A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse

Amin Y. Noaman
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
+1 204 474 8691
noaman@cs.umanitoba.ca

Ken Barker
Department of Computer Science
University of Manitoba
Winnipeg, Manitoba, Canada R3T 2N2
+1 204 474 8669
barker@cs.umanitoba.ca

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 pre-existing 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. The article provides an architectural model for a distributed data warehouse, the formal definition of the relational data model for data warehouse and a methodology for distributed data warehouse design along with a "horizontal" fragmentation algorithm for the fact relation.

Keywords

distributed data warehouse architecture, distributed data warehouse design, horizontal fragmentation.

1 INTRODUCTION

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 model that is integrated with existing warehouse data. Once information is available at the warehouse, queries can be answered and data analysis (DSS and EIS) can be performed.

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. The centralized data warehouse is very expensive and tends to ignore the advantages realized during the past decade in the areas of distribution and support for data localization in a geographically dispersed corporate structure. Further, it would be unwise to enforce a centralized data warehouse when the operational systems exist over a widely distributed geographical area.

The distributed data warehouse supports the decision makers by providing a single view of data even though that data are physically distributed across multiple data warehouses 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.

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

1. Extending the preliminary architecture model that has been presented in [8] by proposing a distributed data warehouse system architecture and describing the functionality of its components.
2. Proposing the formal definition of the relational data model for data warehouse where the relational data model represents the underlying model for the different level of schemas of the proposed system architecture.
3. Proposing a methodology for the distributed data warehouse design and a horizontal fragmentation algorithm that partitions the huge fact relation into a set of fragments.

To the best of our knowledge, this is one of the first works to propose a methodology and a horizontal fragmentation algorithm for the distributed data warehouse design.

The reminder of the paper is organized as follows. Section 2 presents our proposal for the distributed data warehouse system architecture and illustrates how the information flows in the distributed data warehouse. Section 3 provides the data model for data warehouse. Section 4 addresses our proposal for the distributed data warehouse design and presents the horizontal fragmentation algorithm for the fact relation. Finally, Section 5 draws conclusions.

2 DISTRIBUTED DATA WAREHOUSE ARCHITECTURE

This section extends the preliminary architecture model that has been presented in [8]. It proposes distributed data warehouse system architecture, and describes the functionality of its components

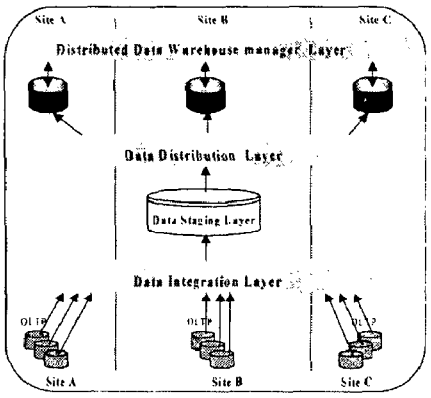

Figure 1: Distributed Data Warehouse System Architecture.

and how the information flow in the distributed data warehouse environment.

The proposed distributed data warehouse architecture represents the classical solution for a large enterprise with various divisions and geographic locations. It is based on the ANSI/SPARC architecture [13] that has three levels of schemas: internal, conceptual and external.

Figure 1 illustrates the distributed data warehouse system architecture. It is a four-tiered architecture consisting of (1) *The data integration layer.* (2) *The data staging layer.* (3) *The data distribution layer.* (4) *The distributed data warehouse manager layer.* The following sections discuss these layers in detail.

2.1 The data integration layer

The data integration layer consists of the sources databases available across the sites and the integration and transformation tools. Figure 1 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* (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 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.

2.2 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 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 mapped into the *Global Conceptual Schema* (GCS). The underlying model for the GCS is the data warehouse model (see Section 3 for the detail description 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 represent the most recent data that are added to the operational systems. The changed data represent the update occurring on existing data in the operational systems.

2.3 The data distribution layer

The data distribution layer provides the following processes: fragmentation, allocation and updating the distributed data warehouse. 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. 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.

2.4 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.

The various sites are connected using a communication protocol. This communication protocol provides a single application 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 [4].

