DISTRIBUTED DATA WAREHOUSE ARCHITECTURE AND DESIGN

By

Amin Yousef Noaman

A Thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in
Computer Science

University of Manitoba
Winnipeg, Manitoba, Canada, 2000

0-612-51662-8

Distributed Data Warehouse Architecture and Design

BY

Amin Yousef Noaman

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree

of

Doctor of Philosophy

Abstract

Data warehousing is one of the major research topics of applied-side database investigators. Most of the work to date has focused on building large centralized systems that are integrated repositories founded on preexisting systems upon which all corporate-wide data are based. Unfortunately, this approach is very expensive and tends to ignore the advantages realized during the past decade in the area of distribution and support for data localization in a geographically dispersed corporate structure. This research investigates building distributed data warehouses with particular emphasis placed on distribution design for the data warehouse environment. This dissertation contributes by providing an architectural model for a distributed data warehouse, the formal definition of the relational data model for data warehouse, a methodology for distributed data warehouse design along with two approaches to horizontally fragment the huge fact relation in the data warehouse and an analytical technique that provides insight into which approach presented is most suited to a particular distributed data warehouse environment.

Acknowledgment

In the name of Allah, the Merciful, the Compassionate. Thanks are to Him that he sent us the prophets to guide us to the straight path.
To my mother and my father, who are always in my heart and mind, and whose my accomplishment in life is non but the result of Allah's answering to their prayers and supplication to Him for my success.
To my dear wife, whom whatever words of thanks I say, they would not be enough to do justice to her. She is the most understanding and patient person I know. To our children, AbdulRahman, Fisal, and Raghad, whom just remembering them, makes me know what I want to do in life.
To my supervisor, Dr. Ken Barker, I would like to thank him for his sincere encouragements and his diplomatic way of getting me back on the right track when I was digressing. I would like to thank Dr. Randal Peters for his useful discussion during the database group meetings. I would like to thank Lynne Romuld who gave me administrative support during my Ph.D. program at the department of computer science. My thanks also go to my close friend Mohsen Madi. We motivated each other during our Ph.D. program. I hope that we will work together in the future.

Contents

List of Figures

Chapter 1

Introduction

1.1 Overview

Decision Support Systems (DSSs) 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 data model that is integrated with existing warehouse data. The underlying data model for the data warehouse is the denormalized relational data model. Once information is available at the warehouse, queries can be answered and data analysis (DSS and EIS) can be performed quickly and effectively.

Most 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. However, this approach 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 force 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 is physically distributed across multiple data warehouses in multiple systems at different branches. Currently, the field of distributed data warehouse in terms of architecture and design is considered an important research problem that needs investigation.

With the broad objective of providing a distributed data warehouse, this research contributes in the following ways:

- Investigates the feasibility of placing the data warehouse on the distributed environment.

- Provides a "near optimal" distribution design for the denormalized relations in the data warehouse environment.

- Describes the advantages and disadvantages of using distributed technology for a data warehouse.
- Provides a formal definition of data model for data warehouse (star schema).
- Describes an architecture model for distributed data warehouse.
- Provides a methodology for distributed data warehouse design.
- Details two horizontal fragmentation algorithms that partition the denormalized relations into a set of fragments.
- Ensures that the semantic characteristics of the denormalized relations do not change as a result of applying the horizontal fragmentation algorithm.
- Provides insight into which algorithm presented is most suited to a particular distributed data warehouse environment.

1.2 Comparison Between OLTP and OLAP

On-Line Transaction Processing (OLTP) systems support day-to-day business operations of an organization by capturing data about basic business processes. *On-Line Analytical Processing* (OLAP) is a decision-support system. It provides the decision-makers of an organization with a multidimensional view of data for the purpose of business analysis. The following characteristics distinguishes the OLTP system from the OLAP system:

- *Type of data*: in an OLTP system, data are at the most detailed level and up-to-date data; in an OLAP system, data are historical, summarized and consolidated from several operational databases and over long periods of time.

- *Size*: in an OLTP system, the size of data are hundreds of megabytes to giga-bytes; in an OLAP system, the size of data are hundreds of giga-bytes to tera-bytes..

- *Typical access*: in an OLTP system, the typical access is standard transaction: insert, retrieve, update and delete individual (or a few) records; in an OLAP system, the typical access is complex and ad-hoc queries.

- *Data model*: in an OLTP system, the data model is a normalized relational data model; in an OLAP system, the data model is a denormalized relational model.

- *Users*: in an OLTP system, users are operational and administrative staff; in an OLAP system, users are managers and analysts.

1.3 Motivation

Given the benefits that can be achieved by distribution in a data warehouse, it is necessary to deploy this system in an effective way.

There are two approaches for distributed database design, the top-down approach and the bottom-up approach [ÖV91, CP84]. The key issue in a designing distributed data warehouse is to determine if the approach should be bottom-up or top-down. The essence of the argument supporting bottom-up design has to do with the need to integrate legacy systems into a "centralized" data warehouse. Thus, the multi-database approach to schema and data integration is required because we are really attempting to bring pre-existing data repositories into a single massive storage location. If this were true, a data warehouse would simply be a *cached* multidatabase system.

Although this design approach is ultimately needed at the time we bring data from the operational systems to the integrated data warehouse, it is not the proper way to "design" a distributed data warehouse. The better approach is to develop a subject-oriented, integrated, non-volatile, time-variant data repository that captures the OLAP application requirements of an organization. Once the core design is completed a decision must be made to either store data in a single centralized data repository or to distribute the data on a distributed database platform. This decision is beyond the scope of this research in that we assume the business environment and application will benefit from a distributed solution.

This thesis addresses the top-down design approach to distributed data warehouse design. There are two fundamental issues for the top-down design approach: fragmentation and allocation. Fragmentation partitions data into a set of data or fragments according to referencing frequency or locality. Allocation distributes these fragments across the network sites of the distributed system.

The problem of fragmentation and allocation has been addressed for distributed relational database system [NKR95, ÖV91, NR89, SW85, CP84, NCWD84, CN83, CNW83, CNP82, CNP82, HN79, HS75] and the distributed object based system [BKS97, EB95, BB95, ÖDV94]. Previous research work on fragmentation and allocation for distributed relational database system has been based on *normalized* relations (that are in at least third normal form).

In the data warehouse environment, the integrated data from different resources are modeled into a *denormalized* relations (that are in at least first normal form) to facilitate on-line data analysis (DSS and EIS) [MHP99, GMR98, CT97a, Kim96, CD97].

The existing distributed relational database and distributed object base design techniques for fragmentation will not work for distributed data warehouses. The reason is the underlying model. It has changed from normalized relational to a denormalized relational model. The additional complexity (fragmenting and allocating denormalized relation) introduced by the data warehouse environment has not yet been sufficiently studied.

There are three fragmentation strategies: horizontal, vertical, and mixed. Horizontal fragmentation is defined as the process of subdividing the tuples of the relation into groups of tuples. Each group has the same attributes as the original relation. Vertical fragmentation is defined as the process of subdividing the attributes of the relation into groups. Each group contains the primary key of the original relation. Mixed fragmentation is defined as the process of performing horizontal fragmentation followed by vertical fragmentation or vice versa. The emphasis of this research is on the horizontal fragmentation.

The following assumptions frame the research effort:

- The data warehouse data has been integrated from all data sources and stored in a data repository.

- The integrated data in the data repository is stored using a *Global Conceptual Schema* (GCS). The underlying model for the GCS is the denormalized relational model (Star Schema).

- The information requirements for distribution design have been pre-defined. These information requirements are: database information, application information, communication information, and computer system information.

The research problem addressed in this dissertation can be stated as: Given an integrated data that has been stored in a homogenous model (i.e., star schema) and a set of applications accessing these data at various sites, design a horizontal fragmentation scheme to partition the given data into fragments that fulfill the correctness rules of fragmentation. These rules are completeness, reconstruction and disjointedness [ÖV91]. Completeness ensures that all tuples from a relation are mapped into at least one fragment without any loss. Disjointedness ensures that the generated fragments are non-overlapping. Finally, reconstruction ensures that the fragments can be reconstructed to reproduce the original relation.

1.4 Organization of this Document

The rest of the document is organized as follows: Chapter 2 presents the background work for this research. It begins with an overview of the information management architecture and describes the variation of information warehouse. It also discusses the centralized approach for building data warehouse. Chapter 3 addresses the current system architectures for building a distributed data warehouse, reviews the traditional data modeling techniques and the previous research work on distributed relational database design. Chapter 4 discusses the advantages and disadvantages of distributed data warehouse and presents an architectural model for the distributed data warehouse. Chapter 5 provides the formal definition for the denormalized relational model for data warehouse and reviews the variation of data warehouse schema design. Chapter 6 addresses the proposed distributed data warehouse design methodology and presents two horizontal fragmentation algorithms. Chapter 7 provides an analytical technique to indicate to the distributed data warehouse designer which of the two algorithms presented in Chapter 6 is most suitable. Chapter 8 summarizes the contribution of this research and outlines future research directions.

Chapter 2

Background Work

This chapter presents the current architectural models for the information warehouse and discusses the advantages and disadvantages of each one. Section 2.1 describes a general information management architecture. Section 2.2 presents a taxonomy of the various types of information warehouse. Further, this chapter discusses, in Section 2.3, the advantages and disadvantages of building a data warehouse in a centralized environment and presents its various architecture models.

2.1 Information Management Architecture

Figure 2.1 illustrates the information flow through a typical information management architecture. The figure is essentially a three tiered architecture consisting of:

- The Integration Environment: This includes the legacy information systems and the processes required to integrate them.

- The Information Warehouse Environment: This is where the integrated, subject-based, and time-oriented data is stored using a homogeneous model.

- The Decision Support Environment: The decision support environment is responsible for interfacing with the information warehouse and providing the necessary tools to support interaction with it.

2.1.1 The Integration Environment

The integration environment consists of the *source databases* and the *integration and transformation tools*. Figure 2.1 depicts the multiple sources that comprise the Information Warehouse. These sources run the gamut from fully functioning *On-Line Transaction Processing (OLTP)* to various unstructured external data sources that could be flat-files or spreadsheets. The integration and transformation tools extract data and information about the data from the source databases, define the relationships among data at multiple sources, detect duplicates and inconsistencies, add any extra desired information such as time-stamps, transform the integrated data to the target database for the information warehouse, and transform the information about the data to the target database in the form of *metadata*.

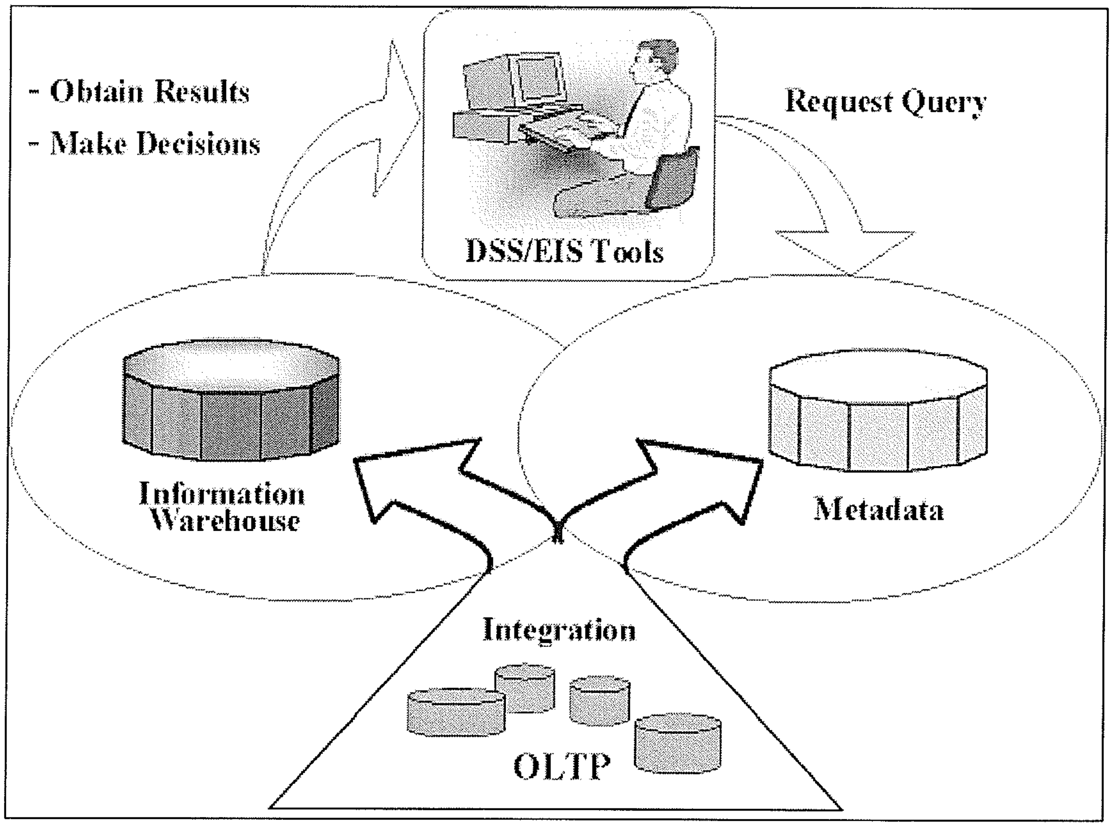


Figure 2.1: The Information Management Architecture.

2.1.2 The Information Warehouse Environment

The information warehouse environment consists of two significant parts namely the *Metadata* and the *Information Warehouse database.*

Metadata: The metadata describes the source data format and how the data is stored in the information warehouse. Tools to manage metadata must meet two types of user needs. First, database administrators must be able to manage data, locate its sources, and ensure that the information warehouse data is logically consistent with the source data. Second, end-users need to know what data exists in the information warehouse and, whenever possible, understand its meaning. Unfortunately, data semantics are a particularly difficult problem that is the subject of much ongoing research, but because this problem is so challenging, results will be slow to appear. Thus, current *ad hoc* approaches will need to suffice into the near future.

Information Warehouse: The information warehouse stores data so queries can be answered and large data volumes analyzed quickly and flexibly. Several tools are available to assist with the following tasks: controlling access to warehouse data, creating summarized data out of warehouse data, maintaining warehouse data, archiving outdated data to other media so that the warehouse does not become overloaded, and ensuring adequate protection against failure.

2.1.3 The Decision Support Environment

The decision support environment provides tools such as *On-Line Analytical Processing (OLAP)*, DSS, and EIS that can query and analyze warehouse data and deliver results to decision makers. OLAP tools perform at least two critical functions. The first allows the end user to form complex queries and do sophisticated analysis. The second function allows the end user to drill-down into lower levels of the detailed data when necessary. The DSS and the EIS tools use OLAP functions to visualize the analyzed data by creating aggregations, consolidating data, and performing trend analysis, statistical analysis, and optimization.

2.2 A Taxonomy of Information Warehouse Architectures

Identifying the organization's objectives and requirements from the information management environment is an important factor to consider in designing and choosing the right architecture for this environment. Two types of informational data may be needed by the organization [BE97, IH94]. These are:

- Data to support immediate decisions and
- Data to support long to medium-term decisions.

If the organization needs to make an immediate decision, the Virtual Data Warehouse [Dev97] or the Operational Data Store architecture will be a suitable approach to building the information management environment [IIB96, Inm95, IH94]. However, if it needs to make a long to medium-term decision and perform historical and trend analysis, then the Data Warehouse and/or Data Marts architectures will be used. Figure 2.2 shows a taxonomy of different types of information warehouses based on the organization's needs.

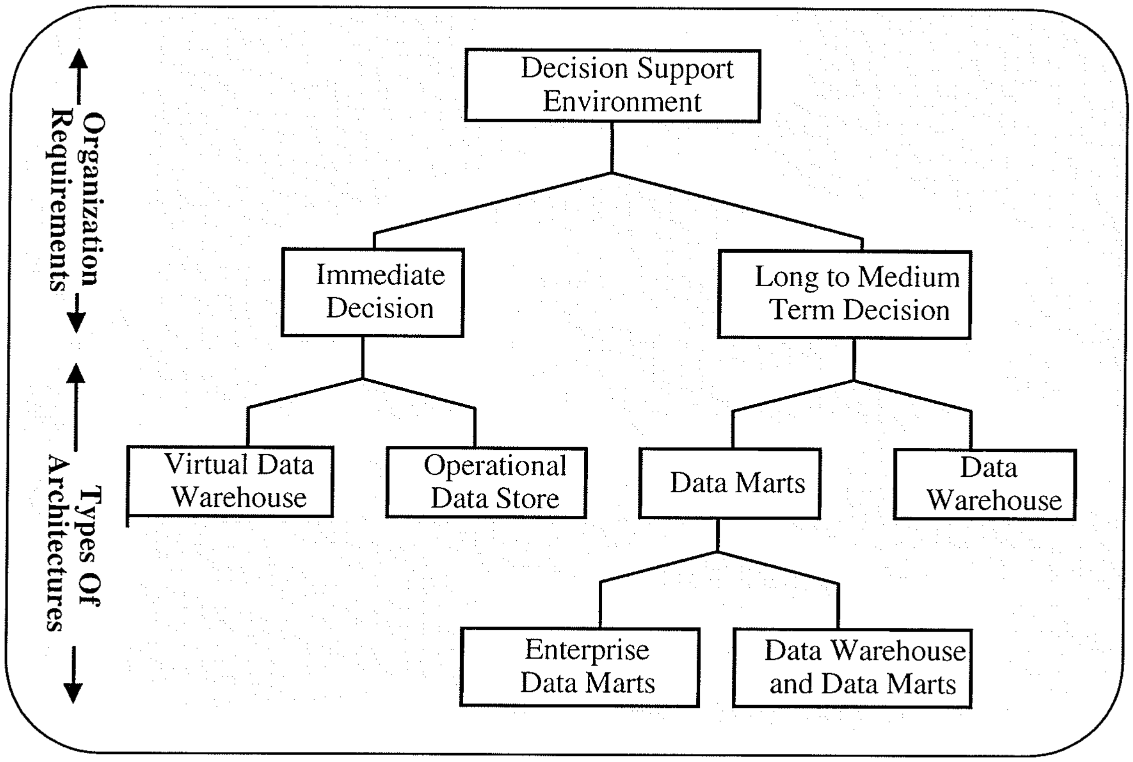


Figure 2.2: A Taxonomy of Information Warehouse Architectures.

Each information warehouse architecture has various characteristics depending on its use. Table 1 shows the types of information warehouse architectures and their characteristics. In general these characteristics are: *subject oriented, integrated, non-volatile, time variant, current value, summarized* and *detailed.*

The *subject oriented* represents a collectively integrated image of data that are of interest across the organization. For example, in a hospital, the subjects could be the patients, the medical staff, and the medications. The subjects for a manufacturer could be the customers, the products, and the vendors. The *integrated* data means having consistent naming conventions, consistent key structures, consistent physical attributes, and consistent measurement of variables. In different databases, the same element is often referred to by different names. For instance, the Customer database may name the customer balance "cubal" while the Accounts Receivable database may name it "balance". The integration ensures a consistent naming conventions such as "balance".

Characteristics	**Types of Information Warehouse**			
	Virtual DW	Oper. Data Store	Data Warehouse	Data Mart
Subject Oriented	√	√	√	√
Integrated	-	√	√	√
Non-Volatile	-	-	√	√
Time Variant	-	-	√	√
Current Value	√	√	-	-
Summarized	-	-	√	√
Detailed	√	√	√	√

Table 1 : The characteristics of information warehouse architectures.

Non-volatile means there is no update operation occurring in the warehouse data (i.e., it contains stable data) and it is available for direct access and analysis. *Time variant* means the warehouse data should include historical data (i.e., it contains elements of time such as day, month and year) to be used in analysis and evaluation of trends. *Current-valued* means the warehouse data contains current or near-current (up-to-date) data. *Summarized* data represents important information for the end-users. It is generated from the detailed data within the warehouse. Detailed data is the core of the warehouse without which the end-users could not have a complete picture of how a specific situation was created.

The following sections discuss the architectures that could be applied to the information management environment.

2.2.1 Virtual Data Warehouse

Virtual data warehouses provide end-users with direct access to multiple operational databases and files through middleware tools [Dev97]. The virtual data warehouse uses a network to allow end-users on terminals or client workstations direct access to operational systems (see Figure 2.3). The virtual data warehouse is a *subject oriented, current-valued* and *detailed-only* collection of data in support of an organization's need for up-to-the-second operational information.

The advantages of virtual data warehouses are:

- provides end-users with the most current corporate information
- no data redundancy and

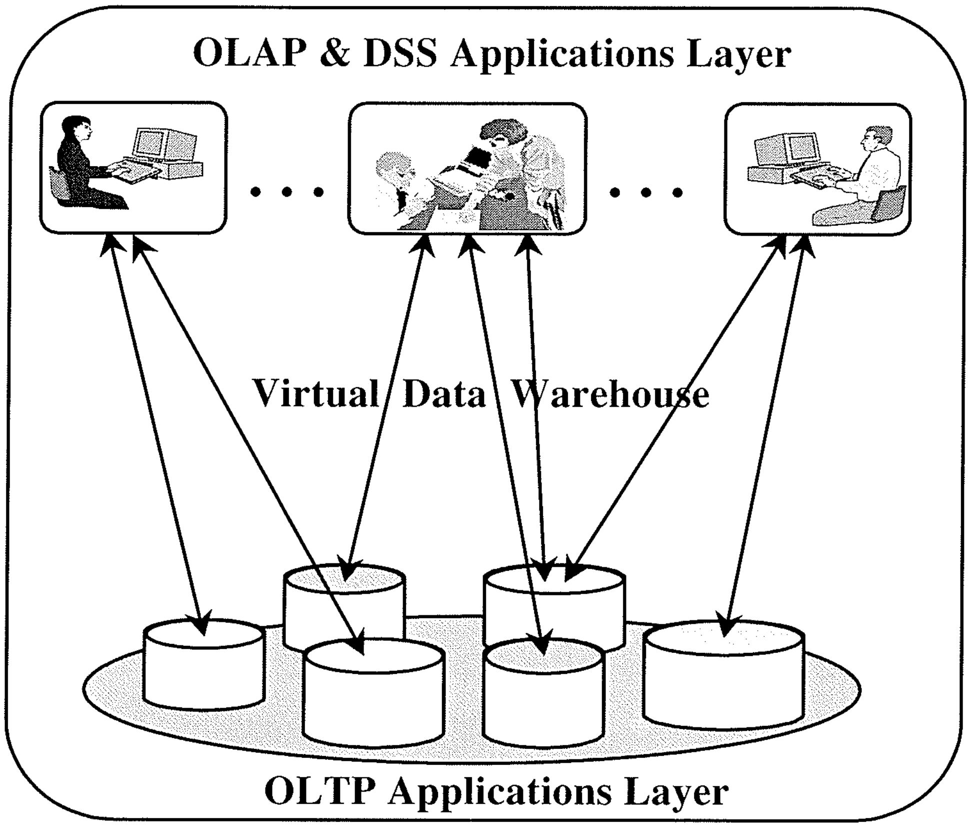


Figure 2.3: Virtual Data Warehouse Architecture.

- no need for additional hardware to handle the analysis processing.

Several disadvantages need to be considered:

- The data in a virtual data warehouse are accessed by using the multidatabase (interoperable) approach whereby queries are translated and shipped to the original source systems for execution. Any interoperable approach requires that the underlying source systems provide some form of locking to ensure serializability of submitted transactions, but this may prove to be extremely costly or nearly impossible (in fully autonomous systems).

- The data in a virtual data warehouse may be inconsistent or incomplete as it is derived directly from the operational databases without any kind of integration.
- End-users access times are unpredictable. They could request meaningless queries or scan all the data as they have a complete access to the entire detailed operational databases.
- The operational database is frequently not in a form the DSS end-users need because it is designed and tuned to support OLTP applications rather than OLAP.

2.2.2 Operational Data Store

An Operational Data Store (ODS) is the second solution. The ODS represents a collective, integrated view of the current operations of the organization. It is fed by the operational databases and serves the role of integrating data within the operational system. The operational data store is a *subject-oriented, integrated, volatile, current-valued*, and *detailed-only* collection of data in support of an organization's need for up-to-the-second, operational, integrated and collective information [IIB96, Inm95, IH94].

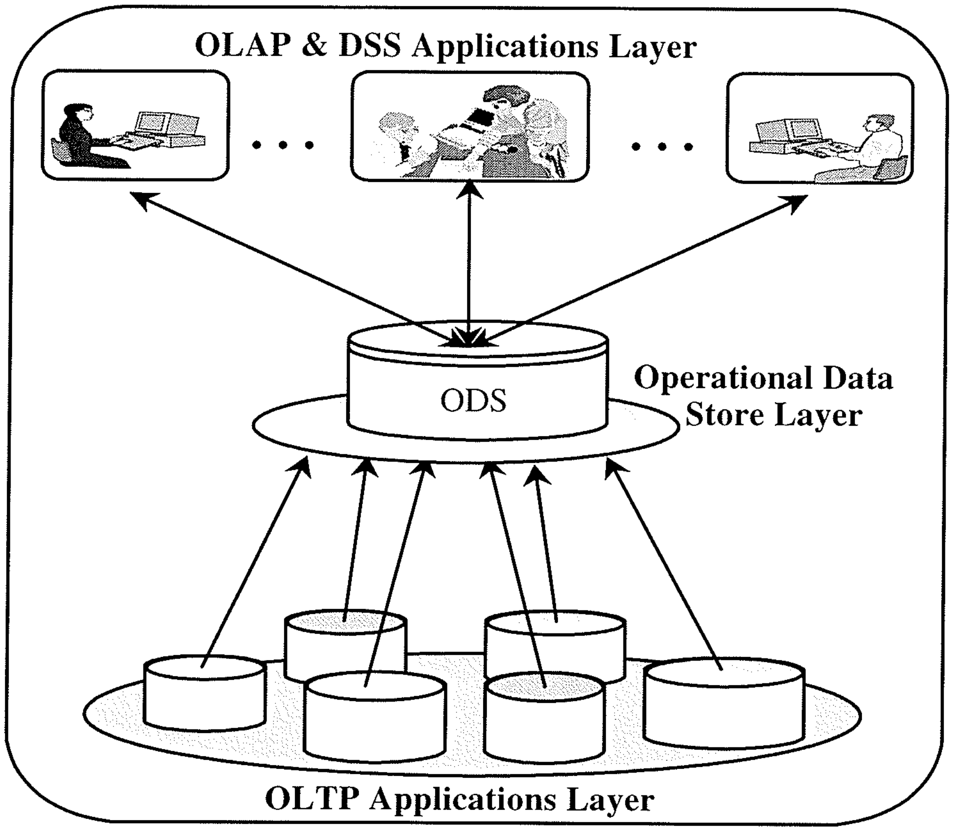


Figure 2.4: Operational Data Store Architecture.

