# View Selection in OLAP Environment

Shi Guang Qiu and Tok Wang Ling

School of Computing
National University of Singapore
Lower Kent Ridge Road, Singapore 119260
{qiushigu,lingtw}@comp.nus.edu.sg

**Abstract.** To select some "valuable" views for materialization is an essential challenge in OLAP system design. Several techniques proposed previously are not very scalable for systems with a large number of dimensional attributes in the very dynamic OLAP environment. In this paper, we propose two filtering methods. Our first method, the functional dependency filter, removes views with redundant summary information based on functional dependencies among the dimensional attributes. The second method, the size filter, is based on the view size to filter out any view that can be either derived from another small materialized view or has almost the same number of tuples as another materialized view from which it can be derived. More over, all useful views are selected by these two view filtering methods, other existing view selection methods can still be applied on the remaining views to further reduce other possible non-essential views from systems. We conduct performance tests to compare our method with other existing methods. The results show our method outperform the others.

## 1. Introduction

On-Line Analytical Processing (OLAP) system is a query subsystem inside a Decision Support System (DSS). It is designed to help users to gain insight into data through fast, consistent, interactive access to a variety of data in the database.

To achieve that, Multi-Dimensional model (MD model) is widely adopted by almost all OLAP systems. Under such model, data attributes are classified according to users' intuitive perception of the business, data are put into a simple and standardized data schema and summary-views**,** some sort of intermediate results, are computed on top of that. User queries are redirected to the smaller summary-views instead of the original sources and better response time can be expected.

On the other hand, the number of possible summary-views in the MD model increases explosively with the increasing number of dimensional attributes. The high storage cost and computation cost make it unfeasible for any system to materialize all of these possible views [OLAP]. Which summary-views should be materialized becomes an essential challenge in OLAP research. Although, there are already quite a number of OLAP products, there are still no good solutions for this problem in the market yet.

Several methods have been proposed to select an appropriate subset of views or indexes for materialization based on a set of user queries. However, it is very difficult to get an accurate set of user queries, at least at the system design stage. If all possible queries were included, the number of view needs to be considered would increase significantly, the complexity of these methods would be increased.

In this paper, we will approach the problem from the other angle. Instead of finding which view is more useful for OLAP system, we propose two methods to filter out views that are not very useful for the OLAP query optimization process.

Based on functional dependencies among dimensional attributes, we propose a filtering method, **functional dependency filter,** to trim out views holding duplicate summary information. As functional dependencies are normally available at the system design stage, the functional dependency filter can get ride of many views at this initial stage. Functional dependencies are independent on the actual data, so we do not need to review the filtered views unless there are changes to the set of functional dependencies. Also in this paper, we solve the problem for computing functional dependency filter set for a set of functional dependencies.

Then we move our interest to the view size. In [BPT97], a simple method is proposed to filter out those "big" views, which are almost the same size of the ancestor views. In this paper, we extend this technology to "small" views. With today's technology, computers can easily handle complex queries on a view if its size is smaller enough to be kept in the main memory. It is not very useful for a system to materialize any view that can be derived from another already materialized small view.

In these two methods, all useful views are selected. We can still apply other existing view selection methods easily to further reduce those non-essential views from the remaining views selected by these two methods. In the real environment, these two methods are very efficient and a large percentage of views can be filtered out, we might able to directly materialize all the views in this greatly reduced view set.

In section 2, we define summary view and summary view lattice. In section 3, we propose two filtering algorithms and show the experimental results. Finally, the conclusion is presented in section 4.

## 2. Multi-Dimensional Model and MD Query Model

Before we begin our discussion on view selection, we could like to define some major concepts of MD model and develop some notations and definitions.

### 2.1 Summary-View and Summary-View Lattice

**Multi-Dimensional Model (MD model)** [KIM97] is a "business oriented" model. Under such model, all attributes are divided into two major groups, **measure attributes** and **dimensional attributes,** according to their roles in the business analysis. Measure attributes are numeric measurements about the business processes, while dimensional attributes are describing dimensions of the business processes. In an

OLAP SQL queries, measure attributes are normally acting as input for summary operations such as SUM, while dimensional attributes are appearing in Group-by clause and Where clause [KIM97].