3 THE DATA MODEL FOR DATA WAREHOUSE

The data model for a data warehouse should be designed to structure the data in a manner that could handle the *On-Line Analytical Processing* (OLAP). There are two techniques for modeling the data warehouse: the *multidimensional* data model [1, 7] and the *relational* data model [3, 6]. These two modeling techniques provide a multidimensional view of data to support and facilitate the OLAP applications. The distributed data warehouse design presented in Section 4 is based on the relational data model. The schema that is used for physical structures data warehouse data using the relational data model is the star schema. The following sections discuss the relational data warehouse schema and its formal definition in detail.

3.1 Data Warehouse Schema

The data warehouse schema (star schema) design was introduced by Kimball [6]. 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 [3, 12]. The large central tables

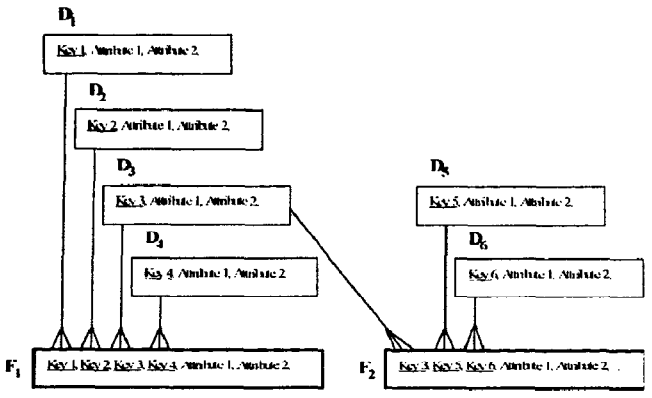

Figure 2: The data warehouse schema.

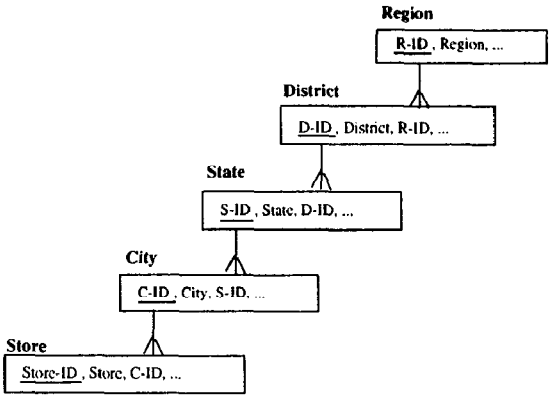

Figure 3: The OLTP store schema.

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:

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

The data warehouse schema is a set of relation schemas. Two types of relation schemas are available in data warehouse: Dimensions and Facts. The relationship between the dimension relation and fact relation is one to many as shown in Figure 2.

3.1.1 *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 containing 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.

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 [11]:

- 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 legacy system. Figure 3 illustrates normalized relations based on geographical regions. Clearly these are in at least 3rd NF. 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 3. The normalized relations (Region, District, State, City and Store) have a hierarchical 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 4 shows the store dimension relation after it has been denormalized out of the normalized relation which was shown in Figure 3. 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 represents the moving along the attributes hierarchy to decrease or increase the level of the aggregation [3]. 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 $S = (\mathcal{D}, \mathcal{F})$.

S-Key	Store	City	State	District	Region
1	S.No.1	New York	NY	New York	Eastern
2	S.No.2	L. Angeles	CA	L. Angeles	Pacific
3	S.No.3	Boston	MA	Suffolk	Eastern
4	S.No.4	Miami	FL	Dade	South East
...	...	...	...	...	...

Figure 4: The Store Dimension relation

Definition 3 *(Dimension):*
The dimension relation schema for eny D_i is represented by $D_i(A_{i1}^D, A_{i2}, A_{i3}, ..., A_{ij}, ..., A_{im})$ where $A_{ij} : D_i \rightarrow Q_{ij}$ is an attribute of D_i ($D_i \in \mathcal{D}$) and Q_{ij} is its domain for $1 \leq i \leq n$ and $1 \leq j \leq m$. ■

The *dimension relation key* A_{i1}^D is functionally determines all attributes of the dimension relation D_i, that is, $A_{i1}^D \longrightarrow A_{ij}$ for $1 \leq i \leq n$ and $2 \leq j \leq m$. The superscript letter D in A_{i1}^D implies that A_{i1}^D 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 involve in the dimension attributes hierarchy.
2. *Hierarchical attribute*: it contains descriptive information about the dimension and involves in the dimension attributes hierarchy.

