dynamic phase is Q1. The individual optimal plan of Q1, Q19, Q22 and Q31 are shown in Figure B.1

![Figure B.1 Relational Algebra Query Tree of Query Q1, Q19, Q22 and Q31](image-url)
Figure B.1 (Continued)

Suppose that the re-optimized existing MVPP of initial requirements, Q19, Q22 and Q31, is shown in Figure B.2. The optimal dynamic MVPP after Q1 is added into existing MVPP is shown in Figure B.3.

Figure B.4 is the re-optimized MVPP generated by rerunning the static approach for all queries, Q1, Q19, Q22 and Q31.

Figure B.2 The Re-Optimized Existing MVPP in Static Phase for Q19, Q22 and Q31
Figure B.3 The Optimal MVPP of Dynamic Phase by Adding New Query Q1

Figure B.4 The Re-Optimized Cheapest MVPP by the Static Approach for Q19, Q22, Q31 and Q1
Table B.1 shows the query processing cost of all queries for the static approach and the dynamic approach after the materialized view are selected. Table B.2 shows the comparison of the result from the static approach and the dynamic approach.

**Table B.1** The Query Processing Cost

<table>
<thead>
<tr>
<th>Query</th>
<th>Static Approach</th>
<th>Dynamic Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number 1 (Q1)</td>
<td>9,105,190</td>
<td>9,105,190</td>
</tr>
<tr>
<td>Query number 19 (Q19)</td>
<td>276,083,393,841</td>
<td>276,083,393,841</td>
</tr>
<tr>
<td>Query number 22 (Q22)</td>
<td>1,725,517,188,621</td>
<td>1,725,513,451,014</td>
</tr>
<tr>
<td>Query number 31 (Q31)</td>
<td>5,770,009,600</td>
<td>5,763,780,255</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>2,007,379,697,252</strong></td>
<td><strong>2,007,369,730,300</strong></td>
</tr>
</tbody>
</table>

**Table B.2** The Total Query Processing Cost, Maintenance Cost and Total Cost

<table>
<thead>
<tr>
<th>Approach</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>2,007,379,697,252</td>
<td>1,365,589,907,765</td>
<td>3,372,969,605,017</td>
</tr>
<tr>
<td>Dynamic</td>
<td>2,007,369,730,300</td>
<td>1,726,723,846,934</td>
<td>3,734,093,577,234</td>
</tr>
</tbody>
</table>

In conclusion, from the existing MVPP shown in Figure B.2, the node Tmp5 is the subgraph of existing MVPP. Tmp5 is the subsumption of new query Q1 node Tmp4 in Figure B.1 (a). Because the selection condition of Tmp5 in Figure B.2 same as Tmp4 in Figure B.1 (a), which is LINEITEM and ORDERS with selection \((σ_{o\_orderdate >= '1994-01-01' \ and \ o\_orderdate< '1995-01-01')\), plus the additional selection \((σ_{l\_commitdate < l\_receiptdate})\) that is node Tmp2 in Figure B.2.

In Figure B.2 the materialized views are Tmp5 and Tmp15. After the optimal MVPP generated by dynamic approach, Figure B.3, Tmp20 is the new materialized view selected to support new query Q1. Comparing the dynamic approach result with static approach, Figure B.4, the number of materialized views of dynamic approach is
more than that of static approach. In Figure B.4, the result of MVPP by rerunning static approach, Tmp4 is the sharable conjunctive joined operation among the queries. All three queries can be derived from Tmp4 whilst Tmp5 in Figure B.3 are used for Q19 and Q22 only. So, Tmp4 in Figure B.4 is simple common subexpression than Tmp5 in Figure B.3. Therefore, the materialized view maintenance cost of dynamic materialized view approach is higher than that of the static approach.

The result of Table B.2 shows that after selecting view to be materialized, the total cost of static approach, 3,372,969,605,017, is less than dynamic approach, 3,734,093,577,234. Although the total query processing cost of dynamic approach is less than that of static approach, the query processing cost is not less than enough to make the total cost less than static approach. Therefore, the existing query should be rewritten to sharable subexpression with new query that the saving of the materialized view maintenance cost.

**B.2 Example B**

Normally there possibly to have more than one alternative plans when new query is merged into search space. This example shows the experiment how to choose the query processing plan for new query when new query is merged into the existing MVPP if new query has many query processing plan. We use the experiment in section 4.6.3 to explain this situation. According to Figure 4.43 (e) and (f), there are two possible plan of Q13 when Q13 is merged into the existing MVPP that Q12 already merged in the MVPP.

**Plan 1:** \{\{(REGION \Join NATION \Join SUPPLIER \Join PARTSUPP \Join PART)\}\} join with \{\{(LINEITEM) \Join ORDERS \Join CUSTOMER)\}\}. The first subgraph is Tmp19 and the second subgraph is Tmp25. Plan 1 provides query processing cost of Q13 = 869,818,249,255. Figure B.5 shows the MVPP when merge Q13 into the existing MVPP that Q12 already merged in the MVPP. Figure B.6 shows the optimized MVPP by all select, project and aggregation function are pushed down as deep as possible.
represents materialized view node selected by static approach for Q1-Q7

Figure B.5  Plan1: The Dynamic MVPP after Q13 is Merged into the Existing Re-Optimized MVPP

represents materialized view node selected by static approach for Q1-Q7
represents new materialized view node selected by dynamic approach

Figure B.6  Plan 1: The Dynamic Cheapest MVPP of Q8-Q13 after Optimized
Plan 2: \( \{(\text{REGION} \bowtie \text{NATION} \bowtie \text{SUPPLIER} \bowtie \text{PARTSUPP} \bowtie \text{PART}) \bowtie \text{LINEITEM}\} \bowtie \text{ORDERS} \bowtie \text{CUSTOMER} \).  
\( \{(\text{REGION} \bowtie \text{NATION} \bowtie \text{SUPPLIER} \bowtie \text{PARTSUPP} \bowtie \text{PART}) \bowtie \text{LINEITEM}\} \) is the intermediate result of Q12.

**Figure B.7** Plan 2: The Dynamic MVPP after Q13 is merged into the Existing Re-Optimized MVPP

**Figure B.8** Plan 2: The Re-Optimized Cheapest Dynamic MVPP of Q8-Q13
The first subgraph, \{(REGION \bowtie NATION \bowtie SUPPLIER \bowtie PARTSUPP \bowtie PART) \bowtie LINEITEM\}, is the intermediate result of Q12. Next, the result is joined with ORDERS and later joined with CUSTOMER, respectively. Figure Plan 2 in Figure B.7 shows the MVPP when merge Q13 into the existing MVPP that Q12 already merged in MVPP. Plan 2 provides the query processing cost of Q13 is 817,447,115,675. Figure B.8 shows the optimize MVPP by all select, project and aggregation function are pushed down as deep as possible.

As we match the optimal individual plan of query from leaf node to the root node with the MVPP and merge to subgraph of MVPP which provides the number of base relations that are joined conjunctively as much as possible. Therefore, the second plan is chosen because the first subgraph of second plan is constructed on 6 base relations more than others subgraphs. Also the second plan provides the query processing cost less than the first plan. The query processing cost of Q13 for Plan 1 and Plan 2 are calculated and shown in Table B.3.

**Table B.3** The Query Processing Cost of Q13

<table>
<thead>
<tr>
<th>Plan</th>
<th>Accessed from the nodes</th>
<th>Cost of node</th>
<th>Query processing cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp33, Tmp24(materialized view), Tmp12, Tmp25, Tmp34, Tmp35 and result13</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 160240, 32192, 910519, 150000, 13657785000, 910519, 3733997148, 215</td>
<td>869,827,064,180</td>
</tr>
<tr>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32, Tmp33, Tmp9, Tmp10, Tmp34, Tmp12, Tmp35 and result13</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 6000000, 6000000, 122113784, 832,152424, 1500000, 227597, 6998835347, 150000, 711150000</td>
<td>817,446,813,290</td>
</tr>
</tbody>
</table>
APPENDIX C

Result of Merging Queries to Construct Dynamic MVPPs

In this appendix, the details to merge new queries, Q8 to Q13, into the existing re-optimized MVPP are presented. The existing re-optimized MVPP is generated in the static phase for initial requirements, Q1 to Q7, shown in Figure 4.23.

C.1 The First Dynamic MVPP

The first order list of the first dynamic MVPPs of this query set is {Q10, Q8, Q11, Q12, Q9, and Q13}. The details of constructing the first dynamic MVPP already have been described as Figure 4.41 in section 4.6.2. We represent the first dynamic MVPP again in Figure C.1.

![Diagram showing the representation of the first dynamic MVPP](image)

* represents materialized view node selected in static phase for Q1-Q7

Figure C.1 The First Dynamic MVPP after All New Queries are Merged
We push down select, project and aggregate function for the first dynamic MVPP as deep as possible to optimize MVPP. The first dynamic MVPP is shown in Figure C.2. The query processing cost of each query after the first dynamic MVPP is optimized shown in Table C.1.

![Diagram](image)

**Figure C.2** The First Dynamic MVPP after Optimized

bullet represents materialized view node selected in static phase for Q1-Q7
Table C.1  The Query Processing Cost of the First to the Fourth Dynamic MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>f_q</th>
<th>List of Accessed Nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19 and result1</td>
<td>2003, 800000, 1602400000, 200000, 3204800000 and 160240</td>
<td>67,303,124,486</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp15(materialized view) and result2</td>
<td>184082 and 184082</td>
<td>2,208,984</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp11(materialized view) and result3</td>
<td>182183 and 182183</td>
<td>2,550,562</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17,Tmp12, Tmp23 and result4</td>
<td>2003, 800000, 1602400000, 150000, 24036000000, and 967519280</td>
<td>53,213,742,566</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp21(materialized view) and result5</td>
<td>36276 and 36276</td>
<td>362,760</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp21(materialized view), Tmp5, Tmp22, and result6</td>
<td>36276, 10000, 3627600000 and 36276</td>
<td>3,265,582,968</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp24(materialized view), Tmp12, Tmp25 and result7</td>
<td>910519, 150000, 136577850000 and 910519</td>
<td>409,739,463,114</td>
</tr>
<tr>
<td>Q8</td>
<td>6</td>
<td>Tmp7, Tmp27, Tmp18, Tmp26, Tmp28 and result8</td>
<td>6000000, 6000000, 2000000, 200000, 151951851168 and 758746</td>
<td>911,790,059,484</td>
</tr>
<tr>
<td>Q9</td>
<td>4</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14 and result 9</td>
<td>5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251 and 46008</td>
<td>27,488,935,648</td>
</tr>
<tr>
<td>Q10</td>
<td>5</td>
<td>Tmp24(materialized view) and result10</td>
<td>910519, 910519</td>
<td>9,105,190</td>
</tr>
<tr>
<td>Q11</td>
<td>5</td>
<td>Tmp24(materialized view), Tmp29, Tmp16, Tmp30 and result11</td>
<td>910519, 575169, 800000, 460135200000 and 575169</td>
<td>2,300,691,981,035</td>
</tr>
</tbody>
</table>
Table C.1 (Continued)

<table>
<thead>
<tr>
<th>Query</th>
<th>( f_q )</th>
<th>List of Accessed Nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q12</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32 and result12</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 160240, 600000, 6000000, 122113784832 and 152424</td>
<td>311,554,998,998</td>
</tr>
<tr>
<td>Q13</td>
<td>5</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp7, Tmp27, Tmp31, Tmp32, Tmp33, Tmp9, Tmp10, Tmp34, Tmp12, Tmp35 and result13</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 160240, 6000000, 6000000, 122113784832, 152424, 1500000, 227597, 6998835347, 150000, 711150000 and 215</td>
<td>817,446,813,290</td>
</tr>
</tbody>
</table>

The total query processing cost of the first MVPP: 4,902,508,929,085
C.2 The Second Dynamic MVPP

After the first dynamic MVPP is generated, the first element of the list is moved to the end of the list. So Q10 is moved to the end of list, the list becomes \{Q8, Q11, Q12, Q9, Q13 and Q10\}.

We start the second dynamic MVPP with Q8, as there is no conjunctively join available in existing MVPP for PART and LINEITEM. Then, a new intermediate node is introduced. The second dynamic MVPP when Q8 is merged is shown as Figure C.3 (a). Next, when Q11 is merged, the existing join conjunctively intermediate node is available for subtree of Q11 that is Tmp24. There is remaining base relation, PARTSUPP, then the new node is introduced as a join operation between Tmp24 and PARTSUPP, \{Tmp24 \Join PARTSUPP\}. The second dynamic MVPP when Q11 is merged, is shown as Figure C.3 (b).

![Figure C.3 (a) The Second Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP](image-url)
Figure C.3 (b) The Second Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP

Figure C.3 (c) The Second Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP
represents materialized view node selected in static phase for Q1-Q7

Figure C.3 (d) The Second Dynamic MVPP after Merging Q9 into the Existing Re-Optimized MVPP

Next, when Q12 is merged the existing join conjunctively node is available for subtree of Q12 that is Tmp19. The remaining base relation is LINEITEM, then the new node is introduced as a join operation between Tmp19 and LINEITEM. The second dynamic MVPP, when Q12 is merged, is shown as Figure C.3 (c). Later, when Q9 is merged, the existing join conjunctively intermediate node is available for Q9 that is Tmp14. There is no new node created as Q9 is subsumption of the existing MVPP. The dynamic MVPP, when Q9 is merged, is shown as Figure C.3 (d). The next query in the list is Q13, we merge Q13 at the intermediate result of Q12, then join with ORDERS, and CUSTOMER shown in Figure C.3 (e). The last query in the list is Q10. Q10 has conjunctively join (ORDERS \(\bowtie\) LINEITEM) that is already available in the existing MVPP, Tmp24. However, Tmp24 includes the select operation \(\sigma_{\text{o_orderdate}='1994-01-01'}\) and \(\sigma_{\text{o_orderdate} < '1995-01-01'}\) ORDER). Then, this select operation of Q10 is push down. So, Q10 is merged into the existing MVPP at Tmp4 and no new intermediate node created for Q10 as shown in Figure C.3 (f).
\textbf{Figure C.3 (e)} The Second Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP

\textbf{Figure C.3 (f)} The Second Dynamic MVPP after All New Queries are Merged
Next step of the merging new queries into the existing MVPP process, we push down select, project and aggregate function for the second dynamic MVPP as deep as possible to optimize MVPP. The second dynamic MVPP after optimized is shown in Figure C.4. The query processing cost of each query after the second dynamic MVPP is optimized same as the query processing cost of the first dynamic MVPP shown in Table C.1.

![Diagram](image)

*Figure C.4* The Second Dynamic MVPP after Optimized

### C.3 The Third Dynamic MVPP

After the second dynamic MVPP is generated, the first element of the list is moved to the end of the list. So, Q8 is moved to the end of list, the list becomes \{Q11, Q12, Q9, Q13, Q10 and Q8\}.

We start the third dynamic MVPP with Q11, the existing join conjunctively intermediate node is available for subtree of Q11 that is Tmp24. The remaining base relation is PARTSUPP, then the new node is introduced as a join operation between Tmp24 and PARTSUPP, \{Tmp24 \Join PARTSUPP\}. The third dynamic MVPP, when Q11 is merged, is shown as Figure C.5 (a). Next, when Q12 is merged, the existing
conjunctively joined node is available that is Tmp19. The remaining base relation is LINEITEM, then the new node is introduced as a join operation between Tmp19 and LINEITEM. The third dynamic MVPP when Q12 is merged shown as Figure C.5 (b). Later, when Q9 is merged, the existing conjunctively joined node is available for Q9 that is Tmp14. As Q9 is subsumption of existing MVPP then no new node is introduced. The third dynamic MVPP when Q9 is merged shown as Figure C.5 (c). Thereafter, we merge Q13 at the intermediate result of Q12, then join with ORDERS, and CUSTOMER shown in Figure C.5 (d). The next query in the list is Q10. Q10 has conjunctively joined (ORDERS \[ LINEITEM \]) that is already available in the existing MVPP, Tmp24. The select operation (\( \sigma_{o\text{\_orderdate}>=`1994-01-01'\text{ and } o\text{\_orderdate} <`1995-01-01'\text{ ORDER}) of Q10 is pushed down as Tmp24 has this operation. New intermediate node are not created for Q10 as Q10 is subsumption of existing MVPP. Figure C.5(e) show dynamic MVPP when Q10 is merged into the existing MVPP. The last query in the list is Q8, when Q8 is merged, Q8 is classified as nothing in common with existing MVPP then the sharable expression is not available for Q8. So, the new intermediate node is introduced to join PART and LINEITEM. The dynamic MVPP when Q8 is merged shown as Figure C.5 (f).
Figure C.5 (b) The Third Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP

Figure C.5 (c) The Third Dynamic MVPP after Merging Q9 into the Existing Re-Optimized MVPP
Figure C.5 (d) The Third Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP

Figure C.5 (e) The Third Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP

● represents materialized view node selected in static phase for Q1-Q7
Next step of merging is to push down select, project and aggregate function for first dynamic MVPP as deep as possible to optimize MVPP. The third dynamic MVPP after optimized is shown in Figure C.6. The query processing cost of each query after the third dynamic MVPP is optimized same as the query processing cost of the first dynamic MVPP shown in Table C.1.
We repeat these steps of merging algorithm to construct the fourth to sixth dynamic MVPP based on following order list of query:

The fourth dynamic MVPP : \{Q12, Q9, Q13, Q10, Q8 and Q11\}
The fifth dynamic MVPP : \{Q9, Q13, Q10, Q8, Q11 and Q12\}
The sixth dynamic MVPP : \{Q13, Q10, Q8, Q11, Q12 and Q9\}

The pictorial views of merging each query into MVPP show in section C.7 to C.12 the fourth to the sixth dynamic MVPP respectively.

**C.4 The Fourth Dynamic MVPP**

The fourth dynamic MVPP : \{Q12, Q9, Q13, Q10, Q8 and Q11\}

![Diagram](image)

- \(\bullet\) represents materialized view node selected in static phase for Q1-Q7

**Figure C.7 (a) The Third Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP**
represents materialized view node selected in static phase for Q1-Q7

**Figure C.7 (b) The Third Dynamic MVPP after Merging Q9 into the Existing Re-Optimized MVPP**

represents materialized view node selected in static phase for Q1-Q7

**Figure C.7 (c) The Third Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP**
Figure C.7 (d)  The Third Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP

Figure C.7 (e)  The Third Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP
represents materialized view node selected in static phase for Q1-Q7

Figure C.7 (f) The Fourth Dynamic MVPP after All New Queries are Merged

The fourth dynamic MVPP after optimized is shown in Figure C.8. The query processing cost of each query after the fourth dynamic MVPP is optimized same as the query processing cost of the first dynamic MVPP shown in Table C.1.

represents materialized view node selected in static phase for Q1-Q7

Figure C.8 The Fourth Dynamic MVPP after Optimized
C.5 The Fifth Dynamic MVPP

The fifth dynamic MVPP: \{ Q9, Q13, Q10, Q8, Q11 and Q12 \}

![Diagram of the fifth dynamic MVPP]

Figure C.9 (a) The Fifth Dynamic MVPP after Merging Q9 into the Existing Re-Optimized MVPP

![Diagram of the fifth dynamic MVPP after merging Q9]

Figure C.9 (b) The Fifth Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP

![Diagram of the fifth dynamic MVPP after merging Q13]
Figure C.9 (c) The Fifth Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP

Figure C.9 (d) The Fifth Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP
Figure C.9 (e) The Fifth Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP

Figure C.9 (f) The Fifth Dynamic MVPP after All New Queries are Merged
The fifth dynamic MVPP after optimized is shown in Figure C.10. The query processing cost of each query after the fifth dynamic MVPP is optimized shown in Table C.2.

