Optimized Design of Materialized Views in a Real-Life Data Warehousing Environment

Gorettiv K.Y. Chan
Computer & Information Systems
Rix Pumps Limited, Tai Po
Industrial Estate,N.T.,
Hong Kong, China
g.chan@rix.com.hk

Qing Li
Dept of Computer Science
City University of Hong Kong
Tat Chee Ave, Kowloon,
Hong Kong, China
csqli@cityu.edu.hk

Ling Feng
InfoLab, Tilburg University
B 302, PO Box 90153
5000 LE Tilburg
The Netherlands
ling@kub.nl

Abstract

In this paper, we describe the design of a data warehousing system for an engineering company 'R'. This system aims to assist users in retrieving data for business analysis in an efficient manner. The structural design of this data warehousing system employs the dimensional modeling concepts of star and snowflake schemes. Furthermore, frequently accessed dimension keys and attributes are stored in various summary views (materialized views) in order to minimize the query processing cost. A cost model was developed to enable the evaluation of the total cost and benefit involved in selecting each materialized view. Using the cost analysis methodology for evaluation, an adapted greedy algorithm has been implemented for the selection of materialized views. This algorithm takes into account all of the cost variables associated with the materialized views selection method, including query access frequencies, base-data update frequencies, query access costs, view maintenance costs and the availability of the system's storage. The algorithm and cost model have been applied to a set of real-life database items extracted from company 'R'. By selecting the most cost effective set of materialized summary views, the total cost of the maintenance, storage and query processing of the system is optimized, thereby resulting in an efficient data warehousing system.

Keywords: Data warehouse, materialized views selection, query processing cost, storage cost, maintenance cost, query and update frequencies.

1. Introduction

A data warehouse is an information base that stores a large volume of extracted and summarized data for On-Line Analytical Processing and Decision Support Systems [1]. The basic architecture of a data warehousing system given in [2] is shown in Figure 1. To reduce the cost of executing aggregate queries in a data warehousing environment, frequently used aggregates are often pre-computed and materialized into summary views so that future queries can utilize them directly. Undoubtedly, materializing these summary views can minimize query response time. However, if the source data changes frequently, keeping these materialized views updated will inevitably incur a high maintenance cost. Furthermore, for a system with limited storage space and/or with thousands of summary views, we may be able to materialize only a small fraction of the views. Therefore, a number of parameters, including *users' query frequencies, base relation update frequencies, query costs, view maintenance costs* and *the availability of the system's storage,* should be considered in order to select an optimal set of summary views to be materialized.

To motivate the discussion of data warehouse design and materialized view selection, consider a data warehouse which contains the following fact and dimension tables:

INV (Co_no, Inv_no, Inv_date, P_no, Qty, Amt)
CO (Co_no, Co_name, R_no)
PD (P_no, P_name, Mfr_no, Type_no, Cat_no)

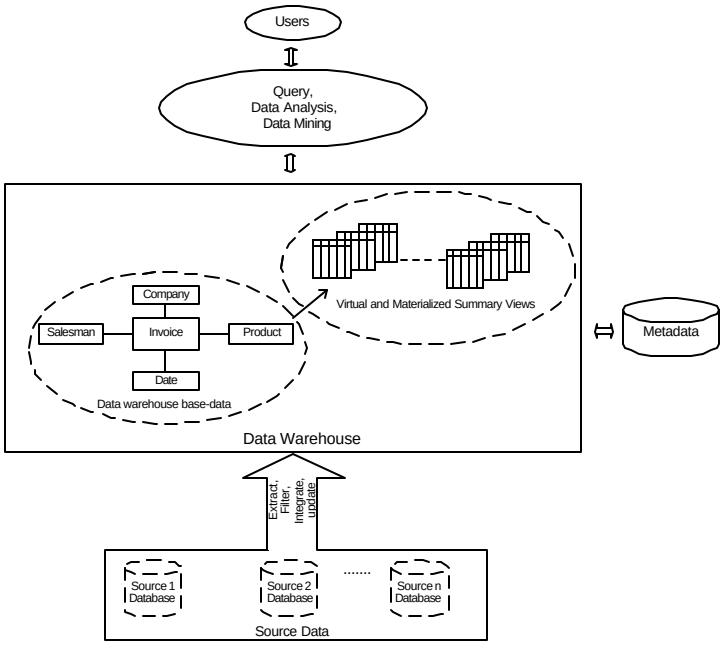

Figure 1: The basic architecture of a data warehousing system

Assume the sizes of the fact and dimension tables 'INV', 'CO' and 'PD' are 114B, 12B and 6B, respectively, where B denotes the data block size which is 2K in the database system (e.g., Oracle). Given a subset of typical user queries as illustrated in Table 1, we can calculate the total cost C_{total} and each cost component (i.e. query processing, maintenance and storage costs) for the following three view materialization strategies:

- the *all-virtual-views* method
- the *all-materialized-views* method
- the *selected-materialized-views* method

Table 5 illustrated the storage cost calculation for summary table 'CO-P-DAY'. Table 2 presents the calculation results, from which we make the following observations: (i) The *all-virtual-views* method requires the highest query processing cost but no view maintenance and storage costs are incurred. (ii) The *all-materialized-views* method can provide the best query performance since this method requires the minimum query processing cost. However, its total maintenance and storage expenses are the highest. (iii) The *selected-materialized-views* method requires a slightly higher query processing cost than the *all-materialized-views* method, but its total cost C_{total} is the least.

Based on the above cost analysis, apparently, the *selected-materialized-views* method is the most effective in terms of both query performance and maintenance cost of data warehousing systems.

Users' Queries	Query Frequency f_{qi}	Summary Views	No. of records in summary table	Size of Summary View (in B)
SELECT INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, CAT_NO, MFR_NO, R_NO, INV_DT, SUM(AMT) AMT, SUM(QTY) QTY FROM INV , CO, PD WHERE INV.CO_NO=CO.CO_NO AND INV.P_NO = PD.P_NO GROUP BY INV.CO_NO,CO_NAME,INV.P_NO,P_NAME,TYPE_NO, INV_DT ORDER BY TYPE_NO	2	CO-P-DAY	3845	240.00
SELECT INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, CAT_NO, MFR_NO, R_NO, TO_CHAR(INV_DT,'MM-YY'), SUM(AMT) AMT, SUM(QTY)QTY FROM INV , CO, PD WHERE INV.CO_NO=CO.CO_NO AND INV.P_NO = PD.P_NO GROUP BY INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, TO_CHAR(INV_DT,'MM-YY') ORDER BY TYPE_NO	1	CO-P-MTH	3560	209.00
SELECT INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, CAT_NO, MFR_NO, R_NO, TO_CHAR(INV_DT,'Q-YY'), SUM(AMT) AMT, SUM(QTY)QTY FROM INV , CO, PD WHERE INV.CO_NO=CO.CO_NO AND INV.P_NO = PD.P_NO GROUP BY INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, TO_CHAR(INV_DT,'Q-YY') ORDER BY TYPE_NO	1	CO-P-QTR	3331	196.00
SELECT INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, CAT_NO, MFR_NO, R_NO, TO_CHAR(INV_DT,'YY'), SUM(AMT) AMT, SUM(QTY) QTY FROM INV , CO, PD WHERE INV.CO_NO=CO.CO_NO AND INV.P_NO = PD.P_NO GROUP BY INV.CO_NO, CO_NAME, INV.P_NO, P_NAME, TYPE_NO, TO_CHAR(INV_DT,'YY') ORDER BY TYPE_NO	1	CO-P-YR	1087	64.00

Table 1: A subset of users' queries. Note f_{qi} denotes the query frequency between every two updates.[1]

	Total query processing cost $Total(C_{qr})$	Total maintenance cost $Total(C_{mT})$	Total storage cost $Total(C_{storeT})$	C_{total} = $Total(C_{qr})$ + $Total(C_{mT})$ + $Total(C_{storeT})$
All-virtual-views	10920	0	0	10920
All- materialized-views	949	2829	709	4487
Selected-materialized-views	1200	2184	240	3624

Table 2: The query, maintenance and storage costs for three view materialization strategies

Recently, materialized view selection problem has sparked vigorous discussions in the database research community. Harinarayan, Rajaraman and Ullman [3] presented a greedy algorithm for the selection of materialized views so that query evaluation costs can be optimized in the special case of

[1] A complete list of annotations used in our case study can be found in the appendix of the paper.