3.1.2 *The Fact Relations*

The fact relation normally contains millions of rows and is highly normalized [6]. 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 for eny F_i is represented by $F_i(K_i^F, A_{i1}^m, A_{i2}^m, ..., A_{ij}^m, ..., A_{ip}^m)$ where $A_{ij}^m : F_i \rightarrow Q_{ij}$ 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 $1 \leq j \leq p$. The A_{ij}^m attribute contains a numeric value. The superscript letter m in A_{ij}^m implies that A_{ij}^m is a measurement attribute. ■

The *fact relation key*, represented by the foreign key set $K_i^F = \{A_1^D, A_2^D, ..., A_l^D, ..., A_g^D\}$ where A_l^D ($1 \leq l \leq g$) is a foreign key references a primary key on one of the dimension relations and g is the number of the dimension relations referenced by the fact relation F_i. Each element A_l^D of the K_i^F set functionally determines all the measurement attributes of the fact relation F_i, that is, $A_l^D \longrightarrow A_{ij}^m$ for $1 \leq l \leq g$, $1 \leq i \leq q$ and $1 \leq j \leq p$.

3.1.3 *The Relations Size*

The size of the dimension relations is smaller than the size of the fact relations [6]. 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 (see Figure 5). The number of records of the time dimension relation would be ($3\ Years * 365\ Days = 1095\ Days$ (records)). Also suppose that there are 200 stores (records) in the store dimension relation and 50, 000 products in the product dimension relation. Let us assume that the daily sales of each stores are 6000 items. Then the number of records in the sales fact relation would be ($1095 * 200 * 6000 = 1314\ millions\ records$). Therefore, it is clear that the size(D) $\ll$ size(F).

4 DISTRIBUTED DATA WAREHOUSE DESIGN

The distributed data warehouse design proposed in this research is based on the top-down design approach. There are two fundamental issues in the top-down design approach: fragmentation and allocation. The problem of fragmentation and allocation has been addressed for distributed relational database system [2, 10] and the distributed object based system [5]. Previous research work on fragmentation and allocation for distributed relational database system has been based on *highly normalized* relations[1].

In the data warehouse environment, the integrated data from different resources are modeled into a *denormalized* relations to facilitate the on-line data analysis (see Section 3). The existing distributed relational database design techniques for fragmentation will not work for distributed data warehouse because of the underlying model.

This section proposes a methodology for the distributed data warehouse design and a horizontal fragmentation algorithm for the huge fact relation. The main idea of the methodology will be discussed first, followed by a detailed description of the algorithm.

4.1 The Design Methodology

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 size of the dimension relations is relatively small compared to the fact relations (see Section 3.1.3 for details).
- The dimension relations are changing slowly, the cost of updating the 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.

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

There are two approaches for horizontal fragmentation: *primary* and *derived* [2, 10]. Applying the primary horizontal fragmentation requires a set of simple predicates used in user queries against the relation. The fact relation represents the numerical measurements

[1] Clearly much work has been done in the area of horizontal fragmentation in the distributed relational database systems. Instead of providing a complete review, we will point to the relevant work as we present the algorithms in subsequent sections.