**Figure C.10** The Fifth Dynamic MVPP after Optimized
Table C.2 The Query Processing Cost of the Fifth and the Sixth Dynamic MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>( f_q )</th>
<th>List of Accessed Nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19 and result1</td>
<td>2003, 800000, 16024000000, 200000, 32048000000 and 160240</td>
<td>67,303,124,486</td>
</tr>
<tr>
<td>Q2</td>
<td>6</td>
<td>Tmp15(materialized view) and result2</td>
<td>184082 and 184082</td>
<td>2,208,984</td>
</tr>
<tr>
<td>Q3</td>
<td>7</td>
<td>Tmp11(materialized view) and result3</td>
<td>182183 and 182183</td>
<td>2,550,562</td>
</tr>
<tr>
<td>Q4</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17,Tmp12, Tmp23 and result4</td>
<td>2003, 800000, 16024000000, 150000, 24036000000, and 967519280</td>
<td>53,213,742,566</td>
</tr>
<tr>
<td>Q5</td>
<td>5</td>
<td>Tmp21(materialized view) and result5</td>
<td>36276 and 36276</td>
<td>362,760</td>
</tr>
<tr>
<td>Q6</td>
<td>9</td>
<td>Tmp21(materialized view), Tmp5, Tmp22, and result6</td>
<td>36276, 10000, 362760000 and 36276</td>
<td>3,265,582,968</td>
</tr>
<tr>
<td>Q7</td>
<td>3</td>
<td>Tmp24(materialized view), Tmp12, Tmp25 and result7</td>
<td>910519, 150000, 136577850000 and 910519</td>
<td>409,739,463,114</td>
</tr>
<tr>
<td>Q8</td>
<td>6</td>
<td>Tmp7, Tmp27, Tmp18, Tmp26, Tmp28 and result8</td>
<td>6000000, 6000000, 200000, 200000, 151951851168 and 758746</td>
<td>911,790,059,484</td>
</tr>
<tr>
<td>Q9</td>
<td>4</td>
<td>Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14 and result 9</td>
<td>5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251 and 46008</td>
<td>27,488,935,648</td>
</tr>
<tr>
<td>Q10</td>
<td>5</td>
<td>Tmp24(materialized view) and result10</td>
<td>910519, 910519</td>
<td>9,105,190</td>
</tr>
<tr>
<td>Q11</td>
<td>5</td>
<td>Tmp24(materialized view), Tmp29, Tmp16, Tmp30 and result11</td>
<td>910519, 575169, 800000, 460135200000 and 575169</td>
<td>2,300,691,981,035</td>
</tr>
</tbody>
</table>
Table C.2 (Continued)

<table>
<thead>
<tr>
<th>Query</th>
<th>( f_q )</th>
<th>List of Accessed Nodes</th>
<th>Cost of Nodes</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q12</td>
<td>2</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32 and result12</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 160240, 600000, 600000, 122113784832 and 152424</td>
<td>311,554,998,998</td>
</tr>
<tr>
<td>Q13</td>
<td>5</td>
<td>Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp24(materialized view), Tmp12, Tmp25, Tmp34, Tmp35 and result13</td>
<td>2003, 800000, 1602400000, 200000, 32048000000, 160240, 32192, 910519, 150000, 136577850000, 910519, 3733997148 and 215</td>
<td>869,827,064,180</td>
</tr>
</tbody>
</table>

The total query processing cost of the first MVPP: 4,954,889,179,975
C.6 The Sixth Dynamic MVPP

The sixth dynamic MVPP: \{Q13, Q10, Q8, Q11, Q12 and Q9\}

Figure C.11 (a) The Sixth Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP

Figure C.11 (b) The Sixth Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP
Figure C.11 (c) The Sixth Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP

Figure C.11 (d) The Sixth Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP
Figure C.11 (e) The Sixth Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP

Figure C.11 (f) The Sixth Dynamic MVPP after All New Queries are Merged
The sixth dynamic MVPP after optimized is shown in Figure C.12. The query processing cost of each query after the sixth dynamic MVPP is optimized same as the fifth dynamic MVPP as shown in Table C.2.

![Diagram](image)

Figure C.12 The Sixth Dynamic MVPP after Optimized
APPENDIX D

Result of Affected Node Identification Algorithm

In this appendix, we are presenting the result of directly and indirectly affected node that are identified by an affected node identification algorithm. The affected node identification algorithm is shown in Figure D.1.

begin
  1. Initial list $M_{direct}$ and $M_{indirect} = \phi$
     $M_{direct}$ is the set of directly affected node
     $M_{indirect}$ is the set of indirectly affected node
  2. For each new query
     2.1 Depth first search from the root to base relations to determine the existing intermediate nodes, $v_i$, used to construct the new query.
     2.2 Calculate weight $w(v)$ of each node $v_i$, $v_i$, that are conjunctively joined with positive weight or project operation that is not the ancestor of base relation or select operation, are inserted into the list $M_{direct}$.
  3. For each node $v_i$ in list $M_{direct}$ search its ancestor node $u_j$, $u_j \notin M_{direct}$, up to the query node
     3.1 calculate weight of node $u_j$.
     3.2 if ( weight $v_i > $ weight $u_j$ ) and $u_j$ is existing materialized view then put $u_j$ into list $M_{indirect}$
     3.3 if ( weight $v_i < $ weight $u_j$ ) then
         traverse in bottom-up way to find the node that return maximum weight $u_j$ of each branch.
         put $u_j$ into list $M_{indirect}$
end;

Figure D.1 The Affected Node Identification Algorithm

$$w(v) = \sum_{q \in O_v} \left[ f_q(q) \ast \left(C^d_{a}(v)\right) \right] - \sum_{r \in I_v} \left( f_u(r) \ast \left(C^r_{m}(v)\right) \right)$$

$w(v)$ denotes weight of node

$O_v$ denotes the queries which use view $v$. 
The accessing cost \( a \) for query \( q \) using view \( v \). The cost of answering query \( q \) is the number of rows presented in the relation used to construct \( q \).

\( f_q \) denotes the frequency of executing a query.

\( I_v \) denotes the base relations which are used to produce view \( v \).

\( C'_m \) denotes the maintenance cost \( m \) for materialized view \( v \) based on base relation \( r \), which is occasionally updated.

\( f_u \) denotes the frequency of updating base relation

D.1 The Affected Node of The Subsumption Data Set

![Figure D.2](image)

- **○** represents materialized view node selected in static phase for Q1-Q7
- **●** represents new materialized view node selected in dynamic phase

**Figure D.2** The Existing Re-Optimized MVPP with Q9 and Q10 by Dynamic Approach

Considering Figure D.2, The existing nodes used to construct Q9 are {Tmp14, Tmp10, Tmp9, Tmp13, Tmp4, Tmp3, Tmp2, Tmp1 and Tmp12} and Q10 are {Tmp24, Tmp10, Tmp9 and Tmp7}. According to Table 4.14 showing the weight of the existing node used to construct Q9 and Q10, the details of weight calculation are shown in Table D.1. The details of weight calculation of each node according to Table 4.15, the weight of ancestor node of directly affected node, are shown in Table D.2.
### Table D.1 The Weight of the Existing Node to Construct Q9 and Q10

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight (w(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q1,Q2,Q3,Q4,Q9</td>
<td>Tmp1</td>
<td>1</td>
<td>(2+6+7+2+4)(5) – (1)(5)</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q1,Q2,Q3,Q4,Q9</td>
<td>Tmp1, 2</td>
<td>1</td>
<td>(2+6+7+2+4)(1) – (1)(5+1)</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q1,Q2,Q3,Q4,Q9</td>
<td>Tmp1,2,3</td>
<td>1</td>
<td>(2+6+7+2+4)(25) – (1)(25)</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q1,Q2,Q3,Q4,Q9</td>
<td>Tmp1,2, 3, 4</td>
<td>2</td>
<td>(2+6+7+2+4)(25) – (2)(5+1+25+25)</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q2,Q3,Q7,Q10</td>
<td>Tmp7</td>
<td>1</td>
<td>(6+7+3+5) (6000000) – (1)(6000000)</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q2,Q3,Q7,Q9,Q10</td>
<td>Tmp9</td>
<td>1</td>
<td>(6+7+3+5+4) (15000000) – (1)(15000000)</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q2,Q3,Q7,Q9,Q10</td>
<td>Tmp9, 10</td>
<td>1</td>
<td>(6+7+3+5+4)(227597) – (1)(1500000+227597)</td>
</tr>
<tr>
<td>Tmp12</td>
<td>Q4, Q7, Q2, Q9</td>
<td>Tmp12</td>
<td>1</td>
<td>(2+3+6+4)(150000) – (1)(150000)</td>
</tr>
<tr>
<td>Tmp13</td>
<td>Q2,Q9</td>
<td>Tmp1, 2, 3, 4, 12, 13</td>
<td>3</td>
<td>(6+4)(750000) – (3)(5+1+25+25+150000+750000)</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q2,Q9</td>
<td>Tmp1, 2, 3, 4, 12, 13, 9, 10, 14</td>
<td>4</td>
<td>(6+4)(6869560251)</td>
</tr>
</tbody>
</table>

 TMP24 Q7,Q10 Tmp7, 9, 10, 24 2 (3+5)(136558200000) – 8,193,476,544,806 (2)(6000000+1500000+227597+1365582000000)
Table D.2 The Weight of Ancestor Node of Directly Affected Node of Q9, Q10

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp6</td>
<td>Q1,Q3,Q4</td>
<td>Tmp1, 2, 3, 4, 5, 6</td>
<td>3</td>
<td>(2+7+2) (50000) – (3)*(5+1+25+25+10000) +50000)</td>
</tr>
<tr>
<td>Tmp8</td>
<td>Q7</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8</td>
<td>4</td>
<td>(7)(12018000000) – (4)(5+1+25+25+10000 +50000 +6000000+12018000000)</td>
<td></td>
</tr>
<tr>
<td>Tmp11</td>
<td>Q7</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10</td>
<td>5</td>
<td>(7)(273369715461) – (5) (5+1+25 +25 +10000+ 50000 +600000 +12018000000+ 1500000 +227597+273369715461)</td>
<td></td>
</tr>
<tr>
<td>Tmp17</td>
<td>Q1,Q4</td>
<td>Tmp1, 2, 3, 4, 5, 6, 16, 17</td>
<td>4</td>
<td>(2+2)(16024000000) – (4)(5+1+25+25+10000 +50000 +800000+16024000000)</td>
<td></td>
</tr>
<tr>
<td>Tmp19</td>
<td>Q1</td>
<td>Tmp1, 2, 3, 4, 5, 6, 16, 17, 19</td>
<td>5</td>
<td>(2)(320480000000) – (5)(5+1+25+25+10000 +50000 +8000000+1602400000 +200000+32048000000)</td>
<td></td>
</tr>
<tr>
<td>Tmp23</td>
<td>Q4</td>
<td>Tmp1, 2, 3, 4, 5, 6, 16, 17, 12, 23</td>
<td>5</td>
<td>(2)(240360000000) – (5)(5+1+25+25+10000 +50000 +8000000+1602400000 +150000+24036000000)</td>
<td></td>
</tr>
</tbody>
</table>

-3,440,224
-104,160,300,280
-80,125,050,280
Therefore, the affected nodes are:

Directly affected nodes: Tmp14, Tmp13, Tmp10, Tmp9, Tmp4, Tmp2, Tmp1 and Tmp24

Indirectly affected nodes: Tmp11, Tmp15
D.2 The Affected Node of The Partially Overlapping Data Set Data Set

represents materialized view node selected in static phase for Q1-Q7
represents new materialized view node selected in dynamic phase

Figure D.3 The Existing Re-Optimized MVPP with Q11 and Q12 by Dynamic Approach

Considering Figure D.3, the existing nodes used to construct Q11 are \{Tmp16, Tmp24, Tmp10, Tmp9 and Tmp7\} and Q12 are \{Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1 and Tmp7\}. According to Table 4.17 showing the weight of the existing node used to construct Q11 and Q12, the details of weight calculation are shown in Table D.3. The details of weight calculation of each node according to Table 4.18, the weight of ancestor node of directly affected node, are shown in Table D.4.
Table D.3 The Weight of the Existing Node to Construct Q11 and Q12

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight (w(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q1,Q2,Q3,Q4,Q12</td>
<td>Tmp1</td>
<td>1</td>
<td>(2+6+7+2+2)(5) – (1)(5)</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q1,Q2,Q3,Q4,Q12</td>
<td>Tmp1,2</td>
<td>1</td>
<td>(2+6+7+2+2)(1) – (1)(5+1)</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q1,Q2,Q3,Q4,Q12</td>
<td>Tmp3</td>
<td>1</td>
<td>(2+6+7+2+2)(25) – (1)(25)</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q1,Q2,Q3,Q4,Q12</td>
<td>Tmp1, 2, 3, 4</td>
<td>2</td>
<td>(2+6+7+2+2)(25) – (2)(5+1+25+25)</td>
</tr>
<tr>
<td>Tmp5</td>
<td>Q1,Q3,Q4,Q6,Q12</td>
<td>Tmp5</td>
<td>1</td>
<td>(2+7+2+9+2)(10000) – (1)(10000)</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Q1,Q3,Q4,Q12</td>
<td>Tmp1, 2, 3, 4, 5, 6</td>
<td>3</td>
<td>(2+7+2+2)(50000) – (3)(5+1+25+25+10000 + 50000)</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q2,Q3,Q7,Q11</td>
<td>Tmp7</td>
<td>1</td>
<td>(6+7+3+5)(6000000) – (1)(6000000)</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q2,Q3,Q7,Q11</td>
<td>Tmp9</td>
<td>1</td>
<td>(6+7+3+5)(1500000) – (1)(1500000)</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q2,Q3,Q7,Q11</td>
<td>Tmp9, 10</td>
<td>1</td>
<td>(6+7+3+5)(227597) – (1)(1500000+227597)</td>
</tr>
<tr>
<td>Tmp16</td>
<td>Q1,Q4,Q5,Q6,Q12</td>
<td>Tmp16</td>
<td>1</td>
<td>(2+2+5+9+2)(800000) – (1)(800000)</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Q1,Q4,Q12</td>
<td>Tmp1,2,3, 4, 5, 6, 16, 17</td>
<td>4</td>
<td>(2+2+2)(1602400000) – (4)(5+1+25+25+10000+800000+1602400000)</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Q1,Q5,Q6,Q12</td>
<td>Tmp18</td>
<td>1</td>
<td>(2+5+9+2)(2000000) – (1)(2000000)</td>
</tr>
<tr>
<td>Tmp19</td>
<td>Q1,Q12</td>
<td>Tmp1,2,3,4,5,6,16,17,18,19</td>
<td>5</td>
<td>(2+2)(32048000000) – (5)(5+1+25+25+10000+50000+800000+1602400000+2000000+32048000000)</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Q7,Q11</td>
<td>Tmp7, 9, 10, 24</td>
<td>2</td>
<td>(3+5)(1365582000000) – (2)(6000000+1500000+227597+1365582000000)</td>
</tr>
<tr>
<td>Directly Affected Node</td>
<td>Ancestor Node</td>
<td>Query</td>
<td>Derived Node</td>
<td>Number of Base Relations</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-------</td>
<td>--------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>Q2</td>
<td>Tmp1, 2, 3, 4, 12, 13</td>
<td>3</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q2</td>
<td>Tmp1,2,3, 4, 12, 9, 10, 13, 14</td>
<td>4</td>
<td>(6)(6869560251) – (4)(5+1+25+25+150000 +750000 + 1500000 +227597+6869560251) 13,728,609,890</td>
</tr>
<tr>
<td>Tmp15</td>
<td>Q2</td>
<td>Tmp1,2,3,4, 7,12,9,10,13,14, 15</td>
<td>5</td>
<td>(6)(276048000000) – (5)(5+1+25+25+ 150000 +750000 + 1500000 +227597+6869560251 +6000000 +276048000000) 241,657,060,480</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>Q3</td>
<td>Tmp1,2,3,4,5,6,7,8</td>
<td>4</td>
</tr>
<tr>
<td>Tmp11</td>
<td>Q3</td>
<td>Tmp1,2,3,4,5,6,7,8,9,10,11</td>
<td>5</td>
<td>(7)(273369715461) – (5)(5+1+25+25+10000 +50000+6000000+12018000000+ 150000 +227597+273369715461) 486,610,492,657</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Tmp19</td>
<td>Q1,Q12</td>
<td>Tmp1,2,3,4,5,6,16,17,18,19</td>
<td>5</td>
</tr>
<tr>
<td>Tmp23</td>
<td>Q4</td>
<td>Tmp1,2,3,4,5,6, 12,16,17, 23</td>
<td>5</td>
<td>(2)(240360000000) – (5)(5+1+25+25 +10000 +50000 +800000+1602400000 +150000 + 24036000000) 80,125,050,280</td>
</tr>
</tbody>
</table>
Therefore, the affected nodes are:

- **Directly affected nodes:** Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp17 and Tmp24
- **Indirectly affected nodes:** Tmp11, Tmp15
D.3 The Affected Node of Deleting the Query

represents materialized view node selected in static phase for Q1-Q7

Figure D.4 The Re-Optimized MVPP with Deleting Q3 by Dynamic and Static Approach

Considering Figure D.4, The existing nodes used to construct Q3 are {Tmp11, Tmp10, Tmp9, Tmp8, Tmp7, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1}. After Q3 deleted, the nodes used to construct only Q3 are deleted that are Tmp11 and Tmp8, the nodes used to construct Q3 and the other queries are remained and might be identified as the directly affected node. According to Table 4.20 showing the weight of the existing node for Q3, the details of weight calculation are shown in Table D.5. The details of weight calculation of each node according to Table 4.21, the weight of ancestor node of directly affected node, are shown in Table D.6.
### Table D.5  The Weight of the Existing Node of Q3

