Development of New Data Partitioning and Allocation Algorithms for Query Optimization of Distributed Data Warehouse Systems

A thesis submitted in partial fulfilment of the requirements of

London Metropolitan University for the degree of

Doctor of Philosophy

Hassan Ismail Abdalla

June 2008

Dedication

This dissertation is dedicated to my family, my beloved brothers and sisters for their patience and support.

Abstract

Distributed databases and in particular distributed data warehousing are becoming an increasingly important technology for information integration and data analysis. Data Warehouse (DW) systems are used by decision makers for performance measurement and decision support. However, although data warehousing and on-line analytical processing (OLAP) are essential elements of decision support, the OLAP query response time is strongly affected by the volume of data need to be accessed from storage disks.

Data partitioning is one of the physical design techniques that may be used to optimize query processing cost in DWs. It is a non redundant optimization technique because it does not replicate data, contrary to redundant techniques like materialized views and indexes. The warehouse partitioning problem is concerned with determining the set of dimension tables to be partitioned and using them to generate the fact table fragments.

In this work an enhanced grouping algorithm that avoids the limitations of some existing vertical partitioning algorithms is proposed. Furthermore, a static partitioning algorithm that allows fragmentation at early stages of schema design is presented.

The thesis also, investigates the performance of the data warehouse after implementing a combination of Genetic Algorithm (GA) and Simulated Annealing (SA) techniques to horizontally partition the data warehouse star schema. It, then presents the experimentation and implementation results of the proposed algorithm.

This research presented different approaches to optimize data fragments allocation cost using a greedy mathematical model and a combination of simulated annealing and genetic algorithm to determine the site by site allocation leading to optimal solutions for fragments distribution.

Throughout this thesis, the term fragmentation and partitioning will be used interchangeably.

Preface

All work presented here is the original work of the author unless otherwise indicated. Some parts of this report include revised versions of the following published papers:

F. Marir, M. Tounsi H. and Abdalla, "Using a Greedy-Based Approach for Solving Data Allocation Problem in a Distributed Environment", To appear in the Proceedings of the 2008 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'08).

F. Marir, H. Abdalla and M. AlFares, "An Enhanced Grouping Algorithm for Vertical Partitioning Problem in DDBs" IEEE Computer and Information Sciences, 2007. ISCIS-2007. 22nd International Symposium, 7-9 Nov. 2007, Pages:1–6, Digital Object Identifier 10.1109/ISCIS.2007.4456833.

H. Abdalla, E. Abuelyaman and F. Marir, "A Static Attribute-Based Partitioning Algorithm for Vertical Fragmentation in DDBs", Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07), Volume II, pp 1017-1022, Las Vegas, June, 2007.

H. Abdalla, M. AlFares and F. Marir, "Vertical Partitioning for Database Design: A Grouping Algorithm", 16th International Conference on Software Engineering and Data Engineering July 9-11, 2007, Nevada USA, Las Vegas, July 2007.

L. Bellatreche, H. Abdalla and K. Boukhalfa, "A Combination of Genetic and Simulated Annealing Algorithms for Physical Data Warehouse Design", Proceedings of the 23rd British National Conference on Databases, Queen's University Belfast, Northern Ireland, 18-20 July, 2006.

H. Abdalla and F. Marir, "Vertical Partitioning Impact on Performance and Manageability of Distributed Database Systems: A Comparative study of some vertical partitioning algorithms", Proceedings of the 18th NCC, Riyadh, Saudi Arabia, pp 85 - 92, March 26 - 29, 2006.

L. Bellatreche, K. Boukhalfa, and H. Abdalla, "Algorithms for Physical Data Warehouse Design to Speed up Decision-making Processes", Proceedings of the 18th NCC, Riyadh, Saudi Arabia, pp 93 -110, March 26 - 29, 2006.

Acknowledgements

In preparing this thesis, I am highly indebted to pass my heartfelt thanks to the many people who helped me in one way or another. First, I would like to acknowledge my sincerest, special, deep appreciation, and true thanks to Dr. F. Marir for his insights and guidance without which this thesis would have not been possible. Through many long discussions, he provided me with many insightful technical suggestions and helped me clarify my often-confused thoughts.

My thanks also go to the members of my Ph.D. supervision team, Dr. M. Affendi, Dr. F. Cai and Dr. S. Sahiti, who provided valuable suggestions and guidance to my research topics, that helped greatly to improve the presentation and contents of this dissertation.

There is always a major turning point in the life of any one of us. For me this point occurred when I first come to know Dr. Ladjel Bellatreche and started communicating with him to discuss various issues in the area of data warehousing. From then and on, Ladjel was continuously there to give me a friendly scientific and professional advice in my research work.

Table of Contents

List of Figures *Page*

List of Tables *Page*

Abbreviations and Meanings in Alphabetic Order

3NF	Third Normal Form
AAM	Attribute Affinity Matrix
AG	Affinity Graph
ALF	Attributes Link Factor
ANSI/SPARC	American National Standards Institute/ Standards Planning And Requirements Committee
AUM	Attribute Usage Matrix
BEA	Bond Energy Algorithm
CAM	Clustered Affinity Matrix
COMM-MIN	Completeness-Minimality
CQP	Constrained Quadratic Program
DBD	Data-Base Designer
DBMS	Database Management Systems
DDBMSs	Distributed Database Management Systems
DDBS	Distributed Database Systems
DSS	Decision Support Systems
DW	Data Warehouse
DWA	Data Warehouse Administrator
EISs	Executive Information Systems
ERM	Entity-Relationship Modelling
ETL	Extracted, Transformed and Loaded
FL	Fragment Limit
GA	Genetic Algorithms
GBAs	Graph-Based Algorithms
GHz	Giga Hertz
GLF	Group Link Factor
HP	Horizontal Partitioning
I/O	Input/Output
MB	Mega Byte
OLAP	On-Line Analytical Processing

OLTP	On-line Transaction Processing
OODB	Object Oriented Database
PS	Page Size
PT	Processing Time
RDB	Relational Database
RNG	Random Number Generator
SA	Simulated Annealing
SAPA	Static Attribute-Based Partitioning Algorithm
SQL	Structured Query Language
SSPD	Set of Simple Predicates
VP	Vertical Partitioning
WAN	Wide Area Network

Chapter 1

Introduction

Data Warehouses (DWs) are large, special purpose databases that contain historical data integrated from a number of independent sources, supporting users who wish to analyze the accumulated data. The analysis is usually done by queries that aggregate, select and group the data in a number of ways. Efficient query processing is critical because the data warehouse is very large, queries are often complex, and decision support applications typically require least response time.

The concept of data warehousing has evolved out of the need for easy access to a structured store of quality data that can be used for decision making [Ahmed *et al* 02]. It is globally accepted that information is a very powerful asset that can provide significant benefits to any organization and a competitive advantage in the business world. Organizations have vast amounts of data but have found it increasingly difficult to access and maintain. This is because data is in many different formats, exists on many different platforms, and resides in many different file and database structures developed by different vendors. Thus organizations have had to write and maintain perhaps hundreds of programs that are used to extract, prepare, and consolidate data for use by many different applications for analysis and reporting. Also, decision makers often want to dig deeper into the data once initial findings are made [Datta *et al* 98]. This would typically require modification of the existing programs or development of new ones. This process is costly, inefficient and very time consuming. Data warehousing offers a better approach.

Data warehouse applications deal with enormous data sets in the range of Gigabytes or Terabytes. Queries usually either select a very small set of this data or perform aggregations on a fairly large data set. Materialized views storing pre-computed aggregates are used to efficiently process queries with aggregations. This approach increases resource requirements in disk space and slows down updates because of the view maintenance problem.

The emergence of data warehousing was initially a consequence of the observation that, operational-level on-line transaction processing (OLTP) and decision support applications

cannot coexist efficiently in the same database environment, mostly due to their very different transaction characteristics [Inmon 96].

Operational databases are focused on recording transactions, thus they are prevalently characterized by an online transaction processing (OLTP) workload. Conversely, data warehouses allow complex analysis of data aimed at decision support; the workload they support has completely different characteristics, and is widely known as online analytical processing (OLAP) [Rizzi 08].

The design of a data warehouse database system of an OLAP nature is fundamentally different from the operational database system of OLTP nature. The following list summarizes the major differences between OLTP and OLAP system design

- OLTP contains up-to-date detailed information while OLAP contains historical, summarized, multidimensional, integrated and consolidated data.
- OLTP is highly normalized with many tables to ensure consistency while OLAP is typically de-normalized with fewer tables (use of star and/or snowflake schemas).
- OLTP focus on single record access while OLAP focus on multiple record data access.
- OLTP Emphasise on update speed while OLAP emphasise on search speed.
- OLTP is application oriented while OLAP is subject-oriented.
- OLTP is used for day to day operations while OLAP is used for decision support.
- OLTP queries are relatively standardized and simple queries returning relatively few records (Quantitative) while OLAP queries are often complex involving aggregations (Qualitative).

The term "Data Warehouse" was first used by Barry Devlin [Devlin 96], but Bill Inmon has won the most acclaim for introducing the concept, defined as follows: "A Data Warehouse is a subject oriented, integrated, non-volatile and time-variant collection of data in support of management's decisions" [Inmon 96]. These properties can be defined as follows [Inmon 96]:

Subject Oriented: Subject orientation means that the data warehouse (unlike the operational systems where data is organized to support specific business processes) does not include data

that won't be used for Decision Support System (DSS) processing. Data in DW is organized by subject rather than by function. The data warehouse is oriented to the major subject areas of the corporation that have been defined in the high-level corporate data model. Typical subject areas include: customer, product, transaction or activity, claims, etc.
Each major subject area is physically implemented as a series of related tables in the data warehouse. A subject area may consist of 10, 100, or even more physical tables that are all related.

Integrated: A business typically employs many different operational systems, each optimized for a special business process, and each with its own data store. Whatever the design issue, the data in a data warehouse needs to be stored in a singular, globally acceptable fashion, regardless of the data source it is coming from. In the DW, data from various sources is integrated, both by definition, i.e., the same data type, and by content, i.e., the same value sets, wherever these occur. When data is moved to the data warehouse from application-oriented operational systems, the data is integrated before entering the data warehouse. So that the focus of the data warehouse users is on using the data in the data warehouse, rather than on wondering about the credibility or consistency of the data.

Non-volatile: In a typical operational system, data is often kept only for a short period of time as it is only interesting for the daily business during that short period. However, in a data warehousing environment the need to discover trends as been done in business and comparing them with those of previous periods strengthen the need to keep data for longer periods of time. Once data is loaded into the data warehouse, then most operations for using the data will be data querying, rather than inserting, deleting, and changing. That is, data in data warehouse is normally kept for long-term and is not overwritten but instead it is appended.

Time-variant: Time variant means that the data in a data warehouse is accurate not only for some period of time, but for the whole data history. A data warehouse contains enterprise information of each stage from the time the data warehouse began to current. This means, not only the current value of data is stored, but also, a snapshot of data at specific points in time, or a complete history of all changes occurred to the data. Thus, the representation of the history of fact values across a given lapse of time, at a given granularity, is directly supported in DWs [Golfarelli and Rizzi 08].

Using this information, DSS analyst quantitatively analyzes and forecast the development progress and future trend of an enterprise. In contrast to data warehouse, application-oriented operational databases mainly consider the data over specified periods.

Management's decisions: Data in a DW is optimized for data analysis and used by top management at the strategic level for setting the course for the entire business. Therefore, managing redundancy of data is usually appropriate in a DW because it simplifies the database schema and improves analysis performance.

1.1 Background

The origin of the concept of data warehousing can be traced back to the early 1980s, when relational database management systems emerged as commercial products [Mohania *et al* 00]. The foundation of the relational model with its simplicity, together with the query capabilities provided by the SQL language, supported the growing interest in what then was called end-user computing or decision support.

To enhance end-user computing environments, data was extracted from the organizations existing databases and stored in newly created database systems dedicated to supporting ad hoc end-user queries and reporting functions of all kinds. One of the prime concerns underlying the creation of these systems was the performance impact of end-user computing on the operational data processing systems. This concern prompted the requirement to separate end-user computing systems from transactional processing systems.

Data warehouse systems represent a single source of information to analyse the development and results of an organization [Reeves *et al* 98]. Measures such as the number of transactions per customer or the increase of sales during a promotion period are used to recognise warning signs and to decide on future investments with regard to the strategic goals of the organization.

Decision Support Systems (DSS) and Executive Information Systems (EISs) can only be effective tools if the data used are readily available and represent the integration of all pertinent corporate wide data. Data warehouses provide this integrated environment by extracting, filtering, and integrating relevant information from all available data sources.

Further, as new or additional relevant information becomes available, or the underlying source data are modified by the operational systems, the new data are extracted from its autonomous, distributed and heterogeneous sources into a common model that is integrated with existing warehouse data. Once information is available at the warehouse, queries can be answered and data analysis (DSS and EIS) can be performed.

Much of the work to date has focused on building large centralized systems that are integrated repositories founded on pre-existing systems upon which all corporate-wide data is based. The centralized data warehouse is very expensive and tends to ignore the advantages realized during the past decade in the areas of distribution and support for data localization in a geographically dispersed corporate structure. Further, it would be unwise to enforce a centralized data warehouse when the operational systems exist over a widely distributed geographical area.

The distributed data warehouse supports the decision makers by providing a single view of data even though that data are physically distributed across multiple data warehouses on multiple systems at different sites. Currently, the field of distributed data warehouse in terms of architecture and design is considered a potential research area with great scope for future investigation.

1.2 Motivation and Problem Statement

Partitioning of a global schema before allocating it in a distributed database or data warehouse can be performed either in vertical fragmentation or horizontal fragmentation schemes. In this work, both vertical and horizontal partitioning methodologies for improving the performance of distributed database systems are adopted.

The work presented in this thesis is motivated by the need to demonstrate the feasibility of using a combination of Simulated Annealing and Genetic Algorithms (SAGA) in solving horizontal partitioning selection problem based on the observations that data warehouse can have a large number of fragmentation schemas.

Given a set of dimension tables $D = \{D_1, D_2, ..., D_d\}$ and a set of OLAP queries $Q = \{Q_1, Q_2, ..., Q_m\}$. The horizontal partitioning selection problem is concerned with determining a set of dimension tables $D' \subseteq D$ to be partitioned and generating a set of fragments that can be used to partition the fact table F into a set of horizontal fragments (called fact fragments) $\{F_1, F_2, ..., F_N\}$ such that:

– The sum of the query cost when executed on the partitioned star schema is minimized and
– $N \leq W$, where N is the number of possible fragments of the fact table, and W is a threshold fixed by the Data Warehouse Administrator (DWA) representing the maximal number of fragments that can be maintained by him/her. The obedience of this constraint avoids an explosion of the number of the fact fragments.

A combination of genetic and simulated annealing algorithms is also used to tackle the fragment allocation problem by determining the optimal allocation of a data fragment in a distributed environment based on the fragment access patterns and the cost of moving data fragments from one site to the other.

1.3 Aims and Objectives

A data warehouse stores large volumes of data from many sources, with the purpose of efficiently implementing decision support or OLAP queries. One of the most important decisions in designing a data warehouse is the selection of optimal fragments to be maintained at the warehouse. The goal is to select an appropriate set of fragments that minimizes total query response time, query processing cost and the cost of maintaining the selected fragments, given a limited amount of resources such as CPU time, storage space, etc.

In this work, a conceptual framework for the general problem of fragments selection in a data warehouse has been developed. The goal is to choose a design, such that the performance of a given query is optimized, subject to a threshold constraint on the maximum number of fragmentations that can be maintained by a data warehouse administrator and that would have to be determined by him. Given a set of queries Q that will be executed on a relational data warehouse modelled by a star schema S with D dimension tables and a fact table F, by partitioning the fact table based on the fragmentation schemas of the fragmented dimension

tables (derived fragmentation), the author aims to reduce irrelevant data accesses to the star schema and allow the OLAP queries to be executed efficiently.
On the other hand, the objective of the proposed fragment allocation method is to determine which fragments are used by each query being hosted at specific sites such that all queries are satisfied while minimizing the communication cost.

1.4 Contribution

This research contributes to the solution of the problem of distributed warehouse database design and architecture by:

1. Providing a simple to understand, easy to implement and efficient algorithms compared to existing algorithms used for vertical partitioning problem. This is obtained by adding some factors that allow for more control on the final produced partitions based on the problem specifications.

2. Proposing a static vertical partitioning algorithm for improving the performance of database systems using the number of occurrences of an attribute in a set of queries rather than the frequency of queries accessing the attributes. This enables the database designer to perform partitioning and consequent distribution of fragments before the database enters operation.

3. Proposing a methodology for the distributed data warehouse design using a horizontal fragmentation algorithm to partition the huge fact relation into a set of fragments based on the fragmentation schema of the dimension tables. Thus, relations those need not to be accessed are identified and unnecessary processing is avoided.

4. Addressing and formalizing horizontal fragmentation schema selection problem in relational data warehouse, using a genetic and simulated annealing algorithms in a star schema to select the right solution that improves the performance of OLAP queries by avoiding unnecessary processing, and reduces the maintenance cost.

5. Contributing in determining best possible allocation of a data fragment in a distributed environment using a greedy mathematical modelling approach and SAGA algorithm that considers network communication and data movement costs to allocate fragments site by site, leading to a better solution for the optimal fragments distribution problem.

1.5 Thesis Organization

In this chapter, the background to the proposed research has been described, the subject matter of the thesis is outlined and the scope of the work is specified. The rest of this thesis is organized as follows. In Chapter 2, basic concepts in the area of distributed databases, data warehouses and some distributed database design issues will be provided. Chapter 3 specifies how the data model for a data warehouse should be planned to structure the data in a manner that could handle the On-Line Analytical Processing (OLAP). Chapter 4 describes the data warehouse functions and the distributed data warehouse design methodology. In Chapter 5, related work and developments in the area of horizontal and vertical partitioning in relational database systems are discussed. An enhanced grouping algorithm that avoids the limitations of some existing vertical partitioning algorithms is provided in Chapter 6. Enhancements are continued in Chapter 7 by providing a static partitioning algorithm that allows fragmentation at early stages of schema design. Chapter 8 investigates the performance of the data warehouse after implementing a combination of Genetic Algorithm (GA) and Simulated Annealing (SA) techniques to horizontally partition the data warehouse star schema. It, then presents the experimentation and implementation results of the SAGA algorithm. Chapter 9 optimizes the data fragments allocation cost using a greedy mathematical model and SAGA algorithm to determine the site by site allocation leading to optimal solutions for fragments distribution. Finally, Chapter 10 provides an overall conclusion, a list of contributions and future work suggestions.

Chapter 2

Review of Basic Concepts

2.1 Introduction

The advent of telecommunication era and the constant development of hardware and network structures have encouraged the decentralization of data while increasing the need to access information from different sites leading to significant advances in distributed database systems [Muthuraj 92]. A distributed database system is a collection of sites connected on a common high-bandwidth network. Logically, data belongs to the same system but physically it is spread over the sites of the network, making the distribution invisible to the user [Ceri and Weiderhold 89]. Each site is an autonomous database with its own processing capability and data storage capacity. The advantage of this distribution resides in achieving availability, modularity, performance, response time improvement and reliability.

Distributed and parallel processing on database management systems is an efficient way of improving performance of applications that manipulate large volumes of data. This may be accomplished by removing irrelevant data accessed during the execution of queries and by reducing the data exchange among sites, which are the two main goals of the design of distributed databases [Özsu and Valduriez 99].

The primary concern of distributed database systems is to design the fragmentation and allocation of the underlying database. Distribution involves making decisions on the fragmentation and placement of data across the sites of a computer network. The first phase of designing the distribution is the fragmentation phase, which is the process of physically distributing data that is logically linked to make it closer to the user for fast access. The fragmentation phase is then followed by the allocation phase, which handles the physical storage of the generated fragments among the nodes of a computer network, and then replication of fragments.

2.2 Distributed Databases

A distributed database is the term used collectively for distributed database systems (DDBSs) and distributed database management systems (DDBMSs). These systems were developed in response to the current trend toward distributed models of computation. Unlike traditional centralized database systems, DDBSs are spread over many sites. These sites are connected by a communications network.

Portions of the entire database are spread out over multiple computers, called sites or nodes. The computers are connected by a communications network with a given topology. Each local site may have its own local database, which can be maintained by a traditional DBMS. Each site may also contain fragments, or portions of the distributed global database. Fragments are managed by application and communication processing software.

The advantages of DDBs include reliability and availability. Reliability is loosely defined as the probability that a system is up at a particular moment in time. Availability refers to the probability that a system is continuously available during some time interval. In a traditional centralized database system, the failure of a single site means failure of the entire system. In a DDB, the failure of a single site will only affect access to data located at that site. Clearly, this leads to improved reliability and availability. Another advantage of DDBs is the performance improvement obtained by distributed processing. Local queries and transactions accessing data at a single site are much faster since the local database is smaller. Transactions involving different sites can be processed concurrently, reducing execution and response time. This is especially an advantage when the database is naturally distributed over different locations, such as in a business with databases used by regional offices which are all accessible from the corporate headquarters. These types of database systems are typically dominated by local queries and transactions. Further, DDBs allow sharing of data while at the same time retaining localized control. This can be an important issue in database security when maintaining a 'need to know' authorization scheme.

A potential drawback in a DDB is the added complexity and overhead involved in its design and implementation. The DDB must be designed to preserve consistency in the database while providing acceptable response time for transactions across many different sites. Strategies must be developed to handle distributed queries and transactions. The distribution

as part of the design, involves fragmentation of relations and allocation of these fragments. The objective of fragmentation is to achieve better units of distribution. Allocation is concerned with optimal placement of the fragments among the available sites. Special care must be taken in the placement of replicated fragments to maintain consistency and efficient access. Another important aspect is that the DDB must be able to gracefully recover from failures such as site crashes or network hang ups.

2.3 Data Warehousing and OLAP

Data warehousing is the design and implementation of processes, tools, and facilities to manage and deliver complete, timely, accurate, and understandable information for decision making. It includes all the activities that make it possible for an organization to create, manage, and maintain a data warehouse or data mart.

The amount of information available in today's large-scale enterprises has been growing explosively. New data are being rapidly and continuously generated by various operational sources, such as auction databases and order processing systems. In order to make intelligent business decisions, complex analytical queries will be issued and answered across all data sources [Chaudhuri and Dayal 97]. Since the modern data sources are becoming increasingly heterogeneous and are often distributed over a large network, it is often preferred that the data may have need to be Extracted, Transformed and Loaded (ETL) [Chaudhuri and Dayal 97] before complex analytical queries can be executed. However, such ETL processes are very expensive.

Data warehouses are thus proposed to support on-line analytical processing (OLAP) [Chaudhuri and Dayal 97]. A data warehouse extracts and integrates data from independent data sources and then stores the integrated data in a central database. The warehouse data is extracted from multiple operational databases, external sources or legacy sources. The extracted data is often further aggregated under different granularities in order to provide a summary of the underlying data.

The granular data found in the data warehouse is the key to reusability, because it can be used by many people in different ways. With a data warehouse, different organizations are able to look at the data as they wish to see it.

Looking at the data in different ways is only one advantage of having a solid foundation. A related benefit of a low level of granularity is flexibility, containing a history of activities and events across the corporation, and that data can be reshaped across the corporation for many different needs.

Perhaps the largest benefit of data warehousing is that future unknown requirements can be accommodated. Various front-end tools have been developed to analyze such summary data for decision making, for example, query reporting, analysis and data mining [Chaudhuri and Dayal 97].

Most analytical queries over the warehouse data are fairly complex involving multiple joins and aggregations [Zaharioudakis *et al* 00]. Such queries are typically operated over huge volumes of data (many terabytes). Furthermore, most of these queries require interactive response, i.e., response time typically in a few seconds [Zaharioudakis *et al* 00]. Traditional query optimization techniques often fail to meet such new requirements. The proposed solution is to horizontally partition the warehouse to vastly improve the query performance.

2.4 Query Optimization

Query optimization is of great importance for the performance of a relational data warehouse [Ziyati *et al* 07], especially for the execution of complex SQL statements. In query optimization process the query optimizer chooses and determines the best strategy for performing each query, for example, whether or not to use indexes for a given query, and which join techniques to use when joining multiple tables. These decisions have a tremendous effect on SQL performance, and query optimization is a key technology for every application, from operational systems to data warehouse and analysis systems to content-management systems.

The query optimizer is entirely transparent to the application and the end-user. However, because applications may generate very complex SQL, query optimizers must be extremely sophisticated and robust to ensure good performance. For example, query optimizers transform SQL statements, so that these complex statements can be transformed into equivalent, but better performing SQL statements.

Query optimization is mostly cost-based (it sometimes compromise between cost and response time). In a cost-based optimization strategy, multiple execution plans are generated for a given query, and then an estimated cost is computed for each plan. The query optimizer chooses the plan with the lowest estimated cost.

2.4.1 Query Optimization and Execution in Data Warehousing

After the set of data sources has been selected for a given query, a key problem is to find the optimal query execution plan for this query. A query execution plan is an imperative program that specifies exactly how to evaluate the query. In particular, the plan specifies the order in which to perform different operations in a query (join, selection, and projection), a specific algorithm to use for each operation and the scheduling of different operators. Typically, the optimizer selects a query execution plan by searching a space of possible plans and comparing their estimated cost [Bennett *et al* 91]. To evaluate the cost of a query execution plan the optimizer relies on extensive statistics about the underlying data, such as sizes of relations, sizes of domains and selectivity of predicates. The query execution plan is passed to the query execution engine, which evaluates the query. This problem is analogous to the query optimization problem faced in database systems, except that it is more complicated in Data Warehousing because [Bennett *et al* 91]:

1) Since the sources are autonomous, the optimizer may have no statistics about the sources or have unreliable ones. Therefore, the optimizer cannot compare between different plans because their costs cannot be estimated.

2) The data sources are not necessarily database systems, the sources may appear to have different processing capabilities. For example, one data source may be a Web interface to a legacy information system, while the other may be a program that scans data stored in a structured file (for example bibliography entries). Therefore, the query optimizer needs to consider the possibility of exploiting query processing capabilities of a data source. Note that query optimizers in distributed data warehouse systems also evaluate which parts of the query should be executed, but in a context when the different processors have identical capabilities.

3) In a data warehouse system, data is often transferred over a Wide Area Network (WAN) and hence delays may occur for many reasons as opposed to traditional system where the optimizer can reliably estimate the time to transfer data from the disk to the main memory. Therefore, even a plan that appears to be the best based on cost estimates may turn out to be inefficient if there are unexpected delays in transferring data from one of the sources accessed early in the plan.

2.5 Data Warehouse Partitioning

The size of a data warehouse varies, but they are typically quite large because of volumes of historical data (measured in terabytes) [Boehnlein and Ende 99]. To deal with this issue it is vital to consider data partitioning in the data architecture.

Partitioning is the process of dividing a relation into fragments which contain sufficient information to reconstruct the relation back to its original status. It is an important aspect of physical database design that has significant impact on performance and manageability [Bellatreche *et al* 06]. In a data warehouse, issues surrounding partitioning do not focus on whether partitioning should be done, but rather how it should be done. Partitioning data allows data to grow and to be easily managed.

Generally speaking, proper partitioning can benefit a data warehouse in several ways:

- Loading data
- Accessing data
- Archiving data
- Deleting data
- Monitoring data
- Storing data

The purpose of partitioning data is to break data up into small fragments because operation staff and the designer can manage small physical units of data better than large ones. Some of

the tasks that can easily be performed when data resides in small fragment units includes restructuring, indexing, sequential scanning, reorganization, recovery, monitoring, etc.

Data can be partitioned by criteria, such as date, line of business, geography, organizational unit, etc. There are two main types of partitioning, vertical and horizontal. However, vertical and horizontal partitions can be mixed and fragments may be successively fragmented to an arbitrary depth.

Like indexes and materialized views, both kinds of partitioning can significantly impact the performance of the queries by reducing cost of accessing and processing data.

2.5.1 Vertical Partitioning

Vertical partitioning (VP) allows a table to be partitioned into disjoint sets of columns. It is the process that divides a relation into fragments called vertical fragments containing subsets of the original attributes. It allows tuples to be split so that each part of the tuple is stored where it is most frequently accessed. It also allows parallel processing on a relation.

2.5.2 Horizontal Partitioning

Horizontal partitioning (HP) is the process that divides a global relation into subsets of tuples called horizontal fragments. It allows parallel execution of On-Line Analytical Processing (OLAP) queries on fragments of a relation and allows a relation to be split so that tuples are located where they are most frequently accessed. HP aims to reduce irrelevant data access during query processing [Bellatreche 08], [Özsu and Valduriez 1999].

Horizontal partitioning is an important aspect of physical database design that has significant impact on performance and manageability. Two versions of HP are cited by researchers [Ceri and Pelagatti]: *primary HP* and *derived HP*. Primary HP of a relation is performed using predicates that are defined on that relation. On the other hand, derived HP is the partitioning of a relation that results from predicates defined on any other relation.

DBAs today also use horizontal partitioning extensively to make database servers easier to manage. If the indexes and the underlying tables are partitioned identically, database operations such as backup and recovery become much easier. Therefore, there is a need to incorporate manageability while arriving at the right physical design for data warehouses.

2.6 Summary

This chapter reviewed some basic concepts related to distributed databases and data warehousing that will be extensively used in this work.

Chapter 3 presents different data modelling techniques employed in data warehousing environment and explores how data modelling is important in enabling relational data warehouses to efficiently handle on-line processing.

Chapter 3

Data Warehouse Data Models and Architecture

3.1 Introduction

Data warehouse modelling is a very important stage in the data warehousing process. It provides necessary information for designing and implementing the data warehouse and defining its functions.

Data warehouse modelling is a process that produces abstract data models for one or more database components of the data warehouse. It is one part of the overall data warehouse development process that comprises other processes such as data warehouse architecture and design with the aim of producing an abstract model representing the reality which the data warehouse is intending to support and interact with.

The data model for a data warehouse should be designed to structure the data in a manner that could efficiently handle the (OLAP) queries. There are two techniques for modelling the data warehouse: the dimensional data model and the relational data model [Ballard *et al* 98]. These two modelling techniques provide a broader view of data to support and facilitate the OLAP applications.

The schema proposed for physical structures of the relational data warehouse model is the star schema [Kimball 96]. The following sections discuss the relational data warehouse modelling and schema definitions in detail.

3.2 Data Warehouse Modelling

In the process of building a data warehouse, modelling is an important step prior to the data warehouse implementation. Although the models in the data sources may vary, the data warehouse itself must use a single consistent model that accommodates the needs of users [Richardo 04]. Two data modelling techniques being used in a data warehousing environment, despite of their variations in terms of semantic representation are entity-

relationship (ER) modelling and dimensional modelling. ER modelling produces a data model using three basic concepts: entities, attributes and relationships among entities. ER modelling is a tool for representing conceptual data and primarily focus on eliminating data redundancy and keeping consistency among the different data sources and applications. It is a transaction based modelling tool that is mainly used to design a data model for relational databases [Ballard *et al* 98]. However, dimensional modelling is a tool that is used for designing data warehouses on top of a pre-existing database and focuses on the subject area data rather than on transactions. Both of the two modelling techniques, ER and dimensional modelling, has their own strengths and weaknesses, and each of them can be used in the appropriate situation. The distributed data warehouse design presented in this research is based on the dimensional data model.

The importance of data modelling in a data warehouse could be highlighted through the following points:

- A data model should represent the real world, because it is an abstraction and reflection of the real world. Actually, data is nothing but a simple representation of all business activities, resources, and facts of the organization. Thus, the data model is a well-organized abstraction of the data it reflects the structure, functionality, and work flow of the data. Through the data model, we have the ability to visualize the relationship between the warehouse data so that these data can be used in higher efficiency.
- A single, well-integrated, and easy to understand data model of the data warehouse is crucial because the data warehouse integrates data being gathered from multiple autonomous and heterogeneous data sources with different data structures.
- Data modelling step could be considered as a guideline to implement the data warehouse. Consolidating the data models of each business area before real implementation can help in reducing the cost of implementation.