Volatile means whenever the data in the operational databases changes, the data in the operational data store is updated. The operational data store contains only detailed data and does not store summarized data because the data is constantly changing values. However, the summarized data can be calculated.

The operational data store overcomes the drawbacks of the virtual data warehouse by transforming and integrating operational data into a consistent single repository (see Figure 2.4). This repository is directly available to answer queries and perform data analysis. Moreover, if the organization needs to build a data warehouse later on, the operational data store could be used to feed this data warehouse. The disadvantages of operational data stores are the high cost of hardware and software needed for their construction and slower retrieval for summarized data.

2.2.3 Data Warehouse

The data warehouse contains consolidated data extracted from multiple operational systems and/or merged with data from other external information systems (see Figure 2.5). It supports long to medium-term decisions.

The data warehouse is a *subject oriented, integrated, non-volatile, time variant, summarized*, and *detailed* collection of data supporting managerial's decision making for the entire organization [Inm96].

The advantages of the data warehouse architecture model are:

- It provides data that is common across the organization and of interest to the entire organization in one central location.
- It provides consistent data, so that decision makers are referencing the same data when they are making decisions.
- It protects the operational systems from complex queries that could slow down the performance of these systems.

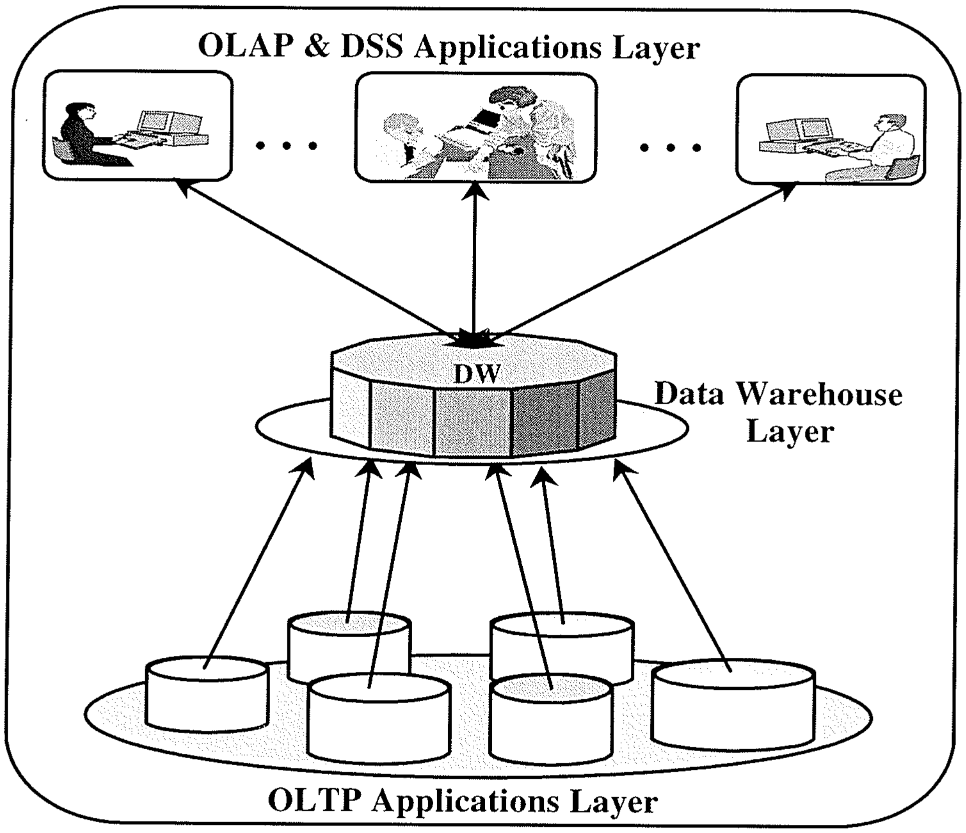


Figure 2.5: Data Warehouse Architecture.

The disadvantages of the model are:

- The data warehouse usually contains terabytes of data that increase with time.
- This huge amount of data requires expensive hardware and software to handle intensive analysis processing against it.
- As the numbers of the DSS end-users increase, the performance of the data warehouse will decrease because of the nature of the analysis queries that would

require access to the entire data warehouse data to answer the requested analysis query.

- The integration and transformation stage in building the data warehouse is consuming a lot of time and requires high levels of cooperation among the various divisions of the organization.

2.2.4 Data Marts

Data marts provide data that is of interest to a specific user or a group of users. For example, individual departments or divisions could create their own data mart that address their specific data requirements.

Data marts provide the DSS end-users with more control over their own departmental data in terms of requesting the required data, handling the data (summarizing, sorting, aggregating, etc.), and selecting the analytical software that is tailored to handle the data and fit their needs. Furthermore, data marts permit DSS processing to be done on local systems which improve both performance and availability.

Data marts are *subject oriented, integrated, non-volatile, time variant, summarized*, and *detailed* collection of data in support of divisional decision making.

The characteristics of data marts are similar to those of data warehouse but they are different in the following points:

- The data mart data is at a refined level while the data warehouse data is at very "detailed" level.

- The data marts focus on a few specific subjects in the organization while the data warehouse focuses on all subjects in the organization.

- Following from both points above, the data mart is smaller than the data warehouse. The small size of the data mart improves the performance of DSS processing, providing both fast responses to DSS end-user's requests and reduced storage cost.

There are two architectural models for building data marts. The first architecture is the *Enterprise Data Marts.* The second is the *Data Warehouse and Data Marts* architecture [Dem94, Whi95].

Enterprise Data Marts: The basic idea behind the enterprise data marts architecture is to create various data marts according to departmental and divisional needs. Each data mart will contain data that is captured, cleaned up and integrated from one or more operational systems and other external informational systems (see Figure 2.6).

The advantages of this architectural model are:

- Speed and ease associated with the integration and transformation processes.

- Existing hardware in the organization could handle the small size of the data marts.

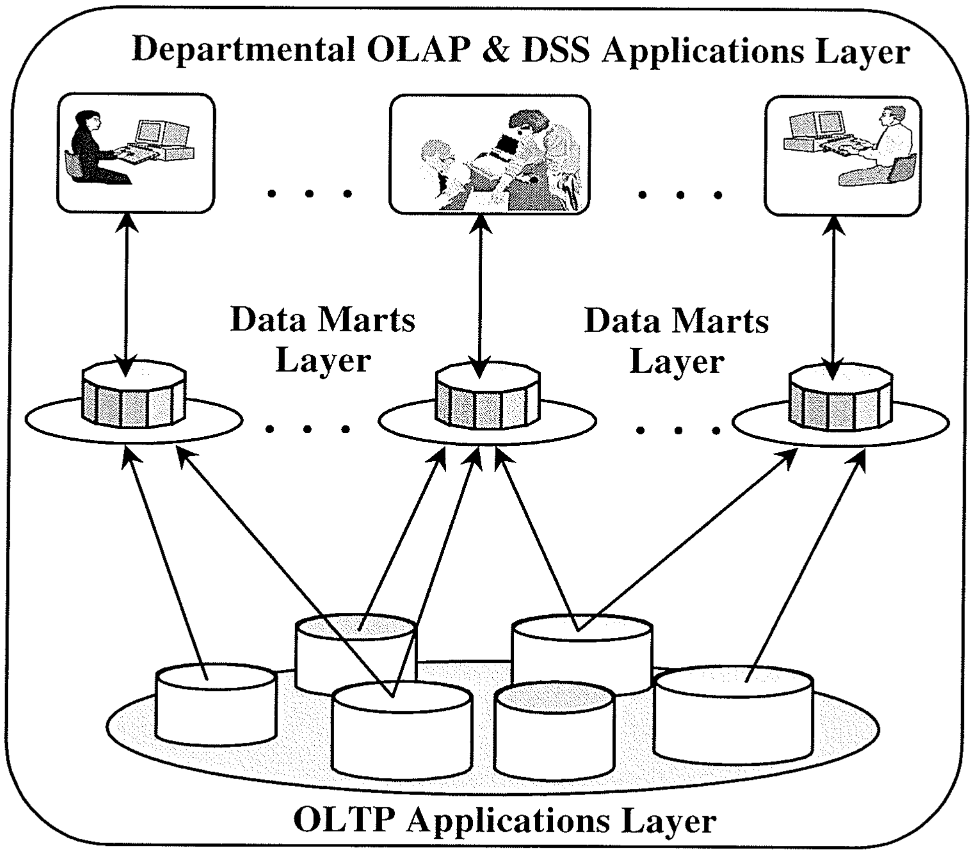


Figure 2.6: The Enterprise Data Marts Architecture.

- Low construction cost.

One disadvantage of this architecture is the enterprise data marts cannot provide a global view of the entire organization. For example, when there is a need for making a decision covering the entire organization and the various data marts are built in a non-standardized way (i.e., there is no consistency in the integration "no consistent naming convention, no consistent key structure, etc."), the organization

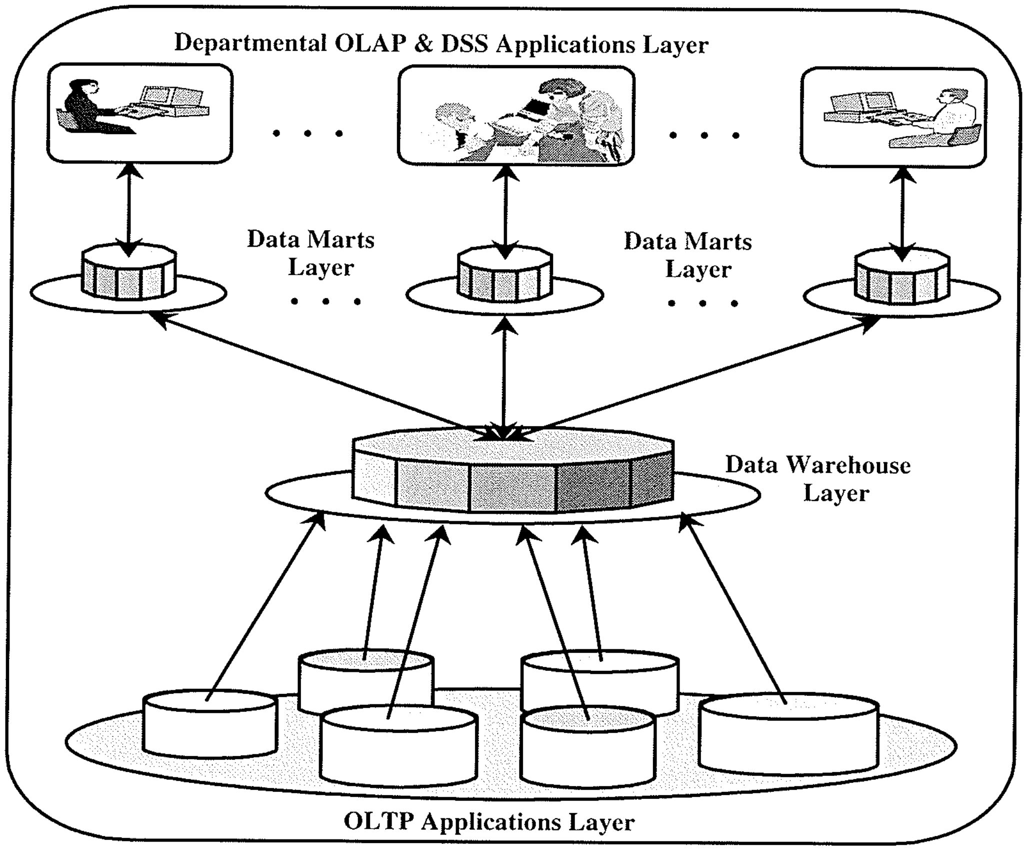


Figure 2.7: Data Warehouse and Data Marts Architecture.

will not be able to generate a global view of the data existing in the data marts. In order to support this need, another integration layer should exist to consolidate and standardize the data. Moreover, as the number of data marts grows, the complexity of managing them will increase as will the data redundancy.

Data Warehouse and Data Marts: This model is a combination of data warehouse and data marts. The data warehouse contains data that is integrated and cleaned up from operational systems and/or other external informational systems as discussed in Section 2.2.3. The data mart contains data that is extracted, transformed and loaded from the data warehouse according to the DSS end-user's requests. As the detailed data is loaded from the data warehouse into the data mart, it will be customized, selected and summarized (see Figure 2.7).

The data warehouse and data marts architecture has the following advantages:

- The data marts data does not require any kind of integration because it has already been integrated in the data warehouse.
- Reconciling and recovering the existing data marts will be affordable because the source for the data marts is the data warehouse.
- It is affordable to build a data mart that provides a global view for the entire organization because the data warehouse contains the complete detailed data of the organization.

This architecture, however, has some disadvantages similar to the enterprise data marts including the redundancy of data and the complexity of managing growing numbers of data marts.

2.3 Centralized Data Warehouse

Since the emphasis of this research is on the data warehouse, this section presents the various approaches for the centralized data warehouse. A centralized data warehouse is a single physical repository that contains integrated data extracted from multiple operational systems and merged with data from external information systems. Currently, most organizations build central data warehouses, even if their organizations are geographically distributed.

2.3.1 Advantages and Disadvantages of Centralized Data Warehouse

The major advantages of a centralized data warehouse are:

- Security: The centralized data warehouse offers a high degree of security and control over data access.
- Ease of Management: A centralized database is easier to manage because operations to analyze and answer complex queries does not involve the additional complexity of a network.
- Experience: A centralized data warehouse can draw upon the technical experience available for centralized database systems. Unfortunately, there is very little experience in managing this type of system on a distributed platform.

The disadvantages of a centralized data warehouse are:

- Performance: Many users must compete to access the data, resulting in delays caused by queuing requests. In addition, if the users are geographically distributed, delays also may occur due to transmitting access requests and responses to and from the central data warehouse location.

- Expandability: Expansion is expensive in a centralized data warehouse. For example, if the volume of the data exceeds the capacity of the warehouse, or the amount of processing against the data has increased, then the organization must replace the existing centralized data warehouse with a larger one. This is costly and may not be possible within an organization's fiscal constraints, which may cause the data warehouse to become outdated prematurely.

- Reliability: The centralized data warehouse is the single point of failure. When the whole system goes down, it usually takes some time to bring it back [Kau89].

- Cost: Experience over the past decade has shown that it is cheaper to have a number of smaller computers linked together rather than a large central machine.

- Vendor Dependency: Vendor dependency is an unfortunate consequence of any centralized approach. Typically, purchasing a single centralized system binds future expansion to the initial choice. Obviously, an inability to use multiple vendors, at least for negotiation, can be expensive, and migrating to a third-party vendor can be cost-prohibitive [Kau89].

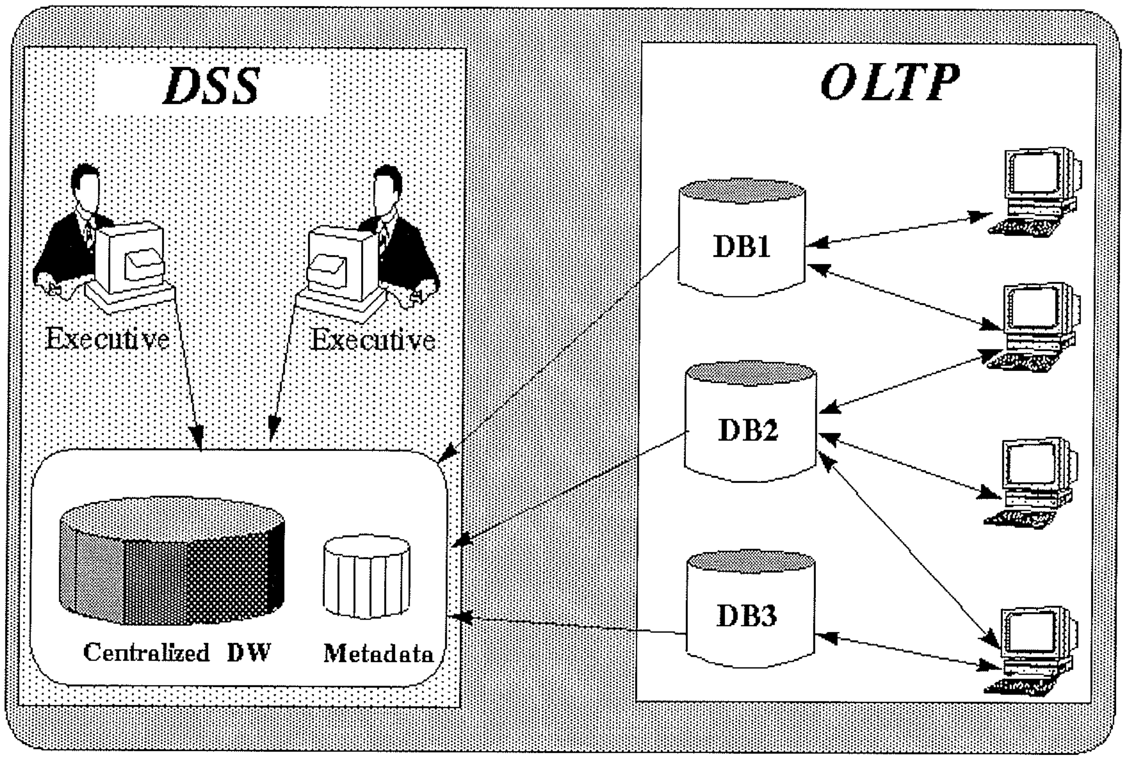


Figure 2.8: The whole organization located in the same environment.

2.3.2 Variation of Centralized Data Warehouse Architectures

A centralized data warehouse is the best choice if the organization uses a mainframe as its main information source. Figure 2.8 depicts a centralized data warehouse containing integrated data from multiple operational systems, which process on-line transactions (OLTP). Further, the centralized data warehouse contains metadata information describing the warehouse and source data, as described earlier. Finally, DSS end users run on-line analytical processing (OLAP) tools against the centralized data warehouse. The key feature of this architecture is that all processing is handled at a single location.

This architecture does not require that all data sources be located at the site of the data warehouse. A centralized data warehouse can also be applied to a large organization that has different divisions and geographic locations. For example, consider an organization with a headquarters and three geographically dispersed branches. Each branch processes OLTP-type applications against its databases. At the headquarters, the data warehouse environment receives data extracted from the branches and then integrates and stores them in the centralized data warehouse (see Figure 2.9). Also, it stores information about the data into the metadata repository. The warehouse data are then available for DSS end users.

There are two disadvantages to this approach. First, a branch will not have a

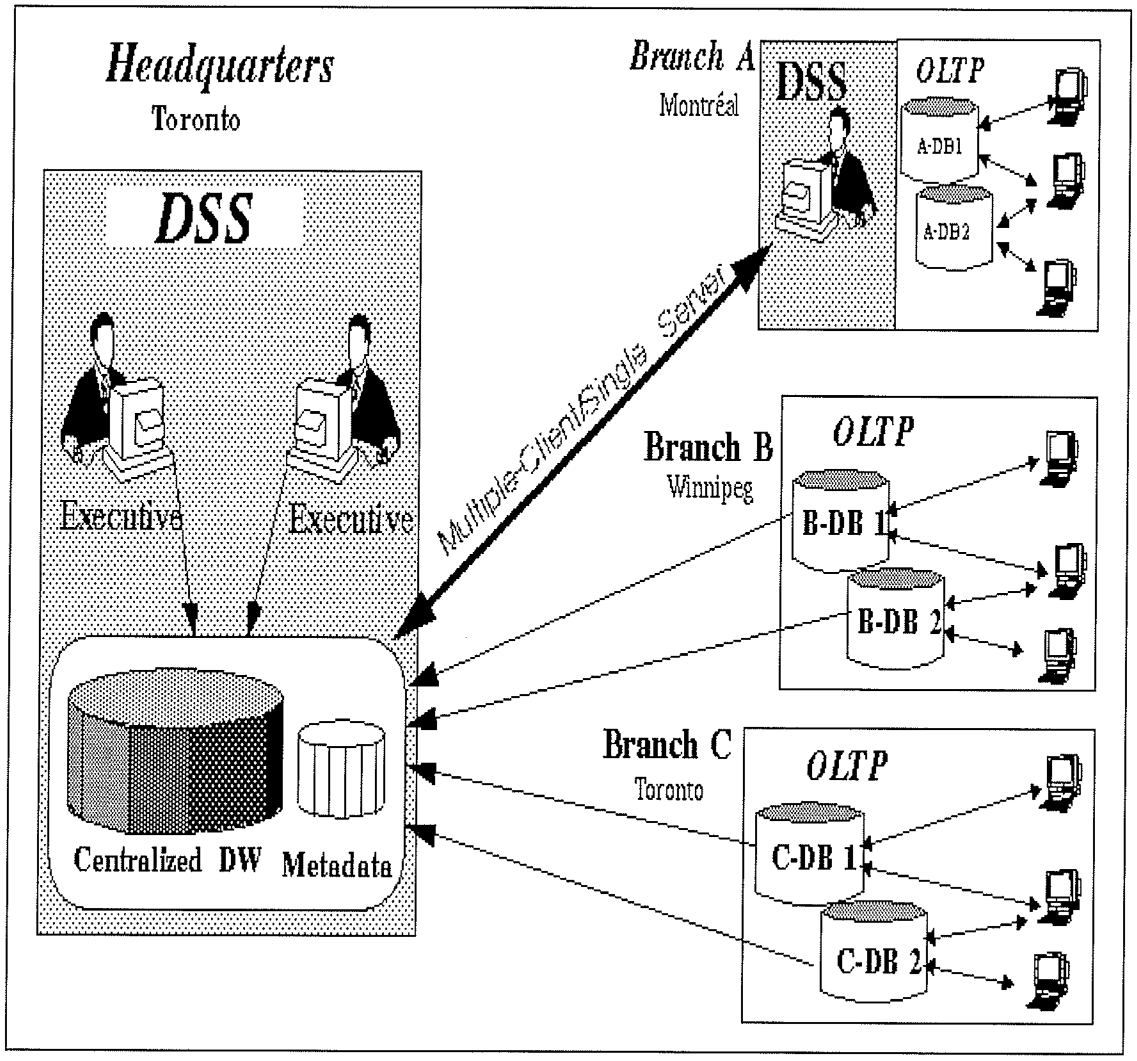


Figure 2.9: Multiple Client/Single Server Architecture.

chance to process "local" DSS request that may be important at the site containing the data. Second, the decision makers in a branch will be restricted to using only the DSS applications available at the headquarters.

The second disadvantage can be alleviated by applying a multiple-client/single server architecture to the distributed organization to form a communication bridge between the DSS tools (clients) in various branches and warehouse data (the server) in the headquarters. The client in each branch accepts requests from the DSS application and generates the proper protocol to convey those requests to the server, which accepts requests and passes them on to the warehouse's database engine. There, the requested service is performed and the results are returned to the client [IH94, Atr92] (see Figure 2.9). Unfortunately, this architecture suffers from the poor performance discussed earlier.

Another architecture that can be applied by a large organization with divisions in several geographic locations is the fully replicated data warehouse (or a Master data warehouse) [ÖV91, NB97]. It extracts data from disparate operational systems and integrates it into one central repository. This strategy is similar to the centralized data warehouse discussed earlier. Once the Master data warehouse and its metadata repository is created, copies are placed at each site designated by the organization. The duplicated copies are called *local replica data warehouses (LR-DWs)*. Each LR-DW represents the data warehouse server for the DSS end users at each site. The

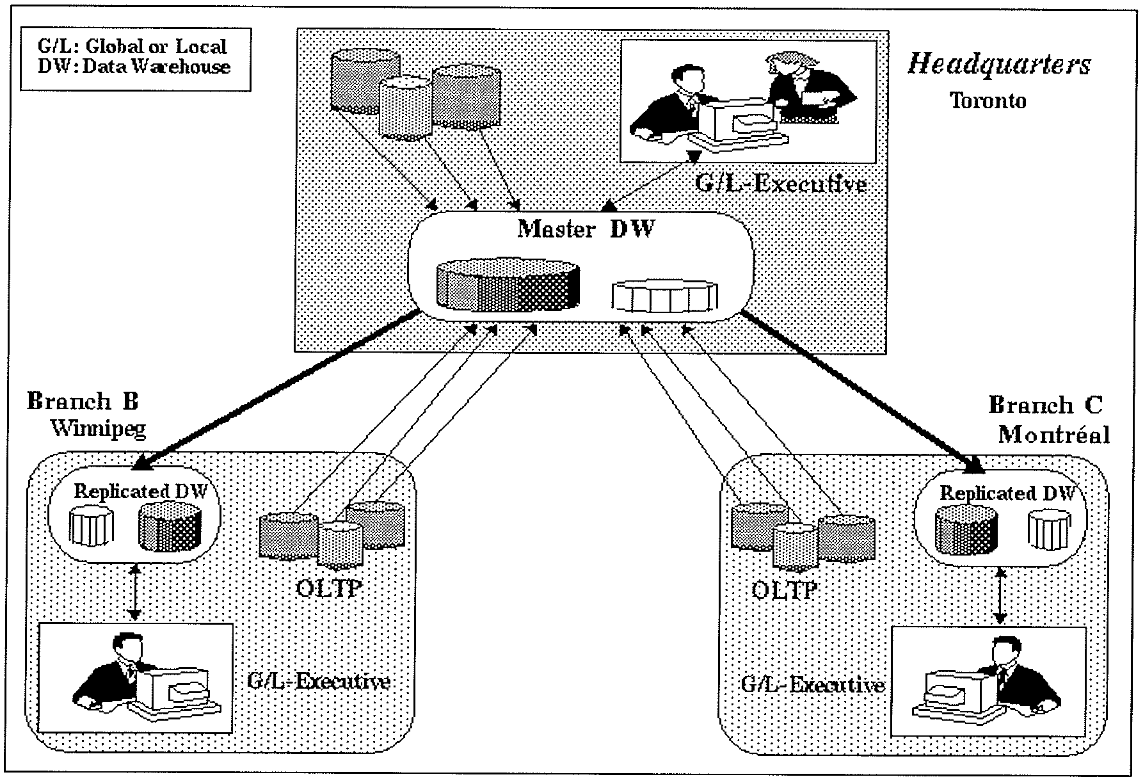


Figure 2.10: Replicated Data Warehouse Architecture.

local replica data warehouse is synchronized with the Master data warehouse regularly [ÖV91].

