

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 [Imm93] 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

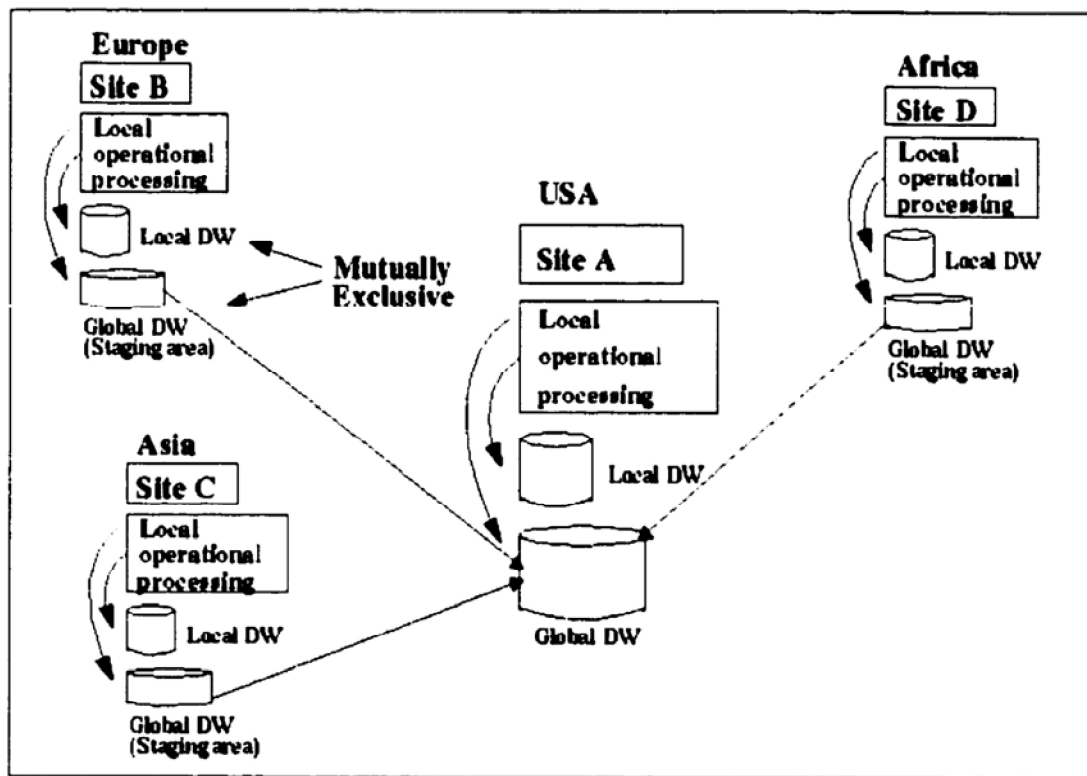


Figure 3.1: Inmon's Approach to DDW [Inm93].

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” [Imm93]. 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-

