uOttawa

L'Université canadienne
Canada's university

FACULTÉ DES ÉTUDES SUPÉRIEURES ET POSTOCTORALES

uOttawa

L'Université canadienne
Canada's university

FACULTY OF GRADUATE AND POSDOCTORAL STUDIES

Jianwei Wang

AUTEUR DE LA THÈSE / AUTHOR OF THESIS

M.C.S. (Master of Computer Science)

GRADE / DEGREE

School of Information Technology and Engineering

FACULTÉ, ÉCOLE, DÉPARTEMENT / FACULTY, SCHOOL, DEPARTMENT

Composing Dimension and Fact Mappings in Peer Data Warehouses

TITRE DE LA THÈSE / TITLE OF THESIS

Iluju Kiringa

DIRECTEUR (DIRECTRICE) DE LA THÈSE / THESIS SUPERVISOR

CO-DIRECTEUR (CO-DIRECTRICE) DE LA THÈSE / THESIS CO-SUPERVISOR

EXAMINATEURS (EXAMINATRICES) DE LA THÈSE / THESIS EXAMINERS

Liam Peyton

Tony White

Gary W. Slater

Le Doyen de la Faculté des études supérieures et postdoctorales / Dean of the Faculty of Graduate and Postdoctoral Studies

Composing Dimension and Fact Mappings in Peer Data Warehouses

by

Jianwei Wang

Thesis submitted to the

Faculty of Graduate and Postdoctoral Studies

In partial fulfillment of the requirements

For the M.Sc. degree in

Computer Science

School of Information Technology and Engineering

Faculty of Engineering

University of Ottawa

Your file Votre référence
ISBN: 978-0-494-25839-2
Our file Notre référence
ISBN: 978-0-494-25839-2

Dedication

To my parents and my beloved wife

Abstract

Semantic mappings are correspondences that are established between instance or schema level vocabularies of autonomous and heterogeneous data sources. The importance of semantic mappings is steadily increasing in recent data sharing architectures as these mappings enable query answering across heterogeneous boundaries. Peer data management systems (PDBMSs) are one such architecture that has semantic mappings at its core. These systems are made up of fully autonomous network nodes, called peers, which contain data sources to be shared with other peers, called acquaintances. A peer data warehouse (PDW) is a traditional data warehouse (with appropriate dimensions and facts) which is associated with a peer and managed by a PDBMS. We investigate the problem of composing semantic mappings between dimensions and facts of acquainted PDWs. Moreover, we give a distributed algorithm for composing these mappings expressed in the mapping language LAV (Local-As-View). Furthermore, we show that the complexity of the algorithm is quadratic and prove the correctness of the latter. Finally, we conduct several experiments that illustrate the robustness of the algorithm.

Acknowledgments

First of all, I would like to gratefully thank my supervisor Dr. Iluju Kiringa for his professional guidance and academic support. Every time when I make progress on this thesis, he accurately points out the next step that I should go to. He is a great researcher. A special thanks goes to Dr. Tony White and Dr. Liam Peyton for being the examiners. They took their time to make valuable suggestion to improve this thesis. Moreover, I also would like to thank my beautiful wife for her love and remote support though she is currently thousands of miles away from me. Finally, I do not forget the constant encouragement from parents.

Contents

List of Figures

Chapter 1

INTRODUCTION

Over the last two decades, data sharing between several, possibly heterogeneous sources has been a recurrent problem in many data management systems. Multidatabase systems [17] are the classical example of systems built around the idea of data sharing. Data integration systems [15] are another well-known data sharing architecture. Furthermore, more recently, *peer data management systems* (PDBMSs) [4, 19, 2, 22] have been proposed as an architecture for data sharing that has semantic mappings at its core. Semantic mappings are correspondences that are established between instance [2] or schema [22] level vocabularies of autonomous and heterogeneous data sources. "A data source is a system from which data are collected, in order to be integrated into the data warehouse" [14]. An autonomous data source is a data source that is independently and functionally operational. Heterogeneity of data sources may occur in various forms in terms of languages, data structures, operations, managements and etc. The PDBMSs are made up of fully autonomous network nodes, called *peers*, which act as data sources (called peer databases) shared with other peers, called *acquaintances*. Over

the last years, mappings between data sources have been studied with a focus on appropriate mapping languages and query processing algorithms using these mappings to enable data sharing over heterogeneous acquaintances. Finally, peer databases that are data warehouses, called *peer data warehouses* (PDWs), have been modeled in [8]. Here, a model for multidimensional data scattered in a peer-to-peer (P2P) network is presented, along with a query processing mechanism that propagate Online Analytical Processing (OLAP) queries throughout such a network.

This thesis presents a technique to deal with dimension mappings and fact mappings when a new PDW is added to a network of PDWs. A PDW system (PDWS) is a collection of PDWs. In a PDWS, each peer is associated with a data warehouse. The latter is fully autonomous and is run by an independent DBMS. Furthermore, each PDW has its own data sources and Extraction Transformation Loading (ETL) process. A PDW is organized in *data marts*, *dimensions* and *facts* in the usual way of traditional data warehouses [14] and, as such, is a traditional data warehouse that exists at a peer in the P2P network. A dimension is an entity that describes a real-world domain. A dimension consists of a set of attributes, called *hierarchy levels*. Hierarchy levels of a dimension are placed in a partial order. A fact describes the factual information of an organization process by numeric representations (real numbers or integers). A fact consists of a set of numerical attributes, called *fact measures*, and a set of attributes that are associated with a set of dimensions. Fact measures of a fact are additive or semi-additive. A data mart is the description of an organization process; it is a fundamental unit of a data warehouse. In general, a data mart consists of a set of dimensions and a set of facts. In

this thesis, we use the *star schema* [14] to represent a data mart. A star schema is the structure of a multidimensional relational model; it allows us to store data in a "star" structure to describe an organizational behavior; it consists of one central fact and a set of dimensions. For simplicity, we assume that the multidimensional models of the PDWs are isomorphic even though those PDWs might have different sources, ETL process, or schemas. An isomorphism [11] between the structures of any two of those multidimensional models shows that the two multidimensional models are structurally identical. In other words, the structures of the two multidimensional models can be mapped onto each other, in such a way that to each part (dimension, fact or data mart) of the structure of one multidimensional model there is a corresponding part in the structure of the other multidimensional model. Those two parts play similar roles in their respective structures of the two multidimensional models. In a PDWS, data sharing occurs through OLAP queries that are propagated along the acquaintances to collect answers.

We consider two problems that are closely related. The first problem is the DIMENSION MAPPING COMPOSITION problem. That is, given complete dimension mappings between PDWs p_A and p_B, and between PDWs p_B and p_C, build direct dimension mappings between p_A and p_C that are also complete. Completeness of a mapping between dimensions d_1 and d_2 means that all the hierarchy levels of dimensions d_1 and d_2 appear exactly once in the dimension mapping of d_1 and d_2. Intuitively, we should determine the completeness of dimension mappings prior to dimension mapping composition. That is, given two dimensions of two PDWs that get acquainted and a mapping of those two dimensions, we must determine whether this dimension mapping is complete

with respect to all the hierarchy levels of the two dimensions. The second problem is the FACT MAPPING COMPOSITION problem. That is, given complete fact mappings between PDWs p_A and p_B, and between PDWs p_B and p_C, build direct fact mappings between p_A and p_C that are also complete. Similarly, completeness of a mapping between facts f_1 and f_2 means that all the fact measures of facts f_1 and f_2 appear exactly once in the fact mapping of f_1 and f_2. We also should determine the completeness of fact mappings prior to fact mapping composition. That is, given two facts of two PDWs that get acquainted and a mapping of those two facts, we must determine whether this fact mapping is complete with respect to all the fact measures of the two facts.

Mappings that are used in current PDBMSs for the purpose of data sharing are established either only at the instance level or only at the schema level. However, complete mappings used in a PDWS are built at both instance and schema levels. Mapping composition [18] is a technique used in a P2P system when mappings are built based on the existing mapping information. This technique deals with the case in which the existing mapping information is retrieved from the acquaintances and a new mapping is composed at run time when a query is posed against a local peer. We use a similar concept of mapping composition in the thesis. However, in a PDWS, new mappings are composed when a new PDW is added to the system; furthermore, the problem of mapping composition becomes more complicated because it is processed at both instance and schema levels.

1.1 Motivating Example

Consider the following simple scenario. There are two universities located in the same city. They offer similar programs separately and plan to propose new joint programs in which the courses are offered by both universities. Students entering the programs are from both universities, and two corresponding faculties are responsible for these programs. Both universities have autonomous centralized data warehouses to independently evaluate education quality and performance. These two data warehouses both have relational multidimensional models and conform to the usual star schema. Therefore, we must deal with dimension mappings and fact mappings. In order to evaluate the new joint programs properly and accurately, we may need to modify the architectures of the two autonomous data warehouses so that they can communicate with each other and exchange academic information.

One possible solution would be to build a centralized data mart to store the information carried from the joint programs. In this case, the former data marts storing the information of the programs in the two autonomous data warehouses would be no longer in use, and the cost of building a new data mart would be expensive. Moreover, even though the new data mart was physically located in either university, users at the other university would remotely connect to the data mart, and the communication cost would be expensive too. Also, the remote connection would greatly reduce the performance of the systems due to the overhead of the amount of data transferred through the network when such OLAP operations, like *roll-up*, *drill-down* and etc are performed. Overall, such a new data mart would violate the purpose of building a data warehouse for effec-

tive decision-making support.

An alternate solution would be to add a schema mediator between those two data marts [12]. The schema mediator only stores schemas, possibly common schemas, rather than instances of the two data marts. Every query regarding the joint programs is answered through the schema mediator. In this case, the schema mediator needs to justify which part of a query should be answered at which data mart, then waits for the answers from both data marts, and finally returns a complete answer. This schema mediator would badly affect the performance of data warehousing since there was no actual data stored in the schema mediator. Again, this solution would violate the purpose of effectiveness of data warehousing.

Figure 1.1 shows that our solution is to keep the two data warehouses as they were, and then to build mappings at both instance and schema levels between the dimensions and facts of both data warehouses. That is, we treat both data warehouses as PDWs participating in a PDWS. Both data warehouses are not physically merged. The information stored in the two PDWs regarding the new joint programs is still in use. Mappings are built at both PDWs which are local. Dimension and fact information is propagated from the local PDW of either university to the other. All queries regarding the joint programs against either PDW are first posed locally and then propagated through the P2P network to the other PDW.

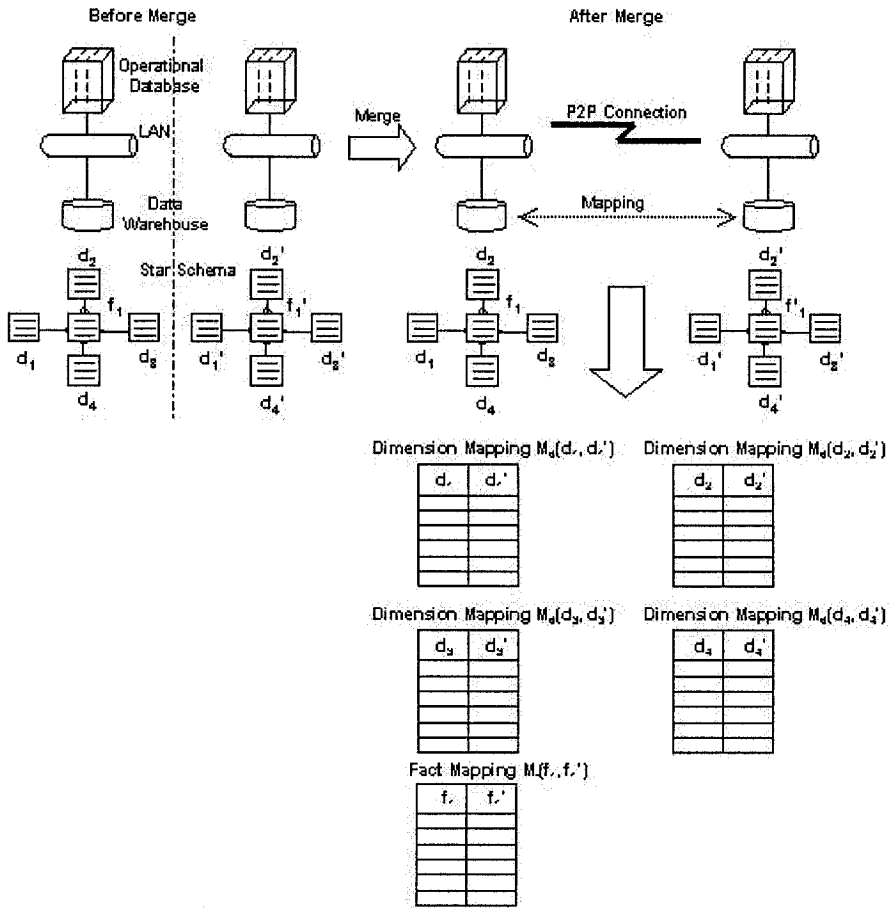


Figure 1.1: System Architecture

1.2 Problem

Consider again our running scenario. The PDW at University A has a data mart called *Enrollment*, where students registration information is stored. The data mart consists of *Students*, *Courses* and *Professors* dimensions, and *Grades* fact. The *Students* dimension has hierarchy levels *StudentID*, *Program*, *Department* and *Faculty*; the *Courses* dimension has hierarchy levels of *CourseID*, *Type*, *Program*, *Department* and *Faculty*; the *Professors* dimension has hierarchy levels *ProfID*, *Title*, *Department* and *Faculty*; the *Grades* fact

has an additive fact measure *Grade* and a semi-additive fact measure *Rank*. The PDW at University B has a data mart called *Registration*, where students registration information is stored. The data mart is similar to the data mart of the PDW at University A, and consists of *Student*, *Course* and *Professor* dimensions, and *Marks* fact. However, the hierarchy levels of *Student* and *Course* dimensions are slightly different. In the *Student* dimension, it has an attribute called *Option* indicating which program option a student majors in. At University A, on the other hand, students do not have program options. In other words, the students there have only one option, and it is described in their programs. The same situation happens in the *Course* dimension. The *Course* dimension has an attribute called *Option* explicitly indicating which program option a course is required by. When a student in a joint program from University A wants to register in a course offered by University B, the student only needs to register with the academic unit at University A (and vice versa). Figure 1.2 lists the dimension hierarchy levels of the dimensions of the two data marts.

Figure 1.3 gives sample dimension mappings for our university PDWS scenario. We have three mappings: the *Students* dimension is mapped to the *Student* dimension, the *Courses* dimension is mapped to the *Course* dimension, and the *Professors* dimension is mapped to the *Professor* dimension. The mapping of the *Students* and *Student* dimensions has four pairs of hierarchy levels, (*StudentID, SID*),(*Program, Prog*), (*Department, Dept*) and (*Faculty, Faculty*); the mapping of the *Courses* and *Course* dimensions has five pairs of hierarchy levels, (*CourseID, CID*), (*Type, Type*), (*Program, Prog*), (*Department, Dept*) and (*Faculty, Faculty*), the mapping of the *Professors* and *Professor* dimensions

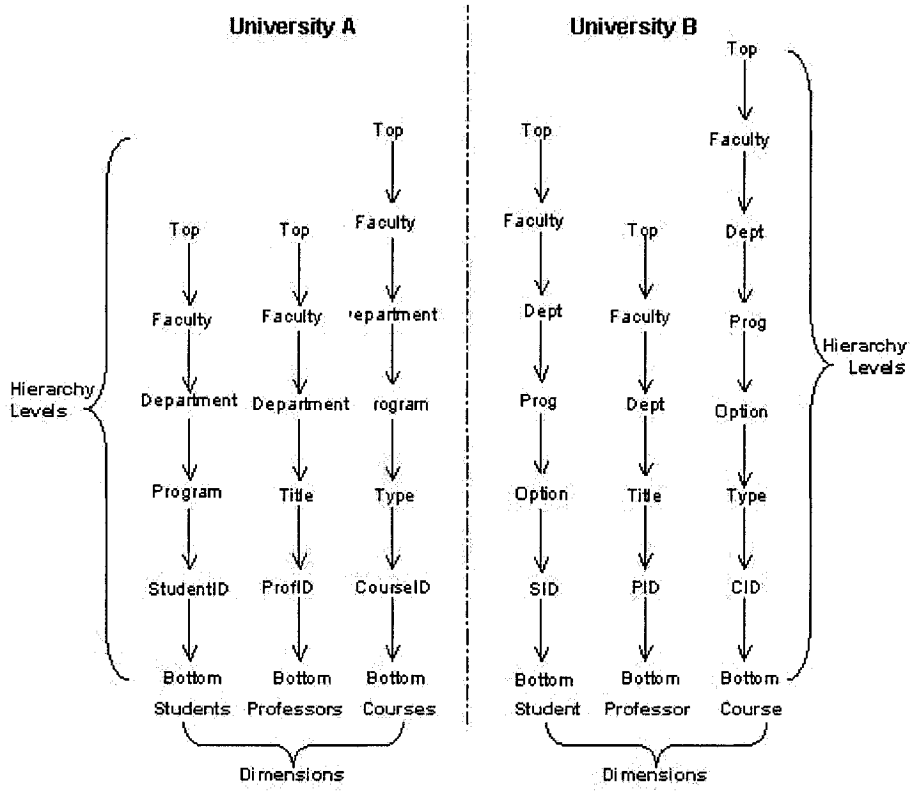


Figure 1.2: Dimension Hierarchy Levels

has four pairs of hierarchy levels, (*ProfessorID, PID*), (*Title, Title*), (*Department, Dept*) and (*Faculty, Faculty*).

Since those two data marts record the registration information of the joint programs, they certainly contain similar academic information regardless of the differences between students and between academic policies in both universities. However, each university only has the registration information regarding its own students. Thus, if a secretary wants to have complete information regarding all the students who registered in a course of the joint programs, a query is posed to retrieve both local data and remote data

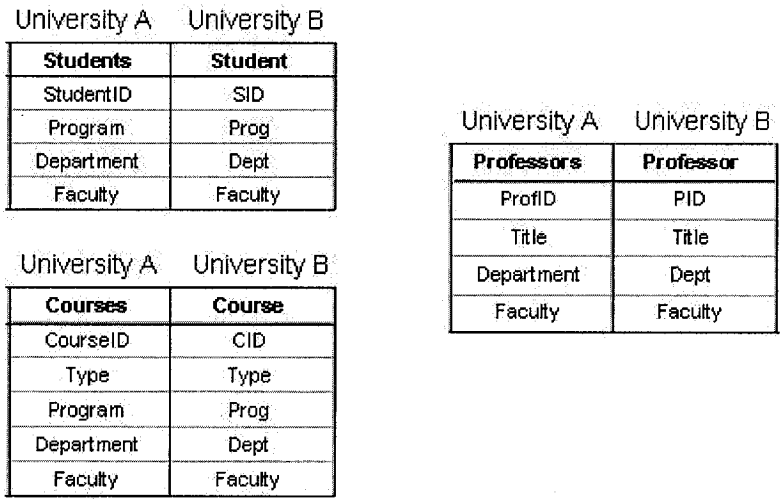


Figure 1.3: Dimension Mappings

through the mappings that are in place between Universities A and B.

Now suppose the secretary wants to find the average grade of each course in each term grouping by those students who are from different program options. Since the *Students* dimension in the data mart at University A does not have the *Option* attribute that represents different program options, the query cannot be answered directly from the data warehouse at University A. However, the query can be partially answered from the data warehouse at University B because the *Student* dimension in the data mart at University B does store such information. Therefore, one solution would be to just have an incomplete answer. Nevertheless, if possible, we would prefer to get the complete answer for such a query. In order to achieve this, we need to determine whether the existing dimension mappings are complete. If any one of them is incomplete, then we need to "rebuild" it by building *extra dimension mapping tables* to make it complete. Extra di-

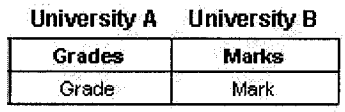


Figure 1.4: Fact Mappings

mension mapping tables of an incomplete dimension mapping are the supplement of the incomplete dimension mapping and contain the missing hierarchy levels at the instance level with respect to the two dimensions of the incomplete dimension mapping.