3.2.1 Dimensional Data Modelling

Dimensional modelling is a technique for conceptualizing and visualizing data models as a set of measures that are described by common aspects of the business. It is especially used for

summarizing and rearranging the data and presenting views of the data to support data analysis [Ballard *et al* 98].

Particularly, dimensional modelling focuses on numeric data, such as values, counts and cost [Richardo 04]. While conventional database modelling uses transactions/functions such as automating stock control, calculating VAT, etc, the dimensional data modelling uses numeric data to aggregate facts to answer the queries, such as, why a state agent selling is going down (Question) for properties (Subject). So the sales of properties make the fact which is influenced by features (dimensions) of the properties, such as location, time, etc.

Dimensional data modelling has three basic concepts: facts, dimensions, and measures. A fact is a collection of related data items, consisting of measures and context data. Fact units and their values are referred to as measures. In a data warehouse, facts are implemented in the core tables in which all of the numeric data is stored. Dimensions are single perspectives on the data that determines the granularity (data detail level) to be adopted for fact representation

3.2.2 Dimensional Data Modelling for Multidimensional Data Analysis

Multidimensional analysis allows decision makers to efficiently and effectively use data analysis tools, which mainly depend on multidimensional structures of a data warehouse [Mazón *et al* 08]. The data models for designing traditional OLTP systems are not well suited for modelling complex queries in data warehousing environment. This is because, in data warehousing queries tend to use joins on more tables and have a much more computation time than that in OLTP systems. This kind of processing environment requires a new perspective to data modelling. The dimensional data modelling proved to be the suitable model for OLAP applications because it provides a way to aggregate facts along multiple attributes, called dimensions [Agrwal *et al* 97], [Mohania *et al* 00] .

In the dimensional model, data could be thought of as residing in a multidimensional matrix called a data cube [Boehnlein and Ende 99]. OLAP tools provide an environment for decision making and business modelling activities by supporting ad-hoc queries. There are two ways to implement dimensional data model.

1. By using the underlying relational architecture and
2. By using a true multidimensional data structures like arrays.

Multidimensional analysis requires a data model that will enable the data to easily and quickly be viewed from many possible perspectives, or dimensions. It enables users to look at a large number of interdependent factors involved in a business problem and to view the data in complex relationships which can be analyzed through an iterative process that includes drilling down to lower levels of detail or rolling up to higher levels of summarization and aggregation. That is, rather than submitting multiple queries, data is structured to enable fast and easy access to answers for typically asked questions. For example, the data would be structured to include answers to the question, How much of each of our products was sold on a particular day, by a particular sales person, in a particular store? Each separate part of that query represents a dimension.

Figure 3.1 shows a three-dimensional data cube called sales. The basis of the multidimensional data model [Wiese 05] is rooted in the difference between qualifying and quantifying data that are reflected by two key concepts: dimensions and measures. Dimensions in dimensional models serve for the unambiguous, orthogonal structuring of the data space and describe different ways of looking at the information (e.g. time, database objects and workload).

The intersection of dimensions acts as an index and identifies the data points the analysts intend to analyze, the so-called measures.

In contrast to the descriptive, textual and qualifying dimension attributes, the fact attributes are mostly numerical, additive and quantifying information.

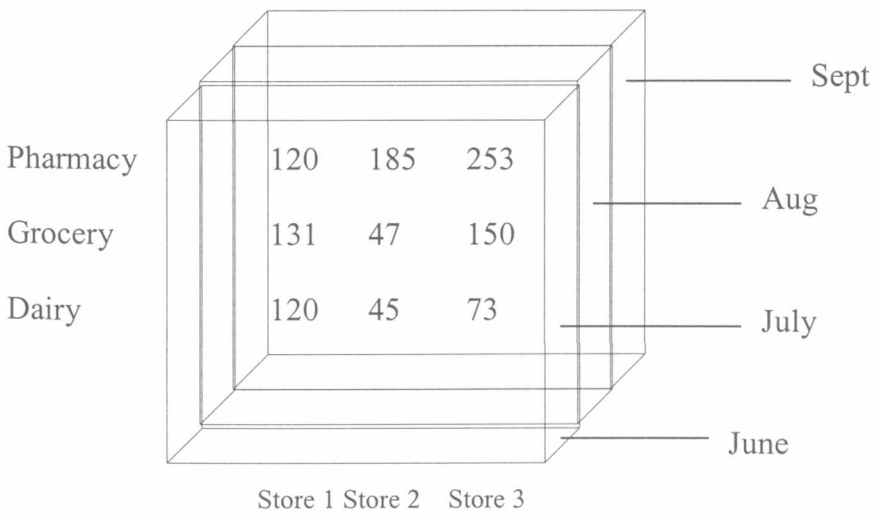

Figure 3.1: Sales Data Cube

3.3 Data Warehouse Schema

The data warehouse schema design was introduced by Kimball [Kimball *et al* 98]. The basic principle behind is building a de-normalized data structure to improve database performance for OLAP applications.

The data warehouse schemas are physical database structures that store quantitative or factual data about the organization in large central tables surrounded by a group of smaller tables that describe the dimension of the organization [Chaudhuri and Dayal 97], [Shoshani 97].

There are two basic models that can be used in dimensional modelling: star schema model and snowflake schema model. The star schema is widely used in data warehousing as the basic structure for dimensional modelling. Typically, it has one large central table of raw data, called the fact table that stores un-aggregated observed data, and has some attributes that represent dimensions, and other dependent attributes that are of interest. Each dimension is represented by its own table, and the dimension tables can be thought of as the points of the star whose centre is the fact table. In contrast, the snowflake model is actually the result of normalizing one or more of the dimensions of the star model causing the dimension tables themselves to have dimensions.

Figures 3.2 and 3.3 below show an example for star model and snowflake model in dimensional modelling quoted from [Richardo 04].

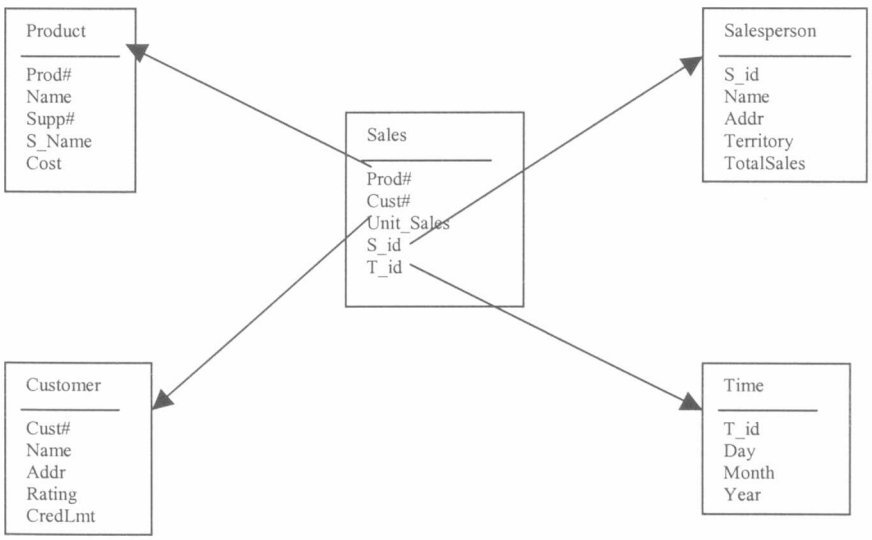

Figure 3.2: A Star Model (Quoted from [Richardo 04])

The primary justification for using the star schema is performance and understandability. The simplicity of the star schema has been one of its attractions [Bekrid *et al* 08]. Data in a multi-dimensional database is stored as business people view it, allowing them to slice and dice the data to answer business questions. When star schema is designed correctly, an OLAP database will provide must faster response times for analytical queries.

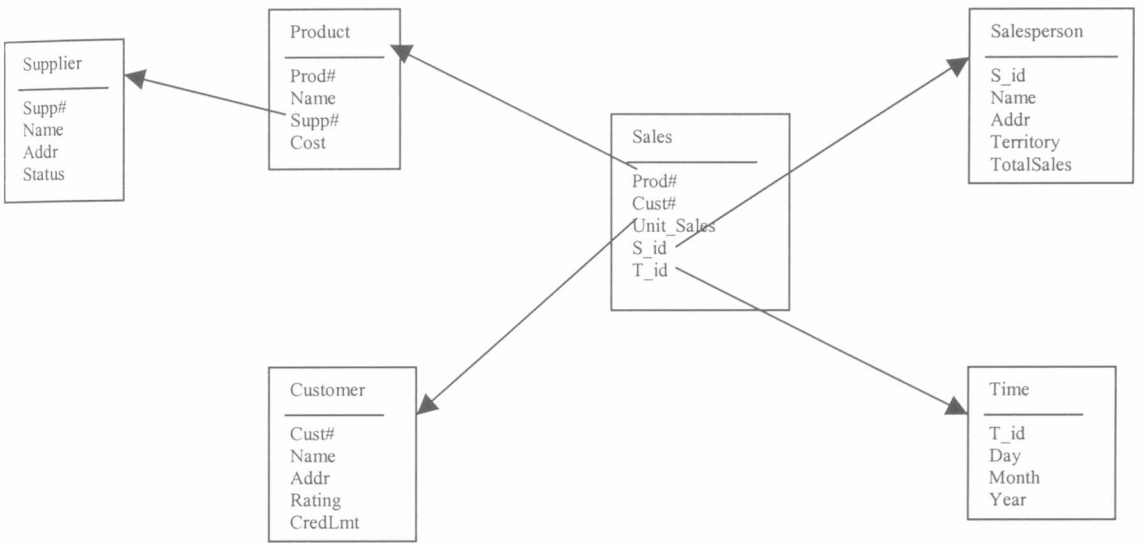

Figure 3.3: Snowflake Model (Quoted from [Richardo 04])

Definition 1 (Data Warehouse Schema):

Data warehouse schema S is an ordered pair (D, F) [Noaman and Barker 99], $S = (D, F)$ where:

$D = \{D_1, D_2,...,D_n)$. is a set of dimension relation schemas

$F = (F_1, F_2, . , F_t)$ is a set of fact relation schemas.

The data warehouse schema is a set of relation schemas. Two types of relation schemas are available in data warehouse: Dimensions and Facts. The relationship between the fact relation and dimension relations is one to many.

Definition 2 (De-normalization):

De-normalization is the process of re-joining relations in a careful way to introduce controlled redundant data into already normalized relations, thereby improving database performance.

The advantages of using de-normalization in building data warehouses are [Poe 96]:

1. Reducing the number of joined relations required to answer queries. As a result, the run-time application is improved.
2. Mapping the physical database structure closely to the user queries thereby improving the database performance.

The dimension relation represents the joining of more than one normalized relation from legacy system. Clearly these are in at least 3rd NF to perform OLAP analysis that should be "pre-joined" to enhance the performance of the OLAP queries.

3.4 Data Warehouse Architecture

Data warehouse architecture exhibits various layers of data in which data of each layer are derived from the lower layer data. The proposed distributed data warehouse architecture represents the classical solution for a large enterprise with various divisions and geographic locations. It is based on the ANSI/SPARC architecture [Tsichritzis 78] that has three levels of schemas: internal, conceptual and external.

Figure 3.4 illustrates typical data warehouse system architecture. It is a three-tiered architecture consisting of (1) data sources layer (2) data warehouse layer (that includes staging area, detail data, summarized data, data marts, and meta data) (3) end-user layer.
The following sections discuss these layers in detail [Noaman and Barker 99].

Data source layer is the origin of the data in the data warehouse. One feature of data warehouses is integrating data from multiple autonomous and heterogeneous data sources. Furthermore, warehouse data could come from either remote or local data sources. Such arbitrary make challenges to data warehouse builders for creating a uniform repository to store these multi-structure data, and designing an easy-understanding modelling language to express the schemas of data sources and data warehouse repositories, and the transformations between these schemas.

Data Warehouse layer is the central layer of the architecture. The global data warehouse keeps a historical record of data that results from the transformation, integration, and

aggregation of detailed data found in the data sources. Usually, a data store of volatile, low granularity data is used for the integration of data from the various sources.
The actual data warehouse is the database which contains the integrated collection of data used to support strategic decision-making processes. It contains several components such as:

- *Staging Area:* Staging area keeps whole copies of the source data and brings them under the control of the data warehouse administrator. Naturally, staging area stores heterogeneous data and may contain duplicate and inconsistent data. Data in staging area is the direct source of data in the data warehouse

- *Detailed Data:* Detailed data includes current detailed data and older detailed data [Inmon 00]. Far and away the current detailed data is the major concern of data warehousing. It is the exact lowest level source of the information supporting DSS processing. Normally, data here is stored in a singular, globally acceptable fashion. From the staging area to the detail data repository, data need to be extracted, cleaned, transformed, loaded and integrated. Such activities are the main processes of data warehousing

- *Summarized Data:* Summarized data is the data that is distilled from the low level detail data. It is divided into two levels, lightly and highly summarized data. Both of these could be treated as virtual or materialized views over the detailed data or other views. Mostly, DSS processing is based on these views.

- *Data Marts:* Data warehouse users have different information requirements from different departments. Data flows from the data warehouse to various departments for their customized DSS usage. These departmental DSS databases are called data marts. A data mart is actually a body of DSS data for a department that has an architectural foundation of a data warehouse [Inmon 00].

- *Meta-Data:* Meta-data plays a special and very important role in the data warehouse and serves as a roadmap that provides a trace of all design choices and a history of changes performed on its architecture and components [Inmon 00]..

End-user applications layer is the interface used by data warehouse user to access warehouse data. It contains a series of tools, such as OLAP servers, query applications, analytic applications, data mining tools, and so on.

The conceptual architecture of a data warehousing system shown in figure 3.4, describes the proposed data warehouse architecture used in this work. It presents the functionality of its components and how the information flows in the distributed data warehouse environment.

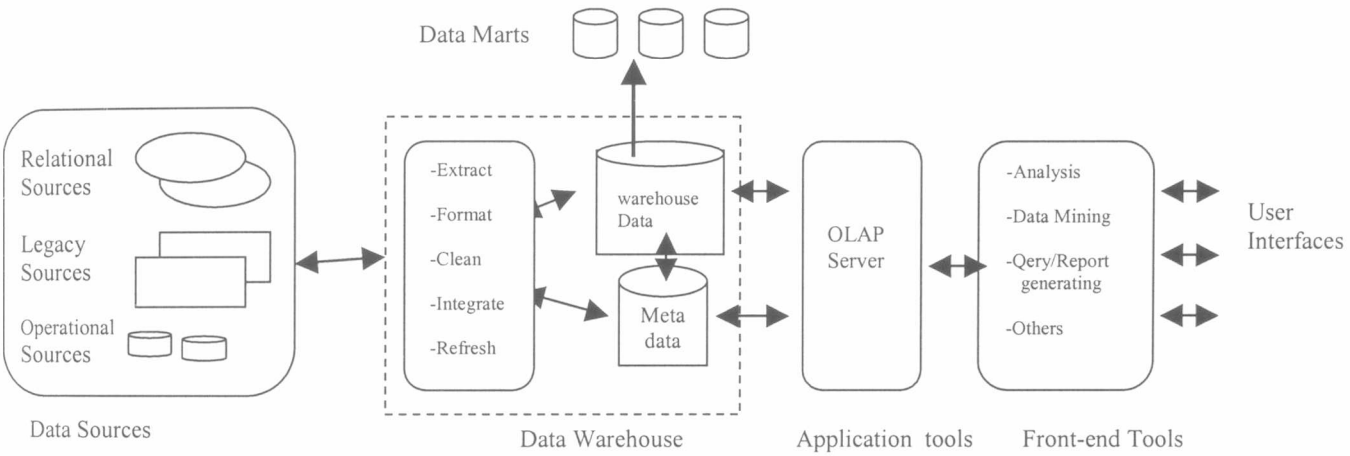

Figure 3.4: Data warehouse architecture

3.5 Summary

In addition to the benefit of visualization, the data model plays the role of a guideline, or plan, to implement the data warehouse. Traditionally, ER modelling has primarily focused on eliminating data redundancy and keeping consistency among the different data sources and applications. Consolidating the data models of each business area before the real implementation can help assure that the result will be an effective data warehouse and can help reduce the cost of implementation.

ER and dimensional modelling, although related, are different from each other. There can be no definite answer on which is best, but there are guidelines on which would be the better selection in a particular set of circumstances or in a particular environment.

In this chapter, the importance of data modelling in a data warehousing environment has been highlighted and some analytical modeling techniques used in data warehousing have been presented.

Because physical schema design is one of the major factors that affect the performance of query processing in data warehousing environment, in the next chapter, the data warehouse physical design will be reviewed and defined as an optimization problem requiring solutions to several interrelated problems.

Chapter 4

Distributed Data Warehouse Design

4.1 Introduction

A data warehouse is a repository of data that has been extracted and integrated from heterogeneous and autonomous distributed sources. DWs stores large volumes of data used for decision support applications and usually owned by centrally coordinated organizations to process a more complex OLAP queries.

Data warehouses are constructed in a heuristic manner, where one phase of development depends entirely on the results attained from the previous phase. First, one portion of data is populated. It is then used and examined by the DSS analyst. Next, the data is modified and/or other data is added. Then another portion of the data warehouse is built, and so forth. This loop continues throughout the entire life of the data warehouse.

Therefore, data warehouses cannot be designed the same way as the classical requirements-driven system. On the other hand, anticipating requirements is still important. Design begins with the considerations of placing data in the data warehouse. There are many considerations to be made concerning the placement of data into the data warehouse from the operational environment.

When the existing operational applications were constructed, no thought was given to possible future integration. Each application had its own set of unique and private requirements. It is no surprise, then, that some of the same data exists in various places with different names or some data is labeled the same way in different places, or all data is in the same place with the same name but reflects a different measurement, and so on.

Pulling the data into the data warehouse without integrating it is a grave mistake. However, extracting data from many places and integrating it into a unified picture is a complex problem.

In data warehousing approach (also known as an *eager* or *in-advance* approach for data integration), information from each source that may be of interest is extracted in advance, translated and filtered as appropriate, merged with relevant information from other sources, and stored in a (logically) centralized repository. When a query is posed, the query is evaluated directly at the repository, without accessing the original information sources. Therefore, in this approach, the integrated information is available for immediate querying and analysis by clients. Thus, the warehousing approach is appropriate for [Widom 95]:

- Clients who require specific, mostly predictable portions of the available information.
- Clients who require high query performance (the data is available locally at the warehouse), but not necessarily requiring the most recent state of the information.
- Environments in which native applications at the information sources require high performance (large multi-source queries are executed at the warehouse instead)
- Clients wanting access to private copies of the information so that it can be modified, annotated, summarized, and so on, or clients wanting to save information that is not maintained at the sources (such as historical information).

The main characteristics of data warehouses are: their data complexity due to the presence of historical cascade of data, the huge amount of data, and the complexity of their queries due to the presence of join and aggregate operations.

4.2 Data Warehouse Functions

The main function of a data warehouse is providing data to support DSS processing. Additional data warehouse functionalities could include the following[1]:

1 To perform as a server whose tasks are associated with querying and reporting on the data not used by transaction processing systems.

2 To use data models and/or server technologies that speed up querying and reporting the data that are not appropriate for transaction processing.

[1] http://www.dwinfocenter.org/casefor.html

3 To provide a means to speed up the writing and maintaining of the querying and reporting data.

4 To provide an area for cleaning transaction processing data and making them appropriate to the requirement of DSS processing.

5 To make it easier, on a regular basis, to query and report data from multiple transaction processing systems, from external data sources, and from data that must be stored for query/report purposes only.

6 To provide a repository of transaction processing system data that contains data from a longer span of time which can efficiently be held in a transaction processing system, and to be able to generate reports "as was" as of a previous point in time.

7 To prevent people who only need to query and report transaction processing system data from having any access to maintain and update databases.

These functionalities contain the main consideration aspects of building a data warehouse, and imply the architecture of data warehouses.

4.3 Distributed Data Warehouse Design Process and Techniques

The design of data warehouses is an optimization problem requiring solutions to several interrelated problems that include: designing the conceptual schema of the integrated database, mapping the conceptual schema to storage areas and determining appropriate access methods i.e. designing the physical database, designing data fragmentation, designing the allocation of fragments, and local optimization. Each problem phase can be solved with several different approaches thereby making the design a difficult task.

The distributed data warehouse design involves making decisions on the fragmentation and placement of data across the sites of a computer network. However, it is not possible to determine the optimal fragmentation and allocation by solving the two problems independently, since they are interrelated.

To fragment a set of relations, it is possible to use two basic techniques: horizontal and vertical fragmentation [Özsu and Valduriez, 99], which may be combined and applied in many different ways to define the final fragmentation schema.

This chapter addresses the fragmentation phase of data warehouses. The author believes that, by outputting good fragmentation schemas with improved performance, data allocation and replication may then be carried out more efficiently, since the fragmentation schema will adequately reflect appropriate units of distribution according to the application access patterns, and thus may significantly reduce the search space of the allocation phase. However, the generation of a good fragmentation schema of a data warehouse is a not an easy task, because it is not yet a well-defined problem and it could take many parameters into account with conflicting goals. However, the designer may concentrate on semantic relationships leaving physical distribution design to the last phase.

The distributed data Warehouse design process consists of three phases: initial design, redesign and materialization of the redesign [Bellatreche *et al* 99]. The initial design consists of the fragmentation and allocation algorithms that minimize the total query processing cost for a given set of transactions. The redesign problem consists of generating new fragmentation and allocation schemes from current fragmentation and allocation schemes. The materialization of redesign is accomplished by a sequence of operations to materialize the new fragmentation and allocation scheme from the current design.

Efficient query processing is critical because the data warehouse is very large, queries are often complex, and decision support applications typically require interactive response times. Because physical schema design is one of the major factors that affects the performance of query processing in data warehousing environment, our motivation is to study this effect.

A highly normalized schema offers superior performance and efficient storage where only a few attributes are accessed by a particular query. The star schema [Noaman and Barkerl. 99] provides similar benefits for data warehouses where most queries aggregate a large amount of data.

Horizontal and vertical partitioning are important aspects of logical and physical database designs that have significant impact on performance and manageability [Sanjay *et al* 04].

Both kinds of partitioning can significantly impact the performance of the queries executed against the data warehouse system, by reducing cost of accessing and processing data. Horizontal fragmentation of a relation is the partitioning of the relation based on the values of its attributes such that each fragment contains only a subset of the tuples in this relation [Ezeife 00]. To horizontally partition a relational data warehouse [Informix 97] several choices of partitioning schemas are available for a star or snowflake schemas, examples of some available partitioning scenarios are:

1. Partition only the dimension tables using simple predicates defined on these tables (a simple predicate p is defined by: $p : Ai\ \theta$ value, where Ai is an attribute, $\theta \in \{=, <, >, \leq, \geq\}$, and Value $\in$ Dom(Ai)).

   This scenario is not suitable for OLAP queries, because the sizes of dimension tables are generally small compared to that of fact table. Most of OLAP queries access the fact table, which is very huge. Therefore, any partitioning that does not take into account the fact table is discarded.

2. Partition only the fact table using simple predicates defined on this table because it normally contains millions of rows and is highly normalized. The fact relation stores time-series factual data. It is composed of foreign keys and raw data. Each foreign key references a primary key on one of the dimension relations. In a data warehouse modelled by a star schema, most of OLAP queries access dimension tables first and after that the fact table is accessed. This choice is also discarded.

3. Partition some/all dimension tables using their predicates, and then partition the fact table based on the fragmentation schemas of dimension tables.

   This study opted for the last solution for data warehouse partitioning because it takes into consideration the star join queries requirements and the relationship between the fact table and dimension tables (these queries impose restrictions on the dimension values that are used for selecting specific facts, these facts are further grouped and aggregated according to the user demands. The major bottleneck in evaluating such

queries has been the join of a large fact table with the surrounding dimension tables [St¨ohr *et al* 00]).

4.4 Distributed Data Warehouse Physical Design Problem

Physical design is the process to come up with proper structuring of data in storage so that good performance is guaranteed. A highly normalized schema offers superior performance and efficient storage where only a few records are accessed by a particular transaction. It is not possible to come up with better physical schema unless prior knowledge of workload is known. The information required should consist of the nature of queries and their expected frequencies. For each query the following should be specified:

- The relations that will be accessed by the query.
- The attributes on which any selection predicates are specified.
- The attributes whose values will be retrieved by the query.

Distributed data warehouse design is concerned with optimizing the storage and allocation of data with respect to most frequently executed queries on the warehouse.

4.4.1 Objectives of Distributed Data Warehouse Design

When designing a distributed data warehouse, the following objectives should be considered:

- Maximizing processing locality, this simply refers to the principle of placing data as close as possible to the applications which use them.
- Maximizing the degree of parallelism of execution of applications by efficiently distributing workload over the sites. However, it is necessary to consider the trade-off between workload distribution and processing locality as the first could have an adverse effect on the latter.
- Maximizing availability and reliability of distributed data achieved by storing multiple copies of the same information
- Minimizing storage cost by considering the limitations of available storages at different sites.

4.4.2 Approaches of Distributed Data Warehouse Design

Two alternative strategies that have been identified for data warehouse design are the top-down and the bottom-up approaches. The top-down approach [Firestone 97] starts by designing the global schema, designing the database fragmentation, and then allocating fragments to the sites. In this approach the data is obtained from the primary sources every time a query is executed. The bottom-up approach [Bellatreche *et al* 00], on the other hand, is based on the integration of existing database schemas into a single, global schema by aggregating existing databases. Integration refers to the process of merging common data definitions and the resolution of conflicts among different representations given to the same data. In this approach, the data is obtained from the primary sources based on the profile of the likely to be executed queries, which are typically known in advance. These two approaches are similar to *lazy* and *eager* approaches discussed in [Widom 95].

In the lazy approach, where information is extracted from the sources only when queries are posed, the data integration problem is based on the following very general two-step process:

1. Accept a query, determine the appropriate set of information sources to answer the query, and generate the appropriate sub-queries or commands for each information source.

2. Obtain results from the information sources, perform appropriate translation, filtering, and merging of the information, and return the final answer to the user or application.

On the other hand, the steps for the eager data integration approach which is commonly referred to as *data warehousing*, since the repository serves as a warehouse storing the data of interest, are as follows:

1. Information from each source that may be of interest is extracted in advance, translated and filtered as appropriate, merged with relevant information from other sources, and stored in a (logically) centralized repository.

2. When a query is posed, the query is evaluated directly at the repository, without accessing the original information sources.

A lazy approach to integration is appropriate for information that changes rapidly, for users with unpredictable needs, and for queries that operate over vast amounts of data from very large numbers of information sources. The warehousing approach, on the other hand, is appropriate for users requiring specific, mostly predictable portions of the available information requiring high query performance (the data is available locally at the warehouse), but not necessarily requiring the most recent state of the information [Boehnlein and Ende 99].

The top-down and the bottom-up approaches are both feasible design solutions to the data integration problem, and each is appropriate for certain scenarios. In both approaches, fragmentation [Ladjel 08] can play an important role, by fragmenting the data warehouse into a number of fragments that can be used as a data allocation unit to sites. These fragments can be allocated to sites so that most of the queries posed on a given site are executed locally, thus, minimizing communication cost. This problem can be seen as data allocation problem in distributed environment and will be discussed later in this thesis. The database research community has focused primarily on top-down (*lazy*) approaches to integration. The distributed data warehouse design proposed in this research is based on the bottom-up (*eager*) design approach. This approach is based on the integration of the existing schemata into a single global schema. The bottom-up approach is used in data warehousing because user queries can be answered immediately and data analysis can be done efficiently since data will always be available in the warehouse. Hence, this approach is feasible and improves the performance of the system. There are two fundamental issues in the bottom-up design approach: fragmentation and allocation. The problem of fragmentation and allocation has been addressed for distributed relational database system [Ceri and Pelagatti 84] and the distributed object database system [Ezeife and Barker 95]. Previous research work on fragmentation and allocation for distributed relational database system has been based on highly normalized relations.

In the data warehouse environment, the integrated data from different resources are modelled into de-normalized relations to facilitate the on-line data analysis. The existing distributed relational database design techniques for fragmentation will not work quite well for distributed data warehouse because of the underlying model and also because of the difference in the type of queries in both environments (quantitative in DDBs while they are qualitative in Data warehouse environment).

The proposed design goal is to choose a configuration (a configuration is a valid set of physical design structures that can be realized in a data warehouse), such that the performance of a given workload Q is optimized, subject to a threshold constraint W representing the maximum number of fragments manageable by the data warehouse administrator. Figure 4.1 shows the physical design problem formulation.

Given a data warehouse D, a workload Q and a threshold constraint W, find a fragmentation design configuration:

$N = \prod_{i=1}^{g} m_i$ whose total fragments does not exceed the threshold W,

such that the total cost $TC(Q) = \sum_{k=1}^{m} Cost(Q_k)$ is minimized and the response time for a given query workload is improved.

Figure 4.1: Physical Design Problem.

4.5 The Design Methodology

Several methodologies have been proposed for the initial design phase of the distributed data warehouse design process. Some aspects of this initial design, such as vertical and horizontal fragmentation, have been researched before. The purpose of fragmentation design is to determine non-overlapping fragments which are proper starting point for the subsequent data allocation problem. However, before going into the methodological aspects of distributed data warehouse design let us present the different options of vertical and horizontal partitioning scenarios that are possible.

Figure 4.2 shows a relation and different ways of partitioning a relation. Given a relation, the subsets of its tuples form its horizontal fragments defined by its horizontal partitioning, the subsets of its attribute columns form its vertical fragments defined by its vertical partitioning.

Vertical partitioning of each of the horizontal fragments gives rise to row biased fragments and is known as HV partitioning. Horizontal partitioning of each of the vertical fragments gives rise to column biased fragments and is known as VH partitioning.

Simultaneously applying vertical partitioning and horizontal partitioning on a relation gives rise to what is known as mixed partitioning.

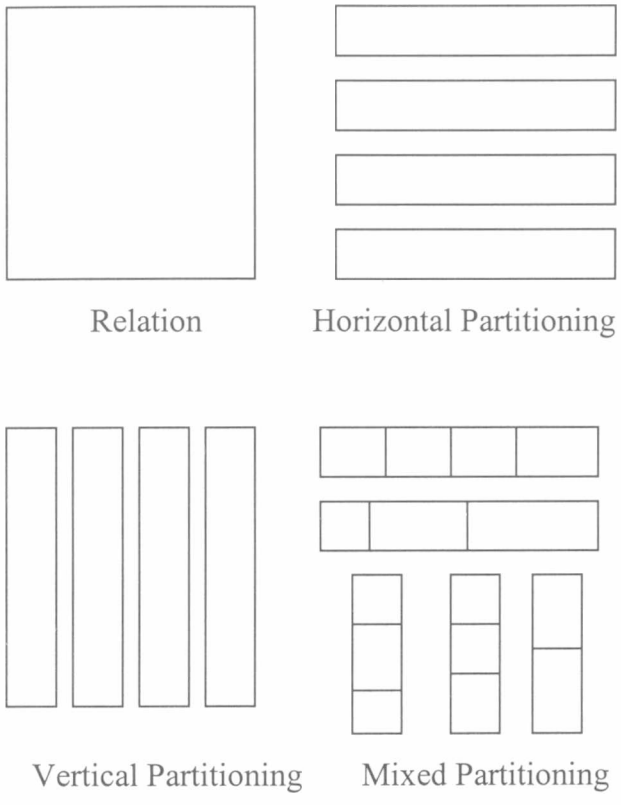

Figure 4.2: Different relation partitioning types

The main ideas of the design methodology are (1) to replicate the dimension relations across the network and (2) to generate horizontal fragments of the fact relation. This is because the size of the dimension relations are relatively small compared to the fact relation and since the dimension relations are changing slowly [Bellatreche *et al* 06], the cost of updating the replicas is relatively low.