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q1,Q2,Q4</td>
<td>Tmp1</td>
<td>1</td>
<td>((2+6+7+2)(5) - (1)(5))</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q1,Q2,Q4</td>
<td>Tmp1,2</td>
<td>1</td>
<td>((2+6+7+2)(1) - (1)(5+1))</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q1,Q2,Q4</td>
<td>Tmp3</td>
<td>1</td>
<td>((2+6+7+2)(25) - (1)(25))</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q1,Q2,Q4</td>
<td>Tmp1,2,3,4</td>
<td>2</td>
<td>((2+6+7+2)(25) - (2)(5+1+25+25))</td>
</tr>
<tr>
<td>Tmp5</td>
<td>Q1,Q4,Q6</td>
<td>Tmp5</td>
<td>1</td>
<td>((2+2+9)(10000) - (1)(10000))</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Q1,Q4</td>
<td>Tmp1,2,3,4,5,6</td>
<td>3</td>
<td>((2+2)(50000) - (3)(5+1+25+25+10000+50000))</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q2,Q7</td>
<td>Tmp7</td>
<td>1</td>
<td>((6+3)(6000000) - (1)(6000000))</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q2,Q7</td>
<td>Tmp9</td>
<td>1</td>
<td>((6+3)(1500000) - (1)(1500000))</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q2,Q7</td>
<td>Tmp9,10</td>
<td>1</td>
<td>((6+3)(227597) - (1)(1500000+227597))</td>
</tr>
</tbody>
</table>
### Table D.6 The Weight of Ancestor Node of Directly Affected Node of Q3

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>Q2</td>
<td>Tmp1, 2, 3, 4, 12, 13</td>
<td>3</td>
<td>(6)(750000) – (3)(5+1+25+25+150000+750000) – (2)(16024000000) – (4)(5+1+25+25+10000+50000+800000+16024000000)</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q2</td>
<td>Tmp1,2,3, 4, 12, 9, 10, 13, 14</td>
<td>4</td>
<td>(6)(6869560251) – (4)(5+1+25+25+150000+750000 + 1500000 + 227597 + 6869560251)</td>
<td>13,728,609,890</td>
</tr>
<tr>
<td>Tmp15</td>
<td>Q2</td>
<td>Tmp1,2,3,4, 7,12,9,10,13,14, 15</td>
<td>5</td>
<td>(6)(276048000000) – (5)(5+1+25+25+150000+750000 +1500000 + 227597 + 6869560251 + 6000000 + 276048000000)</td>
<td>241,657,060,480</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Tmp17</td>
<td>Q1,Q4</td>
<td>Tmp1,2,3,4,6,16,17</td>
<td>4</td>
<td>(2+2)(16024000000) – (4)(5+1+25+25+10000+50000+800000+16024000000)</td>
</tr>
<tr>
<td>Tmp19</td>
<td>Q4</td>
<td>Tmp1,2,3,4,6,16,17,18,19</td>
<td>5</td>
<td>(2)(32048000000) – (5)(5+1+25+25+10000+50000+800000+16024000000 + 200000+32048000000)</td>
<td>-104,160,300,280</td>
</tr>
<tr>
<td>Tmp23</td>
<td>Q4</td>
<td>Tmp1,2,3,4,6,12,16,17,23</td>
<td>5</td>
<td>(2)(24036000000) – (5)(5+1+25+25+10000+50000+800000+16024000000 + 24036000000)</td>
<td>-80,125,050,280</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Tmp24</td>
<td>Q7</td>
<td>Tmp7, 9,10, 24</td>
<td>2</td>
<td>(3)(1365582000000) – (2)(6000000 + 1500000 + 227597 + 1365582000000)</td>
</tr>
</tbody>
</table>
Table D.6 (Continued)

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp10</td>
<td>Tmp25</td>
<td>Tmp7, 9, 10, 12, 24, 25,</td>
<td>3</td>
<td>(3)(136577850000) – (3)(6000000+1500000+227597+150000+1365582000000+136577850000) -4,096,769,632,791</td>
</tr>
</tbody>
</table>

Therefore, the affected nodes are:

Directly affected nodes: Tmp10, Tmp9, Tmp6, Tmp4, Tmp2 and Tmp1

Indirectly affected nodes: Tmp15 and Tmp24
D.4 The Affected Node of Adding Query Constructed on All Base Relations

Figure D.5 The Existing Re-Optimized MVPP with Q1 by Dynamic Approach

Considering Figure D.5, the existing nodes used to construct Q3 are \{Tmp19, Tmp18, Tmp17, Tmp16, Tmp6, Tmp5, Tmp4, Tmp3, Tmp2, Tmp1, Tmp25, Tmp24, Tmp10, Tmp9, Tmp7, and Tmp12\}. According to Table 4.23 showing the weight of the existing node used to construct Q13, the details of weight calculation are shown in Table D.7. The details of weight calculation of each node according to Table 4.24, the weight of ancestor node of directly affected node, are shown in Table D.8.
### Table D.7 The Weight of the Existing Node to Construct Q13

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q1,Q2,Q3,Q4,Q13</td>
<td>Tmp1</td>
<td>1</td>
<td>((2+6+7+2+5)(5) - (1)(5))</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q1,Q2,Q3,Q4,Q13</td>
<td>Tmp1,2</td>
<td>1</td>
<td>((2+6+7+2+5)(1) - (1)(5))</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q1,Q2,Q3,Q4,Q13</td>
<td>Tmp3</td>
<td>1</td>
<td>((2+6+7+2+5)(25) - (1)(25))</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q1,Q2,Q3,Q4,Q13</td>
<td>Tmp1,2,3,4</td>
<td>2</td>
<td>((2+6+7+2+5)(25) - (2)(5+1+25+25))</td>
</tr>
<tr>
<td>Tmp5</td>
<td>Q1,Q3,Q4,Q6,Q13</td>
<td>Tmp5</td>
<td>1</td>
<td>((2+7+2+9+5)(10000) - (1)(10000))</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Q1,Q3,Q4,Q13</td>
<td>Tmp1,2,3,4,5,6</td>
<td>3</td>
<td>((2+7+2+5)(50000) - (3)(5+1+25+25+10000+50000))</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q2,Q3,Q7,Q13</td>
<td>Tmp7</td>
<td>1</td>
<td>((6+7+3+5)(6000000) - (1)(6000000))</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q2,Q3,Q7,Q13</td>
<td>Tmp9</td>
<td>1</td>
<td>((6+7+3+5)(1500000) - (1)(1500000))</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q2,Q3,Q7,Q13</td>
<td>Tmp9,10</td>
<td>1</td>
<td>((6+7+3+5)(227597) - (1)(1500000+227597))</td>
</tr>
<tr>
<td>Tmp12</td>
<td>Q2, Q4,Q7,Q13</td>
<td>Tmp12</td>
<td>1</td>
<td>((6+2+3+5)(150000) - (1)(150000))</td>
</tr>
<tr>
<td>Tmp16</td>
<td>Q1,Q4,Q5,Q6,Q13</td>
<td>Tmp16</td>
<td>1</td>
<td>((2+2+5+9+5)(800000) - (1)(800000))</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Q1,Q4,Q13</td>
<td>Tmp1,2,3,4,5,6,16,17</td>
<td>4</td>
<td>((2+5)(1602400000) - (4)(5+1+25+25+10000)) (+50000+800000+1602400000)</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Q1,Q5,Q6,Q13</td>
<td>Tmp18</td>
<td>1</td>
<td>((2+5+9+9+5)(200000) - (1)(200000))</td>
</tr>
<tr>
<td>Tmp19</td>
<td>Q1,Q13</td>
<td>Tmp1,2,3,4,5,6,16,17,18,19</td>
<td>5</td>
<td>((2+5)(32048000000) - (5)(5+1+25+25+10000)) (+50000+800000+1602400000+200000+3204800000)</td>
</tr>
</tbody>
</table>
### Table D.7 (Continued)

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight ( $w(v)$ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp24</td>
<td>Q7,Q13</td>
<td>Tmp7,9,10,24</td>
<td>2</td>
<td>$(3+5)(1365582000000) - (2)(6000000+ 1500000+ 227597 + 1365582000000) = 8,193,476,544,806$</td>
</tr>
<tr>
<td>Tmp25</td>
<td>Q7,Q13</td>
<td>Tmp7,9,10,12,24,25</td>
<td>3</td>
<td>$(3+5)(136577850000) - (3)(6000000+ 1500000+ 227597 + 1365582000000+136577850000) = -3,413,880,382,791$</td>
</tr>
</tbody>
</table>

### Table D.8 The Weight of Ancestor Node of Directly Affected Node of Q13

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp4</td>
<td>Tmp13</td>
<td>Q2</td>
<td>Tmp1, 2, 3, 4, 12, 13</td>
<td>3</td>
<td>$(6)(750000) - (3)(5+1+25+25+150000+750000) = 1,799,832$</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q2</td>
<td>Tmp1,2,3, 4, 12, 9, 10, 13, 14</td>
<td>4</td>
<td>$(6)(6869560251) - (4)( 5+1+25+25+150000+750000 + 1500000 + 227597+6869560251) = 13,728,609,890$</td>
<td></td>
</tr>
<tr>
<td>Tmp15</td>
<td>Q2</td>
<td>Tmp1,2,3,4, 7,12,9,10,13,14, 15</td>
<td>5</td>
<td>$(6)(276048000000) - (5)( 5+1+25+25+150000 + 750000 + 1500000 + 227597 + 6869560251+6000000 + 276048000000) = 241,657,060,480$</td>
<td></td>
</tr>
</tbody>
</table>
Table D.8 (Continued)

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>Q3</td>
<td>Tmp1,2,3,4,5,6,7,8</td>
<td>4</td>
<td>(7)(120180000000) – (4)(5+1+25+25+ +10000+50000+6000000+12018000000)</td>
</tr>
<tr>
<td>Tmp11</td>
<td>Q3</td>
<td></td>
<td>Tmp1,2,3,4,5,6,7,8,9,10,11</td>
<td>5</td>
<td>(7)(273369715461) – (5)(5+1+25+25+ +10000+50000+6000000+12018000000+ +15000000+227597+273369715461)</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>Q7,Q13</td>
<td>Tmp7,9,10,12,24,25</td>
<td>3</td>
<td>(3+5)(136577850000) –(3)(6000000+1500000+227597+1500000+1365582000000+136577850000)</td>
</tr>
</tbody>
</table>

Therefore, the affected nodes are:

- Directly affected nodes: Tmp19, Tmp17, Tmp6, Tmp4, Tmp2, Tmp1, Tmp24, Tmp10 and Tmp9
- Indirectly affected nodes: Tmp11 and Tmp15
D.5 The Affected Node for Adding All Queries of the First Experiment

represents materialized view node selected in static phase for Q1-Q7

Figure D.6 The First Dynamic MVPP after Optimized, Queries in the List: \{ Q10, Q8, Q11, Q12, Q9 and Q13 \}

The initial requirements for the first experiment are Q1 to Q7. The new requirements are Q8 to Q13 that are merged into the existing re-optimized MVPP. The existing MVPP is generated from Q1 to Q7 in the static phase by static materialized view selection approach.

According to Table 4.29 showing the weight of the existing node used to construct all new queries, the details of weight calculation are shown in Table D.9. The details of weight calculation of each node according to Table 4.30, the weight of ancestor node of directly affected node, are shown in Table D.10
<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q1,Q2,Q3,Q4,Q9,Q12,Q13</td>
<td>Tmp1</td>
<td>1</td>
<td>(2+6+7+2+4+2+5)(5) – (1)(5)</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q1,Q2,Q3,Q4,Q9,Q12,Q13</td>
<td>Tmp1,2</td>
<td>1</td>
<td>(2+6+7+2+4+2+5)(1) – (1)(5+1)</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q1,Q2,Q3,Q4,Q9,Q12,Q13</td>
<td>Tmp3</td>
<td>1</td>
<td>(2+6+7+2+4+2+5)(25) – (1)(25)</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q1,Q2,Q3,Q4,Q9,Q12,Q13</td>
<td>Tmp1,2,3,4</td>
<td>2</td>
<td>(2+6+7+2+4+2+5)(25) – (2)(5+1+25+25)</td>
</tr>
<tr>
<td>Tmp5</td>
<td>Q1,Q3,Q4,Q12,Q13</td>
<td>Tmp5</td>
<td>1</td>
<td>(2+7+2+9+2+5)(10000) – (1)(10000)</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Q1,Q3,Q4,Q12,Q13</td>
<td>Tmp1,2,3,4,5,6</td>
<td>3</td>
<td>(2+7+2+2+5)(50000) – (3)(5+1+25+25+10000+50000)</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q2,Q3,Q7,Q8,Q10,Q11,Q12,Q13</td>
<td>Tmp7</td>
<td>1</td>
<td>(6+7+3+6+5+5+2+5)(6000000) – (1)(6000000)</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q2,Q3,Q7,Q9,Q10,Q11,Q13</td>
<td>Tmp9</td>
<td>1</td>
<td>(6+7+3+4+5+5+5)(1500000) – (1)(1500000)</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q2,Q3,Q7,Q9,Q10,Q11,Q13</td>
<td>Tmp9,10</td>
<td>1</td>
<td>(6+7+3+4+5+5+5)(227597) – (1)(1500000+227597)</td>
</tr>
<tr>
<td>Tmp12</td>
<td>Q2, Q4,Q7,Q9,Q13</td>
<td>Tmp12</td>
<td>1</td>
<td>(6+2+3+4+5)(150000) – (1)(150000)</td>
</tr>
<tr>
<td>Tmp13</td>
<td>Q2,Q9</td>
<td>Tmp1, 2, 3, 4, 12, 13</td>
<td>3</td>
<td>(6+4)(750000) – (3)(5 + 1 + 25 + 25 + 150000 + 750000)</td>
</tr>
<tr>
<td>Existing Node</td>
<td>Query</td>
<td>Derived Node</td>
<td>Number of Base Relations</td>
<td>Weight ( ( w(v) ) )</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q2,Q9</td>
<td>Tmp1, 2, 3, 4, 12, 13, 9, 10, 14</td>
<td>4</td>
<td>(6+4)(6869560251) – (4)(5+1+25+25+150000+750000+1500000+27597+6869560251)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>41,206,850,894</td>
</tr>
<tr>
<td>Tmp16</td>
<td>Q1,Q4,Q5,Q6,Q11,Q12,Q13</td>
<td>Tmp16</td>
<td>1</td>
<td>(2+2+5+9+5+2+5)(800000) – (1)(800000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>23,200,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Q1,Q4,Q12,Q13</td>
<td>Tmp1,2,3,4,5,6,16,17</td>
<td>4</td>
<td>(2+2+2+5)(1602400000) – (4)(5+1+25+25+10000+50000+800000+1602400000+10000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11,213,359,776</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Q1,Q5,Q6,Q8,Q12,Q13</td>
<td>Tmp18</td>
<td>1</td>
<td>(2+5+9+6+2+5)(200000) – (1)(200000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5,600,000</td>
</tr>
<tr>
<td>Tmp19</td>
<td>Q1,Q12,Q13</td>
<td>Tmp1,2,3,4,5,6,16,17,18,19</td>
<td>5</td>
<td>(2+2+5)(32048000000) – (5)(5+1+25+25+10000+50000+800000+1602400000+200000+3204800000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120,174,699,720</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Q7,Q10,Q11</td>
<td>Tmp7,9,10,24</td>
<td>2</td>
<td>(3+5+5)(1365582000000) – (2)(6000000+1500000+27597+1365582000000)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15,021,386,544,806</td>
</tr>
</tbody>
</table>
Table D.10  The Weight of Ancestor Node of Directly Affected Node of All Queries of the First Experiment

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>Tmp8</td>
<td>Q3</td>
<td>Tmp1,2,3,4,5,6,7,8</td>
<td>4</td>
<td>(7)(12018000000) – (4)(5+1+25+25+10000+50000+6000000+12018000000) – (5+1+25+25+10000+50000+6000000+12018000000+150000+227597+273369715461) = 36,029,759,776</td>
</tr>
<tr>
<td>Tmp11</td>
<td>Q3</td>
<td>Tmp1,2,3,4,5,6,7,8,9,10,11</td>
<td>5</td>
<td>(7)(273369715461) – (5)(5+1+25+25+10000+50000+6000000+12018000000+150000+227597+273369715461) = 486,610,492,657</td>
<td></td>
</tr>
<tr>
<td>Tmp14</td>
<td>Tmp15</td>
<td>Q2</td>
<td>Tmp1,2,3,4,7,12,9,10,13,14,15</td>
<td>5</td>
<td>(6)(276048000000) – (5)(5+1+25+25+150000+750000+1500000+227597+6869560251+6000000+276048000000) = 241,657,060,480</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Tmp23</td>
<td>Q4</td>
<td>Tmp1,2,3,4,5,6,16,17,12,23</td>
<td>5</td>
<td>(2)(240360000000) – (5)(5+1+25+25+10000+500000+800000+1602400000+150000+24036000000) = -80,125,050,280</td>
</tr>
<tr>
<td>Tmp24</td>
<td>Tmp25</td>
<td>Q7</td>
<td>Tmp7,9,10,24</td>
<td>3</td>
<td>(3)(136577850000) – (3)(6000000+1500000+227597+1365558200000+150000+136577850000) = -4,096,769,632,791</td>
</tr>
</tbody>
</table>

Therefore, the affected nodes are:

- Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp9, Tmp10, Tmp13, Tmp14, Tmp17, Tmp19, and Tmp24
- Indirectly affected nodes: Tmp11, Tmp15
D.6 The Affected Node for Adding All Queries of the Second Experiment

represents materialized view node selected in static phase for Q1-Q7

Figure D.9 The Dynamic MVPP after Merged All New Queries and Optimized

The initial requirements for the second experiment are \{Q4, Q15, Q22, Q33, Q40, Q43 and Q50\} shown in section 4.9.1. The new requirements \{Q3, Q6, Q28, Q30, Q31 and Q47\} shown in section 4.9.2 are merged into the existing re-optimized MVPP. The existing MVPP is generated from \{Q4, Q15, Q22, Q33, Q40, Q43 and Q50\} in the static phase by static materialized view selection approach.

