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 ......[illegible]......Major Advisor

(Raweewan Auepanwiriyakul, 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 ......[illegible]......Committee Chairperson

(Surapong Auwatanamongkol, Ph.D.)

Associate Professor ......[illegible]......Committee

(Raweewan Auepanwiriyakul, Ph.D.)

Assistant Professor ......[illegible]......Committee

(Ohm Sornil, Ph.D.)

Assistant Professor ......[illegible]......Committee

(Charun Sanrach, Ph.D.)

Instructor ......[illegible]......Dean

(Siwiga Dusadenoad, 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 Warehouses 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

LIST OF TABLES

LIST OF FIGURES

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 database 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 (Theodorators 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* (Theodorators 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 (Theodorators 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.

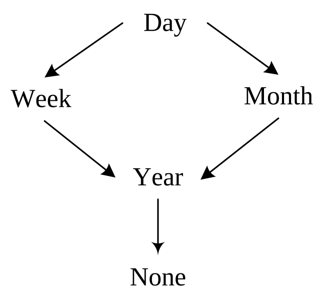


Figure 2.1 Lattice Framework of the Time Dimension
Source: Harinarayan et al., 1996: 209.

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.

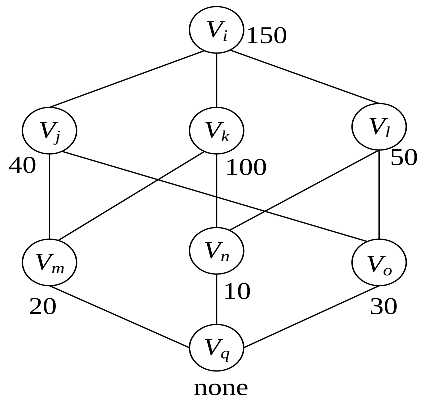


Figure 2.2 Sample Data Cube Lattice Framework for Eight Views
Source: Harinarayan et al., 1996:207.

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 $v_1, v_2, \ldots, v_i$ then all of node $v_1, v_2, \ldots, v_i$ 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*}.

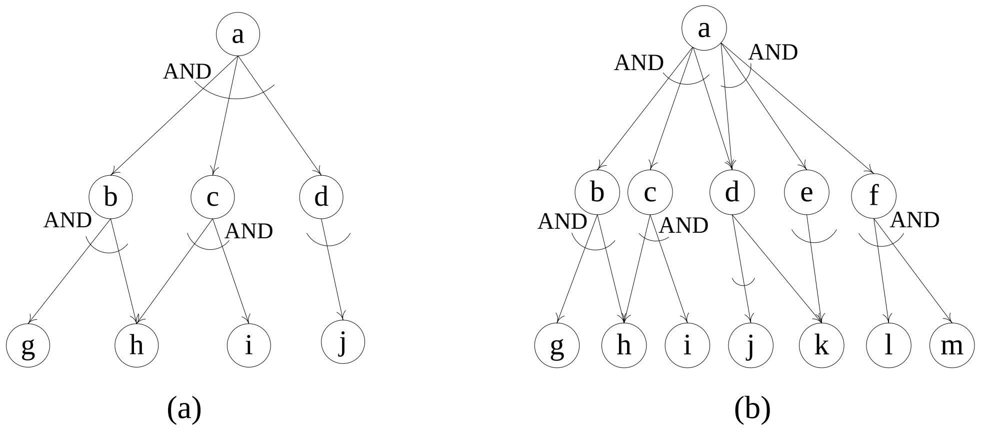


Figure 2.3 (a) An Expression AND DAG (b) An Expression AND-OR DAG
Source: Gupta and Mumick, 2005: 25.

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 $D_1, D_2, \ldots, D_k$. Then G_i be an AND-OR DAG is formed by merging $D_1, D_2, \ldots, D_k$ 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_i.

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.

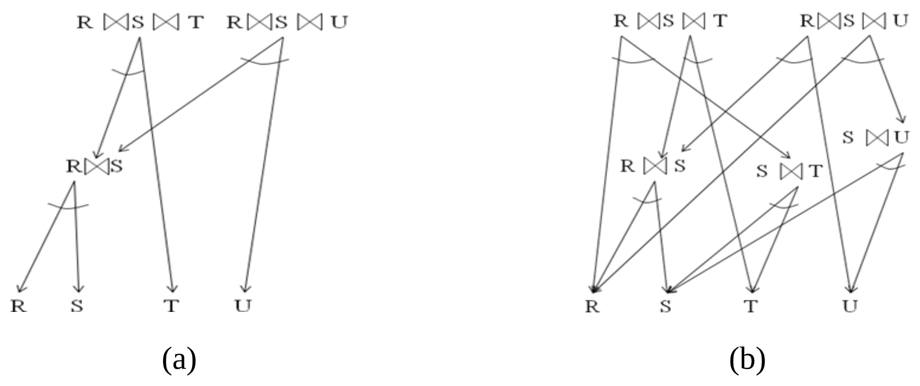


Figure 2.4 (a) An AND DAG (b) An AND-OR DAG

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.

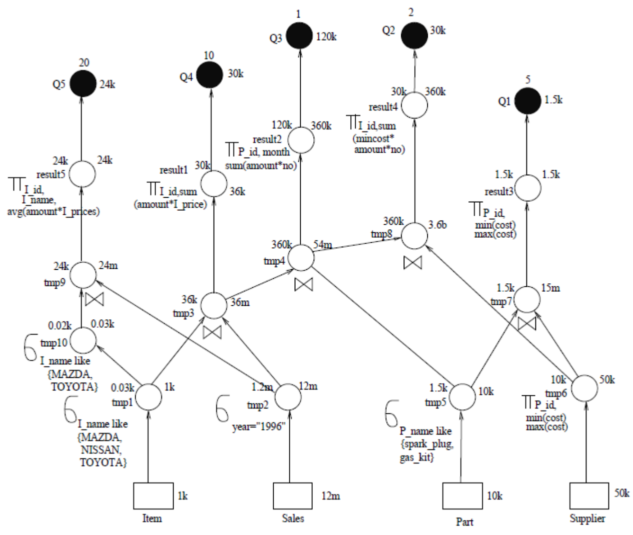


Figure 2.5 Multiple View Processing Plan (MVPP)

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 to be materialized based on AND, OR and AND-OR DAG. Yang et al. (1997: 136) proposed a heuristic algorithm based on MVPP to obtain a minimal total cost which is the summation of query processing cost and maintenance cost.

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, Pellenkoft 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.

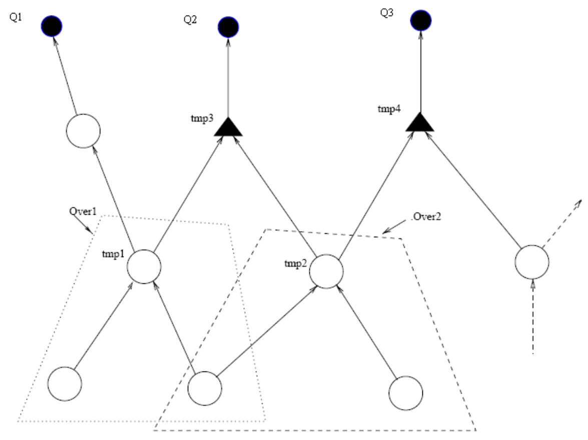


Figure 2.6 Example of the Affected Transference
Source: Zhang and Yang 1999: 253.

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:

$$C_{queryprocessing} = \sum_{q \in Q} f_q C_q(M) \qquad (1)$$

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* is 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:

$$C_{maintenace} = \sum_{v \in M} f_u C_m(v) \tag{2}$$

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* (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:

$$C_{total} = \sum_{q \in Q} f_q C_q(M) + \sum_{v \in M} f_u C_m(v) \tag{3}$$

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 re-computing 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.

$$w(v) = \sum_{q \in O_v} \left\{ f_q(q) * \left(C_a^q(v) \right) \right\} - \sum_{r \in I_v} \left\{ f_u(r) * \left(C_m^r(v) \right) \right\} \qquad (4)$$

O_v	denotes the queries which use view *v*.
I_v	denotes the base relations which are used to produce view *v*.
C_a^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*.
C_m^r	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.

CHAPTER 3

METHODOLOGY

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.

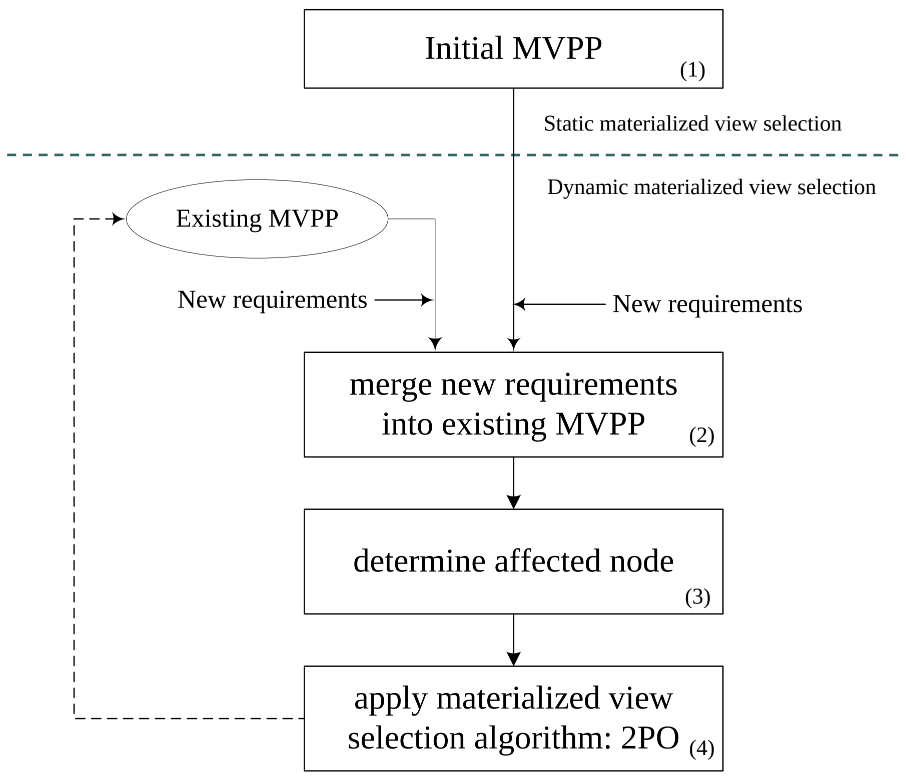

Initial MVPP (1)
Static materialized view selection
Dynamic materialized view selection
Existing MVPP
New requirements
New requirements
merge new requirements into existing MVPP (2)
determine affected node (3)
apply materialized view selection algorithm: 2PO (4)

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.

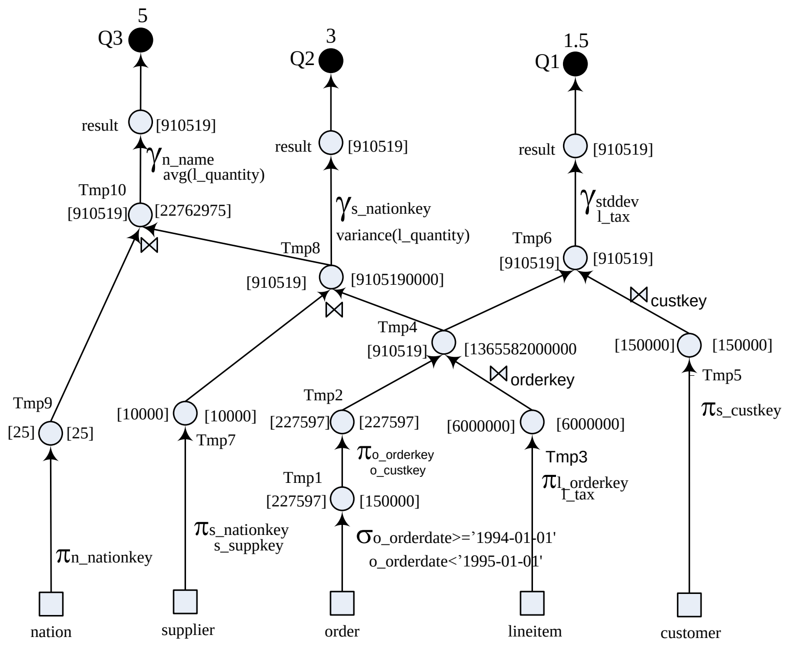


Figure 3.2 The Simple MVPP of Three Queries Q1, Q2 and Q3

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 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.

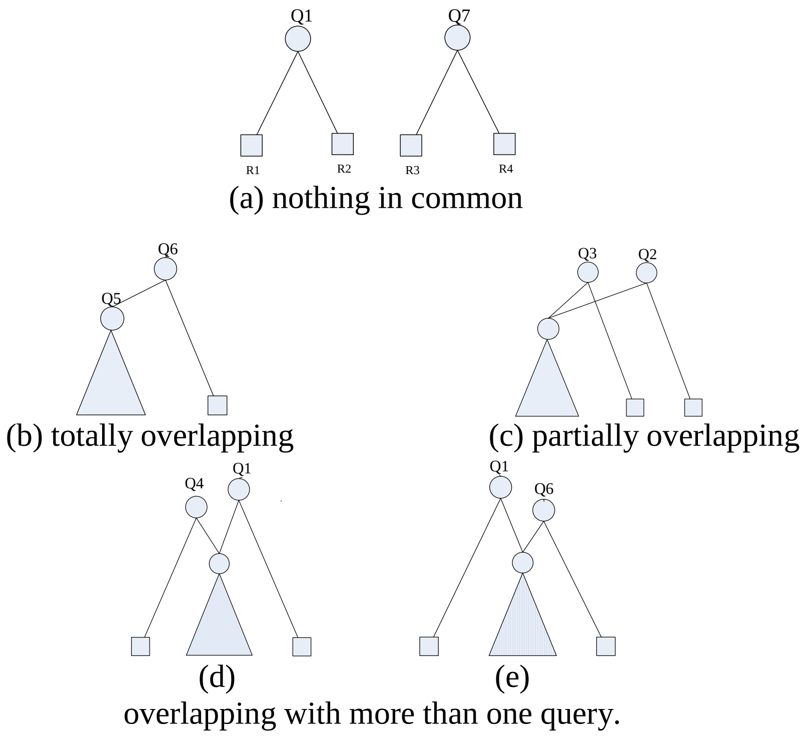


Figure 3.4 The Categories of Subexpression Commonalities

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.

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 nth 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

Cq(i) denotes the query processing cost of query *i* in the cheapest MVPP

$Cq_j'(i)$ denotes the query processing cost of query *i* in the nth MVPP.

If $Cq_j'(i)$ less than *Cq(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.

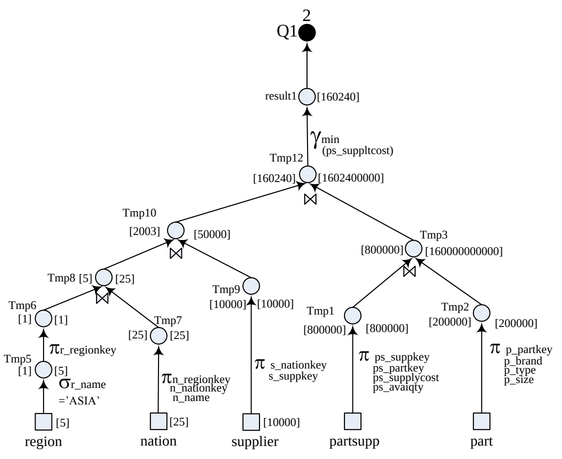
For example, Figure 3.6 illustrates the possible individual plans of query Q1 and MVPP of Q1, Q2 and Q3. Figure 3.6 (a) and (b) are the possible execution plans for Q1. The MVPP of Q1, Q2 and Q3 is shown in Figure 3.6 (c). In MVPP, Q1 is constructed by using Plan A as shown in Figure 3.6(a) that is {Tmp10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

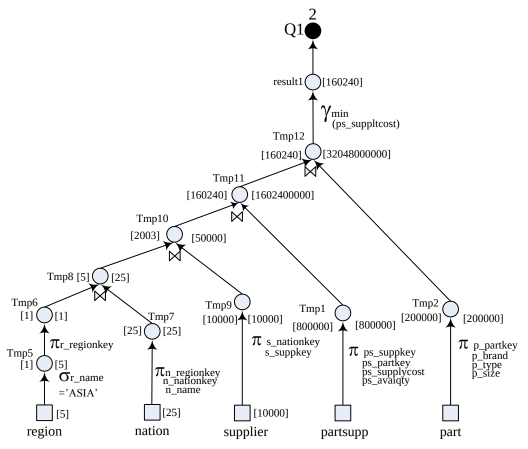
For Plan A in Figure 3.6 (a), Q1 accesses nodes named Tmp1, Tmp2, Tmp3, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp12 and result1. The processing cost of each node is 800000, 200000, 160000000000, 5, 1, 25, 25, 10000, 50000, 1602400000 and 160240, respectively. The frequency of executing the query Q1 is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


2
Q1
result1 [160240]
γ_{min} (ps_suppltcost)
Tmp12 [160240] [1602400000]
Tmp10 [2003] [50000]
Tmp3 [800000] [160000000000]
Tmp8 [5] [25]
Tmp9 [10000] [10000]
Tmp6 [1] [1]
Tmp7 [25] [25]
Tmp1 [800000] [800000]
Tmp2 [200000] [200000]
πr_regionkey
Tmp5 [1] [5]
σr_name ='ASIA'
πn_regionkey n_nationkey n_name
π s_nationkey s_suppkey
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5] [25] [10000]
region nation supplier partsupp part
(a) Plan A of Q1


2
Q1
result1 [160240]
γ_{min} (ps_suppltcost)
Tmp12 [160240] [32048000000]
Tmp11 [160240] [1602400000]
Tmp10 [2003] [50000]
Tmp8 [5] [25]
Tmp6 [1] [1]
Tmp7 [25] [25]
Tmp9 [10000] [10000]
Tmp1 [800000] [800000]
Tmp2 [200000] [200000]
πr_regionkey
Tmp5 [1] [5]
σr_name ='ASIA'
πn_regionkey n_nationkey n_name
π s_nationkey s_suppkey
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5] [25] [10000]
region nation supplier partsupp part
(b) Plan B of Q1

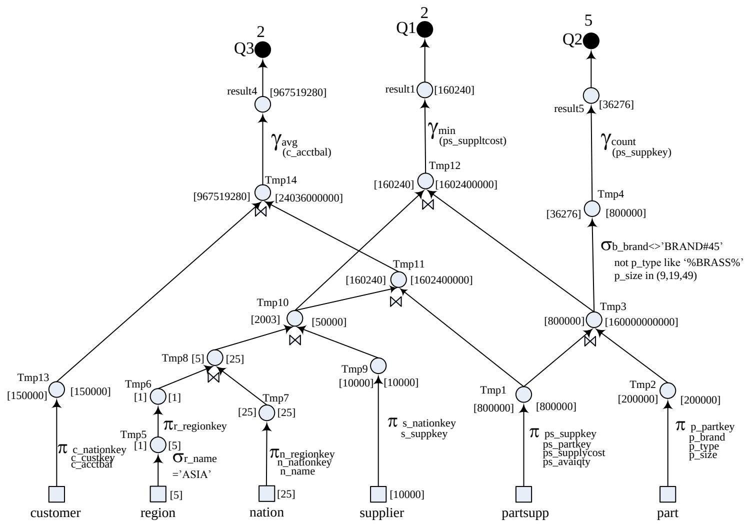

2
Q3
2
Q1
5
Q2
result4 [967519280]
result1 [160240]
result5 [36276]
γ_{avg} (c_acctbal)
γ_{min} (ps_suppltcost)
γ_{count} (ps_suppkey)
Tmp14 [967519280] [24036000000]
Tmp12 [160240] [1602400000]
Tmp4 [36276] [800000]
σb_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp11 [160240] [1602400000]
Tmp10 [2003] [50000]
Tmp3 [800000] [160000000000]
Tmp8 [5] [25]
Tmp9 [10000] [10000]
Tmp13 [150000] [150000]
Tmp6 [1] [1]
Tmp7 [25] [25]
Tmp1 [800000] [800000]
Tmp2 [200000] [200000]
πr_regionkey
Tmp5 [1] [5]
σr_name ='ASIA'
π c_nationkey c_custkey c_acctbal
πn_regionkey n_nationkey n_name
π s_nationkey s_suppkey
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5] [25] [10000]
customer region nation supplier partsupp part
(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
        $S_{min} = S_\infty$; {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($S_{min}$) then $S_{min}$ = S
         }
        return($S_{min}$)
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}
        $S_{min}$ = S;
        while not(time limit) do {
                while not(local minimum(S)) do {
                        S' = random state in neighbor(S)
                        $\Delta C$ = cost(S') - cost(S)
                        if ($\Delta C \leq 0$) then S = S'
                        if ($\Delta C > 0$) then S = S' with probability $e^{-\Delta C / T}$
                        if cost(S) < cost($S_{min}$) then $S_{min}$ = S
                }
                T = reduce(T)
        }
        return{$S_{min}$}
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, $e^{-\Delta C/T}$, ΔC 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:

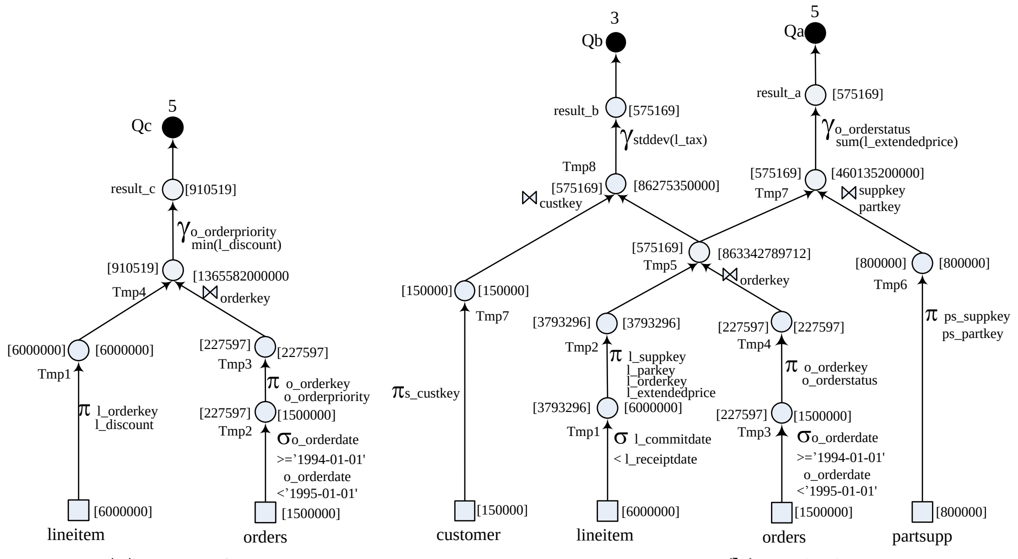
1) The MVPP will be changed for the first and second situation.

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 rewrite 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 (σ_c ORDERS $\bowtie$ LINEITEM) that is Tmp4 in Figure 3.12 (a). So, the conjunctively joined of (σ_c ORDERS $\bowtie$ 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 {σ_cORDERS $\bowtie$ ($\sigma_{l\_commitdate<l\_receiptdate}$ LINEITEM)}.



(a) New Query (b) Existing MVPP

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, (σ_cORDERS ⋈ LINEITEM) to support Qc. If Qa, Qb still use Tmp5 then (σ_cORDERS ⋈ LINEITEM) will be created specific for Qc only. However, if we rewrite Qa, Qb by using (σ_cORDERS ⋈ LINEITEM) then (σ_cORDERS ⋈ 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

```
Begin
    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 MVPP re-optimization algorithm described in section 3.4 to
       the cheapest MVPP.
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:

$$w(v) = \sum_{q \in O_v} \left\{ f_q(q) * \left(C_a^q(v) \right) \right\} - \sum_{r \in I_v} \left\{ f_u(r) * \left(C_m^r(v) \right) \right\}$$

w(v) denotes weight of node

O_v denotes the queries which use view *v*.

C_a^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_m^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 un-materialized, 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
    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
        3.5  Depth first search from the root to base relations to determine the
             existing intermediate nodes, $v_i$ , used to construct the new query.
        3.6  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 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, *uj* 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_i, 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_i, and the weight of u_j is greater than v_i 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 $\sum_{q \in Q} f_q C_q(M)$ 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.

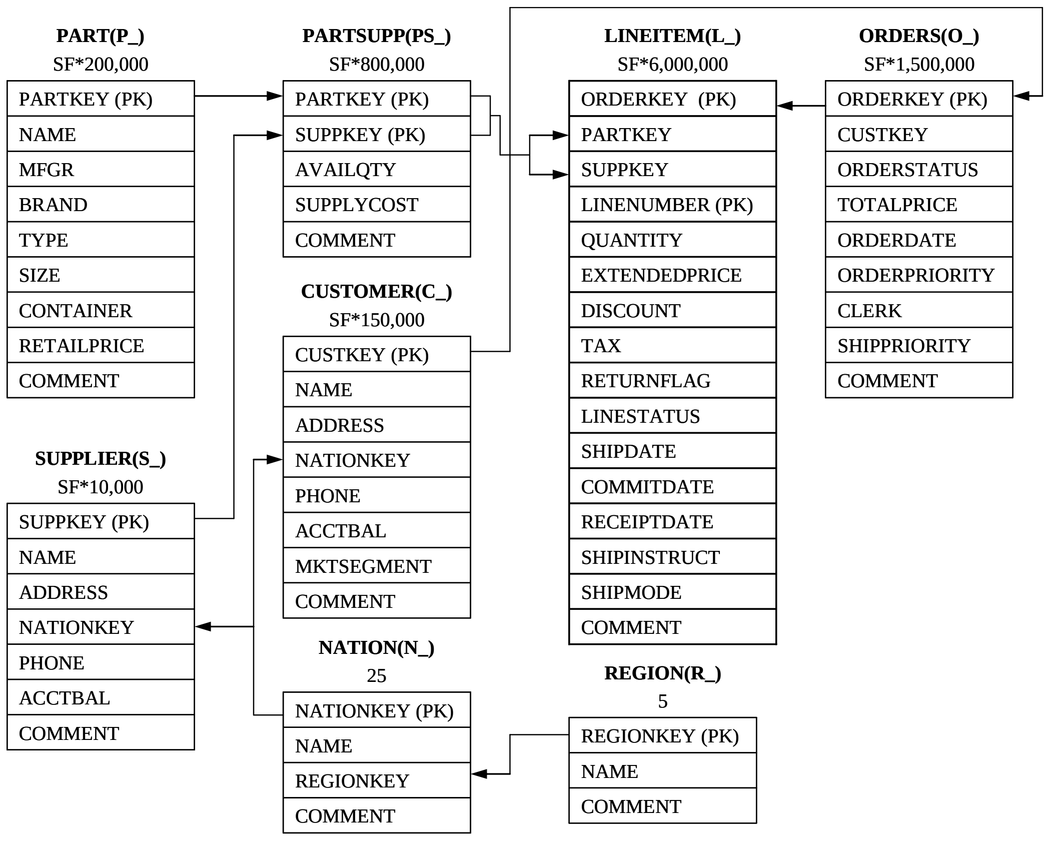


Figure 4.1 The TPC-H Schema Revision 2.14.2

Source: Transaction Processing Performance Council (TPC), 2011: 12.

Table 4.1 The TPC-H Schema Table Size

Table name	Relation Size (in Tuples)	Record Size (in bytes)	Table Size (in MB)
REGION	5	124	<1
NATION	25	128	<1
CUSTOMER	150,000	179	26
SUPPLIER	10,000	159	2
PART	200,000	155	30
PARTSUPP	800,000	144	110
LINEITEM	6,000,000	112	641
ORDERS	1,500,000	104	149

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:

σ_a represents the select operation, where *a* is a selection condition on one or more attributes of a relation.

π_b represents the project operation, where *b* is a list of one or more attributes of a relation.

$\bowtie$ represents the inner join operation.

γ 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

```
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;
```

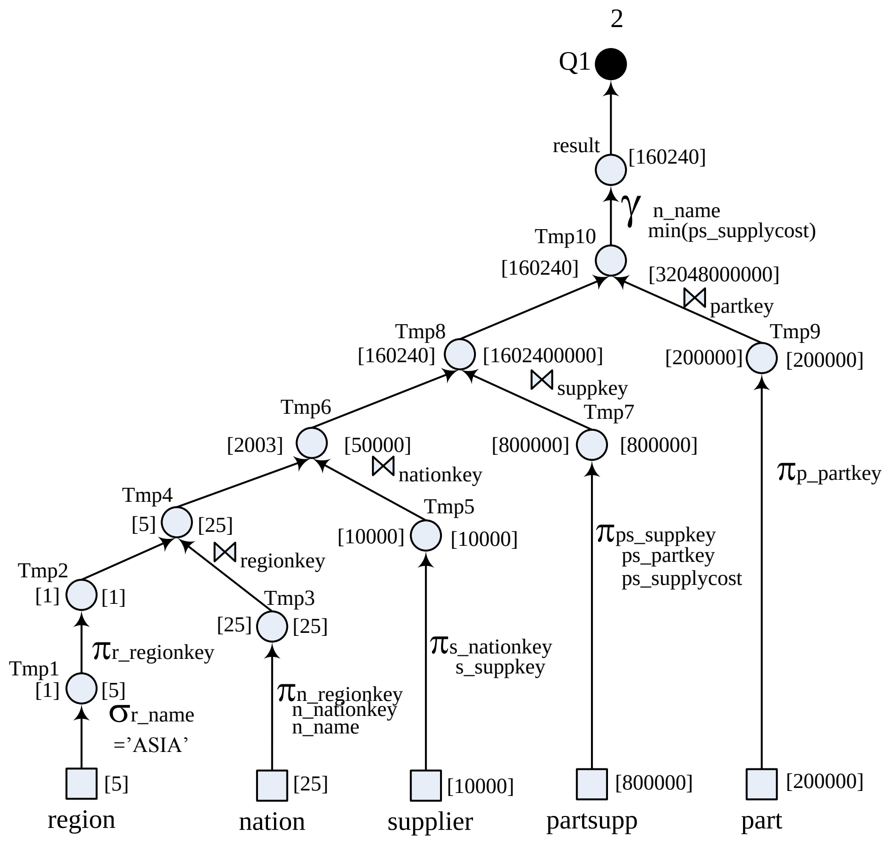


Figure 4.2 Relational Algebra Query Tree of Query Q1

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

```
SELECT      N_NAME, COUNT(L_ORDERKEY)
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;
```

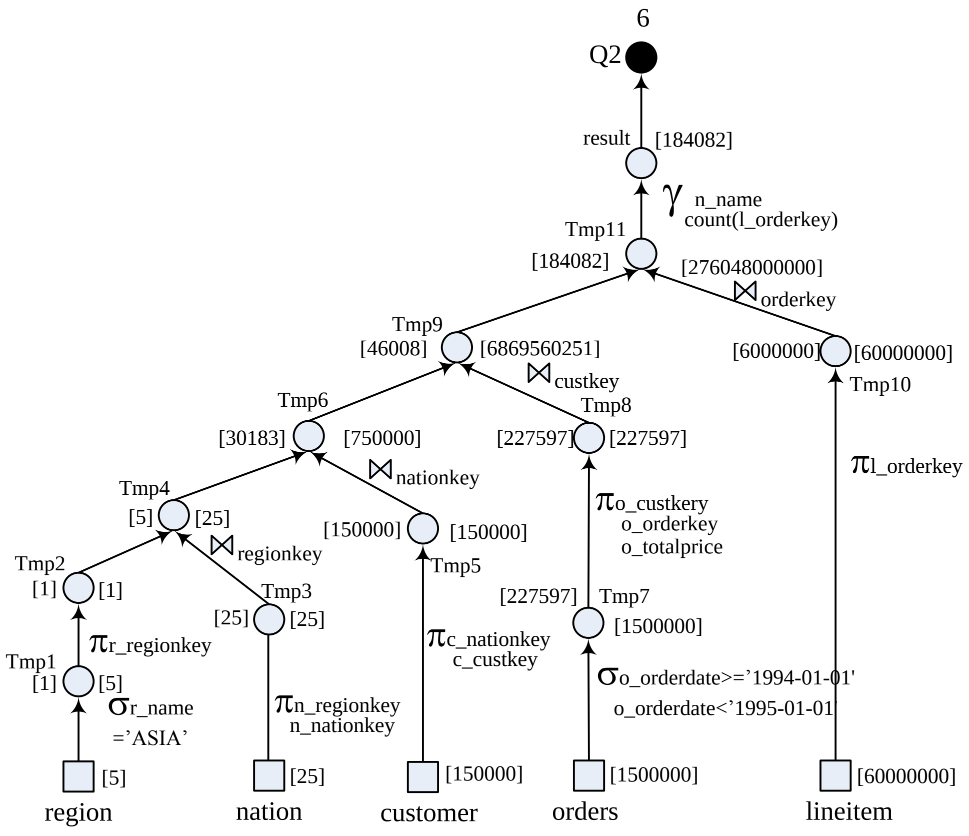


Figure 4.3 Relational Algebra Query Tree of Query Q2

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;
```

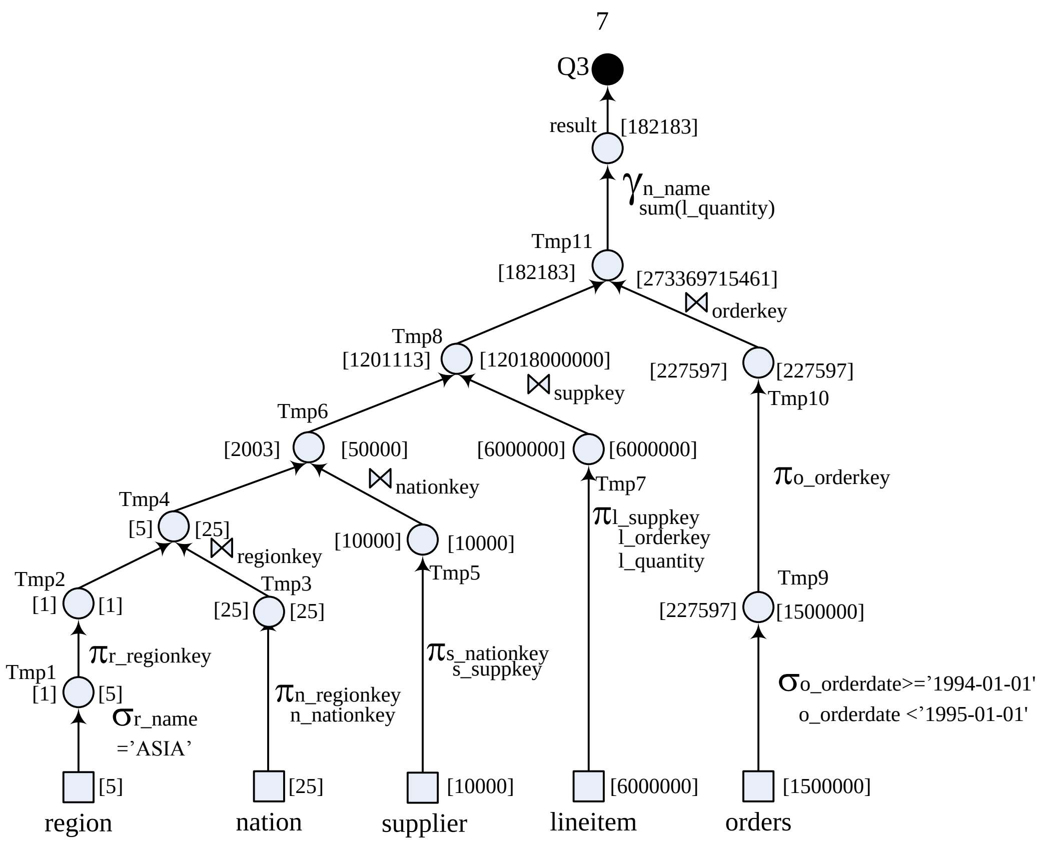


Figure 4.4 Relational Algebra Query Tree of Query Q3

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;
```

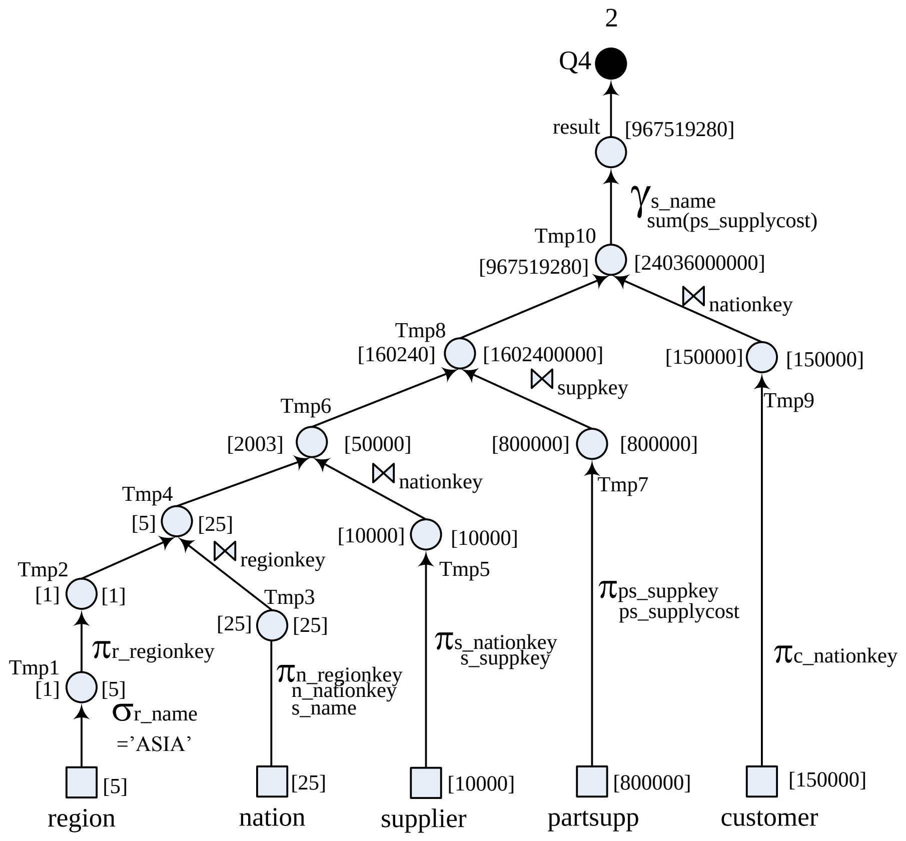


Figure 4.5 Relational Algebra Query Tree of Query Q4

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

```
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);
```

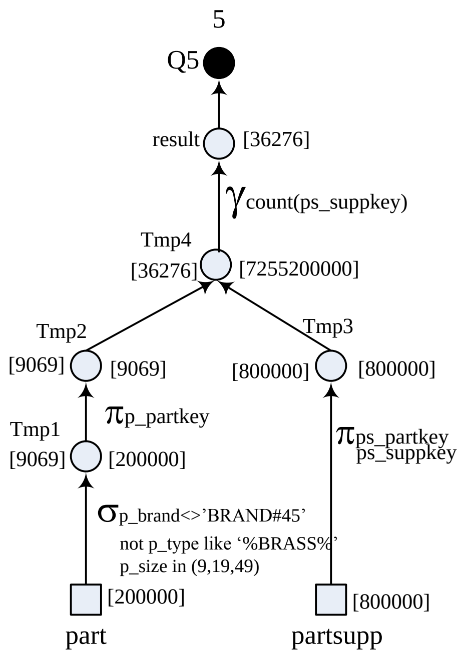


Figure 4.6 Relational Algebra Query Tree of Query Q5

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

```
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;
```

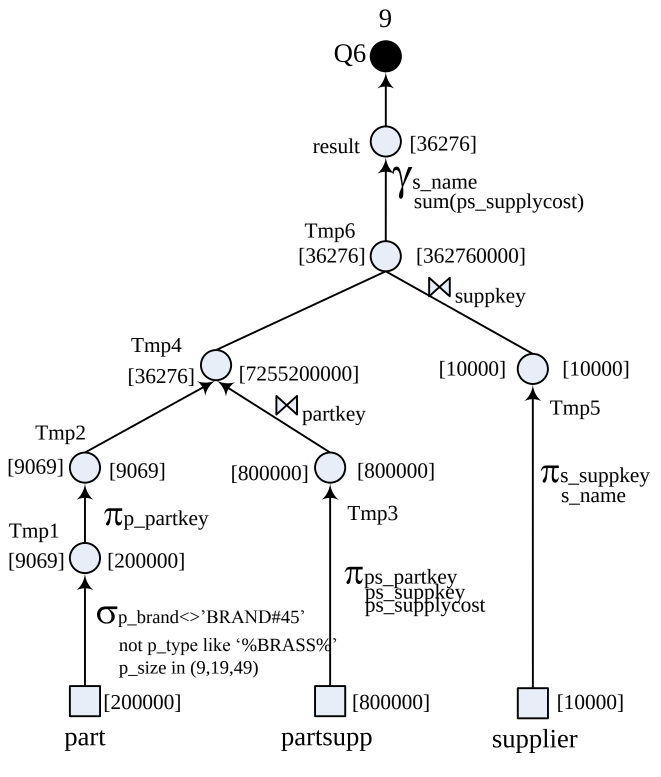


Figure 4.7 Relational Algebra Query Tree of Query Q6

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

```
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;
```

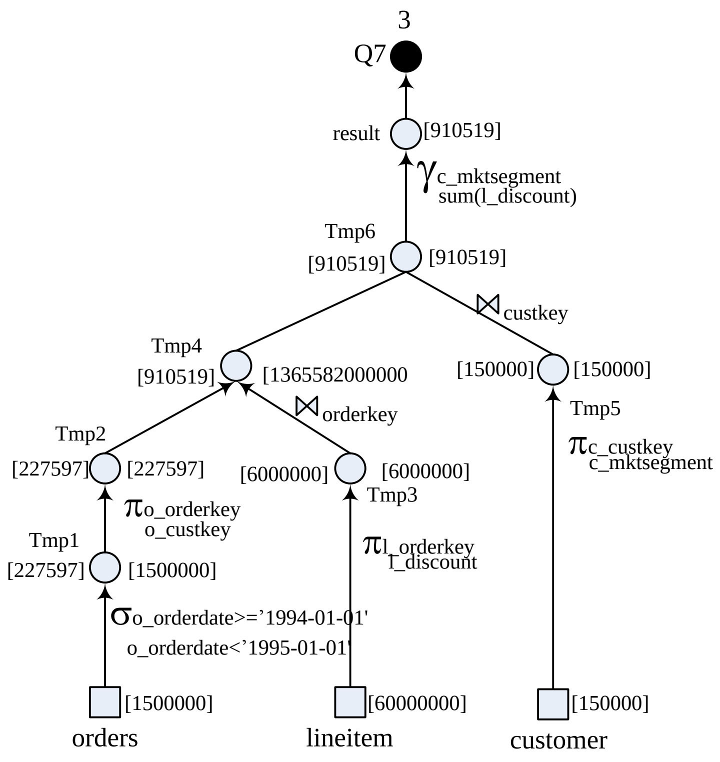


Figure 4.8 Relational Algebra Query Tree of Query Q7

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}.

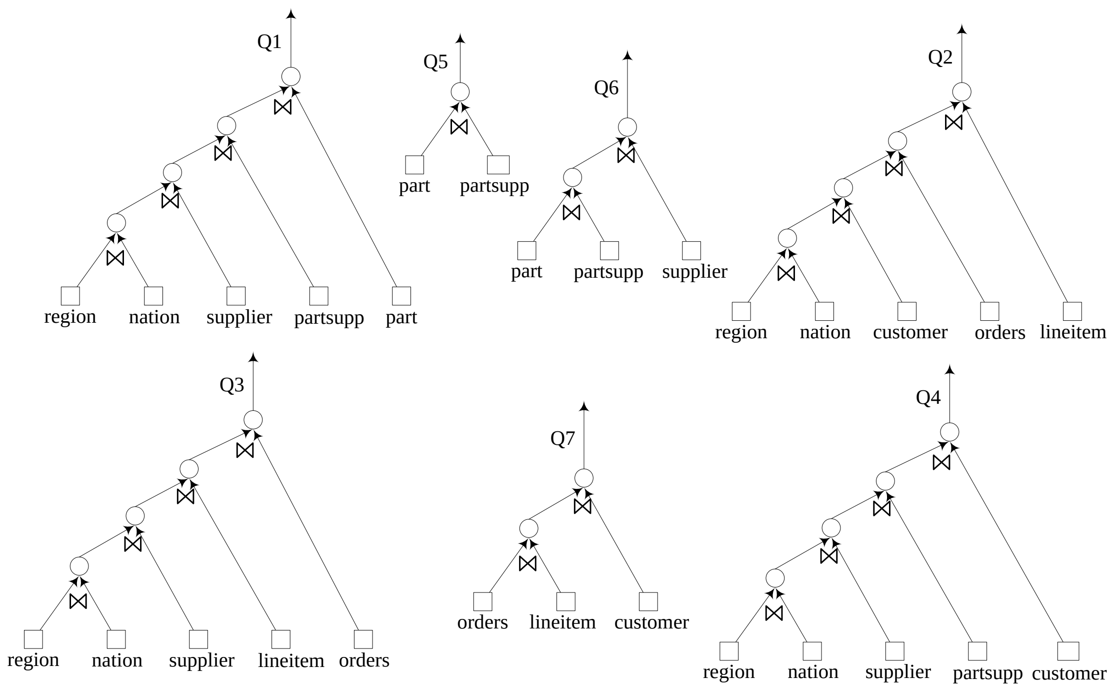
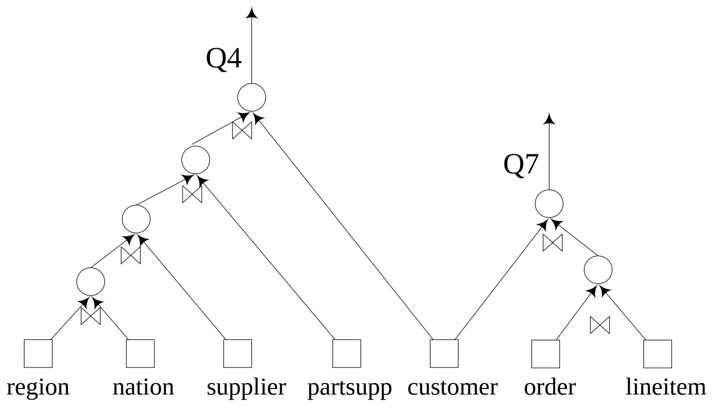


Figure 4.9 The Result of the First Step to Construct MVPP

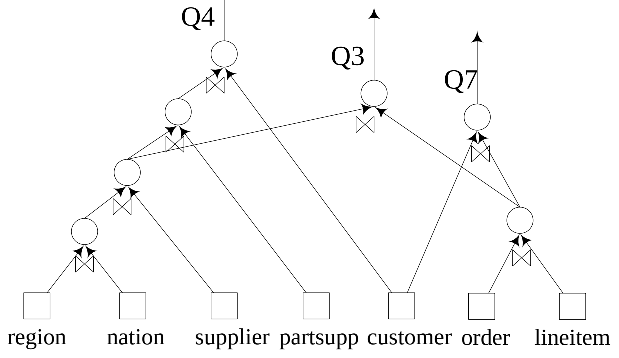
Table 4.2 The Query Access Frequency, Query Cost, and Query Access Frequency Multiplied by Query Cost

Query	**Query Access Frequency(f_q)**	**Query Cost**	**f_q * Query Cost**
Q1	2	160,240	320,480
Q2	6	184,042	1,104,252
Q3	7	182,183	1,275,281
Q4	2	967,519,280	1,935,038,560
Q5	5	36,276	181,380
Q6	9	36,276	326,484
Q7	3	910,519	2,731,557

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 ⋈ NATION ⋈ SUPPLIER ⋈ LINEITEM ⋈ ORDERS). There are two subtrees available for Q3 that are (REGION ⋈ NATION ⋈ SUPPLIER) and (ORDERS ⋈ 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 ⋈ NATION), the remaining base relations are CUSTOMER, ORDERS and LINEITEM conjunctively joined already for Q7.

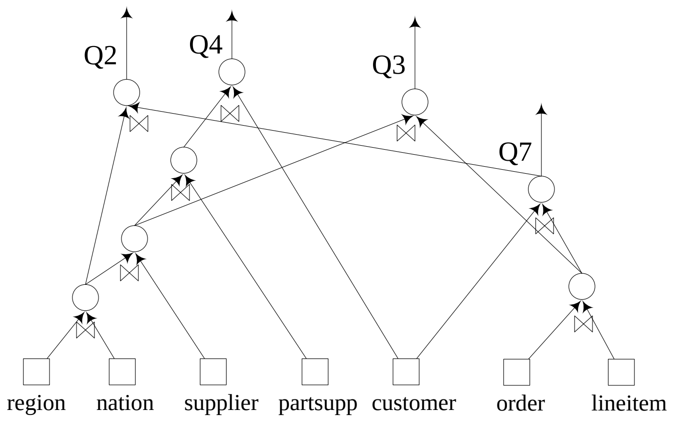


(a) The result after Q7 is merged with Q4 of first MVPP

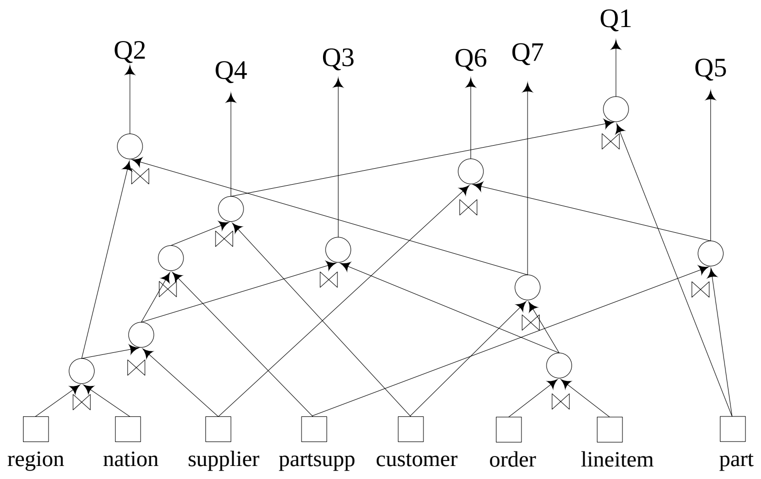


(b) The result after Q3 is merged of the first MVPP

Figure 4.10 The Result of Merging Steps for the First MVPP



(c) The result after Q2 is merged of the first MVPP

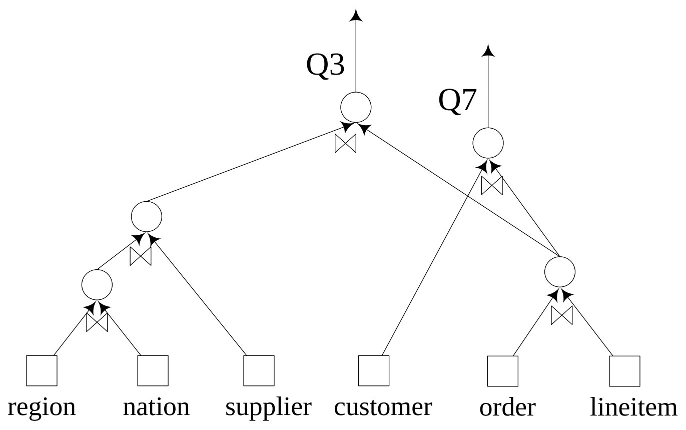


(d) The result of the first MVPP after all Queries Q1-Q7 are merged

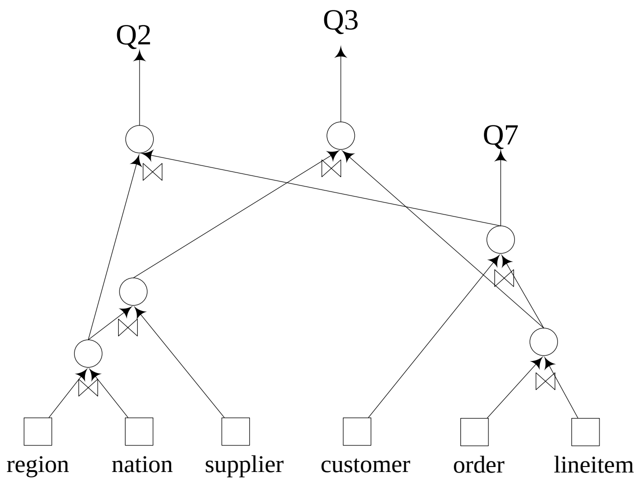
Figure 4.10 (Continued)

Therefore, the new node is introduced for Q2 as a join operation of those results: (REGION ⋈ NATION) and (CUSTOMER ⋈ ORDERS ⋈ 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).

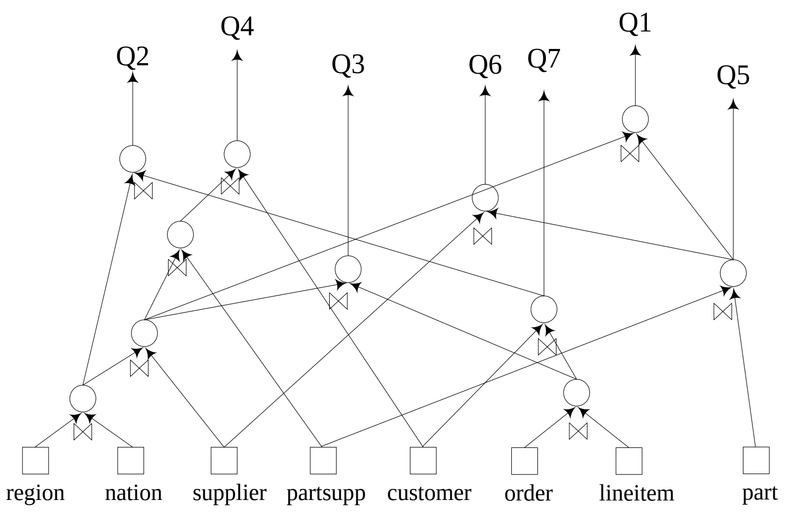


(a) The result after Q3 is merged with Q7 of the second MVPP



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

Figure 4.11 The Result of Merging Steps for the Second MVPP



(c) 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)*(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.

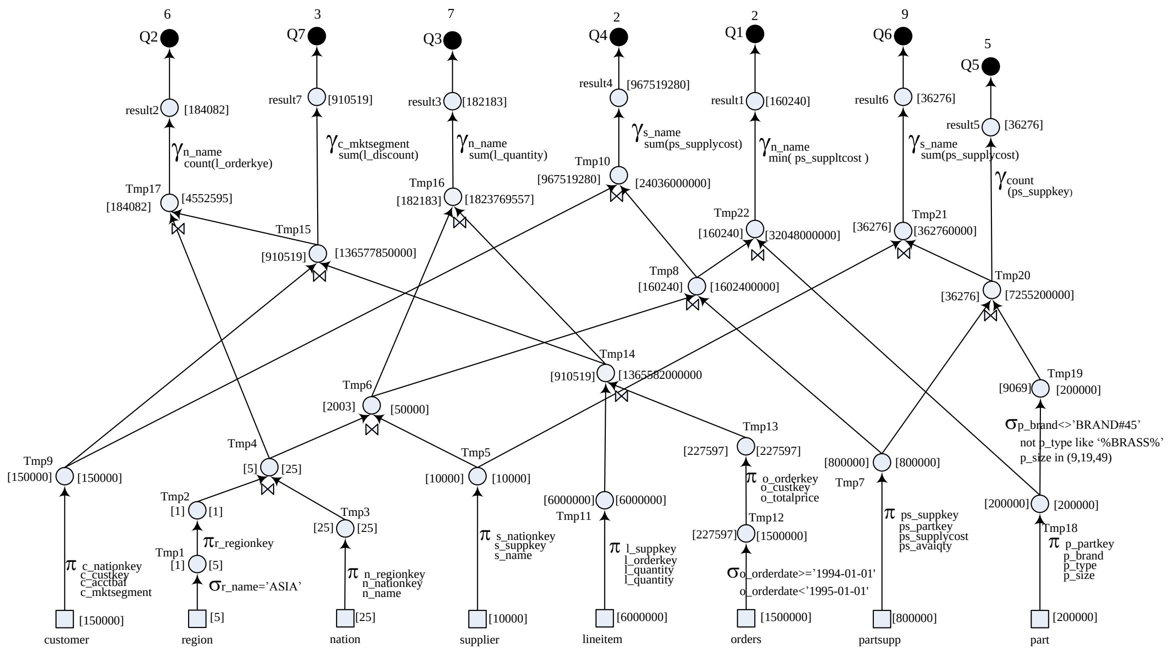


Figure 4.12 The First MVPP, the Queries in the List: {Q4, Q7, Q3, Q2, Q6, Q1, and Q5}

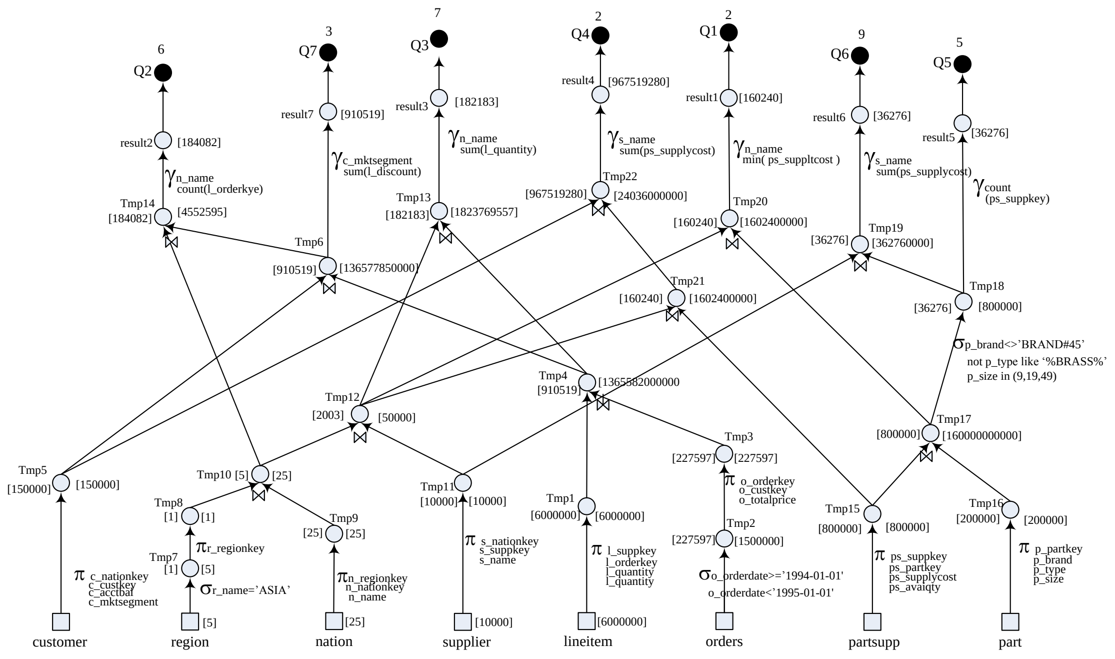


Figure 4.13 The Second MVPP, the Query in the List: {Q7, Q3, Q2, Q6, Q1, Q5, and Q4}

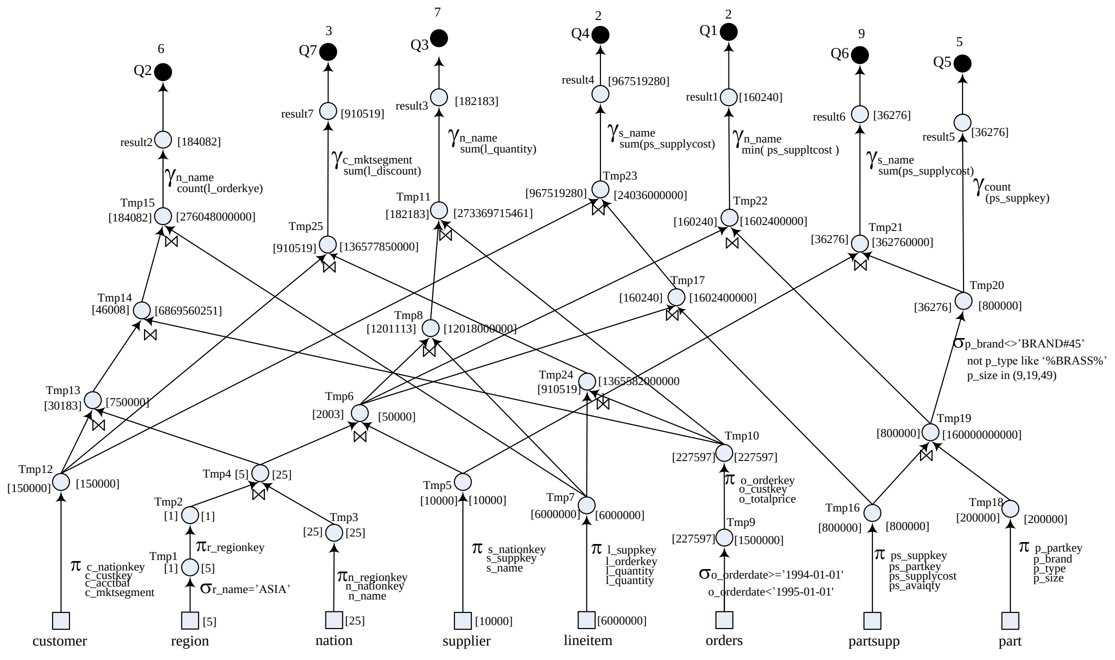


Figure 4.14 The Third MVPP (the Cheapest MVPP), the Query in the List: {Q3, Q2, Q6, Q1, Q5, Q4, and Q7}

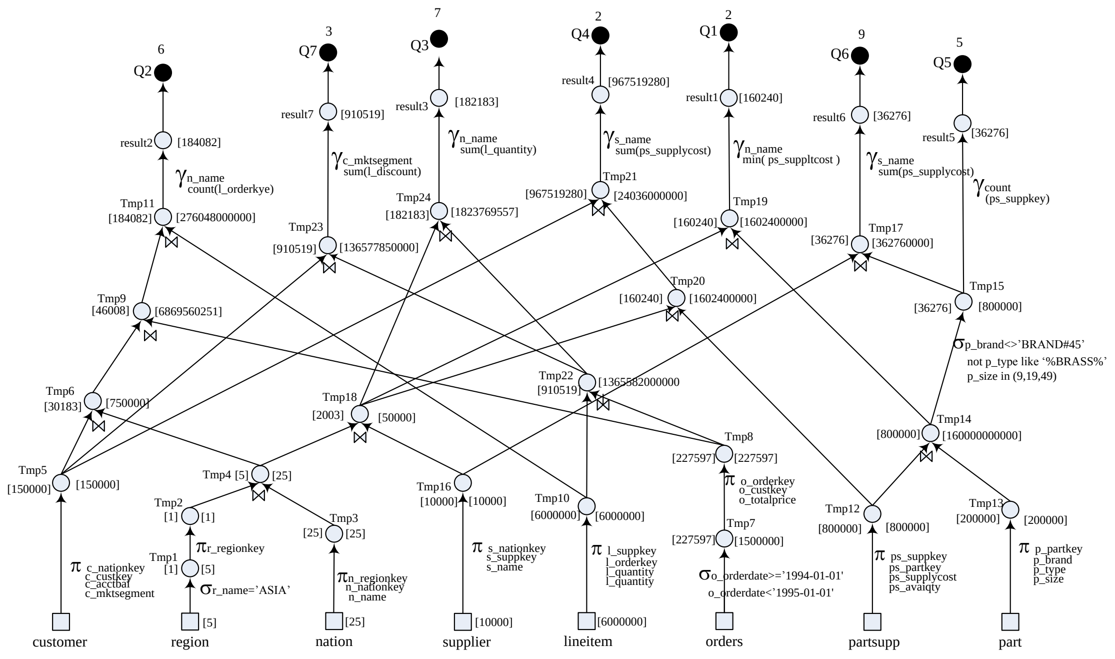


Figure 4.15 The Fourth MVPP, the Query in the List: {Q2, Q6, Q1, Q5, Q4, Q7, and Q3}

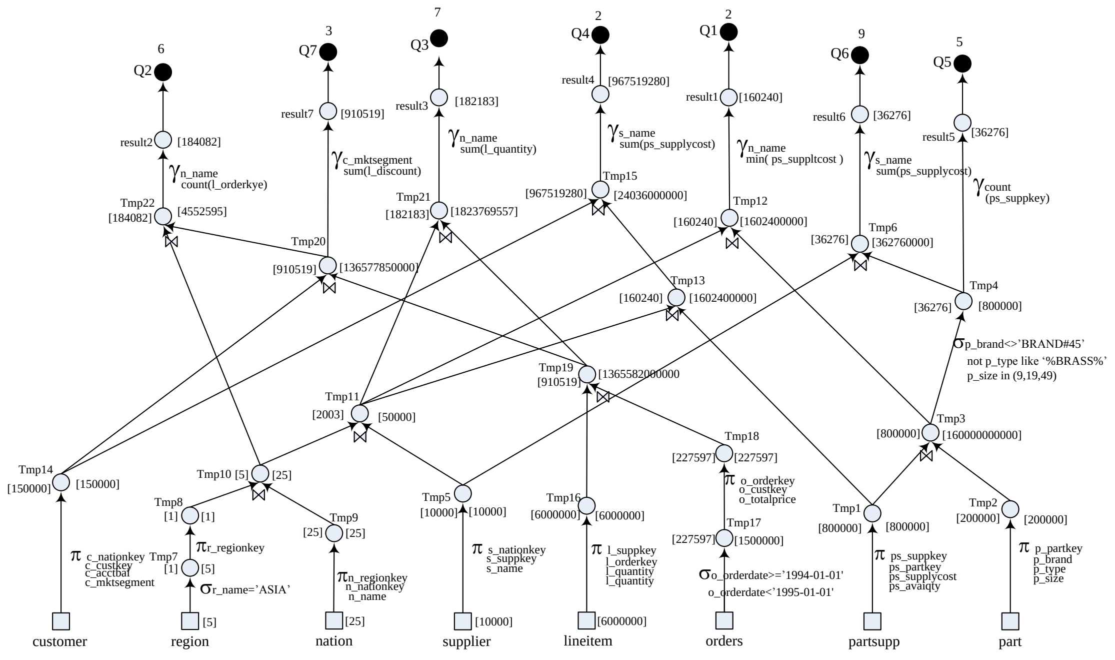


Figure 4.16 The Fifth MVPP, the Query in the List: {Q6, Q1, Q5, Q4, Q7, Q3, and Q2}

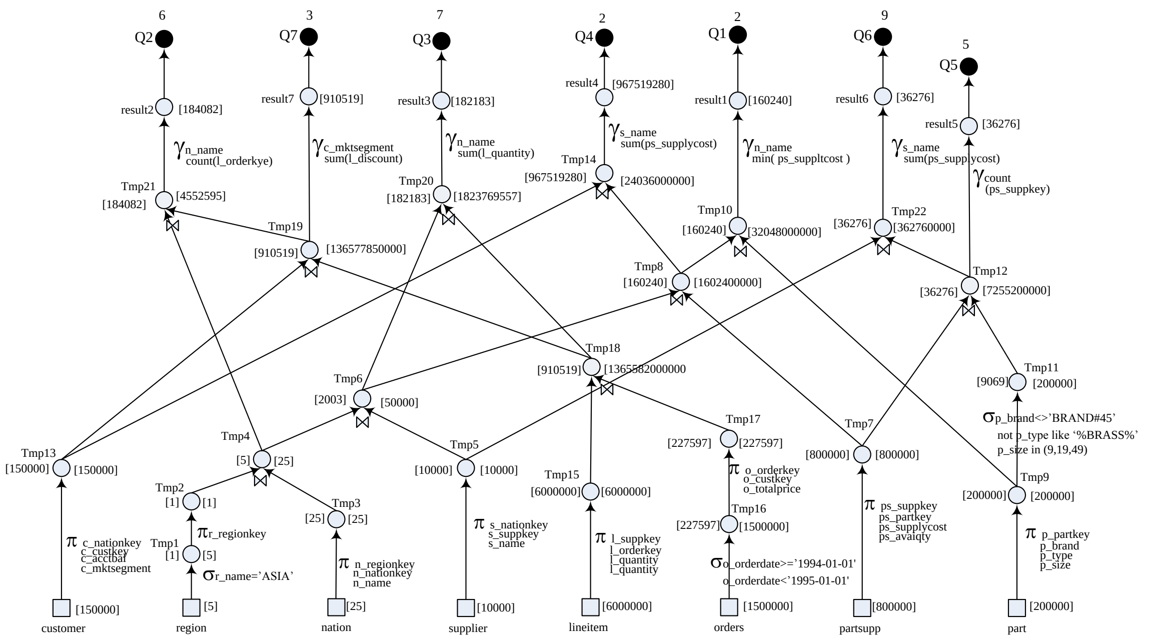


Figure 4.17 The Sixth MVPP, the Query in the List: {Q1, Q5, Q4, Q7, Q3, Q2, and Q6}

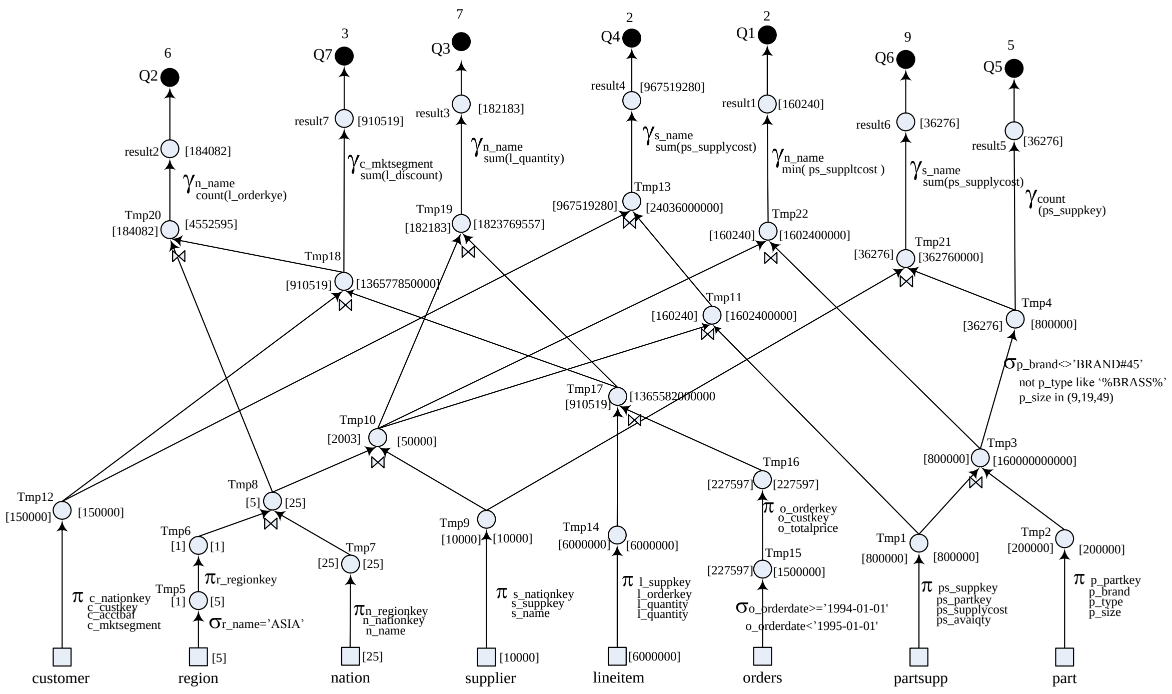


Figure 4.18 The Seventh MVPP, the Query in the List: {Q5, Q4, Q7, Q3, Q2, Q6, and Q1}

Table 4.3 The Query Processing Cost of the First MVPP

Query	**f_q**	**Access from the Nodes**	**Cost of Nodes**	**Query Processing Cost**
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp7, Tmp8, Tmp18, Tmp22 and result1	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15, Tmp17 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp11, Tmp12, Tmp13, Tmp14, Tmp16 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp7, Tmp18, Tmp19, Tmp20 and result5	800000, 200000, 200000, 7255200000 and 36276	36,282,181,380
Q6	9	Tmp7, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6	800000, 200000, 200000, 7255200000, 10000, 362760000 and 36276	68,572,856,484
Q7	3	Tmp11, Tmp12, Tmp13, Tmp14, Tmp9 , Tmp15 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the first MVPP		**23,316,809,013,207**

Table 4.4 The Query Processing Cost of All MVPPs

Query	**1st MVPP**	**2nd MVPP**	**3rd MVPP (Cheapest MVPP)**	**4th MVPP**
Q1	67,303,240,592	323,207,240,592	323,207,240,592	323,207,240,592
Q2	9,013,034,785,980	9,013,034,785,980	1,697,558,231,916	1,697,558,231,916
Q3	9,571,896,175,751	9,571,896,175,751	1,997,769,797,079	9,571,896,175,751
Q4	53,213,858,672	53,213,858,672	53,213,858,672	53,213,858,672
Q5	36,282,181,380	800,009,181,380	800,009,181,380	800,009,181,380
Q6	68,572,856,484	1,443,281,456,484	1,443,281,456,484	1,443,281,456,484
Q7	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348
Total	**23,316,809,013,207**	**25,711,148,613,207**	**10,821,545,680,471**	**18,395,672,059,143**

Query	**5th MVPP**	**6th MVPP**	**7th MVPP**
Q1	323,207,240,592	67,303,240,592	323,207,240,592
Q2	9,013,034,785,980	9,013,034,785,980	9,013,034,785,980
Q3	9,571,896,175,751	9,571,896,175,751	9,571,896,175,751
Q4	53,213,858,672	53,213,858,672	53,213,858,672
Q5	800,009,181,380	36,282,181,380	800,009,181,380
Q6	1,443,281,456,484	68,572,856,484	1,443,281,456,484
Q7	4,506,505,914,348	4,506,505,914,348	4,506,505,914,348
Total	**25,711,148,613,207**	**23,316,809,013,207**	**25,711,148,613,207**

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 $LV = \phi$.
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 $Cq_j'(i)$ 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 $LV=\phi$. 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.

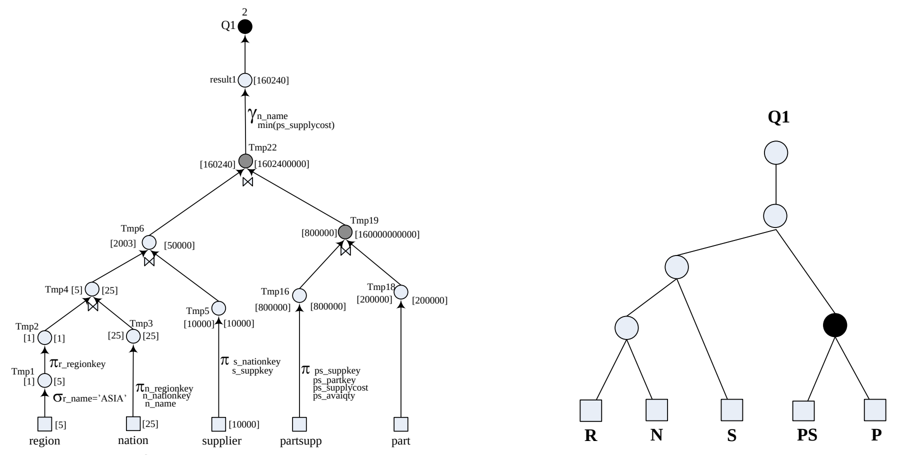
Because Q1 has overlapping with several queries and its query processing cost in the cheapest MVPP is higher than that of the first and the sixth MVPP so Q1 in the cheapest MVPP would be rewritten as follows:

- We match optimal query processing plan of Q1 in bottom-up way from leaf node up to the query node.

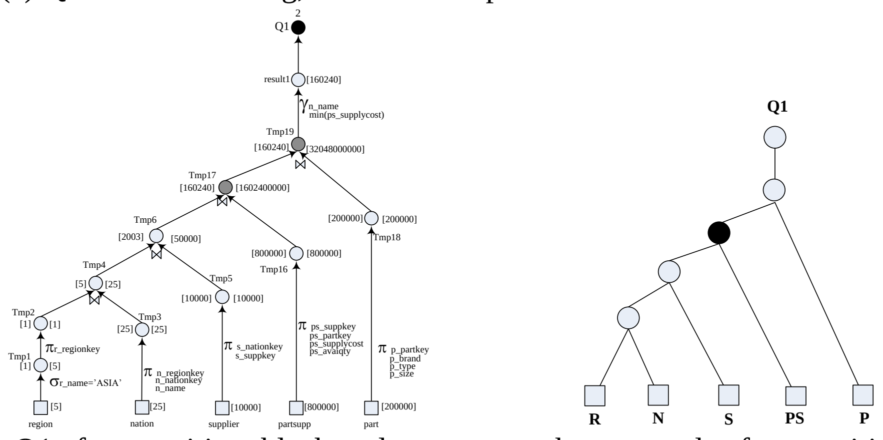
- We can match the node in optimal query processing plan of Q1 with the nodes in the cheapest MVPP at node Tmp6 and Tmp17 in Figure 4.14. Tmp17 in Figure 4.14 is the conjunctively join of the base relations i.e. REGION, NATION, SUPPLIER, and PARTSUPP of Q1.

- 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.



(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.

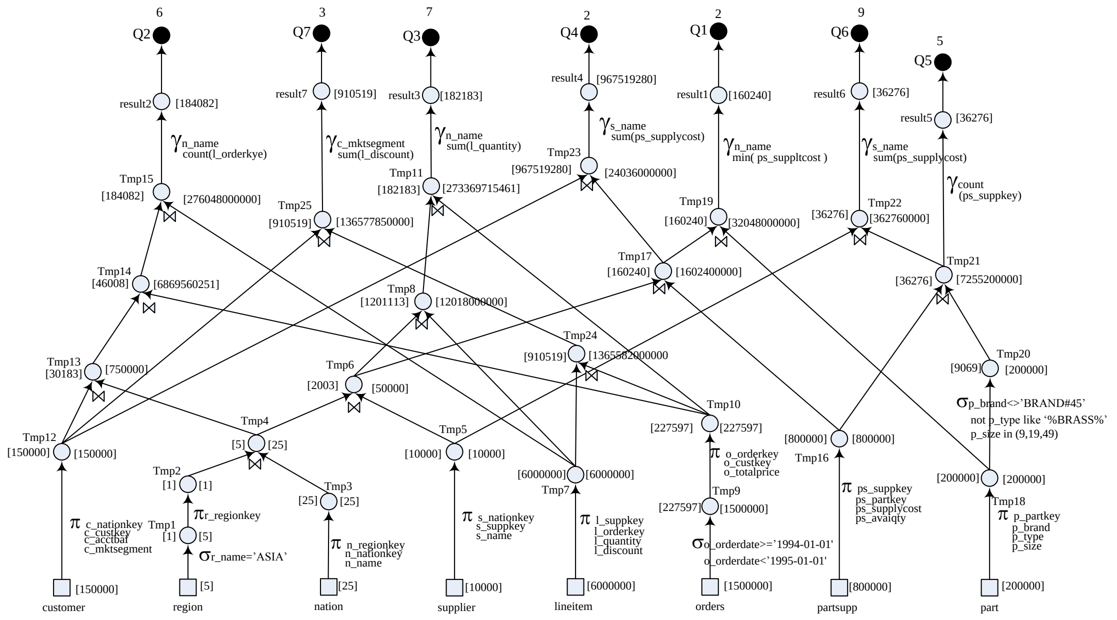


Figure 4.20 The Cheapest MVPP after Applying the MVPP Re-Optimization Algorithm

Table 4.5 The Query Processing Cost of the Cheapest MVPP and the Re-Optimized MVPP

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

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.

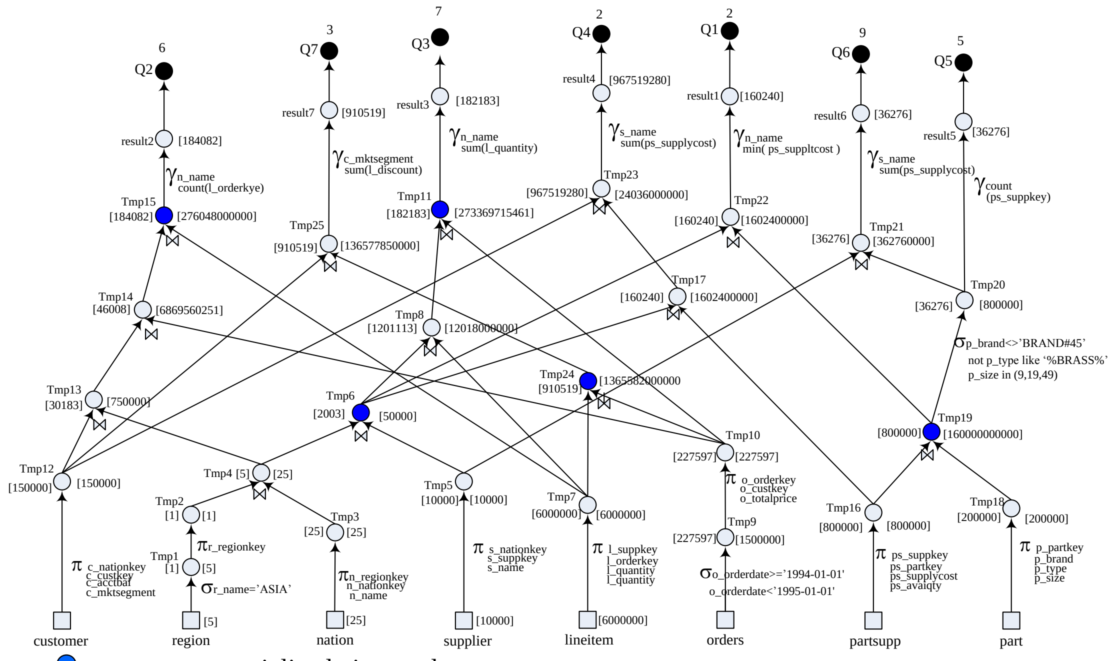
The details of implementing 2PO for materialized view selection are described in Figure 4.21.