Also in a MD model, we can predict queries that users tend to ask and build some intermediate data sets in advance. Based on that, various kinds precomputation methods, e.g. summary-view, range sum cube [HAMS97] and bit-map index [OQ97] have been proposed. Query performance can be improved drastically as we can get the final result by using these smaller precomputed results. Among all, summary-view approach is one of the most popular ones. In the following part of this paper, we concentrate on summary-view only.

**Definition 1. Summary-View.** A summary-view is an aggregate view grouping some measure attributes along various dimensions, i.e. corresponding to different sets of group-by attributes. In a MD model M, let **GA** be a subset of dimensional attributes, $\mathbf{M_1, M_2}$ **..** be measure attributes, $\mathbf{OP_1, OP_2}$ **..** be SQL aggregate functions like SUM, COUNT , MAX, and MIN[1]. A summary-view in M is of the form:

**SELECT** GA, $OP_1(M_1)$, $OP_2(M_2)$, …
**FROM** relations in M
**WHERE** (joins of relations in M)
**GROUP BY** GA

To simplify the discussion, without loss generality, we assume there is only one measure attribute in the M and SUM is the only SQL aggregate function used in the following part of this paper. Thus, we can denote this summary-view as **SV(GA)**.

To represent the relationship among summary-views in a MD model, we will use a lattice framework to represent all possible summary-views in a MD model as a Summary-View Lattice [HRU96].

**Definition 2. Operator $\preceq$.** We define the operator $\preceq$ between summary-views as below: SV(A) $\preceq$ SV(B) iff SV(A) can be derived from SV(B).

By definition of the summary-view, if $A \subseteq B$ where A and B are two sets of dimensional attributes in a MD model, then SV(A) $\preceq$ SV(B), e.g. SV({FAMILY_ID}) $\preceq$ SV({FAMILY_ID,DATE_ID}).

Operator $\preceq$ is a partial order relationship among summary-views, therefore all possible summary-views of a given MD **M** can be presented as a lattice with the summary-view holding all dimensional attributes in its group-by dimensional attribute set as the top element (top view), and the summary-view that aggregates everything together, i.e. nothing in its group-by dimensional attribute set (empty set) as the bottom element.

[1] SQL function AVG can be derived from SUM()/COUNT()

**Example 1.** Let a MD model M which has three dimensioanl attributes A,B,C. The summary-view lattice for M is shown in Figure 1.

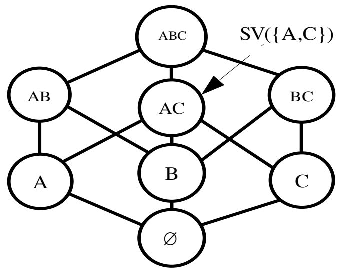

**Fig. 1** A Summary-View Lattice

### 2.2 View Filtering Example

It is obvious that not all these summary-views in summary-view lattice are useful for OLAP query optimization.

**Example 2**. Assuming we have a simple MD model: sales_info. In it, there are four dimensional attributes: Product, Product_type, Branch_id, Date, one measure attribute Unit, and a functional dependency f: Product→Product_type. Let us look at three possible summary-views in the summary-view lattice:

sv1= SV(Product, Product_type, Branch_id, Date)
sv2= SV(Product, Branch_id, Date)
sv3= SV(Product_type, Branch_id, Date)

First, we look at sv1 and sv2. As every Product belongs to only one Product_type, two summary-views have the same number of tuples and hold the same summary information. Only one should be considered for materialization. In this paper, sv1 will be chosen as it can be used to compute all queries answered by sv2.

Note that how sv1 should be materialized will be considered as a separate issue. Whether sv1 should be materialized directly as a flat table or split into sv2 and a small table p=(Product, Product_type), and how these views should be indexed will not be covered in this paper. Here, we concentrate on eliminating redundant summary information for measure attributes, this is, sv1 and sv2 should never be selected at the same time.

Next, we look at sv3, should it be materialized? If sv1 is small, e.g. whole sv1 can be fetched into memory quickly, it may not worth to materialize sv3 separately. All queries running against sv3 can be redirected to sv1, additional I/O cost is negligible. In another case, if there are only few Product belongs to each Product_type, sv1 and sv3 might have almost the same number of tuples. It might be not worth to materialize sv3 also. The benefit gained in query processing might not justify for the storage cost involved.

