Workshop on Databases and Software Engineering

A Partial View Materialization Approach in Data Warehouse Environment

Huei-Huang Chen
hhchen@cse.ttu.edu.tw

Ting-Yi Chen
cti@it01.cse.ttu.edu.tw

Department of Computer Science and Engineering, Tatung University
No. 40, Chung-Shan N. Rd., Sec. 3, Taipei, 104, Taiwan, R.O.C.
Tel: 886-2-25925252 ext.3295 Fax: 886-2-25925252 ext.2288

Abstract

In a data warehouse environment (DWE), users access very large databases to carry out strategic analysis for maintaining business competitiveness by executing OLAP queries. Therefore, efficient query processing becomes a critical issue. The storage space of the dedicated OLAP server is smaller than the total size of all data cubes usually. That is, only parts of the data cubes can be stored. Materialized views selection is the most important decisions in building a Data Warehouse.

At the heart of all OLAP or multidimensional data analysis applications is the ability to simultaneously aggregate across many sets of dimensions. Computing multidimensional aggregates is a performance bottleneck for these applications. In this paper, we address an approach that is combined materialized views selection and computing the views for implementation of materialized views. The Partial View Materialization Approach includes two phases of views selection and views materialization. It also includes the advantages of Extended Reverse Progressive View Materialization Algorithm (ERPVMA) and Overlap Method, so the approach can effectively improve the data warehouse performance.

Keywords: On-Line Analytical Processing; Data Cube; Data Cube Lattice; Materialized View; Partially-Materialized Lattice; Overlap Method.

1. Introduction

In many enterprises, *Decision Support Systems* (DSS) play an important role for businesses. The main reason for DSS's popularity is that DSS are the key to gaining competitive advantage for business. Many enterprises have built or are building unified decision-support databases called *data warehouses* on which users can carry out their analysis. Since queries to data warehouse tend to be queried which identify trends in large multidimensional data, these queries typically make large use of aggregations. This leads to a necessity of multidimensional data analysis: *On-Line Analytical Processing*. OLAP is a technique that guarantees extremely fast response time for multidimensional queries in data warehouse.

Two commonly used techniques to speed up OLAP queries are materialized views and indices. But the storage space of the dedicated OLAP server may be smaller than the total size of all data cubes. That is, only parts of the data cubes can be stored, and so we need to select suitable data cubes that can minimize the query cost and can used to answer more queries. Materialized views selection is the most important decisions in building a Data Warehouse. At the heart of all OLAP is the ability to simultaneously aggregate across many sets of dimensions. Computing multidimensional aggregates is a performance bottleneck for OLAP. So how to improve the question to increase the OLAP performance is one of the most significant studies of the subject.

Hence, in this paper, we address an approach that is combined materialized views selection and computing the views for views materialization. The Partial View Materialization Approach includes the advantages of ERPVMA and Overlap Method, so we expecting the approach can improve the data warehouse performance, and let the performance optimizations.

We consider a warehouse of retail information, with point-of-sale (pos) data from one thousand of stores. The point of sale data is stored in the warehouse in a large pos table, called a fact table. It contains a tuple for each item sold in a sales transaction. Each tuple has the format:

pos (storeID, itemID, quarter, qty, price).

Let the stores and items tables contain store information and item information, respectively. The key of stores is storeID, and the key of items is itemID.

stores(storeID, city, region).

items (itemID, name, category, cost).

The rest of the thesis is organized as follows. In Chapter 2, we survey the related works that is used by the Approach that we address. In Chapter 3, we detail explain the Partial View Materialization Approach that we proposed. In Chapter 4, we experiment on Extended Progressive View Materialization Algorithm and Extended Reverse Progressive View Materialization Algorithm. Then, analysis and explain what are advantages of the approach that we present. In Chapter 5, we conclude this thesis and point out the future work that has arisen from this area.

2. Related Works

2.1. The Lattice Framework

- **Lattice Model [4]:** We can aggregate any combinations of the three dimensions, *stores, items, quarter,* and obtain totally eight possible views (group-by or cubes). The relation of these eight views can be modeled as a cube-lattice shown in Figure 2.1.

- **Dimension Hierarchies and Lattices [4]:** The various dimensions represented by the group-by attributes of a fact table often are organized into dimension hierarchies. The hierarchies are very important to OLAP queries, because they

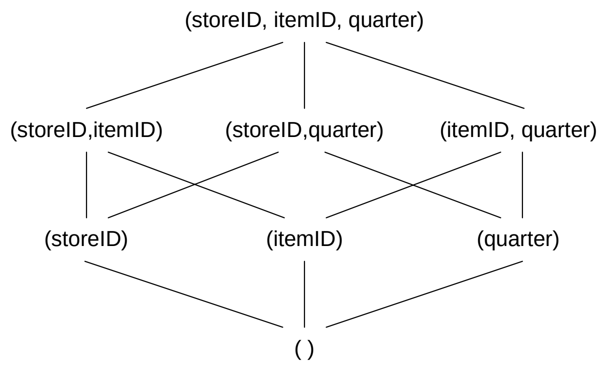

Figure 2.1: Data cube lattice of pos.