Figure 2.10 illustrates a replicated data warehouse. The Master data warehouse at the headquarters contains the integrated data from disparate operational systems together with its metadata. This data is available for the DSS end users and each branch has a local data warehouse containing a copy of the Master data warehouse accessible to DSS end users at those branches. The major gain in the replicated data warehouse approach is performance because DSS applications can operate on a local copy of the Master data warehouse instead of communicating with remote sites. Clearly, the disadvantages include redundancy, expandability, high cost and maintaining synchronization between the Master data warehouse and the local replica data warehouses update cost.

Chapter 3

Distributed Data Warehouse Framework

This chapter reviews other work on current distributed data warehouse architectures, the traditional modeling tools (the entity-relationship model and the relational model), and the distributed database design. Section 3.1 summarizes the current system architectures for building data warehouse in a distributed environment. Section 3.2 presents a review of the data modeling tools. Section 3.3 summarizes contributions made by researchers to the distributed database fragmentation problem in the relational data model.

3.1 Current System Architectures for Building Distributed Data Warehouse

Two approaches to building the distributed data warehouse have been proposed: Inmon [Inm93] and White [Whi95].

3.1.1 Inmon's Approach

Inmon's approach assumes the existence of both local and global data warehouses, with data stored in each being mutually exclusive. The local data warehouse contains data of interest to the local site and includes historical data in addition to local DSS functions. Figure 3.1 shows the local data warehouse fed by its local operational systems.

The global data warehouse contains data common across the corporation and data integrated from the various local staging areas for inclusion into the central location (headquarters). This is accomplished by having each local site stage warehouse data before passing it to the central global data warehouse, which provides the global DSS functionality for corporate-wide queries. Inmon's approach assumes that data found in any local data warehouse are not stored in the global data warehouse and vice versa, thereby guaranteeing no redundancy between them.

Inmon's assumption, about the mutual exclusivity of data between the local and

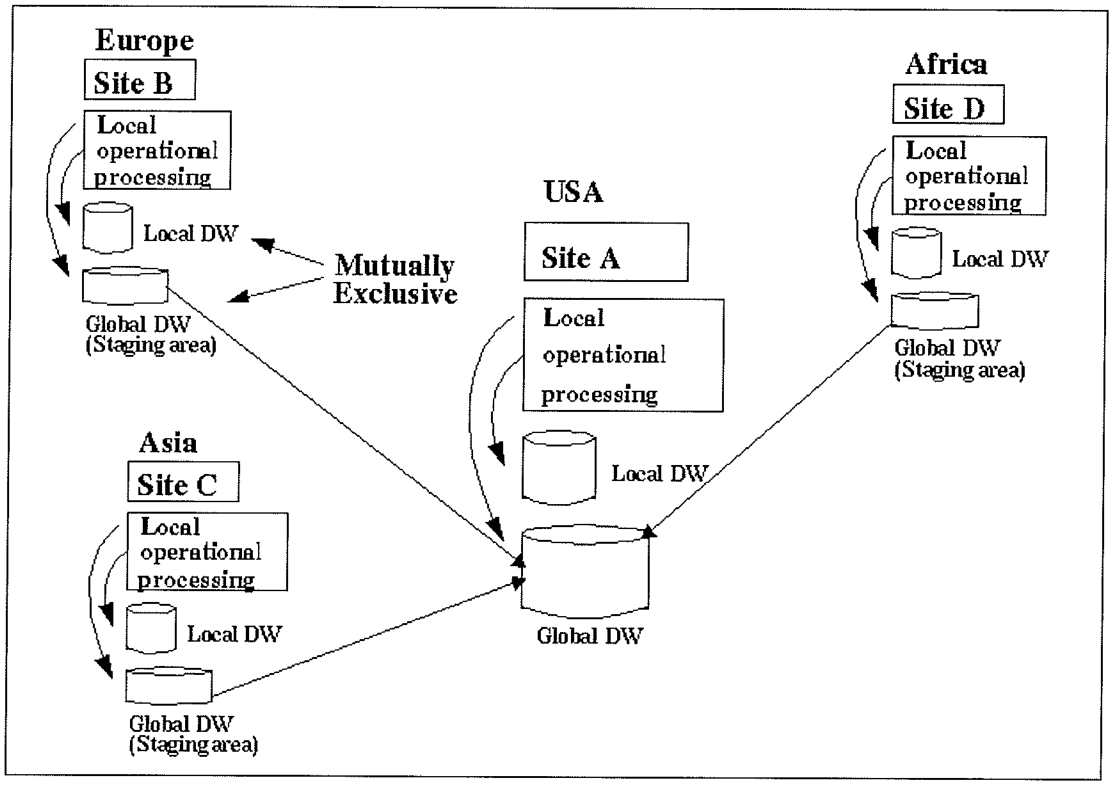


Figure 3.1: Inmon's Approach to DDW [Inm93].

global data warehouses, seems impractical. For example, a corporation with various locations may deal with a vendor that supplies a specific item to each of its branches. The vendor is thus a common attribute between the branches of the corporation and will be found only at the global data warehouse. If a local DSS end user asks the question: "What is the total number of item (x) supplied by vendor (y) during last two years?" the answer will not be available at the local data warehouse. This contradicts the definition of the local data warehouse which is "The local data warehouse contains data that is of interest to the local level" [Inm93]. Further, research into multidatabase systems shows that any such partitioning of data is impractical because the partition is either impossible to accomplish in general or the data tend to migrate to either the local or global level.

3.1.2 White's Approach

White's approach, also known as a "Two-Tier Data Warehouse," is a combination of a centralized data warehouse and a decentralized data mart. The data mart or decentralized data mart contains denormalized and summarized data that is of value to a specific user or user group.

White's central data warehouse contains normalized detailed data captured and cleaned from operational systems at user-defined intervals. The central data warehouse maintains data collections that consist of data derived from the detailed base data. Data collections are the user view of warehouse data and may contain denormalized detailed data as well as summarized data.

A data distribution service is provided by the data warehouse to distribute data collections to decentralized data marts at the various branches or sites of the corporation (Figure 3.2). The data marts are subsequently distributed to the other sites of the corporation.

Data marts permit DSS processing on local systems, which improves both per-

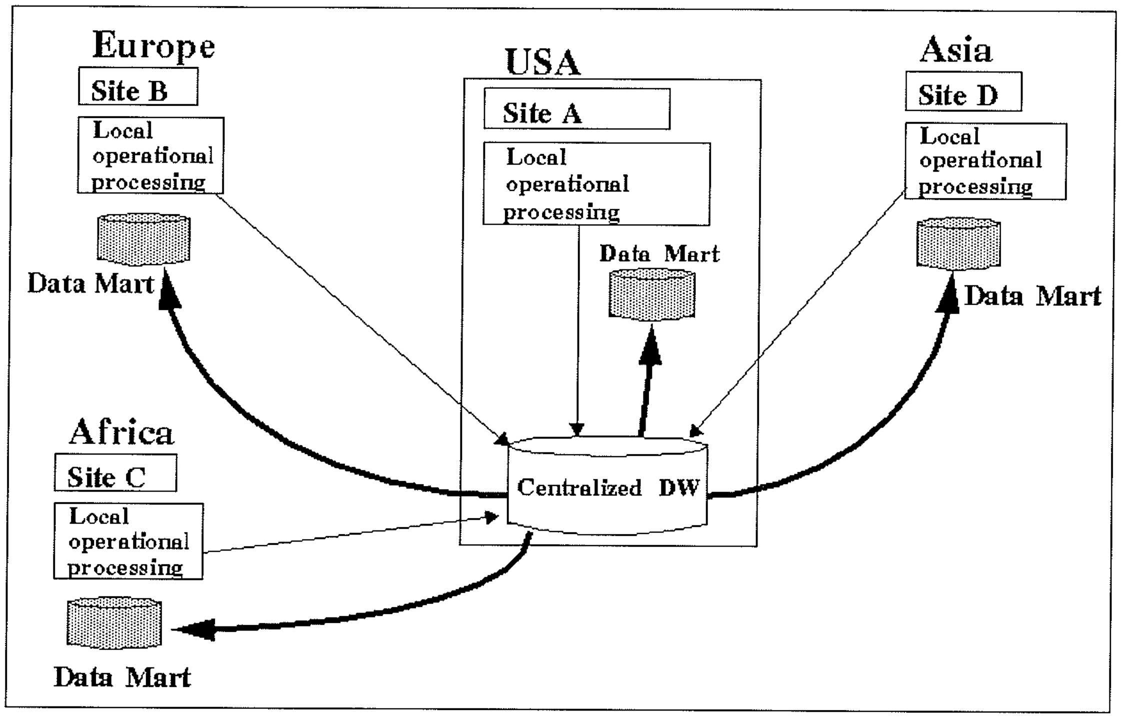

Europe
Site B
Local operational processing
Data Mart
USA
Site A
Local operational processing
Data Mart
Asia
Site D
Local operational processing
Data Mart
Africa
Site C
Local operational processing
Data Mart
Centralized DW

Figure 3.2: White's Approach to DDW.

formance and availability, but they do have some disadvantages. For example, when the number of data marts grows; the data redundancy and data management problems increase. Furthermore, data marts have limited flexibility for satisfying new information requirements.

3.2 Data Modeling

We now turn our attention to the data modeling tools used in the data warehouse development community. This section reviews the entity-relationship model and the relational model respectively, which are two commonly used traditional modeling tools.

3.2.1 The Entity-Relationship Data Model

A data model is the logical/physical representation of how the data is structured and how operations are performed on it. The data model that is most commonly used for logical database design is the entity-relationship data model. It was introduced by Chen [Che76].

The entity-relationship data model views data as entities, relationships, and attributes. Entities are "things" with independent existence. An entity for example could be a person, a house, or a company. Relationships are connections between two or more entities.

Entities and relationships are called objects. Each object has attributes that describe the properties of the object. Each attribute is coupled with a domain. The domain is a set of possible values that the attribute can assume [MR92]. The relationship between entity sets could be in a one-to-one, one-to-many, or many-to-many forms.

The entity-relationship data model is represented in a graphical form called an entity-relationship (ER) diagram. Figure 3.3 shows a simple ER diagram for a sales database.

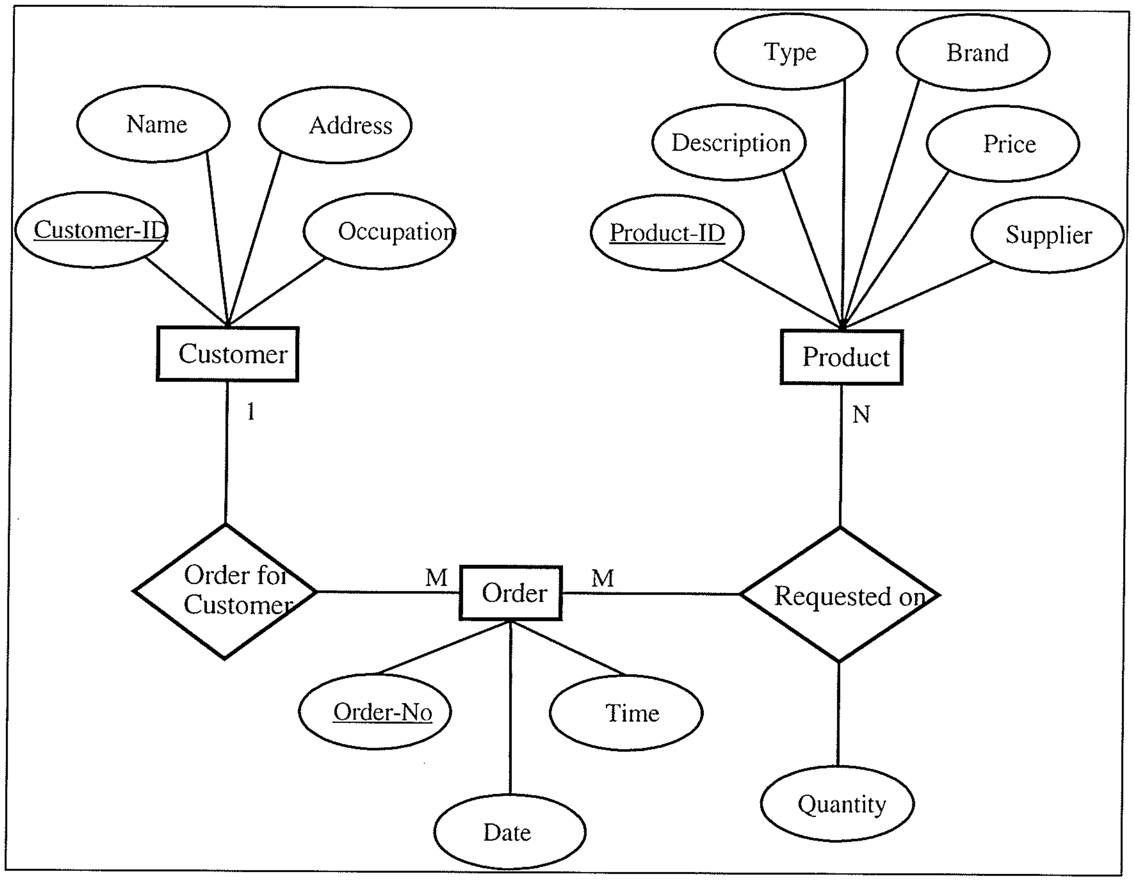


Figure 3.3: A simple entity-relationship diagram for the sales database.

In the diagram the entity sets are represented by rectangles, attributes by ellipses, and relationships among attributes by diamonds. For a given entity type, the attribute(s) that uniquely identify instances (the key) of the entity is underlined.

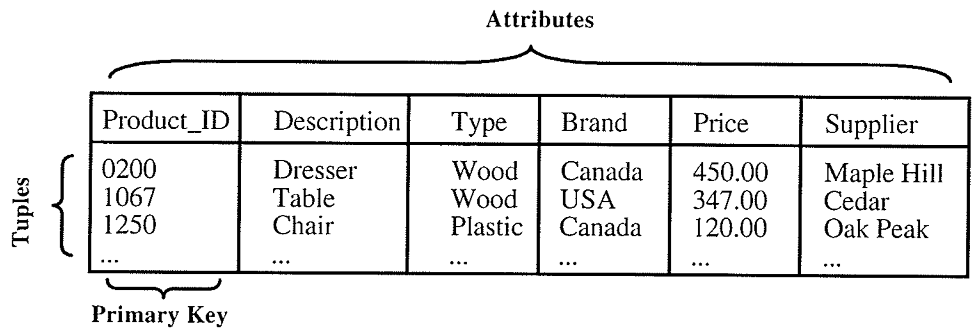
The ER diagram in Figure 3.3 represents the logical design for a simple sales database at the conceptual level. The simple sales database is described by the entities Customer, Order, and Product. The attributes describing the Customer entity includes a unique customer identity number *Customer-ID*, customer name, address, and occupation. The Order entity is described by the order identity number *Order-No* that represents the sale of one or more products at a particular date and time. The attributes describing the product entity are the product identity number *Product-ID*, product description, type, brand, price, and supplier. The relationships between entities in Figure 3.3 include *Order for Customer*, and *Requested on*. The *Requested on* relationship is described by the *Quantity* attribute.

3.2.2 Relational Model of Data

The relational model of data was proposed by Codd [Cod70] in 1970. It is based on the mathematical theory of sets. The relational model can be represented as two dimensional tables such as shown in Figure 3.4. Rows of the relation are referred to as tuples of the relation and the columns are its attributes. Each attribute of a relation represents a set of values known as a domain. The primary key of a relation is an attribute which can be used to uniquely identify a tuple in a relation. In the relational model, the ordering of the tuples within a relation is not defined, but the ordering of the attributes is fixed.

The relation R is represented by the notation $R(A_1, A_2, ..., A_n)$ where $A_i : R \rightarrow D_i$ is an attribute of R and D_i is its domain for $i = 1, 2, ..., n$.

Figure 3.4 shows the product relation. Each tuple of the relation is composed of


Attributes
Tuples
Primary Key

Product_ID	Description	Type	Brand	Price	Supplier
0200	Dresser	Wood	Canada	450.00	Maple Hill
1067	Table	Wood	USA	347.00	Cedar
1250	Chair	Plastic	Canada	120.00	Oak Peak
...	...	...	...	...	...

Figure 3.4: Product relation

"six valued" attribute. The primary key of the relation is the *Product_ID* attribute.

In the relational model, all relations must be in at least *first normal form.* The first normal form relation requires all its attributes to be *atomic.* An atomic attribute cannot be further subdivided and is assigned a single value at any point in time.

Normalization is the process of improving the structure of a given collection of relations into well-structured relations. The well-structured relation is one that is free of redundancy and is at its most decomposed level. It allows users to insert, delete, and modify its rows without errors or inconsistencies.

The process of normalization begins with the original data structure and progresses through first, second, and third normal forms. The original data structure is an unnormalized relation. The unnormalized relation is a relation that may contain repeating values and relations nested within other relations.

Normalized relations are appropriate for On Line Transaction Processing (OLTP) because they eliminate the insert, delete, and update anomalies. However, normalized relations will slow down database performance because join operations may be needed to answer a requested query. In general, normalized relations have been designed and tuned for OLTP applications and are inappropriate for On Line Analytical Processing (OLAP) applications [Sta93, AGS95].

Normalized relations are not ideal for data warehouse or OLAP because the data warehouse requires high performance to answer on-line data analysis queries. Chapter 5 will present in detail the schema used for structuring the warehouse data.

3.3 Distributed Database Design

Recall from the introduction (see Section 1.3) where two approaches for distributed database design were introduced: the top-down approach and the bottom-up approach [ÖV91, CP84]. We now consider these in more detail. The top-down approach is used when the databases are non-existent. However, once the databases exist (for example, the multidatabase environment), the bottom-up design is the appropriate approach.

In the top-down approach, the starting steps of the design process are the *requirement analysis* and *logical design* of the global database. The output of these two steps are the Global Conceptual Schema (GCS) and the access pattern information. These two outputs represent the input for the *distributed design* step. The objective of the distributed design is to devise the Local Conceptual Schema (LCS) by generating entities called fragments. These fragments are then allocated to the distributed sites. The *physical design* is the last step in the top-down design approach process. It maps the LCS with the access information to the distributed physical storage devices.

In the bottom-up approach, the design process consists of [CP84]:

- Selecting a common database model for describing the global schema of the existing databases.
- Translating each local schema into the common data model.
- Integrating local schemas from the existing databases into the global conceptual schema.

Since the emphasis of this research is on the top-down design, the previous works related to this design are reviewed.

There are two fundamental issues in top-down design: fragmentation and allocation. Fragmentation partitions each global relation into a set of local relations (fragments). Allocation distributes these local relations across the sites of the distributed system [ÖV91].

There are three fragmentation strategies: horizontal, vertical, and mixed. Horizontal fragmentation is defined as the process of subdividing (i.e., selecting) the tuples of the relation into groups of tuples. Each group has the same attributes as the original relation. Vertical fragmentation is defined as the process of subdividing (i.e., projecting) the attributes of the relation into groups. Each group contains the primary key of the original relation. Mixed fragmentation is defined as the process of performing horizontal fragmentation followed by vertical fragmentation or vice versa.

To ensure that the semantic characteristics of the relation does not change as a result of applying the fragmentation strategies, three correctness rules are required. These correctness rules are: completeness, disjointedness, and reconstruction [ÖV91]. Completeness ensures that all tuples from a relation are mapped into at least one fragment without any loss. Disjointedness ensures that the generated fragments are non-overlapping. Finally, reconstruction ensures that the union of all fragments should reproduce the original relation.

The following subsections summarize previous work on the three fragmentation strategies in the *relational data model.*

3.3.1 Horizontal Fragmentation

Ceri, Negri, and Pelagati [CNP82] fragment relations horizontally by determining the logical properties of data, such as the fragmentation predicates, and the statistical properties of data, such as the reference of applications to fragments.

Shin and Irani [SI91] fragment relations horizontally using a knowledge-base approach. This approach uses the knowledge of the data to derive precise *user reference* *clusters* (URCs). They proposed a knowledge-base system. It consists of a knowledge-base about the relations and an inference mechanism. The inference mechanism applies the knowledge base to the user queries to produce new equivalent queries from which the highly accurate URCs will be derived.

Ceri, Navathe and Wiederhold [CNW83] define a logical schema model which is needed to generate two types of horizontal fragmentation: *primary* and *derived.* The logical schema is modeled as a directed graph with (1) objects, such as normalized relations for the relational model, that are represented as nodes; (2) links, such as relationships among relations, that are represented as edges. The relation at the tail of a link is called the owner of the link and the relation at the head is called the member. Primary horizontal fragmentation is performed using disjoint predicates that are defined on owner relations while derived horizontal fragmentation is derived from a member relation by using predicates that are defined on an owner relation.

Özsu and Valduriez [ÖV91] specify stronger requirements (database information and application information) to permit the two types of horizontal fragmentation: primary and derived. They provide an algorithm for the primary horizontal fragmentation. The input to the algorithm is a relation with simple predicates that have been determined in accordance with the applications defined on the relation. The output of the algorithm is a set of minterm fragments.

Navathe, Karlapalem and Ra [NKR95] present an algorithm for horizontal fragmentation as a part of a mixed fragmentation methodology. This methodology minimizes the number of disk accesses required to process the distributed transaction. The main idea of the algorithm is similar to the vertical fragmentation algorithm presented in [NR89]. The algorithm is based on a graphical technique and assumes that all simple predicates are previously determined. The algorithm clusters and optimizes the predicates terms. Finally, the horizontal fragments will be generated and adjusted to remove the overlapping ones.

3.3.2 Vertical Fragmentation

There are two approaches for vertical fragmentation in the relational data model. The first approach provides an *optimal* solution for vertical fragmentation with restrictive assumptions such as the file capacity and numbers of attributes [Hof76, ES76, MS77, Sch77]. The second approach is heuristic [HS75, HN79, NCWD84]. Özsu and Valduriez [ÖV91] classify these techniques into two types: *splitting* and *grouping*.

Splitting technique measures the access of applications to the attributes and uses these measures to split the relation into fragments. Hoffer and Severance [HS75] measured the affinity between pairs of attributes and clustered them based on their affinity. They used the bond energy algorithm (BEA), developed by McCormick *et al.* [MSW72], to cluster the attributes. Navathe *et al.* [NCWD84] extend the work of Hoffer and Severance by:

- providing algorithms for splitting attributes into nonoverlapping or overlapping fragments.

- proposing a two phase design for vertical fragmenting. The first phase is based on the logical access frequencies of the transaction. The second phase is based on the cost factors related to the physical environment.

- applying the proposed vertical fragmenting technique into an environment where the database is stored on one level of memory, a memory hierarchy or on a distributed database environment.

Özsu and Valduriez [ÖV91] describe the work of Hoffer and Severance [HS75] in more detail. They provide a partitioning algorithm for the distributed database environment based on the work of Navathe *et al.* [NCWD84]. Navathe and Ra [NR89] proposed a graphical algorithm that represents the attributes affinity values in a complete graph, then a linearly connected spanning tree is formed. Each cycle on the spanning tree represents a fragment.

On the other hand, the *grouping* technique proposed by Hammer and Niamir [HN79] has two phases which iterate until no further improvement is found. The first phase "*grouping*" starts by assigning each attribute to one candidate fragment. Then a set of variations is constructed from the candidate fragments. These variations are submitted to the fragment evaluator for cost evaluation. The variation that represents the greatest improvement over the current candidate becomes the new candidate. This process iterates until no improvement over the current candidate occurs.

The second phase is the attribute "*regrouping*" phase. It starts with the final candidate of the first phase. At each step of this phase several variations are constructed by moving an attribute from its current fragment to another fragment. This process iterates until no improvement is found over the current candidate.

3.3.3 Mixed Fragmentation

Mixed fragmentation is needed when the database users require access to subsets of data that are both vertical and horizontal fragments of global relations. Navathe, Karlapalem and Ra [NKR95] developed graphical algorithms for generating horizontal and vertical fragmentations. The algorithm that generates horizontal fragmentation is based on the *predicate affinities* while the algorithm that generates vertical fragment is based on *attribute affinities.* Moreover, they present a mixed fragmentation methodology that simultaneously applies the horizontal and vertical algorithms to generate a grid on a relation. This grid consists of cells which are merged to form mixed fragments. They argue that the mixed fragmentation methodology is an optimal partitioning technique for distributed databases.

Chapter 4

Distributed Data Warehouse Architecture

This chapter discusses the feasibility of building a data warehouse using the distributed technology and proposes an architecture model for the distributed data warehousing. Section 4.1 discusses the advantages and disadvantages of distributed data warehousing. Section 4.2 presents the proposed architecture model for distributed data warehousing, provides on depth description of its system architecture, and illustrates how the information flows in the distributed data warehouse.

4.1 Advantages and Disadvantages of Distributed Data Warehousing

Arguments over centralized versus distributed systems have occurred since the early 1960s and will probably continue into the foreseeable future [Mar81]. Current approaches suggest building the data warehouse as a single centralized repository even if the organization is geographically distributed. However, if the whole organization is geographically distributed, the data warehouse should ultimately be distributed as well. It would be unwise to force a centralized data warehouse when the operational systems exist over a widely distributed geographical area.

Several advantages are evident in a distributed data warehouse:

- *Local Autonomy*: Each site in a distributed data warehouse is autonomous. The local warehouse data is locally owned, managed and controlled, even if it is accessible from other remote sites. Conversely, a local user is allowed to access remote data but must conform to the requirements imposed by that remote location.

- *Performance*: Data in a distributed data warehouse can be stored close to its normal point of use, which reduces response time and communication costs. Moreover, since each site handles only a portion of the warehouse data, contention for CPU and I/O services is not as acute. Finally, the inherent parallelism in the system can lead to improved throughput and response times [ÖV91].

- *Reliability and Availability*: A distributed data warehouse offers increased reliability and availability. Reliability means that the system can continue to function despite failure of one or more sites. If a distributed data warehouse supports replicated data at more than one site, a crash or failure of a communication link at one or more of the sites does not necessarily make the warehouse data inaccessible.

- *Incremental Growth*: A major advantage to a distributed data warehouse is that as it grows, due to constantly increasing historical data or re-design and extensions, the system can evolve cost effectively. Additional processing and storage can be added incrementally because there is no need to build a new data warehouse on a new and expensive platform. Finally, new sites can be easily and expeditiously added whenever the organization expands.