In the above example, we have made use of two view trimming methods, which can potentially trim out a large percentage of summary-views from a summary-view lattice. In the next sections, we propose these two filtering algorithms formally.

## 3. Summary-View Lattice Trimming

The total number of possible **summy-view** for a MD model with n dimensional attributes will be $2^n$, it is simply impractical to materialize all nodes inside the lattice. We should only materialize those views of the most value for the system performance.

## 3.1 Related works

Most techniques proposed previously concentrate on selecting an appropriate subset of views or indexes based on a given set of user query. Various kinds of heuristics, e.g. greedy, A* algorithm, are used to obtain a near optimal solution [HRU96][GHRU97][G97][YKL97][TS98][BPT97].

In [HRU96], the overall computation cost of the given set queries inside the lattice is considered. By using the Greedy algorithm, the most "valuable" view for the remaining unmaterialized views is picked at each time, until the stop condition is met. In [BPT97], a size filter algorithm is proposed to filter out those summary views with similar number of tuples as their ancestor views.

In the very dynamic OLAP environment, these methods are hard to be implemented.

1) It is very difficult to make good predictions for user queries, especially at the system design stage. If all possible queries were included, the number of possible views will be increased significantly.
2) It is also very difficult to evaluate the size of the views accurately, which are used to estimate the view costs. Before the actual data set is populated, available information is very limited, especially for those expensive big views. After the data is loaded, we still need to do some complex statistics information collection. One of the major burdens is the large number of view need to be evaluated. Sampling methods, e.g. [DNK+97], could be a choice, but its accuracy is depending on the samples. More, data pattern in the OLAP environment is also changing frequently. We need to evaluate view sizes repeatedly.
3) In addition, there is no simple way to verify whether enough views have been selected.
4) At last, various kinds of view selection algorithms are not compatibility with each other. Some useful views selected by one algorithm could be filtered out by another based on different view selection criteria without compensation. It is difficult to use more than one method in the same case.

## 3.2 Functional Dependency Filter

**Definition 3**. Functional Dependency Filter, $\varphi(f)$

Let M be a MD model, $f{:}AL \rightarrow AR$ be a functional dependency where AL and AR are two subsets of dimensional attributes. The functional dependency filter for f, denoted as $\varphi(f)$, is defined as: $\varphi(f)=\{SV(X) \mid AL\subseteq X \wedge AR\not\subseteq X\}$ where X is a subset of dimensional attributes.

**Theorem 1.** For any view SV(X) in $\varphi(f{:}AL\rightarrow AR)$, we can find another summary-view $SV(X\cup AR)$, which is in the summary-view lattice but not in $\varphi(f)$ and holds the same summary information as SV(X). Therefore, all views in $\varphi(f)$ should be filtered out

Proof is included in [QSG].

**Definition 4**. Functional Dependency Filter Set.

Let $F=\{f_1,f_2,\ldots f_n\}$ be a functional dependency set in a MD model M where each $f_i$ is a functional dependency on dimensional attributes. We define the functional dependency filter set for F, denoted as $\varphi(F)$, as a union of $\varphi(f_i)$ where i=1 to n

**Example 3**. Assuming there is a MD model M with four dimensional attributes {A, B, C, D} and $F=\{f_1:A\rightarrow B, f_2:BC\rightarrow D\}$. We can easily get $\varphi(F) = \varphi(f_1)\cup\varphi(f_2)$ = {SV({A}), SV({A,C,D}),SV({A,C}),SV({A,D}),SV({A,B,C}),SV({B,C})} as shown in Fig. 2. Views in $\varphi(F)$ are represented as shaded nodes. Views in bracket below the node are holding the same summary information and replaced by it, e.g. SV({A,B}) is used to replace SV({A}). It is also interesting to look at that derived functional dependency $f_3:AC\rightarrow D$, $\varphi(f_3)$={SV({A,C}),SV({A,B,C})}, $\varphi(f_3)$ is a subset of $\varphi(F)$.

As a next step, we compute a functional dependency filter set for the closure of a set of functional dependencies. There are two problems: completeness and Interference. As there are many derived functional dependencies, will the functional dependency filters for derived functional dependencies further filter out any new summary views? Will the filter sets of functional dependencies in the closure interfere with each other and filter out some useful views?

