Chapter 7

Data Cube Materialization

OLAP (On-line Analytical Processing) operations deal with aggregate data. Hence, materialization or pre-computation of summarized data are often required to accelerate the DSS (Decision Support System) query processing and data warehouse design. Otherwise, DSS queries may take long time due to huge size of data warehouse and complexity of the query, which is not acceptable in DSS environment. Different techniques like query optimizers and query evaluation techniques [CS94, GHQ95, YL95] are being used to reduce query execution time. View materialization is also one very important technique, which is used in DSS to reduce query response time. Therefore, researchers are always in search of better algorithms, which can select best views to be materialized. In this chapter also, there has been attempt to develop a better view materialization technique by exploitation of density concept and association rule mining technique. Different indexing techniques like bit-map index, join index, etc. are also used to reduce the query response time to a great extent.

Query response time largely depends on the data structure used to represent the aggregates. One of the most efficient data structures is data cube [GCB+97], which is used widely to represent multidimensional aggregates in data warehouse systems. A data cube allows data to be modeled and viewed in multiple dimensions. In SQL terminology, data cube is nothing, but collection of group-bys. Let us take an example. Suppose, an organization keeps sales data of a particular product with respect to time(t), location(l) and branch(b) without any hierarchy as has been shown in Figure 7.1 on the next page. Here, the data cube
consists of eight possible group-bys: $tlb, tl, tb, bl, t,l,b$ and none. Each individual group-by is called sub-cube or cuboid or view. In data warehouse systems, query response time largely depends on the efficient computation of data cube. However, creating data cube on the fly is very much time and space consuming.

So, one common technique used in data warehouse is to materialize (i.e., pre-compute) cuboids of a data cube. To do this, there are three possibilities:

1. Materialize the whole data cube: This is the best solution in terms of query response time. However, computing every cuboid and storing them will take maximum space if data cube is very large, which will affect indexing and the query response time.

2. No Materialization: Here, cuboids are computed as and when required. So, the query response time fully depends on database which stores the raw data.

3. Partial materialization: This is the most feasible solution. In this approach, some cuboids or cells of a cuboid are pre-computed. However, the problem
is how to select these cuboids and cells to be pre-computed. Generally, cuboids and cells which can help in computing other cells or cuboids, are pre-computed.

There exists some view materialization algorithms. However, most of them work on some constraints such as space to store the views, time to update the views, etc. Some well known algorithms are \textit{BPUS} [HRU96], \textit{PBS} [SDN98], \textit{PVMA} [URT99], \textit{A\*} [GYC+03], etc. \textit{BPUS} is a greedy algorithm, which selects the views with the highest benefit per unit space. The complexity of the algorithm is $O(k.n^2)$, where $k$ is the number of views to be selected and $n$ is the total number of views. The main disadvantage of the algorithm is that its execution time increases exponentially with the increase of number of views. Otherwise, the algorithm selects better views in terms of benefits. \textit{PBS} (Pick By Size) algorithm selects the views on the basis of view size. However, \textit{PBS} is meant only for SR (Size Restricted)-Hypercube lattice. \textit{A\*} algorithm is one of the recent algorithms. The algorithm is interactive, flexible and robust enough to find the optimal solution under disk space constraint and the algorithm has been found to be useful when disk-space constraint is small. The algorithm has used two powerful pruning techniques (H-pruning and F-pruning) and two sliding techniques (sliding-left and sliding-right) to further improve the running efficiency of the search. Above all, there is one algorithm called \textit{PVMA} (Progressive View Materialization Algorithm) [URT99]. The algorithm is based on the concept of Nearest Materialized Parent Views (NMPV). To the best of our knowledge, this is the first algorithm to have used access frequency of queries to select the views. It also considers updates on views and view size to calculate benefits of the views. So, this algorithm can be found to select better views than other algorithms [URT99].