There are two approaches for horizontal fragmentation: primary and derived [Ceri and Pelagatti 84], [Özsu and Valduriez 99]. In the first approach, applying the primary horizontal fragmentation requires a set of simple predicates used in OLAP queries against the fact relation that represents the numerical measurements of the organization. The queries perform arithmetic operations on the fact relation such as summarization, aggregation, average and so forth.

However, it is unlikely that a set of simple predicates for the fact relation could be determined from these OLAP queries. Therefore, the primary horizontal fragmentation approach could not be applied on the fact relation.

The other approach is to apply the derived horizontal fragmentation on the fact relation. This approach derives the horizontal fragments of the fact relation from the predicates that are defined on all the dimension relations. In this work, derived horizontal fragmentation approach has been used to obtain our horizontal fragments of the fact relation.

4.6 Data Warehouse Partitioning

To discuss horizontal partitioning of a data warehouse that has been modelled using star schema, let's consider the following definitions.

Definition 1 (Distributed Join)

Distributed join [Ceri *et al* 82] is a join between horizontally partitioned relations. When application requires the join between two relations *R* and *S*, all the tuples of *R* and *S* need to be compared, thus comparing all fragments of relation *R* with all fragments of relation *S*.

A distributed join between two relations *R* and *S* can be represented by a graph called *join graph*. The nodes of this graph represent the fragments of *R* and *S*. An edge between nodes exists if these nodes are joinable.

Definition 2 (Total Join Graph)

A join graph is called total [Ceri and Pelagatti 84] when it contains all possible edges between fragments of *R and S*. A join graph is partitioned if it is composed of two or more sub-graphs without edges between them. A join graph is simple if it is partitioned and each sub-graph has just one edge.

Determining that a join has a simple join graph is very important in database design [Ceri and Pelagatti 84]. The simple join graph concept has a great advantage in optimizing selection and joins operation by providing partition elimination [Oracle Corp. 99]. Partition elimination occurs when the database optimizer determines that some of the table fragments are unnecessary to satisfy the query execution.

4.6.1 Partitioning a Star Schema

The proposed algorithm considers fragmenting a star schema with one fact table F and D dimension tables. The algorithm partitions the dimension table and then uses its fragment schema to partition the fact table. To partition a dimension table, an algorithm that uses quantitative and qualitative [Ozsu and P. Valduriez 99] information about applications is used. Quantitative information describes the selectivity factors and the frequency of each query accessing this table. Qualitative information describes selection predicates defined on these dimension tables.

A simple predicate p is defined as:- p: $Ai\ \theta$ value, where Ai is an attribute, $\theta \in \{=, <, \leq, >, \geq \neq\}$, and value $\in$ *Dom (Ai).*

4.6.2 The Horizontal Fragmentation Algorithm for the Fact Relation

The input to the proposed algorithm are set of dimension tables $D = \{ D_1, ..., D_d\}$ and one fact table F and a set of most frequently executed OLAP queries: $Q = \{Q_1, ..., Q_n\}$ with their frequencies. The algorithm consists of some major steps to generate the horizontal fragments of the fact relations. These steps are:

1. Name all simple predicates used by OLAP queries $Q = \{Q_1, ..., Q_n\}$.

2. Assign to each dimension table D_i $(1 \leq i \leq d)$ its set of simple predicates SSP^{Di}.

3. Each dimension table D_i having $SSP^{Di} = \emptyset$ can not be fragmented. Let $D_{candidate}$ be the set of all dimension tables that do not have empty $SSP^{Di.}$ Let g be the cardinality of $D_{candidate.}$

4. Apply the COMM_MIN algorithm [Ozsu and Valduriez 99] to the simple predicate of each dimension table $D_{i\ of}$ $D_{candidate.}$ This algorithm takes a set of simple predicates and then generates a set of complete and minimal predicates. The rule of completeness and minimality states that "a relation which is partitioned into at least two fragments should be accessed differently by at least one application".

5. Apply the primary horizontal fragmentation algorithm, presented in [Ceri *et al* 82], [Ozsu and Valduriez 99] on each dimension relation. The algorithm generates the set of minterm predicates for each dimension relation along with the set of implications defined on each dimension.

6. After fragmentation process, each dimension table D_i of $D_{candidate}$ will have m_i fragments $\{D_{i1}, D_{i2}, ..., D_{imi}\}$, where each fragment D_{ij} is defined as follows:
   $D_{ij} = \sigma_{cl^i_j}(D_i)$ with cl^i_j $(1 \leq i \leq g, 1 \leq j \leq mi)$ represents a clause of simple predicates.

7. Derive the fragments of the fact table using the fragmentation schemas of the dimension tables.

   The number of the fact table fragments is equal to: $N = \prod_{i=1}^{g} m_i$.

Therefore, the star schema S is decomposed into N sub-star schemas $\{S_1, S_2, ...S_N\}$, where each one satisfies a clause of predicates.

Generally, when the derived HP is used for partitioning a table in a database schema, two potential cases of joins exist (simple and partitioned). In the DW context, when the fact table is horizontally partitioned based on the dimension tables, we will never have a partitioned join (i.e. the case where a HF of the fact table has to be joined with more than one HF of the same dimension table will not occur). A simple join operation (see figure 4.3) has three advantages [Bellatreche et al. 99]:

- It avoids costly total distributed join of fact table F with each and every HF D_{ij} $(1 \leq i \leq mi)$ of each and every dimension table D_i.

- It guarantees the elimination of some partitions from join and selection. For example, if the fact table SALES has been partitioned into 12 fragments (using the attribute Month of the dimension table TIME), the system can satisfy a query that asks for only last two months of data by processing only 2 of 12 fragments.

- It facilitates parallel processing of multiple simple joins (each HF *Fi* of the fact table joins with exactly one HF D_{ik} of D_i).

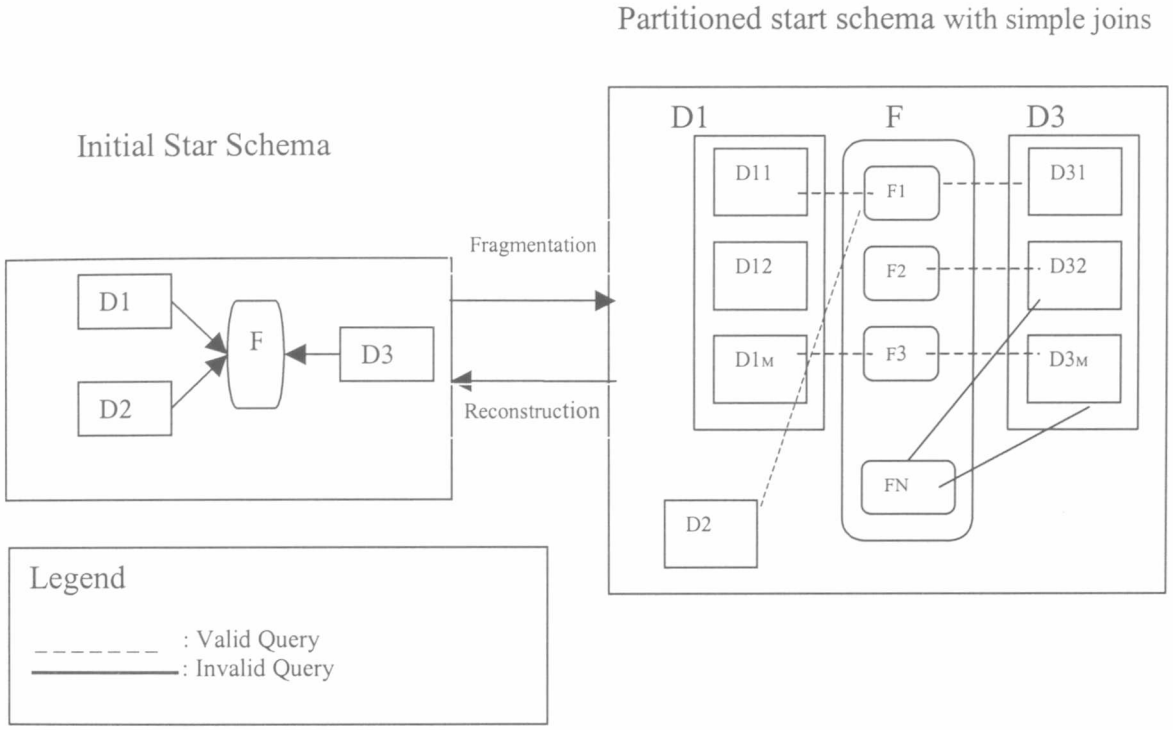

Figure 4.3: Simple join in Star Schema

4.7 Summary

The design of data warehouses is an optimization problem requiring solutions to several interrelated problems of data fragmentation and allocation that include: designing the conceptual schema of the integrated database, mapping the conceptual schema to storage areas and determining appropriate access methods.

In this chapter we have discussed different approaches for distributed data Warehouse design and studied the problem of partitioning the data warehouse when data is modelled using star schema. The chapter presented construction techniques for the design of data warehouse physical structures that can optimize performance of a given workload searching for an optimal fragmentation schema using quantitative and qualitative information of applications.

This work presents a framework to handle the fragmentation problem during the design of distributed data warehouses. The framework works in the conceptual level, and thus uses the

relational data model to capture the application semantics represented by the user and addressing the needs mentioned by [Ozsu and Valduriez 99].

The next chapter will discuss different horizontal and vertical partitioning algorithms that have been proposed in the literature to handle horizontal and vertical fragmentation problem during the design of distributed databases.

Chapter 5

Related Research and Developments

5.1 Introduction

Distributed and parallel processing is an efficient way of improving performance of database management systems (DBMSs) and applications that manipulate large volumes of data. Such improvement comes from limiting queries to data that are relevant to their respective transactions. This is one of the main design goals of distributed databases according to [Ozsu and Valduriez 99].

The primary concern of DBMS design is the fragmentation and allocation of the underlying database. The distribution of data across various sites of computer networks involves making proper fragmentation and placement decisions. The first phase of distribution in a top-down approach is fragmentation which clusters information into fragments for simultaneous access by applications. This process is followed by the allocation phase which distributes, and if necessary, replicates the generated fragments among the nodes of a computer network. The use of data fragmentation to improve performance is not new and commonly appears in file design and optimization literature.

Partitioning based on attributes has been studied earlier in [Babad 77], [Baião 01], [Hoffer 76], [Navathe *et al* 84]. Stocker and Dearnley discussed implementation of a self-reorganizing DBMS that carries out attribute clustering [Stocker and Dearnley 73]. They showed that it is beneficial to cluster attributes of a database where storage cost is low compared to the cost of accessing subfiles. Such is the case because increases in storage costs will be offset by savings in access cost. Hoffer developed a non-linear, zero-one program which minimizes a linear combination of the costs of: storing, retrieving and updating, with capacity constraints for each file [Hoffer 76]. Navathe *et al* used a two-step approach for vertical partitioning. In the first step, they used the given input parameters in the form of an Attribute Usage Matrix (AUM) to construct an Attribute Affinity Matrix (AAM) for clustering [Navathe *et al* 84]. After clustering, an empirical objective function is used to

perform binary partitioning iteratively. In the second step, estimated storage cost factors are considered for further refinement of the partitioning process. Further details about AUM and AAM matrices will be provided in the next paragraph.

Cornell and Yu extended Navathe *et al* approach to decrease the number of disk accesses for optimal binary partitioning [Cornell and Yu 87]. Their extension involved specific physical factors such as: the number of attributes, their length and selectivity, the cardinality of the relation and so on. Navathe and Ra developed a new algorithm that follows graph theory partitioning techniques [Navathe and Ra 89]. Their algorithm starts from the AAM matrix, which is transformed into a graph called the Affinity Graph (AG). An edge in AG is labeled with a weight that represents the affinity between its vertices, where: vertices represent attributes, affinity between vertices represents the number of queries in which the attributes occurred simultaneously. For the interest of clarity of presentation we will define what an AAM matrix is. For more details, interested readers are referred to reference [Navathe *et al* 84]. Basically, an $n \times n$ AAM matrix is one whose AAM(i, j) entry represents the number of queries that simultaneously access the attributes represented by i and j. Based on the AAM, an iterative binary partitioning method has been proposed in [Cornell and Yu 87] and [Navathe *et al* 84]. The authors first clustered the attributes and then applied empirical objective functions and/or mathematical cost functions to perform fragmentation.

5.1 Horizontal Partitioning Algorithms

Although, a lot of work has been done on the partitioning in relational [Ceri and Pelagatti 84], [Ceri *et al* 82], [Ozsu and Valduriez 99] and object models [Bellatreche *et al* 00] but not that much work has been done on horizontal partitioning in relational data warehouses. Ceri, Nergi and Pelagatti [Ceri *et al* 82] show that the main optimization parameter needed for horizontal fragmentation is the number of *accesses* performed by the application programs to different portions of data (file of records). They define applications in terms of boolean *predicates* and use access pattern information to achieve the design. Predicates are collected into sets of *minterms* which form the horizontal fragments. Navathe, Karlapalem and Ra [Navathe *et al* 90] define a scheme for simultaneously applying the horizontal and vertical fragmentation on a relation to produce a grid. A technique similar to the vertical fragmentation schemes discussed in [Navathe and Ra 89]. In 1991, Shin and Irani partition

relations horizontally based on estimated user reference clusters (URCs). URCs are estimated from user queries but are refined using semantic knowledge of the relations. In 1999, Noaman *et al* proposed a construction technique of a distributed data warehouse by adapting the work done by Özsu and Valduriez, 99. Noaman *et al* suggested a horizontal fragmentation algorithm of the fact table based on the fragmentation schemas of the dimension tables. However, they neither took into account the number of fact table fragments, nor showed how horizontal partitioning can be used to speed up query processing and how it can help in fragment allocation. Sufficient effort has been extended to speed up the OLAP query processing in DWs. Materialized views [Gupta and Mumick 95], advance indexes [O'Neil and Quass 97], sampling and parallel computing technologies [Datta *et al* 98] are among the techniques used to enhance DW performance. Ozsu and Valduriez [Ozsu and Valduriez 99] define the database information needed for horizontal fragmentation of the universal relation and show how the database relations are re-constructible using joins. In 1983, Ceri *et al* model this relationship explicitly using directed links drawn between relations via equijoin operations. The relation at the tail of the link is called the *owner* of the link and the relation at the head of the link is the *member*. Primary horizontal fragmentation is performed on all owner relations, while derived horizontal fragmentation is performed on all member relations of links. In 2001, Kalnis and Papadias proposed an architecture for caching dynamically generated results of OLAP queries in a network of cooperating cache servers. The authors considered any horizontal fragmentation schema of the data cube as a candidate for caching. In 2004, Sanjay *et al* suggested using horizontal and vertical partitioning as a part of the physical database design. In 2004, Papadomanolakis and Ailamaki proposed an algorithm called AutPart that automatically partition database tables by using prior knowledge of a representative workload.

In 1994, Karlapalem *et al* identify some of the fragmentation issues in object base including: How are subclasses of a fragment of a class handled? Which objects and attributes of the objects are being accessed by the methods? What type of methods are considered: simple methods that access a set of attributes values of an object or complex methods that access a set of objects and instance variables?[2] Further, they argue that a precise definition of the processing semantics of the application is necessary. They did not present solutions for horizontally fragmenting class objects but argue that techniques used by [Navathe *et al* 90]

[2] They take complex methods as being synonymous with an application

for fragmenting relations could be applied. All these works neither did consider the data partitioning selection problem nor did propose algorithms to solve it.

In this work we will consider that the data warehouse is partitioned using the star schema [Kimball *et al* 98] which has two kinds of tables: dimension tables $D = \{D_1, D_2, ..., D_d\}$ where each table D_i has a primary key K_{Di}, and a fact table F where it is primary key is composed of the concatenation of the dimension tables keys.

To the best of the author's knowledge, the proposed work is one of the first articles that address the problem of selecting a partitioning schema in a relational data warehouse. It proposes an algorithm that minimizes the query processing cost and time under the maintenance threshold constraint representing the number of fragments that the warehouse administrator can maintain.

5.2 Vertical Partitioning Algorithms

Vertical partitioning is the process that divides a relation into sub-relations called vertical fragments, containing subsets of the original attributes [Chen and Su 96], [Niamir 78], [Kittler 76].

Vertical partitioning is used during design of a database to enhance the performance of query execution [Navathe *et al* 84], [Ma *et al* 06]. In order to obtain improved performance, fragments must closely match the requirements of the query workload. The advantage of vertical partitioning is that if query involves only few columns then unnecessary fetching of other columns will be avoided. This saves the I/O bandwidth and avoids unnecessary processing.

The input to most of the exisiting vertical partitioning algorithms is the Attribute Usage Matrix (AUM). AUM is a matrix, which has attributes as columns, and queries as rows and the accesses frequency of the queries as values in the matrix. The attribute usage matrix represents the use of attributes in set of queries. Each row refers to one query, the 1 entry in a column indicates that the query accesses the corresponding attributes. The attribute usage matrix quoted from [Navathe *et al* 84] for 10 attributes and 8 queries is shown in Figure 5.1.

Queries \ Attributes	1	2	3	4	5	6	7	8	9	10	Access Freq
Query 1	1	0	0	0	1	0	1	0	0	0	Acc1=25
Query 2	0	1	1	0	0	0	0	1	1	0	Acc2=50
Query 3	0	0	0	1	0	1	0	0	0	1	Acc3=25
Query 4	0	1	0	0	0	0	1	1	0	0	Acc4=35
Query 5	1	1	1	0	1	0	1	1	1	0	Acc5=25
Query 6	1	0	0	0	1	0	0	0	0	0	Acc6=25
Query 7	0	0	1	0	0	0	0	0	1	0	Acc7=25
Query 8	0	0	1	1	0	1	0	0	1	1	Acc8=15

Figure 5.1: Attribute Usage Matrix (Taken from [Navathe *et al* 84])

Most of earlier data fragmentations algorithms use an Attribute Affinity Matrix (AAM) derived from the AUM. An attribute affinity matrix (see Figure 5.2 quoted from [Navathe *et al* 84]) is a matrix in which for each pair of attributes, the sum total of frequencies of queries accessing that pair of attributes together is stored. Attribute affinity between attributes *i* and *j* is defined as:

$$Affij = \sum_{k \in T} acck_{ij}$$

Where $acck_{ij}$ is the number of accesses of query *k* referencing both attributes *i* and *j*. The summation occurs over all queries that belong to the set of important queries *T*. This definition of attribute affinity measures the strength of the bond between the two attributes, predicated on the fact that attributes are used together by queries. Based on this definition of attribute affinity, the attribute affinity matrix is defined as follows: It is an *n x n* matrix for the *n-attribute* problem whose (*i, j*) element equals *Affij*.

A diagonal element AA(*i, i*) equals the sum of the elements in the attribute usage matrix for the column which represents a_i . This is reasonable since it shows the strength of that attribute in terms of its use by all queries.

Most of the proposed vertical partitioning algorithms do not have a mechanism to evaluate the "goodness" of partitions that they produce. The results of the different algorithms are sometimes different even for the same attribute affinity matrix indicating that the objective functions used by these algorithms are different.

Attributes	1	2	3	4	5	6	7	8	9	10
1	75	25	25	0	75	0	50	25	25	0
2	25	110	75	0	25	0	60	110	75	0
3	25	75	115	15	25	15	25	75	115	15
4	0	0	15	40	0	40	0	0	15	40
5	75	25	25	0	75	0	50	25	25	0
6	0	0	15	40	0	40	0	0	15	40
7	50	60	25	0	50	0	85	60	25	0
8	25	110	75	0	25	0	60	110	75	0
9	25	75	115	15	25	15	25	75	115	15
10	0	0	15	40	0	40	0	0	15	40

Figure 5.2: Attribute Affinity Matrix (Taken from [Navathe *et al* 84])

Some approaches that handled vertical fragmentation problem during the design of distributed databases are: Bond Energy Algorithm (BEA) [Hoffer and Severance 75] which used to group the attributes of a relation based on the attribute affinity values in AAM. This algorithm takes as input the attribute affinity matrix, permutes its rows and columns, and generates a Clustered Affinity Matrix (CAM). Generation of the clustered affinity matrix is done in three steps: initialization, iteration and row ordering. BEA is considered appropriate because the AAM is symmetric and the final groupings are insensitive to the order in which items are presented to the algorithm. Binary vertical partitioning algorithm [Navathe *et al* 84] extended the results of [Hoffer and Severance 75] by giving algorithms to quantitatively cluster the attributes together and taking into account blocks of attributes with similar properties. The binary vertical partitioning algorithm uses the clustered affinity matrix to partition an object into two non-overlapping fragments. The approach of this algorithm is splitting rather than grouping with the objective of finding sets of attributes that are accessed mostly by distinct set of applications.

Navathe and Ra have developed a new Graph-based vertical partitioning algorithm based on a graphical technique [Navathe and Ra 89]. This algorithm starts from the attribute affinity matrix by considering it as a complete graph called the "affinity graph" in which an edge value represents the affinity between the two attributes, and then forms a linearly connected spanning tree. The algorithm generates all meaningful fragments in a single iteration by considering a cycle as a fragment. In this algorithm, a linearly connected tree that has only two ends is constructed by including one edge at a time such that only edges at the "first" and the "last" node of the tree would be considered for inclusion. Then "affinity cycles" are formed in this spanning tree by including the edges of high affinity value around the nodes and "growing" these cycles as

large as possible. After the cycles are formed, partitions are easily generated by cutting the cycles apart along "cut-edges". The major disadvantage of this algorithm is the relative complexity involved in implementation. Figure 5.3 shows the affinity graph corresponding to the affinity matrix of figure 5.2.

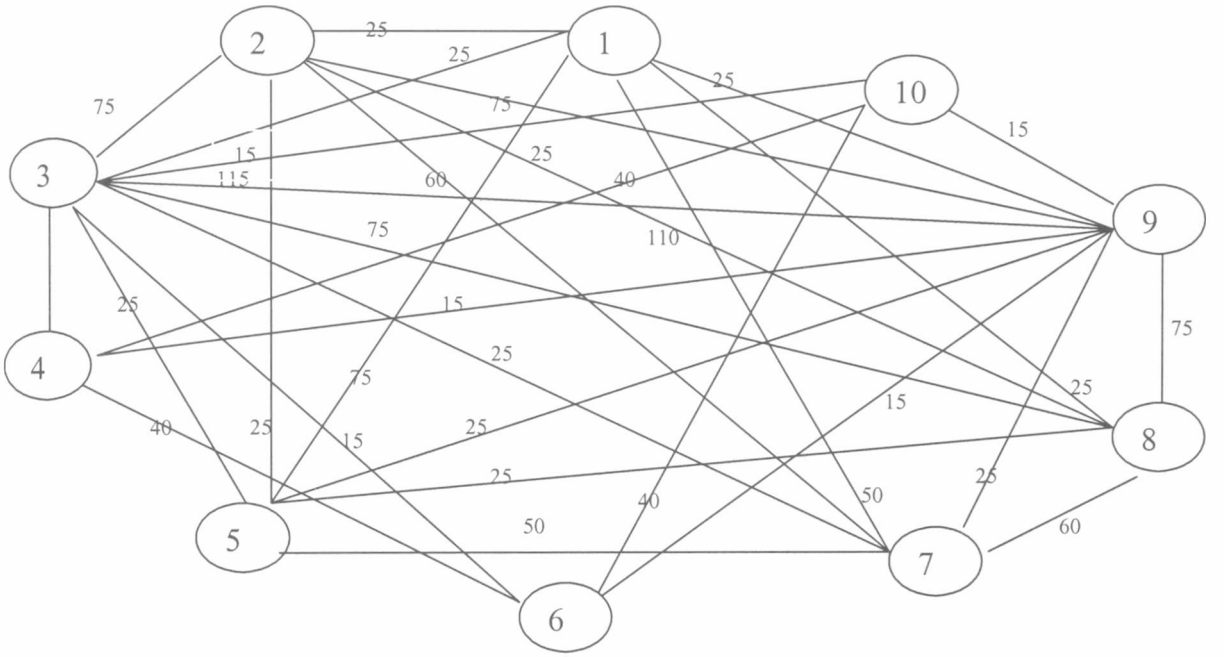

Figure 5.3: Affinity Graph (Taken from [Navathe *et al* 84])

All these Algorithms use the attribute affinity matrix formed from the attribute usage matrix. Attribute affinity, measures the bond between two attributes of a relation according to how they are accessed by applications. Apart from the workload characteristics of queries we must also take into account their expected frequency of invocation. This frequency information along with the attribute information of each query can be used to compute a cumulative statistics of expected access frequency for all the queries. This is expressed as the expected frequency of accessing each attribute in each relation in a selection predicate over all the queries.

5.3 Limitations of the presented Vertical Portioning Algorithms

I believe that the partitioning suggested in the aforementioned vertical partitioning algorithms suffers from the following disadvantages:

1) The database designer has to wait until enough data is collected on the frequencies of queries. Hence, the partitioning is not applicable to a newly designed database and its distribution among various sites of an organization requires another approach.

2) The frequency of queries may be a function of many variables including time, users, and future plans which may call for additional sets of attributes.

3) The bond used to group attributes in a partition is the access frequency, which is of dynamic nature, hence, partitions may not be valid all the time.

Based on the third point above, the author decided to call the frequency based partitioning a dynamic partitioning versus the static partitioning proposed in chapter seven of this thesis.

5.4 Summary

In this chapter, we have presented different approaches that handle horizontal and vertical fragmentation problem during the design of distributed databases by assessing different horizontal and vertical partitioning algorithms. We have studied and compared some vertical partitioning algorithms like bond energy algorithm, binary vertical partitioning algorithm and graph based vertical partitioning algorithm and identified the problems and limitations associated with these algorithms.

In the next chapter an enhancement to the graph algorithm presented in this chapter will be proposed by adopting a grouping algorithm that uses two factors to improve the resulting partitions. The author will verify that different fragmentation algorithms can come-up with the different fragmentation results with varying performance efficiency measures.

Chapter 6

Proposed New Grouping Approach to Enhance Graph Based Vertical Partitioning Algorithms

6.1 Introduction

Distribution design involves making decisions on the fragmentation and allocation of data across the sites of a computer network. Vertical partitioning is the process of subdividing the attributes of a relation to generate fragments. In this chapter, a new vertical partitioning algorithm is proposed. The algorithm uses grouping approach to enhance the previously discussed graph based vertical partitioning algorithms.

In the proposed grouping vertical partitioning algorithm the author explores how vertical partitioning is an important aspect of physical design in relational warehouse database system that has a significant impact on performance. This study addresses the vertical partitioning problem in distributed warehouse database systems during the design phase of distribution. It starts from where previously proposed vertical partitioning algorithms have been stopped by providing solutions and enhancements to the Graph-Based Partitioning Algorithm of Navathe & Ra [Navathe & Ra 89] with regard to database performance. The proposed algorithm starts from the attribute affinity matrix [Ozsu and Valduriez 99] and generates initial groups based on the affinity values between attributes. Then, it attempts to merge the initial groups to produce final groups that will represent the fragments.

6.2 The Enhanced Grouping Algorithm

Before start describing the algorithm, some necessary terms that will be used in the rest of this chapter need to be defined.

6.2.1 Definitions

The following notations and terminologies in the description of the algorithm will be used.

- $a_1, a_2, \ldots, a_n$: denotes attributes.
- *Independent Attribute*: refers to the attribute that has not been joined to any group.
- *Max(Aff)*: refers to the maximum affinity value between two attributes i and j.

- $P(g_k)$: denotes power of a group k.
- $P(A)$: denotes power of the attribute which is the affinity of the attribute to itself $aff(i,i)$.
- *MinMerge*: refers to the difference between the group power value before and after merging.
- *Attributes Link Factor (ALF)*: A factor to avoid having poor grouping between two (or more) attributes.
- *Groups Link Factor (GLF)*: A factor to avoid having poor grouping between two groups.
- *"Best extension"*: refers to the extension that has maximum affinity value and minimum *MinMerge* value and connects attribute A_i in group k to A_j (whether independent or not). If independent check whether $aff(i,j) \geq P(g_k)*GLF/100$ is true. But if $A_j \in P(g_1)$ then check if condition $P(g_l) \geq P(g_k) * GLF/100$ is true.

6.2.2 Description of the Proposed Algorithm

The algorithm starts from the Attribute Affinity Matrix (AAM) generated from the Attribute Usage Matrix (AUM) by considering it as a complete group. Figures 6.1 and 6.2 show AUM and AAM of [Navathe *et al* 84] for a relation containing 10 attributes and 8 queries.

Attribute affinity between attributes i and j with respect to the set of queries $Q = \{q_1, q_2, \ldots, q_q\}$ is defined as: $Aff_{ij} = \sum_{k \in Q} acc_{kij}$

Where acc_{kij} is the number of accesses of query k referencing both attributes i and j.

Queries \ Attributes	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}	Access Freq
Query 1	1	0	0	0	1	0	1	0	0	0	Acc1=25
Query 2	0	1	1	0	0	0	0	1	1	0	Acc2=50
Query 3	0	0	0	1	0	1	0	0	0	1	Acc3=25
Query 4	0	1	0	0	0	0	1	1	0	0	Acc4=35
Query 5	1	1	1	0	1	0	1	1	1	0	Acc5=25
Query 6	1	0	0	0	1	0	0	0	0	0	Acc6=25
Query 7	0	0	1	0	0	0	0	0	1	0	Acc7=25
Query 8	0	0	1	1	0	1	0	0	1	1	Acc8=15

Figure-6.1. Attribute Usage Matrix (Taken from [Navathe *et al* 84])

Attributes	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a_1	75	25	25	0	75	0	50	25	25	0
a_2	25	110	75	0	25	0	60	110	75	0
a_3	25	75	115	15	25	15	25	75	115	15
a_4	0	0	15	40	0	40	0	0	15	40
a_5	75	25	25	0	75	0	50	25	25	0
a_6	0	0	15	40	0	40	0	0	15	40
a_7	50	60	25	0	50	0	85	60	25	0
a_8	25	110	75	0	25	0	60	110	75	0
a_9	25	75	115	15	25	15	25	75	115	15
a_{10}	0	0	15	40	0	40	0	0	15	40

Figure-6.2. Attribute Affinity Matrix (Taken from [Navathe *et al* 84])

6.2.3 Grouping Algorithm Steps

- The algorithm consists of two steps:

Step 1. Iterate starting from the first attribute (first row in affinity matrix) trying to generate a group by joining it to other attribute(s) with highest affinity value (Max($aff(i,j)$)) forming the first initial group. The resulted group will have a power factor P(g) that takes the affinity value $aff(i,j)$. Here three scenarios are possible:

First: the two attributes are independent (do not belong to any initial group), in this case a direct grouping is performed if the selected highest affinity value $aff(i,j)$ is greater than or equal to $P(A_i) * ALF/100$.

Second: one of the attributes i or j belongs to a group k, in this case the independent attribute will be joined to group k if the condition $aff(i,j) \geq P(g_k)$ is true.

Third: having attribute A_i in group k and attribute A_j in group l, then the two groups will be joined if $P(g_k)$ and $P(g_l)$ are equal.

By the end of this step all possible initial groups will be obtained.