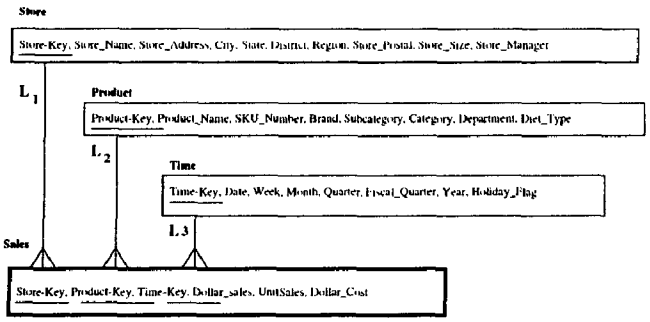

Figure 5: A simple data warehouse schema for sales.

of the organization. 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 apply the derived horizontal fragmentation on the fact relation. In the data warehouse schema, the fact relation is owned by more than one dimension relations (owners) and represents the many-to-many relationship among the dimension relations. For example, in Figure 5 the many-to-many relationship among the dimension relations (Store, Product, Time) is expressed with three links (L_1, L_2, L_3) to the sales fact relation (member).

Since the fact relation is owned by more than one dimension relations, there are two approaches for deriving the horizontal fragments of the fact relation:

- the first approach derives the horizontal fragments of the fact relation from the predicates that are defined on only one dimension relation. This dimension relation has more applications than the other dimension relations. The algorithm for implementing this approach has been presented in [9]. Unfortunately, this approach has the following drawbacks: (1) the generated fragments of this approach does not reflect all the user applications against the data warehouse because they based on the applications of one dimension relation; (2) the number of generated fragments is small but they are large in terms of size. The small number of fragments will decrease the level of executing the query in parallel and therefore the system performance. The large size of fragments may not be desirable if storage is limited in the distributed sits.
- the second approach derives the horizontal fragments of the fact relation from the predicates that are defined on all the dimension relations. The following section proposed an algorithm for implementing this approach.

4.2 The Horizontal Fragmentation Algorithm for the Fact Relation

The algorithm consists of three major steps to generate the horizontal fragments of the fact relations. 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. apply the primary horizontal fragmentation algorithm, presented in [10, 2], on each dimension relation. The algorithm generates the set of minterm predicates for each dimension relation along with the set of implications defined on each dimension.
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 1 OptimizeDimension

Input:

- the set of simple predicates for the 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 dimension relation.

Output:

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

Begin
// Identify the level of the highest attribute in P_{D_k}.
$HighestLevel = 0$
for each $p_j^{D_k} \in P_{D_k}$ **do**
  if A_{ki} of $p_j^{D_k}$ exist in $D_{hk}.AH$ **then**
    if $D_{hk}.Level \geqslant HighestLevel$ **then**
      $HighestLevel = D_{hk}.Level$
end {end for $p_j^{D_k}$}
// Remove the predicates that have a lower level than the
// $HighestLevel$.
for each $p_j^{D_k} \in P_{D_k}$ **do**
  if A_{ki} of $p_j^{D_k}$ exist in $D_{hk}.AH$ **then**
    if $D_{hk}.Level < HighestLevel$ **then**
      $P_{D_k} = P_{D_k} - p_j^{D_k}$
end {end for $p_j^{D_k}$}
Return(P_{D_k})
End {OptimizeDimension}

Algorithm 4 (*FactHorizontalFragments*) provides a formal presentation of these steps. It has been assumed that all the input to the horizontal fragmentation algorithm are determined during the requirement analysis phase of the top-down design approach. 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 application information is required to process this step. There are two types of the user applications information defined on the relation: quantitative and qualitative information. The *quantitative* information is essential for the allocation [10]. 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 *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 P_{D_k} set is defined as: $p_j^{D_k} : A_{ki}\ \theta\ Value$ 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 the value from the domain Q_z of A_{ki} (i.e., $Value \in Q_z$).

The dimension relation has a hierarchical structure among a subset of its attributes (as discussed in Section 3.1.1) and it is associated with the attributes hierarchy relation to show 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 One 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. The benefit of this elimination is to derive fact fragments from the minterm predicates of the dimension relation that (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 between the fragments.

To perform this elimination, the following two phases are required:

1. Identify the attribute that has the highest level among the hierarchical predicates.
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.

Algorithm 1 (*OptimizeDimension*) provides a formal presentation of Step One.

Step Two-*Determine the set of dimension minterm predicates.*

The objective of this step is to generate a set of dimension 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 along with its set of implications $I_{D_k} = \{i_1^{D_k}, i_2^{D_k}, ..., i_s^{D_k}\}$.