Table D.11 shows the details of weight calculation of the existing node used to construct all new queries, \{Q3, Q6, Q28, Q30, Q31 and Q47\}. The details of weight calculation of ancestor node of directly affected node, are shown in Table D.12.
<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight ( w(v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>Q33, Q40, Q50, Q28, Q30, Q31, Q47</td>
<td>Tmp1</td>
<td>1</td>
<td>((6+4+5+5+5+5+5) \cdot (5) - (1)(5)) 165</td>
</tr>
<tr>
<td>Tmp2</td>
<td>Q33, Q40, Q50, Q28, Q30, Q31, Q47</td>
<td>Tmp1, 2</td>
<td>1</td>
<td>((6+4+5+5+5+5) \cdot (1) - (1)(5+1)) 28</td>
</tr>
<tr>
<td>Tmp3</td>
<td>Q33, Q40, Q50, Q28, Q30, Q31, Q47</td>
<td>Tmp3</td>
<td>1</td>
<td>((6+4+5+5+5+5+5) \cdot (25) - (1)(25)) 825</td>
</tr>
<tr>
<td>Tmp4</td>
<td>Q33, Q40, Q50, Q28, Q30, Q31, Q47</td>
<td>Tmp1, 2, 3, 4</td>
<td>2</td>
<td>((6+4+5+5+5+5+5) \cdot (25) - (2)(5+1+25+25)) 738</td>
</tr>
<tr>
<td>Tmp5</td>
<td>Q15, Q33, Q40, Q43, Q50, Q28, Q31, Q47, Tmp5</td>
<td>Tmp5</td>
<td>1</td>
<td>((5+6+4+7+5+5+5+5) \cdot (10000) - (1)(10000)) 450,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>Q28, Q33, Q40, Q31, Q47</td>
<td>Tmp1, 2, 3, 4, 5, 6</td>
<td>3</td>
<td>((5+6+4+5+5)(50000) - (3)(5+1+25+25+10000+50000)) 1,069,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>Q4, Q15, Q22, Q33, Q40, Q43, Q50, Q6, Q28, Q30, Q47</td>
<td>Tmp7</td>
<td>1</td>
<td>((6+5+3+6+4+7+5+4+7+5+5+4+5)(800000) - (1)(800000)) 44,800,000</td>
</tr>
<tr>
<td>Tmp8</td>
<td>Q28, Q33, Q40, Q47</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8</td>
<td>4</td>
<td>((5+6+4+5)(16024000000) - (4)(5+1+25+25+10000+50000+800000+1602400000)) 25,634,959,776</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Q28, Q40, Q50</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td>4</td>
<td>((5+4+5)(160240) - (4)(5+1+25+25+10000+50000+800000+1602400000+160240)) -6,411,437,824</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Q6, Q15, Q22, Q40, Q43, Q50</td>
<td>Tmp10</td>
<td>1</td>
<td>((7+5+3+4+7+5)(200000) - (1)(200000)) 6,000,000</td>
</tr>
<tr>
<td>Tmp11</td>
<td>Q6, Q15, Q40</td>
<td>Tmp10, 11</td>
<td>1</td>
<td>((7+5+4)(200000) - (1)(200000+200000)) 2,800,000</td>
</tr>
<tr>
<td>Existing Node</td>
<td>Query</td>
<td>Derived Node</td>
<td>Number of Base Relations</td>
<td>Weight ( w(v) )</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
<td>--------------</td>
<td>--------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Tmp13</td>
<td>Q4,Q22,Q33,Q43,Q50,Q30,Q31,Q47</td>
<td>Tmp13</td>
<td>1</td>
<td>(6+3+6+7+5+4+4+5+5)(6000000) – (1)(6000000) = 264,000,000</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Q4,Q22,Q33,Q43,Q50,Q31,Q47</td>
<td>Tmp13, 14</td>
<td>1</td>
<td>(6+3+6+7+5+4+5+5)(6000000) – (1)(6000000) = 234,000,000</td>
</tr>
<tr>
<td>Tmp15</td>
<td>Q33,Q47</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 15</td>
<td>5</td>
<td>(6+5)(607837751040) – (5)(5+1+25+125+625+390625+1562500) = 3,638,950,205,960</td>
</tr>
<tr>
<td>Tmp16</td>
<td>Q22,Q43,Q50,Q31,Q47</td>
<td>Tmp16</td>
<td>1</td>
<td>(3+7+5+5+5)(1500000) – (1)(1500000) = 36,000,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>Q22,Q43,Q50,Q31,Q47</td>
<td>Tmp16, 17</td>
<td>1</td>
<td>(3+7+5+5+5)(227597) – (1)(1500000+227597) = 3,962,328</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Q22,Q43,Q50,Q31</td>
<td>Tmp13, 14, 16, 17, 18</td>
<td>2</td>
<td>(3+7+5+5)(863342789712) – (2)(6000000 + 6000000+1500000+227597 + 863342789712) = 15,540,142,759,622</td>
</tr>
<tr>
<td>Tmp22</td>
<td>Q43,Q50,Q30</td>
<td>Tmp22</td>
<td>1</td>
<td>(7+5+4)(150000) – (1)(150000) = 2,250,000</td>
</tr>
<tr>
<td>Tmp27</td>
<td>Q6, Q15</td>
<td>Tmp7, 10, 11, 27</td>
<td>2</td>
<td>(7+5)(32046400000) – (2)(800000+2000000+32046400000) = 320,461,600,000</td>
</tr>
</tbody>
</table>
Table D.12  The Weight of Ancestor Node of Directly Affected Node of the Second Experiment

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Query</th>
<th>Derived Node</th>
<th>Number of Base Relations</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp11</td>
<td>Tmp12</td>
<td>Q40, Q50</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12</td>
<td>5</td>
<td>((4+5)(1288745976) - (5)(5+1+25+25+10000 + 500000 + 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976)) -2,864,117,576</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 24</td>
<td>7</td>
<td>((5)(3733997148) - (7)(5+1+25+25+10000 + 500000 + 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976 + 6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 3733997148)) -6,071,211,579,363</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Q50</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 22, 24, 25</td>
<td>8</td>
<td>((5)(711150000) - (8)(5+1+25+25+10000 + 500000 + 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976 + 6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 3733997148 + 150000+ 711150000)) -6,961,999,295,832</td>
</tr>
<tr>
<td>Tmp14</td>
<td>Tmp26</td>
<td>Q4</td>
<td>Tmp7, 13, 14, 26</td>
<td>2</td>
<td>((6)(3034636800000) - (2)(6000000 + 6000000 + 800000 + 3034636800000)) 12,138,521,600,000</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Tmp19</td>
<td>Q22, Q43</td>
<td>Tmp7, 13, 14, 16, 17, 18, 19</td>
<td>3</td>
<td>((3+7)(460135200000) - (3)(6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 8000000 + 460135200000)) 630,852,848,073</td>
</tr>
<tr>
<td>Directly Affected Node</td>
<td>Ancestor Node</td>
<td>Query</td>
<td>Derived Node</td>
<td>Number of Base Relations</td>
<td>Weight of Ancestor Node</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------</td>
<td>-------------</td>
<td>-------------------------</td>
<td>--------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>Tmp18</td>
<td>Tmp20</td>
<td>Q22, Q43</td>
<td>Tmp7, 13, 14, 16, 17, 18, 19, 10, 20</td>
<td>4</td>
<td>(3+7)(115033800000) – (4)(600000+ 600000+ 1500000+ 227597 + 863342789712 + 800000+ 460135200000 + 200000 + 115033800000) – 4,603,796,869,236</td>
</tr>
<tr>
<td>Tmp21</td>
<td>Q43</td>
<td>Tmp7, 13, 14, 16, 17, 18, 19, 10, 20, 5, 21</td>
<td>5</td>
<td>(7)(5751690000) – (5)(6000000 + 6000000+ 1500000 + 227597 + 863342789712 + 8000000 + 460135200000 + 200000 + 115033800000 + 10000 + 5751690000) – 7,181,165,256,545</td>
<td></td>
</tr>
<tr>
<td>Tmp23</td>
<td>Q43</td>
<td>Tmp7, 13, 14, 16, 17, 18, 19, 10, 20, 5, 21, 22, 23</td>
<td>6</td>
<td>(7)(86275350000) – (6)(6000000 + 6000000+ 1500000 + 227597 + 863342789712 + 8000000 + 460135200000 + 200000 + 115033800000 + 10000 + 5751690000 + 150000 + 86275350000) – 8,579,438,053,854</td>
<td></td>
</tr>
<tr>
<td>Tmp27</td>
<td>Tmp28</td>
<td>Q15</td>
<td>Tmp5, 7, 10, 11, 27, 28</td>
<td>3</td>
<td>(5)(16023200000) – (3)(800000+ 200000 + 200000 + 32046400000 + 10000 + 1602320000) – 92,938,190,000</td>
</tr>
</tbody>
</table>
From Table D.12, as weight of Tmp12, Tmp20, Tmp21, Tmp23, Tmp24, Tmp25 and Tmp28 are negative then they are not the indirectly affected node. Tmp26 is the indirectly affected node as its weight is greater than that of Tmp14. For Tmp19, although its weight is less than that of Tmp18, Tmp19 is the existing materialized view then Tmp19 is identified as the indirectly affected node (align with the condition in line 3.2 of affected node identification algorithm in Figure D.1).

Therefore, the affected nodes are:

Directly affected nodes:  Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp9, Tmp11, Tmp14, Tmp15, Tmp16, Tmp17, Tmp18, Tmp27

Indirectly affected nodes:  Tmp19, Tmp26
APPENDIX E

Result of Selection Algorithm

E.1 Two-Phase Optimization Algorithm

The Two-Phase Optimization (2PO) is the combination of Interactive Improvement (II) and Simulated Annealing (SA) (Ioannidis and Kang, 1990:313). The algorithm is presented in Figure E.1

begin
1. Input the MVPP represented by a DAG
2. Use depth first search from root nodes to base relations to search through all of the nodes in the DAG.
3. Produce the sequence of nodes into a binary string.
4. Call Iterative Improvement
5. Call Simulated Annealing algorithm
6. Present set of views to materialized with minimum cost
end;

Figure E.1 The Materialized View Selection with 2PO

The Interactive Improvement algorithm is shown in Figure E.2. In our experiment, the stop criterion is set to 10 local minimum.

begin
    $S_{\text{min}} = S_{\infty}$; {initial solution}
    while not (stopping condition) do {
        $S$ = random state
        while local minimum not reached do {
            $S' = $ random state in neighbor($S$)
            if cost($S'$) < cost($S$) then $S = S'$
        }
        if cost($S$) < cost($S_{\text{min}}$) then $S_{\text{min}} = S$
    }
    return($S_{\text{min}}$)
end;

Figure E.2 The Iterative Improvement (II) Algorithm
The Simulated Annealing algorithm is shown in Figure E.3.

begin
S = S0 ; {initial state}
T = T0 ; {initial value of time limit}
Smin = S;
while not(time limit) do {
    while not(local minimum(S)) do {
        S' = random state in neighbor(S)
        ΔC = cost(S') - cost(S)
        if (ΔC ≤ 0) then S = S'
        if (ΔC > 0) then S = S' with probability e^{ΔC / T}
        if cost(S) < cost(Smin) then Smin = S
    }
    T = reduce(T)
}
Return{Smin}
end;

Figure E.3 The Simulated Annealing (SA) Algorithm

For our experiment, the value of each parameter includes time limit which is set to 90 at the starting point, decrement factor is set to 0.7.

The Result of the Re-Optimized MVPP

Figure E.4 The Cheapest MVPP after Re-Optimized
We map DAG in Figure E.4 to binary string as [Tmp19,0], [Tmp18,0], [Tmp17,0], [Tmp16,0], [Tmp6,0], [Tmp5,0], [Tmp4,0], [Tmp3,0], [Tmp2,0], [Tmp1,0], [Tmp15,0], [Tmp7,0], [Tmp14,0], [Tmp10,0], [Tmp9,0], [Tmp13,0], [Tmp12,0], [Tmp11,0], [Tmp8,0], [Tmp23,0], [Tmp21,0], [Tmp20,0], [Tmp22,0], [Tmp25,0], [Tmp24,0] that is \{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0\}, indicates that all nodes are virtual views. The result generated by II is shown in Table E.1

<table>
<thead>
<tr>
<th>Total Cost of Initial State</th>
<th>Initial State</th>
<th>Total cost of Local Minimum State</th>
<th>Local Minimum State</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 8,427,206,080,471</td>
<td>[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
<td>6,732,979,232,178</td>
<td>[0,1,0,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>2 8,144,566,532,161</td>
<td>[0,1,1,1,0,1,0,1,0,0,0,0,0,0,0,0,1,1,1,1,1,0,0,0,0]</td>
<td>7,188,391,010,923</td>
<td>[1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>3 8,136,058,882,811</td>
<td>[0,0,0,1,0,0,0,1,0,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0]</td>
<td>7,585,650,363,648</td>
<td>[0,0,1,0,1,1,0,0,0,1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>4 8,359,334,708,009</td>
<td>[0,1,0,0,1,0,0,0,0,0,1,0,1,1,0,0,0,0,0,0,0,0,0,0,0]</td>
<td>7,116,835,738,107</td>
<td>[1,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>5 6,708,459,311,979</td>
<td>[0,0,0,0,0,0,0,0,1,0,1,1,0,1,0,0,0,1,1,1,0,1,0,0,0]</td>
<td>6,657,098,483,266</td>
<td>[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>6 7,935,408,327,177</td>
<td>[1,0,0,0,0,0,0,0,0,0,1,0,1,0,0,1,0,0,0,0,0,0,0,0,0]</td>
<td>6,624,915,356,922</td>
<td>[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>7 7,820,946,154,678</td>
<td>[1,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0]</td>
<td>6,978,758,638,497</td>
<td>[0,0,1,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>8 7,688,298,740,389</td>
<td>[0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0]</td>
<td>6,775,762,552,571</td>
<td>[0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,0]</td>
</tr>
<tr>
<td>9 7,499,403,494,665</td>
<td>[0,1,0,0,0,1,1,1,1,1,1,1,1,1,0,1,0,0,1,0,0,1,0,0,0]</td>
<td>7,106,331,542,563</td>
<td>[1,1,1,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,0,0,0]</td>
</tr>
<tr>
<td>10 6,768,696,656,658</td>
<td>[0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,1,1,1,0,0,0,0]</td>
<td>6,678,699,388,215</td>
<td>[0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,1,1,1,0,0,0]</td>
</tr>
</tbody>
</table>

For 10 local minimum, the minimum cost is 6,624,915,356,922 that is the initial state for SA. The state is \{0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,0,1\} represent the nodes as [Tmp19,0], [Tmp18,0], [Tmp17,0], [Tmp16,0], [Tmp6,0], [Tmp5,0], [Tmp4,0], [Tmp3,0], [Tmp2,0], [Tmp1,0], [Tmp15,0], [Tmp7,0], [Tmp14,0], [Tmp10,1], [Tmp9,0], [Tmp13,0] [Tmp12,0], [Tmp11,0], [Tmp8,0], [Tmp23,1], [Tmp21,0], [Tmp20,0], [Tmp22,1], [Tmp25,0], [Tmp24,1].
The value of arguments:  \( T_0 = 90, T = \text{reduce}(T); \) decrement factor is set to 0.7  
\( S_0 = 6,624,915,356,922; \) the summation of query processing cost and materialized view maintenance cost of initial state. The binary string of initial state is \{0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,1,0,0,1,0,1,0,0,1,1\}

**Table E.2** The Result of SA of 2PO for the Re-Optimized MVPP

<table>
<thead>
<tr>
<th>( T )</th>
<th>( S' )</th>
<th>( \Delta C )</th>
<th>( e^{\Delta C / T} )</th>
<th>( S )</th>
<th>The State of ( S' )</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.0</td>
<td>6,624,915,356,947</td>
<td>25</td>
<td>0.7574</td>
<td>6,624,915,356,922</td>
<td>{0,0,1,0,1,0,1,0,0,1,1,0,0,1,1,0,0,1,0,0,0,1}</td>
</tr>
<tr>
<td>89.3</td>
<td>6,624,915,356,752</td>
<td>-170</td>
<td>-</td>
<td>6,624,915,356,752</td>
<td>{0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1}</td>
</tr>
<tr>
<td>88.6</td>
<td>6,624,915,356,747</td>
<td>-5</td>
<td>-</td>
<td>6,624,915,356,747</td>
<td>{0,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,1,0,0,1}</td>
</tr>
<tr>
<td>87.9</td>
<td>6,624,915,536,915</td>
<td>180,168</td>
<td>0.0000</td>
<td>6,624,915,356,747</td>
<td>{1,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,1,1,0,0,1}</td>
</tr>
<tr>
<td>87.2</td>
<td>6,259,209,787,966</td>
<td>-365,705,568,781</td>
<td>-</td>
<td>6,259,209,787,966</td>
<td>{0,0,1,0,1,0,0,0,0,1,0,0,1,1,0,0,1,0,0,1}</td>
</tr>
<tr>
<td>86.5</td>
<td>6,259,209,842,028</td>
<td>54,062</td>
<td>0.0000</td>
<td>6,259,209,787,966</td>
<td>{0,0,1,0,1,0,0,0,0,1,0,0,1,1,0,0,1,1,0,1,0}</td>
</tr>
<tr>
<td>85.8</td>
<td>6,172,139,022,926</td>
<td>-87,070,765,040</td>
<td>-</td>
<td>6,172,139,022,926</td>
<td>{0,0,0,1,0,1,0,0,1,0,0,1,1,0,0,1,1,0,0,1,0}</td>
</tr>
<tr>
<td>85.1</td>
<td>6,124,035,032,590</td>
<td>-48,103,990,336</td>
<td>-</td>
<td>6,124,035,032,590</td>
<td>{0,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0}</td>
</tr>
<tr>
<td>84.4</td>
<td>6,120,827,925,892</td>
<td>-3,207,106,698</td>
<td>-</td>
<td>6,120,827,925,892</td>
<td>{0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0}</td>
</tr>
<tr>
<td>83.7</td>
<td>6,168,924,166,116</td>
<td>48,096,240,224</td>
<td>0.0000</td>
<td>6,120,827,925,892</td>
<td>{0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,1,1,0,1,0}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.5</td>
<td>6,120,829,653,489</td>
<td>1,727,597</td>
<td>0.0000</td>
<td>6,120,827,925,892</td>
<td>{0,0,0,1,0,0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0}</td>
</tr>
<tr>
<td>1.8</td>
<td>6,120,827,925,897</td>
<td>5</td>
<td>0.0000</td>
<td>6,120,827,925,892</td>
<td>{0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0}</td>
</tr>
<tr>
<td>1.1</td>
<td>6,120,828,075,892</td>
<td>150,000</td>
<td>0.0000</td>
<td>6,120,827,925,892</td>
<td>{0,0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0,1,0,0,0}</td>
</tr>
<tr>
<td>0.4</td>
<td>6,120,827,926,004</td>
<td>112</td>
<td>0.0000</td>
<td>6,120,827,925,892</td>
<td>{0,0,0,1,0,0,0,1,1,0,0,0,0,0,1,0,0,0,1,0,0}</td>
</tr>
</tbody>
</table>

So the output generated by SA is \{0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1\}. The total cost of this state is 6,120,827,925,892

\[\text{[Tmp19,0], [Tmp18,0], [Tmp17,0], [Tmp16,0], [Tmp6,1], [Tmp5,0], [Tmp4,0], [Tmp3,0], [Tmp2,0], [Tmp1,0], [Tmp15,1], [Tmp7,0], [Tmp14,0], [Tmp10,0], [Tmp9,0], [Tmp13,0], [Tmp12,0], [Tmp11,1], [Tmp8,0], [Tmp23,0], [Tmp21,1], [Tmp20,0], [Tmp22,0], [Tmp25,0], [Tmp24,1]}\]
E.2 Deterministic Algorithm

We further implement another selection algorithm, Deterministic Algorithm, aiming to validate our methodologies. We use 2PO and Deterministic Algorithm because 2PO has provided the minimal total cost whereas Deterministic has provided the maximal total cost (Phuboon-ob and Auepanwiriyakul, 2007:171; 2009:103). The Deterministic algorithm is proposed by Yang et al. This algorithm is to find a set of materialized views that provide the minimal sum of query processing cost and view maintenance cost. The Deterministic algorithm is described in Figure E.5.

begin
1. $M = \emptyset$ ;
2. Calculate $w(v)$
   $$w(v) = \sum_{q \in q} f_q(q) * C_q^a(v) - \sum_{r \in I} f_u(r) * C_m^r(v)$$
3. create list $LV$ for all the nodes (with positive value of weights) based on the descending order of their weights $w(v)$;
4. pick up the first one $v$ from $LV$;
5. generate $Ov$, $Iv$, and $Sv$
6. calculate $Cs$ of $v$
   $$C_s = \sum_{q \in q} f_q(q) * C_q^a(v) - \sum_{u : q \in M} C_q^a(u) - \sum_{r \in I} f_u(r) * C_m^r(v)$$
7. if $Cs > 0$, then
   insert $v$ into $M$;
   remove $v$ from $LV$;
else
   remove $v$ and all the nodes listed after $v$ from $LV$ who are in the subtree rooted at $v$;
8. repeat step 3 until $LV = \emptyset$ ;
9. for each $v \in M$, if $D(v) \subset M$, then remove $v$ from $M$;
end;

**Figure E.5** Deterministic Algorithm for Materialized View Selection

**Source:** Yang et al., 1997: 141.

$f_q$ denotes the frequency of executing a query

$f_u$ denotes the frequency of updating on base relation.
\(w(v)\) denotes the weight of a node. The first part of this formula indicates the benefit if node \(v\) is materialized, the second part indicates the cost for materialized view maintenance.