```
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:

[Tmp22,0], [Tmp19,0], [Tmp18,0], [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], [Tmp17,0], [Tmp20,0], [Tmp21,0], [Tmp25,0], [Tmp24,0]



Q2 6
Q7 3
Q3 7
Q4 2
Q1 2
Q6 9
Q5 5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp25 [910519] [136577850000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp22 [160240] [1602400000]
Tmp21 [36276] [362760000]
Tmp20 [36276] [800000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp24 [910519] [136582000000]
Tmp19 [800000] [160000000000]
Tmp10 [227597] [227597]
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp7 [6000000] [6000000]
π o_orderkey o_custkey o_totalprice
Tmp9 [227597] [1500000]
Tmp16 [800000] [800000]
Tmp18 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
π r_regionkey
Tmp1 [1] [5]
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_quantity
σ o_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5]
[25]
[10000]
[6000000]
customer
region
nation
supplier
lineitem
orders
partsupp
part

● represents materialized view node

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} 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, 0, 1, 0, 0, 0, 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, 50000, 800000, 200000, 160000000000, 1602400000 and 160240, respectively. So, the query processing cost of query Q1 is (2)*(5 + 1 + 25 + 25 + 10000 + 50000 + 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*(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*(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*(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.

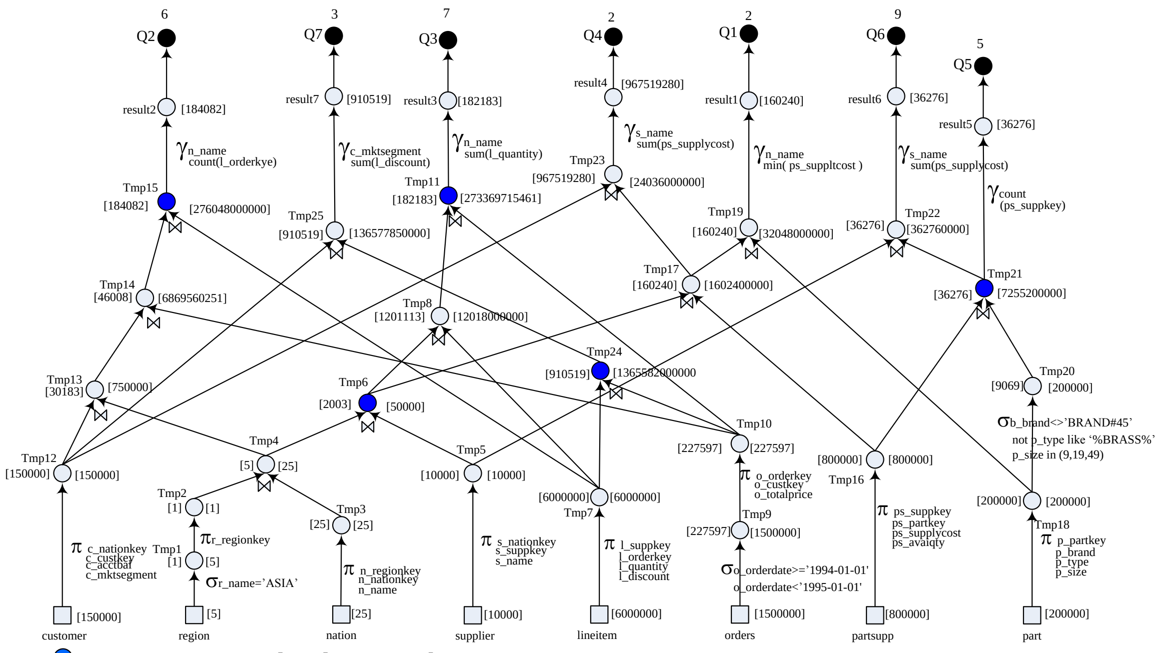
Table 4.6 The Maintenance Cost of the Cheapest MVPP

Materialized View	Number of Base Relations	Constructed from Nodes	Maintenance Cost
Tmp6	3	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6	180,168
Tmp11	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tm10, Tmp11	1,426,977,515,570
Tmp15	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15	1,414,630,939,520
Tmp19	2	Tmp16, Tmp18, Tmp19	320,002,000,000
Tmp24	2	Tmp7, Tmp9, Tmp10, Tmp24	2,731,179,455,194
Total materialized view maintenance cost			**5,892,790,090,452**

Table 4.7 The Query Processing Cost of the Cheapest MVPP

Query	Access from Node	Query Processing Cost
Query number 1 (Q1)	Tmp6, Tmp19, Tmp22, result1	3,206,724,486
Query number 2 (Q2)	Tmp15, result2	2,208,984
Query number 3 (Q3)	Tmp11, result3	2,550,562
Query number 4 (Q4)	Tmp6, Tmp16, Tmp17, Tmp12, Tmp23, result4	53,213,742,566
Query number 5 (Q5)	Tmp19, Tmp20, result5	8,181,380
Query number 6 (Q6)	Tmp19, Tmp20, Tmp21, Tmp5, result6	3,279,656,484
Query number 7 (Q7)	Tmp24, Tmp12, Tmp25, result7	409,739,463,114
Total query processing cost		**469,452,527,576**

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.



6
Q2
3
Q7
7
Q3
2
Q4
2
Q1
9
Q6
5
Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp25 [910519] [136577850000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp21 [36276] [7255200000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
Tmp24 [910519] [1365582000000]
Tmp20 [9069] [200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp10 [227597] [227597]
σ b_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
[800000] [800000] Tmp16
[200000] [200000] Tmp18
Tmp2 [1] [1]
Tmp3 [25] [25]
[6000000] [6000000] Tmp7
π o_orderkey o_custkey o_totalprice
Tmp9 [227597] [1500000]
π c_nationkey c_custkey c_acctbal c_mktsegment
Tmp1 [1] [5]
π r_regionkey
σ r_name='ASIA'
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
σ o_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part
● represents materialized view nodes

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

Materialized View	Number of Base Relations	Constructed from Nodes	Maintenance Cost
Tmp6	3	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6	180,168
Tmp11	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tm10, Tmp11	1,426,977,515,570
Tmp15	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15	1,414,630,939,520
Tmp21	2	Tmp16, Tmp18, Tmp20, Tmp21	14,512,800,000
Tmp24	2	Tmp7, Tmp9, Tmp10, Tmp24	2,731,179,455,194
Total materialized view maintenance cost			**5,587,300,890,452**

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

Query	Access from Node	Query Processing Cost
Query number 1 (Q1)	Tmp6, Tmp16, Tmp17, Tmp18, Tmp19, result1	67,303,124,486
Query number 2 (Q2)	Tmp15, result2	2,208,984
Query number 3 (Q3)	Tmp11, result3	2,550,562
Query number 4 (Q4)	Tmp6, Tmp16, Tmp17, Tmp12, Tmp23, result4	53,213,742,566
Query number 5 (Q5)	Tmp21, result5	362,760
Query number 6 (Q6)	Tmp21, Tmp5, Tmp22, result6	3,265,582,968
Query number 7 (Q7)	Tmp24, Tmp12, Tmp25, result7	409,739,463,114
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

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

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

	Cost of query Processing	**Cost of Maintenance**	**Total Cost**
All-virtual view	8,427,206,080,471	0	8,427,206,080,471
All-materialized views	1,940,978,234	7,686,779,440,303	7,688,720,418,537
2PO	533,527,035,440	5,587,300,890,452	6,120,827,925,892

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

	Total Cost	
	The Cheapest MVPP	**The Re-Optimized MVPP**
All-virtual view	10,821,545,680,471	8,427,206,080,471
All-materialized views	9,092,207,418,537	7,688,720,418,537
2PO	6,362,242,618,028	6,120,827,925,892

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

```
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;
```

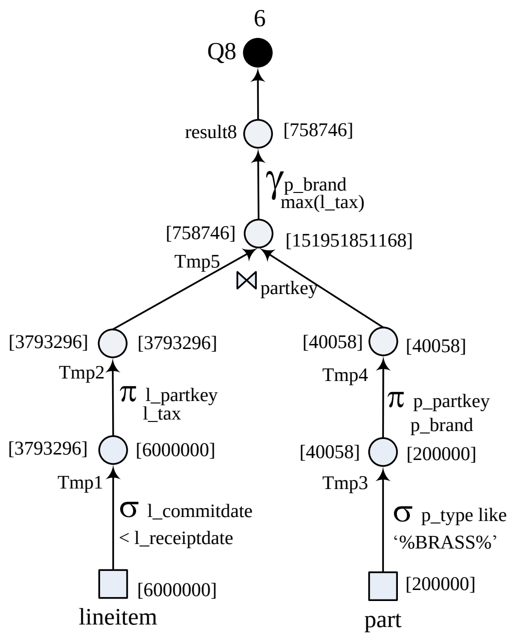


Figure 4.24 Relational Algebra Query Tree of Query Q8

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;
```

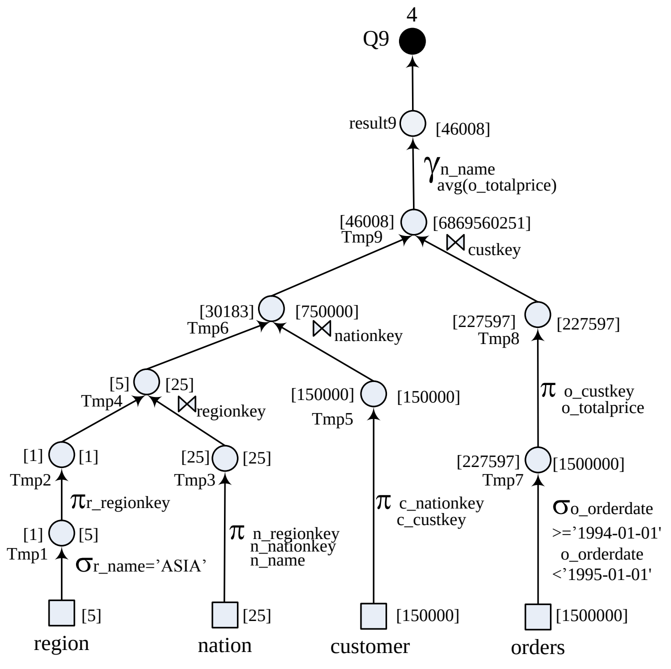


Figure 4.25 Relational Algebra Query Tree of Query Q9

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;
```

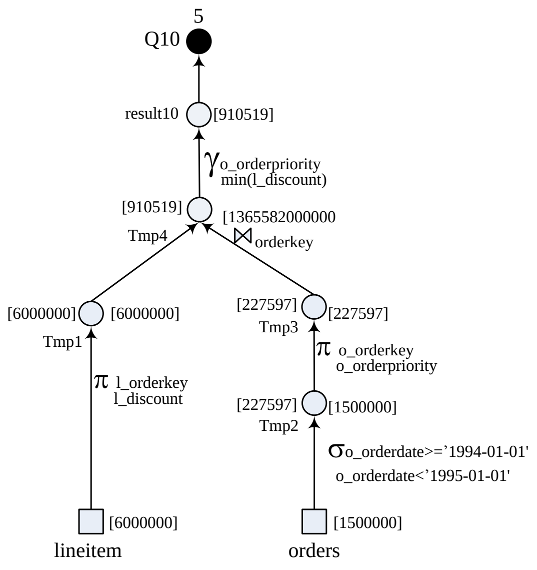


Figure 4.26 Relational Algebra Query Tree of Query Q10

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;
```

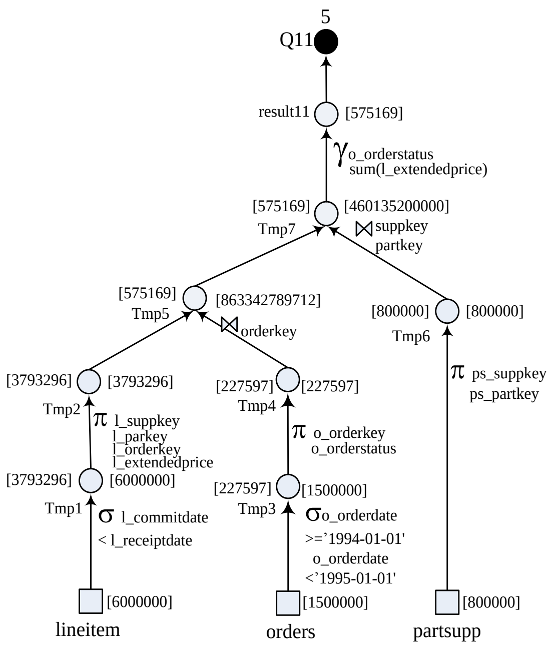


Figure 4.27 Relational Algebra Query Tree of Query Q11

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

```
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;
```

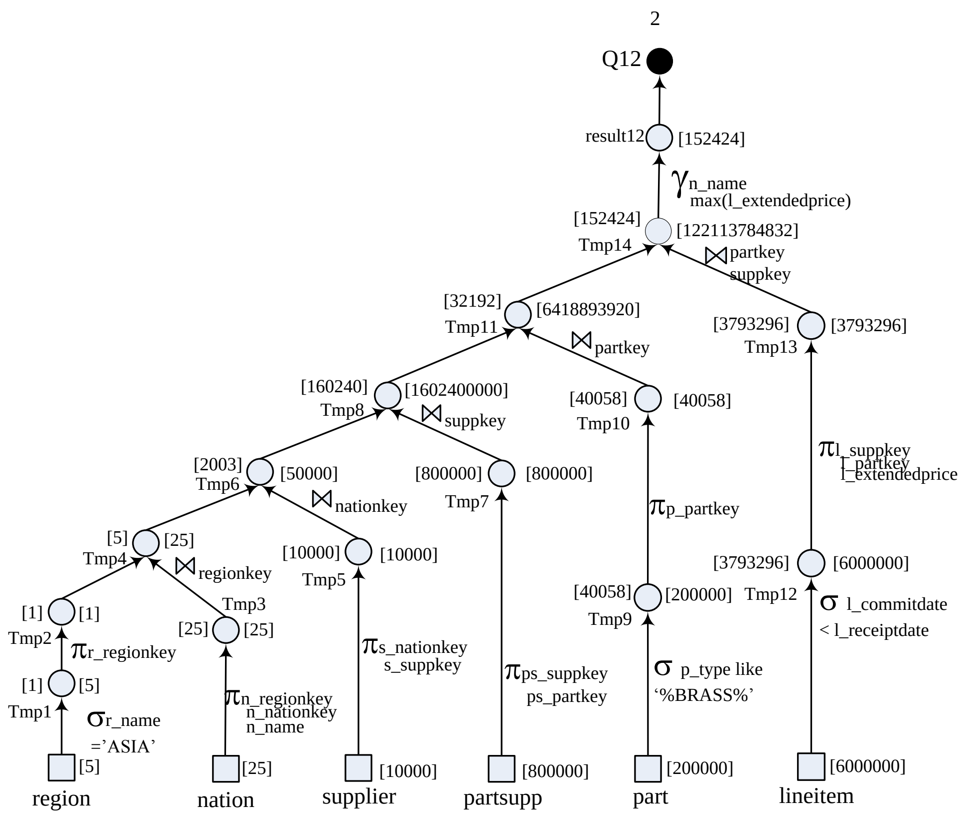


Figure 4.28 Relational Algebra Query Tree of Query Q12

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

```
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;
```

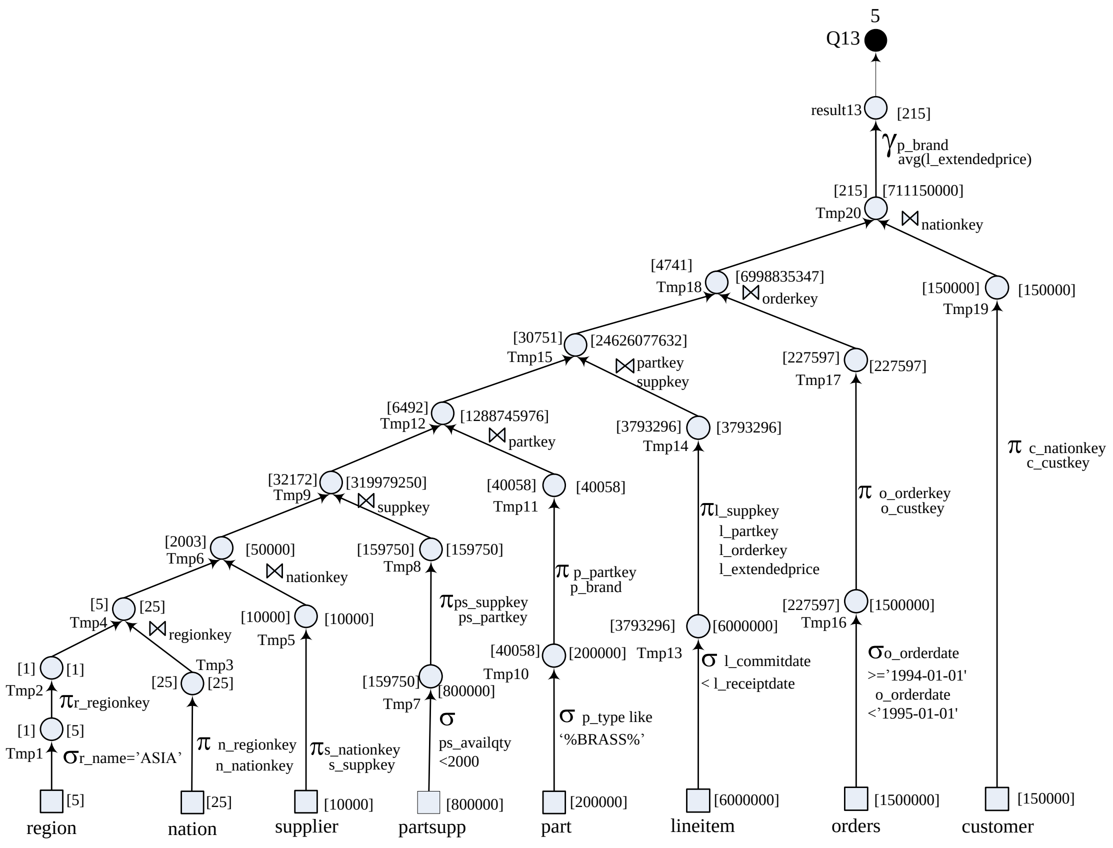


Figure 4.29 Relational Algebra Query Tree of Query Q13

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.

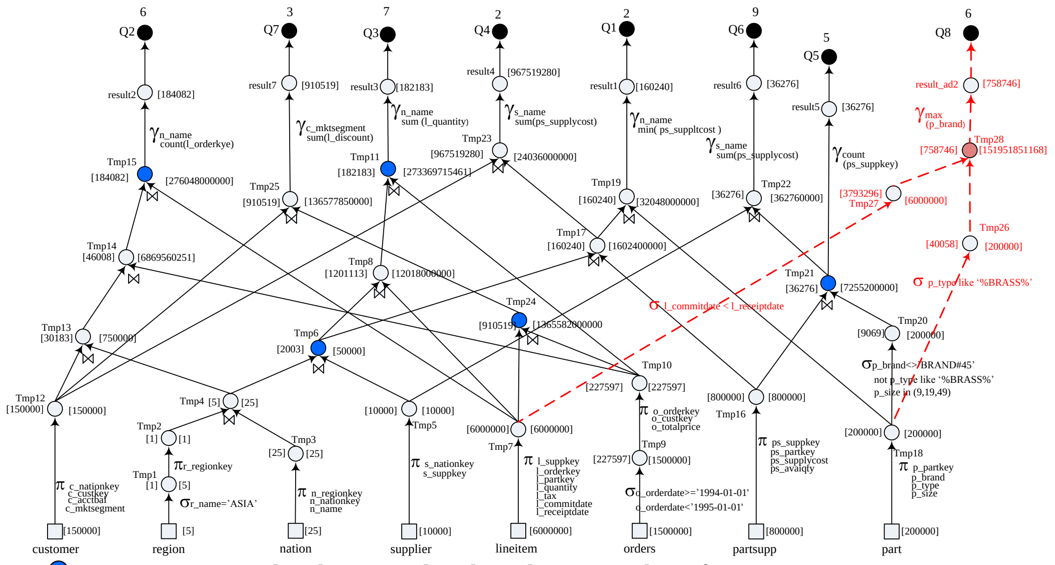
4.6.1.2 Analysis Result of the Nothing in Common Data Set

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.

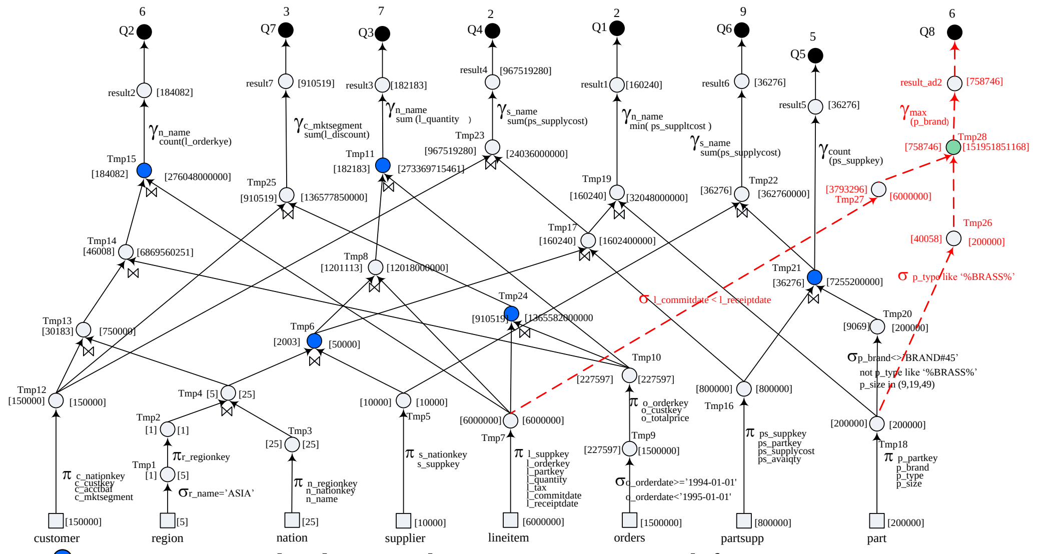
To evaluate the performance of our dynamic approach by the static approach is performed on the set of all queries which includes Q1 to Q7 and Q8. Figure 4.31 shows the re-optimized MVPP generated by static approach for Q1 to Q7 and Q8. The MVPP structure in Figure 4.31 provides the same structure as 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.



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

◐ represents new materialized view node selected in dynamic phase

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



● represents materialized view node same as static approach for Q1-Q7

◐ represents additional materialized view node selected by static approach

Figure 4.31 The Re-Optimized MVPP for Q1-Q7, and Q8 by Static Approach

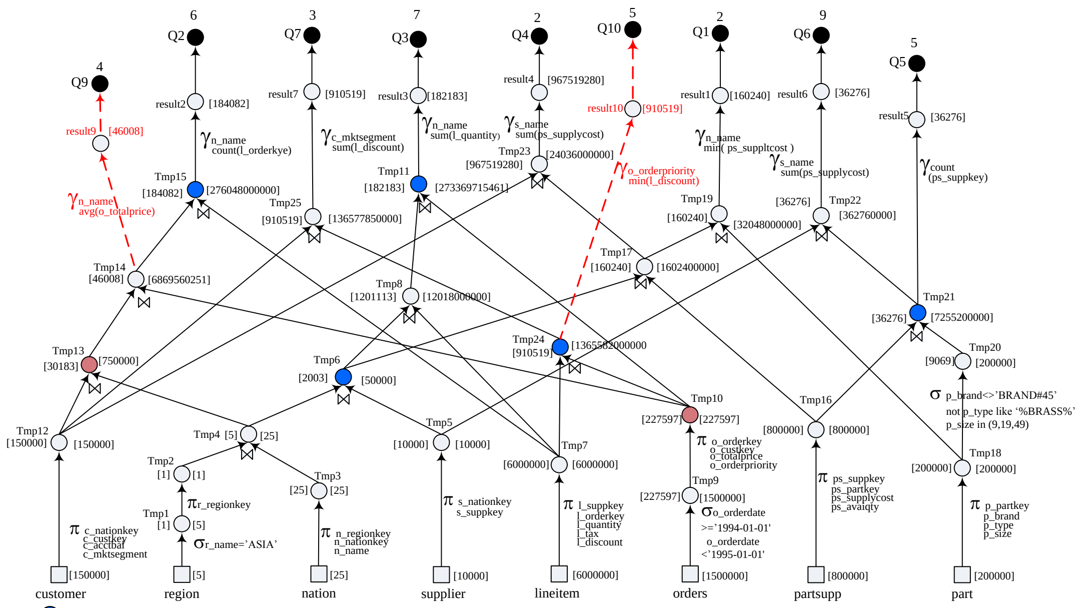
Table 4.13 The Comparison of the Result from the Static Approach and the Dynamic Approach for the Nothing in Common Data Set

Approach	**Number of Nodes to be Selected**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	28	533,536,140,392	5,891,229,392,788	6,424,765,533,180
Dynamic	3	533,536,140,392	5,891,229,392,788	6,424,765,533,180

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:

$$w(v) = \sum_{q \in O_v} \left\{ f_q(q) * \left(C_a^q(v) \right) \right\} - \sum_{r \in I_v} \left\{ f_u(r) * \left(C_m^r(v) \right) \right\}$$

w(v)	denotes weight of node
O_v	denotes the queries which use view *v*.
C_a^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_m^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 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 + 1500000 + 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

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

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 Table4.14, 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

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

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.

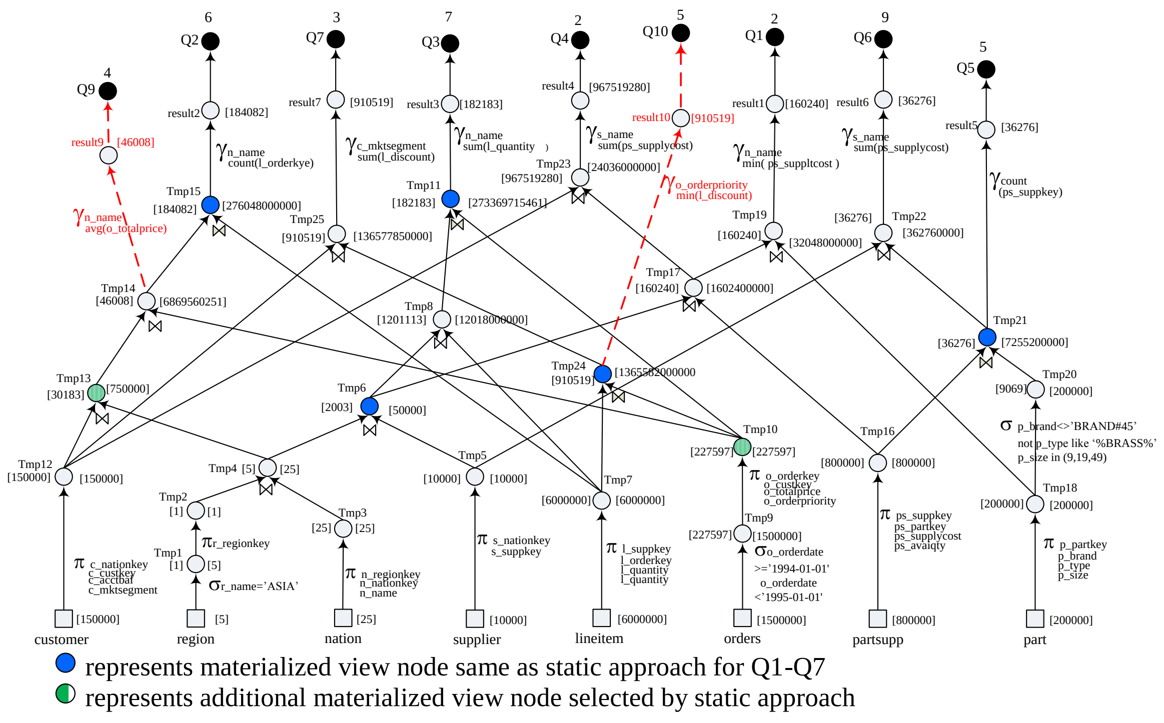


Figure 4.33 The Re-Optimized MVPP for Q1-Q7, and Q9-Q10 by Static Approach

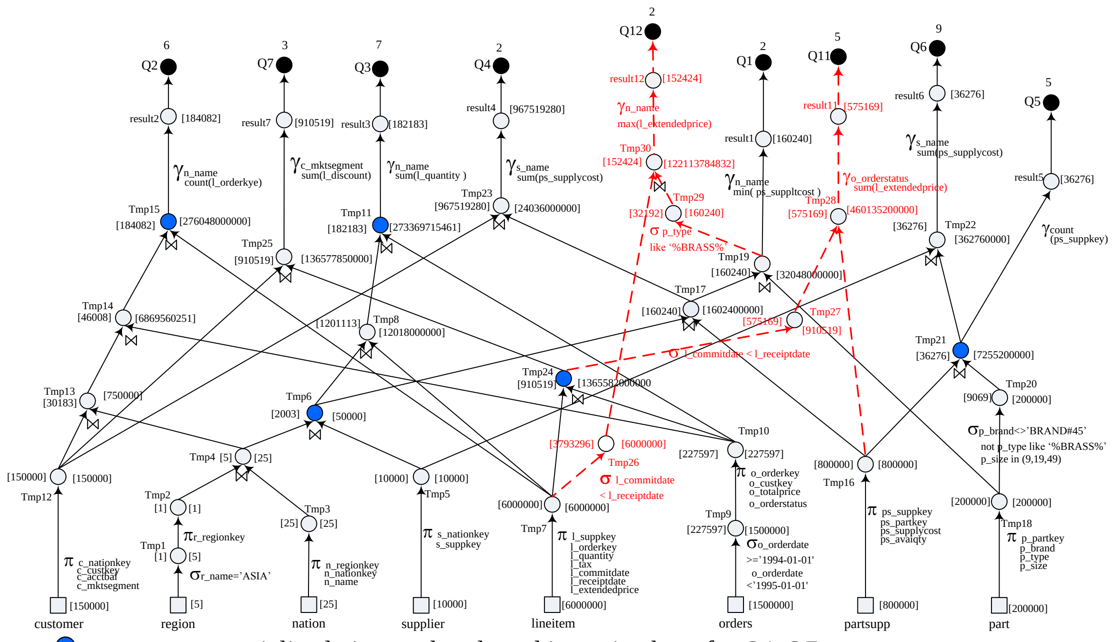
Table 4.16 The Comparison of the Result from the Static Approach and the Dynamic Approach for the Subsumption Data Set

Approach	**Number of Nodes to be Selected**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	25	561,015,596,786	5,587,305,318,217	6,148,320,915,003
Dynamic	10	561,015,596,786	5,587,305,318,217	6,148,320,915,003

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 first search to find the intermediate nodes used to construct Q11 and Q12. The intermediate nodes used to construct Q11 are {Tmp28, Tmp16, Tmp27, Tmp24, 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) * (1602400000) – (4) * (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

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

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

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

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.

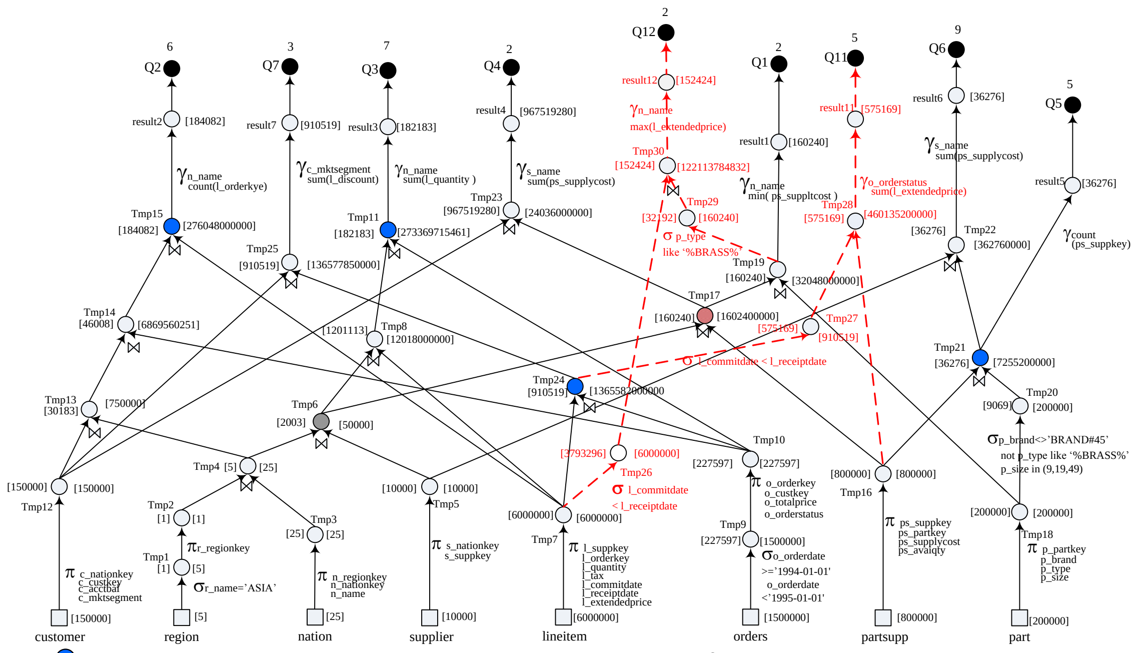
Tmp17, its ancestor nodes are Tmp19 and Tmp23. The weights of Tmp19 and Tmp23 are shown in Table 4.18. As weight of Tmp19 and Tmp23 is negative then Tmp19 and Tmp23 are not the indirectly affected nodes.

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.



● 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 4.35 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.

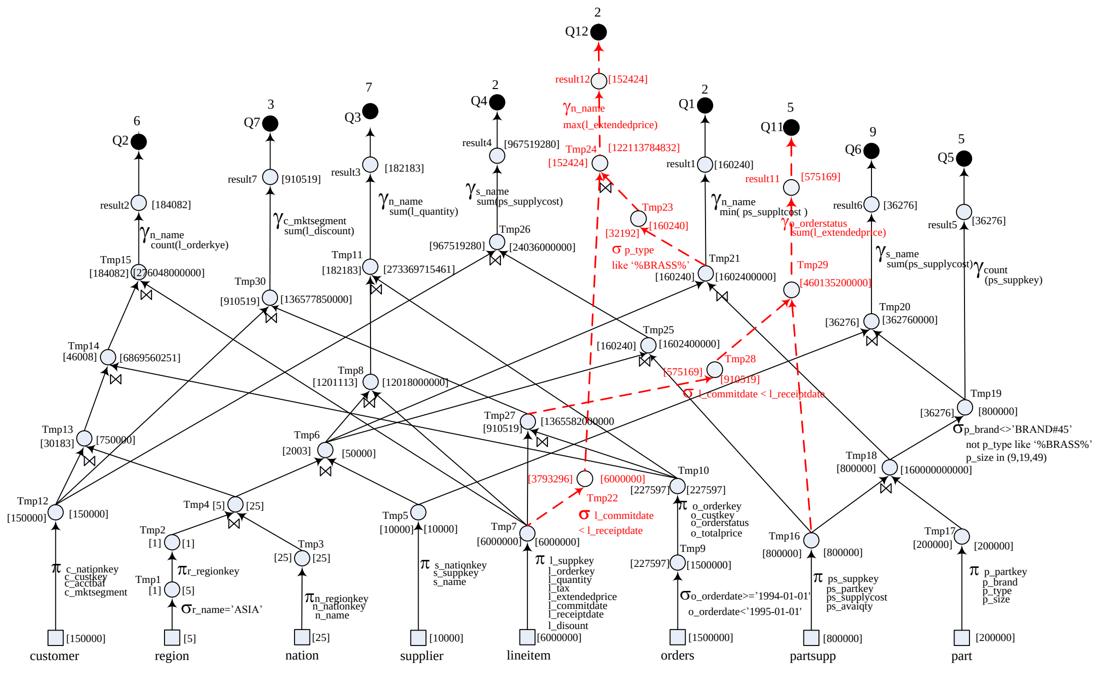


Figure 4.36 The Cheapest MVPP for Q1-Q7, and Q11-Q12 by Static Approach

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.

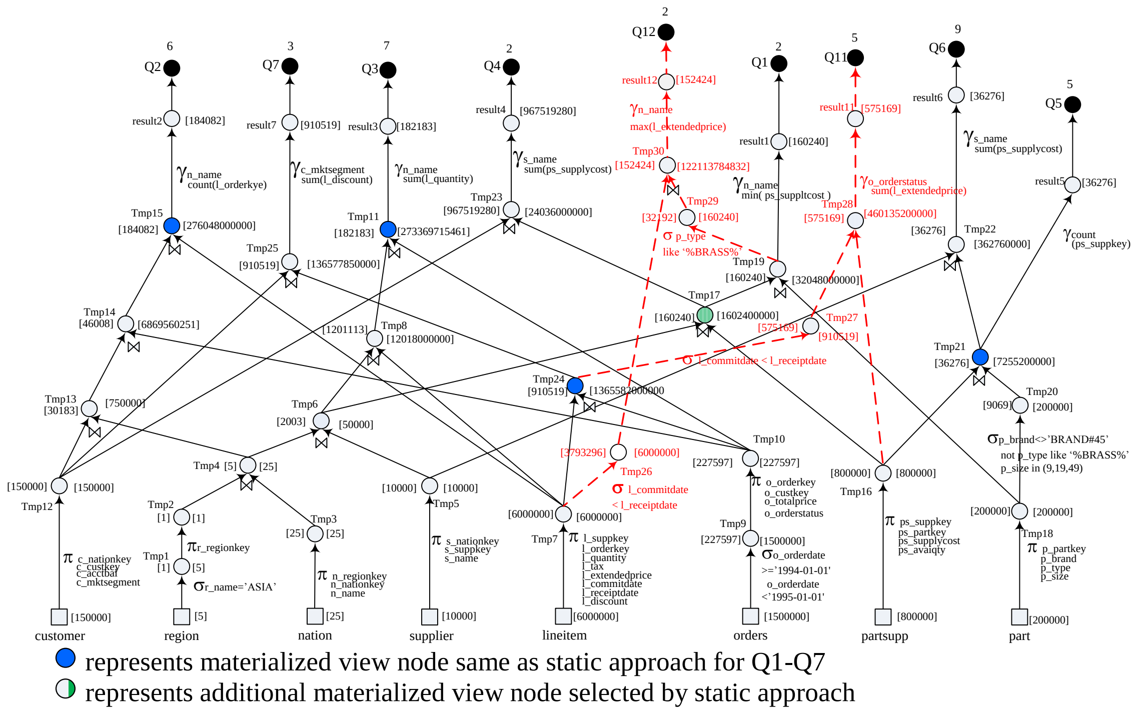


Figure 4.37 The Re-Optimized MVPP for Q1-Q7, and Q11-Q12 by Static Approach

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

Approach	Number of nodes to be selected	Cost of Query Processing	Cost of Maintenance	Total Cost
Static	30	3,352,056,780,640	5,593,713,750,508	8,945,770,531,148
Dynamic	15	3,352,056,780,640	5,593,713,750,508	8,945,770,531,148

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.

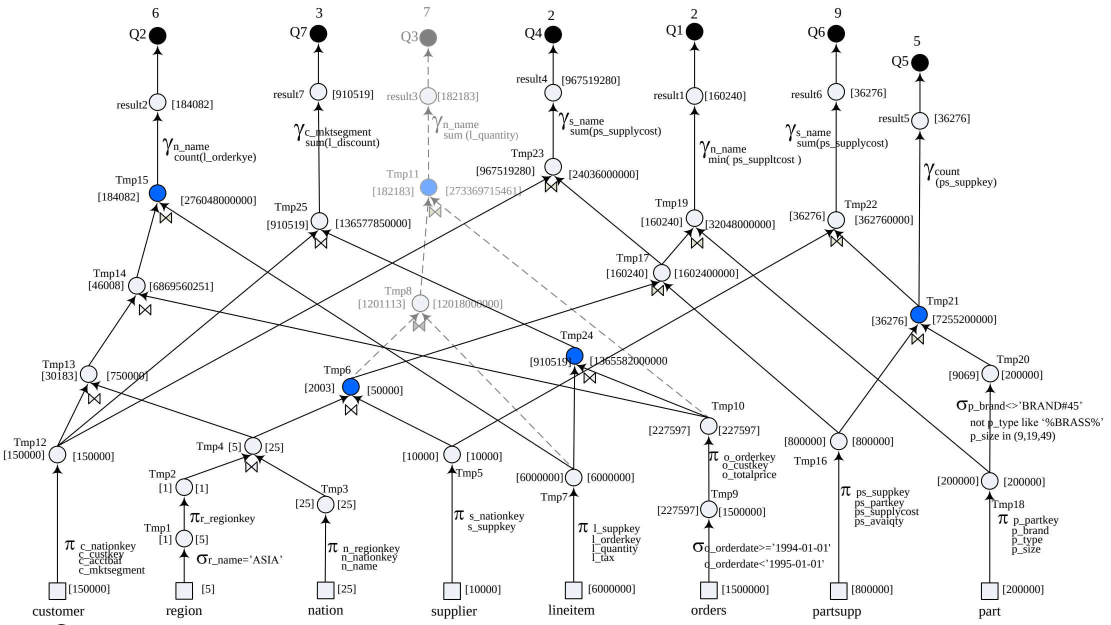

6
Q2
3
Q7
7
Q3
2
Q4
2
Q1
9
Q6
5
Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γn_name count(l_orderkye)
γc_mktsegment sum(l_discount)
γn_name sum (l_quantity)
γs_name sum(ps_supplycost)
γn_name min(ps_suppltcost)
γs_name sum(ps_supplycost)
γcount (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp25 [910519] [136577850000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
Tmp21 [36276] [7255200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp24 [910519] [1365582000000]
Tmp20 [9069] [200000]
σp_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp10 [227597] [227597]
Tmp12 [150000] [150000]
Tmp4 [5] [25]
[10000] [10000] Tmp5
[6000000] [6000000] Tmp7
π o_orderkey o_custkey o_totalprice
[800000] [800000] Tmp16
[200000] [200000] Tmp18
Tmp2 [1] [1]
Tmp3 [25] [25]
πr_regionkey
Tmp1 [1] [5]
σr_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey
π l_suppkey l_orderkey l_quantity l_tax
Tmp9 [227597] [1500000]
σo_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part
● represents materialized view node selected in static phase for Q1-Q7

Figure 4.38 The Re-Optimized MVPP Structure with Deleting Q3 for Dynamic and Static Approach

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

Existing Node	**Weight (*w (v)*)**
Tmp1	45
Tmp2	4
Tmp3	225
Tmp4	138
Tmp5	120,000
Tmp6	19,832
Tmp7	48,000,000
Tmp9	12,000,000
Tmp10	320,776

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

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

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

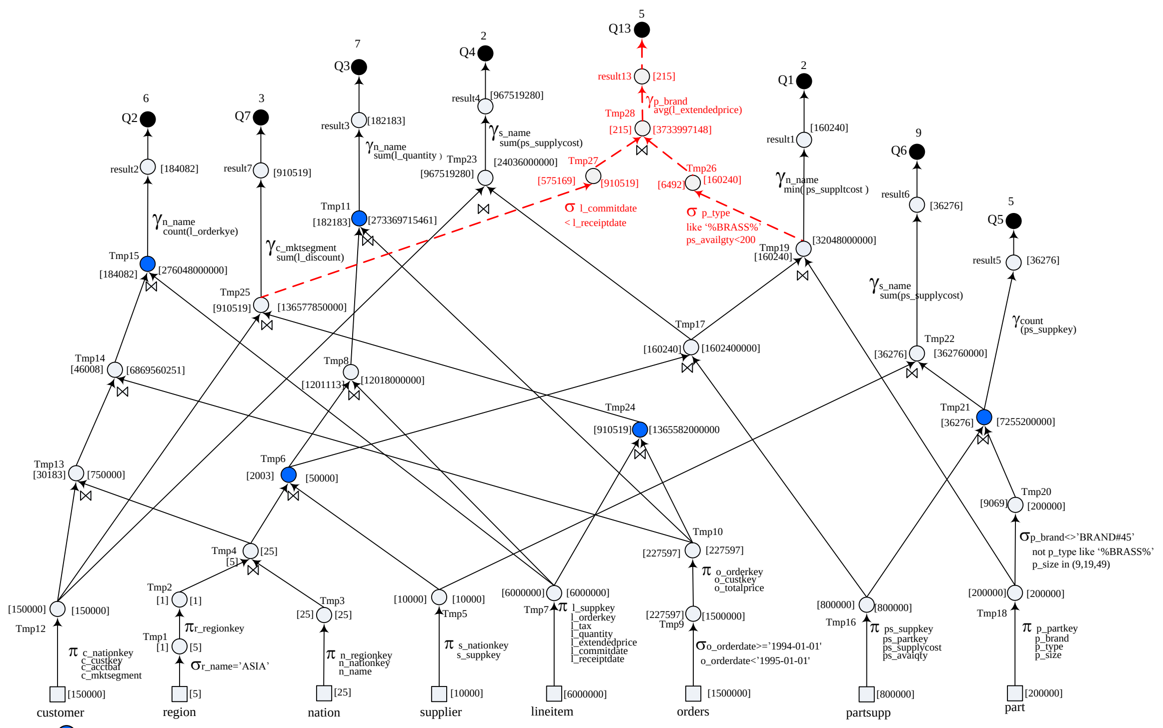
Approach	**Number of Nodes to be Selected**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	23	533,524,484,878	4,160,323,374,882	4,693,847,859,760
Dynamic	8	533,524,484,878	4,160,323,374,882	4,693,847,859,760

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)*, $\sum_{q \in O_v} \left\{ f_q(q) * \left(C_a^q(v) \right) \right\}$, 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.



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

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

Existing Node	**Weight (*w (v)*)**
Tmp1	105
Tmp2	16
Tmp3	525
Tmp4	438
Tmp5	240,000
Tmp6	619,832
Tmp7	120,000,000
Tmp9	30,000,000
Tmp10	3,051,940
Tmp12	2,250,000
Tnp16	17,600,000
Tmp17	8,008,559,776
Tmp18	4,000,000
Tmp19	56,078,699,720
Tmp24	8,193,476,544,806
Tmp25	-3,413,880,382,791

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.

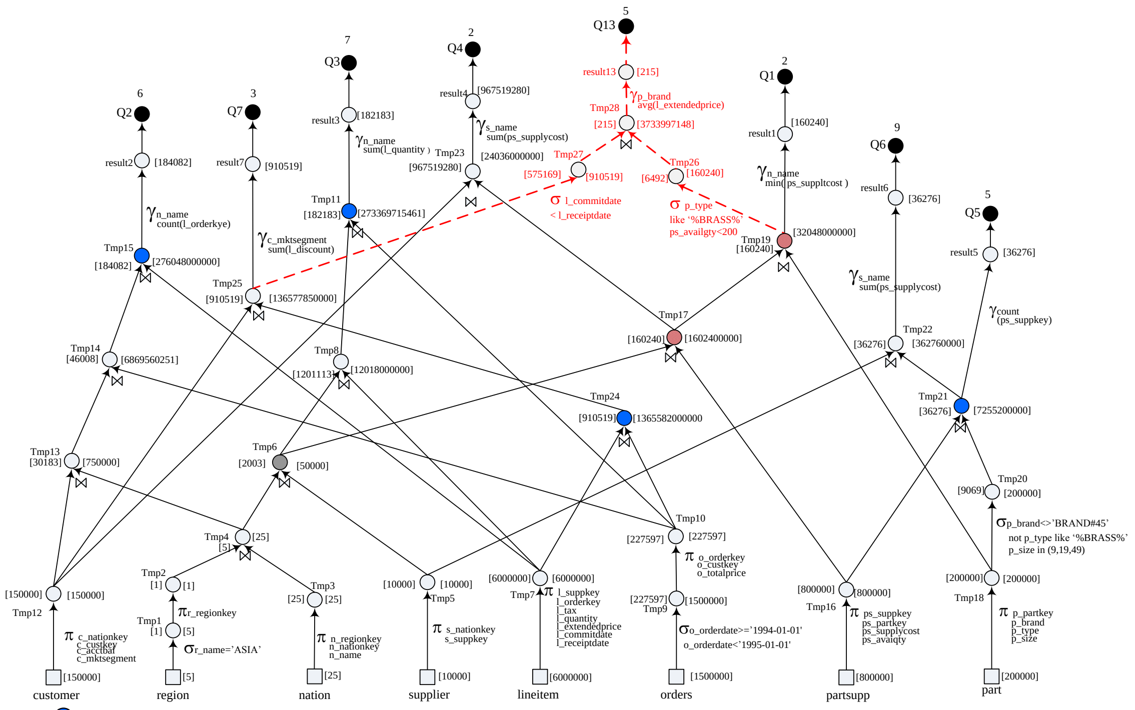
Table 4.24 The Weight of Ancestor Nodes for the Directly Affected Node of Q13

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

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.

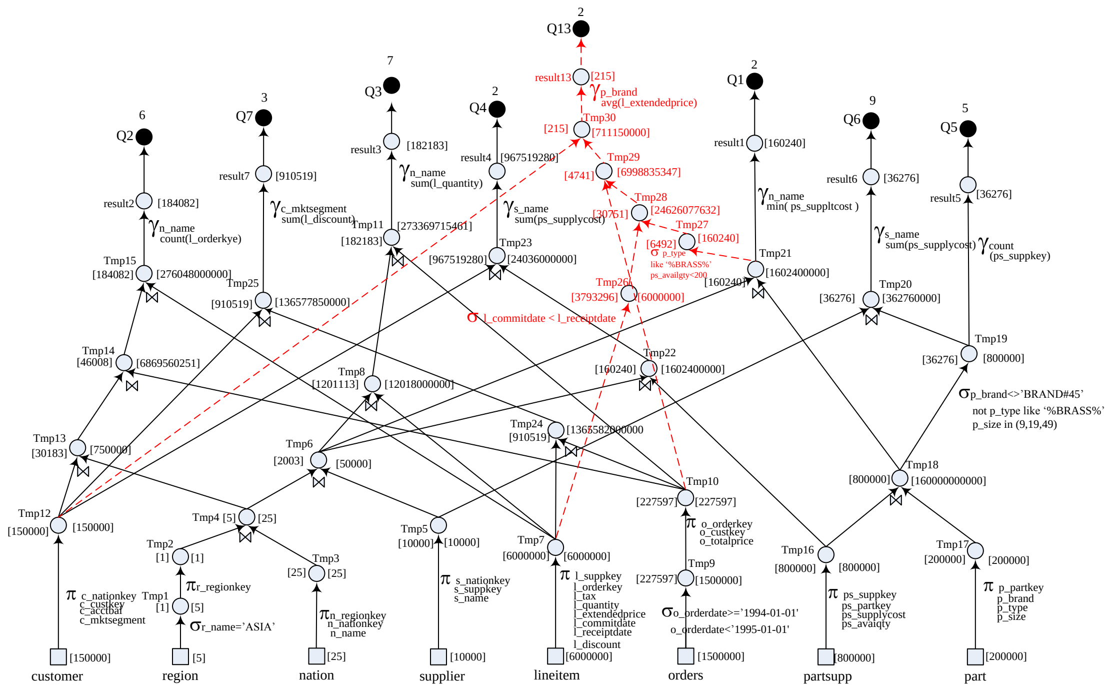
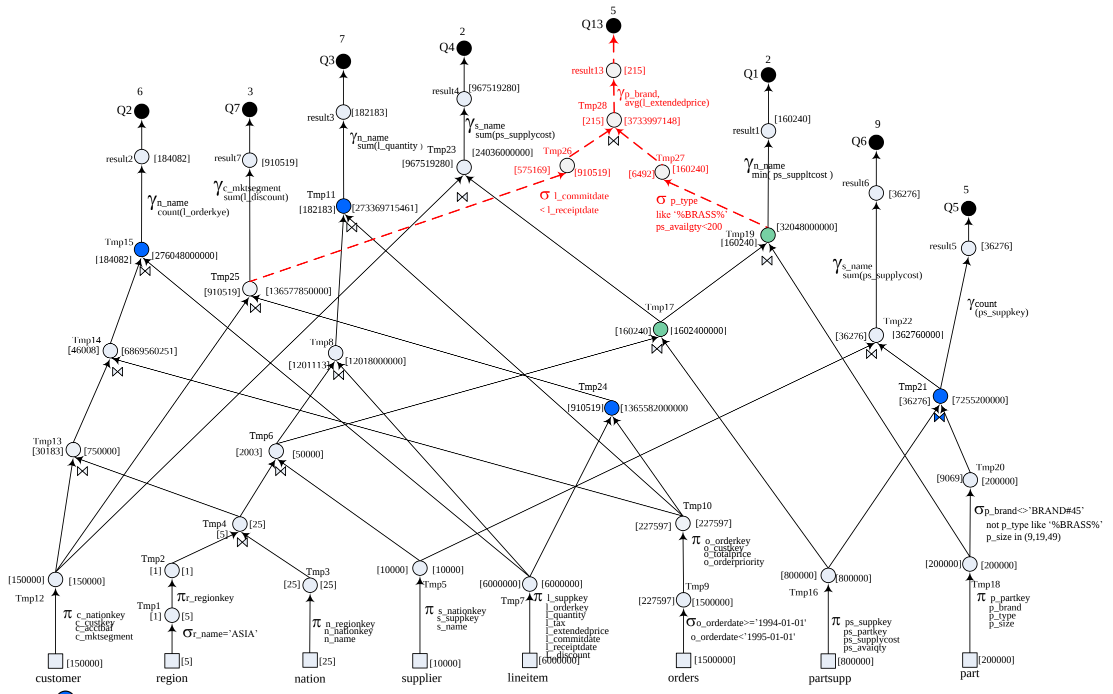


Figure 4.41 The Cheapest MVPP for Query Q1 to Q7, and Q13 by Static Approach



● represents materialized view node same as static approach for Q1-Q7

◐ represents additional materialized view node selected by static approach

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

Approach	**Number of Nodes to be Selected**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	28	1,164,589,162,793	5,593,713,750,508	6,758,302,913,301
Dynamic	14	1,164,589,162,793	5,593,713,750,508	6,758,302,913,301

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 the Existing 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.
```

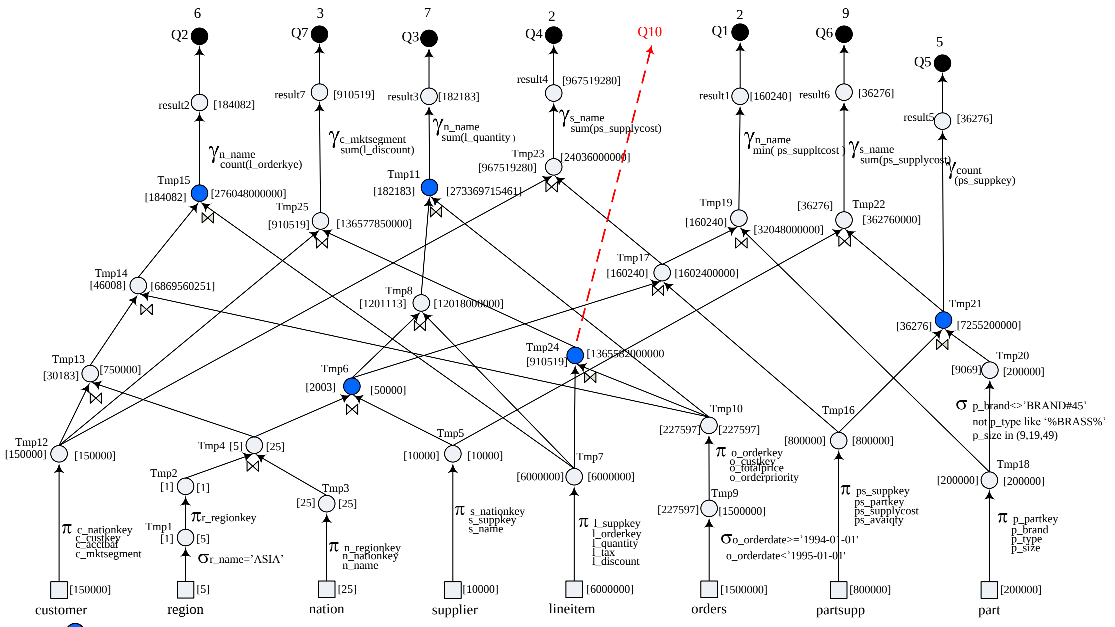
First, for every optimal query plan of the new requirements, select, project operations and aggregate functions are pushed up along the tree, if there is a join operation involved. Thereafter, we calculate the weight which is the frequency of executing the query multiplied with the query cost of queries 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

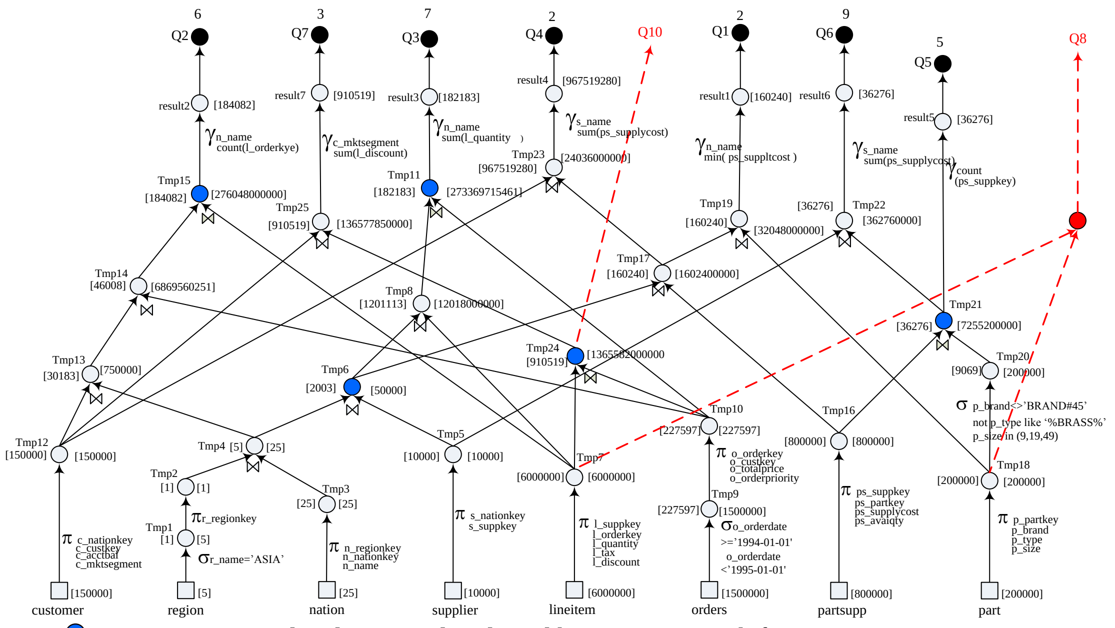
Query	**Frequency of Executing Query(f_q)**	**Query Cost**	**f_q * Query Cost**
Q8	6	758,746	4,552,476
Q9	4	46,008	184,032
Q10	5	910,519	4,552,595
Q11	5	575,169	2,875,845
Q12	2	152,424	304,848
Q13	5	215	1,075

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 ⋈ LINEITEM) that already available in the existing MVPP that is Tmp24. However, Tmp24 includes the select operation ($\sigma_{\text{o\_orderdate>='1994-01-01' and o\_orderdate <'1995-01-01'}}$ 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 ⋈ PARTSUPP}. The first dynamic MVPP, after Q11 is merged, is shown in Figure 4.43 (c).

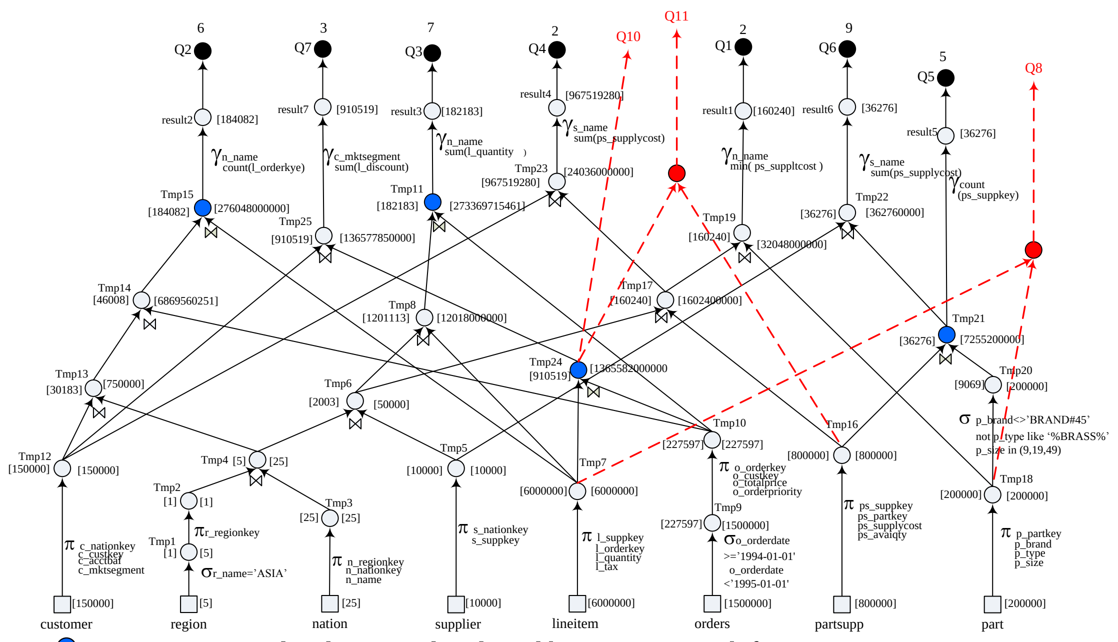

● represents materialized view node selected by static approach for Q1-Q7

(a) The first dynamic MVPP after Q10 is merged into the existing MVPP


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

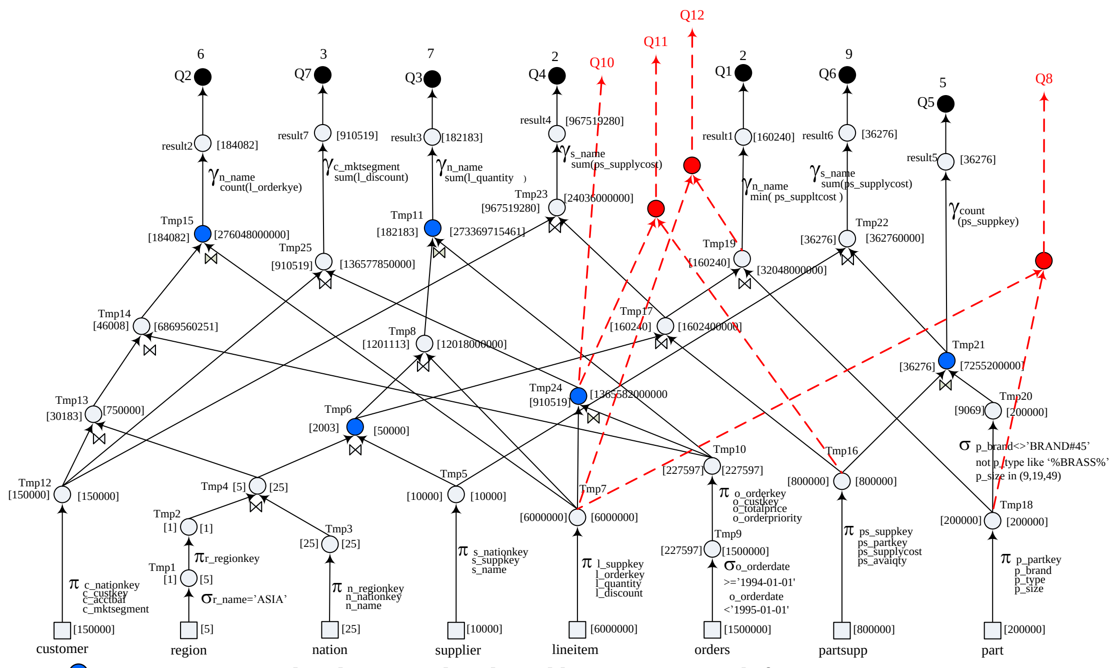
(b) The first dynamic MVPP after Q8 is merged into the existing MVPP

Figure 4.43 The Result of Merging Steps for the First Dynamic MVPP



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

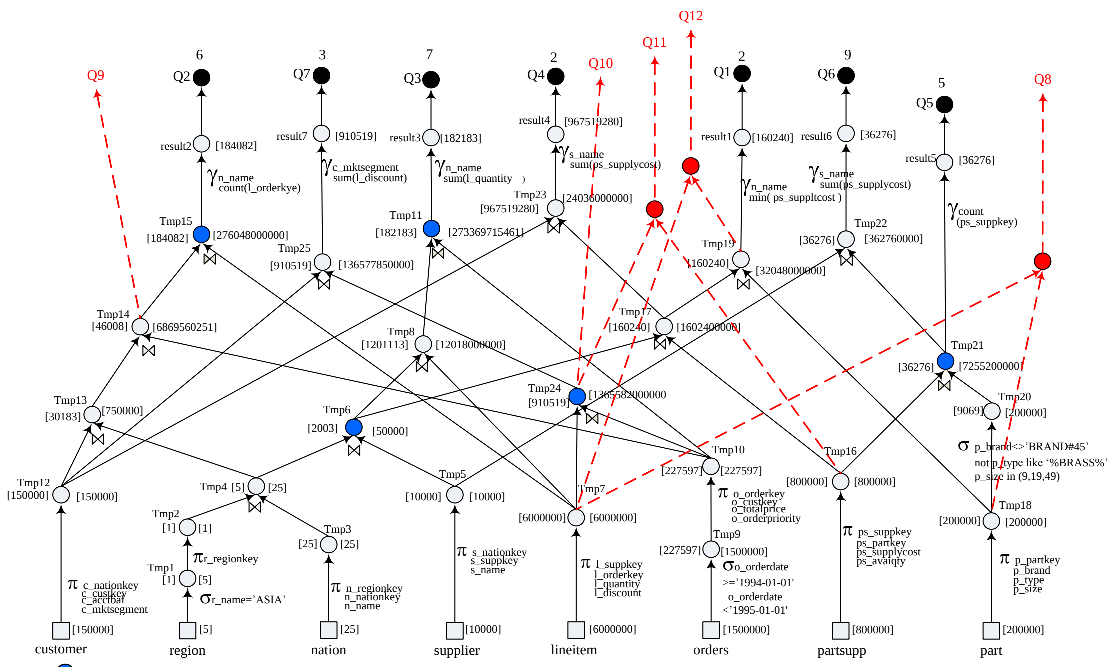
(c) The first dynamic MVPP after Q11 is merged into the existing MVPP



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

(d) The first dynamic MVPP after Q12 is merged into the existing MVPP

Figure 4.43 (Continued)



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

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

Figure 4.43 (Continued)

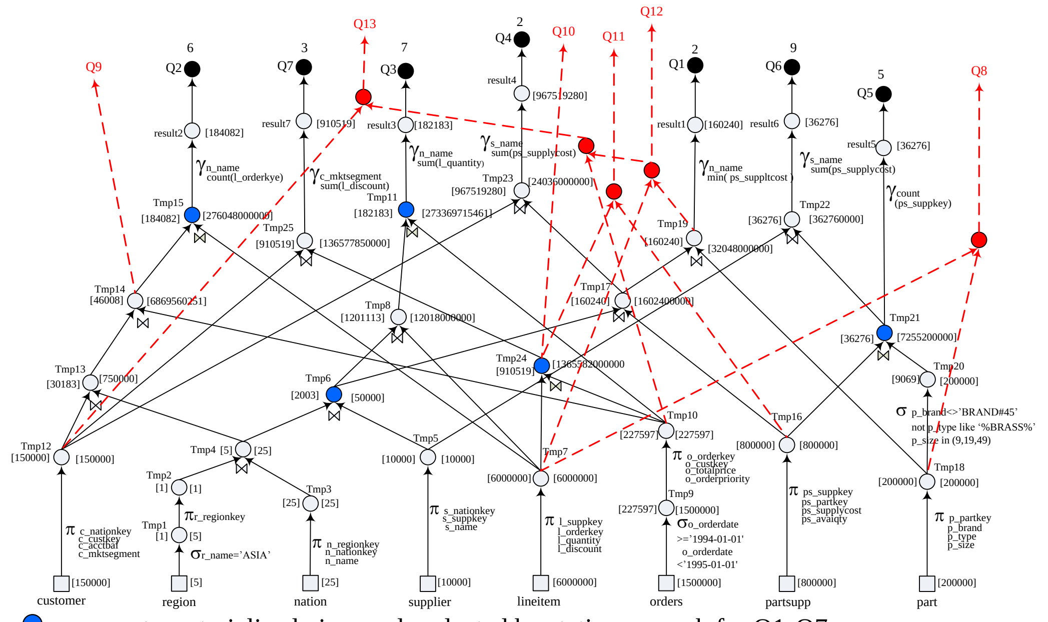
Next, when Q12 is merged, the existing conjunctive joined node is available for subtree of Q12 that is Tmp19. The remaining base relation is LINEITEM then the new node is introduced as a join operation between Tmp19 and LINEITEM, {Tmp19 ⋈ LINEITEM}. The first dynamic MVPP when Q12 is merged is shown 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

Plan	Execution Plan
Plan 1	(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) that is Tmp19, (LINEITEM ⋈ ORDERS ⋈ CUSTOMER) that is Tmp25. Its query processing cost is 869,827,064,180
Plan 2	{(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) ⋈ LINEITEM} that is Q12 and then join with ORDERS ⋈ CUSTOMER. Its query processing cost is 817,446,813,290

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.

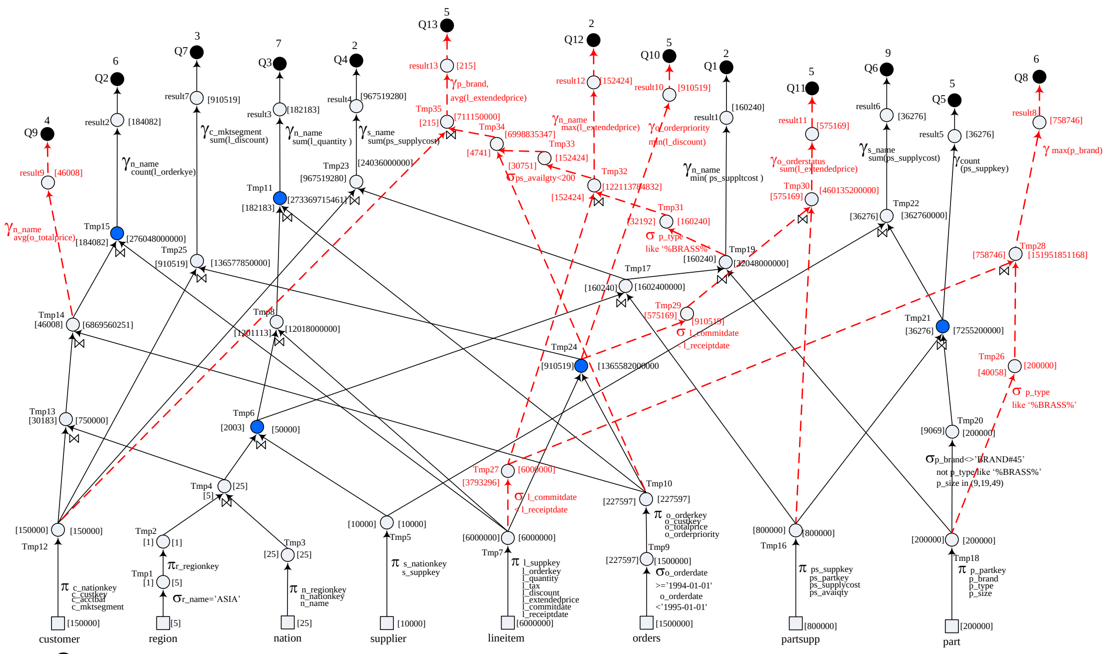


● represents materialized view node selected by static approach for Q1-Q7

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

Figure 4.43 (Continued)

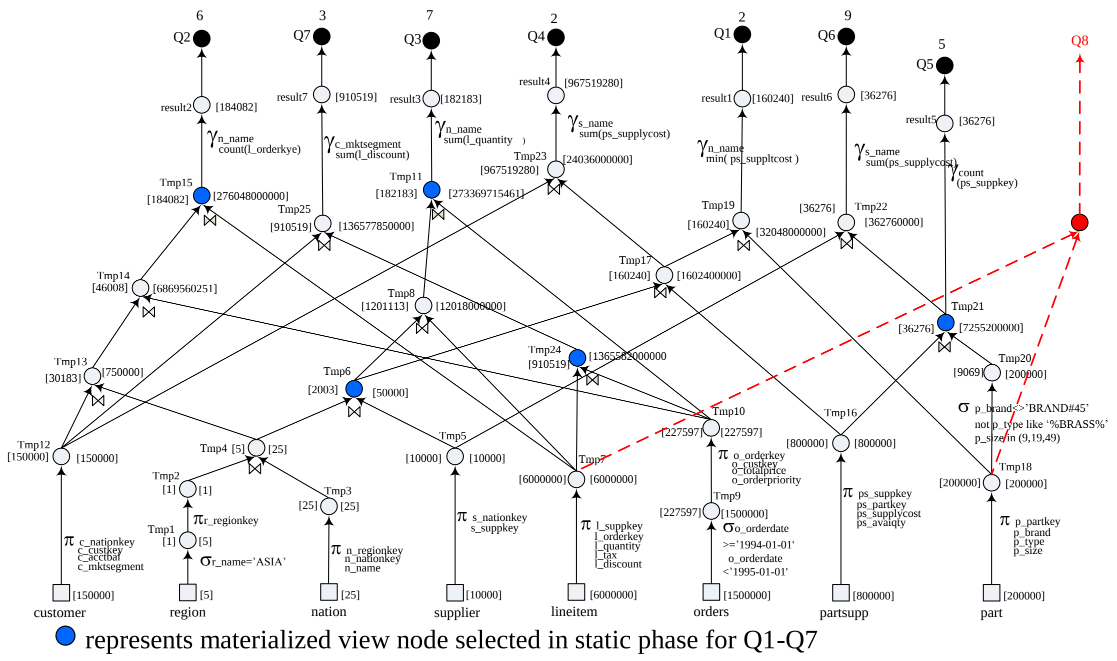
Next step of merging process, we push down the select, project and aggregate functions for the first dynamic MVPP as deep as possible to optimize MVPP. The first dynamic MVPP after optimized is shown in Figure 4.44



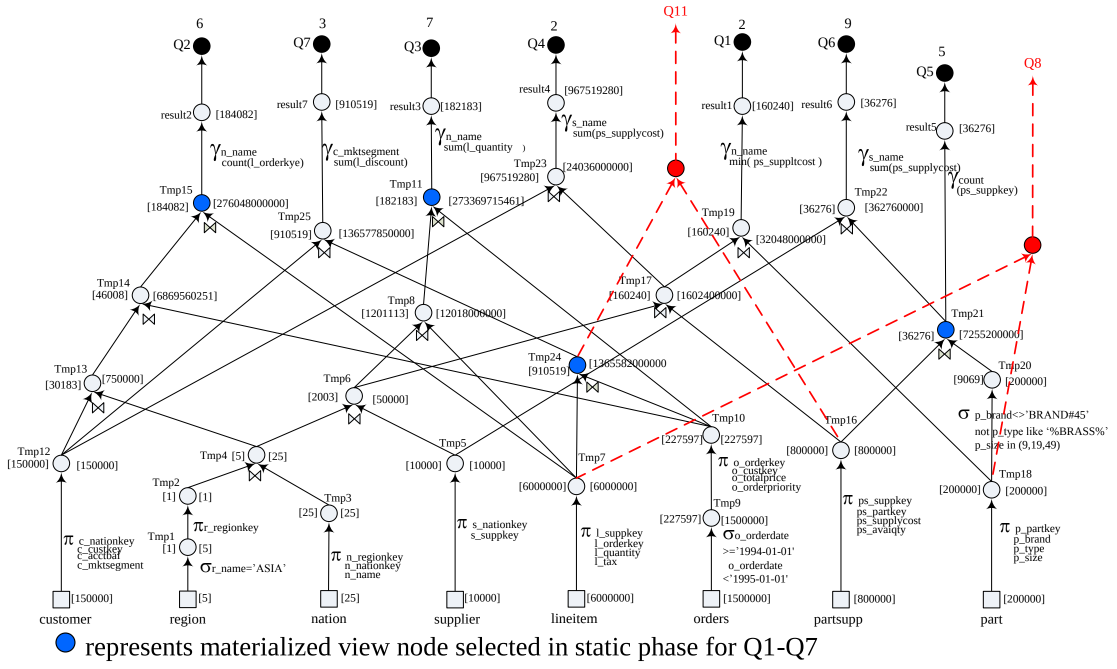
● represents materialized view node selected in static phase for Q1-Q7

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).

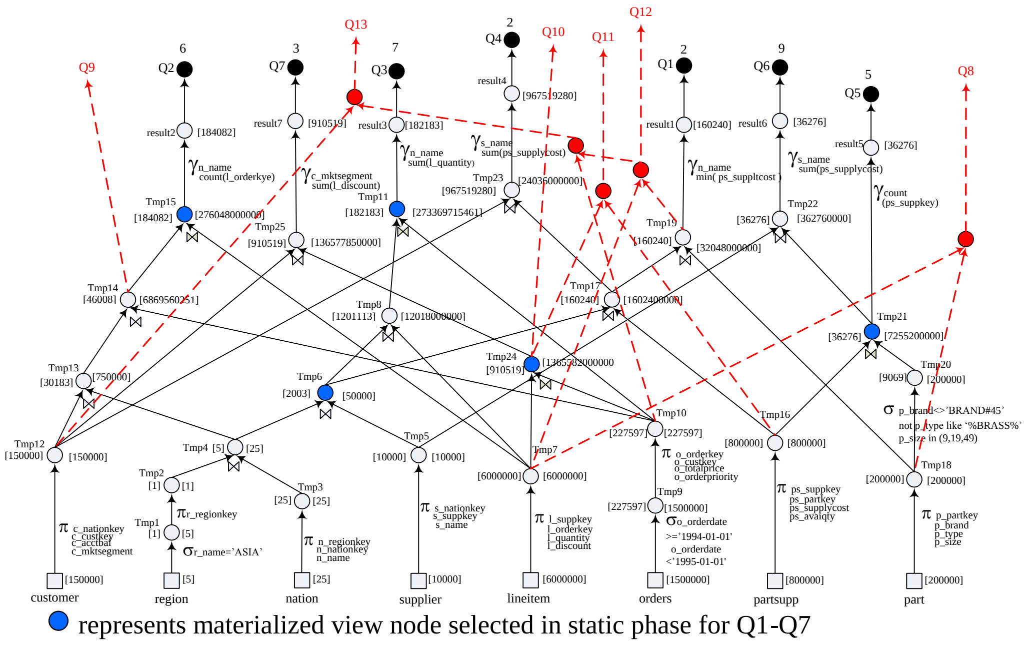


(a) The second dynamic MVPP after Q8 is merged into the existing MVPP



(b) The second dynamic MVPP after Q11 is merged into the existing MVPP

Figure 4.45 The Result of Merging Steps for the Second Dynamic MVPP


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

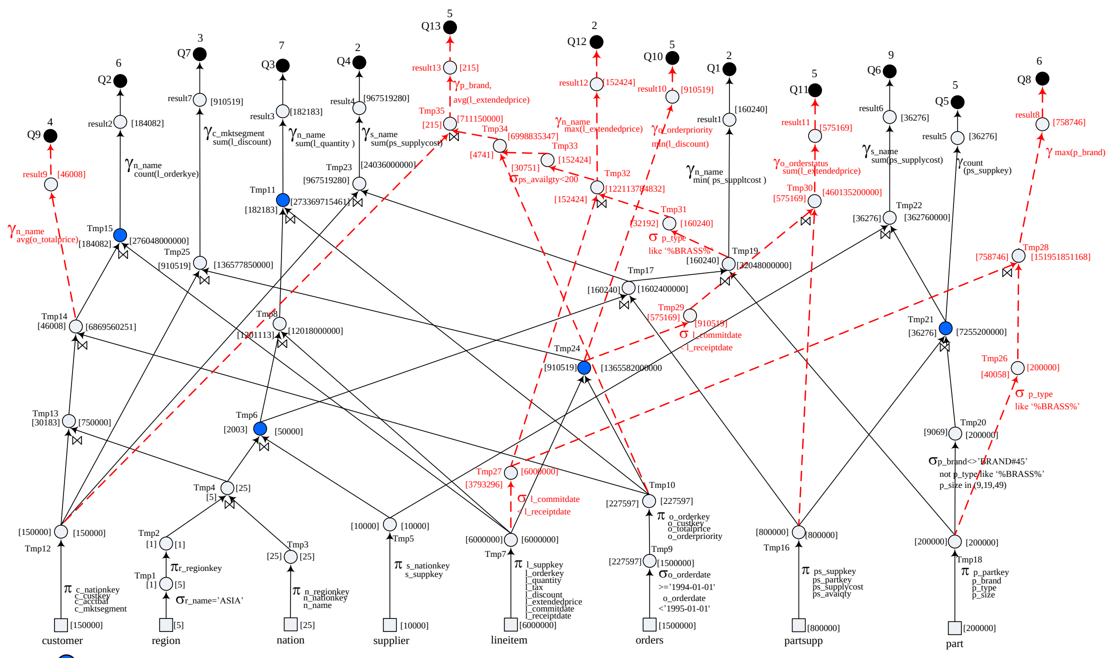
(c) The second dynamic MVPP after all new queries are merged

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.