Consider a second scenario in which the grades information is stored at both PDWs, but the *Grades* fact in the PDW at University A has two fact measures *Grade* and *Rank*, whereas the *Marks* fact in the PDW at University B has only one fact measure, namely *Mark*. Then, Figure 1.4 shows that we have a fact mapping in place. This fact mapping has one pair of fact measures, (*Grade, Mark*). If we want to have a complete mapping between those two facts, we have to deal with the dangling fact measure *Rank*, meaning that if possible, we need to build an *extra fact mapping table* to represent the ranking information in the PDW at University B. Extra fact mapping tables of an incomplete fact mapping are the supplement of the incomplete fact mapping and contain the missing fact measures at the instance level with respect to the two facts of the incomplete fact mapping.

Finally, consider a third scenario in which a new PDW at University C is added to the system. Then, we need to build both dimension mappings and fact mappings either between C and A, or between C and B. Obviously, the designers must make a decision on

between which PDWs the new PDW should be acquainted. It might be the case that C gets acquainted with A. Then, we have dimension mappings and fact mappings in place between C and A, but do not have dimension mappings or fact mappings between C and B. Hence, we can build dimension mappings and fact mappings between C and B based on the mappings between A and B, C and A. That is, we need to deal with the problem of composing the existing mappings to obtain the mappings between B and C.

1.3 Solution

We have made the following assumptions for our solution. We first assume that all the PDWs in our examples are capable of OLAP operations, and the relational multidimensional models in the centralized data warehouses are represented by the star schema. Moreover, different PDWs might have different vocabularies referring to the same semantics, and even the same schema might have different meanings in terms of the semantics. We can use either of two fundamental mapping languages, *Local-As-View*(LAV) and *Global-As-View*(GAV) [15], at the schema level to solve the schema heterogeneity. In a LAV mapping, a local source is expressed as a view over the global schema. In a GAV mapping, on the other hand, the global schema is expressed as a collection of views over the local sources. In this thesis, we assume the LAV approach to the schema mapping problem. The LAV approach has the advantage of allowing easy system extension, meaning that adding a new data source to system does not affect other data sources at the schema level. Compared to the LAV approach, the GAV approach has some difficulties of extending a system, meaning that the global schema might indeed be affected whenever a new data source is added. Furthermore, the reliability of a network is not one

of our concerns in the thesis, so we again simply assume that the network is reliable, meaning that any message from one PDW to another will be properly delivered with a finite delay. Finally, a link between two PDWs in the network is bidirectional, meaning that a message can be sent from one end to the other of the link, or in the opposite direction.

We solve the two closely-related problems of dimension mapping composition and fact mapping composition in a distributed computing fashion, thus the complexity will be evaluated by counting the number of messages exchanged over the network. We give a unified algorithm for both the DIMENSION MAPPING COMPOSITION and FACT MAPPING COMPOSITION problems. Starting with two PDWs, we add PDWs one by one to the system. Once a PDW is added, we build dimension mappings as well as fact mappings between it and all acquainted peers. Then, we examine the completeness of those dimension mappings and fact mappings. Finally, we build dimension mappings and fact mappings for those PDWs that are not acquainted with the added PDW.

Our solution consists of three parts. The first part is the main function written in a distributed computing way to deal with the case in which a PDW is added to the system. That is to notify the system of the event of adding a new PDW and to exchange messages between PDWs so that local computations at the added PDW can be followed. The second part is made of two local functions to determine whether a given dimension mapping is complete and whether a given fact mapping is complete. If the given dimension mapping is incomplete, then for each hierarchy level of both dimensions that does

not appear in the dimension mapping, we need to build an extra mapping table for every one of those missing hierarchy levels to make the dimension mapping complete. On the other hand, similar steps will be followed to check the completeness of the given fact mapping. The third part is made of two local functions to deal with dimension mapping composition to generate dimension mappings as well as to deal with fact mapping composition to generate fact mappings based on the complete mapping information obtained from the added PDW and its acquainted PDWs.

We have observed that building extra dimension mapping tables to get a complete dimension mapping is not always possible because we have to examine semantics of instances, meaning that such a process will be successful only if the semantics of the instances allow us to do so. The same observation we have obtained from building extra fact mapping tables to get a complete fact mapping. Furthermore, although we use the LAV data integration approach to generate dimension mappings and fact mappings,we redefine the data structures of both mappings because complete mappings (either dimension mappings or fact mappings) are not only at the schema level but also at the instance level.

The rest of the thesis is organized as follows. In Chapter 2, we first summarize the necessary background related to data warehousing as well as P2P computing and then present and discuss the related work. Chapter 3 defines a language for precisely formulating the DIMENSION MAPPING COMPOSITION and FACT MAPPING COMPOSITION problems. Chapter 4 presents an algorithm for determining the completeness

of dimension mappings and of fact mappings. Furthermore, Chapter 4 deals with an algorithm for composing dimension mappings and fact mappings, and gives the proof of the correctness of the algorithm as well as its complexity. Chapter 5 gives performance evaluations for our algorithm by showing several experimental results obtained from an implementation of a simulated PDWS. Finally we conclude and point out some directions for further research on the topic of peer data warehousing.

Chapter 2

BACKGROUND AND RELATED WORK

This chapter first summarizes the related background including data warehouses, distributed computing and peer to peer (P2P) systems. Then, it discusses the related work including data integration, mapping composition, dimension and fact mappings as well as other necessary related work.

2.1 Background

2.1.1 Data Warehouses

Most commercial data warehouses use the star schema modeling technique to represent the multidimensional data models. A usual star schema represents a data mart, which is the fundamental unit of a data warehouse and is capable of OLAP operations. Each star schema consists of a single fact and a set of dimensions that are all associated with

the fact. Such an association is represented by a pair of primary key and foreign key to enforce referential integrity.

In this section, we present a version of the multidimensional conceptual data model (MCDM) [7, 14]. MCDM is an incarnation of the star schema modeling technique. MCDM has two fundamental components, the *dimension* and the *fact*. A dimension describes a real-world domain in the form of a relation that is composed of attributes. Attributes of a dimension can be the dates of a year, the students at a university, or the courses offered at a university. Each dimension has a set of hierarchy levels including two constant levels, namely the top and the bottom. Each attribute that belongs to a dimension lies between the top and the bottom levels, and the sequence of the attributes is based on their granularity (for instance, students can be organized into programs, departments and faculties). Moreover, in the dimension, the attributes are related to each other by two functions called *roll-up* and *drill-down*, which are two OLAP operations. The *roll-up* function gives access to attributes with a higher granularity; the *drill-down* function does the reverse. A fact represents an organization process by fact measures and associates the latter with a set of dimensions. For example, a university data warehouse may have a fact called *Grades* that is associated with the *Students*, *Courses* and *Professors* dimensions. Finally, a data mart that usually describes an organization process is the smallest unit of a data warehouse, and is functionally operational to end-user applications. Examples of organization processes are: the students registration information for each term at a university, the monthly sales at a corporation, and the passengers daily flight activities. A typical data warehouse is made up of a collection of data marts.

Now, we give formal and theoretical definitions for dimensions, facts as well as data marts.

Definition 1 (Dimension) *A dimension d is an entity that is represented by a relation. The relation is composed of a surrogate key (primary key that uniquely identifies each tuple) and a finite set of attributes* $L(d) = \{l_1, l_2, ...l_m\}$ *that makes the hierarchy levels in a partial order* $\rightarrow_d$ *on* $L(d)$. *If* $l_1 \rightarrow_d l_2$, *then* l_1 *rolls up to* l_2 *and* l_2 *drills down to* l_1. *By default,* $L(d)$ *includes the top and the bottom elements.*

Let $m : L(d) \rightarrow L(d)$ be a function. For each pair of attributes of a dimension d, if $l_1 \rightarrow_d l_2$, then the following holds for every pair of instances (i_1, i_2):

(1) If i_1 is an instance of l_1, then there is an instance i_2 of l_2 such that $m^{l_1 \rightarrow l_2}(i_1) = i_2$;

(2) If i_2 is an instance of l_2, then there is at least one instance i_1 of l_1 such that $m^{l_1 \rightarrow l_2}(i_1) = i_2$.

The *roll-up* and *drill-down* functions are the two examples of the function m. Therefore, such hierarchy levels are capable of using the *roll-up* and *drill-down* functions along the hierarchy in a dimension. Moreover, the common elements of the top and the bottom guarantee that the *roll-up* and *drill-down* functions will always be well defined for each hierarchy level in a dimension. In a dimension, the base level (excluding the bottom) in $L(d)$ defines the granularity of the dimension. Moreover, we presume that the values of a surrogate key of a dimension are a sequence of natural numbers (1,2,3,...) starting from 1. The values of a surrogate key have no meaning. They are absolutely nothing about the underlying tuples of a dimension. The only purpose of using surrogate key for a dimension is to define uniqueness in a dimension.

Definition 2 (Fact) *A fact f is a relation over a set D of dimensions. It is a pair $(M,FK(f))$ such that $M = \{\langle m_1, ..m_q\rangle\}$ is a set of fact measures and $FK(f) = \{\langle l_1, ..., l_p\rangle\}$ is a set of attributes (recognized as foreign keys) associated with dimensions in D. Each $m_j \in M$ is a distinct fact measure, and each $l_i \in FK(f)$ is a distinct attribute of some dimension in D.*

Most fact measures are additive, meaning that simple mathematical operations such as summation, average and etc can be applied. The rest fact measures are semi-additive, meaning that those fact measures are numbers (real numbers or integers) but normally not computable. A fact measure represents factual data over a set of dimensions. In a fact, the granularities of the set of associated dimensions determine the granularity of the fact.

Definition 3 (Data Mart) *A data mart consists of a set D of dimensions and a set F of facts over the dimensions in D. Each data mart represents a business process. The context and semantics of a data mart are defined by the granularity of the data mart. The granularity of a data mart is the base level of information that the data mart can provide. In a data mart, the granularities of the dimensions and the fact define the granularity of the data mart.*

2.1.2 Distributed Computing

"A distributed computing environment consists of a finite collection of computational units, called entities, communicating by means of messages [20]." The collection of entities work together by the way of messages exchange between each other for a common

target, such as electing a leader, constructing a spanning tree, computing the diameter of a network, counting the number of nodes in a network and etc.

The following rules are applied in a distributed computing environment to ensure the quality of a goal that is going to be achieved.

- Reliability. There is no failure occurred during the process of message transmission. Every message sent by an entity will arrive at its desired destination with a finite delay. Furthermore, the content of every message is undamaged.
- FIFO queuing policy. All the messages from a single entity to a single destination will arrive in the same order they were sent.
- Bidirectional links. A message can be sent through a link from one entity to the other, or in the opposite direction.
- Connectivity. The communication topology is strongly connected. From any node in a network, all the other nodes are reachable.

A typical distributed computation model is composed of entities, external events, actions and behaviors. An entity is capable of the operations of local storage, local processing and message transmission. Each entity has a collection of status registers that indicate the current state of the entity. The value of a status register is obtained from a pre-defined finite set, such as S = {INITIATOR, SLEEP, AWAKEN, IDLE, WAITING, TERMINATION, ...}. An entity that is in the INITIATOR state is a special entity starting a computing process spontaneously; an entity that is in the TERMINATION state finishes the computing process and will no longer exist in the environment. Obviously,

before a computing process starts, there can exist one or more than one entity that are in the INITIATOR state; when the computing process finishes execution, all the entities in the environment are in the TERMINATION state. The operation of changing the state of an entity is included in the operation of local processing. External events of an entity are the event triggers that come from the outside of the entity in the environment, and includes message arrival, spontaneous compulsion and etc. An action of an entity takes place locally when an external event occurs. The action is atomic, meaning that when the action gets started, it either finishes successfully or terminates without any changes. A behavior of an entity is a rule form defined as follow:

$$\text{Status x Event} \longrightarrow \text{Action} \quad [20]$$

This rule form that involves a status, an external event and an action is interpreted as, in a distributed computing environment, when a specific external event occurs, an entity that is in a specific state takes a specific action.

Distributed computing is measured by the amount of communication activities, often known as the number of messages transfered over the environment, and the total execution time of a distributed computation, known as the difference between the start time and termination time of the computing process.

We now highlight some fundamental distributed computing problems and methods that are used in our algorithm as follows:

- The BROADCASTING problem:

  Given a set of entities and a communication topology, define a collection of rules

such that every entity eventually knows the specific information no matter which entity has this information initially. One of the solutions is FLOOD when the restrictions of reliability and connectivity are applied.

- The SPANNING TREE problem:

  Given a set of entities and a communication topology, define a collection of rules such that every entity has a subset of its neighbors as Tree-Neighbors and that the collection of the corresponding links are the links of a spanning tree. One of the solutions is SHOUT, which is the combination of FLOOD and REPLY. In the REPLY method, when an entity receives a query message Q, it always sends an acknowledgment (YES or NO) to the sender.

- The CONVERGE CASTING method:

  In a rooted tree, a leaf sends its message to its parent; each internal node waits until it receives a message from all its children, and then it sends a message to its parent; the root eventually gets the expected information. This method combining with SHOUT as well as FLOOD can be used to solve problems such as counting the number nodes in a network, finding the center of a network, computing the diameter of a network and etc.

2.1.3 P2P Systems

A P2P system is a decentralized collection of computers providing their local resources to everyone in a network and focuses on a specific application domain. In a P2P system, peers rely on each other for services, rather than on the centralized service provider. Each peer in the system plays a role both as service provider and as service consumer. Most

P2P systems are for the purpose of data exchange and data sharing without centralized data sources. In a pure P2P system, there is no centralized coordinator to coordinate data exchange and data sharing between peers and acquaintances. Moreover, there is no centralized data source, and each peer has its own data repository. Furthermore, there is no global schema, and each peer has its own local schema. Finally, peers are autonomous and are independently responsible for maintenance of their own data and schema.

Mapping: Due to the heterogeneity of data stored at different peers in a P2P network, normally the data residing at a peer cannot be directly used by other peers. We must provide an appropriate and efficient mechanism to solve the problem of heterogeneity so that the data can be shared among peers. Building mappings between peers is a widely-used solution. For example, Universities A and B offer some similar programs, but the programs offered at University A have different names from the programs offered at University B. If there are two different programs' names from the two universities respectively refer to the same program, then we build a mapping of those two programs so that the systems at both universities can recognize both of them. Obviously, we must carefully study the semantics of the data before building the mapping. Therefore, such mappings may be built at the instance level. Now, we give a formal definitions for mappings as follow:

Definition 4 (Mapping) *Let X and Y be nonempty disjoint sets of attributes. A mapping is a tuple (x, y) that indicates $x \in X$ is associated with $y \in Y$* [13]. *Furthermore, any attribute in X is associated with at most one attribute in Y, and vice versa.*

Definition 5 (Mapping Table) *Let X and Y be nonempty disjoint sets of attributes.*

A mapping table m from X to Y is a finite set of mappings, $\{(x, y) \mid x \in X, y \in Y\}$*, such that each attribute of X and Y appears in at most one mapping* [13].

Mapping Composition: In a P2P network, when a query is posed against a local peer, one of the solutions is to chain mappings together at query time [18]. Chaining mappings at run time might be overwhelmed by finding all possible paths in the network. Moreover, some of the mapping chains might be redundant, or some of them might not favor efficient query execution planning. Mapping composition in P2P systems is a pre-computing technique that favors run time savings and removes redundancies from the mapping chains. On the other hand, composition is one of the basic operators in model management algebra [3]. Mapping composition at the schema level is desirable to make meta-data management tasks much easier to achieve higher performance of the P2P systems.

Definition 6 (Mapping Composition) *Given three data sources A, B and C, as well as two mappings* $M(A, B)$ *and* $M(A, C)$*, A mapping composition algorithm is to yield a direct mapping* $M(B, C)$ *that is equivalent to* $M(A, B)$ *and* $M(A, C)$*. We have the following composition formula:*

$$M(A, B) \circ M(A, C) = M(B, C)$$

"Equivalence means that for any query $q \in Q$, where Q is a given set of queries, and for any instance of the data sources, the answer obtained by the direct mapping is the same as the answer obtained by the two original mappings [18]."

2.2 Related Work

2.2.1 Data Integration

Lenzerini gives a tutorial to review the two basic modeling techniques used today of data integration, LAV and GAV [15]. In this section, we revisit those two mapping languages and highlight their advantages as well as disadvantages.

Suppose that we have a data integration system $\mathcal{I} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$ [15]. $\mathcal{G}$ is the global schema; $\mathcal{S}$ is the source schema; $\mathcal{M}$ is the mapping between $\mathcal{G}$ and $\mathcal{S}$.

The LAV approach specifies that every local source is expressed as a view over the global schema. In other words, a LAV mapping is a set of assertions of the form:

$$s \rightsquigarrow q_{\mathcal{G}} \text{ [15]},$$

where s is an element of the source schema $\mathcal{S}$ and $q_{\mathcal{G}}$ is a query over the global schema $\mathcal{G}$. The LAV approach favors extensions of the data integration system $\mathcal{I}$: adding a new source is equivalent to modifying the mapping by adding a new assertion, and there are no other changes. However, query answering may become more complicated because the problem of processing a query in a LAV data integration system is the well-known problem of view-based query processing.

On the other hand, the GAV approach specifies that the global schema is expressed as a collection of views over the local sources. That is, a GAV mapping is a set of assertions of the form:

$$g \rightsquigarrow q_S \text{ [15]},$$

where g is an element of the global schema $\mathcal{G}$ and q_S is a query over the source schema $\mathcal{S}$. The GAV approach favors query answering because it directly shows how to retrieve data from the sources. However, adding a new source may indeed affect the global schema. As a result, modifying the global schema may not be so easy because the existing parts of the global schema may also be impacted.

2.2.2 Composing Peer Mappings

In this thesis, we use the concept of *mapping* from [13], but our notation of complete mappings is different from the notation of mappings in [13]. On the other hand, the process of mapping composition we address bears similarities with the one studied in [18].

Kementsietsidis et al treats mappings as constraints on the exchange of information among peers and introduces a language to formulate the constraints [13]. Given a set of mapping constraints, the authors are able to obtain new mapping constraints and check the consistency of the new mapping constraints. Mapping constraint formulas defined by the language allow the authors to combine mapping constraints by using logic operations, such as conjunctive, disjunctive and negation. Furthermore, they study the decision problem of mapping constraint consistency and give an algorithm for the latter. The problem of mapping constraint consistency is NP-complete. The algorithm is a distributed algorithm. However, it has two disadvantages. The first disadvantage is that the algorithm is not efficient since it does unnecessary computation such as redundant work on checking the consistency of mapping constraints; the second is that deadlocks

may occur during the computation.

Moreover, Kementsietsidis et al address semantic and algorithmic issues raised in building mappings in P2P systems. The mappings described in [13] are only at the instance level. That is, the authors are concerned only with mappings between data values. On the other hand, the complete mappings described in the thesis are at both instance and schema levels. That is, we are concerned with both instance and schema level complete mappings. To obtain a complete dimension(or fact) mapping, we might need to deal with two dimensions(or facts) from two PDWs respectively. That is, we need to generate queries over the dimensions (or facts) and to execute those queries at one or both PDWs to get the supplemental mapping information.

In [18], the authors consider the problem of composing semantic mappings. Given mappings between data sources A and B, and between B and C, they present an algorithm to find a direct mapping between A and C such that, for any query over the three data sources, the direct mapping between A and C yields the same answer as the original mappings do. In their algorithm, they define the minimal mapping composition formula such that none of other mapping composition formulas (possibly combined) that yields the same result is smaller than the minimal mapping composition formula. Moreover, they study a particular mapping composition problem for a subclass of conjunctive queries. Given a constant integer k, every expression in the subclass of conjunctive queries has maximum k variables. They then show that this particular mapping composition problem is $\prod_2\mathbf{P}$(It is equivalent to $\mathbf{coNP^{NP}}$: Complement of $\mathbf{NP}$ with an oracle of $\mathbf{NP}$).