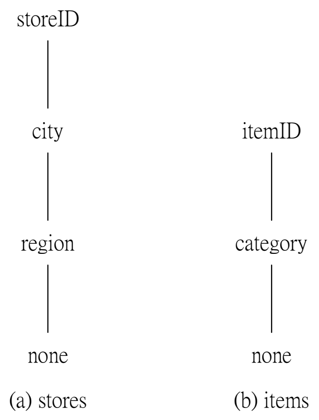

Figure 2.2: Hierarchies for the stores and items

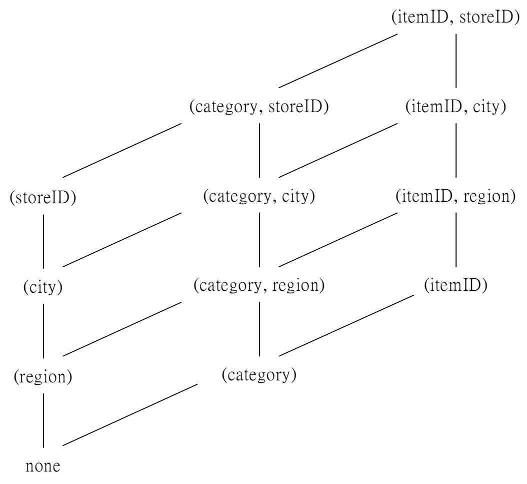

Figure 2.3: Combining two hierarchical

realize the “drill-down” and “roll-up” operations. With the presence of hierarchies, the lattice diagram becomes more complex. For example, consider two dimensions: store and item, whose hierarchies are shown in Figure 2.2. The resulting diagram of combining these two dimensions with hierarchies is shown in Figure 2.3.

- **Generalized Cube Views and Partially Materialized Lattices [6]:** Some views may do aggregation on columns used as dimension attributes in other views. We will call these views *generalized cube views*. A *partially materialized lattice* is obtained by removing some nodes of the lattice, to represent the fact that the corresponding views are not being materialized. When a node n is removed, all incoming and outgoing edges from node n are also removed, and new edges are added between nodes above and below node n.

2.2. Progressive View Materialization Algorithm (PVMA)

The benefit and cost metrics that discussed in the paper [8] use in selecting views to materialize are base on the frequency of updates and accesses on each view and the view size. The PVMA assumes that the selection of each materialized view is done independently. It also assumes that there is no space constraint in the warehouse and that the OLAP uses relational database systems (ROLAP). The benefit $benefit_k(v)$ of selecting a view v is considered for all views v that are not in the set of selected views in iteration k. The profit $benefit_k(v)$ of a view v is the subtraction of benefit and cost of view v. The view that yields the maximum positive profit is selected to materialize. The search terminates when it is not possible to increase the profit further.

- **Data Cube Lattices:** As discussed in the paper [4], a data cube can be presented by using a lattice. For example, consider that there are 2 dimensions: *Product* and *Region* as a sample. The representation of these two dimensions as a lattice is shown in Figure 2.4. The box in Figure 2.4, a number at lower left represents a view size and a number at lower right is the frequency of queries on the view.

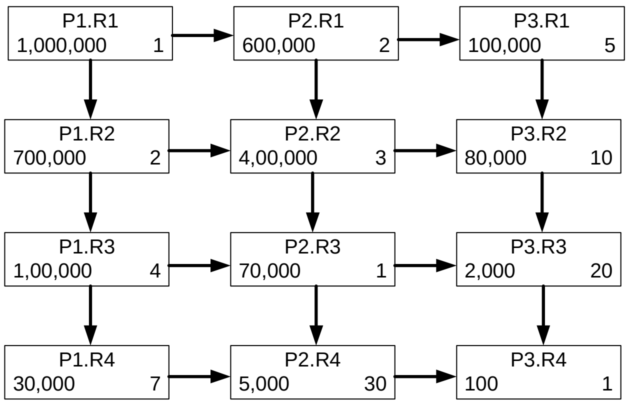

Figure 2.4: Lattice of data cube [8].

- **Nearest Materialized Parent View (NMPV):**

  $NMPV_k(v) = \min(R(v_s), R(v))$ $\forall v_s \in S$, where $v_s \xrightarrow{*} v$

  where $v_s \xrightarrow{*} v$ means that view v_s is one of the parents of view v, and S represents a collection of views which the PVMA algorithm decides to materialize.

- **Benefit Calculation:** The benefit $benefit_k(v)$ of view v in iteration k is:

  $$benefit_k(v) = (\frac{(R(NMPV(v) - R(v))}{bf} \sum_{p \in child(v) \cup v} fp)T_{rba}$$

  where T_{rba} is the time for one random block access, and bf is the blocking factor of the view. The view $child(v)$ represents child views, whose NMPV is v.

- **Cost Calculation:** Each change to the fact table in OLAP involves update to each view - all dimensions and all levels of each dimension are affected. We recommend that when aggregated views are created, their primary key indexes are created as well. We assume that their primary key indexes are implemented as B^+ trees. We derive the time estimation for each update operation as follows, where f represents the frequency of each operation on a view.

  - **Insert and Delete:** (i) read a block of a view, (ii) rewrite a block of the view, (iii) read a leaf node (block), and (iv) rewrite the leaf node. Thus total cost for the insertion of one row is $4f(rba)$.