Step2. Iterate starting from the first initial group produced in step 1, trying to search for "best extension". At this step there are two possible scenarios:

First: the "best extension" connects attribute A_i in group k and attribute A_j that has not been joined to any initial group in step 1, in this case the independent attribute A_j will be joined to group k if the condition $aff(i,j) \geq P(g_k) * GLF/100$ is true, then the extended group's power will be equal to $aff(i,j)$ value.

Second: the "best extension" connects attribute A_i in group k and attribute A_j in group l, in this case the two conditions of $aff(i, j) \geq P(g_k) * GLF/100$ and $P(g_l) \geq P(g_k) * GLF/100$, need to be satisfied. The new group's power will be equal to the power of group l.

This last step will be repeated until there is no possible "best extension" found, and then we will obtain the final groupings of our algorithm as follows.

ALGORITHM: GROUPING

Input: AAM, ALF, GLF
Output: groups of attributes (vertical partitions)
begin
// Step 1
for each A_i in Relation R **do**
- find the maximum affinity Max(*Aff*) value between A_i and A_j where:
  - $i \neq j$
  - $aff(i,j) \geq P(A_i) * ALF/100$
  - $aff(i,j) \geq P(g_k)$ where $A_i \in g_k$ *or* $A_j \in g_k$
- generate initial group by joining A_i to A_j
- let P(g) ← Max(*Aff*)

end-for
// Step 2
while there is "best extension" **do**
for each A_i in Relation R **do**
let "best extension" ← Max(Aff_{ij}) where
- $i \neq j$
- A_i in group k
- $aff(i, j) \geq P(g_k) * GLF/100$ is true
- **if** A_j in group l then
  - check $P(g_l) \geq P(g_k) * GLF/100$ is true
  - $MinMerge \leftarrow P(g_k) - P(g_l)$

  else
  - $MinMerge \leftarrow P(g_k) - aff(i, j)$

  end-if
- *MinMerge* is the minimum value.

end-for
if "best extension" found **then**
if A_j is independent **then**
join A_j to A_i group
let P(g) ← $aff(i, j)$
else
join all attributes in Aj group to Ai group
let P(g) ← Power of A_j group
end-if
end-if
end-while
end-algorithm

6.2.4 Example 1:

To illustrate how this algorithm works, the Attribute Affinity Matrix of Figure 6.2 will be used for the 10 attributes example employed by [Navathe *et al* 84]. In this example, the following values were used to our introduced enhancement factors: *ALF* = 55% and *GLF* = 60%. I used these values (neither very high nor very low) to ensure having attributes that are mostly accessed together be placed in the same fragment and at the same time avoid having poor grouping. To simplify the example explanation the used conditions are referred to by the following short notations:

ALF-Cond ◄_	$aff(i,j) \geq P(A_i) * ALF/100$
GLF-Cond1 ◄_	$aff(i,j) \geq P(g_k) * GLF/100$
GLF-Cond2 ◄_	$P(g_l) \geq P(g_k) * GLF/100$

The implementation of the algorithm on the 10 attributes example is illustrated in the following points:

1. Start from the first row of *AAM* ($i = a_1$) and search for the Max(Aff_{ij}). The Max(Aff_{ij}) found was 75, where $j = a_5$. Here the condition *ALF-Cond* ($75 \geq 75 * 0.55$) is validated and it was checked to be true. Then the first initial group that joins attributes a_1 and a_5 will be created. The power of this group will be 75. As shown in Figure 6.3-(a).

2. Move to the second row of *AAM* ($i = a_2$) and search for the Max(Aff_{ij}). The Max(Aff_{ij}) found was 110, where $j = a_8$. The condition *ALF-Cond* ($110 \geq 110 * 0.55$) is checked. Then the second initial group that includes attributes a_2 and a_8 is created. The power of this group will be 110, as shown in Figure 6.3-(b).

3. For the third row the Max(Aff_{ij}) was 115, where $j = a_9$ and *ALF-Cond* was true. And the third initial group includes attributes a_3 and a_9, as shown in Figure 6.3-(c).

4. For the fourth row the Max(Aff_{ij}) was 40, where $j = a_6$ and *ALF-Cond* was true. And the fourth initial group includes attributes a_4 and a_6. But, here there is another attribute (10) that has the same affinity, so it will be added to this group as shown in Figure 6.3-(d).

5. For the fifth row the Max(Aff_{ij}) was 75, where $j = a_1$ which was already included in the first initial group. So, this affinity value will be skipped and we move to next Max(Aff_{ij}) which equals to 50, where $j = a_7$. However, because the current Max(Aff_{ij}) is less than the power of the first initial group (75) that includes attribute a_5, then the attribute a_7 cannot be added to this group. Since all remaining affinity values are less than the power of the first initial group (75) we discontinue searching in this row and move to the next row.

6. Similar to step 5, no attribute can be added to the group that includes attribute a_6.

7. For the seventh row the Max(Aff_{ij}) was 60, where $j = a_2$ and which has already been included in the second initial group. However, the power of the second initial group (110) is greater than the current Max(Aff_{ij}). So, we will skip attribute a_2 and move to the next attribute (attribute a_8) that has the same affinity value (60). Because attribute a_8 belongs to the same initial group of attribute a_2 it will be skipped as well. Then we look for the second Max(Aff_{ij}) which is 50, where $j = a_1$. Similarly, attribute a_1 belongs to the first initial group, and the power of the first initial group (75) is greater than the current Max(Aff_{ij}). So, we will skip attribute a_1 and move to the next attribute (attribute a_5) which will also be skipped for the same reason. The next Max(Aff_{ij}) equals to 25 for $j = a_3$ and $j = a_9$ and will both be skipped because the power of their group is greater than 25. Thus, attribute a_7 will remain as independent attribute.

8. Attributes a_8, a_9 and a_{10} will be processed in the same way as above.

9. By completing this step four initial groups will be generated as shown in figure-6.3.

10. Moving to step 2, the algorithm will search for the "best extension".

11. It will start from the first row of *AAM* ($i = a_1$) and search for the Max(Aff_{ij}) for attribute j where $i \neq j$, and attributes i and j are not in the same group.
The Max(Aff_{ij}) will be equal to 50 for $j = a_7$, in this case we must validate if ***GLF-Cond1*** is true i.e. if: $(aff(i,j) \geq P(g_k) * GLF/100 \rightarrow 50 \geq 75 \times 60/100)$ is true.

Since the ***GLF-Cond1*** was found to be true, then the Max(Aff_{ij}) will represent the current "best extension" and the *MinMerge* value will be equal to $P(g_k) - Aff_{ij}$ i.e. 75–50 = 25.

12. By moving to the second row ($i = a_2$), and searching for the Max(Aff_{ij}) we found that the Max(Aff_{ij}) was equal to 75 for $j = a_3$. Here, because the algorithm is trying to join two groups it will check if ***GLF-Cond2*** is true i.e. if:

$$(P(g_l) \geq P(g_k) * GLF/100 \rightarrow 115 \geq 10\times60/100) \text{ is true.}$$

And because ***GLF-Cond2*** is true and the current $MinMerge = |P(g_k) - P(g_l)| = 5$ is less than the previous *MinMerge*(25), then the Max(Aff_{ij}) will represent the current "best extension" and the previous one will be discarded. The new *MinMerge value* will be equal to 5.

Now, and along the same row, the next Max(Aff_{ij}) is equal to 75 for $j = a_9$. And ***GLF-Cond2*** is true but the *MinMerge=5* which is not less that the previous *MinMerge*, so attribute 9 will be skipped. By moving to the next Max(Aff_{ij}) = 60 for $j = a_7$. Here, because an independent attribute need to be joined to a group, then the ***GLF-Cond1*** need to be validated i.e. to check if:

($aff(i,j) \geq P(g_k) * GLF/100 \rightarrow 60 \geq 10\times60/100$) is true. However, the condition ***GLF-Cond1*** found to be false and thus attribute a_7 was skipped. Moving along the same row all other attributes were skipped for the same reason as above.

13. By repeating the previous step for the remaining attributes it was found that the "best extension" will not be changed.

14. After getting the "best extension" that connects attributes a_2 and a_3, then their groups will be merged. And the power of the new group will become 110, as shown in figure-6.4.

15. Going back to search for the next "best extension" in the same way as above, it was found that the "best extension" will be the previously discarded one that connects attributes a_1 and a_7 with Max(Aff_{ij}) of 50, then the group of attribute a_1 will be extended to include attribute a_7. And the power of the new group will become 50, as shown in figure-6.5.

16. Continuing the search for the next "best extension" none was found, therefore, the algorithm stops.

17. At the end, the results produced by our algorithm that represents final groupings are presented in figure-6.5.

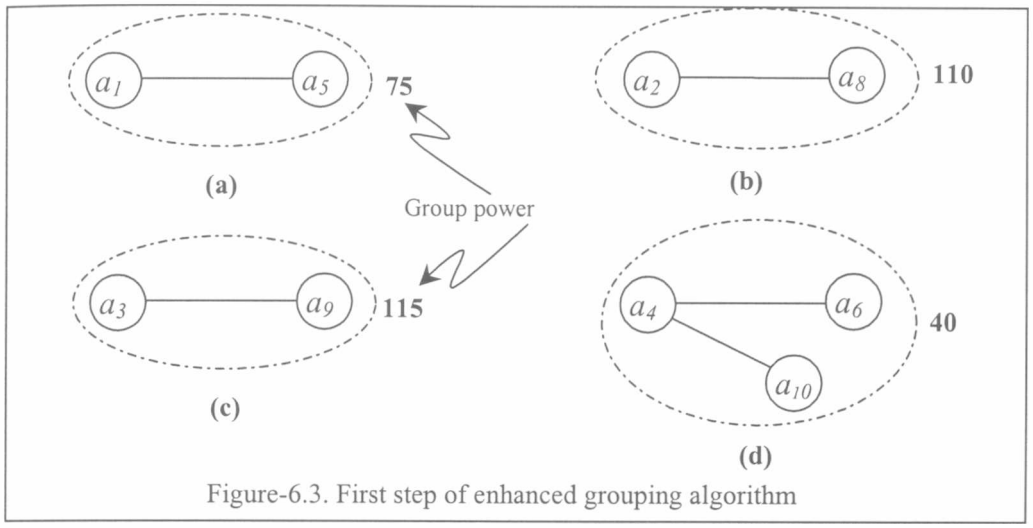

Figure-6.3. First step of enhanced grouping algorithm

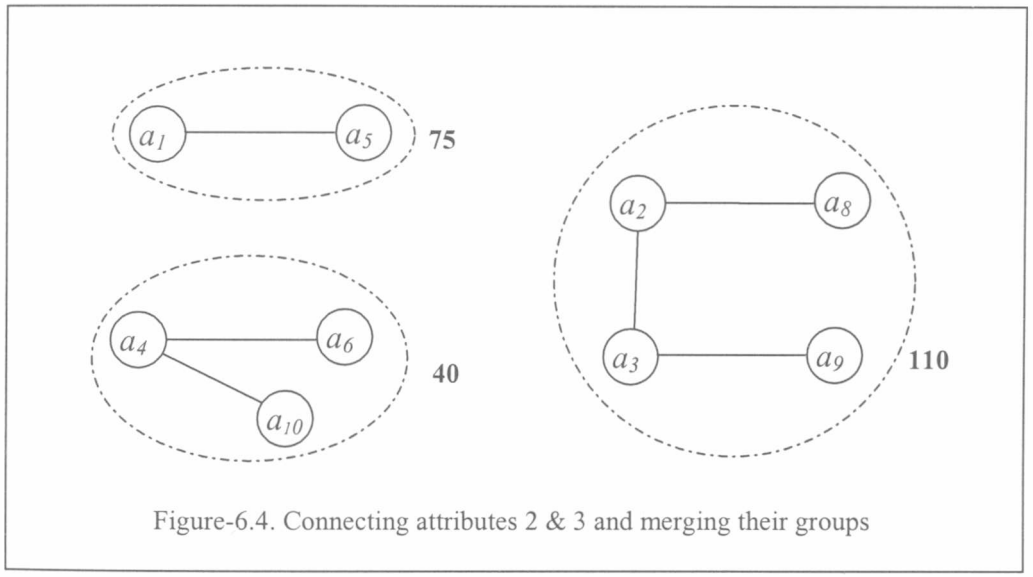

Figure-6.4. Connecting attributes 2 & 3 and merging their groups

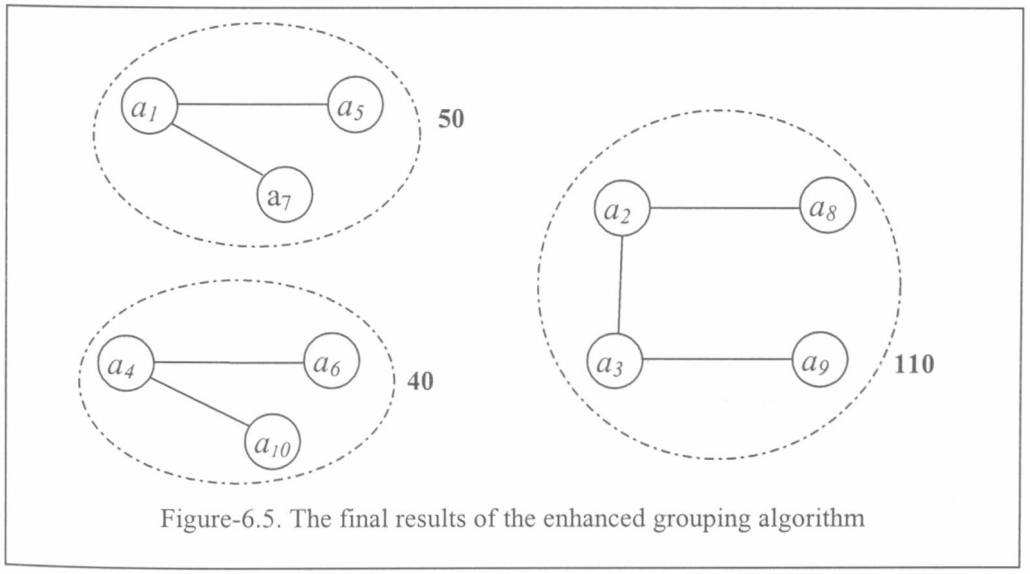

Figure-6.5. The final results of the enhanced grouping algorithm

6.2.5 Example 2:

In the second example the same global relation of 20 attributes and 15 queries of [Navathe *et al* 84] were used. A system developed in C# was used to obtain the final results.

When using the same values for the factors as in example 1 ($ALF = 55\%$ and $GLF = 60\%$), three groups were obtained as shown in table 6.1. However, when changing the second factor value ($GLF = 70\%$), four groups were obtained as shown in table 6.2

$ALF = 55\%$		$GLF = 60\%$
No.	***P* (*g*)**	**Attributes in the group**
1	55	a_1, a_4, a_5, a_6 and a_8
2	65	a_2, a_9, a_{12}, a_{13} and a_{14}
3	65	a_3, a_7, a_{10}, a_{11}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19} and a_{20}

Table-6.1. The final three groups

$ALF = 55\%$		$GLF = 70\%$
No.	***P* (*g*)**	**Attributes in the group**
1	55	a_1, a_4, a_5, a_6 and a_8
2	65	a_2, a_9, a_{12}, a_{13} and a_{14}
3	65	a_3, a_7, a_{10}, a_{11}, a_{17} and a_{18}
4	65	a_{15}, a_{16}, a_{19} and a_{20}

Table-6.2. The final four groups

6.3 Summary

This algorithm is more flexible compared to previous access frequency based grouping algorithms and more efficient for vertical partitioning problem because the added factors provided more control on the final produced groups based on the problem specifications. The major advantage of this algorithm is that it is simple to understand and easy to implement (only two steps).

The final results using the 10 and 20 attributes examples were identical to that obtained by [Navathe *et al* 84] and [Navathe & Ra 89] Graphical algorithms but with easier applicability and more flexibility. This algorithm is more efficient for vertical partitioning problem because it eliminates the limitations of binary partitioning and the complexity of graphical algorithm. The method requires no complementary algorithms such as the SHIFT algorithm of [Navathe

et al 84] that shifts the rows and columns of the affinity matrix, and requires no objective function of [Navathe and Ra 89] to control the process of partitioning.

The values of the enhancement factors for the grouping algorithm are chosen based on several criteria, such as the network bandwidth, number of sites, number of attributes in a relation, the queries frequency and their type (retrieval or update), etc.

In the next chapter a scheme for vertical partitioning of a distributed database at their design cycles is proposed, where the database designer will be in a position to perform partitioning and consequent distribution of fragments before the database is operational. This new proposed scheme is independent of frequencies of queries thus, can be used as a stepping stone for the grouping algorithm presented in this chapter and for its counterpart, the dynamic partitioning technique.

Chapter 7

A Static Partitioning Algorithm for Vertical Fragmentation Problem in a Distributed Environment

7.1 Introduction

Usually, the decision to partition a database is taken when there are problems with an existing centralized data warehouse. Examples of such problems are: delays in query response time, failure of database server, a number of users frequently accessing a particular data, etc. However, there are many situations that occurred recently where there is a critical need to design new database systems in a distributed fashion from the beginning based on the nature of some organizations.

One of the drawbacks of database partitioning techniques that are discussed in previous chapters is the fact that they depend not only on the entries of a database table, but also on their empirical frequencies of use. Such restriction limits the options of a database designer whose task is to distribute a newly designed database across various locations of an organization. The partitioning suggested in the aforementioned algorithms suffers from various limitations that will complicate the task of a database designer. These limitations could be summarized in the following: 1) A database designer has to have sufficient empirical data on frequency of queries 2) Frequency of queries is function of several relatively independent variables that include time, users, and future needs of an organization 3) Attributes are partitioned based on frequency of queries.

The first limitation makes the partitioning inapplicable to newly designed database schemas that have to be distributed among various sites of organization. The second applies to periodical queries. Furthermore, change in organizational structures or business requirements may call for additional attributes. The third limitation comes from the natural dynamicity of frequencies of queries.

In this chapter, an algorithm for partitioning a database at its early stage is proposed. That is, the partitioning is performed at the schema phase of a warehouse database design. The algorithm uses a set of queries that are expected to be employed for accessing a database.

A properly forecasted set of queries will enable a designer to establish partitions that are consistent with those obtained using the frequency based techniques. A simulator was written using C++ to test the proposed algorithm. Results of various simulation runs are consistent with the hypothesis ensuring that partitions can be obtained in advance.

The common denominator in all three limitations is the frequency of queries. Therefore, all partitions that are based on frequency of queries can be classified as dynamic. On the other hand, the only way for a partition to be independent of frequencies of queries is when it is based on a database schema. In this case it will be logical to classify the partitioning as static. However, after the schema design phase is completed, and the database is operational, then an algorithm that uses query frequency approach such as the grouping algorithm discussed in the previous chapter can be used to further tune the generated partitions.

7.2 Static Partitioning

In general, designers very frequently delay important steps to the end of design cycles [Wiese 05]. The design of efficient database system is not an exception because partitioning is based on frequencies of queries. That means, the database has to be operational and a large number of queries have to be performed before the partitioning is decided. Thus, the time of partitioning and the parameters used there are weaknesses for the algorithms discussed thus far.

Several surveys indicate that a significant percentage of data warehouses fail to meet business objectives [Giorgini *et al* 08]. One of the reasons for this is that requirement analysis is typically overlooked in real projects. Successful data warehouse design needs to be based upon a requirement analysis phase in order to adequately represent the information needs of DW users [Mazó *et al* 07]. For a newly developed database system which is to be distributed among different sites of an organization, partitioning should be decided at the design phase immediately after completion of the schema. It can be decided even before the database tables are populated. For such a partition, which the author calls static, the database designer must:

1. Gain sufficient knowledge about the business requirements of an organization to design a stable schema.
2. Gather necessary and sufficient information from potential users of the database to determine a complete set of queries that would be of immediate use. This step requires very careful consideration and thorough understanding of the business requirement of an organization.
3. Gather information about future plans of an organization to determine additional queries that may be needed in the future.

In the proposed static partitioning scheme these issues are considered and an improved strategy is adopted. Our approach starts partitioning right upon completion of schema design. Queries that will be used to access the database must be determined in advance. Once the schema design phase is completed, and the database is in operation then the generated partitions can be further tuned using Grouping Algorithm [Abdalla *et al* 07] or Graph Based Algorithms (GBAs).

A simulator was written to test the algorithm using randomly generated queries. Given a database schema with a set of attributes A = $\{A_1, A_2, ..., A_n\}$, the simulator scheme uses a Random Number Generator (*RNG*) to determine a random set of queries Q = $\{q_1, q_2, ..., q_n\}$. The *RNG* will mainly associate a subset of A with each member of the set Q. Since Q is determined using an *RNG* for simulation purpose, one can say that the proposed scheme is not restricted to specific query parameters. The use of the *RNG* may be replaced by a forecasted set of queries for partitioning real schemas. Such set can be reached by a comprehensive study of the business requirements that would lead a Data-Base Designer (DBD) to predict the set.

In general, if the set Q is chosen successfully, then the partition determined by the scheme can be further enhanced using any query-frequency based algorithm. That is, the results of this scheme would give supersets of the sets of partitions independently generated using frequency based dynamic algorithms. The primary difference between this static scheme and the dynamic algorithms is that the former is based on the number of occurrences of an attribute within a set of queries while the latter is based on

frequencies of query accesses to these attributes. In the next subsection the simulator will be discussed.

7.2.1 The Simulator

The simulator for the proposed scheme is a multi-module package that will enable a database designer to partition a newly designed database at the schema level. The output of the simulator may range from 0 to 100 percent partitioning where 0 percent means the schema could not be partitioned and 100 percent means every attribute is placed in a partition by itself. Along with the schema, a complete set of queries and parameters are needed to run the simulator.

Based on the AUM matrix of [Navathe *et al* 84] presented in Table-7.1 as input, the output of the first module of the simulator is a symmetrical table an example of which is shown in Table-7.2. This table will be referred to as the Symmetry Matrix or SM for short. Entries of the SM matrix are computed using the following equations:

$$SM[j, j] = \sum_{i=1}^{n} AUM[i, j] \text{ for } j = 1 \text{ to } n \quad (1)$$

$$SM[i, j] = \sum_{k=1}^{n} AUM(k, i)*AUM(k, j) \text{ For } i \text{ and } j=1 \text{ to } n \text{ and } i \neq j \quad (2)$$

Attributes / Queries	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
Query 1	1	0	0	0	1	0	1	0	0	0
Query 2	0	1	1	0	0	0	0	1	1	0
Query 3	0	0	0	1	0	1	0	0	0	1
Query 4	0	1	0	0	0	0	1	1	0	0
Query 5	1	1	1	0	1	0	1	1	1	0
Query 6	1	0	0	0	1	0	0	0	0	0
Query 7	0	0	1	0	0	0	0	0	1	0
Query 8	0	0	1	1	0	1	0	0	1	1

Table-7.1. Attribute Usage Matrix (Taken from [Navathe *et al* 84])

An example of attribute usage matrix (AUM) generated by the first simulation module is shown on Table-7.1. The table shows the relationship between 10 attributes and 8 queries

that have been randomly generated by the simulator. Table-7.2 shows the SM Matrix derived from the attribute usage matrix of Table-7.1.

Attributes	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
a1	3	1	1	0	3	0	2	1	1	0
a2	1	3	2	0	1	0	2	3	2	0
a3	1	2	4	1	1	1	1	2	4	1
a4	0	0	1	2	0	2	0	0	1	2
a5	3	1	1	0	3	0	2	1	1	0
a6	0	0	1	2	0	2	0	0	1	2
a7	2	2	1	0	2	0	3	2	1	0
a8	1	3	2	0	1	0	2	3	2	0
a9	1	2	4	1	1	1	1	2	4	1
a10	0	0	1	2	0	2	0	0	1	2

Table-7.2. SM Matrix

Diagonal entries of the SM matrix give the usage degree of an attribute. If the matrix is transformed into a graph, then an attribute would be represented by a vertex. Each attribute must have a usage degree of at least one. In Table-7.1 the usage of attribute "a_2" is "3" (total number of 1's in attribute a_2's column). The same usage value is also shown on the second diagonal entry on the SM matrix. A non-diagonal entry defines the symmetry between the corresponding attributes and is equal to the number of queries that include both. For the corresponding graph, the symmetry is modelled by an edge connecting the two vertices. The SM matrix is itself symmetrical across the diagonal.

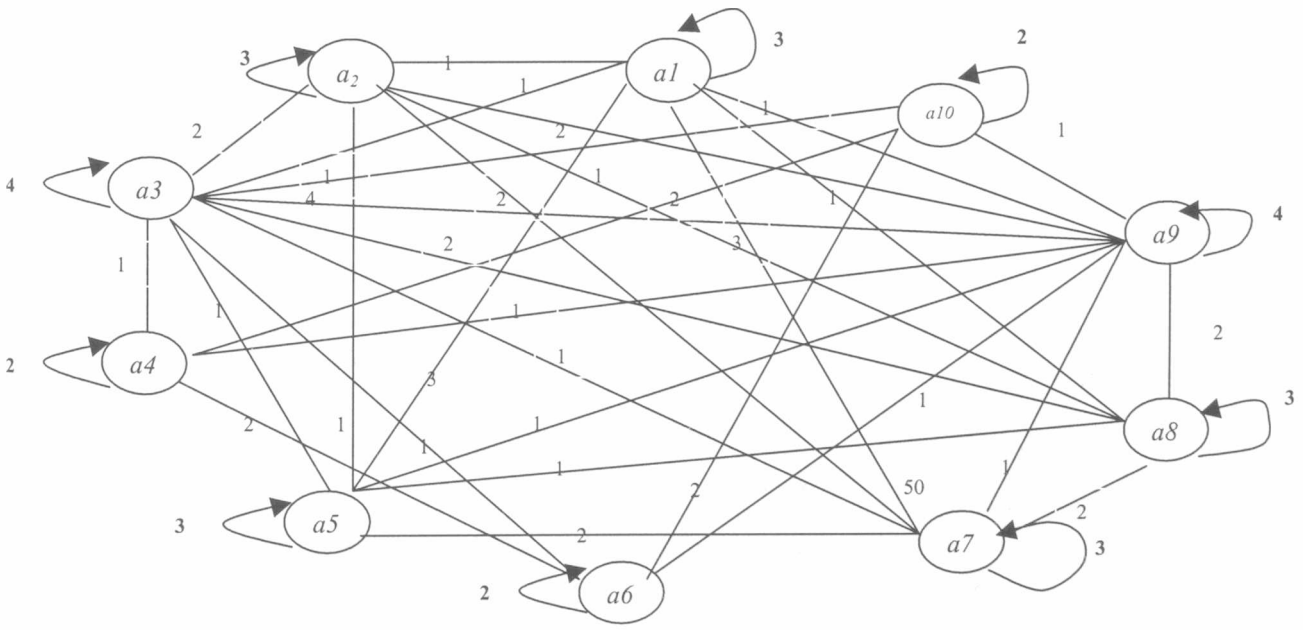

Figure-7.1 Affinity Graph

The second module of the simulator partitions the SM matrix into sub-matrices each corresponds to a partition of the schema.

In the next section more details on how the static partitioning is performed will be given.

7.3 Static Attribute-Based Partitioning (SAPA) Algorithm

The proposed SAPA Algorithm handles some of the drawbacks of its frequency based counterparts that are discussed in the introduction section. In this section SAPA steps will be illustrated. SAPA starts from the SM matrix of Table-7.2 which is a tabular representation of a connected graph (or affinity graph). The affinity graph that corresponds to the SM matrix is shown in Figure-7.1. For discussion of the steps of the algorithm in the next sub section the following definitions are necessary.

Definitions:

1. ***w(v$_i$):*** defines the affinity weight of a vertex *Vi*.
2. ***affinity cycle:*** defines any cycle in the affinity graph.
3. ***w(d$_i$):*** defines the affinity weight of an edge *di*.
4. ***cycle edge:*** is any of the edges forming a cycle.
5. ***extension of a cycle:*** refers to a cycle being extended by pivoting at the cycle vertex.

7.3.1 Description of SAPA Algorithm:

The following steps illustrate how SAPA algorithm works:

a) Construct the affinity graph based on the SM table.

b) Start from the vertex with the highest affinity weight *w(v)*.

c) Branch down along the two (highest and the second highest) weighted edges of the selected vertex in step b starting by the highest weighted edge *w(d)*.

d) Connect to the vertex with the highest affinity weight among the possible choices of vertices with which it has a link. If there are several highest weighted vertices, then any one could be randomly selected.

e) When reaching this step, discontinue moving along this route, go back to step c and extend the cycle along the second highest selected edge route, following the same rules specified in d.

f) After getting to a balanced hierarchy of vertices, to have a further extension of the cycle, start from the vertex with the highest affinity weight among the newly branched two vertices in previous steps, and expand down to include a vertex with largest affinity to the already selected vertex following the same rules specified earlier.

g) This iteration will end when reaching a dead end where the cycle extension process takes you back to one of the already included vertices.

Let us start the explanation of the algorithm to show the mechanism for forming vertical fragments based on the definitions specified above.

1. In Figure 7.2, suppose we started from vertex a_1 where edges '40' and '30' were selected for being the edges with highest and second highest affinity weights among the edges of that vertex. The selected edges '40' and '30' formed the initial "affinity cycle" by connecting vertex a_1 to vertices 'a_2' and 'a_3'. The number between quotes represents the order of connection. For example, because the weight (affinity) between vertices a_1 and a_2 is higher than that of a_1 and a_3, then vertex a_2 was connected first followed by vertex a_3. The order of a connection is shown in between parenthesis next to the vertex number.

2. To further extend the cycle, we have to start from the vertex with the highest affinity weight among the two vertices 'a_2' and 'a_3'.

3. Assuming that vertex 'a_2' was the vertex with highest affinity weight, we select the largest cycle edge emerging from vertex 'a_2' to extend the cycle by connecting vertex 'a_2' to the highest weighted vertex in the remaining vertices linked to it.

4. If there are several vertices with the same weight, then select the vertex that will lead to the best extension, i.e. the vertex that will not take you back to an already selected vertex.

5. Discontinue moving along this route, go back to step 2 and try to extend the cycle using the other selected vertex 'a_3' route, following the same rules specified above.

6. Continue this process until the extension process forces you back to an already selected vertex on one end of the cycle. On figure-7.2, the end vertices are labelled $a_4(1)$ and $a_4(1)$. Clearly this means that both vertices a_2 and a_3 connect to vertex a_4. However, the edge (connection) with the lowest weight should be removed (cut). This is shown by the scribbled line separating vertex a_3 from vertex a_4.

7. If this happens then produce a cut on the last edge that leads to an already selected vertex (a dead end) and consider all connected vertices as a vertical fragment.

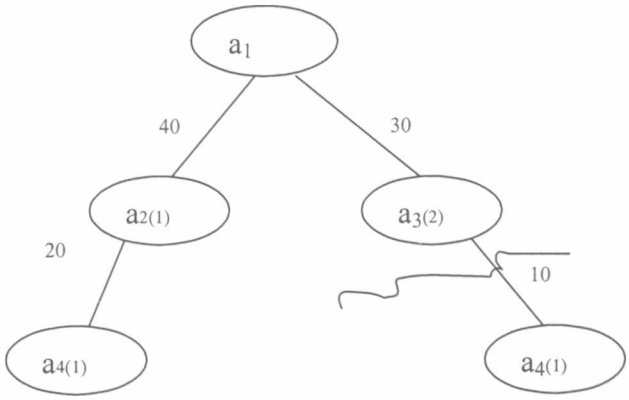

Figure-7.2: Cycle extension leading to a candidate partition

The result of applying the algorithm to affinity graph of Figure-7.1 is shown on Figure-7.3.

