1998

A data warehouse view selection scheme to accommodate dimension hierarchies.

Xiaohong. Meng
University of Windsor

A Data Warehouse View Selection Scheme to Accommodate Dimension Hierarchies

by

Xiaohong Meng

A Thesis
Submitted to the College of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

1998

0-612-52611-9

Abstract

A data warehouse is a central data repository that supports efficient execution of complex business decision queries. Data warehouse views are aggregations or summary tables holding millions of records integrated from a variety of source data systems. An *n*-dimensional data cube is a multidimensional data model used to generate *n* different perspectives of the measure aggregates of interest, and has 2^n subviews. Query response time can be significantly improved by pre-computing and storing needed warehouse views. Owing to disk space constraints and increasing maintenance cost of materialized views, pre-computation and storing of all of the required views may not be feasible. Thus, many algorithms have been proposed for selecting only a subset of views of the data cube most beneficial to materialize for better query response time. When taking the warehouse dimension hierarchies into consideration, the view selection problem in a data warehouse gets more complex. The objective of this thesis is to review and contribute a solution to the view-selection problem to accommodate warehouse dimension hierarchies. The proposed selection scheme recommends a set of warehouse views including the dimension subviews with maximum benefits, to materialize in order to improve the query response time in a data warehouse.

To my parents and my family, with love.

Acknowledgments

I first would like to express my thanks to my principal advisor, Dr. C. I. Ezeife, for her great advice and guidance on my thesis work. I would also like to express my thanks to Dr. R. Frost and Dr. K. Hildebrandt, who served as internal and external readers on my Master's Committee, for their comments and suggestions on this thesis. Finally, I would like to express my gratitude to my parents for their constant encouragement and support, which made it possible for me to complete the study while working and raising my children.

Table of Contents

List of Tables

List of Figures

1 Introduction

A data warehousing system is designed to support on-line analytical processing (OLAP) and decision support systems. Business executives, managers and analysts query huge warehouse data for making better and faster decisions through warehouse user interface. A data warehouse is a collection of subject oriented, integrated, non-volatile, and time variant data [In96]. The warehouse data is organized around major subjects of an enterprise and not around its functions as the online transactional processing system (OLTP) does. For example, an insurance company's major subject areas might be customer, policy, premium and claim. An integrated warehouse table can list the claim made by each customer and premium paid by each customer for each policy over a period of time. OLTP, on the other hand, stores customer data on premiums separately from data on claims and stores only current data. The most important aspect of a data warehouse is integration. The data coming from a number of source operational systems is integrated before loading into the data warehouse. The integration process includes encoding, transforming, merging, cleaning, and computing aggregates. The data warehouse is non-volatile because updates of data generally do not occur in the warehousing environment. Warehouse data is time variant since most of the data in the warehouse consists of collections of historical information over a time horizon of 5 to 10 years. It is a series of snapshots, taken at some moment in time. The key structure of data warehousing always contains some element of time. In general, data warehouses are created to provide users with data access and to support online analytic processing (OLAP) and decision support systems.

Typically, the data warehouse is maintained separately from the operational databases in an organization. The functional and performance requirements of a data warehouse are quite different from those of the on-line transaction processing (OLTP) applications traditionally supported by operational databases. Traditional operational systems are organized around the applications of the company, which typically automate clerical data processing tasks such as order entry and banking transactions. Since data warehouses contain consolidated data, perhaps from several operational databases, over potentially long periods of time, they tend to be orders of magnitude larger than operational databases; enterprise data warehouses are projected to be hundreds of gigabytes to terabytes in size. The workloads (applications accessing) in a data warehouse are query intensive with mostly ad hoc, complex queries that can access millions of records and perform many scans, joins, and aggregates. Query throughput and response times are more important than transaction throughput [ChDa97].

The topic of data warehousing encompasses architectures, algorithms, and tools for bringing together selected data from multiple databases or other information sources into a single repository, the data warehouse, suitable for direct querying or analysis [Wi95]. Although the concept of data warehousing is already prominent in the database industry, there are still many challenging issues in both research and industry. One such challenging area is the data warehouse aggregate materialization problem, also referred to as the view-selection problem. Many queries over data warehouses require aggregate views or summary data, which can be obtained by joining many large tables. The size of the data warehouse and the complexity of queries can cause queries to take very long to execute, and this delay is unacceptable in most decision support systems. One technique used to improve warehouse query response time is pre-computation and storing (materialization) of aggregate views. However, as changes are made to the data sources, all the warehouse views that depend upon this data must be updated to reflect the changed state of the data sources [MuQuMu97, Zhetal95, Hu97]. Gray *et al.* presents the concept of the data cube, a multidimensional representation of a set of aggregate measures. An n-dimensional data cube has 2^n subcubes. When taking warehouse dimension hierarchies into consideration, the number of warehouse aggregate views can be huge. Storing all these huge warehouse views may not be feasible owing to storage space constraints and increased maintenance costs since all stored views need to be refreshed as updates are being made in the source databases.

Defining ways to select an appropriate set of warehouse views to materialize is one of the most important design decisions in designing a data warehouse. Since OLAP queries are complex and the volume of data is large, there is a need to balance the time-space trade-off in order to make the system usable. Carefully selecting and defining a set of views and their indexes to materialize contributes to finding this needed balance between maintenance cost, storage space and query response time [Ez97b]. Thus, given a set of queries and some storage space constraint, the decision on which aggregate views to materialize in a data warehouse to minimize response time and maintenance cost remains a challenging research issue. The purpose of this thesis is to review and contribute a solution to the warehouse view selection problem especially when the dimension hierarchies are taken into consideration.

1.1 Data Warehouse Model

Data warehousing technologies have evolved in the last decade and have been successfully applied in many industries including manufacturing, telecommunications, financial services, retail stores and health care [ChDa97]. This section discusses briefly the typical data warehousing architecture, its components, tables in a star schema, aggregate views modeled as data cube and dimension hierarchies.

1.1.1 Data Warehouse Architecture

Figure 1 shows a typical data warehouse architecture. It consists of the following components: Information sources, Extractor/Monitor, Integrator/Loader, Data Warehouse/Data Marts, Information Delivery/OLAP Tools, and Administration Tools [ChDa97]. The information sources can be conventional database systems, or external sources such as flat files, news wires, HTML and SGML documents, knowledge bases and legacy systems. The extractor is the component responsible for extracting data from multiple information sources while the monitor is responsible for automatically detecting changes of interest in the source data and reporting them to the integrator. The integrator is responsible for installing the information in the warehouse, which may include transforming and filtering the information, summarizing it, or merging it with information from other sources. The loader is responsible for loading data into the data warehouse and periodically refreshing the warehouse to reflect updates at the sources. The data warehouse is the central repository of large volume of integrated, consolidated, and summarized data. In addition to it, there may be several departmental data marts. Data in the warehouse and data marts are stored and managed by one or more warehouse servers, which present multidimensional views of data to a variety of front end information delivery tools: query tools, report writers, analysis tools, and data mining tools. Finally, there are also tools for monitoring and administering the warehousing system.

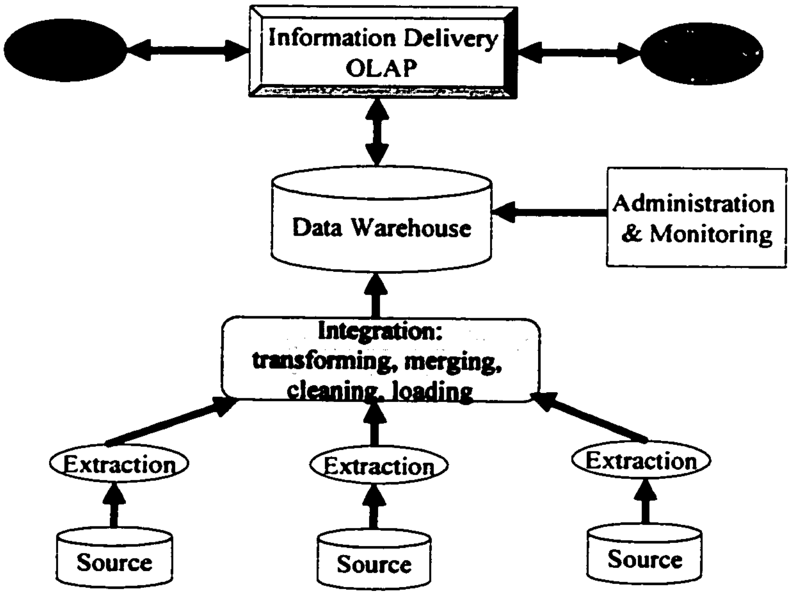


Figure 1 Data Warehouse Architecture

1.1.2 Data Warehouse Design Methodology

Entity Relationship (ER) diagrams and normalization techniques are popularly used for database design in OLTP environments. However, the database designs recommended by ER diagrams are inappropriate for decision support systems where efficiency in querying and in loading data are important [ChDa97].

There are two major types of data warehouse design models: the entity model and the dimensional model. The entity model is a normalized relational database modeling approach that emphasizes the elimination of data redundancies in the design. It produces a database that is complete and easily maintained, making this approach suitable for a data warehouse that is intended for data archiving. The dimensional modeling approach results in a database design that is consistent with paths by which users wish to enter and navigate the data warehouse. Frequently requested aggregates or calculated measures are stored in the database, creating useful data redundancies that make it possible to avoid performance-inhibiting repetitive calculations every time a report is prepared.

Dimensional modeling uses fact tables that store time-series historical data, indexed on dimensional keys, and are described in corresponding dimension tables. Each fact table has an indexed primary key composed of several columns, each of which logically corresponds to a major business dimension such as time period, product, or market. Each dimension key must be represented and described in a corresponding dimension table, which logically joins to the fact table(s) through identical primary key columns. Each dimension table should incorporate multiple attribute columns containing text and codes that further describe the key.

The popular design option in the dimensional data model is the ***star schema***, which is used by most data warehouses to represent the multidimensional data model. The basic premise of the star schema is that information can be classified into two groups: facts and dimensions. Facts are the core data elements being analyzed; dimensions are attributes about the facts. The database consists of a single historical fact table within each category, which contains detail and summary data, and a single dimension table for each dimension indicated in the fact table. The primary key of the fact table contains only one key column from each dimension. Each dimension table consists of columns that correspond to attributes of the dimension. In the typical star schema, the fact table is much larger than any of its dimension tables. Figure 2 shows an example of a star schema.

There are other design options available in the dimensional data-modeling approach, such as partial star schema, fact partitioning schema, dimension partitioning schema and snowflake schema [Ta97]. We are not going to discuss them further in this thesis.

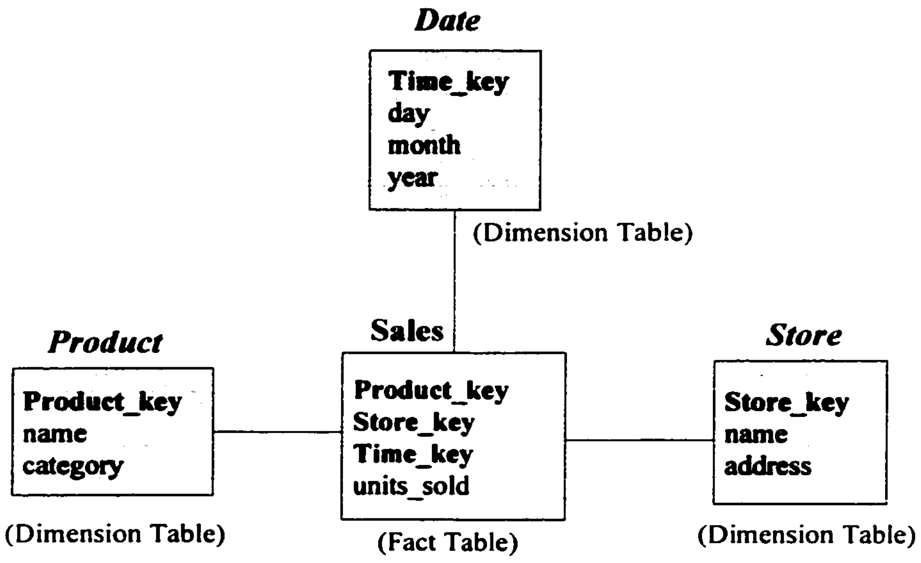


Figure 2 A Star Schema

1.1.2 Aggregate Views and The Data Cube

Aggregation is the process of summarizing granular, or detail, data and physically storing the aggregations as warehouse views along common business hierarchies. In order to answer aggregate queries quickly, a warehouse often stores a number of summary tables, which are materialized views that aggregate the data in the fact table possibly after joining it with one or more dimension tables. Aggregating detail data in advance decreases access times, and speeds up processes such as rolling up and drilling down. A roll-up analysis presents an aggregation first in the lowest level detail but proceeds to present it in a more general level detail (e.g. computing the total sales per day first, then per month, finally per year). A drill-down analysis presents summaries from the coarsest level detail to the lowest level detail (e.g. computing the total sales per year and then per month and finally per day).

Typically, warehouse queries are interested in some measure aggregate value such as total sales along many dimensions. The data cube is a special data structure defined by Gray *et al.* in [Gretal96] for representing multi-dimensions of an aggregate attribute. An n-dimensional data cube represents 2^n aggregate views for a set of n group-by attributes in the main warehouse fact table. Each cell of this data cube is an aggregate view along one or more dimensions. For example, Figure 3 is a graphical presentation of a 3-dimensional data cube with the average grade as the measure aggregate value along the three dimensions: student(S), course(C) and term (T) in a university data warehouse, this 3-D data cube has eight ($2^3 = 8$) subcubes.