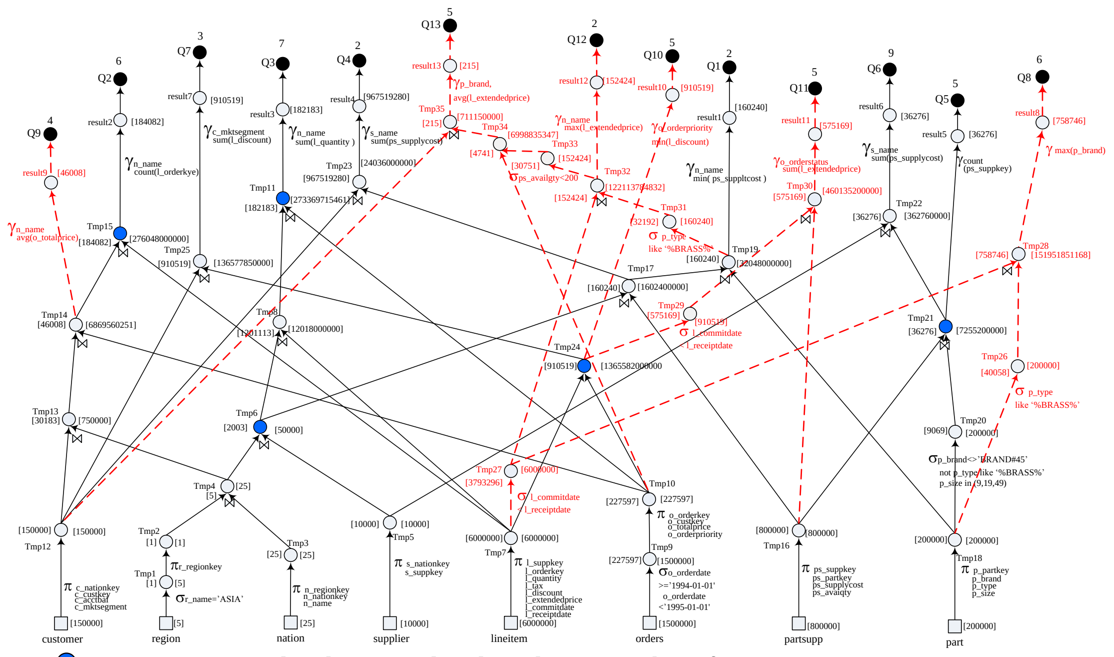
Next step of merging process, we push down select, project and aggregate functions for first dynamic MVPP as deep as possible to optimize MVPP. The second dynamic MVPP after optimized is shown in Figure 4.46.

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.



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

Figure 4.46 The Second Dynamic MVPP: the Queries in the List {Q8, Q11, Q12, Q9, Q13 and Q10}



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

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

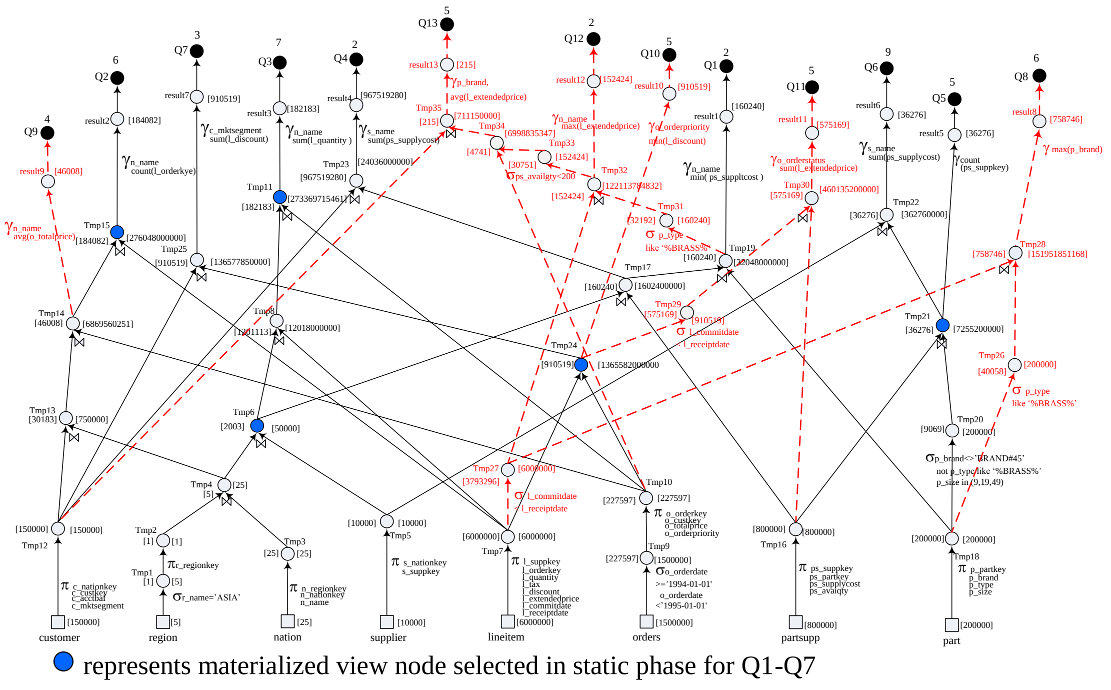


Figure 4.48 The Fourth Dynamic MVPP: the Queries in the List {Q12, Q9, Q13,Q10, Q8 and Q11}

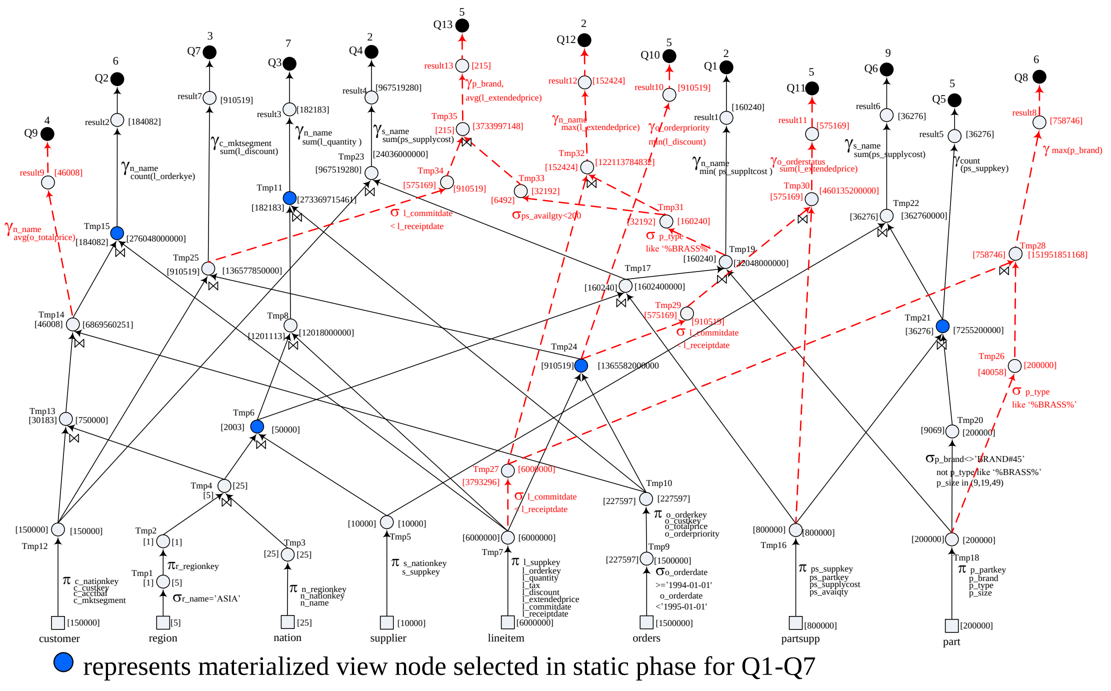
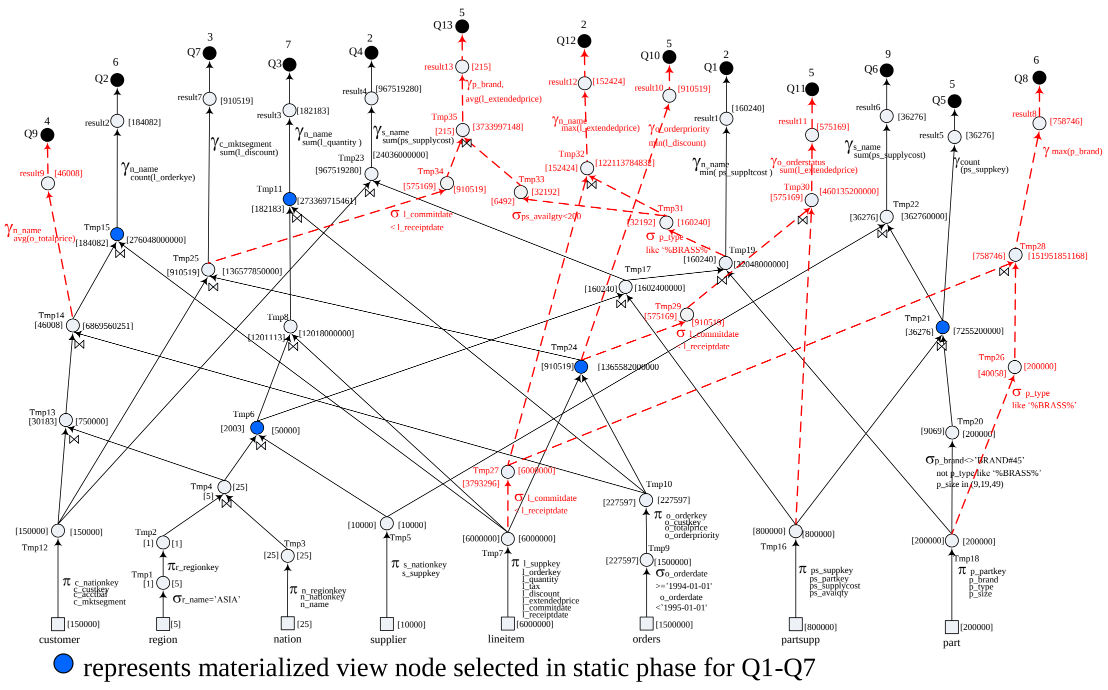


Figure 4.49 The Fifth Dynamic MVPP: the Queries in the List {Q9, Q13, Q10, Q8, Q11 and Q12}



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

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

Query	**1st MVPP**	**2nd MVPP**	**3rd MVPP**	**4th MVPP**	**5th MVPP**	**6th MVPP**
Q1	67,303,124,486	67,303,124,486	67,303,124,486	67,303,124,486	67,303,124,486	67,303,124,486
Q2	2,208,984	2,208,984	2,208,984	2,208,984	2,208,984	2,208,984
Q3	2,550,562	2,550,562	2,550,562	2,550,562	2,550,562	2,550,562
Q4	53,213,742,566	53,213,742,566	53,213,742,566	53,213,742,566	53,213,742,566	53,213,742,566
Q5	362,760	362,760	362,760	362,760	362,760	362,760
Q6	3,265,582,968	3,265,582,968	3,265,582,968	3,265,582,968	3,265,582,968	3,265,582,968
Q7	409,739,463,114	409,739,463,114	409,739,463,114	409,739,463,114	409,739,463,114	409,739,463,114
Q8	911,790,059,484	911,790,059,484	911,790,059,484	911,790,059,484	911,790,059,484	911,790,059,484
Q9	27,488,935,648	27,488,935,648	27,488,935,648	27,488,935,648	27,488,935,648	27,488,935,648
Q10	9,105,190	9,105,190	9,105,190	9,105,190	9,105,190	9,105,190
Q11	2,300,691,981,035	2,300,691,981,035	2,300,691,981,035	2,300,691,981,035	2,300,691,981,035	2,300,691,981,035
Q12	311,554,998,998	311,554,998,998	311,554,998,998	311,554,998,998	311,554,998,998	311,554,998,998
Q13	817,446,813,290	817,446,813,290	817,446,813,290	817,446,813,290	869,827,064,180	869,827,064,180
Total	**4,902,508,929,085**	**4,902,508,929,085**	**4,902,508,929,085**	**4,902,508,929,085**	**4,954,889,179,975**	**4,954,889,179,975**

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

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

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_{direct} , so skip to the next node in the list.

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

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

Tmp6, its ancestor nodes are Tmp8 and Tmp17, as Tmp17 already in M_{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 4.30 The Weight of Ancestor Node of Directly Affected Node of New Queries

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

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:

[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], [Tmp21,1], [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

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0}

The result by 2PO algorithm is {1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0 , 0, 0, 0, 0, 0} 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 4.31 The Query Processing Cost of Dynamic Approach for All Queries

Query	Access from Node	Query Processing Cost
Query number 1 (Q1)	Tmp19, result1	640,960
Query number 2 (Q2)	Tmp15, result2	2,208,984
Query number 3 (Q3)	Tmp11, result3	2,550,562
Query number 4 (Q4)	Tmp17, Tmp12, Tmp23, result4	50,007,659,040
Query number 5 (Q5)	Tmp21, result5	362,760
Query number 6 (Q6)	Tmp21, Tmp5, Tmp22, result6	3,265,582,968
Query number 7 (Q7)	Tmp24, Tmp12, Tmp25, result7	409,739,463,114
Query number 8 (Q8)	Tmp28, result8	9,104,952
Query number 9 (Q9)	Tmp13, result9	27,479,456,156
Query number 10 (Q10)	Tmp24, result10	9,105,190
Query number 11 (Q11)	Tmp24, Tmp29, Tmp16, Tmp30 and result11	2,300,691,981,035
Query number 12 (Q12)	Tmp19, Tmp31, Tmp27, Tmp32 and result12	244,236,102,064

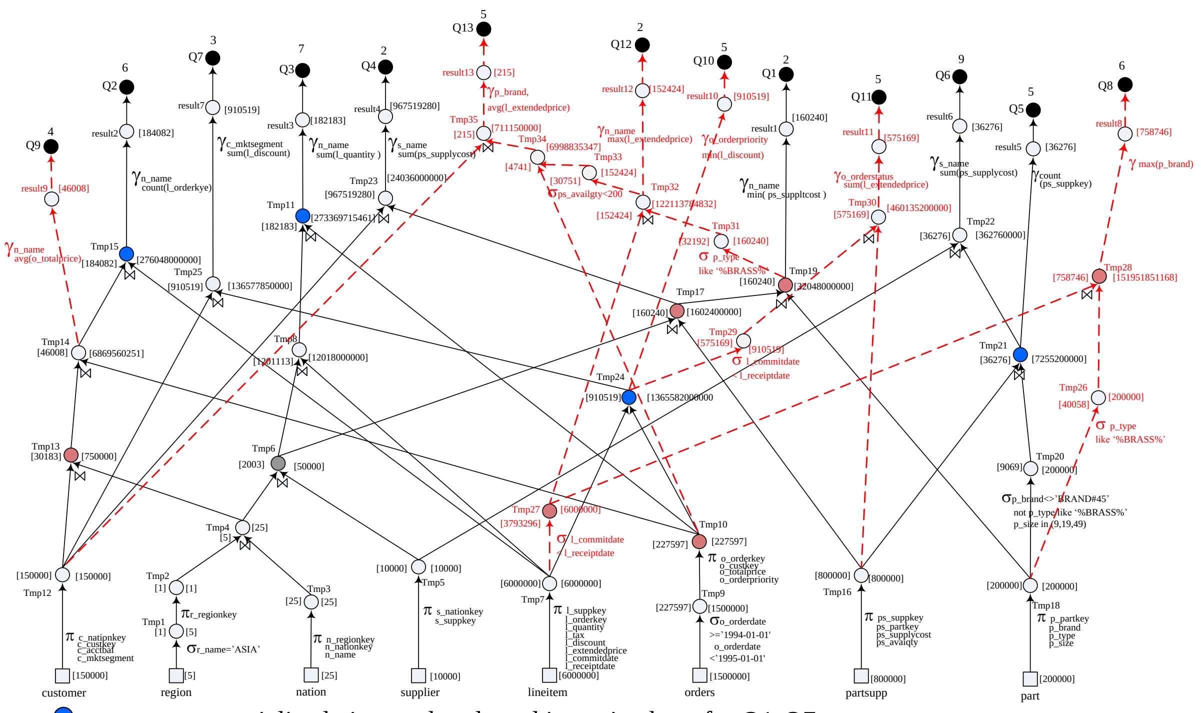
Table 4.31 (Continued)

Query	Access from Node	Query Processing Cost
Query number 13 (Q13)	Tmp19, Tmp31, Tmp27, Tmp32, Tmp33, Tmp10, Tmp34, Tmp35 and result13	649,149,570,955
	Total query processing cost	**3,684,593,788,740**

Table 4.32 The Maintenance Cost of Dynamic Approach for All Queries

Materialized View	Number of Base Relations	Constructed from Nodes	Maintenance Cost
Tmp10	1	Tmp9, Tmp10	1,727,597
Tmp11	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tm10, Tmp11	1,426,977,515,570
Tmp13	3	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13	2,700,168
Tmp15	5	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp7, Tmp15	1,414,630,939,520
Tmp17	4	Tmp1, Tmp2, Tmp3, Tmp4, Tmp6, Tmp16, Tmp17	6,413,040,224
Tmp19	5	Tmp1, Tmp2, Tmp3, Tmp4,Tmp6, Tmp16, Tmp17, Tmp18, Tmp19	168,257,300,280
Tmp21	2	Tmp16, Tmp18, Tmp20, Tmp21	14,512,800,000
Tmp24	2	Tmp7, Tmp9, Tmp10, Tmp24	2,731,179,455,194
Tmp27	1	Tmp7, Tmp27	12,000,000
Tmp28	2	Tmp7, Tmp27, Tmp18 ,Tmp28	303,928,502,336
		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.



● 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 4.51 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.

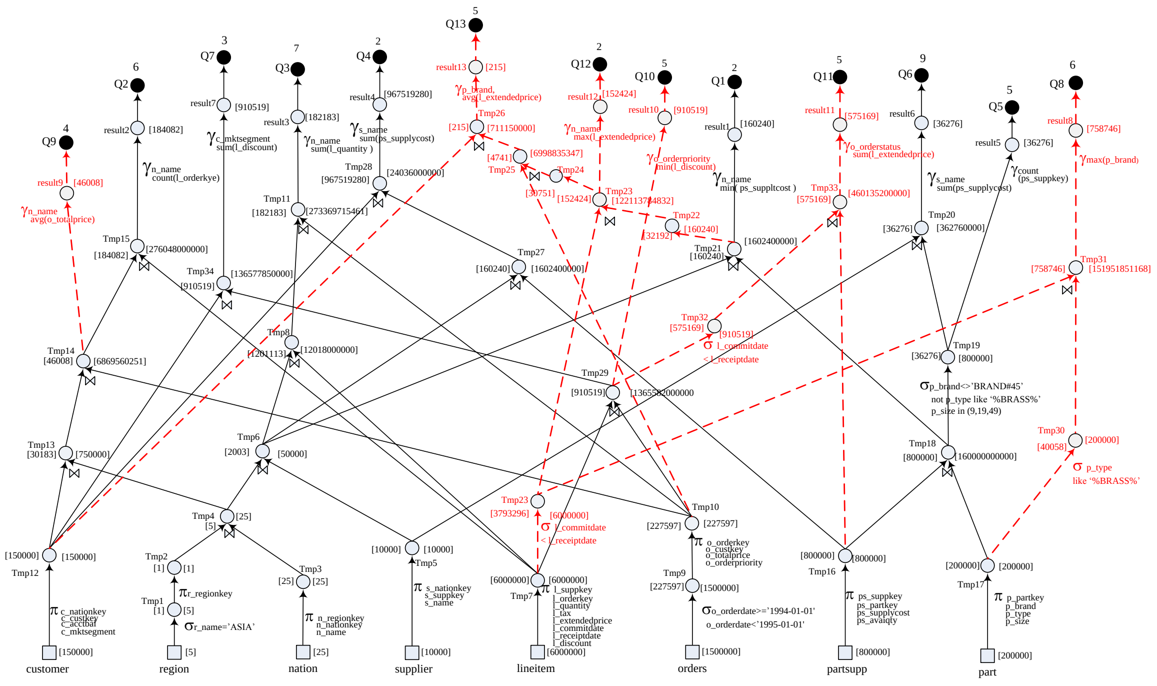
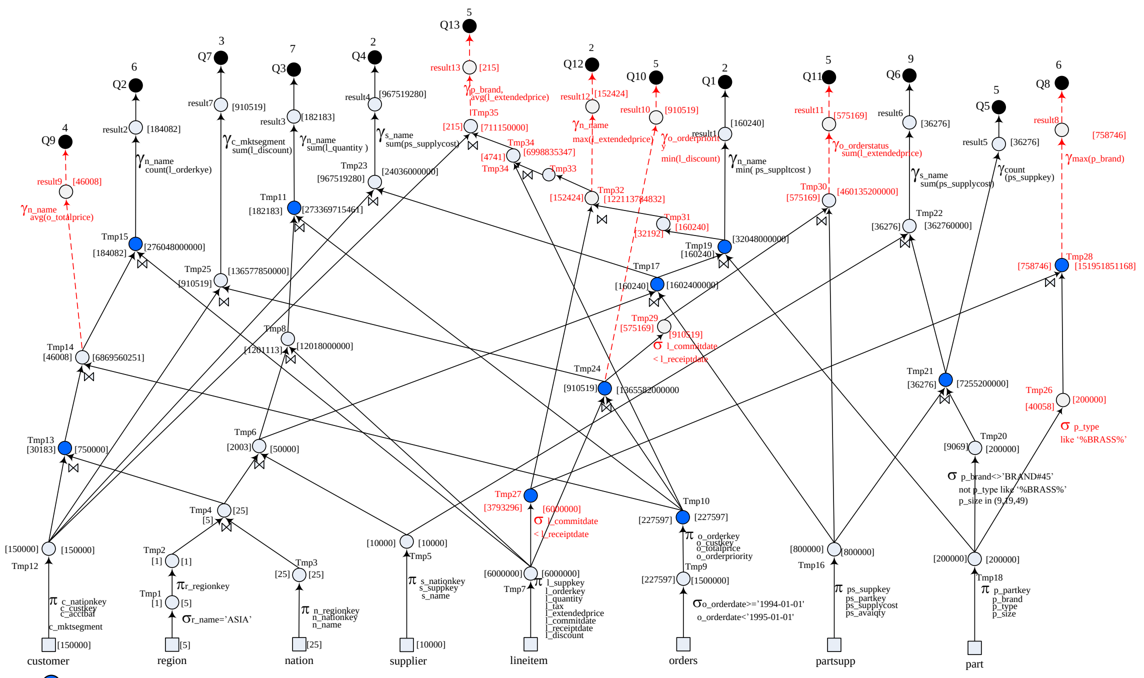


Figure 4.52 The Cheapest MVPP of Q1-Q13 by Static Approach

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.



● represents materialized view node selected by static approach for Q1-Q13

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.

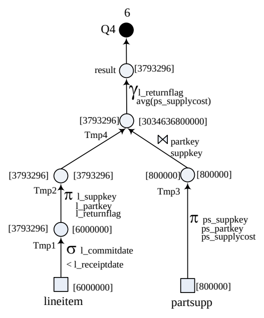
Table 4.33 The Comparison of the Result from the Static Approach and the Dynamic Approach for All Queries

Approach	Number of Nodes	Cost of Query Processing	Cost of Maintenance	Total Cost
Static	35	3,684,593,788,740	6,065,915,980,889	9,750,509,769,629
Dynamic	23	3,684,593,788,740	6,065,915,980,889	9,750,509,769,629

4.9 The Second Experiment for Dynamic Materialized View Selection

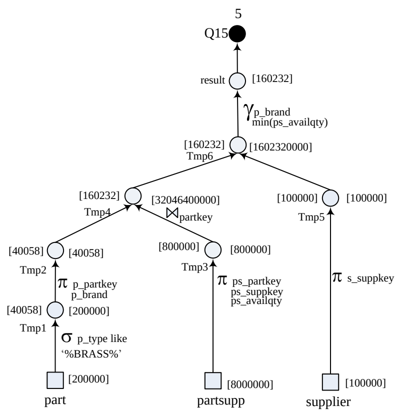
4.9.1 Static Materialized View Selection for the Initial Requirements

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.

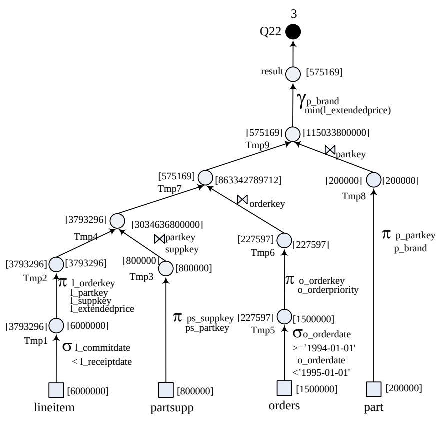


6
Q4
result [3793296]
γ l_returnflag avg(ps_supplycost)
[3793296] Tmp4 [3034636800000]
⋈ partkey suppkey
[3793296] Tmp2 [3793296]
π l_suppkey l_partkey l_returnflag
[800000] Tmp3 [800000]
π ps_suppkey ps_partkey ps_supplycost
[3793296] Tmp1 [6000000]
σ l_commitdate < l_receiptdate
[6000000] lineitem
[800000] partsupp

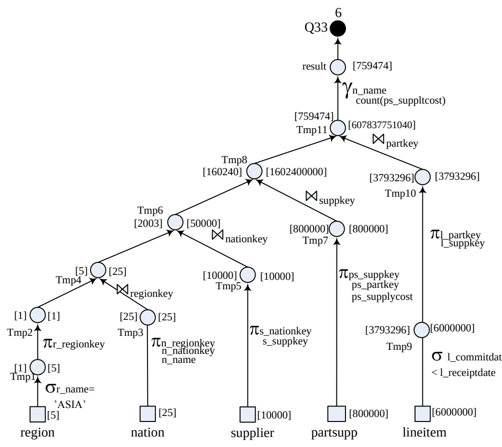
(a) Query Q4



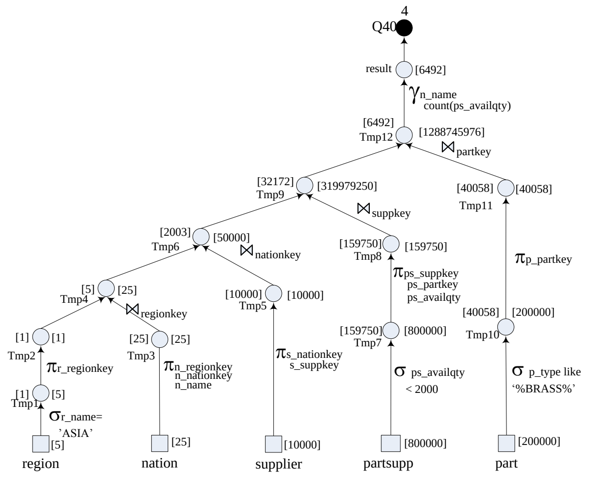
(b) Query Q15



(c) Query Q22

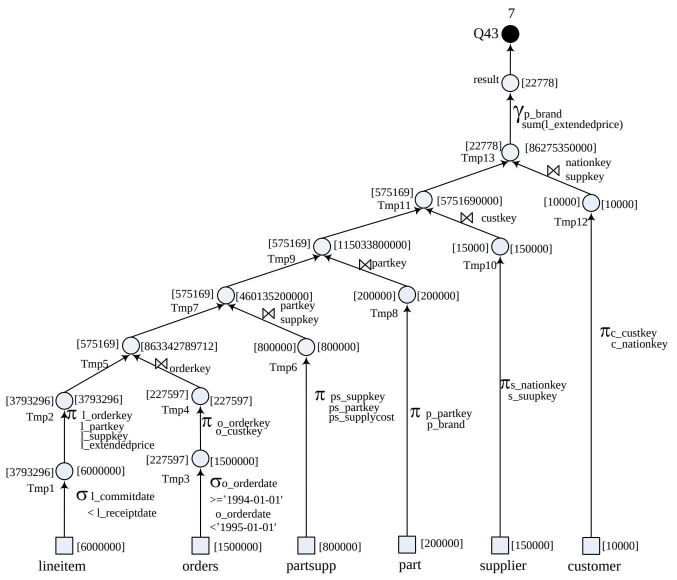


(d) Query Q33



(e) Query Q40

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


(f) Query Q43

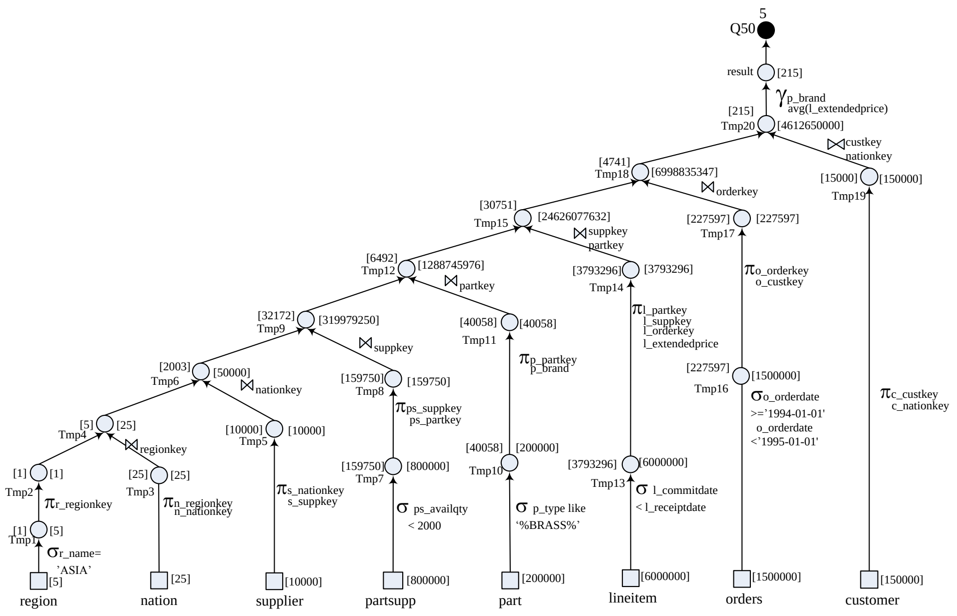

(g) Query Q50

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 Multiply Query Cost

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

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.

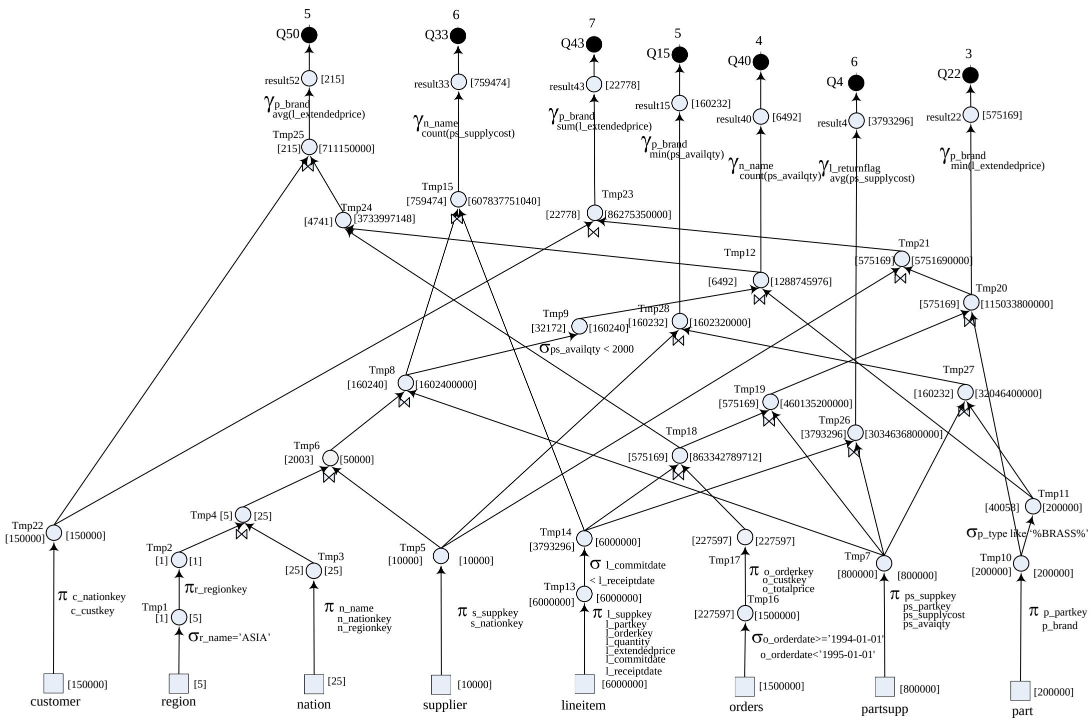


Figure 4.55 The Cheapest MVPP for the Second Experiment: the Queries in the List {Q43, Q40, Q50 Q4, Q33, Q22 and Q15}

Table 4.35 The Query Processing Cost of All MVPPs

Query	1st MVPP	2nd MVPP	3rd MVPP	4th MVPP
Q4	18,207,907,119,552	18,207,907,119,552	18,207,907,119,552	18,207,907,119,552
Q15	168,250,451,160	168,250,451,160	168,250,451,160	840,009,851,160
Q22	12,039,079,457,322	12,039,079,457,322	4,315,574,657,322	12,039,079,457,322
Q33	18,253,477,108,284	3,656,709,383,196	18,253,477,108,284	3,656,709,383,196
Q40	134,600,384,632	11,570,291,056	134,600,384,632	672,007,903,472
Q43	28,735,371,933,681	28,735,371,933,681	10,713,860,733,681	29,065,873,503,681
Q50	20,525,276,724,865	20,525,276,724,865	20,525,276,724,865	20,761,338,241,345
Total	**98,063,963,179,496**	**83,344,165,360,832**	**72,318,947,179,496**	**85,242,925,459,728**

Query	5th MVPP (cheapest)	6th MVPP	7th MVPP
Q4	18,207,907,119,552	18,207,907,119,552	18,207,907,119,552
Q15	168,250,451,160	168,250,451,160	168,250,451,160
Q22	4,315,574,657,322	12,039,079,457,322	12,039,079,457,322
Q33	3,656,709,383,196	3,656,709,383,196	3,656,709,383,196
Q40	11,570,291,056	11,570,291,056	11,570,291,056
Q43	10,713,860,733,681	28,735,371,933,681	28,735,371,933,681
Q50	870,692,174,234	176,201,501,795*	176,201,501,795*
Total	**37,944,564,810,201**	**62,995,090,137,762**	**62,995,090,137,762**

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.

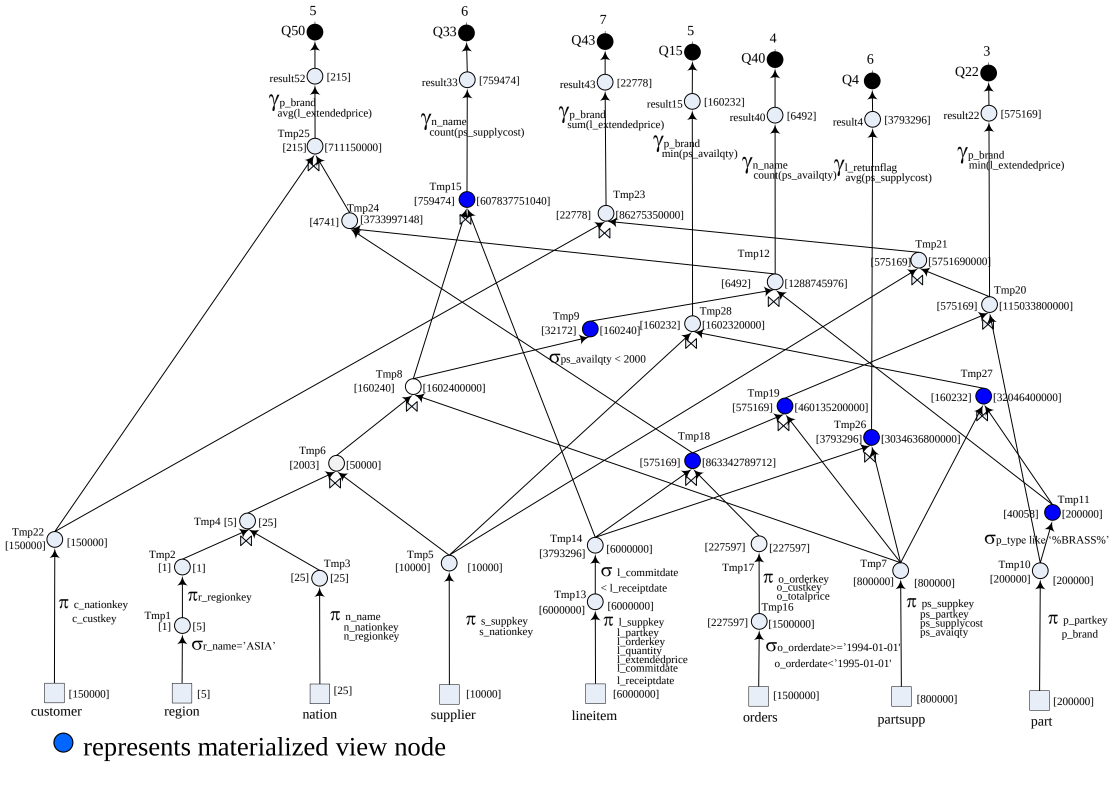


Figure 4.56 The Optimal MVPP with Materialized Views for Initial Requirements

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

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

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.

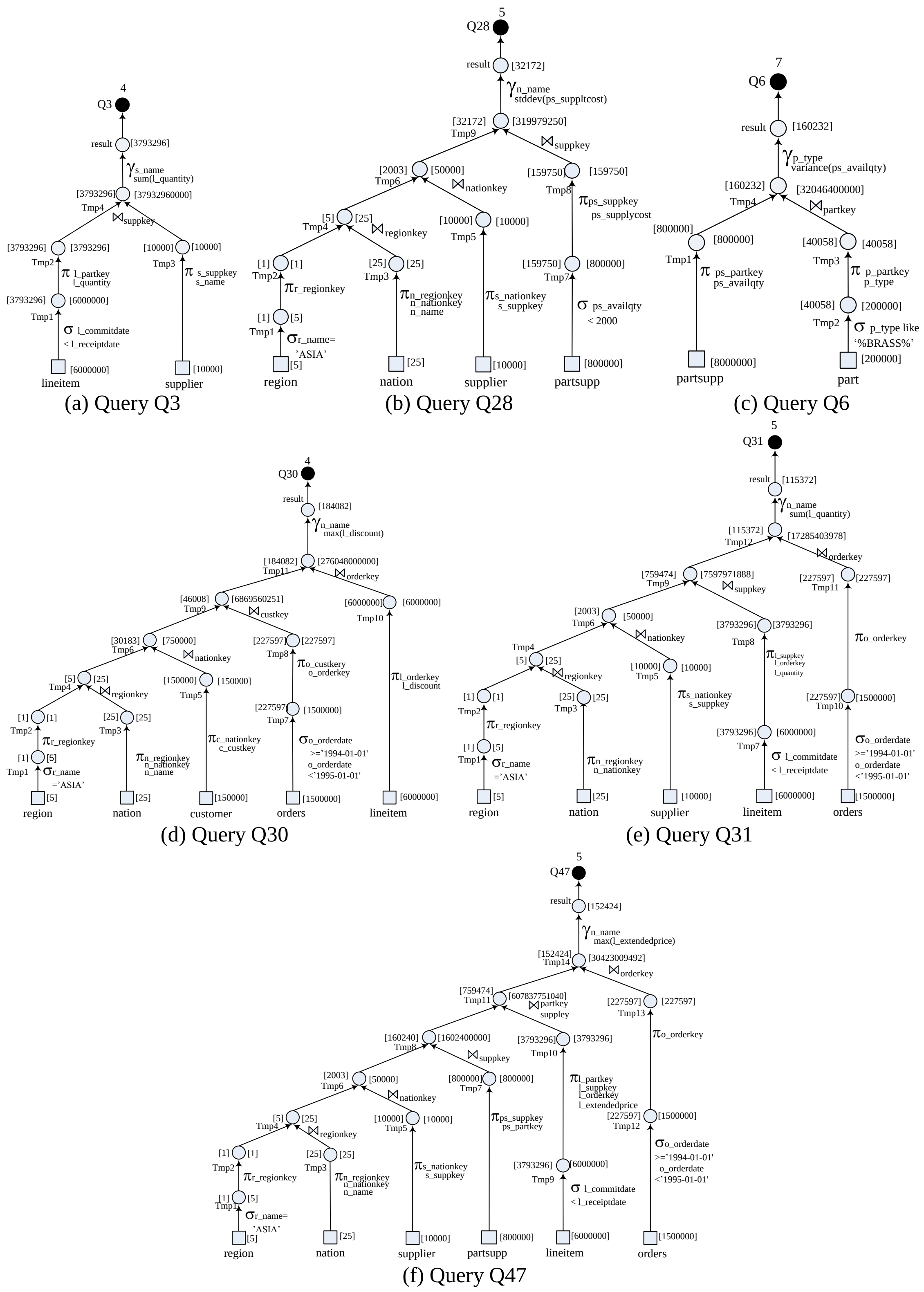
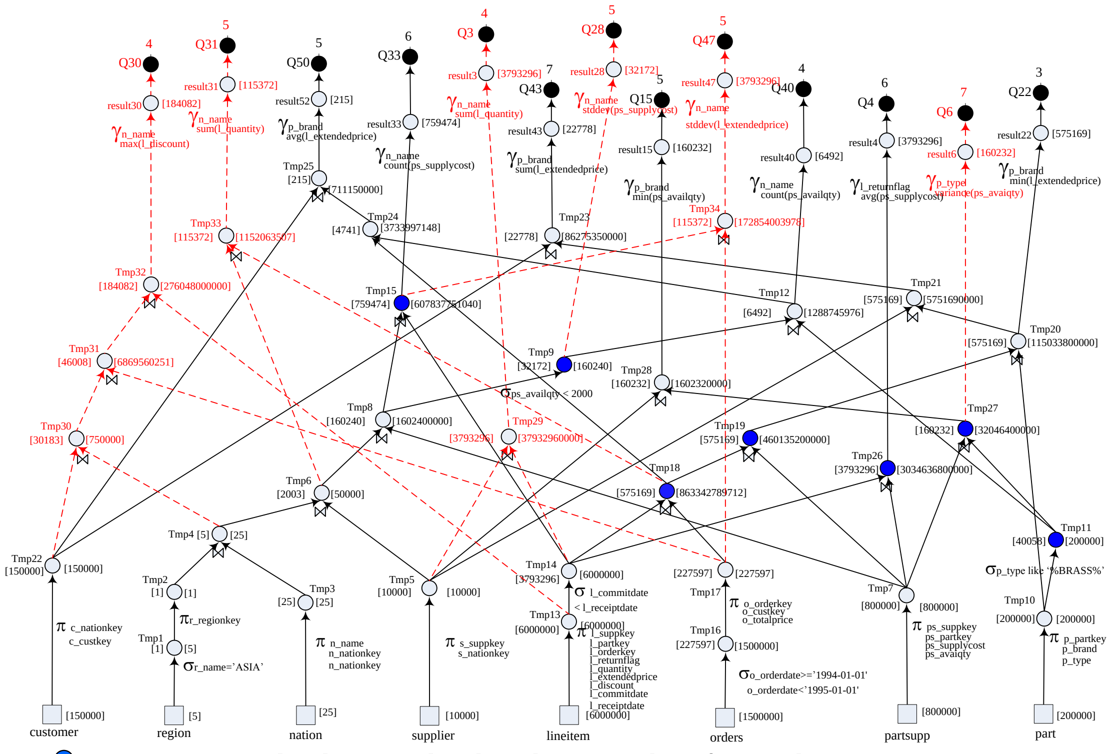


Figure 4.57 The Relational Algebra Query Tree for the New Queries: Q3, Q6, Q28, Q30, Q32 and Q47

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.



Q30
Q31
Q50
Q33
Q3
Q43
Q28
Q15
Q47
Q40
Q4
Q6
Q22
result30 [184082]
result31 [115372]
result52 [215]
result33 [759474]
result3 [3793296]
result43 [22778]
result28 [32172]
result15 [160232]
result47 [3793296]
result40 [6492]
result4 [3793296]
result6 [160232]
result22 [575169]
Tmp34 [115372] [172854003978]
Tmp33 [115372] [1152063507]
Tmp32 [184082] [276048000000]
Tmp31 [46008] [6869560251]
Tmp30 [30183] [750000]
Tmp29 [3793296] [37932960000]
Tmp28 [160232] [1602320000]
Tmp27 [160232] [32046400000]
Tmp26 [3793296] [3034636800000]
Tmp25 [215] [711150000]
Tmp24 [4741] [3733997148]
Tmp23 [22778] [86275350000]
Tmp22 [150000] [150000]
Tmp21 [575169] [5751690000]
Tmp20 [575169] [115033800000]
Tmp19 [575169] [460135200000]
Tmp18 [575169] [863342789712]
Tmp17 [227597] [227597]
Tmp16 [227597] [1500000]
Tmp15 [759474] [6078377751040]
Tmp14 [3793296] [6000000]
Tmp13 [6000000] [6000000]
Tmp12 [6492] [1288745976]
Tmp11 [40058] [200000]
Tmp10 [200000] [200000]
Tmp9 [32172] [160240]
Tmp8 [160240] [1602400000]
Tmp7 [800000] [800000]
Tmp6 [2003] [50000]
Tmp5 [10000] [10000]
Tmp4 [5] [25]
Tmp3 [25] [25]
Tmp2 [1] [1]
Tmp1 [1] [5]
σps_availqty < 2000
σp_type like '%BRASS%'
σl_commitdate < l_receiptdate
σo_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
σr_name='ASIA'
πr_regionkey
π c_nationkey c_custkey
π n_name n_nationkey n_nationkey
π s_suppkey s_nationkey
π l_suppkey l_partkey l_orderkey l_returnflag l_quantity l_extendedprice l_discount l_commitdate l_receiptdate
π o_orderkey o_custkey o_totalprice
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type
customer [150000]
region [5]
nation [25]
supplier [10000]
lineitem [6000000]
orders [1500000]
partsupp [800000]
part [200000]

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

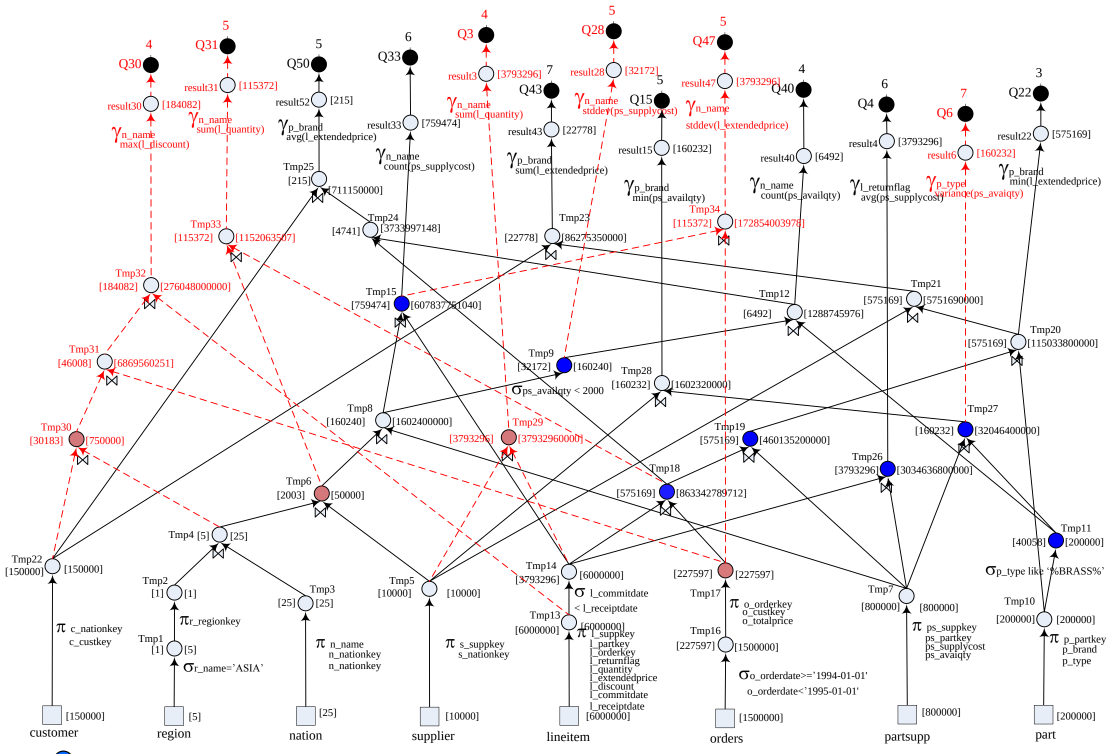
Figure 4.58 The Optimal Dynamic MVPP Constructed by Merging Steps

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.



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

● represents new materialized view node selected in dynamic phase

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.

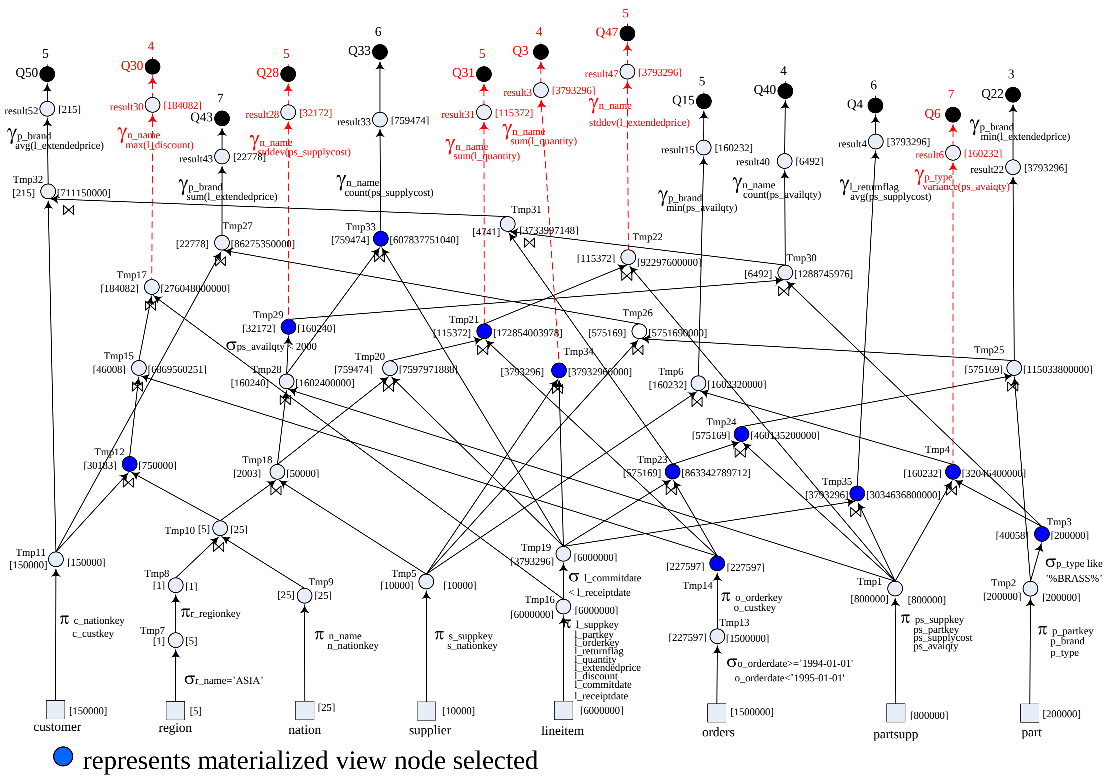


Figure 4.60 The Re-Optimized MVPP by Static Approach for All Queries

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

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

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

Query	f_q	Accessed nodes	Cost of Query Processing
Q4	2	Tmp26(materialized view), result4	45,519,552
Q15	5	Tmp5, Tmp27(materialized view), Tmp28, result15	8,013,252,320
Q22	3	Tmp10, Tmp20, Tmp19(materialized view), result22	345,105,451,014
Q33	6	Tmp15(materialized view), result33	9,113,688
Q40	4	Tmp9(materialized view), Tmp11(materialized view), Tmp12, result40	5,155,298,792
Q43	7	Tmp5, Tmp19(materialized view), Tmp10, Tmp20, Tmp21, Tmp22, Tmp23 and result43	1,449,432,585,629
Q50	5	Tmp9(materialized view), Tmp11(materialized view), Tmp12, Tmp18(materialized view), Tmp22, Tmp24, Tmp25, result50	28,673,453,690
Q3	4	Tmp29(materialized view), result3	30,346,368
Q6	7	Tmp27(materialized view), result6	2,243,248
Q28	5	Tmp9(materialized view), result28	321,720
Q30	4	Tmp13, Tmp17(materialized view), Tmp31, Tmp30(materialized view), Tmp32 and result30	1,131,696,008,452
Q31	5	Tmp6(materialized view), Tmp18(materialized view), Tmp33, result31	5,763,780,255
Q47	5	Tmp15(materialized view), Tmp17(materialized view), Tmp34, result47	864,275,532,105
		Total query processing cost	**3,838,202,906,833**

Table 4.39 The Maintenance Cost of the Dynamic Approach

Materialized View	Number of Base Relation	Derived by Nodes	Maintenance Cost
Tmp6	3	Tmp1, 2, 3, 4, 5, 6	180,168
Tmp9	4	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9	6,413,681,184
Tmp11	1	Tmp10, 11	400,000

Table 4.39 (Continued)

Materialized View	Number of Base Relation	Derived by Nodes	Maintenance Cost
Tmp15	5	Tmp1,2,3 4, 5, 6, 7, 8, 13, 14, 15	3,047,265,055,480
Tmp17	1	Tmp16, 17	1,727,597
Tmp18	2	Tmp13, 14, 16, 17, 18	1,726,713,034,618
Tmp19	3	Tmp7, 13, 14, 16, 17, 18, 19	3,970,477,551,927
Tmp26	2	Tmp7, 13, 14, 26	6,069,299,200,000
Tmp27	2	Tmp7, 10, 11, 27	64,095,200,000
Tmp29	2	Tmp5, 13, 14, 29	75,889,940,000
Tmp30	3	Tmp1, 2, 3, 4, 22, 30	2,700,168
Total materialized view maintenance cost			**14,960,158,671,142**

Table 4.40 The Query Processing Cost of the Static Approach

Query	f_q	Accessed Nodes	Cost of Query Processing
Q4	2	Tmp35(materialized view), result4	45,519,552
Q15	5	Tmp4(materialized view), Tmp5, Tmp6, result15	8,013,252,320
Q22	3	Tmp2, Tmp24(materialized view), Tmp25, result22	345,105,451,014
Q33	6	Tmp33(materialized view), result33	9,113,688
Q40	4	Tmp3(materialized view), Tmp29(materialized view), Tmp30, result40	5,155,298,792
Q43	7	Tmp2, Tmp5, Tmp11, Tmp24(materialized view), Tmp25, Tmp26, Tmp27, result43	1,449,432,585,629
Q50	5	Tmp3(materialized view), Tmp11, Tmp23(materialized view), Tmp29(materialized view), Tmp30, Tmp31, Tmp32, result50	28,673,453,690
Q3	4	Tmp34(materialized view), result3	30,346,368
Q6	7	Tmp4(materialized view), result6	2,243,248
Q28	5	Tmp29(materialized view), result28	321,720

Table 4.40 (Continued)

Query	f_q	Accessed Nodes	Cost of Query Processing
Q30	4	Tmp12(materialized view), Tmp13, Tmp15, Tmp14(materialized view), Tmp16, Tmp17, result30	1,131,672,008,452
Q31	5	Tmp21(materialized view), result31	576,860
Q47	5	Tmp1, Tmp21(materialized view), Tmp22, result47	461,492,576,860
		Total query processing cost	**3,429,632,748,193**

Table 4.41 The Maintenance Cost of the Static Approach

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

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)*(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)*(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)*(10000), (1)*(200000), (1)*(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)*(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)*(711150000) - (8)*(5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 1602400000 + 160240 + 200000 + 200000 + 1288745976 + 6000000 + 6000000 + 1500000 + 227597 + 863342789712 + 150000 + 3733997148 + 711150000) that is (-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)*(1288745976) - (5)*(5 + 1 + 25 + 25 + 10000 + 50000 + 800000 +1602400000 + 160240 + 200000 + 200000 + 1288745976) that is (-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, 460135200000and 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

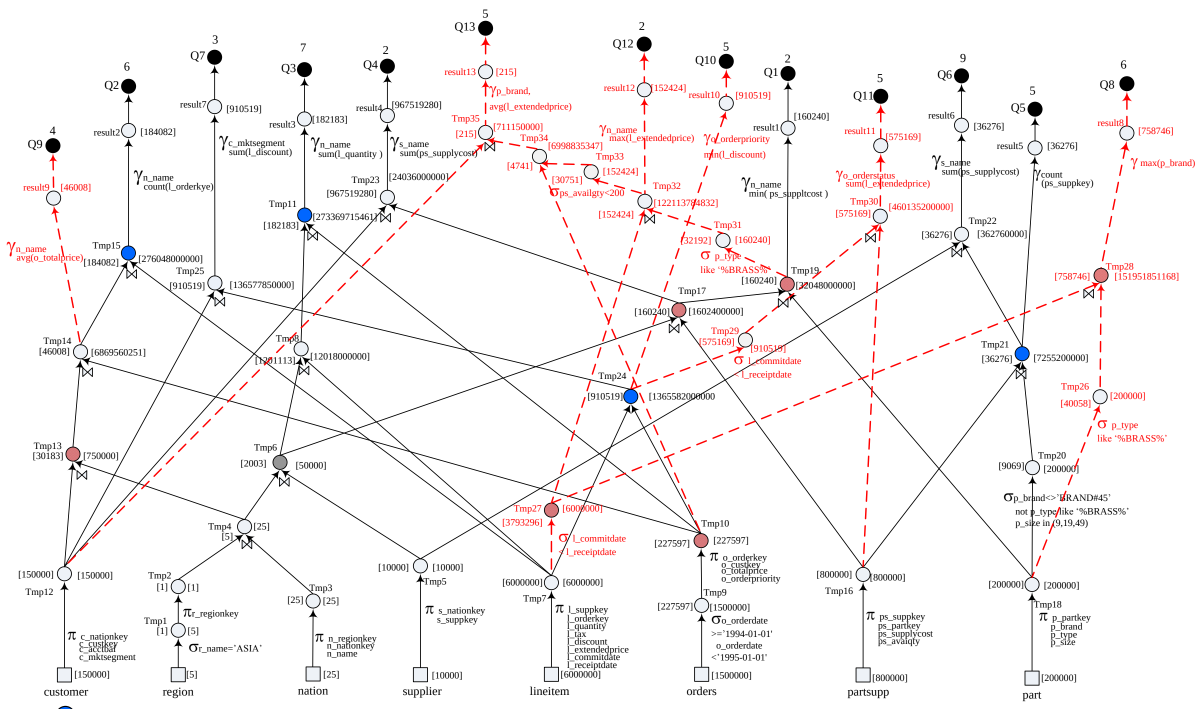
Example	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
*	3,838,202,906,833	14,960,158,671,142	18,798,361,577,975
4	3,809,528,012,678	21,925,713,716,974	25,735,241,729,652
5	3,826,601,006,635	14,974,621,102,502	18,801,222,109,137
6	3,834,617,947,255	14,968,207,421,318	18,802,825,368,573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

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

Materialized view	Cost of Query Processing	Cost of Maintenance	Total Cost
Tmp10, 11, 13, 15, 17, 19, 21, 24, 27, 28	3,684,593,788,740	6,065,915,980,889	9,750,509,769,629

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

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

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, $C_{maintenace}$: 14,877,850,042,025

4th state: Tmp5, 15, 18, 19, 26, 27, $C_{maintenace}$: 14,877,850,052,025

8th state: Tmp10, 15, 18, 19, 26, 27, $C_{maintenace}$: 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 {Tmp15, Tmp18, Tmp19, Tmp26 and Tmp17} 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

State	Materialized View	Cost of Query Processing	Cost of Maintenance	Total Cost
0	Tmp15, 18, 19, 26, 27 (the initial state for SA)	4,012,431,182,390	14,877,850,042,025	18,890,281,224,415
1	Tmp7, 14, 15, 18, 19, 27	22,220,256,782,390	8,808,563,642,025	31,028,820,424,415
2	Tmp7, 13, 15, 18, 19, 27	22,220,256,782,390	8,808,557,642,025	31,028,814,424,415
3	Tmp15, 18, 19, 26, 28	4,228,751,610,766	14,914,704,632,025	19,143,456,242,791
4	Tmp5, 15, 18, 19, 26, 27	4,012,431,182,390	14,877,850,052,025	18,890,281,424,415
5	Tmp7, 11, 15, 18, 19, 26	4,396,994,700,534	14,813,756,042,025	19,210,750,742,559
6	Tmp7, 10, 15, 18, 19, 26	4,397,000,459,606	14,813,755,842,025	19,210,756,301,631
7	Tmp15, 18, 20, 26, 27	2,862,091,182,390	16,661,478,559,334	19,523,569,741,724
8	Tmp10, 15, 18, 19, 26, 27	4,012,431,182,390	14,877,850,242,025	18,890,281,424,415
9	Tmp8, 14, 18, 19, 26, 27	7,637,008,803,006	11,837,010,026,769	19,474,018,829,775
10	Tmp8, 13, 18, 19, 26, 27	7,637,082,663,342	11,837,004,026,769	19,474,086,690,111
11	Tmp6, 15, 18, 19, 26, 27	4,012,430,079,383	14,877,850,222,193	18,890,280,301,576
12	Tmp4, 5, 15, 18, 19, 26, 27	4,012,431,181,217	14,877,850,042,137	18,890,281,223,354
13	Tmp4, 15, 18, 19, 26, 27	4,012,431,181,217	14,877,850,042,137	18,890,281,223,354
14	Tmp6, 12, 15, 18, 19, 26, 27	3,986,400,439,685	14,892,313,053,553	18,878,713,493,238
15	Tmp6, 9, 15, 18, 19, 26, 27	3,989,983,818,331	14,884,263,903,377	18,874,247,721,708

Table 4.45 (Continued)

State	Materialized View	Cost of Query Processing	Cost of Maintenance	Total Cost
16	Tmp6, 9, 11, 15, 18, 19, 26, 27	3,989,980,218,911	14,884,264,303,377	18,874,244,522,288
17	Tmp6, 8, 11, 15, 18, 19, 26, 27	3,989,983,101,991	14,884,263,662,417	18,874,246,764,408
18	Tmp6, 9, 11, 15, 18, 19, 23, 26, 27	2,540,547,952,174	24,067,588,334,828	26,608,139,170,082
19	Tmp6, 9, 11, 15, 18, 19, 22, 21, 26, 27	3,144,480,388,911	22,105,657,267,519	25,250,137,656,430
20	Tmp6, 9, 11, 15, 18, 19, 21, 26, 27	3,144,480,388,911	22,105,657,117,519	25,250,137,506,430
21	Tmp5, 6, 9, 11, 15, 18, 19, 20, 26, 27	3,184,742,218,911	20,638,372,110,210	23,823,114,329,121
22	Tmp6, 9, 11, 15, 18, 20, 26, 27	3,184,742,218,911	20,638,372,100,210	23,823,114,319,121
23	Tmp6, 9, 11, 15, 18, 19, 25, 26, 27	3,961,306,767,371	21,849,821,076,806	25,811,127,844,177
24	Tmp5, 6, 9, 11, 15, 18, 19, 24, 26, 27	3,961,663,290,001	20,104,164,525,348	24,065,827,815,349
25	Tmp6, 9, 11, 15, 18, 19, 24, 26, 27	3,961,663,290,001	20,104,164,515,348	24,065,827,805,349
26	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29	3,838,215,512,095	14,960,155,970,974	18,798,371,483,069
27	Tmp5, 6, 9, 11, 14, 15, 18, 19, 26, 27	3,989,947,392,095	14,884,278,040,974	18,874,225,433,069
28	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 32	2,722,547,177,757	16,374,786,910,494	19,097,334,088,251
29	Tmp6, 9, 11, 13, 15, 18, 19, 26, 27, 29, 31	3,826,762,625,461	14,987,650,722,590	18,814,413,348,051
30	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 31	3,826,762,625,461	14,987,644,722,590	18,814,407,348,051
31	Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 30 (the minimal state)	3,838,202,906,833	14,960,158,671,142	18,798,361,577,975
32	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 30	3,838,208,906,833	14,960,156,943,545	18,798,365,850,378
33	Tmp4, 6, 9, 11, 15, 17, 18, 19, 22, 26, 27, 29	3,838,212,386,121	14,960,156,121,086	18,798,368,507,207
34	Tmp9, 11, 15, 17, 18, 19, 26, 27, 29, 30, 33	3,832,440,280,298	16,691,479,770,464	20,523,920,050,762
35	Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 34	2,973,928,528,448	19,654,011,127,168	22,627,939,655,616
36	Tmp6, 8, 11, 14, 15, 17, 18, 19, 26, 27, 29	6,620,974,199,060	16,606,746,071,688	23,227,720,270,748

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

State	Materialized View	Cost of Query Processing	Cost of Maintenance	Total Cost
0	Tmp15, 18, 19, 26, 27 (the initial state for SA)	4,012,431,182,390	14,877,850,042,025	18,890,281,224,415
1	Tmp14, 15, 18, 19, 27	22,220,256,782,390	8,808,562,842,025	31,028,819,624,415
2	Tmp11, 15, 18, 19, 26	4,396,994,700,534	14,813,755,242,025	19,210,749,942,559
3	Tmp8, 14, 18, 19, 26, 27	7,637,008,803,006	11,837,010,026,769	19,474,018,829,775
4	Tmp6, 15, 18, 19, 26, 27	4,012,430,079,383	14,877,850,222,193	18,890,280,301,576
5	Tmp4, 15, 18, 19, 26, 27	4,012,431,181,217	14,877,850,042,137	18,890,281,223,354
6	Tmp6, 9, 11, 15, 18, 19, 26, 27	3,989,980,218,911	14,884,264,303,377	18,874,244,522,288
7	Tmp6, 8, 11, 15, 18, 19, 26, 27	3,989,983,101,991	14,884,263,662,417	18,874,246,764,408
8	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29	3,838,215,512,095	14,960,155,970,974	18,798,371,483,069
9	Tmp6, 9, 11, 14, 15, 18, 19, 26, 27	3,989,947,392,095	14,884,278,030,974	18,874,225,423,069
10	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 32	2,722,547,177,757	16,374,786,910,494	19,097,334,088,251
11	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 31	3,826,762,625,461	14,987,644,722,590	18,814,407,348,051
12	Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 30 (the minimal state)	3,838,202,906,833	14,960,158,671,142	18,798,361,577,975
13	Tmp6, 9, 11, 15, 18, 19, 26, 27, 29, 30	3,838,208,906,833	14,960,156,943,545	18,798,365,850,378
14	Tmp9, 11, 15, 17, 18, 19, 26, 27, 29, 30, 33	3,832,440,280,298	16,691,479,770,464	20,523,920,050,762
15	Tmp6, 9, 11, 15, 17, 18, 19, 26, 27, 29, 34	2,973,928,528,448	19,654,011,127,168	22,627,939,655,616
16	Tmp6, 8, 11, 14, 15, 17, 18, 19, 26, 27, 29	6,620,974,199,060	16,606,746,071,688	23,227,720,270,748

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

Bello, R. G.; Dias, K.; Downing, A.; Feenan, J.; Finnerty, J.; Norcott, W.D.; Sun, H.; Witkowski, A. and Ziauddin, M. 1998. Materialized views in ORACLE. In **Proceeding of the 24th VLDB Conference.** New York: VLDB. Pp. 659-664.

Chen, F. F. and Dunham, M. H. 1998. Common Subexpression Processing in Multipler-Query Processing. **IEEE Transaction Knowledge Data Engineering**. 10 (May): 493-499.

Choudhari1, Y. D. and Shrivastava, S. K., 2012. Cluster Based Approach for Selection of Materialized Views. **International Journal of Advanced Research in Computer Science and Software Engineering**. 2 (June): 315-318.

Derakhshan, R.; Dehne, F.; Korn, O. and Stantic, B. 2006. Simulated Annealing for Materialized View Selection in Data Warehousing Environment. In **Proceedings of the 24th IASTED International Conference on Database and Applications**. Anaheim, CA: ACTA Press. Pp. 89-94.

Derakhshan, R.; Stantic, B.; Korn, O. and Dehne. F. 2008. Parallel Simulated Annealing for Materialized View Selection in Data Warehousing Environment. In **Proceeding of Algorithms and Architectures for Parallel Processing**, **ICA3PP 2008.** A. Bourgeois and S.Q. Zheng, eds. Berlin, Heidelberg: Springer. Pp. 121-132.

Elmasri, R. and Navathe, S. B. 2010. **Fundamentals of Database Systems**. 6th ed. Boston: Pearson/Addison Wesley.

Finkelstien, S. 1982. Common Expression Analysis in Database Applications. In **Proceeding of ACM-SIGMOD International Conference on Management of Data**. New York: ACM. Pp. 235-245.

Galindo-Legaria, C. A.; Pellenkoft, A. and Kersten, M. L. 1994. Fast, Randomized Join-Order Selection-Why Use Transformations. In **Proceedings of the 20th International Conference on Very Large Data Bases, September 12-15, 1994 (VLDB' 94).** J. B. Bocca, M. Jarke and C. Zaniolo, eds. San Francisco, CA: Morgan Kaufmann Publishers. Pp. 85-95.

Garcia-Molina, H.; Ullman, J. D. and Widom, J. 2009. **Database Systems: The Complete Book**. 2nd ed. New Jersey**:** Prentice Hall.

Gong, A. and Zhao**,** W. **2008. Clustering-Based Dynamic Materialized View Selection Algorithm. Fifth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD '08. Jinan Shandong, China: IEEE. Pp. 391-395.**

Gupta, H. 1997. Selection of Views to Materialize in a Data Warehouse. In **Proceedings of the 6th International Conference on Database Theory, January 08-10, 1997 (ICDT'97).** F. N. Afrati and P. G. Kolaitis, eds. Lecture Notes In Computer Science, vol. 1186. London: Springer-Verlag. Pp. 98-112.

Gupta, H. and Mumick, I. S. 2005. Selection of Views to Materialize in a Data Warehouse. **IEEE Transactions on Knowledge and Data Engineering**. 17 (January): 24-43.

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

Halevy, A. Y. 2001. Answering queries using views: A survey. **The International Journal on Very Large Data Bases**. 10 (September): 270-294.

Inmon, W. H. 2002. **Building the Data Warehouse**. 3rd ed. New York: Wiley.

Ioannidis, Y. E. and Kang, Y. 1990. Randomized Algorithms for Optimizing Large Join Queries. In **Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data**. Atlantic City, New Jersey, May 23 - 26, 1990 (SIGMOD '90). New York: ACM. Pp. 312-321.

Jarke, M. 1984. Common Subexpression Isolation in Multiple Query Optimization. In **Query Processing in Database Systems.** W. Kim, D. Reiner, and D. Batory, Eds. New York: Springer-Verlag. Pp. 191-205.

Kalnis, P.; Mamoulis, N. and Papadias, D. 2002. View Selection Using Randomized Search. **Data & Knowledge Engineering**. 42 (July): 89-111.

Kirkpatrick, S.; Gelatt, C.D. and Vecchi, M.P. 1983. Optimization by Simulated Annealing. **Science**. 220 (May): 671-680.

Kotidis, Y. and Roussopoulos, N. 1999. DynaMat: A Dynamic View Management System for Data Warehouse. In **Proceedings of the 25th ACM SIGMOD International Conference on Management Data**. Philadelphia, PA: ACM. Pp. 371-382.

Lanzelotte, R. S.; Valduriez, P. and Zaït, M. 1993. On the Effectiveness of Optimization Search Strategies for Parallel Execution Spaces. In **Proceedings of the 19th International Conference on Very Large Data Bases**, **August 24-27, 1993**. R. Agrawal, S. Baker and D. A. Bell, eds. San Francisco, CA: Morgan Kaufmann Publishers. Pp. 493-504.

Lawrence, M. and Rau-Chaplin, A. 2008. Dynamic View Selection for OLAP. **International Journal of Data Warehousing & Mining**. 4 (January - March): 47-61.

Lehner, W.; Cochrane, B.; Pirahesh, H. and Zahatioudakis, M. 2001. fAST Refresh Using Mass Query Optimization. In **Proceeding of the 17th International Conference on Data Engineer**. Washington, DC: IEEE Computer Society. Pp. 391-398.

Li, X.; Qian, X.; Jiang, J. and Wang, Z. 2010. Shuffeled Frog Leaping Algorithm for Materialized Views Selection. **Second IEEE International Workshop on Education Technology and Computer Science**. Wuhan, China: IEEE. Pp. 7-10.

Mistry, H.; Roy, P.; Sudarshan, S. and Ramamrithan, K. 2001. Materialized View Selection and Maintenance Using Multi-Query Optimization. In **Proceeding of the ACM SIGMOD International Conference on Management of Data**. Santa Barbara, CA: ACM. Pp. 307-318.

Nahar, S.; Sahni, S. and Shragowitz, E. 1986. Simulated Annealing and Combinatorial Optimization. In **Proceedings of the 23rd ACM/IEEE Conference on Design Automation, Las Vegas, Nevada, United States (DAC'86)**. Piscataway, NJ: IEEE Press. Pp. 293-299.

Phuboon-ob, J. and Auepanwiriyakul, R. 2007. Selecting Materialized Views Using Two-Phase Optimization with Multiple View Processing Plan. **World Academy of Science, Engineering and Technology**. 3: 166-171.

Phuboon-ob, J. 2009. **Materialized Views Selection Using Two-Phase Optimization Algorithm.** Doctoral dissertation, National Institute of Development Administration.

Shukla, A.; Deshpande, P. and Naughton, J. F. 1998. Materialized View Selection for Multidimensional Datasets. In **Proceedings of the 24th International Conference on Very Large Data Bases, August 24-27, 1998**. A. Gupta, O. Shmueli, and J. Widom, eds. San Francisco, CA: Morgan Kaufmann Publishers. Pp. 488-499.

Silberschatz, A.; Korth, H. F. and Sudarshan, S. 2010. **Database System Concepts**. 6th ed. Boston: McGraw-Hill.

Silva, Yasin N.; Larson, P. and Zhou, J. 2012. Exploiting Common Subexpressions for Cloud Query Processing. In **ICDE '12 Proceedings of the 2012 IEEE 28th International Conference on Data Engineering**. Washington DC: IEEE Computer Society. Pp. 1337-1348.

Sun, X. and Wang, Z. 2009. An Efficient Materialized Views Selection Algorithm Based on PSO. **International Workshop on Intelligent Systems and Applications**, **IEEE Conference.** Wuhan, China: IEEE Xplore. Pp. 1-4.

Theodorators, D. and Sellis, T. 1999. Dynamic Data Warehouse Design. In **Data Warehousing and Knowledge Discovery(DaWaK'99)**. Mohania, M.K., Tjoa, A.M., eds. Heidelberg: Springer-Verlag. Pp. 1–10.

Theodoratos, D. and Sellis, T. 2000**.** Incremental Design of a Data Warehouse. **Journal of Intelligent Information Systems**. 15 (July-August): 7-27.

Theodoratos, D.; Dalamagas, T.; Simitsis, A. and Stavropoulos, M. 2001. A Randomized Approach for the Incremental Design of an Evolving Data Warehouse. In **Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling**. Yokohama, Japan: Springer Berlin Heidelberg. Pp. 325–338.

Theodoratos, D. and Xu, W. 2006. Computing Closest Common Subexpressions for View Selection Problem. In **Proceeding of the ACM international workshop on Data warehousing and OLAP (DOLAP 2006)**. New York: ACM. Pp.75-82.

Transaction Processing Performance Council (TPC). 2011. **TPC Benchmark^{TM}H.** Retrieved June, 2011 from http://www.tpc.org/tpch/spec/tpch2.14.2.pdf.

Xu, W.; Theodoratos, D.; Zuzarte, C.; Wu, X. and Oria V. 2007. A Dynamic View Materialization Scheme for Sequences of Query and Update Statements, In **Data Warehousing and Knowledge Discovery(DaWaK 2007)**. Berlin, Heidelberg: Springer-Verlag. Pp. 55–65.

Yang, J.; Karlapalem, K. and Li, Q. 1997. Algorithms for Materialized View Design in Data Warehousing Environment. In **Proceedings of the 23rd International Conference on Very Large Data Bases, August 25-29, 1997 (VLDB'97).** M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos and M.A. Jeusfeld, eds. San Francisco, CA: Morgan Kaufmann Publishers. Pp. 136-145.

Yu, Jeffrey Xu; Yao, Xin; Choi, Chi-Hon and Gou, Gang. 2003. Materialized View Selection as Constrained Evolutionary Optimization. **IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews**. 33 (November): 458-467.

Zhang, C.; Yao, X. and Yang, J. 2001. An Evolution Approach to Materialized Views Selection in a Data Warehouse Environment. **IEEE Transaction on Systems, Man, and Cybernet, Part C. Applications and Reviews**. 31 (August): 282-294.

Zhang, C. and Yang, J. 1999. Materialized View Evolution Support in Data Warehouse Environment. In **Proceeding of 6th International Conference on Database Systems for Advanced Applications**. Hsinchu, Taiwan: IEEE. Pp. 247-254.

Zhang,C. and Yang, J. 1999. Genetic algorithm for materialized view selection in data warehouse environments. In **Proceedings of the International Conference on Data Warehousing and Knowledge Discovery**. London: ACM. Pp. 116–125.

Zhang, C.; Yang, J. and Karlapalem, K. 2003. Dynamic Materialized View Selection in Data Warehouse Environment. **Informatica** (Slovenia) 27(4): 451-460.

Zhang, Q.; Sun, X. and Wang, Z. 2009. An Efficient MA-Based Materialized Views Selection Algorithm. In **IITA International Conference on Control, Automation and Systems Engineering**. Zhangjiajie, China: IEEE. Pp.315-318.

Zhou, J.; Larson, P.; Freytag, J. and Lehner, W. 2007. Efficient exploitation of similar subexpressions for query processing. In **Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data**. Beijing, China: ACM. Pp. 533-544.

Zhou, L.; Geng, H. and Xu, M. 2011. An Improved Algorithm for Materialized View Selection. **Journal of Computers**. 6 (January): 130-138.

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.