"data cubes". However, the costs for view maintenance and storage were not addressed in this piece of work. Yang, Karlapalem and Li [4] proposed a heuristic algorithm which utilizes a Multiple View Processing Plan (MVPP) to obtain an optimal materialized view selection, such that the best combination of good performance and low maintenance cost can be achieved. However, this algorithm did not consider the system storage constraints. Gupta [5] further developed a greedy algorithm to incorporate the maintenance cost and storage constraint in the selection of data warehouse materialized views. "And-Or" view graphs were introduced to represent all the possible ways to generate warehouse views such that the best query path can be utilized to optimize query response time. In this paper, we discuss our experiences in designing and selecting appropriate materialized views for data warehousing systems. The latest dimensional modeling methodologies are applied in our case study to design an efficient data warehousing system for an engineering company 'R'. The greedy algorithm presented by Gupta [5] has been adopted and modified for the selection of materialized views. A cost model was developed to enable the evaluation of the total cost and benefit involved in selecting each materialized view. We apply the algorithm and cost model to a set of real-life database items extracted from this company. Due to the constraints in data storage and computational costs involved, a subset of sales records (i.e. the yearly sales records of 1996) was adopted to estimate the size of each summary view. Based on the cost analysis, a set of materialized views are selected to optimize the total cost including the query, maintenance and storage costs of the warehousing system. This view selection methodology was tested and proved to be very cost effective for the optimization of the data warehouse. General guidelines for data warehouse design and materialized views selection based on this work are presented.

The reminder of the paper is organized as follows. Section 2 describes the schema design of the data warehousing system. The cost model and adapted greedy algorithm for the selection of materialized views are presented in section 3. In section 4, various view materialization strategies are analyzed and their performances are tested. Guidelines for the design and selection of materialized views for data warehousing systems are discussed in section 5. Section 6 concludes the paper with a brief discussion of future work.

2. Data Warehouse Design

In this section, the application characteristics and performances of star, fact constellation and snowflake schemes [6] are reviewed and discussed, based on which the benefit of integrating these schemes into the design of our data warehousing system is then examined.

2.1 Star, fact constellation and snowflake schemes

2.1.1 Star schema

The two major components of the star schema are fact and dimension tables, as shown in Figure 2. The center of this star schema is represented by the fact table 'INV' and the points of the star schema are represented by dimension tables. The attributes of these dimension tables can often be organized into hierarchies [7].

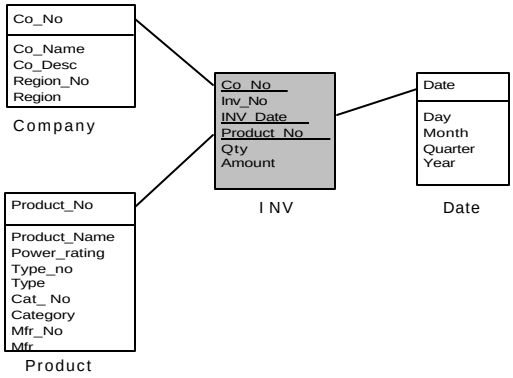

Figure 2: The star schema for data warehousing systems

In order to maintain a simple data structure, the fact table of this star schema keeps both base-data and summarized data, while the dimension tables are un-normalized. As a result, the number of joins required for processing each query can be effectively reduced. However, when large volumes of new data and pre-calculated data are added to the data warehouse, the fact table can become extremely large. Likewise, if the un-normalized dimension tables have many records, their sizes will increase significantly because of the repeated attribute values. Thus, large disk storage and long query processing time will be required due to the large data quantity in the fact and dimension tables.

2.1.2 Fact constellation schema

The fact table of the fact constellation schema is partitioned horizontally according to the group-by attributes in order to reduce query processing time, as shown in Figure 3. Furthermore, the amount of summarized data to be materialized can be adjusted according to the users' query frequencies and the system's storage constraint. The main disadvantages of this design are: 1) the structure of the fact constellation schema is more complex. Therefore, it is very difficult to maintain a large number of summary views; and 2) various views may have to be accessed in order to process one user's query.

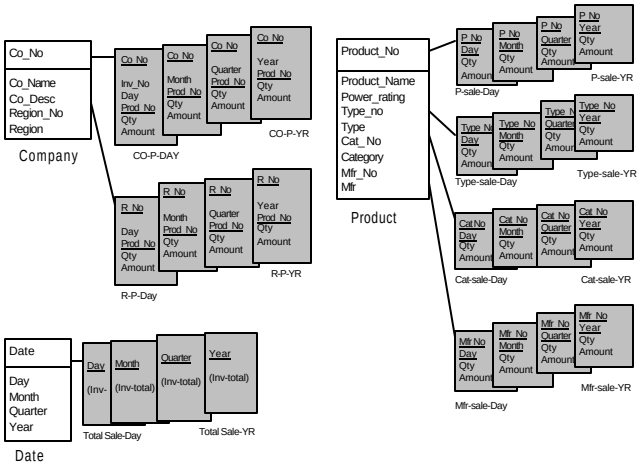

Figure 3: The fact constellation schema for data warehousing systems

2.1.3 Snowflake schema

The snowflake schema was developed in an attempt to further improve query performance by normalizing the dimension tables since smaller dimension tables can effectively reduce the cost of join operations. This schema is illustrated in Figure 4. The main disadvantages of this design are: 1) it is a very complex data structure, with many summary views and normalized dimension tables; and 2) various summary views and dimension tables may also need to be accessed in order to process one user's query.

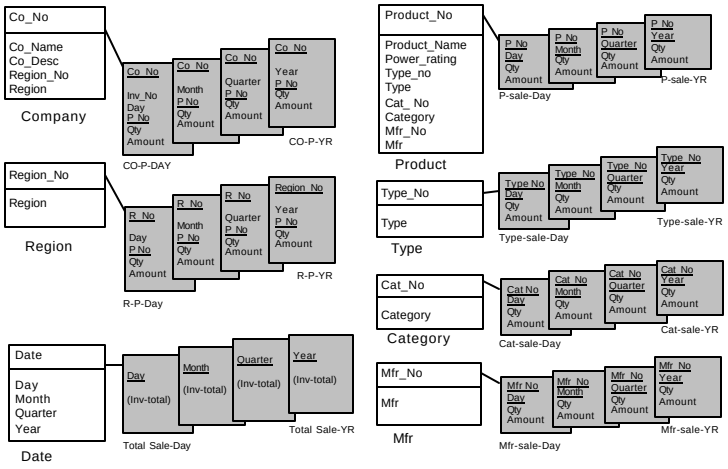

Figure 4: The snowflake schema for data warehousing systems

2.2 The application requirements of company `R'

The database system of company 'R' actually contains a large number of tables. However, in this case study, analysis is conducted mainly on the sales data of the invoice database system. The ER diagram of this system is illustrated in Figure 5. The data warehousing design methodology developed based on this simplified data model can be easily incorporated into the company's present system.

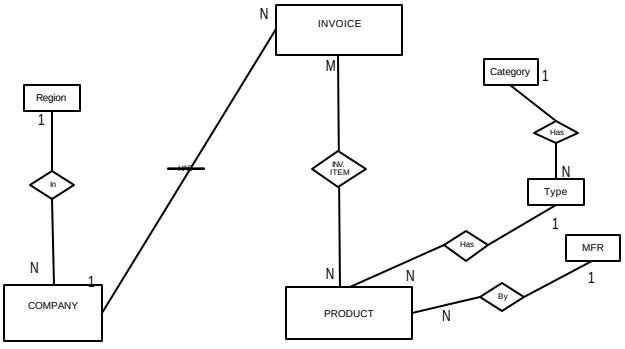

Figure 5: ER diagram for the invoice database system

After the user requirements were collected, it was found that the measures of sales in relation to each dimension attribute needed to be analyzed. The measures of sales are in terms of the amount charged for each invoice item and the quantity of products sold. The information required by users can be obtained by aggregating the sales data with various dimension attributes, including Co_no and R_no of the Company dimension, Product_no, Type_no, Category_no and Mfr_no of the Product dimension, and day, month, quarter and year of the Date dimension. For example, the summary view 'Co-P-Yr' is calculated by summing up numbers of each product sold to different companies in a year. Furthermore, these aggregated sales data will be sorted by various dimension attributes for generating reports.

The summary views which aggregate sales data with various group-by attributes and the estimated query frequencies between every two updates are listed in Table 3. Table 4 gives the notations used in these summary views.