This chapter has discussed performance analysis of \textit{PVMA} algorithm in detail for the reasons given above and attempted to present a faster view materialization algorithm (\textit{DVMAFC}) based on the notion of density and frequency count (support count) of the views. The algorithm basically forms clusters of views and selects the core views for materialization. The concept of density has been taken from the algorithm \textit{DBSCAN} [EKS+96], which is a well-known clustering algorithm. The algorithm \textit{DVMAFC} also has applied the concept of cost/benefit of \textit{PVMA} to form the clusters of views. In addition to that, the algorithm has
used the supports of the frequent (or large) sub-views to calculate the benefits, because it has been observed that supports of the frequent (or large) views plays an important role to select better views to be materialized. At the end, the chapter has compared the performance of DVMAFC with PVMA. It has been observed that in most of the cases DVMAFC selects better views and works much faster than PVMA.

### 7.1 Data Cube Lattice

All the view materialization algorithms are required to use some data structures to represent the data cube. One useful data structure is data cube lattice [HRU96], which has been used by many algorithms to represent a data cube. Let us consider the above example of sales data. The group-bys (views) can be organized in the form of a lattice as shown in Figure 7.2, which is a directed acyclic graph. The top view \( tlb \) is known as fact table. An edge from a view \( u \) to view \( v \) in the graph means that \( v \) can be calculated from \( u \). [HRU96] also has shown that this relationship is in partial order. DVMAFC has also used data cube lattice to represent the views.

![Figure 7.2: A Lattice](image)
7.2 Progressive View Materialization Algorithm (PVMA)

PVMA [URT99] assumes that data cube is represented in the form of lattice as discussed in [HRU96] and selects the appropriate views to be materialized, which minimizes the query response time and maintenance cost. The feature, which distinguishes the algorithm from other algorithms is the use of size of the views, access frequency of queries (views), updates (insert, edit and delete) on each view to select the views for materialization. The algorithm also uses number of rows affected by each of the update operations. These parameters information are usually available and can be kept track easily in a data warehouse system by the warehouse administrator, considering the fact that data warehouse is updated in off-peak period. Followings are some of the concepts used in the algorithm:

- **Nearest Materialized Parent Views (NMPV):** A view \( u \) is a parent view of \( v \), if \( v \) can be computed from \( u \). NMPV of of \( v \) at iteration \( k \), denoted by \( \text{NMPV}_k(v) \) is a materialized view \( u \) such that the difference between size of view \( v \) and size of view \( u \) is minimum among all materialized views in the iteration \( k \) of the algorithm. So, \( \text{NMPV}_k(v) = \min(R(u), R(v))\forall u \in S \) and \( u \rightarrow v \).

- **Benefit:** If a view \( v \) is materialized then view \( v \) and its children receive the benefits because children can be computed from \( v \) whose size is smaller than the fact table. The benefit of \( v \) in the iteration \( k \) is calculated as

\[
\text{benefit}_k(v) = \left( \frac{R(\text{NMPV}(v)) - R(v)}{bf} \sum_{u \in \text{child}(v) \cup v} f_u \right) T_{rba} \quad (7.1)
\]

- **Cost:** Each change (insert, delete and update) in the fact table results in update to each corresponding view. So, cost calculation includes the number of operations (insert, delete and update), their frequencies and time for random block access. The cost is calculated as:

\[
\text{cost}(v) = \left( \sum_{i \in SI} 4N_i f_i + \sum_{d \in SD} 4N_d f_d + \sum_{u \in SU} 3N_u f_u \right) T_{rba} \quad (7.2)
\]
It is to be noted that the cost is same for all the views because the formula does not contain any information of the view.

- Profit: Profit of a view $v$ at iteration $k$, denoted by $\text{profit}_k(v)$, is calculated as $\text{benefit}_k(v) - \text{cost}(v)$.

### 7.2.1 The Algorithm

The algorithm is very simple and works as follows. Base cuboid (fact table) is always to be materialized because any cuboid can be calculated from the base cuboid. The algorithm calculates cost, benefit and profit of all the views, which are not included in $NR$, where $NR$ is a set of views with negative profit. The views with negative profit are discarded. Then, the algorithm selects the view with maximum positive profit. The process continues until all the views are either discarded or selected for materialization. The algorithm is given in Algorithm 7.1 on the next page.