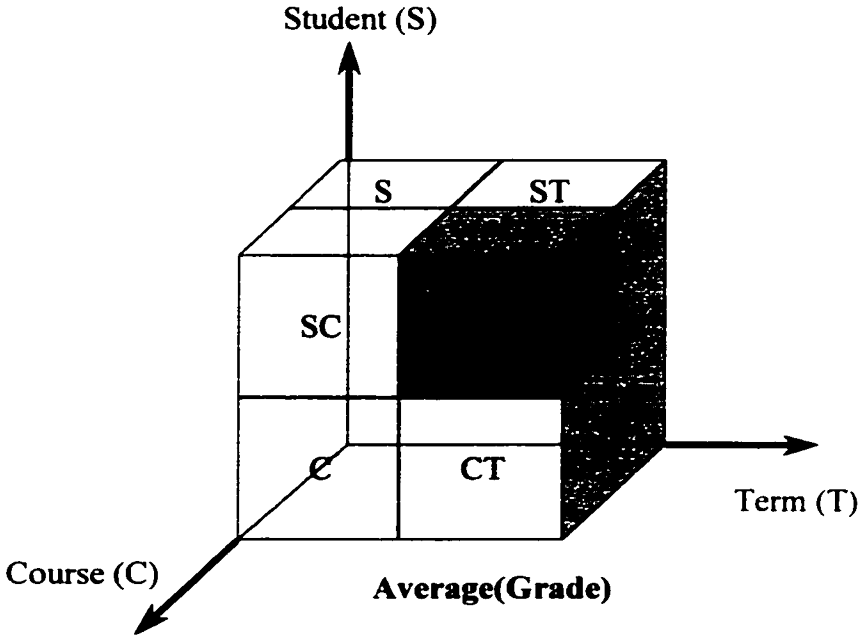


Figure 3 A Graphical Presentation of a 3-Dimensional Data Cube

1.1.3 Dimension Hierarchies

In most real-life applications, dimensions of a data cube consist of more than one attribute, and the dimensions are organized as hierarchies of these attributes. The dimension attributes on both warehouse fact and main cube aggregate tables are foreign keys, each foreign key attribute may have associated with it a dimension hierarchy specifying attributes for describing it. Data in dimension tables often represent dimension hierarchies. A dimension hierarchy is essentially a set of functional dependencies among the attributes of the dimension table. Hierarchies are very important, as they form the basis for two very commonly used querying operations: rolling up and drilling down, as described earlier in section 1.1.2. Hierarchies introduce query dependencies that we must account for when determining what queries to materialize. For example, the time dimension in a data warehouse can have the following attributes: *day*, *week*, *month*, and *year*. Since months and years cannot be divided evenly into weeks, if we do the grouping by *week* we cannot determine the grouping by *month* or *year*. This time dimension hierarchy is shown in Figure 4.

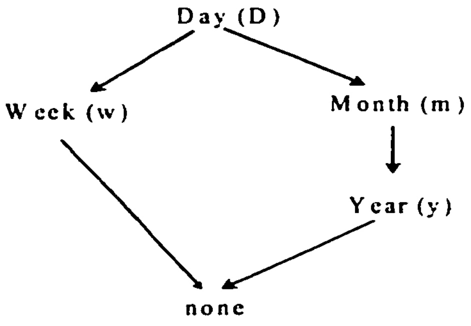


Figure 4 A Time Dimension Hierarchy

1.2 The Thesis Objective and Scope

A data warehouse is a repository of integrated information available for querying and analysis. It usually contains many warehouse views, referred to as materialized views, derived from the data in the sources. This thesis will investigate the view-selection problem in a data warehousing environment. That is, given a set of common queries and some storage space constraints, the goal is to select an appropriate set of materialized views to minimize total query response time. In particular, this thesis will present a view selection scheme for warehouse views that accommodates dimension hierarchies. This selection will not only consider the warehouse views grouped-by on primary key dimension attributes but also on non-primary key dimension attributes. The thesis first reviews and studies the related work in this area before presenting a warehouse view selection technique that accommodates dimension hierarchies in a data warehousing environment. Examples are used to demonstrate this selection technique. The proposed selection technique is implemented and results of the experimentation conducted are reported. The proposed scheme is compared with the greedy algorithm introduced in [HaRaUl96] to show that it has a better query response time than the latter. The scope of the thesis is limited to focusing on the selection of the warehouse aggregate views including both main subviews and dimension subviews, while leaving the consideration of indexes as a future work.

1.3 Thesis Outline

The thesis is organized as follows. Chapter 2 reviews the related work on warehouse view selections. Chapter 3 presents an example based on a simple university warehousing system which keeps track of student grades. It shows how a combined cube lattice and a partial combined cube lattice can be generated to accommodate dimension hierarchies. It also demonstrates how decisions can be made about selection of views using some common queries. Chapter 4 presents the View-Selection-DH Greedy scheme for selecting warehouse views to accommodate dimension hierarchies. The performance of the proposed algorithm is compared with that of the greedy algorithm. Finally the conclusions and contributions of the thesis as well as future work are outlined in Chapter 5. The sample data for the fact table and dimension in the university data warehouse system are listed in Appendix A.

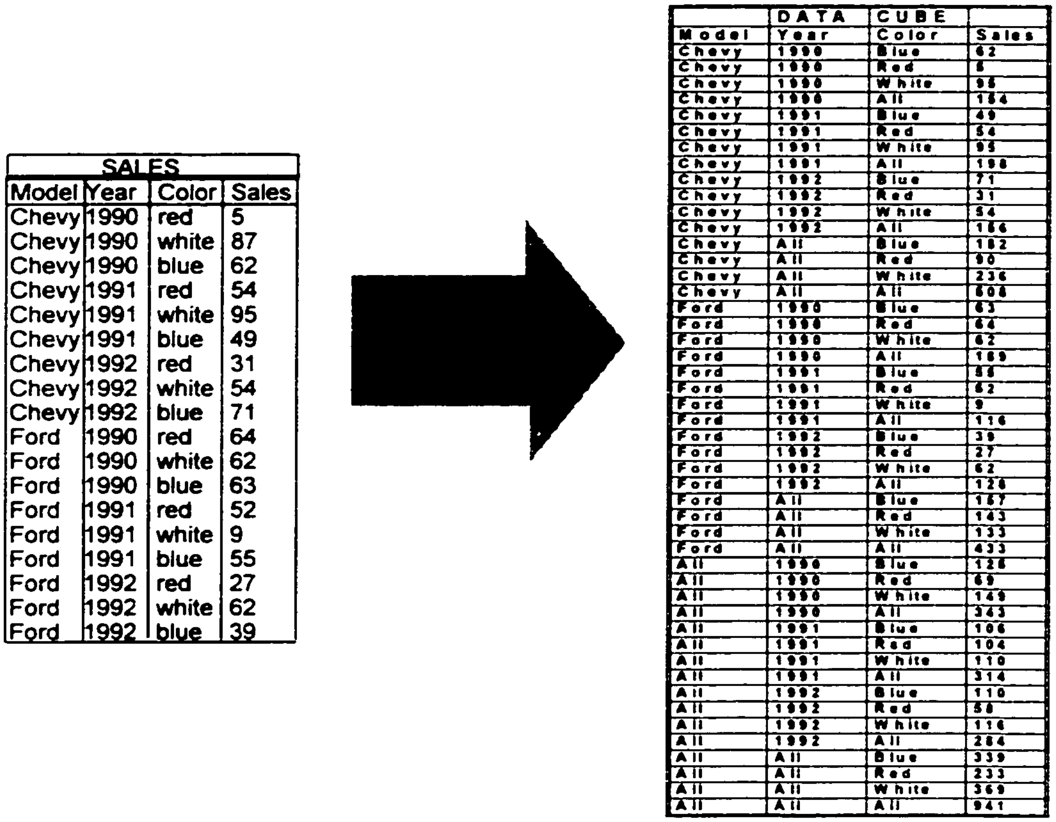
2 Related Work on Warehouse View Selections

This chapter presents a review and study of related work and researches being done recently on warehouse view selections.

Gray *et al.* overcomes one limitation of the standard SQL aggregate functions and defines the ***data cube*** operator for generating aggregation of *n*-dimensions in [Gretal96]. The SQL aggregate functions and the ***group by*** operator only produce zero-dimensional or one-dimensional answers. Multidimensional database system applications (OLAP) require viewing the data from many different business perspectives (dimensions) and need the *n*-dimensional generalization of these operators. A data cube is a multidimensional representation of a set of aggregate measures in a database. Each cell of the data cube corresponds to a unique set of values for the different dimensions. An *n*-dimensional data cube is a multidimensional data model used to generate *n* different perspectives of the measure aggregates of interest, and has 2^n subviews. Along with the data cube operator, Gray *et al.* also introduces the notion of ***ALL*** in the domain of each dimension. As an example shown in Figure 5, sales of automobiles can be organized by several dimensions, such as by model, color, and year of sale. The 3-D data cube on the right of Figure 5 can be built from the fact table at the left by the following CUBE statement:

```
Select  Model, Year, Color, SUM(sales) AS Sales
From    Sales
Where   Model in ('Ford', 'Chevy')
and     year Between 1990 and 1992
Group By Model, Year, Color With Cube;
```

This 3-D data cube has a total of eight ($2^3 = 8$) subcubes to represent sum of sales grouped by (1) nothing (2) model (3) year (4) color (5) model and year (6) model and color (7) year and color, and (8) model, color and year.



SALES			
Model	Year	Color	Sales
Chevy	1990	red	5
Chevy	1990	white	87
Chevy	1990	blue	62
Chevy	1991	red	54
Chevy	1991	white	95
Chevy	1991	blue	49
Chevy	1992	red	31
Chevy	1992	white	54
Chevy	1992	blue	71
Ford	1990	red	64
Ford	1990	white	62
Ford	1990	blue	63
Ford	1991	red	52
Ford	1991	white	9
Ford	1991	blue	55
Ford	1992	red	27
Ford	1992	white	62
Ford	1992	blue	39

	DATA	CUBE	
Model	Year	Color	Sales
Chevy	1990	Blue	62
Chevy	1990	Red	5
Chevy	1990	White	95
Chevy	1990	All	154
Chevy	1991	Blue	49
Chevy	1991	Red	54
Chevy	1991	White	95
Chevy	1991	All	198
Chevy	1992	Blue	71
Chevy	1992	Red	31
Chevy	1992	White	54
Chevy	1992	All	156
Chevy	All	Blue	182
Chevy	All	Red	90
Chevy	All	White	236
Chevy	All	All	508
Ford	1990	Blue	63
Ford	1990	Red	64
Ford	1990	White	62
Ford	1990	All	189
Ford	1991	Blue	55
Ford	1991	Red	52
Ford	1991	White	9
Ford	1991	All	116
Ford	1992	Blue	39
Ford	1992	Red	27
Ford	1992	White	62
Ford	1992	All	128
Ford	All	Blue	157
Ford	All	Red	143
Ford	All	White	133
Ford	All	All	433
All	1990	Blue	125
All	1990	Red	69
All	1990	White	149
All	1990	All	343
All	1991	Blue	106
All	1991	Red	104
All	1991	White	110
All	1991	All	314
All	1992	Blue	110
All	1992	Red	58
All	1992	White	116
All	1992	All	284
All	All	Blue	339
All	All	Red	233
All	All	White	369
All	All	All	941

Figure 5 An Example of a 3-D Data Cube

Implementation of a multi-dimensional data cube can be accomplished in three ways: 1) physically materialize the whole data cube; 2) materialize nothing; 3) materialize only part of the data cube. In [HaRaUl96], Harinarayan *et al.* presents a lattice framework and introduces the greedy algorithm for selecting a set of views to materialize in order to improve the query response time.

To define the lattice framework and propose the greedy algorithm, a simple linear cost model is used in [HaRaUl96]. The assumption is the cost of answering a query is proportional to the number of rows examined. The lattice framework shows the dependencies between subviews of the data cube. It imposes a partial ordering on the queries. The paper also proposes a technique for combining the main cube lattice and dimension hierarchies into a combined cube lattice.

In this paper, the space cost is the number of rows in the view. The greedy algorithm starts by selecting the top level view into a set *S*. Then the benefit of each of the descendant cube views relative to the selected set *S* is calculated. The benefit of a view, *v* not in the *S* is computed as the sum of all positive differences between the sizes of the smallest parent *u* already in the set *S* of each view *v*'s descendant view *w* including *v* itself and the size of view *v*. If the size difference between the smallest parent *u* of a descendant view *w* of *v* and the size of view *v* is negative, the benefit of view *w* is considered zero. The benefits of all remaining views in the lattice are computed each time and the view with the highest benefit is included in the set *S*.

For example, consider the cube lattice of Figure 6. Eight views, named *a* through *h*, have space costs as indicated on the figure. The top view *a*, with cost 100, must be chosen. Suppose we wish to choose three more views. The first choice will be view *b* because it has the highest benefit of 250 among all the views, which represents the cost reduction of (100-50)*5 for each of the descendant views *d*, *e*, *g*, *h* and *b* itself. Once view *b* is picked, the benefit of each remaining view is calculated again and the one with the maximum benefit is selected for materialization. As a result, the greedy algorithm selects the three subviews *b*, *f* and *d* as the first, second, and third choice in addition to the top view *a* to materialize in this example.