Furthermore, in [18], mapping composition is achieved at query processing time. The authors assume that all peers have been already in a network and that mappings between two peers have been built. Then, the process of mapping composition starts when a query is posed against a local peer. On the other hand, in our solution, mapping composition is achieved at construction time. We start with two PDWs in a network. We assume that dimension mappings and fact mappings between two PDWs are already in place. Then, we add the third PDW, and assume that dimension mappings and fact mappings between the added PDW and the existing PDWs are manually built by the designers. After that, we come up with the problems of dimension mapping composition and fact mapping composition which can be done automatically based on the existing dimension mappings and fact mappings. We continue this process until all the PDWs are added to the network.

2.2.3 Mapping Dimensions and Facts in Peer Data Warehouses

To the best of our knowledge, the problem of dimension mappings and fact mappings in a PDWS was first and only addressed in [8]. The problems of DIMENSION MAPPING COMPOSITION and FACT MAPPING COMPOSITION are partially similar to the mapping composition problem stated in [8]; however, our approach is quite different from theirs. We first summarize their contributions and then give the major differences between our solution and theirs.

Vaisman et al give a systemic revise and map strategy for defining an instance of a

dimension in an acquaintance from the local peer's point of view [8]. The revision phase redefines the instance of the acquaintance dimension such that the instance of the acquaintance dimension adapts the meaning of the local dimension. The map phase checks the consistency of every mapping with respect to the revised instance. Furthermore, they define the class of P2P-OLAP queries and present a rewriting technique for those queries. A P2P-OLAP query over a multidimensional data model involves a set of dimensions, a set of facts and a collection of OLAP operations. The set of dimensions and the set of facts are distributed in a P2P network. The revise and map strategy favors the query rewriting process because we can follow the corresponding revision formulas to rewrite the queries whenever a revision occurs.

In [8], for two given data warehouses, the authors give a common name for two dimensions that refer to the same meaning, and then give a common name for every two corresponding dimension hierarchy levels. Moreover, the authors also give a common name for two facts that refer to the same meaning, and then give a common name for every two corresponding fact measures. Those common names of dimensions and facts therefore are used in queries posed against a acquainted data warehouse; so are the common names of hierarchy levels and fact measures. Consider a scenario where some applications rely on the old schemas. Name modifications would heavily affect the use of those applications; even worse, those applications might be partially re-developed. As a result, the cost of re-development would be unexpectedly high. By contrast, in our approach, we do not explicitly give a common name for two dimensions (or facts) if those two dimensions (or facts) refer to the same meaning; neither do we explicitly give

a common name for two corresponding dimension hierarchy levels (or fact measures) if those two dimension hierarchy levels (or fact measures) refer to the same meaning. As a result, old applications that rely on each individual centralized data warehouse run as usual since we do not change the old schemas of the data warehouses, and there is no extra cost of rebuilding those applications.

2.2.4 Further Related Work

Recently, much research work has focused on the issues of data sharing. We summarize and briefly discuss the major related work as follows.

Firstly, Gribbe et al were the first to address research directions related to database management systems [10]. They mainly introduce the dynamic data placement problem: "data must be placed in strategic locations and then used to improve query performance." They classify that each peer in a P2P network may play as many or all of four roles as data origin (the source of data), storage provider (where data physically stores), query evaluator (it consumes the CPU to create query execution plans) and query initiator (where queries are initially posed). Serafini et al propose the first data model, called *Local Relational Model* (LRM), for peer database management systems [4, 21]. In the LRM, they define translation rules between data items in a relational space(a set of relational peer databases) and coordination formulas to define semantic dependencies among relational peer databases. The LRM focuses on inconsistent databases in a P2P network, and it allows those inconsistent peer databases in the absence of a global schema to co-

ordinate and cooperate with each other for the purpose of data sharing.

Secondly, Halevy et al present a hybrid data integration technique, called *Global-Local-As-View* (GLAV), which combines LAV and GAV. A GLAV mapping uses both LAV and GAV assertions to represent the relationships between the global schema and the source schemas. They show that GLAV is more expressive than both LAV and GAV because either the LAV or the GAV mapping language can only partially describe sources in a many-to-many fashion. They further show that query answering by using GLAV is no harder than LAV because LAV can be reduced to GLAV by an extension of inverse rules. Ives et al develop a language, called *Peer-Programming Language* (PPL), for the schema mediation problems in PDBMSs [12]. In PPL, they describe peer schemas by the LAV and GAV mapping languages. They also present a query reformulation algorithm for answering queries respectively based on both the LAV and GAV data integration approaches. The reformulation takes a query posed at a local peer and mapping formulas that represent relationships between acquainted peers as inputs, and it creates another query that is only concerned about the relations stored at the acquainted peers. In [19], the authors extend the existing data integration approach and introduce *Both-As-View* (BAV) to study the problems of data integration at the schema level in P2P systems. In BAV, they express both the global schema and the local source schemas in terms of views of each other. BAV takes both advantages of LAV and GAV. Moreover, a BAV mapping can fully be translated to either a LAV mapping or a GAV mapping; a LAV mapping or a GAV mapping can be partially translated to a BAV mapping.

Thirdly, Lenzerini introduces several principles that underly P2P data integration. He argues that each peer should be fully autonomous in a P2P network (the principle of modularity), that any two peers in a P2P network may get acquainted (the principle of generality), and that not all of the query answering mechanisms are decidable(the principle of decidability). Furthermore, Calvanese et al discuss the semantics of a P2P data integration system using first-order and epistemic logics [6, 5]. Then, they conclude that, compared to epistemic logic based approaches, the properties of modularity, generality as well as decidability are more weakly supported by first-order logic based approaches. They particularly investigate the decidability of query answering in a P2P data integration system for the cases of using first-order and epistemic logic approaches. They show that if the system has decidable schemas and conjunctive mappings, then the former is undecidable [6] and the latter [5] is decidable. Finally, in [5], the authors present an epistemic logic based framework for achieving data integration in P2P systems by building mappings at the schema level between acquainted peers. In this framework, they characterize new semantics of P2P data integration systems such that the semantic characterizations strongly support the properties of modularity, generality and decidability.

Finally, in [22], the authors propose techniques to optimize the query reformulation over XML queries and composed XML mappings in a P2P environment. They construct an AND-OR tree of nodes, where each node represents a query on a specific peer. For a query on an AND node, the answer of the query is gained by joining the answers of the AND nodes' children. For a query on an OR node, the answer of the query is the union

of the answers of the OR nodes' children. Such optimization techniques aim to prune redundant reformulation nodes and minimize the constructed reformulations. Cabibbo et al define the dimension compatibility for data mart integration upon the *drill-across* operation and study the performance of answering *drill-across* queries over autonomous data marts [7]. "*Drill-across* queries are based on joining multiple data marts over common dimensions." They conclude that if the compatibility of data marts is identified, then the performance of answering *drill-across* queries over those compatible data marts are extremely high. They also argue that if incompatible data marts are related, then the related data marts can possibly be made compatible according to relevant external information.

Chapter 3

THE LANGUAGE AND THE PROBLEMS

This chapter presents DMFML, a language for representing our mapping composition problems. Our language is adapted from [8]. Then, this chapter generalizes the problems DIMENSION MAPPING COMPOSITION and FACT MAPPING COMPOSITION. Finally, we give three important theorems and their complete proofs.

3.1 The Language DMFML

We first define concepts of dimension mappings and fact mappings. Then, we define additional concepts of extra dimension mapping tables and extra fact mapping tables. Finally, we formally introduce the idea of mapping composition of both dimension mappings and fact mappings.

In this thesis, we denote a schema-level mapping between two dimensions as a dimension mapping and a schema-level mapping between two facts as a fact mapping. A dimension mapping between two dimensions indicates that some hierarchy levels in one of two dimensions are associated with some hierarchy levels in the other dimension. A fact mapping between two facts indicates that some fact measures in one of two facts are associated with some fact measures in the other fact. On the other hand, we denote an instance-level mapping table between two hierarchy levels of two dimensions as a dimension mapping table and an instance-level mapping table between two fact measures of two facts as a fact mapping table.

Dimension Mappings and Fact Mappings: Dimension mappings and fact mappings provide semantic associations between the schemas of different data marts at different PDWs in a PDWS. Dimension mappings and fact mappings are closely related. If a dimension mapping between two dimensions does not exist, then the corresponding fact mapping between two associated facts is meaningless.

Let d_i and d_j be dimensions of data marts DM_i and DM_j, stored at PDWs p_i and p_j, respectively. Then, we use the notation $M_f(f_i, f_j)$ to denote a fact mapping of the fact f_i of DM_i to the fact f_j of DM_j. Such a fact mapping is definable if f_i is associated with every dimension d_i of data mart DM_i, if f_j is associated with every dimension d_j of data mart DM_j, and if for every dimension d_i (or d_j) of data mart DM_i (or DM_j) there exists a dimension d_j (or d_i) of data mart DM_j (or DM_i) such that a dimension mapping $M_d(d_i, d_j)$ has been built. If a mapping(either a dimension mapping or a fact

mapping) is built between two acquainted PDWs, it must be stored at both sides of the PDWs.

Definition 7 (Dimension Mapping) *Let d_1 and d_2 be two dimensions of two data marts DM_1 and DM_2 at two PDWs p_1 and p_2, respectively. Furthermore, let d_1 have hierarchy levels $L(d_1) = \{l_1^1, l_1^2, ..., l_1^m\}$, and let d_2 have hierarchy levels $L(d_2) = \{l_2^1, l_2^2, ..., l_2^n\}$. Then, a dimension mapping $M_d(d_1, d_2)$ between d_1 and d_2 is the set $\{(l_1^i, l_2^j) \mid l_1^i \in L(d_1), l_2^j \in L(d_2)\}$, where every $l_1^i \in L(d_1)$ appears in $M_d(d_1, d_2)$ at most once and every $l_2^j \in L(d_2)$ appears in $M_d(d_1, d_2)$ at most once as well.*

Example 1: Figure 1.3 shows dimension mappings between the data marts *Enrollment* and *Registration* at PDWs of Universities A and B:

$$M_d(Students, Student) = \{(StudentID, SID), (Program, Prog),$$

$$(Department, Dept), (Faculty, Faculty)\};$$

$$M_d(Professors, Professor) = \{(ProfID, PID), (Title, Title),$$

$$(Department, Dept), (Faculty, Faculty)\};$$

$$M_d(Courses, Course) = \{(CourseID, CID), (Type, Type),$$

$$(Program, Prog), (Department, Dept), (Faculty, Faculty)\}.$$

Definition 8 (Complete Dimension Mapping) *A complete dimension mapping between two dimensions indicates that every hierarchy level in one of two dimensions is*

associated with a hierarchy level in the other dimension, and vice versa. A complete dimension mapping is a dimension mapping $M'_d(d_1, d_2) = \{(l_1^i, l_2^j) \mid l_1^i \in L(d_1), l_2^j \in L(d_2)\}$, *where every* $l_1^i \in L(d_1)$ *appears in* $M'_d(d_1, d_2)$ *exactly once and every* $l_2^j \in L(d_2)$ *appears in* $M'_d(d_1, d_2)$ *exactly once as well. Otherwise, it is an incomplete dimension mapping.*

Example 2: In Example 1, $M_d(Professors, Professor)$ is a complete dimension mapping. Whereas, $M_d(Students, Student)$ is an incomplete dimension mapping since the *Option* attribute of the *Student* dimension does not appear in $M_d(Students, Student)$; $M_d(Courses, Course)$ is also an incomplete dimension mapping since the *Option* attribute of the *Course* dimension does not appear in $M_d(Courses, Course)$ either.

Definition 9 (Fact Mapping) *Let* f_1 *and* f_2 *be two facts of two data marts* DM_1 *and* DM_2 *at two PDWs* p_1 *and* p_2, *respectively. Furthermore, let* f_1 *have fact measures* $f_1 = \{m_1^1, m_1^2, ..., m_1^q\}$, *and let* f_2 *have fact measures* $f_2 = \{m_2^1, m_2^2, ..., m_2^n\}$. *Then, a fact mapping* $M_f(f_1, f_2)$ *is the set* $\{(m_1^i, m_2^j) \mid m_1^i \in f_1, m_2^j \in f_2\}$, *where every* $m_1^i \in f_1$ *appears in* $M_f(f_1, f_2)$ *at most once and every* $m_2^j \in f_2$ *appears in* $M_f(f_1, f_2)$ *at most once as well.*

Example 3: Figure 1.4 shows a fact mapping between the data marts *Enrollment* and *Registration* at PDWs of Universities A and B:

$$M_f(Grades, Marks) = \{(Grade, Mark)\}.$$

Definition 10 (Complete Fact Mapping) *A complete fact mapping between two facts indicates that every fact measure in one of two facts is associated with a fact measure in the other fact, and vice versa. A complete fact mapping is a fact mapping* $M'_f(f_1, f_2) = \{(m_1^i, m_2^j) \mid m_1^i \in f_1, m_2^j \in f_2\}$, *where every* $m_1^i \in f_1$ *appears in* $M'_f(f_1, f_2)$

exactly once and every $m_2^j \in f_2$ appears in $M'_f(f_1, f_2)$ exactly once as well. Otherwise, it is an incomplete fact mapping.

Example 4: In Example 3, $M_f(Grades, Marks)$ is an incomplete fact mapping since the *Rank* fact measure of the *Grades* fact does not appear in $M_f(Grades, Marks)$. Suppose that the *Grades* fact of the data mart *Registration* at University B has another fact measure called *MarkRank*. Then the fact mapping:

$$M'_f(Grades, Marks) = \{(Grade, Mark), (Rank, MarkRank)\},$$

is a complete fact mapping.

Extra Dimension Mapping Table and Extra Fact Mapping Table: Given two dimensions and a dimension mapping of those two dimensions, if the dimension mapping is incomplete, then a collection of extra dimension mapping tables should be built to make it complete. An extra dimension mapping table is generated by a specific conjunctive query. The query is generated by carefully studying semantics of the incomplete dimension mapping. It is a design decision to determine what should be written in the query. Semantic studies determine if a hierarchy level of a dimension has sufficient information to generate such a query. An extra dimension mapping table is a supplement to a dimension mapping, which reveals the missing dimension information of a hierarchy level. The number of extra dimension mapping tables of an incomplete dimension mapping is the number of the attributes of the two dimensions that do not appear in the dimension mapping.

Definition 11 (Dimension Mapping Table) *Let d_1 and d_2 be two dimensions of two data marts DM_1 and DM_2 at two PDWs p_1 and p_2, respectively. Furthermore, let d_1*

have hierarchy levels $L(d_1) = \{l_1^1, l_1^2, ..., l_1^m\}$, *and let* d_2 *have hierarchy levels* $L(d_2) = \{l_2^1, l_2^2, ..., l_2^n\}$. *Then, a dimension mapping table* $MT_d(l_1^i, l_2^j)$ *between* d_1 *and* d_2 *is the set of the values of* $\{(l_1^i, l_2^j) \mid l_1^i \in L(d_1), l_2^j \in L(d_2)\}$, *where every value of* l_1^i *appears in* $MT_d(l_1^i, l_2^j)$ *at most once and every value of* l_2^j *appears in* $MT_d(l_1^i, l_2^j)$ *at most once as well.*

Definition 12 (Conjunctive Query) *The basic form of relational expressions is*

$$\{t \mid \psi(t)\},$$

where t is a tuple variable or constant (here, t denotes a tuple of some fixed arity), and ψ *is a formula built according to the conventional first order predicate calculus rules* [1]. *Let R be a relational database schema. A conjunctive query q over R is an expression of the form*

$$q : ans(t) \leftarrow R_1(t_1) \wedge ... \wedge R_n(t_n);$$

where $R_1, ...R_n$ *are relation names in R; ans is a relation name not in R;* $t, t_1, ..., t_n$ *are free tuples defined previously. Variables appear in t must also appear at least once in* $t_1, ..., t_n$ [1].

Definition 13 (Extra Dimension Mapping Table) *Let* $M_d(d_1, d_2) = \{(l_1^1, l_2^1), (l_1^2, l_2^2)\}$ *be a dimension mapping, where* d_1 *is a dimension of data mart* DM_1 *at PDW* p_1, d_2 *is a dimension of data mart* DM_2 *at PDW* p_2; *also let* d_1 *have hierarchy levels* l_1^1, l_1^2, *and let* d_2 *have hierarchy levels* l_2^1, l_2^2, l_2^3. *Furthermore, let* $K(d_1)$ *be a set of surrogate keys of* d_1, *and* $q(l_2^3)$ *be a conjunctive query over hierarchy level* l_2^3 *of dimension* d_2. *Then, an extra dimension mapping table* $T_d(K(d_1), q(l_2^3))$ *generated by a conjunctive query* $q(K(d_1), q(l_2^3))$

over d_1 reveals the missing information of l_2^3 for d_1. As a result, a complete dimension mapping is $M'_d(d_1, d_2) = \{(l_1^1, l_2^1)(l_1^2, l_2^2)(T_d(K(d_1), q(l_2^3)), l_2^3)\}$.

The conjunctive query $q(l_2^3)$ over l_2^3 of d_2 is to get a set of values of l_2^3 of d_2 so that we can decide what should be written in $q(K(d_1), q(l_2^3))$ to generate $T_d(K(d_1), q(l_2^3))$. Both $q(l_2^3)$ and $q(K(d_1), q(l_2^3))$ are conjunctive queries because each query deals with a single attribute of a dimension.

Example 5: In Example 2, we have given:

$$M_d(Students, Student) = \{(StudentID, SID), (Program, Prog),$$

$$(Department, Dept), (Faculty, Faculty)\},$$

which is an incomplete dimension mapping. After we build an extra dimension mapping table $T_d(K(Students), q(Option))$, the complete dimension mapping is:

$$M'_d(Students, Student) = \{(StudentID, SID), (T_d(K(Students), q(Option)), Option),$$

$$(Program, Prog), (Department, Dept), (Faculty, Faculty)\}.$$

Figure 3.1 shows the complete dimension mapping $M'_d(Students, Student)$. We will describe the data structure of a complete dimension mapping shortly.

Similarly, given two facts and a fact mapping of those two facts, if the fact mapping is incomplete, then a collection of extra fact mapping tables should be built to make it complete. An extra fact mapping table is generated by a specific non-recursive query. The query is generated by carefully considering the semantics of the incomplete fact

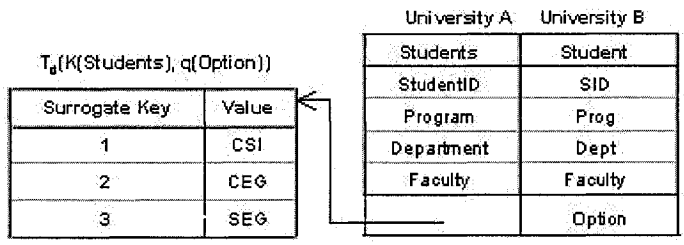


Figure 3.1: Complete Dimension Mapping

mapping. Again, it is a design decision to determine what should be written in the query. This semantics determines the relationship between two fact measures of a fact. Such a relationship can be converted into a non-recursive query. An extra fact mapping table is a supplement to the fact mapping, which reveals the missing fact information of a fact measure. The number of extra fact mapping tables of an incomplete fact mapping is the number of the fact measures of the two facts that do not appear in the fact mapping.