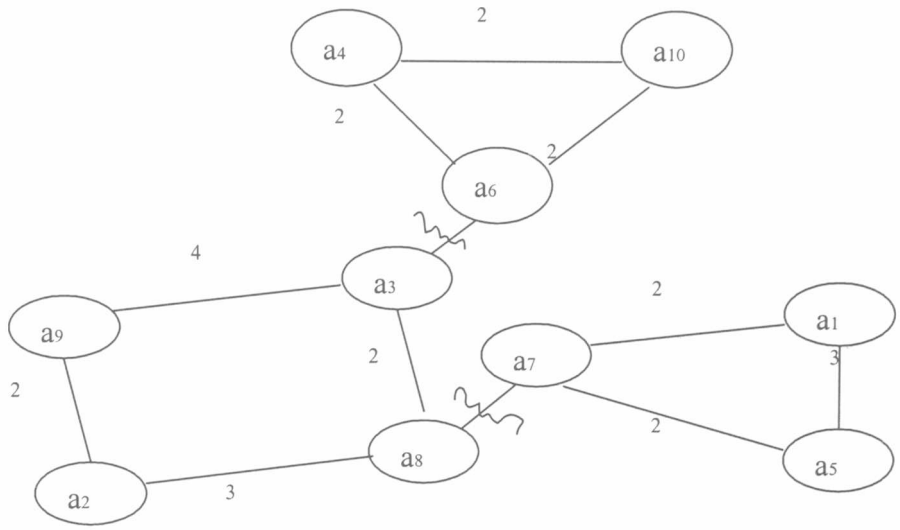

Figure-7.3: Partitioning Result of Figure-7.1

A third module of the simulator can optionally be chosen to reduce (collapse) the number of attributes before the actual partitioning takes place. It is recommended in situations where the number of attributes is extremely large. The use of this option will significantly reduce the size of the corresponding graph and the time taken for partitioning.

In the next section the techniques used for collapsing attributes will be briefly discussed.

7.4 Attribute Partitioning

The collapsing technique uses vertices affinity weights as a basis for fragmentation. It groups vertices (attributes) according to their weights, by collapsing attributes having the same affinity weight values together. Thus, vertices with identical weights will end up in the same fragment. For example, after applying the algorithm on Table-7.2, attributes (a_3 and a_9) are grouped in one partition, attributes (a_1, a_2, a_5, a_7, a_8) in another partition and attributes (a_4, a_6 and a_{10}) in a third partition with weights 4, 3 and 2 respectively.

After collapsing the affinity graph in Figure-7.1 we end up with a fragmentation scheme as shown in Figure-7.4.

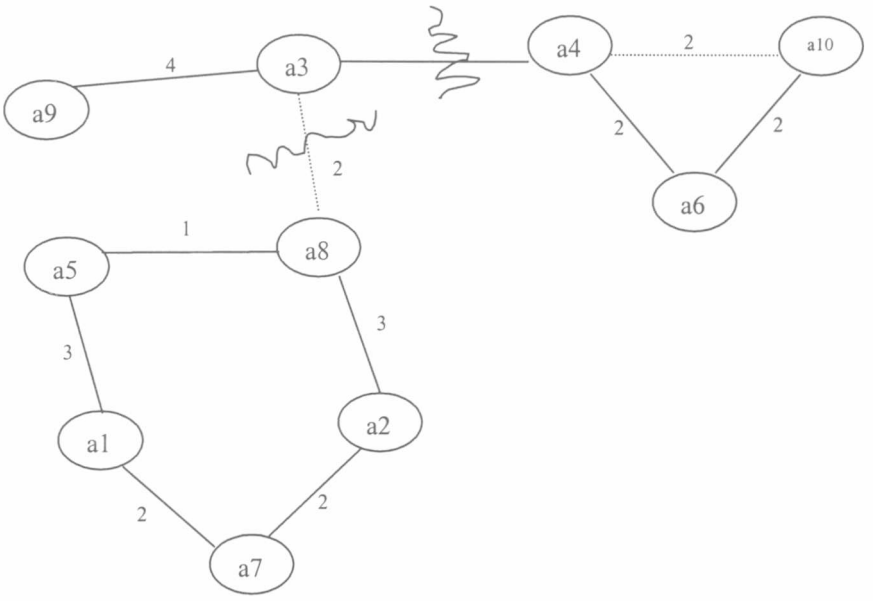

Figure-7.4: Vertical fragments generated by collapsing algorithm

7.5 Summary

In this work, a vertical partitioning algorithm for improving the performance of warehouse database systems was proposed. The proposed SAPA algorithm used the number of occurrences of an attribute in a set of queries rather than the frequency of queries accessing these attributes. This enabled the fragmentation of a database schema even before its tables are populated. Thus, a database designer will be in a position to perform partitioning and consequent distribution of fragments before the database enters operation. This approach facilitated the possibility of building a distribution design that could be complemented at a later stage with frequency based algorithms. The significant advantage of the SAPA algorithm was that a database designer doesn't have to wait for empirical data on query frequencies before partitioning a database.

In this chapter and the previous one we focused on vertical fragmentation problem in distributed environment. In the following chapter a horizontal partitioning method that combines genetic and simulated annealing algorithms to solve the data partitioning problem in a data warehousing environment will be considered.

Chapter 8

SAGA for Physical Warehouse Design and Implementation

8.1 Introduction

A data warehouse stores large amounts of consolidated, historical data. It is especially designed to support complex business decision queries. This complexity is due to the presence of hierarchies between attributes of the warehouse and the number of join and aggregation operations. Data warehouses are typically updated only periodically, hence they are mostly of read-only in nature.

Several query optimization methods were proposed: materialized views [Yang and Karlapalem 97], indexes [Chaudhuri and Narasayya 98], clustering [Jagadish *et al* 99], data partitioning [Sanjay *et al* 04], and parallel processing [St¨ohr *et al* 00]. Data partitioning is an important aspect of physical data warehouse design [Sanjay *et al* 04], [Papadomanolakis and Ailamaki 04]. It has a significant impact on performance and manageability. Contrary to materialized views and indexes, data partitioning does not replicate data, thereby reducing space requirements and minimizing update overhead [Papadomanolakis and Ailamaki 04].

The main characteristic of data partitioning is its ability to be combined with other optimization structures like indexes and materialized views [Bellatreche *et al* 04]. In the context of relational warehouses, horizontal partitioning allows tables, indexes and materialized views to be partitioned into disjoint set of rows that are physically stored and accessed separately [Sanjay *et al* . 04]. It can also be used for parallel data warehousing [St¨ohr *et al* 00].

To partition a relational warehouse modelled by a star schema (or snowflake schema) having a set of dimension tables and a fact table, we advocate a fragmentation of the fact table based on the fragmentation schemas of the dimension tables. Concretely, it consists of partitioning some or all dimension tables using their simple selection predicates

defined in the set of most frequently executed queries, and then partition the fact table using the fragmentation schemas of the fragmented dimension tables (the fragmentation of the fact table is called derived horizontal fragmentation [Ceri *et al* 82]).

8.2 Horizontal Partitioning Selection Problem

In this section, the horizontal partitioning design will be highlighted. It is extensively used to make the management of database servers easier. Several works and commercial systems have shown its impact on optimizing OLAP queries [Sanjay et al 04], [Kalnis and Papadias 01], [St¨ohr et al 00], [Bellatreche et al 04]. But few studies have formalized the problem of selecting a horizontal partitioning schema to speed up a set of queries and proposed selection algorithms.

The performance of query processing depends on the physical schema design. A highly normalized schema offers superior performance and efficient storage where only a few attributes are accessed by a particular query. The star schema [Sanjay et al 04] provides similar benefits for data warehouses where most queries aggregate a large amount of data. Such a schema is an intuitive way to represent the multidimensional data of a typical business, in a relational system. The queries usually filter rows based on dimensional attributes, then group and aggregate the attributes of the fact table using some dimensional attributes.

Suppose that warehouse modelled by a star schema with d dimension tables and a fact table F. Among these dimension tables g tables were fragmented ($g \leq d$). Suppose that the dimension table D_i ($1 \leq i \leq g$) is partitioned into mi fragments: $\{D_{i1}, D_{i2}, ..., D_{imi}\}$, where each fragment D_{ij} is defined as:
$D_{ij} = \sigma_{cl^i_j}(D_i)$ with cl^i_j and σ ($1 \leq i \leq g$, $1 \leq j \leq m_i$) represent a conjunction of simple predicates and selection operator respectively.
Thus, the fragmentation schema of the fact table F is defined as follows:
$F_i = F \ltimes D_{1i} \ltimes D_{2i} \ltimes \ ... \ltimes \ D_{gi}$, ($1 \leq i \leq m_i$) with $\ltimes$ represents the semi join operation.

To illustrate the procedure of fragmenting the fact table based on the fragmentation schemas of dimension tables, let us consider the star schema presented in [Informix 97] with three dimension tables: *Customer*, *Time* and *Product* and one fact table *Sales*. The tables and attributes of the schema are shown in Figure 8.1. Suppose that the dimension table *Customer* is fragmented into two fragments $Cust_1$ and $Cust_2$ defined by the following clauses: $Cust_1 = \sigma_{Sex='M'}$ (*Customer*) and $Cust_2 = \sigma_{Sex='F'}$ (*Customer*).

Therefore, the fact table Sales is fragmented based on the partitioning of Customer into two fragments $Sales_1$ *and* $Sales_2$ *such that:* $Sales_1 = Sales \ltimes Cust_1$ *and* $Sales_2 = Sales \ltimes Cust_2$. After partitioning *Sales* and *Customer* tables the initial star schema (*Sales, Customer, Product, Time)* is represented as the union of two sub star schemas S_1 and S_2 such as: S_1: ($Sales_1$, $Cust_1$, *Product, Time*) (sales activities for only male customers) and S_2 : ($Sales_2$,$Cust_2$, *Product, Time*) (sales activities for only female customers).

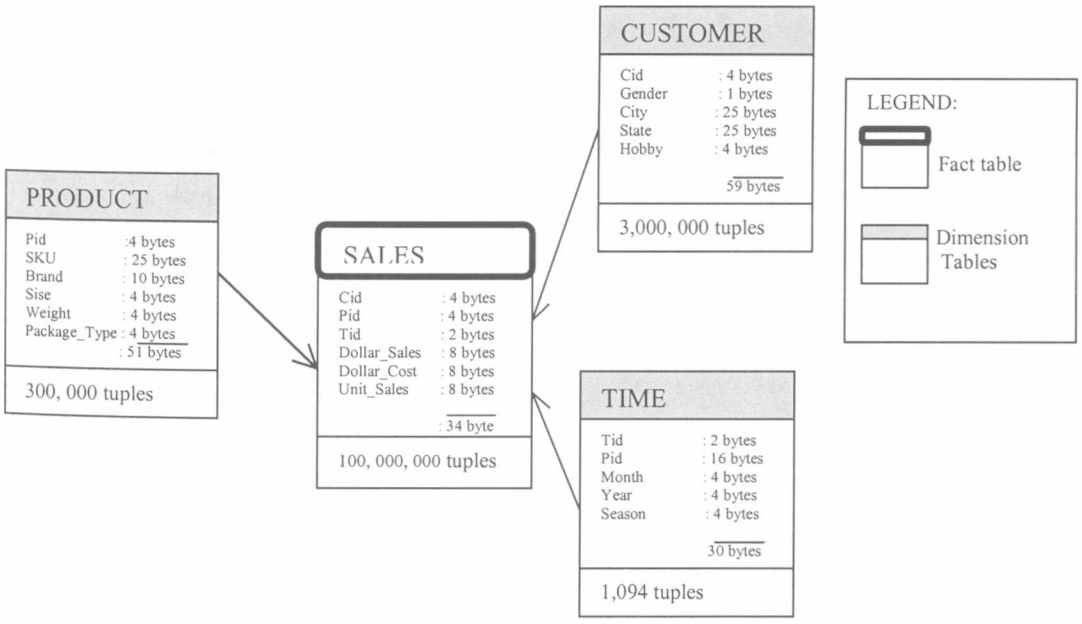

Figure 8.1: An example of a Star Schema

The number of horizontal fragments of the fact table generated by the partitioning procedure (denoted by N) is given by the following equation:

$$N = \prod_{i=1}^{g} m_i$$

Where m_i is the number of fragments of the dimension table D_i. This fragmentation technique may generate a large number of fragments of the fact table. For example, suppose we have: *Customer* dimension table partitioned into 50 fragments using the State attribute (using 50 states of the U.S.A), *Time* into 36 fragments using the Month attribute (if the sales analysis is done based on the last three years), and *Product* into 80 fragments using Package type attribute, therefore the fact table will be fragmented into 144000 fragments (50 × 36 × 80). Consequently, instead of managing one star schema, the data warehouse administrator will manage 144000 sub star schemas. Indeed, it will be very hard for her/him to maintain all these sub-star schemas. Therefore, it is necessary to reduce the number of fragments of the fact table in order to guarantee two main objectives: (1) avoid an explosion of the number of the fact fragments and (2) ensure a good performance of OLAP queries. To satisfy the first objective, the DWA is given the possibility to choose the maximum number of fragment that she/he can maintain (threshold W). For the second one, the number of fragments can be increased to a limit where the global performance will be satisfied. The problem of selecting an optimal fragmentation schema includes finding a compromise between the maintenance cost and the performance cost as shown in Figure 8.2 quoted fromy [Noaman and Barker 99]. In order to satisfy this compromise, genetic algorithm [B¨ack 95] is been used since it explores a large search space.

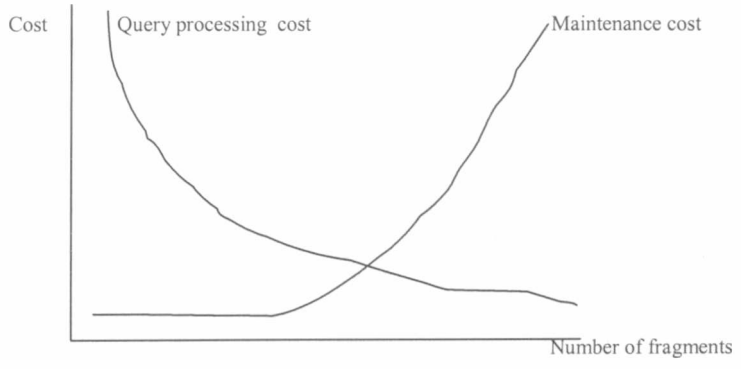

Figure 8.2: Compromise between maintenance and the processing cost

8.2.1 A Formulation of Horizontal Partitioning Selection Problem

A formulation of the horizontal partitioning problem is the following: given (1) a set of dimension tables $D = \{D_1, D_2, ..., D_d\}$ and a fact table F, (2) a set of OLAP queries $Q = \{Q_1, Q_2, ..., Q_m\}$, where each query Q_i ($1 \leq i \leq m$) has an access frequency, and (3) a threshold (W fixed by the DWA) representing the maximum number of fragments that she/he can maintain. The horizontal partitioning problem is about determining a set of dimension tables $D' \subseteq D$ to be partitioned and using their fragmentation schemas to partition the fact table F into a set of horizontal fragments $\{F_1, F_2, ..., F_N\}$ such that: the sum of the query cost when executed on top of the partitioned star schema is minimized, and the maintenance constraint ($N \leq W$) is satisfied, where W is a threshold fixed by the DWA representing the maximal number of fragments that she/he can maintain. The respect of this constraint avoids an explosion of the number of the fact fragments.

To respond to horizontal partitioning selection problem, a combination of genetic and simulated annealing algorithms [B¨ack 95 and Bennett *et al* 91] was used.

8.2.2 Simulated Annealing and Genetic Algorithm (SAGA) Approach

Our approach, SAGA, first uses a genetic algorithm (GA) and then a simulated annealing algorithm (SA). GAs has been widely used in database physical design, they are particularly suitable for solving complex optimization problems and for applications that require adaptive problem solving strategies. It is an adaptive search technique based on the principles and mechanisms of natural selection and 'survival of the fittest' from natural evolution. It can effectively search the problem domain and easily solve complex problems.

Join query optimization problem [Ioannidis and Kang 90], materialized views selection problem [Zhang and Yang 99] and automation of physical design of parallel databases [Rao *et al* 02] can be cited. It has been adopted for optimizing join operation, and due to the similarity between index selection problem and horizontal fragmentation selection

problem in the data warehouse environment [Bellatreche *et al* 04], GA have been provoked to be used for this problem.

Genetic algorithms (first proposed by Holland in 1975 [Holland 75]) are a class of computational models that mimic natural evolution to solve problems in a wide variety of domains. They are search methods based on the evolutionary concept of natural mutation and the survival of the fittest individuals. Given a well-defined search space they apply three different genetic search operations, namely, *selection, crossover, and mutation*, to transform an initial population of chromosomes, with the objective to improve their quality. After the process of selection, recombination and mutation is complete, the next population can be evaluated.

The process of evaluation, selection, recombination and mutation forms one generation in the execution of a genetic algorithm [Whitley 93]. Fundamental to the GA structure is the notion of chromosome, which is an encoded representation of a feasible solution. Before the search process starts, a set of chromosomes is initialized to form the first generation. Then the three genetic search operations are repeatedly applied, in order to obtain a population with better characteristics. Figure 8.3 shows the basic structure of our GA.
An outline of a generic GA is as follows:

Generate initial population
Perform selection step
while** stopping criterion not met **do
    Perform crossover step
    Perform mutation step
    Perform selection step
***end while**.*
Report the best chromosome as the final solution.

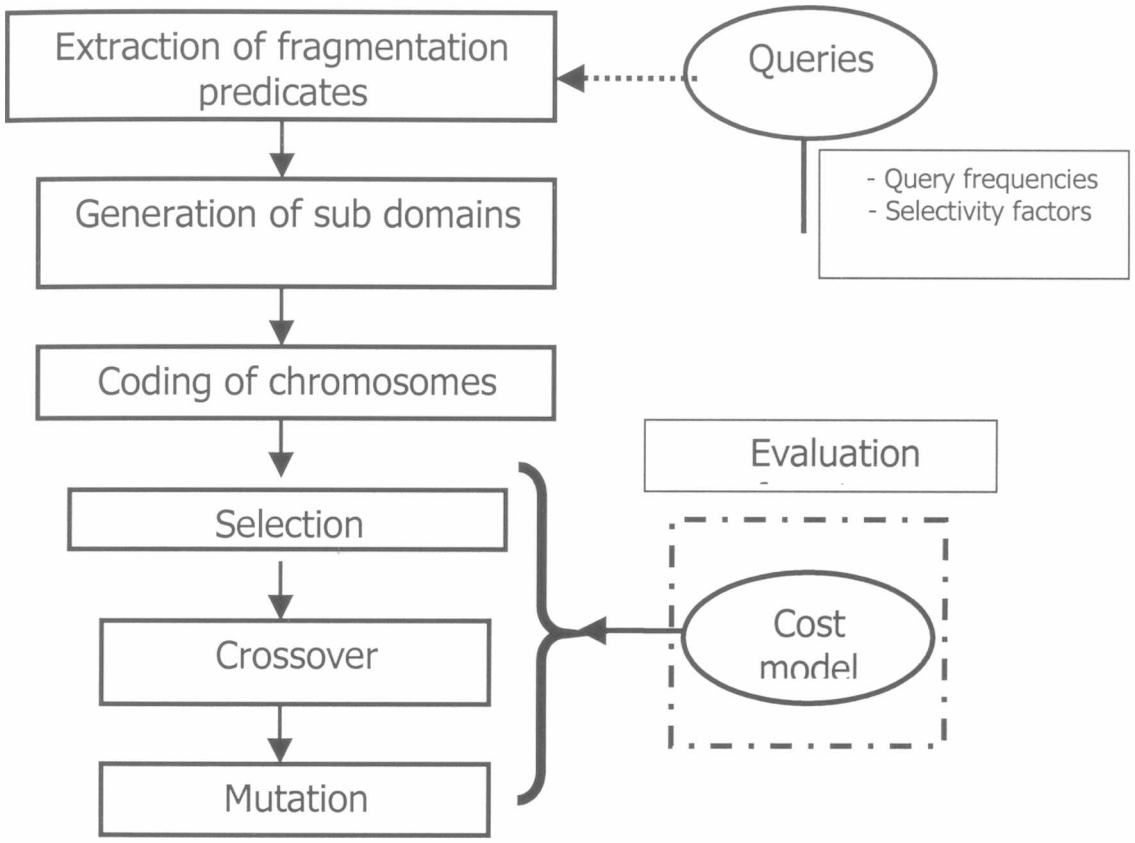

Figure 8.3: The structure of the genetic algorithm

On the other hand, the **Simulated Annealing (SA)** is an evolution strategy that is also a population-based form of search that has largely been developed for parameter optimization problems. Evolution strategies generally use real-valued encodings and emphasize the use of mutation rather than recombination. They use strategy parameters that control mutation step size for each parameter on the chromosome.

SA avoids the problem of premature convergence inherent to GA by allowing uphill moves to solutions of worse fitness. SA [Kirkpatrick *et al* 83] is a randomized technique for finding a near-optimal solution of difficult combinatorial optimization problems. The algorithm was originally proposed as a means of finding the equilibrium configuration of a collection of atoms at a given temperature. It starts with a randomly solution candidate, then it repeatedly attempts to find a better solution by moving to a neighbour with higher fitness, until it reaches a solution where none of its neighbours has a higher fitness.

SA's major advantage over other methods is an ability to avoid becoming trapped at local minima. The algorithm improves this strategy through the introduction of different mechanisms. The most famous one is the so-called "Metropolis algorithm" [Metropolis *et al* 58], in which some trades that do not lower the cost are accepted under certain conditions when they allow the solver to explore more of the possible space of solutions.

Simulated annealing takes a variety of forms, but the method presented here is one of the simpler ones. Simulated annealing basically involves perturbing the independent variables by a random value, and keeping track of the value with the least error. To avoid getting trapped in poor local optima, SA allows occasionally an uphill moves to solutions with lower fitness by using a temperature parameter to control the acceptance of the moves. The temperature takes on the form of the standard deviation used by the random number generator. At a high temperature, a large range is sampled, but as the temperature decreases so does the sampling range and as a result the error becomes smaller and smaller.

The principal difference between a genetic algorithm and an evolutionary strategy like simulated annealing is that the former relies on crossover to locate better solutions, while the latter uses mutation as the primary search mechanism.

Algorithm 1 illustrates the behaviour of SA. In the inner loop, the temperature is kept constant. A downhill move is always allowed. An uphill move is allowed with some probability that depends on the temperature and the difference between the actual state's cost and the new state's cost. The inner loop is finished when an equilibrium condition is met.

Then the temperature is reduced and the inner loop is started again. The outer loop is finished when a freezing condition is met. The parameters of SA are: initial state, initial temperature, temperature reduction, equilibrium condition, and freezing condition depends on the implementation of the process.

Algorithm 1 Simulated Annealing Algorithm

Input: initial state, initial temperature;
Output: minstate;
begin
minstate:= initial state; cost := Cost(initial state); mincost := cost;
temp := initial temperature;
repeat
repeat
newstate:=state after random move; newcost:= Cost(newstate);
if (newcost ≤cost) **then** state := newstate; cost := newcost
else with probability $e^{(\frac{newcost-cost}{temperature})}$ > rand(0,1)
state := newstate; cost := newcost
end;
if (*cost* < *mincost*) **then** minstate := state; mincost := cost
end
until equilibrium not reached;
reduce temperature
until not frozen;
return minstate

end

8.3 Implementation of Genetic Algorithm

The most difficult part of applying GA is the representation of the solutions that represent fragmentation schemas. Representation of a solution (chromosome) is one of the key issues in problem solving. Traditionally, GAs uses binary string as their chromosomes representation. In our study, a solution represents a fragmentation schema.

8.3.1 Coding Mechanism

To accomplish chromosomes representation, then application information need to be defined on the tables to be partitioned [Özsu and Valduriez 99], like selectivity factors of selection predicates and the frequencies of queries accessing these tables (Q = $\{Q_1,\ldots,Q_m\}$).

Before presenting the proposed coding mechanism of dimension fragments, dimension tables that will participate in fragmenting the fact table should be identified. To do so, the following procedure is employed:

(1) Extract all simple predicates used by the *m* queries.

(2) Assign to each dimension table D_i ($1 \leq \leq d$) its set of simple predicates, denoted by $SSPD_i$.

(3) Dimension tables having an empty *SSPD* will not be considered in the partitioning process.

(4) Apply the COMM-MIN algorithm [Özsu and Valduriez 99] to each dimension table D_i of all dimension tables having a non-empty *SSPD*.

Note that each fragmentation attribute has a domain of values. The clauses of simple predicates representing horizontal fragments define partitions of each attribute domain into sub domains. The cross product of fragmentation attributes by all applicable sub domains determines the total number of fragments schemas of the facts table.

Using the star schema of figure 8.1, consider two fragmentation attributes, *Age* and *Gender* of dimension table *CUSTOMER* and one attribute *Season* of dimension table *TIME*. The domains of these attributes are defined as follows:

Dom (Age) = {*0 - 120*}. *Dom* (Season) = {*'Summer'*, *'Spring'*, *'Autumn'*, *'Winter'*}. *Dom* (Gender) = {*'M'*, *'F'*}. Suppose that on attribute Age, three simple predicates are defined as follows: p1: (*Age* ≤ 18), p2: (18 < *Age* < 60), and p3: (*Age* ≥ 60). The domain of this attribute is then partitioned into three sub domains: *Dom*(Age) = $d_{11} \cup d_{12} \cup d_{13}$, with d_{11} = [0 - 18], d_{12} = [18 - 60], d_{13} = [60 - 120]. Similarly, the domain of Gender attribute is decomposed into two sub domains: *Dom*(Gender) = $d_{21} \cup d_{22}$, with d_{21} = {'M'}, d_{22} = {'F'}.

Finally, domain of Season is partitioned into four sub domains: $Dom(\text{Season}) = d_{31} \cup d_{32} \cup d_{33} \cup d_{34}$, where d_{31} = {'Summer'}, d_{32} = {'Spring'}, d_{33} = {'Autumn'}, and d_{34} = {'Winter'}. The different sub domains of all three fragmentation attributes are represented in Figure 8.4.

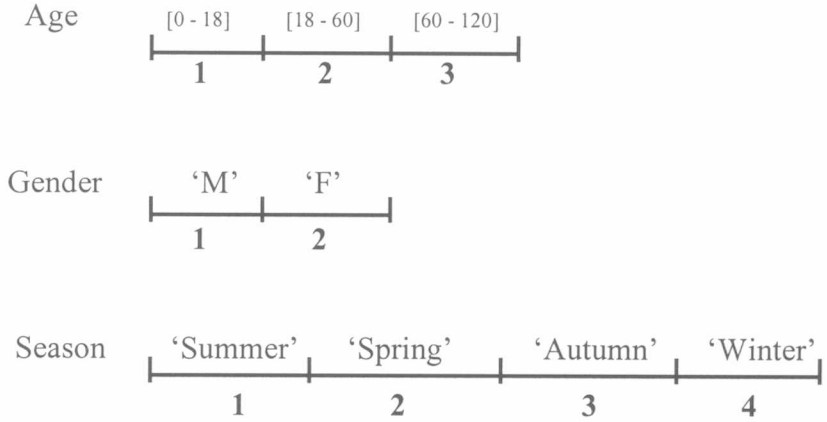

Figure 8.4: Sub domains of a fragmentation attributes

Each fragmentation attribute can be represented by an array with n cells, where n corresponds to number of its sub domains. The values of these cells are between 1 and n. Consequently, each chromosome (fragmentation schema or solution) can be represented by a multi-dimensional array.

Gender	1	1		
Season	2	1	3	3
Age	2	1	2	

Table 8.1: An example of a possible solution

From this representation, the obtainment of horizontal fragments is just a matter of generating all conjunctive clauses. If two cells of the same array have the same value, then they will be merged.

For example, in table 8.1, it can be deduced that the fragmentation of the data warehouse is not performed using the attribute Gender, because all its sub domains have the same value. Consequently, the warehouse will be fragmented using only Season and Age. For

Season attribute, three simple predicates are possible: P_1: Season = "Summer", P_2: Season = "Spring", and P_3 : (Season = "Autumn") $\vee$ (Season = "Winter").

For Age attribute, two predicates are possible: P_4 : (Age $\leq$18) $\vee$ (Age $\geq$60) and P_5: (18 < Age < 60) Therefore, the data warehouse can be fragmented into six fragments defined by the following clauses: Cl_1 : ($P_1 \cap P_4$), Cl_2 : ($P_1 \cap P_5$), Cl_3 : ($P_2 \cap P_4$), Cl_4 : ($P_2 \cap P_5$), Cl_5 : ($P_3 \cap P_4$), and Cl_6 : ($P_3 \cap P_5$). Note that each fragment is represented by a conjunctive of simple predicates defined on fragmentation attributes.

The proposed coding satisfies the correctness rules of [Özsu and Valduriez, 99] (completeness, reconstruction and disjointness) and the new chromosomes generated by crossover operations belongs to the relevant sub domains. This coding can be used to represent fragments of dimension tables and fact table. This multidimensional array representation may be used to generate all possible fragmentation schemas (using an exhaustive search). This number is calculated as:

$2^{\sum_{i=1}^{k} n_i}$, where K represents the number of fragmentation attributes and n corresponds to number of their sub domains. Considering the three fragmentation attributes of *Age*, *Gender* and *Season* (in the previous example), the number of all possible fragmentation schemas is $2^{(3+2+4)} = 2^9$.

This number is quite similar to the possible minterms predicates generated by horizontal fragmentation algorithm proposed by [Özsu and Valduriez, 99].

8.3.2 Fitness Value

The quality of each chromosome is measured by computing its fitness value represented by a cost model which calculates the sum of page accesses (inputs/outputs) incurred by each query executed on the fragmented data warehouse. A cost model calculating the

number of inputs and outputs (IOs) for each query is used. To estimate the cost of queries, it is assumed that all dimension tables are in the main memory. Let $D^{sel} = \{D_1^{sel}, ..., D_k^{sel}\}$ be the set of dimension tables having selection predicates, where each selection predicate *pj* (defined on a dimension table *Di*) has a selectivity factor denoted by $Sel_{D_i}^{P_j}$ ($Sel_{D_i}^{P_j} \in [0,1]$). For each predicate p_j, its selectivity factor on the fact table is defined, denoted by $Sel_F^{P_j}$ *where* $Sel_{D_i}^{P_j} \neq Sel_F^{P_j}$.

Let's consider the selection predicate Gender = "Female" defined on the dimension table *Customer*. Suppose that its selectivity factor is 0.4. This means that 40% of customers are female and 60% are male.