Summary views generated by users' queries	Query Frequency f_{qi}	Summary views generated by users' queries	Query Frequency f_{qi}
CO-P-DAY	2	CO-SALE-DAY	5
CO-P-MTH	1	CO-SALE-MTH	5
CO-P-QTR	1	CO-SALE-QTR	5
CO-P-YR	1	CO-SALE-YR	5
R-P-DAY	1	R-TYPE-DAY	0.5
R-P-MTH	1	R-TYPE-MTH	0.5
R-P-QTR	1	R-TYPE-QTR	0.5
R-P-YR	1	R-TYPE-YR	0.5
P-SALE-DAY	6	TYPE-SALE-DAY	1
P-SALE-MTH	6	TYPE-SALE-MTH	1
P-SALE-QTR	6	TYPE-SALE-QTR	1
P-SALE-YR	6	TYPE-SALE-YR	1
CO-MFR-DAY	0.5	R-CAT-DAY	0.5
CO-MFR-MTH	0.5	R-CAT-MTH	0.5
CO-MFR-QTR	0.5	R-CAT-QTR	0.5
CO-MFR-YR	0.5	R-CAT-YR	0.5
R-MFR-DAY	1	CAT-SALE-DAY	2
R-MFR-MTH	1	CAT-SALE-MTH	2
R-MFR-QTR	1	CAT-SALE-QTR	2
R-MFR-YR	1	CAT-SALE-YR	2
MFR-SALE-DAY	3	R-SALE-DAY	2
MFR-SALE-MTH	3	R-SALE-MTH	2
MFR-SALE-QTR	3	R-SALE-QTR	2
MFR-SALE-YR	3	R-SALE-YR	2
CO-TYPE-DAY	1	TOT-SALE-DAY	3
CO-TYPE-MTH	1	TOT-SALE-MTH	3
CO-TYPE-QTR	1	TOT-SALE-QTR	3
CO-TYPE-YR	1	TOT-SALE-YR	3
CO-CAT-DAY	1	CO-CAT-QTR	1
CO-CAT-MTH	1	CO-CAT-YR	1

Table 3: Summary views generated by users' queries and related query frequencies

Notations in summary views	Group-by attributes	Names of group-by attributes
Co	Co_no	Company_number
R	R_no	Region_number
P	P_no	Product_no
Mfr	Mfr_no	Manufacturer_no
Type	Type_no	Type_no
Cat	Cat_no	Category_no
Day	Inv_dt	Invoice_date
Mth	To_char(Inv_dt,'mm-yy')	Invoice_date
Qtr	To_char(Inv_dt,'Q-yy')	Invoice_date
Yr	To_char(Inv_dt,'yy')	Invoice_date
Amt	Amt	Amount
Qty	Qty	Quantity

Table 4: Notations in summary views

2.3 Storing frequently accessed dimension keys and attributes in the summary views

Long query processing time is required for joining large fact and dimension tables. However, when the frequently accessed dimension keys and attributes are stored into summary views, the number of joins and query processing time can be effectively reduced. The storage cost, benefit and storage effectiveness associated with adding various frequently accessed dimension keys and attributes into the fact table 'INV' are calculated and listed in Table 5. The cost analysis here assumes that the difference in maintenance cost is negligible. The query frequencies used for calculating the total query cost are listed in Table 3. The set of dimension attributes (P_name, Co_name, R_no, Type_no, Cat_no, and Mfr_no) are chosen for storage in 'CO-P-DAY' and various summary views to speed up query processes, since this set yields the greatest benefit (i.e., total query cost savings) and storage effectiveness. Detailed calculations of benefit and effectiveness in our case study will be given in Section 3.1.4.

Fact table	Table size (in B)	Extra storage space (in B)	Total Cost for evaluating 115 queries (in B)	Benefit (in B)	Effectiveness (in B)
'INV' (Co_no, Inv_no, Inv_date, P_no, Qty, Amt)	114		126270.00		
The following dimension keys and attributes are added to the fact table 'INV'					
X ={R_no, Type_no, Cat_no, Mfr_no}	154	40	99754.60	26515.40	662.89
{P_name} ∪ X	197	83	107858.36	18411.64	221.83
{Co_name} ∪ X	197	83	63902.36	62367.64	751.42
{P_name, Co_name} È X	**240**	**126**	**30087.12**	**96182.88**	**763.36**
{Region, P_name, Co_name}∪ X	269	155	33478.40	92791.60	598.66
{Type, P_name, Co_name} ∪ X	280	166	33779.22	92490.78	557.17
Category, P_name, Co_name} ∪ X	280	166	34785.20	91484.80	551.11
{Manufacturer, P_name, Co_name} ∪ X	280	138	34088.32	92181.68	667.98

Table 5 : The costs of adding various dimension keys and attributes to the fact table 'INV'. (X represents a set of dimension keys { R_no, Type_no, Cat_no, Mfr_no}.)

2.4 System design and characteristics

In our case study, the data warehouse structural design is based on a combination of star and snowflake schemes. Source data from the invoice database is integrated into the data warehousing system to form its base-data. These base-data are stored in fact and dimension tables in the form of a star schema. The fact table is horizontally partitioned into many summary views. Furthermore, frequently accessed dimension keys and attributes are stored in the summary views so that query processing costs can be reduced. The dimension tables are normalized according to various dimension attributes in order to reduce the table sizes. The sizes of these normalized tables are listed in Table 6. Figure 6 illustrates the hybrid schema of the data warehousing system for company 'R'.

Dimension tables	Record number	Attributes in dimension table	Table size $S(V_i)$ (in B)
CO	760	Co_no, Co_name, R_no	12
COMPANY	760	Co_no, Co_name, Co_desc, R_no, Region	45
REGION	4	R_no, Region	0.045
PRODUCT	190	P_no, P_name, P_desc, Power_rating, Type_no, Type, Cat_no, Cat, Mfr_no, Mfr	102
PD	190	P_no, P_name, Mfr_no, Type_no, Cat_no	6
PD_MFR	14	Mfr_no, Mfr	0.2
PD_TYPE	31	Type_no, Type, Cat_no	0.47
PD_CAT	6	Cat_no, Cat	0.08
P_MFR	190	P_no, Mfr_no, Mfr	3
P_TYPE	190	P_no, Type_no, Type, Cat_no	3
P_CAT	190	P_no, Cat_no, Cat	3

Table 6: Sizes of different dimension tables

3. Materialized Views Selection

We now move on to address the related issue of data warehouse design for our case study, namely, the selection of summary views to be stored/materialized in the data warehouse. Benefits of materializing summary views selectively have been articulated in the literature [5, 8]. For our case study, a cost model is established to enable the evaluation of query cost, maintenance cost, storage cost and benefit associated with materializing each summary view in the data warehouse. An adapted greedy algorithm using the cost analysis methodology for evaluation is then presented for selecting an optimal set of materialized views.

3.1 Cost model

The estimated query, maintenance and storage costs in the following descriptions will be calculated in terms of data block size B. For simplicity, other factors such as computational cost and communication cost are ignored in our estimation.

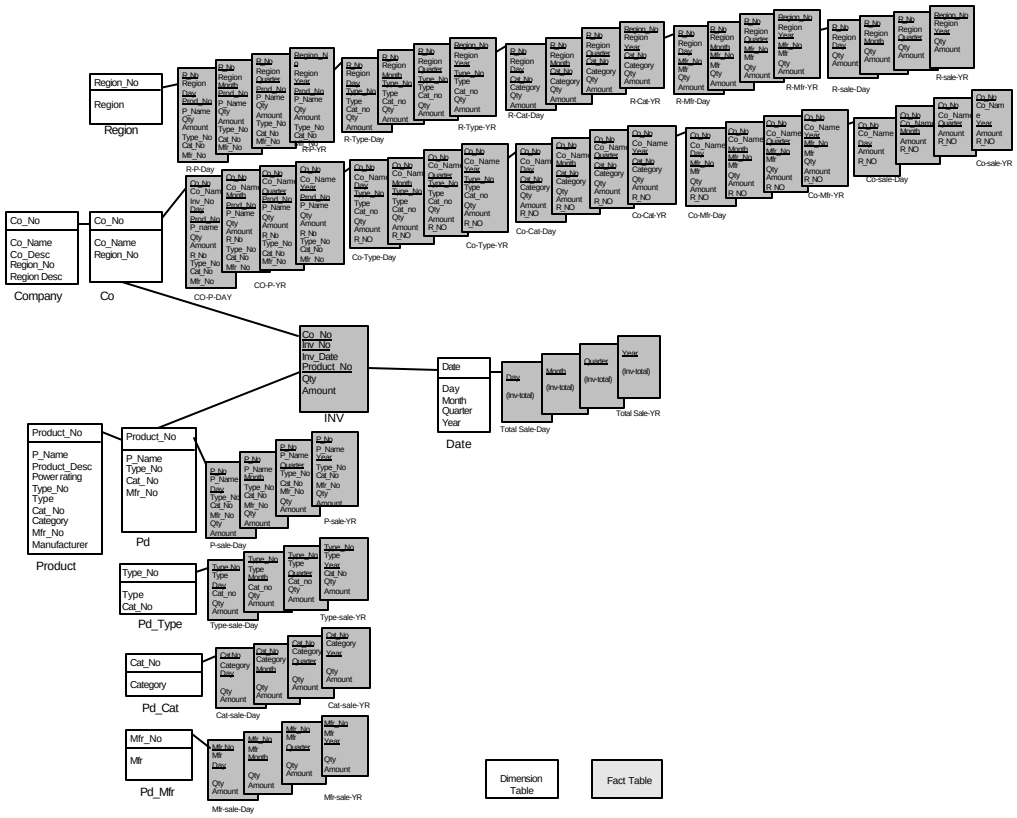

Figure 6: The hybrid schema of the data warehousing system for company 'R'

3.1.1 Query processing cost for selection, aggregation and joining

The analysis assumes that there is no index or hash key in any of the summary views, therefore linear search and nested loop approach are used for the selection and join operations, respectively. In the worst case, the analysis estimates that all the records in a summary view will be scanned once in order to process one user's query which involves selection and aggregation. Thus, to access a summary view V_i, the estimated query processing cost in terms of block access is equal to the size of V_i, i.e.,

$$C_g(V_i) = S(V_i)$$

The estimated query cost involving the joining of n dimension tables [9] with the summary view V_i is:

$$C_j(V_{d1}, V_{d2}, \ldots, V_{dn}, V_i) = (S(V_{d1}) + S(V_{d1}) * S(V_i)) + (S(V_{d2}) + S(V_{d2}) * S(V_i)) + \ldots + (S(V_{dn}) + S(V_{dn}) * S(V_i))$$