Definition 14 (Fact Mapping Table) *Let f_1 and f_2 be two facts of two data marts DM_1 and DM_2 at two PDWs p_1 and p_2, respectively. Furthermore, let f_1 have fact measures $f_1 = \{m_1^1, m_1^2, ..., m_1^q\}$, and let f_2 have fact measures $f_2 = \{m_2^1, m_2^2, ..., m_2^n\}$. Then, a fact mapping table $MT_f(m_1^i, m_2^j)$ between f_1 and f_2 is the set of the values of $\{(m_1^i, m_2^j) \mid m_1^i \in f_1, m_2^j \in f_2\}$, where every value of m_1^i appears in $MT_f(m_1^i, m_2^j)$ at most once and every value of m_2^j appears in $MT_f(m_1^i, m_2^j)$ at most once as well.*

Definition 15 (Non-recursive Query) *A non-recursive query q is an expression of the form*

$$q : S(u) \leftarrow L_1, ..., L_n,$$

where S is a relation name; u is a free tuple defined in Definition 12; $L_1, ...L_n$ are literals.

A literal is an expression of the form $R(t)$, where R is a relation name and t is a free tuple defined in Definition 12. Moreover, S does not occur in the body of R [1].

Definition 16 (Extra Fact Mapping Table) *Let $M_f(f_1, f_2) = \{(m_1^1, m_2^1)\}$ be a fact mapping, where f_1 is a fact of data mart DM_1 at PDW p_1, f_2 is a fact of data mart DM_2 at PDW p_2; also let f_1 have a fact measure m_1^1, and let f_2 have fact measures m_2^1 and m_2^2. Furthermore, let $r(m_2^1, m_2^2)$ be a relationship between m_2^1 and m_2^2. (the relationship $r(m_2^1, m_2^2)$ can be empty, meaning that there is no relationship between m_2^1 and m_2^2). Let $FK(f_1)$ be a set of foreign keys of f_1, and $q(r(m_2^1, m_2^2), m_1^1)$ be a non-recursive query over f_1. Then, an extra fact mapping table $T_f(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ generated by a query $q(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ over f_1 determines the missing information of m_2^2 for f_1. As a result, the complete fact mapping is:*

$$M'_f(f_1, f_2) = \{(m_1^1, m_2^1), (T_f(FK(f_1), q(r(m_2^1, m_2^2), m_1^1)), m_2^2)\}.$$

The non-recursive query $q(r(m_2^1, m_2^2), m_1^1)$ over m_1^1 of f_1 is to get a set of values of "m_1^2" of f_1, which is discovered by the relationship of m_1^1 and "m_1^2", by examining $r(m_2^1, m_2^2)$. When $q(r(m_2^1, m_2^2), m_1^1)$ is in place, we can write a straightforward query $q(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ to generate $T_f(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$. The following query:

$$q(r(m_2^1, m_2^2), m_1^1)$$

is a non-recursive query because we do have no idea to predict what the relationship of m_2^1 and m_2^2 is. For instance, the relationship of the fact measures *Grade* and *Rank* is to sort *Grade* in ascending order, and the relationship of the fact measures *Grade* and *Midterm* is the percentage. Every such relationship, however, can be manually

expressed in SQL by a non-recursive query [1]. On the other hand, even though the only purpose of $q(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ is to add the set of foreign keys of f_1 to $T_f(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$, we observe that $q(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ is a nested query in which the sub-query $q(r(m_2^1, m_2^2), m_1^1)$ is a non-recursive query, and therefore $q(FK(f_1), q(r(m_2^1, m_2^2), m_1^1))$ is a non-recursive query.

Example 6: In Example 3, we have given:

$$M_f(Grades, Marks) = \{(Grade, Mark)\},$$

which is an incomplete fact mapping. After we build an extra fact mapping table

$$T_f(FK(Marks), q(r(Grade, Rank), Mark)),$$

the complete fact mapping is:

$$M'_f(Grades, Marks) = \{(Grade, Mark),$$

$$(Rank, T_f(FK(Marks), q(r(Grade, Rank), Mark)))\}.$$

Figure 3.2 shows the complete fact mapping $M'_f(Grades, Marks)$. We will describe the data structure of a complete fact mapping shortly.

It is important to know that during the process of building an extra dimension mapping table, the generated query over a hierarchy level might return an empty set. This means that the extra dimension mapping table cannot be built because there is no sufficient information stored in the data warehouses for us to discover the missing hierarchy level.

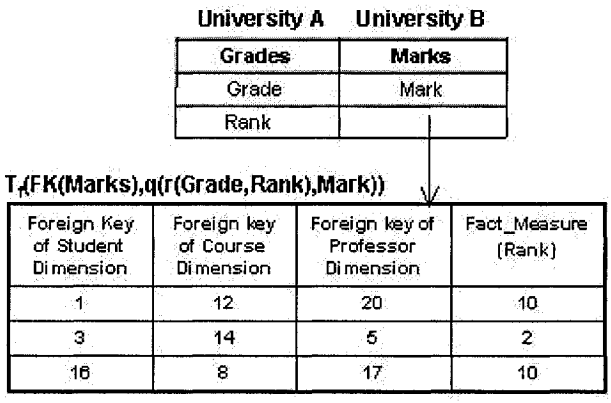


Figure 3.2: Complete Fact Mapping

Example 7: Suppose that the data mart *Enrollment* at University A has a dimension *Students(StudentID, Program, Department, Faculty)* and that the data mart *Registration* at University B has a dimension *Student(SID, Email, Option, Prog, Dept, Faculty).* Then we have the following incomplete dimension mapping:

$$M_d(Students, Student) = \{(StudentID, SID), (Program, Prog),$$

$$(Department, Dept), (Faculty, Faculty)\},$$

and we further have the following dimension mapping:

$$M'_d(Students, Student) = \{(StudentID, SID), (T_d(K(Students), q(Option)), Option),$$

$$(Program, Prog), (Department, Dept), (Faculty, Faculty)\}.$$

However, $M'_d(Students, Student)$ is still incomplete because the information of student email addresses is not stored in the PDW at University A. Figure 3.3 shows a scenario of an incomplete dimension mapping.

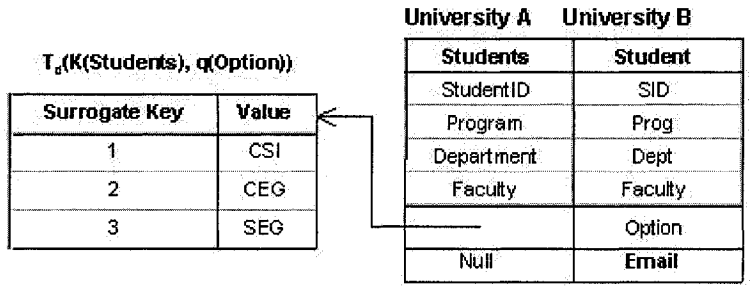


Figure 3.3: Incomplete Dimension Mapping

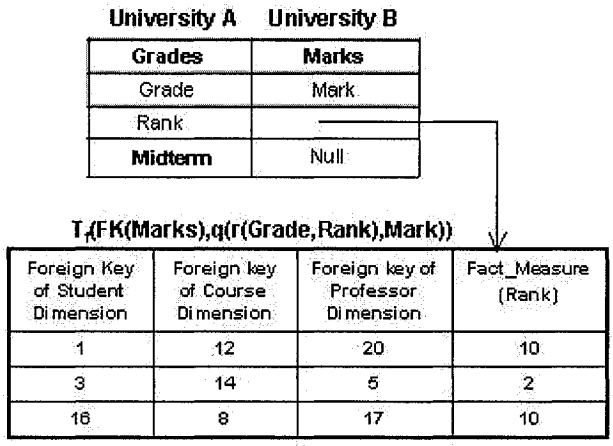


Figure 3.4: Incomplete Fact Mapping

Similarly, during the process of building an extra fact mapping table, the generated query over a relationship of two fact measures might return an empty set. This means that the extra fact mapping table cannot be built because there is no sufficient information stored in the data warehouses for us to discover the missing fact measure.

Example 8: Suppose that the data mart *Enrollment* at University A has a fact *Grades(Midterm, Grade, Rank)* and that the data mart *Registration* at University B has

a fact *Marks(Mark)*. Then, we have the following incomplete fact mapping:

$$M_f(Grades, Marks) = \{(Grade, Mark)\},$$

and we further have the following fact mapping:

$$M'_f(Grades, Marks) = \{(Grade, Mark), (Rank, T_f(FK(Marks),$$

$$q(r(Grade, Rank), Mark)))\}.$$

However, $M'_f(Grades, Marks)$ is still incomplete because the information of student midterm grades is not stored in the PDW at University B. Figure 3.4 shows a scenario of an incomplete fact mapping.

Then, we might need to look at the data sources to discover such missing information. This topic is out of the scope of the thesis.

Dimension Mapping Composition and Fact Mapping Composition: Suppose that we have PDWs p_A and p_B in a PDWS and that there is an acquaintance between p_A and p_B. Furthermore, suppose that we have dimension mappings and fact mappings of data marts DM_A at p_A and DM_B at p_B. Moreover, suppose that a new PDW p_C is added to the system and that an acquaintance between p_B and p_C is built (there is no acquaintance between q_A and q_C). Finally, suppose that we have a collection of dimension mappings and a collection of fact mappings between DM_A and DM_B that are in place and that we have a collection of dimension mappings and a collection of fact mappings between DM_B and DM_C that are in place. Although there is no acquaintance between p_A and p_C, we may have sufficient information to build dimension mappings and fact

mappings between p_A and p_C. That is, we can establish dimension mappings and fact mappings between DM_A and DM_C based on the existing dimension mappings and fact mappings between DM_A and DM_B as well as those between DM_B and DM_C.

Definition 17 (Dimension Mapping Composition) *Let $M_d(d_1, d_2)$ and $M_d(d_2, d_3)$ be dimension mappings, where d_1 is a dimension of data mart DM_1 at PDW p_1, d_2 is a dimension of data mart DM_2 at PDW p_2, and d_3 is a dimension of data mart DM_3 at p_3, respectively. Furthermore, let $Q[d_1, d_2]$ be a set of conjunctive queries to generate extra dimension mapping tables for $M_d(d_1, d_2)$, and $Q[d_2, d_3]$ be a set of conjunctive queries to generate extra dimension mapping tables for $M_d(d_2, d_3)$. Finally, let $M'_d(d_1, d_2)$ be a complete dimension mapping that consists of $M_d(d_1, d_2)$ and $Q[d_1, d_2]$, and $M'_d(d_2, d_3)$ be a complete dimension mapping that consists of $M_d(d_2, d_3)$ and $Q[d_2, d_3]$. Dimension mapping composition consists in directly building a dimension mapping $M'_d(d_1, d_3)$ by using $M'_d(d_1, d_2)$ and $M'_d(d_2, d_3)$. The dimension mapping $M'_d(d_1, d_3)$ is made of a dimension mapping $M_d(d_1, d_3)$ between p_1 and p_3 as well as a set of conjunctive queries $Q[d_1, d_3]$ between DM_1 and DM_3. Moreover, the set of conjunctive queries $Q[d_1, d_3]$ is rewritten by $Q[d_1, d_2]$ or $Q[d_2, d_3]$.*

Example 9: Figure 3.5 shows a scenario of composing complete dimension mappings. In Example 5, we have:

$$M_d(Students, Student) = \{(StudentID, SID), (Program, Prog),$$

$$(Department, Dept), (Faculty, Faculty)\},$$

and a set of conjunctive queries $Q[Students, Student] = \{q(K(Students), q(Option))\}$ (in this example, the set of conjunctive queries has only one query) to generate the

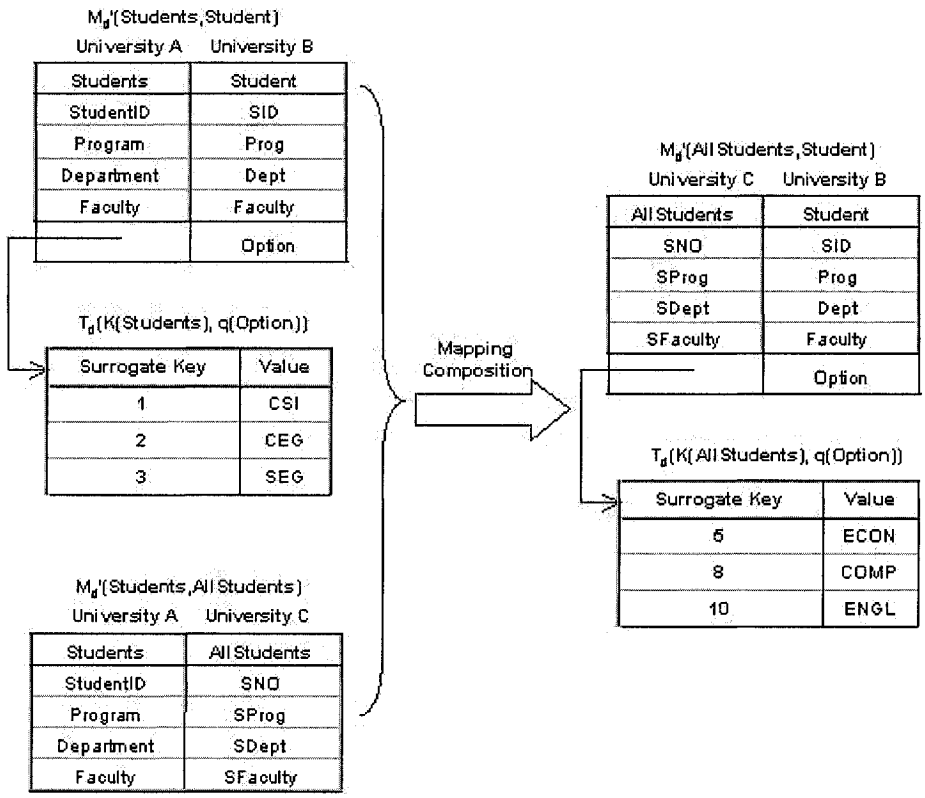


Figure 3.5: Dimension Mapping Composition

extra dimension mapping table $T_d(K(Students), q(Option))$. Suppose that we have a data mart *StudentRegister* at the PDW of University C and that the data mart *StudentRegister* has dimensions *AllStudents*, *AllCourses* and *AllProfessors*. The dimension *AllStudents* has hierarchy levels *SNO*, *SProg*, *SDept* and *SFaculty*. Furthermore, suppose that an acquaintance is established between Universities A and C, and that the following dimension mapping:

$$M_d(Students, AllStudents) = \{(StudentID, SNO), (Program, SProg),$$

$$(Department, SDept), (Faculty, SFaculty)\},$$

is in place ($Q[Students, AllStudents] = \phi$). Then, we can use $M_d(Students, Student)$ and $M_d(Students, AllStudents)$ together with $Q[Students, Student]$ to generate the following dimension mapping:

$$M'_d(AllStudents, Student) = \{(SNO, SID), (T_d(K(AllStudents), q(Option)), Option),$$

$$(SProg, Prog), (SDept, Dept), (SFaculty, Faculty)\}.$$

$T_d(K(AllStudents), q(Option))$ is an extra dimension mapping table and is generated by $q(K(AllStudents), q(Option))$. $q(K(AllStudents), q(Option))$ is rewritten by:

$$q(K(Students), q(Option)).$$

Definition 18 (Fact Mapping Composition) *Let $M_f(f_1, f_2)$ and $M_f(f_2, f_3)$ be fact mappings, where f_1 is a fact of data mart DM_1 at PDW p_1,f_2 is a fact of data mart DM_2 at PDW p_2, and f_3 is a fact of data mart DM_3 at p_3. Furthermore, let $Q[f_1, f_2]$ be a set of non-recursive queries to generate extra fact mapping tables for $M_f(f_1, f_2)$, and $Q[f_2, f_3]$ be a set of non-recursive queries to generate extra fact mapping tables for $M_f(f_2, f_3)$. Finally, let $M'_f(f_1, f_2)$ be a complete fact mapping that consists of $M_f(f_1, f_2)$ and $Q[f_1, f_2]$, and $M_f(f_2, f_3)$ be a complete fact mapping that consists of $M_f(f_2, f_3)$ and $Q[f_2, f_3]$. Fact mapping composition consists in building a fact mapping $M'_d(f_1, f_3)$ by using $M'_f(f_1, f_2)$ and $M'_f(f_2, f_3)$. The fact mapping $M'_d(f_1, f_3)$ is made of a fact mapping $M_d(f_1, f_3)$ between p_1 and p_3 as well as a set of non-recursive queries $Q[f_1, f_3]$ between DM_1 and DM_3. Moreover, the set of non-recursive queries $Q[f_1, f_3]$ is rewritten by $Q[f_1, f_2]$ or $Q[f_2, f_3]$.*

Example 10: Figure 3.6 shows a scenario of composing complete fact mappings. In Example 6, we have:

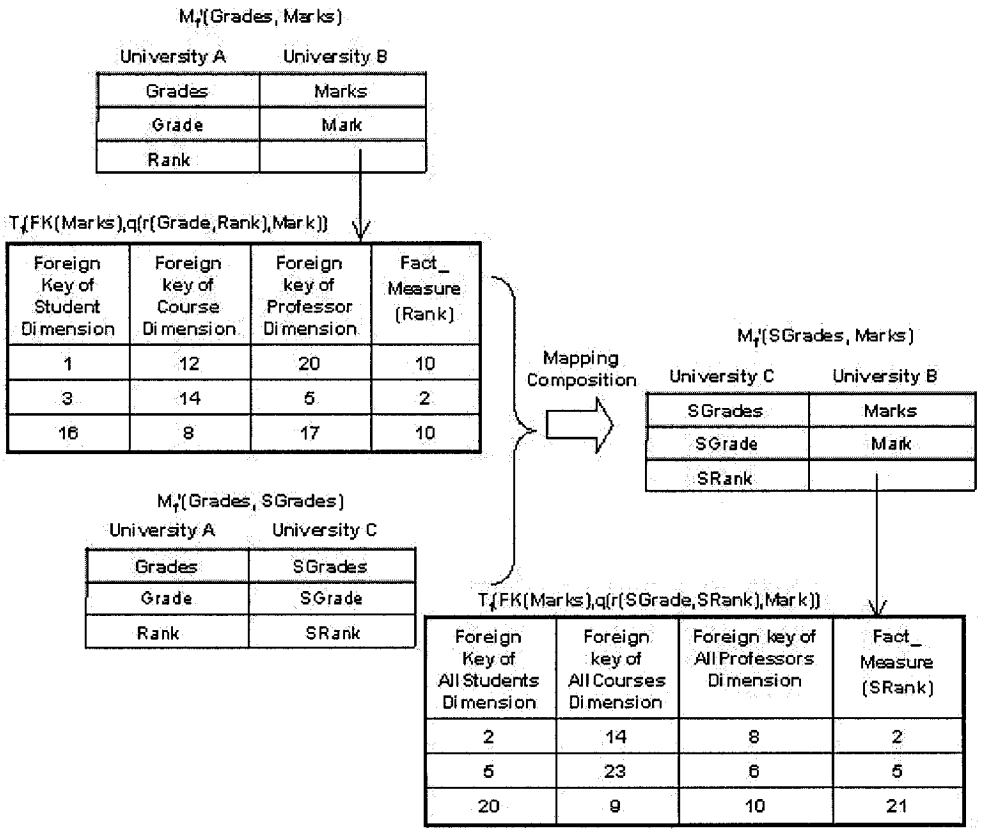


Figure 3.6: Fact Mapping Composition

$$M_f(Grades, Marks) = \{(Grade, Mark)\},$$

and a set of non-recursive queries:

$$Q[Grades, Marks] = \{q(FK(Marks), q(r(Grade, Rank), Mark))\}$$

(in this example, the set of non-recursive queries has only one query) to generate the following extra fact mapping table:

$$T_f(FK(Marks), q(r(Grade, Rank), Mark)).$$

Suppose that the data mart *StudentRegister* at the PDW of University C has a fact

SGrades. The fact *SGrades* has fact measures $SGrade$ and $SRank$. Furthermore, suppose that we have the following fact mapping:

$$M_f(Grades, SGrades) = \{(Grade, SGrade), (Rank, SRank)\},$$

is in place ($Q[Grades, SGrades] = \phi$). Then, we can use $M_f(Grades, Marks)$ and $M_f(Grades, SGrades)$ together with $Q[Grades, Marks]$ to generate the following fact mapping:

$$M'_f(SGrades, Marks) = \{(SGrade, Mark),$$

$$(SRank, T_f(FK(Marks), q(r(SGrade, SRank), Mark)))\}.$$

$T_f(FK(Marks), q(r(SGrade, SRank), Mark))$ is an extra mapping table and is generated by $q(FK(Marks), q(r(SGrade, SRank), Mark))$. The query:

$$q(FK(Marks), q(r(SGrade, SRank), Mark))$$

is rewritten by:

$$q(FK(Marks), q(r(Grade, Rank), Mark)).$$

3.2 The Problems and the Theorems

We generalize aforementioned closely-related problems as follows:

DIMENSION MAPPING COMPOSITION Problem: Suppose that we have a set of PDWs P and that each PDW $p_i \in P$ has a set of dimensions D_i. Furthermore, suppose that we have two dimension mappings $M_d(d_i, d_j)$ and $M_d(d_j, d_k)$, where $d_i \in D_i$, $d_j \in D_j$ and $d_k \in D_k$. Find a dimension mapping $M_d(d_i, d_k)$ such that if both $M_d(d_i, d_j)$

and $M_d(d_j, d_k)$ are complete, then $M_d(d_i, d_k)$ is also complete.