Algorithm 2 (*DimenMintermPredicates*) provides a formal presentation of Step Two. The input to this step are the set of simple predicates P_{D_k} that has been refined in Step One and the dimension relation D_k. There are three phases required to process this step [10, 2]:

1. Apply the COM_MIN algorithm [10]. The objective of this algorithm is to generate a complete and minimal set of predicates P'_{D_k} from the optimized set of simple predicates P_{D_k}. The COM_MIN algorithm includes, into the set of predicates P'_{D_k}, predicates that partition the relation or fragment into at least two parts which are accessed differently by at least one application.
2. Generate the set of dimension minterm predicates $M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_v^{D_k}, ..., m_z^{D_k}\}$. The minterm predicate

Algorithm 2 DimenMintermPredicates

Input:

- The dimension relation D_k .
- the set of optimized simple predicates for the dimension relation D_k ($P_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_m^{D_k}\}$).

Output:

- A set of dimension minterm predicates $M_{D_k} = \{m_1^{D_k}, m_2^{D_k}, ..., m_v^{D_k}, ..., m_z^{D_k}\}$.
- A set of implication $I_{D_k} = \{i_1^{D_k}, i_2^{D_k}, ..., i_s^{D_k}\}$

Begin
  $P'_{D_k} \longleftarrow$ **COM_MIN**(D_k, P_{D_k})
  Generate the minterm predicates set M_{D_k}
  Determine the set I_{D_k} of implications among $p_j^{D_k} \in P'_{D_k}$
  // Eliminate the contradictory minterm from M_{D_k}
  for each $m_v^{D_k} \in M_{D_k}$ **do**
    if $m_v^{D_k}$ is contradictory according to I_{D_k} **then**
      $M_{D_k} \longleftarrow M_{D_k} - m_v^{D_k}$
  end {end for $m_v^{D_k}$}
  Return(M_{D_k}, I_{D_k})
End {*DimenMintermPredicates*}

($m_v^{D_k}$) is the conjunction of simple predicates that exist in the set of simple predicates $P'_{D_k} = \{p_1^{D_k}, p_2^{D_k}, ..., p_j^{D_k}, ..., p_m^{D_k}\}$ of the dimension relation D_k. Each simple predicate can occur in its natural form or its negated form. The number of minterm predicates in the M_{D_k} set is $z = 2^m$ where m is the number of simple predicates in P'_{D_k} set. The set of minterm predicates defined as follows $M_{D_k} = \{m_v^{D_k} | m_v^{D_k} = \wedge_{p_j^{D_k} \in P_{D_k}} p_j^{D_k *}\}$ where $p_j^{D_k *} = p_j^{D_k}$ or $p_j^{D_k *} = \neg p_j^{D_k}$, $1 \leq k \leq n$ (n 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) and $1 \leq j \leq m$ (m is the number of complete and minimal predicates defined on D_k).

3. Determine the set of implication $I_{D_k} = \{i_1^{D_k}, i_2^{D_k}, ..., i_s^{D_k}\}$ and eliminate the meaningless minterm predicates. The set of implication I_{D_k} will be determined depending on the semantic of the domains and not on the current values of the database. The implication set will reduce the large (exponential) number of minterm predicates by eliminating the meaningless (contradictory) minterm predicates.

Step Two is performed on all the dimension relations that are owners of the fact relation and have applications on them (i.e., set of simple predicates). The determined dimension minterm predicates sets will be collected in one set called $DMset$ where $DMset = \cup_{i=1}^{s} M_{D_i}$, ($s$ is the number of dimension relations that has applications and owner links with the fact relation F_i). The implication sets of the dimension relations will be collected also in one set called $Iset$. Algorithm 4 (*FactHorizontalFragments*) shows the details of generating the $DMset$ and the $Iset$.