  - **Update:** (i) read a leaf node, (ii) read a block of a view, and (iii) rewrite the block of a view. Thus total update cost for one update is $3f(rba)$.

  We can derive a formula of the cost of update operations on view v as $\cos t = (\sum_{i \in I} 4N_i f + \sum_{d \in D} 4N_i f + \sum_{u \in U} 3N_u f_u)T_{rba}$. T_{rba} is the time for one *random block access*.

- **Profit Calculation:** PVMA calculates benefit and cost for each view in each iteration. The function $profit_k(v)$ that is the profit of view v, and is defined as $profit_k(v) = benefit_k(v) - \cos t(v)$.

2.3. Overlap Method

- **Choosing a Parent to Compute a View [1]:** Each view in the view DAG (Directed Acyclic Graph) has more than one parent from which it could be computed. We need to choose one of these parents thus converting the DAG to a rooted tree. The root of the tree is the base view and each view's parent is the view to be used for computing it. For example, one possible tree for computing the DAG in Figure 2.5 is as shown in Figure 2.6.
- **Choosing a Set of Views for Overlapped Computation:** The next step is to choose a set of views that can be computed concurrently within the memory constraints. To compute a view in memory, we need memory equal to the size of its partition. We assume that we have estimates of sizes of the views. We have shown that finding an overall optimal allocation scheme for our view tree is NP-hard. So, instead of trying to find the optimal allocation we do the allocation by using the heuristic of traversing the tree in a breadth first (BF) search order:
    - Views to the left have smaller partition sizes, and require less memory. So consider these before considering views to the right.
    - Views at a higher level tend to be bigger. Thus, these should be given higher priority for allocation than views at a lower level in the tree.
- **Example Computation of a CUBE:** Consider the CUBE to be computed on {A, B, C, D}. The tree of views and the estimates of the partition sizes of the views are shown in. If the memory available is 25 pages, BF allocation will generate three subtrees, each of which is computed in one pass. These subtrees are shown

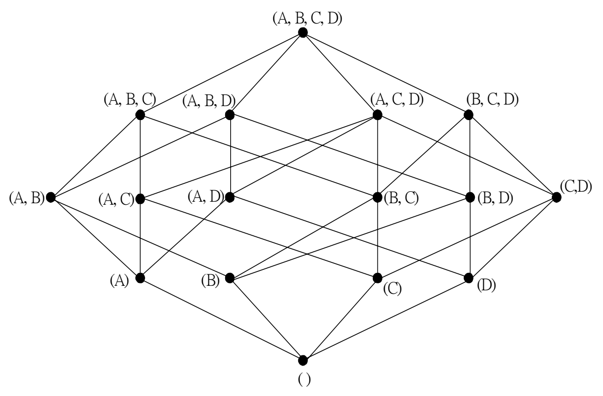

Figure 2.5: Sort orders enforced on the views.

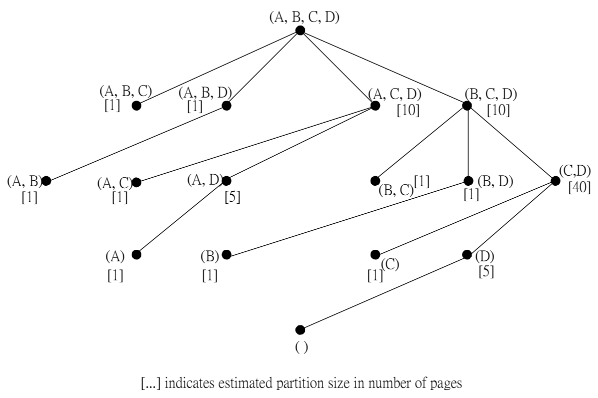

Figure 2.6: Estimates of partition sizes.

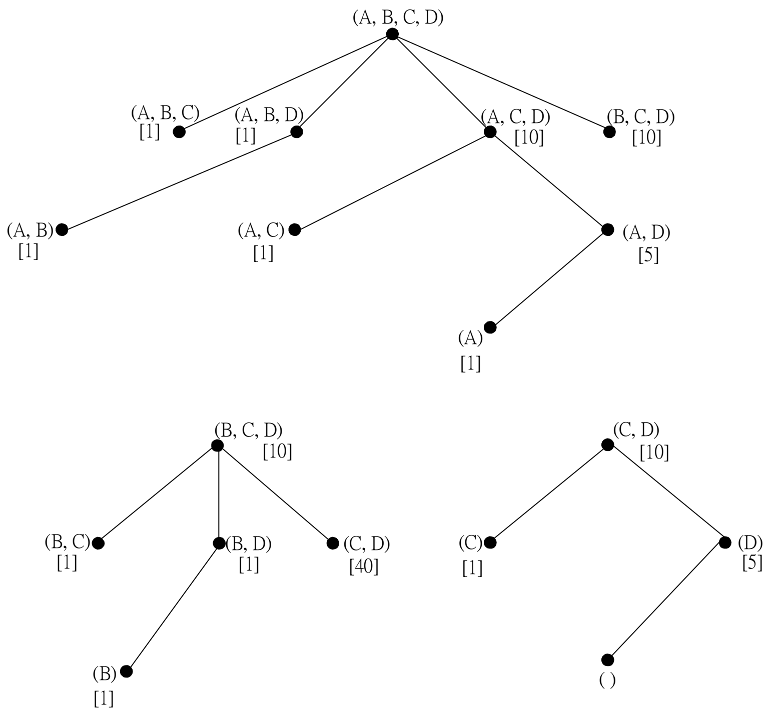

Figure 2.7: Steps of the algorithm [1].

in Figure 2.7. In the second and third steps the views (B, C, D) and (C, D) are allocated 10 Pages as there are 9 sorted runs to merge.