To execute a query Q_k over a partitioned star schema $\{S_1, S_2, ..., S_N\}$, the relevant sub star schema(s) on which query Q_k will be executed should be identified. To do so, we introduce a boolean variable denoted by $valid(Q_k, S_i)$ and defined as follows:

$$Valid(Q_k, S_i) = \begin{cases} 1 \text{ if the sub star schema } S_i \text{ is needed for query } Q_k \\ 0 \text{ otherwise} \end{cases}$$

The number of IOs for executing a query Q_k over a partitioned star schema is given by the following equation:

$$Cost(Q_k) = \sum_{j=1}^{n} valid(Q_k, S_i) \prod_{i=1}^{Mj} \left\lceil \frac{Sel_F^{p_i} \times \| F \| \times L}{PS} \right\rceil \qquad (1)$$

where M_j represent the number of selection predicates defining the fact fragment of the sub star schema S_j, F the cardinality of the fact table (number of tuples), L the width in bytes of a tuple of a table F and PS the page size of the file system (in bytes). The total cost (TC) of executing a set of queries Q is given by:

$$TC(Q) = \sum_{k=1}^{m} Cost(Q_k) \qquad (2)$$

8.3.3 Selection Operation

Selection in GAs determines the probability of chromosomes being selected for reproduction. The principle is to assign higher probabilities to filter chromosomes. The roulette wheel method (see Figure 8.5) proposed by [Holland 75] is used in our algorithm. It works as follows:

- *Add up the fitness of all chromosomes*
- *Generate a random number R in that range*
- *Select the first chromosome in the population that, when all previous fitnesses are added - gives you at least the value R*

Individual i will have a $\frac{f(i)}{\sum_i f(i)}$ probability to be chosen

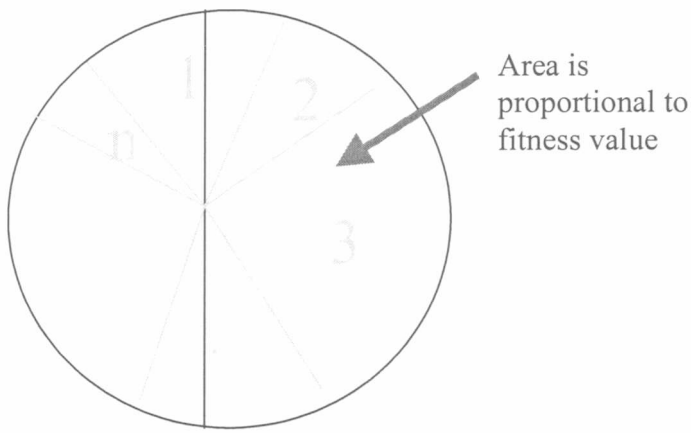

Figure 8.5: Roulette wheel selection

In this method, each chromosome is associated with its fitness value calculated using the cost model defined in section 8.3.2. The chromosomes with high fitness values have better chances to be selected. Table 8.2 and Figure 8.6 show a roulette wheel method example of chromosomes selection.

No.	Chromosome	Fitness
1	12121233	1
2	11121223	2
3	12111122	1
4	12111234	3
5	11223344	3
6	11121111	5
7	12211224	1
8	22122344	2

Table 8.2: An example of chromosomes' fitness representation

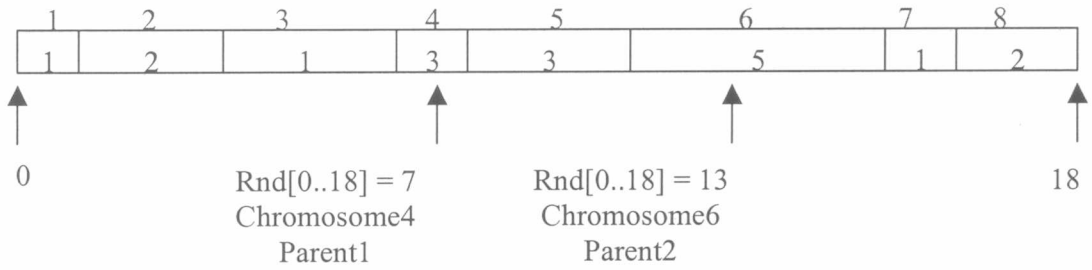

Figure 8.6: Selection of the chromosomes

8.3.4 Crossover Operation

A two-point crossover mechanism has been used in the proposed GA to avoid that attributes with high number of sub domains, like *Season* (with four sub domains) having a greater probability to be crossed over than attributes with low number of sub domains, like *gender* (with two sub domains). The rationale behind crossover operation is that after the exchange of genetic materials, it is very likely that the two newly generated chromosomes will possess the good characteristics of their both parents (building-block hypothesis [Holland 75]). An example of a crossover operation is illustrated` in Figure 8.7.

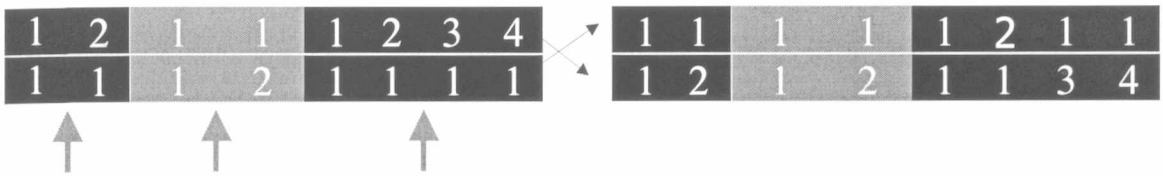

Figure 8.7: Crossover of chromosomes 4 and 6

8.3.5 Mutation Operation

Although crossover can put good chromosome together to generate better offspring, it cannot generate new genes. Mutation is needed to create new genes that may not be present in any member of a population. It is an operation aiming at restoring lost genetic material and is performed in our algorithm by simply flipping the selected bit with certain probability, called the mutation rate.

A mutation rate between 6 to 30 percent has been chosen. Mutation rate of 30% gave good performance as it enabled the algorithm to reach a reasonable number of possible solutions in the search space. However, after several generations a mutation rate of 6% was used to avoid redundant search [Bellatreche *et al* 06]. Initialization of the first generation is performed by randomly generating half of the population while the rest is obtained from solutions previously found by the algorithm.

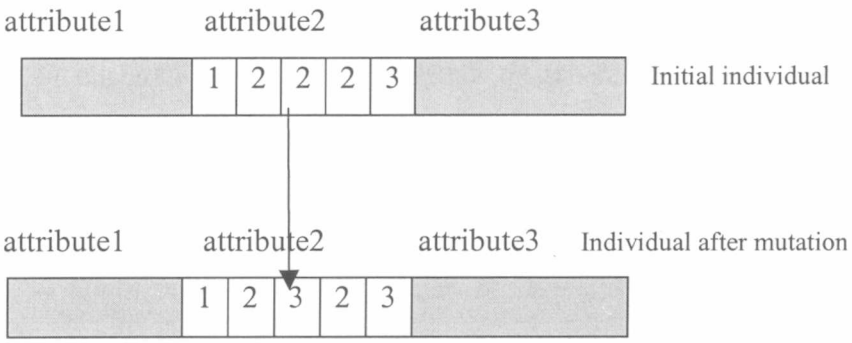

Figure 8.8: An example of a mutation

In Figure 8.8, the initial chromosome has three fragments for the attribute 2. After the mutation process, the resulting individual has also three fragments, but the intervals are not the same. In practice, there could be more distinct intervals or merged intervals. In the same way, mutations could occur on several attributes of the individual.

8.4 Implementation of Simulated Annealing Algorithm

The SA is applied on the final solution obtained by the GA. This means that the initial state of SA is the fragmentation schema generated by the GA. Note that this solution is represented by multidimensional arrays. Random moves used by SA are applied on the final multidimensional array of GA. In order to facilitate their implementation, this array is transformed into one dimensional array, by concatenating all its rows. This gives a new representation of the fragmentation schema of fact table. The random moves generate a new problem, called the validation of a solution. This is due to the fact that our SA is concerned with modifying cell values of the array by incrementing or decrementing them. However, this may cause an overflow of domain values of cells (Figure 8.9). To solve this problem, a function that checks the validity of each solution generated by SA has been developed.

Simulated annealing avoids the problem of being trapped in a local minimum through the introduction of a method by which some "bad" random moves that do not lower the cost are accepted when they serve to allow exploring more of possible solutions. Such "bad" moves are allowed using the criterion that: $e^{-\Delta D/T} > R(0,1)$

Where ΔD is the change of cost implied by the move (negative for a "good" move, positive for a "bad" move), T is a control parameter, which by analogy with the original application is known as the system "temperature", R (0, 1) is a random number in the interval [0, 1] and D is the "cost function".

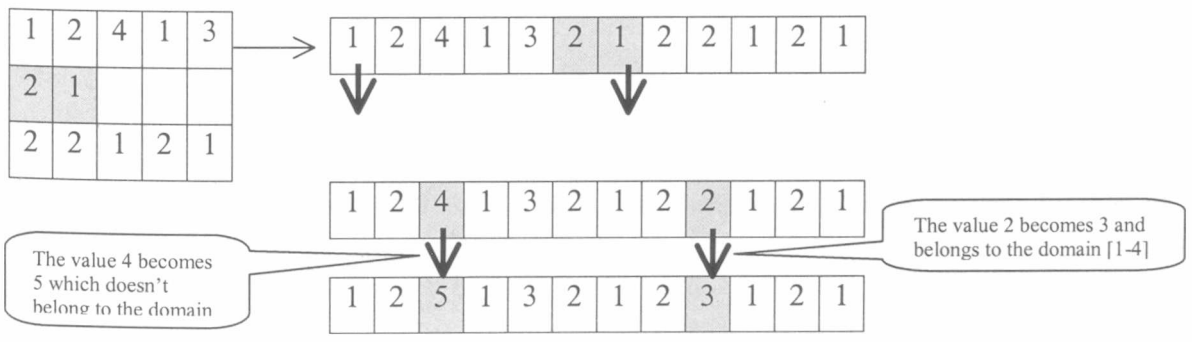

Figure 8.9: Example of SA

The fitness of each solution is calculated using the cost model (TC) developed in Section 8.3.2. The steps of our SA are shown in Algorithm 2.

8.4.1 A Concise Description of SA

1. Select a fragmentation schema using GA.

2. Check that chromosomes are transformed into an array of one dimension by concatenating the rows of the array. This array gives a new representation of the fragmentation schemas.

3. Choose two positions of the fragmentation schema and apply a smooth transformation (random move) by applying a coefficient between 0 and 1 and add (or subtract) to the result of the old value.

4. If the score of the resulting fragmentation schema is better than the previous one then save the new result, otherwise, compute the difference between the scores of the current and the previous fragmentation schemas, called Delta and compute the deterioration as: $e^{\frac{-Delta}{Temperature}}$.

5. A random number is drawn, if the deterioration is greater than this number then still save the new result and continue the search, otherwise go to 2.

6. Reduce the temperature.

7. Repeat step 2-6 for t number of temperatures until certain number of iterations without improvement.

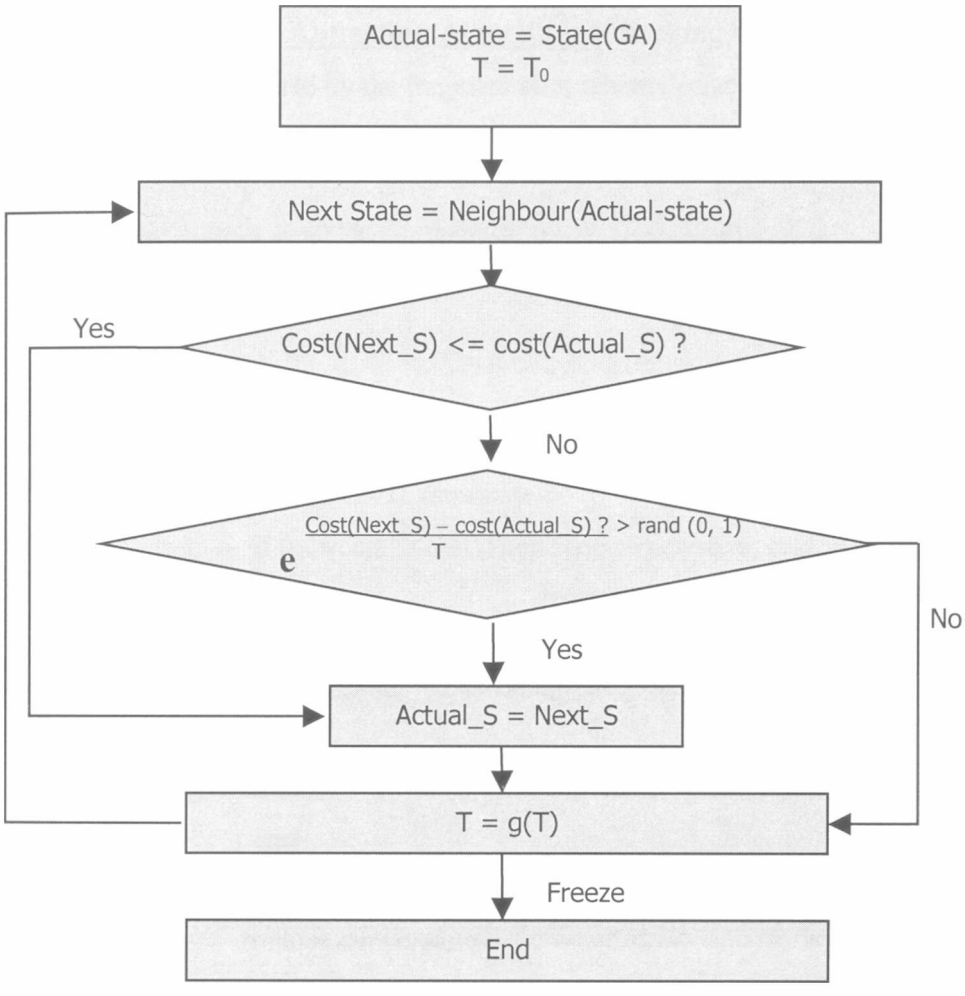

Figure 8.10: The structure of the SA algorithm

The implementation of the SA algorithm is remarkably easy. Figure 8.10 shows its basic structure. The following elements must be provided:

- a representation of possible solutions,
- a generator of random changes in solutions,
- a means of evaluating the problem functions, and
- an *annealing schedule* - an initial temperature and rules for lowering it as the search progresses.

Algorithm 2 Simulated Annealing Algorithm (Checking validity of each solution)

Input: initial state represented by the fragmentation schema generated by GA, initial temperature;

Output: minstate;

begin

minstate:= initial state; cost := **TC**(initial state); mincost := cost;

temp := initial temperature;

repeat

repeat

newstate:=state after random move; *validation_ check(newstate);*

newcost:= TC(newstate);

if (newcost ≤cost) **then** state := newstate; cost := newcost

else with probability $e^{(\frac{newcost-cost}{temperature})}$ > rand(0,1)

state := newstate; cost := newcost

end;

if (*cost* < *mincost*) **then** minstate := state; mincost := cost

end

until equilibrium not reached;

reduce temperature

until not frozen;

return minstate

end

8.5 Experimental Studies

Our proposed solution has been implemented using APB-1 benchmark [OLAP Council 98]. The experimental results are encouraging and show the applicability of the approach. The star schema of this benchmark has one fact table: **Actvars** (Product level, Customer level, Time level, Channel level, UnitsSold, DollarSales, DollarCost), and four dimension tables: **Prodlevel** (Code level, Class level, Group level, Family level, Line level, Division level), **Custlevel** (Store level, Retailer level), **Timelevel** (Tid, Year level, Quarter level, Month level, Week level, Day level), **Chanlevel** (Base level, All level). The Star Schema of APB-1 Benchmark is shown in Figure 8.11.

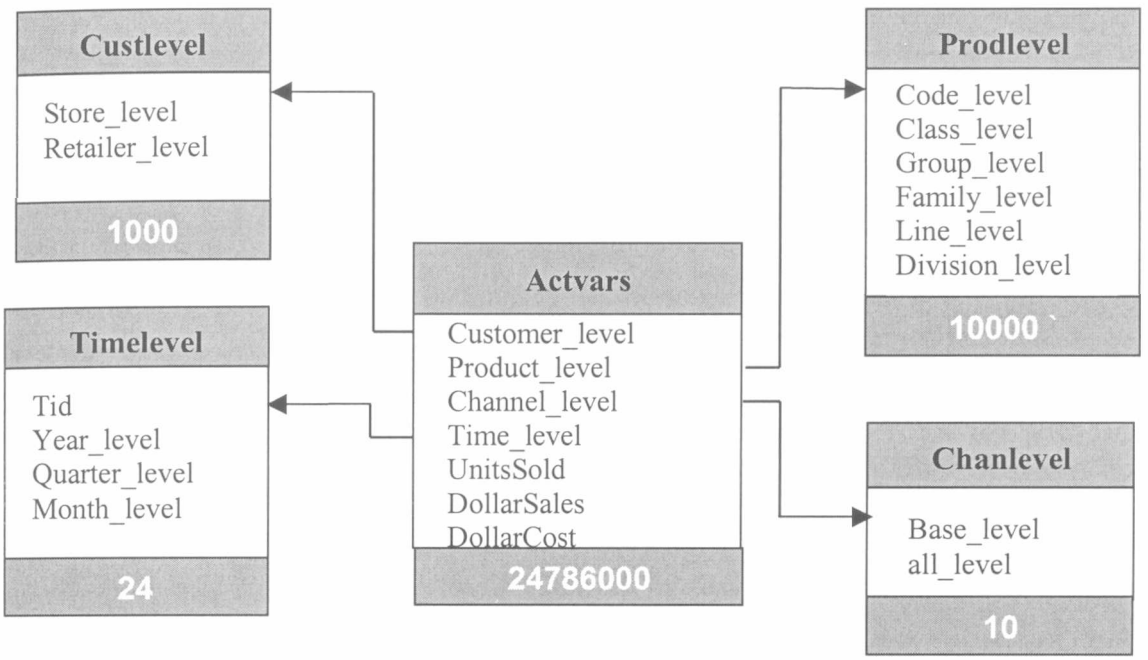

Figure 8.11: Star Schema of APB-1 Benchmark

The experimental studies went through three steps: (1) firstly the good parameters for the genetic algorithms were identified, (2) secondly, experiments based on these parameters were run, (3) finally, experiment by combining genetic algorithm and simulated annealing algorithm was conducted. This warehouse has been populated using the generation module of APB1. Each dimension table can be joined with the fact table through its first attribute.

Our simulation software was performed using C++ on a DW built on Oracle 9i platform on a Pentium IV 1.5 GHz microcomputer (with a 256 MB memory). The architecture of our software is described in Figure 8.12. Oracle DBMS was chosen because it is equipped with enough features and technologies by which data warehouse can scale to very large data volumes for analysis.

Table	Number of tuples	Width of tuples
Actvars	24786000	74
Chanlevel	10	24
Custlevel	1000	24
Prodlevel	10000	72
Timelevel	24	36

Table 8.3: Sizes of tables

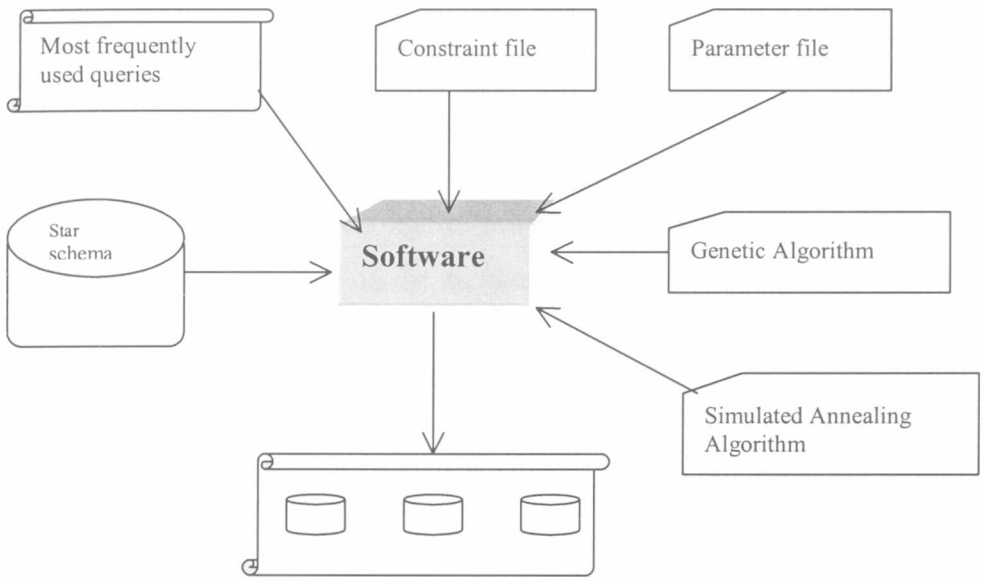

Figure 8.12: Architecture of SAGA

8.6 Experimental Setup and Configuration of GA and SA Parameters

15 queries have been considered. Each query has selection predicates, where each one has its selectivity factor. The page size (*PS*) is 8192 bytes and 9 fragmentation attributes were used. The number of sub domains generated by these attributes is 40. An exhaustive algorithm should generate 2^{40} fragmentation schemas to get the optimal solution. Due to this large number an exhaustive search was not considered. After conducting our experiments the identified good parameters of the GA were: (1) number of generations, (2) number of chromosomes, (3) crossover rate and (4) mutation rate. Using these parameters, GA was run and then the SA

The good parameters obtained by the first experimentation were: 500 generations (40 chromosomes per generation), crossover and mutation rates were 70% and 30%, respectively, in the beginning. After several generations, the mutation rate of 6% was used to ovoid a redundant search.

8.6.1 Experimentation of GA

In Figure 8.13, generation number was varied from 1 to 10000, and for each generation, the cost of evaluating the set of 15 queries was computed. A conclusion reached was that more than 500 generations are enough to produce good results. Note that the number of final fragments does not depend on the number of generations (see Figure 8.14).

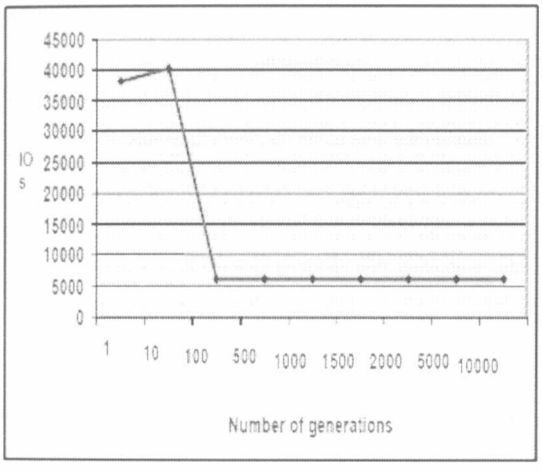

Figure 8.13: Number of generations vs. query processing cost

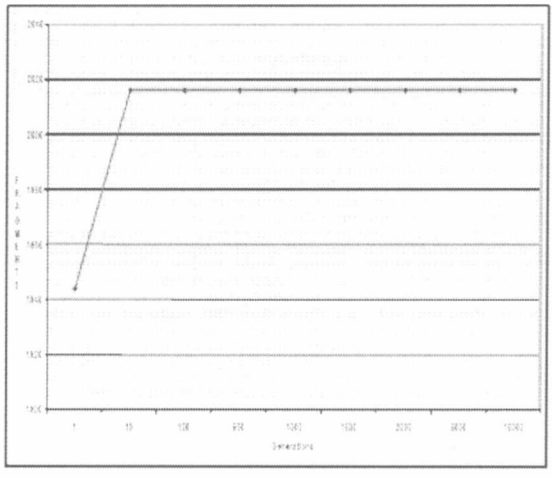

Figure. 8.14: Number of final fragments vs. generation

In Figure 8.15, the variation of the number of chromosomes in each generation was studied. The result shows the importance of this parameter to get a better performance of the GA. A reduced number of chromosomes do not allow the exploration of a large space of search and thus do not improve the quality of the final solution because it increases the number of identical chromosomes. Therefore we varied this number from 1 to 400. We concluded that 40 chromosomes can be enough to obtain good results.

In Figure 8.16, we study the effect of varying the crossover rate and see its impact on performance. Our results confirm the theoretical studies which states that a small crossover rate does not allow for an improvement of the solution. In our experimentation, we vary this rate from 0 to 100. Starting from 60 per cent, the performance becomes better.

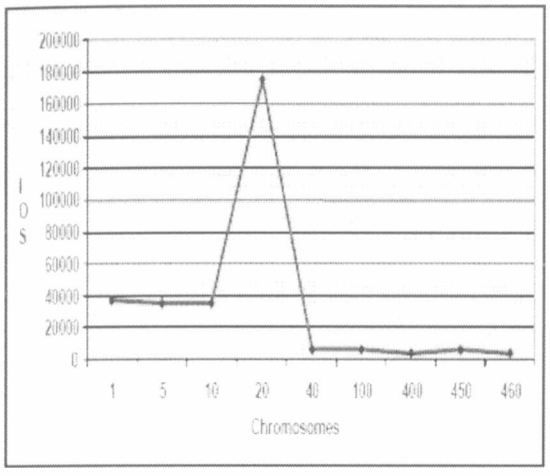

Fig. 8.15: Number of chromosomes per generation

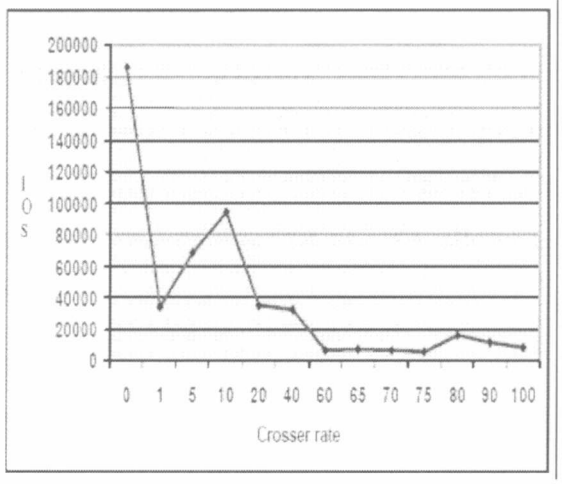

Fig. 8.16: The impact of crossover rate on Performance

Similarity, experimentation to see the impact of mutation rate has been conducted. The results show that a mutation rate with 30% gives a good performance (Figures 8.17, 8.18).

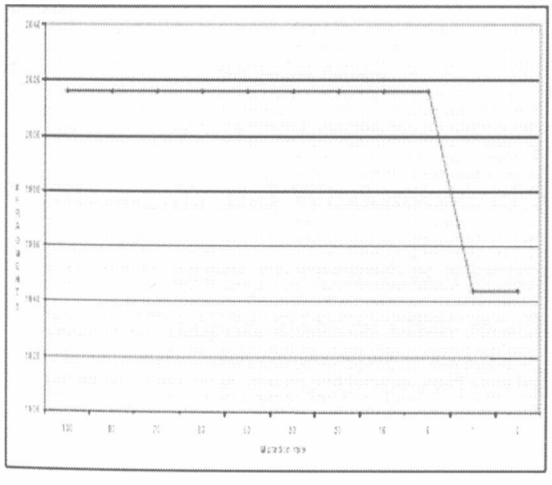

Fig.8.17: The impact of mutation rate on fragment number

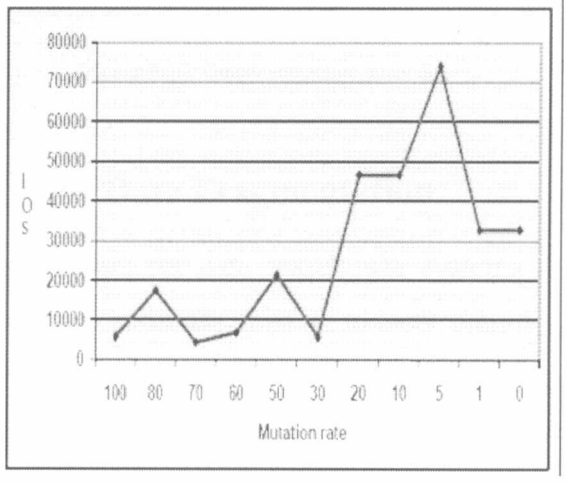

Fig. 8.18: The impact of mutation rate on IOs

8.6.2 Experimentation of the SA

Parameters used in our SA are: initial temperature was 400 which is decremented by 2 every 100 iterations, threshold was 2000, and the number of generations with improvement was fixed at 21000. The effect of GA and SA on different queries has been studied. The reduction was significant when using the SA.

It has been noticed that among the initial set of queries (15 queries), some queries do not get benefit from the application of SA. This is because, their selection predicates do not match the fragmentation predicates generated by SA (see Figure 8.19). Finally, Figure 8.20 shows the impact of SA on global query processing cost reduction. It was reduced by 44% after applying the SA. It can be concluded that combining GA and SA allows us to get good results (SA is complementary to GA).

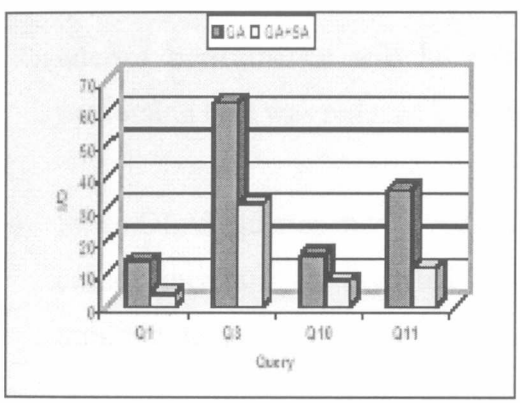

Figure 8.19: Profitable queries

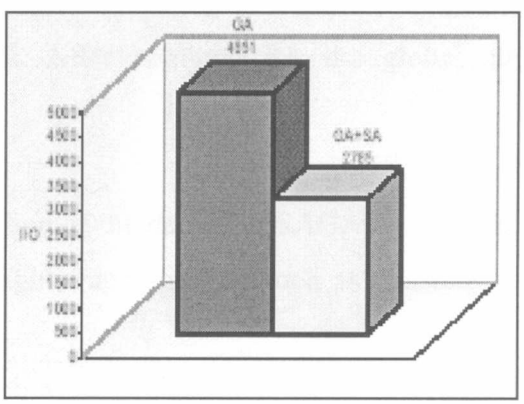

Figure 8.20: Query reduction after SA

8.7 Strengths and Weaknesses of SAGA for Data Fragmentation

Efficient query processing is a critical requirement for data warehouse systems as decision support systems often require minimum response time to answer complex queries having aggregations and multi-joins over vast repositories. This objective can be achieved by fragmenting warehouse data. In this chapter we introduced a new technique for horizontal partitioning a data warehouse star schema. The SAGA algorithm for horizontal partitioning presented in this chapter is characterized by many strengths and little weaknesses that can be summarized in the following:

Strengths:

1. Reducing irrelevant data accesses and queries execution time. The star schema partitions generated by the algorithm facilitate parallel execution of queries.

2. The choice of the best dimension tables, performed by the algorithm, for fragmenting the fact table plays an important role on the overall performance. And the number of partitioned dimension tables had a great impact on reducing query processing cost.

3. The algorithm gives good results when the concern is about the overall data warehouse performance, this is because even if the performance of some few queries will be degraded by the horizontal partitioning process, the majority of queries performance will be enhanced. After applying SA the global query processing cost was reduced by 44%.

4. Since OLAP queries mostly require portion of the data then SAGA fragmentation causes queries to be executed on the right fragments as much as possible thus, minimizing query response time.

5. The produced fragments can be allocated to sites in such a way where each fragment can be a functional unit of allocation.

Weaknesses:

1. The algorithm may degrade the performance for the queries that have selection predicates that does not appear in the partitioning specifications or that doesn't have selection predicates at all. This is because to evaluate such queries the algorithm need to access all sub-star schemas and then perform union operation

8.8 Summary

Physical data warehouse design includes materialized views, advanced indexing schemes, and data partitioning. In this chapter the focus was on horizontal partitioning of relational warehouses. The problem was formalized as an optimization problem and its complexity was presented. It has been shown how the number of fact fragments generated by the proposed partitioning methodology can be very huge and thus would be difficult for the

data warehouse administrator to maintain all resulted fragments. To solve the data partitioning problem, a hybrid (SAGA) method combining genetic and simulated annealing algorithms was proposed.

Genetic algorithms are good for optimization problems with a large search space. Simulated annealing algorithms are used on solutions obtained by genetic algorithms in order to avoid the problem of premature convergence inherent to genetic algorithm by allowing uphill moves to solutions of worse fitness.

The fitness is calculated by cost model evaluating the cost of a set of frequently used queries on the partitioned relational warehouse schema. This model is also used to measure the quality of the final solution.

Our experimental studies went through three steps: firstly good parameters for the genetic algorithms were identified, secondly, experiments based on these parameters were run, finally, experiment by combining genetic algorithm and simulated annealing algorithm was conducted. The experimental results are encouraging and show the feasibility and applicability of the approach.