The disadvantages of a distributed data warehouse are:

- *Security*: A distributed data warehouse utilizes a network, which introduces a weak security link [ÖV91].

- *Complexity*: Since data are stored at different sites, access and management is more challenging. Technology advances and research into distributed database management systems (DDBMS) provide many useful insights into this problem.

- *Cost*: Each site must have people to maintain the system [ÖV91].

Large organization that have different divisions and geographic location will benefit from the distributed data warehouse. The distributed data warehouse will provide on-line analytical processing for the decision makers at each division and at the headquarters of the organization.

4.2 Distributed Data Warehouse Architecture Model

This section proposes a distributed data warehouse architecture model. The proposed architecture represents the classic solution for a large enterprise with various divisions and geographic locations. Section 4.2.1 presents the architecture model [NB97]. The architecture provides the geographically distributed organization with a Distributed Detailed Data Warehouse and a Centralized Summarized Data Warehouse. Since the emphasis of this research is on the distributed data warehouse, Section 4.2.2 provides, in depth, the system architecture for the distributed detailed data warehouse.

4.2.1 The Architecture Model

The architectural model for distributed data warehouses is based on the decision makers' needs.

The Decision Makers' Needs: The primary objective of a decision maker is to make the corporation run better, increase market share, reduce cost and expenses, and increase revenues [IH94]. To do this, decision-makers need to know when things are going wrong (and why) or when to recognize an opportunity (e.g., an acquisition or new product) [PC93]. Decision makers have developed several strategies for extracting the most important data patterns from huge amounts of information. Strategies such as using subordinates to collect and filter information and using various industry summaries, reports, and newsletters to provide summarized information are examples.

The huge volumes of information are potentially relevant to any problem but it must be filtered and summarized. Unfortunately, summarization throws away potentially useful information. The solution to this paradox is to support the decision makers with a system that provides summaries about the corporation status that are linked to the underlying detailed information, which permits "drill-through" to the detailed data when required. Typically a Decision Support System (DSS) and an Executive Information System (EIS) are used as part of this decision support.

In a geographically distributed organization, there are two types of decision-makers:

- those who make decisions for the entire organization, and
- those who make decisions for one of the organization's branches.

The decision-makers whose scope is the entire organization usually emphasize long-term trends in their information processing. They require highly summarized information that is strategic in nature. On the other hand, decision-makers at the branch level tend to be limited to a branch-based view so the emphasis is on short-term trends. Obviously, immediate and detailed information is required to make the best possible decision for the branches [IH94].

Since there are two categories of decision-makers in a geographically distributed organization, the warehouse should mirror the users' needs. Therefore, a distributed warehouse architecture that provides the geographically distributed organization with a Distributed Detailed Data Warehouse (DDDW) and a Centralized Summarized Data Warehouse (CSDW) is ultimately required.

Distributed Detailed Data Warehouse: In the DDDW, the dispersed branches of the organization will be connected by the peer-to-peer approach that provides a fully distributed data warehouse system. At each branch in the distributed data warehouse, the data will be integrated from the local operational systems, fragmented, and allocated into one or more of the local data warehouses in the organization (see Figure 4.1).

The distributed data warehouse supports the two categories of decision makers by providing a single view of data even though that data is physically distributed across multiple data warehouses in multiple systems at different branches (i.e., distribution

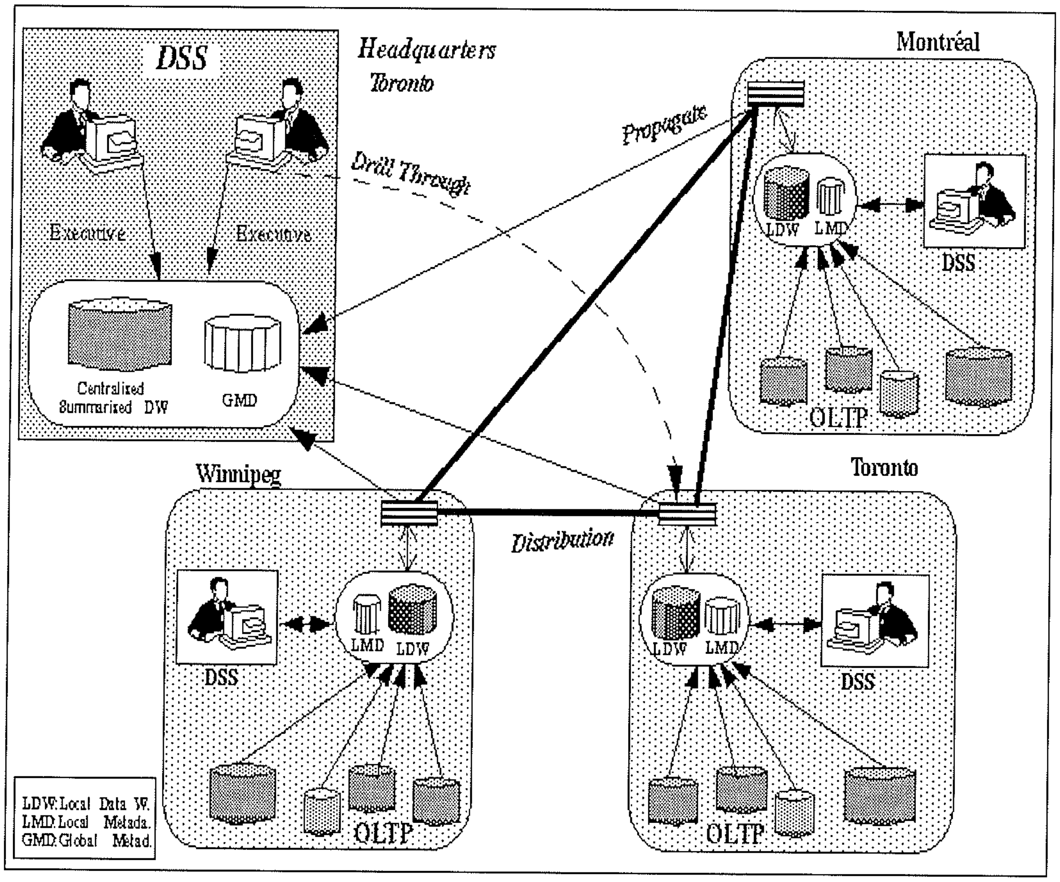


Figure 4.1: Architectural Model to DDW [NB97].

transparency). Distribution transparency allows the local decision makers to run DSS application against their local data warehouse to fulfill their local needs, and the decision makers at the headquarters can drill through to the detailed data of any branch whenever necessary. Section 4.2.2 will present the system architecture for the distributed data warehouse.

Centralized Summarized Data Warehouse: The CSDW is located at the headquarters (see Figure 4.1). Each branch contains the summarized data created by selecting, projecting, and aggregating relational data into views more convenient and useful to the decision makers. This summarized data is then propagated to the headquarters, where it is aggregated with summary data from other branches in a central repository and as spreadsheets, text documents, and charts.

The CSDW provides quick response time as the decision-makers are at the same location as the data so they will not be affected by network delays. It provides efficient analysis because the summarized data are updated directly when newly integrated data are loaded into the current detailed data warehouse. Further, it provides high security over the most sensitive data in an environment that is easy to manage.

4.2.2 The Distributed System Architecture

The distributed data warehouse system architecture is based on the ANSI/SPARC architecture [TK78] that has three level of schemas: internal, conceptual and external.

Figure 4.2 illustrates the distributed data warehouse architecture model. It is a four-tiered architecture consisting of (1) *The data integration layer* which includes the source database systems and the processes required to integrate them. (2) *The data staging layer* integrates subject oriented and current detailed data using a homogeneous model. (3) *The data distribution layer* fragments, allocates and updates the distributed data warehouse with changes occurring in the operational systems. (4)

The distributed data warehouse manager layer is responsible for interfacing with the decision support environment by providing a corporate-wide view of the distributed data across the network. The following sections discuss these layers in detail.

The Data Integration Layer

The data integration layer consists of the source databases available across the sites and the integration and transformation tools. Figure 4.2 shows the multiple source databases available across the sites. These sources run the gamut from full functioning On-Line Transaction Processing (OLTP) to various unstructured external data sources such as flat files or spreadsheets. The individual source at each site has its own *Local Internal Schema* (LIS) and *Local Conceptual Schema*[1] (LCS). The LIS defines the physical data organization on the source database while the LCS represents the abstract definition of the data and the relationships between them. The LIS and LCS in each source database has its own native data model. The integration and transformation tools extract data and information about the data from the source databases, define the relationships among data at multiple sources, detect duplicates and inconsistencies, add any extra desired information such as timestamps and transform the integrated data to the target database in the data staging layer.

[1] Conceptual schema represents the data and the relationships between data without considering the requirements of individual applications or the restrictions of the physical storage media [ÖV91].

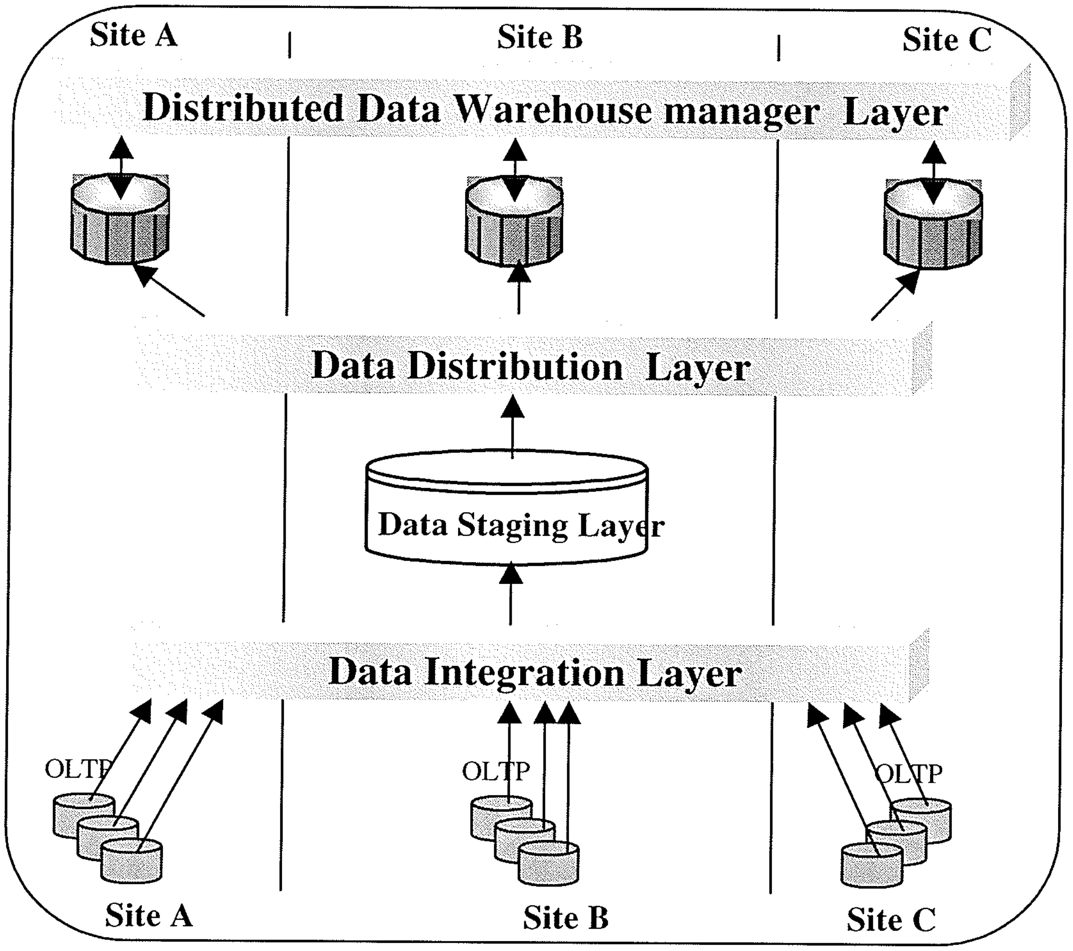


Figure 4.2: Distributed Data Warehouse System Architecture.

The Data Staging Layer

The data staging layer stores the integrated, subject oriented, current-value and detailed data. The integrated data in the staging layer is stored using the *Integrated Conceptual Schema* (ICS), which defines the local schema of the entire source databases. The underlying model for the ICS is a canonical data model. The ICS will be transformed[2] into the *Global Conceptual Schema* (GCS) (see Figure 4.3). The underlying model for the GCS is the data warehouse model (see Chapter 5 for the detail of the data model for data warehouse).

The data stored in the data staging layer can be classified into new data and changed data. The new data represents the most recent data that is added to the operational systems. The changed data represent the update occurring on existing data in the operational systems.

The Data Distribution Layer

The data distribution layer provides the following processes: fragmentation, allocation and updating the distributed data warehouse. The fragmentation process applies its algorithms to the data that have been mapped into the data warehouse model (GCS). The fragmentation algorithms partition these data into fragments. The allocation process applies its algorithms to distribute the fragments to the sites available across the network. Finally, the update process applies its algorithms to add the new information to the history of existing data available in the fragments in the distributed data warehouse.

The main objective of the fragmentation and allocation processes is to minimize the total transaction processing cost for a given set of transactions. When the transaction processing cost increases and reaches to a specific threshold, the fragmentation

[2] Transformation between different schemas is performed by mappings which specify how a definition at one schema can be obtained from a definition at another schema [ÖV91].

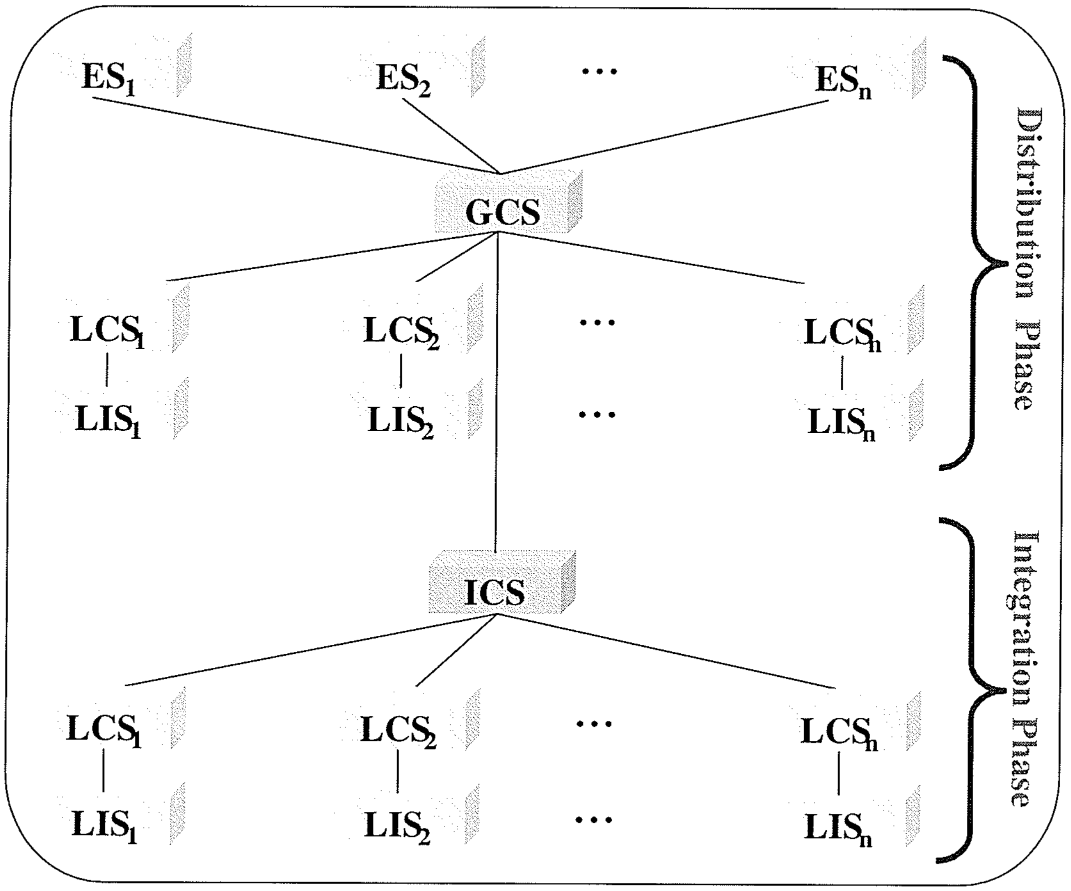


Figure 4.3: Distributed Data Warehouse Reference Architecture.

and allocation processes will be reapplied.

The Distributed Data Warehouse Manager Layer

The distributed data warehouse manager layer manages the fragments at each site. These fragments represent the *integrated, subject oriented, non-volatile, time variant and detailed data.* This layer also provides a single view of the fragments even though these fragments are physically distributed across multiple sites on multiple systems.

At every site, the logical organizations of the fragments are defined by the Local Conceptual Schema (LCS). Each LCS is mapped to a Logical Internal Schema (LIS), which defines the physical organization of the fragments stored at the local site. The DSS end users at each local site are supported by *External Schema* (ES) to allow them to execute the DSS applications against the data warehouse (see Figure 4.3). The GCS represents the union of the local conceptual schemas. It is defined between the external schemas and the local conceptual schemas to support network transparency between the various distributed data warehouse sites.

The various sites are connected using a Peer-to-Peer approach. Peer-to-Peer means that a single application is able to operate "transparently" on data spread across different data warehouses, managed by a variety of different DBMSs, run on a variety of different machines, supported by a variety of different operating systems, and connected by a variety of different communication networks [Dat95].

A distributed data warehouse implemented using the peer-to-peer approach can be viewed as a partnership among various sites and their individual local DBMSs. A new software component at each site, logically an extension of the local DBMS, provides the necessary partnership functions. The combination of this new component with the existing DBMS is called the distributed data warehouse management system (DDWMS).

It is important to distinguish true, distributed data warehouse systems (DDWS) from systems that merely provide remote data access. In a remote data access system, the user might be able to operate on data at a remote site, or even data at several remote sites simultaneously, but the user is aware that the data is remote so no support is provided for the distribution transparency that a full DDWS provides.

Chapter 5

The Data Model for A Data Warehouse

The data model for a data warehouse should be designed to structure the data in a manner that supports the *On-Line Analytical Processing* (OLAP). There are two techniques for modelling the data warehouse: the *multidimensional* data model [LAW98, Leh98, CT97a, GL97, LW96, AGS95] and the *relational* data model [CD97, Sho97, Kim96]. These two modelling techniques provide multidimensional views of data to support and facilitate the OLAP applications. This chapter presents the relational data model for a data warehouse. Section 5.1 discusses the relational data warehouse schema and its formal definition. Section 5.2 presents a review of the variation of the data warehouse schema design.

5.1 Data Warehouse Schema

The data warehouse schema (star schema) design was introduced by Kimball [Kim96]. The basic principle behind it is building a highly redundant data structure to improve database performance for OLAP applications.

The *data warehouse schemes* 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 [AM97, CD97]. The large central tables are called the fact tables and the surrounding tables are called the dimensional tables.

Definition 1 *(Data Warehouse Schema).*
Data warehouse schema S is an ordered pair ($\mathcal{D}$, $\mathcal{F}$); $S = (\mathcal{D}, \mathcal{F})$ where:

- $\mathcal{D} = \{D_1, D_2, ..., D_u\}$ *is a set of dimension relation schemes;*
- $\mathcal{F} = \{F_1, F_2, ..., F_q\}$ *is a set of fact relation schemes.* ■

The data warehouse schema is a set of relation schemes. Two types of relation schemes are available in data warehouse: Dimensions and Facts. The relationship between the dimension relation and fact relation is one to many as shown in Figure 5.1.

5.1.1 The Dimension Relations

The dimension relations are descriptive relations that add value to the quantitative data available in the fact relation. They contain multiple attribute columns con-

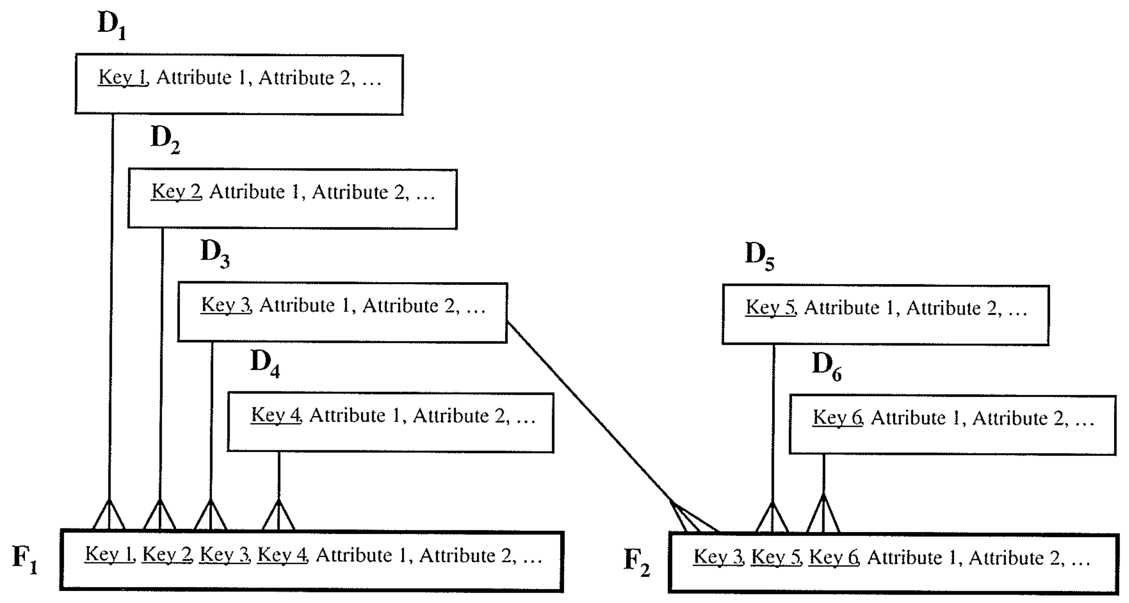

D1
Key 1, Attribute 1, Attribute 2, ...
D2
Key 2, Attribute 1, Attribute 2, ...
D3
Key 3, Attribute 1, Attribute 2, ...
D4
Key 4, Attribute 1, Attribute 2, ...
D5
Key 5, Attribute 1, Attribute 2, ...
D6
Key 6, Attribute 1, Attribute 2, ...
F1
Key 1, Key 2, Key 3, Key 4, Attribute 1, Attribute 2, ...
F2
Key 3, Key 5, Key 6, Attribute 1, Attribute 2, ...

Figure 5.1: The data warehouse schema.

taining text and codes that describe the dimension key. The dimension relations are heavily attributed to support the "what-if" queries required to derive decision-making information. They also represent the "by" criteria. For example, the DSS end user may wish to see "sales by region" or "monthly sales by sales person".

The dimension relation is a denormalized relation. It is constructed by applying a denormalization process [Bur97].

Definition 2 *(Denormalization).*

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

The advantages of using denormalization in building data warehouses are [Poe96]:

- Reducing the number of joined relations required to answer queries. As a result, the run time application is improved.
- 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 relations from the *On-Line Transaction Processing* (OLTP) schema. Figure 5.2 illustrates normalized relations based on geographical regions. Clearly these are in at least third normal form (3NF). To perform OLAP analysis these should be "pre-joined" to enhance the performance of the OLAP queries. The normalized relations often have a hierarchical structure among themselves. This hierarchical structure will be captured during the denormalization process. Consider, for example, the hierarchical structure between the normalized relations of the OLTP store schema in Figure 5.2. The normalized relations (Region, District, State, City and Store) have a hierarchal structure between them where the highest level of the hierarchy is the Region relation and the lowest level is the Store relation.

The denormalization process produces two outputs:

1. The *dimension relation*: Figure 5.3 shows the store dimension relation after it has been denormalized out of the normalized relations which was shown in

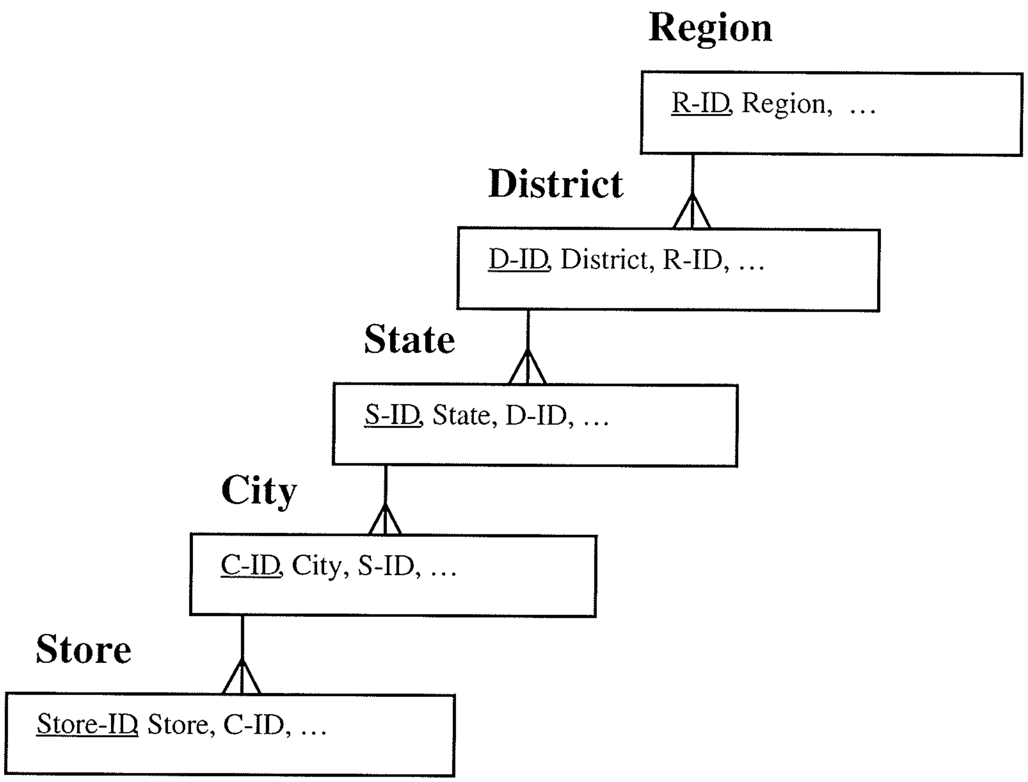


Figure 5.2: The OLTP store schema.