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

The objectives of this step are to

1. determine the set of fact minterm predicates $M_{F_i} = \{m_{f1}, m_{f2}, ..., m_{fj}, ..., m_{fz}\}$;
2. eliminate the meaningless (contradictory) fact minterm predicates from the generated set;

Algorithm 3 FactDHorizontal

Input:

- The set of all dimension minterm predicates $DMset$.
- The set of all dimension implications $Iset$.
- The semijoin predicates set $SemiPset$.
- The set of the dimension relations $Dset$
- 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
Generate the fact minterm predicates set M_{F_i}
// Eliminate the contradictory minterm from M_{F_i}
for each $m_{f_j} \in M_{F_i}$ **do**
if m_{f_j} is contradictory according to $Iset$ **then**
$M_{F_i} \longleftarrow M_{F_i} - m_{f_j}$
end {end for m_{f_j}}
// Derived the fact horizontal fragments
for each $m_{f_j} \in M_{F_i}$ **do**
$F_i^j = F_i$
for each $m^{D_k} \in m_{f_j}$ **AND** $D_K \in Dset$ **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}}
Return(f_{F_i})
End {FactDHorizontal}

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

Algorithm 3 (*FactDHorizontal*) provides a formal presentation of Step Three. The input to this step are the set of all dimension minterm predicates $DMset$, the set of all dimension implications $Iset$, the semijoin predicates set $SemiPset$, the set of the dimension relations $Dset$ and the fact relation F_i. There are three phases required to process this step:

1. Generate the set of fact minterm predicates $M_{F_i} = \{m_{f1}, m_{f2}, ..., m_{fj}, ..., m_{fz}\}$. The fact minterm predicate m_{fj} represents the conjunction of the dimensions minterm predicates in $DMset$ that have been generated from Step Two. Each dimension minterm predicate can occur in its natural form or its negated form. The number of fact minterm predicates in the M_{F_i} set is $z = 2^p$ where p is the total number of all the dimension minterm predicates in $DMset$. The set of fact minterm predicates defined as $M_{F_i} = \{m_{fj} | m_{fj} = \wedge_{m_j^{D_k} \in DMset} m_j^{D_{k^*}}\}$ where $m_j^{D_{k^*}} = m_j^{D_k}$ or $m_j^{D_{k^*}} = \neg m_j^{D_k}$, $1 \leq k \leq s$ (s is the number of dimension relations in the dimension set $Dset$), $1 \leq fj \leq z$ (z is the number of fact minterm predicates defined on the dimension relations in $Dset$) and $1 \leq j \leq p$ (p is the total number of all the dimension minterm predicates 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 $DMset = \{m_1^{D_k}, ..., m_z^{D_k}, m_1^{D_l}, ..., m_y^{D_l}, m_1^{D_u}, ..., m_{u'}^{D_u}\}$. As a result, the set of fact minterm predicates will be $M_{F_i} = \{m_{f1}, m_{f2}, ..., m_{fj}, ..., m_{fz}\}$ where $m_{fj} = m_v^{D_k} \wedge m_r^{D_l} \wedge m_s^{D_u}$.

Algorithm 4 FactHorizontalFragments

Input:

- The set of the dimension relations that have applications and owner links with the fact relation F_i where $Dset \subseteq \mathcal{D}$ ($Dset = \{D_1, D_2, ..., D_k, ..., D_S\}$).
- 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 $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$.

Var
$Temp\_DPset$: dimension predicates set.
$DMset$: dimension minterm set.
$Iset$: Implications set.
Begin
for each $P_{D_k} \in DPset$ **do**
P_{D_k} = OptimizeDimension(P_{D_k}, D_{hk})
$Temp\_DPset \longleftarrow Temp\_DPset \cup P_{D_k}$
end {end for P_{D_k}}
// Derived the horizontal fragments of the fact relation
for each $D_K \in Dset$ **AND** $P_{D_k} \in Temp\_DPset$ **do**
DimenMintermPredicates($D_K, P_{D_k}, M_{D_k}, I_{D_k}$)
$DMset \longleftarrow DMset \cup M_{D_k}$
$Iset \longleftarrow Iset \cup I_{D_k}$
end {end for D_K}
f_{F_i} = FactDHorizontal($DMset, Iset, SemiPset, Dset, F_i$)
End {FactHorizontalFragments}

2. Eliminate the contradictory fact minterm predicates. The generated fact minterm predicates will be reduced by eliminating the contradictory fact minterm predicates with the set of implication $Iset$ that have been generated from Step Two.