To process a user's query q_i, which requires not only selection and aggregation of the summary view V_i, but also the joining of V_i with other dimension tables, the query cost $C_q(q_i)$ is:

$$C_q(q_i) = C_g(V_i) + C_j(V_{d1}, V_{d2}, ..., V_{dn}, V_i)$$
$$= S(V_i) + (S(V_{d1}) + S(V_{d1}) * S(V_i)) + (S(V_{d2}) + S(V_{d2}) * S(V_i)) + ... +$$
$$(S(V_{dn}) + S(V_{dn}) * S(V_i))$$

$$Total\ (C_{qr}) = \sum_{i=1}^{r} f_{qi} * C_q(q_i)$$

Thus, the total query cost $Total(C_{qr})$ for processing r users' queries between every two updates is:

3.1.2 Data warehouse maintenance cost

As the source data set of the invoice system is subject to constant changes, these changes are recorded by a set of auxiliary views and conveyed to the 'INV' fact table and dimension tables during the maintenance window [10]. Based on the updated 'INV' fact table and dimension tables, all the sales summary views within the data warehouse need to be re-computed. There are usually many ancestor views from which a sales summary view can be evaluated. However, the best choice is the smallest possible ancestor view that requires the least number of joining processes for the computation of its descendant views. As illustrated in Figure 7, the summary view of product monthly sales 'P-sale-Month' can be evaluated by aggregating the summary view of either the regional product monthly sales 'R-P-Month' or the product daily sales 'P-sale-Day'. However, aggregating 'R-P-Month' will result in the least query cost. The optimal query and maintenance paths for these summary views are illustrated in Figure 7.

Assume that re-computation of each summary view V_i requires selection and aggregation from its ancestor summary view V_{ai}, and the joining of V_{ai} with n dimension tables V_{d1}, V_{d2}, ..., V_{dn}. Thus, the cost for re-computing summary view V_i can be calculated by :

$$C_m(V_i) = C_g(V_{ai}) + C_j(V_{d1}, V_{d2}, ..., V_{dn}, V_{ai})$$
$$= S(V_{ai}) + (S(V_{d1}) + S(V_{d1}) * S(V_{ai})) + (S(V_{d2}) + S(V_{d2}) * S(V_{ai})) + ... +$$
$$(S(V_{dn}) + S(V_{dn}) * S(V_{ai}))$$

If there are j summary views in the warehouse which are materialized, the total maintenance cost $Total(C_m)$ for these materialized views is then:

$$Total\ (C_m) = \sum_{i=1}^{j} f_{ui} * C_m(V_i)$$

where f_{ui} is the update frequency of summary view V_i. In our case study, we assume that all sales summary views are updated once within a fixed time interval, thus $f_{ui} = 1$ for any $1 < i < j$.

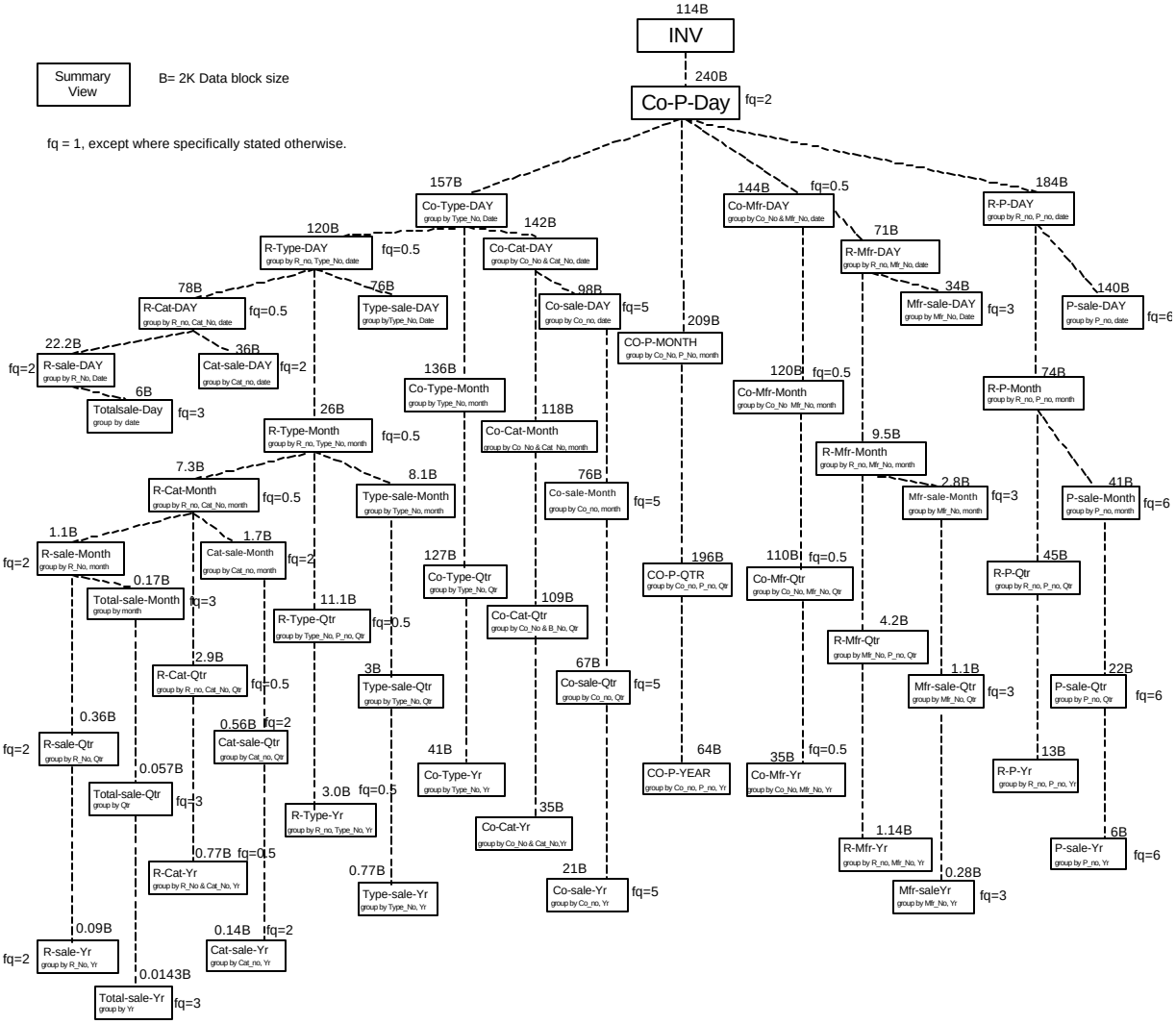

Figure 7: The optimal query and maintenance paths for sales summary views.

3.1.3 Storage cost

The analysis estimates that the cost for storing materialized views depends on the availability of hard disk space within the data warehousing system. The storage factor *U* represents the estimated ratio of the storage capacity required by the data warehouse to the availability of hard disk space:

$$U = (Total(C_{store}) + (1+Q) * Y * S_a) \ / Total\ available\ storage\ capacity$$

where '$(1+Q) * Y * S_a$' estimates the total increase in storage capacity for accommodating new data during the estimated life cycle of the data warehouse. Here, '*Q*' denotes the estimated increase rate in data volume per year within the data warehouse, '*Y*' denotes the estimated life cycle of the data

warehouse, and 'S_a' denotes the storage space required to store yearly added new data and their summarized data.

The storage cost of summary view V_i in terms of data block B is:

$C_{store}(V_i) = U * S(V_i)$

In our case study, $U = 1$, meaning that storage space is readily available for storing materialized views, thus, $C_{store}(V_i) = S(V_i)$.

3.1.4 The net benefit and storage effectiveness

In order to determine an optimal set of materialized summary views, the net benefit $Net(B_i)$ and the storage effectiveness h_i (i.e. the net benefit per unit of storage space occupied by a materialized view) associated with each summary view (cf. Figure 7) need to be calculated.

i) The Net Benefit of materializing view V_i can be calculated using the following formula:

Net Benefit = Benefit – Maintenance cost – Storage cost

where *benefit* measures the total query cost savings that a materialized summary view brings to all its descendent views, i.e.,

$$B_i = \sum_{n=1}^{m} f_q(V_{ni}) * [C_t(V_{ni} \leftarrow V_{ai}) - C_t(V_{ni} \leftarrow V_i)]$$

Here, V_{ni} ($1 < n < m$) represents one of the descendent views of V_i, and m denotes the total number of descendent views of V_i. $f_q(V_{ni})$ is the query frequency of V_{ni}. For simplicity, we use C_t ($V_{ni} \neg V_{ai}$) to denote the cost of accessing V_{ni} from V_{ai}, the ancestor of V_i, in case that V_i has not been materialized, and $C_t(V_{ni} \neg V_i)$ the cost of accessing V_{ni} from V_i directly.