Figure 5.2. In the store dimension relation the store-key (of the lowest level relation in the hierarchy of the normalized relations) has been associated with the non-key attributes of the other normalized relations.

2. The *attributes hierarchy* for the dimension relation: The attributes hierarchy will be represented by the relation $D_{hi}(A_{i1}, A_{i2})$ where D_{hi} is the attributes hierarchy relation of the dimension D_i, the domain Q_{i1} of A_{i1} represents the attributes of the dimension D_i involved in the hierarchy and the domain Q_{i2} of

A_{i2} represents the level of each attribute in the hierarchy. For instance, the attributes hierarchy of the store dimension relation is *Store(Attributes-Hierarchy, Level)* and it has the following values:

Attributes Hierarchy	**Level**
Store	1
City	2
State	3
Distrect	4
Region	5

The attributes hierarchy is used in rolling-up and drilling-down operations of the OLAP applications. These two operations represent moving along the attributes hierarchy to decrease or increase the level of the aggregation [CD97]. The *roll-up* operation increases the level of aggregation, such as viewing the sales data from sales by city to sales by region. The *drill-down* operation decreases the level of aggregation, such as viewing the sales data from sales by region to sales by district. The attributes hierarchy will be used later in the proposed algorithm for fragmenting the data warehouse schema horizontally.

The context for the formal definition of a dimension's relation schema requires recalling Definition 1 of the data warehouse schema where $S = (\mathcal{D}, \mathcal{F})$.

Store-Key	Store	City	State	District	Region	Store-Manager
1	Store No. 1	New York	NY	New York	Eastern	Jones
2	Store No. 2	Philadelphia	PA	Philadelphia	Eastern	Williams
3	Store No. 3	Pittsburgh	PA	Allegheny	Eastern	Erickson
4	Store No. 4	Pittsburgh	PA	Allegheny	Eastern	Davis
5	Store No. 5	Los Angeles	CA	Los Angeles	Pacific	Smith
6	Store No. 6	San Francisco	CA	San Francisco	Pacific	Mona
7	Store No. 7	Boston	MA	Suffolk	Eastern	Amin
8	Store No. 8	Miami	FL	Dade	South East	Cobb
9	Store No. 9	Miami	FL	Dade	South East	Cobb
...	...	...	...	...	...	...

Figure 5.3: The Store Dimension Relation.

Definition 3 *(Dimension).*

The dimension relation schema for any D_i is represented by $D_i(A^D_{i1}, A_{i2}, A_{i3}, ..., A_{ij}, ..., A_{im})$ where $A_{ij} : D_i \rightarrow Q_{ij}$ is an attribute of D_i and Q_{ij} is its domain for $1 \leq i \leq u$ and $1 \leq j \leq m$. ■

The *dimension relation key* A^D_{i1} functionally determines all attributes of the dimension relation D_i, that is, $A^D_{i1} \longrightarrow A_{ij}$ for $1 \leq i \leq u$ and $2 \leq j \leq m$. The superscript letter D in A^D_{i1} implies that A^D_{i1} is a dimension key.

There are two types of the *dimension relation attributes*:

1. *Non-hierarchical attribute*: it contains descriptive information about the dimension but it is not involved in the dimension attributes hierarchy.

2. *Hierarchical attribute*: it contains descriptive information about the dimension and involves the dimension attributes hierarchy.

The relation scheme for the Store dimension relation (see Figure 5.3) is (*StoreKey,* *StoreName, City, State, District, Region, StoreManager*). The Store dimension relation is in first normal form. The functional dependencies in this relation are {*StoreKey* → *(StoreName, City, State, District, Region, StoreManager), StoreName* → *(City, State, District, Region, StoreManager), City* → *(State, District, Region), State* → *(District, Region), District* → *Region*}. The functional dependencies in the Store dimension relation are transitive dependencies. The presence of these transitive dependencies are acceptable in the data warehouse schema to support the on-line analytical processing.

5.1.2 The Fact Relations

The fact relation normally contains millions of rows and is highly normalized [Kim96]. The fact relation stores time-series factual data. These data represent real values that the DSS end users need to track. The fact relation is composed of foreign keys and raw data. Each foreign key references a primary key on one of the dimension relations. These dimension relations could be time, product, market, vendor, customer, etc. The raw data represent the numerical measurement of the organization such as sales amount, number of units sold, prices and so forth. The raw data usually never contain descriptive (textual) attributes because the fact relation is designed to perform arithmetic operations such as summarization, aggregation, average and so forth on such data.

The context for the formal definition of a fact's relation schema requires recalling

Definition 1 of the data warehouse schema where $S = (\mathcal{D}, \mathcal{F})$.

Definition 4 *(Fact).*

The fact relation schema F_i is represented by $F_i(A^D_{i1}, A^D_{i2}, ..., A^D_{il}, ..., A^D_{ig}, ..., A^m_{ij}, ..., A^m_{ip})$ where A^D_{il} is a foreign key references a primary key for one of the dimension relations $(1 \leq i \leq q$ and $1 \leq l \leq u)$, and $A^m_{ij} : F_i \rightarrow Q_{ij}$ is a measurement attribute of F_i $(F_i \in \mathcal{F})$ and Q_{ij} is its domain for $1 \leq i \leq q$ and $\mid A^D_{ig} \mid \leq j \leq p$. ■

The fact relation's primary key is a composite key of all the foreign keys (null values should not be allowed in these foreign keys). The measurement attribute A^m_{ij} contains a numeric value. The superscript letter m in A^m_{ij} implies that A^m_{ij} is a measurement attribute.

Figure 5.4 shows an example of a data warehouse schema for sales. The sales data warehouse schema has four dimension relations: time, store, product, and promotion. They will be denoted as follows:

Time(<u>Time_ Key</u>, Year, Quarter, Month, Day, ...)

Store(<u>Store_ Key</u>, Store_ Name, Store_ Number, Store_ Address, Store_ City, ...)

Product(<u>Product_ Key</u>, Description, Prod_ Type, Prod_ Size, Prod_ Weight, ...)

Promotion(<u>Promotion_ Key</u>, Description, Ad_ Type, Coupon_ Type, ...).

These dimension relations surround the sales fact relation. The sales fact relation contains foreign keys referencing each of the four dimensions along with three factual data values (sales amount, units sold, and cost). The sales fact relation will be

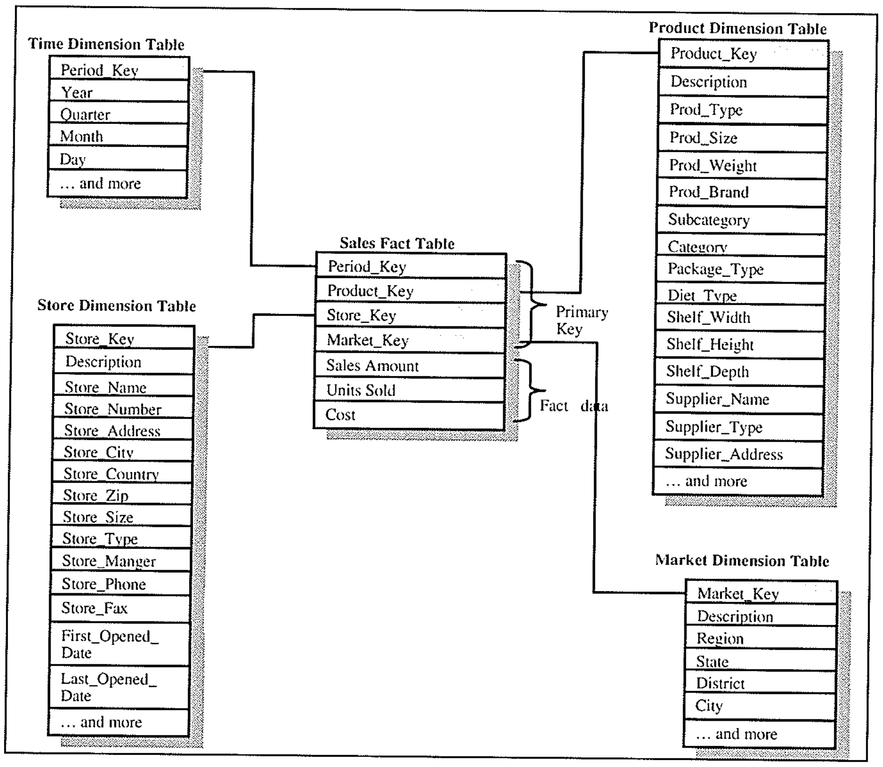


Figure 5.4: A simple data warehouse schema for sales [Kim96].

denoted as follows:

Sales(Time_ Key, Product_ Key, Store_ Key, Promotion_ Key, Sales Amount, Units Sold, Cost).

5.1.3 The Relations Size

The size of the dimension relations is smaller than the size of the fact relations [Kim96, MHP99]. The size of the dimension relations could be thousands of rows while the fact relation could be millions of rows. For example, suppose that the sales history in a sales data warehouse holds data for three years. The number of rows of the time dimension relation would be (3 $Years * 365$ $Days = 1095$ $Days$ (rows)). Also suppose that there are 200 stores (rows) in the store dimension relation and 50,000 products (rows) in the product dimension relation. Let us assume that the daily sales of each store is 6000 items, then the number of rows in the sales fact relation would be (1095 Days $*$200 Stores $*$6000 active product $= 1,314,000,000$ rows). This example illustrates that the size of the fact relation can contain over 1 billion rows. Therefore, it is clear that the size(D) $\ll$ size(F).

5.2 Variation of Data Warehouse Schema Design

There are various extensions of the data warehouse schema (star schema). These various extensions are required to suit more complex modeling problems [Poe96]. The following subsections describe briefly two such extensions of the star schema: snowflake schema and multi-star schema.

5.2.1 Snowflake Schema

The snowflake schema is an extension of the star schema, where dimensional relations are normalized. The snowflake schema diagram forms a shape similar to that of a snowflake. Figure 5.5 shows an example of a snowflake schema of a data warehouse for sales. This example is similar to the one in Figure 5.4, except that here some of the dimension relations are normalized. For example, the product dimension relation is normalized by a link to the supplier relation which contains the information about

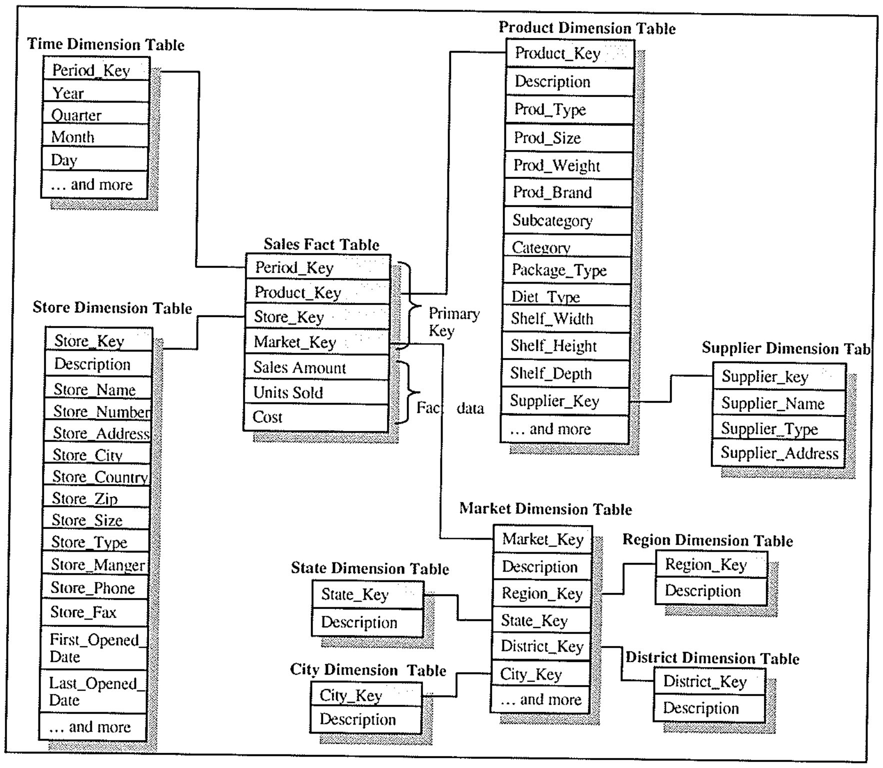


Figure 5.5: A simple snowflake schema design for sales database.

the suppliers.

The snowflake schema minimizes disk storage for the data because the denormalized dimension relation tends to be large and contains redundant information. On the other hand, the snowflake schema increases the number of relations which naturally increases the complexities of some of the queries.

5.2.2 Multi-Star Schema

The fact relation in the star schema is composed of an order set of a foreign keys that form a primary key and raw data. However, in some applications the selected

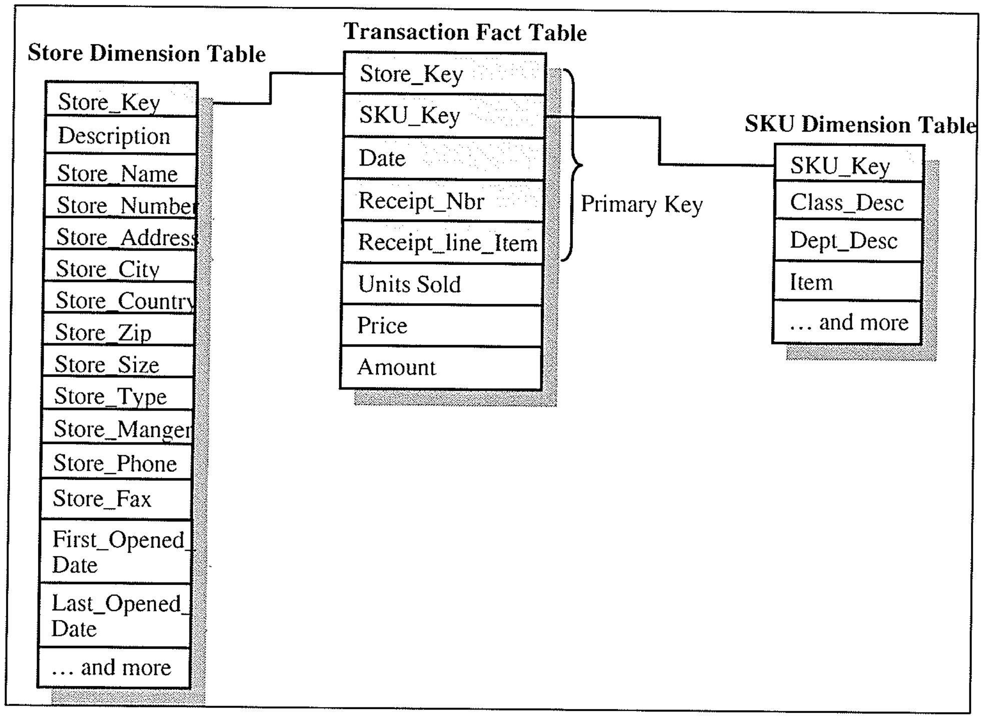


Figure 5.6: Multi-Star schema.

primary key might not represent a unique identifier for the fact relation rows. Multi-star schema solves this problem by providing a set of foreign keys that reference dimension relations, and a primary key that consists of one or more columns to identify the fact relation rows [Poe96].

Figure 5.6 shows a multi-star schema for a retail sales database. The transaction fact relation records daily sales. It is composed of two foreign keys, a primary key, and three factual data. The two foreign keys *Store_Key* and *SKU_Key*, which refer to the Store and SKU (Stock Keeping Unit) relations, can occur in multiple rows. For instance, the same store could have multiple sales of the same item on the same day. Therefore these two foreign keys are not enough to identify the fact relation row. Therefore, the primary key for the transaction fact relation consists of the three columns *Date*, *Receipt_Nbr*, and *Receipt_Line_Item*. This type of relational design has been introduced to support the on-line analytical processing.

5.3 Discussion and Summary

From the above description of the data warehouse schema model, we could say that the data warehouse schema is a particular implementation of the relational data model. The fact relation represents the associative entity that links the instances of the various dimensions. The associative entity is an entity that associates the instances of one or more entities and contains attributes that are peculiar to the

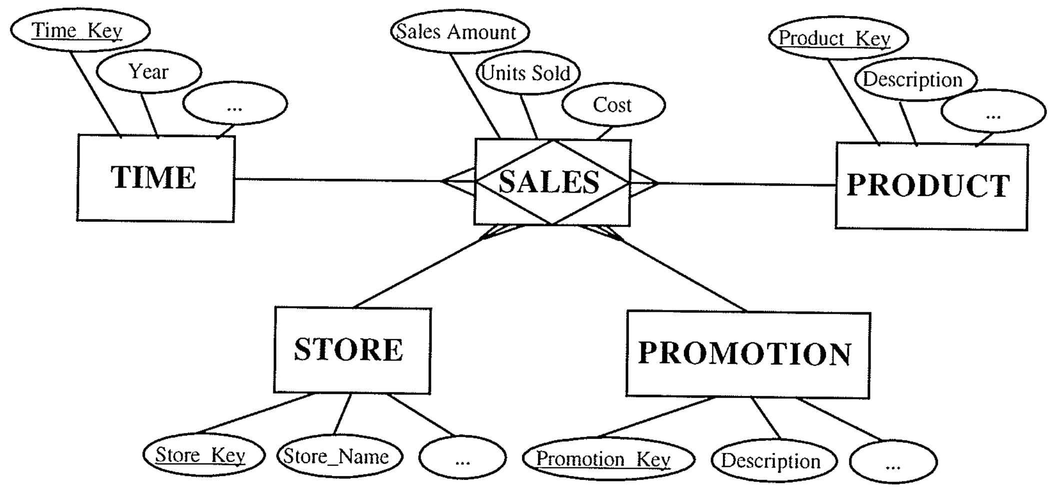


Figure 5.7: The entity relationship digram for the sales data warehouse.

relationship between those entity instances [MHP99].

Figure 5.7 shows the entity relationship diagram for the sales data warehouse. The associative entity SALES is represented with diamond relationship symbol enclosed within the entity box. The associative entity SALES forms an association between instances of the TIME, STORE, PRODUCT, and PROMOTION entities. There are three attributes on the associative entity SALES: Sales Amount, Units Sold, and Cost. The identifier of the associative entity SALES is a composite identifier whose components will consist of the identifier of the TIME, STORE, PRODUCT, and PROMOTION entities.

The dimension relation in the data warehouse schema (star schema) is denormalized. The main idea behind denormalization is to improve database performance for on-line analytical processing applications.

Snowflake schema same as star schema, but with normalization of dimension relations. The Multi-Star schema is the same as the star schema where the fact relation may contain a partial key.

Chapter 6

The Distributed Data Warehouse Design

This chapter presents a methodology for the distributed data warehouse design and two approaches to fragment the huge fact relation horizontally. Section 6.1 addresses the main idea of the design methodology. Section 6.2 presents the first approach that partitions the fact relation in an algorithmic way along with its time complexity and an example. In the same manner, Section 6.3 presents the second approach. Finally, Section 6.4 validates the generated fragments from the fact relation with respect to the three correctness rules of fragmentation: *completeness*, *reconstruction* and *disjointedness*.

6.1 The Design Methodology

This section presents a design methodology for distributed data warehouse. The main ideas of the methodology are (1) to replicate the dimension relations across the network and (2) to generate horizontal fragments of the huge fact relation only. The reasons are:

- The sizes of the dimension relations are relatively small compared to the fact relations (see Section 5.1.3 for details).
- The dimension relations are changing slowly. Therefore, the cost of updating these replicas is relatively low.
- If the dimension relations are fragmented and allocated across the network, query processing will be costly because the DSS user queries usually run against the fact relation and its dimension relations. Consider, for example, the following user query "*Give the total sales amounts for each product in each quarter of 1997 in each region*". To answer this query, the data from product, time and store dimensions should be joined with the sales fact relation. Then the total sales would be determined. Therefore, the improved performance query processing is realized by replicating the dimension relations.

Since the dimension relations will be replicated across the network, the main objective of the proposed algorithm is to generate horizontal fragments of the huge fact relation.

There are two approaches for horizontal fragmentation: primary and derived. Applying the primary horizontal fragmentation requires a set of applications (simple predicates) used by user queries against the relation.

In the data warehouse, the fact relation represents the numerical measurements of the organization and the DSS user queries perform arithmetic operations on the fact relation such as summarization, aggregation, average and so forth.

It is unlikely that a set of simple predicates for the fact relation could be determined from the DSS user queries. Therefore, the primary horizontal fragmentation approach could not be applied on the fact relation.

The other approach is to derive the horizontal fragments of the fact relation from the applications that are defined on a dimension relation. In the data warehouse schema, the fact relation is owned by more than one dimension relation (owner) and represents the many-to-many relationship among the dimension relations. For example, in Figure 6.1 the many-to-many relationship between the dimension relations (Store, Product, Time) is expressed with three links (L_1, L_2, L_3) to the sales fact relation (member).

The two functions: *Owner* and *Member*[1] defined in [ÖV91] have an analog in the data warehouse schema. This analog is that the *Owner* function always maps the given link to a dimension relation where the *Member* function always maps the link set $\mathcal{L}$ (where $\mathcal{L} = \{L_1, L_2, ..., L_k, ..., L_n\}$) to the fact relation in the data warehouse schema as follows:

[1] They provide mapping from the set of links to the set of relations.

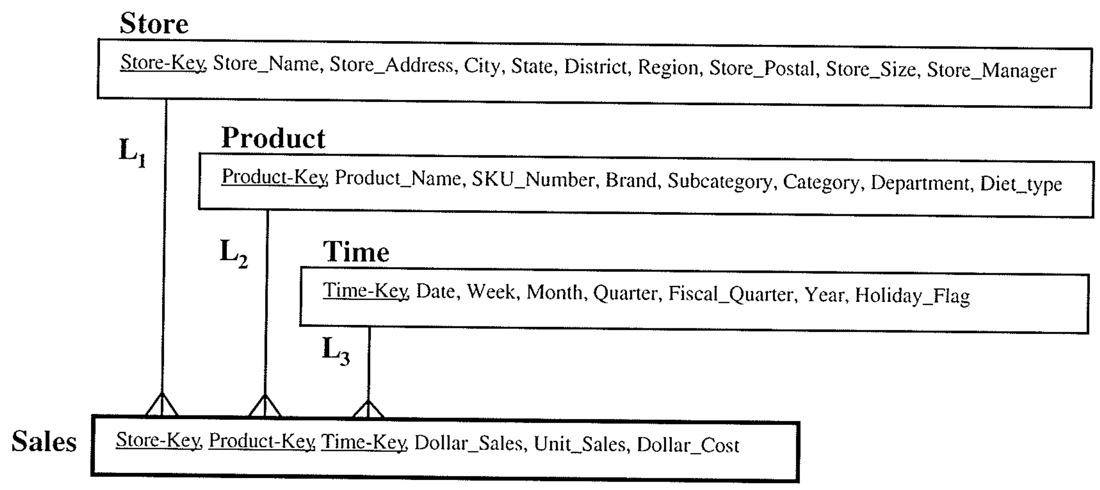

Store
Store-Key, Store_Name, Store_Address, City, State, District, Region, Store_Postal, Store_Size, Store_Manager
L1
Product
Product-Key, Product_Name, SKU_Number, Brand, Subcategory, Category, Department, Diet_type
L2
Time
Time-Key, Date, Week, Month, Quarter, Fiscal_Quarter, Year, Holiday_Flag
L3
Sales
Store-Key, Product-Key, Time-Key, Dollar_Sales, Unit_Sales, Dollar_Cost

Figure 6.1: A simple data warehouse schema foe sales.

- $Owner(L_k) = D_i$ where L_k is one of the links in the link set $\mathcal{L}$ and $D_i \in \mathcal{D}$ is a dimension relation.

- $Member(L_k) = F_i$ where L_k is one of the links in the link set $\mathcal{L}$ and $F_i \in \mathcal{F}$ is a fact relation.

For example, in Figure 6.1 the output values of the $Owner$ and $Member$ functions of the link L_1 are $Owner(L_1) = Store$ and $Member(L_1) = Sales$. Figure 6.1 also shows that the $Member$ function maps the links (L_1, L_2 and L_3) to the sales fact relation.

Since the fact relation is owned by more than one dimension relations, this thesis presents two new approaches for deriving the horizontal fragments of the fact relation:

The **first approach** derives the horizontal fragments of the fact relation from the applications that are defined on only one dimension relation. This dimension relation (owner) has to be selected among the other dimension relations. There are two criteria that could be used to select the owner relation for performing derived horizontal fragments on the fact relation. These two criteria have been presented in [ÖV91] but the methodology for how to implement them has not yet appeared. These two criteria are to select:

- the relation fragments with better join characteristic, or
- the relation fragments used in more applications.

The objectives of the first criterion are to choose the relation fragments that provide answers to the query by performing joins on smaller relations (i.e., fragments) and by performing joins between fragments in one site which execute the query in parallel. This criterion could not be applied to generate the horizontal fragments of the fact relation because of the basic prior assumption to replicate the dimension relations. Therefore, there is no benefit in choosing the dimension (owner) relation that provides better join characteristics.

The objective of the second criterion is to choose the relation fragments that are used in more applications. The derived fragments of the fact relation will represent the segments of the fact relations that are heavily accessed by the DSS users. This criterion, therefore, seems to be the preferable choice. The methodology for implementing it will be presented below in Section 6.2. The selected dimension relation, according to this criterion, will be used temporarily. It still has to be replicated with the other dimension relations though.

Unfortunately, the drawback of this approach is that the generated fragments do not reflect all the user applications against the data warehouse because they are based on the applications of one dimension relation.

The **second approach** alleviates the drawback of the first approach by deriving the horizontal fragments of the fact relation from the applications of all the dimension relations that have owner links with the fact relation. Section 6.3 presents an algorithm for implementing this approach.

6.2 The First Approach

This section presents an algorithm for implementing the first approach. The idea behind this approach is to derive the horizontal fragments of the fact relation based on the applications of one selected dimension relation.

The algorithm consists of three major steps to derive the horizontal fragments of the fact relations. These steps are:

1. Select the dimension relation that has the highest access frequency among the dimension relations.

2. Optimize the set of simple predicates for the selected dimension relation by eliminating the predicates that have a lower position in the attributes hierarchy.

3. Derive the horizontal fragments of the fact relation based on the minterm predicates of the selected dimension relation.

This algorithm is different from the relational distributed database [ÖV91] because (1) it uses access frequencies of queries to measure the importance of the dimension relations; and (2) it refines the set of simple predicates to support the requirements of the OLAP applications.