FACT MAPPING COMPOSITION Problem: Suppose that we have a set of PDWs P and that each PDW $p_i \in P$ has a set of facts F_i. Furthermore, suppose that we have two fact mappings $M_f(f_i, f_j)$ and $M_d(f_j, f_k)$, where $f_i \in F_i$, $f_j \in F_j$ and $f_k \in F_k$. Find a fact mapping $M_f(f_i, f_k)$ such that if both $M_f(f_i, f_j)$ and $M_f(f_j, f_k)$ are complete, then $M_f(f_i, f_k)$ is also complete.

Theorem 1 *Suppose that we have dimensions d_1 and d_2, as well as a dimension mapping $M_d(d_1, d_2)$. Determining whether $M_d(d_1, d_2)$ is a complete dimension mapping of d_1 and d_2 is in log space. On the other hand, suppose that we have facts f_1 and f_2 as well as a fact mapping $M_f(f_1, f_2)$. Determining whether $M_f(f_1, f_2)$ is a complete fact mapping of f_1 and f_2 is also in log space.*

PROOF: The first part: Let the total size of the inputs of d_1, d_2 and $M_d(d_1, d_2)$ be n. Construct a deterministic Turing machine TM_d. TM_d reads elements of d_1 and d_2 one by one. If all the elements of d_1 and d_2 can be found in $M_d(d_1, d_2)$, then TM_d returns YES. Otherwise, TM_d returns NO. It can be done in log space with respect to the size of n because all the elements of d_1 and d_2 are read and checked one by one in memory. The second part: a similar proof can be given.

Theorem 2 *Suppose that we have dimensions d_1, d_2 and d_3 as well as that complete dimension mappings $M'_d(d_1, d_2)$ and $M'_d(d_2, d_3)$. Furthermore, suppose that $M'_d(d_1, d_2)$ consists of a dimension mapping $M_d(d_1, d_2)$ as well as a set of conjunctive queries $Q[d_1, d_2]$ and that $M'_d(d_2, d_3)$ consists of a dimension mapping $M_d(d_2, d_3)$ as well as a set of conjunctive queries $Q[d_2, d_3]$. Finally, suppose that we have a dimension mapping $M'_d(d_1, d_3)$*

that consists of a dimension mapping $M_d(d_1, d_3)$ and a set of conjunctive queries $Q[d_1, d_3]$. Determining whether $M'_d(d_1, d_3)$ is a composition of $M'_d(d_1, d_2)$ and $M'_d(d_2, d_3)$, as well as $M'_d(d_1, d_3)$ whether is complete is in log space. On the other hand, suppose that we have facts f_1, f_2 as well as f_3 and that fact mappings $M'_f(f_1, f_2)$ and $M'_f(f_2, f_3)$. Furthermore, suppose that $M'_f(f_1, f_2)$ consists of a fact mapping $M_f(f_1, f_2)$ as well as a set of non-recursive queries $Q[f_1, f_2]$ and that $M'_f(f_2, f_3)$ consists of a fact mapping $M_f(f_2, f_3)$ as well as a set of non-recursive queries $Q[f_2, f_3]$. Finally, suppose that we have a fact mapping $M'_f(f_1, f_3)$ that consists of a fact mapping $M_f(f_1, f_3)$ and a set of non-recursive queries $Q[f_1, f_3]$. Determining whether $M'_f(f_1, f_3)$ is a composition of $M'_f[f_1, f_2]$ and $M'_f[f_2, f_3]$, as well as is complete is in PTIME.

PROOF: The first part: Let the total size of inputs of d_1,d_2,d_3,$M'_d(d_1, d_2)$,$M'_d(d_2, d_3)$ and $M'_d(d_1, d_3)$ be n. Construct a deterministic Turing machine TM_d. TM_d reads the pairs in $M'_d(d_1, d_3)$ one by one into the memory. Then, TM_d checks if for any pair (l_1^k, l_3^k) in $M'_d(d_1, d_3)$, we can find two pairs (l_1^k, l_2^i) in $M'_d(d_1, d_2)$ and (l_2^j, l_3^k) in $M'_d(d_2, d_3)$ such that l_2^i and l_2^j are equal if both are hierarchy levels of d_2 or such that l_2^i and l_2^j yield the same result if both are queries in $Q[d_1, d_2]$ or $Q[d_2, d_3]$. If for all the pairs in $M'_d(d_1, d_3)$ it is true, and furthermore if $M'_d(d_1, d_3)$ is complete, then TM_d returns YES. Otherwise, TM_d returns NO. Because all the pairs are read and checked one by one, because a conjunctive query can be answered in log space [1], as well as because checking the completeness of $M'_d(d_1, d_3)$ can be done in log space(by Theorem 1), the whole process can be done in log space with respect to the size of n.

The second part: a similar proof follows except that a non-recursive query can be answered in PTIME [1]. Thus, the whole process can be done in PTIME.

Proposition 1 *A composition of two complete dimension mappings is a complete dimension mapping; a composition of two complete fact mappings is a complete fact mapping.*

PROOF: The first part: by contradiction. Let d_1, d_2 and d_3 be dimensions. Furthermore, let $M'_d(d_1, d_2)$ that consists of a dimension mapping $M_d(d_1, d_2)$ as well as a set of conjunctive queries $Q[d_1, d_2]$ and $M'_d(d_2, d_3)$ that consists of a dimension mapping $M_d(d_2, d_3)$ as well as a set of conjunctive queries $Q[d_2, d_3]$ be complete dimension mappings. Finally, let $M'_d(d_1, d_3)$ that consists of a dimension mapping $M_d(d_1, d_3)$ and a set of conjunctive queries $Q[d_1, d_3]$) be a composition of $M'_d(d_1, d_2)$ and $M'_d(d_2, d_3)$. Assume that $M'_d(d_1, d_3)$ is incomplete. Then, there exists an element l in d_1, d_3 or $Q[d_1, d_3]$ that does not appear in $M'_d(d_1, d_3)$. If l is in d_1, then it reaches a contradiction because $M'_d(d_1, d_2)$ is complete. If l is in d_3, then it reaches a contradiction because $M'_d(d_2, d_3)$ is complete. If l is in $Q[d_1, d_3]$, then it reaches a contradiction because $Q[d_1, d_3]$ is rewritten by either $Q[d_1, d_2]$ or $Q[d_2, d_3]$. Therefore, $M'_d(d_1, d_3)$ is complete.

The second part: a similar proof can be given.

Chapter 4

DIMENSION AND FACT MAPPING COMPOSITION

This chapter first proposes an idea of solving our mapping composition problems. Then it presents a unified distributed algorithm for solving both the DIMENSION MAPPING COMPOSITION and the FACT MAPPING COMPOSITION problems. In the distributed algorithm, we mainly have four local functions. Two of them determine the completeness of a dimension mapping and a fact mapping, respectively. The rest two functions compose complete dimension mappings and complete fact mappings, respectively.

4.1 The Algorithm

This section presents a unified distributed algorithm for solving both the DIMENSION MAPPING COMPOSITION and the FACT MAPPING COMPOSITION problems that

are closely related. We first briefly describe the whole process of the algorithm, and then present the detailed steps of the algorithm (except the local functions). We present the local functions in the following sections.

4.1.1 The Idea

This unified algorithm is written in a distributed way. It consists of five steps in total. The algorithm is executed as a sequence of those five steps. We build the corresponding graph G for a PDWS. Each node in G corresponds to a PDW in the system, and each edge in G corresponds to an acquaintance between two PDWs. The algorithm starts when a new PDW is added to the system. The variables declared in a previous step can be used in any following step. At the beginning of each step, the execution states, the initial state and the termination state of each node as well as new variables used in the step are listed. Except for the last step, the termination state in a step is usually the initial state in the following step.

Given: A strongly connected graph $G(P, E)$ (P is a set of peer data warehouses, and E is a set of acquaintances between PDWs in P), each PDW p_i with a set of dimensions D_i, a set of facts F_i, a collection of complete dimension mappings $\mathcal{M}'^{i}_{\mathcal{D}}$ as well as a collection of complete fact mappings $\mathcal{M}'^{i}_{\mathcal{F}}$, and a new PDW p_u with a set of dimensions D_u, a set of facts F_u, a collection of dimension mappings $\mathcal{M}^{u}_{\mathcal{D}}$ as well as a collection of fact mappings $\mathcal{M}^{u}_{\mathcal{F}}$.

First Step: When a new node p_u is added to G, we treat p_u as the INITIATOR of the

computation. The INITIATOR sends a message to its neighbors to wake them up. The process continues until all the nodes in G are awake. This is to construct a spanning tree T for G, and the INITIATOR is recognized as the root. The algorithm for constructing a spanning tree is adapted from [20].

Second Step: The root gets the total number of nodes in G.

Third Step: The root gets dimensions, facts, complete dimension mappings and complete fact mappings of all the other nodes.

Fourth Step: The root invokes two local functions to determine the completeness of its dimension mappings and fact mappings respectively, and to make those incomplete dimensions mappings and fact mappings complete respectively. Then, the root invokes another two local functions to compose dimension mappings and fact mappings respectively.

Fifth Step: The root sends the composed dimension mappings and fact mappings to its Tree-Neighbors. If a Tree-Neighbor is not the destination, it forwards them to its Tree-Neighbors except the sender. The process continues until all the messages arrive at their expected destinations.

4.1.2 The Distributed Algorithm

Now, we give the details of each step of the distributed algorithm as follows.

First Step:

States: {INITIATOR, IDLE, ACTIVE, DONE};

Initial States: {INITIATOR, IDLE};

Termination State: {DONE};

Message Q: A message that the current node p_v sends to its neighbors to ask if they would like to be in T;

Message YES: A message that a neighbor sends to its sender to say that it would like to be in T;

Message NO: A message that a neighbor sends to its sender to say that it has answered the message Q and has been in T;

$N(p_v)$: All the neighbors of p_v in G;

Tree-Neighbors: The neighbors of p_v in T;

sender: The node that sends a message to its neighbors;

parent: The parent of p_v in T;

counter: A counter indicating the number of neighbors of p_v has been reached;

root: A boolean variable indicating if a node is the root of T or not;

Initially, for $\forall p_v \in G$, Tree-Neighbors$(p_v) = \Phi$.

INITIATOR:

Spontaneously

root : = true;

Tree-Neighbors := Φ;

Send(Q) to $N(p_v)$;

counter := 0;

become ACTIVE;

IDLE:

Receiving(Q)

root := false;

parent := sender;

Tree-Neighbors := {sender};

Send(YES) to sender;

counter := 1;

IF counter = $|N(p_v)|$ THEN

Become DONE;

ELSE

Send(Q) to $N(p_v)$ - {sender};

Become ACTIVE;

END IF

ACTIVE:

Receiving(Q)

Send(NO) to sender;

Receiving(YES)

Tree-Neighbors := Tree-Neighbors$\cup${sender};

counter := counter + 1;

IF counter = $|N(p_v)|$ THEN

Become DONE;

END IF

Receiving(NO)

counter := counter + 1;

IF counter = $|N(p_v)|$ THEN

Become DONE;

END IF

Second Step:

States: {DONE, PROCESSING, FINISHED};

Initial State: {DONE};

Termination State: {FINISHED};

Message I: A message that the current node p_v in T sends to its parent indicating the number of its descendants plus itself;

neighbors: A temporary set variable storing the neighbors of v;

c: A counter indicating the number of descendants plus itself, initially only itself.

DONE:

c := 1;

neighbors := Tree-Neighbors;

IF |neighbors| = 1 THEN

I :=(c);

Send(I) to parent;

Become FINISHED;

ELSE

Become PROCESSING;

END IF

PROCESSING:

Receiving(I)

c := c + I.c;

neighbors := neighbors - {sender};

IF |neighbors| = 0 THEN ;the node is the root

Become FINISHED;

RETURN c;

END IF

IF |neighbors| = 1 THEN

I :=(c);

Send(I) to parent;

Become FINISHED;

END IF

Third Step:

States: {FINISHED, SLEEPING, READY};

Initial State: {FINISHED};

Termination State: {SLEEPING, READY};

Message S: A message that the current node p_v in T sends to its parent containing dimensions, facts, complete dimension mappings as well as complete fact mappings;

count: A temporary variable indicating how many messages need to be received by the

root;

$\mathcal{M}'^{v}_{\mathcal{D}}$: A collection of complete dimension mappings of p_v;

$\mathcal{M}'^{v}_{\mathcal{F}}$: A collection of complete fact mappings of p_v;

D_v: A set of dimensions of p_v in $\mathcal{M}'^{v}_{\mathcal{D}}$;

F_v: A set of facts of p_v in $\mathcal{M}'^{v}_{\mathcal{F}}$;

$\mathcal{D}$: a family of dimensions;

$\mathcal{F}$: a family of facts;

$\mathcal{M}'_{\mathcal{D}}$: a family of complete dimension mappings;

$\mathcal{M}'_{\mathcal{F}}$: a family of complete fact mappings;

FINISHED:

IF parent is NOT Null THEN

$S := (D_v, F_v, \mathcal{M}'^{v}_{\mathcal{D}}, \mathcal{M}'^{v}_{\mathcal{F}})$;

Send(S) to parent;

ELSE

count := c;

END IF

Become SLEEPING;

SLEEPING:

Receiving(S)

IF parent is NOT Null THEN

Send(S) to parent;

Become SLEEPING;

ELSE

Store $S.D_v$ in $\mathcal{D}$;

Store S.F_v in $\mathcal{F}$;

Store $S.\mathcal{M}'^{v}_{\mathcal{D}}$ in $\mathcal{M}'_{\mathcal{D}}$;

Store $S.\mathcal{M}'^{v}_{\mathcal{F}}$ in $\mathcal{M}'_{\mathcal{F}}$;

count := count - 1;

END IF

IF count = 1 THEN

Become READY;

END IF

Fourth Step:

States: {READY, GOOD};

Initial State: {READY};

Termination State: {GOOD};

READY:

FOR each dimension mapping $M_d(d_u, d_{v'}) \in \mathcal{M}^{u}_{\mathcal{D}}$

Get d_u from D_u;

IF CompleteDimensionMapping($M_d(d_u, d_{v'}), d_u, d_{v'}$) is TRUE THEN

FOR each collection of dimension mappings $\mathcal{M}'^{v}_{\mathcal{D}}$ in $\mathcal{M}'_{\mathcal{D}}$

FOR each dimension mapping $M_d(d_v, d_w) \in \mathcal{M}'^{v}_{\mathcal{D}}$

IF d_v is $d_{v'}$ THEN

Get d_v from D_v of $\mathcal{D}$;

Get d_w from D_w of $\mathcal{D}$;

DimensionMappingComposition($M_d(d_u, d_{v'}), M_d(d_v, d_w), d_u, d_v, d_w$);

END IF

END FOR

END FOR

ELSE

Do nothing;

END IF

END FOR

FOR each fact mapping $M_f(f_u, f_{v'}) \in \mathcal{M}^u_{\mathcal{F}}$

Get f_u from F_u;

IF CompleteFactMapping($M_f(f_u, f_{v'}), f_u, f_{v'}$) is TRUE THEN

FOR each collection of fact mappings $\mathcal{M}'^v_{\mathcal{F}}$ in $\mathcal{M}'_{\mathcal{F}}$

FOR each fact mapping $M_f(f_v, f_w) \in \mathcal{M}'^v_{\mathcal{F}}$

IF f_v is $f_{v'}$ THEN

Get f_v from F_v of $\mathcal{F}$;

Get f_w from F_w of $\mathcal{F}$;

FactMappingComposition($M_f(f_u, f_{v'}), M_d(f_v, f_w), f_u, f_v, f_w$);

END IF

END FOR

END FOR

ELSE

Do nothing;

END IF

END FOR

Become GOOD;

Fifth Step:

States: {GOOD, SLEEPING, TERMINATION};

Initial State: {GOOD, SLEEPING};

Termination State: {TERMINATION};

M: Either a dimension mapping or a fact mapping that will be sent to its destination;

Message H: A message containing the destination and M;

children(p_v): children of p_v in T;

p_x: The destination that M should be sent to;

GOOD:

Get children(root);

FOR each mapping M

Retrieve the destination p_x;

$H := (p_x, M)$;

Send(H) to children(root);

END FOR

Become TERMINATION;

SLEEPING:

Receiving(H)

Get children(p_v);

IF $p_v = H.p_x$ THEN

Store $H.M$;

ELSE IF |Tree-Neighbors| ≥ 2 THEN

Send(H) to children(p_v);

END IF

c := c - 1;

IF c = 0 THEN

Become TERMINATION;

END IF

4.2 Functions: Complete Dimension and Fact Mappings

This section describes two local functions to determine the completeness of a dimension mapping and of a fact mapping, respectively. In the following section, we present the other two local functions to compose dimension mappings and fact mappings, respectively, when we have complete mappings (either dimension or fact mappings).

4.2.1 Function: Complete Dimension Mappings

Input:

A dimension mapping $M_d(d_v, d_w)$ as well as two dimensions d_v and d_w.

Name:

Bool CompleteDimensionMapping(Dimension Mapping $M_d(d_v, d_w)$,

Dimension d_v, Dimension d_w);

Variable $M_d(d_v, d_w)$: Uses the passing by reference method because $M_d(d_v, d_w)$ may be eventually updated inside the function if it is incomplete.

Variables d_v and d_w: Use the passing by value method because d_v and d_u will not be updated inside the function.

Return:

The value of TRUE or FALSE as well as $M_d(d_v, d_w)$.

Given a dimension mapping $M_d(d_v, d_w)$ as well as two dimensions d_v and d_w, in order to determine whether $M_d(d_v, d_w)$ is complete or not, we need to check whether every hierarchy level of d_v and d_w appears in $M_d(d_v, d_w)$ exactly once or not. If every hierarchy level of d_v and d_w appears in $M_d(d_v, d_w)$ exactly once, then $M_d(d_v, d_w)$ is a complete dimension mapping; if any one of the hierarchy levels of d_v or d_w does not appear in $M_d(d_v, d_w)$, then $M_d(d_v, d_w)$ is an incomplete dimension mapping. If $M_d(d_v, d_w)$ is incomplete, then we try to make it complete by building a collection of extra dimension mapping tables at the instance level. We shall deal with three possible cases as follows:

- $M_d(d_v, d_w)$ is complete;
- $M_d(d_v, d_w)$ is incomplete, and it is possible to make it complete;
- $M_d(d_v, d_w)$ is incomplete, and it is impossible to make it complete.