In the next chapter we will consider the fragment allocation problem which is a distribution design techniques that also aims at improving system performance.

Chapter 9

Using a Greedy-Based and SAGA Approaches for Solving Data Allocation Problem in a Distributed Environment

9.1 Introduction

Data allocation is a key performance factor for distributed database and data warehouse systems. A major cost in executing queries in a distributed environment is the data transfer cost of fragments to sites [Boukhalfa *et al* 08]. Therefore, a primary objective of a data allocation algorithm is to locate the fragments at different sites to minimize the total data transfer cost incurred in executing a set of queries while increasing performance.

Finding optimal solutions for data allocation in a distributed environment is a difficult problem to deal with. This is mainly because many design factors are considered in allocation for optimal data distribution design.

Assuming that the database is properly fragmented, the designer has to decide on the optimal allocation of the fragments to various sites on the network and determine which copy or copies of data to access, where to process and how to route the data.

Typically, users at each site or node have their own set of information requirements. Some of these involve data that is unique to users at a single node. Others require data that is shared among users at multiple nodes.

However, to satisfy a user request in a distributed environment, you need to determine where the needed data is located and a strategy that specifies which copy of the data to be accessed and where it will be processed should be identified. Furthermore, final proces and respond to the requesting node via an optimal route must be specified.

For a distributed warehouse system to function efficiently, the fragments of the database need to be located carefully at various sites across the relevant communications network. The problem of allocating these fragments to the most appropriate sites is a difficult one to solve, however, most available approaches rely on heuristic techniques usually based on mathematical programming and formulations.

The proposed greedy approach presents a mathematical modelling technique for the data allocation in a distributed warehouse environment that considers network communication, local processing, and data storage costs.

Similarly, the SAGA approach contributes in determining best possible allocation of a data fragment in a distributed environment based on the fragment access patterns and the cost of moving data fragments from one site to the other.

9.2 Related Work

A distributed database comprises a set of fragments of databases stored at multiple sites that work together and appear as a single database to the user. Each database server in the distributed database is controlled by its local database management system. The objective of data distribution is to meet the information needs of business organizations having different sites with one or more computer systems connected via some communications network.

In a distributed warehouse database system the allocation of data over different sites or nodes of the network is a critical aspect of database design effort. A poor distribution can lead to higher loads and hence higher costs in the nodes or in the communication network, so that the system cannot handle the required set of transactions efficiently.
Fragments allocation problem has been extensively studied in both static and dynamic environments. In a static environment where the access probabilities of nodes to the fragments never change, a static allocation has been proposed prior to the design of a database depending on some static data access patterns. However, in a dynamic

environment where these probabilities change over time, the static allocation solution would degrade the database performance. Initial studies on dynamic data allocation give a framework for data redistribution and demonstrate how to perform the redistribution process in a minimum possible time. In [Brunstroml *et al* 95] a dynamic data allocation algorithm for non-replicated database systems is proposed named optimal algorithm, but no modelling is done to analyze the algorithm. In [Ulus and Uysal 03] the threshold algorithm is proposed for dynamic data allocation algorithm which reallocates data with respect to changing data access patterns with special focus on load balancing issues.

Many authors have considered various aspects of the allocation problem, in a variety of contexts. For example, [Mei *et al* 03] incorporate security considerations into the fragment allocation process [Sto¨hr *et al* 00] consider allocation in the context of multidimensional databases [Lin and Veeravalli] present an algorithm for allocation and replication that adapts to the changing patterns of online requests [Agrawal *et al* 04] consider incorporating partitioning into an automatic design framework, and [Chin 02] considers incremental allocation and reallocation based on changes in workload.

[Zhuo *et al* 03] consider the related problem of distributing the documents of a Web site among the server nodes of a geographically distributed Web server. The problem of replica placement is considered in [Cook *et al* 02] for networks using a read-one-write-all policy, and in [Karlsson and Karamanolis 04] for wide-area systems, while [Buchholz and Buchholz 04] consider it in the context of content delivery networks.

Various approaches have already been adopted to solve the data allocation problem in distributed systems [Corcoran and Hale 94], [Chin 02], [Buchholz and Buchholz 04]. Some approaches are limited in their theoretical and implementation parts [Apers 88], [Huang and Chen 01]. Other approaches present exponential time of complexity and test their performance on specific types of network connectivity [Ahmed *et al* 02].
A major cost in executing queries in a distributed database system is the data transfer cost incurred in transferring fragments accessed by a query from different sites to the site where the query is initiated. The objective of a data allocation algorithm is to determine

the assignment of fragments at different sites so as to minimize the total data transfer cost incurred in executing a set of queries.

In this chapter a new approach for data allocation based on constrained quadratic program (CQP) and SAGA is going to be presented and analyzed.

9.3 The Allocation Problem:

Fragment allocation is a distribution design technique to improve the system performance by reducing the total query costs. The allocation problem involves finding the optimal distribution of fragments to sites. Optimality can be defined with respect to two measures, cost and performance. In the proposed work it is assumed that the fragments have been determined, and the focus will be on the problem of allocating them in such a way as to minimize the total cost resulting from transmissions generated by user queries.

The objective of the proposed fragment allocation method is to determine which fragments are used by each query being hosted at specific sites such that all queries are satisfied while minimizing the communication cost, processing time, and storage costs, and in the same time not violating storage capacity and processing time constraints.

To describe fragment allocation problem, let's assume that we have a distributed database which is composed of m sites S = {S_1, S_2, …., S_m), where $1 \leq i \leq m$ and each site S_i is characterized by memory, CPU and a DBMS and all sites are connected by a communication network and assigned a set of F fragments, where each fragment F_j, is characterized by its size z_j

$$F = \{z_1, z_2, z_3, \ldots, z_j, \ldots, z_n\}.$$

Each fragment is requested by at least one of the sites. The site requirements for each fragment are indicated by the query matrix,

$$Q = \begin{vmatrix} q_{1,1} & q_{1,2} & .... & q_{1,n} \\ q_{2,1} & q_{2,2} & .... & q_{2,n} \\ . & . & . & . \\ q_{m,1} & q_{m,2} & .... & q_{m,n} \end{vmatrix}$$

Where $q_{i,j}$ indicates the requirement for fragment j by site i.

A theoretical framework is provided for this problem within the context of relational model that deals with it as a space optimization problem to be solved using a greedy algorithm and then SAGA algorithm.

9.4 The Cost Function Model:

For any distributed warehouse database system to work well, the fragments have to be dispersed over the available sites in such a way as to minimize the total cost of query processing. Query processing cost consists of processing cost and transmission cost.

Thus, one major aspect to be considered in the data allocation problem is the unit data transfer cost among sites. Hence, our main objective will be to obtain a site that for a given number of fragments minimizes the transmission cost.

In a partitioned (non-replicated) database environment, Let us assume that n fragments need to be placed into m sites and we want to find the optimum placement for n_i fragments in each site, where t_{ij} is the transmission cost for fragment j to site i for the n fragments so that the capacity of any site is not exceeded, given that each site is being characterized by its storage capacity C_i and fragment limit FL. And each fragment is characterized by its size z_j and storage cost s_{ij} associated with maintaining fragment j at site i. Each fragment j is required by at least one site.

The volume of data transmitted due to a query depends on the query type. While it might be possible to handle some queries locally, others might require communication with and among sites other than the query originating site. The data transfer cost model represents

the unit data transfer cost from one site to another following the minimum cost path such that the site capacity and limit constraints are not violated.

9.5 The Problem Formulation:

The allocation problem, attempts to find an allocation schema that minimizes a combined cost function (the cost of querying F_j at site S_i, the cost of storing each F_j at site S_i and the cost of data communication) that has two components: query processing and storage. Subject to the storage capacity (C_i), fragments limit *(FL)* and the processing time (*PT*) constraints of queries.

The proposed model has the following form: The decision variable is q_{ij}.

$$q_{ij} = \begin{cases} 1 & \text{if fragment } j \text{ is requested by site } S_i \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Minimize} \quad \text{Total Cost } (TC) = \sum_{i=1}^{m} \sum_{j=1}^{n} q_{ij} \times t_{ij} + \sum_{i=1}^{m} \sum_{j=1}^{n} q_{ij} \times s_{ij} \tag{1}$$

Subject to:

$$\sum_{j=1}^{n} q_{ij} \times z_j \le C_i \quad \forall_i, 1 \le i \le m \tag{2}$$

$$\sum_{i=1}^{m} q_{ij} = 1 \quad \forall_i, 1 \le j \le n \tag{3}$$

$$\sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij} \le PT \tag{4}$$

$$\sum_{i=1}^{m} q_{ij} \le FL \quad \forall_i, 1 \le i \le n \tag{5}$$

$$s_{ij} \ge 0 \quad \forall, 1 \le i \le n \tag{6}$$

Equation (1) represents the overall cost (TC) of the allocation problem to be minimized. The capacity constraint (equations 2) specifies that no site should receive more than its capacity i.e. the total size of all fragments in site i should be less than or equal to the storage capacity C_i. Equation (3) states that each allocated fragment should be stored in only one site (no replication). The constraints represented by equation (4) represents the total transmission time for all fragments is considered as the processing time PT for the overall allocation process. The overall allocation time must not exceed PT, i.e. the transmission time must not be more than expected. Such constraint forces the fragments allocation to be faster than traditional approaches of [Chin 02], [Corcoran and Hale 94] and [Zhuo *et al* 03]. The constraint in equation (5) specifies that each site should not receive more than a given number of fragments, denoted by fragment limit (FL). And s_{ij} in equation (6) is the cost of maintaining fragment j at site i.

This new formulation is an improvement to that considered by [Menon 03]. This model wants to limit the fragments transmission throughout the data warehouse. Note that this approach is of the form of the constrained quadratic program (CQP), with the addition of a single cardinality constraint $s_{ij} \geq 0$. This observation motivates the use of efficient greedy heuristic methods based on CQP to be applied to the problem considered here.

While the proposed model appears to be a natural representation of the problem, several solution methods proposed in the literature for solving data allocation problems are based not on the quadratic model but instead on an equivalent linearization of the form being used by [Menon 03].

There are some heuristic approaches that have recently been reported in the literature for a non-restricted method of fragment allocation where multiple copies of the same data fragments are allocated over the sites without restriction [Hababeh *et al* 06]. These methods are not applicable to the model considered here where a constrained non-replicated approach is adopted.

A greedy approach to fit our problem formulation is presented. The greedy algorithm works in an iterative fashion. In the first iteration, all the M sites are investigated to find the least occupied site(s) for a total of N fragments.

Consider that fragment j was chosen for allocation. The algorithm recursively makes calculations based on the assumption that all the users in the system request for fragment j. Thus, the algorithm has to pick a site that yields the lowest cost of allocation for the fragment j. In the second iteration, based on the choice of the already allocated fragment j, the algorithm now would identify the next fragment for allocation, which, when added to the fragment already being picked, yields the lowest allocation cost that satisfies equation (1). Note that this assignment may or may not be for the same site i. The algorithm proceeds in its iteration until either one of the constraints are violated.

ALGORITHM: FRAGMENT ALLOCATION

Input: *S*et of fragments $\{F_1, F_2, \ldots, F_n\}$, *S*et of sites $\{S_1, S_2, \ldots, S_m\}$, *PT, FL.*

Output: Fragments allocation to sites

begin

while there are still fragments to be allocated **do**

for each F_j in the set of fragments F **do**

- Investigate all the M sites to find the least occupied site(s) for a total of N fragments
- Pick a site that yields the lowest cost
- Identify the next fragment for allocation
- Add to the fragment already being picked if it satisfies equation 1 and doesn't violate constraints 2,3,4,5
- Calculate total cost
- Recursively make calculations for fragment j

end-for

end-while

end-algorithm

9.6 Experiments

In order to evaluate our work, the model was tested on a benchmark where sensitivity analysis has been carried out by varying *PT* and *FL* parameters in a problem of 12 fragments that need to be allocated at 6 sites to see how sensitive the solution to the changes in these two parameters.

Our greedy-based quadratic modelling approach was used to solve a standard test problem. This problem is similar to some problems appearing in [Menon 03]. The fragments and sites specifications are given in Table 9.1 and Table 9.2 respectively.

Fragment	**f1**	**f2**	**f3**	**f4**	**f5**	**f6**	**f7**	**f8**	**f9**	**f10**	**f11**	**f12**
Size	25	28	30	45	18	87	16	28	17	58	17	11

Table 9.1: Fragments specifications

Site	**Capacity C_i**	**FL**
1	100	6
2	90	7
3	80	5
4	70	3
5	60	3
6	50	2

Table 9.2: Sites specifications

For each site, a capacity C_i and a fragment limit *FL* values are given, which correspond to the capacity and a fragment limit that a given site can handle. Also, for each fragment a specific size is allocated.

Table 9.3 summarizes the fragments allocations to each site. The first column of the table gives the site number. The following twelve columns give the allocated fragments for each site. For instance, for site 3, the allocated fragments are {f2, f5, f7 and f9}. The

solution found respects the capacity constraint and the fragment limit constraint (recall Table 9.1 and 9.2). The overall initial process time (*PT*) constraint used for this solution was 60s. The computational results reported for our CQP approach were obtained by running our heuristic on a 1.96 GHz PC.

Site	Fragments											
	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12
1	x		x	x								
2						x						
3		x			x		x		x			
4								x				x
5										x		
6											x	

Table 9.3: Fragments allocations

Since the results were based on a constant value of *PT*, then best values need to be found for *PT* and *FL* of our problem. By varying the initial values of *PT* and *FL* with a specific delta Δ_{PT} and Δ_{FL}, the best values that fit the current problem could be found.

As presented in table 9.4, the first column presents the percentage changes of the initial value of *PT*. The second column presents the percentage changes of the initial values of *FL* (recall table 9.2). The last column presents the change of the solution found for the problem. As shown, the best values which gave best solution were: Δ_{PT} = +30% and Δ_{FL} = +30%, which means, with approximately the same values of the *FL* provided in table 9.2 and with *PT* = 78s, a better solution could be reached.

It is also obvious from table 9.4 that, as both process time and fragment limit increases better solutions are obtained. This is similar to a relaxed problem, where the process time constraint and the fragment limit constraint are removed.

Δ_{PT}	Δ_{FL}	Quality
+10%	+10%	+12.5%
	-10%	-18%
+20%	+20%	+26%
	-20%	-47.3%
+30%	+30%	**+78.1%***
	-20%	-112.6%
-10%	+10%	+13.3%
	-10%	-12.8%
-20%	+20%	+0.3%
	-20%	-89.5%
-30%	+30%	-178.6%
	-20%	-144.2%

Table 9.4: Sensitivity analysis

9. 7 Using SAGA Approach for Data Allocation

SAGA algorithm is a combination of the two algorithms of simulated annealing (SA) and genetic algorithm (GA). In SAGA model for data allocation, the GA algorithm starts first by generating the initial population that will be used in the implementation process. It assigns a fitness value for each chromosome (solution) for further filtration of solutions. Then, the SA will be incorporated to allow for occasional uphill moves to solutions with lower fitness by using a temperature parameter to control the acceptance of the moves

9.7. 1 Genetic Algorithm for Data Allocation

In computing terms, genetic algorithms map strings of numbers to each potential solution. Each solution becomes an individual in the population, and each string becomes a representation of an individual. There should be a way to derive each individual from its string representation. The genetic algorithm then manipulates the most promising strings in its search for an improved solution. The algorithm operates through a simple cycle:

1. Creation of a population of strings.
2. Evaluation of each string.

3. Selection of the best strings.

4. Genetic manipulation to create a new population of strings.

Figure 9.1 shows how these four stages interconnect. Each cycle produces a new generation of possible solutions (individuals) for a given problem. At the first stage, a population of possible solutions is created as a start point. Each individual in this population is encoded into a string (chromosome) to be manipulated by the genetic operators. In the next stage, the individuals are evaluated to determine how fit this individual is in relation to the others in the population. Based on each individual's fitness, a selection mechanism chooses the best pairs for the genetic manipulation process. The selection policy is responsible to assure the survival of the fittest individuals.

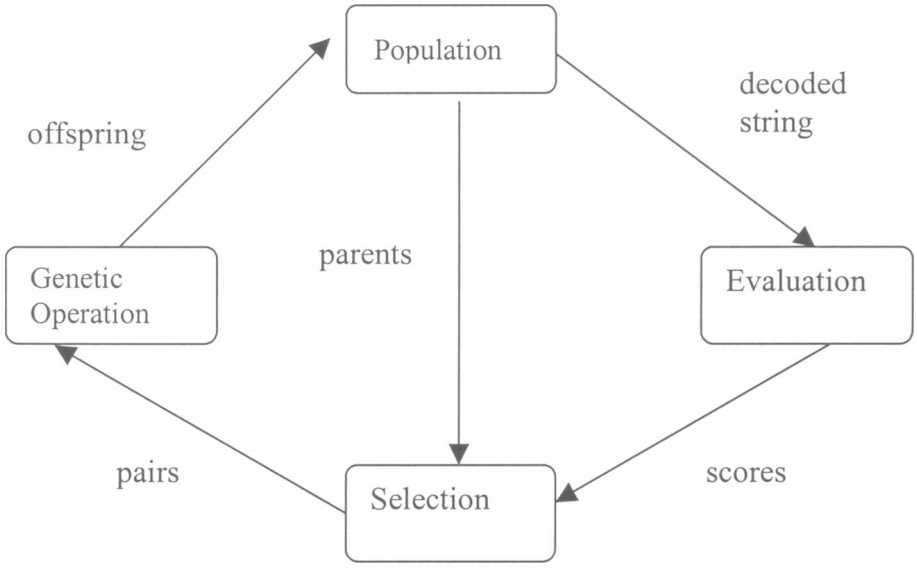

Figure 9.1: Reproduction Cycle Taken from [Basseda and Tasharofi 05]

Below is a brief description of how Genetic Algorithm for data allocation works [Basseda and Tasharofi 05]:

(1) Initialize population. Each individual of the population is a concatenation of the binary representations of the initial random allocation of each data fragment.

(2) Evaluate population.

(3) No of generation = 0

(4) WHILE no of generation < MAX GENERATION DO

(5) Select individuals for next population.

(6) Perform crossover and mutation for the selected individuals.

(7) Evaluate population.

(8) No of generation ++;

(9) ENDWHILE

(10) Determine final allocation by selecting the fittest individual. If the final allocation is not feasible, then consider each over-allocated site to migrate the data fragments to other sites so that the increase in cost is the minimum.

9.7.2 Simulated Annealing for Data Allocation

Simulated Annealing is a kind of single point based search strategy that is an iterative improvement scheme with hill-climbing ability, which allows it to reject inferior local solutions and find more globally near-optimal solutions. Similarly, it does not guarantee to find the global functional optima as well. But if the function optimization problem has many good near-optimal solutions, SA should find one near optimal solution.

The generic problem-space simulated evolution is as below [Basseda and Tasharofi 05]:

(1) Construct the first chromosome based on the problem data and perturb this chromosome to generate an initial population;

(2) Use the mapping heuristic to generate a solution for each chromosome;

(3) Evaluate the solutions obtained;

(4) No of generations = 0;

(5) WHILE no of generations < MAX GENERATION DO

(6) Select chromosomes for next population;

(7) Perform crossover and mutation for these set of chromosomes;

(8) Use the mapping heuristic to generate a solution for each chromosome;

(9) Evaluate the solutions obtained;

(10) No of generations = no of generations+1;

(11) ENDWHILE

(12) Output the best solution found so far;

9.8 SAGA Algorithm for Data Allocation

In the proposed SAGA algorithm, GA starts by generating 100 solutions (chromosomes) randomly as the initial population. This population is then used to produce the next generation using the GA operations of selection, crossover and mutation. The newly generated population will contain 100 offspring (new solutions). The process of producing new generations will be repeated 50 times. We will refer to the process of generating 100 initial chromosomes to reproduce 50 generations as Test-1. This operation of Test-1 will be run 100 times to obtain $100 \times 50 \times 100$ chromosomes, accumulating to 500,000 solutions, and each solution has a cost of allocation (fitness value) calculated according to Table 9.6.

SA role in SAGA starts at the parents' selection step of GA. Mainly, SA forces GA to select the parents from a wider space of population by accepting low fitness chromosomes (bad solutions) with the hope to improve solutions in future generations.

We implemented the entire operation of producing 500,000 solutions 4 times, every time using different method:

- First, using GA with random single-point crossover (GA).
- Second, using GA + SA by increasing the temperature from 0 to 100 (SAGA 0-100).
- Third, using GA + SA by decreasing the temperature from 100 to 0 (SAGA 100-0).
- Fourth, using GA + SA by fixing the temperature at 100 (SAGA 100).

Note: The first implementation (GA) is equivalent to SAGA when fixing the temperature at zero. And the fourth implementation (SAGA 100) is equivalent to GA when we select the pair of parents from the entire population (zero fitness).

To obtain accurate and comparable results for the four implementations we used the same input (the initial population) in the four methods. Therefore, we created a repository containing 100 compartments and in each one we randomly generated and saved 100 initial chromosomes. Then we forced each of the four implementations to get its initial population from the generated repository instead of creating it randomly (step 1 of GA described in section 9.7.1).

For example, in SAGA 0-100 method, to understand the effect of SA on how the GA creates a set of parents, let's assume that we want to produce 25 generations in the whole test and we have 100 chromosomes in the initial population from which GA selected the fittest 25% (25 chromosomes) in the generation number 1. Thus, the remaining 75 chromosomes that represent bad solutions should be included gradually to the set of parents in the subsequent generations controlled by the temperature parameter of SA. The SA will gradually increase the temperature from 0 to 100 at an interval of 4 per generation. At each step of temperature increment the SA forces GA to accept more bad solutions (3 in our example) from the remaining 75%. So, in generation number 2 we will include 3% more chromosomes to the already selected 25% to have a total of 28%. This process will continue until the entire range (100%) of the chromosomes is included by the end of the 25th generation.

Similarly, if we reverse the selection of chromosomes from 100% to 25% then this will represent movement from high temperature to low temperature (100-0). At a high temperature, a large range is sampled, but as the temperature decreases so does the sampling range until the algorithm stops when a freezing condition is met.

9.9 SAGA Implementation Results

To test the SAGA algorithm we used the same data sample used by [Corcoran and Hale 94] as a benchmark and adopted their butterfly network topology (see Figure 9.2).

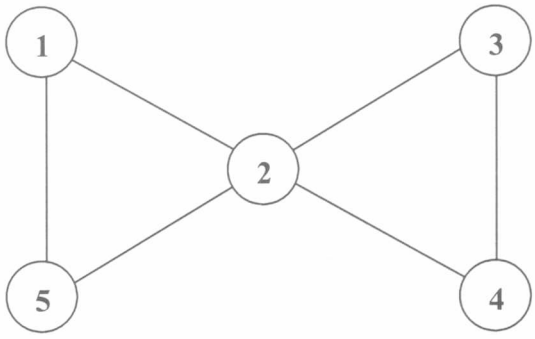

Figure 9.2: Butterfly Topology

The distributed database used by [Corcoran and Hale 94] was containing 15 fragments that need to be allocated over 5 sites, and it was assumed that each site requires specific fragments as presented in Table 9.5.

Site	Required Fragments
1	6, 9, 10, 12, 13, 14
2	7, 11
3	3, 4, 5, 6, 10, 12, 13, 14
4	2, 4, 5, 8, 9, 10, 11, 14
5	1, 2, 3, 6, 10, 15

Table 9.5: Required Fragments

In [Corcoran and Hale 94] there was a constraint that each site can host only up to 3 fragments. This constraint has been relaxed in our SAGA approach for data allocation to obtain more feasible solutions. The proposed SAGA allocation model considers only the communication cost and attempts to find an allocation schema that minimizes the total cost function of query processing.

$$\text{i.e. } TC = \sum_{i=1}^{m} \sum_{j=1}^{n} q_{ij} \times t_{ij} \quad \text{is minimum.}$$

Where $q_{i,j}$ indicates the requirement for fragment j by site i and t_{ij} is the transmission cost for fragment j to site i for the n fragments.

It is also assumed that the cost of data movement from one site to the other is only one unit (see Table 9.5). According to these assumptions the cost of allocating fragments to sites based on data movements is shown in Table 9.6.

S/F	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	3	3	4	4	3	1	2	2	5	3	2	2	4	1
2	1	2	2	2	2	3	0	1	2	4	1	2	2	3	1
3	2	3	2	1	1	4	1	1	3	5	2	2	2	3	2
4	2	2	3	1	1	5	1	0	2	5	1	3	3	3	2
5	0	2	2	4	4	3	1	2	3	5	3	3	3	5	0

Table 9.6: Cost of Fragment Allocation

To see how the communication cost presented in table 9.6 is calculated let's see for example how the allocation cost for fragment 9 is calculated.

If we allocate fragment 9 to site 1, given that fragment 9 was required by sites 1 and 4 (according to table 9.5). Then the total cost of fragment 9 allocation is composed of two costs:

$$\text{Cost}(9) = \text{Cost}(1, 9) + \text{Cost}(4, 9) = 0 + 2 = 2$$

As fragment 9 was already allocated to site 1, then the cost of requesting fragment 9 from site 1 to site 1 will be zero as it will be local request i.e. Cost(1, 9) = 0. However, the Cost(4, 9) = 2 because the request for fragment 9 from site 4 passes through site 2, i.e. it involves two movements, from site 1 to site 2 and then from site 2 to site 4.

Similarly, if we want to allocate fragment 10 to site 4 then the total cost will be:

$$\text{Cost}(10) = \text{Cost}(1, 10) + \text{Cost}(3, 10) + \text{Cost}(4, 10) + \text{Cost}(5, 10) = 2 + 1 + 0 + 2 = 5$$

Any chromosome (solution) consists of 15 genes (15 fragments) and each gene will take a value between 1 and 5 representing a site number (5 sites). And the total cost for a solution will be calculated according to Table 9.6. For example, the solution (543435241221135) that cost 23 is calculated in table 9.7.

Fragment	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Site	5	4	3	4	3	5	2	4	1	2	2	1	1	3	5
Cost	0	2	2	1	1	3	0	0	2	4	1	2	2	3	0

Table 9.7: Calculating the cost for a sample solution

After executing the aforementioned four implementations we obtained two million solutions (500,000 for each implementation). Table 9.8 illustrates the number of solutions for each cost resulted for the four implementations which is also represented graphically in Figure9.3.

Cost	GA	SAGA(0-100)	SAGA(100-0)	SAGA(100)
23	2	3	5	429
24	4	5	12	4227
25	9	11	22	16112
26	15	32	133	33622
27	38	41	1522	45268
28	1686	2104	3784	63096
29	2632	3154	4255	64666
30	14327	15757	17702	55075
31	29343	21259	25488	46955
32	30898	26034	45802	39372
33	39297	59258	51252	33640
34	68097	70634	65991	26983
35	67394	66529	66106	21918
36	64491	47294	53069	16548
37	64015	48025	43202	12066
38	43514	60717	46003	8132
39	32238	31425	34558	5551
40	22585	24403	25602	3223
41	12611	13054	12503	1690
42	3865	5876	2053	953
43	2835	2902	887	322
44	40	56	27	103
45	38	1396	15	38
46	26	31	7	11

Table 9.8: The number of solutions per cost

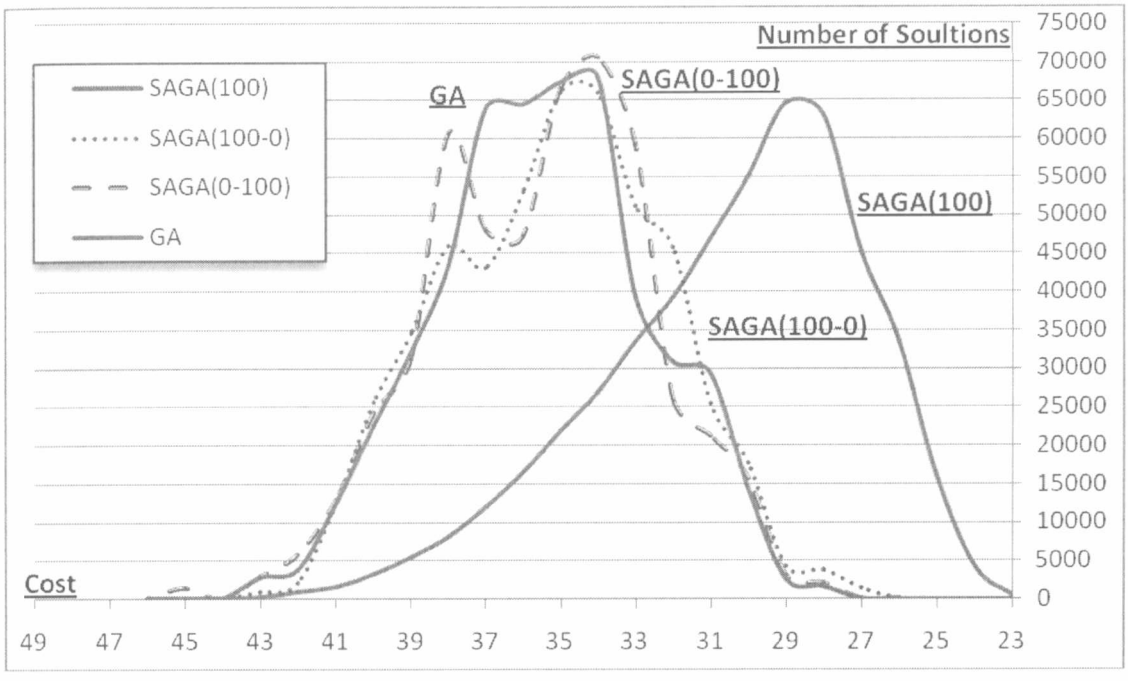

Figure 9.3: The number of solutions per cost

9.9.1 Comparing the Results of Different Methods of the Proposed SAGA Algorithm

At the beginning we started with GA and compared its results with that of SAGA 0-100. A little improvement was found when using SAGA 0-100 compared to GA. In the next phase we used SAGA 100-0 which outperformed both GA and SAGA 0-100. However, both of the used SAGA methods didn't significantly enhance GA results and were far below of our expectations. Therefore, we introduced the SAGA 100 method by fixing the temperature at 100 to enforce GA to accept bad solutions with the hope (which come to be true) to improve the obtained solutions as the reproduction process proceeds.

If we look at Table 9.8 and Figure 9.3, it can be noticed that the number of low cost solutions (like 23, 24 ...etc) are more when using SAGA (both SAGA 100-0 and SAGA 0-100) compared to GA. However, when comparing both SAGA results it is found that SAGA 100-0 is more efficient than SAGA 0-100. This mainly because SAGA 100-0

starts with the entire range of initial solutions and iteratively filters them in the subsequent generations based on the fitness value.

Furthermore, SAGA 100 produced much better results (low cost solutions) compared to all other methods, mainly because it used the entire range of initial solutions without filtration.

Additionally, when calculating the average cost for each generation (50 generations), it was found that in SAGA 100 the average cost was noticeably decreasing all through the implementation process. Table 9.9 and Figure 9.4 support the reached conclusions.