Algorithm 4 Fact Horizontal Fragments Approach One (*FHF_Approach1*) provides a formal presentation of these steps. It is assumed that all the input to the horizontal fragmentation algorithm of the fact relation is determined during the requirement analysis phase of the top-down design approach (see Section 3.3). The input to the algorithm are (1) the set of the dimension relations ($Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$) that have applications and owner links with the fact relation F_i where $Dset \subseteq \mathcal{D}$; (2) the attributes hierarchy relation $D_{hk}(AH, Level)$ for each dimension relation in the $Dset$ (see Section 5.1.1); (3) the dimension relations predicates set ($DPset = \{P_{D_1}, P_{D_2}, ..., P_{D_k}, ..., P_{D_S}\}$); (4) the dimension relations DSS user queries set $DQset = \{Q_{D_1}, Q_{D_2}, ..., Q_{D_k}, ..., Q_{D_S}\}$ where $Q_{D_k} = \{q_1, q_2, ..., q_v\}$ and their access frequencies; (5) the set of semijoin predicates between the dimension relations (owners) and the fact relation (member) ($SemiPset = \{P_1, P_2, ..., P_k, ..., P_s\}$);

and (6) the fact relation F_i. The output of the algorithm is the set of the horizontal fragments for the fact relation. A detailed description of each step of the algorithm is given below.

Algorithm 1 SelectDimension

Input:

- The set of the dimension relations $Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$.
- The dimension relations predicates set $DPset = \{P_{D_1}, P_{D_2}, ..., P_{D_k}, ..., P_{D_S}\}$ where $P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$.
- The dimension relations DSS user queries set $DQset = \{Q_{D_1}, Q_{D_2}, ..., Q_{D_k}, ..., Q_{D_S}\}$ where $Q_{D_k} = \{q_1, q_2, ..., q_v\}$ and their access frequencies.

Output:

- D_k the dimension relation that has the largest access frequency on its simple predicates.

```
Begin
(1)     for each $P_{D_k} \in DPset$ AND $Q_{D_k} \in DQset$ do
(2)     // Compute the total access frequency ($TAF_{D_k}$) of the dimension $D_k$
(3)     for each $q_i \in Q_{D_k}$ do
(4)         for each $p_j^{D_k} \in P_{D_k}$ do
(5)             $TAF_{D_k} \longleftarrow TAF_{D_k} + (use(q_i, p_j^{D_k}) * acc(q_i))$
(6)         end {end for $p_j^{D_k}$}
(7)     end {end for $q_i$}
(8)     // Insert the $TAF_{D_k}$ into the dimensions access frequency ($DAF$) set
(9)     $DAF \longleftarrow DAF \cup TAF_{D_k}$
(10)    end {end for $P_{D_k}$ and $Q_{D_k}$}
(11)    // Get the $D_k$ that has the largest $TAF_{D_k}$ from the $DAF$ set
(12)    $D_k \longleftarrow$ GetMaxDAF($DAF$)
(13)    Return($D_k$)
End {SelectDimension}
```

Step One-*Select the dimension relation that has the highest access frequency among the dimension relations.*

The application information is required to process this step. There are two types of user application information defined on the relation: qualitative and quantitative information. The *qualitative* information of user applications, defined on relation R_i, is represented by a set of simple predicates used in user queries against R_i. This set will generate a set of optimal minterm fragments $M_i = \{m_{i1}, m_{i2}, ..., m_{in}\}$ for fragmenting the relation R_i (where the relation R_i is an owner relation).

In the data warehouse schema, by using the DSS user queries, a set of simple predicates will be identified for each dimension relation involved in the queries. Let the set $P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$ represents the applications defined on the dimension relation D_k. The simple predicate $p_j^{D_k}$ in the P_{D_k} set is defined as: $p_j^{D_k} : A_{ki}$ θ $Value$ where $p_j^{D_k}$ is the j^{th} predicate of the P_{D_k} simple predicates set, A_{ki} is the i^{th} attribute of the dimension relation D_k, θ is a relation operation from the set $\{=, <, \neq, \leq, >, \geq\}$, and $Value$ is a value from the domain Q_{ki} of A_{ki} (i.e., $Value \in Q_{ki}$).

The *quantitative* information is essential for allocation [ÖV91]. Quantitative information is represented by two sets of data: minterm selectivity and access frequency. The *minterm selectivity* is the cardinality of the minterm predicate in the relation and the *access frequency* is the number of times the user application accesses data in a given period. The access frequency is used in Step One.

The objective of this step is to select the dimension relation that has the highest access frequency among the dimension relations. Algorithm 1 (*SelectDimension*) gives a formal presentation of this step. The input to the algorithm are the set of the dimension relations $Dset$, the dimension relations predicates set $DPset$, and the dimension relations DSS user queries set $DQset$. Three phases are required to process this step:

Phase one generates the *predicate usage matrix* for each dimension relation. It is assumed that a procedure that generates the predicate usage matrix is available to this algorithm. The predicate usage matrix represents the use of predicates in the queries. The predicate usage matrix elements defined by the predicate usage value.

Definition 5 *(Predicate Usage Value):*

Let $Q_{D_k} = \{q_1, q_2, ..., q_v\}$ *be the set of the DSS user queries that will use the set of predicates* $P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$*. This set of predicates is defined on the dimension relation* D_k*. A value of 1 or 0 is associated with each query* q_i *and each simple predicate* $p_j^{D_k}$*. This value is called the predicate usage value and is defined as follows:*

$$use(q_i, p_j^{D_k}) = \left\{ \begin{array}{ll} 1 & \textit{if predicate } p_j^{D_k} \textit{ is referenced by query } q_i \\ 0 & \textit{otherwise} \end{array} \right\} \quad \blacksquare$$

Phase two calculates the Total Access Frequency (TAF_{D_k}) of the predicates in the dimension relation (lines 1-10). This is done by multiplying the query access frequency $acc(q_i)$ by the predicate usage value $use(q_i, p_j^{D_k})$. The result of this multiplication will be summed in the TAF_{D_k} as follows:

$$TAF_{D_k} = \sum_{q_i \in Q_{D_k}} \sum_{p_j^{D_k} \in P_{D_k}} (use(q_i, p_j^{D_k}) * acc(q_i))$$

Each TAF_{D_k} will be inserted into a set called the Dimension Access Frequency (DAF) set. **Phase three** selects the dimension relation that has the largest total access frequency (TAF_{D_k}) from the DAF set (lines 11-12). It is assumed that the procedure GetMaxDAF( ) is available to this algorithm.

Step Two- *Optimize the set of simple predicates for the selected dimension relation* D_k.

The dimension relation has a hierarchical structure among a subset of its attributes (as discussed in Section 5.1.1). Each dimension relation is associated with the attributes hierarchy relation that shows the level of each attribute in the hierarchy. The set of simple predicates P_{D_k} of the dimension relation could contain hierarchical predicates. The *hierarchical predicate* is the predicate whose attribute is involved in the attributes hierarchy of the dimension relation.

Step Two eliminates the hierarchial predicate from the P_{D_k} set, if and only if there is another hierarchial predicate in the P_{D_k} set with a higher level. To perform this elimination, the following two phases are required (Algorithm 2 (*OptimizeDimension*) provides a formal presentation of these phases):

Algorithm 2 OptimizeDimension

Input:

- the set of simple predicates of the selected dimension relation $P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$.
- the attributes hierarchy relation $D_{hk}(AH, Level)$ of the selected dimension relation.

Output:

- P_{D_k} an optimize set of simple predicates for the D_k relation.

```
Begin
(1)   // Identify the level of the highest attribute in $P_{D_k}$
(2)   $HighestLevel = 0$
(3)   for each $p_j^{D_k} \in P_{D_k}$ do
(4)      if $A_{ki}$ of $p_j^{D_k}$ exist in $D_{hk}.AH$ then
(5)         if $D_{hk}.Level \geq HighestLevel$ then
(6)            $HighestLevel = D_{hk}.Level$
(7)   end {end for $p_j^{D_k}$}

(8)   // Remove the predicates that have a lower level than the $HighestLevel$
(9)   for each $p_j^{D_k} \in P_{D_k}$ do
(10)     if $A_{ki}$ of $p_j^{D_k}$ exist in $D_{hk}.AH$ then
(11)        if $D_{hk}.Level < HighestLevel$ then
(12)           $P_{D_k} \longleftarrow P_{D_k} - p_j^{D_k}$
(13)  end {end for $p_j^{D_k}$}
(14)  Return($P_{D_k}$)
End {OptimizeDimension}
```

1. Identify the attribute that has the highest level among the hierarchical predicates (see algorithm 2 lines 1-7).

2. Remove from the predicate set ($P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$) all the predicates whose attributes have a lower level than the highest attribute identified

in the previous phase (see algorithm 2 lines 8-13).

Consider, for example, the store dimension relation that has the following set of simple predicates P_{Store} = {City = "New York", City = "Boston", City = "Miami", Region = "Eastern", Region = "South East"}. Consider the attributes hierarchy relation ($D_{hStore}(AH, Level)$) associated with the store dimension relation below:

D_{hStore}

AH	**Level**
Store	1
City	2
State	3
District	4
Region	5

From the set of simple predicates contained in P_{Store} and the attributes hierarchy relation D_{hStore}, the attribute that has the highest level among the hierarchical predicates is "Region". Therefore, the set of simple predicates include only the following predicates P_{Store} = {Region = "Eastern", Region = "South East"}.

The advantages of this optimization is the derived horizontal fragments of the fact relation drawn from the minterm predicates of the dimension relation. They (1) allow the *roll-up* and *drill-down* operations of the OLAP applications to be performed in the same fragment; and (2) reduce the number of *join* operations among the fragments.

The disadvantage of this optimization is that the number of the generated fragments is small. However, they are large in terms of size. The large size of fragments may not be desirable if storage is limited in the distributed sites. Therefore, we believe that optimizing the set of simple predicates is an optional step in the horizontal fragmentation design. The decision of whether to optimize or not will be based on the properties of the computer systems at each site. In this way, fragmentation is not wholly orthogonal to allocation.

Step Three-*Derive the horizontal fragments of the fact relation.*

The objectives of this step are to:

1. generate a set of minterm predicates ($M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_j^{D_k}, ..., m_z^{D_k}\}$) from the set of simple predicates P_{D_k} that has been refined in Step Two; and

2. derive the horizontal fragments of the fact relation ($f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$).

The formal presentation of this step is given in Algorithm 4 (*FHF_Approach1).* Three phases are required to process this step:

Phase One: Apply the COM_MIN algorithm [ÖV91] (see line 5 of Algorithm 4). The objective of this algorithm is to generate a complete and minimal set of predicates $P'_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$ from the optimized set of simple predicates P_{D_k}. The COM_MIN algorithm includes, in the set of predicates P'_{D_k}, those predicates that partition the relation or fragment into at least two parts which are accessed differently by at least one application.

Phase Two: Generate the set of dimension minterm predicates M_{D_k}.

Algorithm 3, *DimenMintermPredicates*, provides a formal presentation of the steps required for generation of the M_{D_k} set ($M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_j^{D_k}, ..., m_z^{D_k}\}$). These steps are:

- *Group the simple predicates in sets according to their attributes* (lines 1-13). Suppose the set $P'_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$ has simple predicates[2] which belong to two different attributes A_{ki} and A_{kj} of the dimension relation D_k, the P'_{D_k} set will, therefore, be subdivided into $P_{D_k}^{A_{ki}} = \{p_1^{A_{ki}}, p_2^{A_{ki}}, ..., p_s^{A_{ki}}\}$ and $P_{D_k}^{A_{kj}} = \{p_1^{A_{kj}}, p_2^{A_{kj}}, ..., p_w^{A_{kj}}\}$.

- *Generate the minterm predicates* (lines 14-37). There are two cases for generating the minterm predicates:

  Case 1: If the P'_{D_k} set is grouped to one set, i.e. the simple predicates in P'_{D_k} set belong to the same attribute, the set of minterm predicates will be the elements of the P'_{D_k} set. The minterm predicates set is defined as follows: $M_{D_k} = \{m_j^{D_k} | m_j^{D_k} = p_j^{D_k}\}$ where $1 \leq k \leq |\mathcal{D}|$ ($|\mathcal{D}|$ is the number of dimension relations in $\mathcal{D}$ set) and $1 \leq j \leq |P'_{D_k}|$ ($|P'_{D_k}|$ is the number of simple predicates in the P'_{D_k} set).

[2] Recall that the simple predicate defined as $p_j^{D_k} : A_{ki}\ \theta\ Value$.

Algorithm 3 DimenMintermPredicates

Input: The set of simple predicates of the dimension relation $P'_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_n^{D_k}\}$.
Output: The set of dimension minterm predicates $M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_j^{D_k}, ..., m_z^{D_k}\}$
Declare
m : is a temporary matrix.
$P_{D_k}^{A_{kz}}$: declare temporary sets. Each set $P_{D_k}^{A_{kz}}$ represents the predicates of a specific attribute A_{kz}. The maximum number of these sets is r where r is the number of the attributes of the D_k relation.
M_{D_k} : is the set of dimension minterm predicates.
Begin

```
(1)  //Group the simple predicates in sets according to their attributes.
(2)  $r = 0$;    $size = |P'_{D_k}|$
(3)  do
(4)      $r = r + 1$
(5)      $CurrentAttribute = A_{k\ell}$ of $p_1^{D_k}$
(6)      for each $p_j^{D_k} \in P'_{D_k}$ do
(7)          if $A_{k\ell}$ of $p_j^{D_k} = CurrentAttribute$ then
(8)              $P_{D_k}^{A_{kr}} \longleftarrow P_{D_k}^{A_{kr}} \cup p_j^{D_k}$
(9)              $P'_{D_k} \longleftarrow P'_{D_k} - p_j^{D_k}$
(10)             $size = size - 1$
(11)         end {end if}
(12)     end {end for $p_j^{D_k}$}
(13) while $size > 0$
(14) //Generate the minterm predicates for the first set (case 1).
(15) for $j = 1$ to $|P_{D_k}^{A_{k1}}|$ do
(16)     $M_{D_k} \longleftarrow M_{D_k} \cup (p_j^{A_{k1}} \in P_{D_k}^{A_{k1}})$
(17) end {end for $j$}
(18) $q = |P_{D_k}^{A_{k1}}|$
(19) for $z = 2$ to $r$ do
(20)     //Conjunct the simple predicates of $P_{D_k}^{A_{kz}}$ with the previous elements of $M_{D_k}$ (case 2).
(21)     for $i = 1$ to $q$ do
(22)         for $j = 1$ to $|P_{D_k}^{A_{kz}}|$ do
(23)             $m_{ij} = (m_i^{D_k} \in M_{D_k}) \wedge (p_j^{A_{kz}} \in P_{D_k}^{A_{kz}})$
(24)         end {end for $j$}
(25)     end {end for $i$}
(26)     //Remove the old minterm predicates from $M_{D_k}$.
(27)     for $i = 1$ to $q$ do
(28)         $M_{D_k} \longleftarrow M_{D_k} - m_i^{D_k}$
(29)     end {end for $m^{D_k}$}
(30)     //Add the new minterm predicates to $M_{D_k}$.
(31)     for $i = 1$ to $q$ do
(32)         for $j = 1$ to $|P_{D_k}^{A_{kz}}|$ do
(33)             $M_{D_k} \longleftarrow M_{D_k} \cup m_{ij}$
(34)         end {end for $j$}
(35)     end {end for $i$}
(36)     $q = q * |P_{D_k}^{A_{kz}}|$
(37) end {end for $z$}
End {DimenMintermPredicates}
```

Case 2: If the P'_{D_k} set is grouped to more than one set, the minterm predicates will be generated by the conjunction among the simple predicates of the grouped sets. The number of minterm predicates in the M_{D_k} set is $z = |P_{D_k}^{A_{k1}}| * \ldots * |P_{D_k}^{A_{ki}}| * \ldots * |P_{D_k}^{A_{kr}}|$. The set of simple predicates is defined as follows:

$$M_{D_k} = \{m_v^{D_k} | m_v^{D_k} = \bigwedge_{p_j^{A_{ki}} \in P_{D_k}^{A_{ki}}} p_j^{A_{ki}}\}$$

where:

- $1 \leq k \leq |\mathcal{D}|$ ($|\mathcal{D}|$ is the number of dimension relations in $\mathcal{D}$ set),

  $1 \leq v \leq z$ (z is the number of minterm predicates defined on D_k),

  $1 \leq j \leq |P_{D_k}^{A_{ki}}|$ ($|P_{D_k}^{A_{ki}}|$ is the number of simple predicates in the $P_{D_k}^{A_{ki}}$ set) and

  $1 \leq i \leq r$ (r is the number of simple predicates sets that have different attributes).

The method of generating the minterm predicates (shown above) is different from the one presented by Özsu and Valduriez [ÖV91]. They defined the minterm predicates as $M = \{m_i | m_i = \bigwedge_{p_j \in P} p_j^*\}$ where $p_j^* = p_j$ or $p_j^* = \neg p_j$. This research does not consider the negated form of simple predicates ($p_j^* = \neg p_j$) in generating the minterm predicates. The reasons are:

- The **COM_MIN** algorithm ensures that the simple predicates are complete and minimal.

- Even if we consider the negated form of the simple predicates, we will end up with minterm predicates that did not have any simple predicates in the negated form.

Phase three: Derive the horizontal fragments set of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$ (see lines 7-11 of Algorithm 4). The derived horizontal fragments of the fact relation (member) are based on the dimension minterm predicates defined on the selected dimension relation D_k that has an owner links with the fact relation. Given (1) a link l_j (where $owner(l_j) = D_k$ and $member(l_j) = F_i$); (2) the selected dimension relation D_k along with its set of the minterm predicates $M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_j^{D_k}, ..., m_z^{D_k}\}$; (3) the fact relation F_i; and (4) the semi-join predicate ($p : D_k.key = F_i.key$) between the dimension relation (owner) and the fact relation (member), the derived horizontal fragments of F_i are defined as $F_i^j = F_i \ltimes (\sigma_{m_j^{D_k}}(D_k))$ where:

- $1 \leq i \leq q$ (q is the number of fact relations in $\mathcal{F}$ set);
- $1 \leq k \leq n$ (n is the number of dimension relations in $\mathcal{D}$ set); and
- $1 \leq j \leq z$ (z is the number of minterm predicates defined on D_k relation).

Algorithm 4 FHF_Approach1

Input:

- The set of the dimension relations $Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$ that have applications and owner links with the fact relation F_i where $Dset \subseteq \mathcal{D}$.
- The attributes hierarchy relation $D_{hk}(AH, Level)$ for each dimension relation in the $Dset$.
- The dimension relations predicates set $DPset = \{P_{D_1}, P_{D_2}, ..., P_{D_k}, ..., P_{D_S}\}$.
- The dimension relations DSS user queries set $DQset = \{Q_{D_1}, Q_{D_2}, ..., Q_{D_k}, ..., Q_{D_S}\}$ where $Q_{D_k} = \{q_1, q_2, ..., q_n\}$ and their access frequencies.
- The set of semijoin predicates between the dimension relations (owners) and the fact relation (member) $SemiPset = \{P_1, P_2, ..., P_k, ..., P_s\}$.
- The fact relation F_i.

Output:

- A set of derived horizontal fragments of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$.

```
Begin
(1)    SelectDimension($Dset, DPset, DQset, D_k$)
(2)    // Optimize the selected dimension relation $D_k$
(3)    OptimizeDimension($P_{D_k}, D_{hk}$)
(4)    // Generate the dimension minterm predicates
(5)    $P'_{D_k} \longleftarrow$ COM_MIN($D_k, P_{D_k}$)
(6)    $M_{D_k} \longleftarrow$ DimenMintermPredicates($P'_{D_k}$)
(7)    // Derive the horizontal fragments of the fact relation
(8)    for each $m_j^{D_k} \in M_{D_k}$ do
(9)        $F_i^j \longleftarrow F_i \ltimes (\sigma_{m_j^{D_k}}(D_k))$
(10)       $f_{F_i} \longleftarrow f_{F_i} \cup F_i^j$
(11)   end {end for $m_j^{D_k}$}
End {FHF_Approach1}
```

6.2.1 The Algorithm Complexity

This section presents the computation of the Fact Horizontal Fragmentation Algorithm 4 (*FHF_Approach1*). Assume s represents the largest number of dimension relations that have applications and owner links with the fact relation, n represents the largest number of simple predicates in a dimension relation, v represents the largest number of queries against one of the dimension relation, r represents the number of simple predicates sets that have been grouped out from the n simple predicates of the selected dimension relation, z is the number of minterm predicates defined on the selected dimension relation.

The horizontal fragmentation algorithm of the fact relation *(FHF_ Approach1)* composed of four algorithms. The time complexity of these algorithms are: (1) the (*SelectDimension*) algorithm is $O(svn + s)$; (2) the (*OptimizeDimension*) algorithm is $O(2n)$; (3) the (*COM_ MIN*) algorithm is $O(n + n^2)$; and (4) the (*DimenMintermPredicates*) algorithm is $O(n^2 + n^r)$. Therefore, the time complexity of the *(FHF_ Approach1)* is $O(svn + n^2 + n^r + z)$. The time complexity of the algorithm can be simplified to $O(n^r)$. Therefore, the algorithm is exponential in the number of simple predicates sets (r) that have been grouped out from the n simple predicates of the selected dimension relation. The time complexity of the algorithm will be reasonable if the size of r remains small.

The worst-case analysis is when r is equal m ($r = m$) where m is the number of attributes in the dimension relation ($D_i(A^D_{i1}, A_{i2}, A_{i3}, ..., A_{ij}, ..., A_{im})$). The algorithm will not reach to the worst-case because the (*OptimizeDimension*) algorithm eliminates the hierarchal predicate from the simple predicates set. Thus, this will reduce the number of the simple predicates sets (r). Therefore, r is less than m ($r < m$).

The time complexity of the algorithm is costly because of the characteristic of the input values. However, time is not a critical issue in this type of design because the fragmentation algorithm addresses a solution to a static fragmentation design problem (i.e., the fragmentation algorithm will run off line). Any fragmentation algorithm is beneficial only if it generates optimal fragments that reduce the transaction processing cost in the distributed data warehouse system.

6.2.2 An Example

Consider a simple data warehouse for sales with historical records of every sales transaction. Figure 6.1 shows three dimension relations: Store, Product and Time. The sales fact relation contains foreign keys referencing each of the three dimensions along with three factual data values: sales amount, units sold and cost. Figure 6.2 shows the attributes hierarchy relations associated with each dimension relations.

Consider the following applications q_1 to q_5 that will be input to the horizontal fragmentation design process:

q_1: This application requires information about the sales by city. The predicates of this application will be defined from the *Store* dimension relation as follows: $\{p_1$: City = "Atlanta", p_2 : City = "Miami", p_3 : City = "Dallas"$\}$.

q_2: This application requires information about the sales by region. The predicates of this application will be defined from the *Store* dimension relation as follows:

$\{p_4$: Region = "South East", p_5 : Region = "South West"$\}$.

q_3: This application requires information about the sales by the product subcategory. The predicates of this application will be defined from the *Product* dimension relation as follows:

$\{p_1$: Subcategory = "Frozen Foods", p_2 : Subcategory = "Candy",

p_3 : Subcategory = "Salty Snacks", p_4 : Subcategory = "Soft Drinks",

p_5 : Subcategory = "Cleaning Supplies"$\}$.

q_4: This application requires information about the sales by the product category. The predicates of this application will be defined from the *Product* dimension relation as follows:

$\{p_6$: Category = "Foods", p_7 : Category = "Drinks", p_8 : Category = "Supplies"$\}$.

q_5: This application requires information about the sales by year. The predicates of this application will be defined from the *Time* dimension relation as follows:

$\{p_1$: Year = "1997", p_2 : Year = "1998"$\}$.

These applications are issued at three sites and their total access frequencies at the three sites are $acc(q_1) = 25$, $acc(q_2) = 85$, $acc(q_3) = 30$, $acc(q_4) = 25$, $acc(q_5) = 40$.

D_{hStore}

AH	Level
Store	1
City	2
State	3
District	4
Region	5

$D_{hProduct}$

AH	Level
Product	1
Brand	2
Subcategory	3
Category	4
Department	5

D_{hTime}

AH	Level
Date	1
Week	2
Month	3
Quarter	4
Fiscal_Quarter	5
Year	6

Figure 6.2: The attributes hierarchy relations for Store, Product and Time dimension relations.

From applications q_1 and q_2 the set of simple predicates for the *Store* dimension relation is $P_{Store} = \{p_1, p_2, p_3, p_4, p_5\}$ and the user queries set is $Q_{Store} = \{q_1, q_2\}$. From applications q_3 and q_4 the set of simple predicates for the *Product* dimension relation is $P_{Product} = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$ and the user quires set is $Q_{Product} = \{q_3, q_4\}$. From application q_5 the set of simple predicates for the *Time* dimension relation is $P_{Time} = \{p_1, p_2\}$ and the user quires set is $Q_{Time} = \{q_5\}$.

The horizontal fragmentation design process:

Apply the three major steps of algorithm 4 *(FHF_ Approach1)* to the above input as follows:

Step One: Select the dimension relation that has the highest access frequency among the dimension relations.

This step will calculate the total access frequency for each dimension relation and return the highest one. By applying Algorithm 1 (*SelectDimension*) to the set of simple predicates and the set of user queries for each dimension relation, the total access frequency of the *Store* dimension relation is $TAF_{Store} = 245$. It has been calculated as follows:

- Generate the predicates usage matrix of the *Store* dimension relation

$$\begin{array}{c c} & \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{array}{c} q_1 \\ q_2 \end{array} & \left[\begin{array}{ccccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

- Multiply the queries access frequency ($acc(q_1) = 25$ and $acc(q_2) = 85$) by the predicates usage value

$$\begin{array}{c c} & \begin{array}{ccccc} p_1 & p_2 & p_3 & p_4 & p_5 \end{array} \\ \begin{array}{c} q_1 \\ q_2 \end{array} & \left[\begin{array}{ccccc} 25 & 25 & 25 & 0 & 0 \\ 0 & 0 & 0 & 85 & 85 \end{array}\right] \end{array}$$

- Sum the matrix elements. The result is $TAF_{Store} = 245$. Then insert the result to the dimension access frequency set ($DAF =< 245 >$).