#### Example 7.1

Let us consider the lattice given in Figure 7.2 on page 145. Let size (number of rows) of the cuboids $tlb$, $tl$, $tb$, $lb$, $t$, $l$, $b$ be 100, 70, 60, 50, 40, 30, 20 respectively. Let access frequency of the cuboids be 10, 5, 5, 6, 5, 3, 1 respectively. Let us also take $bf$, $Trba$ and $cost$ as 100, 10msec and 5msec respectively. Based on the above assumptions, three iterations of the algorithm are shown in Table 7.1 on page 149, where $s$ and $nr$ represents the cuboid is selected and included in $NR$ respectively. In the first step $tlb$ is selected because it is the fact table; in the second step $lb$ is selected; in the third step $tb$ is selected and $b$ is included in $NR$ because the profit is negative.
CHAPTER 7. DATA CUBE MATERIALIZATION

Input: Lattice of the views, \( V \), Access frequency of the views.
Output: \( S \).

1. \( S=V_1; \) (\( V_1 \) is the base cuboid)
2. \( NR = \phi \);
3. 
   \[
   \text{cost} = \left( \sum_{v \in S} 4N_v f_v + \sum_{d \in SD} 4N_d f_d + \sum_{u \in SU} 3N_u f_u \right) T_{rba}
   \]
4. For \( k=1 \) to \( |V| \)
5. Begin
6. For all views \( v \)
7. Begin
8. If \( (v \in S \& v \notin NR) \) then
9. 
   \[
   \text{benefit}_k(v) = \left( \frac{R(NM PV(v)) - R(v)}{bf} \right) \sum_{u \in \text{child}(v) \cup u} f_u \right) T_{rba}
   \]
10. \( \text{profit}_k(v) = \text{benefit}_k(v) - \text{cost}; \)
11. If \( \text{profit}_k(v) \leq 0 \) then add \( v \) into \( NR; \)
12. End
13. End
14. Find \( P_{\text{view}} \) from all the views \( v \in S; \)
15. Add \( P_{\text{view}} \) to \( S; \)

Algorithm 7.1: PVMA
6.1 Data Cube Materialization

<table>
<thead>
<tr>
<th>Cuboid</th>
<th>First step</th>
<th>Second step</th>
<th>Third step</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Benefit</td>
<td>Profit</td>
<td>Benefit</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tlb</td>
<td>-</td>
<td>s</td>
<td>-</td>
</tr>
<tr>
<td>tl</td>
<td>-</td>
<td>-</td>
<td>39</td>
</tr>
<tr>
<td>tb</td>
<td>-</td>
<td>-</td>
<td>44</td>
</tr>
<tr>
<td>lb</td>
<td>-</td>
<td>-</td>
<td>50</td>
</tr>
<tr>
<td>t</td>
<td>-</td>
<td>-</td>
<td>30</td>
</tr>
<tr>
<td>l</td>
<td>-</td>
<td>-</td>
<td>21</td>
</tr>
<tr>
<td>b</td>
<td>-</td>
<td>-</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 7.1: PVMA Example

7.2.2 Analysis

The complexity of the algorithm is $O(V^2 + SI + SD + SU)$. So, it is clear that the complexity heavily depends on $V$. Complexity increases exponentially with the increase of $V$. However, the algorithm has been found to be superior to other algorithms and considers access frequency of the views, size of the views and maintenance cost of the views to select the views [URT99]. The algorithm performs better in situations which involve databases with more dimensions and different access frequencies of views.

7.3 Density-based View Materialization Algorithm using Frequency Count (DVMAFC)

DVMAFC also assumes that data cube is represented in the form of lattice as discussed in [HRU96] and selects the appropriate views to be materialized, which minimizes the query response time and maintenance cost. Like PVMA, it also uses size of the views, access frequency of queries (views), frequency of updates (insert, edit and delete) on each view and number of rows affected by each of the update operations to select the views for materialization.
CHAPTER 7. DATA CUBE MATERIALIZATION

The important concept used in DVMAFC is the use of concept of density [EKS+96] to form clusters of views and then select the views to be materialized in a data warehouse system. A cluster in DVMAFC consists of views. The main characteristic of the clusters is that the benefit of the neighborhood of any view in a cluster must be at least some pre-defined value. Another new concept is the use of frequency/supports of the frequent sub-views to select the views because it has been observed that supports of the sub-views help select better views for materialization.