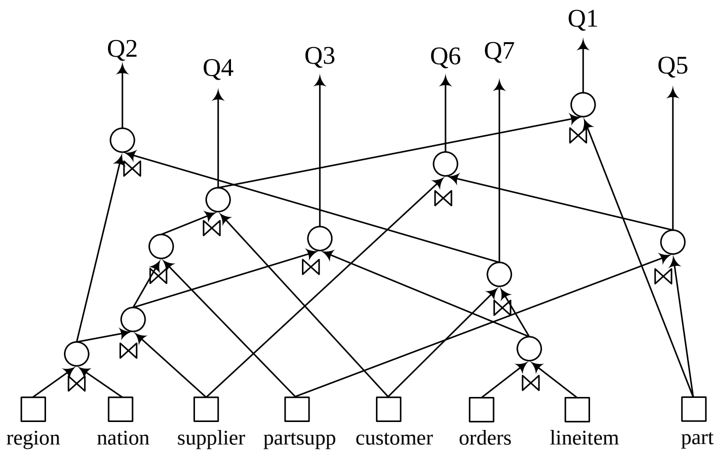


Figure A.1 The Result of the First MVPP

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

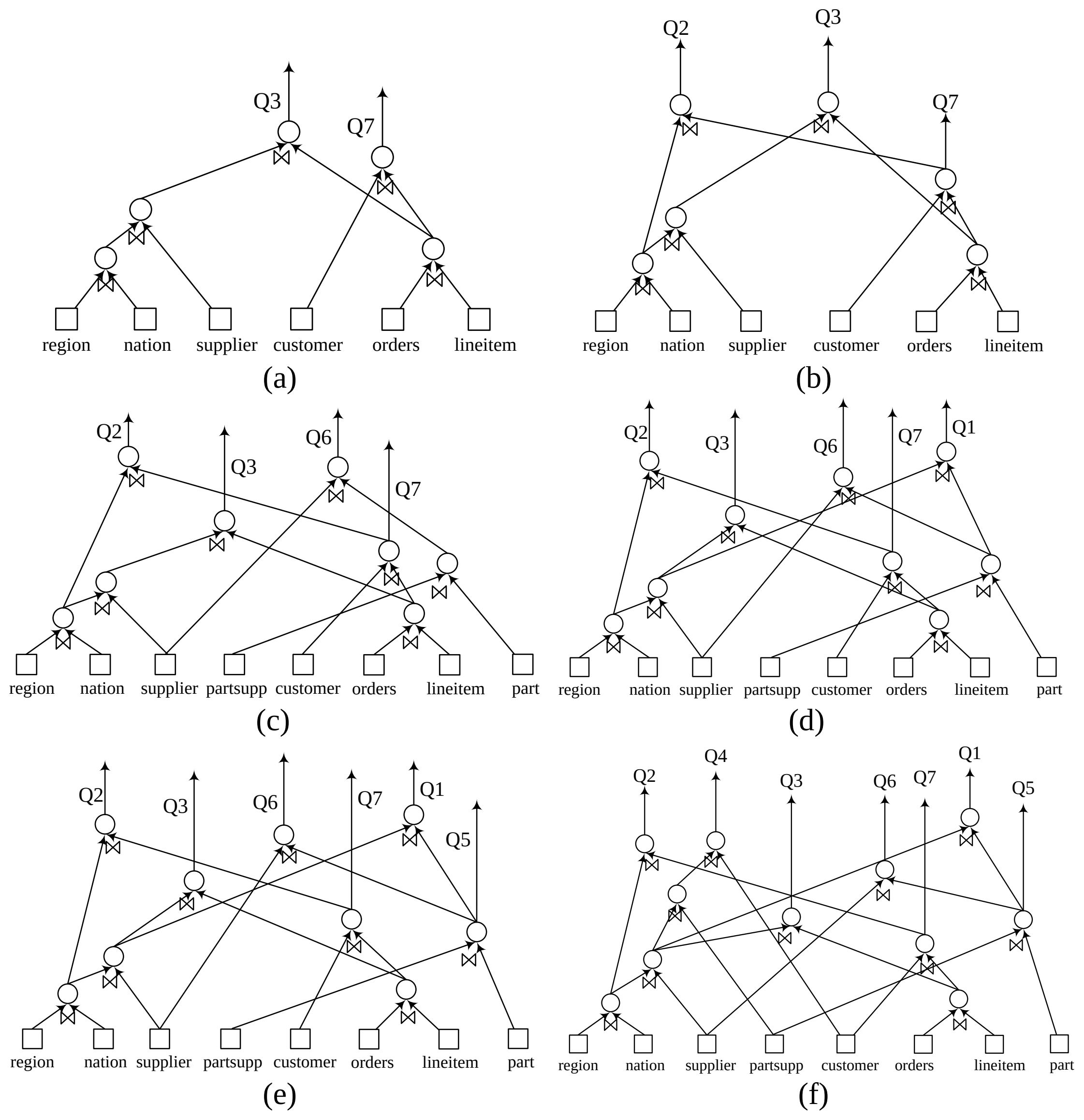


Figure A.2 The Result of the Second MVPP Based on the Order List { Q7, Q3, Q2, Q6, Q1, Q5 and Q4}

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 ⋈ NATION ⋈ SUPPLIER) and (LINEITEM ⋈ 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 ⋈ NATION) and (LINEITEM ⋈ ORDERS ⋈

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 ⋈ SUPPIER) 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 node 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

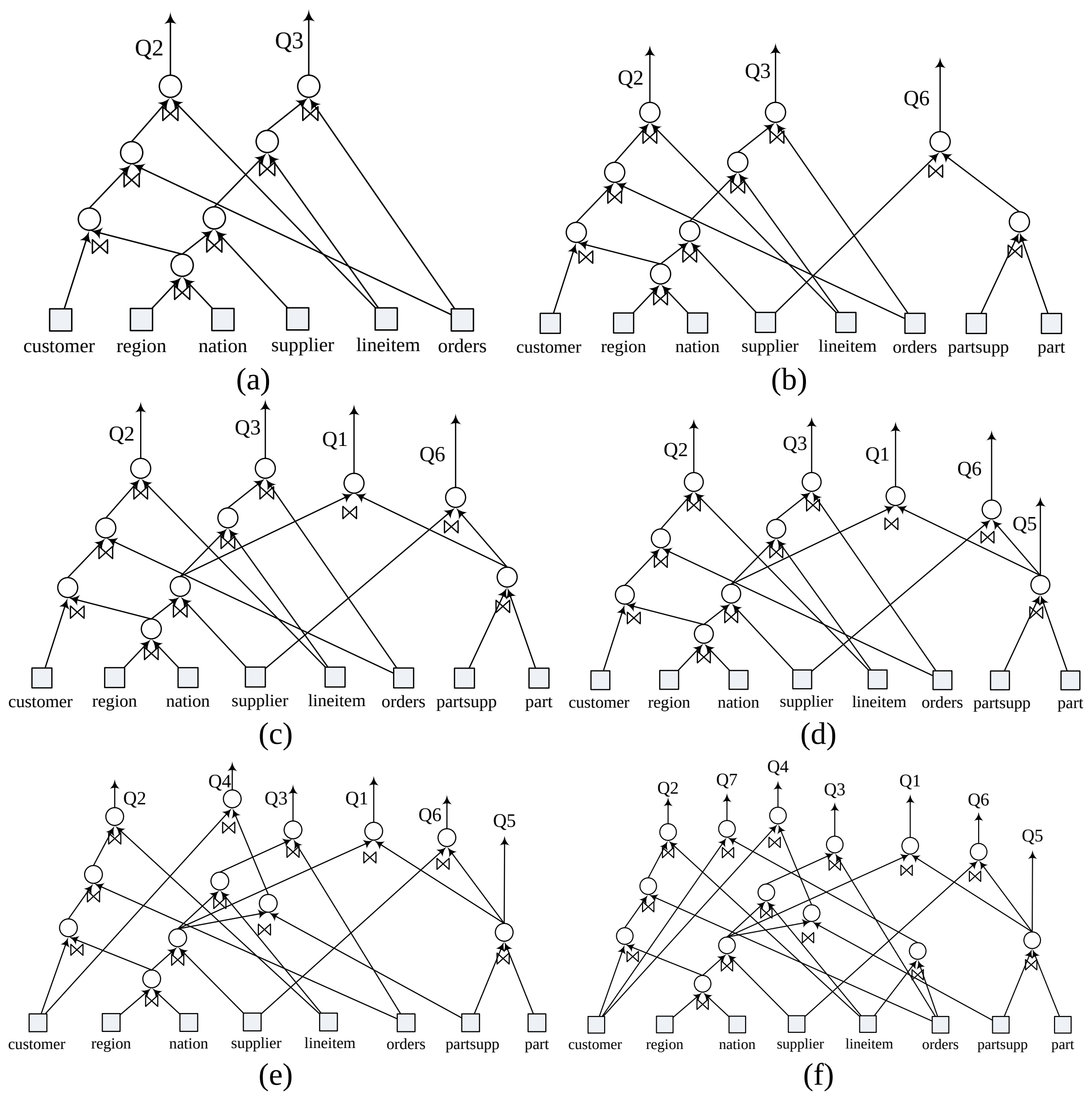


Figure A.3 The Result of the Third MVPP Based on the Order List {Q3, Q2, Q6, Q1, Q5, Q4 and Q7}

operation for existing sharable subexpression and base relation that are the result of (REGION ⋈ 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 (LINEITEM ⋈ 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

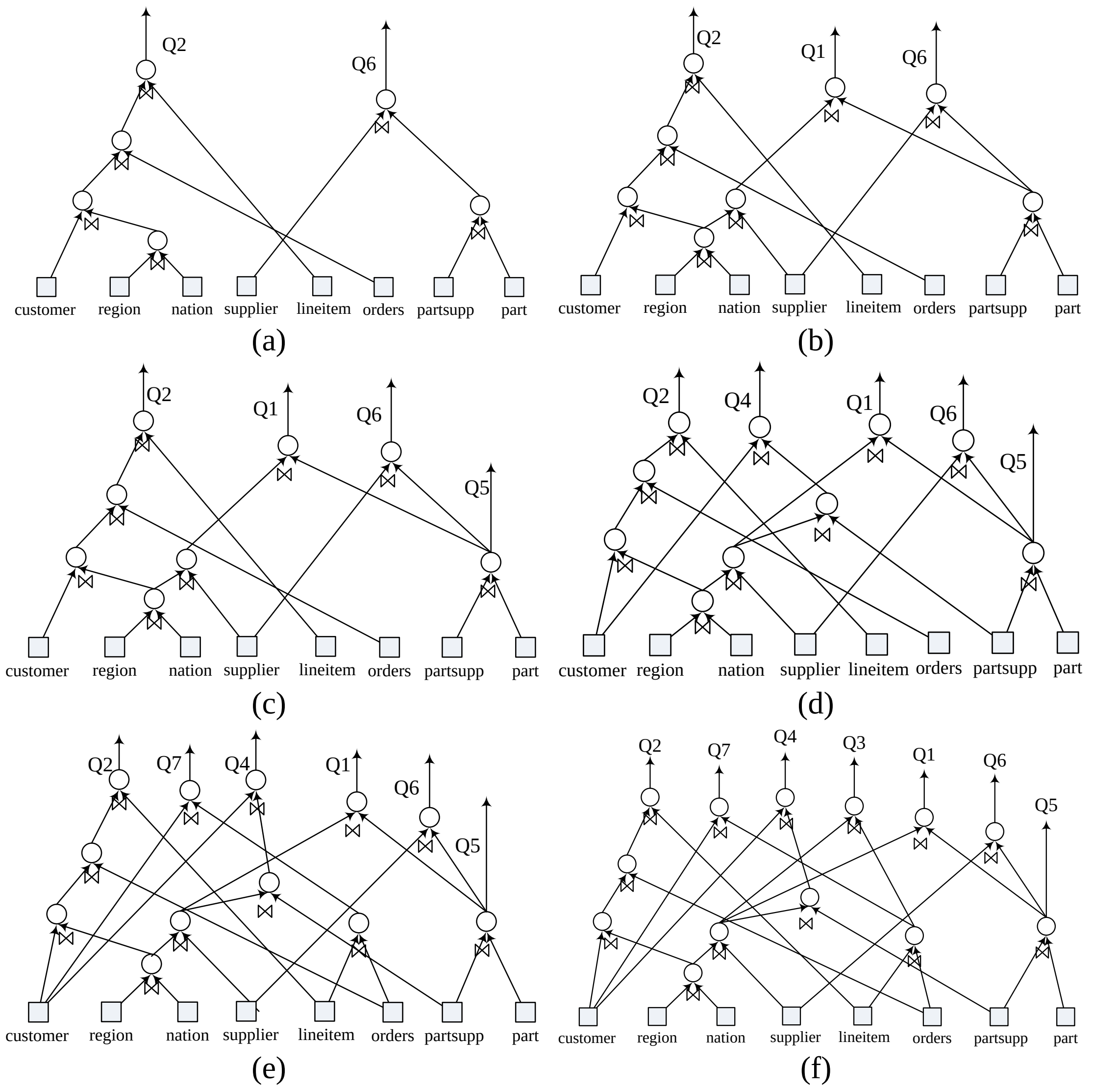


Figure A.4 The Result of the Fourth MVPP Based on the Order List {Q2, Q6, Q1, Q5, Q4, Q7 and Q3}

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 ⋈ 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 ⋈ NATION ⋈ 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 ⋈ NATION ⋈ SUPPLIER) and (LINEITEM ⋈ 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.

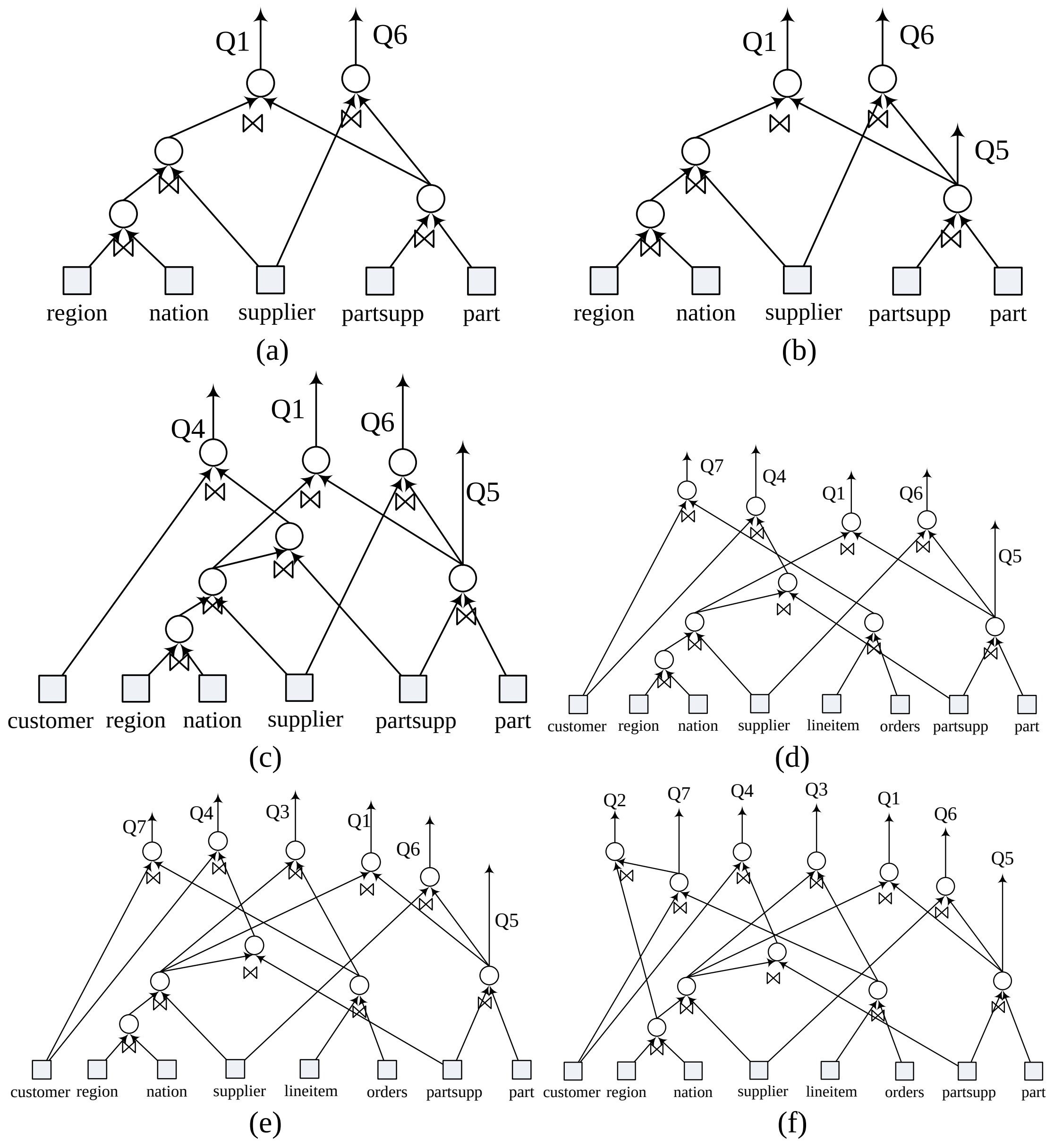


Figure A.5 The Result of the Fifth MVPP Based on the Query List {Q6, Q1, Q5, Q4, Q7, Q3 and Q2}

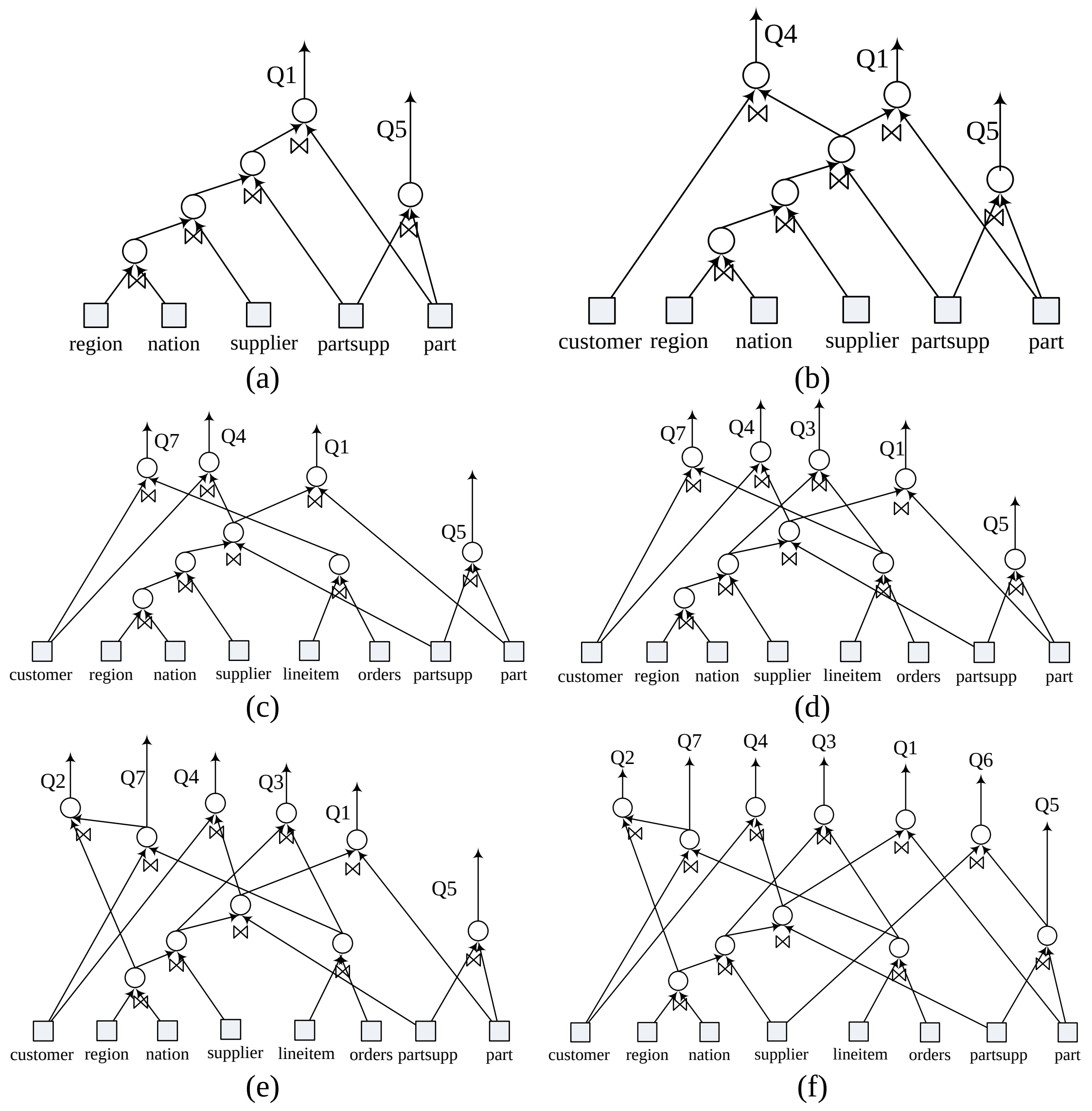


Figure A.6 The Result of the Sixth MVPP Based on the Query List {Q1, Q5, Q4, Q7, Q3, Q2 and Q6}

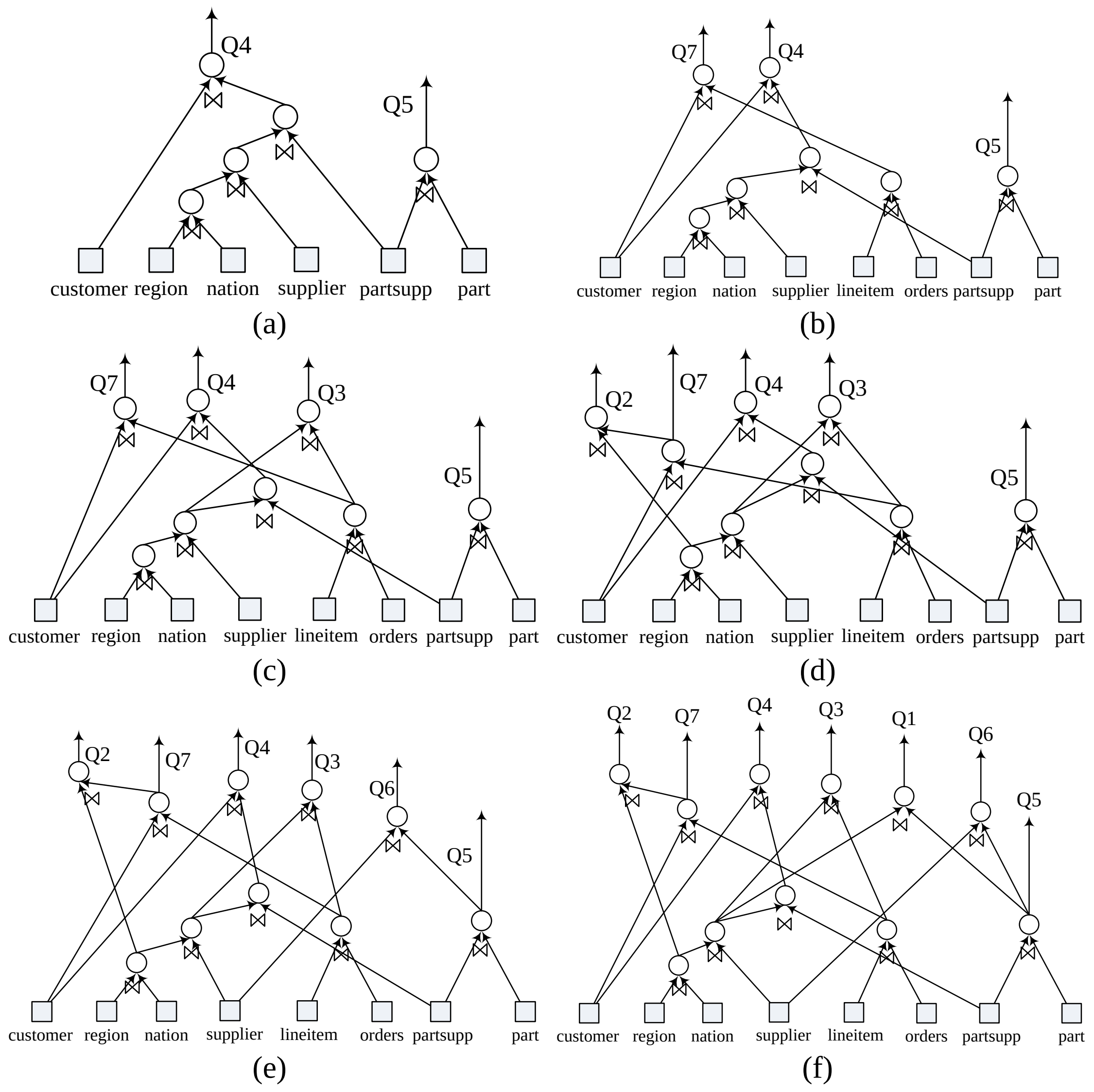
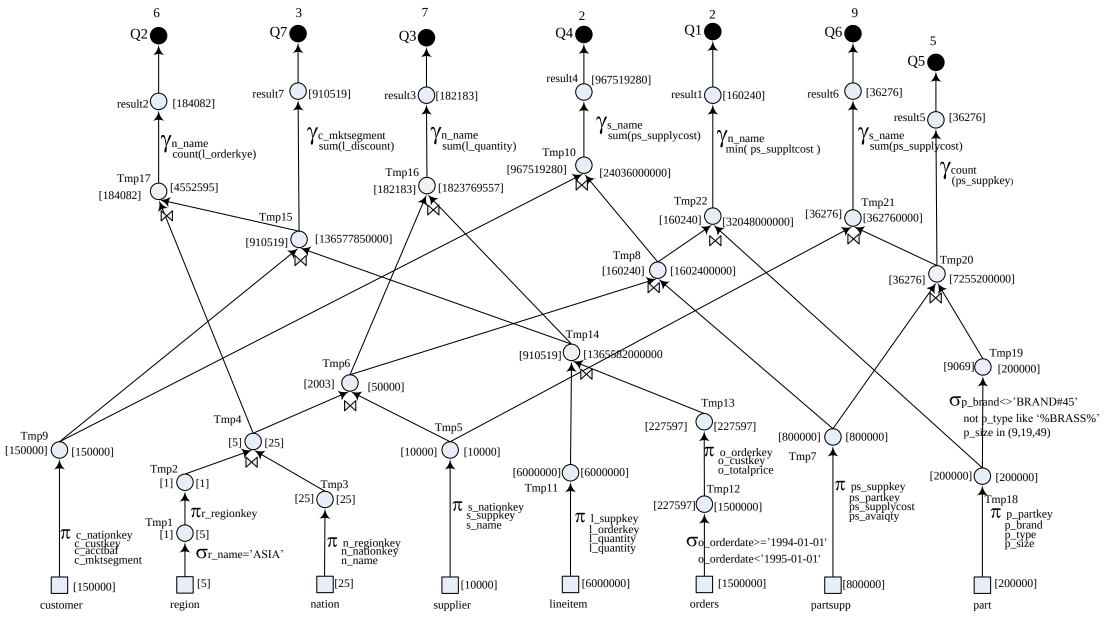

Q4
Q5
customer region nation supplier partsupp part
(a)
Q7
Q4
Q5
customer region nation supplier lineitem orders partsupp part
(b)
Q7
Q4
Q3
Q5
customer region nation supplier lineitem orders partsupp part
(c)
Q2
Q7
Q4
Q3
Q5
customer region nation supplier lineitem orders partsupp part
(d)
Q2
Q7
Q4
Q3
Q6
Q5
customer region nation supplier lineitem orders partsupp part
(e)
Q2
Q7
Q4
Q3
Q1
Q6
Q5
customer region nation supplier lineitem orders partsupp part
(f)

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

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.



Q2 6
Q7 3
Q3 7
Q4 2
Q1 2
Q6 9
Q5 5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp17 [184082] [4552595]
Tmp15 [910519] [136577850000]
Tmp16 [182183] [1823769557]
Tmp10 [967519280] [24036000000]
Tmp22 [160240] [32048000000]
Tmp21 [36276] [362760000]
Tmp8 [160240] [1602400000]
Tmp20 [36276] [7255200000]
Tmp14 [910519] [1365582000000]
Tmp6 [2003] [50000]
Tmp19 [9069] [200000]
Tmp13 [227597] [227597]
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp9 [150000] [150000]
[800000] [800000] Tmp7
π o_orderkey o_custkey o_totalprice
Tmp2 [1] [1]
Tmp3 [25] [25]
[6000000] [6000000] Tmp11
Tmp12 [227597] [1500000]
[200000] [200000] Tmp18
π r_regionkey
Tmp1 [1] [5]
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_quantity
σ o_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part

Figure A.8 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

Query	fq	Access from the nodes	Cost of Each Nodes	Query Processing Cost
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp7, Tmp8, Tmp18, Tmp22 and result1	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp11, Tmp12, Tmp13, Tmp14, Tmp9, Tmp15, Tmp17 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp11, Tmp12, Tmp13, Tmp14, Tmp16 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp7, Tmp18, Tmp19, Tmp20 and result5	800000, 200000, 200000, 7255200000 and 36276	36,282,181,380
Q6	9	Tmp7, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6	800000, 200000, 200000, 7255200000, 10000, 362760000 and 36276	68,572,856,484
Q7	3	Tmp11, Tmp12, Tmp13, Tmp14, Tmp9 , Tmp15 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the first MVPP		**23,316,809,013,207**

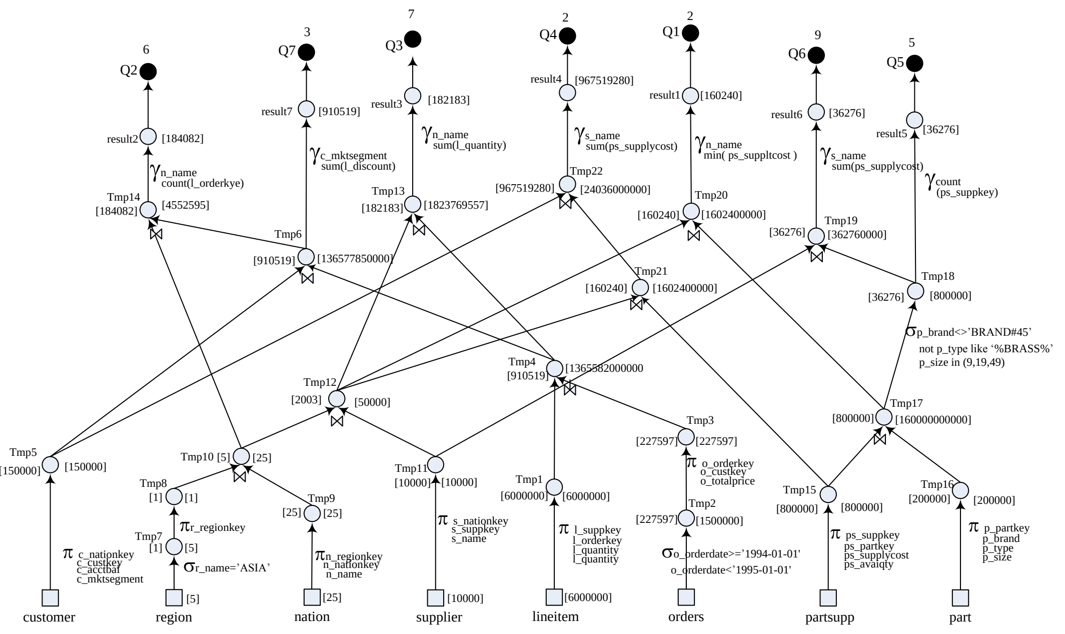


Figure A.9 The Second MVPP, Query in the List: {Q7, Q3, Q2, Q6, Q1, Q5 and Q4}

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

Query	fq	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp16, Tmp17, Tmp20 and result1	5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240	323,207,240,592
Q2	6	Tmp7, Tmp8, Tmp9, Tmp10, Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp14 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp1, Tmp2, Tmp3, Tmp4, Tmp13 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp7, Tmp8, Tmp9, Tmp10, Tmp11, Tmp12, Tmp15, Tmp21, Tmp5, Tmp22 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp15, Tmp16, Tmp17, Tmp18 and result5	800000, 200000, 160000000000, 800000 and 36276	800,009,181,380
Q6	9	Tmp15, Tmp16, Tmp17, Tmp18, Tmp11, Tmp19 and result6	800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276	1,443,281,456,484
Q7	3	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
The total query processing cost of the second MVPP				**25,711,148,613,207**

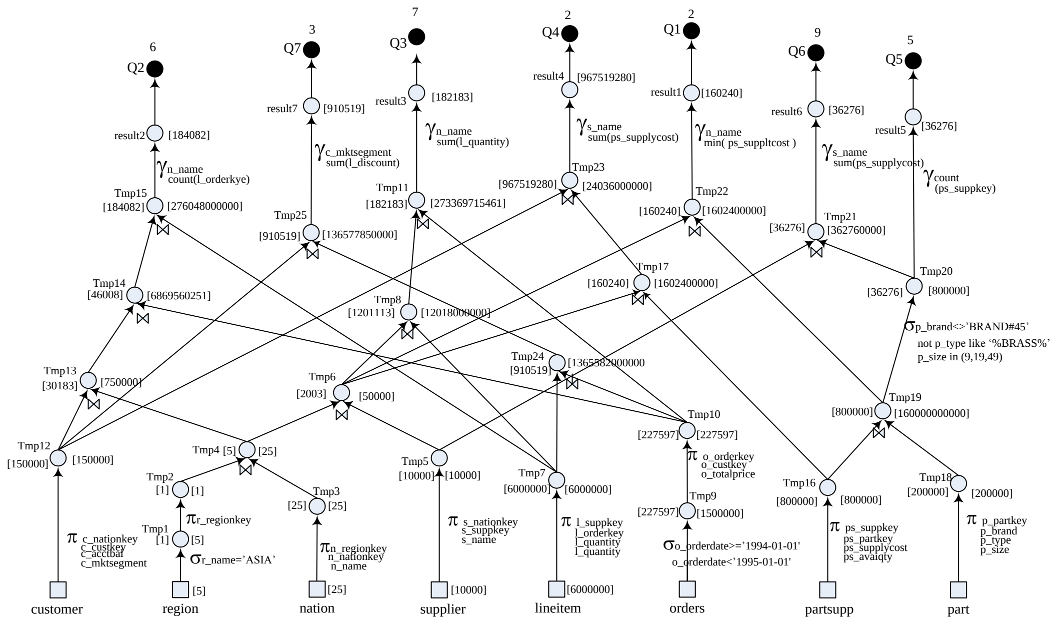


Figure A.10 The Third MVPP (the Cheapest MVPP), Query in the List: {Q3, Q2, Q6, Q1, Q5, Q4 and Q7}

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

Query	fq	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp18, Tmp19, Tmp22 and result1	5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240	323,207,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp15 and result2	5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251, 6000000, 276048000000 and 184082	1,697,558,231,916
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2	5, 1, 25, 25, 10000, 50000, 6000000, 12018000000, 1500000, 227597 , 273369715461 and 182183	1,997,769,797,079
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp12, Tmp23	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp16, Tmp18, Tmp19, Tmp20 and result5	800000, 200000, 160000000000, 800000 and 36276	800,009,181,380
Q6	9	Tmp16, Tmp18, Tmp19, Tmp20, Tmp5, Tmp21 and result6	800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276	1,443,281,456,484
Q7	3	Tmp7, Tmp9, Tmp10, Tmp24, Tmp12, Tmp25 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the third MVPP		**10,821,545,680,471**

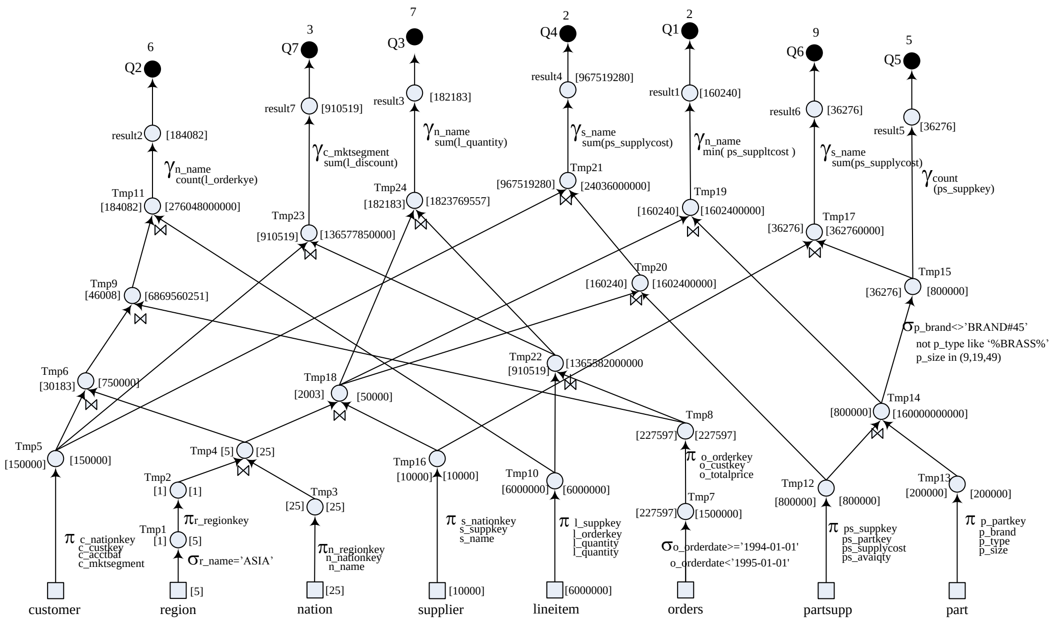
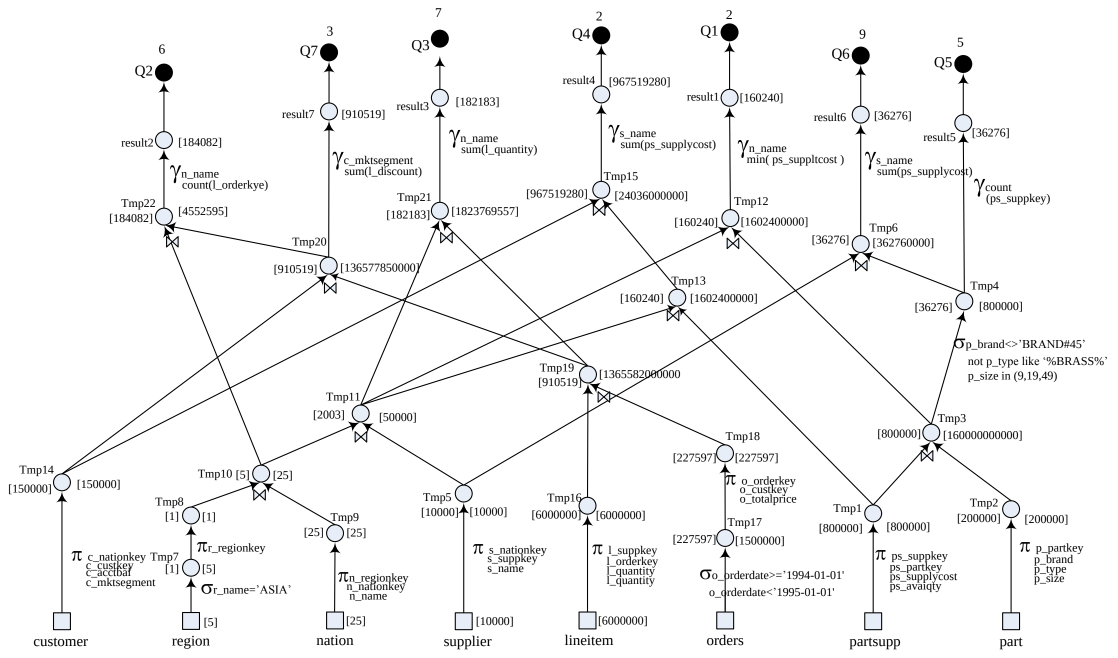

Q2
Q7
Q3
Q4
Q1
Q6
Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γn_name count(l_orderkye)
γc_mktsegment sum(l_discount)
γn_name sum(l_quantity)
γs_name sum(ps_supplycost)
γn_name min(ps_suppltcost)
γs_name sum(ps_supplycost)
γcount (ps_suppkey)
Tmp11 [184082] [276048000000]
Tmp23 [910519] [136577850000]
Tmp24 [182183] [1823769557]
Tmp21 [967519280] [24036000000]
Tmp19 [160240] [1602400000]
Tmp17 [36276] [362760000]
Tmp20 [160240] [1602400000]
Tmp15 [36276] [800000]
Tmp9 [46008] [6869560251]
Tmp22 [910519] [1365582000000]
σp_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp6 [30183] [750000]
Tmp18 [2003] [50000]
Tmp14 [800000] [160000000000]
Tmp8 [227597] [227597]
Tmp5 [150000] [150000]
Tmp4 [5] [25]
Tmp16 [10000] [10000]
Tmp10 [6000000] [6000000]
π o_orderkey o_custkey o_totalprice
Tmp12 [800000] [800000]
Tmp13 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp7 [227597] [1500000]
πr_regionkey
Tmp1 [1] [5]
π c_nationkey c_custkey c_acctbal c_mktsegment
σr_name='ASIA'
πn_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_quantity
σo_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5]
[25]
[10000]
[6000000]
customer
region
nation
supplier
lineitem
orders
partsupp
part
6
3
7
2
2
9
5

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

Query	f_q	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp16, Tmp18, Tmp12, Tmp13, Tmp14, Tmp19 and result1	5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240	323,207,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result2	5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251, 6000000, 276048000000 and 184082	1,697,558,231,916
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp16, Tmp18, Tmp10, Tmp7, Tmp8, Tmp22 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp16, Tmp18, Tmp12, Tmp20, Tmp5, Tmp21 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp12, Tmp13, Tmp14, Tmp15 and result5	800000, 200000, 60000000000, 800000 and 36276	800,009,181,380
Q6	9	Tmp12, Tmp13, Tmp14, Tmp15, Tmp16, Tmp17 and result6	800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276	1,443,281,456,484
Q7	3	Tmp10, Tmp7, Tmp8, Tmp22, Tmp5, Tmp23 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the fourth MVPP		**18,395,672,059,143**



6 Q2
3 Q7
7 Q3
2 Q4
2 Q1
9 Q6
5 Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γn_name count(l_orderkye)
γc_mktsegment sum(l_discount)
γn_name sum(l_quantity)
γs_name sum(ps_supplycost)
γn_name min(ps_suppltcost)
γs_name sum(ps_supplycost)
γcount (ps_suppkey)
Tmp22 [184082] [4552595]
Tmp20 [910519] [136577850000]
Tmp21 [182183] [1823769557]
Tmp15 [967519280] [24036000000]
Tmp12 [160240] [1602400000]
Tmp6 [36276] [362760000]
Tmp13 [160240] [1602400000]
Tmp4 [36276] [800000]
σp_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp19 [910519] [1365582000000]
Tmp11 [2003] [50000]
Tmp18 [227597] [227597]
Tmp3 [800000] [160000000000]
Tmp14 [150000] [150000]
Tmp10 [5] [25]
Tmp8 [1] [1]
Tmp9 [25] [25]
Tmp5 [10000] [10000]
Tmp16 [6000000] [6000000]
π o_orderkey o_custkey o_totalprice
Tmp17 [227597] [1500000]
Tmp1 [800000] [800000]
Tmp2 [200000] [200000]
π c_nationkey c_custkey c_acctbal c_mktsegment
Tmp7 [1] [5]
πr_regionkey
σr_name='ASIA'
πn_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_quantity
σo_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[5] [25] [10000] [6000000]
customer
region
nation
supplier
lineitem
orders
partsupp
part

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

Query	f_q	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp1, Tmp2, Tmp3, Tmp12 and result1	5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240	323,207,240,592
Q2	6	Tmp7, Tmp8, Tmp9, Tmp16, Tmp17, Tmp18, Tmp19, Tmp14, Tmp20, Tmp22 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp21, Tmp16, Tmp17, Tmp18, Tmp19, result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp7, Tmp8, Tmp9, Tmp10, Tmp5, Tmp11, Tmp1, Tmp13, Tmp14, Tmp15 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp1, Tmp2, Tmp3, Tmp4 result5	800000, 200000, 160000000000, 800000 and 36276	800,009,181,380
Q6	9	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6 and result6	800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276	1,443,281,456,484
Q7	3	Tmp16, Tmp17, Tmp18, Tmp19, Tmp14, Tmp20, result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the fifth MVPP		**25,711,148,613,207**

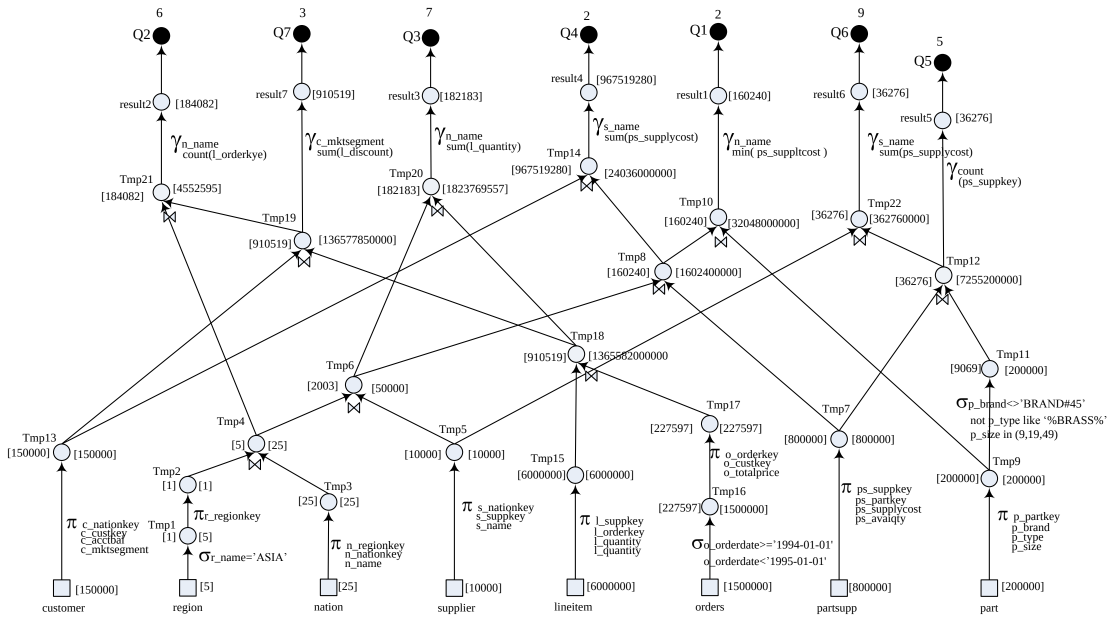


Figure A.13 The Sixth MVPP, Query in the List: {Q1, Q5, Q4, Q7, Q3, Q2, and Q6}

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

Query	f_q	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10 and result1	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp15, Tmp16, Tmp17, Tmp18, Tmp13, Tmp19, Tmp11 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp15, Tmp16, Tmp17, Tmp18, Tmp20 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp13, Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp14 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp7, Tmp9, Tmp11, Tmp12 result5	800000, 200000, 200000, 7255200000 and 36276	36,282,181,380
Q6	9	Tmp7, Tmp9, Tmp11, Tmp12 , Tmp5, Tmp22 result6	800000, 200000, 200000, 7255200000, 10000, 362760000 and 36276	68,572,856,484
Q7	3	Tmp15, Tmp16, Tmp17, Tmp18, Tmp13, Tmp19 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the sixth MVPP		**23,316,809,013,207**

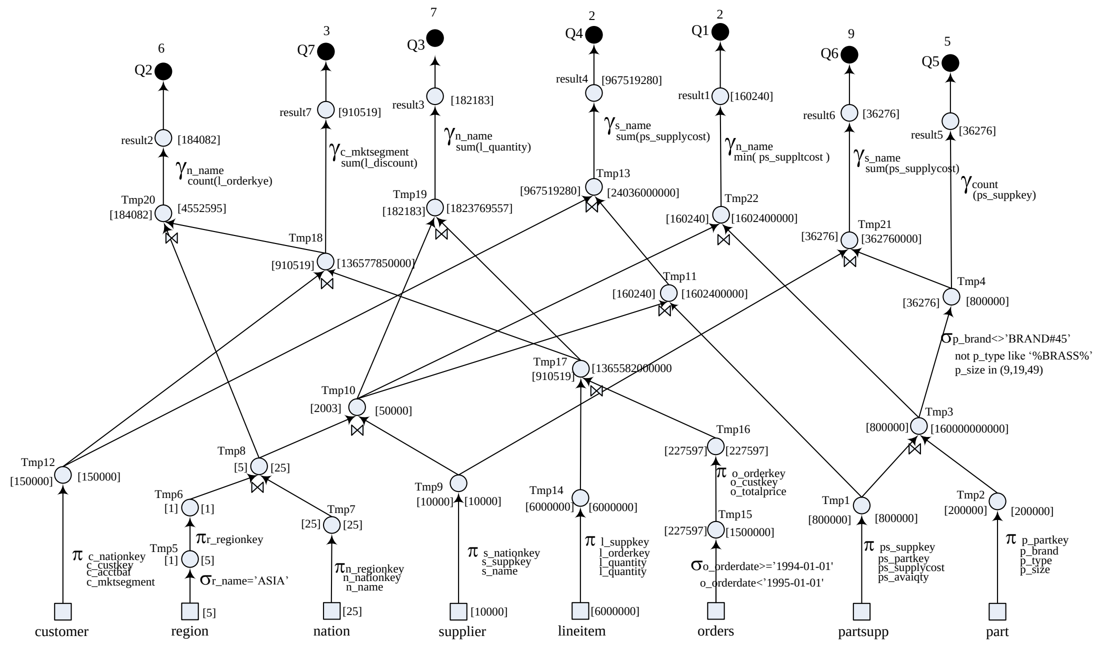
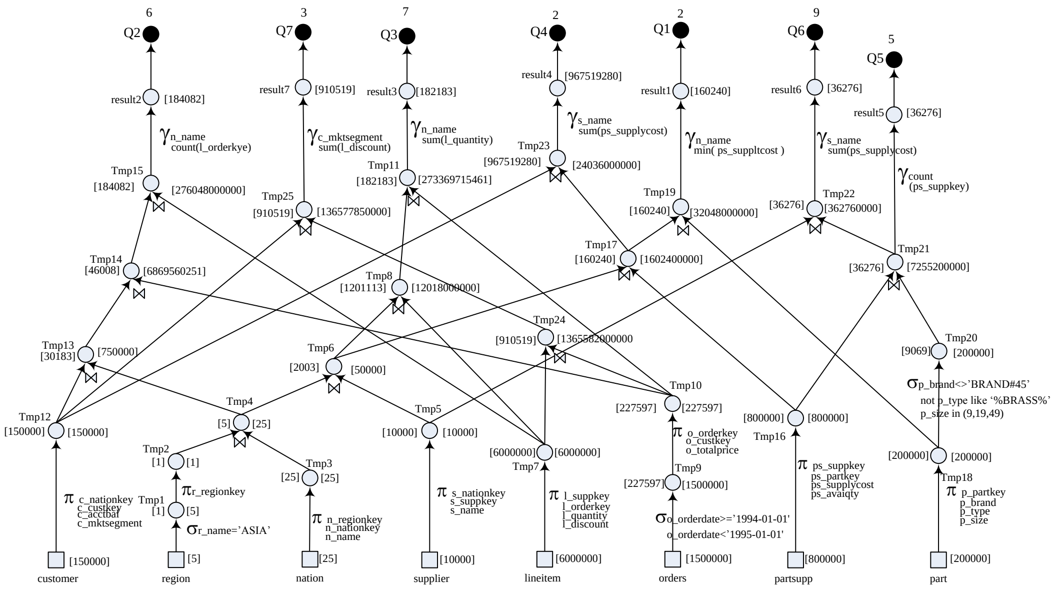


Figure A.14 The Seventh MVPP, Query in the List: {Q5, Q4, Q7, Q3, Q2, Q6, and Q1}

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

Query	f_q	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp1, Tmp2, Tmp3, Tmp22 and result1	5, 1, 25, 25, 10000, 50000, 800000, 200000, 160000000000, 1602400000 and 160240	323,207,240,592
Q2	6	Tmp5, Tmp6, Tmp7, Tmp8, Tmp14, Tmp15, Tmp16, Tmp17, Tmp12, Tmp18, Tmp20 and result2	5, 1, 25, 25, 6000000, 1500000, 227597, 1365582000000, 150000, 136577850000, 4552595 and 184082	9,013,034,785,980
Q3	7	Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp14, Tmp15, Tmp16, Tmp17, Tmp19 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 1500000, 227597, 1365582000000, 1823769557 and 182183	9,571,896,175,751
Q4	2	Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp1, Tmp11, Tmp12, Tmp13 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp1, Tmp2, Tmp3, Tmp4 and result5	800000, 200000, 160000000000, 800000 and 36276	800,009,181,380
Q6	9	Tmp1, Tmp2, Tmp3, Tmp4, Tmp9, Tmp21 and result6	800000, 200000, 160000000000, 800000, 10000, 362760000 and 36276	1,443,281,456,484
Q7	3	Tmp14, Tmp15, Tmp16, Tmp17, Tmp12, Tmp18 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
		The total query processing cost of the seventh MVPP		**25,711,148,613,207**



Q2 6
Q7 3
Q3 7
Q4 2
Q1 2
Q6 9
Q5 5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp25 [910519] [136577850000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp21 [36276] [7255200000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
Tmp24 [910519] [1365582000000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp20 [9069] [200000]
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp10 [227597] [227597]
Tmp16 [800000] [800000]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp7 [6000000] [6000000]
π o_orderkey o_custkey o_totalprice
Tmp9 [227597] [1500000]
Tmp18 [200000] [200000]
π c_nationkey c_custkey c_acctbal c_mktsegment
Tmp1 [1] [5]
π r_regionkey
σ r_name='ASIA'
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
σ o_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
customer [150000]
region [5]
nation [25]
supplier [10000]
lineitem [6000000]
orders [1500000]
partsupp [800000]
part [200000]

Figure A.15 The Cheapest MVPP after Re-Optimized.

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

Query	f_q	Access from the nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp18, Tmp19 and result1	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,240,592
Q2	6	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14, Tmp15 and result2	5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251, 6000000, 276048000000 and 184082	1,697,558,231,916
Q3	7	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp7, Tmp8, Tmp9, Tmp10, Tmp11 and result3	5, 1, 25, 25, 10000, 50000, 6000000, 12018000000, 1500000, 227597 , 273369715461 and 182183	1,997,769,797,079
Q4	2	Tmp1, Tmp2, Tmp3, Tmp4, Tmp5, Tmp6, Tmp16, Tmp17, Tmp12, Tmp23 and result4	5, 1, 25, 25, 10000, 50000, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,858,672
Q5	5	Tmp16, Tmp18, Tmp20, Tmp21 and result5	800000, 200000, 200000, 7255200000 and 36276	36,282,181,380
Q6	9	Tmp16, Tmp18, Tmp20, Tmp21, Tmp5, Tmp22 and result6	800000, 200000, 200000, 7255200000, 10000, 362760000 and 36276	68,572,856,484
Q7	3	Tmp7, Tmp9, Tmp10, Tmp24, Tmp12, Tmp25 and result7	6000000, 1500000, 227597, 1365582000000, 150000, 136577850000 and 910519	4,506,505,914,348
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 querieswhen 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

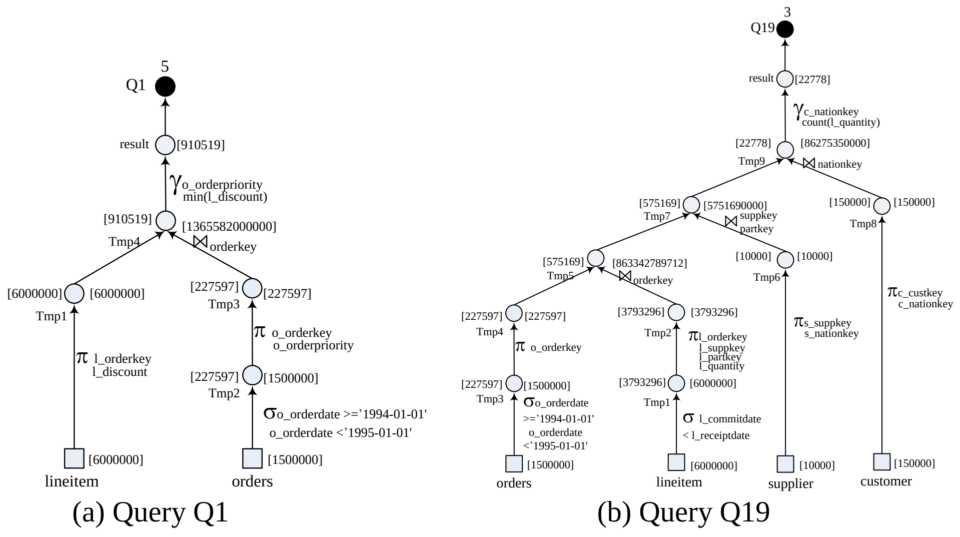
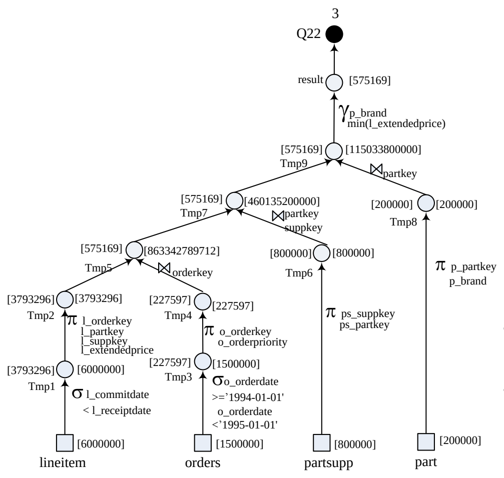


Figure B.1 Relational Algebra Query Tree of Query Q1, Q19, Q22 and Q31


(c) Query Q22

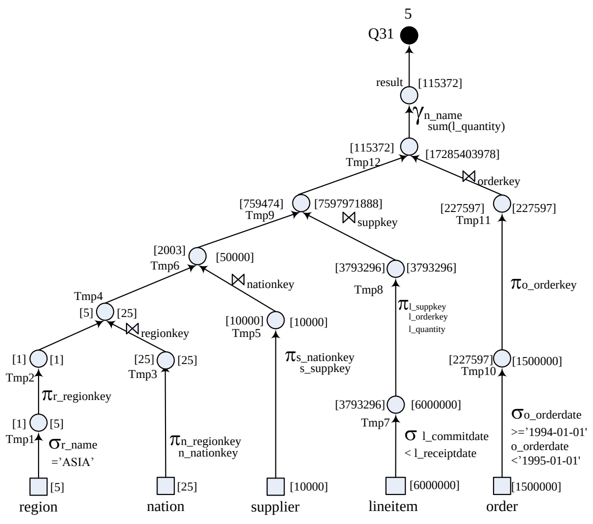

(d) Query Q31

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.

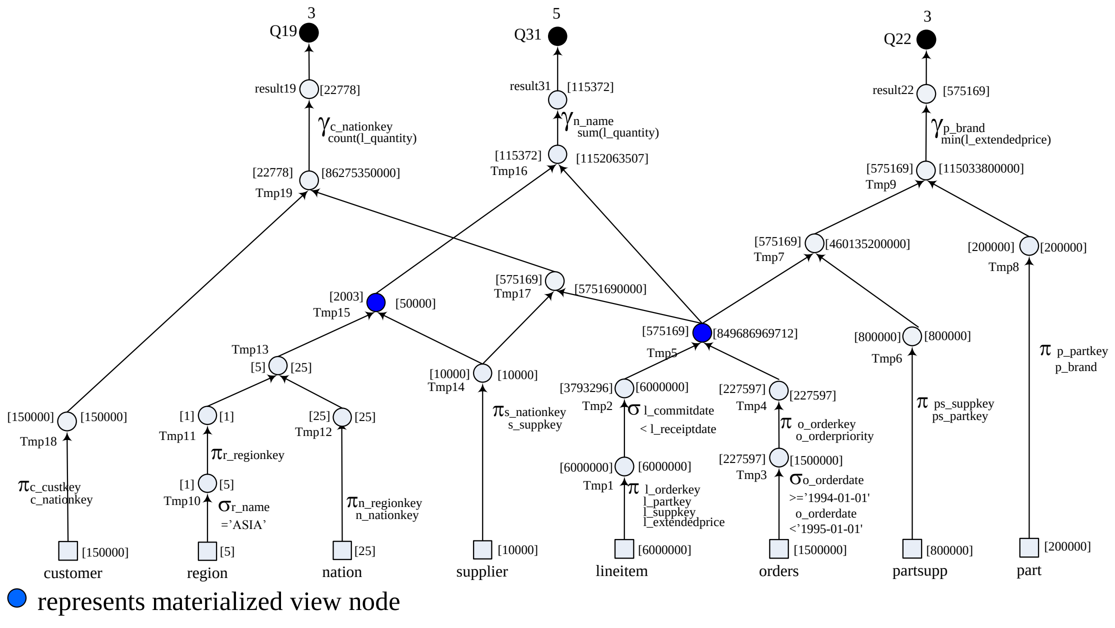

● represents materialized view node

Figure B.2 The Re-Optimized Existing MVPP in Static Phase for Q19, Q22 and Q31

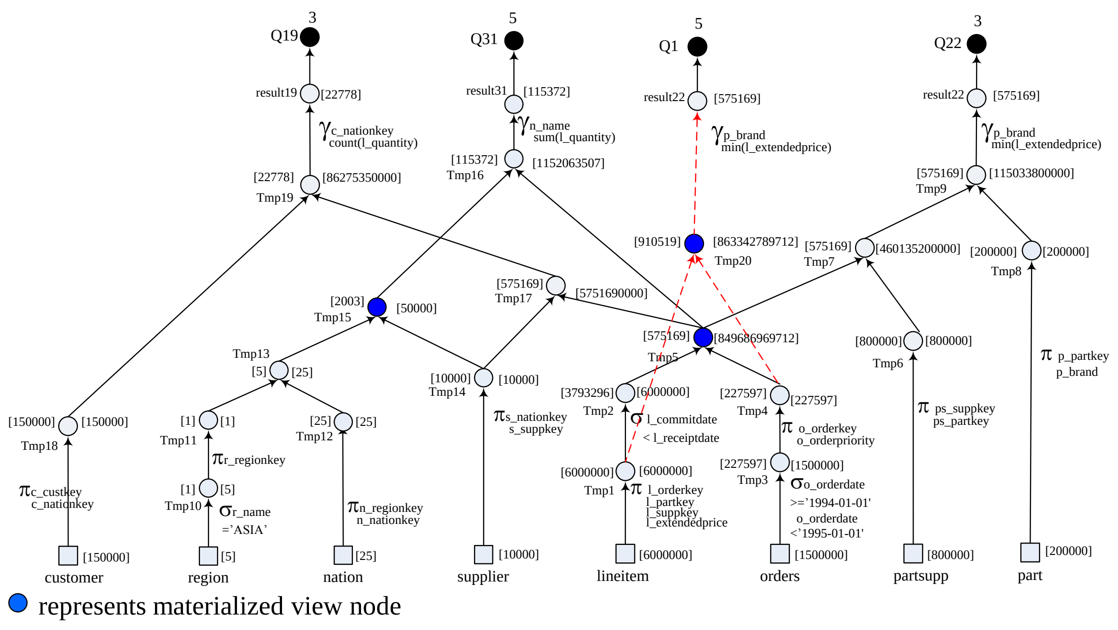


Figure B.3 The Optimal MVPP of Dynamic Phase by Adding New Query Q1

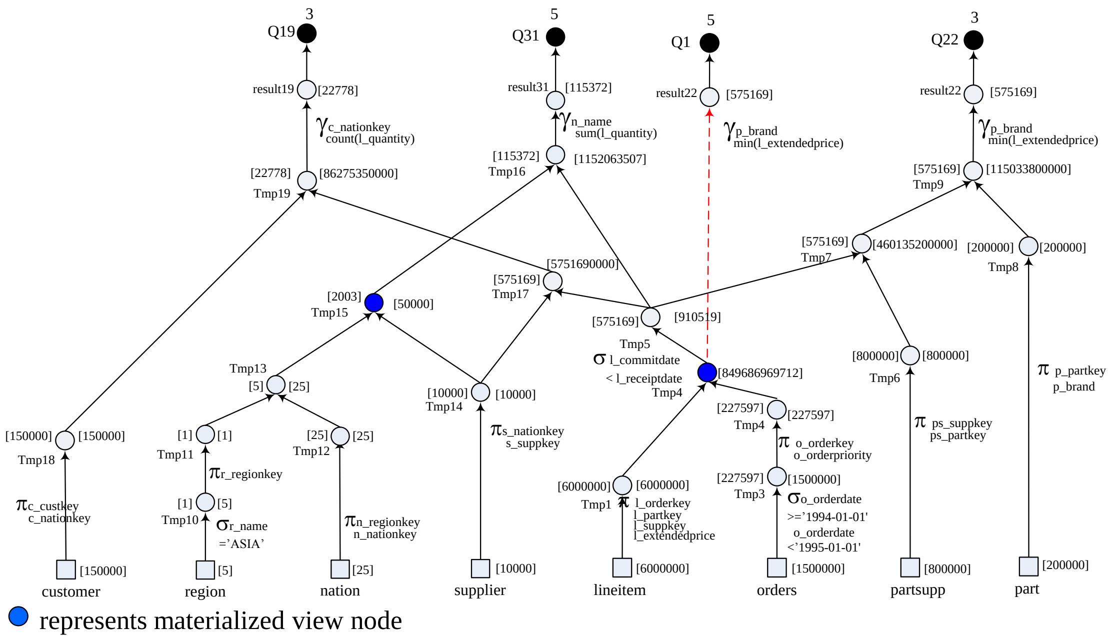


Figure B.4 The Re-Optimized Cheapest MVPP by the Static Approach for Q19, Q22, Q31 and Q1

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

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

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

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

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 ($\sigma_{\text{o\_orderdate >= '1994-01-01' and o\_orderdate<'1995-01-01'}}$), plus the additional selection ($\sigma_{\text{l\_commitdate < l\_receiptdate}}$) that is node Tmp2 in Figure B.2.

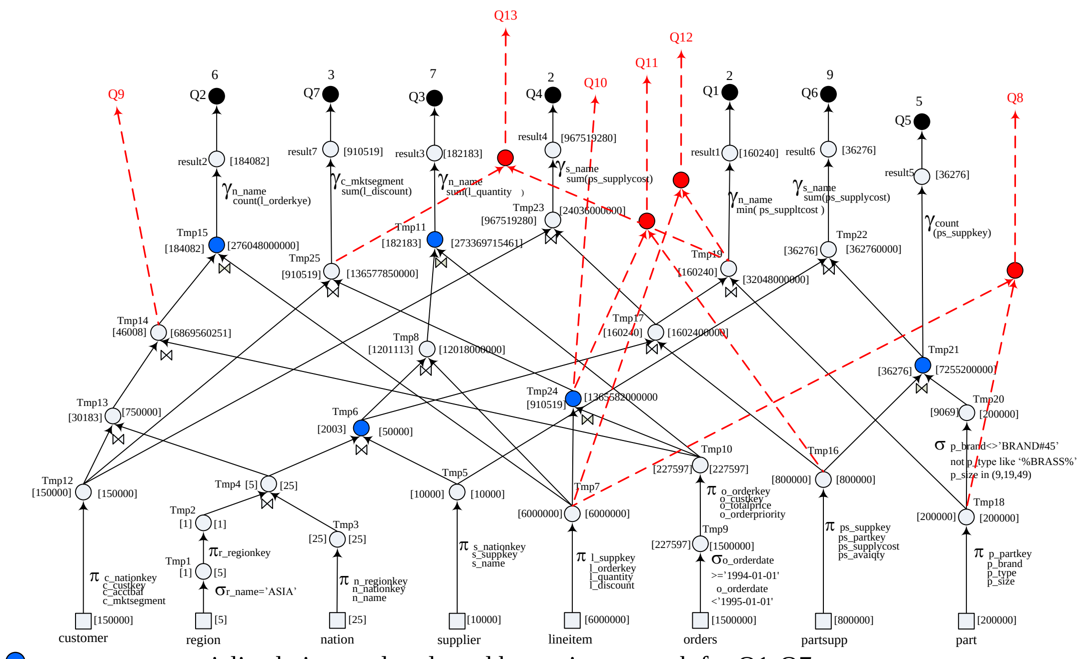
In Figure B.2 the materialized views are Tmp5 and Tmp15. After the optimal MVPP generated by dynamic approach, Figure B.3, Tmp20 is the new materialized view selected to support new query Q1. Comparing the dynamic approach result with static approach, Figure B.4, the number of materialized views of dynamic 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

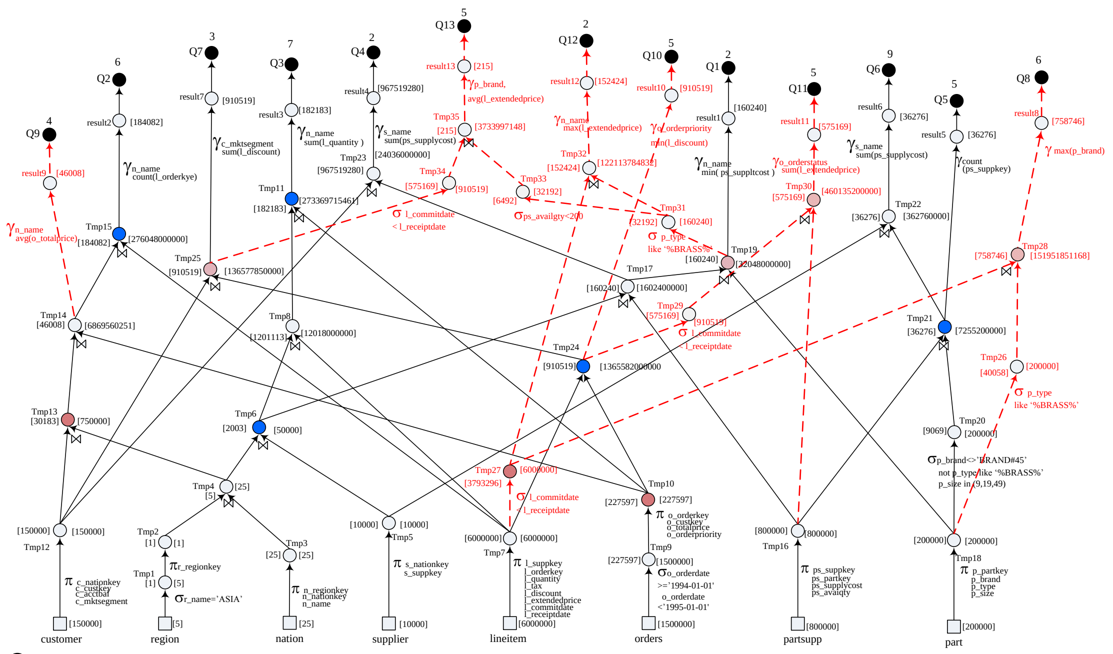
Normally there possibly to have more than one alternative plans when new query is merged into search space. This example shows the experiment how to choose the query processing plan for new query when new query is merged into the existing MVPP if new query has many query processing plan. We use the experiment 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 MVPP that Q12 already in the MVPP.

Plan 1: {(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART)} join with {(LINEITEM) ⋈ ORDERS ⋈ 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

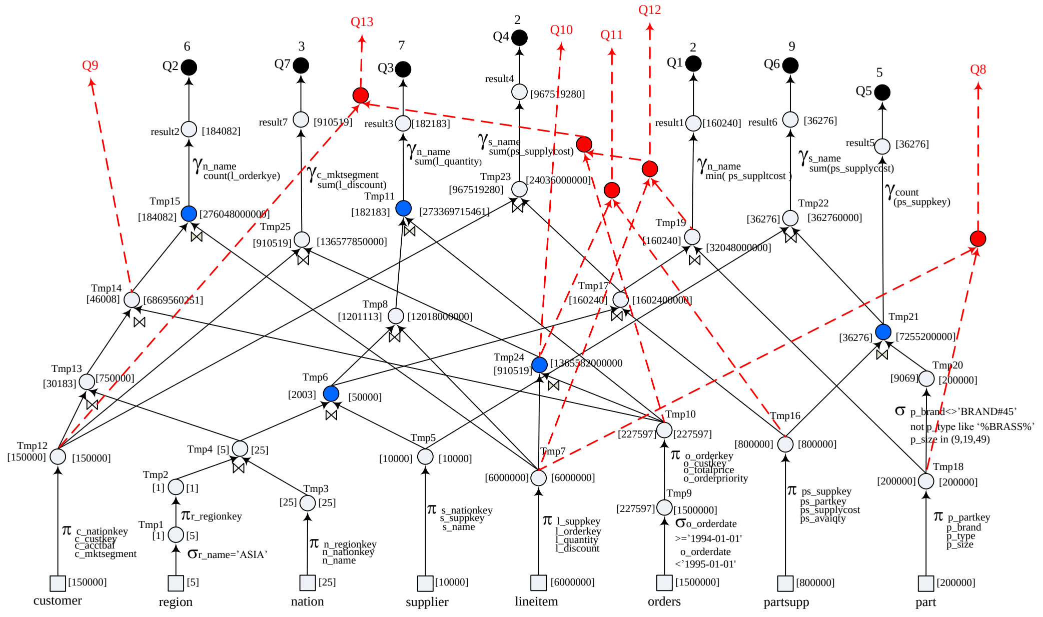


● represents materialized view node selected by static approach for Q1-Q7
● represents new materialized view node selected by dynamic approach

Figure B.6 Plan 1: The Dynamic Cheapest MVPP of Q8-Q13 after Optimized

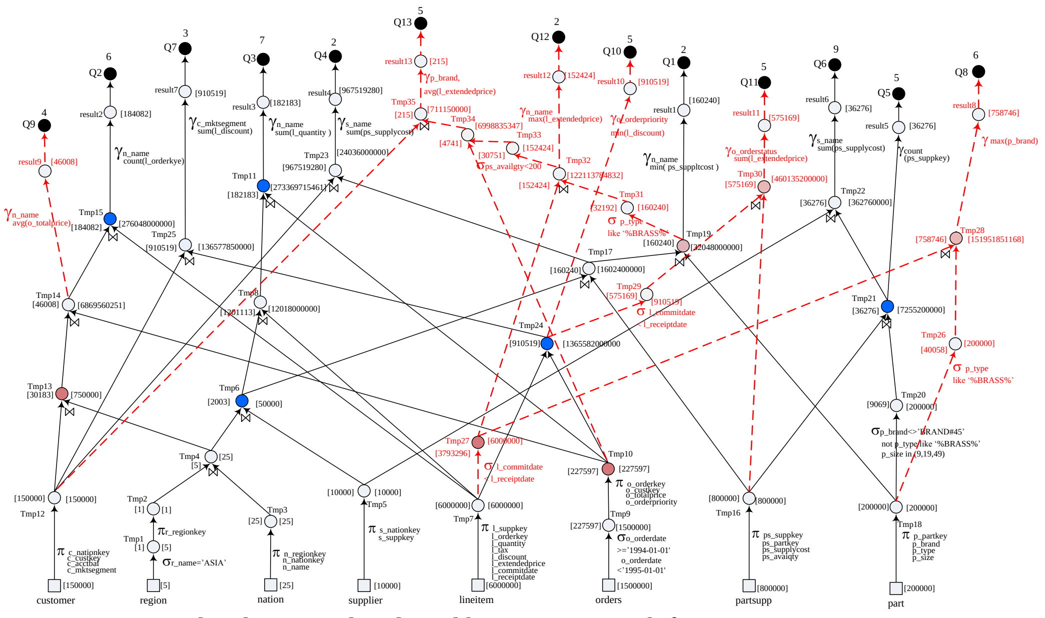
Plan 2: {(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) ⋈ LINEITEM} ⋈ ORDERS ⋈ CUSTOMER.

{(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) ⋈ LINEITEM} is the intermediate result of Q12



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

Figure B.7 Plan 2: 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.8 Plan 2: The Re-Optimized Cheapest Dynamic MVPP of Q8-Q13

The first subgraph, {(REGION ⋈ NATION ⋈ SUPPLIER ⋈ PARTSUPP ⋈ PART) ⋈ 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

Plan	Accessed from the nodes	Cost of node	Query processing cost
1	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp33, Tmp24(materialized view), Tmp12, Tmp25, Tmp34, Tmp35 and result13	2003, 800000, 1602400000, 200000, 32048000000, 160240, 32192, 910519, 150000, 136577850000, 910519, 3733997148, 215	869,827,064,180
2	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32, Tmp33, Tmp9, Tmp10, Tmp34, Tmp12, Tmp35 and result13	2003, 800000, 1602400000, 200000, 32048000000, 160240, 6000000,6000000,122113784832,152424, 1500000, 227597, 6998835347, 150000, 711150000, 215	817,446,813,290

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.

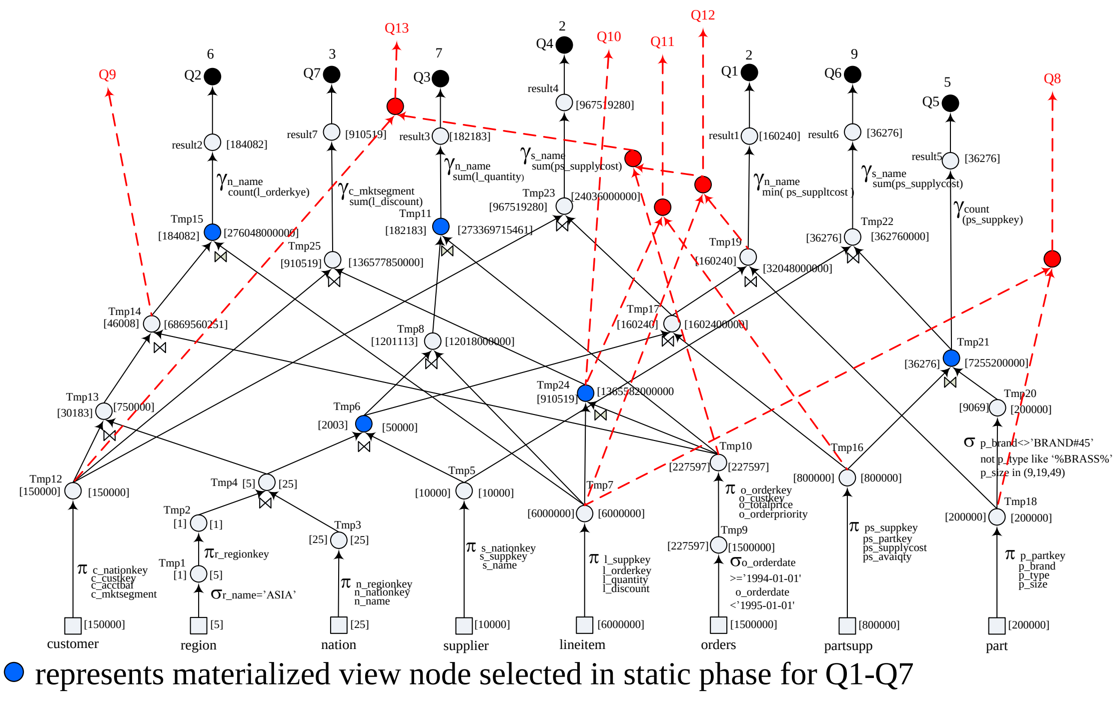


Figure C.1 The First Dynamic MVPP after All New Queries are Merged

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.

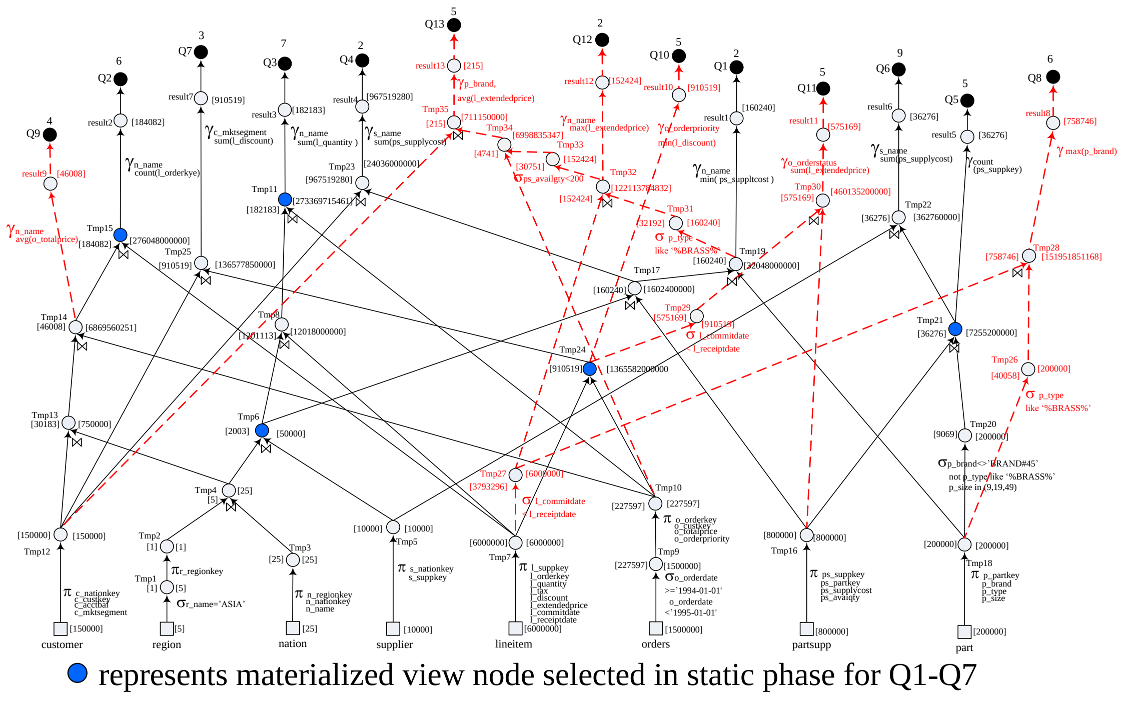


Figure C.2 The First Dynamic MVPP after Optimized

Table C.1 The Query Processing Cost of the First to the Fourth Dynamic MVPP

Query	f_q	List of Accessed Nodes	Cost of Nodes	Query Processing Cost
Q1	2	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19 and result1	2003, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,124,486
Q2	6	Tmp15(materialized view) and result2	184082 and 184082	2,208,984
Q3	7	Tmp11(materialized view) and result3	182183 and 182183	2,550,562
Q4	2	Tmp6(materialized view), Tmp16, Tmp17,Tmp12, Tmp23 and result4	2003, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,742,566
Q5	5	Tmp21(materialized view) and result5	36276 and 36276	362,760
Q6	9	Tmp21(materialized view), Tmp5, Tmp22, and result6	36276, 10000, 362760000 and 36276	3,265,582,968
Q7	3	Tmp24(materialized view), Tmp12, Tmp25 and result7	910519, 150000, 136577850000 and 910519	409,739,463,114
Q8	6	Tmp7, Tmp27, Tmp18, Tmp26, Tmp28 and result8	6000000, 6000000, 200000, 200000, 151951851168 and 758746	911,790,059,484
Q9	4	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14 and result 9	5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251 and 46008	27,488,935,648
Q10	5	Tmp24(materialized view) and result10	910519, 910519	9,105,190
Q11	5	Tmp24(materialized view), Tmp29, Tmp16, Tmp30 and result11	910519, 575169, 800000, 460135200000 and 575169	2,300,691,981,035

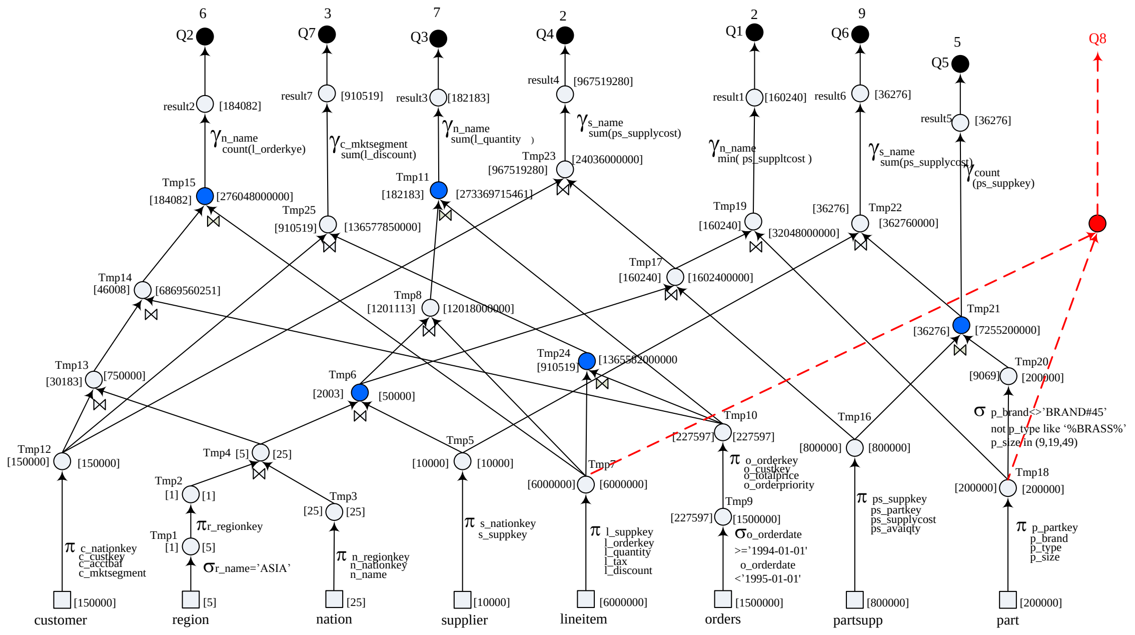
Table C.1 (Continued)

Query	f_q	List of Accessed Nodes	Cost of Nodes	Query Processing Cost
Q12	2	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32 and result12	2003, 800000, 1602400000, 200000, 32048000000, 160240, 6000000, 6000000, 122113784832 and 152424	311,554,998,998
Q13	5	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp7, Tmp27, Tmp31, Tmp32, Tmp33, Tmp9, Tmp10, Tmp34, Tmp12, Tmp35 and result13	2003, 800000, 1602400000, 200000, 32048000000, 160240, 6000000, 6000000, 122113784832, 152424, 1500000, 227597, 6998835347, 150000, 711150000 and 215	817,446,813,290
The total query processing cost of the first MVPP				**4,902,508,929,085**

C.2 The Second Dynamic MVPP

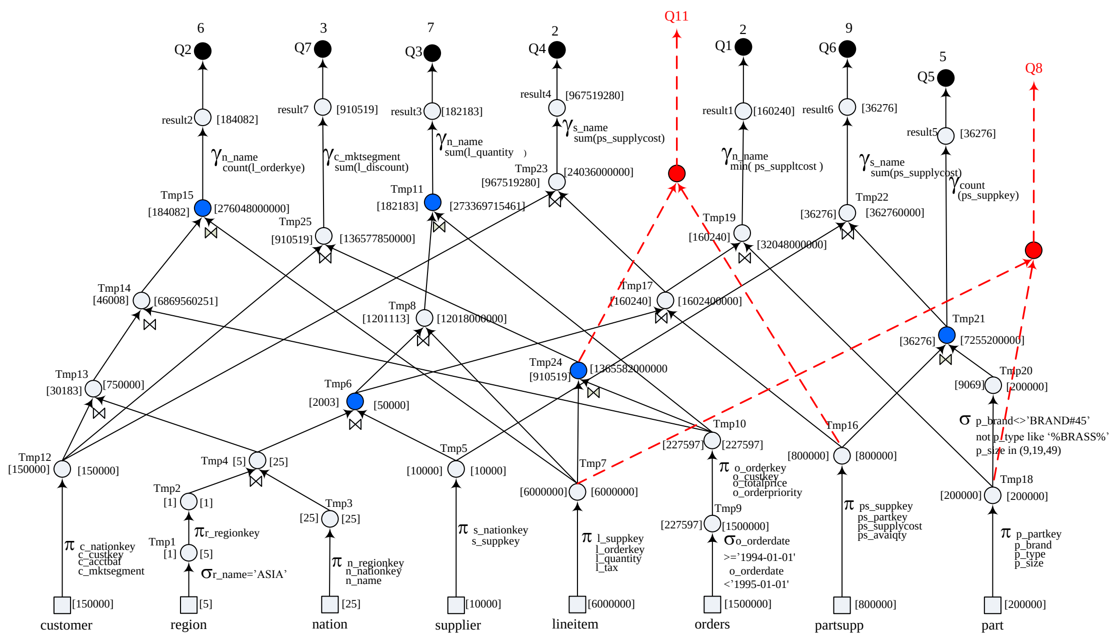
After the first dynamic MVPP is generated, the first element of the list is moved to the end of the list. So Q10 is moved to the end of list, the list becomes {Q8, Q11, Q12, Q9, Q13 and Q10}.