The total access frequency of the *Product* dimension relation is $TAF_{Product} = 225$. It has been calculated as follows:

- Generate the predicates usage matrix of the *Product* dimension relation

$$\begin{array}{c} \\ q_3 \\ q_4 \end{array}\begin{array}{c} \begin{array}{cccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array} \\ \left[\begin{array}{cccccccc} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array}$$

- Multiply the queries access frequency ($acc(q_3) = 30$ and $acc(q_4) = 25$) by the predicates usage value

$$\begin{array}{c} \\ q_3 \\ q_4 \end{array}\begin{array}{c} \begin{array}{cccccccc} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 \end{array} \\ \left[\begin{array}{cccccccc} 30 & 30 & 30 & 30 & 30 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25 & 25 & 25 \end{array}\right] \end{array}$$

- Sum the matrix elements. The result is $TAF_{Product} = 225$. Then insert the result to the dimension access frequency set ($DAF =< 245, 225 >$).

The total access frequency of the *Time* dimension relation is $TAF_{Time} = 80$. It has been calculated as follows:

- Generate the predicates usage matrix of the Time dimension relation

$$\begin{array}{cc} & \begin{array}{cc} p_1 & p_2 \end{array} \\ q_5 & \left[\begin{array}{cc} 1 & 1 \end{array}\right] \end{array}$$

- Multiply the queries access frequency ($acc(q_5) = 40$) by the predicates usage value

$$\begin{array}{cc} & \begin{array}{cc} p_1 & p_2 \end{array} \\ q_5 & \left[\begin{array}{cc} 40 & 40 \end{array}\right] \end{array}$$

- Sum the matrix elements. The result is $TAF_{Time} = 80$. Then insert the result to the dimension access frequency set ($DAF =< 245, 225, 80 >$).

From the DAF set the *Store* dimension relation will be selected because it has the largest TAF.

Step Two: Optimize the set of simple predicates for the selected dimension relation.

From Step One the selected dimension relation is the *Store* relation. By applying Algorithm 2 (*OptimizeDimension*) to the set of simple predicates P_{Store} and the attributes hierarchy relation D_{hStore} (see Figure 6.2), the attribute with the highest level among the hierarchical predicates is "Region". Therefore, the P_{Store} set will be optimized to have the following predicates

$$P_{Store} = \{\text{Region} = \text{"South East"}, \text{Region} = \text{"South West"}\}.$$

Step Three: Derive the horizontal fragments of the fact relation.

The following processes will be performed on the output of Step Two:

- The COM_MIN algorithm will return the complete and minimal set of predicates $P'_{Store} = \{\text{Region} = \text{"South East"}, \text{Region} = \text{"South West"}\}$.

- Generate the minterm predicates. By applying Algorithm 3 (*DimenMintermPredicates*) to the set of simple predicates P'_{Store}, the following minterm predicates set $M_{Store} = \{m_1, m_2\}$ can be defined where:

  m_1 : (Region = "South East")

  m_2 : (Region = "South West")

- Derive the horizontal fragments of the *Sales* fact relation from the minterm predicates of the *Store* dimension relation as follows:

  $$Sales^1 = Sales \ltimes (\sigma_{Region=\text{``South East''}}(Store))$$

  $$Sales^2 = Sales \ltimes (\sigma_{Region=\text{``South West''}}(Store))$$

The above example shows that the *Sales* fact relation has been fragmented into two fragments. Each fragment represents the segment of the fact relation that is heavily accessed by the most requested applications of the DSS users queries.

6.3 The Second Approach

This section presents an algorithm for implementing the second approach. The main idea behind this approach is to derive the horizontal fragments of the fact relation based on the applications of all dimensional relations.

The algorithm consists of three major steps. These steps are:

1. Optimize the set of simple predicates for each dimension relation by eliminating the predicates that have a lower position in the attributes hierarchy.

2. Generate the minterm predicates for each dimension relation that has applications and owner links with the fact relation.

3. Generate the set of fact minterm predicates and derive the horizontal fragments of the fact relation from the set of fact minterm predicates.

Algorithm 6 Fact Horizontal Fragments Approach Two (*FHF_Approach2*) provides a formal presentation of these steps. It has been assumed that all the input to the fact horizontal fragmentation algorithm are determined during the requirement analysis phase of the top-down design approach (see Section 3.3). The input to the algorithm are (1) the set of the dimension relations ($Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$) that have applications and owner links with the fact relation F_i where $Dset \subseteq \mathcal{D}$; (2) the attributes hierarchy relation $D_{hk}(AH, Level)$ for each dimension relation in the $Dset$; (3) the dimension relations predicates set ($DPset = \{P_{D_1}, P_{D_2}, ..., P_{D_k}, ..., P_{D_S}\}$); (4) the set of semijoin predicates between the dimension relations (owners) and the fact relation (member) ($SemiPset = \{P_1, P_2, ..., P_k, ..., P_s\}$); and (5) the fact relation F_i. The output of the algorithm is the set of the horizontal fragmentation for the fact relation.

Step One-*Optimize the set of simple predicates for each dimension relation D_k.*

The objective of this step is to eliminate the hierarchial predicate from the set of simple predicates P_{D_k} set, if and only if there is another hierarchial predicate in the P_{D_k} set with a higher level.

The detail description of this step has been presented in Section 6.2 (Step Two). Algorithm 2 (*OptimizeDimension*) provides a formal presentation of this step.

Step Two-*Determine the minterm predicates for each dimension relation.*

The objective of this step is to generate a set of minterm predicates $M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_v^{D_k}, ..., m_z^{D_k}\}$ for each dimension relation that has applications on it (i.e., set of simple predicates P_{D_k}) and owner link with the fact relation. The determined dimension minterm predicates sets will be collected into a set called the $DMset$ where the $DMset = \cup_{i=1}^{s} M_{D_i}$, ($s$ is the number of dimension relations that has applications and owner links with the fact relation F_i).

The detail description of this step has been presented in Section 6.2 (Step Three-Phase One and Two). The COM_MIN algorithm and Algorithm 3 (*DimenMintermPredicates*) provide a formal presentation of this step.

Step Three-*Determine the set of fact minterm predicates and derive the fact horizontal fragments.*

There are two phases required to process this step:

1. Generate the set of fact minterm predicates $M_F = \{m_{f_1}, m_{f_2,}..., m_{f_j}, ..., m_{f_t}\}$.

   The fact minterm predicate m_{f_j} represents the conjunction of the dimensions minterm predicates in the $DMset$ that have been generated from Step Two. The number of fact minterm predicates in the M_F set is $t = |M_{D_1}| * ... * |M_{D_k}| * ... * |M_{D_s}|$. The set of fact minterm predicates is defined as

$$M_F = \{m_{f_j} | m_{f_j} = \bigwedge_{m_v^{D_k} \in M_{D_k}} m_v^{D_k}\}$$

where:

- $1 \leq j \leq t$ (t is the number of fact minterm predicates defined on all dimension relations in the $Dset$),
- $1 \leq v \leq z$ (z is the number of the dimension minterm predicates in M_{D_k}) and
- $1 \leq k \leq s$ (s is the number of dimension minterm predicates sets in $DMset$).

Consider, for example, the following dimension minterm predicates generated from Step Two ($M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_v^{D_k}, ..., m_z^{D_k}\}$, $M_{D_l} = \{m_1^{D_l}, m_2^{D_l}, ..., m_r^{D_l}, ..., m_y^{D_l}\}$, $M_{D_u} = \{m_1^{D_u}, m_2^{D_u}, ..., m_s^{D_u}, ..., m_w^{D_u}\}$) and the set that unite them is $DMset = \{M_{D_k}, M_{D_l}, M_{D_u}\}$. As a result, the set of fact minterm predicates will be $M_F = \{m_{f_1}, m_{f_2}, ..., m_{f_j}, ..., m_{f_t}\}$ where for instance, $m_{f_1} = m_1^{D_k} \wedge m_1^{D_l} \wedge m_1^{D_u}, m_{f_2} = m_1^{D_k} \wedge m_1^{D_l} \wedge m_2^{D_u}, m_{f_j} = m_v^{D_k} \wedge m_r^{D_l} \wedge m_s^{D_u}, m_{f_t} = m_z^{D_k} \wedge m_y^{D_l} \wedge m_w^{D_u}$. Algorithm 5 (*FactMintermPredicates*) provides a formal presentation of this phase.

```
Algorithm 5 FactMintermPredicates
Input: The set of all dimension minterm predicates $DMset$.
Output: A set of fact minterm predicates $M_F = \{m_{f_1}, m_{f_2}, ..., m_{f_j}, ..., m_{f_t}\}$.
Declare
$fm$ : is a temporary matrix for conjuncting the dimension minterm predicates.
$M_F$ : is the set of fact minterm predicates.
Begin
    //Initialize the $M_F$ by the first dimension minterm predicates set.
    for $j = 1$ to $|M_{D_1}|$ do
        $M_F \longleftarrow M_F \cup (m_j \in M_{D_1})$
    end {end for $j$}
    $r = |DMset|$
    $q = |M_{D_1}|$
    for $z = 2$ to $r$ do
        //Conjunct the minterm predicates of $M_{D_z}$ with the previous elements of $M_F$.
        for $i = 1$ to $q$ do
            for $j = 1$ to $|M_{D_z}|$ do
                $fm_{ij} = (m_{f_i} \in M_F) \wedge (m_j \in M_{D_z})$
            end {end for $j$}
        end {end for $i$}
        //Remove the old fact minterm predicates from $M_F$.
        for $i = 1$ to $q$ do
            $M_F \longleftarrow M_F - m_{f_i}$
        end {end for $m^{D_k}$}
        //Add the new fact minterm predicates to $M_F$.
        for $i = 1$ to $q$ do
            for $j = 1$ to $|M_{D_z}|$ do
                $M_F \longleftarrow M_F \cup fm_{ij}$
            end {end for $j$}
        end {end for $i$}
        $q = q * |M_{D_z}|$
    end {end for $z$}
End {FactMintermPredicates}
```

2. Derive the horizontal fragments set of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^t\}$. The derived horizontal fragments of the fact relation (member), in this

algorithm, are based on the dimension minterm predicates defined on the dimension relations that have owner links with the fact relation. Given (1) a link set $L = \{l_1, l_2, ..., l_n\}$ (where $L \subseteq \mathcal{L}$); (2) a set of dimension relations $Dset = \{D_1, D_2, ..., D_s\}$; (3) a set of fact minterm predicates $M_F = \{m_{f_1}, m_{f_2}, ..., m_{f_j}, ..., m_{f_t}\}$; (4) the set of semijoin predicates between the dimension relations and the fact relation $SemiPset = \{P_1, P_2, ..., P_k, ..., P_s\}$ where $(P_k : D_k.key = F_i.key)$ and (5) the fact relation F_i, the derived horizontal fragments of F_i are defined as

$$F_i^j = F_i \ltimes (\sigma_{m_{f_j} \in M_F}(Dset)).$$

Consider, for example, the link set is $L = \{l_1, l_2, l_3\}$, the dimension relations set is $Dset = \{D_k, D_l, D_u\}$, the fact relation is F_i and the set of fact minterm predicates is $M_F = \{m_{f_1}, m_{f_2}, ..., m_{f_j}, ..., m_{f_t}\}$ where $owner(l_1) = D_k, owner(l_2) = D_l, owner(l_3) = D_u$ and $member(l_1) = F_i, member(l_2) = F_i, member(l_3) = F_i$. Consider, also, the fact minterm predicate (fragment) m_{f_j} is $m_{f_j} = m_v^{D_k} \wedge m_r^{D_l} \wedge m_s^{D_u}$ then

$$F_i^j = F_i \ltimes (\sigma_{m_{f_j} \in M_F}(Dset))$$

$$F_i^j = (((F_i \ltimes (\sigma_{m_v^{D_k}}(D_k)) \ltimes (\sigma_{m_r^{D_l}}(D_l))) \ltimes (\sigma_{m_s^{D_u}}(D_u)))$$

Algorithm 6 (*FHF_Approach2*) provides a formal presentation of this phase. To the best of our knowledge deriving the horizontal fragments of a member relation from the conjunction of all the minterm predicates of the owner relations has not yet appeared.

6.3.1 The Algorithm Complexity

This section presents the computation of the horizontal fragmentation algorithm 6 "*FHF_Approach2*". Assume s represents the largest number of dimension relations that have applications and owner links with the fact relation, n represents the largest number of simple predicates in a dimension relation, r represents the number of simple predicates sets that has been grouped out from the n simple predicates of the dimension relation, t is the number of fact minterm predicates defined on all dimension relations. The horizontal fragmentation algorithm of the fact relation (*FHF_Approach2*) is composed of four algorithms. The time complexity of these algorithms are: (1) the "OptimizeDimension" algorithm is $O(sn)$; (2) the "COM_MIN" algorithm is $O(sn + sn^2)$; (3) the "DimenMintermPredicates" algorithm is $O(sn^2 + sn^r)$; and (4) the "FactMintermPredictes" algorithm is $O(n^s)$. Therefore, the time complexity of the "*FHF_Approach2*" is $O(2sn + 2sn^2 + sn^r + n^s + t)$. The time complexity of the algorithm can be simplified to $O(n^r + n^s)$. Therefore, the algorithm is exponential in the number of the simple predicates sets (r) and the largest number of the dimension relations that have applications and owner links with the fact relation (s). The time complexity of the algorithm will be reasonable if the size of r and s remain small.

Algorithm 6 FHF_Approach2

Input:

- The set of the dimension relations $Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$ that has applications and owner links with the fact relation F_i where $Dset \subseteq \mathcal{D}$.
- The attributes hierarchy relation $D_{hk}(AH, Level)$ for each dimension relation in the $Dset$.
- The dimension relations predicates set $DPset = \{P_{D_1}, P_{D_2}, ..., P_{D_k}, ..., P_{D_S}\}$.
- The set of semijoin predicates between the dimension relations (owners) and the fact relation (member) $SemiPset = \{P_1, P_2, ..., P_k, ..., P_s\}$.
- The fact relation F_i.

Output:

- A set of derived horizontal fragments of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^t\}$.

```
Var
$Temp\_DPset$: dimension predicates set.
$DMset$: dimension minterm predicates set.
Begin
    // Optimize the set of simple predicates for each dimension relation
    for each $P_{D_k} \in DPset$ do
        $P_{D_k} \longleftarrow$ OptimizeDimension($P_{D_k}, D_{hk}$)
        $Temp\_DPset \longleftarrow Temp\_DPset \cup P_{D_k}$
    end{end for $P_{D_k}$}
    // Determine the minterm predicates for each dimension relation
    for each $D_K \in Dset$ AND $P_{D_k} \in Temp\_DPset$ do
        $P'_{D_k} \longleftarrow$ COM_MIN($D_k, P_{D_k}$)
        $M_{D_k} \longleftarrow$ DimenMintermPredicates($P'_{D_k}$)
        $DMset \longleftarrow DMset \cup M_{D_k}$
    end {end for $D_K$}
    // Determine the fact minterm predicates
    $M_F \longleftarrow$ FactMintermPredicates($DMset$)
    // Derived the horizontal fragments of the fact relation
    for each $m_{f_j} \in M_F$ do
        $F_i^j = F_i$
        for each $m^{D_k} \in m_{f_j}$ do
            $F_i^j \longleftarrow F_i^j \ltimes (\sigma_{m^{D_k}}(D_k))$
        end {end for $m^{D_k}$}
        $f_{F_i} \longleftarrow f_{F_i} \cup F_i^j$
    end {end for $m_{f_j}$}
End {FHF_Approach2}
```

The worst-case analysis is when

- r is equal m ($r = m$) where m is the number of the dimension relation attributes ($D_i(A^D_{i1}, A_{i2}, A_{i3}, ..., A_{ij}, ..., A_{im})$). Section 6.2.1 shows that r is less than m ($r < m$) where m is the number of attributes in the dimension relation ($D_i(A^D_{i1}, A_{i2}, A_{i3}, ..., A_{ij}, ..., A_{im})$).
- s is equal u ($s = u$) where u is the number of the dimension relation in the dimension relation schemes $\mathcal{D}$ (recall Definition 1 of the data warehouse schema where $S = (\mathcal{D}, \mathcal{F})$ and $\mathcal{D} = \{D_1, D_2, ..., D_u\}$ is a set of dimension relation schemes). The algorithm will not reach to the worst-case because in the data warehouse schema, each fact relation has links with a subset of the dimension relations. Therefore, s is less than u ($s < u$).

The algorithm is still a feasible solution for the static fragmentation design problem because it will run off line. The time complexity of this algorithm costs more than the algorithm of the first approach (see Section 6.2.1). This is because the algorithm considers all the dimension relations that have applications and owner links with the fact relation.

6.3.2 An Example

Consider the same example presented in Section 6.2.2. Consider the same applications q_1 to q_5 will be input to the horizontal fragmentation design process of the second approach.

The horizontal fragmentation design process:

Apply the three major steps of Algorithm 6 (*FHF_Approach2*) to the input, given in the example, as follows:

Step One: Optimize the set of simple predicates for each dimension relation.

Store dimension: By applying Algorithm 2 (*OptimizeDimension*) to the set of simple predicates P_{Store} and the attributes hierarchy relation D_{hStore} (see Figure 6.2), the attribute with the highest level among the hierarchical predicates is "Region". Therefore, the P_{Store} set will be refined to have the following predicates

$$P_{Store} = \{\text{Region} = \text{``South East''}, \text{Region} = \text{``South West''}\}.$$

Product dimension: By applying Algorithm 2 (*OptimizeDimension*) to the set of simple predicates $P_{Product}$ and the attributes hierarchy relation $D_{hProduct}$ (see Figure 6.2), the attribute that has the highest level among the hierarchical predicates is "Category". Therefore, the $P_{Product}$ set will be refined to the following predicates

$$P_{Product} = \{\text{Category} = \text{``Foods''}, \text{Category} = \text{``Drinks''}, \text{Category} = \text{``Supplies''}\}.$$

Time dimension: By applying Algorithm 2 (*OptimizeDimension*) to the set of simple predicates P_{Time} and the attributes hierarchy relation D_{hTime} (see Figure 6.2), the P_{Time} set will not be changed because the simple predicates in P_{Time} set have the same attribute "Year" ($P_{Time} = \{\text{Year} = \text{``1997''}, \text{Year} = \text{``1998''}\}$).

Step Two: Determine the set of dimension minterm predicates for each dimension relation.

Store dimension: The following processes will be performed on the output of step one of the *Store* dimension:

- The COM_MIN algorithm will return the complete and minimal set of predicates $P'_{Store} = \{$Region = "South East", Region = "South West"$\}$.
- Generate the minterm predicates. By applying Algorithm 3 "DimenMintermPredicates" to the set of simple predicates P'_{Store}, the following minterm predicates set $M_{Store} = \{m_1, m_2\}$ can be defined where:

  m_1 : (Region = "South East")

  m_2 : (Region = "South West")

Product dimension: The following processes will be performed on the output of step one of the *Product* dimension:

- The COM_MIN algorithm will return the complete and minimal set of predicates $P'_{Product} = \{$Category = "Foods", Category = "Drinks", Category = "Supplies"$\}$.
- Generate the minterm predicates. By applying Algorithm 3 "DimenMintermPredicates" to the set of simple predicates $P'_{Product}$, the following minterm predicates set $M_{Product} = \{m_1, m_2, m_3\}$ can be defined where:

m_1 : (Category = "Foods")

m_2 : (Category = "Drinks")

m_3 : (Category = "Supplies")

Time dimension: The following processes will be performed on the output of step one of the *Time* dimension:

- The COM_MIN algorithm will return the complete and minimal set of predicates P'_{Time} = {Year = "1997", Year = "1998"}.
- Generate the minterm predicates. By applying Algorithm 3 "DimenMintermPredicates" to the set of simple predicates P'_{Time}, the following minterm predicates set $M_{Time} = \{m_1, m_2\}$ can be defined where:

  m_1 : (Year = "1997")

  m_2 : (Year = "1998")

Step Three: Generate the set of fact minterm predicates and derive the horizontal fragments of the fact relation.

To process this step all the dimension minterm predicates will be collected into one set $DMset = \{M_{Store}, M_{Product}, M_{Time}\}$. Then, the following processes will be performed:

Generate the fact minterm predicates. Based on the $DMset$ set, 12 fact minterm predicates will be determined. The sales fact minterm predicates are $M_{Sales} = \{m_{Sales1}, m_{Sales2}, ..., m_{Sales12}\}$ where for example:

m_{Sales1} : (Category = "Foods") $\wedge$ (Region = "South East") $\wedge$ (Year = "1997")

m_{Sales2} : (Category = "Foods") $\wedge$ (Region = "South East") $\wedge$ (Year = "1998")

m_{Sales3} : (Category = "Foods") $\wedge$ (Region = "South West") $\wedge$ (Year = "1997")

m_{Sales4} : (Category = "Foods") $\wedge$ (Region = "South West") $\wedge$ (Year = "1998")

m_{Sales5} : (Category = "Drinks") $\wedge$ (Region = "South East") $\wedge$ (Year = "1997")

m_{Sales6} : (Category = "Drinks") $\wedge$ (Region = "South East") $\wedge$ (Year = "1998")

m_{Sales7} : (Category = "Drinks") $\wedge$ (Region = "South West") $\wedge$ (Year = "1997")

m_{Sales8} : (Category = "Drinks") $\wedge$ (Region = "South West") $\wedge$ (Year = "1998")

m_{Sales9} : (Category = "Supplies") $\wedge$ (Region = "South East") $\wedge$ (Year = "1997")

$m_{Sales10}$: (Category = "Supplies") $\wedge$ (Region = "South East") $\wedge$ (Year = "1998")

$m_{Sales11}$: (Category = "Supplies") $\wedge$ (Region = "South West") $\wedge$ (Year = "1997")

$m_{Sales12}$: (Category = "Supplies") $\wedge$ (Region = "South West") $\wedge$ (Year = "1998")

Derive the horizontal fragments of the fact relation. Based on the twelve fact minterm predicates in the M_{Sales} set, twelve horizontal fragments will be derived from the *Sales* fact relation. For instance, to derive the first fragment from the sales fact relation, we issue the following semijoin:

$$Sales^1 = (((Sales \ltimes (\sigma_{Category=\text{“}Foods\text{”}}(Product)) \ltimes (\sigma_{Region=\text{“}South\ Eest\text{”}}(Store))) \ltimes (\sigma_{Year=\text{“}1997\text{”}}(Time)))$$

The above example shows that the derived horizontal fragments of the *Sales* fact relation reflect all the applications of the DSS users.

6.4 Correctness of the Horizontal Fragmentation Algorithms of the fact relation

This section validates the two horizontal fragmentation algorithms of the fact relation (*FHF_Approach1 and FHF_Approach2)* with respect to the three correctness rules of fragmentation: *completeness*, *reconstruction* and *disjointedness.*

6.4.1 Completeness

Completeness ensures that all tuples from a relation are mapped into at least one fragment without any loss.

The completeness of the primary horizontal fragmentation is guaranteed as long as the set of simple predicates are complete and minimal. The COM_MIN algorithm

[ÖV91] ensures the complete and minimality of the simple predicates.

The completeness of derived horizontal fragmentation is guaranteed as long as the referential integrity rule is satisfied among the relations involved in the fragmentation design.

The referential integrity between the dimension relations and the fact relation are satisfied as discussed in the data model for data warehouse (see Section 5.1). Moreover the two functions: *Owner* and *Member* identify the type of the relations through the join link between the relations. This must be completed before the fragmentation design process.

Since the input relations to the two derived horizontal fragmentation algorithms (*FHF_Approach1 and FHF_Approach2)* follow the referential integrity rule, then the derived horizontal fragments of the fact relation are complete.

6.4.2 Reconstruction

Reconstruction of a relation from its fragments ensures that constraints defined on the data are preserved. The reconstruction is performed by the union operator. For example, the derived horizontal fragments of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^t\}$ can be reconstructed as follows: $F_i = \bigcup F_i^j$ for $1 \leq j \leq t$.

6.4.3 Disjointedness

Disjointedness ensures that the generated fragments are non-overlapping. The disjointedness of the derived horizontal fragments are guaranteed as long as the minterm predicates that determine the fragments of the owner relation are mutually exclusive. The following points support the claim that the derived horizontal fragments of the fact relation are disjoint:

- The COM_MIN algorithm ensures that the set of simple predicates is minimal (i.e., non-overlap) which ensures that the generated minterm predicates from the set of simple predicates are non-overlapping.
- The dimension relations themselves are mutually exclusive because each dimension relation represents a specific dimension of the organization (see Section 5.1 data warehouse schema).

Since the generated dimension minterm fragments and the fact minterm predicates are ensured to be non-overlapping, the derived horizontal fragments of the fact relation from both algorithms (*FHF_Approach1 and FHF_Approach2)* are disjoint.

Chapter 7

Analysis

Chapter 6 presents the two approaches for deriving the horizontal fragments of the fact relation. The first approach derives the horizontal fragments of the fact relation based on the applications that are defined on one selected dimension relation. The second approach derives the horizontal fragments of the fact relation based on the applications that are defined on all the dimension relations. These two approaches raise the question: When should the distributed data warehouse designer choose approach one over approach two or vice versa?.

Knowing the quantitative information about each dimension relation and analyzing their distribution will help the distributed data warehouse designer to choose one approach over the other.

The following steps are required for making a decision:

1. Generating the Dimension Access Frequency (DAF) vector: The DAF vector represents the quantitative information of the dimension relations. Each ele-

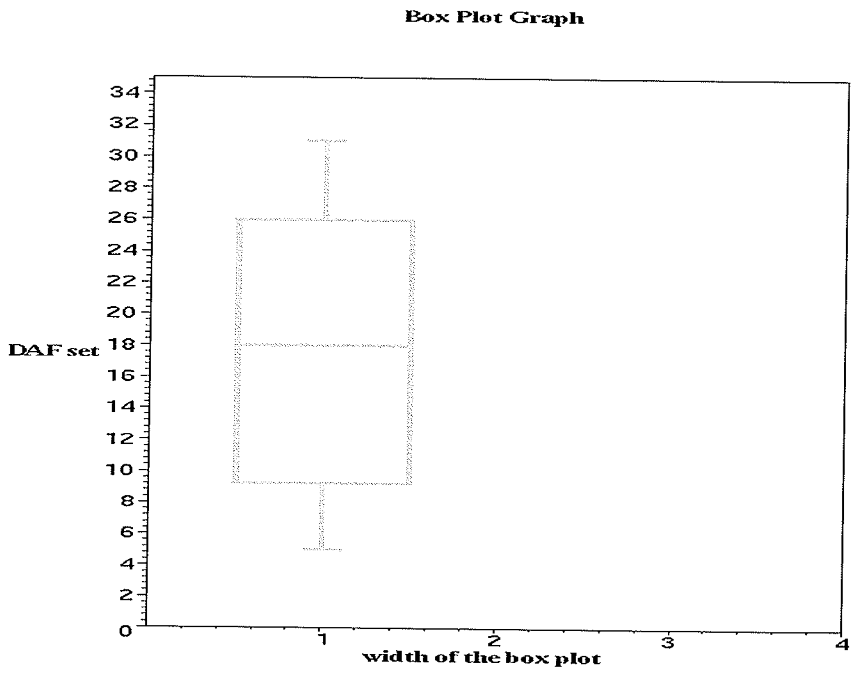


Figure 7.1: The Box Plot Graph.

ment of the DAF vector represents the total access frequency of the user applications against a dimension relation. The process of generating the DAF vector is presented in Section 6.2.