3. Derive the horizontal fragments set of the fact relation $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$. 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 $L = \{l_1, l_2, ..., l_n\}$; (2) a set of dimension relations $Dset = \{D_1, D_2, ..., D_s\}$; and (3) the fact relation F_i, the derived horizontal fragments of F_i are defined as $F_i^j = F_i \ltimes (\sigma_{m_{fj} \in M_{F_i}}(Dset))$.

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 $M_{F_i} = \{m_{f1}, m_{f2}, ..., m_{fj}, ..., m_{fz}\}$ 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_{fj} is $m_{fj} = m_v^{D_k} \wedge m_r^{D_l} \wedge m_s^{D_u}$ then

$F_i^j = F_i \ltimes (\sigma_{m_{fj} \in M_{F_i}}(Dset))$

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

4.3 Correctness of the Horizontal Fragments of the Fact Relation

This section validates the horizontal fragmentation algorithm of the fact relation with respect to the three correctness rules of fragmentation: completeness, reconstruction and disjointness.

4.3.1 *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 [10] 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 3). Moreover the two functions: *Owner* and *Member* identify the type of the relations through the join link between the relations [10]. This must be completed before the fragmentation design process.

Since the input relations to the fact horizontal fragmentation algorithm follows the referential integrity rule, then the fact derived horizontal fragments are complete.

4.3.2 *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 $f_{F_i} = \{F_i^1, F_i^2, ..., F_i^j, ..., F_i^z\}$ can be reconstructed as follows: $F_i = \bigcup F_i^j$ for $1 \leq j \leq z$.

4.3.3 *Disjointness*

Disjointness ensures that the generated fragments are non-overlapping. The disjointness of the primary horizontal fragments is guaranteed as long as the minterm predicates that determining the fragments are mutually exclusive. The COM_MIN algorithm ensures that the set of simple predicates is minimal (i.e., non-overlap). This, also, ensures that the generated minterm predicates from the set of simple predicates are non-overlapping. Therefore, the primary horizontal fragments are disjoint.

The disjointness of the derived horizontal fragments are guaranteed as long as the minterm predicates that determining the fragments of the owner relation are mutually exclusive. The following points support the claim that the fact table derived horizontal fragments are disjoint:

- Algorithm 3 "FactHorizontalFragments" fragments the fact relation based on fact minterm predicates. These fact minterm predicates generated from all the dimension minterm predicates of the dimension relations that have owner links with the fact relation.
- The dimension minterm predicates of each dimension relation are guaranteed to be non-overlapping and this is taken care of by Algorithm 2 "DimenMintermPredicates".
- The dimension relations themselves are mutually exclusive because each dimension relation represents a specific dimension of the organization (see Section 3).

Therefore, the generated fact minterm predicates are non-overlapping, which ensures that the fact derived horizontal fragments are disjoint.

5 CONCLUSION

This paper presented the framework for applying distributed technology to data warehousing and OLAP systems. We have proposed a system architecture for distributed data warehouse, a formal definition of the relational data model for data warehouse, and a methodology for distributed data warehouse design. The proposed methodology for distributed data warehouse design replicates the dimension relations and provides an algorithm to fragments the fact relation horizontally. The algorithm derives the horizontal fragments of the fact relation based on the applications that are defined on all the dimension relations reference by the fact relation.

Future research in this area includes (1) devising meaningful metrics to judge the performance of the proposed algorithm; (2) developing an algorithm to allocate the generated fragments of the fact relation across the sites of the distributed system. There are many technical issues of distributed data warehouse have yet to be posed and answered.

References

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

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

[3] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, 26(1), March 1997.

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

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

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

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

[8] A.Y. Noaman and K. Barker. Distributed data warehouse architectures. *Journal of Data Warehousing*, 2(2):37–50, April 1997.

[9] A.Y. Noaman and K. Barker. Distributed data warehouse architecture and design. *Fourteenth International Symposium on Copmuter and Information Sciences (ISCIS'99), Kusadasi, Turkey*, oct 18-20 1999.

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

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

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

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