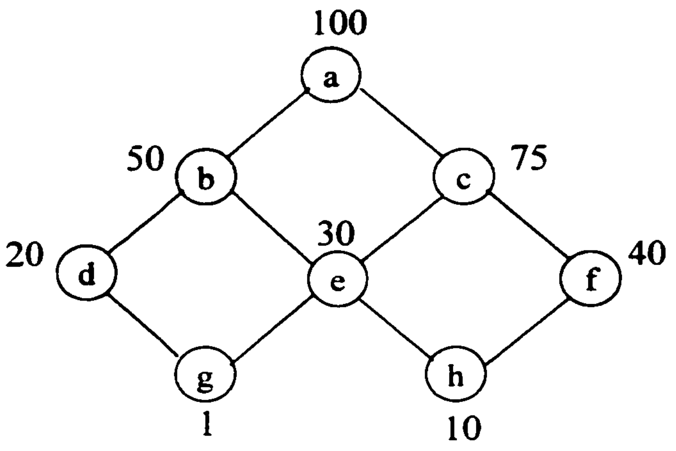


Figure 6 Example Lattice with Space Cost

This paper investigates the problem of deciding which set of views in the data cube to materialize in order to minimize query response times but it does not consider any indexes during the selection. The future work is to define more realistic cost models; use data cubes stored in MDDB systems instead of relational database systems; and generate dynamic materialization.

Gupta in [Gu97] proposes a framework for the general problem of selecting views to materialize in a data warehouse. Given some storage space constraint, the problem is to select a set of derived views to minimize total query response time and the cost of maintaining the selected views. Gupta introduces the *AND-OR* view graph in this paper and presents some polynomial-time heuristics for selection of views to optimize total query response time. In particular, the greedy algorithm, the greedy-interchange algorithm and the inner-level greedy algorithm are discussed for some of the special cases of the general data warehouse scenario as below.

1) Selection of views in an *AND* view graph, where each query/view has a unique evaluation

2) Selection of views in an *OR* view graph, where any view can be computed from any one of its related views

Gupta also extends the algorithms to the case when there is a set of indexes associated with each view and discusses the heuristic for the general case of *AND-OR* view graphs.

In most commercial OLAP systems today, the summary tables (subviews of the data cube) are materialized first, this is followed by the selection of appropriate indexes on them. In [Guetal97], Gupta *et al.* states that this two-step process can perform very poorly and the selection of both the summary tables and indexes should be done together for the most efficient use of space. This paper extends the greedy algorithm introduced in [HaRaUl96] and presents a family of algorithms of increasing time complexities to select both views and indexes to materialize in order to improve OLAP query performance. The algorithms with higher complexities have better performance bounds. In particular, [Guetal97] presents the r-greedy algorithm and shows how it works with different values of r. It also shows that the inner-level greedy algorithm of moderate complexity can perform fairly close to the optimal.

Gupta *et al.* argues that indexes are useful in reducing query costs and that a data cube with n group-by attributes has associated with it about $3n!$ possible indexes, $2n!$ of which are fat indexes. The fat indexes use more than one key combination. For example, the index I_{ps} indicates the search key of this index is a concatenation of the p and s dimensions. The order of the dimensions in the indexes matters. That is, index I_{ps} is completely different from index I_{sp}.

Careful selection of aggregate views and some of their most used indexes to materialize in a data warehouse reduces the warehouse query response time as well as warehouse maintenance cost under some storage space constraint. In [Ez97a] Ezeife defines a uniform approach for selecting both warehouse views and indexes based on a comprehensive cost model. Specifically, the approach considers the top view of a data cube as the main class object, and all its 2^n subviews as its attributes. The top view of the data cube is the view with all the group-by attributes which is called the base level view. Then the net benefits of keeping each of its subviews are computed using a cost/benefit model and stored in a matrix. Using the application access frequency to these subviews, application usage of these subviews and their computed net benefit matrix as well as the cube lattice, the scheme produces a view affinity matrix which shows how closely any pair of views are needed together by warehouse applications. The algorithm applies vertical clustering techniques, the Bond Energy algorithm (BEA) to generate the clustered view affinity matrix, which is now partitioned to generate two non-overlapping view fragments. The partitioning process aims at finding the best point along the diagonal of the clustered affinity matrix to split the views so as to minimize the incidence of sets of applications needed to cross-reference views/indexes in a different group. Then a fragment cleaning operation is applied to drop any view that is not used by any application as recorded in the application usage matrix information. Thereafter, the total benefit of each cleaned fragment is computed and the fragment with the highest total benefit is selected as the one containing all subviews that should be materialized.

Once a subcube of the data cube is selected, to select its indexes, the subcube in turn is made the main class object while the list of its indexes are made its attributes. Then the appropriate three matrices of index usage, index net benefit and index application frequency are used to define fragments of indexes to materialize. In all cases, a selected and cleaned fragment may need to be refragmented if the available storage space is less than the total space requirement for the views in this fragment.

In [Ez97b], Ezeife provides an extension to the vertical partitioning scheme proposed in [Ez97a] for handling views and indexes to accommodate dimension hierarchies. [Ez97b] introduces a dimension view interclass dependency graph (IDG) and a cost/benefit model that includes the joins on dimension tables needed to generate dimension views. This approach accommodates views that are grouped on dimension attributes of subjects and improves system performance by cutting off the enormous amount of time spent performing joins with dimension tables. The steps involved in identifying the set of views/indexes to materialize are summarized below:

1. Generate a combined cube lattice from the cube lattice and the dimension hierarchy lattices.
2. Generate an interclass dependency graph (IDG) to determine the join factor of a dimension view.
3. Generate the three matrices: view/index usage matrix (VUM), application frequency matrix (AFM), and the net benefit matrix (NBM) using the cost/benefit formulae.
4. Compute the view/index affinity matrix (VAM).
5. Based on the view/index affinity matrix, generate a clustered view/index affinity matrix using the Bond Energy algorithm (BEA).
6. Using the vertical partition algorithm on the clustered affinity matrix to generate non-overlapping fragments of views/indexes.
7. Perform fragment cleaning operation to drop every view not accessed by a query.
8. Select fragment with higher total net benefits.
9. Repeat steps 3 to 8 for the cube class object with all selected views included in its original views.

Ezeife and Baksh in [EzBa98] investigate the problem of reducing the size of warehouse views through horizontal fragmentation. They present a horizontal partitioning scheme for selecting warehouse views. The partitioning scheme first horizontally fragments the data cube views starting with the base level view. This horizontal fragmentation scheme enhances the one defined in [OzVa91] by taking more realistic approach because it uses access frequency of queries to attach an importance value to the predicates arising from the queries. The importance value (IP) of each predicate is defined by multiplying the cardinality of the predicate by application access frequency. The selection algorithm incorporates the benefits of fragmenting the parent view to select subsequent views tc materialize. For every selected parent view, a re-computed size of the parent view, derived from access of its fragments by queries, is used to select further views with the greedy algorithm. Every selected view is in turn partitioned and the size is re-computed.

[Ez98] enhances the performance of the partition-selection scheme for views presented in [EzBa98] by defining an algorithmic component named the fragment-advisor, which recommends the best set of fragments of a materialized view that answers any warehouse aggregate query. The input to the scheme are partition attribute, analysis attribute, measure attribute and set of predicates of the query as well as the set of materialized views with their fragments.

Baralis *et al.* in [BaPaTe97] proposes another technique to perform the selection of views to materialize in a multidimensional database. It states that the elements of the multidimensional lattice represent the solution space of the problem. Some view selection techniques proposed in the past do not scale well when the number and complexity of dimensions increase. Their proposed technique reduces the solution space by considering only the relevant elements of the multidimensional lattice.

Johnson and Shasha in [JoSh96] state that a two-step process of picking subcubes first, followed by the selections of indexes, is typically adopted in commercial OLAP systems. They propose an ad hoc approach in dividing the space and in picking indexes. Indexes are built on the most frequently used dimensions.

3 A View Selection Example

This chapter uses a simple university database to show how a combined cube lattice can be defined from the cube lattice and dimension hierarchies. It introduces the concept of partial combined cube lattice and shows how a partial combined cube lattice can be generated from the combined cube lattice and common warehouse queries. It also presents the proposed view selection scheme, although the formal presentation and discussion of the proposed algorithm is made in Chapter 4. This chapter demonstrates how the warehouse views in a partial combined cube lattice can be selected for materialization using both the greedy algorithm and the proposed selection scheme to improve query response time under some storage space constraints. A comparison of the total query response time based on the selection using both algorithms and relevant definitions used in the thesis are presented as well.

3.1 A Simple University Data Warehouse

The university database keeps track of the grade information for students in all the academic terms. It has information on students, the courses they have taken and the term in which the courses are offered. This university data warehousing system is designed using the star schema model, which uses a main fact table and multiple dimension tables. The fact table holds all the integrated, time-variant data while the dimension tables describe the foreign key attributes of the fact tables. The fact table in this warehouse is given below:

grade(student_id (S), course_no (C), term(T), grade)

In addition to the fact table, the following three dimension tables are stored in the warehouse with the ***student_id*, *course_no*** and ***term*** as the primary keys in each of the dimension tables, respectively:

student(student_id (S), name (n), gender (g))
course(course_no (C), course_name (d))
terminfo(term (T), year (y), season(o))

Here, a single letter abbreviation is used for each dimension attribute. A single uppercase letter is used to denote the primary key attribute for each dimension, and a lowercase letter is used to denote the non-primary key attributes for each dimension.

To demonstrate how aggregate views can be selected in a data warehouse, we will use the university data warehouse example with the sample fact table and corresponding dimension table data as shown in Appendix A.

For the fact table grade(student_id (S), course_no (C), term(T), grade), the domain of *student_id* is c0001, c0002, ..., c9999. The domain of *course_no* is cs60-100, cs60-101, ..., cs60-799. The *term* in which the courses are offered is recorded as yyyyW, yyyyF, yyyyS to indicate Winter, Fall, and Summer term in each year respectively. The sample fact table data shown in Appendix A only has 25 records although table holds millions of rows in reality.

The warehouse is designed using the star schema as shown in Figure 7. The fact table *grade* has a composite primary key, which consists of the three primary keys on the three dimension tables.

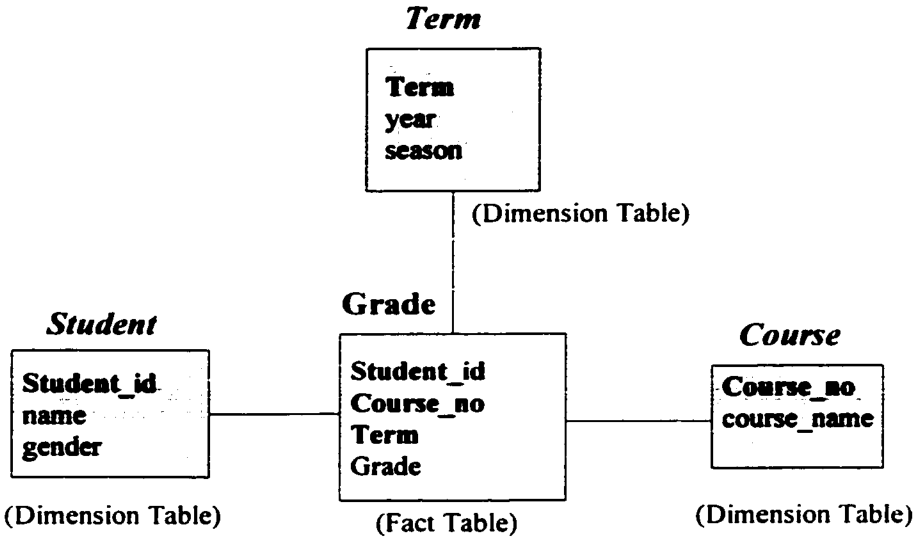


Figure 7 The Star Schema for the University Data Warehouse

3.2 Cube Lattice and Dimension Hierarchies

Consider the average grade as the measure aggregate value in this university warehouse example. There are three dimensions in the university data warehouse upon which we can perform business analysis. The data cube for this example is a 3-dimensional cube on average grade and has eight ($2^3 = 8$) main subviews, which have group-by attributes on the primary key attribute of each dimension: *student*, *course* and *term*. The cube lattice of these eight main subviews, which are denoted by the attributes in the *group by* clause as follows is given as Figure 8.