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 ⋈ PARTSUPP}. The second dynamic MVPP when Q11 is merged, is shown as Figure C.3 (b).



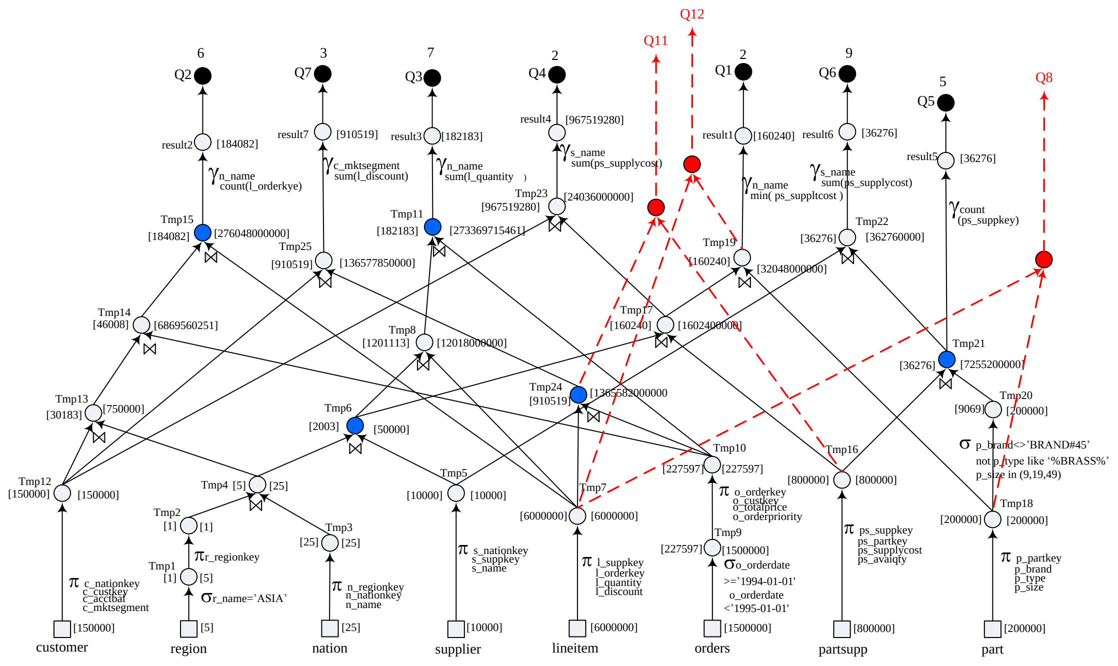
● represents materialized view node selected in static phase for Q1-Q7

Figure C.3 (a) The Second Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP



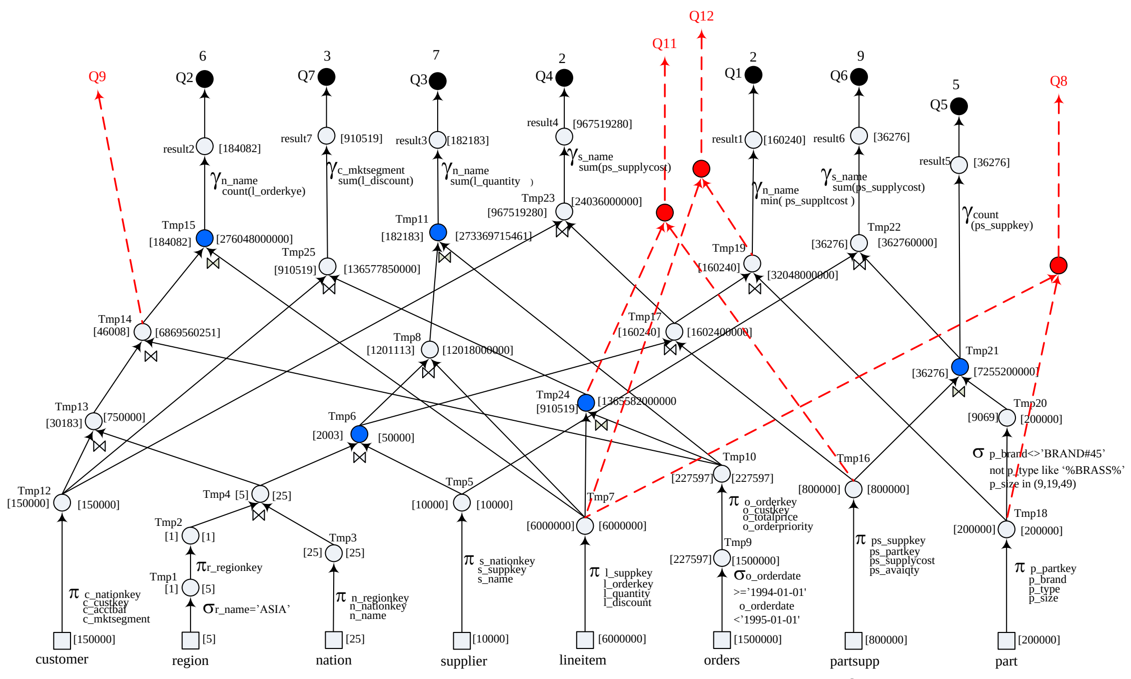
● represents materialized view node selected in static phase for Q1-Q7

Figure C.3 (b) The Second Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP



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

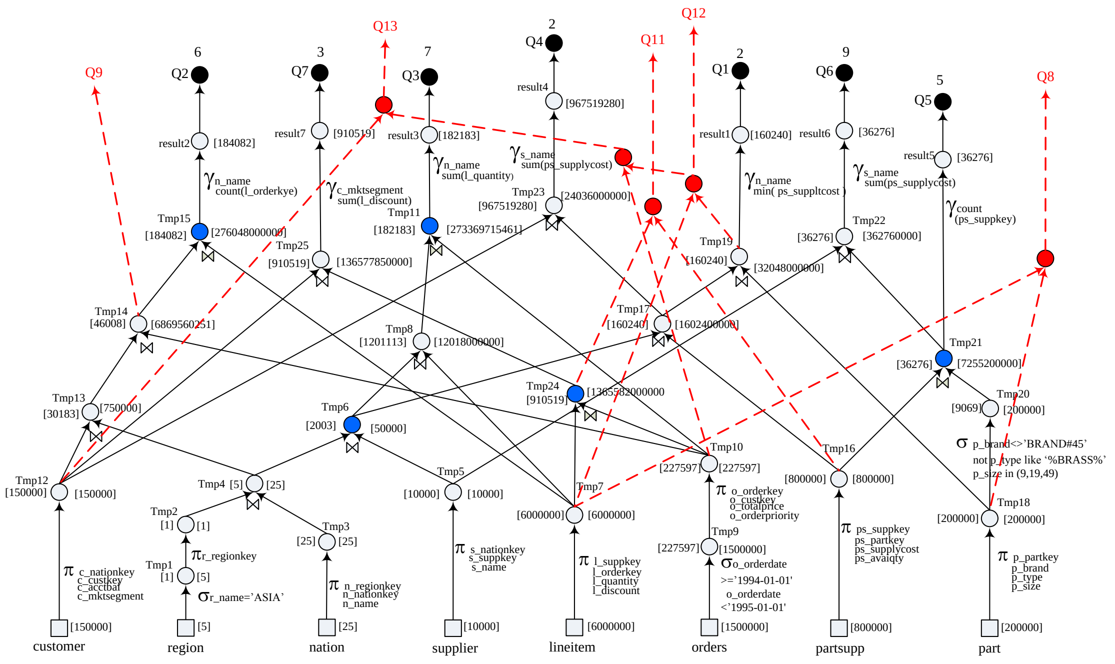
Figure C.3 (c) The Second Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP



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

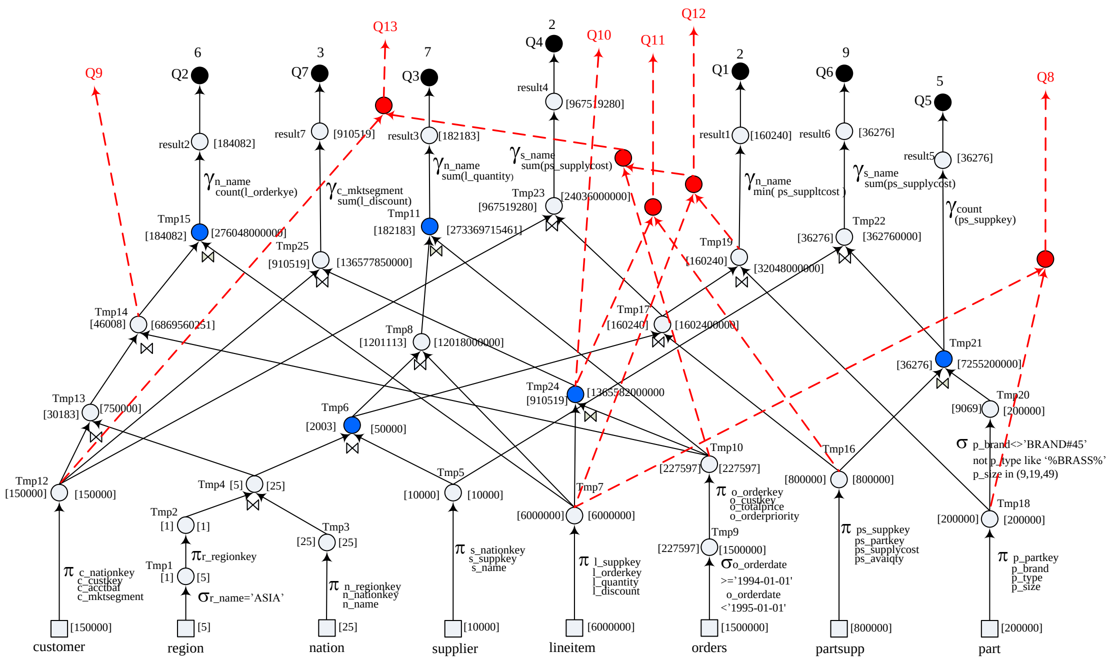
Figure C.3 (d) The Second Dynamic MVPP after Merging Q9 into the Existing Re-Optimized MVPP

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 ⋈ LINEITEM) that is already available in the existing MVPP, Tmp24. However, Tmp24 includes the select operation ($\sigma_{\text{o\_orderdate>='1994-01-01' and o\_orderdate <'1995-01-01'}}$ 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).



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

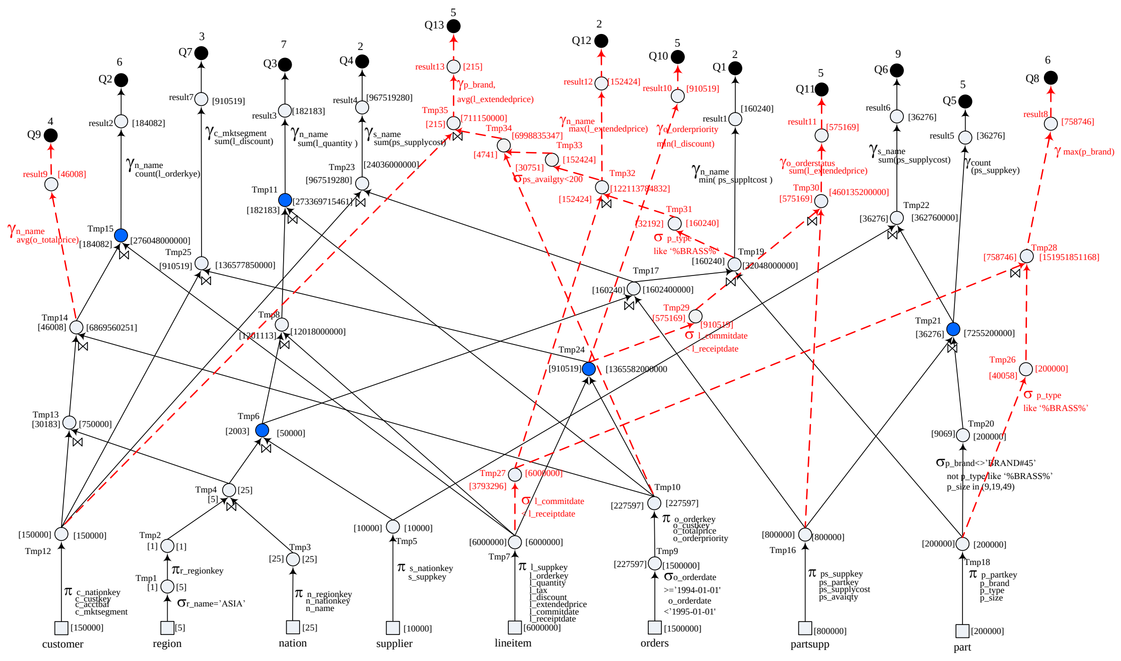
Figure C.3 (e) The Second Dynamic MVPP after Merging Q13 into the Existing Re-Optimized MVPP



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

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.



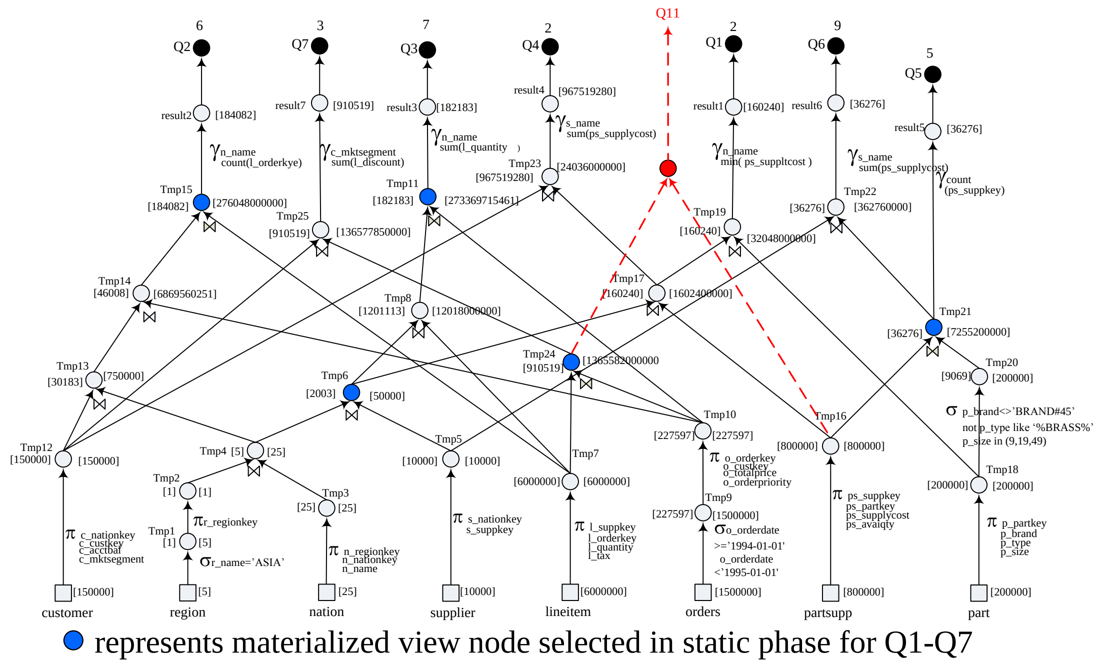
● represents materialized view node selected in static phase for Q1-Q7

Figure C.4 The Second Dynamic MVPP after Optimized

C.3 The Third Dynamic MVPP

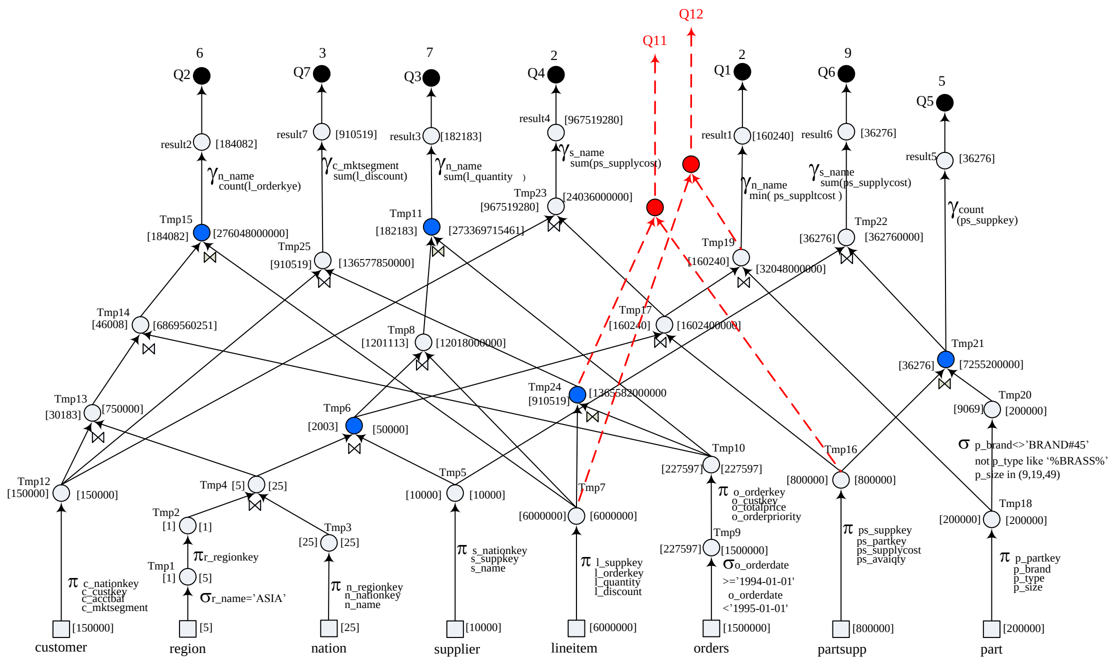
After the second dynamic MVPP is generated, the first element of the list is moved to the end of the list. So, Q8 is moved to the end of list, the list becomes {Q11, Q12, Q9, Q13, Q10 and Q8}.

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 ⋈ 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 ($\sigma_{\text{o\_orderdate>='1994-01-01' and o\_orderdate <'1995-01-01'}}$ 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).



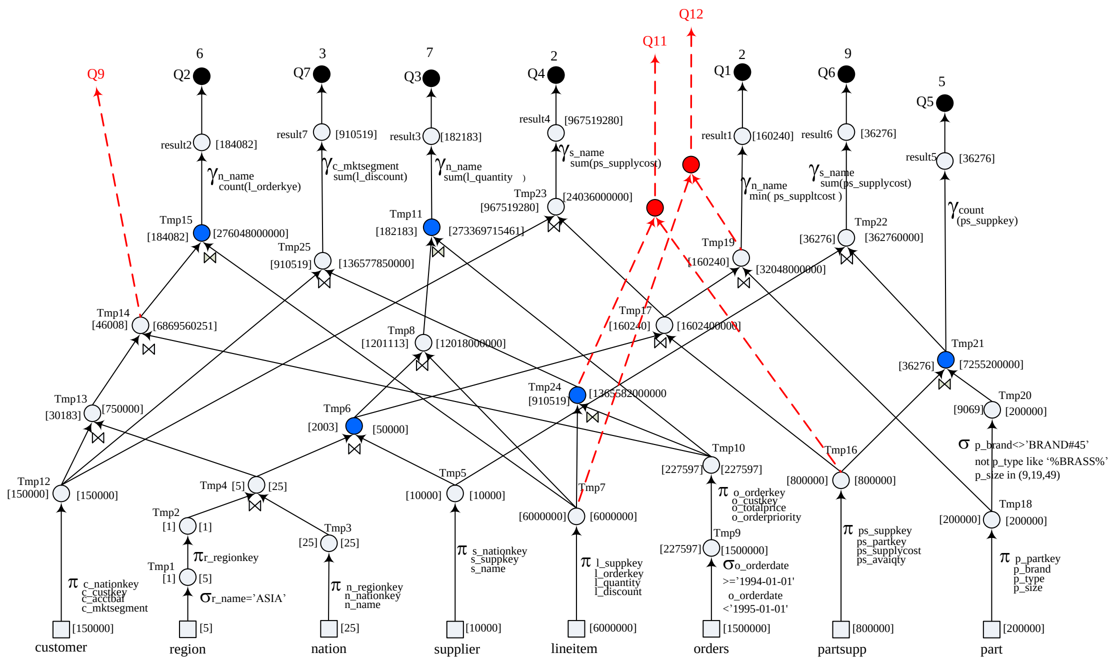
● represents materialized view node selected in static phase for Q1-Q7

Figure C.5 (a) The Third Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP



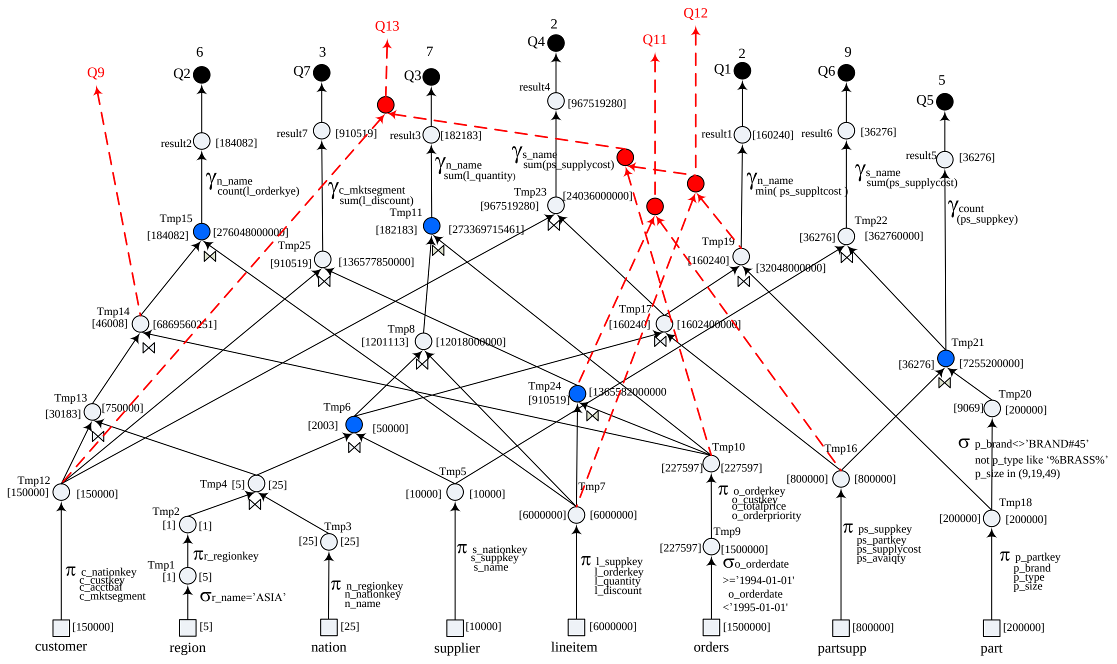
● represents materialized view node selected in static phase for Q1-Q7

Figure C.5 (b) The Third Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP



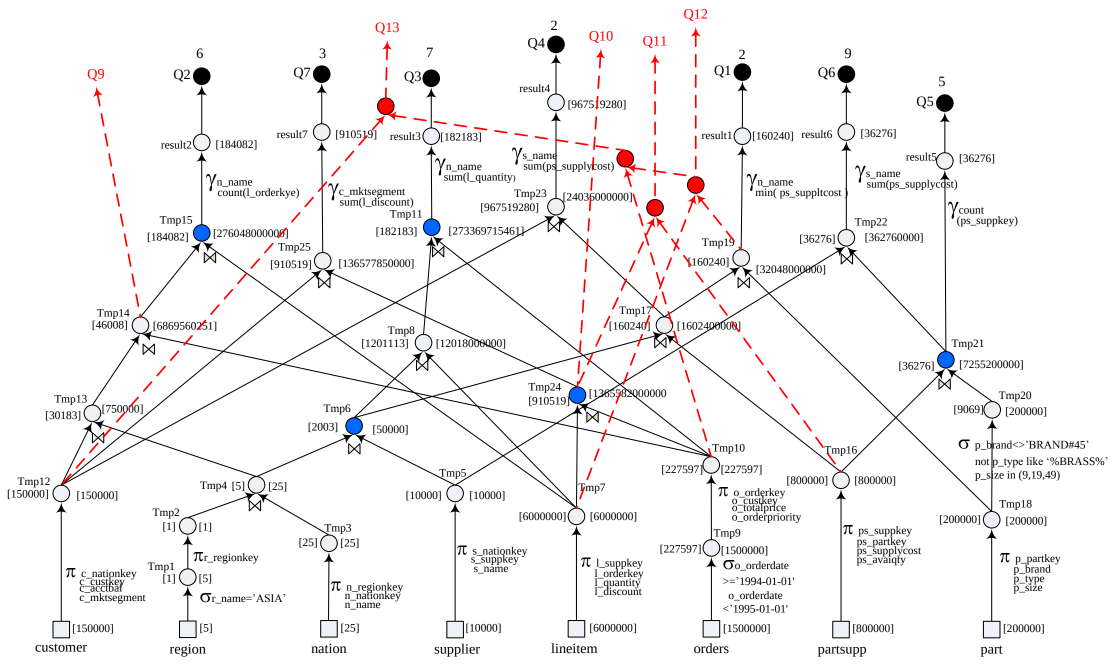
● represents materialized view node selected in static phase for Q1-Q7

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



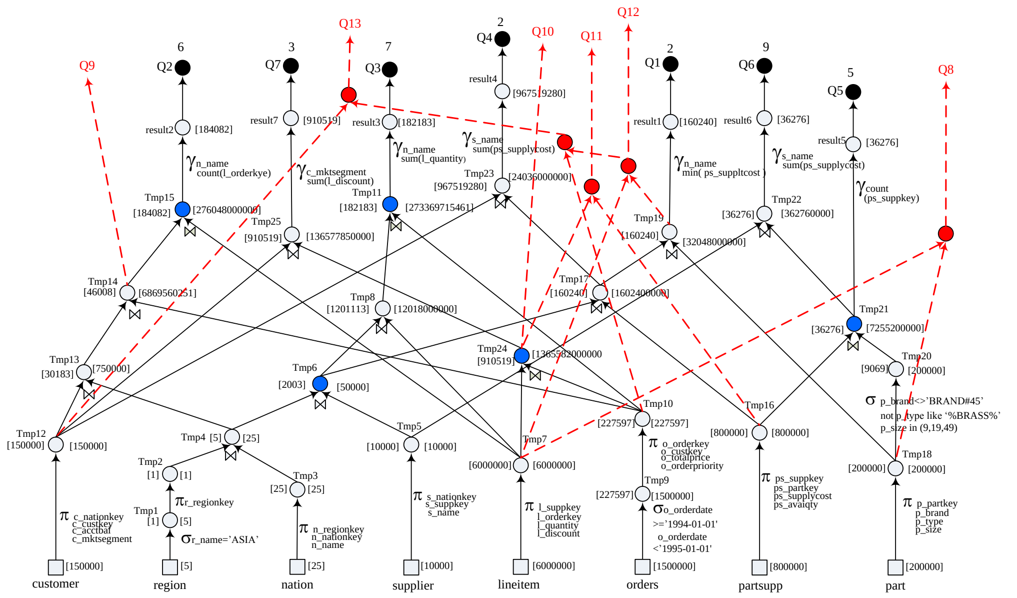
● represents materialized view node selected in static phase for Q1-Q7

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



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

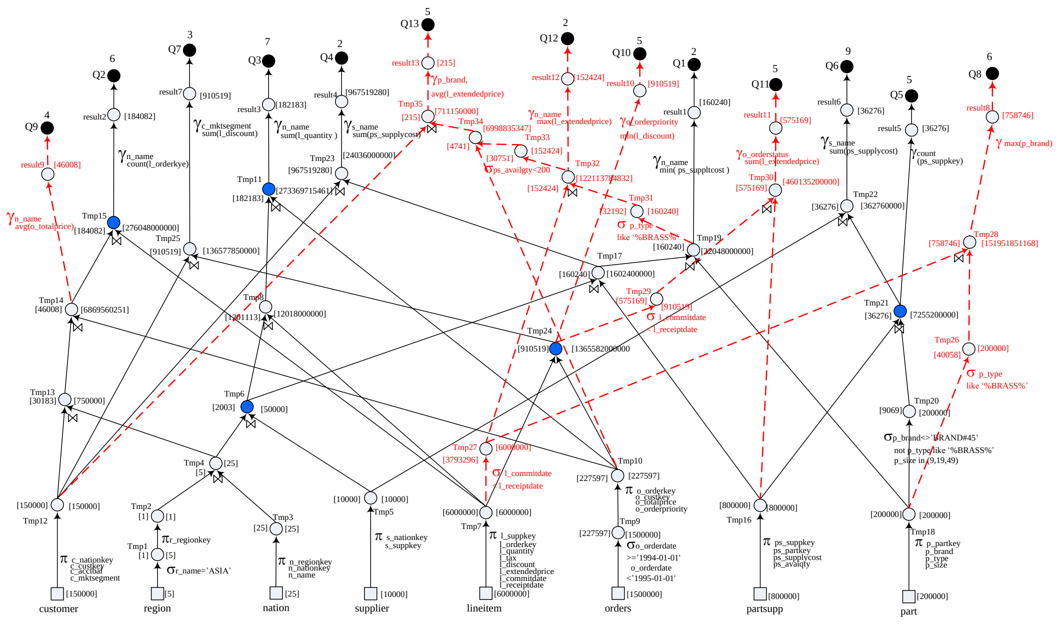
Figure C.5 (e) The Third Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP



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

Figure C.5 (f) The Third Dynamic MVPP after All New Queries are Merged

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.



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

Figure C.6 The Third Dynamic MVPP after Optimized

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}

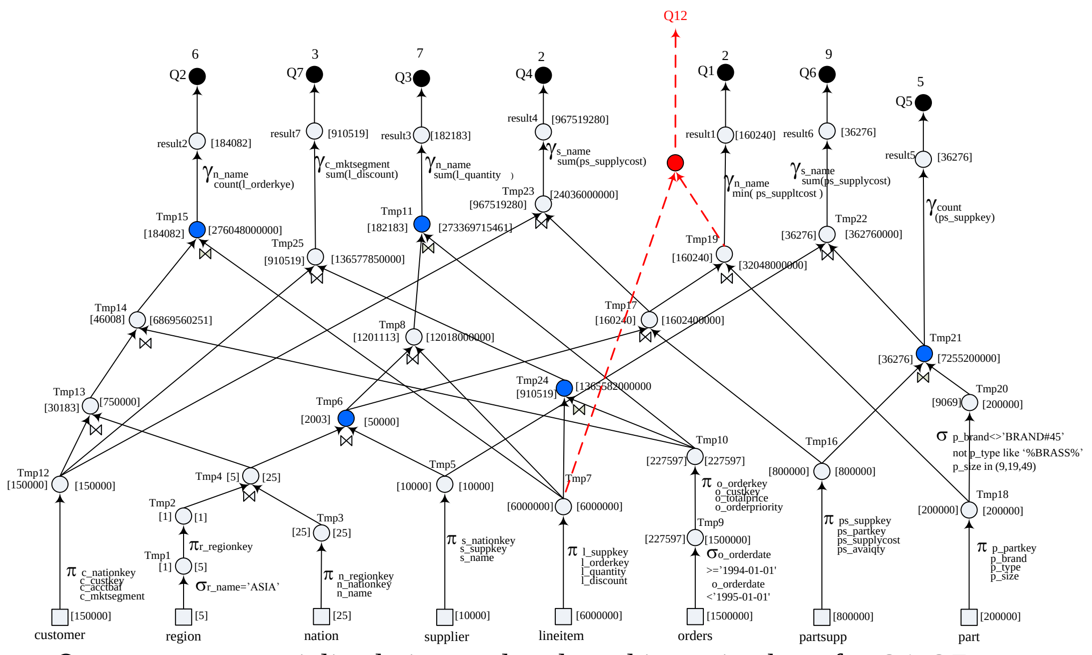
The fifth dynamic MVPP : {Q9, Q13, Q10, Q8, Q11 and Q12}

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.

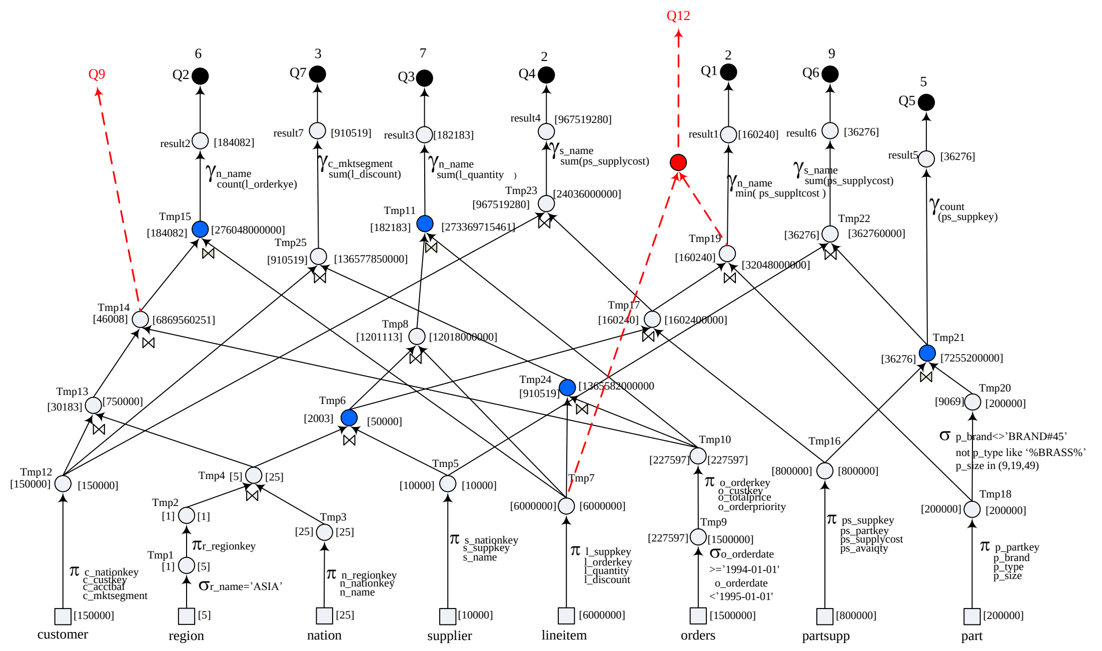
C.4 The Fourth Dynamic MVPP

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



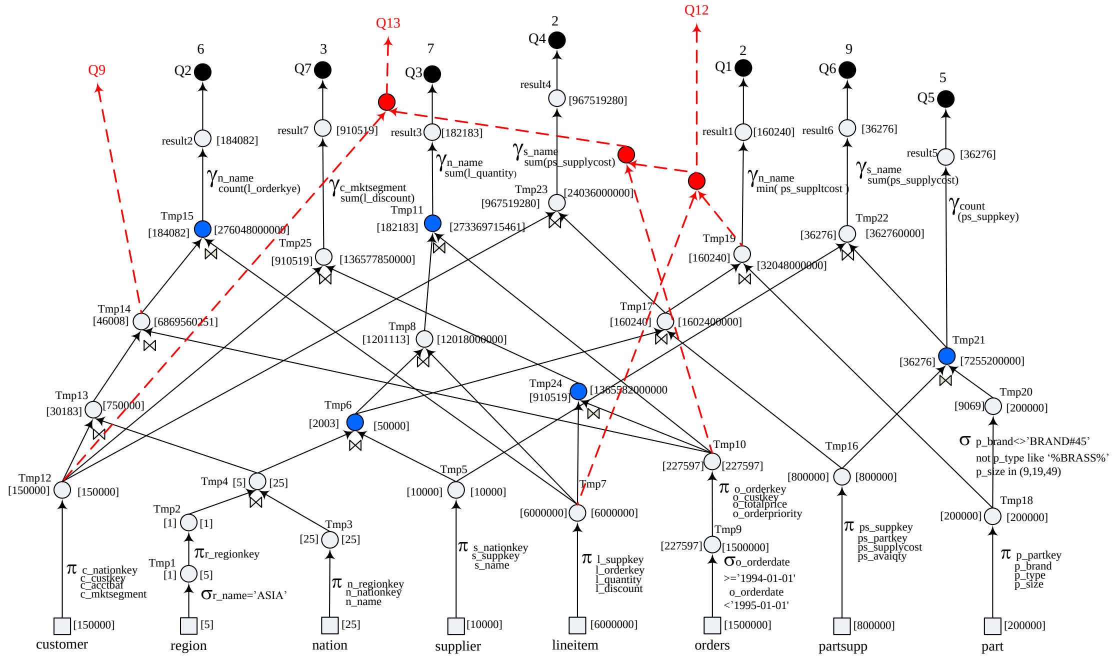
● represents materialized view node selected in static phase for Q1-Q7

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

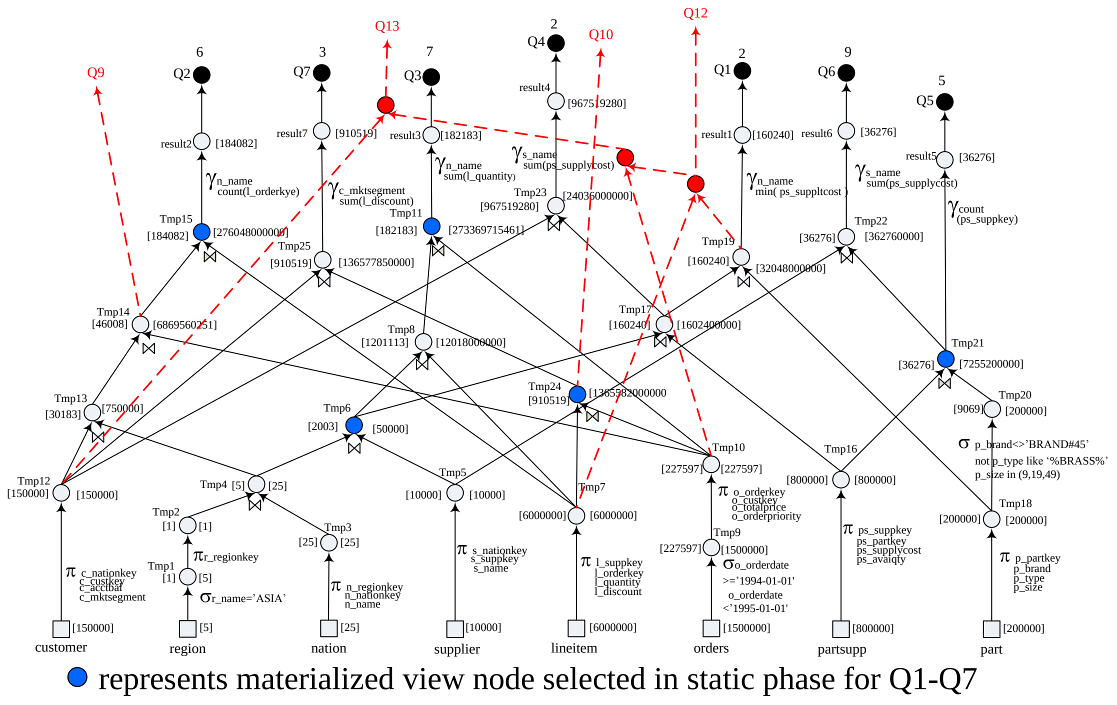


Figure C.7 (d) The Third Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP

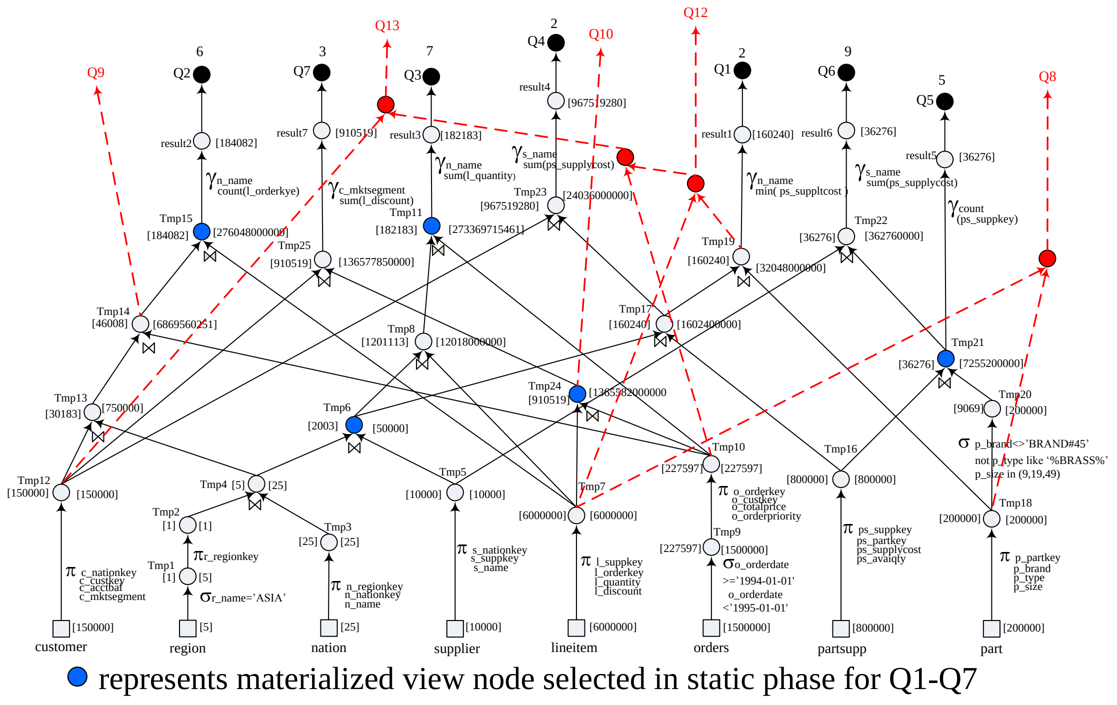
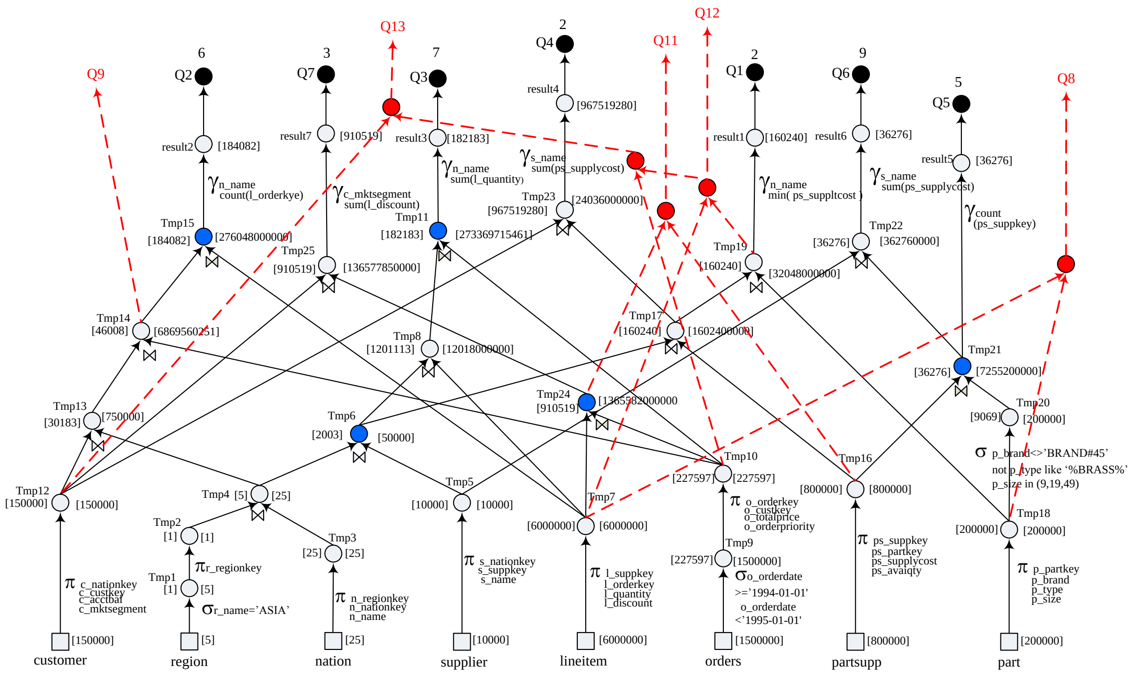


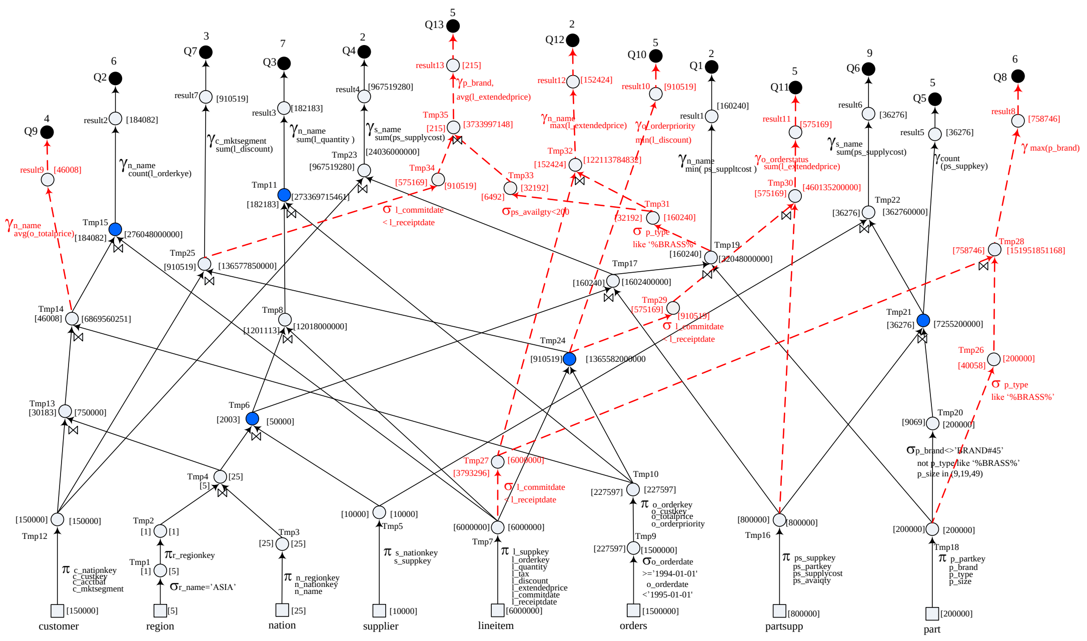
Figure C.7 (e) The Third Dynamic MVPP after Merging Q8 into the Existing Re-Optimized MVPP



● 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.

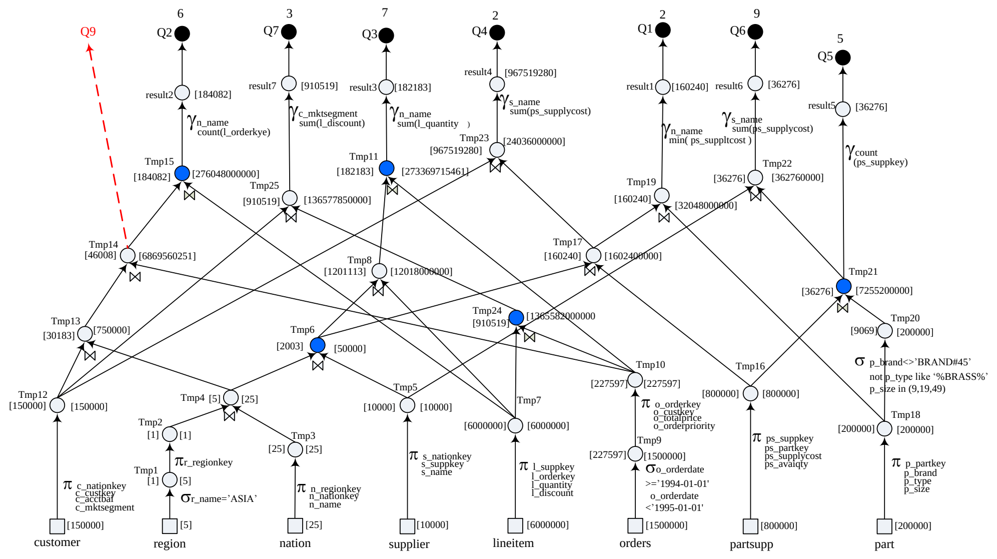


● represents materialized view node selected in static phase for Q1-Q7

Figure C.8 The Fourth Dynamic MVPP after Optimized

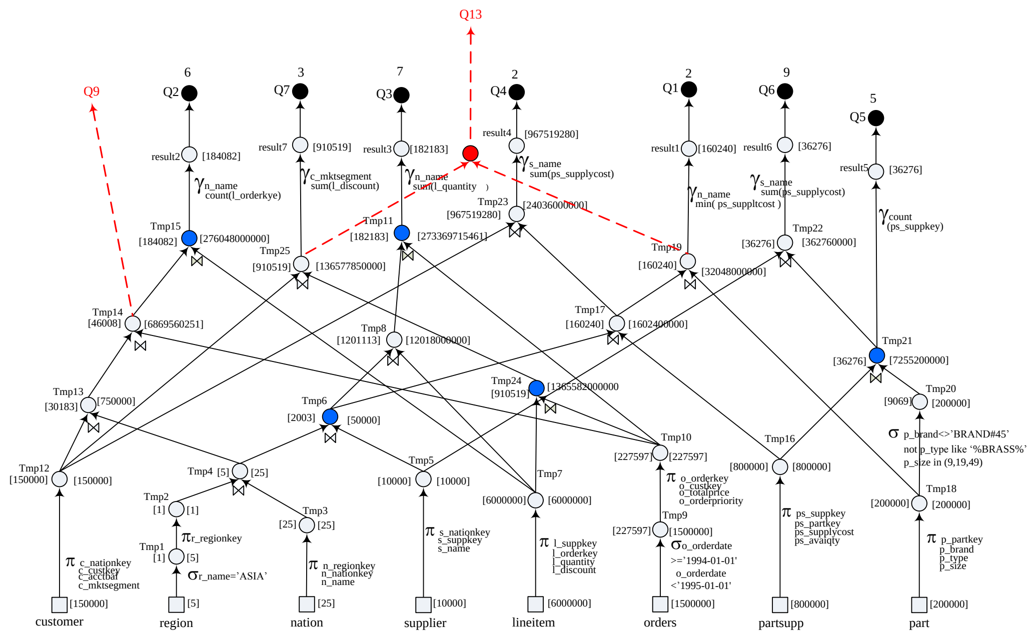
C.5 The Fifth Dynamic MVPP

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



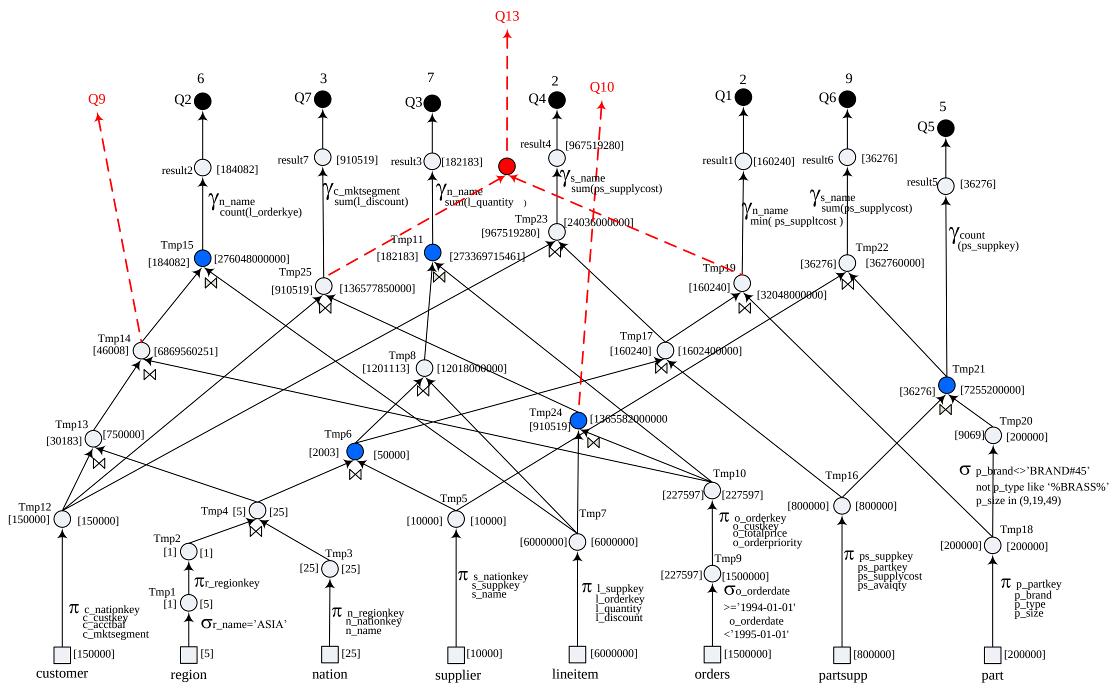
● represents materialized view node selected in static phase for Q1-Q7

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



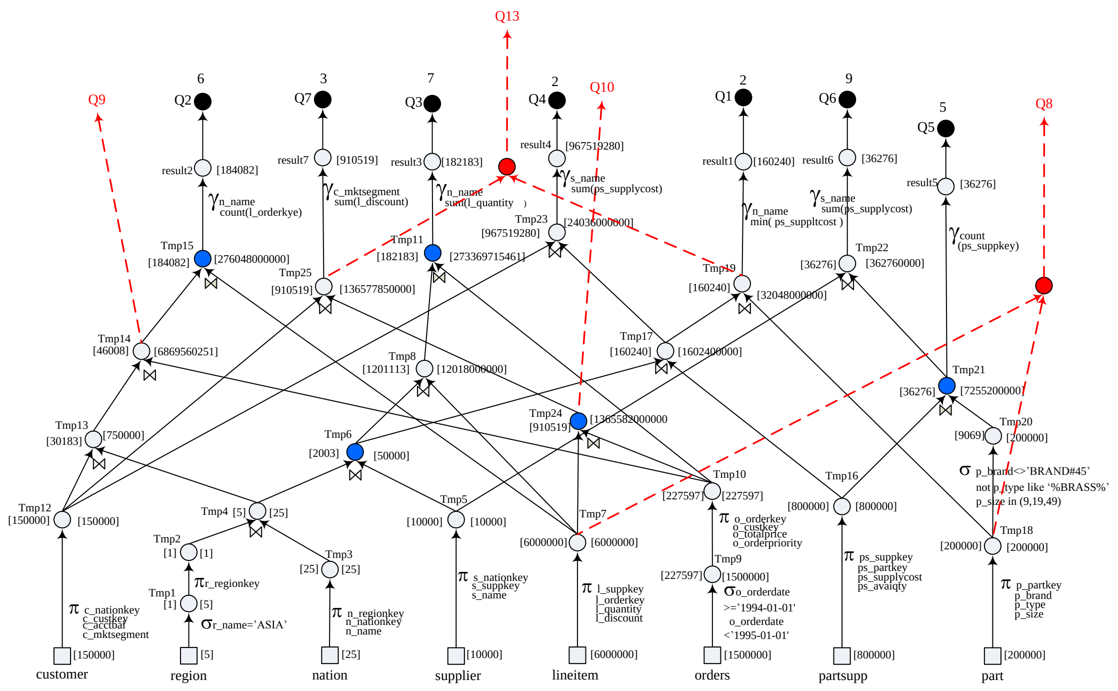
● represents materialized view node selected in static phase for Q1-Q7

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


Q13
Q9
Q10
6 Q2
3 Q7
7 Q3
2 Q4
2 Q1
9 Q6
5 Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp25 [910519] [136577850000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
Tmp21 [36276] [7255200000]
Tmp24 [910519] [136558200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp20 [9069] [200000]
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp10 [227597] [227597]
Tmp16 [800000] [800000]
Tmp7 [6000000] [6000000]
Tmp18 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
π o_orderkey o_custkey o_totalprice o_orderpriority
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
Tmp9 [227597] [1500000]
σ o_orderdate >='1994-01-01' o_orderdate <'1995-01-01'
π r_regionkey
Tmp1 [1] [5]
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part

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

Figure C.9 (c) The Fifth Dynamic MVPP after Merging Q10 into the Existing Re-Optimized MVPP


Q13
Q9
Q10
Q8
6 Q2
3 Q7
7 Q3
2 Q4
2 Q1
9 Q6
5 Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp25 [910519] [136577850000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [1602400000]
Tmp21 [36276] [7255200000]
Tmp24 [910519] [136558200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp20 [9069] [200000]
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp10 [227597] [227597]
Tmp16 [800000] [800000]
Tmp7 [6000000] [6000000]
Tmp18 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
π o_orderkey o_custkey o_totalprice o_orderpriority
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
Tmp9 [227597] [1500000]
σ o_orderdate >='1994-01-01' o_orderdate <'1995-01-01'
π r_regionkey
Tmp1 [1] [5]
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part

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

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

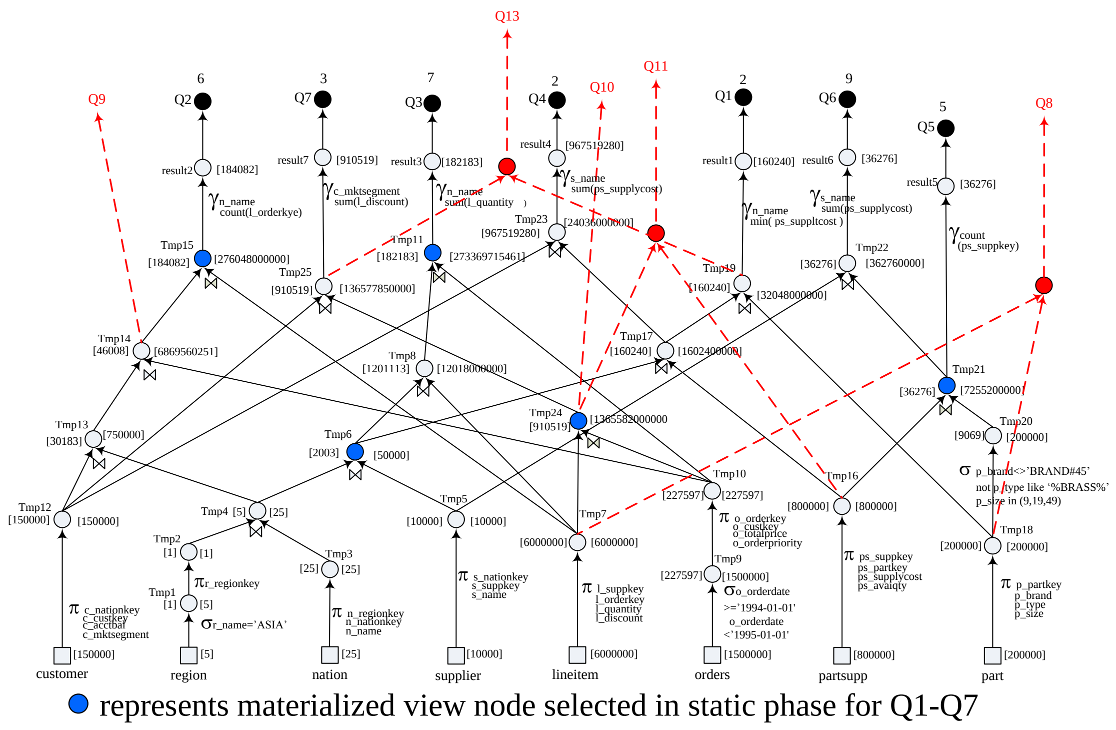


Figure C.9 (e) The Fifth Dynamic MVPP after Merging Q11 into the Existing Re-Optimized MVPP

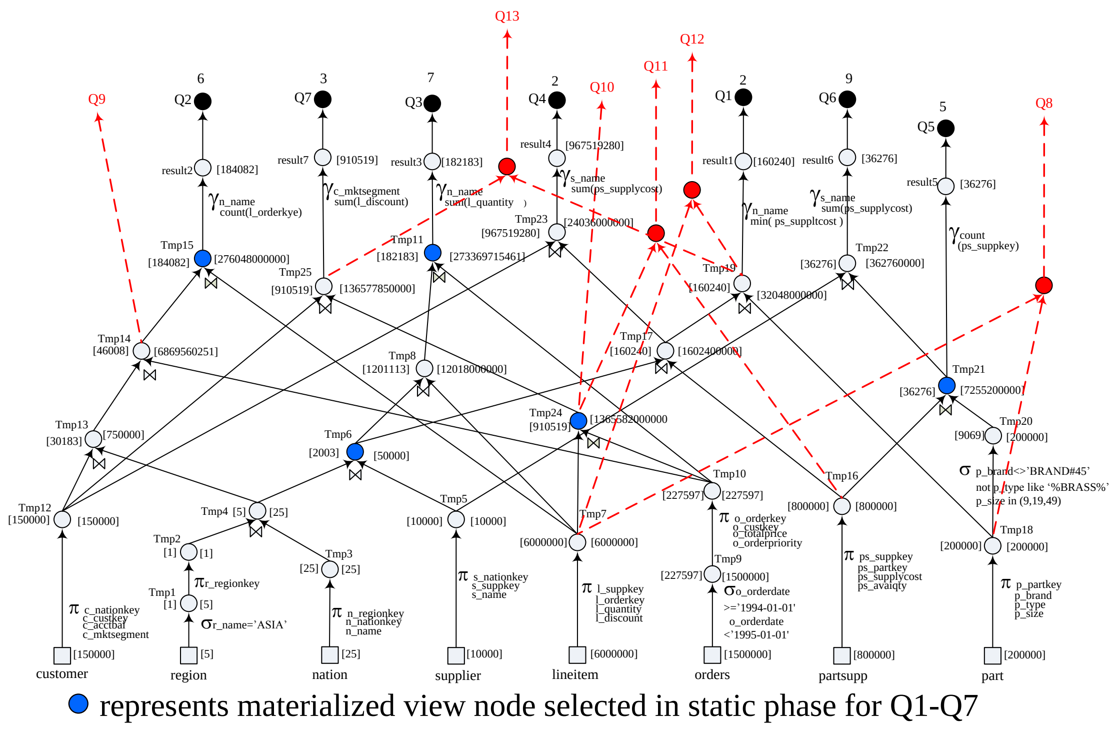


Figure C.9 (f) The Fifth Dynamic MVPP after All New Queries are Merged

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.

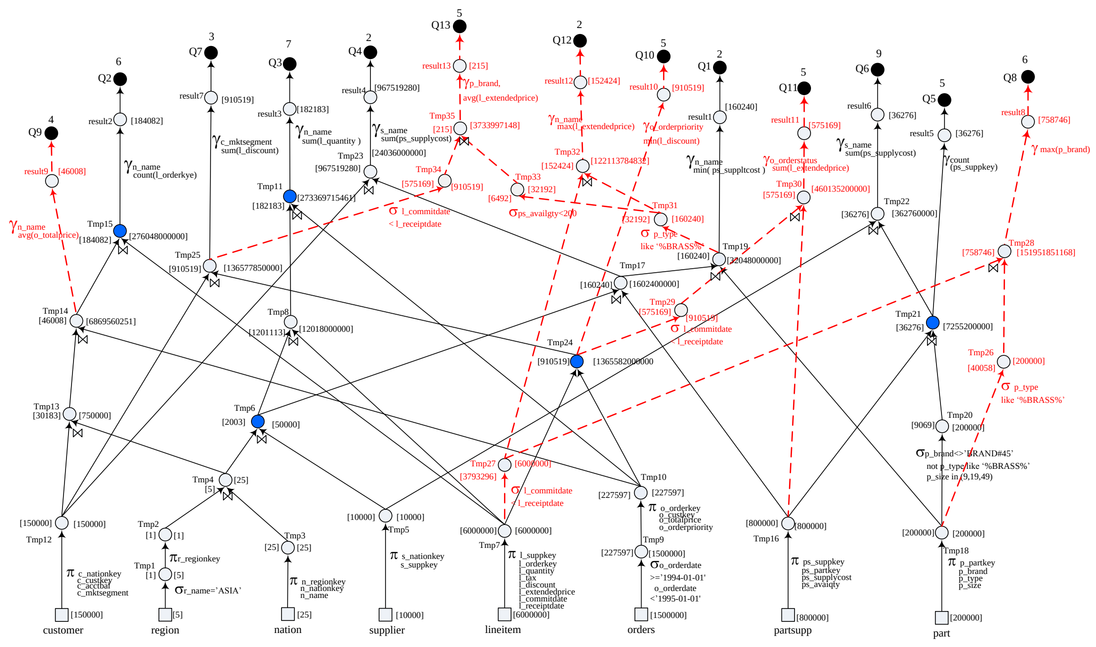

● represents materialized view node selected in static phase for Q1-Q7

Figure C.10 The Fifth Dynamic MVPP after Optimized

Table C.2 The Query Processing Cost of the Fifth and the Sixth Dynamic MVPP

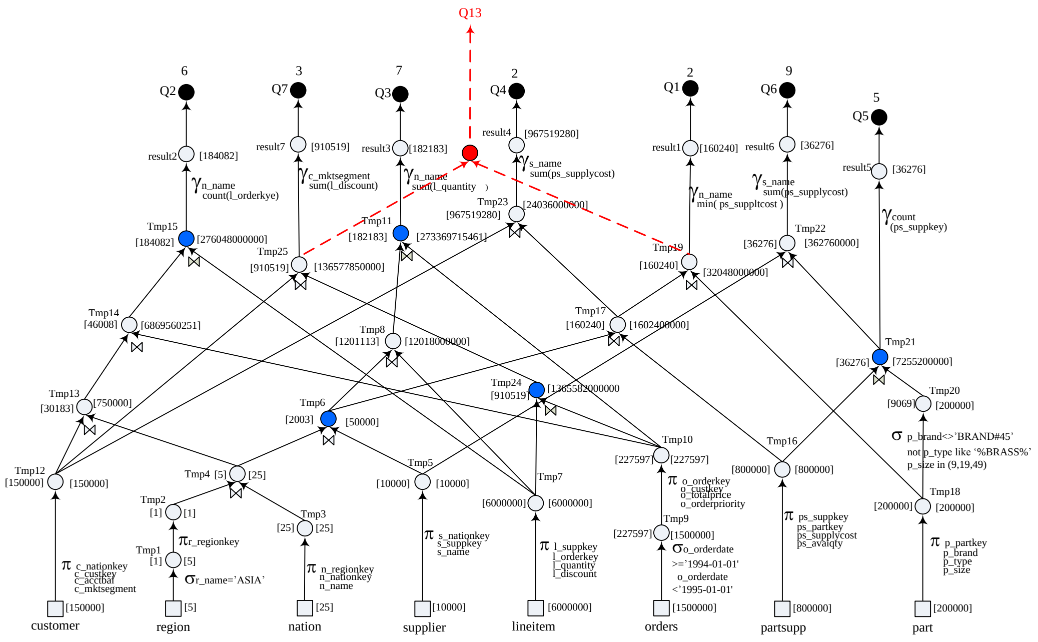
Query	**f_q**	**List of Accessed Nodes**	**Cost of Nodes**	**Query Processing Cost**
Q1	2	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19 and result1	2003, 800000, 1602400000, 200000, 32048000000 and 160240	67,303,124,486
Q2	6	Tmp15(materialized view) and result2	184082 and 184082	2,208,984
Q3	7	Tmp11(materialized view) and result3	182183 and 182183	2,550,562
Q4	2	Tmp6(materialized view), Tmp16, Tmp17,Tmp12, Tmp23 and result4	2003, 800000, 1602400000, 150000, 24036000000, and 967519280	53,213,742,566
Q5	5	Tmp21(materialized view) and result5	36276 and 36276	362,760
Q6	9	Tmp21(materialized view), Tmp5, Tmp22, and result6	36276, 10000, 362760000 and 36276	3,265,582,968
Q7	3	Tmp24(materialized view), Tmp12, Tmp25 and result7	910519, 150000, 136577850000 and 910519	409,739,463,114
Q8	6	Tmp7, Tmp27, Tmp18, Tmp26, Tmp28 and result8	6000000, 6000000, 200000, 200000, 151951851168 and 758746	911,790,059,484
Q9	4	Tmp1, Tmp2, Tmp3, Tmp4, Tmp12, Tmp13, Tmp9, Tmp10, Tmp14 and result 9	5, 1, 25, 25, 150000, 750000, 1500000, 227597, 6869560251 and 46008	27,488,935,648
Q10	5	Tmp24(materialized view) and result10	910519, 910519	9,105,190
Q11	5	Tmp24(materialized view), Tmp29, Tmp16, Tmp30 and result11	910519, 575169, 800000, 460135200000 and 575169	2,300,691,981,035

Table C.2 (Continued)

Query	f_q	List of Accessed Nodes	Cost of Nodes	Query Processing Cost
Q12	2	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp7, Tmp27, Tmp32 and result12	2003, 800000, 1602400000, 200000, 32048000000, 160240, 6000000, 6000000, 122113784832 and 152424	311,554,998,998
Q13	5	Tmp6(materialized view), Tmp16, Tmp17, Tmp18, Tmp19, Tmp31, Tmp33, Tmp24(materialized view), Tmp12, Tmp25, Tmp34, Tmp35 and result13	2003, 800000, 1602400000, 200000, 32048000000, 160240, 32192, 910519, 150000, 136577850000, 910519, 3733997148 and 215	869,827,064,180
		The total query processing cost of the first MVPP		**4,954,889,179,975**

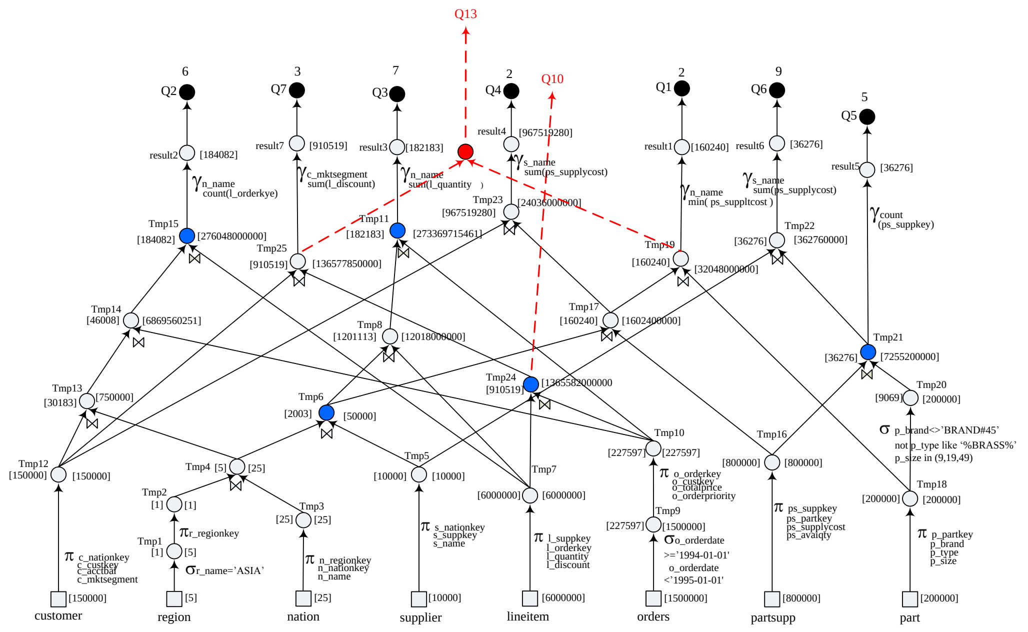
C.6 The Sixth Dynamic MVPP

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



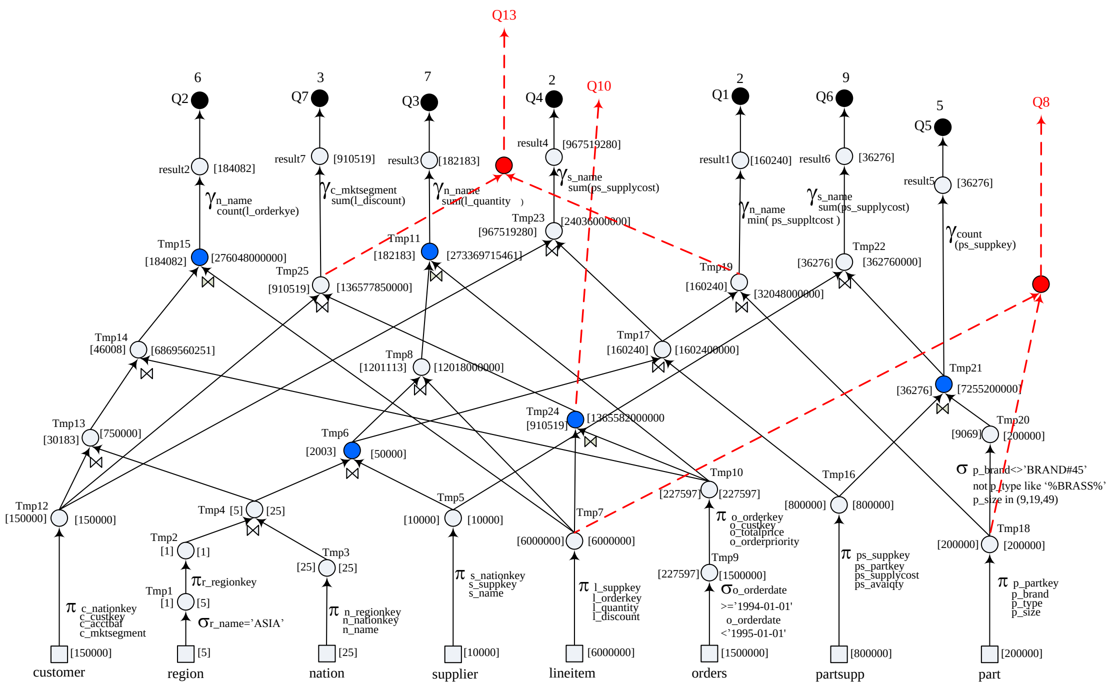
● represents materialized view node selected in static phase for Q1-Q7

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



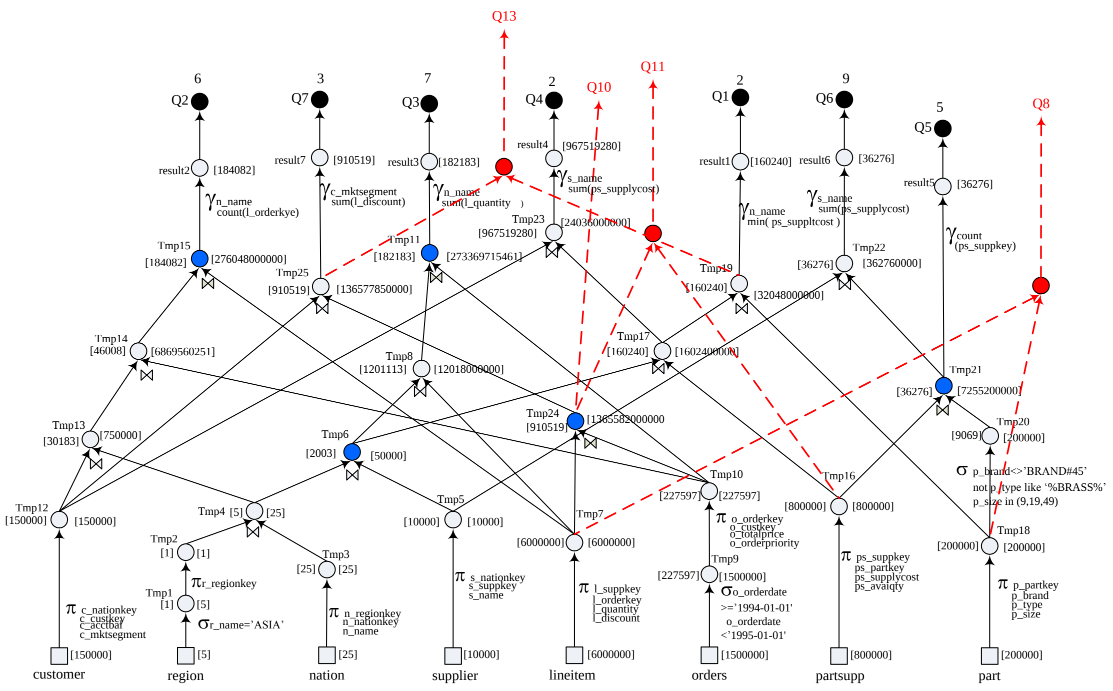
● represents materialized view node selected in static phase for Q1-Q7

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



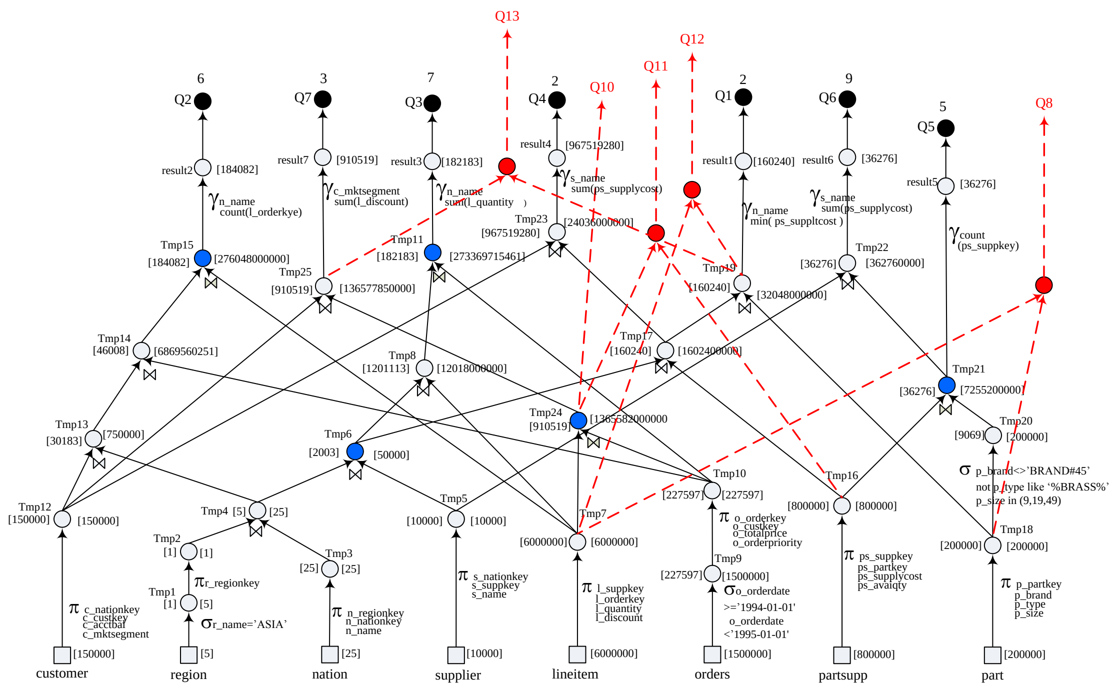
● represents materialized view node selected in static phase for Q1-Q7

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



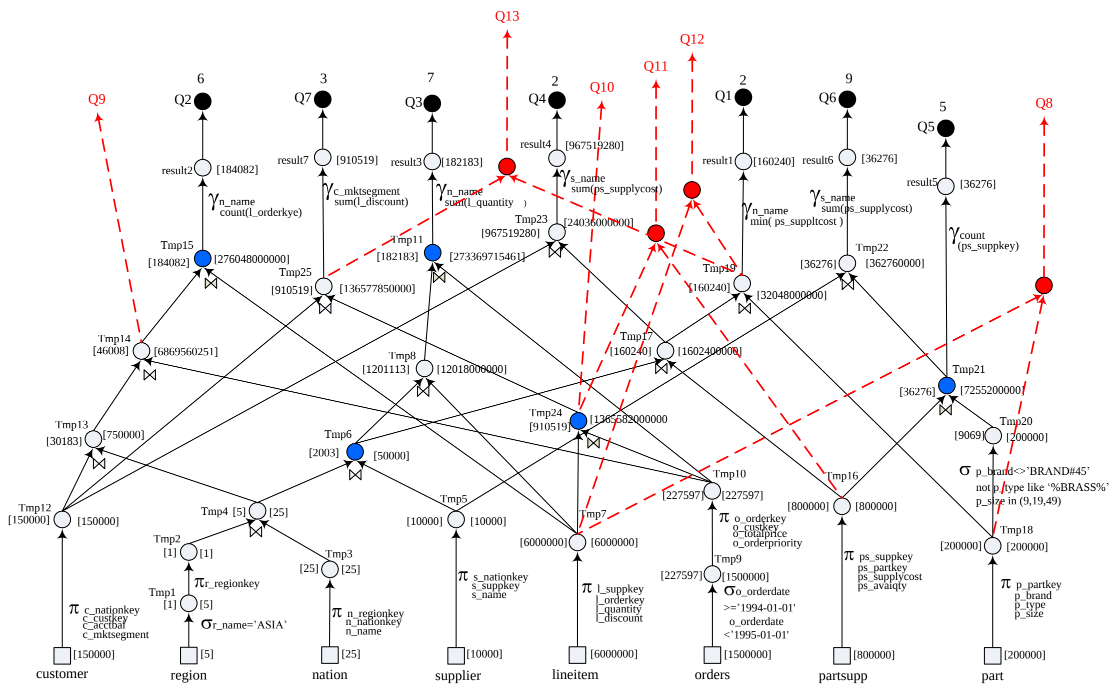
● represents materialized view node selected in static phase for Q1-Q7

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


Q13
Q12
Q11
Q10
Q8
Q2
Q7
Q3
Q4
Q1
Q6
Q5
6
3
7
2
2
9
5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp11 [182183] [273369715461]
Tmp25 [910519] [136577850000]
Tmp23 [967519280] [240360000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [160240000]
Tmp21 [36276] [7255200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp24 [910519] [136582000000]
Tmp20 [9069] [200000]
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp7 [6000000] [6000000]
Tmp10 [227597] [227597]
Tmp16 [800000] [800000]
Tmp18 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp9 [227597] [1500000]
Tmp1 [1] [5]
π r_regionkey
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
π o_orderkey o_custkey o_totalprice o_orderpriority
σ o_orderdate >='1994-01-01' o_orderdate <'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
π p_partkey p_brand p_type p_size
[150000]
[5]
[25]
[10000]
[6000000]
[1500000]
[800000]
[200000]
customer
region
nation
supplier
lineitem
orders
partsupp
part

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

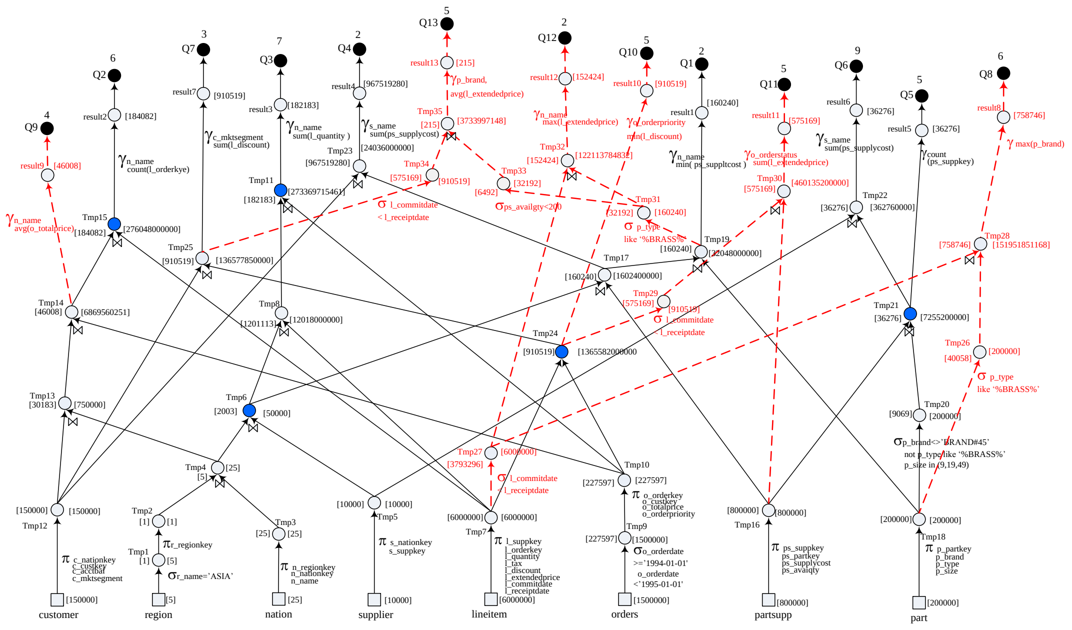
Figure C.11 (e) The Sixth Dynamic MVPP after Merging Q12 into the Existing Re-Optimized MVPP


Q9
Q13
Q12
Q11
Q10
Q8
Q2
Q7
Q3
Q4
Q1
Q6
Q5
6
3
7
2
2
9
5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γ n_name count(l_orderkye)
γ c_mktsegment sum(l_discount)
γ n_name sum(l_quantity)
γ s_name sum(ps_supplycost)
γ n_name min(ps_suppltcost)
γ s_name sum(ps_supplycost)
γ count (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp11 [182183] [273369715461]
Tmp25 [910519] [136577850000]
Tmp23 [967519280] [240360000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp17 [160240] [160240000]
Tmp21 [36276] [7255200000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp24 [910519] [136582000000]
Tmp20 [9069] [200000]
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp7 [6000000] [6000000]
Tmp10 [227597] [227597]
Tmp16 [800000] [800000]
Tmp18 [200000] [200000]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp9 [227597] [1500000]
Tmp1 [1] [5]
π r_regionkey
σ r_name='ASIA'
π c_nationkey c_custkey c_acctbal c_mktsegment
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_discount
π o_orderkey o_custkey o_totalprice o_orderpriority
σ o_orderdate >='1994-01-01' o_orderdate <'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
σ p_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
π p_partkey p_brand p_type p_size
[150000]
[5]
[25]
[10000]
[6000000]
[1500000]
[800000]
[200000]
customer
region
nation
supplier
lineitem
orders
partsupp
part

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

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.