Data Structure: Complete Dimension Mapping

Figure 3.1 shows the data structure of a complete dimension mapping. It consists of two columns, namely $Dimension\_d_v$ and $Dimension\_d_w$. For example, in Figure 1.3, we have a dimension mapping $M_d(Students, Student)$. Because it is incomplete, we add the attribute *Option* to $M_d(Students, Student)$ and build the extra dimension mapping table $T_d(K(Students), q(Option))$. Figure 3.1 also shows the data structure of an extra dimension mapping table consists of two columns, namely $Surrogate\_Key$ and $Value$. The values of the pair of $Surrogate\_Key$ and $Value$ are generated by the conjunctive query $q(K(Students), q(Option))$.

Algorithm:

Bool CompleteDimensionMapping($M_d(d_v, d_w)$, d_v, d_w)

BEGIN

Bool Complete := TRUE;

CheckCompletenessD(Complete, $M_d(d_v, d_w)$, d_v);

IF Complete is FALSE THEN

RETURN Complete;

END IF

CheckCompletenessD(Complete, $M_d(d_v, d_w)$, d_w);

Return Complete;

END

Function:

Void CheckCompletenessD(Bool C, Dimension Mapping $M_d(d, d')$, Dimension d)

Input: A dimension mapping $M_d(d, d')$, a dimension d, and a boolean variable C.

Return: The boolean variable C as well as the dimension mapping $M_d(d, d')$.

Variable $M_d(d, d')$: Uses the passing by reference method because $M_d(d, d')$ may be eventually updated inside the function if it is incomplete.

Variable C: Uses the passing by reference method because C may be updated inside the function if the dimension mapping is incomplete.

Variable d: Uses the passing by value method because d will not be updated inside the function.

Void CheckCompletenessD(Bool C, Dimension Mapping $M_d(d, d')$, Dimension d)

BEGIN

FOR each hierarchy level $l \in d$

IF l does NOT appear in $M_d(d, d')$ THEN

Get $q(K(d'), q(l))$;

IF $q(K(d'), q(l))$ is NOT Null THEN

Put l in the column *Dimension_d* of $M_d(d, d')$;

Put $q(K(d'), q(l))$ in the column *Dimension_d'* of $M_d(d, d')$;

ELSE IF $q(K(d'), q(l))$ is Null THEN

C := FALSE;

END IF

END IF

END FOR

END

4.2.2 Function: Complete Fact Mappings

This function is similar to the function of determining the completeness of a dimension mapping.

Input:

A fact mapping $M_f(f_v, f_w)$ as well as two facts f_v and f_w.

Name:

Bool CompleteFactMapping(Fact Mapping $M_f(f_v, f_w)$, Fact f_v, Fact f_w);

Variable $M_f(f_v, f_w)$: Uses the passing by reference method because $M_f(f_v, f_w)$ may be eventually updated inside the function if it is incomplete;

Variables f_v and f_w: Use the passing by value method because f_v and f_u will not be updated inside the function.

Return:

The value of TRUE or FALSE as well as $M_f(f_v, f_w)$.

Given a fact mapping $M_f(f_v, f_w)$ as well as two facts f_v and f_w, in order to determine

whether $M_f(f_v, f_w)$ is complete or not, we need to check whether every fact measure of f_v and f_w appears in $M_f(f_v, f_w)$ exactly once or not. If every fact measure of f_v and f_w appears in $M_f(f_v, f_w)$ exactly once, then $M_f(f_v, f_w)$ is a complete fact mapping; if any one of the fact measures of f_v or f_w does not appear in $M_f(f_v, f_w)$, then $M_f(f_v, f_w)$ is an incomplete fact mapping. If $M_f(f_v, f_w)$ is incomplete, then we try to make it complete by building a collection of extra fact mapping tables at the instance level. Again, we shall deal with three possible cases as follows:

- $M_f(f_v, f_w)$ is complete;
- $M_f(f_v, f_w)$ is incomplete, and it is possible to make it complete;
- $M_f(f_v, f_w)$ is incomplete, and it is impossible to make it complete.

Data Structure: Complete Fact Mapping

Figure 3.2 shows the data structure of a complete fact mapping. It consists of two columns, namely $Fact\_f_v$ and $Fact\_f_w$. For example, in Figure 1.4, we have a fact mapping $M_f(Grades, Marks)$. Because it is incomplete, we add the fact measure *Rank* to $M_f(Grades, Marks)$ and build the extra fact mapping table:

$$T_f(FK(Marks), q(r(SGrade, SRank), Mark)).$$

Figure 3.2 also shows the data structure of an extra fact mapping table. It is slightly different from the data structure of an extra dimension mapping table. The number of columns of the extra table is various because it depends on how many foreign keys the corresponding fact has. If the number of foreign keys of the fact is n, then the number of columns of the extra table is $n + 1$. n columns are for all the foreign keys, and one is

for $Fact\_Measure$. The values of the foreign keys and $Fact\_Measure$ are generated by the non-recursive query $q(FK(Marks), q(r(SGrade, SRank), Mark))$.

Algorithm:

Bool CompleteFactMapping($M_f(f_v, f_w)$, f_v, f_w)

BEGIN

Bool Complete := TRUE;

CheckCompletenessF(Complete, $M_f(f_v, f_w)$, f_v);

IF Complete is FALSE THEN

RETURN Complete;

END IF

CheckCompletenessF(Complete, $M_f(f_v, f_w)$, f_w);

Return Complete;

END

Function:

Void CheckCompletenessF(Bool C, Fact Mapping $M_f(f, f')$, Fact f)

Input: A fact mapping $M_f(f, f')$, a fact f, and a boolean variable C.

Return: The boolean variable C as well as the fact mapping $M_f(f, f')$.

Variable $M_f(f, f')$: Uses the passing by reference method because $M_f(f, f')$ may be eventually updated inside the function if it is incomplete.

Variable C: Uses the passing by reference method because C may be updated inside the function if the dimension mapping is incomplete.

Variable f: Uses the passing by value method because f will not be updated inside the function.

Void CheckCompletenessF(Bool C, Fact Mapping $M_f(f, f')$, Fact f)

BEGIN

FOR each fact measure $m \in f$

IF m does NOT appear in $M_f(f, f)$ THEN

IF there exists $m^i \in f$ ($m^i \neq m$, and m^i does appear in $M_f(f, f')$) such that m and m^i has a relationship $r(m, m^i)$ THEN

Find the pair $(m^i, m^{i'})$ in $M_f(f, f')$;

Get $q(r(m, m^i), m^{i'})$;

IF $q(r(m, m^i), m^{i'})$ is NOT Null THEN

Get $q(FK(f'), q(r(m, m^i), m^{i'}))$;

Put m in the column $Fact\_f$ of $M_f(f, f')$;

Put $q(FK(f'), q(r(m, m^i), m^{i'}))$ in the column $Fact\_f'$ of $M_f(f, f')$;

ELSE IF $q(r(m, m^i), m^{i'})$ is Null THEN

Complete := FALSE;

END IF

ELSE IF there is NO such $m^i \in f$ ($m^i \neq m$, and m^i does appear in $M_f(f, f')$) such that m and m^i has a relationship $r(m, m^i)$ THEN

Complete := FALSE;

END IF

END IF

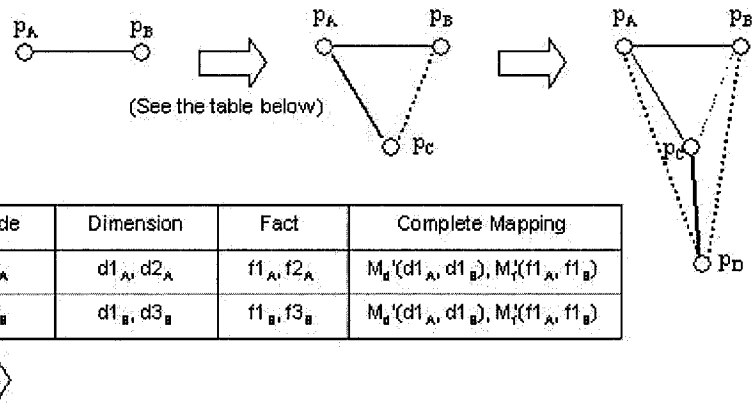
END FOR

END

4.3 Functions: Dimension and Fact Mapping Composition

Figure 4.1 shows a scenario of composing dimension mappings and fact mappings when a new PDW is added to the system. There are four PDWs p_A, p_B, p_C and p_D. They have data marts DM_A, DM_B, DM_C as well as DM_D, respectively. Furthermore, p_A that has already been in a PDWS has dimensions $d1_A$ and $d2_A$ as well as facts $f1_A$ and $f2_A$; p_B that also has already been in the system has dimensions $d1_B$ and $d3_B$ as well as facts $f1_B$ and $f3_B$; p_C that is going to be firstly added to the system has dimensions $d1_C$ and $d4_C$ as well as facts $f1_C$ and $f4_C$; p_D that is going to be secondly added to the system has dimensions $d1_D$ and $d5_D$ as well as facts $f1_D$ and $f5_D$. Finally, A complete dimension mapping $M'_d(d1_A, d1_B)$ and a complete fact mapping $M'_f(f1_A, f1_B)$ are in place.

When p_C is added, p_C gets acquainted with p_A. There is no acquaintance between p_B and p_C. Then, a complete dimension mapping $M'_d(d1_A, d1_C)$ and a complete fact mapping $M'_f(f1_A, f1_C)$ are built. Now, we can build a dimension mapping $M'_d(d1_B, d1_C)$ by using the existing complete dimension mappings $M'_d(d1_A, d1_B)$ and $M'_d(d1_A, d1_C)$. On the other hand, we also can build a fact mapping $M'_f(f1_B, f1_C)$ by using the existing complete fact mappings $M'_f(f1_A, f1_B)$ and $M'_f(f1_A, f1_C)$. By Proposition 1, both $M'_d(d1_B, d1_C)$ and



Node	Dimension	Fact	Complete Mapping
p_A	$d1_A, d2_A$	$f1_A, f2_A$	$M_d'(d1_A, d1_B), M_f'(f1_A, f1_B)$
p_B	$d1_B, d3_B$	$f1_B, f3_B$	$M_d'(d1_A, d1_B), M_f'(f1_A, f1_B)$

Node	Dimension	Fact	Complete Mapping
p_A	$d1_A, d2_A$	$f1_A, f2_A$	$M_d'(d1_A,d1_B), M_f'(f1_A,f1_B), \mathbf{M_d'(d1_A,d1_C), M_f'(f1_A,f1_C)}$
p_B	$d1_B, d3_B$	$f1_B, f3_B$	$M_d'(d1_A,d1_B), M_f'(f1_A,f1_B), \mathbf{M_d'(d1_B,d1_C), M_f'(f1_B,f1_C)}$
p_C	$d1_C, d4_C$	$f1_C, f4_C$	$\mathbf{M_d'(d1_A,d1_C), M_f'(f1_A,f1_B), M_d'(d1_B,d1_C), M_f'(f1_B,f1_C)}$

Figure 4.1: Dimension and Fact Mapping Composition Scenario

$M'_f(f1_B, f1_C)$ are complete. It turns out that a complete mapping(either a complete dimension mapping or a complete fact mapping) is transitive. Similarly, when p_D is added, p_D gets acquainted with p_C. There is no acquaintance between p_D and p_A ,or between p_D and p_B. Then, a complete dimension mapping $M'_d(d1_C, d1_D)$ and a complete fact mapping $M'_f(f1_C, f1_D)$ are built between p_C and p_D. Now, we have complete dimension mappings $M'_d(d1_A, d1_C)$, $M'_d(d1_B, d1_C)$ and $M'_d(d1_C, d1_D)$, as well as complete fact mappings $M'_f(f1_A, f1_C)$, $M'_f(f1_B, f1_C)$ and $M'_f(f1_C, f1_D)$. We observe that we can build complete dimension mappings $M'_d(d1_A, d1_D)$ and $M'_d(d1_B, d1_D)$, as well as complete fact mappings $M'_f(f1_A, f1_D)$ and $M'_f(f1_B, f1_D)$ by using those complete dimension mappings and complete fact mappings respectively.

Moreover, suppose that besides dimensions $d1_C$ and $d1_4$, C has another dimension $d3_C$. Then, we would have a dimension mapping $M_d(d3_B, d3_C)$ between p_B and p_C in place when p_C was added. In our scenario described above, there is no acquaintance between p_B and p_C but between p_A and p_C only. Unfortunately, we can not retrieve the mapping information of $d3_B$ from p_A (or any other nodes than p_B). The mapping composition technique is not applied to this case. We need to directly establish an acquaintance between p_B and p_C. Obviously, it is a design decision. Therefore, we claim that when a PDW is added to the network, its neighbors have sufficient mapping information for us to run our algorithm to compose mappings (either dimension mappings or fact mappings) for the added PDW.

4.3.1 Function: Dimension Mapping Composition

Suppose that we have three dimensions $d1_A$, $d1_B$ and $d1_C$, as well as two complete dimension mappings $M'_d(d1_A, d1_B)$ and $M'_d(d1_B, d1_C)$. Furthermore, suppose that $M'_d(d1_A, d1_B)$ consists of a dimension mapping $M_d(d1_A, d1_B)$ and a set of conjunctive queries $Q[d1_A, d1_B]$; suppose that $M'_d(d1_B, d1_C)$ consists of a dimension mapping $M_d(d1_B, d1_C)$ and a set of conjunctive queries $Q[d1_B, d1_C]$. We want to establish a complete dimension mapping $M'_d(d1_A, d1_C)$. First, we look at every pair (l^i_A, l^i_B) of $M'_d(d1_A, d1_B)$ and every pair (l^j_B, l^j_C) of $M'_d(d1_B, d1_C)$. The definition of complete dimension mappings guarantees that any pair in a complete dimension mapping can be a pair of two hierarchy levels of two dimensions respectively, or one hierarchy level of a dimension and an extra dimension mapping table. Then, on one hand, If l^i_B and l^j_B are equal, meanwhile if l^i_A and l^j_C are hierarchy

levels, then we put l_A^i in the column *Dimension_*$d1_A$ of $M'_d(d1_A, d1_C)$ meanwhile put l_C^j in the column *Dimension_*$d1_C$ of $M'_d(d1_A, d1_C)$. If l_A^i is a hierarchy level and l_C^j is an extra dimension mapping table, then we get a query $q \in Q[d1_B, d1_C]$ over l_C^j, rewrite q to a query q' such that l_A^i and the extra dimension mapping table that is generated by q' are a pair in $M'_d(d1_A, d1_C)$. We follow a similar step to deal with the case in which l_A^i is an extra dimension mapping table and l_C^j is a hierarchy level. Moreover, If l_B^i and l_B^j are extra dimension mapping tables and contain the same values, then both l_A^i of the pair (l_A^i, l_B^i) and l_C^j of the pair (l_B^j, l_C^j) must be hierarchy levels of $d1_A$ and $d1_C$ respectively. As a result, we also put l_A^i in the column *Dimension_*$d1_A$ of $M'_d(d1_A, d1_C)$ meanwhile put l_C^j in the column *Dimension_*$d1_C$ of $M'_d(d1_A, d1_C)$. On the other hand, if for a pair (l_A^i, l_B^i) of $M'_d(d1_A, d1_B)$ we cannot find any pair (l_B^j, l_C^j) of $M'_d(d1_B, d1_C)$ in which l_B^j is equal to l_B^i, then by the definition of complete dimension mappings, the pair (l_A^i, l_B^i) must consist of a hierarchy level of $d1_A$ and a query $q \in Q[d1_A, d1_B]$ representing l_B^j. In this case, we need to add l_A^i to the column of *Dimension_*$d1_A$ meanwhile to rewrite q to a query q' such that l_A^i and the extra dimension mapping table that is generated by q' are a pair in $M'_d(d1_A, d1_C)$. Finally, a similar process can be followed when we deal with those pairs of $M'_d(d1_B, d1_C)$ that $M'_d(d1_A, d1_B)$ does not have.

Input:

Three dimensions d_u,d_v as well as d_w, a complete dimension mappings $M'_d(d_u, d_v)$ that consists of a dimension mapping $M_d(d_u, d_v)$ and a set of conjunctive queries $Q[d_u, d_v]$, and a complete dimension mappings $M'_d(d_v, d_w)$ that consists of a dimension mapping $M_d(d_v, d_w)$ and a set of conjunctive queries $Q[d_v, d_w]$.

Name:

DimensionMappingComposition(Dimension Mapping $M'_d(d_u, d_v)$,

Dimension Mapping $M'_d(d_v, d_w)$, Dimension d_u, Dimension d_v, Dimension d_w);

All those variables use the passing by value method because they will not be updated inside the function.

Return:

A complete dimension mapping $M'_d(d_u, d_w)$ that consists of a dimension mapping $M_d(d_u, d_w)$ and a set of conjunctive queries $Q[d_u, d_w]$.

Algorithm:

BEGIN

FOR every element l in the column *Dimension_*d_v of $M'_d(d_u, d_v)$

Get the pair (l^i_u, l) from $M'_d(d_u, d_v)$;

IF there exists the pair (l, l^j_w) in $M'_d(d_v, d_w)$ THEN

Get the pair (l, l^j_w) from $M'_d(d_v, d_w)$;

IF both l^i_u and l^j_w are hierarchy levels THEN

Put l^i_u in the column *Dimension_*d_u of $M'_d(d_u, d_w)$;

Put l^j_w in the column *Dimension_*d_w of $M'_d(d_u, d_w)$;

ELSE IF l^i_u is a hierarchy level THEN

Put l^i_u in the column *Dimension_*d_u of $M'_d(d_u, d_w)$;

Get the query $q(K(d_w), q(l))$ from $Q[d_v, d_w]$;

Rewrite a query $q(K(d_w), q(l_u^i))$ by $q(K(d_w), q(l))$;

Use $q(K(d_w), q(l_u^i))$ to generate $T_d(K(d_w), q(l_u^i))$;

Put $T_d(K(d_w), q(l_u^i))$ in the column *Dimension_*d_w of $M'_d(d_u, d_w)$;

ELSE IF l_w^j is a hierarchy level THEN

Put l_w^j in the column *Dimension_*d_w of $M'_d(d_u, d_w)$;

Get the query $q(K(d_u), q(l))$ from $Q[d_u, d_v]$;

Rewrite a query $q(K(d_u), q(l_w^j))$ by $q(K(d_u), q(l))$;

Use $q(K(d_u), q(l_w^j))$ to generate $T_d(K(d_u), q(l_w^j))$;

Put $T_d(K(d_u), q(l_w^j))$ in the column *Dimension_*d_u of $M'_d(d_u, d_w)$;

END IF

ELSE IF there does NOT exist the pair (l, l_w^j) in $M'_d(d_v, d_w)$ THEN

IF l is an extra dimension mapping table THEN

Get the query $q(K(d_v), q(l_u^i))$ from $Q[d_u, d_v]$;

Rewrite a query $q(K(d_w), q(l_u^i))$ by $q(K(d_v), q(l_u^i))$;

ELSE IF l_u^i and l are hierarchy levels THEN

Write a query $q(K(d_w), q(l_u^i))$;

END IF

Put l_u^i in the column *Dimension_*d_u of $M'_d(d_u, d_w)$;

Use $q(K(d_w), q(l_u^i))$ to generate $T_d(K(d_w), q(l_u^i))$;

Put $T_d(K(d_w), q(l_u^i))$ in the column *Dimension_*d_w of $M'_d(d_u, d_w)$;

END IF

END FOR

FOR every element l in the column *Dimension_*d_v of $M'_d(d_v, d_w)$

Get the pair (l, l_w^j) from $M'_d(d_v, d_w)$;

IF l_w^j does NOT exist in the column *Dimension_*d_w of $M'_d(d_u, d_w)$ THEN

IF l is an extra dimension mapping table THEN

Get the query $q(K(d_v), q(l_w^j))$ from $Q[d_v, d_w]$;

Rewrite a query $q(K(d_u), q(l_w^j))$ by $q(K(d_v), q(l_w^j))$;

ELSE IF l and l_w^j are hierarchy levels THEN

Write a query $q(K(d_u), q(l_w^j))$;

END IF

Put l_w^j in the column *Dimension_*d_w of $M'_d(d_u, d_w)$;

Use $q(K(d_u), q(l_w^j))$ to generate $T_d(K(d_u), q(l_w^j))$;

Put $T_d(K(d_u), q(l_w^j))$ in the column *Dimension_*d_u of $M'_d(d_u, d_w)$;

END IF

END FOR

END

4.3.2 Function: Fact Mapping Composition

Suppose that we have three facts $f1_A$, $f1_B$ and $f1_C$, as well as two complete fact mappings $M'_f(f1_A, f1_B)$ and $M'_f(f1_B, f1_C)$. Furthermore, suppose that $M'_f(f1_A, f1_B)$ consists of a fact mapping $M_f(f1_A, f1_B)$ and a set of non-recursive queries $Q(f1_A, f1_B)$; suppose that $M'_f(f1_B, f1_C)$ consists of a fact mapping $M_f(f1_B, f1_C)$ and a set of non-recursive queries $Q(f1_B, f1_C)$. We want to establish a complete fact mapping $M'_f(f1_A, f1_C)$. As we have argued in the previous subsection of dimension mapping composition, a sim-

ilar argument holds.