SCT - student_id, course_no, term

SC - student_id, course_no

ST - student_id, term

CT - course_no, term

S - student_id

C - course_no

T - term

( ) - none

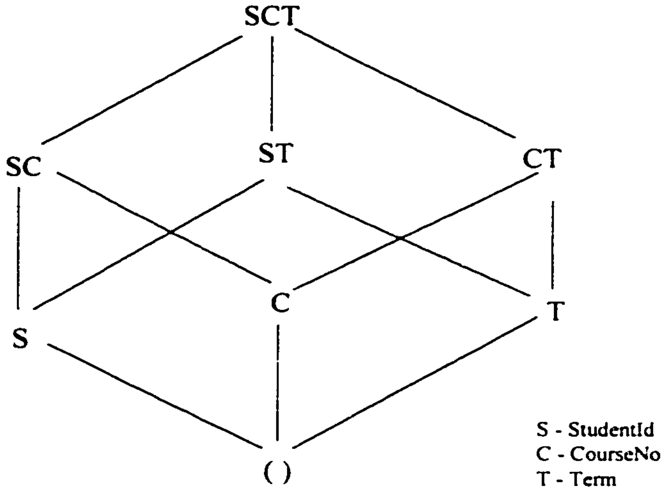


Figure 8 The Cube Lattice for the University Warehouse

Among these subviews, the very top one, which has group-by attributes on all the dimension keys is called the base level view. The very bottom one, which has only one row and denoted as ( ), is called the *ALL* view. These two views can be generated from the SQL statements below:

```
SCT:  select   student_id, course_no, term, avg(grade) as avg_grade
      from     grade
      group by student_id, course_no, term;
ALL:  select   avg(grade) as avg_grade
      from     grade;
```

The subview SC corresponds to the following SQL statement and can answer this query: *What is the average grade achieved by each student for each course in the university?*

```
SC:   select        student_id, course_no, avg(grade) as avg_grade
      from          grade
```

```
group by          student_id, course_no;
```

Here, some lattice notations in the context of warehouse views and queries as described in [HaRaUl96] are used. Consider two queries Q_1 and Q_2. We say that Q_1 is dependent on Q_2 if and only if query Q_1 can be answered using the results of query Q_2, and this dependency relation is denoted as $Q_1 \preceq Q_2$. The $\preceq$ operator imposes a partial ordering on the queries. In order to be a lattice, any two elements on the lattice must have a least upper bound and a greatest lower bound according to the $\preceq$ ordering.

We denote a lattice with a set of elements L and dependence relation $\preceq$ by $\langle L, \preceq \rangle$. For elements a and b of a lattice $\langle L, \preceq \rangle$, $a \prec b$ means that $a \leq b$ and $a \neq b$. The ancestors and descendants of an element of a lattice $\langle L, \preceq \rangle$, are defined as follows:

$$\text{Ancestor}(a) = \{\ b \mid a \preceq b\ \}$$

$$\text{Descendant}(a) = \{\ b \mid b \preceq a\ \}$$

In the cube lattice shown in Figure 8 the base level view is SCT. The view SC can be derived using only the results of view SCT as shown in the following SQL statement. Thus, SC is said to be dependent on SCT, denoted as SC ≤ SCT. Assume SCT view has the following columns: *student_id*, *course_no*, *term*, and *avg_grade*. The SC view can be achieved from this SCT view instead of from the fact table *grade* as follows.

```
Select  student_id, course_no, avg(avg_grade) as average_grade
from    SCT
group by student_id, course_no;
```

The dimension hierarchies for the university warehouse are given in Figure 9.

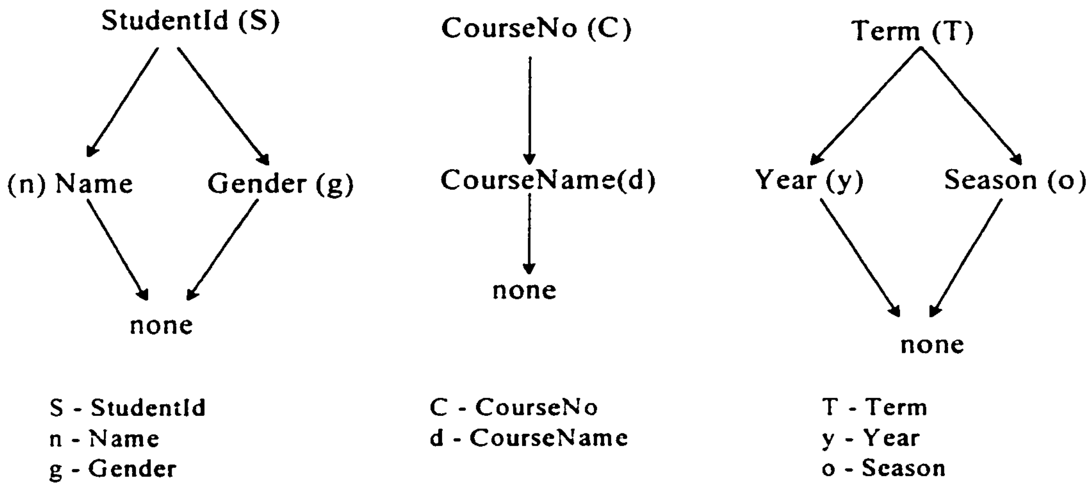


Figure 9 The Dimension Hierarchies for the University Warehouse

3.3 Combined Cube Lattice

In many situations, queries demand grouping by attributes on non-primary key attributes other than the primary key for each dimension. For example, the following query will need to group-by on the attributes *student_id* (S) and *course_name* (n), and the query is denoted as *Sn*.

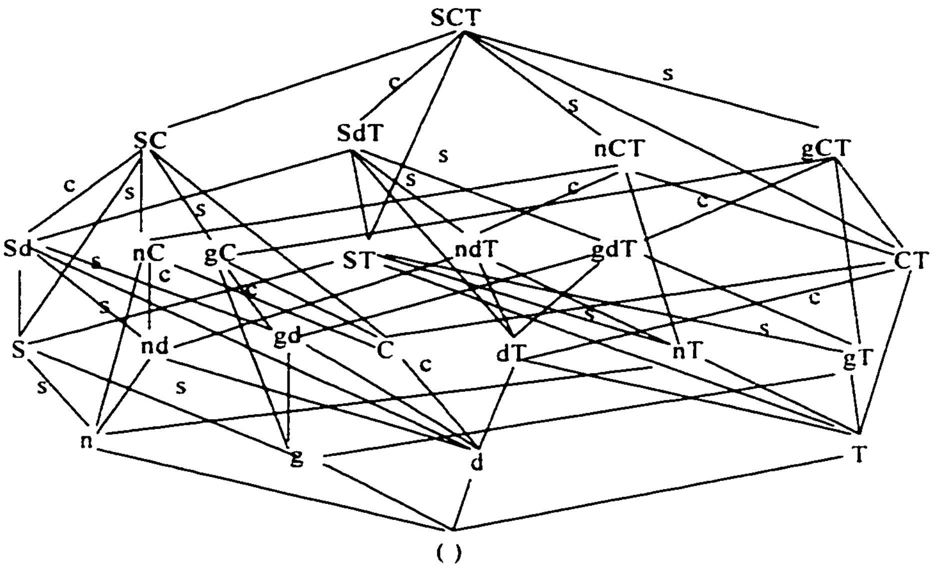
> *What is the average grade achieved by each student for the courses "Data Structure" and "File Structure" in 97W term?*

The following view can be generated to answer this query:

```
select   student_id, course_name, avg(grade) as avg_grade
from     grade g, course c
where    g.course_no = c.course_no
and      c.course_name in ('Data Structure', 'File Structure')
group by student_id, course_name;
```

In the above example, if the view *Sn* is materialized, it can be used to answer the query directly and this avoids the need to perform table join between the fact table *grade* and the dimension table *course*. Thus, the system performance can be improved considerably. In order to consider the dimension hierarchies in the view-selection problem, we first need to generate the combined cube lattice based on the cube lattice for the fact table and the dimension hierarchies.

The combined cube lattice is a direct product of the cube lattice for the fact table and the dimension hierarchies as stated in [HaRaUl96]. Here, we give further description of the steps to generate the combined cube lattice similar to those used in [Ez97b]. Starting with the base level view (i.e. SCT), we first list all the possible main subviews which have group-by on the primary key attributes as denoted by capital letters in our university warehousing example, and all the dimension subviews, which have group-by on the non-primary key attributes, then connect edges downward from a parent view or node to its child view or node according to the lattices, to either reduce one group-by attributes (i.e. SCT -> SC), or substitute one dimension attribute with the non-primary key attribute (i.e. SCT -> SdT). Repeat the connecting process for all the views. If the edge is drawn due to the substitution of a dimension attribute, we label that edge with a corresponding lower-case letter of the primary key attribute. This lower case letter on the edge between the parent and child views indicates that a table join is required to derive the child view using the result of the parent view. We call this lower case letter the table join link. The top element in the combined cube lattice is always the base level view. The bottom element in the combined cube lattice is always the *ALL* view, which is denoted as ( ). The combined cube lattice of the university warehouse for combining only the two dimensions *student* and *course* is given as Figure 10.



S=StdId, C=CourseNo, T=Term, n=StdName, g=Gender, d=CourseName

Figure 10 The Combined Cube Lattice of the University Warehouse (combined for only dimensions S and C)

Notice that for the 3-dimensional university data warehouse example, there are a total of 24 views in this combined cube lattice when only the dimension hierarchies *student* (S) and *course* (C) are included. As mentioned before, an *n*-dimensional data cube has 2^n main subviews. As the dimension *n* gets large in a data warehouse, the number of main subviews will grow exponentially. For example, a four-dimensional data cube has $2^4=16$ main subviews, a five-dimensional data cube has $2^5= 32$ main subviews. When considering all the dimension attributes for the *n* dimensions, the total number of views in the combined cube lattice will increase even faster. Thus, the view-selection problem to accommodate the dimension hierarchies in a data warehouse is much more complex than just considering the main subviews.

3.4 Partial Combined Cube Lattice

This thesis introduces the concept of partial combined cube lattice in order to reduce the number of warehouse views considered in the view-selection problem when dimension hierarchies are taken into account in a data warehouse. Although the total number of views in a combined cube lattice can be very large, only a portion of them need to be considered in the warehouse view-selection process due to either storage space constraints or business requirements.

In a data warehouse system, we can always identify a set of common warehouse queries as those that have either of the following characteristics: (1) high access frequency (which are accessed most frequently) or (2) a critical or high query value (which requires a fast response time). Note that some queries that have a critical or high query value and are included in the set of common queries may have low access frequencies.

The steps to generate the partial combined cube lattice are described as follows. On the combined cube lattice, given a set of common warehouse queries, identify and mark all the smallest views except the *ALL* view, which can be used to answer these common queries. From each of the marked views, highlight the edge from this view *v* to its ancestor views including the table join link denoted by a lower case letter, and mark all of its ancestor views as well. Repeat this marking process for all the ancestor views of view *v* until no more views are left to be marked. Then remove all the nodes that have not been marked as well as any dangling edges from the combined cube lattice. The resultant (marked) combined cube lattice is the partial combined cube lattice.

It is obvious that the top element of the partial combined cube lattice is always the base level view. For the purpose of reducing the views to be considered, we do not include the *ALL* view in the partial combined cube lattice even if the *ALL* view is in the list of views accessed by common queries. As described in a later section, the proposed view selection scheme always materializes the *ALL* view along with the selected the set. The importance of generating the partial combined cube lattice is to allow only relevant views, which is a subset of views in the combined cube lattice to be considered in the selection process and to provide scalability for the proposed selection technique. The views appearing on the partial combined cube lattice are called the candidate views. Note that each leaf node on the partial combined cube lattice are the views associated with a warehouse query, and each non-leaf node on the partial combined cube lattice may or may not be associated with a warehouse query.

Consider some of the warehouse queries and views with corresponding SQL statement to answer these queries in the university data warehouse below.

Q1: (ST) Get the list of all students who have maintained an average grade of 90 or above in any term.

```
Select   student_id, term, avg(grade) as avg_grade
From     grade
Group by student_id
Having   avg(g.grade) >= 90;
```

This query can be answered by the results of view ST.

Q2: (CT) Get the average grade for each cs60-300 level or above course each term.

```
Select  course_no, term, avg(grade) as avg_grade
From    grade
Where   course_no > = 'cs60-300'
Group by course_no, term;
```

This query can be answered by the result of view CT.

Q3: (Sd) Get the average grade for each student on each course taken, listed by student id and course name.

```
Select  g.student_id, c.course_name, avg(g.grade) as avg_grade
From    grade g, course c
Where   g.course_no = c.course_no
Group by g.student_id, c.course_name;
```

This query can be answered by the view Sd.

Q4: (SdT) Get the average grade for each female student on each course name taken in each term.

```
Select   g.student_id, c.course_name, g.term, avg(g.grade) as avg_grade
From     grade g, course c, student s
Where    g.course_no = c.course_no
and      g.student_id = s.student_id
and     s.gender = 'F'
Group by g.student_id, c.course_name, g.term;
```

This query can be answered by the result of view SdT.

Q5: (gdT) Get the average grade for both male and female students on each course name in each term.

```
Select   s.gender, c.course_name, g.term, avg(g.grade) as avg_grade
From     grade g, course c, student s
Where    g.course_no = c.course_no
And      g.student_id = s.student_id
Group by s.gender, c.course_name, g.term;
```

This query can be answered by the view gdT.

Q6: (nCT) Get the average grade for each student on each course in any term, listed by student name, course number and the term offered.

```
Select   s.name, g.course_no, g.term, avg(g.grade) as avg_grade
From     grade g, student s
Where    g.student_id = s.student_id
Group by s.name, g.course_no, g.term;
```

This query can be answered by the view nCT.

Suppose the above six queries are the common warehouse queries that are accessed in the university data warehouse and their corresponding access frequencies are as follows:

Q1	Q2	Q3	Q4	Q5	Q6
80	90	100	75	80	85

The partial combined cube lattice for the university data warehouse example has all the candidate views as shown in Figure 11. The number of rows in each of the candidate views is enclosed in the parenthesis beside the view. They are obtained using the sample fact table and the dimension tables data listed in Appendix A.