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

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

$$w(v) = \sum_{q \in O_v} \left\{ f_q(q) * \left(C_a^q(v) \right) \right\} - \sum_{r \in I_v} \left\{ f_u(r) * \left(C_m^r(v) \right) \right\}$$

w(v) denotes weight of node

O_v denotes the queries which use view *v*.

C_a^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*.

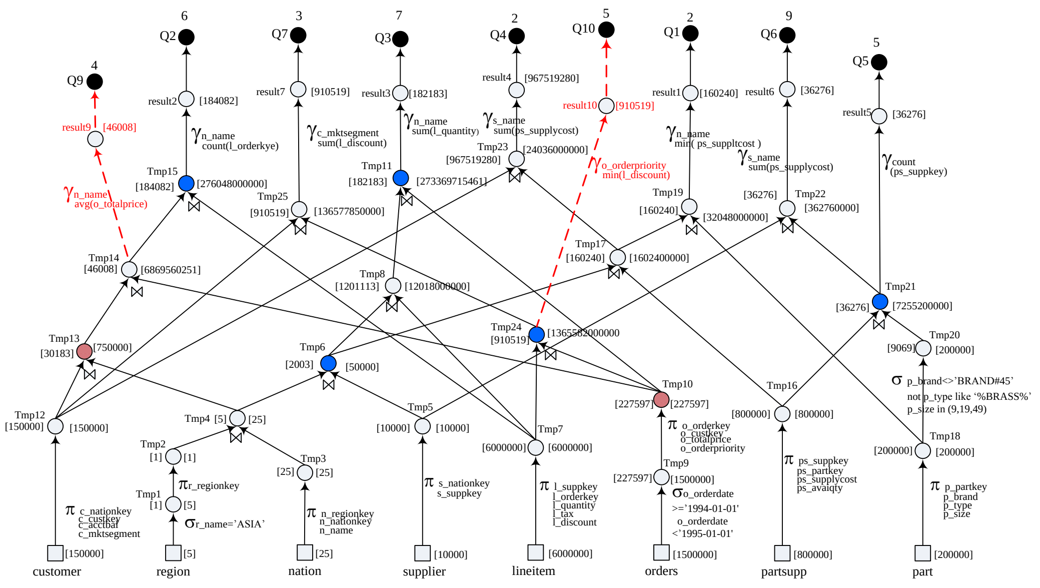
f_q denotes the frequency of executing a query.

I_v denotes the base relations which are used to produce view *v*.

C_m^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

D.1 The Affected Node of The Subsumption Data Set



● 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

Existing Node	Query	Derived Node	Number of Base Relations	Weight (*w(v)*)	
Tmp1	Q1,Q2,Q3,Q4,Q9	Tmp1	1	(2+6+7+2+4)(5) – (1)(5)	100
Tmp2	Q1,Q2,Q3,Q4,Q9	Tmp1, 2	1	(2+6+7+2+4)(1) – (1)(5+1)	15
Tmp3	Q1,Q2,Q3,Q4,Q9	Tmp1,2,3	1	(2+6+7+2+4)(25) – (1)(25)	500
Tmp4	Q1,Q2,Q3,Q4,Q9	Tmp1,2, 3, 4	2	(2+6+7+2+4)(25) – (2)(5+1+25+25)	413
Tmp7	Q2,Q3,Q7,Q10	Tmp7	1	(6+7+3+5) (6000000) – (1)(6000000)	120,000,000
Tmp9	Q2,Q3,Q7,Q9,Q10	Tmp9	1	(6+7+3+5+4) (1500000) – (1) (1500000)	36,000,000
Tmp10	Q2,Q3,Q7,Q9,Q10	Tmp9, 10	1	(6+7+3+5+4)(227597) – (1)(1500000+227597)	3,962,328
Tmp12	Q4, Q7, Q2, Q9	Tmp12	1	(2+3+6+4)(150000) – (1)(150000)	2,100,000
Tmp13	Q2,Q9	Tmp1, 2, 3, 4, 12, 13	3	(6+4)(750000) – (3)(5+1+25+25+150000+750000)	4,799,832
Tmp14	Q2,Q9	Tmp1, 2, 3, 4, 12, 13, 9, 10, 14	4	(6+4)(6869560251) – (4)(5+1+25+25+150000+750000+ +1500000+227597+6869560251)	41,206,850,894
Tmp24	Q7,Q10	Tmp7, 9, 10, 24	2	(3+5)(1365582000000) – (2)(6000000+1500000+227597+1365582000000)	8,193,476,544,806

Table D.2 The Weight of Ancestor Node of Directly Affected Node of Q9, Q10

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp4	Tmp6	Q1,Q3,Q4	Tmp1, 2, 3, 4, 5, 6	3	(2+7+2) (50000) – (3)*(5+1+25+25+10000 +50000)	369,832
	Tmp8	Q7	Tmp1, 2, 3, 4, 5, 6, 7, 8	4	(7)(12018000000) – (4)(5+1+25+25 +10000 +50000 +6000000+12018000000)	36,029,759,776
	Tmp11	Q7	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10	5	(7)(273369715461) – (5) (5+1+25 +25 + 10000+ 50000 +6000000 +12018000000+ 1500000 +227597+273369715461)	486,610,492,657
	Tmp17	Q1,Q4	Tmp1, 2, 3, 4, 5, 6, 16, 17	4	(2+2)(1602400000) – (4)(5+1+25+25 +10000+50000+800000+1602400000)	-3,440,224
	Tmp19	Q1	Tmp1, 2, 3, 4, 5, 6, 16, 17, 19	5	(2)(32048000000) – (5)(5+1+25+25 +10000+50000+800000+1602400000 +200000+32048000000)	-104,160,300,280
	Tmp23	Q4	Tmp1, 2, 3, 4, 5, 6, 16, 17, 12, 23	5	(2)(24036000000) – (5)(5+1+25+25 +10000 +50000+800000+1602400000 +150000+24036000000)	-80,125,050,280

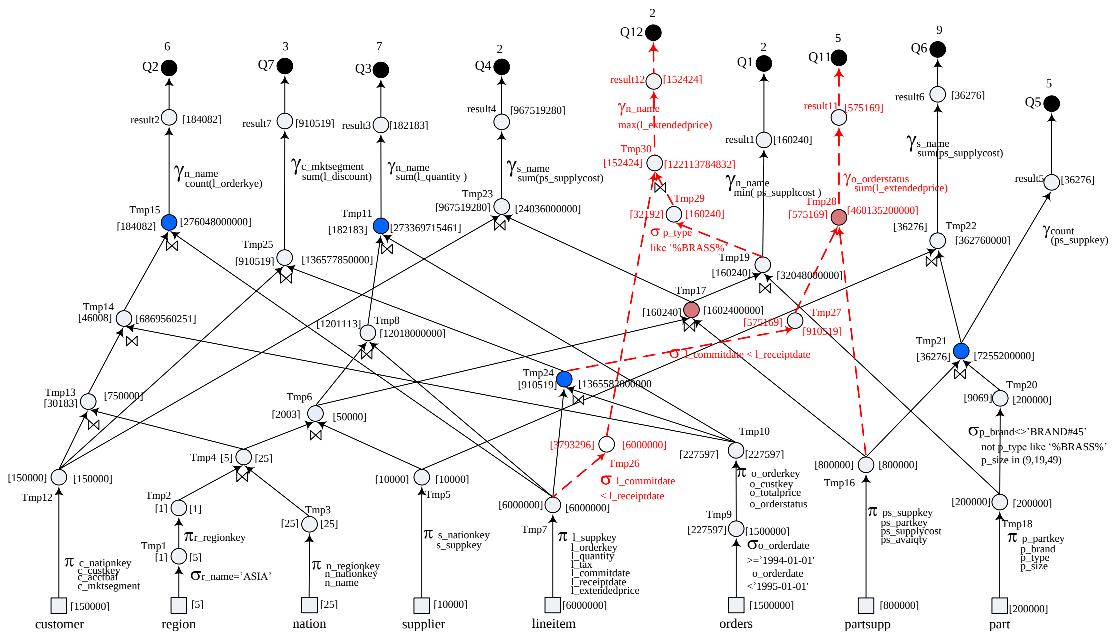
Table D.2 (Continued)

Directly Affected Node	**Ancestor Node**	**Query**	**Derived Node**	**Number of Base Relations**	**Weight of Ancestor Node**	
Tmp14	Tmp15	Q2	Tmp1, 2, 3, 4, 12, 13, 9, 10, 14, 7, 15	5	(6)(276048000000) – (5)(5+1+25+25+ 150000+750000+ 1500000 +227597 + 6869560251+6000000 +276048000000)	241,657,060,480
Tmp24	Tmp25	Q7	Tmp7, 9, 10, 24	3	(3)(136577850000) – (3)(6000000+1500000+227597 +1365582000000+150000+136577850000	-4,096,769,632,791

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

Existing Node	Query	Derived Node	Number of Base Relations	Weight (*w(v)*)	
Tmp1	Q1,Q2,Q3,Q4,Q12	Tmp1	1	(2+6+7+2+2)(5) – (1)(5)	90
Tmp2	Q1,Q2,Q3,Q4,Q12	Tmp1,2	1	(2+6+7+2+2)(1) – (1)(5+1)	13
Tmp3	Q1,Q2,Q3,Q4,Q12	Tmp3	1	(2+6+7+2+2)(25) – (1)(25)	450
Tmp4	Q1,Q2,Q3,Q4,Q12	Tmp1, 2, 3, 4	2	(2+6+7+2+2)(25) – (2)(5+1+25+25)	363
Tmp5	Q1,Q3,Q4,Q6,Q12	Tmp5	1	(2+7+2+9+2)(10000) – (1)(10000)	210,000
Tmp6	Q1,Q3,Q4,Q12	Tmp1, 2, 3, 4, 5, 6	3	(2+7+2+2)(50000) – (3) (5+1+25+25 +10000 + 50000)	469,832
Tmp7	Q2,Q3,Q7,Q11	Tmp7	1	(6+7+3+5)(6000000) – (1)(6000000)	120,000,000
Tmp9	Q2,Q3,Q7,Q11	Tmp9	1	(6+7+3+5)(1500000) – (1) (1500000)	30,000,000
Tmp10	Q2,Q3,Q7,Q11	Tmp9, 10	1	(6+7+3+5)(227597) – (1) (1500000+227597)	3,051,940
Tmp16	Q1,Q4,Q5,Q6, Q12	Tmp16	1	(2+2+5+9+2)(800000) – (1)(800000)	15,200,000
Tmp17	Q1,Q4,Q12	Tmp1,2,3, 4, 5, 6, 16, 17	4	(2+2+2)(1602400000) – (4) (5+1+25+25+ +10000+50000+800000+1602400000)	3,201,359,776
Tmp18	Q1,Q5,Q6,Q12	Tmp18	1	(2+5+9+2)(200000) – (1) (200000)	3,400,000
Tmp19	Q1,Q12	Tmp1,2,3,4,5,6,16,17,18, 19	5	(2+2)(32048000000) – (5) (5+1+25+25+10000 +50000 +800000+1602400000 +200000 +32048000000)	-40,065,300,280
Tmp24	Q7,Q11	Tmp7, 9, 10, 24	2	(3+5)(1365582000000) – (2)(6000000 +1500000+227597+1365582000000)	8,193,476,544,806

Table D.4 The Weight of Ancestor Node of Directly Affected Node of Q11, Q12

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp4	Tmp13	Q2	Tmp1, 2, 3, 4, 12, 13	3	(6)(750000) – (3)(5+1+25+25+150000+ 750000)	1,799,832
	Tmp14	Q2	Tmp1,2,3, 4, 12, 9, 10, 13, 14	4	(6)(6869560251) – (4)(5+1+25+25+150000 +750000 + 1500000 +227597+6869560251)	13,728,609,890
	Tmp15	Q2	Tmp1,2,3,4, 7,12,9,10,13,14, 15	5	(6)(276048000000) – (5)(5+1+25+25+ 150000 + 750000 + 1500000 +227597 +6869560251 +6000000 +276048000000)	241,657,060,480
Tmp6	Tmp8	Q3	Tmp1,2,3,4,5,6,7,8	4	(7)(12018000000) – (4) (5+1+25+25+ 10000 +50000+6000000+12018000000)	36,029,759,776
	Tmp11	Q3	Tmp1,2,3,4,5,6,7,8,9,10,11	5	(7)(273369715461) – (5) (5+1+25+25 +10000 +50000+6000000+12018000000+ 1500000 +227597+273369715461)	486,610,492,657
Tmp17	Tmp19	Q1,Q12	Tmp1,2,3,4,5,6,16,17,18,19	5	(2+2)(32048000000) – (5) (5+1+25+25+10000 +50000+800000+1602400000 +200000 +32048000000)	-40,065,300,280
	Tmp23	Q4	Tmp1,2,3,4,5,6, 12,16,17, 23	5	(2)(24036000000) – (5) (5+1+25+25 +10000 +50000 +800000+1602400000 +150000 + 24036000000)	-80,125,050,280

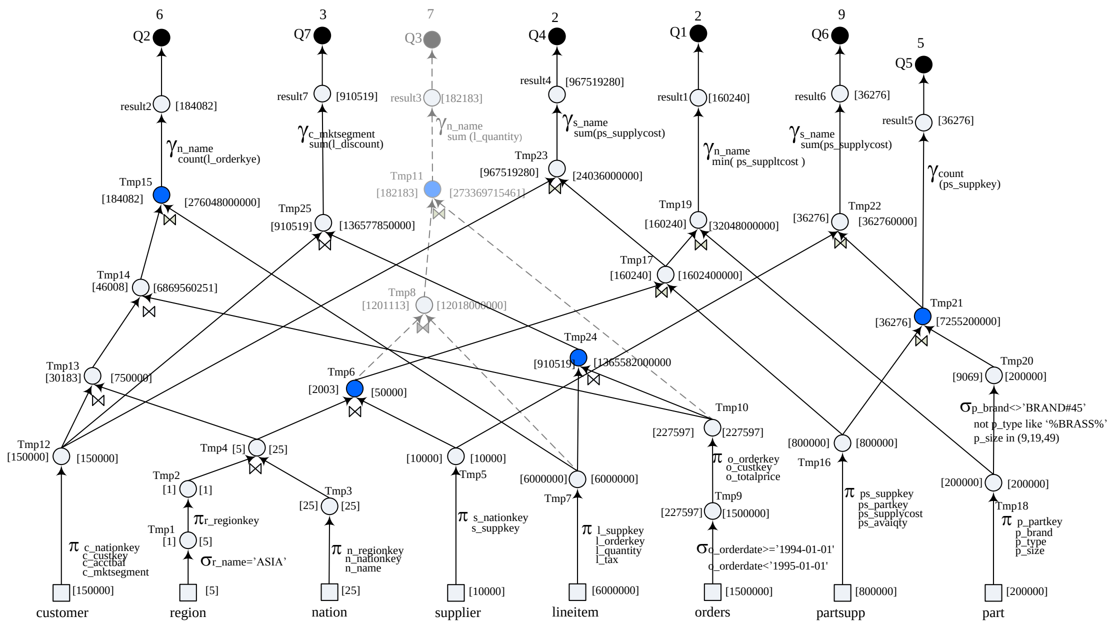
Table D.4 (Continued)

Directly Affected Node	**Ancestor Node**	**Query**	**Derived Node**	**Number of Base Relations**	**Weight of Ancestor Node**	
Tmp24	Tmp25	Q7	Tmp7, 9,10, 12,24,25,	3	(3)(136577850000) – (3)(6000000+1500000 +227597+150000+ 1365582000000+ 136577850000)	-4,096,769,632,791

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

Existing Node	**Query**	**Derived Node**	**Number of Base Relations**	**Weight (*w(v)*)**	
Tmp1	Q1,Q2,Q4	Tmp1	1	(2+6+7+2)(5) – (1)(5)	45
Tmp2	Q1,Q2,Q4	Tmp1,2	1	(2+6+7+2)(1) – (1)(5+1)	4
Tmp3	Q1,Q2,Q4	Tmp3	1	(2+6+7+2)(25) – (1)(25)	225
Tmp4	Q1,Q2,Q4	Tmp1,2,3,4	2	(2+6+7+2)(25) – (2)(5+1+25+25)	138
Tmp5	Q1,Q4,Q6	Tmp5	1	(2+2+9)(10000) – (1)(10000)	120,000
Tmp6	Q1,Q4	Tmp1,2,3,4,5,6	3	(2+2)(50000) – (3)(5+1+25+25+10000+50000)	19,832
Tmp7	Q2,Q7	Tmp7	1	(6+3)(6000000) – (1)(6000000)	48,000,000
Tmp9	Q2,Q7	Tmp9	1	(6+3)(1500000) – (1)(1500000)	12,000,000
Tmp10	Q2,Q7	Tmp9,10	1	(6+3)(227597) – (1)(1500000+227597)	320,776

Table D.6 The Weight of Ancestor Node of Directly Affected Node of Q3

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp4	Tmp13	Q2	Tmp1, 2, 3, 4, 12, 13	3	(6)(750000) – (3)(5+1+25+25+150000+ 750000)	1,799,832
	Tmp14	Q2	Tmp1,2,3, 4, 12, 9, 10, 13, 14	4	(6)(6869560251) – (4)(5+1+25+25+150000 +750000 + 1500000 +227597+6869560251)	13,728,609,890
	Tmp15	Q2	Tmp1,2,3,4, 7,12,9,10,13,14, 15	5	(6)(276048000000) – (5)(5+1+25+25+ 150000 + 750000 + 1500000 +227597 +6869560251+6000000 +276048000000)	241,657,060,480
Tmp6	Tmp17	Q1,Q4	Tmp1,2,3,4,6,16,17	4	(2+2)(1602400000) – (4) (5+1+25+25+ +10000+50000+800000+1602400000)	-3,440,224
	Tmp19	Q4	Tmp1,2,3,4,6,16,17,18,19	5	(2)(32048000000) – (5) (5+1+25+25+ +10000+50000+800000+1602400000 +200000+32048000000)	-104,160,300,280
	Tmp23	Q4	Tmp1,2,3,4,6,12,16,17,23	5	(2)(24036000000) – (5) (5+1+25+25+ +10000+50000+800000+1602400000 +150000 + 24036000000)	-80,125,050,280
Tmp10	Tmp24	Q7	Tmp7, 9,10, 24	2	(3)(1365582000000) – (2)(6000000 +1500000+227597+1365582000000)	1,365,566,544,806

Table D.6 (Continued)

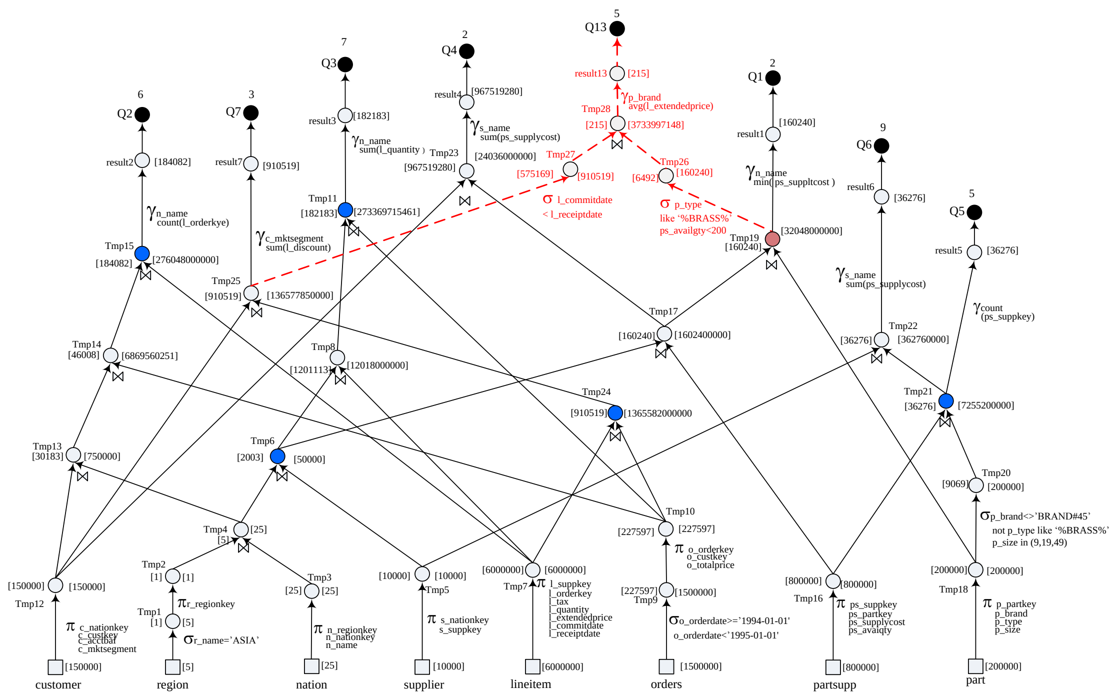
Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp10	Tmp25	Q7	Tmp7, 9,10, 12,24,25,	3	(3)(136577850000) – (3)(6000000+1500000 +227597+150000+ 1365582000000+ 136577850000)	-4,096,769,632,791

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



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

● represents materialized view node selected in dynamic phase

Figure D.5 The Existing Re-Optimized MVPP with Q13 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

Existing Node	**Query**	**Derived Node**	**Number of Base Relations**	**Weight (*w(v)*)**	
Tmp1	Q1,Q2,Q3,Q4,Q13	Tmp1	1	(2+6+7+2+5)(5) – (1)(5)	105
Tmp2	Q1,Q2,Q3,Q4,Q13	Tmp1,2	1	(2+6+7+2+5)(1) – (1)(5+1)	16
Tmp3	Q1,Q2,Q3,Q4,Q13	Tmp3	1	(2+6+7+2+5)(25) – (1)(25)	525
Tmp4	Q1,Q2,Q3,Q4,Q13	Tmp1,2,3,4	2	(2+6+7+2+5)(25) – (2)(5+1+25+25)	438
Tmp5	Q1,Q3,Q4,Q6,Q13	Tmp5	1	(2+7+2+9+5)(10000) – (1)(10000)	240,000
Tmp6	Q1,Q3,Q4,Q13	Tmp1,2,3,4,5,6	3	(2+7+2+5)(50000) – (3)(5+1+25+25+10000+50000)	619,832
Tmp7	Q2,Q3,Q7,Q13	Tmp7	1	(6+7+3+5)(6000000) – (1)(6000000)	120,000,000
Tmp9	Q2,Q3,Q7,Q13	Tmp9	1	(6+7+3+5)(1500000) – (1)(1500000)	30,000,000
Tmp10	Q2,Q3,Q7,Q13	Tmp9,10	1	(6+7+3+5)(227597) – (1)(1500000+227597)	3,051,940
Tmp12	Q2, Q4,Q7,Q13	Tmp12	1	(6+2+3+5)(150000) – (1)(150000)	2,250,000
Tnp16	Q1,Q4,Q5,Q 6, Q13	Tmp16	1	(2+2+5+9+5)(800000) – (1)(800000)	17,600,000
Tmp17	Q1,Q4,Q13	Tmp1,2,3,4,5,6,16,17	4	(2+2+5)(1602400000) – (4)(5+1+25+25+10000 +50000+800000+1602400000)	8,008,559,776
Tmp18	Q1,Q5,Q6,Q13	Tmp18	1	(2+5+9+5)(200000) – (1)(200000)	4,000,000
Tmp19	Q1,Q13	Tmp1,2,3,4,5,6,16,17,18,19	5	(2+5)(32048000000) – (5)(5+1+25+25+10000 +50000+800000+1602400000+200000+32048000000)	56,078,699,720

Table D.7 (Continued)

Existing Node	Query	Derived Node	Number of Base Relations	Weight (*w(v)*)	
Tmp24	Q7,Q13	Tmp7,9,10,24	2	(3+5)(1365582000000) –(2)(6000000+ 1500000+ 227597+ 1365582000000)	8,193,476,544,806
Tmp25	Q7,Q13	Tmp7,9,10,12,24,25	3	(3+5)(136577850000) –(3)(6000000+ 1500000+ 227597+ 150000+1365582000000+136577850000)	-3,413,880,382,791

Table D.8 The Weight of Ancestor Node of Directly Affected Node of Q13

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp4	Tmp13	Q2	Tmp1, 2, 3, 4, 12, 13	3	(6)(750000) – (3)(5+1+25+25+150000+ 750000)	1,799,832
	Tmp14	Q2	Tmp1,2,3, 4, 12, 9, 10, 13, 14	4	(6)(6869560251) – (4)(5+1+25+25+150000 +750000 + 1500000 +227597+6869560251)	13,728,609,890
	Tmp15	Q2	Tmp1,2,3,4, 7,12,9,10,13,14, 15	5	(6)(276048000000) – (5)(5+1+25+25+ 150000 + 750000 + 1500000 +227597 +6869560251+6000000 +276048000000)	241,657,060,480

Table D.8 (Continued)

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp6	Tmp8	Q3	Tmp1,2,3,4,5,6,7,8	4	(7)(12018000000) – (4) (5+1+25+25+ +10000+50000+6000000+12018000000)	36,029,759,776
	Tmp11	Q3	Tmp1,2,3,4,5,6,7,8,9,10,11	5	(7)(273369715461) – (5) (5+1+25+25+ +10000+50000+6000000+12018000000+ 1500000 +227597+273369715461)	486,610,492,657
Tmp24	Tmp25	Q7,Q13	Tmp7,9,10,12,24,25	3	(3+5)(136577850000) –(3)(6000000+ 1500000+ 227597+ 150000+1365582000000+136577850000)	-3,413,880,382,791

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

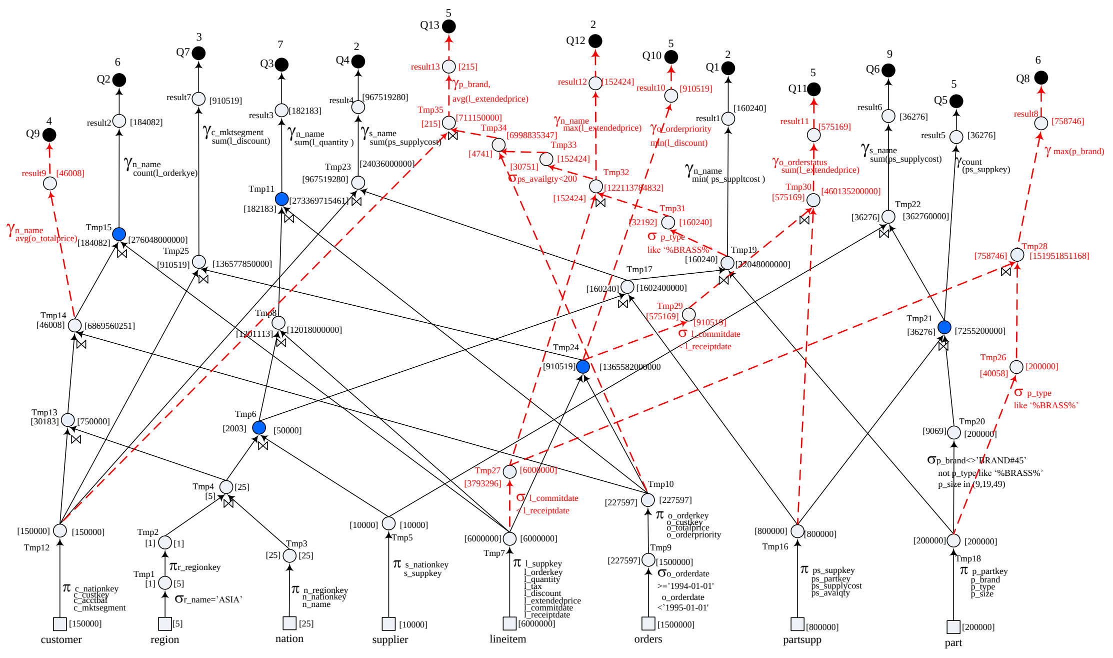

● represents materialized view node selected in static phase for Q1-Q7

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 D.9 The Weight of the Existing Node of All Queries of the First Experiment

Existing Node	**Query**	**Derived Node**	**Number of Base Relations**	**Weight (*w(v)*)**	
Tmp1	Q1,Q2,Q3,Q4,Q9,Q12,Q13	Tmp1	1	(2+6+7+2+4+2+5)(5) – (1)(5)	135
Tmp2	Q1,Q2,Q3,Q4,Q9,Q12,Q13	Tmp1,2	1	(2+6+7+2+4+2+5)(1) – (1)(5+1)	22
Tmp3	Q1,Q2,Q3,Q4,Q9,Q12,Q13	Tmp3	1	(2+6+7+2+4+2+5)(25) – (1)(25)	675
Tmp4	Q1,Q2,Q3,Q4,Q9,Q12,Q13	Tmp1,2,3,4	2	(2+6+7+2+4+2+5)(25) – (2)(5+1+25+25)	588
Tmp5	Q1,Q3, Q4,Q12,Q13	Tmp5	1	(2+7+2+9+2+5)(10000) – (1)(10000)	260,000
Tmp6	Q1,Q3,Q4,Q12,Q13	Tmp1,2,3,4,5,6	3	(2+7+2+2+5)(50000) – (3)(5+1+25+25+10000+50000)	719,832
Tmp7	Q2,Q3,Q7,Q8,Q10,Q11,Q12,Q13	Tmp7	1	(6+7+3+6+5+5+2+5)(6000000) – (1)(6000000)	228,000,000
Tmp9	Q2,Q3,Q7,Q9,Q10,Q11,Q13	Tmp9	1	(6+7+3+4+5+5+5)(1500000) – (1)(1500000)	51,000,000
Tmp10	Q2,Q3,Q7,Q9,Q10,Q11,Q13	Tmp9,10	1	(6+7+3+4+5+5+5)(227597) – (1)(1500000+227597)	6,238,298
Tmp12	Q2, Q4,Q7,Q9,Q13	Tmp12	1	(6+2+3+4+5)(150000) – (1)(150000)	2,850,000
Tmp13	Q2,Q9	Tmp1, 2, 3, 4, 12, 13	3	(6+4)(750000) – (3) (5 + 1 + 25 + 25 + 150000 + 750000)	4,799,832

Table D.9 (Continued)

Existing Node	**Query**	**Derived Node**	**Number of Base Relations**	**Weight (*w(v)*)**	
Tmp14	Q2,Q9	Tmp1, 2, 3, 4, 12, 13, 9, 10, 14	4	(6+4)(6869560251) – (4)(5+1+25+25+150000+750000+ +1500000+227597+6869560251)	41,206,850,894
Tmp16	Q1,Q4,Q5,Q 6,Q11,Q12,Q13	Tmp16	1	(2+2+5+9+5+2+5)(800000) – (1)(800000)	23,200,000
Tmp17	Q1,Q4,Q12,Q13	Tmp1,2,3,4,5,6,16,17	4	(2+2+2+5)(1602400000) – (4) (5+1+25+25+ +10000+50000+800000+1602400000)	11,213,359,776
Tmp18	Q1,Q5,Q6,Q8,Q12,Q13	Tmp18	1	(2+5+9+6+2+5)(200000) – (1) (200000)	5,600,000
Tmp19	Q1,Q12,Q13	Tmp1,2,3,4,5,6,16,17,18,19	5	(2+2+5)(32048000000) – (5)(5+1+25+25 +10000+50000+800000+1602400000 +200000+32048000000)	120,174,699,720
Tmp24	Q7,Q10,Q11	Tmp7,9,10,24	2	(3+5+5)(1365582000000) – (2)(6000000+ 1500000+ 227597+ 1365582000000)	15,021,386,544,806

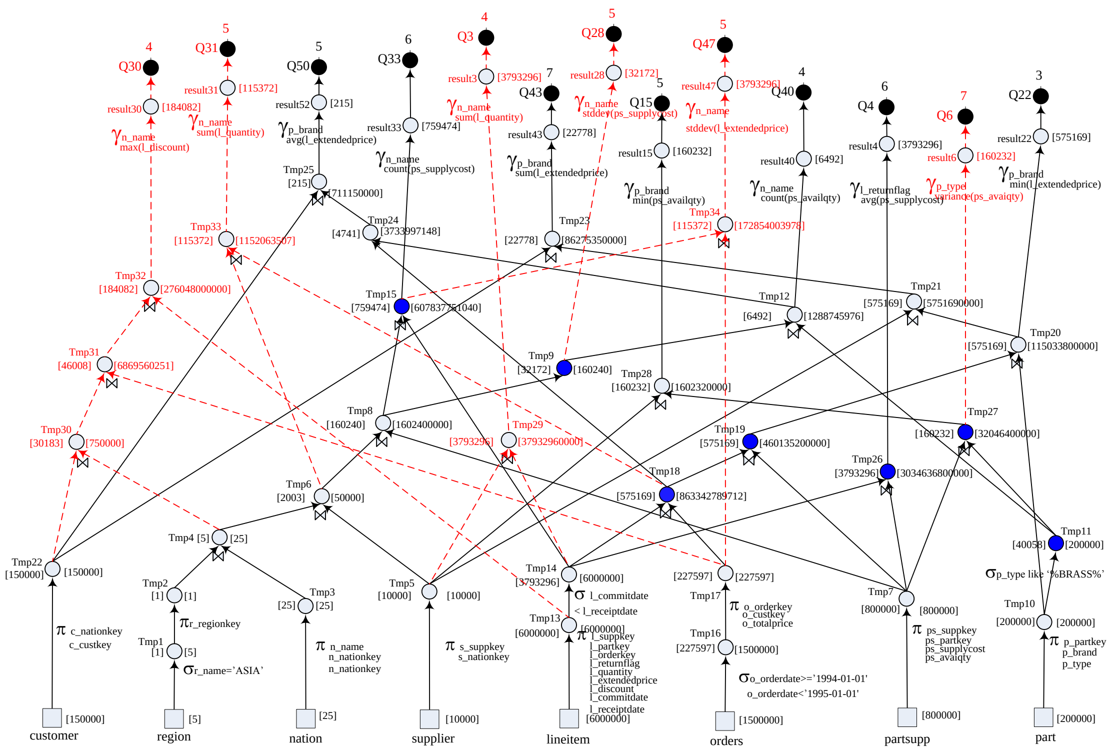
Table D.10 The Weight of Ancestor Node of Directly Affected Node of All Queries of the First Experiment

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp6	Tmp8	Q3	Tmp1,2,3,4,5,6,7,8	4	(7)(12018000000) – (4) (5+1+25+25+ +10000+50000+6000000+12018000000)	36,029,759,776
	Tmp11	Q3	Tmp1,2,3,4,5,6,7,8,9,10,11	5	(7)(273369715461) – (5) (5+1+25+25+ +10000+50000+6000000+12018000000+ 1500000 +227597+273369715461)	486,610,492,657
Tmp14	Tmp15	Q2	Tmp1,2,3,4, 7,12,9,10,13,14, 15	5	(6)(276048000000) – (5)(5+1+25+25+ 150000 + 750000 + 1500000 +227597 +6869560251+6000000 +276048000000)	241,657,060,480
Tmp17	Tmp23	Q4	Tmp1, 2, 3, 4, 5, 6, 16, 17, 12, 23	5	(2)(24036000000) – (5)(5+1+25+25+10000 +50000+800000+1602400000 +150000 +24036000000)	- 80,125,050,280
Tmp24	Tmp25	Q7	Tmp7, 9, 10, 24	3	(3)(136577850000) – (3)(6000000+1500000 +227597 +1365582000000 +150000 +136577850000	-4,096,769,632,791

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 D.11 The Weight of the Existing Node of the Second Experiment of the Second Experiment

Existing Node	Query	Derived Node	Number of Base Relations	Weight (*w(v)*)	
Tmp1	Q33,Q40,Q50,Q28,Q30,Q31,Q47	Tmp1	1	(6+4+5+5+4+5+5)(5) – (1)(5)	165
Tmp2	Q33,Q40,Q50,Q28,Q30,Q31,Q47	Tmp1, 2	1	(6+4+5+5+4+5+5) (1) – (1)(5+1)	28
Tmp3	Q33,Q40,Q50,Q28,Q30,Q31,Q47	Tmp3	1	(6+4+5+5+4+5+5) (25) – (1)(25)	825
Tmp4	Q33,Q40,Q50,Q28,Q30,Q31,Q47	Tmp1, 2, 3, 4	2	(6+4+5+5+4+5+5) (25) – (2)(5+1+25+25)	738
Tmp5	Q15 Q33,Q40,Q43,Q50,Q28,Q31, Q47	Tmp5	1	(5+6+4+7+5+5+5+5)(10000) – (1)(10000)	450,000
Tmp6	Q28,Q33,Q40, Q31,Q47	Tmp1, 2, 3, 4, 5, 6	3	(5+6+4+5+5)(50000) – (3)(5+1+25+25+10000+50000)	1,069,832
Tmp7	Q4,Q15,Q22,Q33,Q40,Q43,Q50, Q6, Q28,Q30,Q47	Tmp7	1	(6+5+3+6+4+7+5+4+7+5+4+5)(800000) – (1)(800000)	44,800,000
Tmp8	Q28,Q33,Q40, Q47	Tmp1, 2, 3, 4, 5, 6, 7, 8	4	(5+6+4+5)(1602400000) – (4)(5+1+25+25+10000+50000+ 800000 + 1602400000)	25,634,959,776
Tmp9	Q28,Q40, Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9	4	(5+4+5)(160240) – (4)(5+1+25+25+10000+50000+ 800000 + 1602400000 + 160240)	-6,411,437,824
Tmp10	Q6,Q15,Q22,Q40,Q43,Q50	Tmp10	1	(7+5+3+4+7+5)(200000) – (1)(200000)	6,000,000
Tmp11	Q6, Q15, Q40	Tmp10, 11	1	(7+5+4)(200000) – (1)(200000+200000)	2,800,000

Table D.11 (Continued)

Existing Node	Query	Derived Node	Number of Base Relations	Weight (*w(v)*)	
Tmp13	Q4,Q22,Q33,Q43,Q50,Q30,Q3, Q31,Q47	Tmp13	1	(6+3+6+7+5+4+4+5+5)(6000000) – (1)(6000000)	264,000,000
Tmp14	Q4,Q22,Q33,Q43,Q50,Q3,Q31, Q47	Tmp13, 14	1	(6+3+6+7+5+4+5+5) (6000000) – (1)(6000000+6000000)	234,000,000
Tmp15	Q33,Q47	Tmp1, 2, 3, 4, 5, 6, 7, 8, 13, 14, 15	5	(6+5)(607837751040) – (5)(5+1+25+ 25 +10000+50000+ 800000 + 1602400000 + 6000000 + 6000000 + 607837751040)	3,638,950,205,960
Tmp16	Q22,Q43,Q50,Q31,Q47	Tmp16	1	(3+7+5+5+5)(1500000) – (1)(1500000)	36,000,000
Tmp17	Q22,Q43,Q50,Q31,Q47	Tmp16, 17	1	(3+7+5+5+5)(227597) – (1)(1500000+227597)	3,962,328
Tmp18	Q22,Q43,Q50,Q31	Tmp13, 14, 16, 17, 18	2	(3+7+5+5)(863342789712) – (2)(6000000 + 6000000+ 1500000+227597 + 863342789712)	15,540,142,759,622
Tmp22	Q43,Q50,Q30	Tmp22	1	(7+5+4)(150000) – (1)(150000)	2,250,000
Tmp27	Q6, Q15	Tmp7, 10, 11, 27	2	(7+5)(32046400000) – (2)(800000+ 200000 +200000 + 32046400000)	320,461,600,000

Table D.12 The Weight of Ancestor Node of Directly Affected Node of the Second Experiment

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp11	Tmp12	Q40,Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	5	(4+5)(1288745976) – (5)(5+1+25+25 +10000 +50000+ 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976)	-2,864,117,576
	Tmp24	Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 24	7	(5)(3733997148) - (7)(5+1+25+25 +10000 +50000+ 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976 + 6000000 + 6000000+ 1500000+227597 + 863342789712 +3733997148)	-6,071,211,579,363
	Tmp25	Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 22, 24, 25	8	(5)(711150000) - (8)(5+1+25+25 +10000 +50000+ 800000 + 1602400000 + 160240+ 200000+ 200000+ 1288745976 + 6000000 + 6000000+ 1500000+227597 + 863342789712 +3733997148 + 150000+ 711150000)	-6,961,999,295,832
Tmp14	Tmp26	Q4	Tmp7, 13, 14, 26	2	(6)(3034636800000) - (2) (6000000 + 6000000 +800000 + 3034636800000)	12,138,521,600,000
Tmp18	Tmp19	Q22,Q43	Tmp7, 13, 14, 16, 17, 18, 19	3	(3+7)(460135200000) – (3)(6000000 + 6000000 + 1500000+227597 + 863342789712 + 8000000+ 460135200000)	630,852,848,073

Table D.12 (Continued

Directly Affected Node	Ancestor Node	Query	Derived Node	Number of Base Relations	Weight of Ancestor Node	
Tmp18	Tmp20	Q22,Q43	Tmp7, 13, 14, 16, 17, 18, 19, 10, 20	4	(3+7)(115033800000) – (4)(6000000 + 6000000+ 1500000+227597 + 863342789712 + 8000000+ 460135200000 + 200000 + 115033800000)	-4,603,796,869,236
	Tmp21	Q43	Tmp7, 13, 14, 16, 17, 18, 19, 10, 20, 5, 21	5	(7)(5751690000) – (5)(6000000 + 6000000+ 1500000+227597 + 863342789712 + 8000000 + 460135200000 + 200000 + 115033800000 + 10000 + 5751690000)	-7,181,165,256,545
	Tmp23	Q43	Tmp7, 13, 14, 16, 17, 18, 19, 10, 20, 5, 21, 22, 23	6	(7)(86275350000) – (6)(6000000 + 6000000+ 1500000 + 227597 + 863342789712 + 8000000 + 460135200000 + 200000 + 115033800000 + 10000 + 5751690000 + 150000 + 86275350000)	-8,579,438,053,854
Tmp27	Tmp28	Q15	Tmp5, 7, 10, 11, 27, 28	3	(5)(1602320000) – (3)(800000+ 200000 +200000 + 32046400000 + 10000 + 1602320000)	-92,938,190,000

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
      $S_{min} = S_\infty$; {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($S_{min}$) then $S_{min}$ = S
      }
      return($S_{min}$)
end;
```

Figure E.2 The Iterative Improvement (II) Algorithm

The Simulated Annealing algorithm is shown in Figure E.3.

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

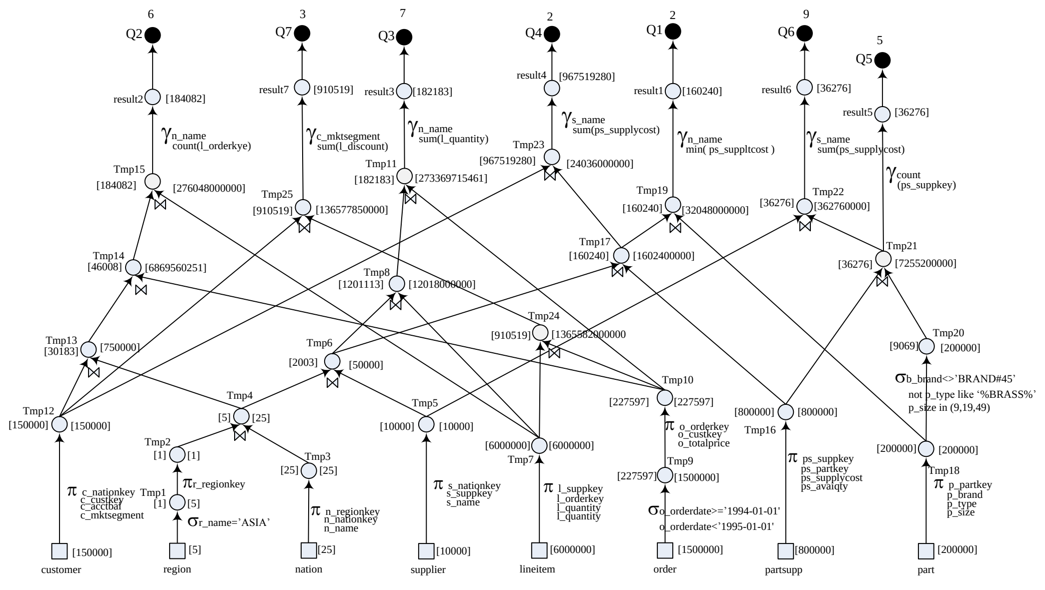

6
Q2
3
Q7
7
Q3
2
Q4
2
Q1
9
Q6
5
Q5
result2 [184082]
result7 [910519]
result3 [182183]
result4 [967519280]
result1 [160240]
result6 [36276]
result5 [36276]
γn_name count(l_orderkye)
γc_mktsegment sum(l_discount)
γn_name sum(l_quantity)
γs_name sum(ps_supplycost)
γn_name min(ps_suppltcost)
γs_name sum(ps_supplycost)
γcount (ps_suppkey)
Tmp15 [184082] [276048000000]
Tmp25 [910519] [136577850000]
Tmp11 [182183] [273369715461]
Tmp23 [967519280] [24036000000]
Tmp19 [160240] [32048000000]
Tmp22 [36276] [362760000]
Tmp21 [36276] [7255200000]
Tmp17 [160240] [1602400000]
Tmp14 [46008] [6869560251]
Tmp8 [1201113] [12018000000]
Tmp24 [910519] [1365582000000]
Tmp13 [30183] [750000]
Tmp6 [2003] [50000]
Tmp20 [9069] [200000]
σb_brand<>'BRAND#45' not p_type like '%BRASS%' p_size in (9,19,49)
Tmp10 [227597] [227597]
Tmp12 [150000] [150000]
Tmp4 [5] [25]
Tmp5 [10000] [10000]
Tmp16 [800000] [800000]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp7 [6000000] [6000000]
Tmp9 [227597] [1500000]
Tmp18 [200000] [200000]
Tmp1 [1] [5]
π c_nationkey c_custkey c_acctbal c_mktsegment
πr_regionkey
σr_name='ASIA'
π n_regionkey n_nationkey n_name
π s_nationkey s_suppkey s_name
π l_suppkey l_orderkey l_quantity l_quantity
π o_orderkey o_custkey o_totalprice
σo_orderdate>='1994-01-01' o_orderdate<'1995-01-01'
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type p_size
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] order
[800000] partsupp
[200000] part

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 E.1 The Result of II of 2PO for the Re-Optimized MVPP

	Total Cost of Initial State	Initial State	Total cost of Local Minimum State	Local Minimum State
1	8,427,206,080,471	{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}	6,732,979,232,178	{0,1,0,0,1,1,1,1,1,1,1,0,0,1,0,0,0,0,0,0,0,1,1,0,1}
2	8,144,566,532,161	{0,1,1,1,0,1,1,0,1,0,1,0,0,0,0,0,1,0,1,1,0,0,1,1,0}	7,188,391,010,923	{1,1,1,1,0,1,0,0,0,0,0,1,1,0,0,1,0,0,0,1,0,0,1,0,1}
3	8,136,058,882,811	{0,0,0, 1,0,0,0,0,1,0,1,1,0,1,0,0,1,0,0,1,1,0,0,1,0}	7,585,650,363,648	{0,0,1,0,0,1,1,1,0,0,0,1,1,1,0,0,1,1,0,0,0,1,0,0,1}
4	8,359,334,708,009	{0,1,0,0, 1,0,1,0,0,0,0,0,1,1,1,0,1,0,0,1,1,0,1,0,0}	7,116,835,738,107	{1,1,0,0,0,1,0,0,0,1,0,0,0,0,0,0,1,0,1,1,1,1,0,0,1}
5	6,708,459,311,979	{0,0,0,0,0,1,0,1,0,0,1,0,0,0,1,1,0,0,1,1,0,1,1,0, 1}	6,657,098,483,266	{0,1,0,0,1,0,1,0,0,0,1,1,1,1,1,0,0,0,1,0,0,1,1,0,1}
6	7,935,408,327,177	{1,1,0,0,0,0,0,0,0,1,0,0,1,0,1,1,0,1,0,1, 1,0,0,0,0}	6,624,915,356,922	{0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1}
7	7,820,946,154,678	{1,1,0,0,0,1,0,0,0,0,0,0,0,1,0,0,1,1,0,0, 1,0,0,1,0}	6,978,758,638,497	{0,0,1,0,0,0,0,0,0,1,0,1,1,1,0,0,0,0,0,1,1,0,0,0,1}
8	7,688,298,740,389	{0,0,0,0, 1,0,1,1,0,0,0,0,1,1,1,0,0,1,0,0,1,0,0,1,0}	6,775,762,552,571	{0,0,0,0,0,1,0,1,0,0,1,0,0,0,1,1,0,0,1,1,0,1,1,0,0}
9	7,499,403,494,665	{0,1,0,0,0,1,1,0,1,1,1,1,1,1,1,0,0,1,0,0,0,1,0,1, 0}	7,106,331,542,563	{1,1,1,1,0,1,0,0,0,0,0,1,1,0,0,1,0,0,0,0,0,0,1,0,1}
10	6,768,696,656,658	{0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0,1, 1,1,0,1,1,0}	6,768,699,388,215	{0,0,1,0,0,1,1,0,0,0,0,1,0,0,0,1,0,0,1,1,1,1,0,1,0}

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, 0,0,0,0,1,0,0,0,1,0, 0,0,1,0,1,0,0,1,0,1} represent the nodes as [Tmp19,0], [Tmp18,0], [Tmp17,0], [Tmp16,0], [Tmp6,0], [Tmp5,0], [Tmp4,0], [Tmp3,0], [Tmp2,0], [Tmp1,1], [Tmp15,0], [Tmp7,0], [Tmp14,0], [Tmp10,1], [Tmp9,0], [Tmp13,0] [Tmp12,0], [Tmp11,1], [Tmp8,0], [Tmp23,1], [Tmp21,0], [Tmp20,0], [Tmp22,1], [Tmp25,0], [Tmp24,1].

The value of arguments: $T_0 = 90$, T = 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, 0,0,0,0,1,0,0,0,1,0, 0,0,1,0,1,0,0,1,0,1}

Table E.2 The Result of SA of 2PO for the Re-Optimized MVPP

T (T_{i-1}-0.7)	**S′**	**Cost(S) – Cost(S′)** **ΔC**	**probability** $e^{-\Delta C / T}$	**S**	**The State of S′**
				6,624,915,356,922	
90.0	6,624,915,356,947	25	0.7574	6,624,915,356,947	{0,0,1,0,1,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,1,0,0,1}
89.3	6,624,915,356,752	-170	-	6,624,915,356,752	{0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1}
88.6	6,624,915,356,747	-5	-	6,624,915,356,747	{0,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1}
87.9	6,624,915,536,915	180,168	0.0000	6,624,915,356,747	{1,0,0,0,1,0,1,0,0,0,0,0,0,1,0,0,0,1,0,1,0,0,1,0,1}
87.2	6,259,209,787,966	-365,705,568,781	-	6,259,209,787,966	{0,0,1,0,0,1,1,0,0,0,0,1,0,0,1,0,0,1,1,0,0,1,0,0,1}
86.5	6,259,209,842,028	54,062	0.0000	6,259,209,787,966	{0,0,1,0,1,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,0,1,0,0,1}
85.8	6,172,139,022,926	-87,070,765,040	-	6,172,139,022,926	{0,0,0,0,1,0,1,0,0,0,1,1,0,0,1,0,0,1,1,0,1,1,0,0,1}
85.1	6,124,035,032,590	-48,103,990,336	-	6,124,035,032,590	{0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1}
84.4	6,120,827,925,892	-3,207,106,698	-	6,120,827,925,892	{0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1}
83.7	6,168,924,166,116	48,096,240,224	0.0000	6,120,827,925,892	{0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,0,1,0,0,0,1}
:	:	:	:		:
:	:	:	:	6,120,827,925,892	:
2.5	6,120,829,653,489	1,727,597	0.0000	6,120,827,925,892	{0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,1,0,0,0,1}
1.8	6,120,827,925,897	5	0.0000	6,120,827,925,892	{0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1}
1.1	6,120,828,075,892	150,000	0.0000	6,120,827,925,892	{0,0,0,0,1,0,0,0,0,1,1,0,0,0,0,0,1,1,0,0,1,0,0,0,1}
0.4	6,120,827,926,004	112	0.0000	6,120,827,925,892	{0,0,0,0,1,0,1,0,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1}

So the output generated by SA is {0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0,1}. The total cost of this state is 6,120,827,925,892 [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. $M = \varphi$ ;
    2. Calculate $w(v)$
        $w(v) = \sum_{q \in O_v} f_q(q) * C_a^q(v) - \sum_{r \in I_v} f_u(r) * C_m^r(v)$
    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 $v$
        $C_s = \sum_{q \in O_v} \{ f_q(q) * C_a^q(v) - \sum_{u \in S_v \cap M} C_a^q(u) \} - \sum_{r \in I_v} f_u(r) * C_m^r(v)$
    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 $LV = \varphi$ ;
    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:

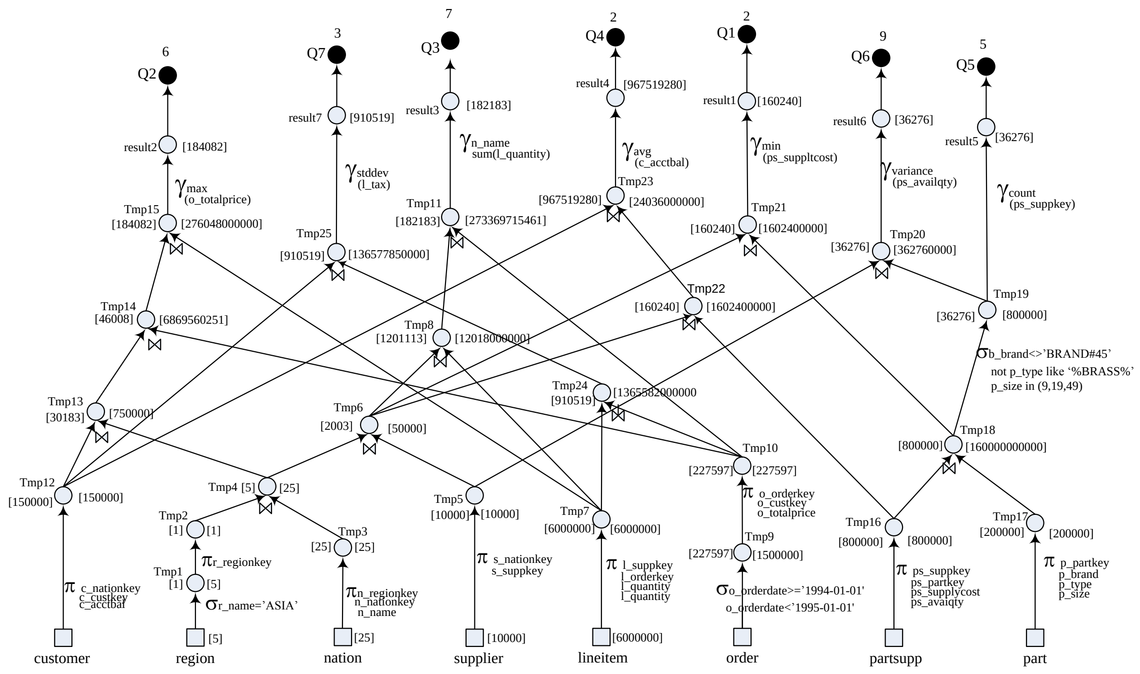


Figure E.6 The Third MVPP (the Cheapest MVPP), Query in the List: {Q3, Q2, Q6, Q1, Q5, Q4, and Q7}

Initially LV = {Tmp18, Tmp24, Tmp11, Tmp15, Tmp8, Tmp14, Tmp7, Tmp9, Tmp16, Tmp17, Tmp10, Tmp13, Tmp12, Tmp6, Tmp5, Tmp3, Tmp4, and Tmp1}

$M = \phi$

Starting with Tmp18;

O_{Tmp18} = {Q1, Q5, Q6},

I_{Tmp18} = {PART, PARTSUPP},

S_v = {Tmp16, Tmp17}

C_s = ((2+9+5) * (160000000000) - (1000000)) - 320,002,000,000

= 2,239,997,000,000 > 0, so Tmp18 is inserted into M.

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, 800000, 200000, 160000000000, 1602400000 and 160240 respectively. So, the query processing cost of query Q1 is

2 * (5 + 1 + 25 + 25 + 10000 + 50000 + 800000 + 200000 + 160000000000 + 1602400000 + 160240) = 323,207,240,592.

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 * (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

Materialized View	Maintenance Cost
Tmp11	1,426,977,515,570
Tmp15	1,414,630,939,520
Tmp18	320,002,000,000
Tmp24	2,731,167,475,194
Total	**5,892,777,930,284**

Table E.4 The Query Processing of the Cheapest MVPP

Query	Query Processing Cost
Query number 1 (Q1)	3,206,840,592
Query number 2 (Q2)	2,208,984
Query number 3 (Q3)	2,550,562
Query number 4 (Q4)	53,213,858,672
Query number 5 (Q5)	8,181,380
Query number 6 (Q6)	3,279,656,484
Query number 7 (Q7)	409,739,463,114
Total	**469,452,759,788**

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

```
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;
```

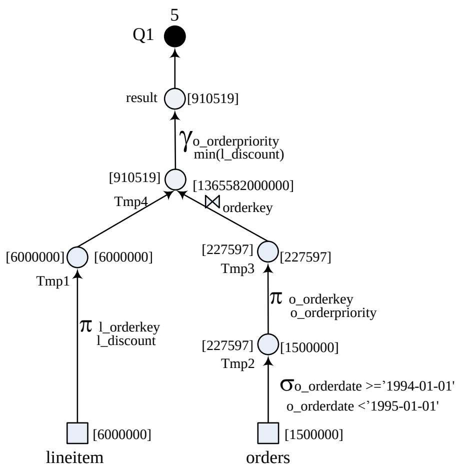


Figure F.1 Relational Algebra Query Tree of Query Q1

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

```
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;
```

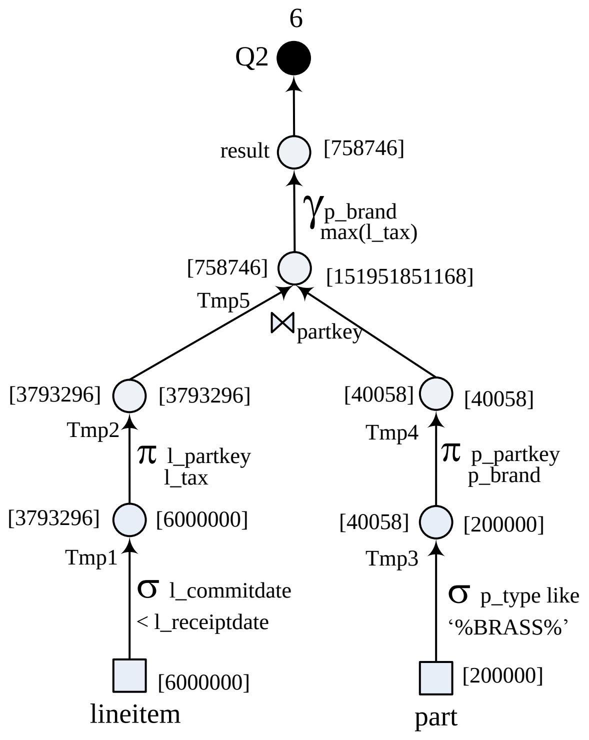


Figure F.2 Relational Algebra Query Tree of Query Q2

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;
```

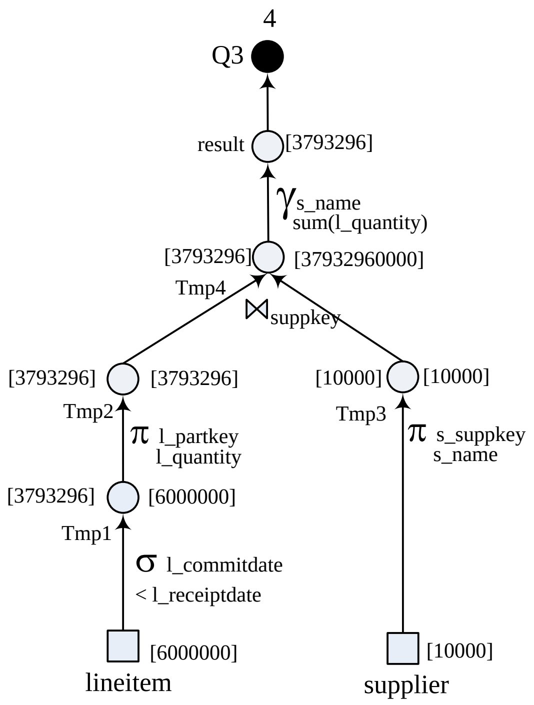


Figure F.3 Relational Algebra Query Tree of Query Q3

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

```
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;
```

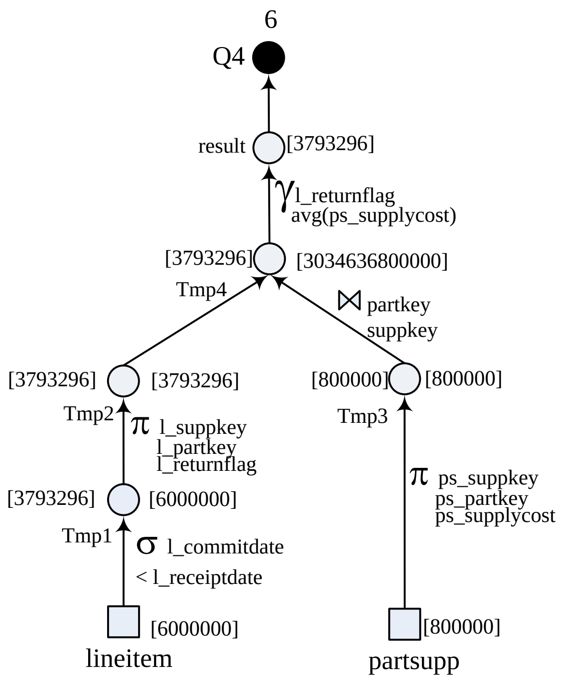


Figure F.4 Relational Algebra Query Tree of Query Q4

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

```
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;
```

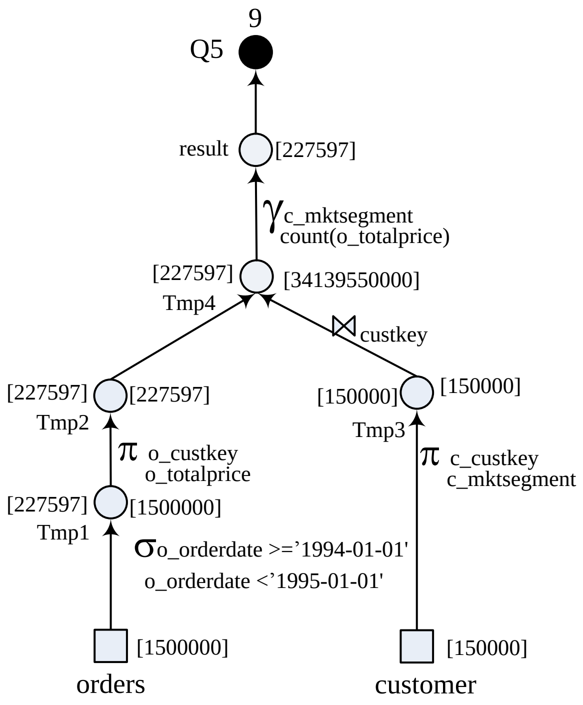


Figure F.5 Relational Algebra Query Tree of Query Q5

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;
```

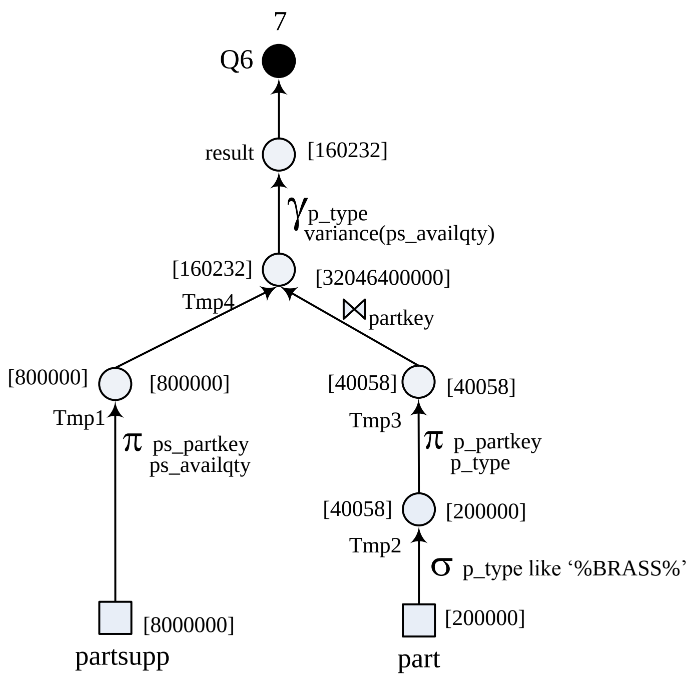


Figure F.6 Relational Algebra Query Tree of Query Q6

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;
```

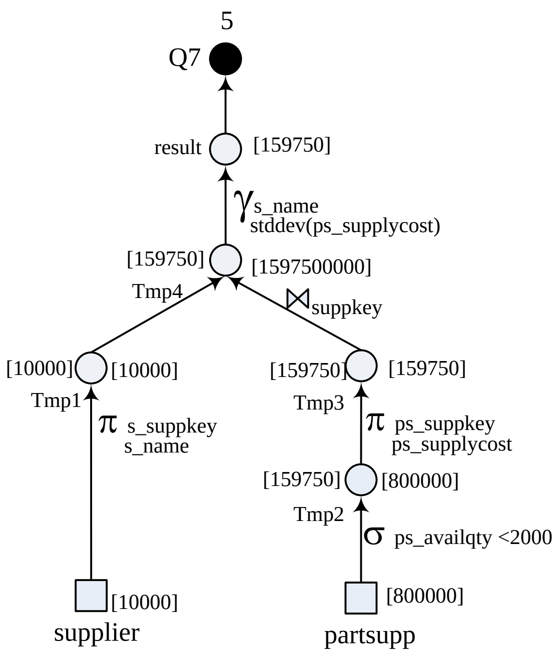


Figure F.7 Relational Algebra Query Tree of Query Q7

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

```
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;
```

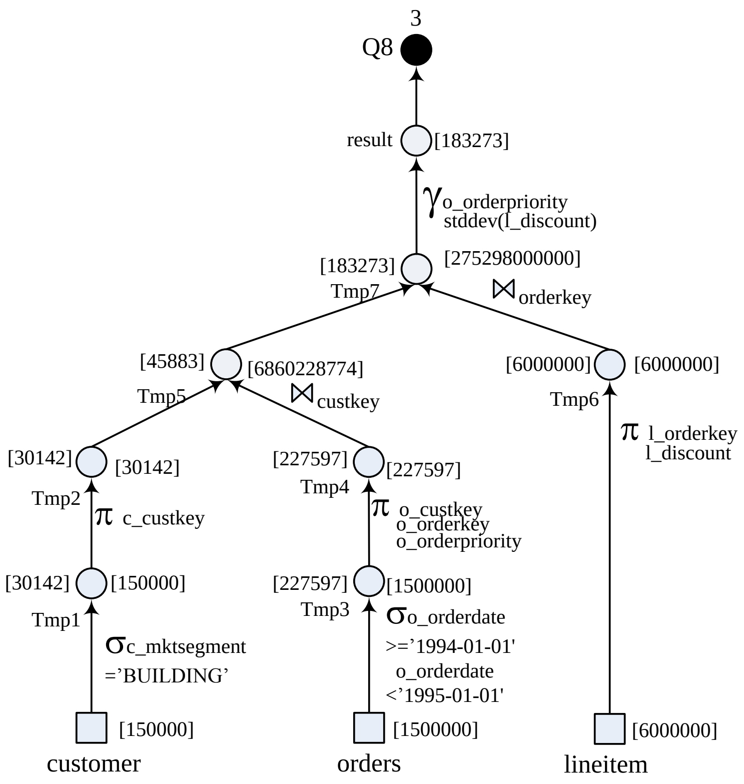


Figure F.8 Relational Algebra Query Tree of Query Q8

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

```
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;
```

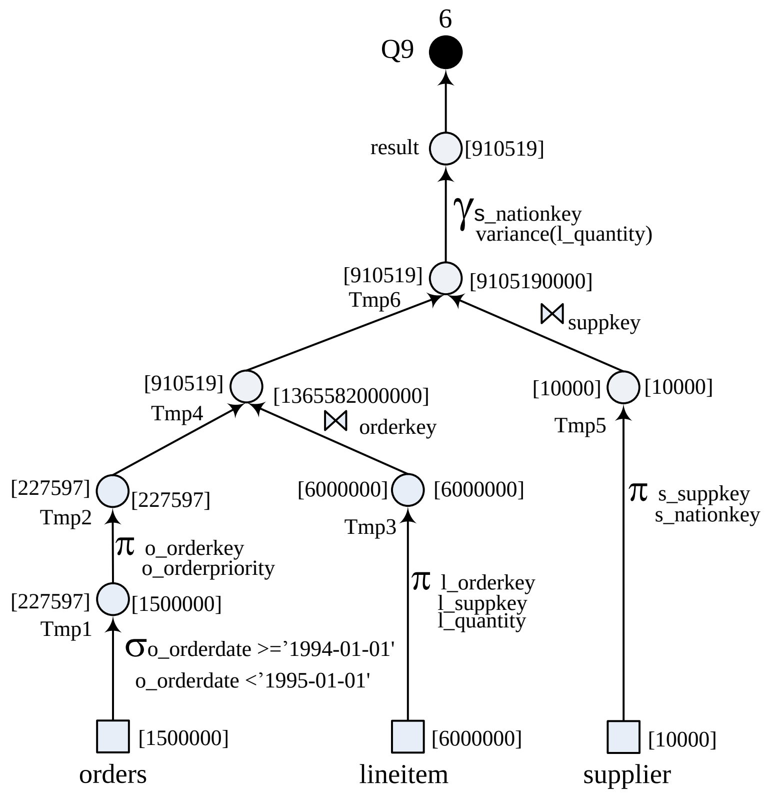


Figure F.9 Relational Algebra Query Tree of Query Q9

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

```
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;
```

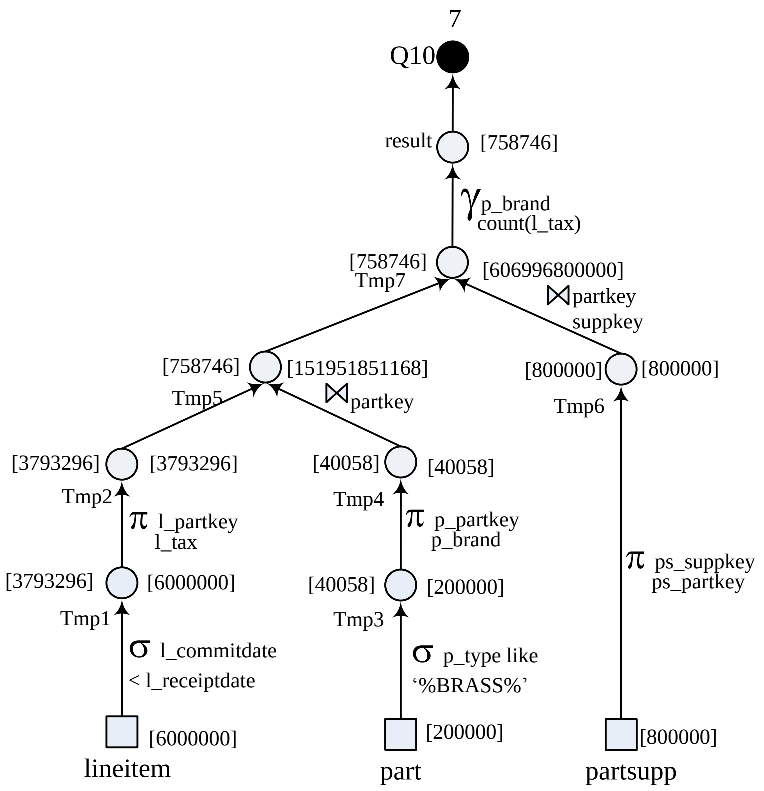


Figure F.10 Relational Algebra Query Tree of Query Q10

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;
```

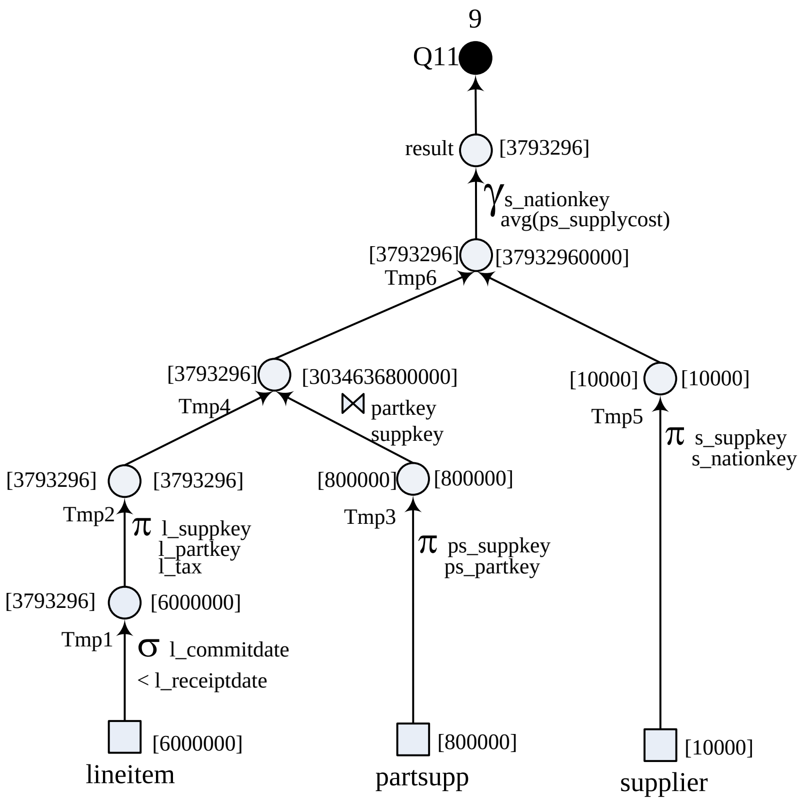


Figure F.11 Relational Algebra Query Tree of Query Q11

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

```
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;
```

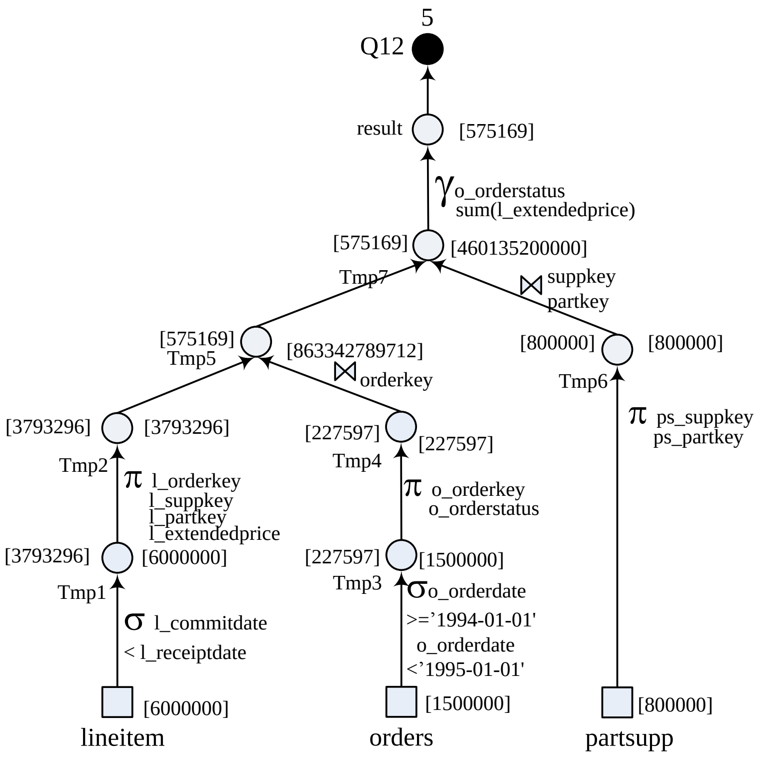


Figure F.12 Relational Algebra Query Tree of Query Q12

Query Q13 with the query frequency of 7 produces the standard deviation of tax for each types of order's priority that customer ordered in 1994. Its relational algebra tree is shown in Figure F.13.

Query Q13

```
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;
```

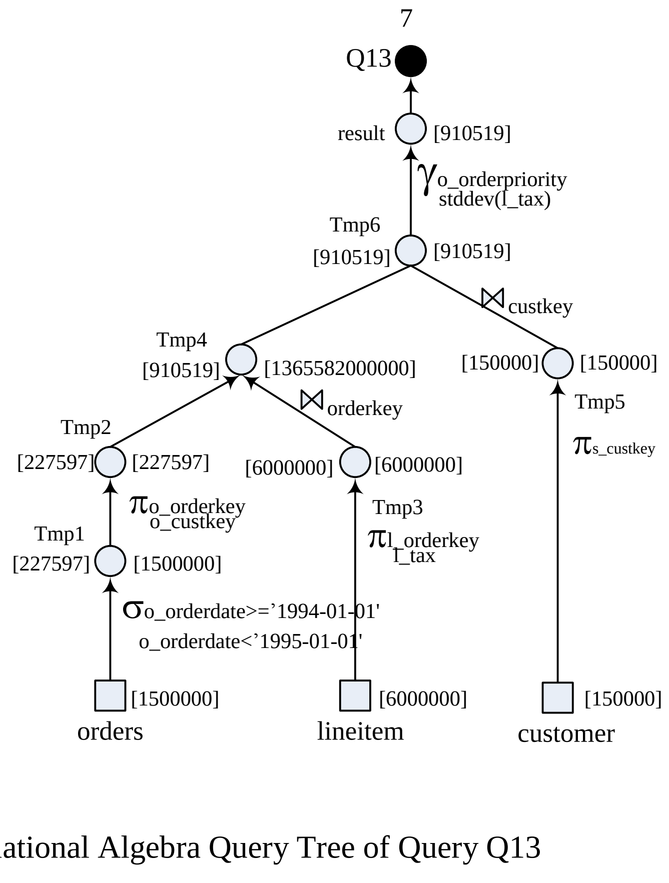


Figure F.13 Relational Algebra Query Tree of Query Q13

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