3. Partial View Materialization Approach

In order to combine the views selection and views materialization to improve the performance of OLAP or multidimensional data analysis applications in data warehouse environment. We address an approach that combines materialized views selection and computing the views for implementation of materialized views.

Figure 3.1 illustrated the *Partial View Materialization Approach* that we proposed in this paper. First, we build the EGCVL from the base tables. Second, through ERPVMA we select the appropriate set of materialized views. Third, according to the set, we can build Partially-Materialized Lattice as the input in the Overlap Method. Then we utilize Overlap Method to compute these views.

3.1. View Selection

3.1.1. Extended Generalized Cube Views Lattices (EGCVL)

Base on the notion as mentioned in the Section 2.1, we can get the *generalized cube views lattices* (GCVL). Then we let GCVL combine with the data cube lattice that is

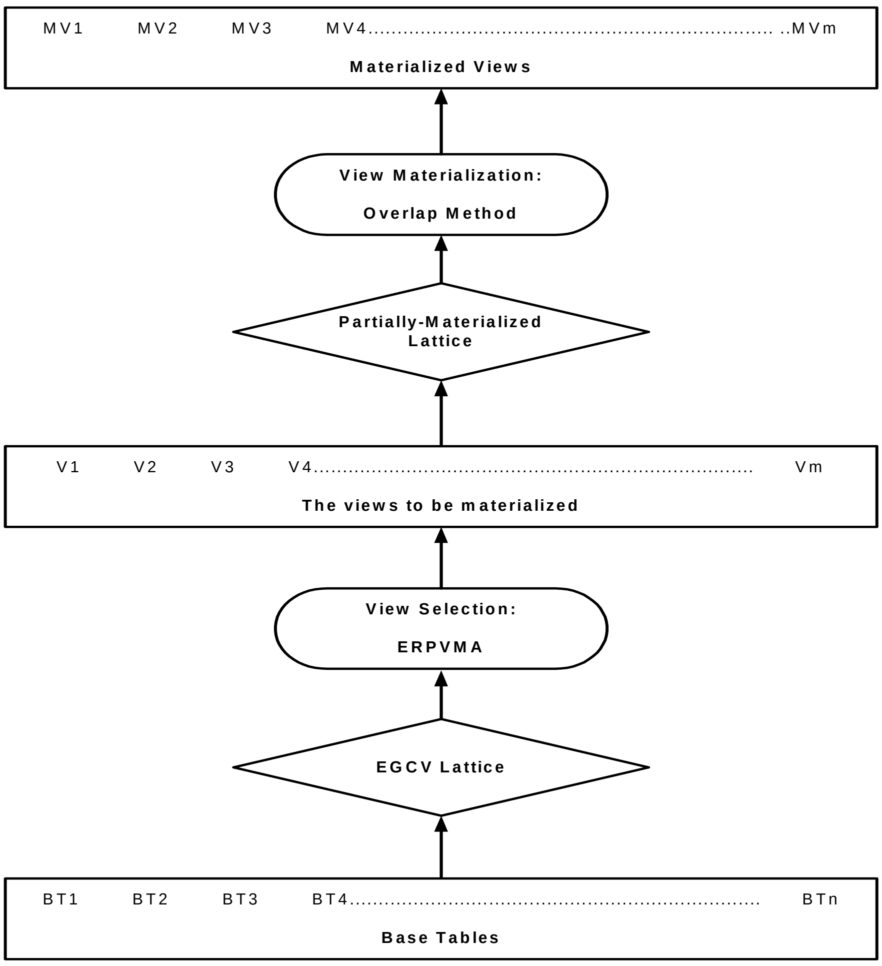
MV1 MV2 MV3 MV4................................ ..MVm
Materialized Views
View Materialization:
Overlap Method
Partially-Materialized Lattice
V1 V2 V3 V4................................ Vm
The views to be materialized
View Selection:
ERPVMA
EGCV Lattice
BT1 BT2 BT3 BT4................................ BTn
Base Tables

Figure 3.1: The Partial View Materialization approach.

mentioned in Section 2.2. The new data cube lattice that is produced by us to be called *Extended Generalized Cube Views Lattices (EGCVL)*. Figure 3.2 shows the EGCVL for pos.

A number at upper middle of the box in Figure 3.2 represents the index of a view. A number at lower left of the box in Figure 3.2 represents the size of a view. A number at lower right of the box in Figure 3.2 is the frequency of queries on the view. The top view (storeID, itemID, quarter) corresponds to the fact table. EGCVL is our main framework to analysis and selection materialized views.

3.1.2. Cost Model

The cost model in our approach is base on that is proposed by [8]. It has described in Section 2.2. But we modify the cost model a little bit. It is described as follows.