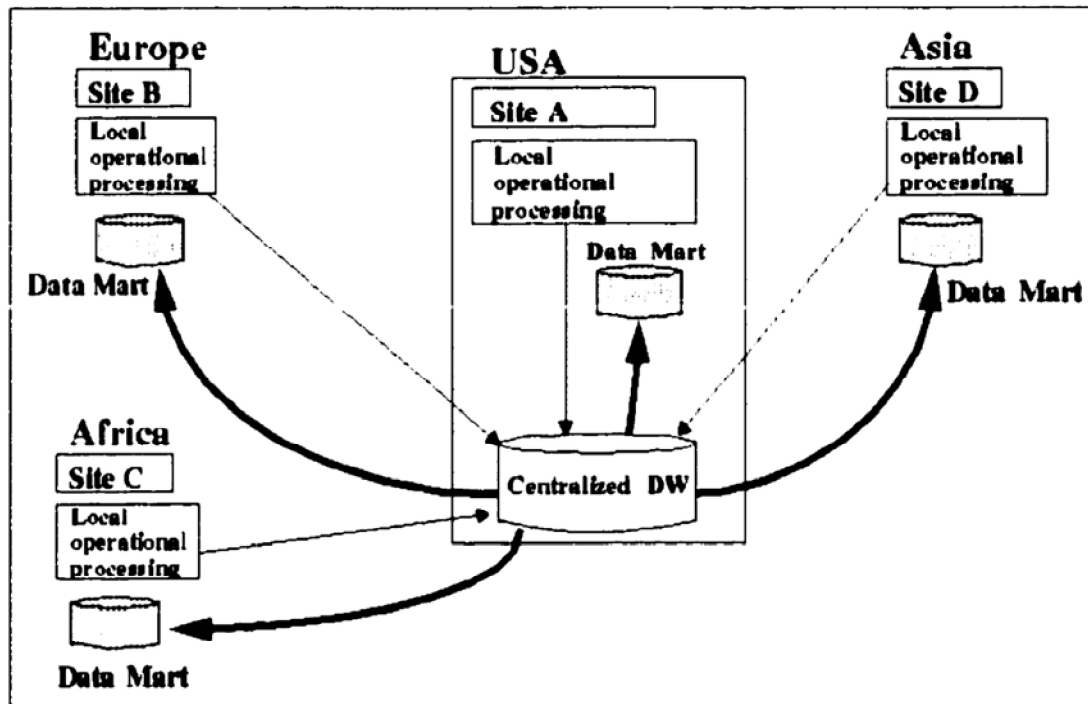


Figure 3.2: White's Approach to DDW.

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.

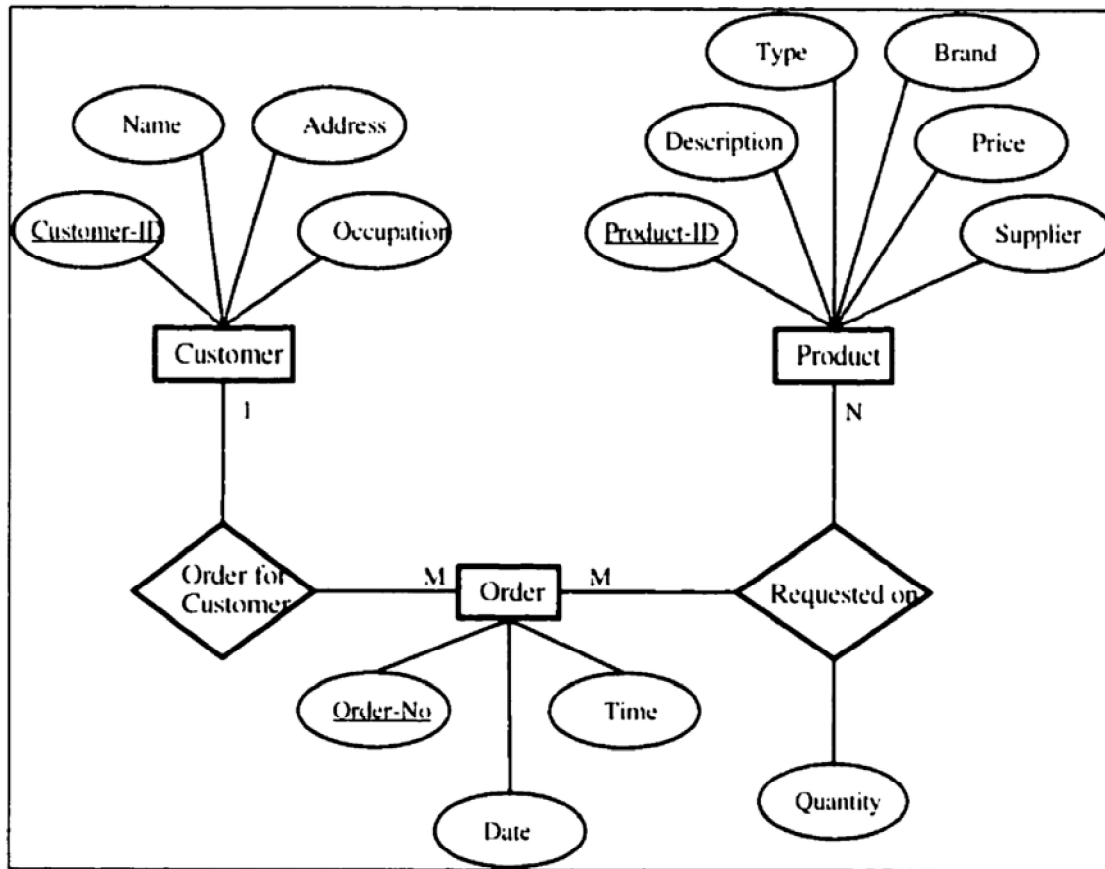


Figure 3.3: A simple entity-relationship diagram for the sales database.