Therefore, we can calculate and get the net benefit for a materialized view V_i, as follows:

$$Net(B_i) = B_i - C_m(V_i) - C_{store}(V_i)$$
$$= \{\sum_{n=1}^{m} f_q(V_{ni}) * [C_t(V_{ni} \leftarrow V_{ai}) - C_t(V_{ni} \leftarrow V_i)]\} - C_m(V_i) - C_{store}(V_i)$$

ii) The storage effectiveness of summary view V_i can be obtained by the formula:

$$h_i = Net(B_i) / S(V_i)$$

Table 7 lists the storage effectiveness h_i, net benefit $Net(B_i)$, storage cost $C_{store}(V_i)$, maintenance cost $C_m(V_i)$, and query frequencies f_{qi} of summary view V_i. These views are sorted in a descending order of storage effectiveness. For easy explanation, we name an ordered sequence of views using $V_1, V_2, \ldots, Vi$ without loss of generality. In other words, h_{i+1} h_i for any $i > 0$.

The C_{total} in Table 7 is the total cost in the presence of one or more materialized views. It includes the total query cost for processing *r* number of users' queries, total maintenance cost and total storage cost for all the materialized views.

Let $Total(C_{qall})$ denote the total cost for processing *r* number of users' queries when no views are materialized in the data warehouse. Initially, $C_{total} = Total(C_{qall})$. Each time, when a view *Vi* is materialized, we calculate and obtain a new value for C_{total} , i.e.,

$$C_{total} = C_{total} - B_i + C_m(V_i) + C_{store}(V_i).$$

Based on i), Net $(B_i) = B_I - C_m(V_i) + C_{store}(V_i)$, we have $C_{total} = C_{total} - Net(B_i)$.

Hence, after materializing a series of views $V_1, V_2, \ldots, V_i$, the total cost will become

$$C_{total} = Total(C_{qall}) - \sum_{x=1}^{i} Net(B_x)$$

The minimum value of the total cost C_{total} obtained is underlined in Table 7. As shown, as long as the $Net(B_i)$ of view V_i is greater than 0, the total cost C_{total} will continuously decrease after materializing summary view V_i.

	Cost evaluation for one materialized view							
	V_i Summary View	h	B_i	$Net(B_i)$	$C_{store}(V_i)$	$C_m(V_i)$	f_{qi}	Total cost C_{total}
		Storage Effectiveness	Benefit	Net Benefit	Storage Cost	Maintenance Cost	Query Frequency	
1	CO-P-DAY	396.96	97694.88	95270.88	240	2184	2	30999.12
2	R-TYPE-MTH	92.09	2540.40	2394.40	26	120	0.5	28604.72
3	R-SALE-MTH	76.91	93.00	84.60	1.1	7.3	2	28520.12
4	R-MFR-MTH	69.21	738.00	657.50	9.5	71	1	27862.62
5	R-CAT-MTH	55.00	436.95	401.49	7.3	28.16	0.5	27461.13
6	TOT-SALE-MTH	41.76	8.37	7.10	0.17	1.1	3	27454.03
7	CO-TYPE-DAY	32.33	5586.48	5076.21	157	353.27	1	22377.82
8	R-P-MTH	27.73	2310.00	2052.00	74	184	1	20325.82
9	MFR-SALE-MTH	17.14	60.30	48.00	2.8	9.5	3	20277.82
10	CO-MFR-DAY	15.14	2612.88	2180.68	144	288.2	0.5	18097.14
11	CAT-SALE-MTH	14.47	33.60	24.60	1.7	7.3	2	18072.54
12	R-MFR-DAY	13.70	1194.10	972.58	71	150.525	1	17099.97
13	P-SALE-MTH	11.68	594.00	479.00	41	74	6	16620.97
14	P-SALE-YR	11.33	96.00	68.00	6	22	6	16552.97
15	R-SALE-DAY	8.05	279.00	178.80	22.2	78	2	16374.17
16	TOT-SALE-QTR	7.91	0.68	0.45	0.057	0.17	3	16373.71
17	P-SALE-QTR	7.50	228.00	165.00	22	41	6	16208.71
18	CO-SALE-YR	6.76	230.00	142.00	21	67	5	16066.71
19	CO-SALE-DAY	6.53	880.00	640.00	98	142	5	15426.71
20	R-P-DAY	6.39	1611.38	1176.54	184	250.845	1	14250.18
21	R-TYPE-DAY	5.99	1002.92	718.81	120	164.11	0.5	13531.37
22	MFR-SALE-QTR	5.73	10.20	6.30	1.1	2.8	3	13525.07
23	R-SALE-QTR	4.17	2.96	1.50	0.36	1.1	2	13523.57

24	CAT-SALE-QTR	4.11	4.56	2.30	0.56	1.7	2	13521.27
25	TOT-SALE-YR	3.97	0.13	0.06	0.0143	0.057	3	13521.21
26	MFR-SALE-YR	3.86	2.46	1.08	0.28	1.1	3	13520.13
27	TOT-SALE-DAY	3.40	48.60	20.40	6	22.2	3	13499.73
28	TYPE-SALE-MTH	2.42	53.70	19.60	8.1	26	1	13480.13
29	CO-SALE-MTH	2.05	330.00	156.00	76	98	5	13324.13
30	R-CAT-DAY	1.46	321.20	113.52	78	129.68	0.5	13210.61
31	R-SALE-YR	1.00	0.54	0.09	0.09	0.36	2	13210.52
32	CAT-SALE-YR	1.00	0.84	0.14	0.14	0.56	2	13210.38
33	CO-CAT-DAY	0.70	410.56	98.92	142	169.64	1	13111.46
34	MFR-SALE-DAY	0.18	111.00	6.00	34	71	3	**13105.46**
35	TYPE-SALE-QTR	-0.30	10.20	-0.90	3	8.1	1	13106.36
36	P-SALE-DAY	-0.43	264.00	-60.00	140	184	6	13166.36
37	R-MFR-QTR	-0.74	10.60	-3.10	4.2	9.5	1	13169.46
38	CO-SALE-QTR	-0.79	90.00	-53.00	67	76	5	13222.46
39	CAT-SALE-DAY	-0.83	84.00	-30.00	36	78	2	13252.46
40	R-P-QTR	-1.36	58.00	-61.00	45	74	1	13313.46
41	CO-CAT-MTH	-1.59	72.00	-188.00	118	142	1	13501.46
42	CO-TYPE-MTH	-1.69	63.00	-230.00	136	157	1	13731.46
43	CO-P-MTH	-1.70	93.00	-356.00	209	240	1	14087.46
44	CO-MFR-MTH	-1.90	36.00	-228.00	120	144	0.5	14315.46
45	CO-CAT-QTR	-1.92	18.00	-209.00	109	118	1	14524.46
46	CO-TYPE-QTR	-1.93	18.00	-245.00	127	136	1	14769.46
47	CO-P-QTR	-1.93	26.00	-379.00	196	209	1	15148.46
48	R-MFR-YR	-2.00	3.06	-2.28	1.14	4.2	1	15150.74
49	R-CAT-QTR	-2.00	4.40	-5.80	2.9	7.3	0.5	15156.54
50	CO-P-YR	-2.00	132.00	-128.00	64	196	1	15284.54
51	R-P-YR	-2.00	32.00	-26.00	13	45	1	15310.54
52	CO-MFR-QTR	-2.00	10.00	-220.00	110	120	0.5	15530.54
53	CO-TYPE-YR	-2.00	86.00	-82.00	41	127	1	15612.54
54	CO-CAT-YR	-2.00	74.00	-70.00	35	109	1	15682.54
55	R-TYPE-QTR	-2.00	14.90	-22.20	11.1	26	0.5	15704.74
56	TYPE-SALE-DAY	-2.00	44.00	-152.00	76	120	1	15856.74
57	TYPE-SALE-YR	-2.00	2.23	-1.54	0.77	3	1	15858.28
58	CO-MFR-YR	-3.07	37.50	-107.50	35	110	0.5	15965.78
59	R-TYPE-YR	-3.35	4.05	-10.05	3	11.1	0.5	15975.83
60	R-CAT-YR	-3.38	1.07	-2.61	0.77	2.9	0.5	15978.44

Table 7: Cost evaluation for selecting an optimal set of materialized views. The minimum value of $C_{total}(V_i)$ is underlined.

3.2 Adapted greedy algorithm for materialized summary view selection

Let T be the set of all sales summary views involved in users' queries, and $|T|$ be the number of sales summary views in T. Based on the greedy algorithm of [5], we develop an adapted greedy algorithm for determining an optimal set of materialized summary views L, a subset of T, such that the total cost C_{total} is minimized. The algorithm is based on the cost model presented in Section 3.1.

Materialized views selection algorithm:

1. Determine the optimal query and maintenance paths for computing all summary views in the data warehouse (as illustrated in Figure 7).