\(LV\) is the list of nodes based on descending order of \(w(v)\).

\(S_v\) is the set of leaf nodes and intermediate nodes which are used to produce \(v\).

\(M\) is the set of materialized views.

\(D_v\) is the set of ancestors of \(v\).

\(C_s\) is the cost of a node. The first part of this formula is the saving in access cost if \(v\) is to be materialized. The second part is the additional view maintenance cost for \(v\).

We provide simple explanation for executing the Deterministic Algorithm to the cheapest MVPP described in section 4.3 shown again in Figure E.2 as follows:

![Diagram](image_url)
Initially \( LV = \{\text{Tmp18, Tmp24, Tmp11, Tmp15, Tmp8, Tmp14, Tmp7, Tmp9, Tmp16, Tmp17, Tmp10, Tmp13, Tmp12, Tmp6, Tmp5, Tmp3, Tmp4, and Tmp1}\} \)

\( M = \emptyset \)

Starting with Tmp18:
\[
\begin{align*}
O_{\text{Tmp18}} &= \{Q1, Q5, Q6\}, \\
I_{\text{Tmp18}} &= \{\text{PART, PARTSUPP}\}, \\
S_v &= \{\text{Tmp16, Tmp17}\} \\
C_s &= ( (2+9+5) * (160000000000) - (1000000) ) - 320,002,000,000 \\
&= 2,239,997,000,000 > 0, \text{ so Tmp18 is inserted into } M.
\end{align*}
\]

For Tmp24, Tmp11 and Tmp15, their costs of are greater than zero, so those views are inserted into \( M \) too.

For Tmp8, although its cost is greater than zero, its parent Tmp11 is already in \( M \), then Tmp8 is ignored. By the same reason, Tmp7, Tmp6, Tmp5, Tmp4 and Tmp1 are ignored accordingly. For Tmp14, its parent Tmp15 is already in \( M \) then Tmp14 is ignored. By the same reason, Tmp13, Tmp12, Tmp9, and Tmp10 are ignored accordingly. Tmp16 and Tmp7 also are ignored as its parent Tmp18 already in \( M \). As a result of this algorithm, the selected views are Tmp11, Tmp15, Tmp18 and Tmp24.

Considering query Q1 in the cheapest MVPP in Figure E.2, its query access frequency is 2, before materializing the intermediate node, this query accesses the nodes named Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp18 Tmp21 and result1. The cost of each node is 5, 1, 25, 25, 10000, 50000, 80000, 20000, 160000000000, 1602400000 and 160240 respectively. So, the query processing cost of query Q1 is
\[
\]

After Tmp18 is materialized, the query processing cost for Query Q1 is 2*1,603,420,296 that is 3,206,840,592. It would be beneficial to materialize them, reducing the processing cost from 323,207,240,592 to 3,206,840,592. However, these views have maintenance cost whenever an update of an involved base relation occurs.
The view maintenance cost is $2 \times (800000 + 200000 + 160000000000)$ that is $320,002,000,000$.

Using these four materialized view selected by Deterministic algorithm, we achieve $469,452,759,788$ as query processing cost, and $5,892,777,930,284$ as materialized view maintenance cost. It would be beneficial to materialize them, reducing the total cost from $9,353,211,451,044$ to $6,362,230,690,072$. Table E.1 and Table E.2 show the maintenance cost of each materialized node and the query processing cost of each query of cheapest MVPP respectively.

Table E.3 The Maintenance Cost of the Cheapest MVPP

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Maintenance Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp11</td>
<td>1,426,977,515,570</td>
</tr>
<tr>
<td>Tmp15</td>
<td>1,414,630,939,520</td>
</tr>
<tr>
<td>Tmp18</td>
<td>320,002,000,000</td>
</tr>
<tr>
<td>Tmp24</td>
<td>2,731,167,475,194</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>5,892,777,930,284</strong></td>
</tr>
</tbody>
</table>

Table E.4 The Query Processing of the Cheapest MVPP

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Processing Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query number 1 (Q1)</td>
<td>3,206,840,592</td>
</tr>
<tr>
<td>Query number 2 (Q2)</td>
<td>2,208,984</td>
</tr>
<tr>
<td>Query number 3 (Q3)</td>
<td>2,550,562</td>
</tr>
<tr>
<td>Query number 4 (Q4)</td>
<td>53,213,858,672</td>
</tr>
<tr>
<td>Query number 5 (Q5)</td>
<td>8,181,380</td>
</tr>
<tr>
<td>Query number 6 (Q6)</td>
<td>3,279,656,484</td>
</tr>
<tr>
<td>Query number 7 (Q7)</td>
<td>409,739,463,114</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>469,452,759,788</strong></td>
</tr>
</tbody>
</table>

The Deterministic algorithm is used to select the set of views to be materialized views for the data set in Appendix F. The comparison of the result of Deterministic and 2PO are also provided in Appendix F.
APPENDIX F

Result of Testbed

In this appendix, we provide the experiments to evaluate our approach that are running with 50 queries (Phuboon-ob, 2009: 133) on TPC-H schema. The details of queries and experiment results are described as follows.

F.1 Query Set of Testbed

Query Q1 with the query frequency of 5 produces the minimum discount of items for each type of order’s priority that are ordered in 1994. Its relational algebra tree is shown in Figure F.1.

**Query Q1**

```sql
SELECT O_ORDERPRIORITY, MIN(L_DISCOUNT)
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY O_ORDERPRIORITY;
```

![Figure F.1 Relational Algebra Query Tree of Query Q1](image.png)
Query Q2 with the query frequency of 6 produces the maximum tax for each brand with specific part type and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.2.

**Query Q2**

```sql
SELECT P_BRAND, MAX(L_TAX)
FROM PART, LINEITEM
WHERE P_PARTKEY = L_PARTKEY
AND P_TYPE LIKE '%BRASS%'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

**Figure F.2** Relational Algebra Query Tree of Query Q2
Query Q3 with the query frequency of 4 produces the summation of quantity of item for each supplier with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.3.

**Query Q3**

```
SELECT S_NAME, SUM(L_QUANTITY)
FROM SUPPLIER, LINEITEM
WHERE S_SUPPKEY = L_SUPPKEY
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY S_NAME;
```

**Figure F.3** Relational Algebra Query Tree of Query Q3
Query Q4 with the query frequency of 6 produces the average cost of supply for type of returned item with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.4.

**Query Q4**

```sql
SELECT L_RETURNFLAG, AVG(PS_SUPPLYCOST)
FROM PARTSUPP, LINEITEM
WHERE PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY L_RETURNFLAG;
```

**Figure F.4** Relational Algebra Query Tree of Query Q4
Query Q5 with the query frequency of 9 produces the summation of total price of orders for each market segment of customer that are ordered in 1994. Its relational algebra tree is shown in Figure F.5.

**Query Q5**

```sql
SELECT C_MKTSEGMENT, COUNT(O_TOTALPRICE)
FROM CUSTOMER, ORDERS
WHERE C_CUSTKEY = O_CUSTKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
GROUP BY C_MKTSEGMENT;
```

Figure F.5 Relational Algebra Query Tree of Query Q5
Query Q6 with the query frequency of 7 produces the variance of available quantity for each part type with specific part. Its relational algebra tree is shown in Figure F.6.

**Query Q6**

```
SELECT P_TYPE, VARIANCE(PS_AVAILQTY)
FROM PART, PARTSUPP
WHERE P_PARTKEY = PS_PARTKEY
  AND P_TYPE LIKE '%BRASS%'
GROUP BY P_TYPE;
```

![Figure F.6 Relational Algebra Query Tree of Query Q6](image)
Query Q7 with the query frequency of 5 produces the standard deviation of cost of supply for each supplier and available quantity for supply part more than 2000. Its relational algebra tree is shown in Figure F.7.

**Query Q7**

```
SELECT S_NAME, STDDEV(PS_SUPPLYCOST)
FROM SUPPLIER, PARTSUPP
WHERE S_SUPPKEY = PS_SUPPKEY
AND PS_AVAILQTY < 2000
GROUP BY S_NAME;
```

**Figure F.7** Relational Algebra Query Tree of Query Q7
Query Q8 with the query frequency of 3 produces the standard deviation of discount for each priority type of orders that are ordered in 1994 for customer market segment BUILDING. Its relational algebra tree is shown in Figure F.8.

**Query Q8**

```sql
SELECT O_ORDERPRIORITY, STDDEV(L_DISCOUNT)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
AND C_MKTSEGMENT = 'BUILDING'
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY O_ORDERPRIORITY;
```

**Figure F.8** Relational Algebra Query Tree of Query Q8
Query Q9 with the query frequency of 6 produces the variance of item quantity for each nation of suppliers and items are ordered in 1994. Its relational algebra tree is shown in Figure F.9.

**Query Q9**

```sql
SELECT S_NATIONKEY, VARIANCE(L_QUANTITY)
FROM SUPPLIER, ORDERS, LINEITEM
WHERE S_SUPPKEY = L_SUPPKEY
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY S_NATIONKEY;
```

**Figure F.9** Relational Algebra Query Tree of Query Q9
Query Q10 with the query frequency of 7 produces number of items for each part brand with specific part type and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.10.

**Query Q10**

```sql
SELECT P_BRAND, COUNT(L_TAX)
FROM PART, LINEITEM, PARTSUPP
WHERE
  P_PARTKEY = L_PARTKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND P_TYPE LIKE '%BRASS%'
  AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

**Figure F.10** Relational Algebra Query Tree of Query Q10
Query Q11 with the query frequency of 9 produces the average of cost of supply for each nation of supplier with specific the committed date is before receipt date. Its relational algebra tree is shown in Figure F.11.

**Query Q11**

```
SELECT S_NATIONKEY, AVG(PS_SUPPLYCOST)
FROM SUPPLIER, PARTSUPP, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY S_NATIONKEY;
```

![Figure F.11 Relational Algebra Query Tree of Query Q11](image-url)
Query Q12 with the query frequency of 5 produces the summation of extended price for each type of order status ordered in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.12.

**Query Q12**

```sql
SELECT O_ORDERSTATUS, SUM(L_EXTENDEDPRICE)
FROM ORDERS, PARTSUPP, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY O_ORDERSTATUS;
```

**Figure F.12** Relational Algebra Query Tree of Query Q12
Query Q13 with the query frequency of 7 produces the standard deviation of tax for each type of order’s priority that customer ordered in 1994. Its relational algebra tree is shown in Figure F.13.

**Query Q13**

```sql
SELECT O_ORDERPRIORITY, 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'
GROUP BY O_ORDERPRIORITY;
```

**Figure F.13** Relational Algebra Query Tree of Query Q13
Query Q14 with the query frequency of 7 produces the minimum total price of order for each nation of customer that customer have same nation of supplier and customer ordered in 1994. Its relational algebra tree is shown in Figure F.14.

**Query Q14**

```sql
SELECT C_NATIONKEY, MIN(O_TOTALPRICE)
FROM SUPPLIER, CUSTOMER, ORDERS
WHERE C_NATIONKEY = S_NATIONKEY
    AND C_CUSTKEY = O_CUSTKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
GROUP BY C_NATIONKEY;
```

![Relational Algebra Query Tree of Query Q14](image)

**Figure F.14** Relational Algebra Query Tree of Query Q14
Query Q15 with the query frequency of 5 produces the minimum of available quantity for each type brand for specific part type. Its relational algebra tree is shown in Figure F.15.

**Query Q15**

```
SELECT P_BRAND, MIN(PS_AVAILQTY)
FROM SUPPLIER, PART, PARTSUPP
WHERE S_SUPPKEY = PS_SUPPKEY
AND P_PARTKEY = PS_PARTKEY
AND P_TYPE LIKE '%BRASS%'
GROUP BY P_BRAND;
```

![Figure F.15 Relational Algebra Query Tree of Query Q15](image-url)
Query Q16 with the query frequency of 8 produces the maximum of supply cost for each brand with specific part type and available quantity more than 2000. Its relational algebra tree is shown in Figure F.16.

**Query Q16**

```sql
SELECT P_BRAND, MAX(PS_SUPPLYCOST)
FROM SUPPLIER, PARTSUPP, PART
WHERE S_SUPPKEY = PS_SUPPKEY
   AND PS_PARTKEY = P_PARTKEY
   AND PS_AVAILQTY < 2000
   AND P_TYPE LIKE '%BRASS%'
GROUP BY N_NAME;
```

**Figure F.16** Relational Algebra Query Tree of Query Q16
Query Q17 with the query frequency of 8 produces the summation of discount for each nation of customer specific market segment BUILDING and ordered in 1994. Its relational algebra tree is shown in Figure F.17.

**Query Q17**

```sql
SELECT N_NAME, SUM(L_DISCOUNT)
FROM NATION, CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
AND C_NATIONKEY = N_NATIONKEY
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND C_MKTSEGMENT = 'BUILDING'
GROUP BY N_NAME;
```

*Figure F.17* Relational Algebra Query Tree of Query Q17
Query Q18 with the query frequency of 4 produces the average of item quantity for each nation of supplier that supplier were ordered in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.18.

**Query Q18**

```sql
SELECT N_NAME, AVG(L_QUANTITY)
FROM NATION, SUPPLIER, ORDERS, LINEITEM
WHERE S_SUPPKEY = L_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY N_NAME;
```

**Figure F.18** Relational Algebra Query Tree of Query Q18
Query Q19 with the query frequency of 3 produces the number of item for each customer nation for customers who have nation same as supplier and ordered in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.19.

**Query Q19**

```
SELECT C_NATIONKEY, COUNT(L_QUANTITY)
FROM SUPPLIER, CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
AND O_ORDERKEY = L_ORDERKEY
AND S_SUPPKEY = L_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY C_NATIONKEY;
```
Query Q20 with the query frequency of 7 produces the variance of supply cost for each nation of supplier and the item have the committed date is before receipt date. Its relational algebra tree is shown in Figure F.20.

**Query Q20**

```
SELECT N_NAME, VARIANCE(PS_SUPPLYCOST)
FROM NATION, SUPPLIER, PARTSUPP, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND PS_PARTKEY = L_PARTKEY
    AND PS_SUPPKEY = L_SUPPKEY
    AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY N_NAME;
```

**Figure F.20**  Relational Algebra Query Tree of Query Q20
Query Q21 with the query frequency of 6 produces the standard deviation of extended price for each supplier nation for items that have the committed date is before receipt date and specific part type. Its relational algebra tree is shown in Figure F.21.

**Query Q21**

```
SELECT S_NATIONKEY, STDDEV(L_EXTENDEDPRICE)
FROM SUPPLIER, PARTSUPP, PART, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND PS_PARTKEY = P_PARTKEY
AND P_TYPE LIKE '%BRASS%'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY S_NATIONKEY;
```

**Figure F.21** Relational Algebra Query Tree of Query Q21
Query Q22 with the query frequency of 3 produces the minimum of extended price for each part brand for items that were ordered in 1994 with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.22.