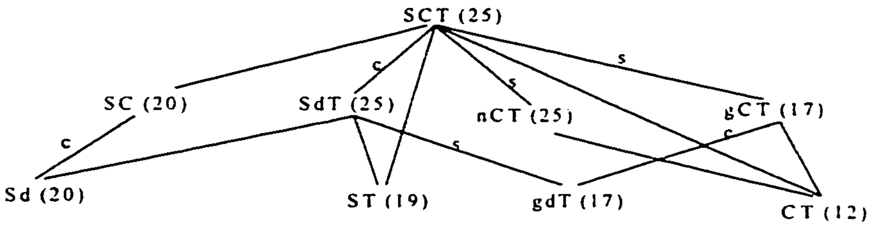


Figure 11 The Partial Combined Cube Lattice of the University Warehouse

3.5 *Applying View Selection Schemes to Example Warehouse*

Before presenting the proposed view selection scheme in this thesis, we first apply the greedy selection scheme introduced in [HaRaUl96] directly to the university data warehouse example. The thesis identifies some limitations in the greedy selection scheme and introduces the proposed approach for selecting the warehouse views to accommodate dimension hierarchies.

3.5.1 The Greedy Selection Scheme on Example Warehouse

The greedy selection scheme presented in [HaRaUl96] simply computes the benefit of a view by considering how it can improve the space cost of evaluating itself and other views, which depend on it. The scheme starts by selecting the base level view into a set. Then the benefit of each of the candidate views is calculated. The view with the maximum benefit considering the views that are already in the set is selected for materialization. In order to show the difference between the greedy selection scheme and the proposed selection scheme, the definition used in the greedy selection scheme to calculate the benefit of a view is given below. Let *C(v)* be the cost of view *v*. After selecting some set *S* of views, the benefit of view *v* relative to *S* denoted as *B(v, S)* is defined as follows:

1. For each $w \leq v$, define the quantity B_W by
   Let *u* be the view of least cost in *S* such that $w \leq u$.
   if $C(v) < C(u)$, then $B_W = C(u) - C(v)$, otherwise $B_W = 0$
2. Define $B(v, S) = \sum_{w \leq v} B_W$

Apply the greedy selection scheme to the partial combined cube lattice of Figure 11. Suppose we want to choose three views in addition to the base level view to materialize. We must make three successive choices of views. When calculating the benefit, we begin with the assumption that each view is evaluated using the base level view SCT and therefore has a cost of 25. The benefits of each view calculated at each round for our example are tabulated in Table 1. It can be seen from the table that the winner of the second round is the view gCT, which has the highest benefit of 24. If view gCT is materialized, we reduce its cost and that of each of its descendant views gdT and CT by 8. Since the space costs of view SdT and view nCT are the same as that of the base level view SCT, there is no benefit gained by materializing either of them. For the third round, the benefits of each candidate views are calculated considering the set of materialized views as {SCT, gCT}. The winner for the second round is the view SC with a maximum benefit of 10. When calculating the benefit for view gdT in the second round, its smallest ancestor view gCT with a space cost of 17 is used, therefore the benefit is (17-17)*1=0. Similarly, the benefit of view CT is calculated from its smallest ancestor view gCT instead of the base level view SCT, resulting (17-12)*1=5. For the fourth round, view ST is the winner with a maximum benefit of 6. Thus, the greedy selection for our example is the following views: SCT, gCT, SC and ST.

View	Second Round	Third Round	Fourth Round
SC	(25-20)*2=10	**(25-20)*2=10**	▨
SdT	(25-25)*4=0	(25-25)*3+0=0	(25-25)*2+0+0=0
nCT	(25-25)*2=0	(25-25)*1+0=0	(25-25)*1+0=0
gCT	**(25-17)*3=24**	▨	▨
Sd	(25-20)*1=5	(25-20)*1=5	(20-20)*1=0
ST	(25-19)*1=6	(25-19)*1=6	**(25-19)*1=6**
gdT	(25-17)*1=8	(17-17)*1=0	(17-17)*1=0
CT	(25-12)*1=13	(17-12)*1=5	(17-12)*1=5

Table 1 View Benefits at Each Round for Greedy Selection

From the above example, when applying the greedy selection [HaRaUl96] directly to the warehouse views the following limitations have been observed: No consideration was given to the common warehouse queries and their access frequencies. Each view on the lattice is treated equally irrespective of whether it is a main subview or a dimension subview. The number of rows in a view is the only factor considered for making the selection decision. When calculating the benefit gain of a dimension subview, there is no consideration given to time spent on table joins required to answer the dimension subview with its ancestor views.

3.5.2 Proposed Selection Scheme on Example Warehouse

This thesis proposes a view selection scheme that takes into consideration of the following elements:

- The common warehouse queries and their access frequencies
- The candidate views associated with a query are treated differently from other views. They are given a higher weight factor to increase their chance of being selected.
- The dimension subviews are treated differently from other views. They are also given a higher weight factor to increase their chance of being selected because of the need to perform table joins if they are not selected.
- The number of table joins involved in deriving a dimension subview from one of its ancestor views is also taken into account when calculating the benefit of a view.

Before presenting the proposed selection scheme, consider the definition of an *m*-join subview in Table 2 below.

Definition of An *m*-join Subview

Let *a* and *b* be the two views on a given lattice $\langle L, \unlhd \rangle$, and *a* is dependent on *b*, that is, $a_{<}$ *b*, if the minimum number of table joins required to drive *a* from *b* is *m*, where *m* is an integer, then we call *a* an *m*-join subview of *b*.

Table 2 Definition of An *m*-join Subview

For example, on the partial combined cube lattice in Figure 11, view Sd is a 1-join subview of SC, but it is a 0-join subview of SdT.

Notice that some of the candidate views on the partial combined cube lattice are associated with a query with known access frequencies while others are not. To apply the proposed selection technique, the following method to determine the access frequency of the views that are not associated directly with a query is used.

Let *u* be a candidate view on the partial combined cube lattice that is not associated with a query, and u is the immediate (smallest) ancestor of the following views: $v_1, v_2, \ldots v_n$. If the access frequencies of these views are $f(v_1), f(v_2), \ldots f(v_n)$ respectively, then the access frequency of u is the maximum of the frequency set.

That is, $f(u) = \max \{ f(v_1), f(v_2), \ldots f(v_n) \}$

Now, the access frequencies of the candidate views: SC, gCT, and SCT are as follows.

f(SC) = max{f(Sd)} = max{100} = 100

f(gCT) = max{f(gdT), f(CT)} = max{80, 90} = 90

f(SCT) = max{f(SC), f(SdT), f(nCT), f(gCT), f(ST)} = max{100, 75, 85, 90, 80}= 100

For convenience, we will denote the candidate views on the partial combined cube lattice in the form of *V(rows, frequency)* if view *V* is not associated with a query or *V(rows, frequency, Q_n)* if view *V* is associated with a query as shown in Figure 12.

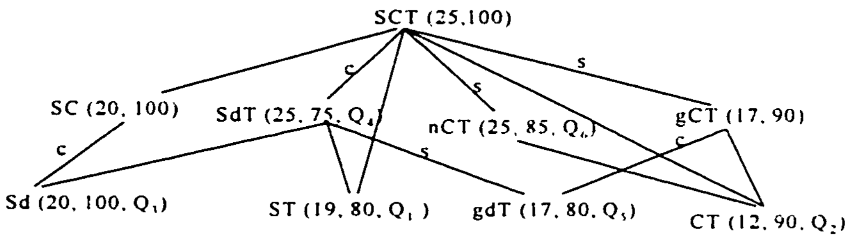


Figure 12 Partial Combined Cube Lattice with rows, frequencies indicated

Now, the thesis introduces the formula and definition that are used in the proposed selection scheme to calculate the benefit of a view relative to a selected set as shown in Table 3. This definition consists of three parts. The first part calculates the benefit of the view itself. The number of table joins required to derive the view from its smallest ancestor view in the selected set as well as whether the view itself is a query-associated view are considered when calculating the benefit. The second part calculates the benefits of all of its descendant views that are not already selected into the set. Again, the numbers of table joins required to derive the dimension subview from its ancestor views are considered when calculating the benefit. The third part takes the summation of the first two parts as the benefit of a view relative to a selected set.

Definition of The Benefit of A View

Let *C(v)* be the cost of view *v*, and let *F(v)* be the access frequency of view *v*. The benefit of view *v* relative to some selected set *S*, which we denote as *B(v,S)*, is defined as:

For view *v* itself, define the quantity B_v by

> Let *u* be the view of least space cost in *S* such that $v < u$,
> and *v* is *k*-join dimension subview of *u*,
> Let *C(u)* be the space cost of view *u*,
> If $C(v) < C(u)$, then $B_v = [C(u) - C(v)]*F(v)*(k + l + 1)$,
> > where l=1 if view *v* is associated with a query, otherwise l=0,
>
> Otherwise, $B_v = 0$.

For each view $w < v$, and $w \notin S$, define the quantity $B_{w<v}$ by

> Let *x* be the view of least space cost in *S* such that $w < x$,
> Let *C(x)* be the space cost of view *x*,
> and *w* is *m*-join dimension subview of *x*.
> Let *F(w)* be the access frequency of view *w*,
> and view *w* is *n*-join dimension subview of *v*
> If $C(v) < C(x)$, then $B_{w<v} = [C(x) - C(v)]*F(w)*(m - n + 1)$,
> Otherwise, $B_{w<v} = 0$.

Define $B(v,S) = B_v + \sum_{w<v} B_{w<v}$.

Table 3 Definition of Benefit of A View in the Proposed Selection Scheme

The proposed selection scheme is briefly described here. It starts by selecting the base level view into a set. Then the benefit of each candidate view on the partial combined cube lattice is calculated using the new definition of the benefit of a view in Table 3. The views with the maximum benefits in each round considering the views that are already in the set are selected for materialization.

Now, apply the proposed selection scheme to our university warehouse example with the partial combined cube lattice shown in Figure 12. Suppose we again want to select three (3) views to materialize in addition to the base level view SCT. The detailed calculation of the benefit for each view is listed below.

The first round:

$S = \{SCT\}$

The base level view SCT is selected into the set S.

The second round:

$B(SC, S) = B_{SC} + B_{Sd<SC} = (25-20)*100*1 + (25-20)*100*(1-1+1)=1000$

$B(SdT, S) = B_{SdT} + B_{Sd<SdT} + B_{ST<SdT} + B_{gdT<SdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT<nCT} = 0$

$\mathbf{B(gCT, S)} = B_{gCT} + B_{CT<gCT} + B_{gdT<gCT}$

$= (25-17)*9*(1+1) + (25-17)*90*(0-0+1) + (25-17)*80*(2-1+1)$

$=1440 + 720 + 1280 = \mathbf{3440}$

$B(Sd, S) = B_{Sd} = (25-20)*100*(1+1+1) = 1500$

$B(ST, S) = B_{ST} = (25-19)*80*(1+1) = 960$

$B(gdT, S) = B_{gdT} = (25-17)*80*(2+1+1) = 2560$

$B(CT, S) = B_{CT} = (25-12)*90*(1+1) = 2340$

$S = \{SCT, gCT\}$

The view gCT with a maximum benefit of 3440 is obviously the winner for the second round.

The third round:

$B(SC, S) = B_{SC} + B_{Sd<SC} = (25\text{-}20)*100*1 + (25\text{-}20)*100*(1-1+1)=1000$

$B(SdT, S) = B_{SdT} + B_{Sd<SdT} + B_{ST<SdT} + B_{gdT<SdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT<nCT} = 0$

$\mathbf{B(Sd, S)} = B_{Sd} = (25\text{-}20)*100*(1+1+1) = \mathbf{1500}$

$B(ST, S) = B_{ST} = (25\text{-}19)*80*(1+1) = 960$

$B(gdT, S) = B_{gdT} = (17\text{-}17)*80*(1+1+1) = 0$

$B(CT, S) = B_{CT} = (17\text{-}12)*90*(1+1) = 900$

S = { SCT, gCT, Sd }

The winner for the third round is view Sd with a maximum benefit of 1500.

The fourth round:

$B(SC, S) = B_{SC} = (25\text{-}20)*100*1=500$

$B(SdT, S) = B_{SdT} + B_{ST<SdT} + B_{gdT<SdT} = 0$

$B(nCT, S) = B_{nCT} + B_{CT<nCT} = 0$

$\mathbf{B(ST, S)} = B_{ST} = (25\text{-}19)*80*(1+1) = \mathbf{960}$

$B(gdT, S) = B_{gdT} = 0$

$B(CT, S) = B_{CT} = (17\text{-}12)*90*(1+1) = 900$

S = { SCT, gCT, Sd, ST }

From above calculation, the fourth choice is view ST with a maximum benefit of 960.

Thus, the selection of views using the proposed technique is the following views: SCT, gCT, Sd, and ST. The benefit of each view calculated using the proposed selection technique is listed in Table 4.

View Name	***Second Round***	***Third Round***	***Fourth Round***
SCT	▨	▨	▨
SC	1000	1000	500
SdT	0	0	0
nCT	0	0	0
gCT	**3440**	▨	▨
Sd	1500	**1500**	▨
ST	960	960	**960**
gdT	2560	0	0
CT	2340	900	900

Table 4 View Benefits at Each Round for Proposed Selection

The proposed technique produces a different selection set from the greedy selection made earlier. Since the major purpose of the warehouse view selection scheme is to improve the warehouse query response time, in the following section, the total query response time of the six common queries using the greedy selection is compared with the one using the selection resulting from the proposed scheme.