2. Calculate the net benefit and storage effectiveness for each summary view in T.

   foreach V_i in T **do**

   Calculate the *Net* (B_i) and storage effectiveness h_i of V_i :

$$Net\ (B_i) = B_i - C_m(V_i) - C_{store}(V_i)$$

$$= \{\sum_{n=1}^{m} f_q(V_{ni}) * [C_t(V_{ni} \leftarrow V_{ai}) - C_t(V_{ni} \leftarrow V_i)]\} - C_m(V_i) - C_{store}(V_i)$$

   $h_i = Net\ (B_i) / S(V_i)$

   endfor.

3. Sort the summary views in T in a descending order of storage effectiveness such that those views with the best storage effectiveness will be chosen first (as shown in Table 7). Without loss of generality, assume $V_1, V_2, \ldots, V_{|T|}$ represents an ordered list of summary views.

4. Calculate the total cost C_{total} when a summary view is materialized, and get $Min(C_{total})$ as the optimal cost for materialized views selection.

   $C_{total} = Total(C_{qall})$; /* Initially, no views are materialized. */
   for (i = 1; i ≤ $|T|$; i ++) **do**
   if ($Net\ (B_i) > 0$) **then** $C_{total} = C_{total} - Net\ (B_i)$;
   else break; /* The minimum C_{total} has been found, exit *for* loop. */
   $Min(C_{total}) = C_{total}$.

5. Select the best materialized view set L.

   $L = \varnothing$;
   $C_{total} = Total(C_{qall})$;
   for (i = 1; i ≤ $|T|$; i ++) **do**
   select V_i from the summary view set $T - L$ with the highest storage effectiveness;
   if ($S(L) + S(V_i) < S$) and ($C_{total} - Net\ (B_i) > Min(C_{total})$) **then**
   $L = L \cup \{V_i\}$;
   else break;
   endfor;
   return L.

Figure 8 shows the set L of optimal materialized views (shadowed boxes) thus chosen.

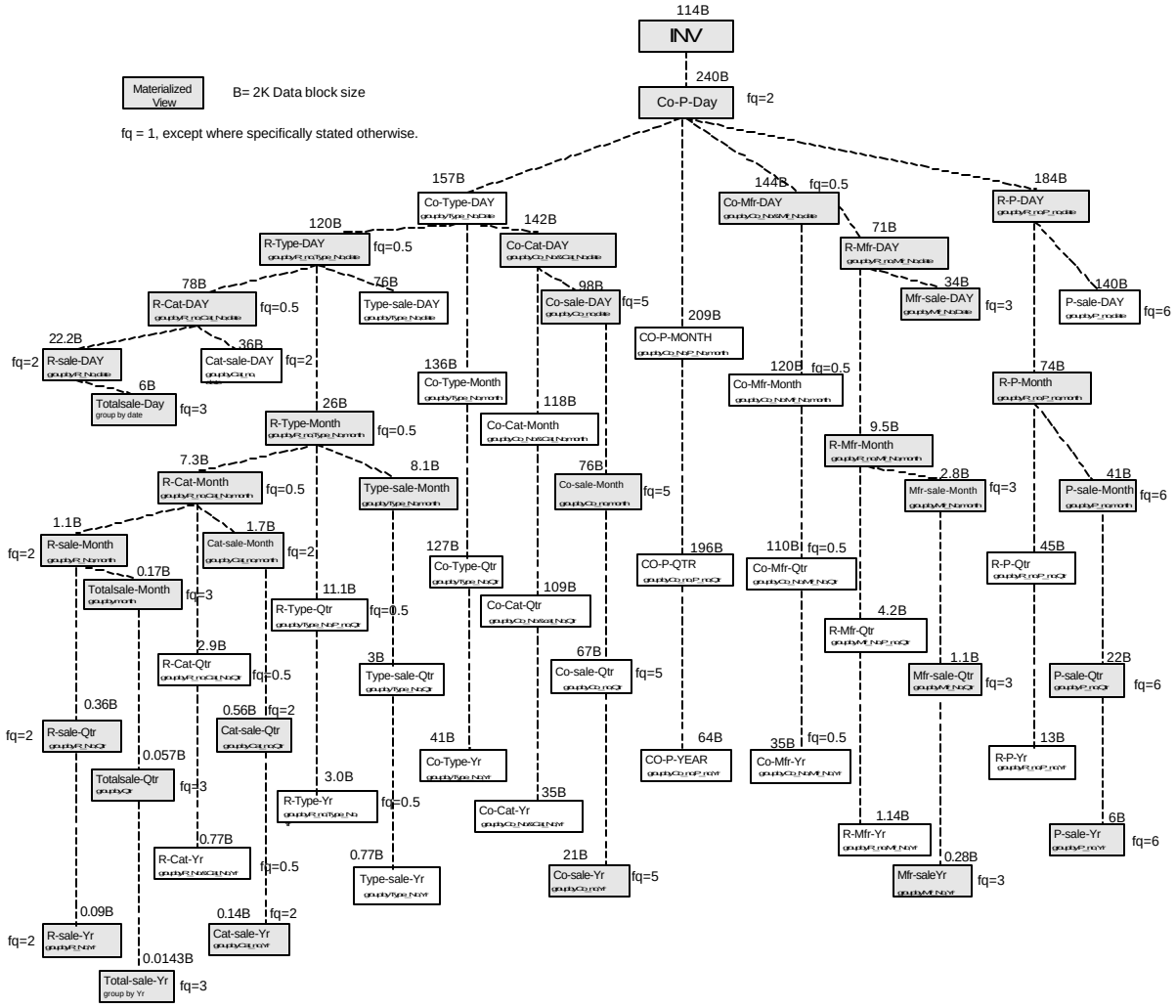

Figure 8: The set of optimum materialized summary views and their query paths

3.3 Cost analysis

The summary views to be materialized are sorted in a descending order according to the corresponding storage effectiveness h_i listed in Table 7. The top thirty-four summary views listed in this table constitute the optimal set of materialized views L. The total cost C_{total} and cost components versus overall storage size of the materialized views are plotted in Figure 9.

We observe that the C_{total} is dominated by the $Total(C_{qr})$ before reaching the optimum point. This optimal point occurs at a cost of 13105.46B and is designated as the minimum total cost $Min(C_{total})$. The $Total(C_{qr})$ drops drastically after materializing the first summary view 'CO-P-DAY', reducing by more than 75% while utilizing only 15% of the total storage space required by the set of optimal materialized views L. Therefore, materializing summary view 'CO-P-DAY' is very cost effective for

improving the query performance of the data warehouse. After this first view has been chosen, there is little reduction in the $Total(C_{qr})$ when more summary views are materialized.

The sum of total maintenance and storage costs, $C_m(V_i)+C_{store}(V_i)$, increases linearly as the number of materialized summary views increases. However, its magnitude is relatively small compared with the $Total(C_{qr})$ before reaching the optimum point $Min(C_{total})$. After reaching this optimal point, C_{total} is dominated by the sum $C_m(V_i)+C_{store}(V_i)$. This is because materializing additional summary views (i.e. summary views with negative net benefit $Net(B_i)$) beyond the optimal point $Min(C_{total})$ cannot reduce query cost, but increases the storage and maintenance costs. Therefore, it is not cost effective to materialize additional views after reaching $Min(C_{total})$.

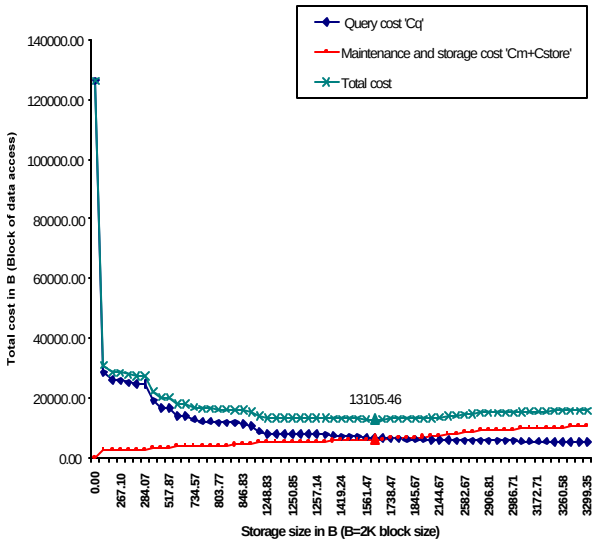

Figure 9: Total costs Ctotal, total query processing cost and the sum of maintenance and storage costs vs. storage size of the materialized views.

If all the summary views of the data warehouse are materialized, query performance can be optimized. However, this method requires the highest maintenance and storage cost. For a data warehouse with limited hard disk storage space and small maintenance window, materializing a few summary views which have the greatest storage effectiveness h_i (e.g., 'CO-P-DAY' for this case study) can effectively reduce query response time since they yield the greatest benefit yet require the least amount of storage space and maintenance costs. In the situation of a data warehouse which can be taken off-line for view maintenance and can have very large disk space available for the storage of materialized views, storing the set of optimal materialized views L can minimize query and maintenance cost while achieving good query performance.