Generation	GA	SAGA(100-0)	SAGA(0-100)	SAGA(100)
1	35.194500	35.194500	35.194500	35.194500
2	35.467146	34.984095	35.466175	35.005905
3	35.335128	35.008849	35.461381	34.770667
4	35.453791	35.313231	35.349877	34.370095
5	35.498764	35.406731	35.398894	34.132857
6	35.405499	35.314650	35.450188	33.819524
7	35.353878	35.276196	35.465882	33.570476
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
24	35.626768	35.177943	35.557281	29.918286
25	35.595252	35.190128	35.576672	29.858667
26	35.651043	35.159493	35.608881	29.689048
27	35.657369	35.172636	35.591152	29.524571
28	35.686418	35.140526	35.599723	29.396571
29	35.630267	35.148423	35.602039	29.287429
30	35.601424	35.109528	35.640348	29.211714
---	---	---	---	---
---	---	---	---	---
---	---	---	---	---
44	35.342795	35.118235	35.431312	28.247238
45	35.332905	35.071217	35.475564	28.191810
46	35.380835	35.050242	35.422750	28.124381
47	35.247651	35.060435	35.459422	28.061714
48	35.269668	35.066640	35.449016	27.996952
49	35.292738	34.985655	35.462185	27.959810
50	35.266693	34.926487	35.483195	27.950571

Table 9.9: The average cost per generation

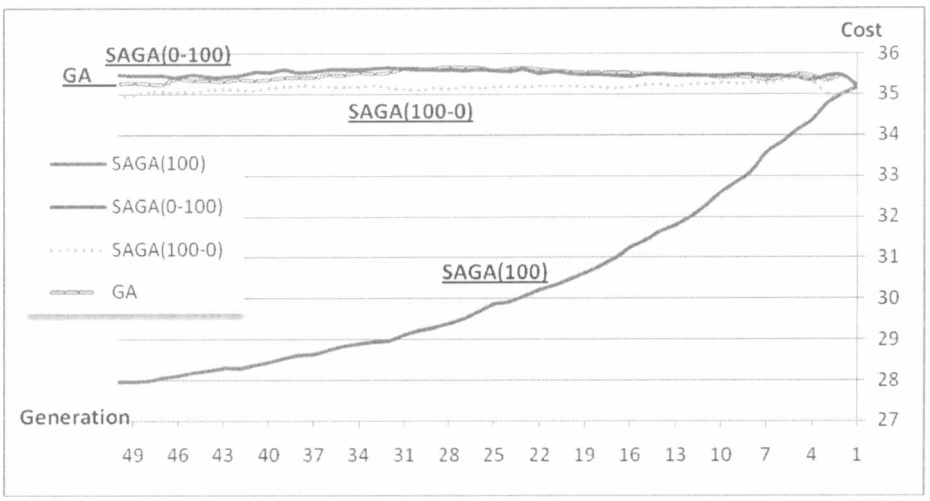

Figure 9.4: The average cost per generation

9.9.2 Proposed GA Results VS [Corcoran and Hale 94] GA Results

After comparing the different methods of the proposed SAGA algorithm among themselves, I now present a brief comparison of the proposed GA to that proposed by [Corcoran and Hale 94] with regard to the best result obtained for the allocation cost.

Both GA algorithms were tested using generational population model where GA saves offspring in a temporary location until the end of a generation where offspring replace the entire current population.

[Corcoran and Hale 94] used different combinations of crossover operators, however, this study is only interested in the result obtained using single point crossover as it is the crossover method being used in the proposed GA implementation.

The proposed GA implementations started with the same benchmark of [Corcoran and Hale 94] considering a problem with 5 sites and 15 fragments, where the fragment size is 1. A fragment matrix was generated in both GA implementations to represent the requirement of each fragment by the different sites (see Table 9.5).

The objective of both GA implementations was to place each fragment to the location that yields the least cost.

The table below summarizes the result of the best result obtained by each GA implementation using single point crossover (simple crossover).

	Corcoran (GA)	The proposed (GA)
Model	**Single Crossover** (Simple)	**Best Result Obtained** (Least cost)
Generational	26	23

Table 9. 10 The Proposed GA Results VS Corcoran and Hale's GA Results

Using single-point crossover, it is clear from Table 9.9 that the implementation of the proposed GA results has outperformed the ones being produced by Corcoran's GA, as the best allocation cost obtained by the proposed GA was 23 while Corcoran's best result obtained was 26.

9.10 Summary

This chapter contributes by allocating data fragments to their optimal location, in a distributed network, based on the access patterns for fragments. Two different approaches for data allocation were discussed in this chapter. First, a greedy approach that allocates fragments to nodes using a mathematical modelling method leading to the best possible solution for fragments distribution was presented. Second, a SAGA approach for optimal allocation of data fragments in a distributed environment was proposed, where the mechanism for achieving this optimality relied on knowing the cost involved in moving data fragments from one site to the other.

The computational results of implementing the greedy approach showed that for an improved solution, the fragment limit constraint should be relaxed and better values for the parameters PT and FL should be used.

In SAGA approach, different SAGA methods for data allocation were employed. However, the implementation confirmed that SAGA 100 outperformed all other SAGA and GA methods as it helped us to reach low cost solutions much faster.

When comparing the proposed GA results with the results obtained by [Corcoran and Hale 94], it was found that the implementation of the proposed GA has produced better results for the allocation cost compared to the one produced by Corcoran's for a single-point crossover

However, extending our work using simulated annealing with genetic algorithm (SAGA) has certainly improved the quality of solution for the data allocation problem.

Chapter 10

Conclusions and Future Work

10.1 Summary of contributions

The overall contribution of this thesis is in considering various algorithms for data fragmentation and fragment allocation that are useful for data warehouse efficiency. These contributions are related to the data warehouse performance improvement. A summary of the main contributions of this thesis is as follows:

- Provided a simple to understand, easy to implement and efficient algorithms compared to existing algorithms used for vertical partitioning problem. This is obtained by adding some factors that allow for more control on the final produced partitions based on the problem specifications.

- Proposed a static vertical partitioning algorithm for improving the performance of distributed systems using the number of occurrences of an attribute in a set of queries rather than the frequency of queries accessing the attributes. This enabled the database designer to perform partitioning and consequent distribution of fragments at early stages before the database enters operation.

- Proposed a methodology for the distributed data warehouse design using a horizontal fragmentation algorithm to partition the huge fact relation into a set of fragments based on the fragmentation schema of the dimension tables. Thus, relations those need not to be accessed are identified and unnecessary processing is avoided.

- Addressed and formalized horizontal fragmentation schema selection problem in relational data warehouse, using a genetic and simulated annealing algorithms in a star

schema to select the right solution that improves the performance of OLAP queries by avoiding unnecessary processing, and reduces the maintenance cost.

- Contributed in determining best possible allocation of a data fragment in a distributed database environment using a greedy mathematical modelling approach and SAGA algorithm that considers network communication cost to allocate site by site, leading to a better solution for the optimal fragments distribution problem.

10.2 Conclusions

Executing an OLAP query in a data warehouse can be very expensive if the data is not modelled properly. Moreover, if OLAP queries need only a portion of data, it is advisable to fragment data so that a set of queries can be executed on the right fragment as much as possible. Thus, leading to improvement in system performance by reducing query response time and maintenance cost. This is mainly because queries will scan fewer fragments than all, and in turn scans fewer rows than are stored in the original tables.

In this work I showed that data partitioning and allocation when properly and adequately performed, may significantly improve query response time for applications running on relational DWs.

Our work focused on understanding the difficulties involved in the fragmentation and allocation of relational warehouse databases, evaluating different strategies and techniques to handle the problem and proposing a unique methodology to obtain distributed relational data warehouse that provides optimal application performance.

This work has highlighted the importance of data modelling in a data warehousing environment and presented some analytical modeling techniques used in data warehousing. It has been emphasized that consolidating the data models of each business area before the real implementation can help assure that the result will be an effective data warehouse and can help reduce the cost of implementation.

The thesis reviewed and defined data warehouse physical design techniques and provided some guidelines to come up with proper structuring of data storage that guarantees good performance. It showed that the design of data warehouses is an optimization problem requiring solutions to several interrelated problems of data fragmentation and allocation that include: designing the conceptual schema of the integrated database, mapping the conceptual schema to storage areas and determining appropriate access methods.

A grouping algorithm for vertical partitioning problem was introduced where some added factors provided more control on the final produced groups based on the problem specifications. Also, an algorithm to handle the fragmentation problem during the design of distributed data warehouses is presented. The algorithm used the number of occurrences of an attribute in a set of queries rather than the frequency of queries accessing these attributes. This enabled a database designer to perform partitioning and consequent distribution of fragments before the database enters operation. Results of simulations were consistent with those obtained using frequency based partitioning algorithms. The significant advantage of the proposed algorithm is that a database designer doesn't have to wait for empirical data on query frequencies before partitioning a database.

In this work we have studied the problem of partitioning the data warehouse when data is modeled using star schema. The horizontal partitioning problem of relational warehouses was formalized as an optimization problem and its complexity was shown. This work illustrated how the number of fact fragments generated by the proposed partitioning methodology can be very huge and thus would be difficult for the data warehouse administrator to maintain all resulted fragments. To solve the data partitioning problem, a hybrid method combining genetic and simulated annealing algorithms was adopted.

Our proposed solution has been implemented using APB-1 benchmark. Experimental studies were conducted to identify good parameters for the genetic algorithms, then experiments were run based on these parameters combining genetic algorithm and

simulated annealing algorithm. The experimental results were encouraging and showed the feasibility of the obtained solutions.

Finally, this research presented an approach that allocates fragments to sites using a mathematical modelling technique and SAGA algorithm leading to optimal solutions for fragments distribution based on the cost of moving data fragments from one site to the other. When comparing the proposed GA results with the results obtained by [Corcoran and Hale 94], it was found that the implementation of the proposed GA has produced better results for the allocation cost compared to the one produced by Corcoran's for a single-point crossover

10.3 Future Work:

There are a number of promising future directions beyond the work presented in this research. Future works in this direction includes the adaptation of simulated annealing and genetic algorithm (SAGA) algorithm to handle the dynamic aspect of a warehouse due to the evolution of the schema and queries when query access information change.

There is no doubt that extending our work of using SAGA for data allocation has certainly improved the quality of the obtained solution. However, the work of using SAGA for data allocation need to be tested with a wider sample of data collection to achieve better results.

I also, believe that the proposed fragmentation and allocation algorithms need to be tested in a real life scenario where the interaction between the DW and organizations is established and the effect of proper DW partitioning and distribution in supporting the organization's strategic goals is presented.

Finally, I am planning to further extend the greedy modelling approach for data allocation to compare the implementation results obtained by the proposed greedy method with some other methods.

References

[Abdalla *et al* 07] H. Abdalla, E. Abuelyaman and F. Marir, "A Static Attribute-Based Partitioning Algorithm for Vertical Fragmentation in DDBs", Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'07), Volume II, pp 1017-1022, Las Vegas, June, 2007.

[Abdalla and Marir 06] H. Abdalla and F. Marir, "Vertical partitioning impact on performance and manageability of distributed database systems: A Comparative study of some vertical partitioning algorithms", Proceedings of the 18th NCC, Riyadh, Saudi Arabia, pp 85 - 92, March 26 - 29, 2006.

[Ahmad *et al* 02] I. Ahmad, K. Karlapalem, Y. Kwok, and K. So, Evolutionary Algorithms for Allocating Data in Distributed Database Systems, International Journal of Distributed and Parallel Databases, 11: 5-32, The Netherlands, 2002.

[Agrwal *et al* 97] A. Agrwal, A. Gupta, and S. Sarawagi. Modelling Multidimensional Databases. Technical report research. 1997.

[Agrwal *et al* 04] S. Agrawal, V. Narasayya, and B. Yang, "Integrating Vertical and Horizontal Partitioning into Automated Physical Database Design," Proc. 2004 ACM SIGMOD Int'l Conf. Management of Data, pp. 359-370, 2004.

[Apers 88] P. Apers, "Data allocation in distributed database systems," ACM Transactions on Database Systems, vol. 13, no. 3, 263–304, 1988.

[Babad 77] M. Babad. A record and file partitioning model. Commun. ACM 20, 1(Jan 1977).

[Baião 01] F. Baião "A Methodology and Algorithms for the Design of Distributed Databases using Theory Revision" D.Sc. Thesis, COPPE/UFRJ, Dec 2001. http://www.cos.ufrj.br/~baiao/thesis/baiaoDSc.pdf (Last accessed June 2005)

[B¨ack 95] B¨ack. Evolutionnary algorithms in theory and practice. Oxford University Press, New York, 1995.

[Ballard *et al* 98] C. Ballard, D. Herreman, D. Schau, R. Bell, E. Kim, and A. Valencic. Data Modeling Techniques for Data Warehousing. International Technical Support Organization, IBM, February 1998

[Basseda and Tasharofi 05] Basseda, R. and Tasharofi, S., Data Allocation in Distributed Database Systems, Technical Report No. DBRG.RB-ST.A50715, 2005.

[Basseda *et al* 06] Basseda, R., Tasharofi, S., Rahgozar, M., Near Neighborhood Allocation (NNA): A Novel Dynamic Data Allocation Algorithm in DDB, In proceedings of 11th Computer Society of Iran Computer Conference (CSICC2006), Tehran, 2006

[Bellatreche 08] L. Bellatreche: Horizontal Data Partitioning: Past, Present and Future, to appear in Encyclopedia of Database Technologies and Applications , 2008

[Bellatreche *et al* 06] L. Bellatreche, H. Abdalla and K. Boukhalfa, "A Combination of Genetic and Simulated Annealing Algorithms for Physical Data Warehouse Design", Proceedings of the 23rd British National Conference on Databases, Queen's University Belfast, Northern Ireland, 18-20 July, 2006.

[Bellatreche *et al* 06] L. Bellatreche, K. Boukhalfa, and H. Abdalla, "Algorithms for Physical Data Warehouse Design to Speed up Decision-making Processes", Proceedings of the 18th NCC, Riyadh, Saudi Arabia, pp 93 -110, March 26 - 29, 2006.

[Bellatreche *et al* 00] L. Bellatreche, K. Karlapalem, M. Mohania and and Michel Schneider,.: What can partitioning do for your data warehouse and data marts?, International Database Engineering and Applications Symposium(IDEAS'2000), 2000, pp. 437-446

[Bellatreche *et al* 05] L. Bellatreche and Boukhalfa K. An evolutionary approach to schema partitioning selection in a data warehouse environment. Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2005), pages 115–125, August 2005.

[Bellatreche *et al* 04] L. Bellatreche, M. Schneider, H. Lorinquer, and M. Mohania. Bringing Together partitioning, materialized views and indexes to optimize performance of relational data warehouses. Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2004), pages 15–25, September 2004.

[Bellatreche *et al* 02] L. Bellatreche, M. Schneider, M. Mohania, and B. K. Bhargava. Partjoin: An efficient storage and query execution for data warehouses. Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'2002), pages 296–306, September 2002.

[Benkrid *et al* 08] Soumia Benkrid, Ladjel Bellatreche and Habiba Drias, A Combined Selection of Fragmentation and Allocation Schemes in Parallel Data Warehouses, 4th International Workshop on Data Management in Global Data Repositories (GREP'08), edited by IEEE Computer Society Press, 2008

[Bennett *et al* 91] K. P. Bennett, M. C. Ferris, and Y. E. Ioannidis. A genetic algorithm for database query optimization. in Proceedings of the 4th International Conference on Genetic Algorithms, pages 400–407, July 1991.

[Boehnlein and Ende 99] M. Boehnlein and A. Ulbrich-vom Ende, "Deriving Initial Data Warehouses Structures from the Conceptual Data Models of the Underlying Operational Information Systems", 2nd International Workshop on Data Warehousing and OLAP (DOLAP), Kansas City, MO, USA, 1999.

[Boukhalfa *et al* 08] Kamel Boukhalfa, Ladjel Bellatreche and Pascal Richard, Primary and derived Fragmentation: study of Complexity, Algorithms of Selection and Validation under ORACLE10g, LISI, Research paper, No 01 2008, March, 2008

[Brunstroml 95] A. Brunstroml, S. Leutenegger and R. Simhal, Experimental Evaluation of Dynamic Data Allocation Strategies in a Distributed Database with changing Workloads, ACM Transactions on Database Systems, 1995.

[Buchholz 04] S. Buchholz and T. Buchholz, "Replica Placement in Adaptive Content Distribution Networks," Proc. 2004 ACM Symp. Applied Computing, pp. 1705-1710, 2004.

[Chaudhuri and Dayal 97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, 26(1):65–74, 1997.

[Ceri *et al* 82] S. Ceri, M. Negri and G. Pelagatti. Horizental Data Partitioning in Database Design. Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGPLAN Notices. pages 128–136, 1982.

[Ceri *et al* 83] S. Ceri, S. Navathe, and G. Weiderhold. Distributed Design of Logical Database Schemes. IEEE Transactions on Software engineering, 9(4), 1983

[Ceri and Pelagatti 84] S. Ceri and G. Pelagatti. Distributed Databases: Principles and Systems. McGraw-Hill International Editions, 1984.

[Ceri *et al* 89] S. Ceri, S. Pernici, and G. Weiderhold. Optimization Problems and Solution Methods in the Design of Data distribution. Information Sciences Vol 14, No. 3, p 261-272, 1989.

[Chaudhuri and Dayal 97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. SIGMOD Record, 26(1):65–74, 1997.

[Chaudhuri and Narasayya 98] S. Chaudhuri and V. Narasayya. Autoadmin 'what-if' index analysis utility. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 367–378, June 1998.

[Chen and Su 96] Y. Chen and S. Su, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object Oriented Databases", International Journal of Distributed and Parallel Databases, Kluwer Academic Publishers, vol. 4(2), 1996, pp. 107-142.

[Chin 01] Chin, A. G., Incremental Data Allocation and Re Allocation in Distributed Database Systems, Journal of Database Management, Jan-Mar 2001; 12, 1; ABI/INFORM Global pg. 35.

[Cook *et al* 02] S. Cook, J. Pachl, and I. Pressman, "The Optimal Location of Replicas in a Network Using a READ-ONE-WRITE-ALL Policy," Distributed Computing, vol. 15, no. 1, pp. 57-66, 2002.

[Corcoran and Hale 94] A. Corcoran and J. Hale, L. C., A Genetic Algorithm for Fragment Allocation in Distributed Database Systems, ACM, 1994.

[Cornell and Yu 87] D. Cornell, and P. Yu, A Vertical Partitioning Algorithm for Relational Databases. Proc. Third International Conference on Data Engineering, Feb. 1987.

[Datta *et al* 98] A. Datta, B. Moon and H. Thomas, A Case of Parallelism in Data Warehousing and OLAP, in the 9th International Workshop on Database and Expert Systems Applications (DEXA 98), pages 226- 231, August 1998.

[Date 95] J. C. Date. An Introduction to Database Systems, 6th edition. Addison-Wesley, 1995.

[Devlin 96] Barry Devlin, Data Warehouse: From Architecture to mplementation, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1996.

[Dodge and Gorman 00] G. Dodge and T. Gorman. Essential Oracle8i Data Warehousing. John Wiley and Sons, New York, 2000.

[Dowdy and Foster 82] L. Dowdy, and Foster, D. V., "Comparative Models of the File assignment Problem," ACM Press,1982.

[Eisner and Severance 76] M. Eisner, and D. Severance. Mathematical techniques for efficient record segmentation in large shared databases. J. ACM 23, 4 (Oct. 1976).

[EImasri and Navathe 99] R. EImasri and S. B. Navathe. Fundamentals of Database Systems, 3rd edition Benjamin/Cummings, 1999.

[Ezeife 00] C. Ezeife. Selecting and materializing horizontally partitioned warehouse views, Data & Knowledge Engineering 36 (2001) 185-210, 19 April 2000.

[Ezeife and Barker 95] C. Ezeife and K. Barker. Comprehensive approach to horizontal class fragmentation in a distributed object based system. lnternationat Journal of Distributed and Parallel Databases, 2, 1995.

[Firestone 97] J. Firestone, Data warehouse and data marts: A dynamic view white paper 3, Executive Information Systems, Inc, March 1997.

[Giorgini *et al* 08] P. Giorgini, S. Rizzi, M. Garzetti. GRAnD: A Goal-Oriented Approach to Requirement Analysis in Data Warehouses. Decision Support Systems, vol. 45, n. 1, pp. 4-21, 2008.

[Golfarelli and Rizzi 08] M. Golfarelli, S. Rizzi. Managing late measurements in data warehouses. In Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications, J. Wang (Ed.), Information Science Reference, pp. 738-754, 2008.

[Gupta and Mumick 95] A. Gupta and I. S. Mumick. Maintenance of materilized views: Problems techniques and applications. Data Engineering Bulletin 18(2):3-18, June 1995.

[Hammer and Niamir 79] M. Hammer, and B. Niamir. A heuristic approach to attribute partitioning. In Proceedings ACM SIGMOD Int. Conf. on Management of Data, (Boston, Mass., 1979), ACM, New York.

[Hoffer and Severance 75] J. Hoffer, and D. Severance. The Uses of Cluster Analysis in Physical Database Design In Proc. 1st International Conference on VLDB, Framingham, MA, 1975.

[Hababeh 06 *et al*] I. Hababeh, M. Ramachandran, N. Bowring, "A Mathematical Approach for Modeling Data Allocation in Distributed Database Systems" The Seventh Annual U.A.E. University Research Conference, 2006.

[Hababeh 05] I. Hababeh, A Method for Fragment Allocation Design in the Distributed Database Systems, The Sixth Annual U.A.E. University Research Conference, 2005.

[Huang and Chen 01] Huang, Y. F. and Chen, J. H., Fragment Allocation in Distributed Database Design, Journal of Information Science and Engineering 17, 2001, 491-506, 2001.

[Hoffer 76] J. Hoffer. An integer programming formulation of computer database design problems. Inf. Sci., July 1976, 29-48.

[Holland 75] J. H. Holland. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan, 1975.

[IEEE97] IEEE, Information Technology, http://standards.ieee.org/catalog/it.html (Last accessed May 2005)

[Informix 97] Informix Corporation. , Informix-online extended parallel server and Informix-universal server: A new generation of decision-support indexing for enterprise data warehouses. *White paper*, 1997.

[Info 97] Informix Inc. The INFORMIX-MetaCube Product Suite. http://www.informix.com/informix/products/new_plo/metabro/metabro2.htm, 1997. (Last accessed March 2008)

[Inmon 96] W. H. Inmon. Building the Data Warehouse. John Wiley & Sons, second edition, 1996.

[Inmon 00] W. H. Inmon. What is a data warehouse? Inmon Enterprises I.I.c., 2000. http://www.business.auc.dk/oekostyr/file/What is a Data Warehouse.pdf.

[Inmon 00] W. H. Inmon. What is a data marts? Inmon Enterprises I.I.c., 2000. http://www.billinmon.com/. (Last accessed May 2007)

[Ioannidis and Kang 90] Y. Ioannidis and Y. Kang. Randomized algorithms algorithms for optimizing large join queries. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 9–22, 1990.

[Jagadish *et al* 99] H. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. Snakes and Sandwiches: Optimal clustering strategies for a data warehouse. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 37–48, June 1999.

[Kalnis and Papadias 01] P. Kalnis and D. Papadias. Proxy-server architecture for olap. Proceedings of the ACM SIGMOD International Conference on Management of Data, 2001.

[Karlsson and Karamanolis 04]M. Karlsson and C. Karamanolis, "Choosing Replica Placement Heuristics for Wide-Area Systems," Proc. Int'l Conference Distributed Computing Systems (ICDCS), pp. 350-359, Mar. 2004.

[Kimball 96] R. Kimball. The Data Warehouse Toolkit. Wiley-QED, New York, 1996.

[Kimball *et al* 98] R. Kimball, R., Reeves, L., Thornthwaite, W., Ross, M., Thornwaite, W.: The Data Warehouse Lifecycle Toolkit. John Wiley & Sons, Inc.(1998).

[Kirkpatrick *et al* 83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, 220(4598):671–680, May 1983.

[Kittler 76] J. Kittler. A locally sensitive method for cluster analysis. Pattern Recognition 8, 22-33, 1976.

[Levy 00] A. Levy. Answering queries using views: A survey. Technical report, Computer Science Dept., Washington University, 2000.

[Levy 00] A. Levy, Logic-Based Techniques in Data Integration. In Minker, J., Logic Based Artificial Intelligence, Kluwer Publishers, 2000.

[Lin and B. Veeravalli 03] W. Lin and B. Veeravalli, "An Adaptive Object Allocation and Replication Algorithm in Distributed Databases," Proc. 23rd Int'l Conf. Distributed Computing Systems Workshops (ICDCSW '03), pp. 132-137, 2003.

[Loney and Koch 00] K. Loney and G. Koch. Oracle8i: The complete reference. Osborne McGraw Hill, Berkeley, 2000.

[Ma *et al* 06] H. Ma; Schewe, K.-D. Schewe and M. Kirchberg, "A heuristic approach to vertical fragmentation incorporating query information" 7th International Baltic Conference on Databases and Information Systems, 2006 3-6 July 2006 Page(s):69 – 76.

[Mazón *et al* 08] N. Mazón, J. Lechtenbörger, J. Trujillo: *Solving Summarizability Problems in Fact-Dimension Relationships for Multidimensional Models* Proc. 11th ACM International Workshop on Data Warehousing and OLAP (DOLAP), 2008.

[Mazón *et al* 07] N. Mazón, J. Lechtenbörger, J. Trujillo: *Reconciling requirement-driven data warehouses with data sources via multidimensional normal forms* (Data & Knowledge Engineering, 63 (3), 2007), 2007.

[Mei *et al* 03] A. Mei, L. Mancini, and S. Jajodia, "Secure Dynamic Fragment and Replica Allocation in Large-Scale Distributed File Systems," IEEE Trans. Parallel and Distributed Systems, vol. 14, no. 9, pp. 885-896, Sept. 2003.

[Menon 03] S. Menon, "Allocating Fragments in Distributed Databases" IEEE Trans. Parallel and Distributed Systems, vol. 164, no. 7, pp. 577-585, Sept. 2003.

[Metropolis *et al* 58] Metropolis, N., Rosenbluth, A.W., Rosenbluth, M. N., Teller, A.H. and Teller, E., Equations of State Calculations by Fast Computing Machines, J. Chem. Phys. 21, 1087- 1092, 1958.

[Mohania *et al* 00] M. Mohania, S. Samatani, J. F. Roddick and Y. Kambayashi. Advances and research directions in data warehousing technology. Austuralian Journal of Information Systems, 2000.

[Muthuraj 92] R. Muthuraj. A formal approach to the vertical partitioning problem in distributed database design. M.S. Thesis, Dept. of Computer Science, Univ. of Florida, Aug. 1992.

[Navathe *et al* 84] S. Navathe, S. Ceri, G. Weiderhold, and J. Dou. Vertical Partitioning Algorithms for Database Design ACM Transactions on Database Systems, Vol. 9, No. 4, 1984.

[Navathe and Ra 89] S. Navathe, and M. Ra. Vertical Partitioning for Database Design: A Graphical Algorithm. ACM SIGMOD, Portland, June 1989.

[Navathe *et al* 90] S. Navathe, K. Karlapalem, and M. Ra. A mixed Fragmentation Methodology for Initial Distributed database Design. In Technical Report. CIS Dept, Univ of Florida, Gainesville, FL, 1990.

[Niamir 78] B. Niamir. Attribute Partitioning in Self-Adaptive Relational Database System. Ph. D. Dissertation, M.I.T. Lab. for Computer Science, Jan. 1978.

[Noaman and Barker 99] A. Y. Noaman and K. Barker. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse. in the 8th International Conference on Information and Knowledge Management (CIKM'99), pages 154–161, November 1999.

[OLAP 97] OLAP Council. OLAP AND OLAP Server Definitions. 1997 Available at http://www.olapcouncil.org/research/glossaryly.htm

[Oracle Corp. 99] Oracle Corp. Oracle8i TM enterprise edition partitioning option. Technecal report, Oracle Corporation, Febeuary 1999.

[Özsu and Valduriez, 99] M. Özsu and P. Valduriez, Principles of Distributed Database Systems, 2nd edition (1st edition 1991), New Jersey, Prentice Hall, 1999.

[O'Neil and Quass 97] P. O'Neil and D. Quass. Improved query performance with variant indexes. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 38–49, May 1997.

[OLAP Council 98] OLAP Council. Apb-1 olap benchmark, release ii. http://www.olapcouncil.org/research/bmarkly.htm, 1998. (Last accessed April 2007)

[Sanjay *et al* 04] A. Sanjay, V. R. Narasayya, and B. Yang. Integrating vertical and Horizontal partitioning into automated physical database design. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 359–370, June 2004.

[Shin and Irani 91] D. Shin and K.B. Irani. Fragmentation Relations Horizontally using a Knowledge-Based Approach. IEEE Transactions on Software Engineering, 17(9), Sept 1991.

[Shoshani 97] A. Shoshani. OLAP and statistical databases: Similarities and differences. in Proc. ACM PODS, pages 185-196, 1997.

[Silberschatz *et al* 02] A. Silberschatz, H., Korth, and S., Sudarshan: Database Syatem Concepts. McGraw Hill, 4th edition 2002.

[St¨ohr *et al* 00] T. St¨ohr, H. M¨artens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. Proceedings of the International Conference on Very Large Databases, pages 273–284, 2000.

[Tsichritzis 78] D. Tsichritzis and A. Klug. The ANSt/X3/SPARC framework. AFIPS Press, Montval, N.J., 1978.

[Ramakrishnan and Gehrke 00] Ramakrishnan and Gehrke, Database Management Systems, 3rd edition, 2000.

[Rao *et al* 02] J. Rao, C. Zhang, G. Lohman, and N. Megiddo. Automating physical database design in a parallel database. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 558–569, June 2002.

[Richardo 04] C. Richardo: Databases Illuminated. Jones and Barlett, 1st edition, 2004.

[Rizzi 08] S. Rizzi. Conceptual modeling solutions for the data warehouse. In Data Warehousing and Mining: Concepts, Methodologies, Tools, and Applications, J. Wang (Ed.), Information Science Reference, pp. 208-227, 2008.

[Papadomanolakis and Ailamaki 04] S. Papadomanolakis and A. Ailamaki. Autopart: Automating schema design for large scientific databases using data partitioning. Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSDBM2004), pages 383–392, June 2004.

[Poe 96] V. Poe. Building A Data Warehouse for Decision Support. Prentice Hall, 1996.

[Stocker and Dearnley 73] M. Stocker and A. Dearnley. Self-organizing Data Management Systems Com-puter Journal. 16, May 1973.

[Ulus and Uysal 03] Ulus, T., and Uysal, M., Heuristic Approach to Dynamic Data Allocation in Distributed Database Systems, Pakistan Journal of Information and Technology, 2003, ISSN 1682-6027, 231-239.

[Wiese 05] Wiese, D.: Framework for Data Mart Design and Implementation in DB2 Performance Expert. Diploma thesis. University of Jena and IBM Böblingen. 2005.

[Widom 95] J. Widom. Research Problems in Data Warehousing. In Proceedings of CIKM, pages 25–30, 1995.

[Whitley 93] D. Whitley, a Genetic Algorithm Tutorial, technical report CS-93-103, November 1993.

[Yang and Karlapalem 97] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. Proceedings of the International Conference on Very Large Databases, pages 136–145, August 1997.

[Zaharioudakis *et al* 00] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering Complex Queries using Automatic Summary Tables. In Proceedings of SIGMOD, pages 105–116, 2000.

[Zhang and Yang 99] C. Zhang and J. Yang. Genetic algorithm for materialized view selection in data warehouse environments. Proceeding of the International Conference on Data Warehousing and Knowledge Discovery (DAWAK'99), pages 116–125, September 1999.

[Ziyati *et al* 07] Elhoussaine Ziyati, Kamel Boukhalfa and Ladjel Bellatreche, The contribution of the not redundant structures of optimization in the physical conception of the warehouses of data, in 8th International Symposium one Programming and Systems (ISPS ' on 2007), in May, on 2007.

[Zhuo 03] L. Zhuo, C. Wang, and F. Lau, "Document Replication and Distribution in Extensible Geographically Distributed Web Server," J. Parallel and Distributed Computing, vol. 63, no. 10, pp. 927-944, 2003.