7.3.1 Definitions

Followings are some definitions [EKS+96], which are of importance in the context of the algorithm. For all these definitions, it is assumed that views are arranged in the form of a lattice as explained in previous sections.

Definition 7.1

*Neighborhood*: Neighborhood of a view \( v \) with respect to \( MaxD \), denoted by \( N(v) \), is defined by \( N(v) = v \cup \{ w | w \in \text{child}(v) \text{ and } R(v) - R(w) \leq MaxD \} \).

Definition 7.2

*Core View*: A view \( v \) is said to be core view if \( \text{benefit}(N(v)) \geq MinBen \), where \( MinBen \) is the minimum benefit.

Definition 7.3

*Directly-Density-Reachable*: A view \( v \) is directly-density-reachable from a view \( w \), if \( w \) is a core view and \( v \) is in the neighborhood of \( w \).

Definition 7.4

*Density-Reachable*: A view \( v_i \) is density reachable from another view \( v_j \) with respect to \( MinBen \). if there exist a chain of views \( v_1, v_2, ... v_k \) such that \( v_1 = v_j \) and \( v_k = v_i \) and \( v_e \) is directly-density-reachable from \( v_{e+1} \).
Definition 7.5

Density-Connected: Two views \( v_1, v_2 \) are density-connected if there exists another view \( v_3 \) such that \( v_1 \) and \( v_2 \) are density reachable from \( v_3 \).

Definition 7.6

Cluster: A cluster \( Cl \) of views with respect to MinBen and MaxD is a non-empty set of views with the following conditions

1. For two views \( v_1, v_2 \in V \), \( v_2 \in Cl \) if \( v_1 \in Cl \) and \( v_2 \) is density-reachable from \( v_1 \).
2. Two views \( v_1, v_2 \in Cl \) are density connected.

\[
\begin{align*}
\text{tlb, tl: Core points.} \\
\text{tl is directly density-reachable from tlb.} \\
\text{t is density-reachable from tlb.} \\
\text{t, lb are density connected to tlb.}
\end{align*}
\]

Figure 7.3: Neighborhood, Core Points, Density-Reachable and Density-Connected

There are three categories of views - classified, unclassified and noise. Classified views are already associated with a cluster; unclassified views are not yet associated with any cluster; noise views do not belong to any cluster. So, it is understood that neighborhoods of classified and noise views are already calculated. Another category of views, called leader view, has been introduced. A leader view is an unclassified view, of which all the parents are either classified (not materialized) or declared noise.
7.3.2 Frequent Sub-views

Sub-views of a view (group-bys) are basically views consisting of the subsets of the view. As for example, sub-views of a view \((u, v, w)\) are \((u, v)\), \((v, w)\), etc. In other words, sub-views of a view are the descendants of the view in the data cube lattice (Figure 7.2 on page 145). It has been observed that frequent sub-views play an important role in predicting future views. As an example, let us consider five views: \((v_1, v_2, v_3)\), \((v_1, v_3)\), \((v_1, v_2, v_3)\), \((v_1, v_2)\) and \((v_1, v_2, v_5)\). Here, the sub-view \((v_1, v_2)\) is frequent and present in 60% views. So, it can be predicted that future queries may be based on views which are superset of the sub-view \((v_1, v_2)\). In other words, views which are superset of the frequent sub-views should be materialized so that any query on those views can be answered instantly. So, supports of the sub-views should also be considered to calculate benefits of the views.

Finding frequent sub-views may be challenging task, particularly when the view (query) database is very large. For this purpose, frequent itemset finding algorithms, as discussed in Chapter 3, can be of great help. To calculate the frequencies of the sub-views, view database can be represented easily in the form of market-basket database. Let us consider the above example again. The equivalent market-basket database of the five views is given in the Table 7.2. Each transaction represents one view, where 1 represents that the corresponding attribute has occurred in the view and 0 represent that corresponding attribute has not occurred in the view. Now, frequent itemset finding algorithms, as discussed in Chapter 3, can be used to find frequent sub-views with the corresponding supports and these supports will be used to calculate the benefits of the views.