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

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

The diagram shows a table representing a Product relation. A bracket above the table is labeled "Attributes". A bracket to the left of the table is labeled "Tuples". A bracket below the first column is labeled "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 frag-

mentation 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 par-

allelism 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) [PC'93]. 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

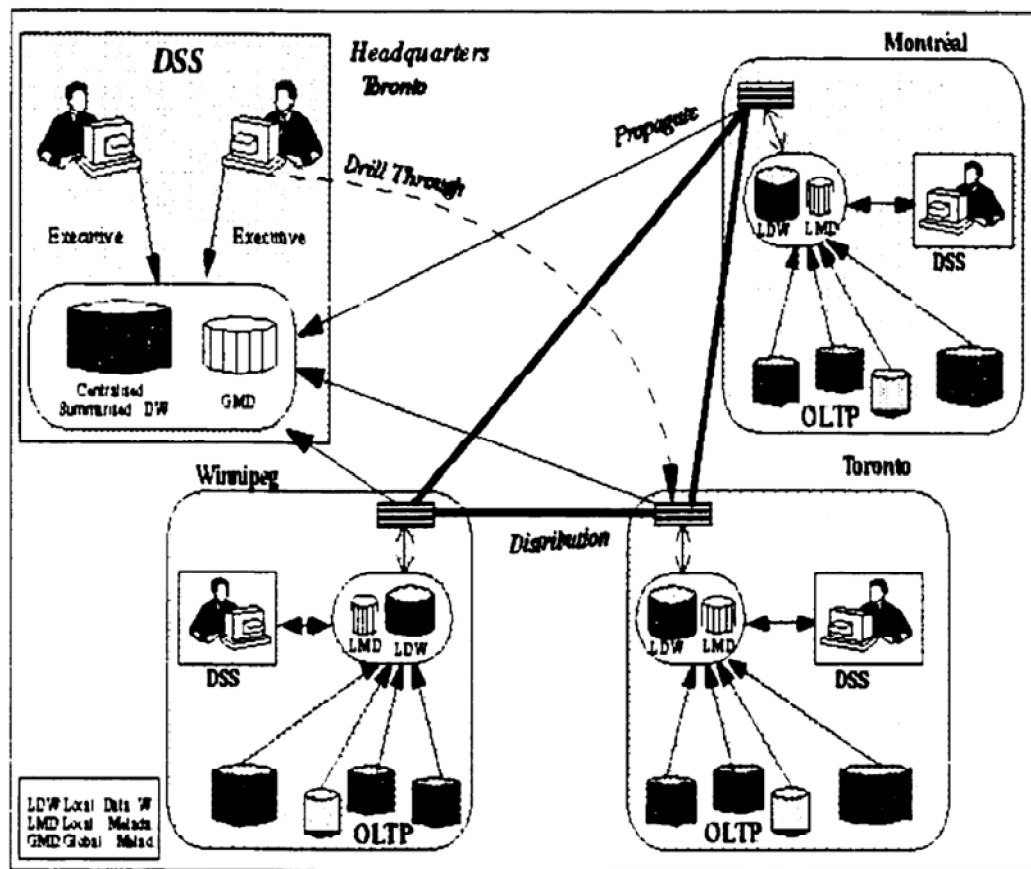


Figure 4.1: Architectural Model to DDW [NB97].

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*<sup>1</sup> (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.

---

<sup>1</sup> 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].