2. Analyzing the distribution of the quantitative information: Analyzing the data distribution of the DAF vector will be based on the *five-number summary* and the *box plot* graph [CT97b, Moo95]. The five-number summary of a data set consists of the *smallest observation*, the *first quartile*, the *median*, the *third quartile*, and the *largest observation*. The first quartile lies one-quarter of the way up the list. The third quartile lies three-quarters of the way up the

list. The five-number summary of a distribution is represented by the box plot graph. Figure 7.1 shows the box plot graph for the following data set $\{5, 7, 10, 14, 16, 18, 20, 25, 29, 31, 31\}$. The central box in the graph has its ends at the quartiles and therefore spans the middle half of the data. The line within the box marks the median. The whiskers at either end extend to smallest and largest observations.

There are five cases that could represent the data available in the DAF vector. The analysis of the five cases will be presented below. In each case, we assume that the dimension access frequency (DAF) vector has been generated previously.

- **Case 1**: Consider that the generated DAF vector has the following elements DAF = <205, 207, 210, 214, 216, 218, 220, 225, 229, 231, 231 >. Figure 7.2 shows the box plot graph of the DAF vector. It is clear from the graph that the distribution of the DAF vector is symmetric as the first and third quartiles are equally distance from the median. This implies that the access frequencies of the dimension relations are balanced. Therefore, generating horizontal fragments of the fact relation based on all the dimension relations (the second approach) will be the preferable choice.

- **Case 2**: Consider that the generated DAF vector has the following elements DAF = <189, 195, 198, 200, 205, 207, 210, 212, 214, 216, 218, 219, 222, 225,

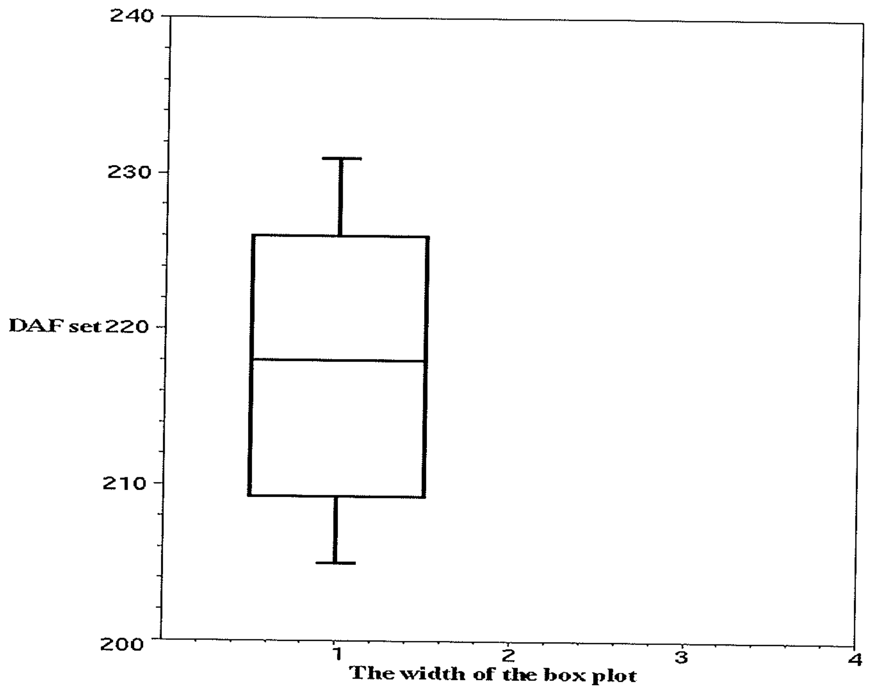


Figure 7.2: The box plot graph for case 1.

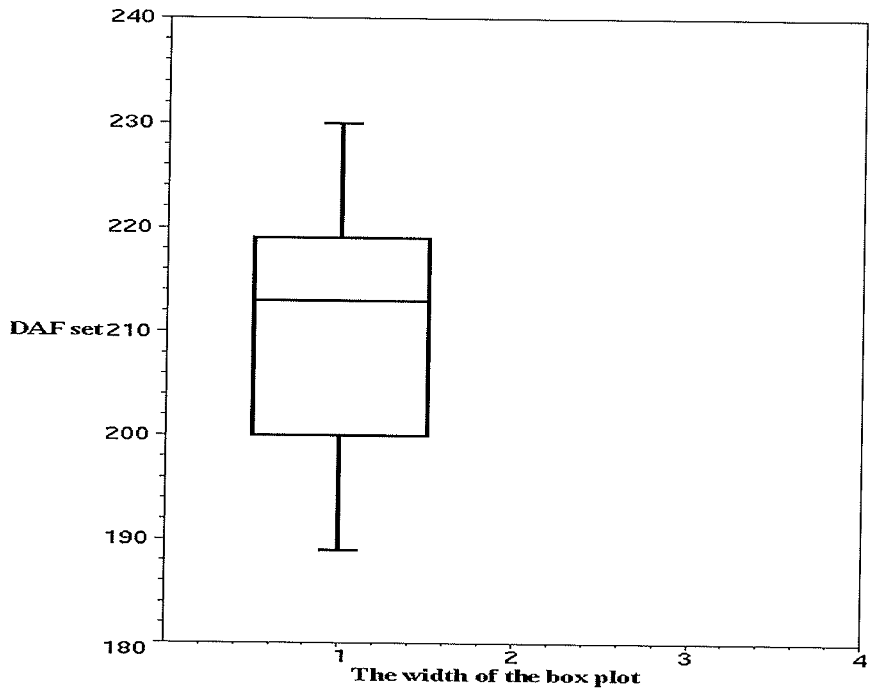


Figure 7.3: The box plot graph for case 2.

229, 230 >. Figure 7.3 shows the box plot graph of the DAF vector. In this case the distribution of the DAF vector is skewed to the left because the first quartile is further down the median than the third quartile. This implies that there is a small group of the dimension relations which has a high access frequencies while the majority of the dimension relations have lower access frequencies. Therefore, generating the horizontal fragments of the fact relation based on the dimension relation with the highest access frequency (the first approach) will be the preferable choice.

- **Case 3**: Consider that the generated DAF vector has the following elements

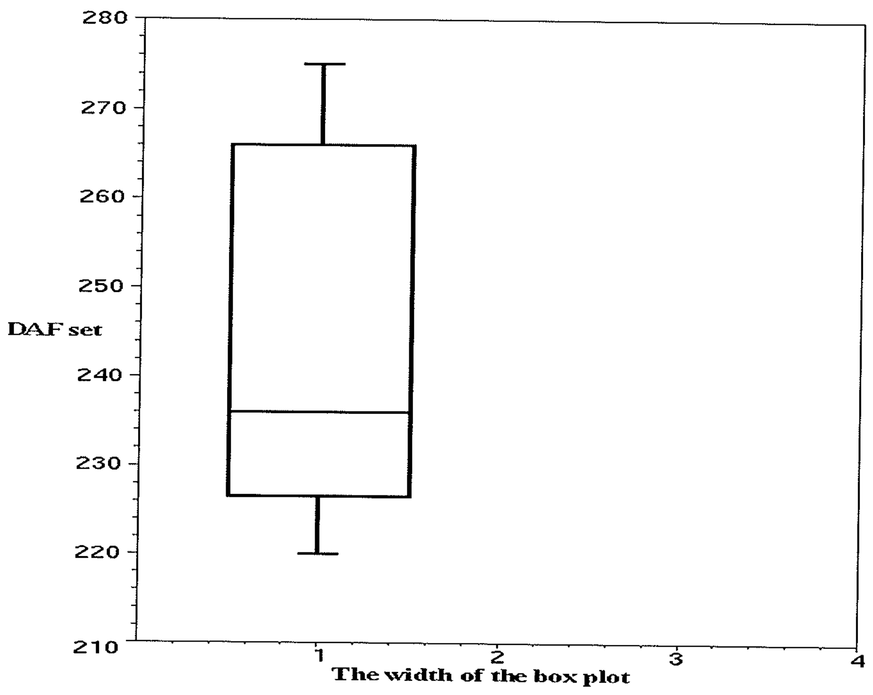


Figure 7.4: The box plot graph for case 3.

DAF = < 220, 221, 222, 225, 227, 228, 231, 233, 234, 236, 250, 255, 259, 265, 269, 271, 271, 273, 275 >. Figure 7.4 shows the box plot graph of the DAF vector. In this case the distribution of the DAF vector is skewed to the right because the third quartile is farther above the median than the first quartile. This implies that there are large groups of the dimension relations which have high access frequencies and a small number of the dimension relations that has lower access frequencies. Therefore, generating the horizontal fragments of the fact relation based on all the dimension relations (the second approach) will be the preferable choice.

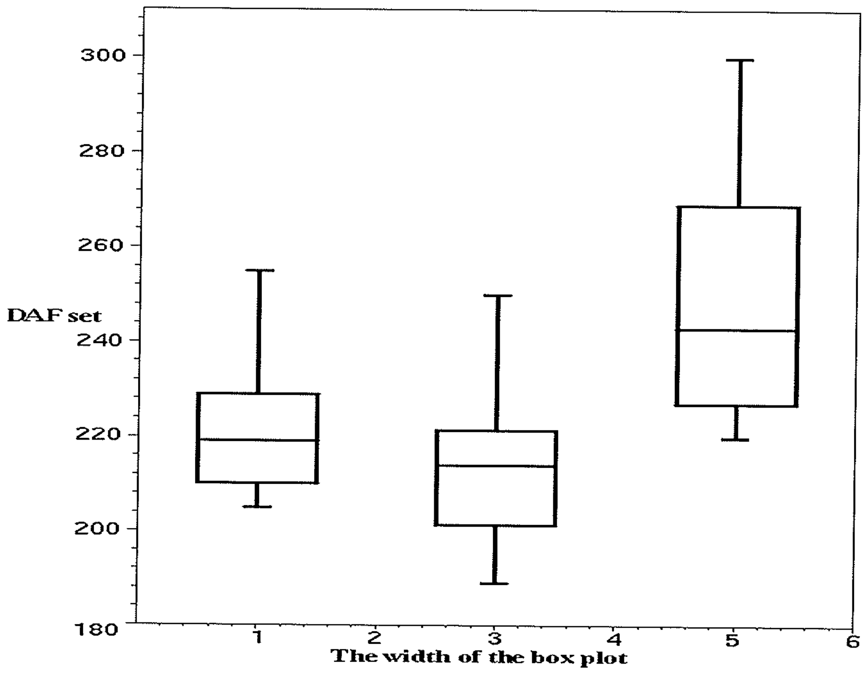


Figure 7.5: The box plot graph for case 4.

- **Case 4**: In this case the distribution of the DAF vector could be similar to case 1, 2, or 3 but the maximum observation is farther above the third quartile (see Figure 7.5). This implies that there is a dimension relation that has a very high access frequency compared to the other dimension relations. Therefore, generating the horizontal fragments of the fact relation based on the highly accessed dimension relation (the first approach) will be the preferable choice.

- **Case 5**: In this case the distribution of the DAF vector could be similar to case 1, 2, or 3 but the minimum observation is farther down the first quartile (see Figure 7.6). This implies that there is a dimension relation that has a very

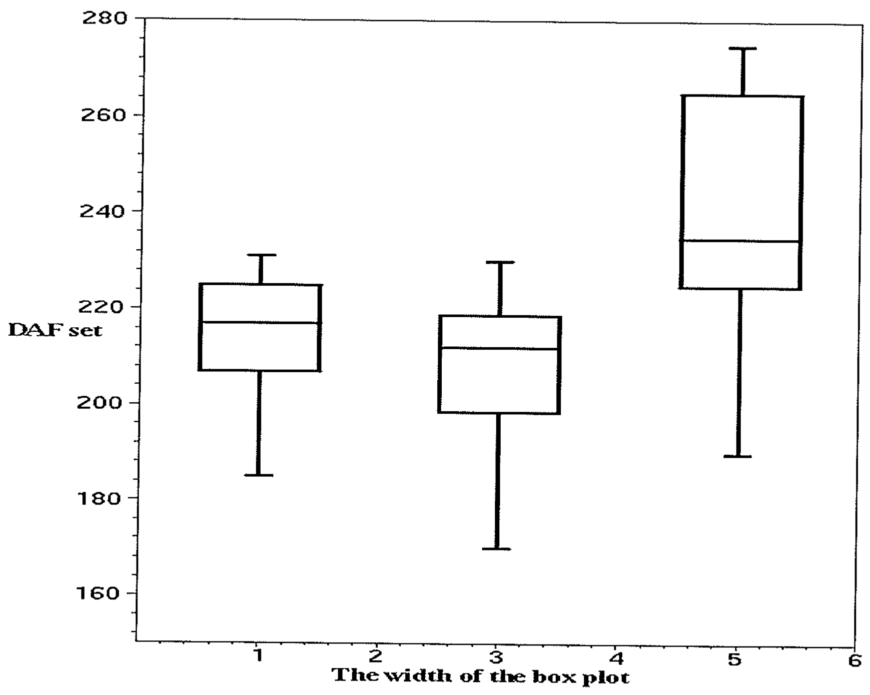


Figure 7.6: The box plot graph for case 5.

low access frequency compared to the other dimension relations. The dimension relation with very low access frequency has no impact on choosing one of the two approaches. Therefore, choosing approach one or two will depend on the distribution of the DAF vector, if it is similar to case 1, 2, or 3.

The above five cases quide the distributed data warehouse designer in selecting the right approach for fragmenting the fact relation. The following table summarizes the five cases that has been presented.

Case	The distribution Characteristics	The recommended Approach
1	Symmetric	Two
2	Skewed to the left	One
3	Skewed to the right	Two
4	Maximum observation	One
5	Minimum observation	Similar to case 1, 2, or 3

Chapter 8

Conclusion

8.1 Summary and Contributions

This dissertation undertakes a comprehensive study of issues in distributed data warehouse architecture and design. The objective of this dissertation is focused on the investigation of the feasibility of placing the data warehouse in a distributed environment with particular emphasis on optimal distribution design for the denormalized relations in the data warehouse environment. The major contribution of this dissertation is its successful attempt to design a solution to the fragmentation problem in the distributed data warehouse systems. The significant contributions made by this research are summarized below.

First, the dissertation proposes a distributed data warehouse architectural model, describes the functionality of its components and how information flows in the distributed data warehouse environment. The architecture represents the solution for a large enterprise with various divisions and geographic locations.

Secondly, the dissertation provides a formal definition of the data model for the data warehouse and describes the characteristics of the data warehouse schema. The formalization and the characteristics of the data warehouse represents the basis of understanding the physical structure of the data warehouse which support the design for an efficient methodology for the distributed data warehouse.

Thirdly, the dissertation provides a methodology for distributed data warehouse design. The main ideas of the methodology are to replicate the dimension relations across the network and to generate horizontal fragments of the huge fact relation. The dissertation proposes two approaches to horizontally fragment the fact relation. The first approach provides an algorithm that derives the horizontal fragments of the fact relation based on selecting the applications of the dimension relation that has the highest access frequency among the dimension relations. The second approach provides an algorithm that derives the horizontal fragments of the fact relation based on the applications of all the dimension relations reference by the fact relation.

Finally, the dissertation provides an analytical technique to support the distributed data warehouse designer in selecting approach one or two to generate the horizontal fragments for the fact relation.

8.2 Future Research Directions

This section outlines the future research directions on the distributed data warehouse design as an extension of the research presented in this dissertation. The following few paragraphs will address briefly some of the open research problem related to this research.

Goodness of the horizontal fragmentation schemes: Measuring the goodness of the horizontal fragmentation schemes requires an objective function similar to the one developed by Chakravarty *et al.* [CMVN93]. The main idea of the objective function is to measure the total cost of local and remote accesses to data on the network by all the applications over a period of time using the horizontal fragments that has been generated from the fragmentation scheme. The objective function developed by Chakravarty measures the goodness of the vertical fragmentation schemes. Developing an objective function for horizontal fragmentation schemes is an important issue for future research.

Fragmentation and allocation are two fundamental issues in the distributed database design. There are three fragmentation strategies: horizontal, vertical, and mixed. The dissertation shows that the current horizontal fragmentation techniques for distributed database did not suit the data warehouse environment. Future research should attempt to investigate the other fragmentation strategies and the various allocation techniques of distributed database and looking at them from the data warehouse perspective.

The two approaches for the horizontal fragmentation that have been presented in this dissertation assume that the underlying model is the star schema. There are various extensions of this schema such as snowflake and Multi-Star schemes (see Section 5.2). Future research should attempt to investigate if the horizontal fragmentation techniques that have been presented in this dissertation, could suit the various extension of the star schema.

In the data distribution layer (see Section 4.2.2), fragmentation and allocation are iterative processes. They are applied whenever the transaction processing cost reaches a specific threshold. Future research should attempt to develop a technique that determines the threshold by taking into consideration the logical and physical changes in the distributed data warehouse environment.

The horizontal fragmentation techniques that have been presented in this dissertation address the solution to the static fragmentation design problem. Future research should attempt to modify the static horizontal fragmentation techniques so they can solve the dynamic fragmentation design problem. The dynamic fragmentation techniques are very complicated because they should re-partition the fragments transparently while the system is being used by the DSS users.

Bibliography

[AGS95] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling Multidimensional Databases. Technical report, IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, 1995.

[AM97] S. Anahory and D. Murray. *Data Warehousing in the Real World.* Addison-Wesley, 1997.

[Atr92] S. Atre. Distributed Databases, Cooperative Processing, & Networking. 1992.

[BB95] S. Bhar and K. Barker. Static Allocation in Distributed Objectbase Systems: A Graphical Approach. *Information Systems and Data Management, 6th International Conference CISMOD 95 Bombay, India, In: Lect. Notes in comp. Sci., No. 1006, Springer-Verlag, Bhalla S. (ed.),* November 1995.

[BE97] R. Barquin and H. Edelstein, editors. *Planning and Designing the Data warehouse.* Prentice Hall PTR, 1997.

[BKS97] L. Bellatreche, K. Karlapalem, and A. Simonet. Horizontal Class Partitioning in Object-Oriented Databases. *In Proceedings of 8th International Conference on Database and Expert Systems Applications (DEXA'97),* pages 58–67, September 1997.

[Bur97] D. Burleson. *High Performance Oracle Data Warehousing.* The Coriolis Group, 1997.

[CD97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), March 1997.

[Che76] P. Chen. The Entity-Relationship Model-Toward a Unified View of Data. *ACM-TODS*, pages 9–36, March 1976.

[CMVN93] S. Chakravarthy, J. Muthuraj, R. Varadarajan, and S. Navathe. An Objective Function for Vertically Partitioning Relations in Distributed Databases and its Analysis. *Distributed and Parallel Databases*, 2(1):183–207, 1993.

[CN83] S. Ceri and S. Navathe. A Methodology for the Distribution Design of Database. *COMPCON*, pages 426–431, 1983.

[CNP82] S. Ceri, M. Negri, and G. Pelagatti. Horizontal Data Partitoning in Database Design. In *Proceeding of the ACM SIGMOD International Conference on Management of data*, pages 128–136, 1982.

[CNW83] S. Ceri, S. Navathe, and G. Wiederhold. Distributed Design of logical Database Schemas. *IEEE Transaction on Software Engeneering*, 9(4):487–503, 1983.

[Cod70] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. *Communication of the ACM*, 13(6):377–387, June 1970.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases Principles and Systems.* McGraw-Hill, 1984.

[CT97a] L. Cabibbo and R. Torlone. Querying Multidimensional Databases. *In Proceedings of 6th DBPL Workshop*, pages 253–269, 1997.

[CT97b] W. Carlson and B. Thorne. *Applied Statistical Methods.* Prentice Hall, 1997.

[Dat95] C.J. Date. *An Introduction to Database Systems.* Addison-Wesley, sixth edition, 1995.

[Dem94] M. Demarest. Building the Data Mart. *DBMS*, 7(8):44–52, July 1994.

[Dev97] B. Devlin. *Data Warehouse from Architecture to Implementation.* Addision-Wesley, 1997.

[EB95] C. Ezeife and K. Barker. A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System. *International Journal of Distributed and Parallel Databases*, 2:247–272, 1995.

[ES76] M.J. Eisner and D.G. Severance. Mathematical Techniques for Efficient Record Segmentation in Large Shared Database. *Journal of the ACM*, 23(4), October 1976.

[GL97] M. Gyssens and L.V.S. Lakshmanan. A foundation for multi-dimensional databases. *Proc. VLDB*, pages 106–115, 1997.

[GMR98] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual Design of Data Warehouses from E/R Schemes. *In Proceedings of the Hawaii International Conference on System Sciences*, January 1998.

[HN79] M. Hammer and B. Niamir. A Heuristic Approach to Attribute Partitioning. *In Proceedings ACM SIGMOD*, pages 93–101, May 1979.

[Hof76] H.A. Hoffer. An Integer Programming Formulation of Computer Database Design Problem. *Information Science*, pages 29–48, July 1976.

[HS75] H.A. Hoffer and D.G. Severance. The Use of Cluster Analysis in Physical Data Base Design. *In Proceedings of First International Conference on Very Large Data Bases*, pages 69–86, 1975.

[IH94] W.H. Inmon and R.D. Hackathorn. *Using the Data Warehouse.* John Wiley & Sons, 1994.

[IIB96] W. H. Inmon, C. Imhoff, and G. Battas. *Building the Operational Data Store.* John Wiley & Sons, 1996.

[Inm93] W.H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, 1993.

[Inm95] W.H. Inmon. The Operational Data Store. *InfoDB Journal*, 9(1):21–26, February 1995.

[Inm96] W.H. Inmon. *Building the Data Warehouse.* John Wiley & Sons, second edition, 1996.

[Kau89] F.J. Kauffels. *Understanding Data Communications.* Ellis Horwood Limited & Halsted Press, 1989.

[Kim96] R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, Inc., 1996.

[LAW98] W. Lehner, J. Albrecht, and H. Wedekind. Multidimensional Normal Forms. *10th International Conference on Scientific and Statistical Data Managment (SSDBM'98), Capri, Italy*, July 1-3 1998.

[Leh98] W. Lehner. Modeling Large Scale OLAP Scenarios. *6th International Conference on Extending Database Technolgy (EDBT'98), Valencia, Spain*, pages 23–27, March 1998.

[LW96] C. Li and X.S. Wang. A Data Model for Supporting On-Line Analytical Processing. *In Proc. Conf. on Information and Knowledge Managment*, pages 81–88, November 1996.

[Mar81] J. Martin. *Design and Strategy for Distributed Data Processing.* Prentice-Hall, 1981.

[MHP99] F. R. McFadden, J. A. Hoffer, and M. B. Prescott. *Modern Database Management.* Addison-Wesley, 1999.

[Moo95] D. S. Moore. *The Basic Practice of Statistics.* W. H. Freeman and Company, 1995.

[MR92] H. Mannila and K.J. Raiha. *The Design of Relational Database.* Addison-Wesley, 1992.

[MS77] S.T. March and D.G. Severance. The Determination of Efficient Record Segmentations and Blocking Factors for Shared Data Files. *ACM Transactions on Database Systems*, 2(3):279–296, September 1977.

[MSW72] W.T. McCormick, P.J. Schweitzer, and T.W. White. Problem Decomposition and Data Reorganization by a Clustering Technique. *Operational Research*, 20(5):993–1009, September 1972.

[NB97] A.Y. Noaman and K. Barker. Distributed Data Warehouse Architectures. *Journal of Data Warehousing*, 2(2):37–50, April 1997.

[NCWD84] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical Partitioning Algorithms for Database Design. *ACM Transactions on Database Systems*, 9(4):680–710, December 1984.

[NKR95] S. Navathe, K. Karlapalem, and M. Ra. A Mixed Fragmentation Methodology for Initial Distributed Database Design. *Journal of Computer and Software Engineering*, 3(4):395–426, 1995.

[NR89] S. Navathe and M. Ra. Vertical Partitioning for Database Design: A graphical Algorithm. *In Proceedings of the ACM SIGMOD*, 18(2):440–450C, June 1989.

[ÖDV94] M.T. Özsu, U. Dayal, and P. Valduriez. *Distributed Object Managment.* Morgan Kaufmann, 1994.

[ÖV91] M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, 1991.

[PC93] K. Parsaye and M. Chignell. *Intelligent Database Tools & Applications.* John Wiley & Sons, 1993.

[Poe96] V. Poe. *Building A Data Warehouse for Decision Support.* Prentice Hall, 1996.

[Sch77] M. Schkolnick. A Clustering Algorithm for Hierachical Structure. *ACM Transactions on Database Systems*, 2(1), March 1977.

[Sho97] A. Shoshani. OLAP and Statistical Databases: Similarities and Differences. *In Proc. ACM PODS*, pages 185–196, 1997.

[SI91] D. Shin and K. Irani. Fragmenting Realtions Horizontally Using a knowledge-based Approach. *IEEE Transactions on Software Engineering*, 17(9):672–883, September 1991.

[Sta93] J.P. Stamen. Structuring Database for Analysis. *IEEE Spectrum*, 30(10):55–58, October 1993.

[SW85] D. Sacca and G. Wiederhold. Database Partitioning in a Cluster of Processors. *ACM Transaction on database systems*, 10(1):29–56, October 1985.

[TK78] D. Tsichritzis and A. Klug. *The ANSI/X3/SPARC Framework*. AFIPS Press, Montval, N.J., 1978.

[Whi95] C. White. A Technical Architecture for Data Warehousing. *InfoDB Journal*, 9(1):5–11, February 1995.