<table>
<thead>
<tr>
<th>(v_1)</th>
<th>(v_2)</th>
<th>(v_3)</th>
<th>(v_4)</th>
<th>(v_5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1 0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0 1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.2: Representation of Views
CHAPTER 7. DATA CUBE MATERIALIZATION

7.3.3 Benefit of a Neighborhood

Benefit is an important concept and the views are selected for materialization on the basis of benefits of the views. The more the benefit of a view, the more likely the view will be selected for materialization. However, benefits of the neighborhoods of the views will be used, instead of the views themselves. Benefit of \( N(v) \), denoted by \( \text{benefit}(N(v)) \) (Formula 7.3), is calculated in the same way as benefit of a view \( v \) is calculated in \( PVMA \) [URT99], which is based on size of the view and access frequencies of the children views. In addition to that, supports of the frequent sub-views have been used, as discussed above, to calculate the benefits of the views. However, only the sub-views of the neighborhoods have been used, because it has been observed that lower level sub-views do not have much effect on the view.

\[
\text{benefit}(N(v)) = \left( R(NMPV(v)) - R(v) \right) \sum_{u \in N(v) \cup v} f_u + \sum_{u \in N(v) \cap Fv} \text{Sup}(u)
\]  

(7.3)

7.3.4 The Algorithm (DVMAFC)

The algorithm centers around forming the clusters of views. While creating clusters, the algorithm has to calculate benefits of neighborhoods. The benefit is based on the view size (number of rows), access frequency of views and frequency count of the sub-views. Frequencies of view access are easily available in any data warehouse system and frequency of the sub-views can be easily calculated using the algorithms discussed in Chapter 3. View sizes can also be calculated easily using the methods given in [SDN98, LS96].

The algorithm assumes that views are selected independently, there is no space constraint and OLAP uses relational database system. The algorithm also assumes that views are organized in the form of lattice as explained the previous sections. The working principle of the algorithm is very simple. It first finds all the clusters of views and then selects the core views of the clusters for materialization. The algorithm always selects the fact table for materialization. So,
top view (fact table) is not included in the creation of the clusters; clusters are created from the rest of the views.

The algorithm works as follows. The algorithm starts with finding the smallest leader view $v$ among the leaders with highest dimensions because clusters are created from the top of the lattice. Then it calculates the benefit of $N(v)$. If the benefit is less than the minimum benefit ($\text{MinBen}$), it is marked as a noise. Otherwise, a cluster starts at $v$, and all the unclassified child views are put into a list of candidate views. Then, one view from the candidate views is picked up and benefit is calculated. If it is a core view, all the unclassified child views are included in the list of candidate views. Otherwise, it is marked as classified. The process continues until the list becomes empty. This way one cluster is formed. Similarly, other clusters are formed. At the end, core views of the clusters are selected for materialization. Here, each view will require to compute the neighborhood only once. So, average run time complexity of the algorithm is $O(V \log V)$.

The algorithm needs two important parameters - $\text{MinBen}$ and $\text{MaxD}$. $\text{MinBen}$ can be set to any arbitrary positive value according to requirement. However, optimum value can be calculated in the same way as cost of a view is calculated in $PVMA$. Similarly, optimum value for $\text{MaxD}$ can be determined in the same way as $\text{Eps}$ has been determined in [EKS+96].

### 7.4 Experimental Results

**Setup**: Performance of $DVMAF C$ and $PVMA$ was compared with two synthetic datasets (TD1 and TD2) and a PIV machine with 256 MB RAM.