```
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;
```

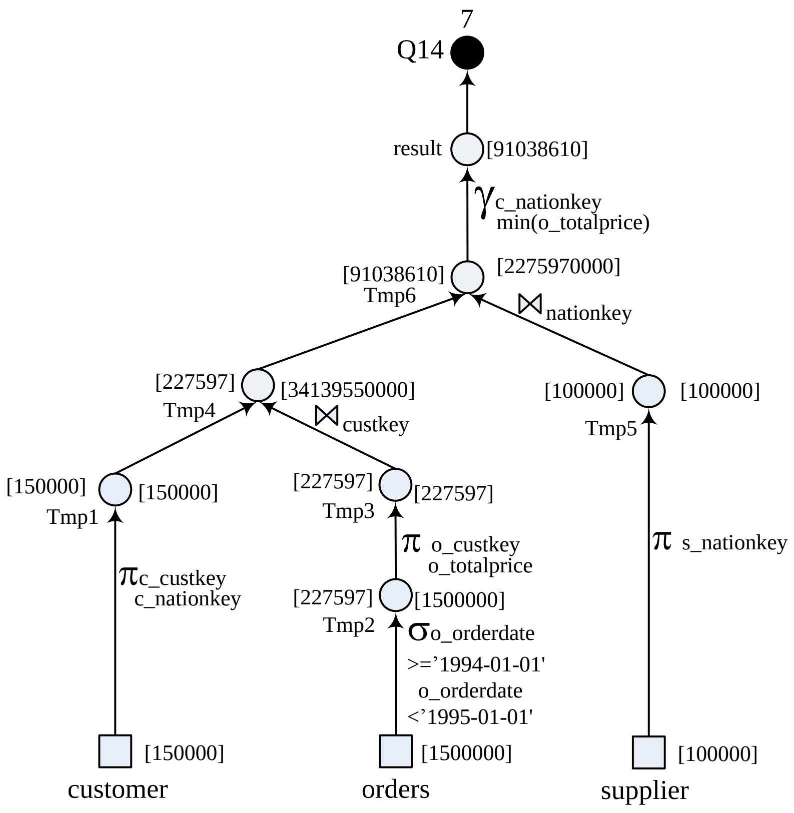


Figure F.14 Relational Algebra Query Tree of Query Q14

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;

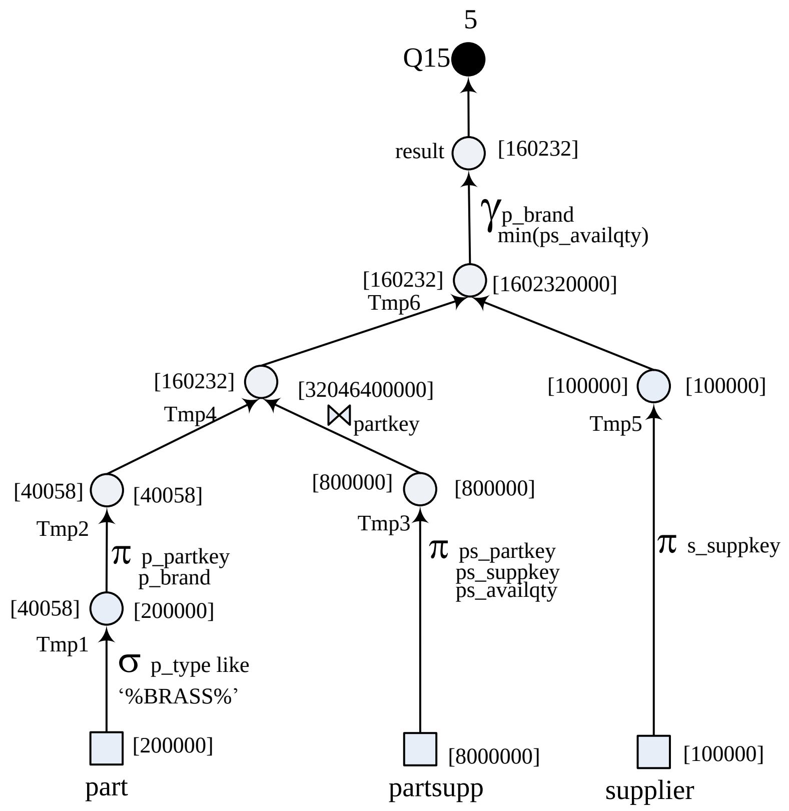


Figure F.15 Relational Algebra Query Tree of Query Q15

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

```
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;
```

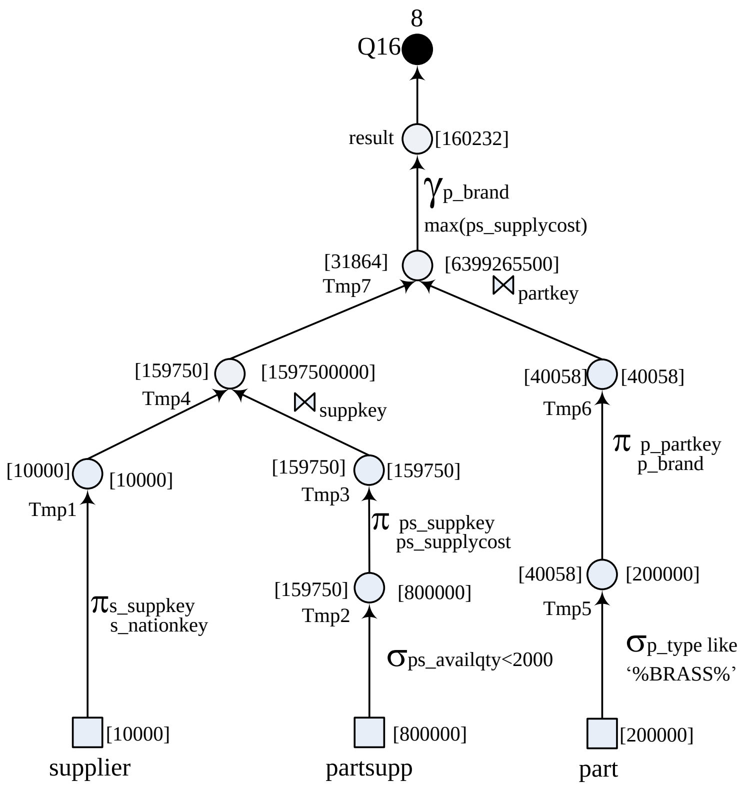


Figure F.16 Relational Algebra Query Tree of Query Q16

Query Q17 with the query frequency of 8 produces the summation of discount for each nation of customer specific market segment BUIILDING and ordered in 1994. Its relational algebra tree is shown in Figure F.17.

Query Q17

```
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;
```

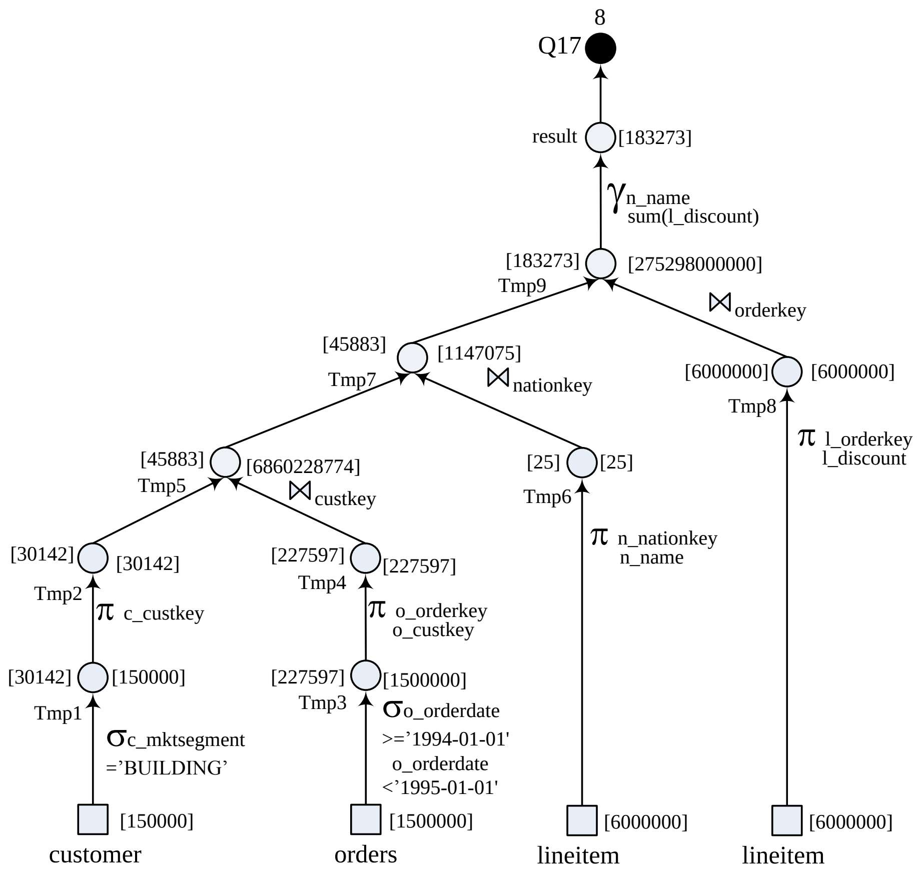


Figure F.17 Relational Algebra Query Tree of Query Q17

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

```
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;
```

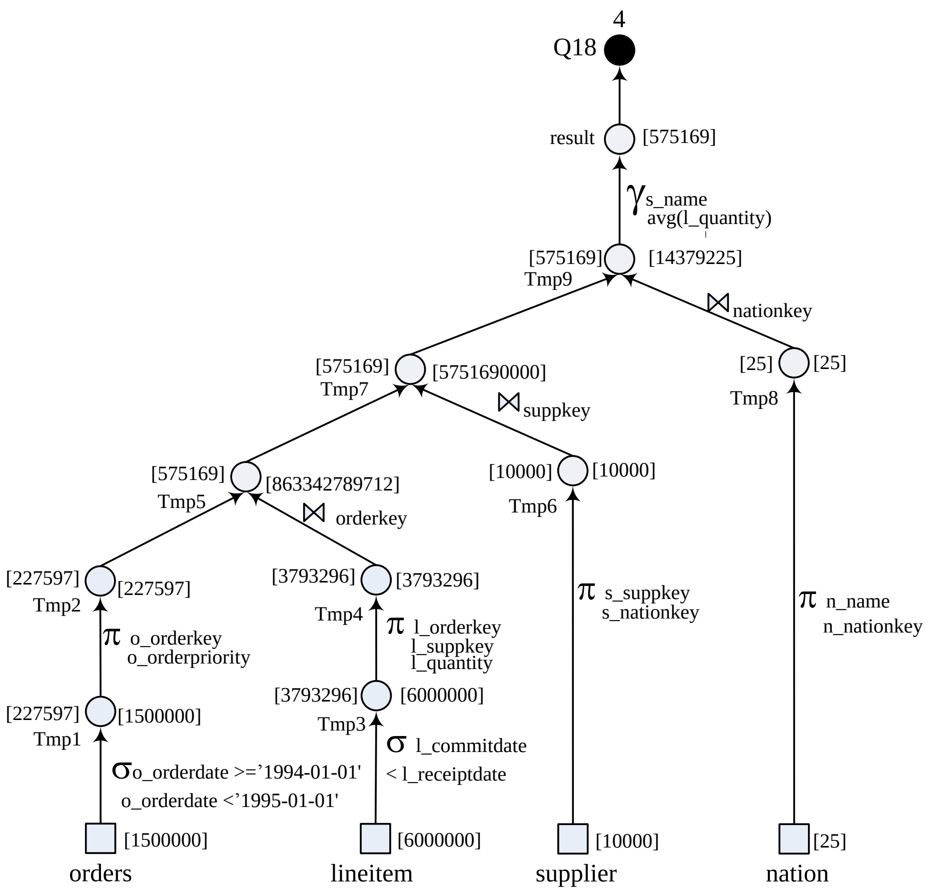


Figure F.18 Relational Algebra Query Tree of Query Q18

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;
```

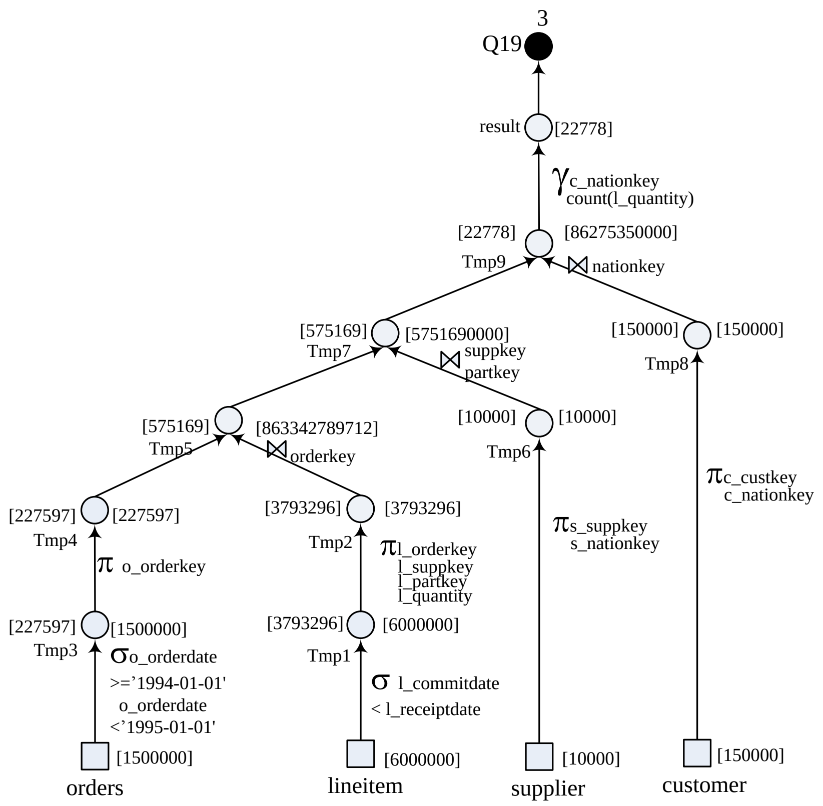


Figure F.19 Relational Algebra Query Tree of Query Q19

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;
```

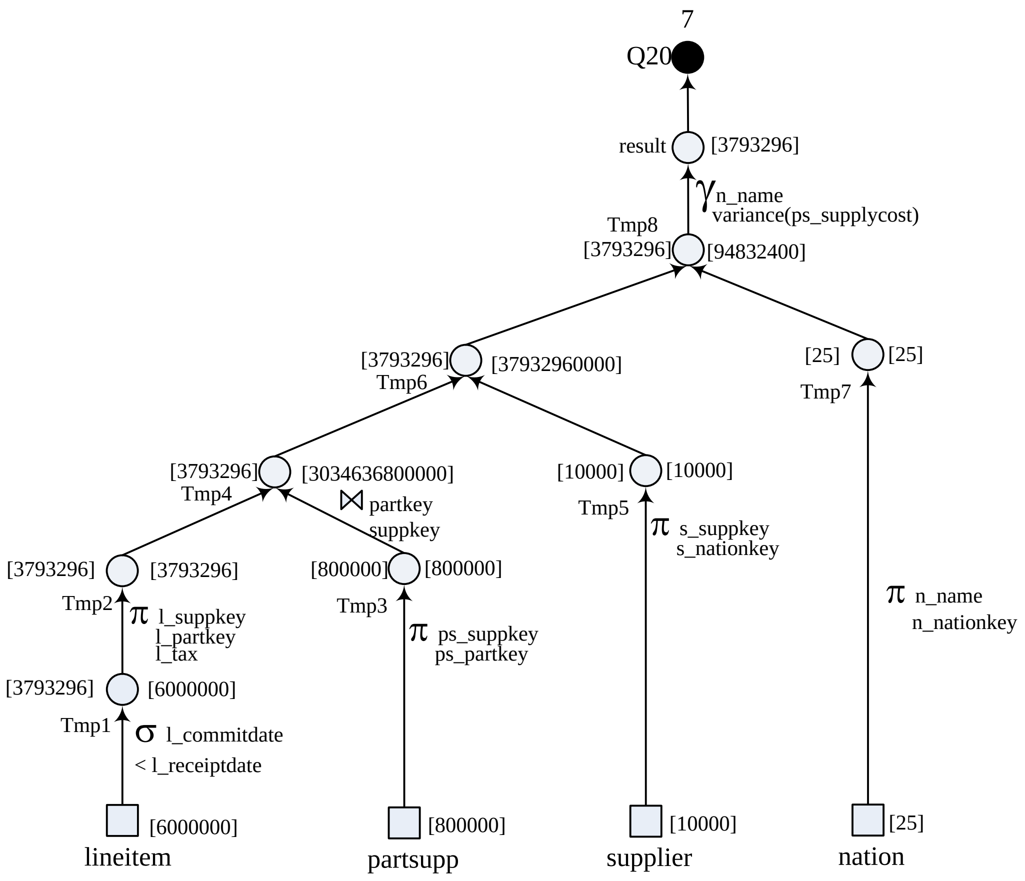


Figure F.20 Relational Algebra Query Tree of Query Q20

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;
```

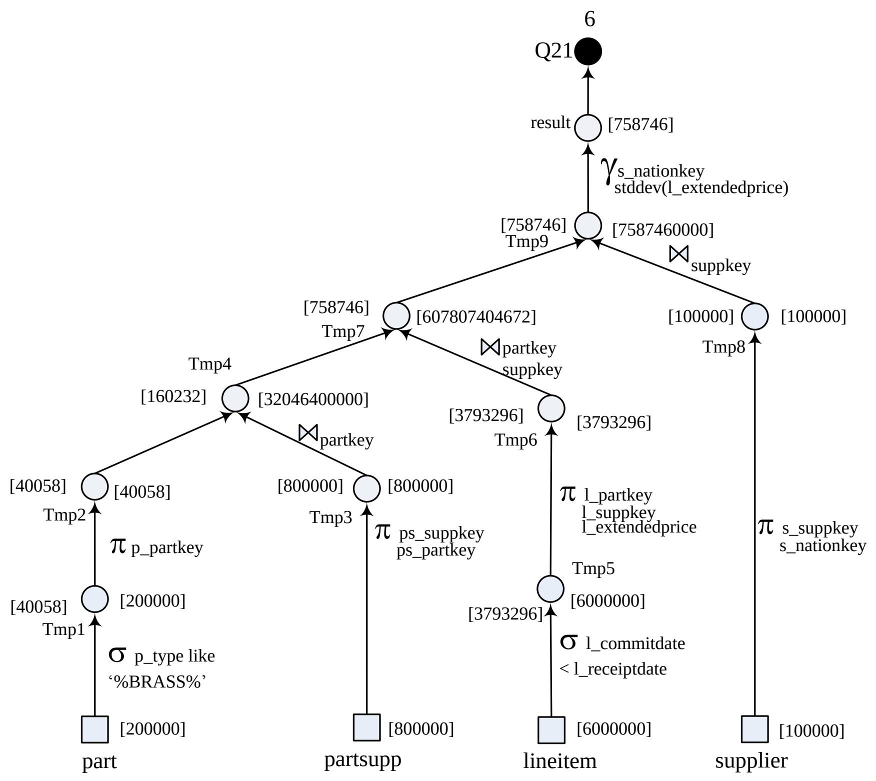


Figure F.21 Relational Algebra Query Tree of Query Q21

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;
```

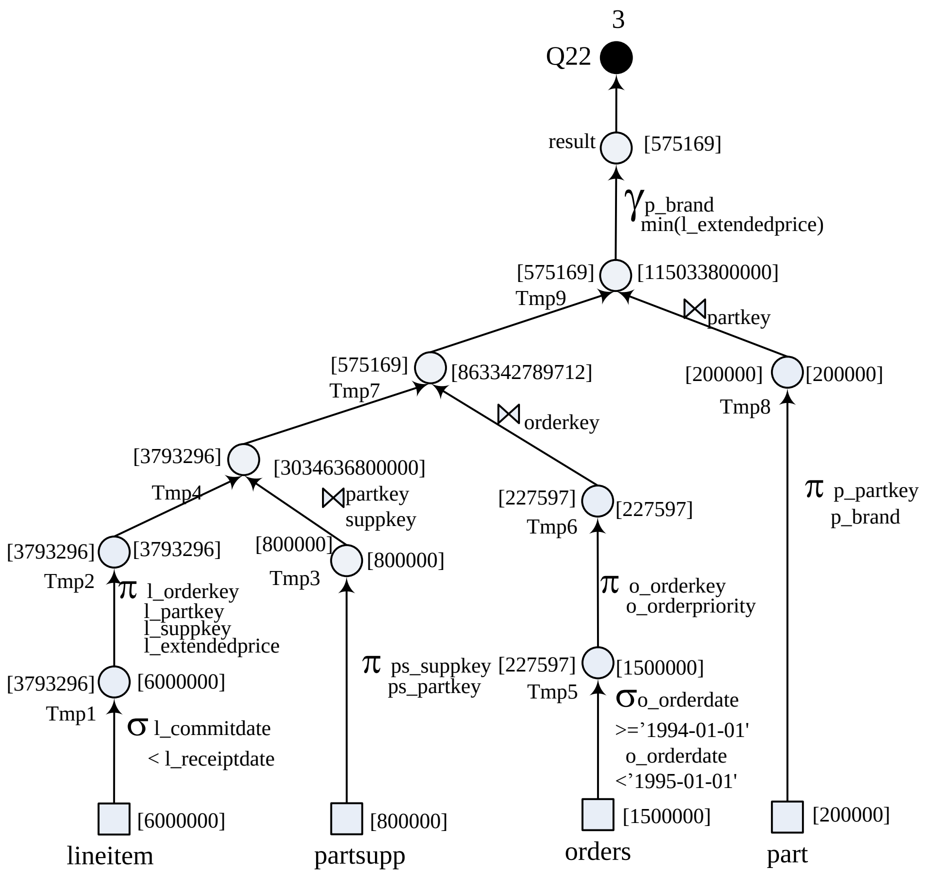


Figure F.22 Relational Algebra Query Tree of Query Q22

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;
```

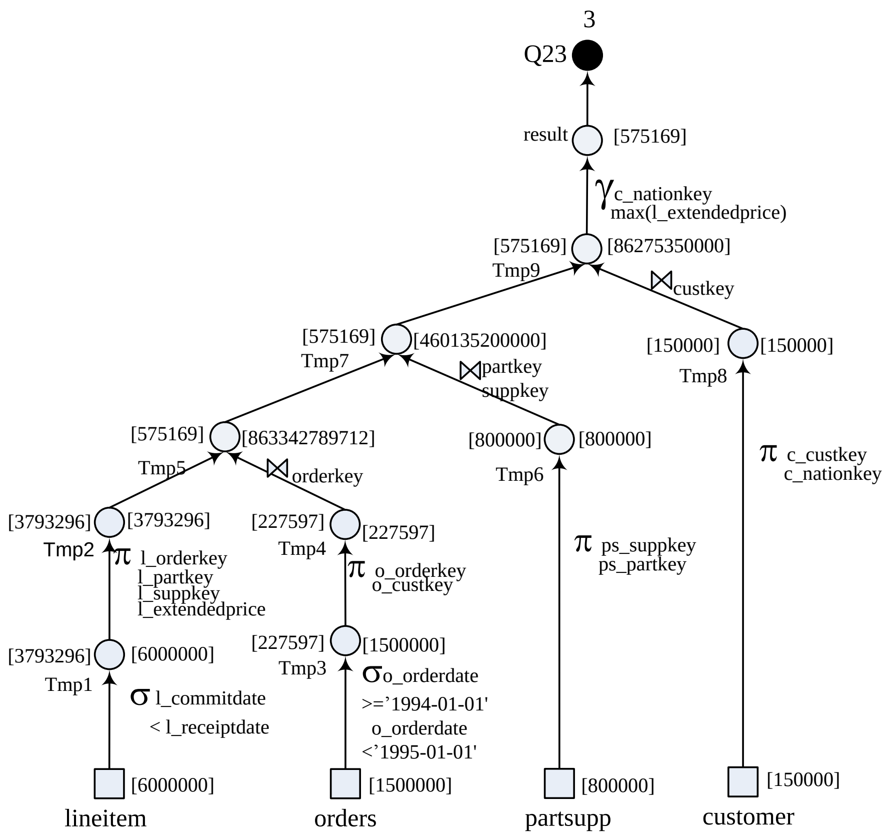


Figure F.23 Relational Algebra Query Tree of Query Q23

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;
```

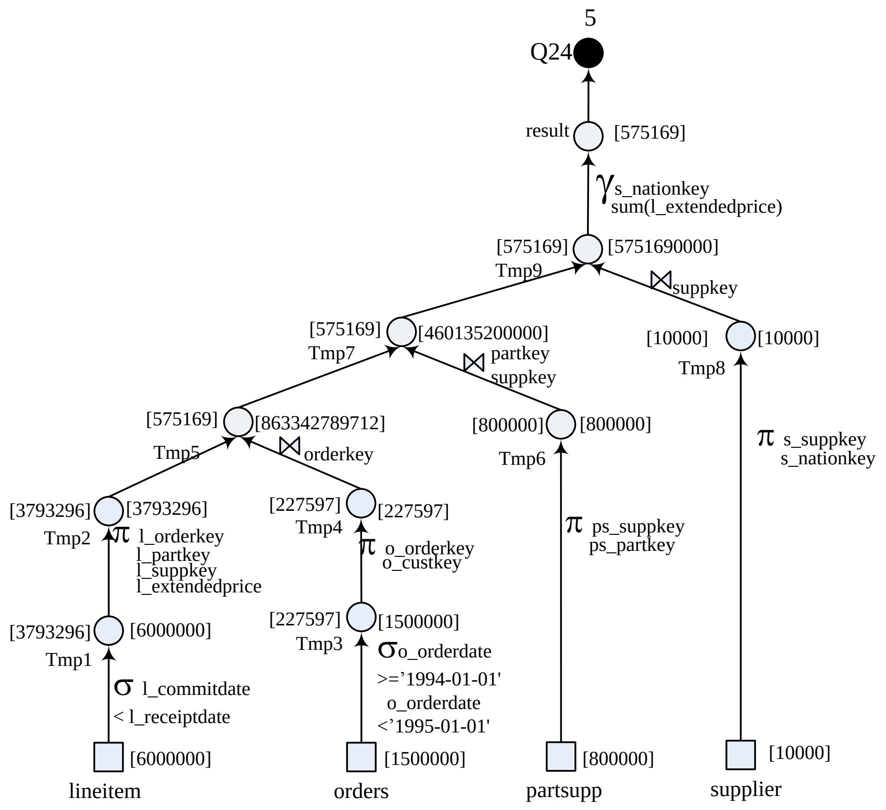


Figure F.24 Relational Algebra Query Tree of Query Q24

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;
```

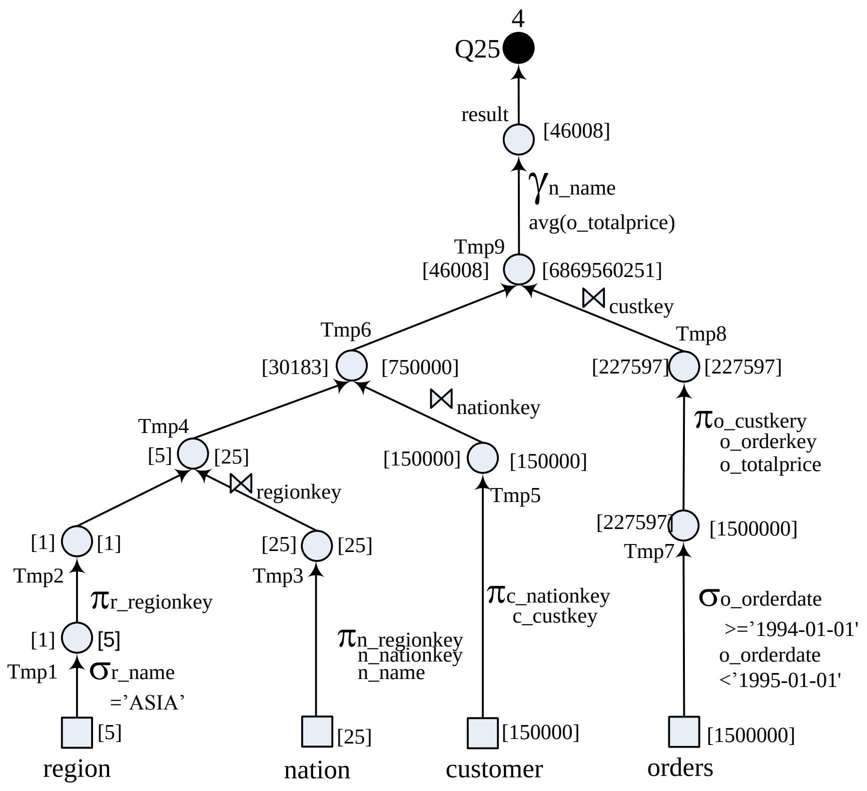


Figure F.25 Relational Algebra Query Tree of Query Q25

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

```
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;
```

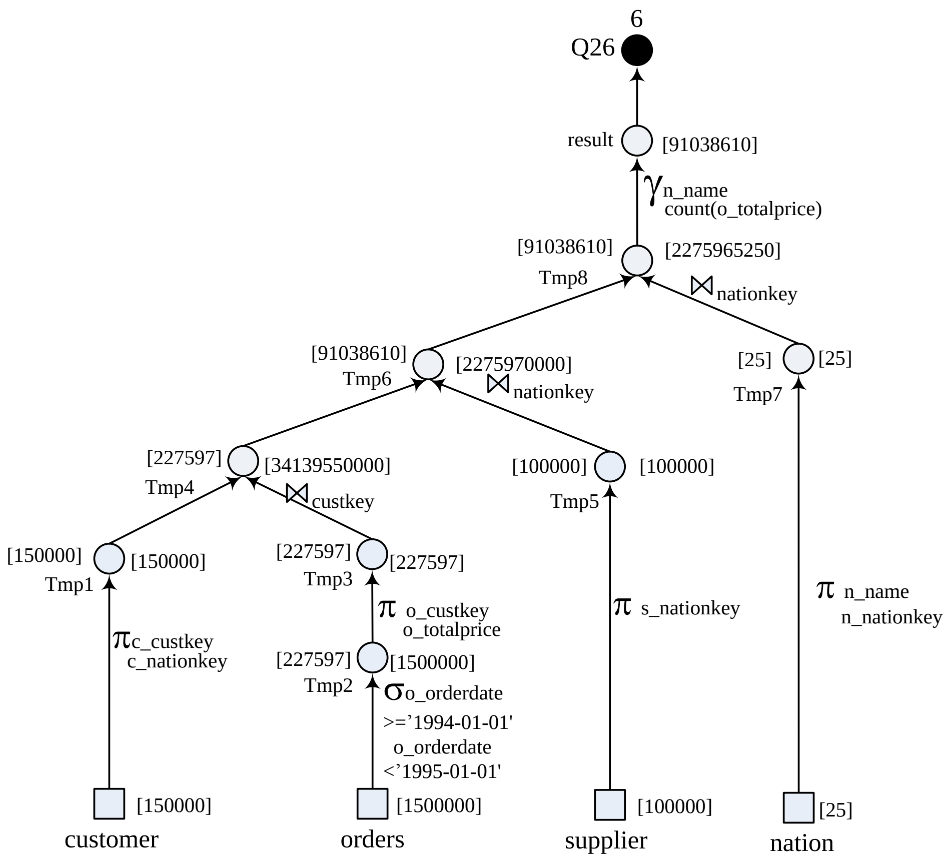


Figure F.26 Relational Algebra Query Tree of Query Q26

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;
```

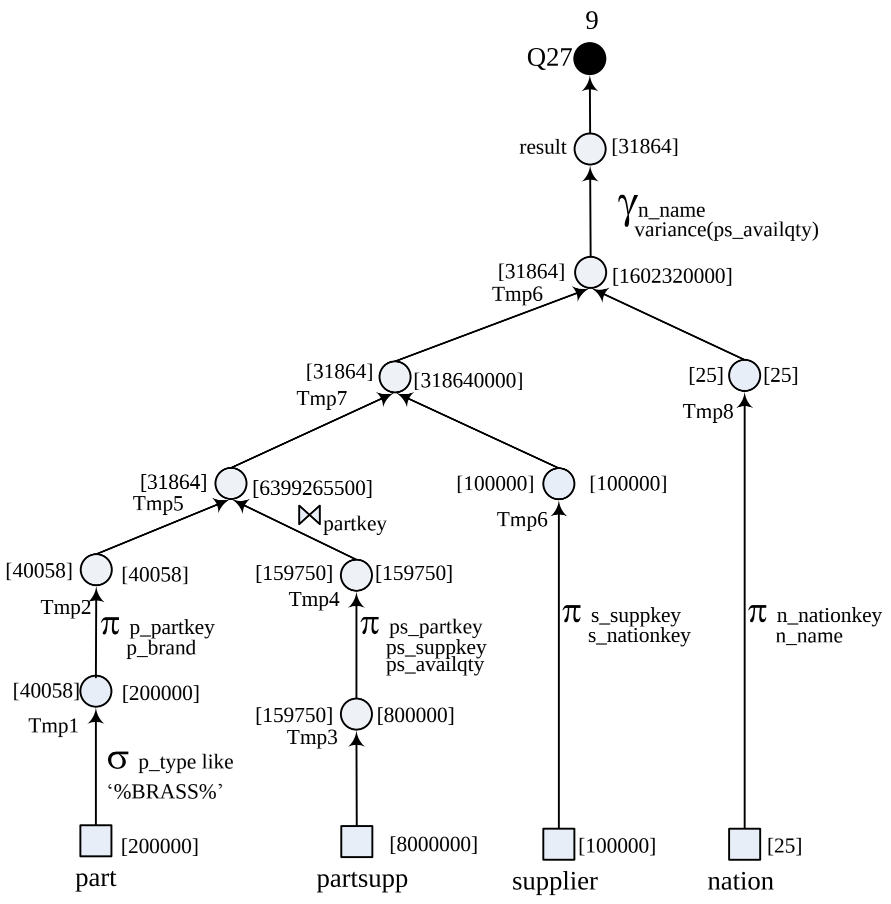


Figure F.27 Relational Algebra Query Tree of Query Q27

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;
```

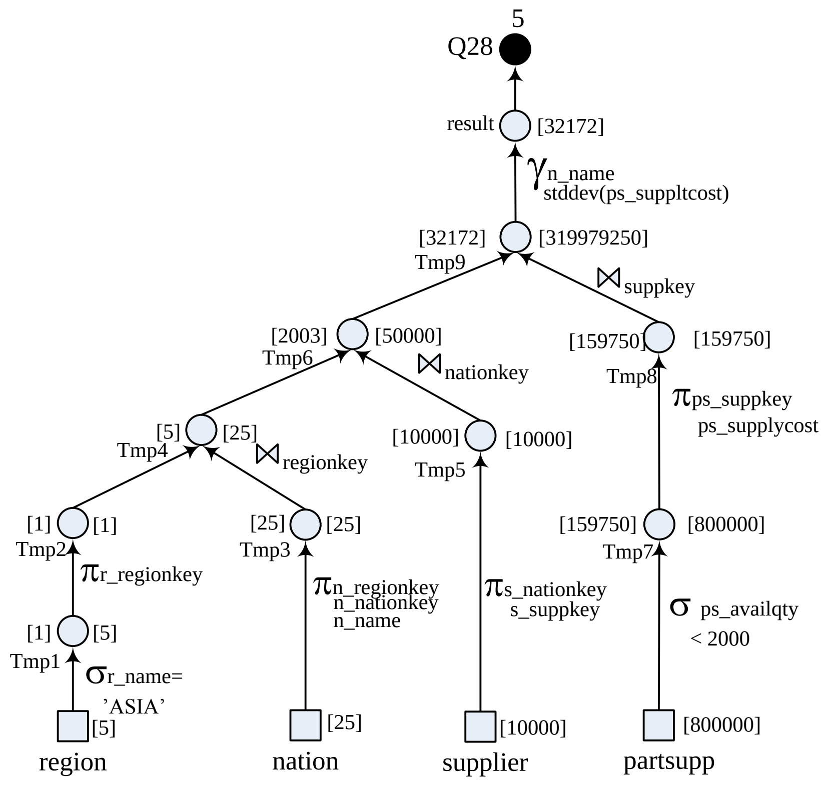


Figure F.28 Relational Algebra Query Tree of Query Q28

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;
```

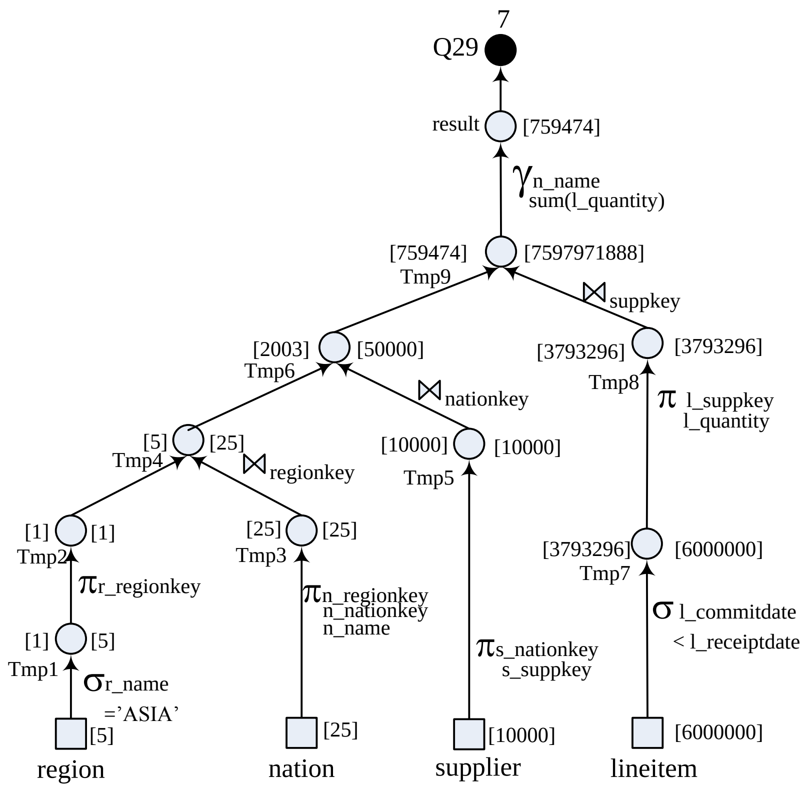


Figure F.29 Relational Algebra Query Tree of Query Q29

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;
```

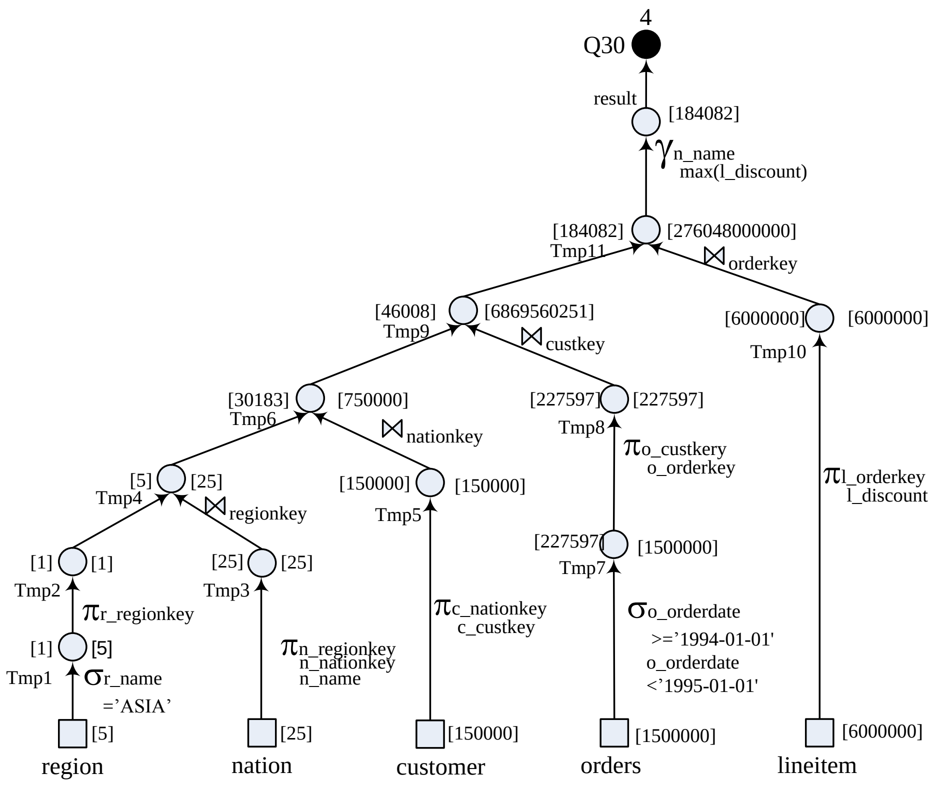


Figure F.30 Relational Algebra Query Tree of Query Q30

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

```
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;
```

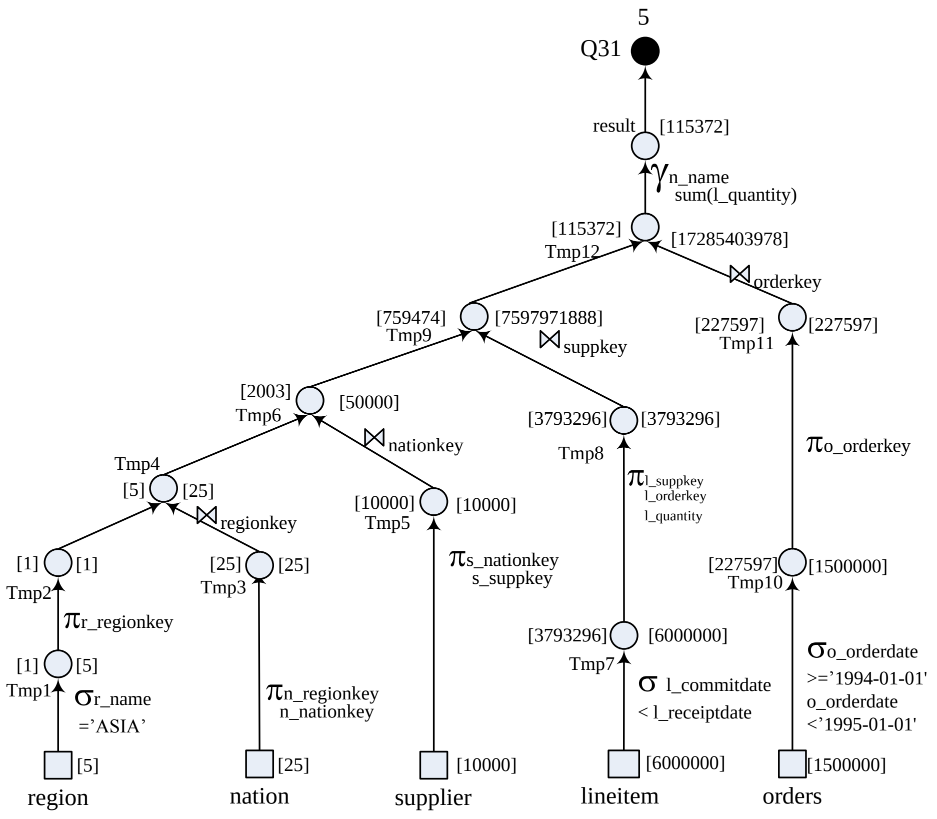


Figure F.31 Relational Algebra Query Tree of Query Q31

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

```
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;
```

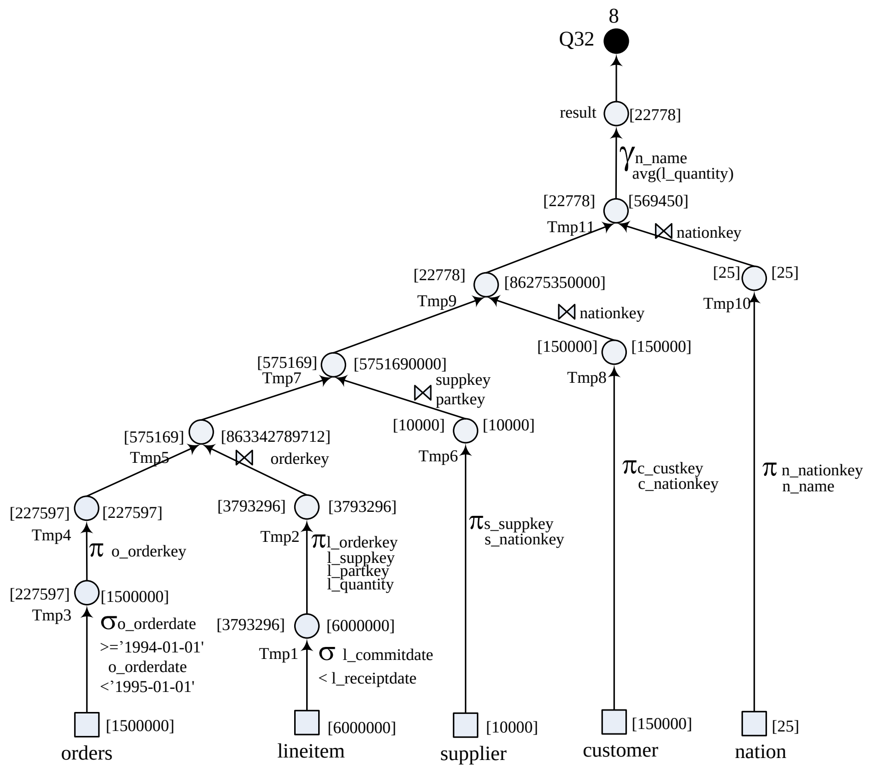


Figure F.32 Relational Algebra Query Tree of Query Q32

Query Q33 with the query frequency of 6 produces the number of supply part 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

```
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;
```

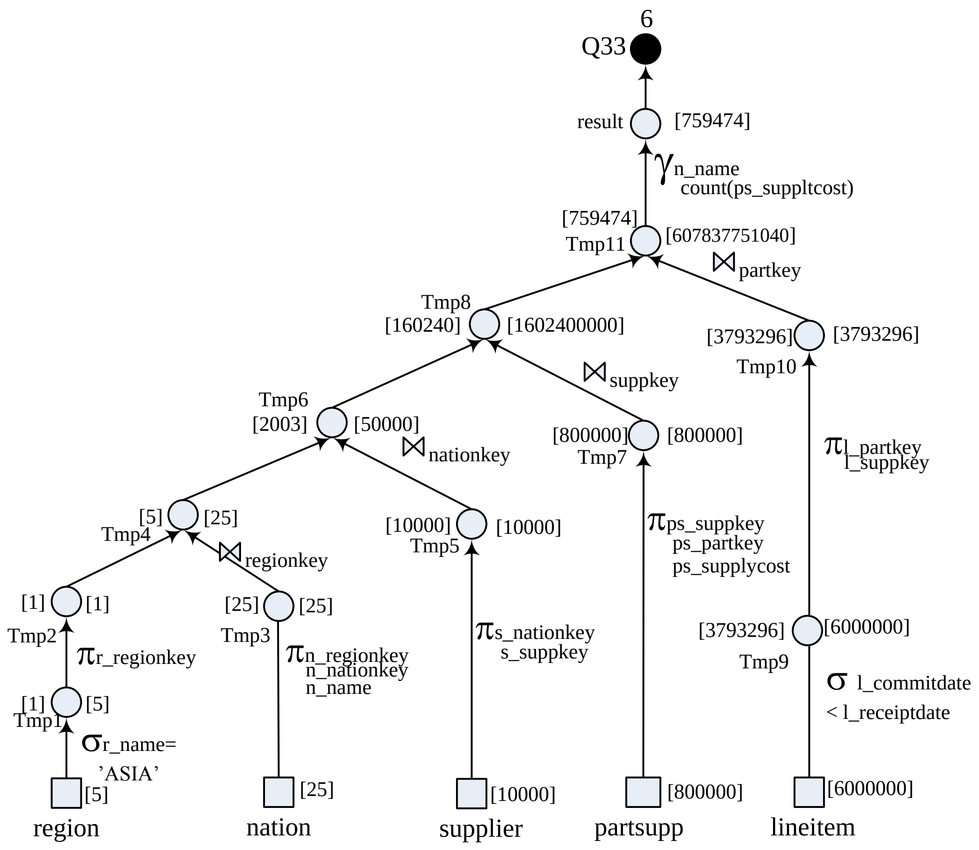


Figure F.33 Relational Algebra Query Tree of Query Q33

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

```
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;
```

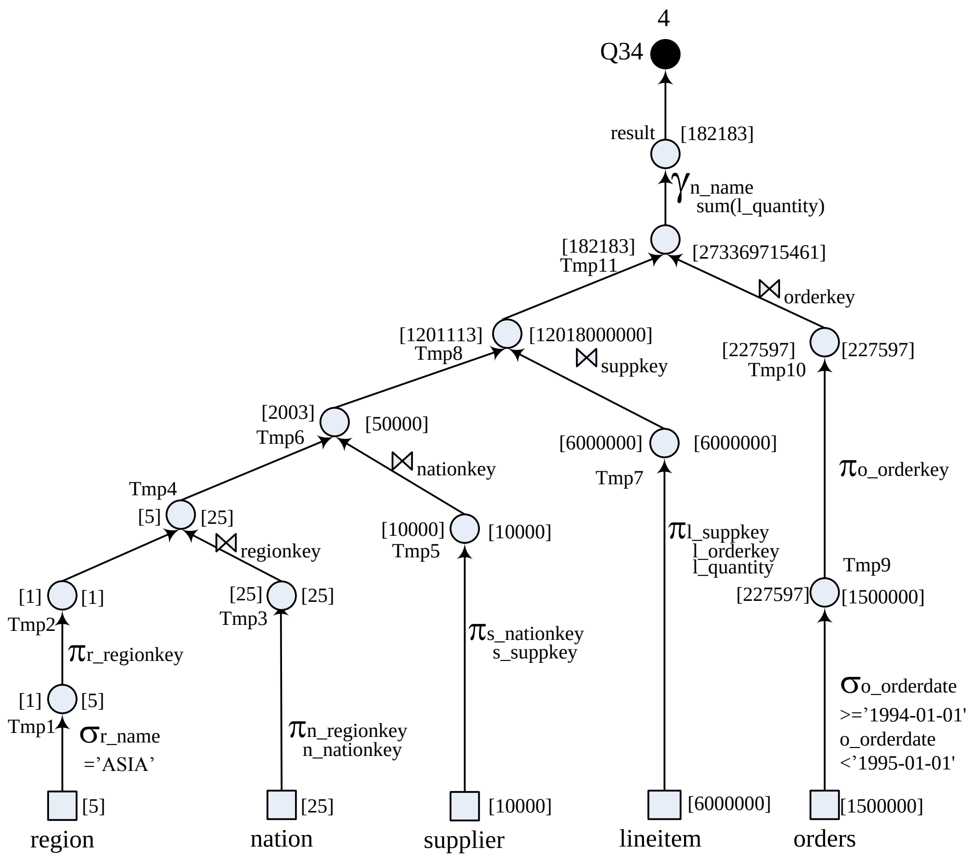


Figure F.34 Relational Algebra Query Tree of Query Q34

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;
```

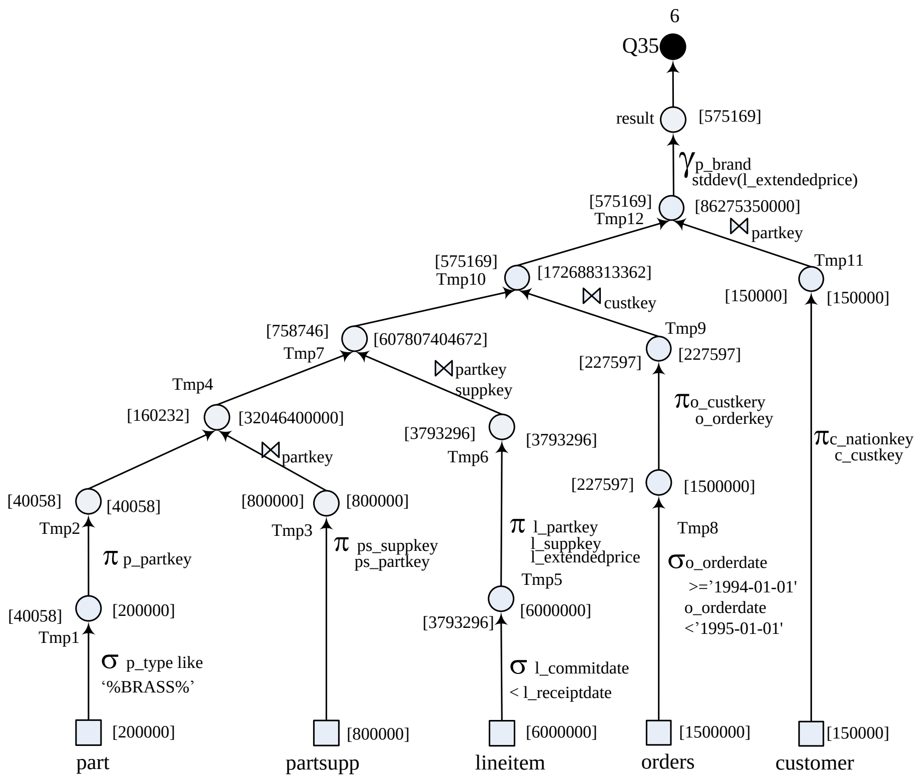


Figure F.35 Relational Algebra Query Tree of Query Q35

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

```
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;
```

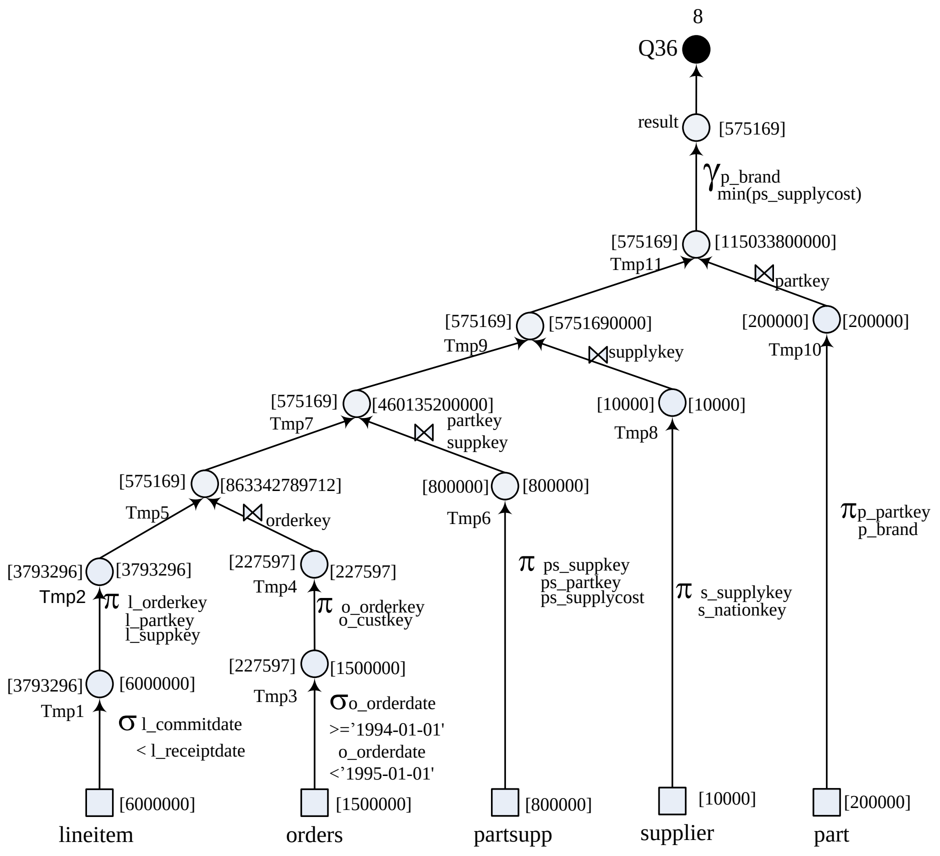


Figure F.36 Relational Algebra Query Tree of Query Q36

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

```
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;
```

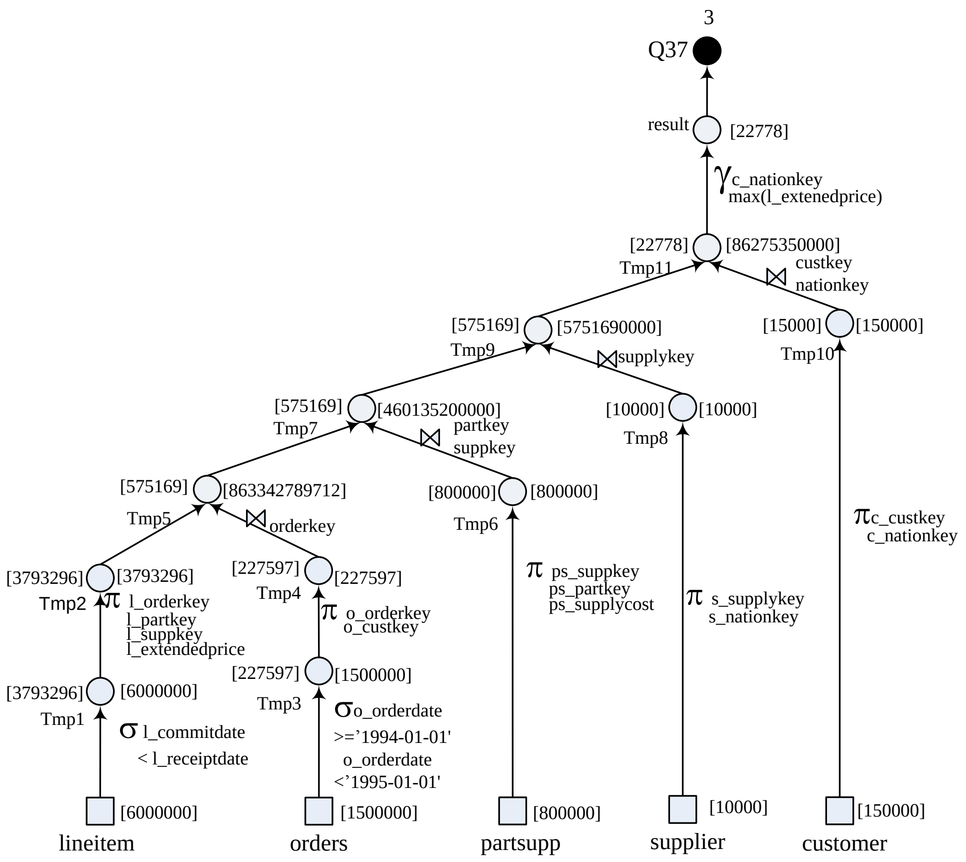


Figure F.37 Relational Algebra Query Tree of Query Q37

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;
```

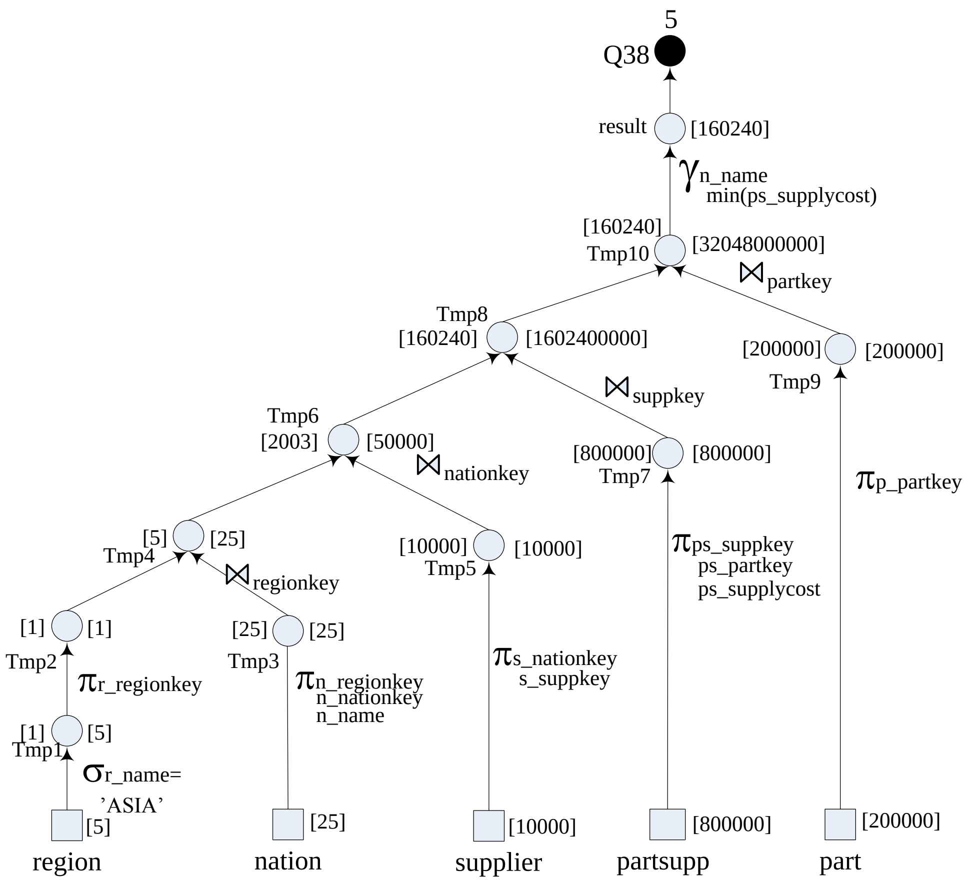


Figure F.38 Relational Algebra Query Tree of Query Q38

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;

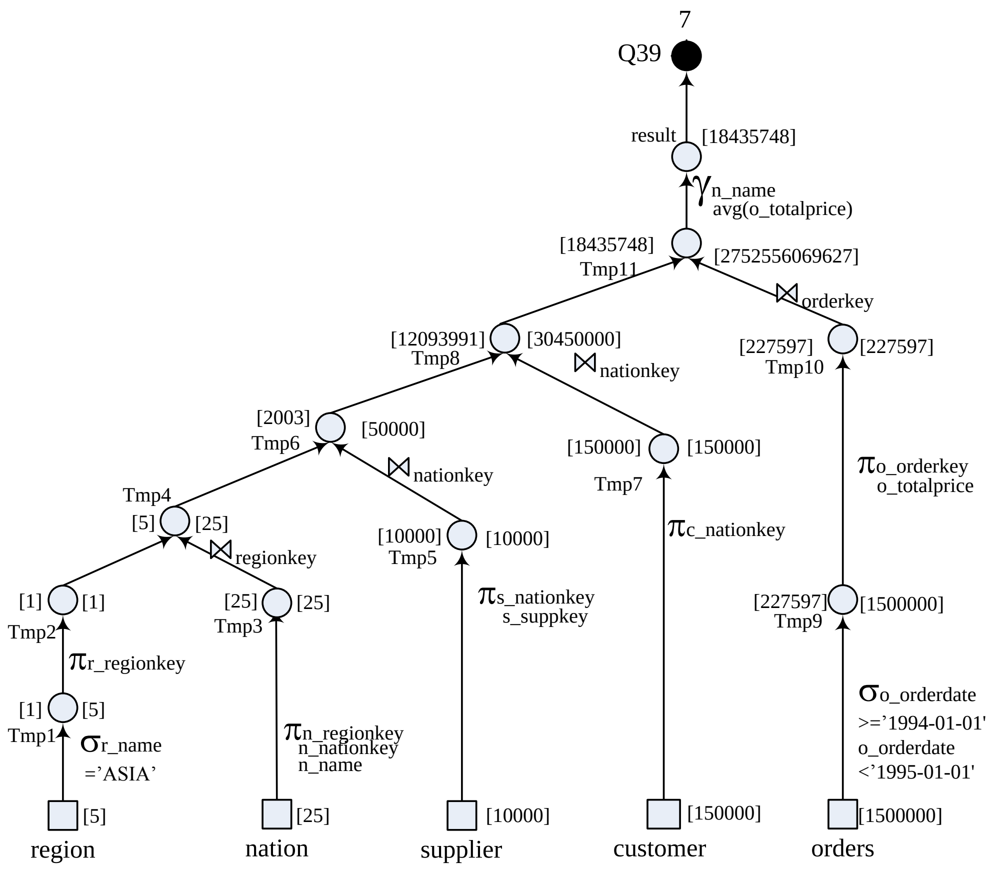


Figure F.39 Relational Algebra Query Tree of Query Q39

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

```
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;
```

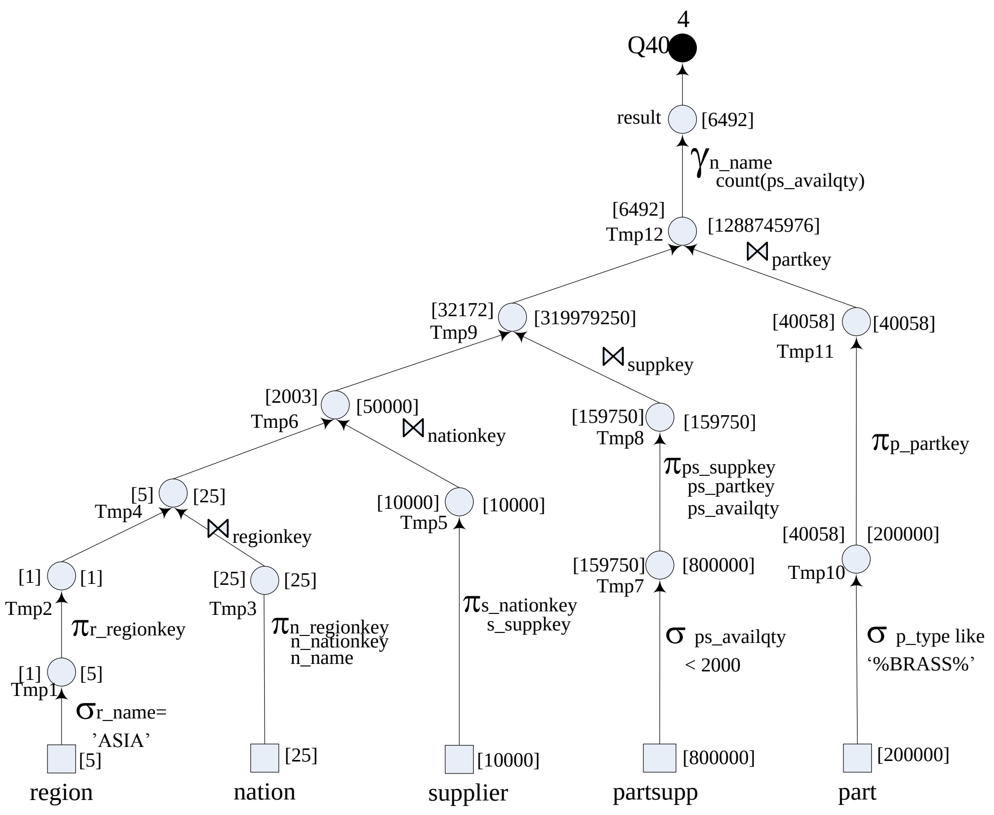


Figure F.40 Relational Algebra Query Tree of Query Q40

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

```
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;
```

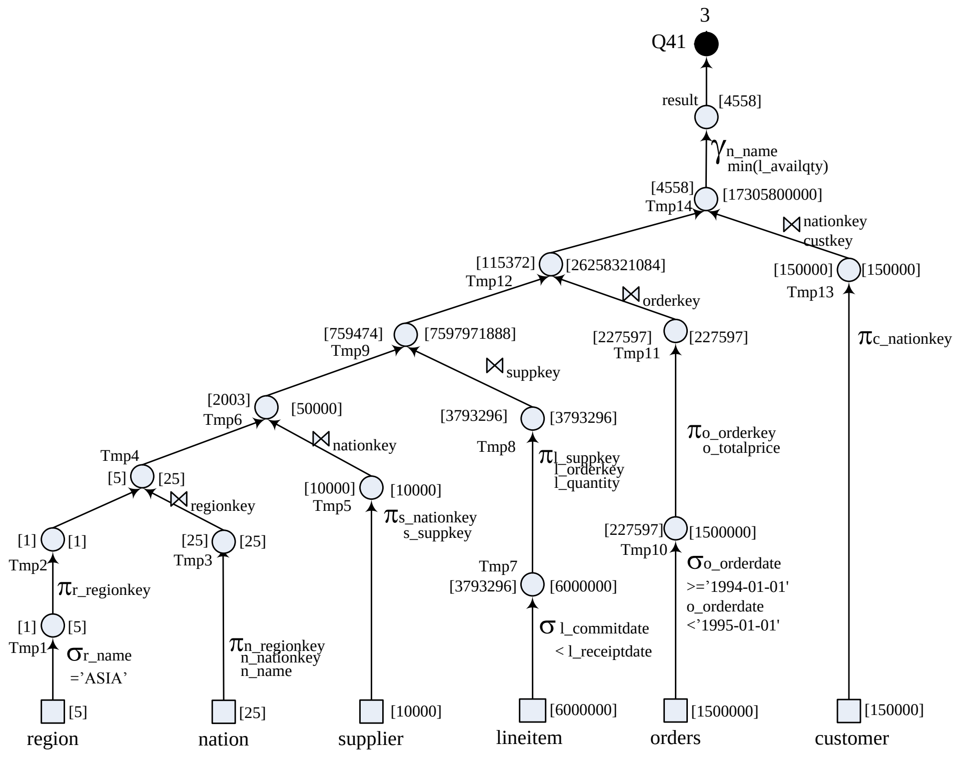


Figure F.41 Relational Algebra Query Tree of Query Q41

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;
```

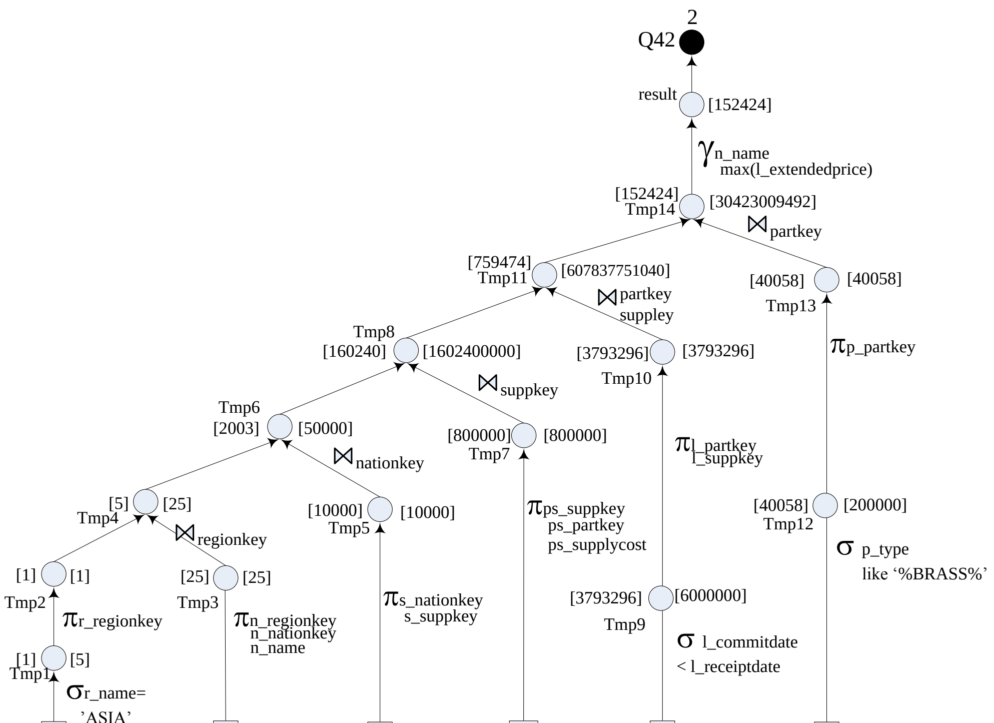


Figure F.42 Relational Algebra Query Tree of Query Q42

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;
```

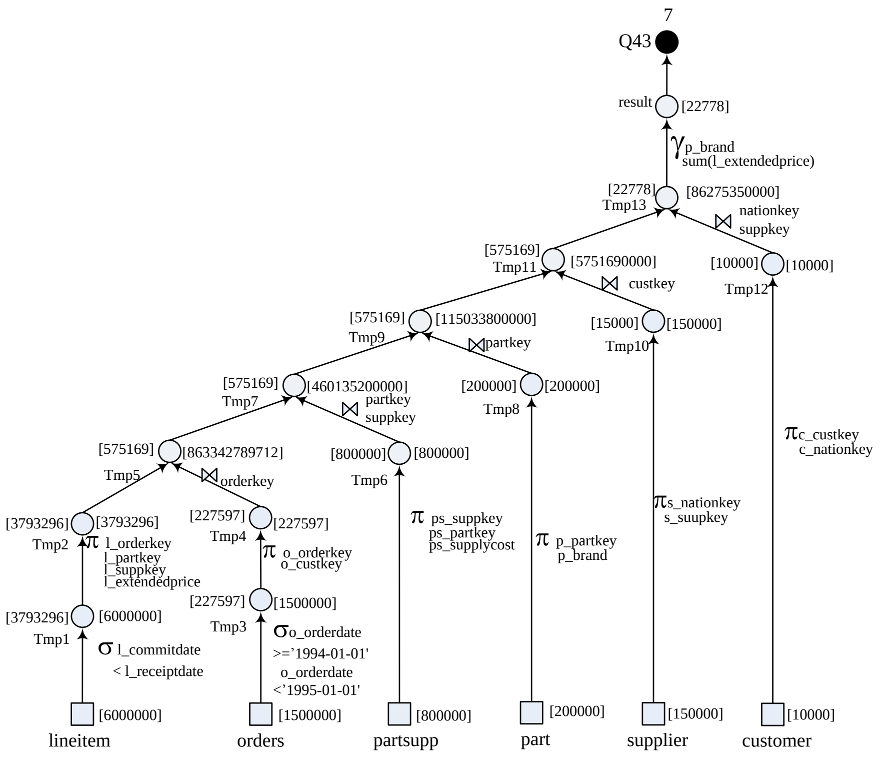


Figure F.43 Relational Algebra Query Tree of Query Q43

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;
```

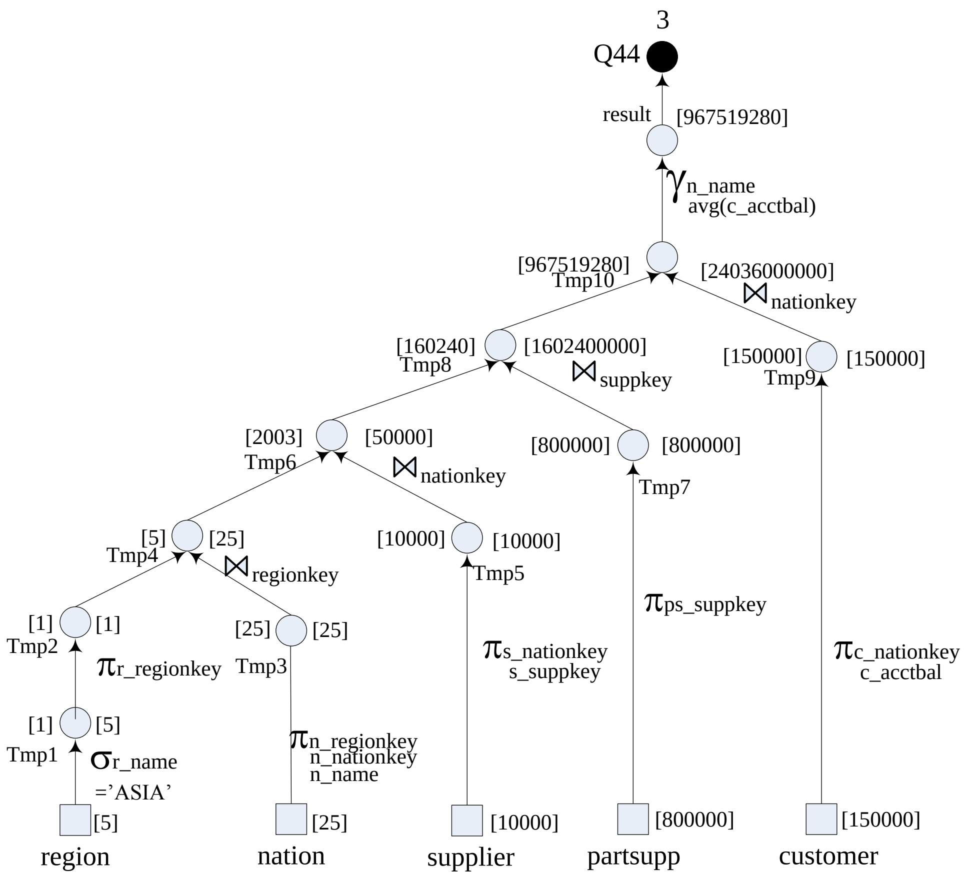


Figure F.44 Relational Algebra Query Tree of Query Q44

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

```
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;
```

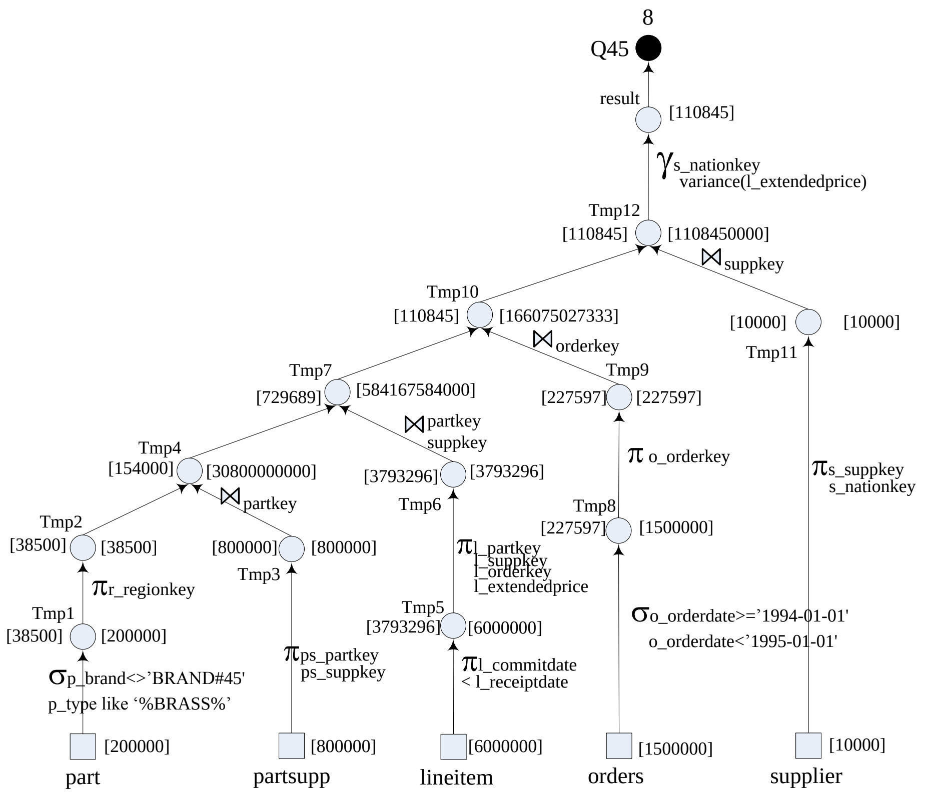


Figure F.45 Relational Algebra Query Tree of Query Q45

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;

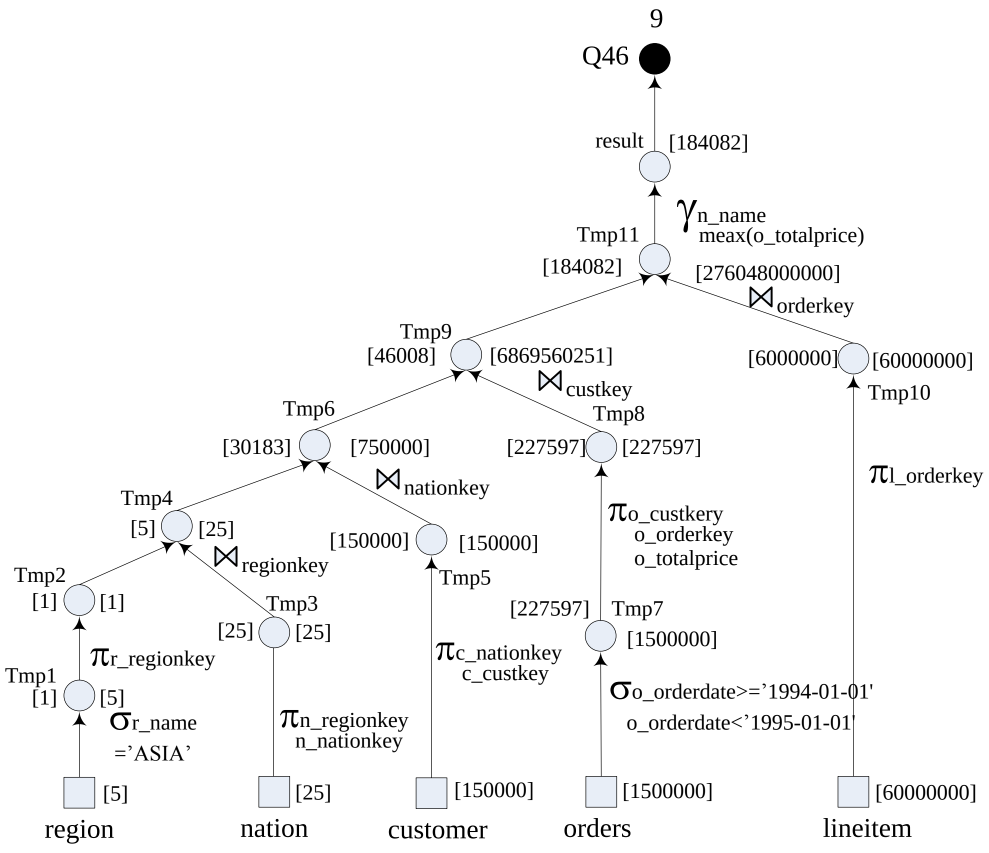


Figure F.46 Relational Algebra Query Tree of Query Q46 (Q2 of 2nd set)

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