Input:

Three facts f_u,f_v as well as f_w, a complete fact mappings $M'_f(f_u, f_v)$ that consists of a fact mapping $M_f(f_u, f_v)$ and a set of non-recursive queries $Q[f_u, f_v]$, and a complete fact mappings $M'_f(f_v, f_w)$ that consists of a fact mapping $M_f(f_v, f_w)$ and a set of non-recursive queries $Q[f_v, f_w]$.

Name:

FactMappingComposition(Fact Mapping $M'_f(f_u, f_v)$, Fact Mapping $M'_f(f_v, f_w)$,

Fact f_u, Fact f_v, Fact f_w);

All those variables use the passing by value method because they will not be updated inside the function.

Return:

A complete fact mapping $M'_f(f_u, f_w)$ that consists of a fact mapping $M_f(f_u, f_w)$ and a set of non-recursive queries $Q[f_u, f_w]$.

Algorithm:

BEGIN

FOR every element m in the column $Fact\_f_v$ of $M'_f(f_u, f_v)$

Get the pair (m^i_u, m) from $M'_f(f_u, f_v)$;

IF there exists the pair (m, m^j_w) in $M'_f(f_v, f_w)$ THEN

Get the pair (m, m^j_w) from $M'_f(f_v, f_w)$;

IF both m_u^i and m_w^j are fact measures THEN

Put m_u^i in the column $Fact\_f_u$ of $M'_f(f_u, f_w)$;

Put m_w^j in the column $Fact\_f_w$ of $M'_f(f_u, f_w)$;

ELSE IF m_u^i is a fact measure THEN

Put m_u^i in the column $Fact\_f_u$ of $M'_f(f_u, f_w)$;

Get the query $q(FK(f_w), q(r(m_v^i, m_v^k), m_w^h))$ from $Q[f_v, f_w]$

such that the pair (m_v^k, m_w^h) exists in $M'_f(f_v, f_w)$;

Rewrite a query $q(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$ by $q(FK(f_w), q(r(m_v^i, m_v^k), m_w^h))$

(the pairs (m_u^i, m_v^i) and (m_u^k, m_v^k) exist in $M'_f(f_u, f_v)$);

Use $q(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$ to generate $T_f(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$;

Put $T_f(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$

in the column $Fact\_d_w$ of $M'_f(f_u, f_w)$;

ELSE IF m_w^j is a fact measure THEN

Put m_w^i in the column $Fact\_f_w$ of $M'_f(f_u, f_w)$;

Get the query $q(FK(f_u), q(r(m_v^i, m_v^k), m_u^h))$ from $Q[f_u, f_v]$

such that the pair (m_v^k, m_u^h) exists in $M'_f(f_u, f_v)$;

Rewrite a query $q(FK(f_u), q(r(m_w^i, m_w^k), m_u^h))$ by $q(FK(f_u), q(r(m_v^i, m_v^k), m_u^h))$

(the pairs (m_v^i, m_w^i) and (m_v^k, m_w^k) exist in $M'_f(f_v, f_w)$);

Use $q(FK(f_u), q(r(m_v^i, m_v^k), m_u^h))$ to generate $T_f(FK(f_u), q(r(m_v^i, m_v^k), m_u^h))$;

Put $T_f(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$

in the column $Fact\_d_u$ of $M'_f(f_u, f_w)$;

END IF

ELSE IF there does NOT exist the pair (m, m_w^j) in $M'_f(f_v, f_w)$ THEN

IF m is an extra fact mapping table THEN

Get the query $q(FK(f_v), q(r(m_u^i, m_u^k), m_v^h))$ from $Q[f_u, f_v]$

such that the pair (m_u^k, m_v^h) exists in $M_f'(f_u, f_v)$;

Rewrite a query $q(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$ by $q(FK(f_v), q(r(m_u^i, m_u^k), m_v^h))$

(the pair (m_v^h, m_w^h) exists in $M_f'(f_v, f_w)$);

ELSE IF m_u^i and m are fact measures THEN

Write a query $q(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$

END IF

Put m_u^i in the column $Fact\_f_u$ of $M_f'(f_u, f_w)$;

Use $q(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$ to generate $T_f(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$;

Put $T_f(FK(f_w), q(r(m_u^i, m_u^k), m_w^h))$ in the column $Fact\_f_w$ of $M_f'(f_u, f_w)$;

END IF

END FOR

FOR every element m in the column $Fact\_f_v$ of $M_f'(f_v, f_w)$

Get the pair (m, m_w^j) from $M_f'(f_v, f_w)$;

IF m_w^j does NOT exist in the column $Fact\_f_w$ of $M_f'(f_u, f_w)$ THEN

IF m is an extra fact mapping table THEN

Get the query $q(FK(f_v), q(r(m_w^i, m_w^j), m_v^k))$ from $Q[f_v, f_w]$

such that the pair (m_v^k, m_w^i) exists in $M_f'(f_v, f_w)$;

Rewrite a query $q(FK(f_u), q(r(m_w^i, m_w^j), m_u^k))$ by $q(FK(f_v), q(r(m_w^i, m_w^j), m_v^k))$

(the pair (m_u^k, m_v^k) exists in $M_f'(f_u, f_v)$);

ELSE IF m_w^j and m are fact measures THEN

Write a query $q(FK(f_u), q(r(m_w^i, m_w^j), m_u^k))$

Put m_w^j in the column $Fact\_f_w$ of $M'_f(f_u, f_w)$;

Use $q(FK(f_u), q(r(m_w^i, m_w^j), m_u^k))$ to generate $T_f(FK(f_u), q(r(m_w^i, m_w^j), m_u^k))$;

Put $T_f(FK(f_u), q(r(m_w^i, m_w^j), m_u^k))$ in the column $Fact\_d_u$ of $M'_f(f_u, f_w)$;

END IF

END FOR

END

4.4 Running Example

This section continues using our running example to trace the most important two functions of the unified algorithm. Those two functions are the functions of composing complete dimension mappings and complete fact mappings.

Firstly, we trace the function of composing complete dimension mappings. In Example 5, we have the complete dimension mapping between the PDWs of Universities A and B:

$$M'_d(Students, Student) = \{(StudentID, SID), (T_d(K(Students), q(Option)), Option),$$

$$(Program, Prog), (Department, Dept), (Faculty, Faculty)\}.$$

$M'_d(Students, Student)$ consists of the dimension mapping:

$$M_d(Students, Student) = \{(StudentID, SID), (Program, Prog),$$

$$(Department, Dept), (Faculty, Faculty)\},$$

and the set of conjunctive queries $Q[Students, Student] = \{q(K(Students), q(Option))\}$ (here, the set of conjunctive queries has only one query). The query:

$$q(K(Students), q(Option))$$

generates the extra dimension mapping table $T_d(K(Students), q(Option))$. Figure 1.2 lists the hierarchy levels of the *Students* and *Student* dimensions. Moreover, in Example 9, we have the dimension $AllStudents(SNO, SProg, SDept, SFaculty)$ of the data mart *StudentRegister* at University C and the complete dimension mapping between the PDWs of Universities of A and C:

$$M'_d(Students, AllStudents) = \{(StudentID, SNO), (Program, SProg),$$

$$(Department, SDept), (Faculty, SFaculty)\}$$

$M'_d(Students, AllStudents)$ consists of the dimension mapping:

$$M_d(Students, AllStudents) = \{(StudentID, SNO), (Program, SProg),$$

$$(Department, SDept), (Faculty, SFaculty)\},$$

and the set of conjunctive queries:

$$Q[Students, AllStudents](Q[Students, AllStudents] = \phi).$$

We get the pair $(StudentID, SID)$ from $M'_d(Students, Student)$. In $M'_d(Students, AllStudents)$, there exists the pair $(StudentID, SNO)$. The pairs $(StudentID, SID)$ and $(StudentID, SNO)$ have a common element, the *StudentID* hierarchy level. The *StudentID* hierarchy level is in the *Students* dimension. Then, we compose the pair (SNO, SID) in which the *SNO* hierarchy level is from the pair

$(StudentID, SNO)$ and the *SID* hierarchy level is from the pair $(StudentID, SID)$. We put the pair (SNO, SID) into the dimension mapping $M'_d(AllStudents, Student)$. Similarly, we get the pairs $(Program, Prog)$, $(Department, Dept)$, $(Faculty, Faculty)$ from $M'_d(Students, Student)$. In $M'_d(Students, AllStudents)$, there exist the pairs $(Program, SProg)$, $(Department, SDept)$, $(Faculty, SFaculty)$, respectively. The pairs $(Program, Prog)$ and $(Program, SProg)$ have a common element, the *Program* hierarchy level; the pairs $(Department, Dept)$ and $Department, SDept$ have a common element, the *Department* hierarchy level; the pairs $(Faculty, Faculty)$ and $(Faculty, SFaculty)$ have a common element, the *Faculty* hierarchy level. The *Program*, *Department* and *Faculty* hierarchy levels are all in the *Students* dimension. Then, we compose the pairs $(SProg, Prog)$, $(SDept, Dept)$ and $(SFaculty, Faculty)$. We put those three pairs into $M'_d(AllStudents, Student)$. For the pair $(T_d(K(Students), q(Option)), Option)$ of $M'_d(Students, Student)$, there does not exist a pair in $M'_d(AllStudents, Students)$ such that those two pairs have a common element, the *Option* hierarchy level, which is from a single dimension. Then, we get the query $q(K(Students), q(Option))$ from $Q[Students, Student]$ and rewrite the query $q(K(Students), q(Option))$ to the query $q(K(AllStudents), q(Option))$. Finally, we use $q(K(AllStudents), q(Option))$ to generate the extra dimension mapping table $T_d(K(AllStudents), q(Option))$, compose the pair $(q(K(AllStudents), q(Option)), Option)$ (The *Option* hierarchy level is in the *Student* dimension), and put this pair into $M'_d(AllStudents, Student)$. As a result, we have the complete dimension mapping between Universities C and B:

$$M'_d(AllStudents, Student) =$$

$$\{(SNO, SID), (T_d(K(ALLStudents), q(Option)), Option),$$

$$(Program, Prog), (Department, Dept), (Faculty, Faculty)\}.$$

$M'_d(AllStudents, Student)$ consists of the dimension mapping:

$$M_d(AllStudents, Student) = \{(SNO, SID), (SProg, Prog),$$

$$(SDept, Dept), (SFaculty, Faculty)\},$$

and the set of conjunctive queries:

$$Q[AllStudents, Student] = \{q(K(AllStudents), q(Option))\}$$

(here, the set of conjunctive queries has only one query). Figure 3.5 shows the the result of the dimension mapping composition.

Secondly, we trace the function of composing complete fact mappings. In Example 6, we have the complete fact mapping between the PDWs of Universities A and B:

$$M'_f(Grades, Marks) = \{(Grade, Mark), (Rank, T_f(FK(Marks),$$

$$q(r(Grade, Rank), Mark)))\}.$$

$M'_f(Grades, Marks)$ consists of the fact mapping:

$$M_f(Grades, Marks) = \{(Grade, Mark)\},$$

and the set of non-recursive queries:

$$Q[Grades, Marks] = \{q(FK(Marks), q(r(Grade, Rank), Mark))\}$$

(here, the set of non-recursive queries has only one query).

The query $q(FK(Marks), q(r(Grade, Rank), Mark))$ generates the extra fact mapping table $T_f(FK(Marks), q(r(Grade, Rank), Mark))$. The *Grades* fact has two fact measures, *Grade* and *Rank*, and the $Marks$ fact has one fact measure, *Mark*. Moreover, in Example 10, we have the fact $SGrades(SGrade, SRank)$ of the data mart *StudentRegister* at University C and the complete fact mapping between the PDWs of Universities A and C:

$$M'_f(Grades, SGrades) = \{(Grade, SGrade), (Rank, SRank)\}.$$

$M'_f(Grades, SGrades)$ consists of the fact mapping:

$$M_f(Grades, SGrades) = \{(Grade, SGrade), (Rank, SRank)\},$$

and the set of non-recursive queries $Q[Grades, SGrades]$ $(Q[Grades, SGrades] = \phi)$.

We get the pair $(Grade, Mark)$ from $M'_f(Grades, Marks)$. In $M'_f(Grades, SGrades)$, there exists the pair $(Grade, SGrade)$. The pairs $(Grade, Mark)$ and $(Grade, SGrade)$ have a common element, the *Grade* fact measure. The *Grade* fact measure is in the *Grades* fact. Then, we compose the pair $(SGrade, Mark)$ in which the *SGrade* fact measure is from the pair $(Grade, SGrade)$ and the *Mark* fact measure is from the pair $(Grade, Mark)$. We put the pair $(SGrade, Mark)$ into the fact mapping $M'_f(SGrades, Marks)$. For the pair $(Rank, T_f(FK(Marks), q(r(Grade, Rank), Mark)))$ of $M'_f(Grades, Marks)$, there exists the pair $(Rank, SRank)$ in $M'_f(Grades, SGrades)$ such that those two pairs have a common element, the *Rank* fact measure, which is in the fact *Grades*. The pair $(Grade, Mark)$ exists in $M'_f(Grades, Marks)$. Then,

we get the query $q(FK(Marks), q(r(Grade, Rank), Mark))$ from $Q[Grades, Marks]$. The pairs $(Grade, SGrade)$ and $(Rank, SRank)$ exist in $M'_f(Grades, SGrades)$. After that, we rewrite the query $q(FK(Marks), q(r(Grade, Rank), Mark))$ to the query $q(FK(Marks), q(r(SGrade, SRank), Mark))$. Finally, we use $q(FK(Marks), q(r(SGrade, SRank), Mark))$ to generate the extra fact mapping table $T_f(FK(Marks), q(r(SGrade, SRank), Mark))$, compose the pair $(SRank, FK(Marks), q(r(SGrade, SRank), Mark))$ (The $SRank$ fact measure is in the $SGrades$ fact), and put this pair into $M'_f(SGrades, Marks)$. As a result, we have the complete fact mapping between Universities C and B:

$$M'_f(SGrades, Marks) =$$

$$\{(SGrade, Mark), (SRank, T_f(FK(Marks), q(r(SGrade, SRank), Mark))\}.$$

$M'_f(SGrades, Marks)$ consists of the fact mapping:

$$M'_f(SGrades, Marks) = \{(SGrade, Mark)\}$$

and the set of non-recursive queries:

$$Q[SGrades, Marks] = \{q(FK(Marks), q(r(SGrade, SRank), Mark))\}$$

(here, the set of non-recursive queries has only one query). Figure 3.6 shows the result of the fact mapping composition.

4.5 Proof of Correctness

In order to ensure that the algorithm runs correctly, we now give the proof of correctness as follows:

First Step: It is recognized as the spanning tree construction for an arbitrary strongly-connected graph. The process is also recognized as SHOUT (the combination of FLOOD and REPLY). As long as the graph is connected, SHOUT is correct.

Second Step: It is recognized as CONVERGE CASTING in a spanning tree. As long as the graph is connected, CONVERGE CASTING is correct.

Third Step: It is the same as the second step. It is recognized as CONVERGE CASTING in a spanning tree.

Fourth Step: Four local functions are invoked at the root. After the third step, the root has all the information of dimension mappings and fact mappings, so we can determine the completeness of those dimension mappings and fact mappings. Furthermore, by Proposition 1, we prove the completeness of a composed dimension (or fact) mapping.

Fifth Step: It is recognized as BROADCASTING in a spanning tree. Once again, as long as the graph is connected, BROADCASTING is correct.

4.6 Complexity

The unified algorithm for the DIMENSION MAPPING COMPOSTION AND FACT MAPPING COMPOSITION problems is a distributed algorithm, so the complexity is estimated by counting the number of messages sent over the network. We do a worst case analysis as follows.

Let m be the number of acquaintances, n be the number of PDWs in a network graph G, and T be a spanning tree of G. Then:

First Step: The total number of Q messages is $2m - n + 1$;

The total number of YES messages is $n-1$;

the total number of NO messages is $2(m-(n-1))$;

thus, the total number of messages is $(2m-n+1)+(n-1)+2(m-(n-1)) = 4m-2n+2$.

The worst case is that G is a complete graph; then, $m = (n-1)n/2$. Thus, the total number of messages is $4m-2n+2 = 4(n-1)n/2-2n+2 = 2n^2-4n+2 = O(n^2)$.

Second Step: It is CONVERGE CASTING in T, and $m = n-1$. Thus, every acquaintance in T is traveled exactly once. The total number of messages is $n-1 = O(n)$.

Third Step: Again, it is CONVERGE CASTING in T. Thus, the total number of messages is $(n-1) = O(n)$.

Fourth Step: There are four local functions that are invoked in this step. There is no message transmission in the network.

Fifth Step: Once the completeness of all the dimension mappings and fact mappings that the root have is determined at the root, as well as all the dimension mappings and fact mappings are composed at the root, the root sends all the updated dimension mappings and updated fact mappings (no matter whether they are eventually complete or incomplete) as well as all the composed dimension mappings and composed fact mappings to the expected destination in T. If a mapping (either a dimension mapping or a fact mapping) has a collection extra mapping tables, then a set of queries that generate those extra mapping tables will be sent together with the mapping. Thus, the number of messages sent by the root is the total number of the updated dimension mappings, updated fact mappings, composed dimension mappings and composed fact mappings. Let k be the total number of the updated dimension mappings, updated fact mappings, composed dimension mappings and composed fact mappings that need to be sent by the

root. In the worst case, a message containing a mapping travels each acquaintance of T exactly once, thus the total number of messages is $O(kn)$.

Therefore, the complexity of the algorithm is $O(n^2 + kn)$.

Chapter 5

EXPERIMENTS

This chapter describes two experiences concerning the performance of our mapping composition algorithm. We evaluate the performance of the algorithm for dimension mapping composition as well as the performance of the algorithm for fact mapping composition. We use Object-Oriented programming methodologies to simulate a P2P network, and implement the unified algorithm in Visual C++. At the end, we set up some measurements to study the robustness of our algorithm.

5.1 The Simulation

This simulation program runs on a Windows XP computer, which has a one-Giga-Hertz processor and five-hundred -twelve-Megabyte memory. The simulation mainly consists of three parts. The first part builds a P2P network as a distributed computing environment; the second part determines the completeness of a dimension mapping as well as the completeness of a fact mapping; the last part composes complete dimension mappings

and complete fact mappings. We have two parameters as the inputs of the algorithm. One is the total number of PDWs in the P2P network; the other is the percentage of dimensions that are used to build dimension mappings.