- **Benefit Calculation:** Sometimes, the child views of view v are not all have the same Nearest Materialized Parent View with v in a data cube lattice when materialized view selection every time. Before we discuss the new benefit formula in the EPVMA, let us introduce a notion Different Nearest Materialized Parent View Child Views (DNMPVC). If v was materialized, we say that the view v_c is one of the child views of v, but $NMPV(v_c) \neq NMPV(v)$. The DNMPVC(v) represents the set of v_c. Contrary, the Same Nearest Materialized Parent View Child Views (SNMPVC) means if v was materialized, the view v_c is one of the child views of v, and $NMPV(v_c) = NMPV(v)$. The SNMPVC(v) represents the set of v_c. Hence, the *new benefit calculation* is:

$$benefit_k(v) = \left(\left(\sum_{v_c \in DNMPVC(v)} \frac{(R(NMPV(v_c)) - R(v))}{bf} \sum_{p \in child(v_c) \cup v_c} fp\right) + \left(\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in SNMPVC(v) \cup v} fp\right)\right) T_{rba}$$

(3.1)

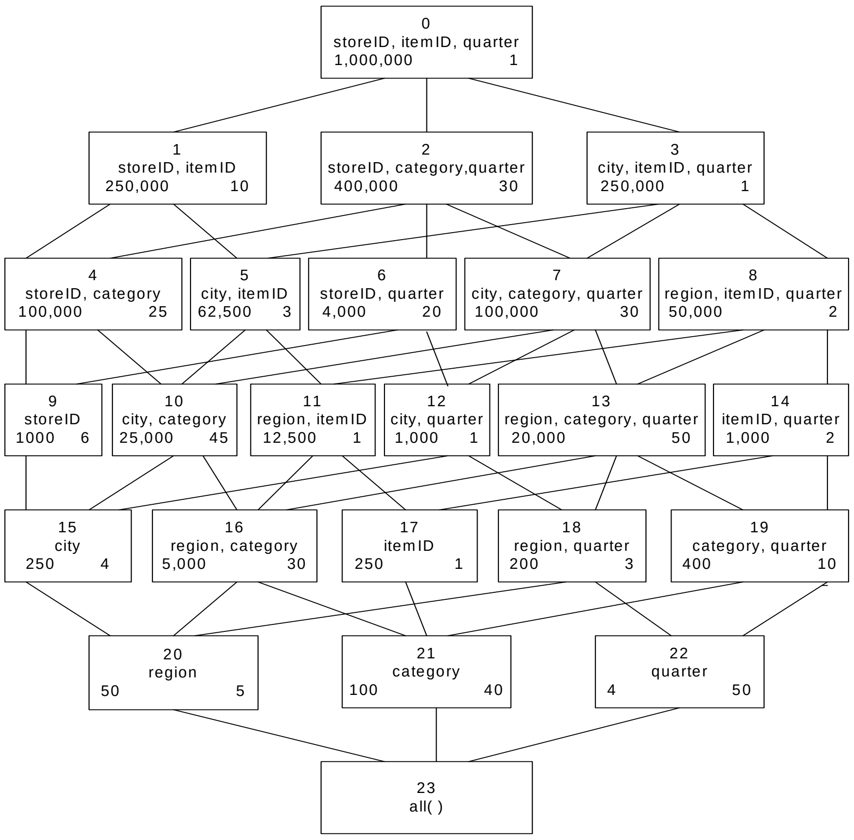

Figure 3.2: Extended Generalized cube views lattices for pos.

- **Cost Calculation:** Due to the source changes are loaded into the warehouse at regular intervals, usually once a day. So we consider the size of views and frequency of queries on views is happen between two update operations. So we remove the factor which frequency of update operations on each view. Hence, the *new cost calculation* is:

$$\cos t = (\sum_{i\in I} 4N_i + \sum_{d\in D} 4N_i + \sum_{u\in U} 3N_u)T_{rba} \qquad \textbf{(3.2)}$$

- **Profit Calculation:** The profit calculation is not change, it is like Eq. (3.3).

$$profit_k(v) = benefit_k(v) - \cos t(v) \qquad \textbf{(3.3)}$$

3.1.3. Extended Progressive View Materialization Algorithm (EPVMA)

The algorithm of PVMA [8] assumes that (1) the selection of each materialized view is done independently; (2) there is no space constraint in the warehouse and (3) that the OLAP uses relational database systems (ROLAP). In our algorithm, we still retain the assumption (1) and (3). But we modify the assumption (2), we add the storage space constraint in our algorithm. Due to at the actual situation, storage space is limited. Thus, the space condition is needed to consider.

In this paper, the first algorithm that we propose is called *Extended Progressive View Materialization Algorithm (EPVMA)*. It is described in Figure 3.3. EPVMA takes the tactic that selects the largest profit view to materialize every time. It assumes that only has the top view (fact table) to be materialized initial. In EPVMA, calculating the views size is represented by $C(x)$. The parameter x is represent a view or the set of the views.