**Query Q22**

```
SELECT P_BRAND, MIN(L_EXTENDEDPRICE)
FROM PART, ORDERS, PARTSUPP, LINEITEM
WHERE P_PARTKEY = PS_PARTKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND O_ORDERDATE >= '1994-01-01'
  AND O_ORDERDATE < '1995-01-01'
  AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

**Figure F.22** Relational Algebra Query Tree of Query Q22
Query Q23 with the query frequency of 3 produces the maximum of extended price for each nation of customer that ordered in 1994 with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.23.

**Query Q23**

```
SELECT C_NATIONKEY, MAX(L_EXTENDEDPRICE)
FROM CUSTOMER, ORDERS, PARTSUPP, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
    AND O_ORDERKEY = L_ORDERKEY
    AND PS_PARTKEY = L_PARTKEY
    AND PS_SUPPKEY = L_SUPPKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
    AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY C_NATIONKEY;
```

**Figure F.23** Relational Algebra Query Tree of Query Q23
Query Q24 with the query frequency of 5 produces the summation of extended price for each nation of supplier that was ordered in 1994 with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.24.

**Query Q24**

```
SELECT    S_NATIONKEY, SUM(L_EXTENDEDPRICE)
FROM      SUPPLIER, ORDERS, PARTSUPP, LINEITEM
WHERE     S_SUPPKEY = PS_SUPPKEY
          AND PS_PARTKEY = L_PARTKEY
          AND PS_SUPPKEY = L_SUPPKEY
          AND O_ORDERKEY = L_ORDERKEY
          AND O_ORDERDATE >= '1994-01-01'
          AND O_ORDERDATE < '1995-01-01'
          AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY  S_NATIONKEY;
```

**Figure F.24** Relational Algebra Query Tree of Query Q24
Query Q25 with the query frequency of 4 produces the average of total price of order for each nation of customer in specific region, ASIA and ordered in 1994. Its relational algebra tree is shown in Figure F.25.

Query Q25

SELECT N_NAME, AVG(O_TOTALPRICE) 
FROM REGION, NATION, CUSTOMER, ORDERS 
WHERE C_NATIONKEY = N_NATIONKEY 
AND N_REGIONKEY = R_REGIONKEY 
AND C_CUSTKEY = O_CUSTKEY 
AND O_ORDERDATE >= '1994-01-01' 
AND O_ORDERDATE < '1995-01-01' 
AND R_NAME = 'ASIA' 
GROUP BY N_NAME;

Figure F.25 Relational Algebra Query Tree of Query Q25
Query Q26 with the query frequency of 6 produces the number of orders for each nation of customer and supplier who have the same nation and order was made in 1994 with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.26.

**Query Q26**

```sql
SELECT N_NAME, COUNT(O_TOTALPRICE)
FROM NATION, SUPPLIER, CUSTOMER, ORDERS
WHERE C_NATIONKEY = S_NATIONKEY
AND S_NATIONKEY = N_NATIONKEY
AND C_CUSTKEY = O_CUSTKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY N_NAME;
```

![Figure F.26 Relational Algebra Query Tree of Query Q26](image)
Query Q27 with the query frequency of 9 produces the variance of available quantity of supply part for each nation of supplier for specific part type and available quantity more than 2000. Its relational algebra tree is shown in Figure F.27.

**Query Q27**

```
SELECT   N_NAME, VARIANCE(PS_AVAILQTY)
FROM      NATION, SUPPLIER, PART, PARTSUPP
WHERE     S_SUPPKEY = PS_SUPPKEY
          AND S_NATIONKEY = N_NATIONKEY
          AND P_PARTKEY = PS_PARTKEY
          AND P_TYPE LIKE '%BRASS%'
          AND PS_AVAILQTY < 2000
GROUP BY  N_NAME;
```

![Relational Algebra Query Tree of Query Q27](image-url)
Query Q28 with the query frequency of 5 produces the standard deviation of supply cost for each nation of supplier in specific region, ASIA and available quantity of supply part more than 2000. Its relational algebra tree is shown in Figure F.28.

Query Q28

SELECT N_NAME, STDDEV(PS_SUPPLYCOST)
FROM REGION, NATION, SUPPLIER, PARTSUPP
WHERE S_SUPPKEY = PS_SUPPKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND PS_AVAILQTY < 2000
    AND R_NAME = 'ASIA'
GROUP BY N_NAME;

Figure F.28 Relational Algebra Query Tree of Query Q28
Query Q29 with the query frequency of 7 produces the summation of item quantity for each nation of supplier in specific region, ASIA with ordered date in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.29.

**Query Q29**

```
SELECT  N_NAME, SUM(L_QUANTITY)
FROM    REGION, NATION, SUPPLIER, LINEITEM
WHERE   S_SUPPKEY = L_SUPPKEY
        AND S_NATIONKEY = N_NATIONKEY
        AND N_REGIONKEY = R_REGIONKEY
        AND O_ORDERKEY = L_ORDERKEY
        AND O_ORDERDATE >= '1994-01-01'
        AND O_ORDERDATE < '1995-01-01'
        AND L_COMMITDATE < L_RECEIPTDATE
        AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

![Figure F.29 Relational Algebra Query Tree of Query Q29](image)
Query Q30 with the query frequency of 4 produces the maximum of discount for each nation of customer in specific region, ASIA with ordered date in 1994. Its relational algebra tree is shown in Figure F.30.

**Query Q30**

```
SELECT N_NAME, MAX(L_DISCOUNT)
FROM REGION, NATION, CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
    AND C_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND O_ORDERKEY = L_ORDERKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
    AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.30** Relational Algebra Query Tree of Query Q30
Query Q31 with the query frequency of 5 produces the summation of item quantity for each nation of supplier in specific region, ASIA with ordered date in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.31.

**Query Q31**

```sql
SELECT N_NAME, SUM(L_QUANTITY)
FROM REGION, NATION, SUPPLIER, ORDERS, LINEITEM
WHERE S_SUPPKEY = L_SUPPKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND N_REGIONKEY = R_REGIONKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND O_ORDERDATE >= '1994-01-01'
  AND O_ORDERDATE < '1995-01-01'
  AND L_COMMITDATE < L_RECEIPTDATE
  AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

![Figure F.31 Relational Algebra Query Tree of Query Q31](image-url)
Query Q32 with the query frequency of 8 produces the average of item quantity for each nation of supplier that have nation same as nation of customer with ordered date in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.32.

**Query Q32**

```sql
SELECT N_NAME, AVG(L_QUANTITY)
FROM NATION, SUPPLIER, CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND S_SUPPKEY = L_SUPPKEY
  AND C_NATIONKEY = S_NATIONKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND O_ORDERDATE >= '1994-01-01'
  AND O_ORDERDATE < '1995-01-01'
  AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY N_NAME;
```

**Figure F.32** Relational Algebra Query Tree of Query Q32
Query Q33 with the query frequency of 6 produces the number of supply parts for each nation of supplier in specific region, ASIA, with the committed date is before receipt date. Its relational algebra tree is shown in Figure F.33.

**Query Q33**

```sql
SELECT N_NAME, COUNT(PS_SUPPLYCOST) 
FROM REGION, NATION, SUPPLIER, PARTSUPP, LINEITEM 
WHERE S_SUPPKEY = PS_SUPPKEY 
AND S_NATIONKEY = N_NATIONKEY 
AND N_REGIONKEY = R_REGIONKEY 
AND PS_PARTKEY = L_PARTKEY 
AND PS_SUPPKEY = L_SUPPKEY 
AND L_COMMITDATE < L_RECEIPTDATE 
AND R_NAME = 'ASIA' 
GROUP BY N_NAME;
```

**Figure F.33** Relational Algebra Query Tree of Query Q33
Query Q34 with the query frequency of 4 produces the summation of quantity of lineitem for each nation of supplier that are ordered in 1994 for specific region, ASIA. Its relational algebra tree is shown in Figure F.34

**Query Q34**

```sql
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;
```

**Figure F.34** Relational Algebra Query Tree of Query Q34
Query Q35 with the query frequency of 6 produces standard deviation of extended price for each brand’s part that customer ordered in 1994, and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.35.

**Query Q35**

```
SELECT    P_BRAND, STDDEV(L_EXTENDEDPRICE)
FROM      CUSTOMER, PART, ORDERS, PARTSUPP, LINEITEM
WHERE     C_CUSTKEY = O_CUSTKEY
       AND O_ORDERKEY = L_ORDERKEY
       AND PS_PARTKEY = L_PARTKEY
       AND PS_SUPPKEY = L_SUPPKEY
       AND P_PARTKEY = PS_PARTKEY
       AND P_TYPE LIKE '%BRASS%'
       AND O_ORDERDATE >= '1994-01-01'
       AND O_ORDERDATE < '1995-01-01'
       AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY  P_BRAND;
```

![Relational Algebra Query Tree of Query Q35](image-url)
Query Q36 with the query frequency of 8 produces the minimum cost supply for each brand ordered in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.36.

**Query Q36**

```sql
SELECT P_BRAND, MIN(PS_SUPPLYCOST)
FROM SUPPLIER, PART, ORDERS, PARTSUPP, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
AND O_ORDERKEY = L_ORDERKEY
AND P_PARTKEY = PS_PARTKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

![Figure F.36 Relational Algebra Query Tree of Query Q36](image-url)
Query Q37 with the query frequency of 3 produces the maximum of extended price for each nation of customer that ordered in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.37.

**Query Q37**

```sql
SELECT C_NATIONKEY, MAX(L_EXTENDEDPRICE)
FROM SUPPLIER, CUSTOMER, ORDERS, PARTSUPP, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
  AND C_NATIONKEY = S_NATIONKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND S_SUPPKEY = PS_SUPPKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND O.OrderDate >= '1994-01-01'
  AND O.OrderDate < '1995-01-01'
  AND L.COMMITDATE < L.RECEIPTDATE
GROUP BY C_NATIONKEY;
```

![Figure F.37](image_url)  
**Figure F.37** Relational Algebra Query Tree of Query Q37
Query Q38 with the query frequency of 5 produces the minimum of cost of supply for nation of suppliers in specific region, ASIA. Its relational algebra tree is shown in Figure F.38.

**Query Q38**

```
SELECT  N_NAME, 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'
GROUP BY N_NAME;
```

**Figure F.38** Relational Algebra Query Tree of Query Q38
Query Q39 with the query frequency of 7 produces the average of total priced for each nation of customer in specific region, ASIA, that ordered in 1994. Its relational algebra tree is shown in Figure F.39.

**Query Q39**

```
SELECT N_NAME, AVG(O_TOTALPRICE)
FROM REGION, NATION, SUPPLIER, CUSTOMER, ORDERS
WHERE C_NATIONKEY = S_NATIONKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND C_CUSTKEY = O_CUSTKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
    AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.39** Relational Algebra Query Tree of Query Q39
Query Q40 with the query frequency of 4 produces the maximum of available quantity of part for each supplier’s nation in specific region, ASIA, specific part type and available quantity is more than 2000. Its relational algebra tree is shown in Figure F.40.

**Query Q40**

```sql
SELECT N_NAME, COUNT(PS_AVAILQTY)
FROM REGION, NATION, SUPPLIER, PART, PARTSUPP
WHERE S_SUPPKEY = PS_SUPPKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND N_REGIONKEY = R_REGIONKEY
  AND P_PARTKEY = PS_PARTKEY
  AND P_TYPE LIKE '%BRASS%'
  AND PS_AVAILQTY < 2000
  AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

*Figure F.40* Relational Algebra Query Tree of Query Q40
Query Q41 with the query frequency of 3 produces the minimum of item quantity for each nation of customers that their nation same nation of supplier in specific region, ASIA, and customer’s orders occurred in 1994 and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.41.

**Query Q41**

```sql
SELECT N_NAME, MIN(L_QUANTITY)
FROM REGION, NATION, SUPPLIER, CUSTOMER, ORDERS, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
    AND O_ORDERKEY = L_ORDERKEY
    AND S_SUPPKEY = L_SUPPKEY
    AND C_NATIONKEY = S_NATIONKEY
    AND S_NATIONKEY = N_NATIONKEY
    AND N_REGIONKEY = R_REGIONKEY
    AND O_ORDERDATE >= '1994-01-01'
    AND O_ORDERDATE < '1995-01-01'
    AND L_COMMITDATE < L_RECEIPTDATE
    AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.41** Relational Algebra Query Tree of Query Q41
Query Q42 with the query frequency of 2 produces the maximum of extended price for each nation of suppliers in specific region, ASIA, specific part type and committed date is before receipt date. Its relational algebra tree is shown in Figure F.42.

**Query Q42**

```
SELECT    N_NAME, MAX(L_EXTENDEDPRICE)
FROM       REGION, NATION, SUPPLIER, PARTSUPP, PART, LINEITEM
WHERE      S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND PS_PARTKEY = P_PARTKEY
AND P_TYPE LIKE '%BRASS%'  
AND L_COMMITDATE < L_RECEIPTDATE
AND R_NAME = 'ASIA'
GROUP BY   N_NAME;
```

**Figure F.42** Relational Algebra Query Tree of Query Q42
Query Q43 with the query frequency of 7 produces the summation of extended price of each brand of part that are ordered in 1994 which committed date is before receipt date, and the nation of customers same that of suppliers. Its relational algebra tree is shown in Figure F.43.

**Query Q43**

```
SELECT P_BRAND, SUM(L_EXTENDEDPRICE)
FROM SUPPLIER, CUSTOMER, PART, ORDERS, PARTSUPP, LINEITEM
WHERE C_CUSTKEY = O_CUSTKEY
  AND C_NATIONKEY = S_NATIONKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND S_SUPPKEY = PS_SUPPKEY
  AND P_PARTKEY = PS_PARTKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND O_ORDERDATE >= '1994-01-01'
  AND O_ORDERDATE < '1995-01-01'
  AND L_COMMITDATE < L_RECEIPTDATE
GROUP BY P_BRAND;
```

**Figure F.43** Relational Algebra Query Tree of Query Q43
Query Q44 with the query frequency of 3 produces the average of account balance for each nation of supplier that same as nation of customer in specific region, ASIA. Its relational algebra tree is shown in Figure F.44

**Query Q44**

```
SELECT N_NAME, 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'
GROUP BY N_NAME;
```

*Figure F.44* Relational Algebra Query Tree of Query Q44
Query Q45 with the query frequency of 8 produces variance of extended price for each nation of supplier which ordered in 1994 for specific part type, brand and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.45

**Query Q45**

```sql
SELECT S_NATIONKEY, Variance (L_EXTENDEDPRICE)
FROM SUPPLIER, PARTSUPP, PART, LINEITEM, ORDERS
WHERE SUPPKEY = PS_SUPPKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND O_ORDERKEY = L_ORDERKEY
AND PS_PARTKEY = P_PARTKEY
AND P_BRAND <> 'BRAND#45'
AND P_TYPE LIKE '%BRASS%'
AND L_COMMITDATE < L_RECEIPTDATE
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
GROUP BY S_NATIONKEY;
```

*Figure F.45* Relational Algebra Query Tree of Query Q45
Query Q46 with the query frequency of 9 produces the maximum of total price for each nation of customers in specific region, ASIA, ordered in 1994. Its relational algebra tree is shown in Figure F.46.

**Query Q46**

```
SELECT N_NAME, 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'
GROUP BY N_NAME;
```

**Figure F.46** Relational Algebra Query Tree of Query Q46 (Q2 of 2nd set)
Query Q47 with the query frequency of 5 produces standard deviation of extended price for each nation of supplier in specific region, ASIA, and orders are occurred in 1994 that committed date is before receipt date. Its relational algebra tree is shown in Figure F.47.

**Query Q47**

```sql
SELECT N_NAME, STDDEV(L_EXTENDEDPRICE)
FROM REGION, NATION, SUPPLIER, ORDERS, PARTSUPP, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
  AND S_NATIONKEY = N_NATIONKEY
  AND N_REGIONKEY = R_REGIONKEY
  AND PS_PARTKEY = L_PARTKEY
  AND PS_SUPPKEY = L_SUPPKEY
  AND O_ORDERKEY = L_ORDERKEY
  AND O_ORDERDATE >= '1994-01-01'
  AND O_ORDERDATE < '1995-01-01'
  AND L_COMMITDATE < L_RECEIPTDATE
  AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.47** Relational Algebra Query Tree of Query Q47
Query Q48 with the query frequency of 6 produces the maximum of extended price for each nation of customers in specific region, ASIA, that ordered in 1994 for specific part type and committed date is before receipt date. Its relational algebra tree is shown in Figure F.48.

**Query Q48**

```
SELECT N_NAME, MAX(L_EXTENDEDPRICE)
FROM REGION, NATION, CUSTOMER, ORDERS, LINEITEM, PARTSUPP, PART
WHERE C_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND PS_PARTKEY = P_PARTKEY
AND O_ORDERKEY = L_ORDERKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
AND P_TYPE LIKE '%BRASS%'
AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.48** Relational Algebra Query Tree of Query Q48
Query Q49 with the query frequency of 4 produces the minimum of cost of supply for each nation of suppliers in specific region, ASIA, that orders occurred in 1994 with committed date is before receipt date. Its relational algebra tree is shown in Figure F.49.

**Query Q49**

```sql
SELECT N_NAME, MIN(PS_SUPPLYCOST)
FROM REGION, NATION, SUPPLIER, PART, ORDERS, PARTSUPP, LINEITEM
WHERE S_SUPPKEY = PS_SUPPKEY
AND S_NATIONKEY = N_NATIONKEY
AND N_REGIONKEY = R_REGIONKEY
AND O_ORDERKEY = L_ORDERKEY
AND P_PARTKEY = PS_PARTKEY
AND PS_PARTKEY = L_PARTKEY
AND PS_SUPPKEY = L_SUPPKEY
AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < '1995-01-01'
AND L_COMMITDATE < L_RECEIPTDATE
AND R_NAME = 'ASIA'
GROUP BY N_NAME;
```

**Figure F.49** Relational Algebra Query Tree of Query Q49
Query Q50 with the query frequency of 5 produces the average of extended price for each brand for customer in region, ASIA, nation of customer and supplier are same and customer ordered in 1994 with specific part type that available quantity is more than 200, and the committed date is before receipt date. Its relational algebra tree is shown in Figure F.50

**Query Q50**

```sql
SELECT P_BRAND, AVG(L_EXTENDEDPRICE) 
FROM REGION, NATION, SUPPLIER, CUSTOMER, PART, ORDERS, PARTSUPP, LINEITEM 
WHERE C_CUSTKEY = O_CUSTKEY 
AND C_NATIONKEY = S_NATIONKEY 
AND S_NATIONKEY = N_NATIONKEY 
AND N_REGIONKEY = R_REGIONKEY 
AND O_ORDERKEY = L_ORDERKEY 
AND S_SUPPKEY = PS_SUPPKEY 
AND P_PARTKEY = PS_PARTKEY 
AND PS_PARTKEY = L_PARTKEY 
AND PS_SUPPKEY = L_SUPPKEY 
AND O_ORDERDATE >= '1994-01-01' 
AND O_ORDERDATE < '1995-01-01' 
AND L_COMMITDATE < L_RECEIPTDATE 
AND P_TYPE LIKE '%BRASS%' 
AND PS_AVAILQTY < 2000 
AND R_NAME = 'ASIA' 
GROUP BY P_BRAND;
```
Figure F.50  Relational Algebra Query Tree of Query Q50
The following sections are the result of query testbed provided in section F.1. We first represent the symbol of the nodes in MVPP as follows.

- represents a materialized view node selected by 2PO algorithm in static phase
- represents a materialized view node selected by Deterministic algorithm in static phase
- represents a new materialized view node selected by 2PO algorithm in dynamic phase

**F.2 The First Query Set**

As our experiment includes two phases, static phase and dynamic phase then the queries are separated into two set for each experiment. The first set, the initial requirement, is for static phase. The second set, new requirements, is for dynamic phase.

The queries for the first query set as follows:

Queries for static phase: {Q7, Q10, Q21, Q27, Q33, Q38 and Q42}
Queries for dynamic phase: {Q6, Q8, Q16, Q30, Q35 and Q50}

**F.2.1 Static Phase with the MVPP Re-Optimization Algorithm**

The order of queries according to their frequency of executing the query multiplied with the query cost is shown in Table F.1. Then, the order of queries of the first MVPP is {Q10, Q33, Q21, Q38, Q7, Q42 and Q27}, and the last order list is {Q27, Q10, Q33, Q21, Q38, Q7 and Q42}. The query processing costs of all MVPPs for query set {Q7, Q10, Q21, Q27, Q33, Q38, and Q42} are shown in Table F.2. The cheapest MVPP is the sixth and seventh MVPP as shown in Figure F.51.