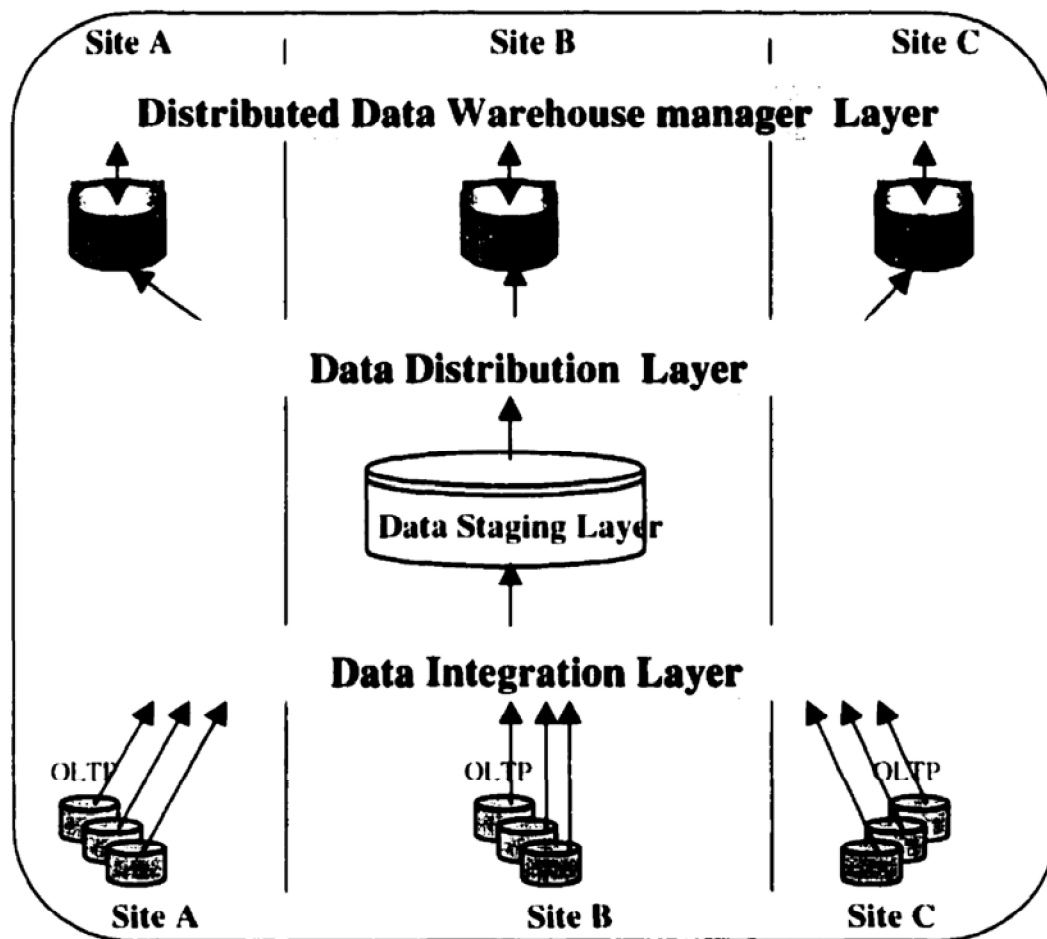


Figure 4.2: Distributed Data Warehouse System Architecture.

### 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<sup>2</sup> 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

---

<sup>2</sup> 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].

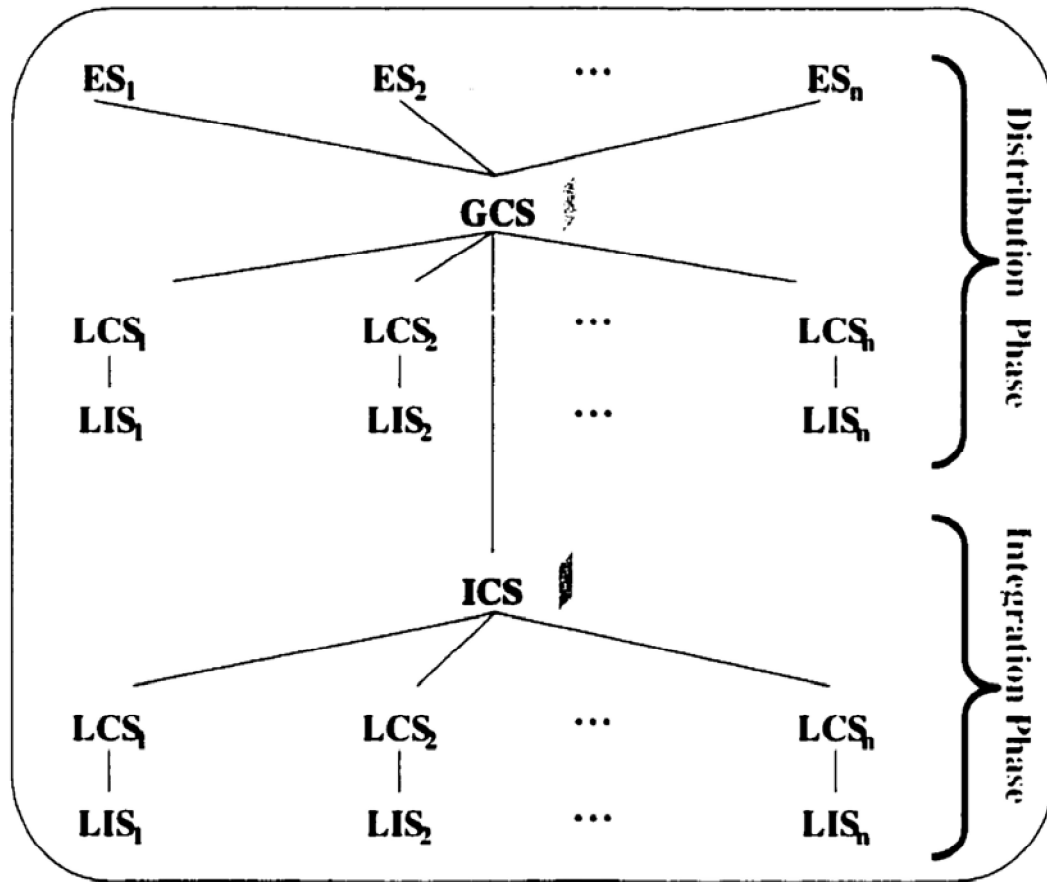


Figure 4.3: Distributed Data Warehouse Reference Architecture.

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.