Initial:

```
    $MVs = \{v0\}$    /* $MVs$ : views which should be materialized ;  $v1$ : The top view
    $NR = \phi$    /* $NR$ : views which should not be materialized
    $\cos t = (\sum_{i \in I} 4N_i + \sum_{d \in D} 4N_d + \sum_{u \in U} 3N_u)T_{rba}$
    $S =$ Total storage space
    $CandidateViews = \phi$
   /* $CandidateViews$ : views which can be selection to materialized)
```

Algorithm:

```
01 WHILE $C(MVs) < S$
02 BEGIN
03      FOR all views $v$
04      BEGIN
05           IF ( $(v \notin MVs)$ & $(v \notin NR)$ & $(C(MVs) + C(v) < S)$ ) THEN
06                Add $v$ into $CandidateViews$
07           END IF
08      End FOR
09      IF $CandidateViews = \phi$ THEN
10           Exit the while loop
11      END IF
12      FOR all views $v$ in $CandidateViews$
13      BEGIN
14           $benefit_k(v) = ((\sum_{v_c \in DNMPVC(v)} \frac{(R(NMPV(v_c)) - R(v))}{bf} \sum_{p \in child(v_c) \cup v_c} fp) + (\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in SNMPVC(v) \cup v} fp))T_{rba}$
15           $profit_k(v) = benefit_k(v) - \cos t(v)$
16           IF ( $benefit_k(v) < 0$ ) THEN
17                Add $v$ into $NR$
18                Remove $v$ from $CandidateViews$
19           END IF
```

```
        END FOR
        $Bview = \phi$   /* $Bview$ : a view with maximum profit
        $Bvalue = 0$   /* $Bvalue$ : profit of $Bview$
        FOR all views $v$ in $CandidateViews$
        BEGIN
            IF ( $Bview = \phi$ ) THEN
                $Bview = v$
                $Bvalue = profit_k(v)$
            END IF
            IF ( $Bvalue < profit_k(v)$ ) THEN
                $Bview = v$
                $Bvalue = profit_k(v)$
            END IF
        END FOR
        Add $Bview$ to $MVs$
        $CandidateViews = \phi$
    END of While Loop
```

Figure 3.3: EPVMA Algorithm.

3.1.4. Reverse Progressive View Materialization Algorithm (RPVMA)

In EPVMA, it calculate the benefit consider the sum of query frequencies on the views that may be materialized and its child views. This may be generated puffiness issue when we calculate the profit in order to selection the materialized view. In order to solve this issue, we address the *Reverse Progressive View Materialization Algorithm* (RPVMA). It is described in Figure 3.4.

The difference between RPVMA and EPVMA is RPVMA assumes that all of the views are materialized initially. A heuristic is to select the smallest profit view to be removed every time. In RPVMA, $NPV(MVs)$ represent whether MVs has any view's profit smaller than zero or not. The algorithm has two stages. First, it removes the smallest profit every time until $NPV(MVs) \neq True$. Second, When MVs has not any view's profit smaller than zero, but $C(MVs) > S$, it is continue remove the view that has the smallest profit until $C(MVs) < S$.

```
Initinal:
        $MVs = \{\text{all views}\}$   /* $MVs$ : views which should be materialized
        $NR = \phi$   /* $NR$ : views which should not be materialized
        $\cos t = (\sum_{i \in I} 4N_i + \sum_{d \in D} 4N_d + \sum_{u \in U} 3N_u)T_{rba}$
        $S =$ Total storage space
```

```
$CandidateViews = \phi$
/* $CandidateViews$ : views which can be selection to remove from $MVs$ )
```

Algorithm:

```
01  WHILE $NPV(MVs) = True$
02  BEGIN
03      FOR all views $v$
04      BEGIN
05          IF ( $(v \in MVs) \& (v \notin NR)$ ) THEN
06              Add $v$ into $CandidateViews$
07          END IF
08      End FOR
09      IF $MVs$ = {top view} THEN
10          Exit the while loop
11      END IF
12      $Bview = \phi$   /* $Bview$ : a view with minimum profit
13      $Bvalue = \infty$   /* $Bvalue$ : profit of $Bview$
14      FOR all views $v$ in $CandidateViews$
15      BEGIN
16          $benefit_k(v) = (\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in child(v) \cup v} f_p) T_{rba}$
17          $profit_k(v) = benefit_k(v) - \cos t(v)$
18          IF ( $Bview = \phi$ ) THEN
19              $Bview = v$,
20              $Bvalue = profit_k(v)$
21          END IF
22          IF ( $Bvalue > profit_k(v)$ ) THEN
23              $Bview = v$
24              $Bvalue = profit_k(v)$
25          END IF
26      END FOR
27      Remove $Bview$ from $MVs$
28      Add $Bview$ into $NR$
29      $CandidateViews = \phi$
30  END of while loop
31  WHILE $C(MVs) > S$
32  BEGIN
33      FOR all views $v$
34      BEGIN
35          IF ( $(v \in MVs) \& (v \notin NR)$ ) THEN
36              Add $v$ into $CandidateViews$
37          END IF
38      End FOR
39      IF $MVs$ = {top view} THEN
40          Exit the while loop
41      END IF
42      $Bview = \phi$   /* $Bview$ : a view with minimum profit
43      $Bvalue = \infty$   /* $Bvalue$ : profit of $Bview$
44      FOR all views $v$ in $CandidateViews$
45      BEGIN
46          $benefit_k(v) = (\frac{(R(NMPV(v)) - R(v))}{bf} \sum_{p \in child(v) \cup v} f_p) T_{rba}$
47          $profit_k(v) = benefit_k(v) - \cos t(v)$
48          IF ( $Bview = \phi$ ) THEN
```

```
                $Bview = v$,
                $Bvalue = profit_k(v)$
            END IF
            IF ( $Bvalue > profit_k(v)$ ) THEN
                $Bview = v$
                $Bvalue = profit_k(v)$
            END IF
        END FOR
        Remove $Bview$ from $MVs$
        Add $Bview$ into $NR$
        $CandidateViews = \phi$
    END of while loop
```

Figure 3.4: Reverse EPVMA Algorithm.