The query processing cost of query of the cheapest MVPP are compared with other MVPPs. The result shows that query processing cost of Q27 of the first to the fifth MVPP is less than that of the cheapest MVPP. Then, Q27 in the cheapest MVPP is possible to be rewritten. The re-optimized MVPP, after Q27 in the cheapest MVPP is rewritten, is shown in Figure F.52.
Table F.1 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Access Frequency($f_q$)</th>
<th>Query Cost</th>
<th>$f_q$ * Query Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q7</td>
<td>5</td>
<td>159,750</td>
<td>798,750</td>
</tr>
<tr>
<td>Q10</td>
<td>7</td>
<td>758,746</td>
<td>5,311,222</td>
</tr>
<tr>
<td>Q21</td>
<td>6</td>
<td>758,746</td>
<td>4,552,476</td>
</tr>
<tr>
<td>Q27</td>
<td>9</td>
<td>31,864</td>
<td>286,776</td>
</tr>
<tr>
<td>Q33</td>
<td>6</td>
<td>759,474</td>
<td>4,556,844</td>
</tr>
<tr>
<td>Q38</td>
<td>5</td>
<td>160240</td>
<td>801,200</td>
</tr>
<tr>
<td>Q42</td>
<td>5</td>
<td>152424</td>
<td>762,120</td>
</tr>
</tbody>
</table>

Table F.2 The Query Processing Cost of the MVPPs for the First Query Set

<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP</th>
<th>4th MVPP</th>
<th>5th MVPP</th>
<th>6th MVPP (cheapest)</th>
<th>7th MVPP (cheapest)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q7</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
<td>7,996,348,750</td>
</tr>
<tr>
<td>Q27</td>
<td>71,996,435,901*</td>
<td>71,996,435,901*</td>
<td>71,996,435,901*</td>
<td>71,996,435,901*</td>
<td>71,996,435,901*</td>
<td>129,618,935,901*</td>
<td>291,303,524,277</td>
</tr>
<tr>
<td>Q38</td>
<td>168,258,100,480,277</td>
<td>168,258,100,480,277</td>
<td>168,258,100,480,277</td>
<td>168,258,100,480,277</td>
<td>840,109,850,480,277</td>
<td>168,258,100,480,277</td>
<td>168,258,100,480,277</td>
</tr>
<tr>
<td>Total</td>
<td>14,128,948,480,277</td>
<td>14,128,948,480,277</td>
<td>14,128,948,480,277</td>
<td>14,128,948,480,277</td>
<td>15,896,610,561,072</td>
<td>12,799,595,116,681</td>
<td>12,799,595,116,681</td>
</tr>
</tbody>
</table>

Note: * query processing cost of nth MVPP less than the cheapest MVPP
Figure F.51 The Cheapest MVPP of the First Query Set in Static Phase

Figure F.52 The Re-Optimized MVPP of the First Query Set in Static Phase
The total query processing cost of the cheapest MVPP is 12,799,595,116,681. After Q27 is rewritten, the total query processing cost is reduced to 12,581,422,028,305.

We further evaluate the MVPP re-optimization algorithm by selecting the set of view to be materialized. Figure F.53 and F.54 show the cheapest MVPP and the re-optimized MVPP after Deterministic and 2PO are applied, respectively. The query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized-views and selection materialized view by Deterministic and 2PO of the cheapest MVPP and the re-optimized MVPP are shown in Table F.3 and Table F.4 respectively.

![Diagram](image-url)

**Figure F.53 (a) The Cheapest MVPP of the First Query Set Selected by Deterministic**
Figure F.53 (b) The Cheapest MVPP of the First Query Set Selected by 2PO

Figure F.54 (a) The Re-Optimized MVPP of the First Query Set by Deterministic
Figure F.54 (b) The Re-Optimized MVPP of the First Query Set by 2PO

Table F.3 The Query Processing Cost, Maintenance Cost and Total Cost of the Cheapest MVPP of the First Query Set

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>12,799,595,116,681</td>
<td>0</td>
<td>12,799,595,116,681</td>
</tr>
<tr>
<td>All-materialized view</td>
<td>16,612,116</td>
<td>8,792,272,746,660</td>
<td>8,792,289,358,776</td>
</tr>
<tr>
<td>2PO</td>
<td>292,686,142,972</td>
<td>5,202,416,789,776</td>
<td>5,495,102,932,748</td>
</tr>
<tr>
<td>Deterministic</td>
<td>581,114,524,945</td>
<td>5,138,321,589,776</td>
<td>5,719,436,114,721</td>
</tr>
</tbody>
</table>
Table F.4 The Query Processing Cost, Maintenance Cost and Total Cost of the Re-Optimized MVPP of the First Query Set

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>12,581,422,028,305</td>
<td>0</td>
<td>12,581,422,028,305</td>
</tr>
<tr>
<td>All-materialized view</td>
<td>16,612,116</td>
<td>8,694,762,487,060</td>
<td>8,694,779,099,176</td>
</tr>
<tr>
<td>2PO</td>
<td>289,800,706,225</td>
<td>5,138,333,589,776</td>
<td>5,428,134,296,001</td>
</tr>
<tr>
<td>Deterministic</td>
<td>297,811,870,883</td>
<td>5,135,123,369,776</td>
<td>5,432,935,240,659</td>
</tr>
</tbody>
</table>

Conclusion result of the MVPP re-optimization algorithm:
The result shows the total cost as follows:

- All-virtual view: reduced from 12,799,595,116,681 to 12,581,422,028,305
- All-materialized view: reduced from 8,792,289,358,776 to 8,694,779,099,176
- Selected views to be materialized:
  - 2PO: reduced from 5,495,102,932,748 to 5,428,134,296,001
  - Deterministic reduced from: 5,719,436,114,721 to 5,432,935,240,659

F.2.2 Dynamic Phase Result
The queries \{Q6, Q8, Q16, Q30, Q35 and Q50\} are merged into the existing re-optimized MVPP, Figure F.54. The result of the dynamic MVPP is shown in Figure F.55.
Figure F.55 The Optimal Dynamic MVPP of the First Query Set

After the dynamic MVPPs are generated and the optimal one is selected, the affected node identification algorithm is applied to identify the affected nodes. The existing nodes used to construct the new queries are shown in Table F.5. Their weights are shown in Table F.6.

Table F.5 The Existing Nodes Used to Construct New Queries

<table>
<thead>
<tr>
<th>New Queries</th>
<th>Existing Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q6</td>
<td>Tmp7, 9, 15, 16</td>
</tr>
<tr>
<td>Q8</td>
<td>Tmp13</td>
</tr>
<tr>
<td>Q16</td>
<td>Tmp5, 7, 11, 12, 15, 19</td>
</tr>
<tr>
<td>Q30</td>
<td>Tmp1, 2, 3, 4, 13</td>
</tr>
<tr>
<td>Q35</td>
<td>Tmp7, 9, 13, 14, 15, 16, 17</td>
</tr>
<tr>
<td>Q50</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 22, 23</td>
</tr>
</tbody>
</table>
Table F.6 The Weight of the Existing Node for Constructing New Queries

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight ( w (v) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>60</td>
</tr>
<tr>
<td>Tmp2</td>
<td>7</td>
</tr>
<tr>
<td>Tmp3</td>
<td>525</td>
</tr>
<tr>
<td>Tmp4</td>
<td>213</td>
</tr>
<tr>
<td>Tmp5</td>
<td>320,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>469,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>31,200,000</td>
</tr>
<tr>
<td>Tmp8</td>
<td>14,418,159,776</td>
</tr>
<tr>
<td>Tmp9</td>
<td>5,600,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>56,078,699,720</td>
</tr>
<tr>
<td>Tmp11</td>
<td>9,600,000</td>
</tr>
<tr>
<td>Tmp12</td>
<td>19,166,780,000</td>
</tr>
<tr>
<td>Tmp13</td>
<td>108,000,000</td>
</tr>
<tr>
<td>Tmp14</td>
<td>102,000,000</td>
</tr>
<tr>
<td>Tmp15</td>
<td>4,000,000</td>
</tr>
<tr>
<td>Tmp16</td>
<td>576,832,800,000</td>
</tr>
<tr>
<td>Tmp17</td>
<td>5,981,895,246,720</td>
</tr>
<tr>
<td>Tmp19</td>
<td>33,597,063,000</td>
</tr>
<tr>
<td>Tmp22</td>
<td>-168,257,781,000</td>
</tr>
<tr>
<td>Tmp23</td>
<td>-620,262,645,936</td>
</tr>
</tbody>
</table>

The intermediate nodes, which are the conjunctively joined nodes with positive weight, project operation that is not the ancestor of base relation and select operation, are inserted into the list of directly affected node. Therefore, the directly affected are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp10, Tmp11, Tmp12, Tmp14, Tmp15, Tmp16, Tmp17 and Tmp19\}.

Next, we identify the indirectly affected nodes. The directly affected nodes that their ancestors are not the directly affected node are Tmp8 and Tmp10. The weights of the ancestor node of those nodes are shown in Table F.7.
Table F.7 The Weight of Ancestor Node of Directly Affected Node of New Queries

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp8</td>
<td>Tmp21</td>
<td>599,761,450,760</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Tmp22</td>
<td>-168,257,781,000</td>
</tr>
<tr>
<td></td>
<td>Tmp23</td>
<td>-620,262,645,936</td>
</tr>
</tbody>
</table>

Tmp21 is identified as the indirectly affected node as its weight greater than that of Tmp8. Tmp22 and Tmp23 are not the indirectly affected node as their weight are negative.

The result of affected nodes show as follows.

Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp10, Tmp11, Tmp12, Tmp14, Tmp15, Tmp16, Tmp17 and Tmp19

Indirectly affected nodes: Tmp21

Therefore, the number of nodes to be the member of set of views to be selected by 2PO is 28 nodes, 14 existing nodes and 14 new created nodes.

Figure F.56 The Materialized View Selected by 2PO for the Dynamic MVPP of the First Query Set
After the affected nodes are identified, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that the existing materialized view \{Tmp10, Tmp12, Tmp14, Tmp17, Tmp19 and Tmp21\} are still materialized, the virtual view \{Tmp16\} is materialized and the new nodes \{Tmp25, Tmp32, Tmp36\} are materialized. Figure F.56 shows the result after the materialized are selected by 2PO.

We rerun static approach for all queries \{Q7, Q10, Q21, Q27, Q33, Q38 and Q42\} and \{Q6, Q8, Q16, Q30, Q35 and Q50\}. The result of the static approach for all queries is shown in Figure F.57. After our MVPP re-optimization algorithm is applied to the cheapest MVPP, the MVPP structure of static approach in Figure F.57 provides same structure as that of dynamic MVPP in Figure F.56.

![Figure F.57](image-url)  
**Figure F.57** The Re-Optimized MVPP by Static Approach for All Queries of the First Query Set

The query processing cost, materialized view maintenance cost and total cost of the static and dynamic approach on the set of materialized views selected by 2PO are shown in Table F.8.
Table F.8  The Comparison of the Result from the Static Approach and the Dynamic Approach for the First Query Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>37</td>
<td>4,709,318,228,610</td>
<td>5,240,142,056,783</td>
<td>9,949,460,285,393</td>
</tr>
<tr>
<td>Dynamic</td>
<td>28</td>
<td>4,709,318,228,610</td>
<td>5,240,142,056,783</td>
<td>9,949,460,285,393</td>
</tr>
</tbody>
</table>

Conclusion result of the dynamic phase:

In Figure F.57, we rerun static approach for all queries; the search space contains 37 intermediated nodes for static materialized view selection whereas our dynamic approach for additional queries, the set of intermediated nodes to be selected is 28 nodes, 14 existing nodes and 14 new nodes. The result in Table F.8 shows that even though all costs of static are same as costs of dynamic approach; the number of nodes to be selected of dynamic is less than static approach.
F.3 The Second Query Set

Queries for static phase: \{Q4, Q12, Q23, Q33, Q36, Q40 and Q41\}
Queries for dynamic phase: \{Q22, Q15, Q21, Q28, Q31 and Q50\}

F.3.1 Static Phase with the MVPP Re-Optimization Algorithm

The order of queries according to their frequency of executing the query multiplied with the query cost is shown in Table F.9. Then, the order of queries of the first MVPP is \{Q4, Q36, Q33, Q12, Q23, Q40 and Q41\} and the last order is \{Q41, Q4, Q36, Q33, Q12, Q23 and Q40\}. The query processing costs of all MVPPs for the query set \{Q4, Q12, Q23, Q33, Q36, Q40 and Q41\} are shown in Table F.10. The cheapest MVPP is the second and the third MVPP as shown in Figure F.58.

Table F.9 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Access Frequency((f_q))</th>
<th>Query Cost</th>
<th>(f_q \times \text{Query Cost})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4</td>
<td>6</td>
<td>3,793,296</td>
<td>22,759,776</td>
</tr>
<tr>
<td>Q12</td>
<td>5</td>
<td>575,169</td>
<td>2,875,845</td>
</tr>
<tr>
<td>Q23</td>
<td>3</td>
<td>575,169</td>
<td>1,725,507</td>
</tr>
<tr>
<td>Q33</td>
<td>6</td>
<td>759,474</td>
<td>4,556,844</td>
</tr>
<tr>
<td>Q36</td>
<td>8</td>
<td>575,169</td>
<td>4,601,352</td>
</tr>
<tr>
<td>Q40</td>
<td>4</td>
<td>6,492</td>
<td>25,968</td>
</tr>
<tr>
<td>Q41</td>
<td>3</td>
<td>4,558</td>
<td>13,674</td>
</tr>
</tbody>
</table>
Table F.10  The Query Processing Cost of The MVPPs for the Second Query Set

<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP (cheapest)</th>
<th>4th MVPP</th>
<th>5th MVPP</th>
<th>6th MVPP</th>
<th>7th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q23</td>
<td>11,952,803, 957,322</td>
<td>4,229,299, 428,870</td>
<td>4,229,299, 428,870</td>
<td>4,229,299, 428,870</td>
<td>11,952,803, 957,322</td>
<td>11,952,803, 957,322</td>
<td>11,952,803, 957,322</td>
</tr>
<tr>
<td>Q40</td>
<td>6,443,239, 04*</td>
<td>11,570,291, 04*</td>
<td>11,570,291, 04*</td>
<td>6,443,239, 04*</td>
<td>6,443,239, 04*</td>
<td>6,443,239, 04*</td>
<td>6,443,239, 04*</td>
</tr>
<tr>
<td>Q41</td>
<td>2,645,437, 66,178</td>
<td>2,645,437, 166,178</td>
<td>2,645,437, 166,178</td>
<td>2,645,437, 166,178</td>
<td>2,645,437, 166,178</td>
<td>2,645,437, 166,178</td>
<td>2,645,437, 166,178</td>
</tr>
<tr>
<td>Total</td>
<td>102,706,25</td>
<td>6,505,602</td>
<td>232,366</td>
<td>232,366</td>
<td>905,602</td>
<td>905,602</td>
<td>200,307</td>
</tr>
</tbody>
</table>

Note: * query processing cost of nth MVPP less than the cheapest MVPP

Figure F.58  The Cheapest MVPP of the Second Query Set in Static Phase
The query processing cost of query of the cheapest MVPP are compared with other MVPPs. The result shows that Q40 and Q41 of the first MVPP is less than the cheapest. So both queries are possible to be rewritten.

For Q40, there is only one possible plan in the cheapest MVPP for Q40 that is \{Tmp8\} \Join PARTSUPP. Tmp8 is sharable conjunctively join with Q33. So Q40 still use the same plan in the cheapest MVPP.

For Q41, there is only one possible plan for Q41 as sharable subexpression already constructed in the cheapest MVPP. That are \{REGION \Join NATION \Join SUPPLIER\}, Tmp6, and \{LINEITEM \Join ORDERS\}, Tmp14. So Q41 still use the same plan in the cheapest MVPP.

Therefore the cheapest MVPP provides the minimal MVPP.

Next, selection algorithms are applied to the optimal MVPP in Figure F.58 to select the set of views to be materialized to be the initial search space for dynamic phase. The result of Deterministic is shown in Figure F.59 (a). The result of 2PO is that \{Tmp6, Tmp10, Tmp11, Tmp14, Tmp15, Tmp19\} are the materialized views as shown in Figure F.59 (b). We show the query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized-views and selection materialized view by 2PO algorithm of the cheapest MVPP in Table F.11.

![Figure F.59](a) The Optimal MVPP of the Second Query Set by Deterministic
Figure F.59 (b) The Optimal MVPP of the Second Query Set by 2PO

Table F.11 The Query Processing Cost, Maintenance Cost and Total Cost of the Static Phase of the Second Query Set

<table>
<thead>
<tr>
<th></th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>34,783,985,485,998</td>
<td>0</td>
<td>34,783,985,485,998</td>
</tr>
<tr>
<td>All-materialized view</td>
<td>21,385,782</td>
<td>31,252,800,172,180</td>
<td>31,252,821,557,962</td>
</tr>
<tr>
<td>2PO</td>
<td>1,292,099,197,067</td>
<td>14,813,755,422,193</td>
<td>16,105,854,619,260</td>
</tr>
<tr>
<td>Deterministic</td>
<td>1,292,099,603,438</td>
<td>14,813,755,242,025</td>
<td>16,105,854,845,463</td>
</tr>
</tbody>
</table>

Conclusion result of the MVPP re-optimization algorithm:

For the second query set, although Q40 and Q41 are possibly to be rewritten, after the MVPP re-optimization algorithm is applied for those queries they are forced to use the sharable subexpression that available in the MVPP rather than created new execution plan equal to the plan in the sixth and the seventh MVPP. Then, query processing plan of Q40, Q41 are the optimal plan for the cheapest MVPP.
F.3.2 Dynamic Phase Result

The queries of the dynamic phase, \{Q22, Q15, Q21, Q28, Q31 and Q50\}, are merged into the existing re-optimized MVPP, Figure F.59 (b). The result of the dynamic MVPP is shown in Figure F.60.

![Diagram showing the optimal dynamic MVPP of the second query set](image)

**Figure F.60** The Optimal Dynamic MVPP of the Second Query Set

After the dynamic MVPPs are generated and the optimal one is selected, the affected node identification algorithm is applied to identify the affected nodes. The existing nodes used to construct new queries are shown in Table F.12. Their weights are shown in Table F.13.