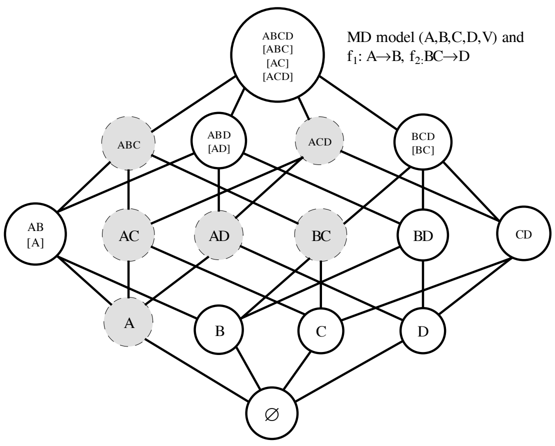

**Figure 2:** Functional Dependency filter

**Theorem 2**. For a set of functional dependencies F in M, $\varphi(F)= \varphi(F^+)$ where $F^+$ is the closure of F.

Proof will be included in [QSG].

Based on the Theorem 2, $\varphi(F)$ has already included all summary-views need to be filtered out in $\varphi(F^+)$ and $\varphi(F)$ is complete. We don't have to worry about the functional dependency filters for functional dependencies derived from F.

If a view SV(X) is filtered out by functional dependency filter $\varphi(f_1:AL_1\rightarrow AR_1)$, then $SV(X\cup AR_1)$, which is not filtered by $\varphi(f_1)$, holds the same summary information as SV(X) , by Theorem 1. Similar, if $SV(X\cup AR_1)$ is filtered out by another functional dependency filter φ($f_2$:$AL_2 \rightarrow AR_2$), then SV($X \cup AR_1 \cup AR_2$), which is not filtered by either φ($f_1$) or φ($f_2$), holds the same summary information as SV($X \cup AR_1$) and SV(X). Similarly, we can prove that for every view filtered out by functional dependency filters, there must be one view remained in the summary-view lattice which holds the same summary information for measure attributes. So we don't have to worry interference among the filter sets for all functional dependencies also.

Hence, we can compute the functional dependency filter set for the functional dependency closure by union the functional dependency filter for each functional dependency inside F at any sequence. Also, assuming F' is a non-redundant covering of F. As $F^+=F'^+$, so φ(F)= φ($F^+$)=φ(F'). Cost to compute φ(F) could be further reduced by computing φ(F') instead.

As functional dependencies are normally available at the system design stage, functional dependency filter can filter out a large number of views in advance before actual data are loaded. Thus, system initialization, system maintenance, and further view selection tasks can be drastically simplified. In addition, we do not have to review the filtered view unless there are changes in the set of functional dependencies.

## 3.3 Size Filter

Not all summary-views are useful for OLAP query optimization. Let the size of a materialized view V be a function: SIZE(V). When SIZE(V) is smaller than a certain value, such as total memory available for the application, it is not a big issue for the OLAP system to process complex queries posted on V within the satisfied response time. So, there is no need to materialize any other summary-view that can be derived from this small view. The extra CPU, I/O cost in query processing to use this slightly bigger view is tolerable. In this paper, we will call the threshold as **Lower_Bound.** In Figure 3, if SV({A,B,C}) is very small and has been materialized, we can directly filter out all tiny views under it in the lattice.

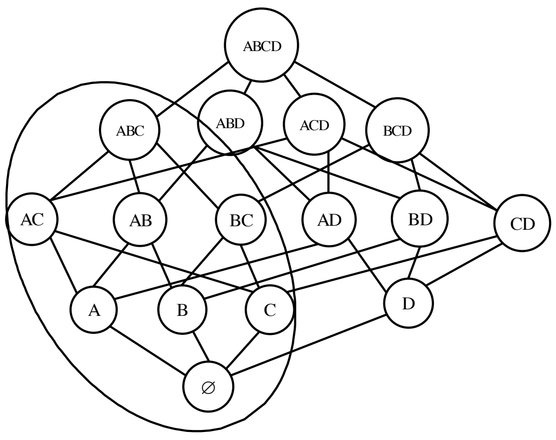

**Fig. 3** Size Filter of Summary-View Lattice

Because of high aggregation, a view with few dimensional attributes in its group-by attribute set will hold similar number of tuples as its domain size, the number of possible distinct tuples in the view. As most of the dimensional attributes have very small domains, e.g. either male or female for human gender, a large number of small views can be filtered out. The OLAP System can be drastically simplified, although the storage space saved might not very significant.