3.1.5. Extended RPVMA (ERPVMA)

Although RPVMA solves the puffiness issue, but its utility rate of storage space is not good. Because the algorithm is doing the while loop until there is not any view's profit smaller than zero in MVs, the surplus storage space maybe enough to add another views that good for query.

In order to make up for the drawback, we provide the algorithm that combines the RPVMA and EPVMA is called Extended *Reverse Progressive View Materialization Algorithm (ERPVMA).* It is described in Figure 3.7. The algorithm runs RPVMA first, and then utilizes the EPVMA.

3.2. View Materialization

3.2.1. Building a Partially-Materialized Lattice

When we process the ERPVMA after, we can get the partial views that will be materialized. Then we build the Partially-Materialized Lattice according to the result. About the notion of Partially-Materialized Lattice was described in Section 2.1.

3.2.2. View Materialization with Overlap method

At the heart of all OLAP or multidimensional data analysis applications is the ability to simultaneously aggregate across many sets of dimensions. Computing multidimensional aggregates is a performance bottleneck for these applications. The

Overlap Method [1] is a fast algorithm for computing a collection of group-bys. The method in [1] focus on a special case of the aggregation problem－computation of the CUBE operator. The CUBE operator requires computing group-bys on all possible combinations of a list of attributes, and is equivalent to the union of a number of standard group-by operations. In this paper, we select the Overlap Method to compute the Partially-Materialized Lattice. We use the partially materialized lattice as the input of the overlap method. About the steps that are how to run the Overlap Method was described in Section 2.3.

4. Experiments and Results Analysis

4.1. Experiments for EPVMA and ERPVMA

4.1.1. Experiments Data

The data of these experiments are base on motivation example. In our experiments, we used conditions shown as follows:

Table 4.1: The execution conditions of experiments.

Lattice	Figure 3.3
Update operations	Table 4.2
Block size	4096 bytes
Record size (fact table)	100 bytes
T_{rba}	16(msec)

Table 4.2: Description of update operation for experiments.

Operation	**Number of updated rows**
Insert I_1	3500
Delete D_1	2500
Update U_1	4000

Running these algorithms before, we calculate the blocking factor (*bf*), cost, and the range of total storage space:

$$bf = \lfloor 4096/100 \rfloor = 40$$

$$\cos t = (4\times 3500 + 4\times 2500 + 3\times 4000)\times 16 = 576000$$

$$0 \le \textit{ total storage space } \le 1283254$$

4.1.2. Experiments Result

The graph in Figure 4.1 shows the total query cost of the two algorithms. We can see the total query cost in ERPVMA is almost smaller than in EPVMA. But has an exception in increasing rate of storage space at 20%.

Although, in the experiments result, it still has some exceptions. We can’t sure that ERPVMA is the best. But it is true that provide the better performance. So, in our opinion that is using the algorithm that we propose to select materialized views is better than others. And it provides user a new selection for materialized views selection. Table 4.3 makes some comparison with PVMA[8], EPVMA and ERPVMA.