3.6 Comparing the Proposed Selection Scheme with the Greedy Scheme using Example Warehouse

3.6.1 Query Response Time

Before the query response times of the two underlying selection schemes are compared, query response time needs to be defined. In this thesis, the query response time is defined as the number of rows of the smallest view used to answer the query multiplied by the unit response time of answering one row in the system. If a table join is involved in answering a query, the number of rows of the smallest view to answer the query is doubled to accommodate the extra time taken for the table joins. If two table joins are involved in answering a query, the number of rows of the smallest view to answer the query is tripled. The definition of query response time and total query response time is listed in Table 5.

Definition of (Total) Query Response Time

Let *C(v)* be the space cost (number of rows) of the smallest view *v* which can be used to answer a query *Q*, let *t* be the unit response time to answer one row in the system, and let *k* be the number of table joins involved to answer query *Q* using view *v*, then the query response time RT_Q for *Q* is defined as follows.

$$RT_Q = C(v) * (1+k) * t$$

The total query response time TRT_Q of the system is the sum of all the query response time for each query. That is, Total Query Response Time

$$TRT_Q = \sum_{i=1}^{n} RT_{Qi} = \sum_{i=1}^{n} (C(v_i) * (1+k)) * t = t * \sum_{i=1}^{n} (C(v_i) * (1+k))$$

Table 5 Definition of (Total) Query Response Time

3.6.2 Calculation of Total Query Response Time for Greedy Selection

Greedy Selection: **S = {SCT, gCT, SC, ST}**

Queries can be directly answered from the set *S* of materialized views: Q_1

Queries can be answered from *S* without table joins: $Q_{2.}$

Queries can be answered from *S* with table joins: Q_3, Q_4, Q_5 and Q_6

Total query response time = $RT_{Q1} + RT_{Q2} + RT_{Q3} + RT_{Q4} + RT_{Q5} + RT_{Q6}$

= [C(ST) + C(gCT) + C(SC)*2 + C(SCT)*2 + C(gCT)*2 + C(SCT)*2] * t

= (19 + 17 + 20*2 + 25*2 + 17*2 + 25*2) * t

= 210t

3.6.3 Calculation of Total Query Response Time for Proposed Selection

Proposed Selection: **S = {SCT, gCT, Sd, ST}**

Queries can be directly answered from the set *S* of materialized views : Q_1 and Q_3

Queries can be answered from *S* without table joins: Q_2.

Queries can be answered from *S* with table joins: Q_4 , Q_5 and Q_6

Total query response time = $RT_{Q1} + RT_{Q2} + RT_{Q3} + RT_{Q4} + RT_{Q5} + RT_{Q6}$

= [C(ST) + C(gCT) + C(Sd) + C(SCT)*2 + C(gCT)*2 + C(SCT)*2] * t

= (19 + 17 + 20 + 25*2 + 17*2 + 25*2) * t

= 190t

From the above comparison, we can see the proposed selection resulted in better total query response time, approximately 9.53% improvement in this example.

4 Proposed View Selection Scheme

4.1 *The Cost/Benefit Model*

To answer a query Q using a view V, we need to process view V table. The space cost for answering Q is a function of the number of rows of the table V used to answer Q. In this thesis, the simple cost/benefit model is chosen. That is, the space cost for answering Q is the number of rows of the table V that must be processed to construct the result of Q.

Space Cost of answering Query $Q = |V|$

If a query Q can be answered by both view V_i and view V_j, and let $F(Q)$ be the access frequency of the query, the benefit gain of answering the query Q using view V_j versus view V_i is the difference of the space costs of answering the query with the two views multiplied by the access frequency of the query.

Benefit Gain = $k * [\,|V_j| - |V_i|\,] * F(Q)$ where k is constant factor

This cost/benefit model is used to define the benefit of a view in the proposed warehouse view selection scheme.

4.2 *The Proposed View Selection Scheme and Algorithms*

The proposed view selection scheme to accommodate dimension hierarchies in a data warehouse is described below.

Given a set of common warehouse queries and their access frequencies, all views on the combined cube lattice, and some storage space constraints, we first construct a partial combined cube lattice by following steps described in Chapter 3. The views contained in the partial combined cube lattice are the only relevant views that are considered for selection. The scheme starts by selecting the base level view into a set. Then the benefit of each of the remaining candidate views on the partial combined cube lattice is calculated using the new definition of the benefit of a view listed in Table 3. The view with the maximum benefit considering the views that are already in the set is selected for materialization. The selection process is repeated until the number of views to be selected is reached under the storage space constraints.

It is recommended that the *ALL* view be materialized always although it is not included on the partial combined cube lattice as a candidate view because it has only a single value and the space cost is minimal.

The algorithm proposed here is called ***View-Selection-DH Greedy*** algorithm, which is a modified version of the greedy algorithm. The input to the algorithm is a set of common warehouse queries and their access frequencies, the combined cube lattice and the storage space constraint. The output of the algorithm is a set of selected views to be materialized in a data warehouse.

The first step of the algorithm is to generate the partial combined cube lattice from a given set of common queries and the combined cube lattice. Only the views appearing on the partial combined cube lattice are considered in order to reduce the number of relevant views being considered in the view-selection process. The algorithm for generating the partial combined cube lattice is presented in Table 6. In this thesis, the space cost of a view is the number of rows in the view. Suppose that there is a limit k on the number of views, in addition to the base level view that we may select due to the storage space constraint. After selecting some set S of views, the benefit of view v relative to S, which we denote $B(v,S)$, is defined earlier in Table 3. The proposed ***View-Selection-DH Greedy*** algorithm for selecting a set of k views to materialize in addition to the base level view is given in Table 7.

```
Algorithm    Partial_CombCubeLattice (Q, L, L')
Input:       A set of common warehouse queries $Q_I = \{Q_1, Q_2, \ldots Q_n\}$
             The combined cube lattice $L_j = \{ V_1, V_2, \ldots V_m\}$
Output:      Partial combined cube lattice $L' \subseteq L_j$
Begin
             $L' \leftarrow \varnothing$ // initial the empty set of partial combined cube lattice L' //
             Identify the set of smallest views $V_i =\{ V_1, V_2, \ldots V_n \}$ to
             Answer the set of common queries $Q_i = \{Q_1, Q_2, \ldots Q_n\}$
             // exclude the ALL view //
             for each view $u \in V_I$
                     if u is the ALL view,
                     then $V_i = V_I - u$
             $V_p \leftarrow \varnothing$ // initial the empty set of ancestor views $V_p$ //
             For each view $v \in V_i$
             Begin
                     $V_p$ = ancestor (v)
                     $L' = L' \cup V_p$
             End
             // the resulting lattice is the partial combined cube lattice L' //
             return L'
End
```

Table 6 Partial Combined Cube Lattice Algorithm

```
Algorithm   View-Selection-DH Greedy (S)
Input:      A set of common warehouse queries $Q_I = \{Q_1, Q_2, \ldots Q_n\}$
            And their access frequencies $F_i = \{F_1, F_2, \ldots F_n\}$
            The combined cube lattice $L_j = \{ V_1, V_2, \ldots V_m\}$
Output:     A set of selected views $S \subseteq L_j$
Begin
            $S \leftarrow \varnothing$ { initial the empty set of S }
            // construct partial combined cube lattice L' //
            $L'_j$ = Partial_CombCubeLattice($Q_i$, $L_j$, $L'_j$)
            Let $V_j$ be the set of all candidate views on the partial combined
            cube lattice $L'_j$
            S = { base level view}
            for  i = 1 to k do
            begin
                 for each view $v \in V_j$ and $v \notin S$ do
                      Compute B(v, S), the benefit of view v relative to S as
                      described in Table 3.
                      Let Vs be the view such that B(Vs, S) = Max( B(v, S) )
                      S = S U Vs
            End
            $V_0 \leftarrow$ the *ALL* view
            $S = S \cup V_0$
            Return S  //the resulting set S is the selection of views for materialization//
End
```

Table 7 View-Selection-DH Greedy Algorithm

4.3 Implementation and Experimentation

The greedy algorithm [HaRaUl96] has been considered as a benchmark in the warehouse view selection area. Harinarayan *et al.* has shown that the greedy algorithm has a performance guarantee of at least 63% of the optimal solution. that is. for no lattice whatsoever does the greedy algorithm give a benefit less than 63% of the optimal benefit. In order to compare the proposed View-Selection-DH greedy algorithm with the greedy algorithm. the View-Selection-DH greedy algorithm has been successfully implemented. Several runs of the implemented programs against the university data warehouse system have been conducted. To examine the behavior of the proposed selection scheme. two experimentation lattices are used during the test runs. The first experimentation lattice has a fewer candidate views than the second experimentation lattice. When determining the common warehouse queries, the access frequency is not the only factor to consider. but the value of a query is also taken into consideration. Our experimentation examples also include some queries that have lower access frequencies but have high query values in the common query set.

4.3.1 Case I - Experimentation Lattice, Input and Output Data

The experimentation lattice for Case I is shown in Figure 13. The ancestor/descendant view dependency relationship on this lattice and the number of table joins required to derive each of the subviews from its ancestor views are represented in Table 8. The third column on Table 8 represents the view numbers of the ancestor views and the number of table joins required to derive the descendant view from each of the ancestor views. Each pair of the ancestor view number and the number of table joins required are separated by the delimiter "&" in this column. For example, the subview Sd with a view number of 6 has three ancestors with the view number 1, 2, and 3, and the numbers of table joins required to derive the subview Sd from each of the ancestor views are 1, 1 and 0, respectively. The input data for this lattice is shown in Table 9. A "1" in the *IsQuery* column of Table 9 indicates that the view is directly associated with a common warehouse query. The last column of the table indicates the query access frequencies. The output benefits calculated for each view in each round for both the greedy algorithm and the proposed algorithm are tabulated in Table 10 and Table 11, respectively.

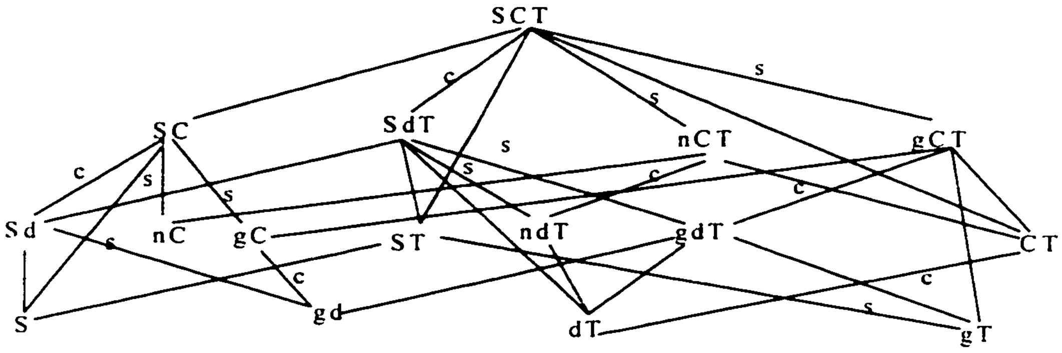


Figure 13 Experimentation Lattice for Case I

ViewName	No	&ancestor1,k-join&ancestor2,k-join…&ancestorN,k-join
SCT	1	
SC	2	&1,0
SdT	3	&1,1
nCT	4	&1,1
gCT	5	&1,1
Sd	6	&1,1&2,1&3,0
nC	7	&1,1&2,1&4,0
gC	8	&1,1&2,1&5,0
ST	9	&1,0&3,0
ndT	10	&1,2&3,1&4,1
gdT	11	&1,2&3,1&5,1
CT	12	&1,0&4,1&5,0
S	13	&1,0&2,0&3,0&6,0&9,0
gd	14	&1,2&2,2&3,1&5,1&6,1&8,1&11,0
dT	15	&1,1&3,0&4,1&5,1&10,0&11,0&12,1
gT	16	&1,1&3,1&5,0&9,1&11,0

Table 8 View Dependency Relationship for Case I Lattice

ViewNo	NumOfRow	Name	NumOfAncestor	IsQuerry	Frequency
1	25	SCT	0	0	75
2	20	SC	1	1	75
3	25	SdT	1	0	50
4	25	nCT	1	0	65
5	17	gCT	1	1	60
6	20	Sd	3	0	45
7	20	nC	3	1	40
8	11	gC	3	0	45
9	19	ST	2	1	50
10	25	ndT	3	0	35
11	17	gdT	3	1	45
12	12	CT	3	1	65
13	5	S	5	1	10
14	11	gd	7	1	45
15	12	dT	7	1	35
16	10	gT	5	1	45

Table 9 Input Data for Case I Lattice

View No.	*View Name*	*#2 Round*	*#3 Round*	*#4 Round*	*#5 Round*	*#6 Round*	*#7 Round*	*#8 Round*	*#9 Round*
1	SCT	▨	▨	▨	▨	▨	▨	▨	▨
2	SC	30	20	▨	▨	▨	▨	▨	▨
3	SdT	0	0	0	0	0	0	0	0
4	nCT	0	0	0	0	0	0	0	0
5	gCT	56	▨	▨	▨	▨	▨	▨	▨
6	Sd	15	10	0	0	0	0	0	0
7	nC	5	5	0	0	0	0	0	0
8	gC	28	12	12	12	▨	▨	▨	▨
9	ST	18	12	7	6	6	6	6	▨
10	ndT	0	0	0	0	0	0	0	0
11	gdT	32	0	0	0	0	0	0	0
12	CT	26	10	10	10	10	▨	▨	▨
13	S	20	20	15	▨	▨	▨	▨	▨
14	gd	14	6	6	6	0	0	0	0
15	dT	13	5	5	5	5	0	0	0
16	gT	15	7	7	7	7	7	▨	▨

Table 10 Output of View Benefits from Greedy Algorithm for Case I

View No.	*View Name*	*#2 Round*	*#3 Round*	*#4 Round*	*#5 Round*	*#6 Round*	*#7 Round*	*#8 Round*	*#9 Round*
1	SCT	▨	▨	▨	▨	▨	▨	▨	▨
2	SC	1675	1225	▨	▨	▨	▨	▨	▨
3	SdT	0	0	0	0	0	0	0	0
4	nCT	0	0	0	0	0	0	0	0
5	gCT	5120	▨	▨	▨	▨	▨	▨	▨
6	Sd	950	500	0	0	0	0	0	0
7	nC	600	600	0	0	0	0	0	0
8	gC	2520	540	540	540	270	270	270	270
9	ST	930	660	610	610	610	610	610	▨
10	ndT	0	0	0	0	0	0	0	0
11	gdT	3800	0	0	0	0	0	0	0
12	CT	2145	825	825	▨	▨	▨	▨	▨
13	S	400	400	300	300	300	300	300	280
14	gd	2520	810	810	810	▨	▨	▨	▨
15	dT	1365	525	525	0	0	0	0	0
16	gT	2025	630	630	630	630	▨	▨	▨

Table 11 Output of View Benefits from Proposed Algorithm for Case I

From the output results listed in Table 10 and Table 11, we get the following selections for both the greedy selection and the proposed selection for different k values (number of materialized views).