In another case as it described in [BPT97], if a view Y derived from another materialized view X has almost the same number of tuples as X, view Y would be similar to a sorted version of view X, there would be little benefit for us to materialize the view Y. In this paper, we define a threshold value as the **Upper_Bound_Ratio**. If SIZE(Y)/SIZE(X) > Upper_Bound_Ratio, we will filter view Y from the summary-view lattice. In an OLAP system with a large number of dimensional attributes, many summary-views, especially those large views near the top of the summary lattice, could be filtered out by this method.

In the size filtering method, the views need to be processed strictly from the top to the bottom of the summary-view lattice in order to avoid the conflict among different steps of the size filter algorithm.

In the OLAP environment, it is difficult to estimate the view size accurately. However, our size filter is running after applying functional dependency filters, much smaller number of views need to be processed.

Size filter is to achieve a tradeoff between performance and cost. Upper_Bound_Ratio and Lower_Bound are determined based on various factors. Beside machine capabilities, e.g. CPU, Memory, harddisk space and system I/O speed, maintenance cost and user requirement are also very important. For example, we could define a bigger Lower_Bound if queries are asked in the system are simple or a faster harddisk array is available. In the actual system, size filter can still get ride of many views in the summary lattice even with conservatively selected bound values, e.g. 95% as the Upper_Bound_Ratio and 10MB as the Lower_Bound.

## 3.4 Experiments

In this test, data and queries are generated by the program downloaded from the website [OLAP][2]. The system used is a Pentium Celeron 366 with 128MB SDRAM, running Windows NT 4.0 and Oracle 8.04. In this example, we use only one fact table, four dimension tables and eight non-redundant functional dependencies among dimensional attributes (Figure 4). All these data in form of raw text files are about 80MB.

In the test, we assume all summary-views will be used equally. Three view sets are built.

1) The first view set is generated by our algorithms. We apply the functional dependency filtering algorithm on this MD model using 8 FDs and filter out 3680 summary-views out of 4096, or 95.89%. For the size filter, we use 80% as the Upper_Bound_Ratio and 10MB as the Lower_Bound, filter out another 135 views and get 33 views[3].

[2] The parameters used as input to the data generator program are: Channel: 10, Density: 0.1% and Users: 10. Refer the [OLAP] for more detail about the data generator.

[3] To simplify the size estimation problem, we actually count the number of tuples in these 168 views, and assume: VIEW SIZE = Records Length × Number of tuples.

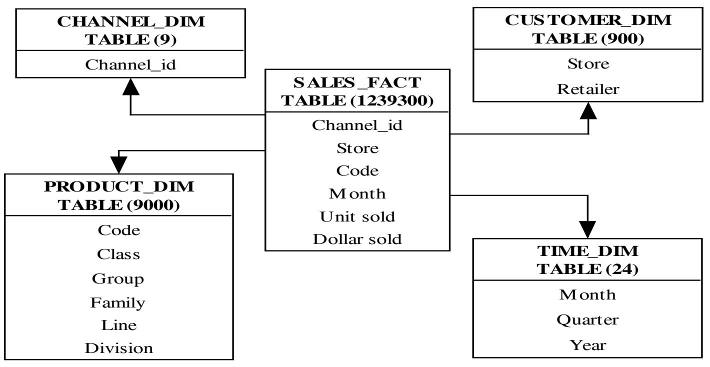

**Fig. 4** Multidimensional model
The number on the right of the table name indicates the number of tuples in that table
FD: Code→Class, Class→Group, Group→Family, Family→Line, Line→Division
Store→Retailer, Month→Quarter, Quarter→Year
*( Year is include as an element in Month, Quarter. e.g. 021997 and Q21997 )

2) The second set is generated by algorithm proposed by [BPT97] running on the 168 views selected by our functional dependency filter.[4] We use 80% as the Upper_Bound_Ratio, get 135 views.
3) The last data set is generated by [HRU96] with Maximum 33 views as the stop condition for the greedy algorithm.