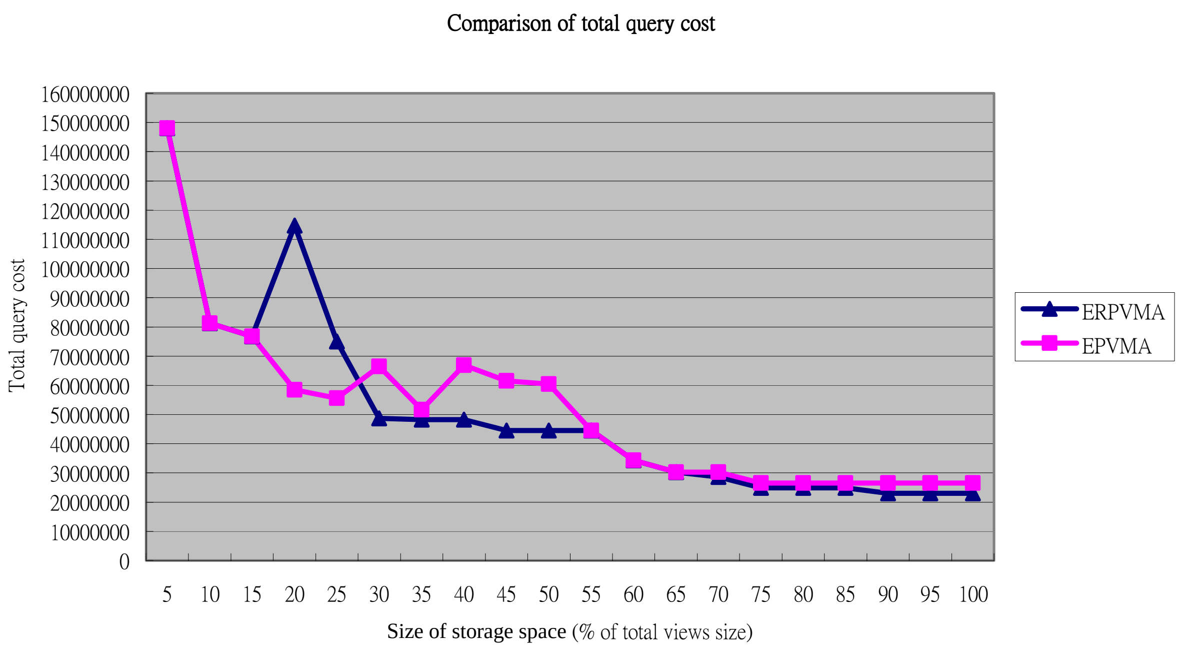

Figure 4.1: Make a comparison between the size of storage space and total query cost.

Table 4.3: makes some comparison with PVMA, EPVMA and ERPVMA.

	PVMA	**EPVMA**	**ERPVMA**
Frequency of selection queries	Yes	Yes	Yes
Cost of update operations	Yes	Yes	Yes
Storage space condition	No	Yes	Yes
Puffiness question	Yes	Yes	No
Size of storage space (% of total views size) $\geq$ 30%	---	Total query cost is bigger than ERPVMA.	Total query cost is smaller than EPVMA.

4.2. Analysis of Partial View Materialization Approach

The approach includes two phases. In the views selection phase, we use the ERPVMA algorithm, the advantages in ERPVMA as the mention in Section 4.2. In the views materialization phase, we use Overlap Method. Its advantages are described as follows.

The Overlap Method is proposed in [1] is a sort-base overlap method. It includes the optimizations share-sorts, smallest-parent, cache-results, and amortize-scans. The Overlap Method computes the view that try to minimize the number of disk accesses by overlapping the computation of the various views. They make use of partially matching sort orders to reduce the number of sorting steps required. And the Overlap Method is a multi-pass method. In each pass, a set of views is selected for computing under memory constraints. These views are computed in an overlapped manner. The tuples generated for a view are used to compute its descendents in the DAG. This pipelining reduces the number of scans needed. The process is repeated until all views get computed.

5. Conclusions

In conclusions, the Extended Reverse Progressive View Materialization Algorithm

(ERPVMA) considers the frequency of selection queries and update cost. It is far better in situations that involve storage space condition. And we also solve the puffiness issue that PVMA maybe happens. That let our algorithm is more perfect. And in the experiments result, it proves the ERPVMA has the better performance than other algorithms for materialized views selection.

In the views materialization phase of the Partial View Materialization Approach, we use the Partially-Materialized Lattice as the input in the overlap method. Letting views selection and views materialization to combine to generate the Partial view materialization approach. The Overlap Method includes the optimizations Smallest-parent, Cache-results, Amortize-scans, and Share-sorts. It has presented one particular sorting based scheme called *Overlap*. This scheme overlaps the computation of different views and minimum the number of scans needed.

The Partial View Materialization Approach includes the advantages of ERPVMA and Overlap Method, so the approach can let the performance of OLAP is optimization.

It maybe selects queries that do not require scanning of views by instead using indexes, and in our paper, we do not consider the index question. In order to group by attributes further along the dimension hierarchy, the fact table must be joined with the dimension tables before doing the aggregation. But our approach does not deal with the factor. So, they are the factors that have to be added in our future work.

References

[1] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi, "On the Computation of Multidimensional Aggregates", *Proceedings of the 22nd VLDB Conference*, pp.506 - 521 Mumbai (Bombay), India, 1996.

[2] P. M. Deshpande, S. Agarwal, J. F. Naughton, and R. Ramakrishnan, "Computation of Multidimensional Aggregates", *Technical Reprot-1314*, University of Wisconsin-Madison, 1996.

[3] J. Han and M. Kamber, "*Data Mining: Concepts and Techniques*", Morgan Kaufmann Publishers, 2001.

[4] V. Harinarayan, A. Raharaman, and J. Ullman, "Implementing Data Cubes Efficiently", *Proceedings of ACM SIGMOD 1996 International Conference on Management of Data*, pp.205 - 216, Montreal, Canada, June 1996.

[5] P. Hass, J. F. Naughton, S. Seshadri, L. Stokes, "Sampling-Based Estimation of the Number of Distinct Values of an Attribute", *In Proceedings of the 21st VLDB Conference*, pp.311 – 322, 1995.

[6] I. S. Mumick, D. Quass, and B. S. Mumick, "Maintenance of Data Cubes and Summary tables in a Warehouse", *Proceedings of ACM SIGMOD 1997 International Conference on Management of Data*, pp.100 - 111, Arizona, United States, 1997.

[7] T. Teorey, "*Data Modeling and Design*", Third edition, Morgan Kaufmann Publishers, Inc., 1999.

[8] H. Uchiyama, K. Runapongsa, and T. J. Teorey, "A Progressive View Materialization Algorithm", *Proceedings of the second ACM international workshop on Data warehouse and OLAP*, pp.36 - 41, Kansas City, Missouri, United States, 1999.