4. Implementation and Testing

As part of this study, we have implemented a prototype system based on the Windows NT 4.0 server and Oracle Workgroup Server 7.3.4 [11], running on a Pentium PC. Data records were extracted from

a Paradox database and a Peach Tree accounting database, and imported to the Oracle database system. Various table spaces were created for storing the base-data and summary views of this data warehouse. Due to the constraints in data storage and computational costs involved, a data sampling method was adopted to estimate the size of each summary view [12]. A subset of sales records (i.e. the yearly sales records of 1996) was extracted from the existing database. Based on these sales records, the required summary views were generated and the sizes of these summary views are listed in Table 7. In this section, we describe both analytical and empirical testing results obtained in our case study.

4.1 Analytical Testing

The C_{total} under five test conditions, composed of different query patterns and frequencies, was evaluated for three different view materialization strategies, i.e., *all-virtual-views* method, *all-materialized-views* method, and *selected-materialized-views* method. The results are summarized in Table 8 and pictorially plotted in Figure 10. Clearly, the C_{total} of the *all-virtual-views* method is much greater than those of the other two methods. This is because the latter two methods utilize the pre-calculated data in the materialized summary views, thus avoiding accessing and processing a large quantity of base-data. On the other hand, the total cost evaluated for the *selected-materialized-views* method is the smallest under all five test conditions (cf. Table 8).

The storage and maintenance costs, and the query processing cost versus query frequency are plotted in Figures 11 and 12 respectively for the *all-materialized-views* and the *selected-materialized-views* methods. The maintenance cost $C_m(L)$ and storage cost $C_{store}(L)$ of the *selected-materialized-views* method in Figure 11 are less than that of the *all-materialized- views* method in all cases. This is because the summary views which are not cost effective (i.e. summary views with negative benefit *Net* (B_i)) will not be materialized in the data warehouse when the *selected-materialized-views* method is applied, hence resulting in a smaller value of C_{total} than the *all-materialized-views* method.

As shown in Figure 12, the total query processing costs for both methods increase steadily as the query frequency increases. However, the *selected-materialized-views* method requires a slightly higher query processing cost than the *all-materialized-views* method. The reason is obvious, as the latter stores all the materialized views in the data warehouse.

Query frequency f_{qi}	*all-materialized-views* Total Cost C_{total} (in B)	*All-virtual-views* Total Cost C_{total} (in B)	Selected-materialized-views Total Cost C_{total} (in B)
3	3030.00	4482.00	3018.00
31	7236.44	31479.00	6201.44
57.5	13251.28	63135.00	9510.60
115	15978.44	126270.00	13105.46
230	21432.75	252540.00	19516.51

Table 8: Total cost for three different view materialization strategies

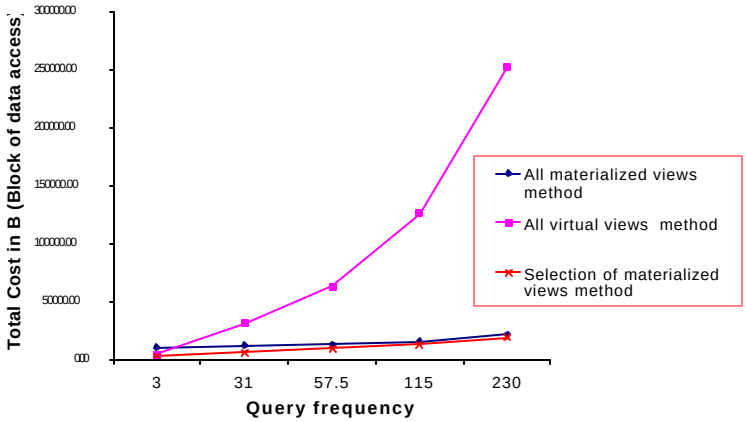

Figure 10:Total cost Ctotal for three view materialization strategies.

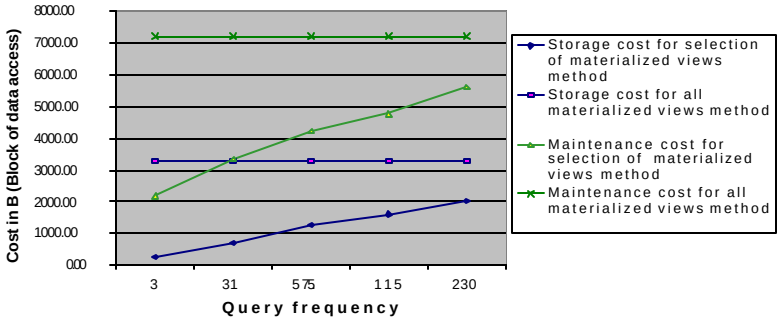

Figure 11:Storage and maintenance costs for all-materialized-views and selected-materialized-views methods

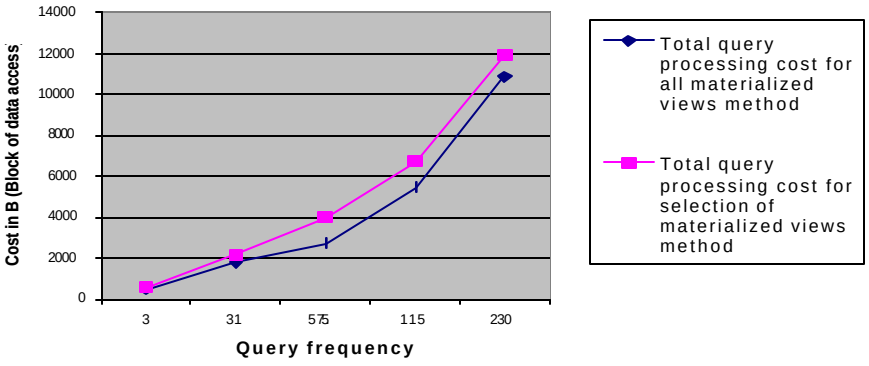

Figure 12:Query processing cost for all-materialized-views and selected-materialized-views methods

4.2 Empirical Testing

Our second test results come from an experimental prototype we have developed for the case study. The experiment was set up to simulate the view maintenance (i.e. re-computation of materialized summary views) and query processing of the data warehouse. The program execution times of the three view materialization strategies for computing 115 queries and completing maintenance processes were recorded. We conducted each test on an Oracle PL/SQL work sheet [13] 4 times so as to obtain a normalized result.

Figure 13 shows the average execution times under three view materialization methods. The shortest total query and maintenance time was recorded by using the *selected-materialized-views* method. The difference in execution time between the *all-materialized-views* method and the *selected-materialized-views* method is about 25 seconds. Even though the *all-materialized-views* method requires the shortest query processing time, the maintenance time required for re-calculating all the summary views is much longer than for the *selected-materialized-views* method. Therefore, the sum of view maintenance and query processing times for the *all-materialized-views* method is longer compared with that for the *selected-materialized-views* method. The *all-virtual-views* method requires the longest program execution time. This is because processing a large amount of base-data requires longer query processing time. In summary, the results obtained from this experiment is coincident with the analytical results discussed in the previous section.

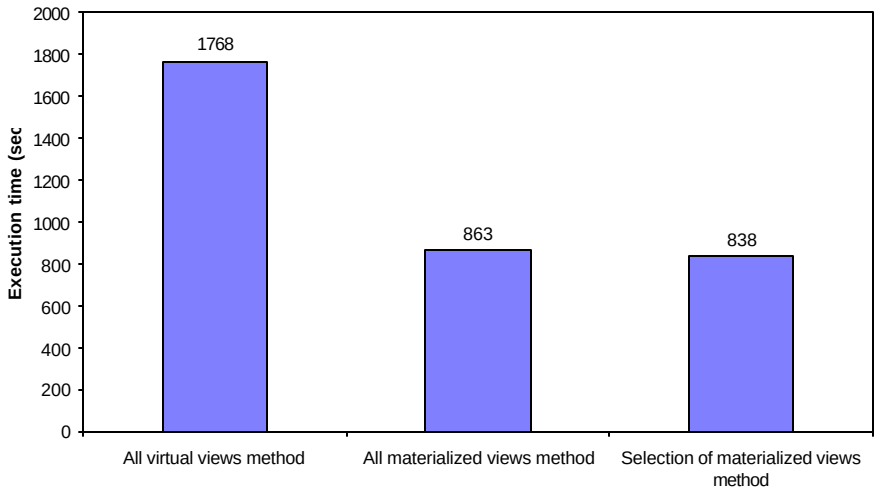

Figure 13: The average execution time of three view materialization strategies

Guidelines for warehouse schema design and materialized views selection

Our experiences gained from this case study can be summarized into the following guidelines for both data warehouse design and materialized view selection.