When k = 4,	Greedy Selection	S = {1, 5, 2, 13}
	Proposed Selection	S' = {1, 5, 2, 12}
When k = 5,	Greedy Selection	S = {1, 5, 2, 13, 8}
	Proposed Selection	S' = {1, 5, 2, 12, 14}
When k = 6,	Greedy Selection	S = {1, 5, 2, 13, 8, 12}
	Proposed Selection	S' = {1, 5, 2, 12, 14, 16}
When k = 7,	Greedy Selection	S = {1, 5, 2, 13, 8, 12, 16}
	Proposed Selection	S' = {1, 5, 2, 12, 14, 16, 9}
When k = 8,	Greedy Selection	S = {1, 5, 2, 13, 8, 12, 16, 9}
	Proposed Selection	S' = {1, 5, 2, 12, 14, 16, 9, 13}
When k = 9,	Greedy Selection	S = {1, 5, 2, 13, 8, 12, 16, 9, 3}
	Proposed Selection	S' = {1, 5, 2, 12, 14, 16, 9, 13, 8}

4.3.2 Case II - Experimentation Lattice, Input and Output Data

The experimentation lattice for Case II is shown in Figure 14. The ancestor/descendant view dependency relationship on this lattice and the number of table joins required to derive each of the subviews from its ancestor views are represented in Table 12. The third column on Table 12 represents the view numbers of the ancestor views and the number of table joins required to derive the descendant view from each of the ancestor views. Each pair of the ancestor view number and the number of table joins required are separated by the delimiter "&" in this column. The input data for this lattice is shown in Table 13. A "1" in the *IsQuery* column of Table 13 indicates that the view is directly associated with a common warehouse query. The last column of the table indicates the query access frequencies. The output benefits calculated for each view in each round for both the greedy algorithm and the proposed algorithm are tabulated in Table 14 and Table 15, respectively.

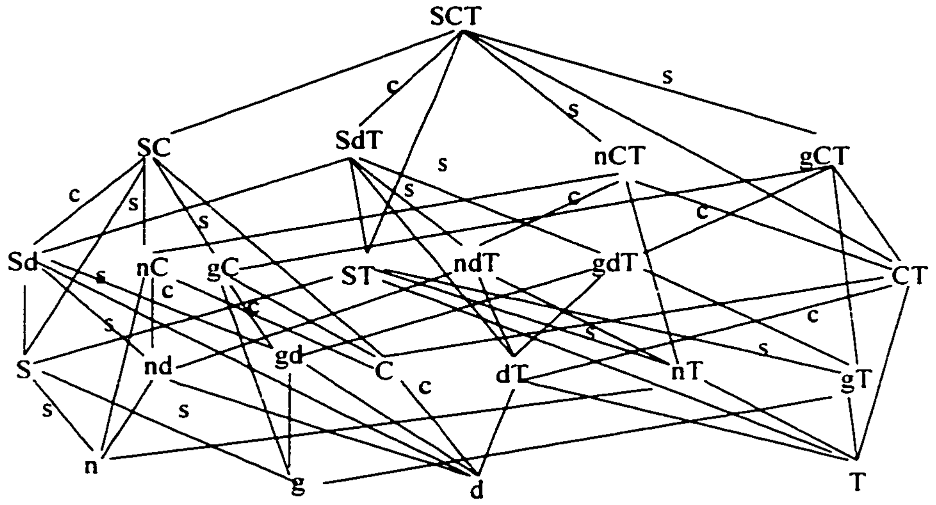


Figure 14 Experimentation Lattice for Case II

ViewName	no	&ancestor1,k-join&ancestor2,k-join … &ancestorN,k-join
SCT	1	
SC	2	&1,0
SdT	3	&1,1
nCT	4	&1,1
gCT	5	&1,1
Sd	6	&1,1&2,1&3,0
nC	7	&1,1&2,1&4,0
gC	8	&1,1&2,1&5,0
ST	9	&1,0&3,0
ndT	10	&1,2&3,1&4,1
gdT	11	&1,2&3,1&5,1
CT	12	&1,0&4,1&5,0
S	13	&1,0&2,0&3,0&6,0&9,0
nd	14	&1,2&2,2&3,1&4,1&6,1&7,1&10,0
gd	15	&1,2&2,2&3,1&5,1&6,1&8,1&11,0
C	16	&1,0&2,0&4,0&5,0&7,0&8,0&12,0
dT	17	&1,1&3,0&4,1&5,1&10,0&11,0&12,1
nT	18	&1,1&3,1&4,0&9,1&10,0
gT	19	&1,1&3,1&5,0&9,1&11,0
n	20	&1,1&2,1&3,1&4,1&6,1&7,0&9,1&10,0&13,1&14,0&18,0
g	21	&1,1&2,1&3,1&5,0&6,1&8,0&9,1&11,0&13,1&15,0&19,0
d	22	&1,1&2,1&3,0&4,1&5,1&6,0&7,1&8,1&10,0&11,0&12,1&14,0&15,0&16,1&17,0
T	23	&1,0&3,0&4,0&5,0&9,0&10,0&11,0&12,0&17,0&18,0&19,0

Table 12 View Dependency Relationship for Case II Lattice

ViewNo	NumOfRow	Name	NumOfAncestor	IsQuerry	Frequency
1	25	SCT	0	0	90
2	20	SC	1	0	75
3	25	SdT	1	1	80
4	25	nCT	1	0	90
5	17	gCT	1	1	85
6	20	Sd	3	1	70
7	20	nC	3	0	55
8	11	gC	3	1	75
9	19	ST	2	0	85
10	25	ndT	3	1	90
11	17	gdT	3	1	65
12	12	CT	3	1	45
13	5	S	5	0	40
14	20	nd	7	0	55
15	11	gd	7	1	50
16	6	C	7	0	55
17	12	dT	7	0	55
18	19	nT	5	1	85
19	10	gT	5	0	30
20	5	n	11	1	40
21	2	g	11	1	30
22	6	d	15	1	55
23	6	T	11	1	20

Table 13 Input Data for Case II Lattice

View No.	View Name	#2 Round	#3 Round	#4 Round	#5 Round	#6 Round	#7 Round	#8 Round	#9 Round	#10 Round
1	SCT	▨	▨	▨	▨	▨	▨	▨	▨	▨
2	SC	55	30	20	20	▨	▨	▨	▨	▨
3	SdT	0	0	0	0	0	0	0	0	0
4	nCT	0	0	0	0	0	0	0	0	0
5	gCT	88	▨	▨	▨	▨	▨	▨	▨	▨
6	Sd	35	20	10	10	0	0	0	0	0
7	nC	25	15	10	10	0	0	0	0	0
8	gC	70	30	24	14	14	▨	▨	▨	▨
9	ST	42	24	12	12	12	12	▨	▨	▨
10	ndT	0	0	0	0	0	0	0	0	0
11	gdT	56	0	0	0	0	0	0	0	0
12	CT	65	25	25	▨	▨	▨	▨	▨	▨
13	S	60	52	▨	▨	▨	▨	▨	▨	▨
14	nd	15	10	5	5	0	0	0	0	0
15	gd	42	18	12	7	7	0	0	0	0
16	C	38	22	22	12	12	10	10	▨	▨
17	dT	39	15	15	0	0	0	0	0	0
18	nT	18	12	6	6	6	6	0	0	0
19	gT	45	21	14	9	9	9	9	9	▨
20	n	20	20	0	0	0	0	0	0	0
21	g	23	15	3	3	3	3	3	3	3
22	d	19	11	11	6	6	5	5	0	0
23	T	19	11	11	6	6	6	6	6	4

Table 14 Output of View Benefits from Greedy Algorithm for Case II

View No.	View Name	#2 Round	#3 Round	#4 Round	#5 Round	#6 Round	#7 Round	#8 Round	#9 Round	#10 Round
1	SCT	▨	▨	▨	▨	▨	▨	▨	▨	▨
2	SC	3000	1675	1475	1475	650	650	650	650	▨
3	SdT	0	0	0	0	0	0	0	0	0
4	nCT	0	0	0	0	0	0	0	0	0
5	gCT	7880	▨	▨	▨	▨	▨	▨	▨	▨
6	Sd	3200	2000	1800	1800	▨	▨	▨	▨	▨
7	nC	2050	1500	1100	1100	550	550	550	550	0
8	gC	6930	2040	2040	▨	▨	▨	▨	▨	▨
9	ST	1980	1500	1260	1260	1060	550	550	550	550
10	ndT	0	0	0	0	0	0	0	0	0
11	gdT	6160	0	0	0	0	0	0	0	0
12	CT	3575	1375	1375	825	825	825	▨	▨	▨
13	S	2200	1600	800	800	600	600	600	600	600
14	nd	1775	1225	825	825	0	0	0	0	0
15	gd	5180	1740	1740	0	0	0	0	0	0
16	C	2090	1210	1210	550	550	550	550	275	275
17	dT	3120	1200	1200	650	650	650	0	0	0
18	nT	2130	2010	1530	1530	1530	▨	▨	▨	▨
19	gT	2100	560	560	380	380	380	280	280	280
20	n	2400	2400	▨	▨	▨	▨	▨	▨	▨
21	g	2070	900	900	540	540	540	540	540	540
22	d	3135	1815	1815	825	825	825	825	▨	▨
23	T	760	440	440	440	440	440	240	240	240

Table 15 Output of View Benefits from Proposed Algorithm for Case II

From the output results listed in Table 12 and Table 13, we get the following selections for both the greedy selection and the proposed selection for different k values (number of materialized views).

When k = 4,	Greedy Selection	S = {1, 5, 13, 12}
	Proposed Selection	S’ = {1, 5, 20, 8}
When k = 5,	Greedy Selection	S = {1, 5, 13, 12, 2}
	Proposed Selection	S’ = {1, 5, 20, 8, 6}
When k = 6,	Greedy Selection	S = {1, 5, 13, 12, 2, 8}
	Proposed Selection	S’ = {1, 5, 20, 8, 6, 18}
When k = 7,	Greedy Selection	S = {1, 5, 13, 12, 2, 8, 9}
	Proposed Selection	S’ = {1, 5, 20, 8, 6, 18, 12}
When k = 8,	Greedy Selection	S = {1, 5, 13, 12, 2, 8, 9, 16}
	Proposed Selection	S’ = {1, 5, 20, 8, 6, 18, 12, 22}
When k = 9,	Greedy Selection	S = {1, 5, 13, 12, 2, 8, 9, 16, 19}
	Proposed Selection	S’ = {1, 5, 20, 8, 6, 18, 12, 22, 2}
When k = 10,	Greedy Selection	S = {1, 5, 13, 12, 2, 8, 9, 16, 19, 23}
	Proposed Selection	S’ = {1, 5, 20, 8, 6, 18, 12, 22, 2, 13}

4.4 Evaluation and Comparison

4.4.1 Evaluation and Comparison for Case I Test Runs

The total query response time for different k values (number of materialized views) using both the greedy algorithm and the proposed algorithm are calculated and the results are tabulated in Table 16. The proposed algorithm provides slightly better total query response time than the greedy algorithm for this example. The last column of the table indicates the improvement ratio of the total query response time using the proposed algorithm versus the greedy algorithm. The average improvement ratio is approximately 3.07% in the test runs of the Case I lattice.

k Value	Greedy Algorithm Total Query Response Time (T_1)	Proposed Algorithm Total Query Response Time (T_2)	Improvement Ratio = $(T_1-T_2)/T_1$ * 100%
4	243t	243t	0
5	231t	220t	4.76
6	216t	213t	1.39
7	209t	206t	1.44
8	203t	192t	5.42
9	203t	192t	5.42

Table 16 Comparison of Total Query Response Time for Different k Values in Case I

4.4.2 Evaluation and Comparison for Case II Test Runs

The total query response time for different k values (number of materialized views) using both the greedy algorithm and the proposed algorithm are calculated and the results are tabulated in Table 17. The proposed algorithm provides consistently better total query response time than the greedy algorithm for this example. The last column of the table indicates the improvement ratio of the total query response time using the proposed algorithm versus the greedy algorithm. The average improvement ratio is approximately 10.68% in the test runs of the Case II lattice and it is much higher than the one resulted in the test runs of the Case I lattice.

k Value	Greedy Algorithm Total Query Response Time (T_1)	Proposed Algorithm Total Query Response Time (T_2)	Improvement Ratio = (T_1-T_2)/T_1 * 100%
4	395t	381t	3.54
5	385t	361t	6.23
6	365t	318t	12.88
7	353t	308t	12.75
8	343t	294t	14.29
9	341t	294t	13.78
10	337t	299t	11.28

Table 17 Comparison of Total Query Response Time for Different k Values in Case II

4.4.3 Special Notes and Observations

From the comparison results in Table 16 and Table 17, the Case II test runs provided better performance results than the Case I test runs. Many factors contribute to these results. For instance, the Case II experimentation lattice includes almost all the possible views in the university data warehouse. The common warehouse query sets (query associated views and access frequencies) are different between these two cases. The view deeper down the partial combined cube lattice has more ancestor views. The number of table joins required to derive a dimension subview is also a major factor contributing to the final performance results. However, based on the experimentation conducted, we cannot provide a systematic prediction on what degree each of the factors affects the performance of the selection scheme. This can be considered for future work.