In the first part, we initially define parameters of the distributed computing environment and characteristics of the PDWs. The parameters include the delay (in milliseconds) of a message transmission, the label of each PDW in the network, the average execution time of a conjunctive query, the average execution time of a non-recursive query and etc. The characteristics of the PDWs include dimensions, facts, the number of dimensions, the number of facts, the number of hierarchy levels of a dimension, the number of fact measures of a fact, the neighbors of a PDW and etc. In the P2P network, each PDW has one data mart that consists of a fact and a set of dimensions. Each PDW has the same number of dimensions; however, the dimensions of every PDW are randomly selected from a pre-defined set of dimensions (for simplicity, we build a dimension mapping for the two dimensions that have the same name.) The number of dimensions is determined by the parameter of the percentage of dimensions. The number of hierarchy levels of a dimension is randomly chosen from a pre-defined range before a dimension mapping is built (once the number of hierarchy levels of the dimension of a PDW is chosen, it is a constant). It means that for the different dimensions with the same name at different PDWs, the numbers of hierarchy levels of those dimensions might be different. The same rule is applied to the numbers of fact measures of different facts with the same name. The neighbors of a PDW is determined by a search algorithm for dimensions. This algorithm is a minimum-finding algorithm. That is, for each dimension of a newly added PDW, we

need to find exactly one PDW in the network that has a dimension with the same name, and then make those two PDWs acquainted. This minimum-finding algorithm guarantees the strong connectivity of the network, the existence of the PDWs in the network, as well as the readiness of building dimension mappings and fact mappings between the PDWs. For two acquainted PDWs, we use an exhaustive search algorithm to build dimension mappings between them. The algorithm for building a fact mapping is partially trivial because each PDW has only one fact in the simulation. As a result, there is only one fact mapping between any two PDWs. We need to follow a similar process to build a fact mapping. Most importantly, it might be the case in which dimension mappings between two PDWs have been built but a fact mapping has not. This is because a data mart is the fundamental unit of a data warehouse. That is, a fact cannot exist without a set of dimensions that associate to it. Therefore, a reasonable assumption is that a fact mapping can be built only if dimension mappings are built for all the associated dimensions. We have taken this assumption into account in the implementation. We start with two PDWs in the network (obviously, those two PDWs are acquainted). Then, we build dimension mappings by using the exhaustive search algorithm and build a fact mapping between the two PDWs. Finally, we add a new PDW to the network, determine all the neighbors of the newly added PDW by using the minimum-finding algorithm and repeat the previous steps to build dimension mappings as well as fact mappings between the newly added PDW and all its neighbors.

In the second part, once the dimension mappings and fact mappings that the newly added PDW have are in place (a mapping resides at both sides of two acquainted PDWs;

nevertheless, the computation is carried out only at the side of the newly added PDW), we are ready to determine the completeness of the dimension mappings and fact mappings. Obviously, determining the completeness of the dimension mappings and fact mappings between the two initial PDWs is partially exceptional because the completeness should be determined initially. For a given dimension mapping, we use a flag to randomly indicate whether the dimension mapping is complete or incomplete. If it is incomplete, then we again randomly choose the number of the missing hierarchy levels for each dimension in the dimension mapping. Intuitively, for each dimension in the dimension mapping, the number of the missing hierarchy levels that the dimension should have is less than (if equal, then the case is trivial) the number of hierarchy levels that the corresponding dimension has. Finally, we use another flag to randomly indicate whether an incomplete dimension mapping can be made complete or not. On the other hand, a similar algorithm is given to determine the completeness of a fact mapping.

In the third part, once the completeness of all the dimension mappings and fact mappings that the newly added PDW have is determined, then we are ready to compose dimension mappings as well as fact mappings by using those mappings and the complete mappings that the newly added PDW has already "received". Given two dimension mappings, we first check the feasibility of composing dimension mappings for them. That is, we check the completeness for the one that resides at the newly added PDW (the other that is "received" from one of the other existing PDWs is complete). Then, if both dimension mappings are complete, we find the common dimension in both. Otherwise, we stop. Moreover, if the common dimension is not found in both dimension mappings,

we also stop. Finally, we follow the algorithm presented previously to generate a new dimension mapping by using those two complete dimension mappings. On the other hand, a similar algorithm is given to compose fact mappings. We repeat the three parts of the process until all the PDWs are added to the network. It is important to know that when the next PDW is added to the network, all the dimension mappings and fact mappings that the added PDW "receive" are complete.

5.2 The Measurements

In this section, we set up four measurements to study the robustness of the unified algorithm. Those four measurements provide different perspectives on algorithm performance.

The first measure is the comparison of different percentages of dimensions (which are obtained from the inputs) in terms of the running time of the algorithm for dimension mapping composition. The running time consists of two parts. One is the execution time of the algorithm for dimension mapping composition, and the other is the total execution time of a set of conjunctive queries that generates a collection of extra dimension mapping tables. Figure 5.1 shows that, with a fixed number of PDWs, when the percentage of dimensions is larger, the running time is higher. Furthermore, Figure 5.1 also shows that when the number of PDWs increases, the running time increases.

The second measure is the comparison of the numbers of the built dimension mappings

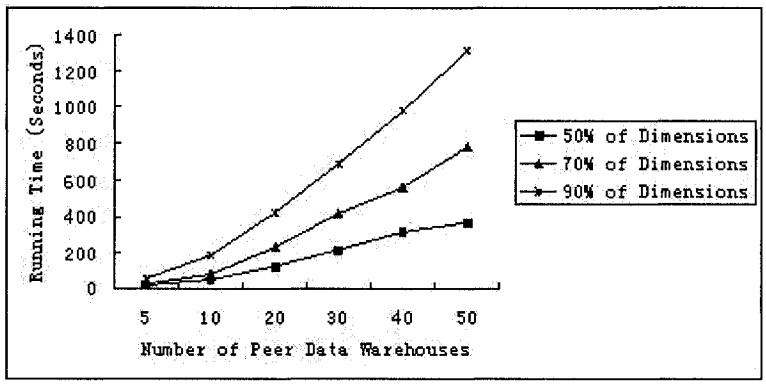


Figure 5.1: Dimension Mapping Composition Running Time

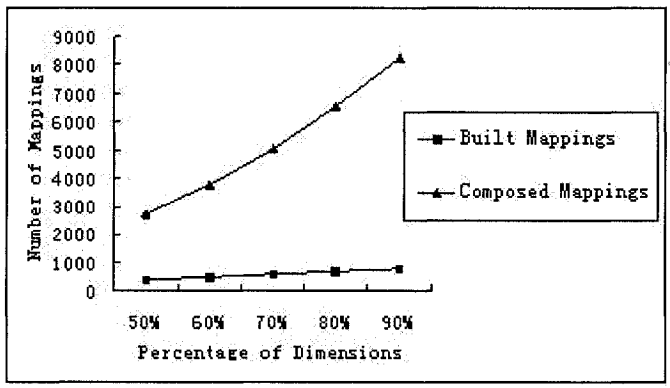


Figure 5.2: Numbers of Built and Composed Dimension Mappings

and the composed dimension mappings. Figure 5.2 shows that, with a fixed percentage of dimensions, the number of the composed dimension mappings is much larger than the number of the built dimension mappings. This is because in our experiments, when a new PDW is added to the network, it is forced to find the minimum set of neighbors such that all the dimensions that the added PDW has is able to be used in the process of building dimension mappings (the first part of the simulation). We can easily relax this requirement so that we can balance the numbers of the built dimension mappings

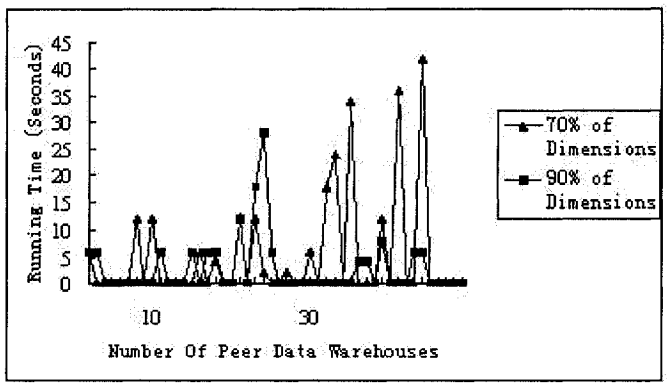


Figure 5.3: Fact Mapping Composition Running Time

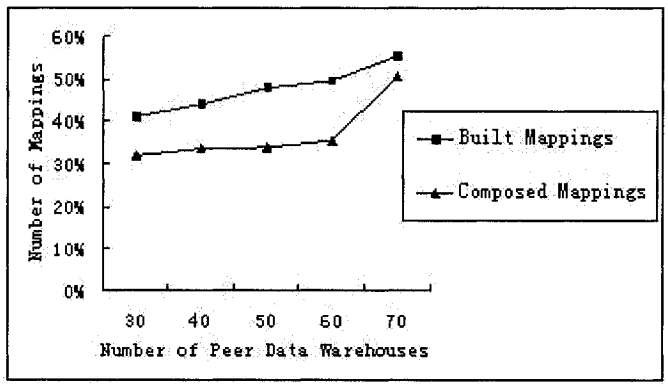


Figure 5.4: Numbers of Built and Composed Fact Mappings

and the composed dimension mappings. Moreover, Figure 5.2 also shows that when the percentage of dimensions increases, the numbers of built dimension mappings and the composed dimension mappings both increase.

The third measure is the comparison of different percentages of dimensions in terms of the running time of the algorithm for fact mapping composition. The running time also consists of two parts. One is the execution time of the algorithm for fact mapping

composition, and the other is the total execution time of a set of non-recursive queries that generates a collection of extra fact mapping tables. However, as shown in Figure 5.3, there is no standard relationship between different percentages of dimensions in this measurement. That is because, as we have mentioned this important assumption earlier, a fact mapping can be built only if dimension mappings are built for all the associated dimensions. Therefore, some PDWs may have complete fact mappings in place to compose fact mappings, and some PDWs may not even have fact mappings.

The last measure is the comparison of the percentages of the built fact mappings and the composed dimension mappings. Figure 5.4 shows that, with a fixed number of PDWs, the difference between the number of the built fact mapping and the number of the composed dimension mapping is very small. Furthermore, Figure 5.4 also shows that when the number of PDWs increase, the percentages of the built fact mappings as well as the composed fact mappings increase, and the increasing ratios are similar.

Chapter 6

CONCLUSION

We have studied the two problems of composing dimension and fact mappings in a PDWS. The first problem focuses on dimension mapping composition when dimension mappings are in place; the second problem focuses on fact mapping composition when fact mappings are in place. The idea of composing such dimension mappings and fact mappings comes from the solution to mapping composition in a P2P system [18].

In Chapter 1, we have introduced the two mapping composition problems using an appropriate running example. Chapter 2 has summarized the related work. Here, we give a brief review of the main concepts of data warehousing as well as those of peer-to-peer computing. Chapter 3 has presented a language for representing our mapping composition problems. Using this language, we define the main concepts of our approach such as dimension mapping, fact mapping, complete dimension mapping, complete fact mapping, extra dimension mapping table, extra fact mapping table, etc. Chapter 4 has presented a unified distributed algorithm for solving both mapping composition prob-

lems in five steps. The first, second and third steps consist of distributed computing algorithms for the purpose of exchanging necessary information between PDWs. The fourth step checks the readiness of mapping composition (that is, they examine the completeness of dimension mappings and fact mappings in order to make them complete if possible). The fourth step also composes those complete dimension mappings and complete fact mappings. The last step transfers the expected composed mappings to the desirable destinations. Moreover, in Chapter 4, we have proved the correctness of the unified algorithm and showed its complexity. Finally, we have used our running example to trace the major part of the unified algorithm (i.e., the two functions of composing complete dimension mappings and complete fact mappings). Chapter 5 has shown experimental results that illustrate the performance of the unified algorithm. There, we have explained the implementation of a simulated PDWS environment and set up four measurements to show the robustness of the algorithm.

As future work, several possible extensions of this work are:

- **Query processing:** Consider an OLAP query being issued at a local PDW in a PDWS. The query requests data from some other PDWs (acquainted or unacquainted PDWs). Then, we must have some appropriate mechanisms to rewrite the query at the local PDW. If the data is from an acquainted PDW, then we need to rewrite the query using proper complete dimension mappings and complete fact mappings that are in place. On the other hand, If the data is from an unacquainted PDW, then we need to rewrite the query using proper composed dimension mappings and fact mappings. Moreover, when an acquainted or unacquainted PDW

receives the rewritten sub-queries, we must appropriately process those sub-queries. Finally, when the local PDW gets the answers of all the sub-queries (if necessary, including the answer of the sub-query processed locally), it must return a correct, possibly complete answer.

- **Dynamics of P2P networks:** Consider the case in which, in a PDWS, a PDW crashes such that all the acquaintances associated with it are lost. This crashed PDW may come back in a while. However, this PDW may get acquainted with any other PDWs. Then, different mapping information would be in place. We want the mapping information currently stored in the PDW to be the same as the mapping information stored before the PDW crashed. We need to carry out appropriate strategies to achieve it.

- **Meta-data management:** The dynamics of P2P networks causes difficulties of meta-data management. Updating the meta-data is one of the most difficult and important tasks when a PDW crashes and comes back later. We want to maintain consistency of the meta-data so that we can accurately deal with query processing afterward.

Appendix A

Notations

P: A set of PWDs in a PDWS.

p_i: A PDW in the PDWS, $p_i \in P$.

DM_i: A data mart at the PDW p_i.

d_i: A dimension of the data mart DM_i at the PDW p_i.

f_i: A fact of the data mart DM_i at the PDW p_i.

D_i: A set of dimensions at the PDW p_i.

F_i: A set of facts at the PDW p_i.

l_i^x: A hierarchy level of the dimension d_i.

m_i^x: A fact measure of the fact f_i.

$K(d_i)$: A set of surrogate keys of the dimension d_i.

$FK(f_i)$ A set of foreign keys of the fact f_i.

$M_d(d_i, d_j)$: A dimension mapping of the dimensions d_i and d_j.

$M'_d(d_i, d_j)$: A complete dimension mapping of the dimensions d_i and d_j.

$M_f(f_i, f_j)$: A fact mapping of the facts f_i and f_j.

$M'_f(f_i, f_j)$: A complete fact mapping of the fact f_i and f_j.

$q[l_i^x]$: A conjunctive query over the hierarchy level l_i^x of the dimension d_i returns the set of values of l_i^x.

$r(m_i^x, m_i^y)$: A relationship between the fact measures m_i^x and m_i^y of the fact f_i. We can translate such a relationship into a SQL expression.

$q(r(m_i^x, m_i^y), m_j^{x'})$: A non-recursive query over the fact measure $m_j^{x'}$ of the fact f_j with the conditions of $r(m_i^x, m_i^y)$ returns the values of the "fact measure $m_j^{y'}$ of the fact f_j" (The pair $(m_i^x, m_j^{x'})$ is in the fact mapping $M_f(f_i, f_j)$, and the fact measure m_i^y is a dangling fact measure in the fact mapping $M_f(f_i, f_j)$).

$T_d(K(d_j), q(l_i^x))$: An extra dimension mapping table that has a set of pairs of the surrogate keys of the dimension d_j and the values of the hierarchy level l_i^x of the dimension d_i.

$q(K(d_j), q(l_i^x))$: A conjunctive query over the dimension d_j to generate an extra dimension mapping table.

$T_f(FK(f_j), q(r(m_i^x, m_i^y), m_j^{x'}))$: An extra fact mapping table that has a set of foreign keys of the fact f_j and the values of the "fact measure $m_j^{y'}$ of the fact f_j".

$q(FK(f_j), q(r(m_i^x, m_i^y), m_j^{x'}))$: A conjunctive query over the fact f_j to generate an extra fact mapping table.

$Q[d_i, d_j]$: A set of conjunctive queries over some hierarchy levels of the dimension d_i or d_j to generate extra dimension mapping tables for the dimension mapping $M_d(d_i, d_j)$.

$Q[f_i, f_j]$: A set of non-recursive queries over some fact measures of the fact f_i or f_j to generate extra fact mapping tables for the fact mapping $M_f(f_i, f_j)$.

$\mathcal{M}_{\mathcal{D}}^i$: A collection of dimension mappings at the PDW p_i.

$\mathcal{M}^i_{\mathcal{F}}$: A collection of fact mappings at the PDW p_i.

$\mathcal{M}'^i_{\mathcal{D}}$: A collection of complete dimension mappings at the PDW p_i.

$\mathcal{M}'^i_{\mathcal{F}}$: A collection of complete fact mappings at the PDW p_i.

$\mathcal{D}$: A collection of sets of dimensions in the PDWS, called a family of dimensions.

$\mathcal{F}$: A collection of sets of facts in the PDWS, called a family of facts.

$\mathcal{M}'_{\mathcal{D}}$: A collection of complete dimension mappings in the PDWS, called a family of complete dimension mappings.

$\mathcal{M}'_{\mathcal{F}}$: A collection of complete fact mappings in the PDWS, called a family of complete fact mappings.

Bibliography

[1] S. Abiteboul, R. Hull and V. Vianu. Foundations of Databases, Addison-Wesley, 1995.

[2] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, and J. Mylopoulos. The Hyperion Project: From Data Integration to Data Coordination. In ACM SIGMOD RECORD, 2003.

[3] P. Bernstein. Applying Model Management to Classical Meta Data Problems. In Proceedings of the Conference on Innovative Data Systems Research (CIDR), 2003.

[4] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data Management for Peer-to-Peer Computing: A Vision. In Proceedings of the 5th International Workshop on the Web and Databases (WebDB), 2002.

[5] D. Calvanese, E. Damaggio, G. De Giacomo, M. Lenzerini and R. Rosati. Semantic Data Integration in P2P Systems. In Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P), 2003.

[6] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical Foundations of Peer-to-Peer Data Integration. In Proceedings of the 21st ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS), 2004.

[7] L. Cabibbo and R. Torlone. Dimension Compatibility for Data Mart Integration. In Proceedings of the 12th Italian Symposium on Advanced Database Systems, 2004.

[8] M. M. Espil and A. A. Vaisman. Aggregate Queries in Peer-to-Peer OLAP. In Proceedings of the 7th International Workshop on Data Warehousing and OLAP, 2004.

[9] M. Friedman, A. Levy, and T. Millstein. Navigational Plans for Data Integration. In Proceedings of the 16th International Conference on Artificial Intelligence (AAAI), 1999.

[10] S. Gribble, A. Halevy, I. Zachary, M. Rodrig, and D. Suciu. What Can Peer-to-Peer Do for Databases? In Proceedings of the 4th International WebDB Workshop, 2001.

[11] D. R. Hofstadter. Godel, Escher, Bach: An Eternal Golden Braid, 20th anniversary edition, HarperCollins, 1999.

[12] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In Proceedings of the 19th IEEE-ICDE International Conference, 2003.

[13] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In Proceedings of the ACM SIGMOD International Conference, 2003.

[14] M. Jarke, M. Lenzerini, Y. Vassiliou, and P. Vassiliadis. Fundamentals of Data Warehouses Second Edition, Springer, 2003.

[15] M. Lenzerini. Data Integration: A Theoretical Perspective. In Proceedings of the 21st ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS), 2002.

[16] M. Lenzerini. Principles of P2P Data Integration. In Proceedings of the 3rd International Workshop on Data Integration over the Web, 2004.

[17] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of Multiple Autonomous Databases. ACM Comp. Surveys, 22(3): 267–293, 1990.

[18] J. Madhavan and A. Y. Halevy. Composing Mappings Among Data Sources. In Proceedings of the 29th International Conference on Very Large Data Bases, 2003.

[19] P. McBrien and A. Poulovassilis. Defining Peer-to-Peer Data Integration Using Both as View Rules. In Proceedings of the 1st DISP2P International Workshop, 2003.

[20] N. Santoro. Design and Analysis of Distributed Algorithms, draft, 2005

[21] L. Serafini, F. Giunchiglia, J. Mylopoulos, and P. Bernstein. The Logical Relational Model: Model and Proof Theory. In Technical Report 0112-23,ITC-IRST, 2001.

[22] I. Tatarinov and A. Halevy. Efficient Query Reformulation in Peer-Data Management Systems. In Proceedings of the ACM SIGMOD International Conference, 2004.