**Table F.12** The Existing Nodes Used to Construct New Queries

<table>
<thead>
<tr>
<th>New Queries</th>
<th>Existing Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q15</td>
<td>Tmp5, 7, 9, 10, 18, 19</td>
</tr>
<tr>
<td>Q21</td>
<td>Tmp5, 7, 9, 10, 18, 19</td>
</tr>
<tr>
<td>Q22</td>
<td>Tmp7, 9, 10, 12, 13, 14, 15, 18, 25</td>
</tr>
<tr>
<td>Q28</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 20</td>
</tr>
<tr>
<td>Q31</td>
<td>Tmp1, 2, 3, 4, 5, 6, 9, 10, 12, 13, 14, 22</td>
</tr>
<tr>
<td>Q50</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10,12,13,14,16, 18,19,21</td>
</tr>
</tbody>
</table>
### Table F.13 The Weight of the Existing Node for Constructing All New Queries

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight ((w(v)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>135</td>
</tr>
<tr>
<td>Tmp2</td>
<td>22</td>
</tr>
<tr>
<td>Tmp3</td>
<td>675</td>
</tr>
<tr>
<td>Tmp4</td>
<td>588</td>
</tr>
<tr>
<td>Tmp5</td>
<td>400,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>1,219,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>40,800,000</td>
</tr>
<tr>
<td>Tmp8</td>
<td>25,634,959,776</td>
</tr>
<tr>
<td>Tmp9</td>
<td>270,000,000</td>
</tr>
<tr>
<td>Tmp10</td>
<td>164,698,320</td>
</tr>
<tr>
<td>Tmp12</td>
<td>46,500,000</td>
</tr>
<tr>
<td>Tmp13</td>
<td>5,555,507</td>
</tr>
<tr>
<td>Tmp14</td>
<td>25,900,260,649,574</td>
</tr>
<tr>
<td>Tmp15</td>
<td>4,772,097,868,185</td>
</tr>
<tr>
<td>Tmp16</td>
<td>1,500,000</td>
</tr>
<tr>
<td>Tmp18</td>
<td>6,000,000</td>
</tr>
<tr>
<td>Tmp19</td>
<td>3,600,000</td>
</tr>
<tr>
<td>Tmp20</td>
<td>-6,412,239,024</td>
</tr>
<tr>
<td>Tmp21</td>
<td>-2,864,117,576</td>
</tr>
<tr>
<td>Tmp22</td>
<td>-4,313,315,662,784</td>
</tr>
<tr>
<td>Tmp25</td>
<td>-4,833,826,082,420</td>
</tr>
</tbody>
</table>

The intermediate nodes, which are the conjunctively joined nodes with positive weight, project operation that is not the ancestor of base relation and select operation, are inserted into the list of directly affected node. Therefore, the directly affected are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp10, Tmp12, Tmp13, Tmp14, Tmp15, Tmp19 and Tmp20\}.
Next, we identify the indirectly affected nodes. The directly affected nodes that their ancestors are not the directly affected node are Tmp6, Tmp8, Tmp10, Tmp15 and Tmp19. The weights of the ancestor node of those nodes are shown in Table F.14.

In table F.14, Tmp11 is the indirectly affected node as its weight greater than Tmp8. Tmp20 is select operation node. The other nodes are not directly affected node as they are conjunctively join node with negative weight.

Table F.14 The Weight of Ancestor Node of Directly Affected Node of New Queries

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp6</td>
<td>Tmp22</td>
<td>-4,313,315,662,784</td>
</tr>
<tr>
<td></td>
<td>Tmp23</td>
<td>-40,375,843,660</td>
</tr>
<tr>
<td>Tmp8</td>
<td>Tmp11</td>
<td>599,772,484,280</td>
</tr>
<tr>
<td>Tmp10</td>
<td>Tmp24</td>
<td>-21,186,592</td>
</tr>
<tr>
<td>Tmp15</td>
<td>Tmp25</td>
<td>-4,833,826,082,420</td>
</tr>
<tr>
<td>Tmp19</td>
<td>Tmp21</td>
<td>-2,864,117,576</td>
</tr>
</tbody>
</table>

The result of affected nodes show as follows.

Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp10, Tmp12, Tmp13, Tmp14, Tmp15, Tmp19 and Tmp20

Indirectly affected nodes: Tmp11

Therefore, the number of nodes to be the member of set of views to be materialized is 18 nodes, 13 existing nodes and 5 new created nodes.

After the affected nodes are identified, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that the existing materialized views {Tmp6, Tmp11, Tmp10, Tmp14, Tmp15 and Tmp19} are still materialized, the existing virtual view {Tmp8} is materialized and the new node {Tmp27} is materialized. Figure F.61 shows the result after 2PO select the set of views to be materialized.

We rerun static approach for all queries {Q4, Q12, Q23, Q33, Q36, Q40 and Q41} and {Q15, Q21, Q22, Q28, Q31 and Q50}. The result of the static approach for
all queries is shown in Figure F.62. After our MVPP re-optimization algorithm is applied to the cheapest MVPP, the MVPP structure of static approach in Figure F.62 provides same structure as that of dynamic MVPP in Figure F.61.

The query processing cost, materialized view maintenance cost and total cost of the static and dynamic approach are computed as shown in Table F.15.

Figure F.61 The Materialized Views Selected by 2PO for the Dynamic MVPP of the Second Query Set
Figure F.62  The Re-Optimized MVPP by Static Approach for All Queries of the Second Query Set

Table F.15  The Comparison of the Result from the Static Approach and the Dynamic Approach of the Second Query Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>31</td>
<td>5,329,761,453,158</td>
<td>14,884,283,889,969</td>
<td>20,214,045,343,127</td>
</tr>
<tr>
<td>Dynamic</td>
<td>18</td>
<td>5,329,761,453,158</td>
<td>14,884,283,889,969</td>
<td>20,214,045,343,127</td>
</tr>
</tbody>
</table>

Conclusion result of the dynamic phase:

In Figure F.62, we rerun static approach for all queries; the search space contains 31 intermediated nodes for static materialized view selection whereas our dynamic approach for additional queries, the set of intermediated nodes to be selected is 17 nodes, 12 existing nodes and 5 new created nodes. The result in Table F.15 shows that even though all costs of dynamic approach are same as cost of static approach, the number of nodes to be selected of dynamic less than static approach.
F.4 The Third Query Set

Queries for static phase: \{Q1, Q8, Q9, Q19, Q30, Q31 and Q41\}
Queries for dynamic phase: \{Q2, Q10, Q25, Q29, Q33 and Q42\}

F.4.1 Static Phase with the MVPP Re-Optimization Algorithm

The order of queries according to their frequency of executing the query multiplied with the query cost is shown in Table F.16. Then, the order of queries of the first MVPP is \{Q9, Q1, Q30, Q31, Q8, Q19 and Q41\} and the last order is \{Q41, Q9, Q1, Q30, Q31, Q8, and Q19\}. The query processing costs of all MVPPs of query set \{Q1, Q8, Q9, Q19, Q30, Q31 and Q41\} are shown in Table F.17. The cheapest MVPP is the third MVPP as shown in Figure F.63.

Table F.16 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost

<table>
<thead>
<tr>
<th>Query</th>
<th>Query Access Frequency ($f_q$)</th>
<th>Query Cost</th>
<th>$f_q \times$ Query Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>5</td>
<td>910,519</td>
<td>4,552,595</td>
</tr>
<tr>
<td>Q8</td>
<td>3</td>
<td>183,273</td>
<td>549,819</td>
</tr>
<tr>
<td>Q9</td>
<td>6</td>
<td>910,519</td>
<td>5,463,114</td>
</tr>
<tr>
<td>Q19</td>
<td>3</td>
<td>22,278</td>
<td>66,834</td>
</tr>
<tr>
<td>Q30</td>
<td>4</td>
<td>184,082</td>
<td>736,328</td>
</tr>
<tr>
<td>Q31</td>
<td>5</td>
<td>115,372</td>
<td>576,860</td>
</tr>
<tr>
<td>Q41</td>
<td>3</td>
<td>4,558</td>
<td>13,674</td>
</tr>
</tbody>
</table>
Table F.17  The Query Processing Cost of the Third Query Set

<table>
<thead>
<tr>
<th>Query</th>
<th>1st MVPP</th>
<th>2nd MVPP</th>
<th>3rd MVPP (cheapest)</th>
<th>4th MVPP</th>
<th>5th MVPP</th>
<th>6th MVPP</th>
<th>7th MVPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>6,827,953</td>
<td>6,827,953</td>
<td>6,827,953</td>
<td>6,827,953</td>
<td>6,827,953</td>
<td>6,827,953</td>
<td>6,827,953</td>
</tr>
<tr>
<td>Q8</td>
<td>4,506,506</td>
<td>4,506,506</td>
<td>4,199,188</td>
<td>4,199,188</td>
<td>4,199,188</td>
<td>4,506,506</td>
<td>4,506,506</td>
</tr>
<tr>
<td>Q9</td>
<td>8,248,175</td>
<td>8,248,175</td>
<td>8,248,175</td>
<td>8,248,175</td>
<td>8,248,175</td>
<td>8,248,175</td>
<td>8,248,175</td>
</tr>
<tr>
<td>Q30</td>
<td>5,572,292</td>
<td>5,572,292</td>
<td>1,131,705</td>
<td>6,008,675</td>
<td>6,008,675</td>
<td>5,572,292</td>
<td>5,572,292</td>
</tr>
<tr>
<td>Q31</td>
<td>6,873,494</td>
<td>6,833,714</td>
<td>902,329,39</td>
<td>902,329,39</td>
<td>6,873,494</td>
<td>902,329,39</td>
<td>4,455</td>
</tr>
<tr>
<td>Q41</td>
<td>4,176,014</td>
<td>4,152,146</td>
<td>593,315,15</td>
<td>593,315,15</td>
<td>4,214,464</td>
<td>4,382,914</td>
<td>593,315,15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>40,577,28</strong></td>
<td><strong>40,664,548</strong></td>
<td><strong>26,119,113</strong></td>
<td><strong>30,996,085</strong></td>
<td><strong>40,590,381</strong></td>
<td><strong>40,794,249</strong></td>
<td><strong>31,023,424</strong></td>
</tr>
<tr>
<td></td>
<td><strong>8,445,643</strong></td>
<td><strong>326,939</strong></td>
<td><strong>241,198</strong></td>
<td><strong>803,827</strong></td>
<td><strong>871,946</strong></td>
<td><strong>309,778</strong></td>
<td><strong>841,659</strong></td>
</tr>
</tbody>
</table>

Figure F.63  The cheapest MVPP of the Third Query Set of the Static Phase
Figure F.64 (a) The optimal MVPP of the Third Query by Deterministic

Figure F.64 (b) The optimal MVPP of the Third Query by 2PO
The query processing cost of query of the cheapest MVPP are compared with other MVPPs. The result is that query processing of all queries of the cheapest MVPP is less than other MVPPs. Therefore the cheapest MVPP is the minimal MVPP.

Next, selection algorithm are applied to the optimal MVPP in Figure F.63 to select the set of views to be materialized to be the initial search space for dynamic phase. The result of the Deterministic is shown in Figure F.64 (a). The result of 2PO is that \{Tmp6, Tmp16, Tmp23, Tmp18\} are the materialized views as shown in Figure F.64 (b). We show the query processing cost, materialized view maintenance cost and total cost of all-virtual-views, all-materialized-views and selection materialized view by 2PO algorithm of the cheapest MVPP in Table F.18.

<table>
<thead>
<tr>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>All-virtual view</td>
<td>26,119,113,241,198</td>
<td>0</td>
</tr>
<tr>
<td>All-materialized view</td>
<td>11,960,724</td>
<td>16,649,310,570,792</td>
</tr>
<tr>
<td>2PO</td>
<td>306,035,672,471</td>
<td>7,832,700,983,345</td>
</tr>
<tr>
<td>Deterministic</td>
<td>306,035,672,471</td>
<td>7,832,699,255,748</td>
</tr>
</tbody>
</table>

**Conclusion result of the MVPP re-optimization algorithm:**

Accordingly to Table F.18, after the MVPP re-optimization algorithm is applied, the query processing of all queries of the cheapest MVPP is less than that of other MVPPs, so the cheapest MVPP is the optimal MVPP.

**F.4.2 Dynamic Phase Result**

The queries of the dynamic phase, \{Q2, Q10, Q25, Q29, Q33 and Q42\}, are merged into the existing re-optimized MVPP, Figure F.64. The result of the dynamic MVPP is shown in Figure F.65.
Figure F.65 The Optimal Dynamic MVPP of the Third Query Set

After the dynamic MVPPs are generated and the optimal one is selected, the affected node identification algorithm is applied to identify the affected nodes. The existing nodes are used to construct the new queries are shown in Table F.19. Their weights are shown in Table F.20.

Table F.19 The Existing Nodes Used to Construct New Queries

<table>
<thead>
<tr>
<th>New Queries</th>
<th>Existing Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q2</td>
<td>Tmp1, 14</td>
</tr>
<tr>
<td>Q10</td>
<td>Tmp10, 14</td>
</tr>
<tr>
<td>Q25</td>
<td>Tmp1, 2, 3, 4, 5, 6, 7, 8, 9</td>
</tr>
<tr>
<td>Q29</td>
<td>Tmp1, 2, 3, 4, 10, 12, 13, 14, 15</td>
</tr>
<tr>
<td>Q33</td>
<td>Tmp1, 2, 3, 4, 10, 12, 13, 14, 15</td>
</tr>
<tr>
<td>Q42</td>
<td>Tmp1, 2, 3, 4, 10, 12, 13, 14, 15</td>
</tr>
</tbody>
</table>
Table F.20  The Weight of the Existing Node for Constructing All New Queries

<table>
<thead>
<tr>
<th>Existing Node</th>
<th>Weight (w(v))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp1</td>
<td>55</td>
</tr>
<tr>
<td>Tmp2</td>
<td>6</td>
</tr>
<tr>
<td>Tmp3</td>
<td>275</td>
</tr>
<tr>
<td>Tmp4</td>
<td>188</td>
</tr>
<tr>
<td>Tmp5</td>
<td>1,050,000</td>
</tr>
<tr>
<td>Tmp6</td>
<td>3,299,832</td>
</tr>
<tr>
<td>Tmp7</td>
<td>42,000,000</td>
</tr>
<tr>
<td>Tmp8</td>
<td>4,872,716</td>
</tr>
<tr>
<td>Tmp9</td>
<td>27,467,730,392</td>
</tr>
<tr>
<td>Tmp10</td>
<td>168,000,000</td>
</tr>
<tr>
<td>Tmp12</td>
<td>160,000</td>
</tr>
<tr>
<td>Tmp13</td>
<td>1,269,832</td>
</tr>
<tr>
<td>Tmp14</td>
<td>36,000,000</td>
</tr>
<tr>
<td>Tmp15</td>
<td>30,343,647,328</td>
</tr>
</tbody>
</table>

The intermediate nodes, which are the conjunctively joined nodes with positive weight, project operation that is not the ancestor of base relation and select operation, are inserted into the list of directly affected node. Therefore, the directly affected are \{Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp9, Tmp13, Tmp14 and Tmp15\}.

Next, we identify the indirectly affected nodes. The directly affected nodes that their ancestors are not the directly affected node are Tmp8, Tmp9 and Tmp15. The weights of the ancestor node of those nodes are shown in Table F.21.
Table F.21 The Weight of Ancestor Node of Directly Affected Node of New Queries

<table>
<thead>
<tr>
<th>Directly Affected Node</th>
<th>Ancestor Node</th>
<th>Weight of Ancestor Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tmp8</td>
<td>Tmp17</td>
<td>136,554,444,806</td>
</tr>
<tr>
<td></td>
<td>Tmp18</td>
<td>3,994,303,717,209</td>
</tr>
<tr>
<td></td>
<td>Tmp19</td>
<td>-4,199,188,282,791</td>
</tr>
<tr>
<td>Tmp8</td>
<td>Tmp20</td>
<td>-5,598,918,620,907</td>
</tr>
<tr>
<td></td>
<td>Tmp21</td>
<td>-5,604,673,082,464</td>
</tr>
<tr>
<td>Tmp9</td>
<td>Tmp11</td>
<td>-310,408,939,520</td>
</tr>
<tr>
<td>Tmp15</td>
<td>Tmp16</td>
<td>480,503,214,229</td>
</tr>
<tr>
<td></td>
<td>Tmp22</td>
<td>-1,134,712,881,114</td>
</tr>
</tbody>
</table>

Tmp18 is identified as the indirectly affected node as its weight is the maximum weight of this branch. Tmp11 is not indirectly affected node as its weight negative. Tmp16 is indirectly affected node as its weight is greater than Tmp15.

The result of affected nodes show as follows.

Directly affected nodes: Tmp1, Tmp2, Tmp4, Tmp6, Tmp8, Tmp9, Tmp13, Tmp14, Tmp15

Indirectly affected nodes: Tmp16, Tmp18

Therefore, the number of nodes to be the member of set of views to be selected by 2PO is 17 nodes, 11 existing nodes and 6 new created nodes.

After the affected nodes are identified, the selection algorithm, 2PO, is applied to select the set of views to be materialized. The result is that the existing materialized views {Tmp8, Tmp16, Tmp18, Tmp23} are still materialized, the existing virtual views {Tmp9, Tmp15} are materialized view, the new nodes {Tmp27, Tmp29, Tmp30} are materialized and the existing materialized view {Tmp6} is un-materialized.

We rerun static approach for all queries {Q1, Q8, Q9, Q19, Q30, Q31 and Q41} and {Q2, Q10, Q25, Q29, Q33 and Q42}. The result of the static approach for
Figure F.66  The Dynamic MVPP of the Third Query Set

Figure F.67  The Re-Optimized MVPP by Static Approach for All Queries of the Third Query Set
all queries is shown in Figure F.67. After our MVPP re-optimization algorithm is applied to the cheapest MVPP, the MVPP structure in Figure F.67 provides same structure as that of dynamic MVPP in Figure F.66. The query processing cost, materialized view maintenance cost and total cost of the static and dynamic approach are computed as shown in Table F.22.

**Table F.22** The Comparison of the Result from the Static Approach and the Dynamic Approach of the Third Query Set

<table>
<thead>
<tr>
<th>Approach</th>
<th>Number of Nodes</th>
<th>Cost of Query Processing</th>
<th>Cost of Maintenance</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>31</td>
<td>184,748,883,173</td>
<td>13,766,731,003,390</td>
<td>13,951,479,886,563</td>
</tr>
<tr>
<td>Dynamic</td>
<td>17</td>
<td>184,748,883,173</td>
<td>13,766,731,003,390</td>
<td>13,951,479,886,563</td>
</tr>
</tbody>
</table>

**Conclusion result of the dynamic phase:**

In Figure F.67, we rerun static approach for all queries; the search space contains 31 intermediated nodes for static materialized view selection whereas our dynamic approach for additional queries, the set of intermediated nodes to be selected is 17 nodes, 11 existing nodes and 6 new created nodes. The result in Table F.22 shows that even though all costs of dynamic approach are same as cost of static approach, the number of nodes to be selected of dynamic less than static approach.
BIOGRAPHY

Name Ms. Boontita Suchyukorn

ACADEMIC BACKGROUND

PUBLICATION