**Test data**: Two synthetic data sets (TD1 and TD2) were used for the experiments. Each of them contained 8 dimensions without any hierarchy, one measure attribute and 2 lacs tuples. Each of 255 possible views was indexed from 1 to 255. Values of each dimension and measure attribute were chosen randomly. It was assumed that queries on any view were equally likely. The analytical formula presented in [SDN98, LS96] was used to estimate the size of views. One view (query)

Set all views of $V$ as leader;
$S = \{ \text{fact table} \}$; $Temp = \text{"True"}$;
$clid = \text{Get a new cluster id}$;
Do while there is a leader view
  Find leader view $v$ with smallest in size $(R(v))$ among leader views;
  with maximum dimensions;
  $Temp = \text{CreateCluster}(V, v, clid, MaxD, MinBen)$;
  If $Temp = \text{"True"}$ then
    $clid = \text{Get a new cluster id}$;
  Endif
End DO

CreateCluster($V, v, clid, MaxD, MinBen$)
(Form the cluster with cluster id as $clid$)
If benefit$(N(v)) < MinBen$ Then
  $v.noise = \text{"True"}$;
  Return "False";
Else
  $v.classified = \text{"True"}$; $S = S \cup v$;
  $seeds = \{ w \mid w \in N(v) \text{ and } w.classified = \text{"False"} \}$;
  For all $s \in seeds$ set $s.classified = \text{True}$;
  While Empty$(seeds) = \text{"False"}$ Do
    For each $s \in seeds$
      If benefit$(N(s)) \geq MinBen$ then
        $S = S \cup s$; $Results = \{ w \mid w \in N(s) \}$;
        For each $r \in Results$
          If $r.classified = \text{"False"}$ then
            $seeds = seeds \cup r$; $r.classified = \text{"True"}$;
          Endif
        Endfor
      Endif
    Endfor
  EndWhile
  Return "True";
Endif

Algorithm 7.2: $DVMAFC$
database was created with about 1000 views to calculate access frequencies of the views and frequent sub-views. Frequent sub-views and their frequency were calculated using Modified_Bit_Assoc algorithm with minimum support as 5%. A constant value for MinBen was taken, because this parameter also does not change even if some views have been materialized. bf and T_{ren} were also not considered because these values are constant for all the views and do not affect the selection of views.

Experimental results are shown in the figures 7.4 on the next page, 7.5 on page 158, 7.6 on page 158 & 7.7 on page 159. Figures 7.4 on the next page & 7.5 on page 158 gives the average query cost (in '000 tuples). Figures 7.6 on page 158 & 7.7 on page 159 reports the execution time.

Observations: Experimental results showed that both the algorithms selected almost same views and average query costs were also almost same for both the algorithms. In case of TD1(Figure 7.4 on the next page), PVMA selected slightly better views than that of DVMFC, resulting in slightly better performance in terms of average query cost. In case of TD2(Figure 7.5 on page 158), PVMA outperformed DVMFC marginally in the beginning, when number of materialized views was small. However, as the number of materialized views increased, DVMFC outperformed PVMA in terms of average query cost. This could be attributed to the selection of better views by DVMFC. Another point to be noted is that average query cost becomes almost constant with the increase of number of materialized views. This shows that materialization of too many views does not reduce the query cost. As far as execution time (Figures 7.6 on page 158 & 7.7 on page 159) is concerned, DVMFC takes much less time than that of PVMA. This is the main advantage of the DVMFC over PVMA. The gain in execution time could be attributed to the difference in the time complexities of the algorithms.

7.5 Discussion

This chapter has presented a view materialization algorithm called DVMFC, which has used density concept to select better views. The most important feature of the algorithm is the use of frequency count of the views to select
better views. To find frequency count of the views, the frequent itemsets finding algorithms reported in the previous chapters (Chapter 3) may be of great help. Followings are the other important features of the algorithm.

- Complexity of the algorithm is only $O(n \log n)$, where $n$ is the number of views.
- As far as view selection is concerned, it selects almost same views as that of $PVMA$.
- The algorithm is scalable due to its low complexity.

![Figure 7.4: Average Query Cost ('000 tuples) of $DVMAFC \& PVMA$ - 1](image)
CHAPTER 7. DATA CUBE MATERIALIZATION

Figure 7.5: Average Query Cost ('000 tuples) of DVMAFC & PVMA - II

Figure 7.6: Execution Times of DVMAFC & PVMA - I
Figure 7.7: Execution Time of DVMAFC & PVMA - II