The proposed selection technique is a static design as is the greedy selection scheme. When common warehouse queries are changed or the warehousing system design is changed, the proposed selection technique will have to be applied again to select a new set of views for materialization. Both the proposed ***View-Selection-DH Greedy*** algorithm and the greedy algorithm are polynomial time algorithms. Using a big "*O*" notation, the performance bound is $O(n^2)$ where *n* is the number of possible views in the data warehouse. However, the generation of the partial combined cube lattice involves some overhead cost in the overall view selection process due to some extra time taken in its generation. Although this overhead cost in generating the partial combined cube lattice has to be considered, it is believed that the performance gain using the proposed selection technique outweighs the overhead cost especially when a system is fairly stable and the selection technique does not have to be applied very frequently.

5 Contributions and Conclusions

5.1 Contributions of the Thesis

The thesis contributes to data warehouse view-selection area by originally introducing the concept of the partial combined cube lattice to reduce the number of the views to be considered for materialization in a data warehousing environment. An algorithm is presented to construct the partial combined cube lattice from a given set of common warehouse queries and the combined cube lattice. As the number of warehouse dimensions increases, the number of main subviews on a cube lattice increases exponentially. When taking warehouse dimension hierarchies into consideration in the view selection problem, the total number of subviews on the combined cube lattice gets even larger. Generating the partial combined cube lattice allows us consider only the relevant views for materialization and provides scalability for the proposed selection technique. Another major contribution of the thesis is defining the benefit of a view relative to a selected set using a simple but practical cost/benefit model. This definition overcomes the limitations of the benefit definition given in [HaRaUl96]. It incorporates many important factors such as common warehouse queries and access frequencies, dimension subviews with degree of table joins, that must be considered in the warehouse view materialization problem. Overall, this thesis contributes a solution for the view selection problem in a data warehousing environment to accommodate dimension hierarchies by providing a new selection technique, the ***View-Selection-DH Greedy*** algorithm.

5.2 Conclusions

This thesis investigates the view selection problem in a data warehousing environment and reviews recent work in the related area. It proposes a new technique to select the warehouse views to accommodate dimension hierarchies. The lattice framework with complex groupings involving dimension hierarchies is used to establish the combined cube lattice, which in turn is used to generate the partial combined cube lattice. The introduction of the partial combined cube lattice allows us consider only the relevant views on the lattice and ensures the scalability of the selection technique in a warehouse environment when the number of dimensions and their attributes become large. Examples to demonstrate the warehousing concept and the warehouse view selection scheme are given. The proposed selection technique improves data warehousing system performance but some overhead cost is involved when generating the partial combined cube lattice. The benefits of the proposed algorithms are demonstrated by comparing with the greedy algorithm that the proposed selection scheme has resulted in better total query response time than the greedy algorithm from the experimentation conducted.

5.3 Future Work

Some of the future work for this view selection scheme are to:

1) compare the performance of the proposed selection technique with the performance of optimal solution that materializes all views. This comparison forms a basis for a more realistic comparison with the performance of the greedy algorithm. Performing a more theoretical comparison that possibly proves this approach has a performance improvement higher than 63% of the optimal performance achieved by the greedy proves in a more concrete way that the approach performs better than the greedy.
2) further evaluate the performance of the proposed selection technique hopefully in a much larger data warehouse environment and predict what factors highly affect the total benefit of the system with this approach.
3) include indexes in the view selection process and generate dynamic warehouse view materialization techniques in a data warehousing environment.

References

[ADS96] Archer Decision Sciences, *Star Schema 101*. White Paper. Available at URL http://members.aol.com/nraden/str101.htm

[Agetal96] S. Agrawal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Ramakrishnan and S. Sarawagi. "*On the Computation of Multidimensional Aggregates*" In Proceedings of the 22nd International Conference on VLDB, Mumbai (Bombay), India, Sept. 1996.

[BaPaTe97] E. Baralis, S. Paraboschi, and E. Teniente. "*Materialized Views Selection in a Multidimensional Database*". Proceedings of the 23rd VLDB Conference, page 156-165, Athens, Greece, 1997.

[ChDa97] S. Chaudhuri and U. Dayal. "*An Overview of Data Warehousing and OLAP Technology*" ACM SIGMOD Record 26(1), March 1997.

[ChSh94] S. Chaudhuri and K. Shim. "*Including Group-By in Query Optimization*". In Proceedings of the 20th International Conference on VLDB, pages 354-366, Santiago, Chile, 1994.

[Ez97a] C. Ezeife. "*A Uniform Approach for Selecting Views and Indexes in a Data Warehouse*", Proceedings of the 2nd International Database Engineering and Applications Symposium (IDEAS 97), pages 151-160, IEEE Publication, Montreal, August 1997.

[Ez97b] C. Ezeife. "*Accommodating Dimension Hierarchies in a Data Warehouse View/Index Selection Scheme*", Proceedings of the 6th International Conference on Information Systems Development - Methods and Tools, Theory and Practice, pages 195-211, Boise, Plennum Press Publishers, August 1997.

[Ez98] C. Ezeife. "*Optimizing Partition-Selection Scheme for Warehouse Views*", submitted to the Information Systems Magazine, University of Economics in Wroclaw, Poland, and Loyola University, Chicago, October 1998

[EzBa98] C. Ezeife and S. Baksh. "*A Partition-Selection Scheme for Warehouse Views*", Proceedings of the 9th International Conference on Computing and Information (ICCI 98), Winnipeg, Canada, IEEE Computer Society Publication, pages 9-16, June 17-20, 1998

[Gretal96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. "*Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals*", Proceedings of the 12th International Conference on Data Engineering, pages 152-159, 1996.

[Gu97] H. Gupta. "*Selection of Views to Materialize in a Data Warehouse*". Proceedings of the International Conference on Database Theory, Athens, Greece, January 1997.

[Guetal97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. "*Index Selection for OLAP*". Proceedings of the International Conference on Data Engineering, Binghamton, UK, April 1997.

[GuHaQu95] A. Gupta, V. Harinarayan, and D. Quass. "*Aggregate-Query Processing in Data Warehousing Environments*". Proceedings of the 21st International VLDB Conference, pages 358-369, 1995.

[Hu97] N. Huyn. "*Multiple-View Self-Maintenance in a Data Warehousing Environments.*" To appear in the Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.

[HaRaUl96] V. Harinarayan, A. Rajaraman, J. Ullman. "*Implementing Data Cubes Efficiently.*" Proceedings of ACM SIGMOD International Conference on Management of Data , Montreal, Canada, June 1996.

[In96] W. H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, Inc., Second Edition, 1996

[JoSh96] T. Johnson and D. Shasha. "*Hierarchically Split Cube Forests for Decision Support: description and tuned design*" 1996. Draft.

[Ki96] Ralph Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, Inc., 1996

[LaQuAd97] W. Labio, D. Quass, B. Adelberg. *"Physical Database Design for Data Warehousing"*. Proceedings of the International Conference on Data Engineering, Binghamton, UK, April 1997.

[MuQuMu97] I. Mumick, D. Quass, B. Mumick. *"Maintenance of Data Cubes and Summary Tables in a Warehouse."* Proceedings of the ACM SIGMOD Conference, Tuscon, Arizona, May 1997.

[OzVa91] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991

[Roetal96] K. Ross, P. Deshpande, J. Naughton, and K. Ramaswamy. *"Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time"*. SIGMOD Conference, pages 47-458, 1996.

[SaAgGu96] S. Sarawagi, R. Agrawal and A. Gupta, *"On Computing the Data Cube"*. Research Report 10026, IBM Almaden Research Center, San Jose, California, 1996.

[ShSrAg97] K. Shim, R. Srikant, R. Agrawal, *"High-dimensional Similarity Joins"*. Proceedings of the 13th International Conference on Data Engineering, Birmingham, U.K., April 1997.

[Shetal96] A. Shukla, P. Deshpande, J. Naughton and K. Ramasamy. *"Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies"* In Proceedings of the 22nd International Conference on VLDB, pages 522-531, Mumbai, September 1996.

[STG95] Stanford Technology Group, Inc. *Designing the Data Warehouse on Relational Databases.* White Paper.

[Ta97] R. Tanler. *The Intranet Data Warehouse.* John Wiley & Sons, Inc., 1997

[TiCh96] S. Tideman and R. Chu. *"Building Efficient Data Warehouses: Understanding the Issues of Data Summarization and Partitioning"*. Proceedings of the 21st Annual SAS Users Group International Conference, SUGI 21, Vol. 1, page 520-527, 1996.

[TPC93] *The Benchmark Handbook for Database and Transaction Processing*

Systems. 2nd Edition. J. Gray (ed.). Morgan Kaufmann Publishers, 1993 or at http://www.tpc.org/

[Wi95] J. Widom, "*Research Problems in Data Warehousing.*" Proceedings of the 4th International Conference on Information and Knowledge Management (CIKM), November 1995.

[YaLa95] W. P. Yan and P. A. Larson. "*Eager Aggregation and Lazy Aggregation*". Proceedings of the 21st International VLDB Conference, pages 345-357, 1995.

[Zhetal95] Y. Zhuge, H. Garcia-Monlina, J. Hammer, and J. Widom. "*View Maintenance in a Warehousing Environment.*" SIGMOD Conference, pages 316-327, 1995.

Appendix A

-- Fact Table: grade --

```
STUDENT COURSE_NO  TERM          GRADE
------- ---------- ------- ----------
C0001   CS60-212   1996W            90
C0001   CS60-255   1997W            92
C0001   CS60-255   1998F            70
C0001   CS60-315   1997F            95
C0001   CS60-330   1997F            87
C0002   CS60-254   1996F            93
C0002   CS60-254   1996W            60
C0002   CS60-315   1997F            98
C0002   CS60-330   1997F            85
C0002   CS60-367   1998W            90
C0003   CS60-212   1996W            91
C0003   CS60-254   1996W            88
C0003   CS60-255   1997W            89
C0003   CS60-315   1996F            70
C0003   CS60-315   1997F            94
C0003   CS60-330   1997F            90
C0004   CS60-212   1996W            85
C0004   CS60-254   1996W            92
C0004   CS60-255   1997W            86
C0004   CS60-330   1996F            70
C0004   CS60-330   1997F            95
C0005   CS60-212   1996W            70
C0005   CS60-212   1997W            96
C0005   CS60-254   1996W            90
C0005   CS60-255   1996F            87
```

-- Dimension Table: student --

```
STUDENT NAME                     G
------- ------------------------ -
C0001   Linda Sharon             F
C0002   Scott Johnson            M
C0003   Edward Green             M
C0004   John Smith               M
C0005   Mary Clark               F
```

-- Dimension Table: course --

```
COURSE_NO  COURSE_NAME
---------- ------------------------------
CS60-104   Computer Concepts
CS60-212   C++ Programming
CS60-254   Data Structures
CS60-255   File Structures
CS60-315   Database Management Systems
CS60-330   Operating Systems
CS60-367   Computer Networks
```

-- Dimension Table: term --

```
TERM          YEAR SEASON
------- ---------- --------
1991F         1991 Fall
1991W         1991 Winter
1992F         1992 Fall
1992W         1992 Winter
1993F         1993 Fall
1993W         1993 Winter
1994F         1994 Fall
1994W         1994 Winter
1995F         1995 Fall
1995W         1995 Winter
1996F         1996 Fall
1996W         1996 Winter
1997F         1997 Fall
1997W         1997 Winter
1998F         1998 Fall
1998W         1998 Winter
1999F         1999 Fall
1999W         1999 Winter
2000F         2000 Fall
2000W         2000 Winter
```

Vita Auctoris

NAME:	Xiaohong Meng
PLACE OF BIRTH:	Jilin Province, P. R. China
YEAR OF BIRTH:	1962
EDUCATION:	GMI Engineering and Management Institute, Flint, Michigan
	1985 – 1989 B. Sc.
	University of Windsor, Windsor, Ontario
	1993 – 1998 M. Sc.