```
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;
```

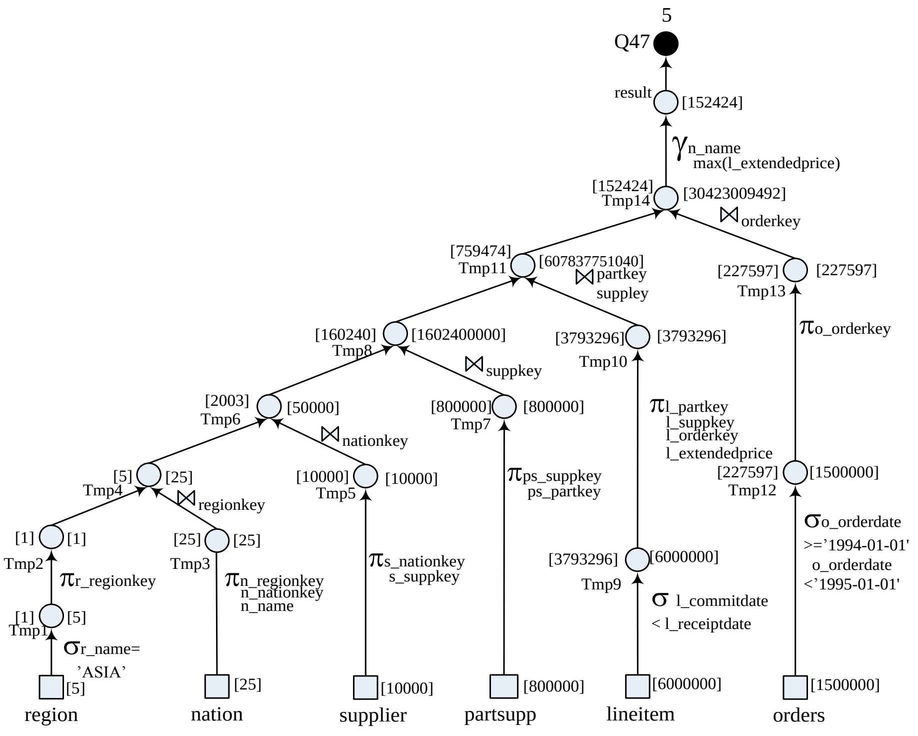


Figure F.47 Relational Algebra Query Tree of Query Q47

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;

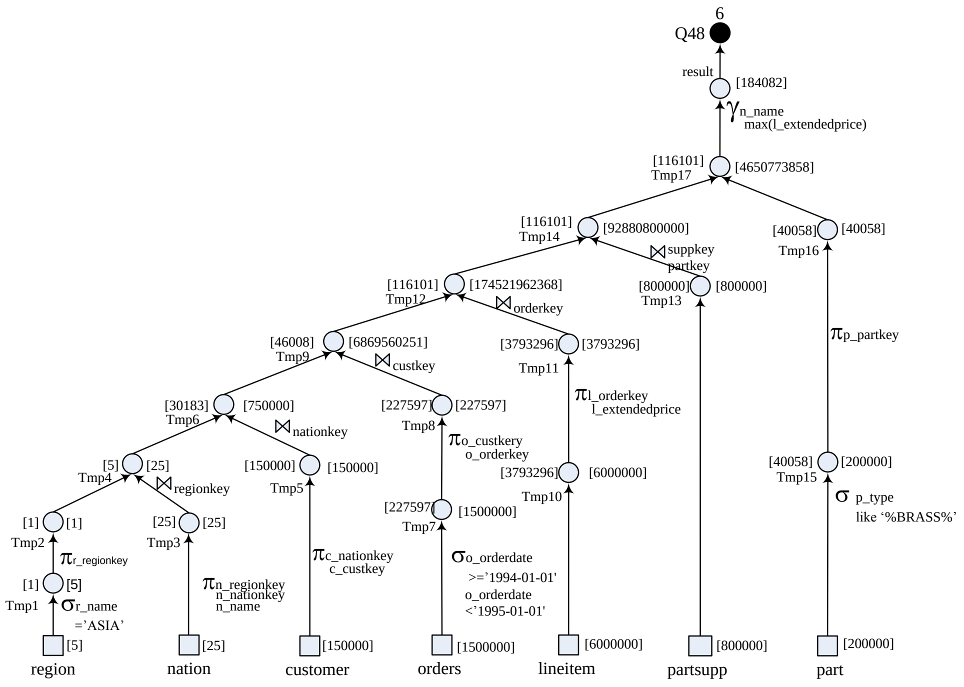


Figure F.48 Relational Algebra Query Tree of Query Q48

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

```
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;
```

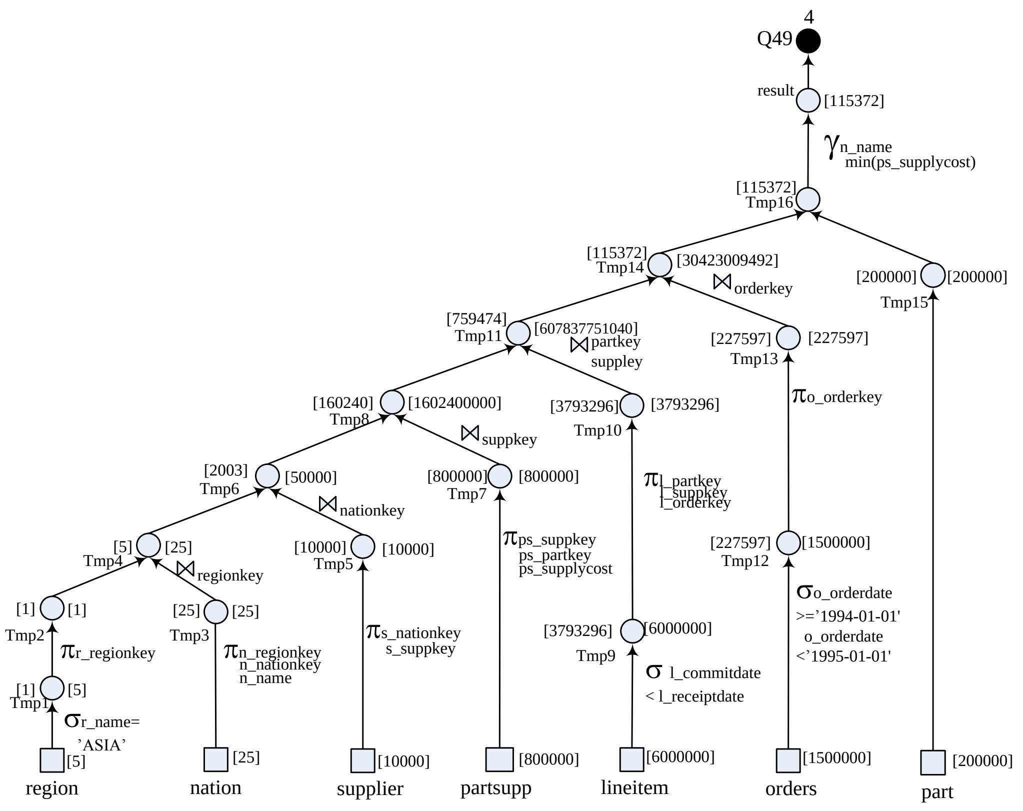


Figure F.49 Relational Algebra Query Tree of Query Q49

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

```
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;
```

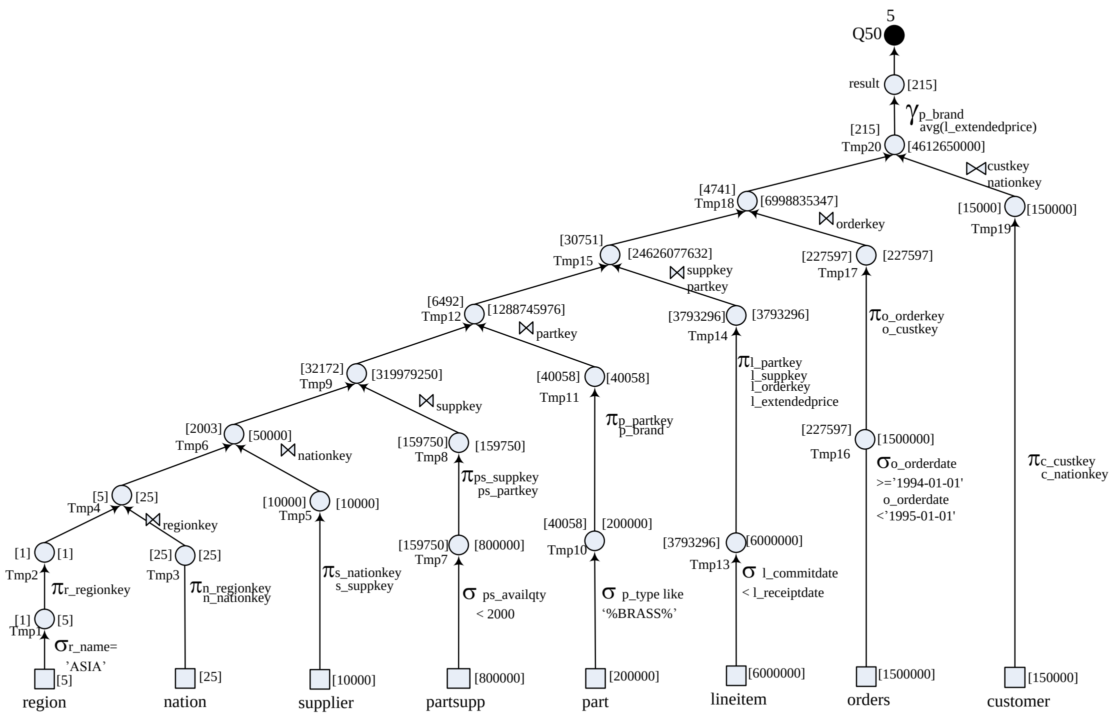


Figure F.50 Relational Algebra Query Tree of Query Q50

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 of 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

Query	**Query Access Frequency(f_q)**	**Query Cost**	**f_q * Query Cost**
Q7	5	159,750	798,750
Q10	7	758,746	5,311,222
Q21	6	758,746	4,552,476
Q27	9	31,864	286,776
Q33	6	759,474	4,556,844
Q38	5	160240	801,200
Q42	5	152424	762,120

Table F.2 The Query Processing Cost of the MVPPs for the First Query Set

Query	**1st MVPP**	**2nd MVPP**	**3rd MVPP**	**4th MVPP**	**5th MVPP**	**6th MVPP (cheapest)**	**7th MVPP (cheapest)**
Q7	7,996,348,750	7,996,348,750	7,996,348,750	7,996,348,750	40,008,848,750	7,996,348,750	7,996,348,750
Q10	5,312,722,822,470	5,312,722,822,470	5,312,722,822,470	5,312,722,822,470	5,312,722,822,470	4,479,058,896,998	4,479,058,896,998
Q21	4,599,282,686,784	4,599,282,686,784	4,599,282,686,784	4,599,282,686,784	4,599,282,686,784	3,884,718,160,284	3,884,718,160,284
Q27	71,996,435,901*	71,996,435,901*	71,996,435,901*	71,996,435,901*	129,618,935,901*	291,303,524,277	291,303,524,277
Q33	3,657,033,383,196	3,657,033,383,196	3,657,033,383,196	3,657,033,383,196	3,695,118,683,196	3,656,709,383,196	3,656,709,383,196
Q38	168,258,101,480	168,258,101,480	168,258,101,480	168,258,101,480	840,109,851,355	168,258,101,480	168,258,101,480
Q42	311,658,701,696	311,658,701,696	311,658,701,696	311,658,701,696	1,279,748,732,616	311,550,701,696	311,550,701,696
Total	**14,128,948,480,277**	**14,128,948,480,277**	**14,128,948,480,277**	**14,128,948,480,277**	**15,896,610,561,072**	**12,799,595,116,681**	**12,799,595,116,681**

Note: * query processing cost of nth MVPP less than the cheapest MVPP

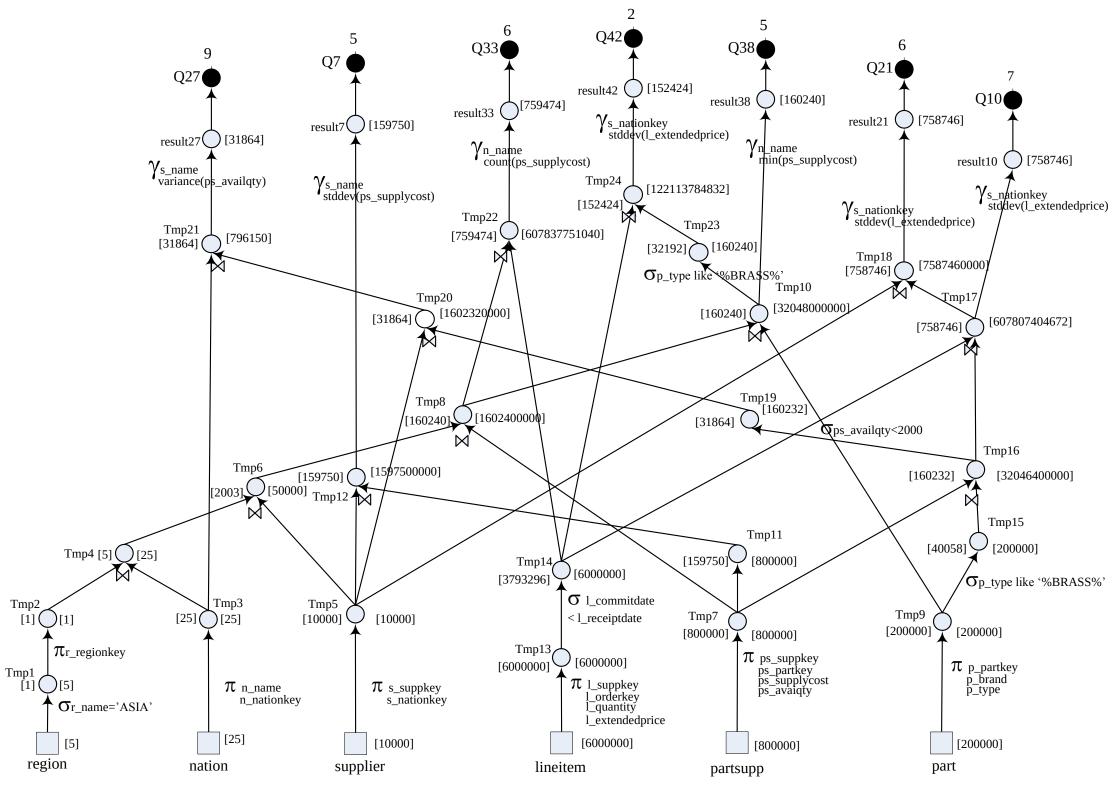


Figure F.51 The Cheapest MVPP of the First Query Set in Static Phase

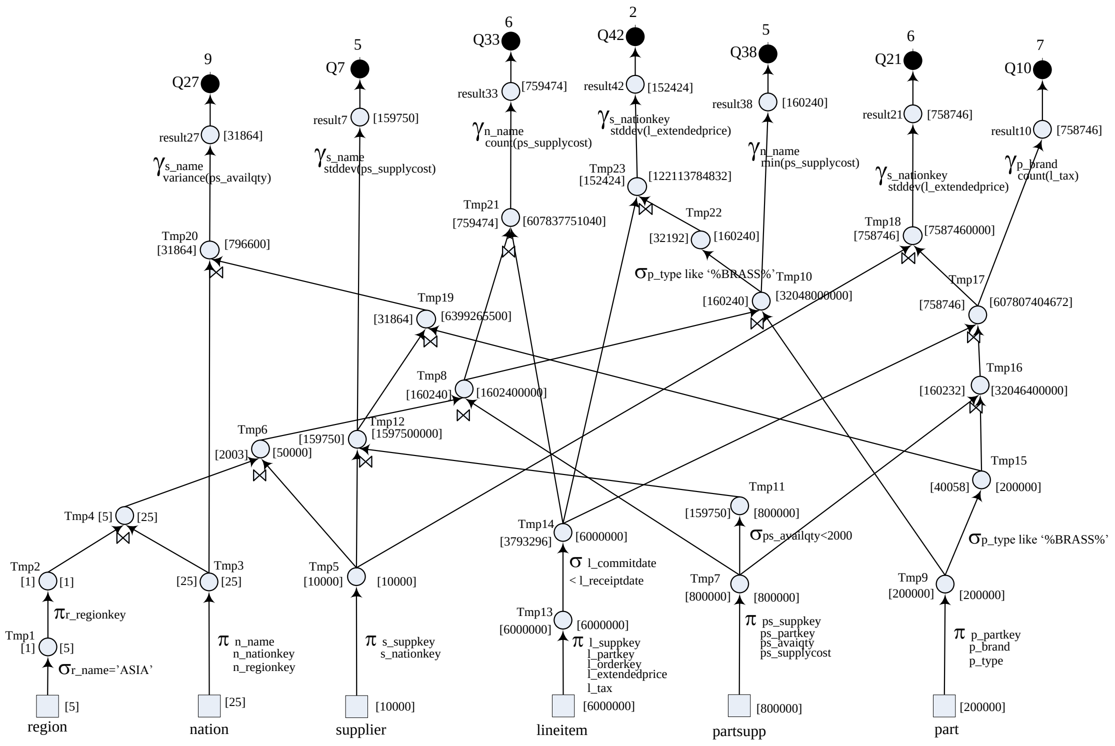


Figure F.52 The Re-Optimized MVPP of the First Query Set in Static Phase

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.

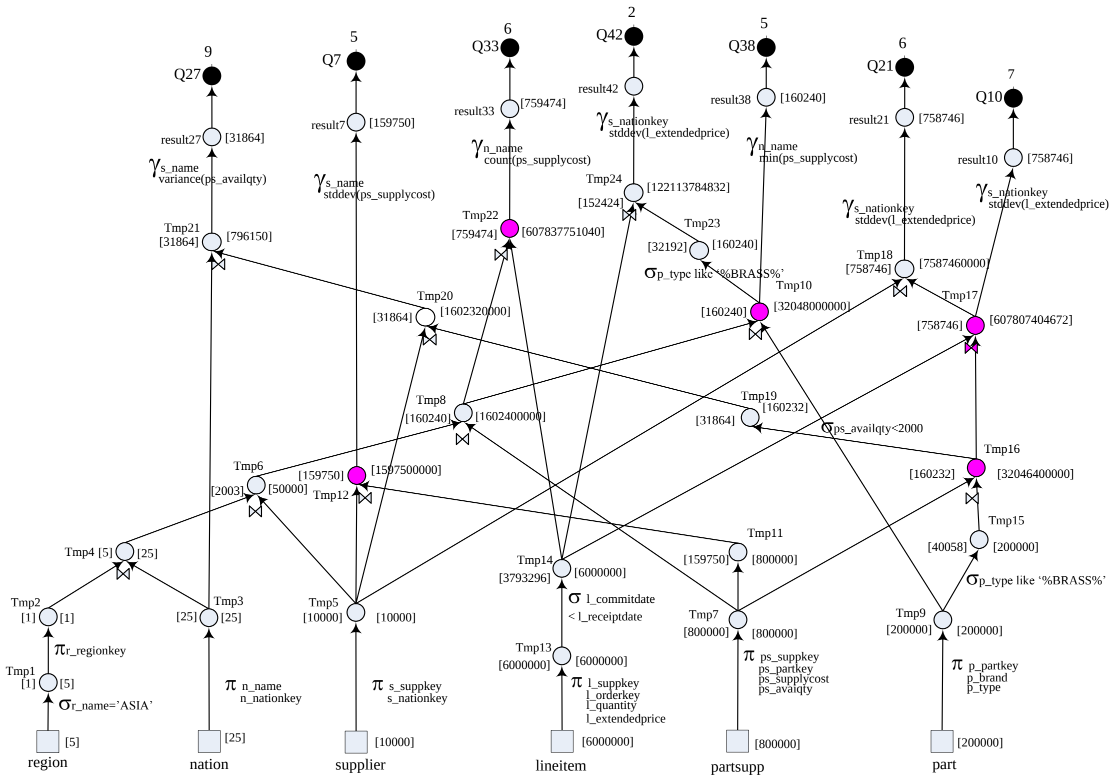


Figure F.53 (a) The Cheapest MVPP of the First Query Set Selected by Deterministic

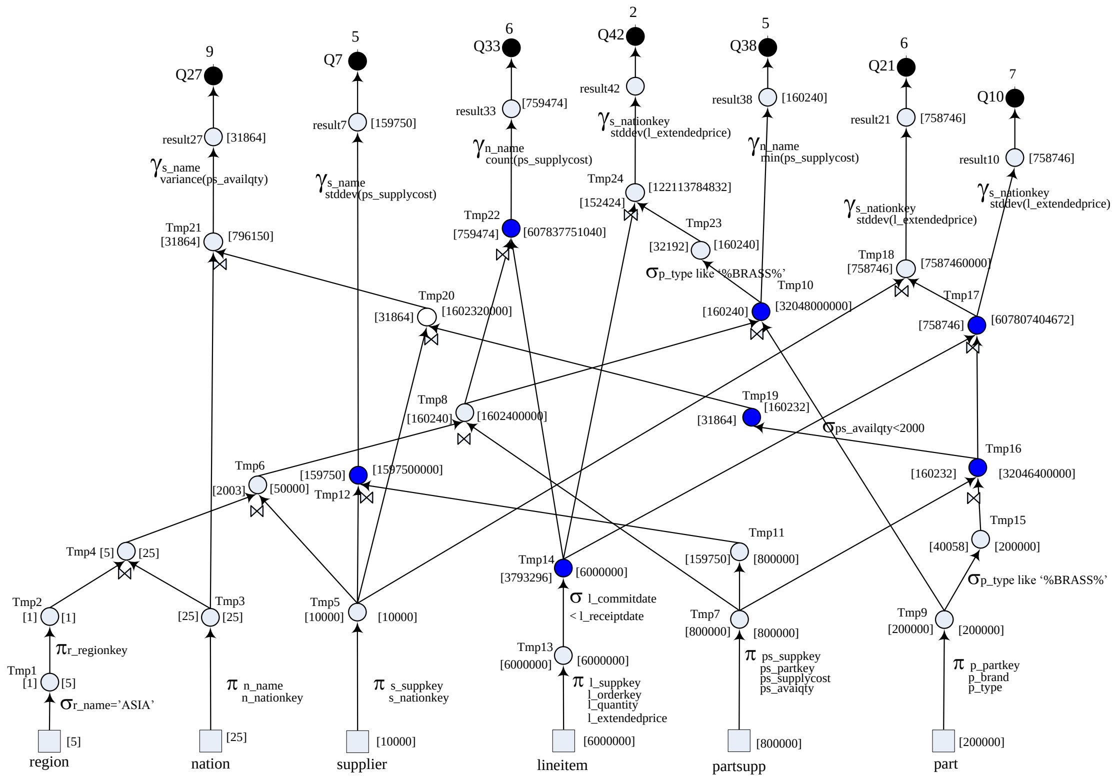


Figure F.53 (b) The Cheapest MVPP of the First Query Set Selected by 2PO

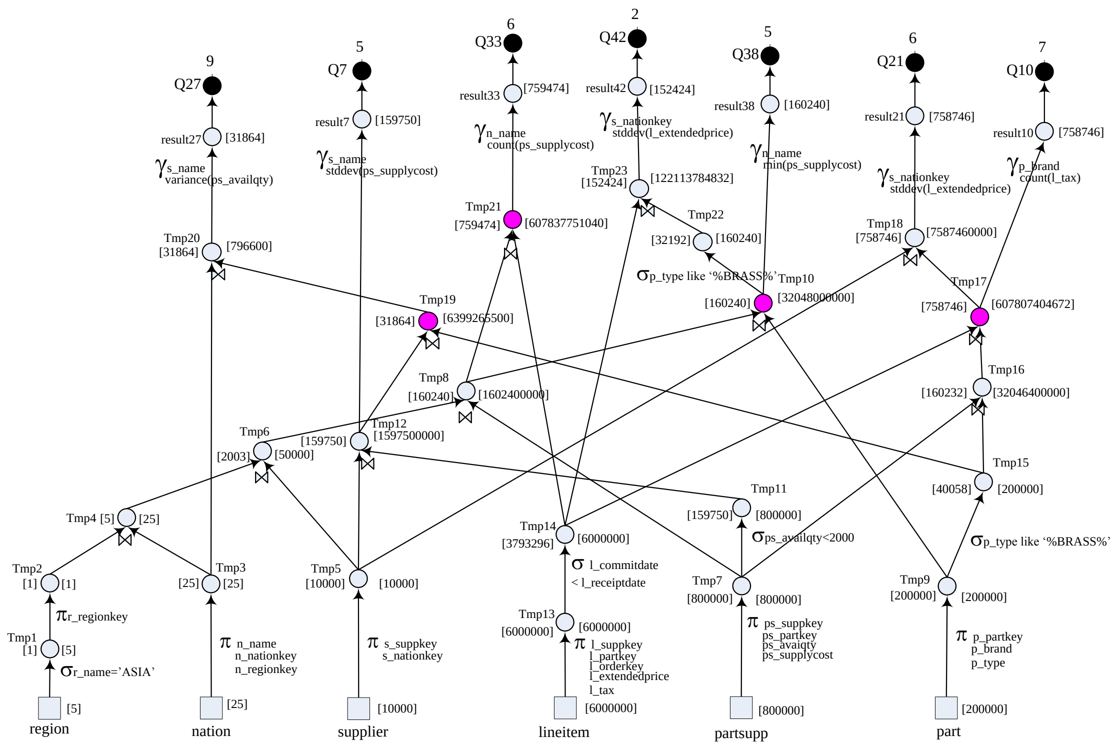


Figure F.54 (a) The Re-Optimized MVPP of the First Query Set by Deterministic

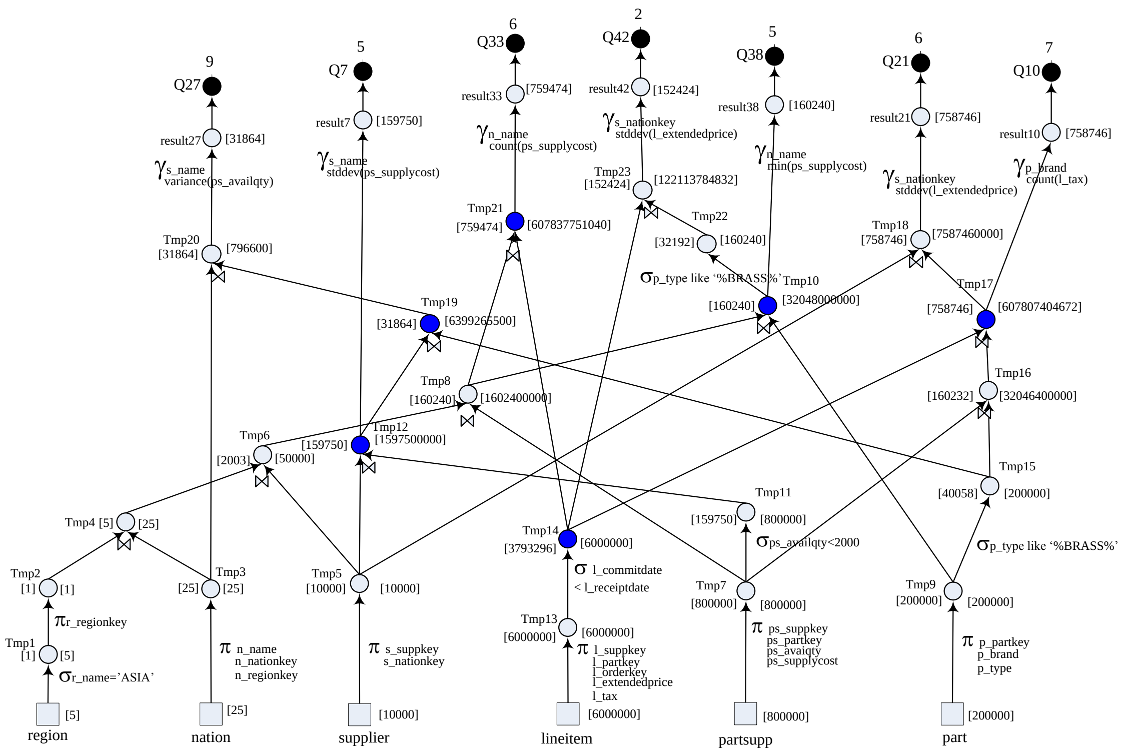


Figure F.54 (b) The Re-Optimized MVPP of the First Query Set by 2PO

Table F.3 The Query Processing Cost, Maintenance Cost and Total Cost of the Cheapest MVPP of the First Query Set

	Cost of Query Processing	**Cost of Maintenance**	**Total Cost**
All-virtual view	12,799,595,116,681	0	12,799,595,116,681
All-materialized view	16,612,116	8,792,272,746,660	8,792,289,358,776
2PO	292,686,142,972	5,202,416,789,776	5,495,102,932,748
Deterministic	581,114,524,945	5,138,321,589,776	5,719,436,114,721

Table F.4 The Query Processing Cost, Maintenance Cost and Total Cost of the Re-Optimized MVPP of the First Query Set

	Cost of Query Processing	**Cost of Maintenance**	**Total Cost**
All-virtual view	12,581,422,028,305	0	12,581,422,028,305
All-materialized view	16,612,116	8,694,762,487,060	8,694,779,099,176
2PO	289,800,706,225	5,138,333,589,776	5,428,134,296,001
Deterministic	297,811,870,883	5,135,123,369,776	5,432,935,240,659

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.

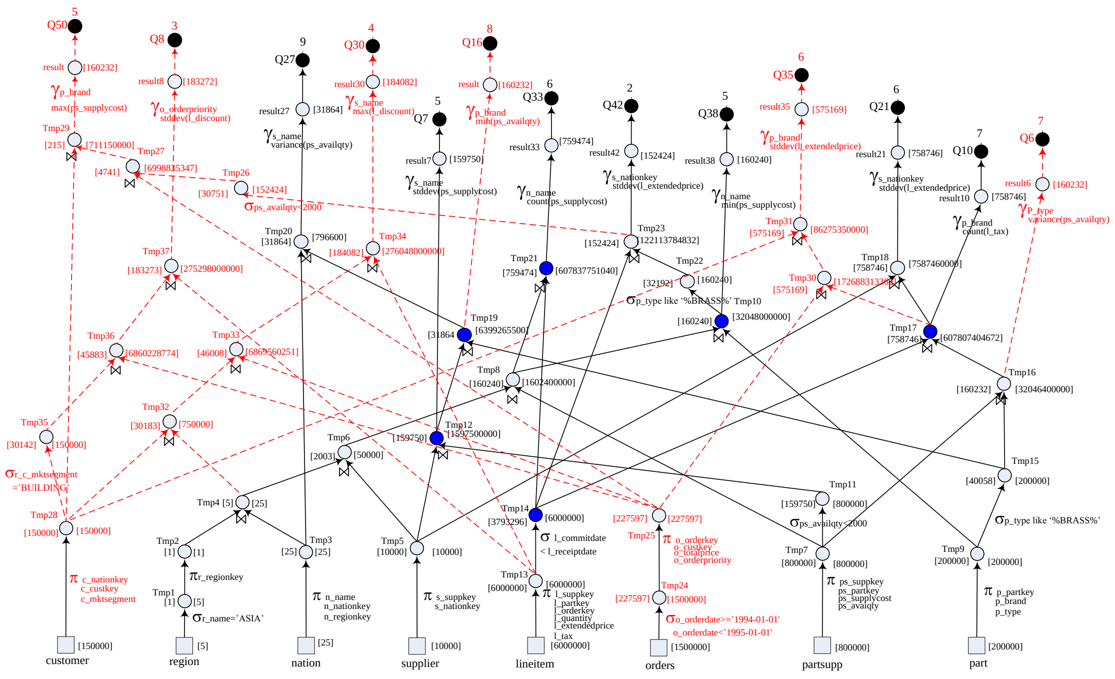


Figure F.55 The Optimal Dynamic MVPP of the First Query Set

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

New Queries	Existing Nodes
Q6	Tmp7, 9, 15, 16
Q8	Tmp13
Q16	Tmp5, 7, 11, 12, 15, 19
Q30	Tmp1, 2, 3, 4, 13
Q35	Tmp7, 9, 13, 14, 15, 16, 17
Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 22, 23

Table F.6 The Weight of the Existing Node for Constructing New Queries

Existing Node	Weight (*w (v)*)
Tmp1	60
Tmp2	7
Tmp3	525
Tmp4	213
Tmp5	320,000
Tmp6	469,832
Tmp7	31,200,000
Tmp8	14,418,159,776
Tmp9	5,600,000
Tmp10	56,078,699,720
Tmp11	9,600,000
Tmp12	19,166,780,000
Tmp13	108,000,000
Tmp14	102,000,000
Tmp15	4,000,000
Tmp16	576,832,800,000
Tmp17	5,981,895,246,720
Tmp19	33,597,063,000
Tmp22	-168,257,781,000
Tmp23	-620,262,645,936

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

Directly Affected Node	Ancestor Node	Weight of Ancestor Node
Tmp8	Tmp21	599,761,450,760
Tmp10	Tmp22	-168,257,781,000
	Tmp23	-620,262,645,936

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.

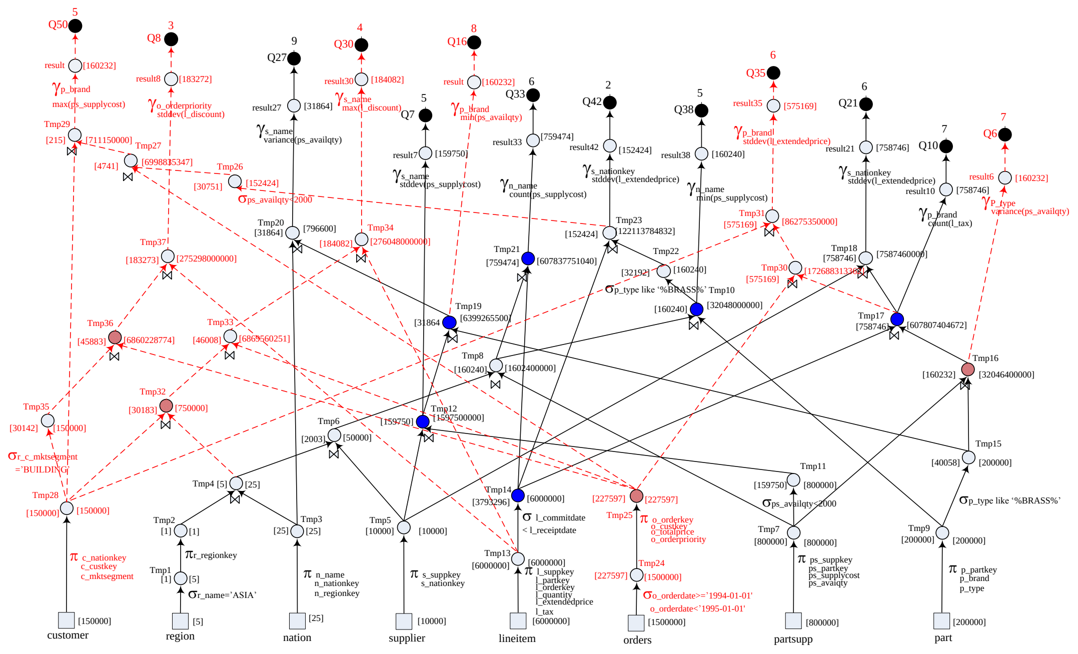


Figure F.56 The Materialized View Selected by 2PO for the Dynamic MVPP of the First Query Set

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.

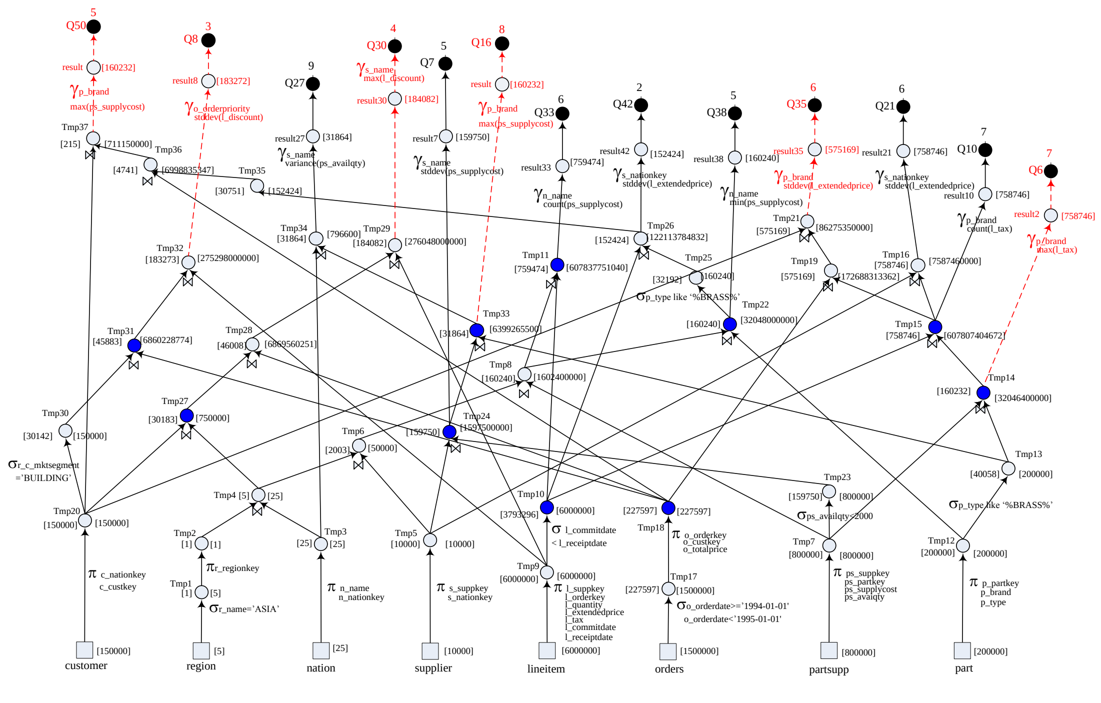


Figure F.57 The Re-Optimized MVPP by Static Approach for All Queries of the First Query Set

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

Approach	**Number of Nodes**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	37	4,709,318,228,610	5,240,142,056,783	9,949,460,285,393
Dynamic	28	4,709,318,228,610	5,240,142,056,783	9,949,460,285,393

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

Query	**Query Access Frequency(f_q)**	**Query Cost**	**f_q * Query Cost**
Q4	6	3,793,296	22,759,776
Q12	5	575,169	2,875,845
Q23	3	575,169	1,725,507
Q33	6	759,474	4,556,844
Q36	8	575,169	4,601,352
Q40	4	6,492	25,968
Q41	3	4,558	13,674

Table F.10 The Query Processing Cost of The MVPPs for the Second Query Set

Query	1st MVPP	2nd MVPP (cheapest)	3rd MVPP (cheapest)	4th MVPP	5th MVPP	6th MVPP	7th MVPP
Q4	18,207,907, 119,552	18,207,907 ,119,552	18,207,907 ,119,552	18,207,907 ,119,552	18,207,907 ,119,552	18,207,907 ,119,552	18,207,907 ,119,552
Q12	19,489,962, 428,870	6,617,454, 428,870	6,617,454, 428,870	6,617,454, 428,870	6,617,454, 428,870	19,489,962 ,428,870	19,489,962 ,428,870
Q23	11,952,803, 957,322	4,229,299, 157,322	4,229,299, 157,322	4,229,299, 157,322	4,229,299, 157,322	11,952,803 ,957,322	11,952,803 ,957,322
Q33	18,253,477, 108,284	3,656,709, 383,196	3,656,709, 383,196	18,253,477 ,108,284	18,253,477 ,108,284	3,656,709, 383,196	3,656,709, 383,196
Q36	32,150,225, 486,192	11,554,212 ,686,192	11,554,212 ,686,192	11,554,212 ,686,192	11,554,212 ,686,192	32,150,225 ,486,192	32,150,225 ,486,192
Q40	6,443,239,2 04*	11,570,291 ,056	11,570,291 ,056	6,443,239, 204	6,443,239, 204	11,570,291 ,056	11,570,291 ,056
Q41	2,645,437,1 66,178	2,645,437, 166,178	2,645,437, 166,178	2,645,437, 166,178	2,645,437, 166,178	593,308,53 4,119*	593,308,53 4,119*
Total	**102,706,25 6,505,602**	**46,922,590 ,232,366**	**46,922,590 ,232,366**	**61,514,230 ,905,602**	**61,514,230 ,905,602**	**86,062,487 ,200,307**	**86,062,487 ,200,307**

Note: * query processing cost of nth MVPP less than the cheapest MVPP

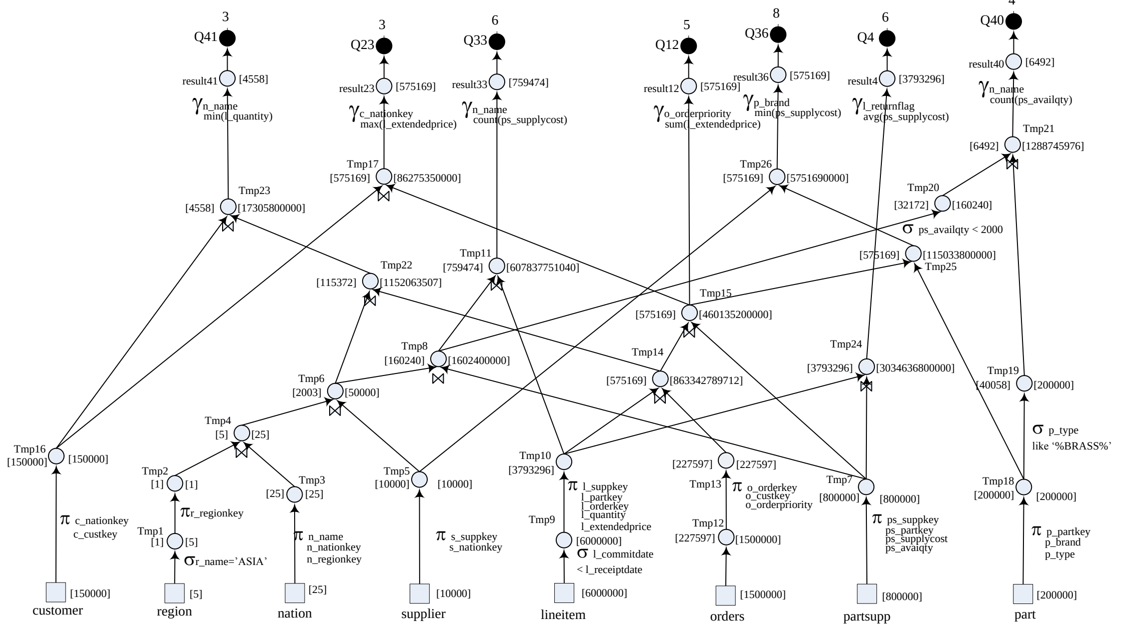

3
Q41
result41
[4558]
γn_name min(l_quantity)
3
Q23
result23
[575169]
γc_nationkey max(l_extendedprice)
6
Q33
result33
[759474]
γn_name count(ps_supplycost)
5
Q12
result12
[575169]
γo_orderpriority sum(l_extendedprice)
8
Q36
result36
[575169]
γp_brand min(ps_supplycost)
6
Q4
result4
[3793296]
γl_returnflag avg(ps_supplycost)
4
Q40
result40
[6492]
γn_name count(ps_availqty)
Tmp21 [6492] [1288745976]
Tmp17 [575169] [86275350000]
Tmp26 [575169] [5751690000]
Tmp23 [4558] [17305800000]
Tmp20 [32172] [160240]
σ ps_availqty < 2000
[575169] [115033800000] Tmp25
Tmp22 [115372] [1152063507]
Tmp11 [759474] [607837751040]
Tmp15 [575169] [460135200000]
Tmp8 [160240] [1602400000]
Tmp14 [575169] [863342789712]
Tmp24 [3793296] [3034636800000]
Tmp6 [2003] [50000]
Tmp19 [40058] [200000]
σ p_type like '%BRASS%'
Tmp4 [5] [25]
Tmp16 [150000] [150000]
Tmp10 [3793296]
[227597] [227597]
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp5 [10000] [10000]
Tmp13
Tmp7 [800000] [800000]
Tmp18 [200000] [200000]
π c_nationkey c_custkey
πr_regionkey
π n_name n_nationkey n_regionkey
π s_suppkey s_nationkey
π l_suppkey l_partkey l_orderkey l_quantity l_extendedprice
π o_orderkey o_custkey o_orderpriority
π ps_suppkey ps_partkey ps_supplycost ps_avaiqty
π p_partkey p_brand p_type
Tmp1 [1] [5]
σr_name='ASIA'
Tmp9 [6000000]
σ l_commitdate < l_receiptdate
Tmp12 [227597] [1500000]
[150000] customer
[5] region
[25] nation
[10000] supplier
[6000000] lineitem
[1500000] orders
[800000] partsupp
[200000] part

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} ⋈ 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 ⋈ NATION ⋈ SUPPLIER}, Tmp6, and {LINEITEM ⋈ 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.

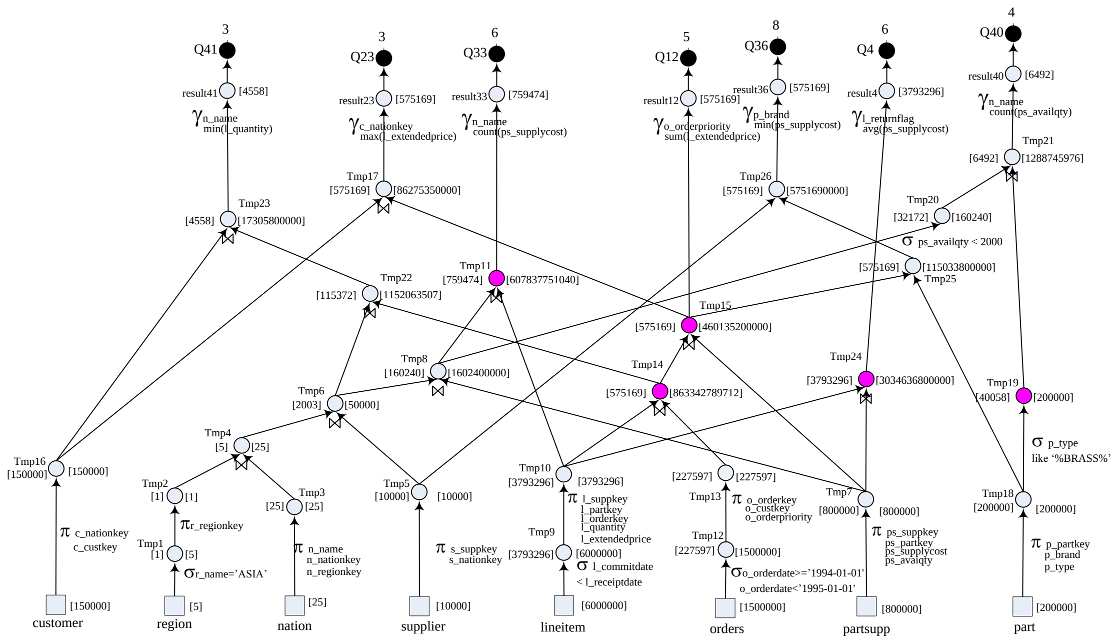


Figure F.59 (a) The Optimal MVPP of the Second Query Set by Deterministic

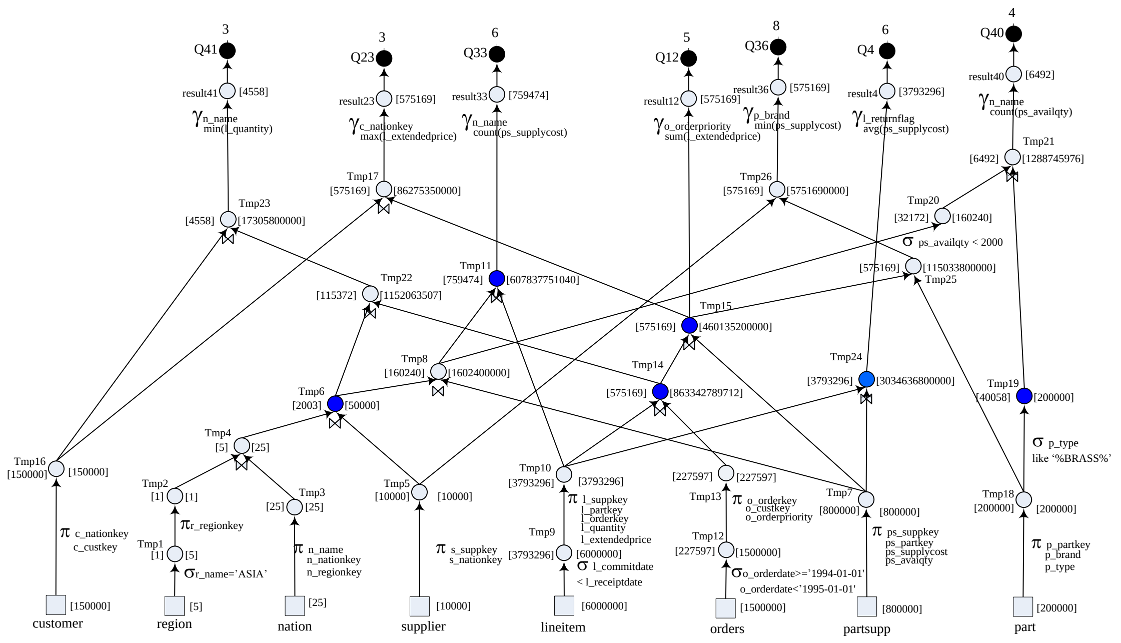


Figure F.59 (b) The Optimal MVPP of the Second Query Set by 2PO

Table F.11 The Query Processing Cost, Maintenance Cost and Total Cost of the Static Phase of the Second Query Set

	Cost of Query Processing	Cost of Maintenance	Total Cost
All-virtual view	34,783,985,485,998	0	34,783,985,485,998
All-materialized view	21,385,782	31,252,800,172,180	31,252,821,557,962
2PO	1,292,099,197,067	14,813,755,422,193	16,105,854,619,260
Deterministic	1,292,099,603,438	14,813,755,242,025	16,105,854,845,463

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.

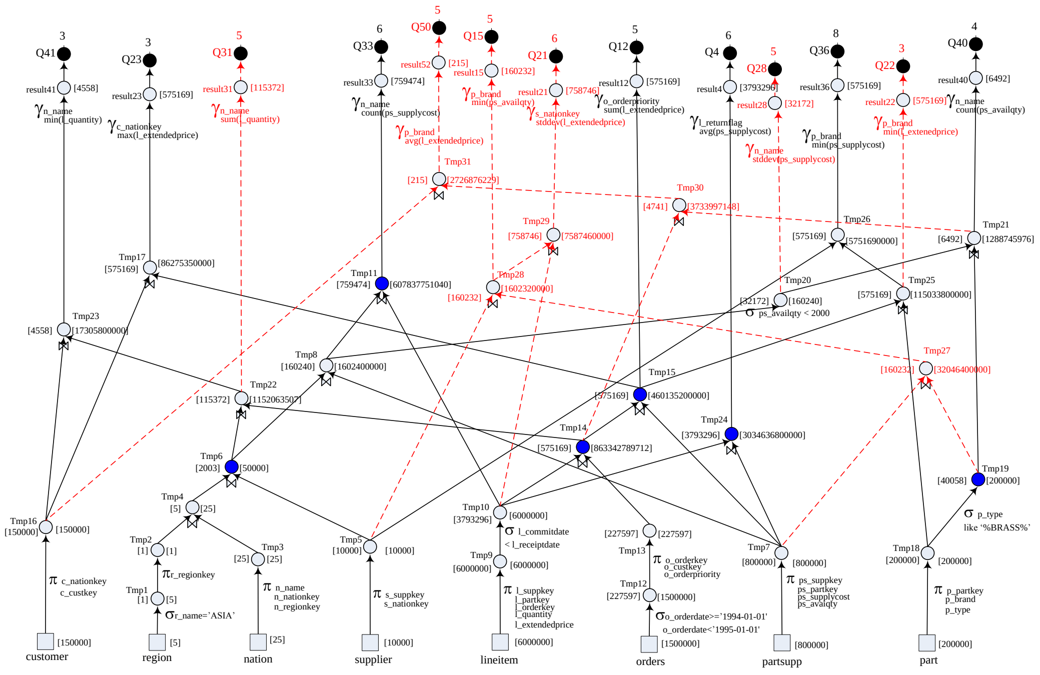


Figure F.60 The Optimal Dynamic MVPP of the Second Query Set

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

New Queries	Existing Nodes
Q15	Tmp5, 7, 9, 10, 18, 19
Q21	Tmp5, 7, 9, 10, 18, 19
Q22	Tmp7, 9, 10, 12, 13, 14, 15, 18, 25
Q28	Tmp1, 2, 3, 4, 5, 6, 7, 8, 20
Q31	Tmp1, 2, 3, 4, 5, 6, 9, 10, 12, 13, 14, 22
Q50	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9, 10,12,13,14,16, 18,19,21

Table F.13 The Weight of the Existing Node for Constructing All New Queries

Existing Node	**Weight (*w (v)*)**
Tmp1	135
Tmp2	22
Tmp3	675
Tmp4	588
Tmp5	400,000
Tmp6	1,219,832
Tmp7	40,800,000
Tmp8	25,634,959,776
Tmp9	270,000,000
Tmp10	164,698,320
Tmp12	46,500,000
Tmp13	5,555,507
Tmp14	25,900,260,649,574
Tmp15	4,772,097,868,185
Tmp16	1,500,000
Tmp18	6,000,000
Tmp19	3,600,000
Tmp20	-6,412,239,024
Tmp21	-2,864,117,576
Tmp22	-4,313,315,662,784
Tmp25	-4,833,826,082,420

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 Tm6, 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

Directly Affected Node	Ancestor Node	Weight of Ancestor Node
Tmp6	Tmp22	-4,313,315,662,784
	Tmp23	-40,375,843,660
Tmp8	Tmp11	599,772,484,280
Tmp10	Tmp24	-21,186,592
Tmp15	Tmp25	-4,833,826,082,420
Tmp19	Tmp21	-2,864,117,576

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.

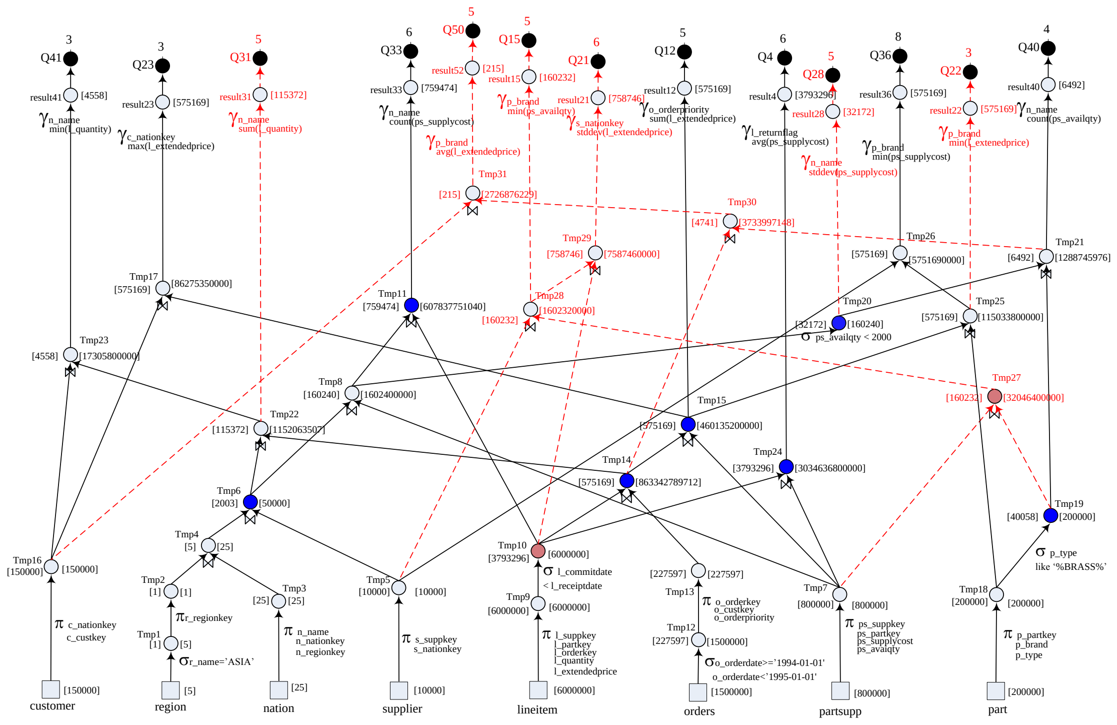


Figure F.61 The Materialized Views Selected by 2PO for the Dynamic MVPP of the Second Query Set

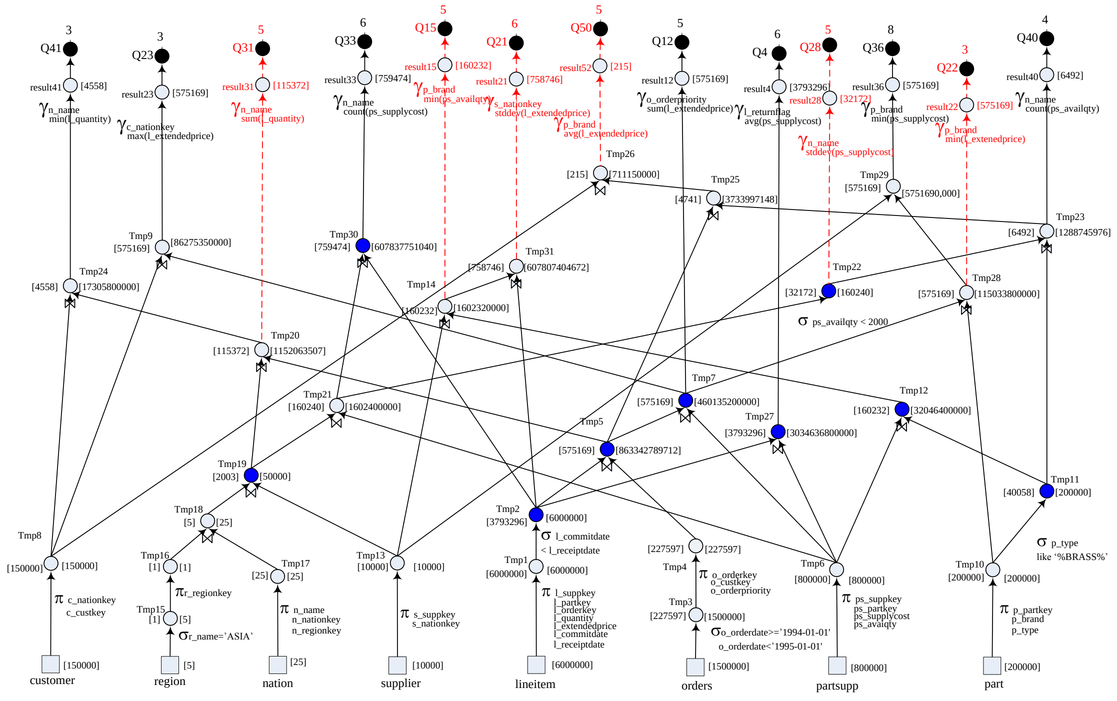


Figure F.62 The Re-Optimized MVPP by Static Approach for All Queries of the Second Query Set

Table F.15 The Comparison of the Result from the Static Approach and the Dynamic Approach of the Second Query Set

Approach	Number of Nodes	Cost of Query Processing	Cost of Maintenance	Total Cost
Static	31	5,329,761,453,158	14,884,283,889,969	20,214,045,343,127
Dynamic	18	5,329,761,453,158	14,884,283,889,969	20,214,045,343,127

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

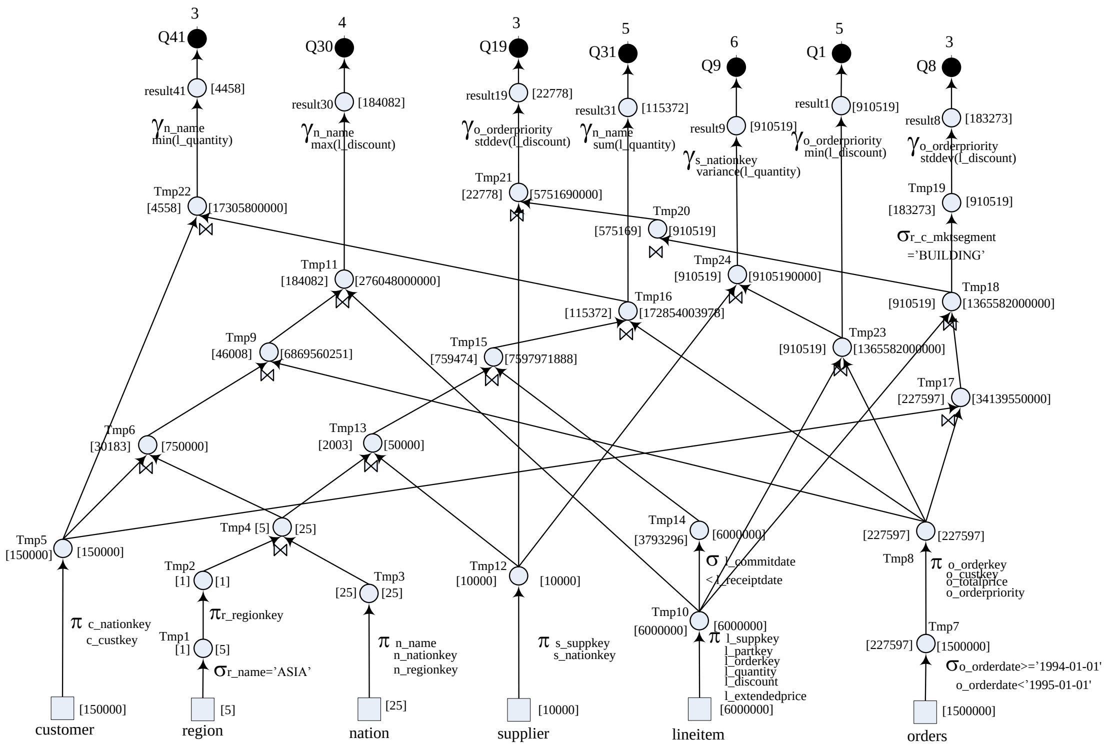
The order of queries according to their frequency of executing the query multiplied with the query cost is shown in Table F.16. Then, the order of queries of the first MVPP is {Q9, Q1, Q30, Q31, Q8, Q19 and Q41} and the last order is {Q41, Q9, Q1, Q30, Q31, Q8, and Q19}. The query processing costs of 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

Query	**Query Access Frequency(f_q)**	**Query Cost**	**f_q * Query Cost**
Q1	5	910,519	4,552,595
Q8	3	183,273	549,819
Q9	6	910,519	5,463,114
Q19	3	22,278	66,834
Q30	4	184,082	736,328
Q31	5	115,372	576,860
Q41	3	4,558	13,674

Table F.17 The Query Processing Cost of the Third Query Set

Query	1st MVPP	2nd MVPP	3rd MVPP (cheapest)	4th MVPP	5th MVPP	6th MVPP	7th MVPP
Q1	6,827,953 ,190,580	6,827,953, 190,580	6,827,953, 190,580	6,827,953, 190,580	6,827,953, 190,580	6,827,953, 190,580	6,827,953, 190,580
Q8	4,506,506 ,464,167	4,506,506, 464,167	4,199,188, 832,610	4,199,191, 564,167	4,199,191, 564,167	4,506,506, 464,167	4,506,506, 464,167
Q9	8,248,175 ,028,696	8,248,175, 028,696	8,248,175, 028,696	8,248,175, 028,696	8,248,175, 028,696	8,248,175, 028,696	8,248,175, 028,696
Q19	4,372,853 ,582,682	4,523,761, 082,682	4,216,446, 182,682	4,216,446, 182,682	4,216,446, 182,682	4,382,914, 082,682	4,372,853, 582,682
Q30	5,572,292 ,026,848	5,572,292, 026,848	1,131,705, 487,944	6,008,675, 289,016	6,008,675, 289,016	5,572,292, 026,848	5,572,292, 026,848
Q31	6,873,494 ,146,945	6,833,714, 385,255	902,329,39 4,455	902,329,39 4,455	6,873,494, 146,945	6,873,494, 146,945	902,329,39 4,455
Q41	4,176,014 ,005,725	4,152,146, 148,711	593,315,15 4,231	593,315,15 4,231	4,216,446, 469,860	4,382,914, 369,860	593,315,15 4,231
Total	**40,577,28 8,445,643**	**40,664,548 ,326,939**	**26,119,113 ,241,198**	**30,996,085 ,803,827**	**40,590,381 ,871,946**	**40,794,249 ,309,778**	**31,023,424 ,841,659**



3
Q41
4
Q30
3
Q19
5
Q31
6
Q9
5
Q1
3
Q8
result41 [4458]
γ_{n_name} min(l_quantity)
result30 [184082]
γ_{n_name} max(l_discount)
result19 [22778]
$\gamma_{o\_orderpriority}$ stddev(l_discount)
result31 [115372]
γ_{n_name} sum(l_quantity)
result9 [910519]
$\gamma_{s\_nationkey}$ variance(l_quantity)
result1 [910519]
$\gamma_{o\_orderpriority}$ min(l_discount)
result8 [183273]
$\gamma_{o\_orderpriority}$ stddev(l_discount)
Tmp22 [4558] [17305800000]
Tmp21 [22778] [5751690000]
Tmp20 [575169] [910519]
Tmp19 [183273] [910519]
$\sigma_{r\_c\_mktsegment}$ ='BUILDING'
Tmp11 [184082] [276048000000]
Tmp24 [910519] [9105190000]
Tmp16 [115372] [172854003978]
Tmp18 [910519] [1365582000000]
Tmp9 [46008] [6869560251]
Tmp15 [759474] [7597971888]
Tmp23 [910519] [1365582000000]
Tmp17 [227597] [34139550000]
Tmp6 [30183] [750000]
Tmp13 [2003] [50000]
Tmp5 [150000] [150000]
Tmp4 [5] [25]
Tmp14 [3793296] [6000000]
σ l_commitdate <l_receiptdate
[227597] [227597]
Tmp8
π o_orderkey o_custkey o_totalprice o_orderpriority
Tmp2 [1] [1]
Tmp3 [25] [25]
Tmp12 [10000] [10000]
$\pi_{r\_regionkey}$
π c_nationkey c_custkey
Tmp1 [1] [5]
$\sigma_{r\_name='ASIA'}$
π n_name n_nationkey n_regionkey
π s_suppkey s_nationkey
Tmp10 [6000000] [6000000]
π l_suppkey l_partkey l_orderkey l_quantity l_discount l_extendedprice [6000000]
Tmp7 [227597] [1500000]
$\sigma_{o\_orderdate>='1994-01-01'}$ o_orderdate<'1995-01-01'
[150000] customer
[5] region
[25] nation
[10000] supplier
lineitem
[1500000] orders

Figure F.63 The cheapest MVPP of the Third Query Set of the Static Phase

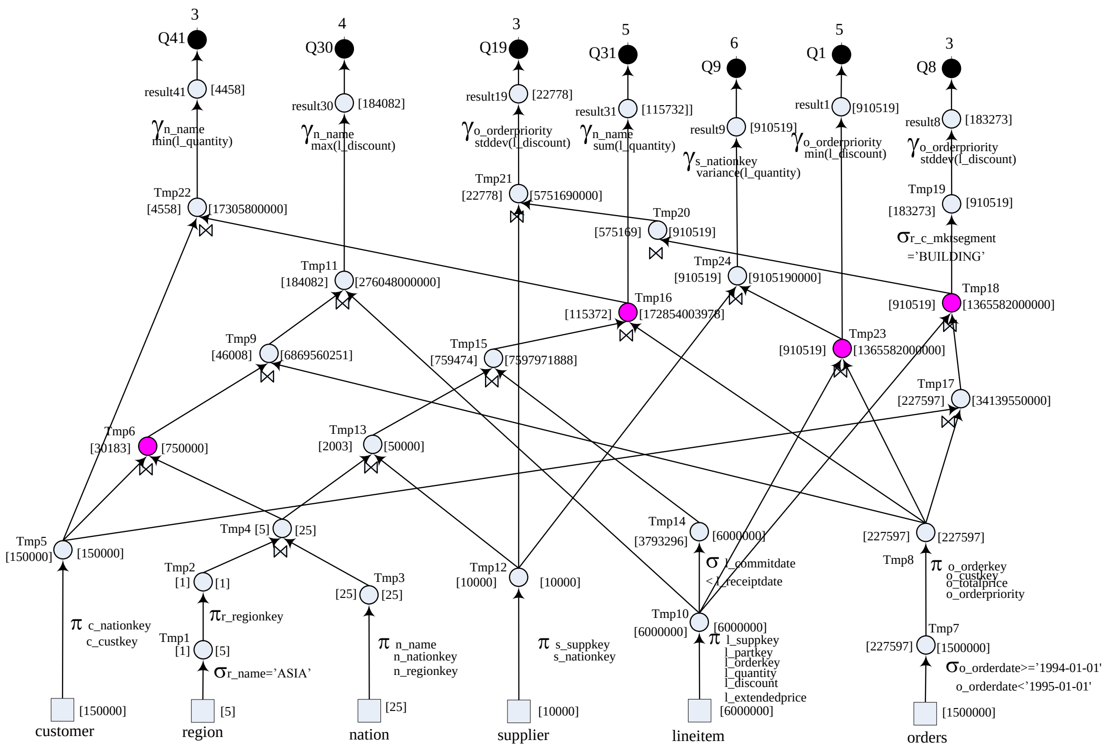


Figure F.64 (a) The optimal MVPP of the Third Query by Deterministic

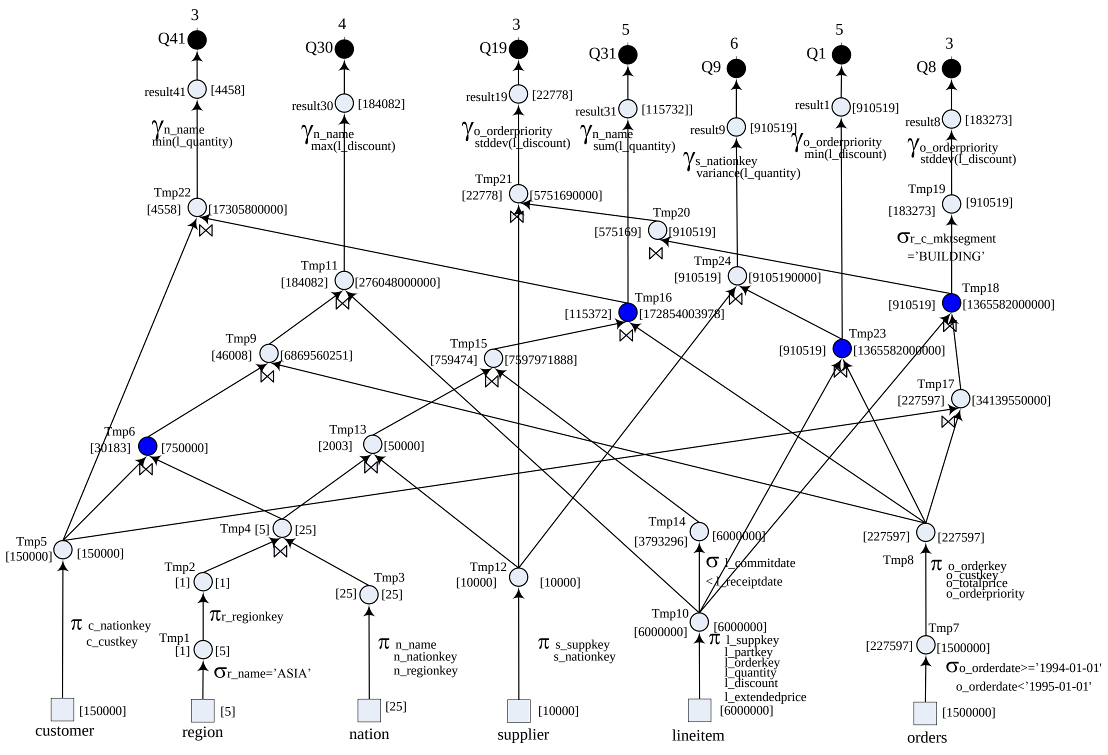


Figure F.64 (b) The optimal MVPP of the Third Query by 2PO

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 F.18 The Query Processing Cost, Maintenance Cost and Total Cost of the Static Phase

	Cost of Query Processing	**Cost of Maintenance**	**Total Cost**
All-virtual view	26,119,113,241,198	0	26,119,113,241,198
All-materialized view	11,960,724	16,649,310,570,792	16,649,322,531,516
2PO	306,035,672,471	7,832,700,983,345	8,138,736,655,816
Deterministic	306,035,672,471	7,832,699,255,748	8,138,736,655,816

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.

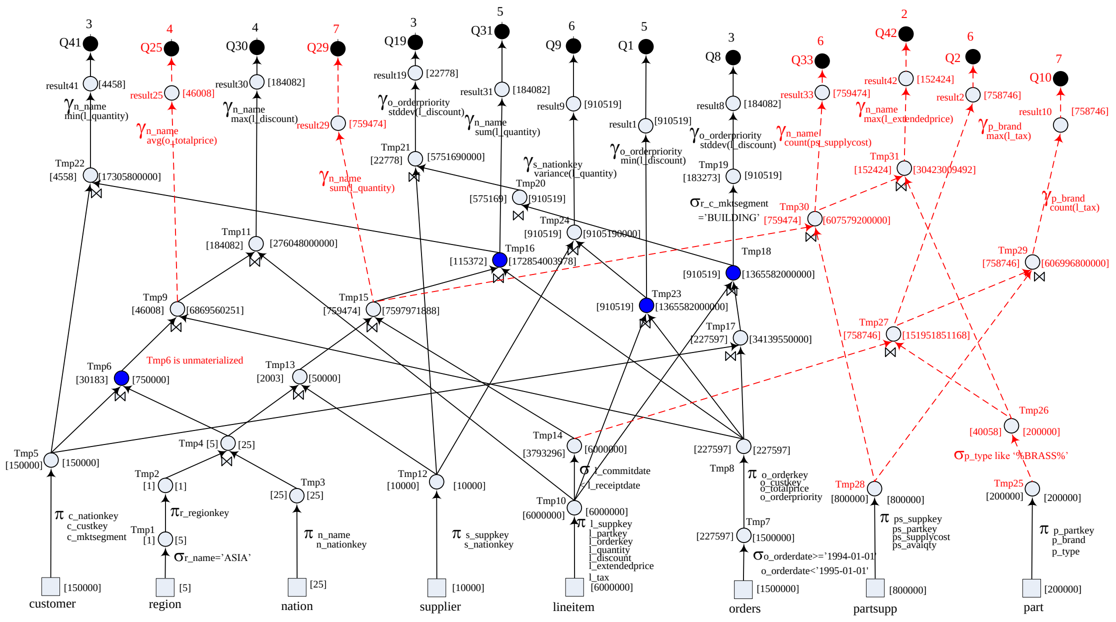


Figure F.65 The Optimal Dynamic MVPP of the Third Query Set

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

New Queries	Existing Nodes
Q2	Tmp10, 14
Q10	Tmp10, 14
Q25	Tmp1, 2, 3, 4, 5, 6, 7, 8, 9
Q29	Tmp1, 2, 3, 4, 10, 12, 13, 14, 15
Q33	Tmp1, 2, 3, 4, 10, 12, 13, 14, 15
Q42	Tmp1, 2, 3, 4, 10, 12, 13, 14, 15

Table F.20 The Weight of the Existing Node for Constructing All New Queries

Existing Node	Weight (*w (v)*)
Tmp1	55
Tmp2	6
Tmp3	275
Tmp4	188
Tmp5	1,050,000
Tmp6	3,299,832
Tmp7	42,000,000
Tmp8	4,872,716
Tmp9	27,467,730,392
Tmp10	168,000,000
Tmp12	160,000
Tmp13	1,269,832
Tmp14	36,000,000
Tmp15	30,343,647,328

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

Directly Affected Node	Ancestor Node	Weight of Ancestor Node
Tmp8	Tmp17	136,554,444,806
	Tmp18	3,994,303,717,209
	Tmp19	-4,199,188,282,791
Tmp8	Tmp20	-5,598,918,620,907
	Tmp21	-5,604,673,082,464
Tmp9	Tmp11	-310,408,939,520
Tmp15	Tmp16	480,503,214,229
	Tmp22	-1,134,712,881,114

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.

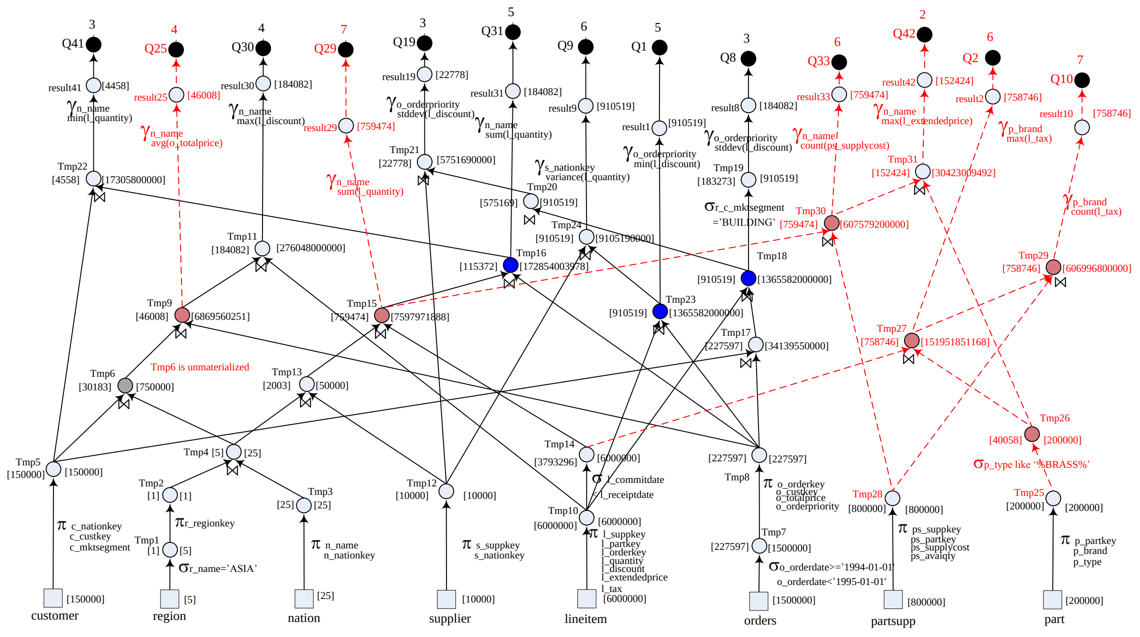
The result of affected nodes show as follows.

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



● represents un-materialized node in dynamic phase

Figure F.66 The Dynamic MVPP of the Third Query Set

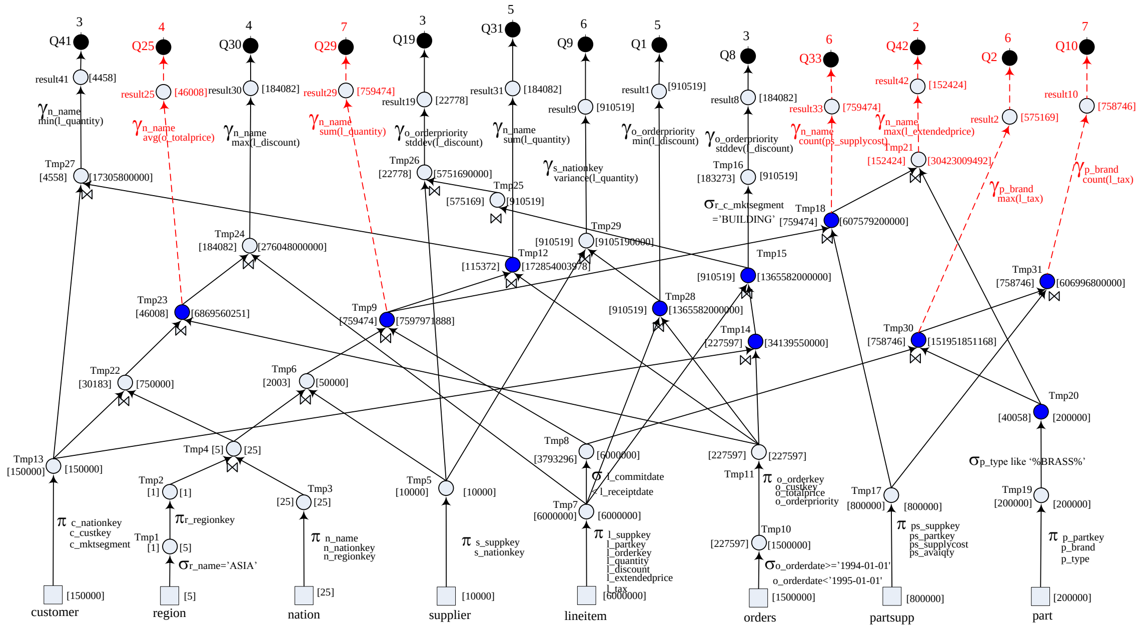


Figure F.67 The Re-Optimized MVPP by Static Approach for All Queries of the Third Query Set

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

Approach	**Number of Nodes**	**Cost of Query Processing**	**Cost of Maintenance**	**Total Cost**
Static	31	184,748,883,173	13,766,731,003,390	13,951,479,886,563
Dynamic	17	184,748,883,173	13,766,731,003,390	13,951,479,886,563

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	B. E. (Computer Engineer) from King Mongkut’s Institute of Technology Thonburi, Bangkok, Thailand in 1994. M.S. (Electronic Business) from King Mongkut's University of Technology Thonburi, Bangkok, Thailand in 2006.
PUBLICATION	1. Boontita Suchyukorn and Raweewan Auepanwiriyakul, “Re-Optimization MVPP using Common Subexpression for Materialized View Selection”, **World Academy of Science Engineering and Technology**, issue 79, July 2013, Pp.1177-1185. 2. Boontita Suchyukorn and Raweewan Auepanwiriyakul, “Dynamic Materialized View Selection Using 2PO Based on Re-Optimized Multiple View Processing Plan”, **International Journal of Advancements in Computing Technology (IJACT)**, Vol.5(14), October 2013, Pp. 150-167