On Data Warehouse Design

i. Use the smallest size of integer or numerical values for the key attributes in dimension tables to minimize storage space and query processing time (cf. Section 2.1).

ii. Normalize dimension tables with large amount of records and hierarchy levels to achieve smaller dimension tables. Thus, the storage size and joining cost can be reduced substantially (cf. Section 2.1).

iii. De-normalize dimension tables with relatively few records and attributes to minimize the number of joins required (cf. Section 2.1).

iv. Horizontally partition the fact table, which has a lot of records, into smaller summary views according to its dimension key attributes so as to improve query performance, and further enable users to select various summary views for materialization based on the query access frequency (cf. Section 2.1).

v. Store foreign keys of dimension tables in the summary views, especially those dimension tables that are frequently accessed to help improve the query performance. Furthermore, data in these summary views can also be easily used by other queries (cf. Section 2.3).

vi Store frequently accessed dimension attributes (e.g. Co_name and P_name in our case study) in the summary views, especially for the dimension tables which have very many records, so as to minimize the number of joins and query processing costs (cf. Section 2.3).

On Materialized Views Selection

i. Materialize summary views that are frequently accessed by users (cf. Section 3.1 and Section 3.3).

ii. Materialize those commonly shared views which are used for generating other summary views (cf. Section 3.1 and Section 4.2).

iii. Materialize those views whose sizes have been substantially reduced from their ancestor's views (cf. Section 3.1).

iv When the storage factor is very small (i.e. large amount of disk storage is available), materializing a set of optimal materialized views 'L' by the selection method as illustrated in Section 3 can achieve the best combination of good query performance and low maintenance cost.

5. Conclusions

In this case study, methods for designing an efficient data warehousing system based on the application requirements of an engineering company 'R' have been investigated. A hybrid schema was designed for this data warehouse by applying dimensional modeling concepts. A cost model was developed to calculate the costs and benefits associated with materializing each data warehouse view.

The total cost under five test conditions, composed of different query patterns and frequencies, were evaluated for three different view materialization strategies: 1) *all-virtual-views* method, 2) *all-materialized-views* method, and 3) *selected-materialized-views* method. The total cost evaluated from using the *selected-materialized-views* method was proved to be the smallest among the three strategies in all cases. Further, an experiment was conducted to record different execution times of the three strategies in the computation of a fixed number of queries and maintenance processes. Again, the *selected-materialized-views* method requires the shortest total processing time.

An adapted greedy algorithm using the cost analysis methodology for evaluation was developed for materialized views selection. This view selection methodology was tested both analytically and experimentally and proved to be very cost effective for the optimization of the data warehouse. General guidelines for data warehouse design and materialized views selection based on this work are presented and a prototype of the data warehouse system was implemented using a commercially available data warehousing software "Oracle-Discoverer" [14, 15].

We plan to apply the cost evaluation methodology and views selection algorithm developed in this case study to other data warehousing applications, such as inventory, production and purchasing analyses, etc. In addition, warehouse view self-maintenance methods [10, 16] other than the view re-calculation method adopted by this work will also be investigated in order to further reduce system maintenance cost and achieve data warehouse optimization.

6. References

[1] S. Chaudhuri and U. Dayal. "An Overview of Data Warehousing and OLAP Technology". SIGMOD Record, 26(1):65-74, 1997.

[2] J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, Y. Zhuge. "TheStanford Data Warehousing Project". IEEE Data Engineering Bulletin, June 1995.

[3] V. Harinarayan, A. Rajaraman, and J. Ullman. "Implementing data cubes efficiently". Proceedings of ACM SIGMOD 1996 International Conference on Management of Data, Montreal, Canada, June 1996, pages 205--216.

[4] J.Yang, K. Karlapalem, and Q. Li. "A framework for designing materialized views in data warehousing environment". Proceedings of 17th IEEE International conference on Distributed Computing Systems, Maryland, U.S.A., May 1997.

[5] H. Gupta. "Selection of Views to Materialize in a Data Warehouse". Proceedings of 1997 International Conference on Database Theory, Athens, Greece 1997.

[6] Red brick systems, Star Schemes and Starjoin Technology, White Paper, http://www.redbrick.com /rbs /whitepapers/star_wp.html#fig1.

[7] I. Mumick, D. Quass, B. Mumick. "Maintenance of Data Cubes and Summary Tables in a Warehouse". Proceedings of the ACM SIGMOD 1997 International Conference on Management of Data,, Tuscon, Arizona, May, 1997.

[8] J.Yang, K. Karlapalem, and Q. Li. "Algorithms for Materialized View Design in Data Warehousing Environment". Proceedings of the 23rd International Conference on Very Large Data Bases, Athens,

Greece 1997, P.136-145.

[9] Ramez Elmasri, Shamkant B. Navathe. "Fundamentals of Database Systems". 2nd edition. The Benjamin/ Cumming Publishing Co.

[10] D.Quass, A. Gupta, I.S. Mumick, J. Widom. "Making Views Self Maintainable for Data Warehouse". Proceedings of the 4th International Conference on Parallel and Distributed Information Systems, 1996.

[11] Oracle WorkGroup Server Administration Guide, Oracle.

[12] A. Shukla, P. Deshpande, J. Naughton and K. Ramasamy. "Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies." Proceedings of 22nd International Conference on Very Large Data Bases, Mumbai, Bombay, India, 1996.

[13] Oracle PL/SQL User's Guide and Reference, Oracle.

[14] Oracle Discoverer 3.0 User's Guide, Oracle.

[15] Oracle Discoverer 3.0 Administration Guide, Oracle.

[16] N.Huyn. "Efficient View Self-Maintenance". Proceeding of ACM Workshop on Materialized Views: Techniques and Applications, Montreal, Canada. 1996.

7. Annotations

	:	Data block size (B =2K, one data block size of the Oracle database system setup in our case study).
T	:	The set of all sales summary views grouped by various dimension key attributes.
$\|T\|$	:	The number of sales summary views in T.
V_i	:	A sales summary view in T.
V_d	:	A dimension table (e.g., Co, Region, Pd, Pd_Cat, Pd_Mfr in the data warehouse).
V_{ai}	:	The ancestor view of V_i.
V_{ni}	:	The descendent view of V_i.
$S(V_d)$	:	The size of dimension table V_d in terms of data block B.
$S(V_i)$	:	The size of summary view V_i in terms of data block B.
$S(V_{ai})$	:	The size of V_{ai} in terms of data block B.
$S(V_{ni})$	:	The size of V_{ni} in terms of data block B.
$C_g(V_i)$	:	The query processing cost which involves selection and aggregation of summary view V_i.
$C_j(V_d, V_i)$	:	The processing cost for joining dimension table V_d with sales summary view V_i.
$C_{store}(V_i)$	:	The storage cost for storing summary view V_i in terms of data block B.
$C_m(V_i)$	:	The maintenance cost for re-computing V_i.
$Ct(V_{ni} \neg V_i)$	:	The cost for evaluating the descendent view V_{ni} from the materialized view V_i.
$Ct(V_{ni} \neg V_{ai})$	:	The cost for evaluating the descendent view V_{ni} from V_{ai}, the ancestor view of V_i.
L	:	An optimal set of materialized summary views ($L \subseteq T$).
$C_{store}(L)$	:	The total storage cost for storing all the materialized summary views in L.

C_m (L)	:	The total maintenance cost for re-computing all the materialized summary views in L.
C_{total}	:	The total cost (i.e. the sum of total query processing cost, maintenance cost, and storage cost) in the presence of zero or more materialized views.
Min (C_{total})	:	The minimum total cost C_{total} evaluated in Table 7.
q_i	:	A user's query which involves selection and aggregation of summary view V_i, and the joining of V_i with dimension tables.
C_q (q_i)	:	The query processing cost for q_i
f_{qi}	:	The query frequency of summary view V_i between every two updates.
f_q (V_{ni})	:	The query frequency of summary view V_{ni}, the descendent view of V_i.
f_{ui}	:	The update frequency of summary view V_i.
m	:	The average number of descendant views of a materialized summary view.
B_i	:	The benefit of materializing summary view V_i.
Net (Bi)	:	The net benefit of materializing summary view V_i.
h_i	:	The storage effectiveness of view V_i (i.e. net benefit per unit of occupied storage space).
r	:	The total number of user's queries between every two updates.
$Total$(C_{qr})	:	The total cost for processing r number of users' queries between every two updates.
$Total$(C_{qall})	:	The total cost for processing r number of users' queries between each update time interval when all the views are kept virtual (i.e. no summary views are materialized).
j	:	The number of materialized summary views in the data warehouse.
$Total$(C_m)	:	The total maintenance cost for j numbers of materialized summary views.
$Total$(C_{mT})	:	The total maintenance cost for all the summary views in T.
$Total$(C_{store})	:	The total storage cost required to store both base-data and all the materialized summary views in the existing system.
S	:	System storage space constraint for the existing data.
$Total$(C_{storeT})	:	The total storage cost required to store all the summary views in T.
U	:	Storage factor: $U = (Total(C_{store}) + Q * Y * S_a)$ / Total available storage space (In our case study, $U = 1$; thus $C_{store}(L) = U * S(L) = S(L)$.)
Y	:	Estimated life cycle of the data warehouse (Y=10 years for our case study).
Q	:	Estimated increase rate in data volume per year within the data warehouse (for our case study, Q=1.2).
S_a	:	Storage space required to store yearly increased new data and their summarized data.