For the benchmark test queries, we use Query Set 1 (Channel Sales Analysis) generated by the data generator program [OLAP] also. There are 2500 queries in this set, we only use the first 500 queries because of the resource limitation. Test results are shown in Table 1.

| Methods | Views selected | Space required | Processing Time (avg) | Queries processed within 3 sec | | Query processed within 3 ~ 20 sec. | | Queries processed more than 20 sec | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | No. of Queries | Processing Time (avg) | No of Queries | Processing Time (avg) | No. of Queries | Processing Time (avg.) |
| **Ours** | 33 | 1.09GB | 9.1 sec. | 312 | 1.16 sec. | 35 | 9.67 sec | 153 | 25.24 sec |
| **[BPT97]*** | 135 | 1.16GB | 9.1 sec. | 312 | 1.16 sec. | 35 | 9.67 sec | 153 | 25.24 sec |
| **[HRU96]** | 33 | 1.93GB | 8.9 sec. | 311 | 1.22 sec | 53 | 12.28 sec | 136 | 25.05 sec |

**Table 1.** Performance Comparison
* [BPT97] is running after applying Functional Dependency Filter

It is clearly that our approach is the overall winner. We have reduced the number of summary-views in the summary-view lattice to 1% of the original size. To achieve a similar performance, we use the same number of views with 56% of storage space required by [HRU96] and only 25% of views required by [BPT97][5]. More testing cases explored in [QSG] shows the similar results.

[4] It is because [BPT97] yields very bad result if it is simply applied on all possible views and also we want to compare the effort of Lower Bound Size Filter.
[5] [BPT97] is running after applying our functional dependency filter

## 4. Conclusion and Future Works

In this paper, we are proposing functional dependency filter and a size filter to filter out a large number of redundant views in the OLAP system with limited cost of query performance. In the test, our algorithms generated an impressive result comparing with those of others.

As one of the future work, we would like to study how to accelerate query which has many dimensional attributes in its where clause and few dimensional attributes in its group-by clause. Summary-view approach may not be a good solution, as heavy aggregations are required to run against a big summary-view. To organize this kind of big views for quick access is still an open problem. Complex indexes [OQ97], Range-Cube [HAMS97], sub-cube and some other complex data structure could be the right direction. Further research in this area is definitely needed.

## Reference:

[BPT97] E.Baralis, S.Paraboschi, E.Teniente. Materialized View selection in a Multidimensional Database, VLDB'97

[DNR+97] P.M.Deshpande, J.F.Naughton, K.Ramasamy, A.Shukla, K.Tufte, Y.Zhao Cubing Algorithms, Storage Estimation, and Storage and Processing Alternatives for OLAP, Bulletin of Technical Committee on Data Engineering Mar 1997, pp. 3 – 11

[G97] H.Gupta, Selection of Views to Materialize in a Data Warehouse, Proc. ICDT'97

[GHRU97] H.Gupta, V.Harinarayan, A.Rajaraman, and J.D.Ullman. Index selection for OLAP. In Proc. ICDE'97, pp. 208 – 219

[HAMS97] C.Ho, R.Agrawal, N.Megiddo, R.Srikant, Range Queries in OLAP Data Cubes SIGMOD'97, pp. 73-88

[HRU96] V.Harinarayan, A.Rajaraman, J.D.Ullman, Implementing Data Cubes Efficiently, ACM SIGMOD '96, pp. 205-216

[KIM97] R.Kimbal A Dimensional Modeling Manifesto AUG 1997 //www.dbmsmag.com

[OQ97] P.O'Neil, D.Quass, Improved Query Performance with Variant Indexes, SIGMOD'97

[OLAP] //www.olapcouncil.org

[QSG] S.G.Qiu, View Selection in OLAP Environment, Thesis for master degree, NUS

[TS98] D.Theodoratos, T.Sellis, Data Warehouse Schema and Instance Design, ER'98

[YKL97] J.Yang, K.Karlapalem, Q.Li, Algorithms for materialized view design in data warehousing environment, Proc. VLDB' 97, pp. 136-145